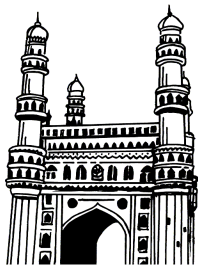
Rahul's ✔
Topper'sVoice

NEW SYLLABUS



B.C.A.

II Year IV Sem

Latest 2023 Edition

DATA SCIENCE USING PYTHON

- Study Manual
- Important Questions
- Short Question & Answers
- Choose the Correct Answers
- Fill in the blanks
- Solved Model Papers
- Lab Programming

` .199/-

- by -

WELL EXPERIENCED LECTURER

Rahul Publications™
Hyderabad. Cell : 9391018098, 9505799122

B.C.A.

II Year IV Sem

DATA SCIENCE USING PYTHON

Price ` 199

Sole Distributors : **Cell : 9391018098, 9505799122**

VASU BOOK CENTRE

Shop No. 2, Beside Gokul Chat, Koti, Hyderabad.

Maternity Hospital Opp. Lane, Narayan Naik Complex, Koti, Hyderabad.

Near Andhra Bank, Subway, Sultan Bazar, Koti, Hyderabad -195.

C O N T E N T S

DATA SCIENCE USING PYTHON

SYLLABUS

UNIT - I

Introduction to data science – Introduction to data science, Data Science Components, Data Science Process, Data Science Jobs Roles, Tools for Data Science, Difference between Data Science with BI (Business Intelligence), Applications of Data science, Challenges of Data science Technology.

Data analysis – Introduction to data analysis, Data Analysis Tools, Types of Data Analysis: Techniques and Methods, Data Analysis Process Introduction to Python, Python features, Python Interpreter, modes of Python Interpreter, Values and Data types, Variables, Key words, Identifiers, Statements.

UNIT - II

Expressions, Input & Output, Comments, Lines & Indentation, Quotations, Tuple assignment, Operators, Precedence of operators. Functions: Definition and use, Types of functions, Flow of execution, Parameters and Arguments, Modules. Conditionals: Conditional(if), Alternative(if-else), Chained Conditionals(if-elif-else), Nested conditionals; Iteration/Control statements: while, for, break, continue, pass; fruitful function vs void function, Parameters/Arguments, Return values, Variables scope(local, global), Function composition.

UNIT - III

Strings: Strings, String slices, Immutability, String functions & Methods, String module; List as array: Array, Methods of array.

Lists: List operations, List slices, List methods, List loops, Mutability, aliasing, Cloning list, List parameters; Tuple: Benefit of Tuple, Operations on Tuple, Tuple methods, Tuple assignment, Tuple as return value, Tuple as argument; Dictionaries: Operations on Dictionary, methods in Dictionary, Difference between List, Tuple and Dictionary; Advanced List processing: List comprehension, Nested List.

UNIT - IV

Introduction to Numpy – The basics of numpy array, computation on numpy arrays, aggregations, computations on arrays, comparisons, masks and Boolean logic, fancy indexing, sorting arrays, structured data.

UNIT - V

Data Manipulation with Pandas – Introducing pandas objects, data indexing and selection, operating on data in pandas, handling missing data, hierarchical indexing, combining datasets, aggregation and grouping.

Contents

Important Questions

UNIT - I

1. **What is Data Science? Explain the steps involved in data science processing.**

Ans :

Refer Unit-I, Q.No. 1

2. **What are the main components of Data science?**

Ans :

Refer Unit-I, Q.No. 2

3. **Explain various processes of data science, what are used to extract information.**

Ans :

Refer Unit-I, Q.No. 3

4. **Write the differences between data science with business intelligence.**

Ans :

Refer Unit-I, Q.No. 6

5. **Explain about various tools used for data analysis.**

Ans :

Refer Unit-I, Q.No. 10

6. **Explain about the various phases of data analysis process.**

Ans :

Refer Unit-I, Q.No. 12

7. **Explain various modes of Python Interpreter.**

Ans :

Refer Unit-I, Q.No. 16

8. **Explain about standard data types used in python with an examples.**

Ans :

Refer Unit-I, Q.No. 18

9. **Write about various types of variables ?**

Ans :

Refer Unit-I, Q.No. 20

UNIT - II

1. Write about Tuple assignment feature?

Ans :

Refer Unit-II, Q.No. 7

2. What are the various types of operators used in python.

Ans :

Refer Unit-II, Q.No. 8

3. What is function? How to define and call a function?

Ans :

Refer Unit-II, Q.No. 11

4. Explain about various types of functions in Python.

Ans :

Refer Unit-II, Q.No. 12

5. What are User-Defined Functions in Python? Write about them?

Ans :

Refer Unit-II, Q.No. 13

6. Explain the Flow of Execution in Python.

Ans :

Refer Unit-II, Q.No. 14

7. Write about how to define a parameters in python.

Ans :

Refer Unit-II, Q.No. 15

8. What are Modules in Python? Explain.

Ans :

Refer Unit-II, Q.No. 16

9. Explain if-elif-else statement with syntax and example.

Ans :

Refer Unit-II, Q.No. 19

10. Explain while loop with syntax and example.

Ans :

Refer Unit-II, Q.No. 22

UNIT - III

1. What is string? and how do you create string?

Ans :

Refer Unit-III, Q.No. 1

2. Explain various String Manipulation Functions.

Ans :

Refer Unit-III, Q.No. 6

3. Define array? Explain about array operations

Ans :

Refer Unit-III, Q.No. 8

4. Explain various array methods in Python.

Ans :

Refer Unit-III, Q.No. 9

5. How to access elements from a list?

Ans :

Refer Unit-III, Q.No. 11

6. Write about various methods used in lists.

Ans :

Refer Unit-III, Q.No. 13

7. Explain how do you perform iterations in loops.

Ans :

Refer Unit-III, Q.No. 14

8. What is tuple in python? What are its advantages

Ans :

Refer Unit-III, Q.No. 19

UNIT - IV

1. What is NumPy? Explain how to create arrays in python using Numpy.

Ans :

Refer Unit-IV, Q.No. 1

2. Explain array creation techniques in Numpy with an example program.

Ans :

Refer Unit-IV, Q.No. 2

3. Explain various operations that can be performed on Numpy Arrays.

Ans :

Refer Unit-IV, Q.No. 4

4. Explain the concept of aggregations in Numpy.

Ans :

Refer Unit-IV, Q.No. 5

5. Expalin various arithmetic operation that can be performed on Numpy.

Ans :

Refer Unit-IV, Q.No. 6

6. Explain briefly about masks array module in Numpy.

Ans :

Refer Unit-IV, Q.No. 8

UNIT - V

1. Explain, how to create and use series in Pandas.

Ans :

Refer Unit-V, Q.No. 2

2. Expalin , how to use frames in Pandas.

Ans :

Refer Unit-V, Q.No. 4

3. Explain about indexing in Pandas.

Ans :

Refer Unit-V, Q.No. 5

4. Explain various operations that can perform on Pandas Data Frames.

Ans :

Refer Unit-V, Q.No. 7

5. Explain, how to handle the missing data.

Ans :

Refer Unit-V, Q.No. 8

6. Explain, how to Combine the data frames in Panda Using Merge() Function.

Ans :

Refer Unit-V, Q.No. 10

UNIT I

Introduction to data science – Introduction to data science, Data Science Components, Data Science Process, Data Science Jobs Roles, Tools for Data Science, Difference between Data Science with BI (Business Intelligence), Applications of Data science, Challenges of Data science Technology.

Data analysis – Introduction to data analysis, Data Analysis Tools, Types of Data Analysis: Techniques and Methods, Data Analysis Process Introduction to Python, Python features, Python Interpreter, modes of Python Interpreter, Values and Data types, Variables, Key words, Identifiers, Statements.

1.1 Introduction to Data Science

1.1.1 Introduction To Data Science

Q1. What is Data Science? Explain the steps involved in data science processing.

Ans : **(Imp.)**

Meaning

Data Science involves obtaining meaningful information or insights from structured or unstructured data through a process of analyzing, programming and business skills. It is a field containing many elements like mathematics, statistics, computer science, etc.

Process

Data science is not a one-step process. Every step has its value and it counts in your model.

- **Problem Statement:** No work start without motivation, Data science is no exception though. It's really important to declare or formulate your problem statement very clearly and precisely. Your whole model and it's working depend on your statement. Many scientist considers this as the main and much important step of Date Science. So make sure what's your problem statement and how well can it add value to business or any other organization.
- **Data Collection:** After defining the problem statement, the next obvious step is to go in search of data that you might require for your model. You must do good research, find all that you need. Data can be in any form i.e unstructured or structured. It might be in various forms like videos, spreadsheets, coded forms, etc. You must collect all these kinds of sources.
- **Data Cleaning:** As you have formulated your motive and also you did collect your data, the next step to do is cleaning. Data cleaning is all about the removal of missing, redundant, unnecessary and duplicate data from your collection. There are various tools to do so with the help of programming in either R or Python.
- **Data Analysis and Exploration:** It's one of the prime things in data science to do and time to get inner Holmes out. It's about analyzing the structure of data, finding hidden patterns in them, studying behaviors, visualizing the effects of one variable over others and then concluding. We can explore the data with the help of various graphs formed with the help of libraries using any programming language.
- **Data Modelling:** Once you are done with your study that you have formed from data visualization, you must start building a hypothesis model such that it may yield you a good prediction in future. Here, you must choose a good algorithm that best fit to your model. There different kinds of algorithms from regression to classification, SVM(Support vector machines), Clustering, etc.
- **Optimization and Deployment:** You followed each and every step and hence build a model that you feel is the best fit. You test your data and find how well it is performing by checking its accuracy. In short, you check the efficiency of the data model and thus try to optimize it for better accurate prediction. Deployment deals with the launch of your model and let the people outside there to benefit from that.

1.1.2 Data Science Components

Q2. What are the main components of Data science?

Ans : **(Imp.)**

The main components of Data Science are:

1. **Statistics:** The essential component of Data Science is Statistics. It is a method to collect and analyze the numerical data in a large amount to get useful and meaningful insights.

   There are two main categories of Statistics:

   (i) **Descriptive Statistics:** Descriptive Statistics helps to organize data and only focuses on the characteristics of data providing parameters. For example, you want to find the average height of students in a classroom, in descriptive statistics, you will record the heights of all students in the class, and then you would find the maximum, minimum and average height of the class.

   (ii) **Inferential Statistics:** Inferential statistics generalizes a large data set and applies probability before concluding. It also allows you to infer parameters of the population based on sample stats and build models on it. For example, if we consider the same example of finding the average height of students in a class, then in Inferential Statistics, you will take a sample set of the class, basically a few people from the entire class.

2. **Visualization:** Visualization means representing the data in visuals such as maps, graphs, etc. so that people can understand it easily. It makes it easy to access a vast amount of data. The main goal of data visualization is to make it easier to identify patterns, trends, and outliers in large data sets. The main benefits of data visualization include:

   ➢ It can absorb information quickly, improve insights, and make faster decisions.

   ➢ It increases understanding of the next steps that must be taken to improve the organization.

   ➢ It provides an improved ability to maintain the audience's interest with the information they can understand.

   ➢ It gives an easy distribution of information that increases the opportunity to share insights with everyone involved.

   ➢ It eliminates the need for data scientists since data is more accessible and understandable.

   ➢ It increases the ability to act on findings quickly and, therefore, achieve success with higher speed and fewer mistakes.

3. **Machine Learning:** Machine Learning acts as a backbone for data science. It means providing training to a machine in such a way that it acts as a human brain. Various algorithms are used to solve the problems. With the help of Machine Learning, it becomes easy to make predictions about unforeseen/future data.

   Machine Learning makes a prediction, analysis patterns, and gives recommendations and is frequently used in fraud detection and client retention.

   For example, a social media platform, i.e., Facebook, where fast algorithms are used to collect the behavioral information of every user available on social media and also recommend them the relevant articles, multimedia files, and much more based on their choice.

   There are four types of Machine learning:

(i) **Supervised Machine Learning:** In this type of machine learning, the machine mainly focuses on regression and classification problems. We already know the correct output and relationship with input and output in this phase. It also deals with labeled datasets and algorithms, and the machine gets the last calculated data on the machine, also known as target data. It includes the data as well as a result. There are two major processes:

   ➢ **Classification:** It is the process in which the input data is labeled based on past data experiences. The machines are also trained with algorithms about the data format, and the algorithms specify the format to recognize by the machine. The examples of classification are weather forecasting and specifying whether tomorrow will be hot or cold. Naive Bayes, Support Vector Machine and Decision Tree are the most popular supervised machine learning algorithms.

   ➢ **Regression:** It is the process to identify the labeled data and calculate the results based on prediction. The machine can learn the data and display real-valued results. These results are based on independent values. For example, a human picture is given to a common man to identify the gender of the person in the image. Another example is the prediction of the temperature of tomorrow based on past data. Linear regression is used for regression problems.

(ii) Unsupervised Machine Learning: Here, the results are unknown and need to be defined. It uses unlabeled data for machine learning, and we have no idea about the types of results. The machine observes the algorithms and then finds the structure of data and has less computational complexity and uses real-time analysis of data. The results are very reliable compared to supervised learning. For example, we can present images of fruits to this model, and this model makes clusters and separates them based on a given pattern and relationships.

There are two types:

- **Clustering**: In clustering, data is found in segments and meaningful groups. It is based in small groups. These groups have their patterns through which data is arranged and segmented. K-means clustering, hierarchical clustering, and density-based spatial clustering are more popular clustering algorithms.
- **Dimensionality Reduction:** The unnecessary data is removed to summaries the distribution of data in groups in this phase.

(iii) Semi-Supervised Machine Learning: Semi-supervised machine learning, also known as hybrid learning, and it lies between supervised and unsupervised learning. This model has a combination of labeled and unlabeled data. This data has fewer shares of labeled data and more shares of unlabeled data. The labeled-data is very cheap in comparison to the unlabeled data. The procedure is that the algorithm uses unsupervised learning algorithms to cluster the labeled data and then uses the supervised learning algorithm.

(iv) Reinforcement Learning: In this learning, there are no training data sets. The machine has special software that works as an agent with the environment to get feedback. The work of an agent is to achieve the target and get the required feedback. An example of a reinforcement learning problem is playing games, in which an agent has a set of goals to get high scores and feedback in terms of punishment and rewards while playing.

1.1.3 Data Science Process

Q3. Explain various processes of data science, what are used to extract information.

Ans : **(Imp.)**

Different processes are included to infer the information from the source like extraction of data, information preparation, model planning, model building and many more.

- **Discovery**

  To begin with, it is exceptionally imperative to get the different determinations, prerequisites, needs and required budget-related with the venture. You must have the capacity to inquire the correct questions like do you have got the desired assets. These assets can be in terms of individuals, innovation, time and information. In this stage, you too got to outline the trade issue and define starting hypotheses (IH) to test.

- **Information Preparation**

  In this stage, you would like to investigate, preprocess and condition data for modeling. You'll be able to perform information cleaning, changing, and visualization. This will assist you to spot the exceptions and build up a relationship between the factors. Once you have got cleaned and arranged the information, it's time to do exploratory analytics on it.

- **Model Planning**

  Here, you may decide the strategies and methods to draw the connections between factors. These connections will set the base for the calculations which you may execute within the following stage. You may apply Exploratory Data Analytics (EDA) utilizing different factual equations and visualization apparatuses.

- **Model Building**

  In this stage, you'll create datasets for training and testing purposes. You may analyze different learning procedures like classification, association, and clustering and at last, actualize the most excellent fit technique to construct the show.

- **Operationalize**

  In this stage, you convey the last briefings, code, and specialized reports. In expansion, now a pilot venture is additionally actualized in a real-time generation environment. This will give you a clear picture of the execution and other related limitations.

➢ **Communicate Results**

Presently, it is critical to assess the outcome of the objective. So, within the final stage, you recognize all the key discoveries, communicate to the partners and decide in the event that the outcomes about the venture are a victory or a disappointment based on the criteria created in Stage 1.

1.1.4 Data Science Jobs Roles

Q4. What are the important job roles in data sceince?

Ans :

Most prominent Data Scientist job titles are:

➢ Data Scientist

➢ Data Engineer

➢ Data Analyst

➢ Statistician

➢ Data Architect

➢ Data Admin

➢ Business Analyst

➢ Data/Analytics Manager

Data Scientist

➢ **Role:** A Data Scientist is a professional who manages enormous amounts of data to come up with compelling business visions by using various tools, techniques, methodologies, algorithms, etc.

➢ **Languages**: R, SAS, Python, SQL, Hive, Matlab, Pig, Spark

Data Engineer

➢ **Role**: The role of a data engineer is of working with large amounts of data. He develops, constructs, tests, and maintains architectures like large scale processing systems and databases.

➢ **Languages**: SQL, Hive, R, SAS, Matlab, Python, Java, Ruby, C + +, and Perl

Data Analyst

➢ **Role:** A data analyst is responsible for mining vast amounts of data. They will look for relationships, patterns, trends in data. Later he or she will deliver compelling reporting and visualization for analyzing the data to take the most viable business decisions.

➢ **Languages:** R, Python, HTML, JS, C, C++, SQL

Statistician

➢ **Role**: The statistician collects, analyses, and understands qualitative and quantitative data using statistical theories and methods.

➢ **Languages**: SQL, R, Matlab, Tableau, Python, Perl, Spark, and Hive

Data Administrator

➢ **Role:** Data admin should ensure that the database is accessible to all relevant users. He also ensures that it is performing correctly and keeps it safe from hacking.

➢ **Languages:** Ruby on Rails, SQL, Java, C#, and Python

Business Analyst

➢ **Role:** This professional needs to improve business processes. He/she is an intermediary between the business executive team and the IT department.

➢ **Languages:** SQL, Tableau, Power BI and, Python

1.1.5 Tools for Data Science

Q5. Write about the tools used for data science.

Ans :

The following are the various tools for data science

1. SAS

It is one of those data science tools which are specifically designed for statistical operations. SAS is a closed source proprietary software that is used by large organizations to analyze data. SAS uses base SAS programming language which for performing statistical modeling.

It is widely used by professionals and companies working on reliable commercial software. SAS offers numerous statistical libraries and tools that you as a Data Scientist can use for modeling and organizing their data.

2. **Apache Spark**

Apache Spark or simply Spark is an all-powerful analytics engine and it is the most used Data Science tool. Spark is specifically designed to handle batch processing and Stream Processing.

3. **BigML**

BigML, it is another widely used Data Science Tool. It provides a fully interactable, cloud-based GUI environment that you can use for processing Machine Learning Algorithms. BigML provides standardized software using cloud computing for industry requirements.

4. **MATLAB**

MATLAB is a multi-paradigm numerical computing environment for processing mathematical information. It is a closed-source software that facilitates matrix functions, algorithmic implementation and statistical modeling of data. MATLAB is most widely used in several scientific disciplines.

In Data Science, MATLAB is used for simulating neural networks and fuzzy logic. Using the MATLAB graphics library, you can create powerful visualizations. MATLAB is also used in image and signal processing.

5. **Excel**

Probably the most widely used Data Analysis tool. Microsoft developed Excel mostly for spreadsheet calculations and today, it is widely used for data processing, visualization, and complex calculations.

Excel is a powerful analytical tool for Data Science. While it has been the traditional tool for data analysis, Excel still packs a punch.

6. **Tableau**

Tableau is a Data Visualization software that is packed with powerful graphics to make interactive visualizations. It is focused on industries working in the field of business intelligence.

The most important aspect of Tableau is its ability to interface with databases, spreadsheets, OLAP (Online Analytical Processing) cubes, etc. Along with these features, Tableau has the ability to visualize geographical data and for plotting longitudes and latitudes in maps.

7. **Jupyter**

Project Jupyter is an open-source tool based on IPython for helping developers in making open-source software and experiences interactive computing. Jupyter supports multiple languages like Julia, Python, and R.

8. **Matplotlib**

Matplotlib is a plotting and visualization library developed for Python. It is the most popular tool for generating graphs with the analyzed data. It is mainly used for plotting complex graphs using simple lines of code. Using this, one can generate bar plots, histograms, scatterplots etc.

9. **NLTK**

Natural Language Processing has emerged as the most popular field in Data Science. It deals with the development of statistical models that help computers understand human language.

10. **Scikit-learn**

Scikit-learn is a library-based in Python that is used for implementing Machine Learning Algorithms. It is simple and easy to implement a tool that is widely used for analysis and data science.

11. **TensorFlow**

TensorFlow has become a standard tool for Machine Learning. It is widely used for advanced machine learning algorithms like Deep Learning. Developers named TensorFlow after Tensors which are multidimensional arrays.

1.2 Difference between Data Science with BI (Business Intelligence)

Q6. Write the differences between data science with business intelligence.

Ans : **(Imp.)**

Below is a table of differences between Data Science and Business Intelligence:

Data Science	Business Intelligence
It is a field that uses mathematics, statistics and various other tools to discover the hidden patterns in the data.	It is basically a set of technologies, applications and processes that are used by the enterprises for business data analysis.
It focuses on the future.	It focuses on the past and present.
It deals with both structured as well as unstructured data.	It mainly deals only with structured data.
Data science is much more flexible as data sources can be added as per requirement.	It is less flexible as in case of business intelligence data sources need to be pre-planned.
It makes use of the scientific method.	It makes use of the analytic method.
It has a higher complexity in comparison to business intelligence.	It is much simpler when compared to data science.
It's expertise is data scientist.	It's expertise is the business user.
It deals with the questions of what will happen and what if.	It deals with the question of what happened.
The data to be used is disseminated in real-time clusters.	Data warehouse is utilized to hold data.
The ELT (Extract-Load-Transform) process is generally used for the integration of data for data science applications.	The ETL (Extract-Transform-Load) process is generally used for the integration of data for business intelligence applications.
It's tools are SAS, BigML, MATLAB, Excel, etc.	It's tools are InsightSquared Sales Analytics, Klipfolio, ThoughtSpot, Cyfe, TIBCO Spotfire, etc.
Companies can harness their potential by anticipating the future scenario using data science in order to reduce risk and increase income.	Business Intelligence helps in performing root cause analysis on a failure or to understand the current status.
Greater business value is achieved with data science in comparison to business intelligence as it anticipates future events.	Business Intelligence has lesser business value as the extraction process of business value carries out statically by plotting charts and KPIs (Key Performance Indicator).
The technologies such as Hadoop are available and others are evolving for handling understanding Its large data sets.	The sufficient tools and technologies are not available for handling large data sets.

1.3 Applications of Data Science

Q7. What are the application sof data science? Explain.

Ans :

The following are the Applications of Data Science

- **Internet Search:** Google search uses Data science technology to search for a specific result within a fraction of a second

- **Recommendation Systems:** To create a recommendation system. For example, "suggested friends" on Facebook or suggested videos" on YouTube, everything is done with the help of Data Science.
- **Image & Speech Recognition:** Speech recognizes systems like Siri, Google Assistant, and Alexa run on the Data science technique. Moreover, Facebook recognizes your friend when you upload a photo with them, with the help of Data Science.
- **Gaming world:** EA Sports, Sony, Nintendo are using Data science technology. This enhances your gaming experience. Games are now developed using Machine Learning techniques, and they can update themselves when you move to higher levels.
- **Online Price Comparison:** PriceRunner, Junglee, Shopzilla work on the Data science mechanism. Here, data is fetched from the relevant websites using APIs.

1.4 Challenges of Data Science Technology

Q8. What are the challenges of data science technology?

Ans :

- A high variety of information & data is required for accurate analysis
- Not adequate data science talent pool available
- Management does not provide financial support for a data science team
- Unavailability of/difficult access to data
- Business decision-makers do not effectively use data Science results
- Explaining data science to others is difficult
- Privacy issues
- Lack of significant domain expert
- If an organization is very small, it can't have a Data Science team

1.5 Data Analysis

1.5.1 Introduction to Data Analysis

Q9. What is data analysis?

Ans :

Meaning

Data analysis is defined as a process of cleaning, transforming, and modeling data to discover useful information for business decision-making. The purpose of Data Analysis is to extract useful information from data and taking the decision based upon the data analysis.

Example

A simple example of Data analysis is whenever we take any decision in our day-to-day life is by thinking about what happened last time or what will happen by choosing that particular decision. This is nothing but analyzing our past or future and making decisions based on it.

1.5.2 Data Analysis Tools

Q10. Explain about various tools used for data analysis.

Ans : **(Imp.)**

With the increasing demand for Data Analytics in the market, many tools have emerged with various functionalities for this purpose. Either open-source or user-friendly, the top tools in the data analytics market are as follows.

- **R programming:** This tool is the leading analytics tool used for statistics and data modeling. R compiles and runs on various platforms such as UNIX, Windows, and Mac OS. It also provides tools to automatically install all packages as per user-requirement.
- **Python:** Python is an open-source, object-oriented programming language that is easy to read, write, and maintain. It provides various machine learning and visualization libraries such as Scikit-learn, TensorFlow, Matplotlib, Pandas, Keras, etc. It also can be assembled on any platform like SQL server, a MongoDB database or JSON.
- **Tableau Public:** This is a free software that connects to any data source such as Excel, corporate Data Warehouse, etc. It then creates visualizations, maps, dashboards etc with real-time updates on the web.

- **QlikView:** This tool offers in-memory data processing with the results delivered to the end-users quickly. It also offers data association and data visualization with data being compressed to almost 10% of its original size.
- **SAS:** A programming language and environment for data manipulation and analytics, this tool is easily accessible and can analyze data from different sources.
- **Microsoft Excel:** This tool is one of the most widely used tools for data analytics. Mostly used for clients' internal data, this tool analyzes the tasks that summarize the data with a preview of pivot tables.
- **RapidMiner:** A powerful, integrated platform that can integrate with any data source types such as Access, Excel, Microsoft SQL, Tera data, Oracle, Sybase etc. This tool is mostly used for predictive analytics, such as data mining, text analytics, machine learning.
- **KNIME:** Konstanz Information Miner (KNIME) is an open-source data analytics platform, which allows you to analyze and model data. With the benefit of visual programming, KNIME provides a platform for reporting and integration through its modular data pipeline concept.
- **OpenRefine:** Also known as GoogleRefine, this data cleaning software will help you clean up data for analysis. It is used for cleaning messy data, the transformation of data and parsing data from websites.
- **Apache Spark:** One of the largest large-scale data processing engine, this tool executes applications in Hadoop clusters 100 times faster in memory and 10 times faster on disk. This tool is also popular for data pipelines and machine learning model development.

1.6 Types of Data Analysis: Techniques and Methods

Q11. Explain about various types of data analysis techniques and methods.

Ans :

There are several types of Data Analysis techniques that exist based on business and technology. However, the major Data Analysis methods are:

- **Text Analysis**

  Text Analysis is also referred to as Data Mining. It is one of the methods of data analysis to discover a pattern in large data sets using databases or data mining tools. It used to transform raw data into business information. Business Intelligence tools are present in the market which is used to take strategic business decisions. Overall it offers a way to extract and examine data and deriving patterns and finally interpretation of the data.

- **Statistical Analysis**

  Statistical Analysis shows "What happen?" by using past data in the form of dashboards. Statistical Analysis includes collection, Analysis, interpretation, presentation, and modeling of data. It analyses a set of data or a sample of data. There are two categories of this type of Analysis – Descriptive Analysis and Inferential Analysis.

- **Descriptive Analysis**

  Descriptive analyses complete data or a sample of summarized numerical data. It shows mean and deviation for continuous data whereas percentage and frequency for categorical data.

- **Inferential Analysis**

  analyses sample from complete data. In this type of Analysis, you can find different conclusions from the same data by selecting different samples.

- **Diagnostic Analysis**

  Diagnostic Analysis shows "Why did it happen?" by finding the cause from the insight found in Statistical Analysis. This Analysis is useful to identify behavior patterns of data. If a new problem arrives in your business process, then you can look into this Analysis to find similar patterns of that problem. And it may have chances to use similar prescriptions for the new problems.

- **Predictive Analysis**

  Predictive Analysis shows "what is likely to happen" by using previous data. The simplest data analysis example is like if last year I bought two dresses based on my savings and if this year my salary is increasing double then I can buy four dresses. So here, this Analysis makes predictions about future outcomes based on current or past data. Forecasting is just an estimate. Its accuracy is based on how much detailed information you have and how much you dig in it.

➢ **Prescriptive Analysis**

Prescriptive Analysis combines the insight from all previous Analysis to determine which action to take in a current problem or decision. Most data-driven companies are utilizing Prescriptive Analysis because predictive and descriptive Analysis are not enough to improve data performance. Based on current situations and problems, they analyze the data and make decisions.

1.6.1 Data Analysis Process

Q12. Explain about the various phases of data analysis process.

Ans : **(Imp.)**

The Data Analysis Process is nothing but gathering information by using a proper application or tool which allows you to explore the data and find a pattern in it. Based on that information and data, you can make decisions, or you can get ultimate conclusions.

Data Analysis consists of the following phases:

1. Data Requirement Gathering
2. Data Collection
3. Data Cleaning
4. Data Analysis
5. Data Interpretation
6. Data Visualization

1. Data Requirement Gathering

First of all, you have to think about why do you want to do this data analysis? All you need to find out the purpose or aim of doing the Analysis of data. You have to decide which type of data analysis you wanted to do! In this phase, you have to decide what to analyze and how to measure it, you have to understand why you are investigating and what measures you have to use to do this Analysis.

2. Data Collection

After requirement gathering, you will get a clear idea about what things you have to measure and what should be your findings. Now it's time to collect your data based on requirements. Once you collect your data, remember that the collected data must be processed or organized for Analysis. As you collected data from various sources, you must have to keep a log with a collection date and source of the data.

3. Data Cleaning

Now whatever data is collected may not be useful or irrelevant to your aim of Analysis, hence it should be cleaned. The data which is collected may contain duplicate records, white spaces or errors. The data should be cleaned and error free. This phase must be done before Analysis because based on data cleaning, your output of Analysis will be closer to your expected outcome.

4. Data Analysis

Once the data is collected, cleaned, and processed, it is ready for Analysis. As you manipulate data, you may find you have the exact information you need, or you might need to collect more data. During this phase, you can use data analysis tools and software which will help you to understand, interpret, and derive conclusions based on the requirements.

5. Data Interpretation

After analyzing your data, it's finally time to interpret your results. You can choose the way to express or communicate your data analysis either you can use simply in words or maybe a table or chart. Then use the results of your data analysis process to decide your best course of action.

6. Data Visualization

Data visualization is very common in your day to day life; they often appear in the form of charts and graphs. In other words, data shown graphically so that it will be easier for the human brain to understand and process it. Data visualization often used to discover unknown facts and trends. By observing relationships and comparing datasets, you can find a way to find out meaningful information.

1.7 Introduction To Python

Q13. What is python? What are the applications of python?

Ans :

Python is a high-level object-oriented programming language that was created by Guido van Rossum. It is also called general-purpose programming language as it is used in almost every domain we can think of as mentioned below:

- Web Development
- Software Development
- Game Development
- AI & ML
- Data Analytics
- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
- Python has a simple syntax similar to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a procedural way, an object-oriented way or a functional way.

1.7.1 Python Features

Q14. What are the features of python?

Ans :

- **Easy-to-learn:** Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
- **Easy-to-read:** Python code is more clearly defined and visible to the eyes.
- **Easy-to-maintain:** Python's source code is fairly easy-to-maintain.
- **A broad standard library:** Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- **Interactive Mode:** Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- **Portable:** Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- **Extendable:** You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- **Databases:** Python provides interfaces to all major commercial databases.
- **GUI Programming:** Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
- **Scalable:** Python provides a better structure and support for large programs than shell scripting.

Apart from the above-mentioned features, Python has a big list of good features, few are listed below:

- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- IT supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

1.7.2 Python Interpreter

Q15. Explain about Python Interpreter.

Ans :

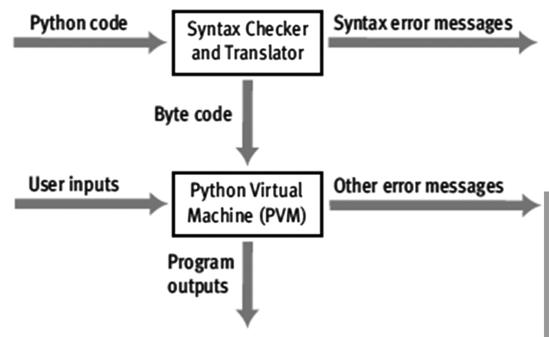
An interpreter is a kind of program that executes other programs. When you write Python programs, it converts source code written by the developer into intermediate language which is again translated into the native language / machine language that is executed.

1. The interpreter reads a Python expression or statement, also called the source code, and verifies that it is well formed. In this step, the interpreter behaves like a strict English teacher who rejects any sentence that does not adhere to the grammar

rules, or syntax, of the language. As soon as the interpreter encounters such an error, it halts translation with an error message.

2. If a Python expression is well formed, the interpreter then translates it to an equivalent form in a low-level language called byte code. When the interpreter runs a script, it completely translates it to byte code.

3. This byte code is next sent to another software component, called the Python virtual machine (PVM), where it is executed. If another error occurs during this step, execution also halts with an error message.



Steps in interpreting a Python program

1.7.3 Modes of Python Interpreter

Q16. Explain various modes of Python Interpreter.

Ans : **(Imp.)**

Python Interpreter is a program that reads and executes Python code. It uses 2 modes of Execution.

1. Interactive mode
2. Script mode

1. Interactive Mode

- ➢ Interactive Mode, as the name suggests, allows us to interact with OS.
- ➢ When we type Python statement, interpreter displays the result(s) immediately.

Advantages

- ➢ Python, in interactive mode, is good enough to learn, experiment or explore.
- ➢ Working in interactive mode is convenient for beginners and for testing small pieces of code.

Drawback

- ➢ We cannot save the statements and have to retype all the statements once again to re-run them.
- ➢ In interactive mode, you type Python programs and the interpreter displays the result:

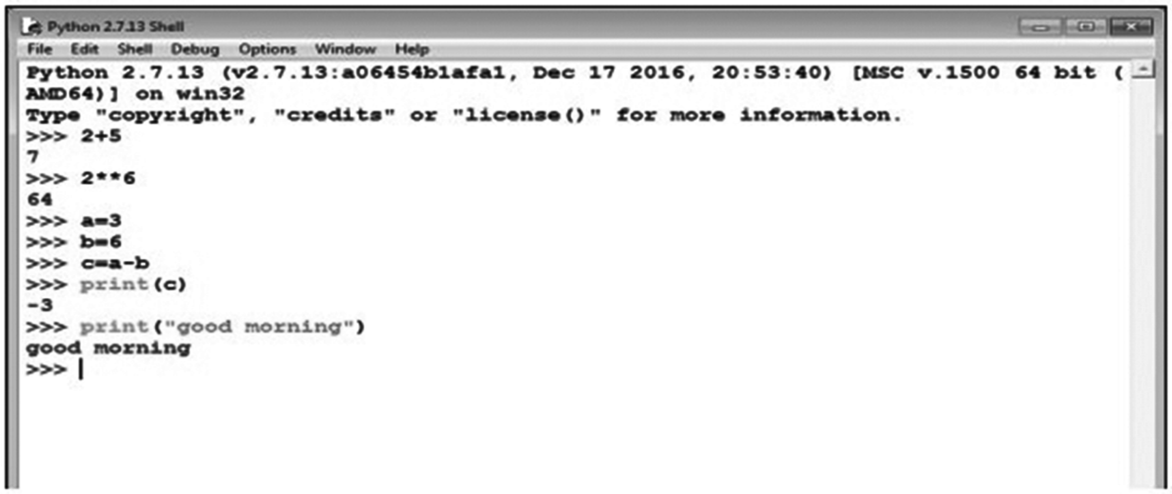
```
>>>  1 + 1
2
```

The chevron, >>>, is the prompt the interpreter uses to indicate that it is ready for you to enter code. If you type 1 + 1, the interpreter replies 2.

```
>>>  print ('Hello, World!')
```

Hello, World!

This is an example of a print statement. It displays a result on the screen. In this case, the result is the words.



```
Python 2.7.13 (v2.7.13:a06454b1afa1, Dec 17 2016, 20:53:40) [MSC v.1500 64 bit (
AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 2+5
7
>>> 2**6
64
>>> a=3
>>> b=6
>>> c=a-b
>>> print(c)
-3
>>> print("good morning")
good morning
>>> |
```

2. Script Mode

- In script mode, we type python program in a file and then use interpreter to execute the content of the file.
- Scripts can be saved to disk for future use. Python scripts have the extension .py, meaning that the filename ends with .py

Example 1 :

```
print (1)
x = 2
print (x)
```

Output:

```
>>>1
   2
```

Q17. State the differences between Interactive Mode and Script Mode.

Ans :

S.No.	Interactive Mode	Script Mode
1.	A way of using the Python interpreter by typing commands and expressions at the prompt.	A way of using the Python interpreer to read and execute statements in a script.
2.	Cant save and edit the code.	Can save and edit the code.
3.	If we want to experiment with the code, we can use interactive mode.	If we are very clear about the code, we can use script mode.
4.	We cannot save the statements for further use and we have to retype all the statements to re-run them.	We can save the statements for further use and we no need to retype all the statements to re-run them.
5.	We can see the resuls immediately.	We cant see the code immediately.

1.7.4 Values and Data Types

Q18. Explain about standard data types used in python with an examples.

Ans : **(Imp.)**

Standard Data Types

The data stored in memory can be of many types. For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters. Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.

Python has five standard data types -

1. Python Numbers

Number data types store numeric values. Number objects are created when you assign a value to them. For example -

```
var1 =1
var2 =10
```

You can also delete the reference to a number object by using the del statement. The syntax of the del statement is -

```
del var1[,var2[,var3[....,varN]]]]
```

You can delete a single object or multiple objects by using the del statement. For example -

```
delvar
del var_a, var_b
```

Python supports four different numerical types -

- int (signed integers)
- long (long integers, they can also be represented in octal and hexadecimal)
- float (floating point real values)
- complex (complex numbers)

Examples

Here are some examples of numbers -

int	Long	Float	complex
10	51924361L	0.0	3.14j
100	-0x19323L	15.20	45.j
-786	0122L	-21.9	9.322e-36j
080	0xDEFABCECBDAECBFBAEl	32.3+e18	.876j
-0490	535633629843L	-90.	-.6545+0J
-0x260	-052318172735L	-32.54e100	3e+26J
0x69	-4721885298529L	70.2-E12	4.53e-7j

- Python allows you to use a lowercase l with long, but it is recommended that you use only an uppercase L to avoid confusion with the number 1. Python displays long integers with an uppercase L.
- A complex number consists of an ordered pair of real floating-point numbers denoted by x + yj, where x and y are the real numbers and j is the imaginary unit.

2. Python Strings

Strings in Python are identified as a contiguous set of characters represented in the quotation marks. Python allows for either pairs of single or double quotes. Subsets of strings can be taken using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.

The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator. For example -

```
#!/usr/bin/python
str = 'Hello World!'
print str          # Prints complete string
print str[0]# Prints first character of the string
print str[2:5]# Prints characters starting from 3rd to 5th
print str[2:]# Prints string starting from 3rd character
print str *2# Prints string two times
print str +"TEST"# Prints concatenated string
```

This will produce the following result "

```
Hello World!
H
llo
llo World!
Hello World!Hello World!
Hello World!TEST
```

3. Python Lists

Lists are the most versatile of Python's compound data types. A list contains items separated by commas and enclosed within square brackets ([]). To some extent, lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data type.

The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1. The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator. For example "

```
#!/usr/bin/python
list =['abcd',786,2.23,'john',70.2]
tinylist =[123,'john']
```

```
print list          # Prints complete list
print list[0]# Prints first element of the list
print list[1:3]# Prints elements starting from 2nd till 3rd
print list[2:]# Prints elements starting from 3rd element
print tinylist *2# Prints list two times
print list + tinylist # Prints concatenated lists
```

This produce the following result -

```
['abcd', 786, 2.23, 'john', 70.200000000000003]
abcd
[786, 2.23]
[2.23, 'john', 70.200000000000003]
[123, 'john', 123, 'john']
['abcd', 786, 2.23, 'john', 70.200000000000003, 123, 'john']
```

4. Python Tuples

A tuple is another sequence data type that is similar to the list. A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.

The main differences between lists and tuples are: Lists are enclosed in brackets ([ ]) and their elements and size can be changed, while tuples are enclosed in parentheses (( )) and cannot be updated. Tuples can be thought of as read-only lists. For example -

```
#!/usr/bin/python
tuple =('abcd',786,2.23,'john',70.2)
tinytuple =(123,'john')
print tuple          # Prints complete list
print tuple[0]# Prints first element of the list
print tuple[1:3]# Prints elements starting from 2nd till 3rd
print tuple[2:]# Prints elements starting from 3rd element
print tinytuple *2# Prints list two times
print tuple + tinytuple # Prints concatenated lists
```

This produce the following result -

```
('abcd', 786, 2.23, 'john', 70.200000000000003)
abcd
(786, 2.23)
(2.23, 'john', 70.200000000000003)
(123, 'john', 123, 'john')
('abcd', 786, 2.23, 'john', 70.200000000000003, 123, 'john')
```

The following code is invalid with tuple, because we attempted to update a tuple, which is not allowed. Similar case is possible with lists -

```
#!/usr/bin/python
tuple = ('abcd',786,2.23,'john',70.2)
list = ['abcd',786,2.23,'john',70.2]
tuple[2] = 1000# Invalid syntax with tuple
list[2] = 1000# Valid syntax with list
```

5. Python Dictionary

Python's dictionaries are kind of hash table type. They work like associative arrays or hashes found in Perl and consist of key-value pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.

Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]). For example -

```
#!/usr/bin/python
dict = {}
dict['one'] = "This is one"
dict[2] = "This is two"
tinydict = {'name':'john','code':6734,'dept':'sales'}
print dict['one']# Prints value for 'one' key
print dict[2]# Prints value for 2 key
print tinydict          # Prints complete dictionary
print tinydict.keys()# Prints all the keys
print tinydict.values()# Prints all the values
```

This produce the following result -

```
This is one
This is two
    {'dept': 'sales', 'code': 6734, 'name': 'john'}
    ['dept', 'code', 'name']
    ['sales', 6734, 'john']
```

Dictionaries have no concept of order among elements. It is incorrect to say that the elements are "out of order"; they are simply unordered.

1.7.5 Variables

Q19. Explain about Python variables.

Ans :

- Variable is a name that is used to refer to memory location. Python variable is also known as an identifier and used to hold value.
- In Python, we don't need to specify the type of variable because Python is a infer language and smart enough to get variable type.

- Variable names can be a group of both the letters and digits, but they have to begin with a letter or an underscore.
- It is recommended to use lowercase letters for the variable name. Rahul and rahul both are two different variables.

Identifier Naming

Variables are the example of identifiers. An Identifier is used to identify the literals used in the program. The rules to name an identifier are given below.

- The first character of the variable must be an alphabet or underscore (_).
- All the characters except the first character may be an alphabet of lower-case(a-z), upper-case (A-Z), underscore, or digit (0-9).
- Identifier name must not contain any white-space, or special character (!, @, #, %, ^ , &, *).
- Identifier name must not be similar to any keyword defined in the language.
- Identifier names are case sensitive; for example, my name, and MyName is not the same.
- Examples of valid identifiers: a123, _n, n_9, etc.
- Examples of invalid identifiers: 1a, n%4, n 9, etc.

Declaring Variable and Assigning Values

Python does not bind us to declare a variable before using it in the application. It allows us to create a variable at the required time.

We don't need to declare explicitly variable in Python. When we assign any value to the variable, that variable is declared automatically.

The equal (=) operator is used to assign value to a variable.

Object References

It is necessary to understand how the Python interpreter works when we declare a variable. The process of treating variables is somewhat different from many other programming languages.

Python is the highly object-oriented programming language; that's why every data item belongs to a specific type of class. Consider the following example.

```
print("John")
```

Output

John

The Python object creates an integer object and displays it to the console. In the above print statement, we have created a string object. Let's check the type of it using the Python built-in type() function.

```
type("John")
```

Output:

```
<class 'str'>
```

In Python, variables are a symbolic name that is a reference or pointer to an object. The variables are used to denote objects by that name.

Let's understand the following example

```
a = 50
```

$$\begin{tikzcd} a \arrow[r] & \boxed{50} \end{tikzcd}$$

In the above image, the variable **a** refers to an integer object.

Suppose we assign the integer value 50 to a new variable b.

```
a = 50
b = a
```

$$\begin{tikzcd} a \arrow[r] & \boxed{50} & b \arrow[l] \end{tikzcd}$$

The variable b refers to the same object that a points to because Python does not create another object.

Let's assign the new value to b. Now both variables will refer to the different objects.

```
a = 50
b =100
```

$$\begin{tikzcd} a \arrow[r] & \boxed{50} \\ b \arrow[r] & \boxed{100} \end{tikzcd}$$

Python manages memory efficiently if we assign the same variable to two different values.

Variable Names

Variable names can be any length can have uppercase, lowercase (A to Z, a to z), the digit (0-9), and underscore character(_). Consider the following example of valid variables names.

```
name = "Devansh"
age = 20
marks = 80.50
 print(name)
print(age)
print(marks)
```

Output:

```
Devansh
20
80.5
```

Consider the following valid variables name.

```
name = "A"
Name = "B"
naMe = "C"
NAME = "D"
n_a_m_e = "E"
_name = "F"
name_ = "G"
_name_ = "H"
na56me = "I"
```

print(name,Name,naMe,NAME,n_a_m_e, NAME, n_a_m_e, _name, name_,_name, na56me)

Output:

```
A B C D E D E F G F I
```

In the above example, we have declared a few valid variable names such as name, _name_ , etc. But it is not recommended because when we try to read code, it may create confusion. The variable name should be descriptive to make code more readable.

The multi-word keywords can be created by the following method.

- **Camel Case:** In the camel case, each word or abbreviation in the middle of begins with a capital letter. There is no intervention of whitespace. For example - nameOfStudent, valueOfVaraible, etc.
- **Pascal Case:** It is the same as the Camel Case, but here the first word is also capital. For example - NameOfStudent, etc.
- **Snake Case:** In the snake case, Words are separated by the underscore. For example - name_of_student, etc.

Multiple Assignment

Python allows us to assign a value to multiple variables in a single statement, which is also known as multiple assignments.

We can apply multiple assignments in two ways, either by assigning a single value to multiple variables or assigning multiple values to multiple variables. Consider the following example.

1. Assigning single value to multiple variables

Eg:

```
x=y=z=50
print(x)
print(y)
print(z)
```

Output:

```
50
50
50
```

2. Assigning multiple values to multiple variables:

Eg:

```
a,b,c=5,10,15
print a
print b
print c
```

Output:

```
5
10
15
```

The values will be assigned in the order in which variables appear.

Delete a variable

We can delete the variable using the del keyword. The syntax is given below.

Syntax -

del <variable_name>

In the following example, we create a variable x and assign value to it. We deleted variable x, and print it, we get the error "variable x is not defined". The variable x will no longer use in future.

Print Single and Multiple Variables in Python

We can print multiple variables within the single print statement. Below are the example of single and multiple printing values.

Example - 1 (Printing Single Variable)

```
# printing single value
a = 5
print(a)
print((a))
```

Output

```
5
5
```

Example - 2 (Printing Multiple Variables)

```
a = 5
b = 6
# printing multiple variables
print(a,b)
# separate the variables by the comma
Print(1, 2, 3, 4, 5, 6, 7, 8)
```

Output

```
5 6
1 2 3 4 5 6 7 8
```

Q20. Write about various types of variables ?

Ans : **(Imp.)**

Python Variable Types

There are two types of variables in Python - Local variable and Global variable. Let's understand the following variables.

1. Local Variable

Local variables are the variables that declared inside the function and have scope within the function. Let's understand the following example.

Example

```
# Declaring a function
def add():
# Defining local variables. They has scope only within a function
a = 20
b = 30
c = a + b
print("The sum is:", c)
# Calling a function
add()
```

Output

```
The sum is: 50
```

In the above code, we declared a function named add() and assigned a few variables within the function. These variables will be referred to as the local variables which have scope only inside the function. If we try to use them outside the function, we get a following error.

```
add()
# Accessing local variable outside the function
print(a)
```

Output:

```
The sum is: 50
    print(a)
NameError: name 'a' is not defined
```

We tried to use local variable outside their scope; it threw the NameError.

2. Global Variables

Global variables can be used throughout the program, and its scope is in the entire program. We can use global variables inside or outside the function.

A variable declared outside the function is the global variable by default. Python provides the global keyword to use global variable inside the function. If we don't use the global keyword, the function treats it as a local variable. Let's understand the following example.

Example

```
# Declare a variable and initialize it
x = 101
# Global variable in function
def mainFunction():
    # printing a global variable
    global x
    print(x)
    # modifying a global variable
    x = 'Welcome To Javatpoint'
    print(x)
    mainFunction()
    print(x)
```

Output

```
101
Welcome To Javatpoint
Welcome To Javatpoint
```

In the above code, we declare a global variable x and assign a value to it. Next, we defined a function and accessed the declared variable using the global keyword inside the function. Now we can modify its value. Then, we assigned a new string value to the variable x.

Now, we called the function and proceeded to print x. It printed the as newly assigned value of x.

1.7.6 Key Words

Q21. What are Python keywords? Explain.

Ans :

Python keywords are unique words reserved with defined meanings and functions that we can only apply for those functions. You'll never need to import any keyword into your program because they're permanently present.

Python's built-in methods and classes are not the same as the keywords. Built-in methods and classes are constantly present; however, they are not as limited in their application as keywords.

Python contains thirty-five keywords in the most recent version, i.e., Python 3.8. Here we have shown a complete list of Python keywords for the reader's reference.

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

In distinct versions of Python, the preceding keywords might be changed. Some extras may be introduced, while others may be deleted. By writing the following statement into the coding window, you can anytime retrieve the collection of keywords in the version you are working on.

```
# Python program to demonstrate the application of iskeyword()
# importing keyword library which has lists
import keyword
# displaying the complete list using "kwlist()."
print("The set of keywords in this version is: ")
print( keyword.kwlist )
```

Output:

```
The set of keywords in this version is :
```

['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']

By calling help(), you can retrieve a list of currently offered keywords:

help("keywords")

1.7.7 Identifiers

Q22. What are identifiers in python?

Ans :

Python Identifiers

A Python identifier is a name used to identify a variable, function, class, module or other object. An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9).

Python does not allow punctuation characters such as @, $, and % within identifiers. Python is a case sensitive programming language. Thus, Manpower and manpower are two different identifiers in Python.

Here are naming conventions for Python identifiers

- ➢ Class names start with an uppercase letter. All other identifiers start with a lowercase letter.
- ➢ Starting an identifier with a single leading underscore indicates that the identifier is private.
- ➢ Starting an identifier with two leading underscores indicates a strongly private identifier.
- ➢ If the identifier also ends with two trailing underscores, the identifier is a language-defined special name.

Q23. Write about python variables.

Ans :

A variable is a location in memory used to store some data (value).

They are given unique names to differentiate between different memory locations. The rules for writing a variable name is same as the rules for writing identifiers in Python.

We don't need to declare a variable before using it. In Python, we simply assign a value to a variable and it will exist. We don't even have to declare the type of the variable. This is handled internally according to the type of value we assign to the variable.

Variable Assignment

We use the assignment operator (=) to assign values to a variable. Any type of value can be assigned to any valid variable.

```
a = 5
b = 3.2
c = "Hello"
```

Here, we have three assignment statements. 5 is an integer assigned to the variable *a*.

Similarly, 3.2 is a floating point number and "Hello" is a string (sequence of characters) assigned to the variables *b* and *c* respectively.

Multiple Assignments

In Python, multiple assignments can be made in a single statement as follows:

```
a, b, c =5,3.2,"Hello"
```

If we want to assign the same value to multiple variables at once, we can do this as

```
x = y = z ="same"
```

This assigns the "same" string to all the three variables.

1.7.8 Statements

Q24. Write about statements in python.

Ans :

Python Statement

Instructions that a Python interpreter can execute are called statements. For example, a = 1 is an assignment statement. if statement, for statement, while statement etc. are other kinds of statements which will be discussed later.

Multi-line Statement

In Python, end of a statement is marked by a newline character. But we can make a statement extend over multiple lines with the line continuation character (\). For example:

```
a =1+2+3+ \
4+5+6+ \
7+8+9
```

This is explicit line continuation. In Python, line continuation is implied inside parentheses ( ), brackets [ ] and braces { }. For instance, we can implement the above multi-line statement as

```
a =(1+2+3+
4+5+6+
7+8+9)
```

Here, the surrounding parentheses ( ) do the line continuation implicitly. Same is the case with [ ] and { }. For example:

```
colors =['red',
'blue',
'green']
```

We could also put multiple statements in a single line using semicolons, as follows

```
a =1; b =2; c =3
```

Short Question and Answers

1. What is Data Science?

Ans :

Data Science involves obtaining meaningful information or insights from structured or unstructured data through a process of analyzing, programming and business skills. It is a field containing many elements like mathematics, statistics, computer science, etc.

2. What is data analysis?

Ans :

Meaning

Data analysis is defined as a process of cleaning, transforming, and modeling data to discover useful information for business decision-making. The purpose of Data Analysis is to extract useful information from data and taking the decision based upon the data analysis.

Example

A simple example of Data analysis is whenever we take any decision in our day-to-day life is by thinking about what happened last time or what will happen by choosing that particular decision. This is nothing but analyzing our past or future and making decisions based on it.

3. What are the challenges of data science technology?

Ans :

- A high variety of information & data is required for accurate analysis
- Not adequate data science talent pool available
- Management does not provide financial support for a data science team
- Unavailability of/difficult access to data
- Business decision-makers do not effectively use data Science results
- Explaining data science to others is difficult
- Privacy issues
- Lack of significant domain expert
- If an organization is very small, it can't have a Data Science team

4. What is python?

Ans :

Python is a high-level object-oriented programming language that was created by Guido van Rossum. It is also called general-purpose programming language as it is used in almost every domain we can think of as mentioned below:

- Web Development
- Software Development
- Game Development
- AI & ML
- Data Analytics
- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
- Python has a simple syntax similar to the English language.

5. Features of python

Ans :

- **Easy-to-learn:** Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
- **Easy-to-read:** Python code is more clearly defined and visible to the eyes.
- **Easy-to-maintain:** Python's source code is fairly easy-to-maintain.
- **A broad standard library:** Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- **Interactive Mode:** Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- **Portable:** Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- **Extendable:** You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.

- **Databases:** Python provides interfaces to all major commercial databases.

6. Python variables

Ans :

- Variable is a name that is used to refer to memory location. Python variable is also known as an identifier and used to hold value.
- In Python, we don't need to specify the type of variable because Python is a infer language and smart enough to get variable type.
- Variable names can be a group of both the letters and digits, but they have to begin with a letter or an underscore.
- It is recommended to use lowercase letters for the variable name. Rahul and rahul both are two different variables.

7. Python keywords

Ans :

Python keywords are unique words reserved with defined meanings and functions that we can only apply for those functions. You'll never need to import any keyword into your program because they're permanently present.

Python's built-in methods and classes are not the same as the keywords. Built-in methods and classes are constantly present; however, they are not as limited in their application as keywords.

8. Identifiers in python

Ans :

A Python identifier is a name used to identify a variable, function, class, module or other object. An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9).

Python does not allow punctuation characters such as @, $, and % within identifiers. Python is a case sensitive programming language. Thus, Manpower and manpower are two different identifiers in Python.

9. Python variables

Ans :

A variable is a location in memory used to store some data (value).

They are given unique names to differentiate between different memory locations. The rules for writing a variable name is same as the rules for writing identifiers in Python.

We don't need to declare a variable before using it. In Python, we simply assign a value to a variable and it will exist. We don't even have to declare the type of the variable. This is handled internally according to the type of value we assign to the variable.

10. Statements in python

Ans :

Python Statement

Instructions that a Python interpreter can execute are called statements. For example, a = 1 is an assignment statement. if statement, for statement, while statement etc. are other kinds of statements which will be discussed later.

Multi-line Statement

In Python, end of a statement is marked by a newline character. But we can make a statement extend over multiple lines with the line continuation character (\). For example:

```
a = 1 + 2 + 3 + \
4 + 5 + 6 + \
7 + 8 + 9
```

This is explicit line continuation. In Python, line continuation is implied inside parentheses ( ), brackets [ ] and braces { }. For instance, we can implement the above multi-line statement as

```
a = (1 + 2 + 3 +
4 + 5 + 6 +
7 + 8 + 9)
```

Here, the surrounding parentheses ( ) do the line continuation implicitly. Same is the case with [ ] and { }. For example:

```
colors = ['red',
'blue',
'green']
```

We could also put multiple statements in a single line using semicolons, as follows

```
a = 1; b = 2; c = 3
```

Choose the Correct Answer

1. Among the following which is not the tool of data science [c]

   (a) Excel (b) Tableau

   (c) MS Word (d) Matlab

2. Among the following which is a Data Visualization software that is packed with powerful graphics to make interactive visualizations [b]

   (a) Excel (b) Tableau

   (c) MS Word (d) Matlab

3. Among the following which step of data science process does investigate and pre -processing of data. [b]

   (a) Discovery (b) Information Preparation

   (c) Model planning (d) Communication

4. Which of the following is the most important language for Data Science? [c]

   (a) Java (b) Ruby

   (c) R (d) None of the mentioned

5. Which of the following is not the type of data analysis [-]

   (a) Diagnostic Analysis (b) Machine Analysis

   (c) Predictive Analysis (d) Prescriptive Analysis

6. Among the following which Analysis combines the insight from all previous Analysis to determine which action to take in a current problem or decision. [d]

   (a) Diagnostic Analysis (b) Predictive Analysis

   (c) Prescriptive Analysis (d) Prescriptive Anlysis

7. Which keyword is used for function in Python language? [b]

   (a) Function (b) Def

   (c) Fun (d) Define

8. Which of the following character is used to give single-line comments in Python? [b]

   (a) // (b) #

   (c) ! (d) /*

9. Which one of the following is the correct extension of the Python file? [a]

   (a) .py (b) .python

   (c) .p (d) None of these

10. What do we use to define a block of code in Python language? [c]

   (a) Key (b) Brackets

   (c) Indentation (d) None of these

Fill in the blanks

1. ________ is all about the removal of missing, redundant, unnecessary and duplicate data from your collection.
2. ________ is a set of technologies, applications and processes that are used by the enterprises for business data analysis.
3. In ________ stage, you'll create datasets for training and testing purposes.
4. ________ Analysis makes predictions about future outcomes based on current or past data.
5. ________ is often used to discover unknown facts and trends.
6. Python supports the creation of anonymous functions at runtime, using a construct called ________.
7. ________ converts source code written by the developer into intermediate language which is again translated into the native language / machine language that is executed.
8. Full form of PVM
9. ________ in Python are identified as a contiguous set of characters represented in the quotation marks.
10. ________ are the variables that declared inside the function and have scope within the function.

ANSWERS

1. Data cleaning
2. Business intelligence
3. Model Building
4. Predictive
5. Data visualization
6. lambda
7. Interpreter
8. Python virtual machine
9. Strings
10. Local variables

UNIT II

Expressions, Input & Output, Comments, Lines & Indentation, Quotations, Tuple assignment, Operators, Precedence of operators.

Functions: Definition and use, Types of functions, Flow of execution, Parameters and Arguments, Modules.

Conditionals: Conditional(if), Alternative(if-else), Chained Conditionals(if-elif-else), Nested conditionals; Iteration/Control statements: while, for, break, continue, pass; fruitful function vs void function, Parameters/Arguments, Return values, Variables scope(local, global), Function composition.

2.1 Expressions

Q1. Write about python expressions.

Ans :

An expression is a combination of values, variables, and operators. A value all by itself is considered an expression, and so is a variable, so the following are all legal expressions (assuming that the variable x has been assigned a value):

```
17
x
x + 17
```

If you type an expression in interactive mode, the interpreter evaluates it and displays the result:

```
>>> 1 + 1
2
```

But in a script, an expression all by itself doesn't do anything! This is a common source of confusion for beginners.

Exercise

Type the following statements in the Python interpreter to see what they do:

```
5
x = 5
x + 1
```

2.1.1 Input & Output

Q2. Explain formatted print function.

Ans :

Output formatting

The special operator % lets you create formatted output. It takes two operands: a formatted string and a value. The value can be a single value, a tuple of values or a dictionary of values. For example:

```
print("pi=%s"%"3.14159")
```

The formatted string has a conversion specifiers that also uses the special characters %s. This conversion specifier tells Python how to convert the value. Here %s means convert the value to a string. In fact, you could even type:

print("pi=%s"%3.14159)

because the right operand (given the conversion specifiers) should be converted with str().

To be more generic, you could include the value within a tuple:

print("pi=%s"%(3.14159))

and print 2 values:

print("%s=%s"%("pi",3.14159))

The conversion specifiers can also convert values into float, integers and so on.

Special Characters

To escape the sign %, just double it:

>>>print"This is a percent sign: %%"

This is a percent sign: %

Other special characters similar to some other languages are summarized in the following table:

Character	Decimal	Description
\		statement continues on next line
\	92	Backslash
\'	39	Single quote
\"	34	Double quote
\a	7	Bell
\b	8	Backspace
\f		Formfeed
\n	10	Newline
\r	13	carriage return
\t	9	Tabulation
\v	11	vertical tabulation
\0 \000		null value
\ooo		octal value o in (0..7)
\xhh		hexadecimal value (0..9, a..f; A..F)
\uxxxx		Unicode character value

More about conversion specifiers

The general syntax for a conversion specifier is:

%[(key)][flags][width][.prec]type

Conversion Types

We have already seen one type: the string type %s. The following table summarizes all the available types:

Character	Description
C	Converts to a single character
d,i	Converts to a signed decimal integer or long integer
U	Converts to an unsigned decimal integer
e,E	Converts to a floating point in exponential notation
F	Converts to a floating point in fixed-decimal notation
G	Converts to the value shorter of %f and %e
G	Converts to the value shorter of %f and %E
O	Converts to an unsigned integer in octal
R	string generated with repr()
S	Converts to a string using the str() function
x,X	Converts to an unsigned integer in hexadecimal

Formatting String with a Dictionary

Let us now look at the key option. This key refer to the keys used in dictionaries. It works as follows:

```
>>>print("%(key1)s and %(key2)%"%{'key1':1,'key2':2})
"1 and 2"
```

Flags

The second type of options are the flags:

character	Description	example	rendering
0	pad **numbers** with leading weros	"(%04d)" % 2	0002
-	left align the results (default is right)		
Space	add a space before a positive number or string		
+	Always begin a number with a sign (+or-)		
#	display numbers in alternate form.		

The width option

The width option is a positive integer specifying the minimum field width. If the converted value is shorter than width, spaces are added on left or right (depending on flags):

```
>>>print("(%10s)"%"example")
(   example)
```

```
>>>print("(%-10s)"%"example")
(example   )
```

Specific number of digits with the prec option

prec is a dot (.) followed by a positive integer specifying the precision. Note that use the %f conversion specifier here:

```
>>>print("%.2f"%2.012)
2.01
```

Dynamic Formatter

Sometimes, you want to format a string but you do not know its size. In such case, you can use a dynamic formatter using the * character as follows:

```
>>>print'%*s : %*s'%(20,"Python",20,"Very Good")
        Python :          Very Good
```

2.1.2 Comments

Q3. Write a short note on usage of comments in python.

Ans :

Python Comments

Comments are very important while writing a program. It describes what's going on inside a program so that a person looking at the source code does not have a hard time figuring it out. You might forget the key details of the program you just wrote in a month's time. So taking time to explain these concepts in form of comments is always fruitful.

In Python, we use the hash (#) symbol to start writing a comment.

It extends up to the newline character. Comments are for programmers for better understanding of a program. Python Interpreter ignores comment.

```
#This is a comment
#print out Hello
print('Hello')
```

Multi-line Comments

If we have comments that extend multiple lines, one way of doing it is to use hash (#) in the beginning of each line. For example:

```
#This is a long comment
#and it extends
#to multiple lines
```

Another way of doing this is to use triple quotes, either ''' or """.

These triple quotes are generally used for multi-line strings. But they can be used as multi-line comment as well. Unless they are not docstrings, they do not generate any extra code.

```
"""This is also a
perfect example of
multi-line comments"""
```

2.1.3 Lines & Indentation

Q4. What is indentation in python?

Ans :

Python Indentation

Most of the programming languages like C, C++, Java use braces { } to define a block of code. Python uses indentation.

A code block (body of a function, loop etc.) starts with indentation and ends with the first unindented line. The amount of indentation is up to you, but it must be consistent throughout that block.

Generally four whitespaces are used for indentation and is preferred over tabs. Here is an example.

```
For i in range(1,11):
  print(i)
  if i == 5:
     break
```

The enforcement of indentation in Python makes the code look neat and clean. This results into Python programs that look similar and consistent.

Indentation can be ignored in line continuation. But it's a good idea to always indent. It makes the code more readable. For example:

```
ifTrue:
print('Hello')
   a =5
```

and

```
ifTrue:print('Hello'); a =5
```

both are valid and do the same thing. But the former style is clearer.

Incorrect indentation will result into IndentationError.

2.1.4 Quotations

Q5. What are the various types of quotations used in python?

Ans :

Quotation in Python

Python accepts single (‘), double (“) and triple (‘’’ or “””) quotes to denote string literals, as long as the same type of quote starts and ends the string.

The triple quotes are used to span the string across multiple lines. For example, all the following are legal -

```
word = 'word'
sentence = "This is a sentence."
paragraph = """This is a paragraph. It is
made up of multiple lines and sentences."""
```

Q6. What are identifiers in python?

Ans :

Python Identifiers

A Python identifier is a name used to identify a variable, function, class, module or other object. An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9).

Python does not allow punctuation characters such as @, $, and % within identifiers. Python is a case sensitive programming language. Thus, Manpower and manpower are two different identifiers in Python.

Here are naming conventions for Python identifiers:

- Class names start with an uppercase letter. All other identifiers start with a lowercase letter.
- Starting an identifier with a single leading underscore indicates that the identifier is private.
- Starting an identifier with two leading underscores indicates a strongly private identifier.
- If the identifier also ends with two trailing underscores, the identifier is a language-defined special name.

2.1.5 Tuple Assignment

Q7. Write about Tuple assignment feature?

Ans : **(Imp.)**

- An assignment to all of the elements in a tuple using a single assignment statement.
- Python has a very powerful tuple assignment feature that allows a tuple of variables on the left of an assignment to be assigned values from a tuple on the right of the assignment.
- The left side is a tuple of variables; the right side is a tuple of values.
- Each value is assigned to its respective variable.
- All the expressions on the right side are evaluated before any of the assignments. This feature makes tuple assignment quite versatile.
- Naturally, the number of variables on the left and the number of values on the right have to be the same.

```
>>>    (a, b, c, d) = (1, 2, 3)
ValueError: need more than 3 values to unpack
```

Example

It is useful to swap the values of two variables. With conventional assignment statements, we have to use a temporary variable. For example, to swap a and b:

Swap two numbers

```
a=2;b=3
print(a,b)
temp = a
a = b
b = temp
print(a,b)
```

Output

```
(2, 3)
(3, 2)
>>>
```

-Tuple assignment solves this problem neatly:

```
(a, b) = (b, a)
```

One way to think of tuple assignment is as tuple packing/unpacking.

In tuple packing, the values on the left are 'packed' together in a tuple:

```
>>> b = ("George", 25, "20000")        # tuple packing
```

In tuple unpacking, the values in a tuple on the right are 'unpacked' into the variables/names on the right:

```
>>> b = ("George", 25, "20000")     # tuple packing
>>> (name, age, salary) = b  # tuple unpacking
>>>     name
'George'
>>>     age
25
>>>     salary
'20000'
```

The right side can be any kind of sequence (string,list,tuple)

Example

To split an email address in to user name and a domain

```
>>>        mailid='god@abc.org'
>>>        name,domain=mailid.split('@')
>>>        print name
god
print (domain)
abc.org
```

2.1.6 Operators

Q8. What are the various types of operators used in python.

Ans : **(Imp.)**

Operators

Operators are the constructs which can manipulate the value of operands.

Consider the expression 4 + 5 = 9. Here, 4 and 5 are called operands and + is called operator.

Types of Operator

Python language supports the following types of operators.

- Arithmetic Operators
- Comparison (Relational) Operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

Arithmetic Operators

Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication etc.

Operator	Meaning	Example
+	Add two operands or unary plus	x + y +2
-	Subtract right operand from the left or unary minus	x - y -2
*	Multiply two operands	x * y
/	Divide left operand by the right one (always results into float)	x / y
%	Modulus - remainder of the division of left operand by the right	x % y (remainder of x/y)
//	Floor division - division that results into whole number adjusted to the left in the number line	x // y
**	Exponent - left operand raised to the power of right	x**y (x to the power y)

Example 1: Arithmetic Operators in Python

```
x = 15
y = 4

# Output: x + y = 19
print('x + y =',x+y)

# Output: x - y = 11
print('x - y =',x-y)

# Output: x * y = 60
print('x * y =',x*y)

# Output: x / y = 3.75
print('x / y =',x/y)

# Output: x // y = 3
print('x // y =',x//y)

# Output: x ** y = 50625
print('x ** y =',x**y)
```

When you run the program, the output will be:

```
x + y = 19
x - y = 11
```

```
x * y = 60
x / y = 3.75
x // y = 3
x ** y = 50625
```

Comparison Operators

Comparison operators are used to compare values. It either returns True or Falseaccording to the condition.

Operator	Meaning	Example
>	Greater that - True if left operand is greater than the right	x > y
<	Less that - True if left operand is less than the right	x < y
= =	Equal to - True if both operands are equal	x = = y
!=	Not equal to - True if operands are not equal	x != y
> =	Greater than or equal to - True if left operand is greater than or equal to the right	x > = y
< =	Less than or equal to - True if left operand is less than or equal to the right	x < = y

Example 2: Comparison operators in Python

```
x = 10
y = 12

# Output: x > y is False
print('x > y  is',x>y)

# Output: x < y is True
print('x < y  is',x<y)

# Output: x = = y is False
print('x = = y is',x= =y)

# Output: x != y is True
print('x != y is',x!=y)

# Output: x > = y is False
print('x > = y is',x> =y)

# Output: x < = y is True
print('x < = y is',x< =y)
```

Logical Operators

Logical operators are the and, or, not operators.

Operator	Meaning	Example
and	True if both the operands are true	x and y
Or	True if either of the operands is true	x or y
not	True if operand is false (complements the operand)	not x

Example 3: Logical Operators in Python

```
x = True
y = False

# Output: x and y is False
print('x and y is',x and y)

# Output: x or y is True
print('x or y is',x or y)

# Output: not x is False
print('not x is',not x)
```

Bitwise Operators

Bitwise operators act on operands as if they were string of binary digits. It operates bit by bit, hence the name.

For example, 2 is 10 in binary and 7 is 111.

In the table below: Let x = 10 (0000 1010 in binary) and y = 4 (0000 0100 in binary)

Operator	Meaning	Example
&	Bitwise AND	x& y = 0 (0000 0000)
|	Bitwise OR	x | y = 14 (0000 1110)
~	Bitwise NOT	~x = -11 (1111 0101)
^	Bitwise XOR	x ^ y = 14 (0000 1110)
> >	Bitwise right shift	x> > 2 = 2 (0000 0010)
< <	Bitwise left shift	x< < 2 = 42 (0010 1000)

Assignment Operators

Assignment operators are used in Python to assign values to variables.

a = 5 is a simple assignment operator that assigns the value 5 on the right to the variable *a* on the left.

There are various compound operators in Python like a + = 5 that adds to the variable and later assigns the same. It is equivalent to a = a + 5.

Operator	Example	Equivalent to
=	x = 5	x = 5
+=	x += 5	x = x + 5
-=	x -= 5	x = x – 5
*=	x *= 5	x = x * 5
/=	x /= 5	x = x / 5
%=	x %= 5	x = x % 5
//=	x //= 5	x = x // 5
**=	x **= 5	x = x ** 5
&=	x &= 5	x = x & 5
|=	x |= 5	x = x | 5
^ =	x ^ = 5	x = x ^ 5
>>=	x >>= 5	x = x >> 5
<<=	x <<= 5	x = x << 5

Special Operators

Python language offers some special type of operators like the identity operator or the membership operator. They are described below with examples.

Identity Operators

is and is not are the identity operators in Python. They are used to check if two values (or variables) are located on the same part of the memory. Two variables that are equal does not imply that they are identical.

Operator	Meaning	Example
Is	True if the operands are identical (refer to the same object)	x is True
is not	True if the operands are not identical (do not refer to the same object)	x is not True

Example 4: Identity operators in Python

```
x1 = 5
y1 = 5
x2 = ‘Hello’
y2 = ‘Hello’
x3 = [1,2,3]
y3 = [1,2,3]
```

```
# Output: False
print(x1 is not y1)

# Output: True
print(x2 is y2)

# Output: False
print(x3 is y3)
```

Here, we see that x1 and y1 are integers of same values, so they are equal as well as identical. Same is the case with x2 and y2 (strings).

But x3 and y3 are list. They are equal but not identical. Since list are mutable (can be changed), interpreter locates them separately in memory although they are equal.

Membership Operators

in and not in are the membership operators in Python. They are used to test whether a value or variable is found in a sequence (string, list, tuple, set and dictionary).

In a dictionary we can only test for presence of key, not the value.

Operator	**Meaning**	**Example**
In	True if value/variable is found in the sequence	5 in x
not in	True if value/variable is not found in the sequence	5 not in x

Example 5: Membership operators in Python

```
x = 'Hello world'
y = {1:'a',2:'b'}

# Output: True
print('H' in x)

# Output: True
print('hello' not in x)

# Output: True
print(1 in y)

# Output: False
print('a' in y)
```

Here, H' is in x but 'hello' is not present in x (remember, Python is case sensitive). Similary, 1 is key and 'a' is the value in dictionary y. Hence, 'a' in y returns False.

2.1.7 Precedence of Operators

Q9. What is operator precedency in python?

Ans :

The following table lists all operators from highest precedence to lowest.

Operator	**Description**
**	Exponentiation (raise to the power)
~ + -	Complement, unary plus and minus (method names for the last two are +@ and -@)
* / % //	Multiply, divide, modulo and floor division
+ -	Addition and subtraction
>> <<	Right and left bitwise shift
&	Bitwise 'AND'
^ |	Bitwise exclusive `OR' and regular `OR'
<= < > >=	Comparison operators
<> == !=	Equality operators
= %= /= //= -= += *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators
not or and	Logical operators

Q10. Write a program to add two numbers

Ans :

```
# This program adds two numbers

# Store input numbers
num1 = input('Enter first number: ')
num2 = input('Enter second number: ')
# Add two numbers
sum = float(num1) + float(num2)
# Display the sum
print('The sum of {0} and {1} is {2}'.format(num1, num2, sum))
```

Output

```
Enter first number: 1.5
Enter second number: 6.3
The sum of 1.5 and 6.3 is 7.8
```

2.2 Functions

2.2.1 Definition and Use

Q11. What is function? How to define and call a function?

Ans : **(Imp.)**

A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.

As you already know, Python gives you many built-in functions like print(), etc. but you can also create your own functions. These functions are called user-defined functions.

Defining a Function

You can define functions to provide the required functionality. Here are simple rules to define a function in Python.

- Function blocks begin with the keyword def followed by the function name and parentheses (( )).
- Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.
- The first statement of a function can be an optional statement - the documentation string of the function or docstring.
- The code block within every function starts with a colon (:) and is indented.
- The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

Syntax of Function

```
def function_name(parameters):
        """docstring"""
        statement(s)
```

Above shown is a function definition which consists of following components.

1. Keyword def marks the start of function header.
2. A function name to uniquely identify it. Function naming follows the same rules of writing identifiers in Python.
3. Parameters (arguments) through which we pass values to a function. They are optional.
4. A colon (:) to mark the end of function header.
5. Optional documentation string (docstring) to describe what the function does.
6. One or more valid python statements that make up the function body. Statements must have same indentation level (usually 4 spaces).
7. An optional return statement to return a value from the function.

Example of a Function

```
def printme( str ):
"This prints a passed string into this function"
print str
return
```

Calling a Function

Defining a function only gives it a name, specifies the parameters that are to be included in the function and structures the blocks of code.

Once the basic structure of a function is finalized, you can execute it by calling it from another function or directly from the Python prompt. Following is the example to call printme() function –

```
#!/usr/bin/python
# Function definition is here
def printme( str ):
"This prints a passed string into this function"
print str
return;
# Now you can call printme function
printme("I'm first call to user defined function!")
printme("Again second call to the same function")
```

When the above code is executed, it produces the following result –

```
I'm first call to user defined function!
Again second call to the same function
```

```
# Define our 3 functions
def my_function():
  print("Hello From My Function!")

def my_function_with_args(username, greeting):
   print("Hello, %s , From My Function!, I wish
you %s"%(username, greeting))
```

```
def sum_two_numbers(a, b):
    return a + b

# print(a simple greeting)
my_function()
```

#prints - "Hello, John Doe, From My Function!, I wish you a great year!"

my_function_with_args("John Doe", "a great year!")

```
# after this line x will hold the value 3!
x = sum_two_numbers(1,2)
```

2.2.2 Types of Functions

Q12. Explain about various types of functions in Python.

Ans : **(Imp.)**

Basically, we can divide functions into the following two types:

1. **Built-in functions -** Functions that are built into Python.
2. **User-defined functions -** Functions defined by the users themselves.

Python Built-in Functions

The Python built-in functions are defined as the functions whose functionality is pre-defined in Python. The python interpreter has several functions that are always present for use. These functions are known as Built-in Functions. There are several built-in functions in Python which are listed below:

1. Python abs() Function

The python abs() function is used to return the absolute value of a number. It takes only one argument, a number whose absolute value is to be returned. The argument can be an integer and floating-point number. If the argument is a complex number, then, abs() returns its magnitude.

Python abs() Function Example

```
#  integer number
integer = -20
print('Absolute value of -40 is:', abs(integer))
  #  floating number
floating = -20.83
print('Absolute value of -40.83 is:', abs(floating))
```

Output:

```
Absolute value of -20 is: 20
Absolute value of -20.83 is: 20.83
```

2. Python all() Function

The python all() function accepts an iterable object (such as list, dictionary, etc.). It returns true if all items in passed iterable are true. Otherwise, it returns False. If the iterable object is empty, the all() function returns True.

Python all() Function Example

```
# all values true
k = [1, 3, 4, 6]
print(all(k))
  # all values false
k = [0, False]
print(all(k))
# one false value
k = [1, 3, 7, 0]
print(all(k))
  # one true value
k = [0, False, 5]
print(all(k))
  # empty iterable
k = []
print(all(k))
```

Output:

```
True
False
False
False
True
```

3. Python bin() Function

The python bin() function is used to return the binary representation of a specified integer. A result always starts with the prefix 0b.

Python bin() Function Example

```
x = 10
y = bin(x)
print (y)
```

Output:

```
0b1010
```

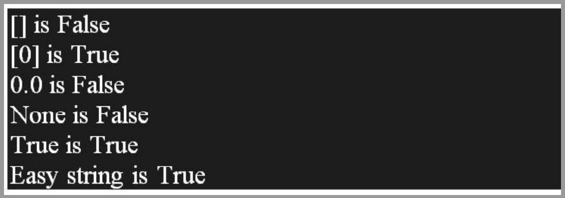
4. Python bool()

The python bool() converts a value to boolean(True or False) using the standard truth testing procedure.

Python bool() Example

```
test1 = []
print(test1,’is’,bool(test1))
test1 = [0]
print(test1,’is’,bool(test1))
test1 = 0.0
print(test1,’is’,bool(test1))
test1 = None
print(test1,’is’,bool(test1))
test1 = True
print(test1,’is’,bool(test1))
test1 = ’Easy string’
print(test1,’is’,bool(test1))
```

Output:



```
[] is False
[0] is True
0.0 is False
None is False
True is True
Easy string is True
```

5. Python bytes()

The python bytes() in Python is used for returning a bytes object. It is an immutable version of the bytearray() function.

It can create empty bytes object of the specified size.

Python bytes() Example

```
string = ”Hello World.”
array = bytes(string, ’utf-8')
print(array)
```

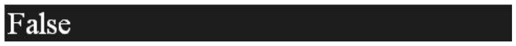
6. Python callable() Function

A python callable() function in Python is something that can be called. This built-in function checks and returns true if the object passed appears to be callable, otherwise false.

Python callable() Function Example

```
x = 8
print(callable(x))
```

Output:



```
False
```

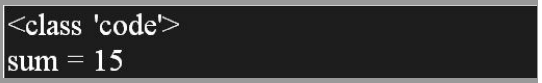
7. Python compile() Function

The python compile() function takes source code as input and returns a code object which can later be executed by exec() function.

Python compile() Function Example

```
# compile string source to code
code_str = ’x=5\ny=10\nprint(“sum =”,x+y)’
code = compile(code_str, ’sum.py’, ’exec’)
print(type(code))
exec(code)
exec(x)
```

Output:



```
<class 'code'>
sum = 15
```

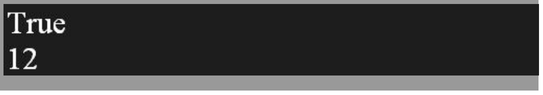
8. Python exec() Function

The python exec() function is used for the dynamic execution of Python program which can either be a string or object code and it accepts large blocks of code, unlike the eval() function which only accepts a single expression.

Python exec() Function Example

```
x = 8
exec(‘print(x==8)’)
exec(‘print(x+4)’)
```

Output:



```
True
12
```

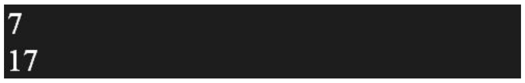
9. Python sum() Function

As the name says, python sum() function is used to get the sum of numbers of an iterable, i.e., list.

Python sum() Function Example

```
s = sum([1, 2,4 ])
print(s)
   s = sum([1, 2, 4], 10)
print(s)
```

Output:



```
7
17
```

10. Python any() Function

The python any() function returns true if any item in an iterable is true. Otherwise, it returns False.

Python any() Function Example

```
l = [4, 3, 2, 0]
print(any(l))
  l = [0, False]
print(any(l))
  l = [0, False, 5]
print(any(l))
  l = []
print(any(l))
```

Output:

```
True
False
True
False
```

11. Python ascii() Function

The python ascii() function returns a string containing a printable representation of an object and escapes the non-ASCII characters in the string using \x, \u or \U escapes.

Python ascii() Function Example

```
normalText = 'Python is interesting'
print(ascii(normalText))
 otherText = 'Pythön is interesting'
print(ascii(otherText))
  print('Pyth\xf6n is interesting')
```

Output:

```
'Python is interesting'
'Pyth\xf6n is interesting'
Pythön is interesting
```

12. Python bytearray()

The python bytearray() returns a bytearray object and can convert objects into bytearray objects, or create an empty bytearray object of the specified size.

Python bytearray() Example

```
string = "Python is a programming language."
 # string with encoding 'utf-8'
arr = bytearray(string, 'utf-8')
print(arr)
```

Output:

```
bytearray(b'Python is a programming language.')
```

13. Python eval() Function

The python eval() function parses the expression passed to it and runs python expression(code) within the program.

Python eval() Function Example

```
x = 8
print(eval('x + 1'))
```

Output:

```
9
```

14. Python float()

The python float() function returns a floating-point number from a number or string.

Python float() Example

```
# for integers
print(float(9))
  # for floats
print(float(8.19))
# for string floats
print(float("-24.27"))
  # for string floats with whitespaces
print(float("     -17.19\n"))
 # string float error
print(float("xyz"))
```

Output:

```
9.0
8.19
-24.27
-17.19
ValueError: could not convert string to float: 'xyz'
```

15. Python format() Function

The python format() function returns a formatted representation of the given value.

Python format() Function Example

```
# d, f and b are a type
  # integer
print(format(123, "d"))
  # float arguments
print(format(123.4567898, "f"))
  # binary format
print(format(12, "b"))
```

Output:

```
123
123.456790
1100
```

16. Python frozenset()

The python frozenset() function returns an immutable frozenset object initialized with elements from the given iterable.

Python frozenset() Example

```
# tuple of letters
letters = ('m', 'r', 'o', 't', 's')
 fSet = frozenset(letters)
print('Frozen set is:', fSet)
print('Empty frozen set is:', frozenset())
```

Output:

```
Frozen set is: frozenset({'o', 'm', 's', 'r', 't'})
Empty frozen set is: frozenset()
```

17. Python getattr() Function

The python getattr() function returns the value of a named attribute of an object. If it is not found, it returns the default value.

Python getattr() Function Example

```
class Details:
    age = 22
    name = "Phill"
  details = Details()
print('The age is:', getattr(details, "age"))
print('The age is:', details.age)
```

Output:

```
The age is: 22
The age is: 22
```

18. Python globals() Function

The python globals() function returns the dictionary of the current global symbol table.

A Symbol table is defined as a data structure which contains all the necessary information about the program. It includes variable names, methods, classes, etc.

Python globals() Function Example

```
age = 22
 globals()['age'] = 22
print('The age is:', age)
```

Output:

```
The age is: 22
```

19. Python len() Function

The python len() function is used to return the length (the number of items) of an object.

Python len() Function Example

```
strA = 'Python'
print(len(strA))
```

Output:

```
6
```

20. Python list()

The python list() creates a list in python.

Python list() Example

```
# empty list
print(list())
  # string
String = 'abcde'
print(list(String))
 # tuple
Tuple = (1,2,3,4,5)
print(list(Tuple))
# list
List = [1,2,3,4,5]
print(list(List))
```

Output:

```
[]
['a', 'b', 'c', 'd', 'e']
[1,2,3,4,5]
[1,2,3,4,5]
```

21. Python locals() Function

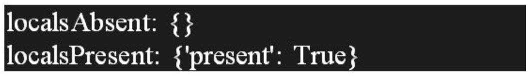
The python locals() method updates and returns the dictionary of the current local symbol table.

A Symbol table is defined as a data structure which contains all the necessary information about the program. It includes variable names, methods, classes, etc.

Python locals() Function Example

```
def localsAbsent():
      return locals()
def localsPresent():
      present = True
      return locals()
   print('localsNotPresent:', localsAbsent())
print('localsPresent:', localsPresent())
```

Output:



```
localsAbsent: {}
localsPresent: {'present': True}
```

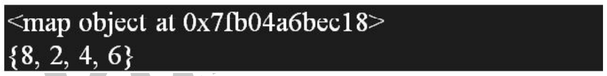
22. Python map() Function

The python map() function is used to return a list of results after applying a given function to each item of an iterable(list, tuple etc.).

Python map() Function Example

```
def calculateAddition(n):
   return n+n
   numbers = (1, 2, 3, 4)
result = map(calculateAddition, numbers)
print(result)
  # converting map object to set
numbersAddition = set(result)
print(numbersAddition)
```

Output:



```
<map object at 0x7fb04a6bec18>
{8, 2, 4, 6}
```

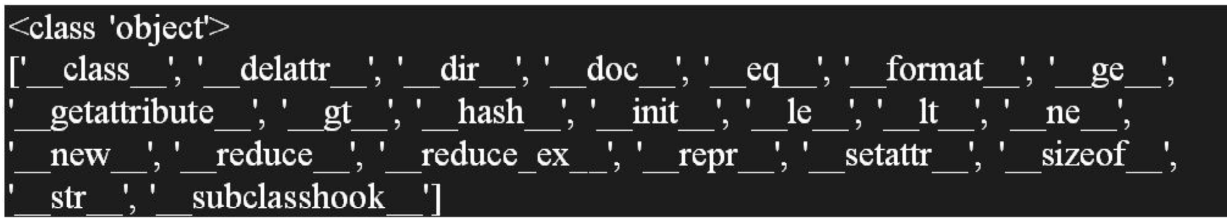
23. Python object()

The python object() returns an empty object. It is a base for all the classes and holds the built-in properties and methods which are default for all the classes.

Python object() Example

```
1.    python = object()
2.       print(type(python))
3.    print(dir(python))
```

Output:



```
<class 'object'>
['__class__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__',
'__getattribute__', '__gt__', '__hash__', '__init__', '__le__', '__lt__', '__ne__',
'__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__',
'__str__', '__subclasshook__']
```

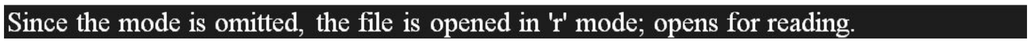
24. Python open() Function

The python open() function opens the file and returns a corresponding file object.

Python open() Function Example

```
# opens python.text file of the current directory
f = open("python.txt")
# specifying full path
f = open("C:/Python33/README.txt")
```

Output:



```
Since the mode is omitted, the file is opened in 'r' mode; opens for reading.
```

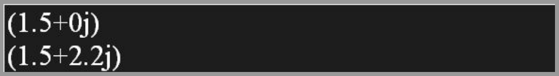
25. Python complex()

Python complex() function is used to convert numbers or string into a complex number. This method takes two optional parameters and returns a complex number. The first parameter is called a real and second as imaginary parts.

Python complex() Example

```
Python complex() function example
# Calling function
a = complex(1) # Passing single parameter
b = complex(1,2) # Passing both parameters
# Displaying result
print(a)
print(b)
```

Output:



```
(1.5+0j)
(1.5+2.2j)
```

26. Python dict()

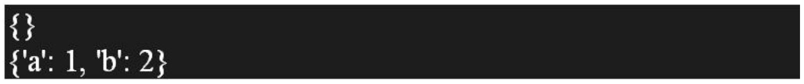
Python dict() function is a constructor which creates a dictionary. Python dictionary provides three different constructors to create a dictionary:

- ➢ If no argument is passed, it creates an empty dictionary.
- ➢ If a positional argument is given, a dictionary is created with the same key-value pairs. Otherwise, pass an iterable object.
- ➢ If keyword arguments are given, the keyword arguments and their values are added to the dictionary created from the positional argument.

Python dict() Example

```
# Calling function
result = dict() # returns an empty dictionary
result2 = dict(a=1,b=2)
# Displaying result
print(result)
print(result2)
```

Output:



```
{}
{'a': 1, 'b': 2}
```

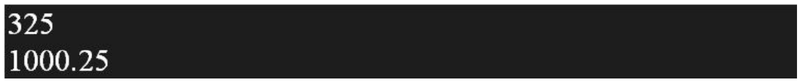
27. Python min() Function

Python min() function is used to get the smallest element from the collection. This function takes two arguments, first is a collection of elements and second is key, and returns the smallest element from the collection.

Python min() Function Example

```
# Calling function
small = min(2225,325,2025) # returns smallest element
small2 = min(1000.25,2025.35,5625.36,10052.50)
# Displaying result
print(small)
print(small2)
```

Output:



```
325
1000.25
```

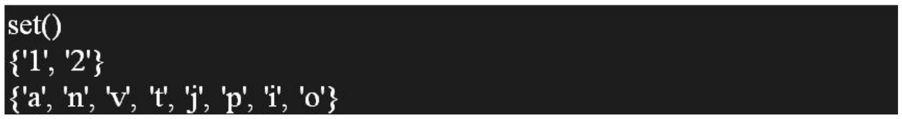
28. Python set() Function

In python, a set is a built-in class, and this function is a constructor of this class. It is used to create a new set using elements passed during the call. It takes an iterable object as an argument and returns a new set object.

Python set() Function Example

```
Calling function
result = set() # empty set
result2 = set('12')
result3 = set('javatpoint')
# Displaying result
print(result)
print(result2)
print(result3)
```

Output:



```
set()
{'1', '2'}
{'a', 'n', 'v', 't', 'j', 'p', 'i', 'o'}
```

29. Python hex() Function

Python hex() function is used to generate hex value of an integer argument. It takes an integer argument and returns an integer converted into a hexadecimal string. In case, we want to get a hexadecimal value of a float, then use float.hex() function.

Python hex() Function Example

```
# Calling function
result = hex(1)
# integer value
```

```
result2 = hex(342)
# Displaying result
print(result)
print(result2)
```

Output:

```
0x1
0x156
```

30. Python slice() Function

Python slice() function is used to get a slice of elements from the collection of elements. Python provides two overloaded slice functions. The first function takes a single argument while the second function takes three arguments and returns a slice object. This slice object can be used to get a subsection of the collection.

Python slice() Function Example

```
# Calling function
result = slice(5) # returns slice object
result2 = slice(0,5,3) # returns slice object
# Displaying result
print(result)
print(result2)
```

Output:

```
slice(None, 5, None)
slice(0, 5, 3)
```

31. Python sorted() Function

Python sorted() function is used to sort elements. By default, it sorts elements in an ascending order but can be sorted in descending also. It takes four arguments and returns a collection in sorted order. In the case of a dictionary, it sorts only keys, not values.

Python sorted() Function Example

```
str = "javatpoint" # declaring string
# Calling function
sorted1 = sorted(str) # sorting string
# Displaying result
print(sorted1)
```

Output:

```
['a', 'a', 'i', 'j', 'n', 'o', 'p', 't', 't', 'v']
```

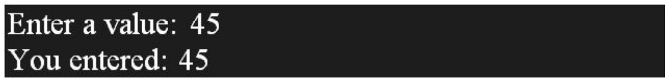
32. Python input() Function

Python input() function is used to get an input from the user. It prompts for the user input and reads a line. After reading data, it converts it into a string and returns it. It throws an error EOFError if EOF is read.

Python input() Function Example

```
# Calling function
val = input("Enter a value: ")
# Displaying result
print("You entered:",val)
```

Output:



```
Enter a value: 45
You entered: 45
```

33. Python int() Function

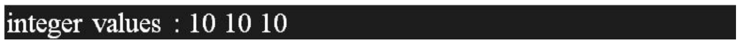
Python int() function is used to get an integer value. It returns an expression converted into an integer number. If the argument is a floating-point, the conversion truncates the number. If the argument is outside the integer range, then it converts the number into a long type.

If the number is not a number or if a base is given, the number must be a string.

Python int() Function Example

```
    # Calling function
    val = int(10) # integer value
    val2 = int(10.52) # float value
    val3 = int('10') # string value
    # Displaying result
print("integer values :",val, val2, val3)
```

Output:



```
integer values : 10 10 10
```

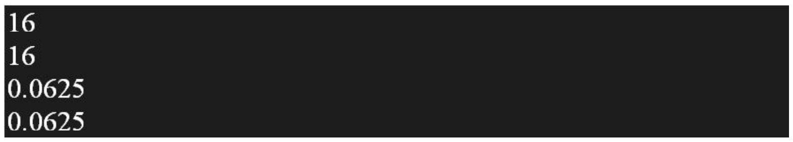
34. Python pow() Function

The python pow() function is used to compute the power of a number. It returns x to the power of y. If the third argument(z) is given, it returns x to the power of y modulus z, i.e. (x, y) % z.

Python pow() function Example

```
# positive x, positive y (x**y)
print(pow(4, 2))
 # negative x, positive y
print(pow(-4, 2))
  # positive x, negative y (x**-y)
print(pow(4, -2))
  # negative x, negative y
print(pow(-4, -2))
```

Output:



```
16
16
0.0625
0.0625
```

35. Python print() Function

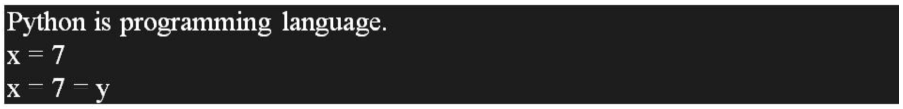
The python print() function prints the given object to the screen or other standard output devices.

Python print() function Example

```
print("Python is programming language.")
 x = 7
```

```
# Two objects passed
print(“x =”, x)
y = x
# Three objects passed
print(‘x =’, x, ’= y’)
```

Output:



```
Python is programming language.
x = 7
x = 7 = y
```

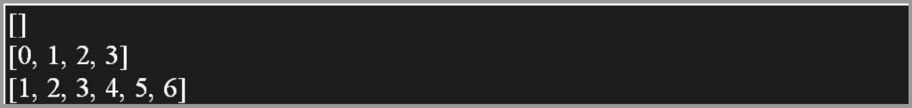
36. Python range() Function

The python range() function returns an immutable sequence of numbers starting from 0 by default, increments by 1 (by default) and ends at a specified number.

Python range() function Example

```
# empty range
print(list(range(0)))
 # using the range(stop)
print(list(range(4)))
  # using the range(start, stop)
print(list(range(1,7 )))
```

Output:



```
[]
[0, 1, 2, 3]
[1, 2, 3, 4, 5, 6]
```

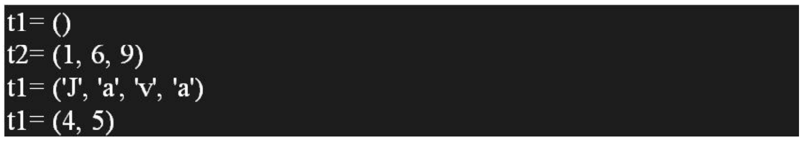
37. Python tuple() Function

The python tuple() function is used to create a tuple object.

Python tuple() Function Example

```
1 = tuple()
print(‘t1=’, t1)
  # creating a tuple from a list
t2 = tuple([1, 6, 9])
print(‘t2=’, t2)
  # creating a tuple from a string
t1 = tuple(‘Java’)
print(‘t1=’,t1)
  # creating a tuple from a dictionary
t1 = tuple({4: ’four’, 5: ’five’})
print(‘t1=’,t1)
```

Output:



```
t1= ()
t2= (1, 6, 9)
t1= ('J', 'a', 'v', 'a')
t1= (4, 5)
```

Q13. What are User-Defined Functions in Python? Write about them?

Ans : **(Imp.)**

Functions that we define ourselves to do the certain specific task are referred to as user-defined functions. The way in which we define and call functions in Python are already discussed.

Functions that readily come with Python are called built-in functions. If we use functions written by others in the form of the library, it can be termed as library functions.

All the other functions that we write on our own fall under user-defined functions. So, our user-defined function could be a library function to someone else.

Advantages of user-defined functions

1. User-defined functions help to decompose a large program into small segments which makes the program easy to understand, maintain and debug.
2. If repeated code occurs in a program. The function can be used to include those codes and execute when needed by calling that function.
3. Programmers working on a large project can divide the workload by making different functions.

Syntax

```
def function_name(argument1, argument2, ...) :
    statement_1
    statement_2
    ....
```

Example

```
# Program to illustrate
# the use of user-defined functions

defadd_numbers(x,y):
   sum=x +y
   returnsum

num1 =5
num2 =6

print("The sum is", add_numbers(num1, num2))
```

Output

```
Enter a number: 2.4
Enter another number: 6.5
The sum is 8.9
```

Next up on this Python Functions blog, let us check out how we can create a simple application using Python.

Python Program To Create A Simple Calculator Application

In this example, you will learn to create a simple calculator that can add, subtract, multiply or divide depending upon the input from the user.

# Program make a simple calculator that can add, subtract, multiply and divide using functions

```
 # This function adds two numbers
defadd(x, y):
   returnx +y
 # This function subtracts two numbers
defsubtract(x, y):
   returnx -y
 # This function multiplies two numbers
defmultiply(x, y):
   returnx *y
 # This function divides two numbers
defdivide(x, y):
   returnx /y
 print("Select operation.")
print("1.Add")
print("2.Subtract")
print("3.Multiply")
print("4.Divide")
 # Take input from the user
choice =input("Enter choice(1/2/3/4):")

num1 =int(input("Enter first number: "))
num2 =int(input("Enter second number: "))
 ifchoice =='1':
    print(num1,"+",num2,"=",
add(num1,num2))

elifchoice =='2':
    print(num1,"-",num2,"=",
subtract(num1,num2))
```

```
elifchoice = ='3':
    print(num1,"*",num2,"=", multiply(num1,num2))
  elifchoice = ='4':
    print(num1,"/",num2,"=", divide(num1,num2))
else:
    print("Invalid input")
```

Output

```
Select operation.
1.Add
2.Subtract
3.Multiply
4.Divide
Enter choice(1/2/3/4): 3
Enter first number: 15
Enter second number: 14
15 * 14 = 210
```

2.2.3 Flow of Execution

Q14. Explain the Flow of Execution in Python.

Ans : **(Imp.)**

- The order in which statements are executed is called the flow of execution
- Execution always begins at the first statement of the program.
- Statements are executed one at a time, in order, from top to bottom.
- Function definitions do not alter the flow of execution of the program, but remember that statements inside the function are not executed until the function is called.
- Function calls are like a bypass in the flow of execution. Instead of going to thenext statement, the flow jumps to the first line of the called function, executes all the statements there, and then comes back to pick up where it left off.

Consider the code of python

```
x=int(input("Enter any no=")) # Line 1
sqr=x * x # Line 2
print(sqr) # Line 3
```

In above code execution of the code is Line 1 2 3. If order is not same the code will produce error.

In above code if we include any branching (if-else or if-elif) or looping (for or while)used in prorgam , then flow of execution may depend on the conditions.

see the code given below

```
x=int(input("Enter any no=")) # Line 1
sqr=0 # Line 2
if x>0: # Line 3
sqr=x * x # Line 4
print(sqr) # Line 5
```

Now if x is positive number then square of number is calculated otherwise not.

Now execution is Line 1 2 3 (Now if x is above 0) 3 4

Eexecution is Line 1 2 3 (Now if x is 0 or below) 5 . Line 4 is not executed .

Now we discuss the function execution.

2.2.4 Parameters and Arguments

Q15. Write about how to define a parameters in python.

Ans : **(Imp.)**

Parameters:

A parameter is the variable defined within the parentheses during function definition. Simply they are written when we declare a function.

Example:

```
# Here a,b are the parameters
defsum(a,b):
  print(a+b)
    sum(1,2)
```

Output:

```
3
```

Arguments:

An argument is a value that is passed to a function when it is called. It might be a variable, value or object passed to a function or method as input. They are written when we are calling the function.

Example:

```
defsum(a,b):
  print(a+b)
    # Here the values 1,2 are arguments
sum(1,2)
```

Output:

```
3
```

Types of Arguments in Python

Python functions can contain two types of arguments:

- Positional Arguments
- Keyword Arguments

Positional Arguments

Positional Arguments are needed to be included in proper order i.e the first argument is always listed first when the function is called, second argument needs to be called second and so on.

Example:

```
defperson_name(first_name,second_name):
  print(first_name+second_name)
```

```
# First name is Ram placed first
# Second name is Babu place second
person_name("Ram","Babu")
```

Output:

```
RamBabu
```

Keyword Arguments:

Keyword Arguments is an argument passed to a function or method which is preceded by a keyword and an equal to sign. The order of keyword argument with respect to another keyword argument does not matter because the values are being explicitly assigned.

```
defperson_name(first_name,second_name):
    print(first_name+second_name)
    # Here we are explicitly assigning the values
person_name(second_name="Babu",first_name="Ram")
```

Output:

```
RamBabu
```

2.2.5 Modules

Q16. What are Modules in Python? Explain.

Ans : **(Imp.)**

A document with definitions of functions and various statements written in Python is called a Python module.

In Python, we can define a module in one of 3 ways:

- Python itself allows for the creation of modules.
- Similar to the re (regular expression) module, a module can be primarily written in C programming language and then dynamically inserted at run-time.
- A built-in module, such as the itertools module, is inherently included in the interpreter.

A module is a file containing Python code, definitions of functions, statements, or classes. An example_module.py file is a module we will create and whose name is example_module.

We employ modules to divide complicated programs into smaller, more understandable pieces. Modules also allow for the reuse of code.

Rather than duplicating their definitions into several applications, we may define our most frequently used functions in a separate module and then import the complete module.

```
# Python program to show how to create a module.
# defining a function in the module to reuse it
def square( number ):
    """This function will square the number passed to it"""
      result = number ** 2
    return result
```

Here, a module called example_module contains the definition of the function square(). The function returns the square of a given number.

Import Modules

In Python, we may import functions from one module into our program, or as we say into, another module.

For this, we make use of the import Python keyword. In the Python window, we add the next to import keyword, the name of the module we need to import. We will import the module we defined earlier example_module.

```
import example_module
```

The functions that we defined in the example_module are not immediately imported into the present program. Only the name of the module, i.e., example_ module, is imported here.

We may use the dot operator to use the functions using the module name. For instance:

Code

```
result = example_module.square( 4 )
print( ”By using the module square of number is: ”, result )
```

Output:

```
By using the module square of number is: 16
```

2.2.6 Conditionals

2.2.6.1 Conditional (IF)

Q17. Explain if statement with syntax and example.

Ans :

Decision making is required when we want to execute a code only if a certain condition is satisfied.

The if…elif…else statement is used in Python for decision making.

Python if Statement Syntax

```
if test expression:
   statement(s)
```

Here, the program evaluates the test expression and will execute statement(s) only if the text expression is True.

If the text expression is False, the statement(s) is not executed.

In Python, the body of the if statement is indicated by the indentation. Body starts with an indentation and the first unindented line marks the end.

Python interprets non-zero values as True. None and 0 are interpreted as False.

Python if Statement Flowchart

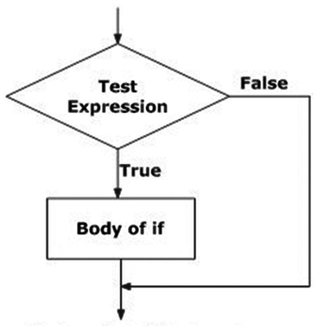


Fig: Operation of if statement

Example: Python if Statement

# If the number is positive, we print an appropriate message

```
num = 3
if num > 0:
   print(num, "is a positive number.")
print("This is always printed.")

num = -1
if num > 0:
   print(num, "is a positive number.")
print("This is also always printed.")
```

When you run the program, the output will be:

```
3 is a positive number
This is always printed
This is also always printed.
```

In the above example, num > 0 is the test expression.

The body of if is executed only if this evaluates to True.

When variable num is equal to 3, test expression is true and body inside body of if is executed.

If variable num is equal to -1, test expression is false and body inside body of if is skipped.

The print() statement falls outside of the if block (unindented). Hence, it is executed regardless of the test expression.

2.2.6.2 Alternative (If-else)

Q18. Explain Python if...else Statement

Ans :

Syntax of if…else

```
if test expression:
   Body of if
else:
   Body of else
```

The if..else statement evaluates test expression and will execute body of if only when test condition is True.

If the condition is False, body of else is executed. Indentation is used to separate the blocks.

Python if..else Flowchart

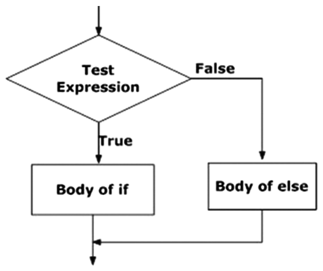


FIg. : Operation of if...else statement

Example of if...else

# Program checks if the number is positive or negative

```
# And displays an appropriate message

num = 3

# Try these two variations as well.
# num = -5
# num = 0

if num >= 0:
   print("Positive or Zero")
else:
   print("Negative number")
```

In the above example, when num is equal to 3, the test expression is true and body of ifis executed and body of else is skipped.

If num is equal to -5, the test expression is false and body of else is executed and body of if is skipped.

If num is equal to 0, the test expression is true and body of if is executed and body of else is skipped.

2.2.6.3 Chained Conditionals (If-elif-else)

Q19. Explain if-elif –else statement with syntax and example.

Ans : **(Imp.)**

Syntax of if...elif...else

```
if test expression:
   Body of if
elif test expression:
   Body of elif
else:
   Body of else
```

The elif is short for else if. It allows us to check for multiple expressions.

If the condition for if is False, it checks the condition of the next elif block and so on.

If all the conditions are False, body of else is executed.

Only one block among the several if...elif...else blocks is executed according to the condition.

The if block can have only one else block. But it can have multiple elif blocks.

Flowchart of if...elif...else

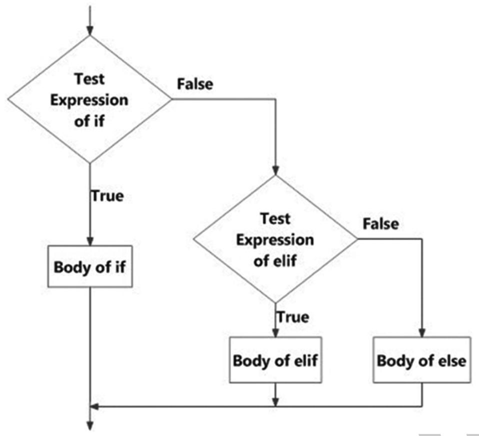


Fig. : Operation of if...elif...else statement

Example of if...elif...else

```
# In this program,
# we check if the number is positive or
# negative or zero and
# display an appropriate message

num = 3.4

# Try these two variations as well:
# num = 0
# num = -4.5

if num > 0:
  print("Positive number")
elif num = = 0:
  print("Zero")
else:
  print("Negative number")
```

When variable *num* is positive, Positive number is printed.

If *num* is equal to 0, Zero is printed.

If *num* is negative, Negative number is printed

2.2.6.4 Nested Conditionals

Q20. Explain nested if statements with syntax and example.

Ans :

Python Nested if statements

We can have a if...elif...else statement inside another if...elif...else statement. This is called nesting in computer programming.

Any number of these statements can be nested inside one another. Indentation is the only way to figure out the level of nesting. This can get confusing, so must be avoided if we can.

Python Nested if Example

```
# In this program, we input a number
# check if the number is positive or
# negative or zero and display
# an appropriate message
# This time we use nested if

num =float(input("Enter a number: "))
if num > =0:
if num = =0:
print("Zero")
else:
print("Positive number")
else:
print("Negative number")
```

Output 1

```
Enter a number: 5
Positive number
```

Output 2

```
Enter a number: -1
Negative number
```

Output 3

```
Enter a number: 0
Zero
```

Q21. Write a program to check whether the given number is prime or not.

Ans :

```
# Python program to check if the input number is prime or not
num = 407
# take input from the user
# num = int(input("Enter a number: "))

# prime numbers are greater than 1
if num > 1:
  # check for factors
  for i in range(2,num):
      if (num % i) = = 0:
         print(num,"is not a prime number")
         print(i,"times",num//i,"is",num)
          break
  else:
     print(num,"is a prime number")
    # if input number is less than
# or equal to 1, it is not prime
else:
  print(num,"is not a prime number")
```

2.3 Iteration / Control Statements

2.3.1 While

Q22. Explain while loop with syntax and example.

Ans : **(Imp.)**

The while loop in Python is used to iterate over a block of code as long as the test expression (condition) is true.

We generally use this loop when we don't know beforehand, the number of times to iterate.

Syntax of while Loop in Python

```
while test_expression:
   Body of while
```

In while loop, test expression is checked first. The body of the loop is entered only if the test_expression evaluates to True. After one iteration, the test expression is checked again. This process continues until the test_expression evaluates to False.

In Python, the body of the while loop is determined through indentation.

Body starts with indentation and the first unindented line marks the end.

Python interprets any non-zero value as True. None and 0 are interpreted as False.

Flowchart of while Loop

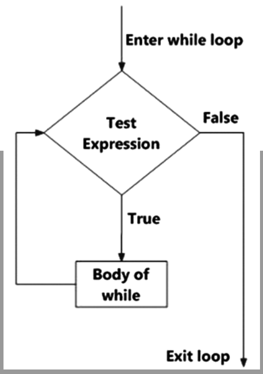


Fig. : Operation of while loop

Example: Python while Loop

```
# Program to add natural
# numbers upto
# sum = 1+2+3+...+n

# To take input from the user,
# n = int(input("Enter n: "))
n = 10
# initialize sum and counter
sum = 0
i = 1

while i < = n:
   sum = sum + i
   i = i+1    # update counter

# print the sum
print("The sum is", sum)
```

When you run the program, the output will be:

```
Enter n: 10
The sum is 55
```

In the above program, the test expression will be True as long as our counter variable i is less than or equal to n (10 in our program).

We need to increase the value of counter variable in the body of the loop. This is very important (and mostly forgotten). Failing to do so will result in an infinite loop (never ending loop).

Finally the result is displayed.

Q23. Explain while loop with else statement

Ans :

while loop with else

Same as that of for loop, we can have an optional else block with while loop as well.

The else part is executed if the condition in the while loop evaluates to False. The while loop can be terminated with a break statement.

In such case, the else part is ignored. Hence, a while loop's else part runs if no break occurs and the condition is false.

Here is an example to illustrate this.

```
# Example to illustrate
# the use of else statement
# with the while loop

counter = 0

while counter < 3:
   print("Inside loop")
   counter = counter + 1
else:
   print("Inside else")
```

Output

```
Inside loop
Inside loop
Inside loop
Inside else
```

Here, we use a counter variable to print the string Inside loop three times.

On the forth iteration, the condition in while becomes False. Hence, the else part is executed.

2.3.2 FOR

Q24. Explain For loop with example.

Ans :

The for statement is in opposition to the "while" loop, which is employed whenever a condition requires to be verified each repetition or when a piece of code is to be repeated indefinitely.

Syntax of for Loop

```
for value in sequence:
    {loop body}
```

On each iteration, the value is the parameter that gets the element's value within the iterable sequence. If an expression statement is present in a sequence, it is processed first. The iterating variable iterating_variable is then allocated to the first element in the sequence. After that, the intended block is run. The statement block is performed until the whole sequence is completed, and each element in the sequence is allocated to iterating_variable. The for loop's material is distinguished from the rest of the program using indentation.

Example of Python for Loop

```
# Code to find the sum of squares of each element of
  the list using for loop
# creating the list of numbers
    numbers = [3, 5, 23, 6, 5, 1, 2, 9, 8]
 # initializing a variable that will store the sum
    sum_ = 0
 # using for loop to iterate over the list
    for num in numbers:
        sum_ = sum_ + num ** 2
      print("The sum of squares is: ", sum_)
```

Output:

```
The sum of squares is: 774
```

Q25. Write a Python Program to Print the Fibonacci sequence.

Ans :

# Program to display the Fibonacci sequence up to n-th term where n is provided by the user

```
# change this value for a different result
nterms = 10
# uncomment to take input from the user
#nterms = int(input("How many terms? "))
# first two terms
n1 = 0
n2 = 1
count = 2
# check if the number of terms is valid
if nterms < = 0:
  print("Please enter a positive integer")
elif nterms = = 1:
    print(“Fibonacci sequence upto”,nterms,”:”)
    print(n1)
  else:
    print(“Fibonacci sequence upto”,nterms,”:”)
    print(n1,”,”,n2,end=’, ‘)
    while count < nterms:
        nth = n1 + n2
        print(nth,end=’ , ‘)
        # update values
        n1 = n2
        n2 = nth
        count + = 1
```

Output

```
Fibonacci sequence upto 10 :
0, 1, 1, 2, 3, 5, 8, 13, 21, 34,
```

Q26. Write a program to calculate a running total in python.

Ans :

Calculating a Running Total

```
balance = float(raw_input(“Outstanding Balance: “))
interestRate = float(raw_input(“Interest Rate: “))
minPayRate = float(raw_input(“Minimum Monthly Payment Rate: “))
paid = 0

for month in xrange(1, 12+1):
  interestPaid = round(interestRate / 12.0 * balance, 2)
  minPayment = round(minPayRate * balance, 2)
  principalPaid = round(minPayment - interestPaid, 2)
  remainingBalance = round(balance - principalPaid, 2)
  paid + = minPayment

  print  # Make the output easier to read.
  print ‘Month: %d’ % (month,)
  print ‘Minimum monthly payment: %.2f’ % (minPayment,)
  print ‘Principle paid: %.2f’ % (principalPaid,)
  print ‘Remaining balance: %.2f’ % (remainingBalance,)

  balance = remainingBalance

print
print ‘RESULTS’
print ‘Total amount paid:’, paid
print ‘Remaining balance: %.2f’ % (remainingBalance,)
```

2.3.3 BREAK

Q27. Write about break and continue statements.

Ans :

In Python, break and continue statements can alter the flow of a normal loop.

Loops iterate over a block of code until test expression is false, but sometimes we wish to terminate the current iteration or even the whole loop without cheking test expression.

The break and continue statements are used in these cases.

Python Break Statement

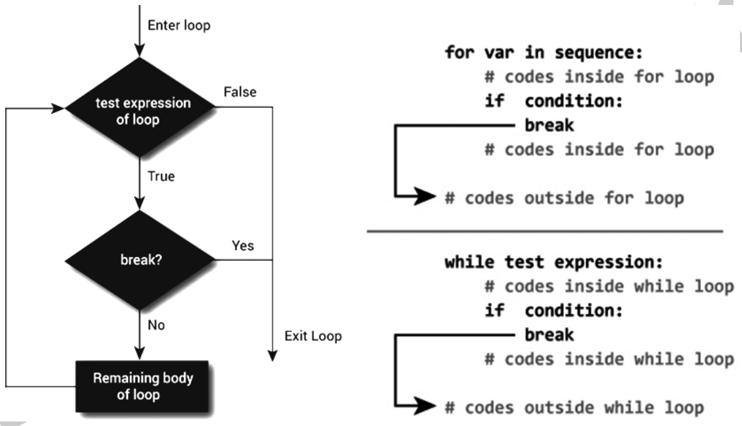
The break statement terminates the loop containing it. Control of the program flows to the statement immediately after the body of the loop.

If break statement is inside a nested loop (loop inside another loop), break will terminate the innermost loop.

Syntax of break

break

Flowchart of break



```
for var in sequence:
    # codes inside for loop
    if  condition:
        break
    # codes inside for loop

# codes outside for loop
```

```
while test expression:
    # codes inside while loop
    if  condition:
        break
    # codes inside while loop

# codes outside while loop
```

The working of break statement in for loop and while loop is shown below.

Example: Python break

```
# Use of break statement inside loop
for val in "string":
    if val = = "i":
        break
    print(val)

print("The end")
```

Output

```
s
t
r
The end
```

In this program, we iterate through the "string" sequence. We check if the letter is "i", upon which we break from the loop. Hence, we see in our output that all the letters up till "i" gets printed. After that, the loop terminates.

2.3.4 Continue

Q28. Explain Python continue statement with example.

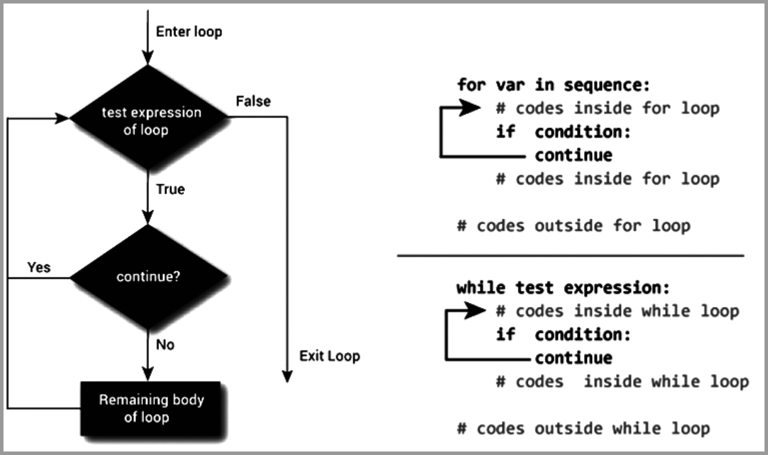
Ans : **(Imp.)**

The continue statement is used to skip the rest of the code inside a loop for the current iteration only. Loop does not terminate but continues on with the next iteration.

Syntax of Continue

continue

Flowchart of continue



```
for var in sequence:
    # codes inside for loop
    if  condition:
        continue
    # codes inside for loop

# codes outside for loop
```

```
while test expression:
    # codes inside while loop
    if  condition:
        continue
    # codes  inside while loop

# codes outside while loop
```

The working of continue statement in for and while loop is shown below.

Example: Python continue

```
# Program to show the use of continue statement inside loops

for val in "string":
  if val = = "i":
    continue
  print(val)
print("The end")
```

Output

```
s
t
r
n
g
The end
```

This program is same as the above example except the break statement has been replaced with continue.

We continue with the loop, if the string is "i", not executing the rest of the block. Hence, we see in our output that all the letters except "i" gets printed.

2.3.5 Pass

Q29. What is Pass Statement in Python?

Ans :

The null statement is another name for the pass statement. A Comment is not ignored by the Python interpreter, whereas a pass statement is not. Hence, they two are different Python keywords.

We can use the pass statement as a placeholder when unsure what code to provide. So, we only have to place the pass on that line. Pass may be used when we don't wish any code to be executed. We can simply insert a pass in places where empty code is prohibited, such as loops, functions, class definitions, or if-else statements.

Syntax of the Pass Keyword

```
Keyword:
    pass
```

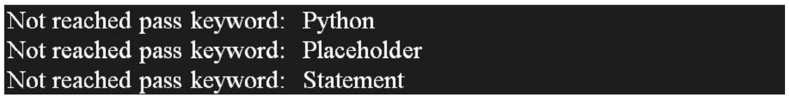
Typically, we utilise it as a reference for the future.

Let's say we have a loop or an if-else statement that isn't to be filled now but that we wish to in the future. The pass keyword cannot have an empty body as it will be syntactically wrong. An error would be displayed by the Python interpreter suggesting to fill the space. Therefore, we create a code block that performs nothing using the pass statement.

Example of the Pass Statement

```
# Python program to show how to use a pass statement in a for loop
""''pass acts as a placeholder. We can fill this place later on''
sequence = {"Python", "Pass", "Statement", "Placeholder"}
for value in sequence:
    if value == "Pass":
        pass # leaving an empty if block using the pass keyword
    else:
        print("Not reached pass keyword: ", value)
```

Output:



```
Not reached pass keyword:  Python
Not reached pass keyword:  Placeholder
Not reached pass keyword:  Statement
```

2.3.6 Fruitful Function Vs Void Function

Q30. Differentiate between fruitful functions and void functions.

Ans : **(Imp.)**

Python provides various functions which may or may not return value.

The function which return any value are called as fruitful function.

The function which does not return any value are called as void function.

Fruitful functions means the function that gives result or return values after execution.

Some functions gets executed but doesn't return any value.

While writing fruitful functions we except a return value and so we must assign it to a variable to hold return value.

In void function you can display a value on screen but cannot return a value.

A void function may or may not have return statement, if void function has return statement then it is written without any expression

Void function without return statement

```
def fun1():
  print("Python")
```

Void function with return statement

```
def fun2():
  print("ITVoyagers")
   return
```

Void function with parameter without return statement

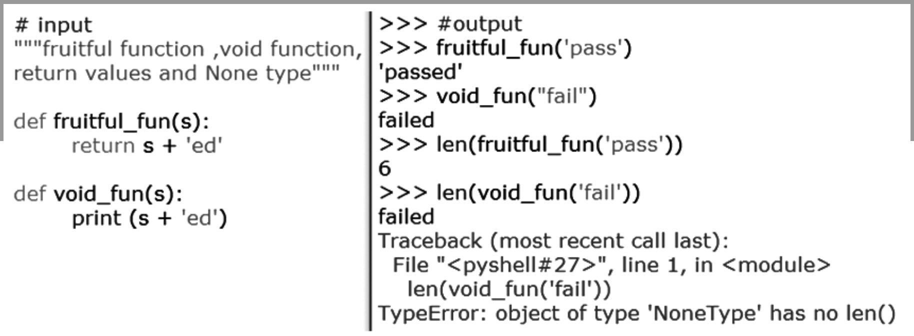
```
def fun1(a,b):
   print("sum of a and b", a+b)
fun1(5,2)
```

In above example main function have passed values to given function and as print statement is used function will not return anything it will just print value.

Fruitful function with parameter and return statement

```
def fun2(a,b):
   return a+b
print(fun2(5,8))
```

In above example we have passed two values to the given function which then returns sum of that values to main function using return keyword.



```
# input
"""fruitful function ,void function,
return values and None type"""

def fruitful_fun(s):
       return s + 'ed'

def void_fun(s):
       print (s + 'ed')
```

```
>>> #output
>>> fruitful_fun('pass')
'passed'
>>> void_fun("fail")
failed
>>> len(fruitful_fun('pass'))
6
>>> len(void_fun('fail'))
failed
Traceback (most recent call last):
  File "<pyshell#27>", line 1, in <module>
    len(void_fun('fail'))
TypeError: object of type 'NoneType' has no len()
```

2.3.7 Parameters/Arguments

Q31. Write about, passing parameters and arguments in python.

Ans : **(Imp.)**

Meaning

Parameters are the variables that we specify inside parentheses at the time of defining a function. In the example, num1 and num2 are function parameters.

Arguments, on the other hand, are the values that are passed for these parameters when calling the function. Arguments allow us to pass information to the function. In the example above, we provided 5 and 6 as the function arguments for parameters num1 and num2, respectively.

Types of Function Arguments in Python

1. Python Default Arguments

Python allows function parameters to have default values. This is useful in case a matching argument is not passed in the function call statement.

```
defsayhello(name = "World"):
print("Hello,", name)
sayhello()
sayhello("Techvidvan")
```

Here, the "sayhello" function's header contains a parameter "name" which has its default value set to the string "World".

As you can see in the function call, the function prints "Hello, World" when no argument is passed. Whereas, if we do pass a value for the argument, the function will print "Hello" followed by the argument value.

The output looks like this:

```
Hello, World
Hello, Techvidvan
>>>
```

One very important thing to remember while dealing with default arguments is :

In a function's definition, a parameter cannot have a default value unless all the parameters to its right have their default values.

2. Python Keyword Arguments

Python offers a way to write any argument in any order if you name the arguments when calling the function. This allows us to have complete control and flexibility over the values sent as arguments for the corresponding parameters.

```
defmultiply(a, b):
return a*b
print(multiply(a = 10, b = 5))
print(multiply(b = 20, a = 9))
```

In the first function call, a gets value 10 and b gets value 5. In the second function call, a gets value 9 and b gets value 20.

The output is:

```
50
180
>>>
```

3. Python Arbitrary Arguments

Arbitrary arguments come in handy when we don't know how many arguments the function will take.

Often at the time of defining the function, we cannot determine the number of arguments our function is going to accept. We place an asterisk (*) before the parameter to denote that the function can take an arbitrary number of arguments.

Example:

```
defsummation(*numbers):
sum1 = 0
for number in numbers:
sum1 + = number
return sum1
print(summation(10,20,30))
```

The function summation takes multiple arguments.

This outputs to the following:

```
60
>>>
```

You can now give any number of arguments while calling the function summation.

2.3.8 Return Values

Q32. Write about the return values in Python

Ans : **(Imp.)**

Meaning

A return statement is used to end the execution of the function call and "returns" the result (value of the expression following the return keyword) to the caller. The statements after the return statements are not executed. If the return statement is without any expression, then the special value None is returned. A return statement is overall used to invoke a function so that the passed statements can be executed.

Syntax:

```
def fun():
  statements
   :
  return [expression]
```

Example:

```
def cube(x):
  r=x**3
  return r
```

Example:

```
# Python program to demonstrate return statement
defadd(a, b):
    # returning sum of a and b
    returna +b
  defis_true(a):
    # returning boolean of a
    returnbool(a)
# calling function
res =add(2, 3)
print("Result of add function is {}".format(res))
  res =is_true(2<5)
print("\nResult of is_true function is {}".format(res))
```

Output:

```
Result of add function is 5
Result of is_true function is True
```

Returning Multiple Values

In Python, we can return multiple values from a function. Following are different ways.

- **Using Object:** This is similar to C/C++ and Java, we can create a class (in C, struct) to hold multiple values and return an object of the class.

Example

```
# A Python program to return multiple  values from a method using class
classTest:
    def__init__(self):
        self.str="geeksforgeeks"
        self.x =20
  # This function returns an object of Test
deffun():
    returnTest()
# Driver code to test above method
t =fun()
print(t.str)
print(t.x)
```

Output

```
geeksforgeeks
20
```

- **Using Tuple:** A Tuple is a comma separated sequence of items. It is created with or without (). Tuples are immutable. See this for details of tuple.

```
# A Python program to return multiple
# values from a method using tuple
# This function returns a tuple
deffun():
    str="geeksforgeeks"
    x =20
    returnstr, x;   # Return tuple, we could also
                    # write (str, x)
# Driver code to test above method
str, x =fun() # Assign returned tuple
print(str)
print(x)
```

Output:

```
geeksforgeeks
20
```

- **Using a list:** A list is like an array of items created using square brackets. They are different from arrays as they can contain items of different types. Lists are different from tuples as they are mutable. See this for details of list.

```
# A Python program to return multiple  values from a method using list
# This function returns a list
deffun():
    str="geeksforgeeks"
    x =20
    return[str, x];
# Driver code to test above method
list=fun()
print(list)
```

Output:

```
['geeksforgeeks', 20]
```

- **Using a Dictionary:** A Dictionary is similar to hash or map in other languages. See this for details of dictionary.

```
        # A Python program to return multiple values
from a method using dictionary

          # This function returns a dictionary
        deffun():
            d =dict();
            d['str'] ="GeeksforGeeks"
            d['x']     =20
            returnd
           # Driver code to test above method
        d =fun()
        print(d)
```

Output:

```
        {'x': 20, 'str': 'GeeksforGeeks'}
```

Function Returning Another Function

In Python, functions are objects so, we can return a function from another function. This is possible because functions are treated as first class objects in Python. To know more about first class objects click here.

In the below example, the create_adder function returns the adder function.

```
        # Python program to illustrate functions can
return another function
          defcreate_adder(x):
              defadder(y):
                    returnx +y
                returnadder
          add_15 =create_adder(15)
          print("The result is", add_15(10))
           # Returning different function
        defouter(x):
              returnx *10
          defmy_func():
                   # returning different function
              returnouter
           # storing the function in res
        res =my_func()

        print("\nThe result is:", res(10))
```

Output:

```
        The result is 25
        The result is: 100
```

2.3.9 Variables Scope (Local Global)

Q33. What is the scope of the variable in python?

Ans :

Scope and Lifetime of Variables

Scope of a variable is the portion of a program where the variable is recognized. Parameters and variables defined inside a function is not visible from outside. Hence, they have a local scope.

Lifetime of a variable is the period throughout which the variable exits in the memory. The lifetime of variables inside a function is as long as the function executes.

They are destroyed once we return from the function. Hence, a function does not remember the value of a variable from its previous calls.

Here is an example to illustrate the scope of a variable inside a function.

```
def my_func():
    x = 10
    print("Value inside function:",x)


x = 20
my_func()
print("Value outside function:",x)
```

Output

```
Value inside function: 10
Value outside function: 20
```

Here, we can see that the value of x is 20 initially. Even though the function my_func()changed the value of x to 10, it did not effect the value outside the function.

This is because the variable x inside the function is different (local to the function) from the one outside. Although they have same names, they are two different variables with different scope.

On the other hand, variables outside of the function are visible from inside. They have a global scope.

We can read these values from inside the function but cannot change (write) them. In order to modify the value of variables outside the function, they must be declared as global variables using the keyword global.

2.3.10 Function Composition

Q34. What is function composition? Explain.

Ans :

Function composition is the way of combining two or more functions in such a way that the output of one function becomes the input of the second function and so on. For example, let there be two functions “F” and “G” and their composition can be represented as F(G(x)) where “x” is the argument and output of G(x) function will become the input of F() function.

Example:

```
# Function to add 2
# to a number
def add(x):
    return x + 2
  # Function to multiply
# 2 to a number
def multiply(x):
    return x * 2

# Printing the result of   composition of add and   multiply to add 2 to a number
# and then multiply by 2
print(“Adding 2 to 5 and multiplying the result with 2: “,
      multiply(add(5)))
```

Output:

```
Adding 2 to 5 and multiplying the result with 2: 14
```

Explanation

First the add() function is called on input 5. The add() adds 2 to the input and the output which is 7, is given as the input to multiply() which multiplies it by 2 and the output is 14.

Short Question and Answers

1. What is function?

Ans :

A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.

As you already know, Python gives you many built-in functions like print(), etc. but you can also create your own functions. These functions are called user-defined functions.

2. Defining a Function

Ans :

You can define functions to provide the required functionality. Here are simple rules to define a function in Python.

- Function blocks begin with the keyword def followed by the function name and parentheses (( )).
- Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.
- The first statement of a function can be an optional statement - the documentation string of the function or docstring.
- The code block within every function starts with a colon (:) and is indented.
- The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

3. Types of Functions

Ans :

Basically, we can divide functions into the following two types:

(i) **Built-in functions:** Functions that are built into Python.

(ii) **User-defined functions:** Functions defined by the users themselves.

4. User-Defined Functions

Ans :

Functions that we define ourselves to do the certain specific task are referred to as user-defined functions. The way in which we define and call functions in Python are already discussed.

Functions that readily come with Python are called built-in functions. If we use functions written by others in the form of the library, it can be termed as library functions.

All the other functions that we write on our own fall under user-defined functions. So, our user-defined function could be a library function to someone else.

5. Advantages of user-defined functions

Ans :

(i) User-defined functions help to decompose a large program into small segments which makes the program easy to understand, maintain and debug.

(ii) If repeated code occurs in a program. The function can be used to include those codes and execute when needed by calling that function.

(iii) Programmers working on a large project can divide the workload by making different functions.

6. Flow of Execution in Python.

Ans :

- The order in which statements are executed is called the flow of execution
- Execution always begins at the first statement of the program.
- Statements are executed one at a time, in order, from top to bottom.
- Function definitions do not alter the flow of execution of the program, but remember that statements inside the function are not executed until the function is called.
- Function calls are like a bypass in the flow of execution. Instead of going to thenext statement, the flow jumps to the first line of the called function, executes all the statements there, and then comes back to pick up where it left off.

7. What are Modules in Python?

Ans :

A document with definitions of functions and various statements written in Python is called a Python module.

In Python, we can define a module in one of 3 ways:

- Python itself allows for the creation of modules.
- Similar to the re (regular expression) module, a module can be primarily written in C programming language and then dynamically inserted at run-time.
- A built-in module, such as the itertools module, is inherently included in the interpreter.

8. Quotations used in python

Ans :

Python accepts single ('), double (") and triple ('' or """) quotes to denote string literals, as long as the same type of quote starts and ends the string.

The triple quotes are used to span the string across multiple lines. For example, all the following are legal -

```
word = 'word'
sentence = "This is a sentence."
paragraph = """This is a paragraph. It is
made up of multiple lines and sentences."""
```

9. Explain nested if statements with syntax and example.

Ans :

Python Nested if statements

We can have a if...elif...else statement inside another if...elif...else statement. This is called nesting in computer programming.

Any number of these statements can be nested inside one another. Indentation is the only way to figure out the level of nesting. This can get confusing, so must be avoided if we can.

Python Nested if Example

```
# In this program, we input a number
# check if the number is positive or
# negative or zero and display
# an appropriate message
# This time we use nested if
num = float(input("Enter a number: "))
if num >= 0:
if num == 0:
```

```
print("Zero")
else:
print("Positive number")
else:
print("Negative number")
```

Output 1

```
Enter a number: 5
Positive number
```

Output 2

```
Enter a number: -1
Negative number
```

Output 3

```
Enter a number: 0
Zero
```

10. What is Pass Statement in Python?

Ans :

The null statement is another name for the pass statement. A Comment is not ignored by the Python interpreter, whereas a pass statement is not. Hence, they two are different Python keywords.

We can use the pass statement as a placeholder when unsure what code to provide. So, we only have to place the pass on that line. Pass may be used when we don't wish any code to be executed. We can simply insert a pass in places where empty code is prohibited, such as loops, functions, class definitions, or if-else statements.

Choose the Correct Answer

1. Which function removes a set's first and the last element from a list? [a]
   (a) pop (b) remove
   (c) dispose (d) discard

2. The output of this Python code would be: [c]
   sum(1,2,3)
   sum([2,4,6])
   (a) 6, 12 (b) Error, Error
   (c) Error, 12 (d) 6, Error

3. Which function doesn't accept any argument? [d]
   (a) re.compile (b) re.findall
   (c) re.match (d) re.purge

4. Which of the following functions is a built-in function in python language? [b]
   (a) val() (b) print()
   (c) print () (d) None of these

5. Which one of the following is a valid Python if statement. [a]
   (a) if a>=2 : (b) if (a >= 2)
   (c) if (a => 22) (d) if a >= 22

6. Which of the following is not used as loop in Python? [c]
   (a) for loop (b) while loop
   (c) do-while loop (d) None of the above

7. Which one of the following is a valid Python if statement : [a]
   (a) if a>=2 : (b) if (a >= 2)
   (c) if (a => 22) (d) if a >= 22

8. What keyword would you use to add an alternative condition to an if statement? [c]
   (a) else if (b) elseif
   (c) elif (d) None of the above

9. Which statement will check if a is equal to b? [b]
   (a) if a = b: (b) if a == b:
   (c) if a === c: (d) if a == b

10. Which of the following is a valid for loop in Python? [b]
    (a) for(i=0; i < n; i++) (b) for i in range(0,5):
    (c) for i in range(0,5) (d) for i in range(5)

Fill in the blanks

1. _________ are used to indicate variables that must not be accessed from outside the class.
2. _________ keyword would you use to add an alternative condition to an if statement?
3. _________ loop is used when multiple statements are to executed repeatedly until the given condition becomes False.
4. _________ determine which statements in the program will be executed and in what order, allowing for statements to be skipped over or executed repeatedly.
5. In Python,. When a statement occurs on a line which is indented less than the previous one, it indicates _________.
6. _________ Keyword can be used to bring control out of the current loop statement is True regarding loops in Python.
7. Else statement after loop will be executed only when the loop condition becomes _________.
8. A loop becomes _________ loop if a condition never becomes FALSE.
9. If the else statement is used with a while loop, the else statement is executed when the condition becomes _________.
10. Python programming language allows to use one loop inside another loop known as _________.

ANSWERS

1. Leading underscores
2. Elif
3. While
4. Control structures
5. The end of a block.
6. Break
7. False
8. Infinite
9. False
10. Nested

UNIT III

Strings: Strings, String slices, Immutability, String functions & Methods, String module; List as array: Array, Methods of array.

Lists: List operations, List slices, List methods, List loops, Mutability, aliasing, Cloning list, List parameters; Tuple: Benefit of Tuple, Operations on Tuple, Tuple methods, Tuple assignment, Tuple as return value, Tuple as argument; Dictionaries: Operations on Dictionary, methods in Dictionary, Difference between List, Tuple and Dictionary; Advanced List processing: List comprehension, Nested List.

3.1 Strings

3.1.1 Strings

Q1. What is string? and how do you create string?

Ans : **(Imp.)**

Meaning

In Python, string is a sequence of Unicode character. Unicode was introduced to include every character in all languages and bring uniformity in encoding.

Creating a string

Strings can be created by enclosing characters inside a single quote or double quotes. Even triple quotes can be used in Python but generally used to represent multiline strings and doc strings.

```
# all of the following are equivalent
my_string = 'Hello'
print(my_string)
my_string = "Hello"
print(my_string)
my_string = '''Hello'''
print(my_string)
# triple quotes string can extend multiple lines
my_string = """Hello, welcome to  the world of
Python"""
print(my_string)
```

When you run the program, the output will be:

```
Hello
Hello
Hello
Hello, welcome to
    the world of Python
```

Access characters in a string

We can access individual characters using indexing and a range of characters using slicing.

Python allows negative indexing for its sequences.

The index of -1 refers to the last item, -2 to the second last item and so on. We can access a range of items in a string by using the slicing operator (colon).

```
str = 'programiz'
print('str = ', str)
#first character
print('str[0] = ', str[0])
#last character
print('str[-1] = ', str[-1])
#slicing 2nd to 5th character
print('str[1:5] = ', str[1:5])
#slicing 6th to 2nd last character
print('str[5:-2] = ', str[5:-2])
```

If we try to access index out of the range or use decimal number, we will get errors.

```
# index must be in range
> > > my_string[15]
...
IndexError:string index out of range
# index must be an integer
> > > my_string[1.5]
...
Type Error:string indices must be integers
```

Slicing can be best visualized by considering the index to be between the elements as shown below.

If we want to access a range, we need the index that will slice the portion from the string.

```
 | P | R | O | G | R | A | M | I | G |
 0   1   2   3   4   5   6   7   8   9
–9  –8  –7  –6  –5  –4  –3  –2  –1
```

Q2. How to change (or) delete a string in python?

Ans :

Strings are immutable. This means that elements of a string cannot be changed once it has been assigned. We can simply reassign different strings to the same name.

```
> > > my_string = 'programiz'
> > > my_string[5] = 'a'
...
TypeError:'str'object does not support item assignment
> > > my_string = 'Python'
> > > my_string
'Python'
```

We cannot delete or remove characters from a string. But deleting the string entirely is possible using the keyword del.

```
> > >del my_string[1]
...
TypeError:'str'object doesn't support item deletion
> > > del my_string
> > > my_string
...
NameError: name 'my_string' is not defined
```

3.1.2 String Slices

Q3. Explain how to do Slicing in Python.

Ans :

You can take subset of string from original string by using [] operator also known as slicing operator.

Syntax: s [start:end]

this will return part of the string starting from index start to index end - 1.

Let's take some examples.

```
> > > s = "Welcome"
> > > s[1:3]
El
```

Some more examples

```
> > > s = "Welcome"
> > > s[ : 6]
'Welcom'
 >>> s[4 : ]
's'
 >>> s[1 : -1]
'elcom'
```

Note: start index and end index are optional. If omitted then the default value of start index is 0 and that of end is the last index of the string.

Q4. Write a program to demonstrate slicing in strings

Ans :

```
s = "abcdefghijklmnopqrs"
# Loop over some indexes.
for n in range(2, 4):
   # Print slices.
  print(n, s[n])
  print(n, s[n:n + 2])
  print(n, s[n:n + 3])
  print(n, s[n:n + 4:2])
  print(n, s[n:n + 6:2])
```

Output

```
2 c
2 cd
2 cde
2 ce
2 ceg
3 d
3 de
3 def
3 df
3 dfh
```

3.1.3 Immutability

Q5. What are called immutable data types in python? Explain.

Ans : **(Imp.)**

In python, the string data types are immutable. Which means a string value cannot be updated. We can verify this by trying to update a part of the string which will led us to an error.

```
# Can not reassign
t= "Tutorialspoint"
print type(t)
t[0] = "M"
```

When we run the above program, we get the following output -

```
t[0] = "M"
TypeError: 'str' object does not support item assignment
```

We can further verify this by checking the memory location address of the position of the letters of the string.

```
x = 'banana'
for idx in range (0,5):
    print x[idx], "=", id(x[idx])
```

When we run the above program we get the following output. As you can see above a and a point to same location. Also N and N also point to the same location.

```
b = 91909376
a = 91836864
n = 91259888
a = 91836864
n = 91259888
```

3.1.4 String Functions & Methods

Q6. Explain various String Manipulation Functions.

Ans : **(Imp.)**

1. **capitalize()**

This function returns the copy of the string passed changing the first character of the string to uppercase.

Syntax:

str.capitalize()

Example:

```
str = "lets test the function";
print "str.capitalize() : ", str.capitalize() // change the first character to uppercase
```

Output:

```
str.capitalize() :   Lets test the function
```

2. **is lower()**

This method checks if the string is in lowercase and returns true if all the characters are in lowercase.

Syntax:

str.islower()

Example:

```
str = "lets test the function";
print str.islower();// Returns true since all characters are in lower case.
str = "lets Test the function";
print str.islower();
```

Out put:

```
True
False
```

3. **isupper()**

This method checks if all the characters in the string are in uppercase. If any character is in lower case, it would return false otherwise true.

Syntax:

str.isupper()

Example:

```
str = "LETS TEST THE FUNCTION";
print str.isupper();//returns true since all characters are capital
str = "LETS TEST THE FUNCTIOn";
print str.isupper(); // Returns false as 'n' is small.
```

Output

```
True
False
```

4. lower()

This method returns a string after converting every character of the string into lower case.

Syntax :

str.lower()

Example:

str = "LETS TEST THE FUNCTION";

print str.lower();//converts the string to lowercase

Output

lets test the function

5. upper()

This method returns string after converting every character of string into lowercase

Syntax:

str.upper()

Example:

str = "lets test the function";

print str.upper(); //Converts the string to uppercase

Output:

LETS TEST THE FUNCTION

6. swapcase()

This method swaps the case of every character i.e. every uppercase is converted to lowercase and vice versa.

Syntax:

str.swapcase()

Example:

str = "LETS test THE function";

print str.swapcase(); //swaps the cases

Output

lets TEST the FUNCTION

7. len()

This methods returns the count of the total number of characters present in the string. It takes a string as argument. This string is the string length you want to calculate.

Syntax:

len(str)

Example:

str2 = "Hello"

print "The length of string is "len(str2);//Returns length of the string

Output:

The length of string is 5

8. split()

This function splits the string given as argument on the basis of the separator provided. If nothing is provided as argument, then it splits based on whitespaces.

Syntax:

str.split()

Example:

str = "Lets test the function";

print str.split() //splits by every space it encounters

print str.split('the')// splits by ever "the" it encounters

Output:

['Lets', 'test', 'the', 'function']

['Lets test ', ' function']

9. replace()

This method returns the copy of the string in which a certain word is replaced by the given word.

Syntax:

str.replace(old,new,max)

- ➢ old: The substring to replace
- ➢ new: This is the substring which is to be replaced in place of the old substring.
- ➢ max: This arguments defines how many substrings would be replaced.

Example:

str = "Lets test the function and the test should be good";

print str.replace('test','changed')//Changes the value of 'test' to 'changed'

Output

Lets change the function and the changed one should be good

10. count()

The count method returns the count of occurrence of the substring in the string.

Syntax:

str.count(sub,start,end)

- ➢ sub: This is the string to search.
- ➢ start. Starting index of the search.
- ➢ end: End index of the search

Example:

str = "Lets test the function and the test should be good";

sub="t"

print "Number of t are",str.count(sub, 1, 20) //Counts total number of 't' present

Output

Number of t are 5

11. lstrip()

This method returns the string after removing all the characters from the beginning of the string.

Syntax:

str.lstrip([chars])

Example:

str = " Lets test the function and the test should be good";

str1 = "000000000Lets test the function and the test should be good000000";

print str.lstrip(' '); // Removes all the beginning spaces.

print str1.lstrip('0');// Removes all the zeros from starting

Output

Lets test the function and the test should be good

Lets test the function and the test should be good000000

12. rstrip()

This method returns the string after removing all the characters from the end of the string.

Syntax:

str.rstrip([chars])

Example:

str = "Lets test the function and the test should be good ";

str1 = "000000000Lets test the function and the test should be good000000";

print str.rstrip(' '); // Removes all the end spaces.

print str1.rstrip('0');// Removes all the zeros from the end

Output

Lets test the function and the test should be good

000000000Lets test the function and the test should be good

13. rfind()

This method returns the last index of the substring found or otherwise -1 if the substring is not present.

Syntax: str.rfind(str, beg,end)

- sub: This is the string to search.
- start. Starting index of the search.
- end: End index of the search

Example:

str = " Lets test the function and the test should be good";

print str.rfind(str);//String matches with itself

print str.rfind(str, 0, 10); //First 10 character matches only hence -1

print str.rfind('test', 0, 20);// Found at index 7

print str.rfind('and', 0, 30); // Found at index 25

Out put :

0

-1

7

25

14. is digit()

The method checks whether the strings consist only of digits and returns true or false accordingly.

Syntax:

str.isdigit()

Example:

str = " 1231231Lets test the function";

print str.isdigit(); //Returns false since string contains alphanumeric characters

str="12390877"

print str.isdigit();// Returns true since all characters are digits

Output

False

True

15. join()

This method is used to join the string based on the separator given.

Syntax:

str.join(seq)

Example:

str= "-"

seq=("This", "string", "will", "be", "joined")

print str.join(seq) // The string is joined by the dash.

3.1.5 String Module

Q7. Explain briefly about Python String Module.

Ans : **(Imp.)**

It's a built-in module and we have to import it before using any of its constants and classes.

String Module Constants

Let's look at the constants defined in the string module.

```
import string
# string module constants
print(string.ascii_letters)
print(string.ascii_lowercase)
print(string.ascii_uppercase)
print(string.digits)
print(string.hexdigits)
print(string.whitespace)  # ' \t\n\r\x0b\x0c'
print(string.punctuation)
```

Output:

```
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
0123456789
0123456789abcdefABCDEF
!"#$%&'()*+,-./:;?@[\]^_'{|}~
```

string capwords() function

Python string module contains a single utility function - capwords(s, sep=None). This function split the specified string into words using str.split(). Then it capitalizes each word using str.capitalize() function. Finally, it joins the capitalized words using str.join(). If the optional argument sep is not provided or None, then leading and trailing whitespaces are removed and words are separated with single whitespace. If it's provided then the separator is used to split and join the words.

```
s = '  Welcome TO  \n\n JournalDev '
print(string.capwords(s))
```

Output : Welcome To Journaldev

Python String Module Classes

Python string module contains two classes - Formatter and Template.

Formatter

It behaves exactly same as str.format() function. This class become useful if you want to subclass it and define your own format string syntax. Let's look at a simple example of using Formatter class.

```
from string import Formatter
formatter = Formatter()
print(formatter.format('{website}', website='JournalDev'))
print(formatter.format('{} {website}', 'Welcome to', website='JournalDev'))
# format() behaves in similar manner
print('{} {website}'.format('Welcome to', website='JournalDev'))
```

Out put:

```
Welcome to Journal Dev
Welcome to Journal Dev
```

3.1.6 List As Array

3.1.6.1 Array

Q8. Define array? Explain about array operations

Ans : **(Imp.)**

Array is an idea of storing multiple items of the same type together and it makes easier to calculate the position of each element by simply adding an offset to the base value. A combination of the arrays could save a lot of time by reducing the overall size of the code. It is used to store multiple values in single variable. If you have a list of items that are stored in their corresponding variables like this:

```
car1 = "Lamborghini"
car2 = "Bugatti"
car3 = "Koenigsegg"
```

If you want to loop through cars and find a specific one, you can use the array.

The array can be handled in Python by a module named array. It is useful when we have to manipulate only specific data values. Following are the terms to understand the concept of an array:

Element - Each item stored in an array is called an element.

Index - The location of an element in an array has a numerical index, which is used to identify the position of the element.

Array Representation

An array can be declared in various ways and different languages. The important points that should be considered are as follows:

- Index starts with 0.
- We can access each element via its index.
- The length of the array defines the capacity to store the elements.

Array operations

Some of the basic operations supported by an array are as follows:

- **Traverse** - It prints all the elements one by one.
- **Insertion** - It adds an element at the given index.
- **Deletion** - It deletes an element at the given index.
- **Search** - It searches an element using the given index or by the value.
- **Update** - It updates an element at the given index.

The Array can be created in Python by importing the array module to the python program.

```
from array import *
arrayName = array(typecode, [initializers])
```

Accessing array elements

We can access the array elements using the respective indices of those elements.

```
import array as arr
a = arr.array('i', [2, 4, 6, 8])
print("First element:", a[0])
print("Second element:", a[1])
print("Second last element:", a[-1])
```

Output :

```
First element: 2
Second element: 4
Second last element: 8
```

deleting elements from an array

The elements can be deleted from an array using Python's del statement. If we want to delete any value from the array, we can do that by using the indices of a particular element.

```
import array as arr
number = arr.array('i', [1, 2, 3, 3, 4])
del number[2]
# removing third element
print(number)
# Output: array('i', [1, 2, 3, 4])
```

Output :

```
array('i', [10, 20, 40, 60])
```

Finding the length of an array

The length of an array is defined as the number of elements present in an array. It returns an integer value that is equal to the total number of the elements present in that array.

Syntax

len(array_name)

Array Concatenation

We can easily concatenate any two arrays using the + symbol.

Example

a=arr.array('d',[1.1, 2.1, 3.1, 2.6, 7.8])

b=arr.array('d',[3.7,8.6])

c=arr.array('d')

c=a+b

print("Array c = ",c)

Out put :

Array c= array('d', [1.1, 2.1, 3.1, 2.6, 7.8, 3.7, 8.6])

3.1.7 Methods of Array

Q9. Explain various array methods in Python.

Ans : **(Imp.)**

Python has a set of built-in methods that you can use on lists/arrays.

1. **append()**- Adds an element at the end of the list

   Syntax : list.append(elmnt)

   Example

   Add a list to a list:

   a = ["apple", "banana", "cherry"]

   b = ["Ford", "BMW", "Volvo"]

   a.append(b)

2. **clear()** - Removes all the elements from the list

   Syntax : list.clear()

   Example

   Remove all elements from the fruits list:

   fruits = ['apple', 'banana', 'cherry', 'orange']

   fruits.clear()

3. **copy() -** Returns a copy of the list

   Syntax : list.copy()

   Example

   Copy the fruits list:

   fruits = ['apple', 'banana', 'cherry', 'orange']

   x = fruits.copy()

4. **count()** - Returns the number of elements with the specified value

   Syntax : list.count(value)

   Return the number of times the value "cherry" appears in the fruits list:

   fruits = ['apple', 'banana', 'cherry']

   x = fruits.count("cherry")

5. **extend()** - Add the elements of a list (or any iterable), to the end of the current list

   Syntax :list.extend(iterable)

   Example

   Add a tuple to the fruits list:

   fruits = ['apple', 'banana', 'cherry']

   points = (1, 4, 5, 9)

   fruits.extend(points)

6. **index()** - Returns the index of the first element with the specified value

   Syntax : list.index (elmnt)

   Example

   What is the position of the value 32:

   fruits = [4, 55, 64, 32, 16, 32]

   x = fruits.index (32)

7. **insert()** - Adds an element at the specified position

   Syntax : list.insert(pos, elmnt)

   Insert the value "orange" as the second element of the fruit list:

   fruits = ['apple', 'banana', 'cherry']

   fruits.insert(1, "orange")

8. **pop()** - Removes the element at the specified position

   Syntax : list.pop(pos)

   eturn the removed element:

   fruits = ['apple', 'banana', 'cherry']

   x = fruits.pop(1)

9. **remove()** -Removes the first item with the specified value

   Syntax : list.remove(elmnt)

   Remove the "banana" element of the fruit list:

fruits = ['apple', 'banana', 'cherry']

fruits.remove("banana")

10. **reverse()** - Reverses the order of the list

Syntax :list.reverse()

Reverse the order of the fruit list:

fruits = ['apple', 'banana', 'cherry']

fruits.reverse()

11. **sort()** - Sorts the list

Syntax : list.sort(reverse=True|False, key = my Func)

Example

Sort the list alphabetically:

cars = ['Ford', 'BMW', 'Volvo']

cars. sort ()

3.2 Lists

3.2.1 List Operations

Q10. What are lists ? How to create a list ?

Ans :

Python offers a range of compound data types often referred to as sequences. List is one of the most frequently used and very versatile data type used in Python.

Creating a list

In Python programming, a list is created by placing all the items (elements) inside a square bracket [ ], separated by commas.

It can have any number of items and they may be of different types (integer, float, string etc.).

```
# empty list
my_list = []
# list of integers
my_list = [1,2,3]
# list with mixed datatypes
my_list = [1,"Hello",3.4]
```

Also, a list can even have another list as an item. This is called nested list.

```
# nested list
my_list = ["mouse", [8, 4, 6], ['a']]
```

Q11. Write about indexing in lists.

Or

How to access elements from a list?

Ans : **(Imp.)**

There are various ways in which we can access the elements of a list. Indexing is one way to access the list.

List Index

We can use the index operator [] to access an item in a list. Index starts from 0. So, a list having 5 elements will have index from 0 to 4.

Trying to access an element other that this will raise an IndexError. The index must be an integer. We can't use float or other types, this will result into TypeError.

Nested list are accessed using nested indexing.

```
my_list = ['p','r','o','b','e']
# Output: p
print(my_list[0])
# Output: o
print(my_list[2])
# Output: e
print(my_list[4])
# Error! Only integer can be used for indexing
# my_list[4.0]
# Nested List
n_list = ["Happy", [2,0,1,5]]
# Nested indexing
# Output: a
print(n_list[0][1])
# Output: 5
print(n_list[1][3])
```

Negative indexing

Python allows negative indexing for its sequences. The index of -1 refers to the last item, -2 to the second last item and so on.

```
my_list = ['p','r','o','b','e']
# Output: e
print(my_list[-1])
```

```
# Output: p
print(my_list[-5])
```

Updating Lists

You can update single or multiple elements of lists by giving the slice on the left-hand side of the assignment operator, and you can add to elements in a list with the append() method. For example "

```
#!/usr/bin/python
list = ['physics','chemistry',1997,2000];
print"Value available at index 2 : "
print list[2]
list[2] = 2001;
print"New value available at index 2 : "
print list[2]
```

Note: append() method is discussed in subsequent section.

When the above code is executed, it produces the following result -

```
Value available at index 2 :
1997
New value available at index 2 :
2001
```

The concatenation (+) and repetition (*) operators work in the same way as they were working with the strings.

Iterating a List

A list can be iterated by using a for - in loop. A simple list containing four strings, which can be iterated as follows.

```
list = ["John", "David", "James", "Jonathan"]
for i in list:
# The i variable will iterate over the elements of the List and contains each element in each iteration.
    print(i)
```

Output:

```
John
David
James
Jonathan
```

Adding elements to the list

Python provides append() function which is used to add an element to the list. However, the append() function can only add value to the end of the list.

Consider the following example in which, we are taking the elements of the list from the user and printing the list on the console.

```
#Declaring the empty list
l =[]
# Number of elements will be entered by the user
n = int(input("Enter the number of elements in the list:"))
# for loop to take the input
for i in range(0,n):
# The input is taken from the user and added to the list as the item
l.append(input("Enter the item:"))
print("printing the list items..")
# traversal loop to print the list items
for i in l:
print(i, end = "  ")
```

Output:

```
Enter the number of elements in the list:5
Enter the item:25
Enter the item:46
Enter the item:12
Enter the item:75
Enter the item:42
printing the list items
25 46 12 75 42
```

Removing elements from the list

Python provides the remove() function which is used to remove the element from the list. Consider the following example to understand this concept.

Example -

```
list = [0,1,2,3,4]
print("printing original list: ");
for i in list:
    print(i,end=" ")
     list.remove(2)
     print("\nprinting the list after the removal of first element...")
     for i in list:
    print(i,end=" ")
```

Out put:

```
printing original list:
0 1 2 3 4
printing the list after the removal of first element...
0 1 3 4
```

Let's see how the list responds to various operators.

Consider a Lists l1 = [1, 2, 3, 4], and l2 = [5, 6, 7, 8] to perform operation.

Operator	Description	Example
Repetition	The repetition operator enables the list elements to be repeated multiple times.	L1*2 = [1, 2, 3, 4, 1, 2, 3, 4]
Concatenation	It concatenates the list mentioned on either side of the operator.	11 + 12 = [1, 2, 3, 4, 5, 6, 7, 8]
Membership	It returns true if a particular item exists in a particular list otherwise false.	print(2 in l1) prints True.
Iteration	The for loop is used to iterate over the list elements.	for i in l1: print (i) **Output** 1 2 3 4
Length	It is used to get the length of the list	len(l1) = 4

3.2.2 LIST SLICES

Q12. How to slice lists in Python?

(OR)

What is list slicing?

Ans :

List Slicing

Python has an amazing feature just for that called *slicing*. Slicing can not only be used for lists, tuples or arrays, but custom data structures as well, with the *slice* object, which will be used later on in this article.

We can access a range of items in a list by using the slicing operator (colon).

Like the list data type that has items that correspond to an index number, each of a string's characters also correspond to an index number, starting with the index number 0.

For the string Sammy Shark! the index breakdown looks like this:

S	A	M	M	Y		S	h	a	r	k	!
0	1	2	3	4	5	6	7	8	9	10	11

As you can see, the first S starts at index 0, and the string ends at index 11 with the ! symbol.

We also notice that the whitespace character between Sammy and Shark also corresponds with its own index number. In this case, the index number associated with the whitespace is 5.

The exclamation point (!) also has an index number associated with it. Any other symbol or punctuation mark, such as *#$&.;?, is also a character and would be associated with its own index number.

The fact that each character in a Python string has a corresponding index number allows us to access and manipulate strings in the same ways we can with other sequential data types.

Accessing Characters by Positive Index Number

By referencing index numbers, we can isolate one of the characters in a string. We do this by putting the index numbers in square brackets. Let's declare a string, print it, and call the index number in square brackets:

```
ss = "Sammy Shark!"
print(ss[4])
```

Output

```
y
```

When we refer to a particular index number of a string, Python returns the character that is in that position. Since the letter y is at index number 4 of the string ss = "Sammy Shark!", when we print ss[4] we receive y as the output.

Index numbers allow us to access specific characters within a string.

Accessing Characters by Negative Index Number

If we have a long string and we want to pinpoint an item towards the end, we can also count backwards from the end of the string, starting at the index number -1.

For the same string Sammy Shark! the negative index breakdown looks like this:

S	A	M	M	Y		S	h	a	r	k	!
-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

By using negative index numbers, we can print out the character r, by referring to its position at the -3 index, like so:

print(ss[-3])

Output

r

Using negative index numbers can be advantageous for isolating a single character towards the end of a long string.

Syntax:

```
a[start:end] # items start through end-1
a[start:] # items start through the rest of the array
a[:end]  # items from the beginning through end-1
a[:] # a copy of the whole array
```

There is also the step value, which can be used with any of the above:

```
a[-1]    # last item in the array
a[-2:]   # last two items in the array
a[:-2]   # everything except the last two items
```

Python is kind to the programmer if there are fewer items than you ask for. For example, if you ask for a[:-2] and a only contains one element, you get an empty list instead of an error. Sometimes you would prefer the error, so you have to be aware that this may happen.

For Example consider the list

```
>>> a = [1, 2, 3, 4, 5, 6, 7, 8]
```

Advanced Python Slicing (Increments)

There is also an optional second clause that we can add that allows us to set how the list's index will increment between the indexes that we've set.

Syntax :

a[start:end:step] # start through not past end, by step

The key point to remember is that the :end value represents the first value that is *not* in the selected slice. So, the difference between end and start is the number of elements selected (if step is 1, the default).

The other feature is that start or end may be a negative number, which means it counts from the end of the array instead of the beginning. So:

In the example above, say that we did not want that 3 returned and we only want nice, even numbers in our list.

```
>>>a[1:4:2]
[2,4]
```

Python Slicing in Reverse

Alright, how about if we wanted our list to be backwards use negative index.

```
>>>a[::-1]
[8,7,6,5,4,3,2,1]
```

Example program for list slicing

```
my_list = ['p','r','o','g','r','a','m','i','z']
# elements 3rd to 5th
print(my_list[2:5])
# elements beginning to 4th
print(my_list[:-5])
# elements 6th to end
print(my_list[5:])
# elements beginning to end
print(my_list[:])
```

3.2.3 List Methods

Q13. Write about various methods used in lists.

Ans : **(Imp.)**

Python List Methods

Methods that are available with list object in Python programming are tabulated below.

They are accessed as list.method(). Some of the methods have already been used above.

1. Append() method

The append() method adds an item to the end of the list.

The append() method adds a single item to the existing list. It doesn't return a new list; rather it modifies the original list.

The syntax of append() method is:

list.append(item)

append() Parameters

The append() method takes a single item & and adds it to the end of the list.

The item can be numbers, strings, another list, dictionary etc.

Return Value from append()

As mentioned, the append() method only modifies the original list. It doesn't return any value.

Example 1: Adding Element to a List

```
# animal list
animal = ['cat', 'dog', 'rabbit']
# an element is added
animal.append('guinea pig')
#Updated Animal List
print('Updated animal list: ', animal)
```

2. Extend() method

The extend() extends the list by adding all items of a list (passed as an argument) to the end.

The syntax of extend() method is:

list1.extend(list2)

Here, the elements of list2 are added to the end of list1.

extend() Parameters

As mentioned, the extend() method takes a single argument (a list) and adds it to the end.

If you need to add elements of other native datatypes (like tuple and set) to the list, you can simply use:

```
# add elements of a tuple to list
list.extend(list(tuple_type))
```

or even easier

list.extend(tuple_type)

Return Value from extend()

The extend() method only modifies the original list. It doesn't return any value.

Example 1: Using extend() Method

```
# language list
language = ['French', 'English', 'German']
# another list of language
language1 = ['Spanish', 'Portuguese']
language.extend(language1)
# Extended List
print('Language List: ', language)
```

3. Insert() method

The insert() method inserts the element to the list at the given index.

The syntax of insert() method is

list.insert(index, element)

insert() Parameters

The insert() function takes two parameters:

- index - position where element needs to be inserted
- element - this is the element to be inserted in the list

Return Value from insert()

The insert() method only inserts the element to the list. It doesn't return any value.

Example 1: Inserting Element to List

```
# vowel list
vowel = ['a', 'e', 'i', 'u']
# inserting element to list at 4th position
vowel.insert(3, 'o')
print('Updated List: ', vowel)
```

4. Remove() method

The remove() method searches for the given element in the list and removes the first matching element.

The syntax of remove() method is:

list.remove(element)

remove() Parameters

The remove() method takes a single element as an argument and removes it from the list.

If the element(argument) passed to the remove() method doesn't exist, valueError exception is thrown.

Return Value from remove()

The remove() method only removes the given element from the list. It doesn't return any value.

Example 1: Remove Element From The List

```
# animal list
animal = ['cat', 'dog', 'rabbit', 'guinea pig']
# 'rabbit' element is removed
animal.remove('rabbit')
#Updated Animal List
print('Updated animal list: ', animal)
```

5. pop() method

The pop() method removes and returns the element at the given index (passed as an argument) from the list.

The syntax of pop() method is:list.pop(index)

➢ **pop() parameter**

The pop() method takes a single argument (index) and removes the element present at that index from the list.

If the index passed to the pop() method is not in the range, it throws IndexError: pop index out of range exception.

The parameter passed to the pop() method is optional. If no parameter is passed, the default index -1 is passed as an argument which returns the last element.

➢ **Return Value from pop()**

The pop() method returns the element present at the given index.

Also, the pop() method removes the element at the given index and updates the list.

Example 1:

Print Element Present at the Given Index from the List

```
# programming language list
language = ['Python', 'Java', 'C++', 'French', 'C']
# Return value from pop()
# When 3 is passed
return_value = language.pop(3)
print('Return Value: ', return_value)
# Updated List
print('Updated List: ', language)
```

Python List clear()

6. Index() method

The index() method searches an element in the list and returns its index.

In simple terms, index() method finds the given element in a list and returns its position.

However, if the same element is present more than once, index() method returns its smallest/ first position.

Note: Index in Python starts from 0 not 1.

The syntax of index() method for list is:

list.index(element)

index() Parametersx

The index method takes a single argument:

➢ element - element that is to be searched.

Return value from index()

The index() method returns the index of the element in the list.

If not found, it raises a ValueError exception indicating the element is not in the list

Example 1: Find position of element in the list

```
# vowels list
vowels = ['a', 'e', 'i', 'o', 'i', 'u']
# element 'e' is searched
index = vowels.index('e')
# index is printed
print('The index of e:', index)
# element 'i' is searched
index = vowels.index('i')
# only the first index of the element is printed
print('The index of i:', index)
```

7. Count() method

The count() method returns the number of occurrences of an element in a list.

In simple terms, count() method counts how many times an element has occurred in a list and returns it.

The syntax of count() method is:

list.count(element)

count() Parameters

The count() method takes a single argument

- element - element whose count is to be found.

Return value from count()

The count() method returns the number of occurrences of an element in a list.

Example 1:

Count the occurrence of an element in the list

```
# vowels list
vowels = ['a', 'e', 'i', 'o', 'i', 'u']
# count element 'i'
count = vowels.count('i')
# print count
print('The count of i is:', count)
# count element 'p'
count = vowels.count('p')
# print count
print('The count of p is:', count)
```

8. Sort() method

The sort() method sorts the elements of a given list.

The sort() method sorts the elements of a given list in a specific order - Ascending or Descending.

The syntax of sort() method is:list.sort(key=..., reverse=...)

Alternatively, you can also use Python's in-built function sorted() for the same purpose.

sorted(list, key=..., reverse=...)

Note: Simplest difference between sort() and sorted() is: sort() doesn't return any value while, sorted() returns an iterable list.

sort() Parameters

By default, sort() doesn't require any extra parameters. However, it has two optional parameters:

- reverse - If true, the sorted list is reversed (or sorted in Descending order)
- key - function that serves as a key for the sort comparison

Return value from sort()

sort() method doesn't return any value. Rather, it changes the original list.

If you want the original list, use sorted().

Example 1: Sort a given list

```
# vowels list
vowels = ['e', 'a', 'u', 'o', 'i']
# sort the vowels
vowels.sort()
# print vowels
print('Sorted list:', vowels)
```

9. reverse() method

The reverse() method reverses the elements of a given list.

The syntax of reverse() method is:

list.reverse()

reverse() parameter

The reverse() function doesn't take any argument.

Return Value from reverse()

The reverse() function doesn't return any value. It only reverses the elements and updates the list.

Example 1: Reverse a List

```
# Operating System List
os = ['Windows', 'macOS', 'Linux']
print('Original List:', os)
# List Reverse
os.reverse()
# updated list
print('Updated List:', os)
```

9. Copy() method

The copy() method returns a shallow copy of the list.

A list can be copied with = operator. For example:

```
old_list = [1, 2, 3]
 new_list = old_list
```

The problem with copying the list in this way is that if you modify the new_list, the old_list is also modified.

```
old_list = [1, 2, 3]
new_list = old_list
# add element to list
new_list.append('a')
print('New List:', new_list )
print('Old List:', old_list )
```

3.2.4 List Loops

Q14. Explain how do you perform iterations in loops.

Ans : **(Imp.)**

Iterating Through a List

However, programs also need to do repetitive things very quickly. We are going to use a for-loop in this exercise to build and print various lists.

Before you can use a for-loop, you need a way to store the results of loops somewhere. The best way to do this is with lists. Lists are exactly what their name says: a container of things that are organized in order from first to last

Syntax:

```
hairs = ['brown', 'blond', 'red']
eyes = ['brown', 'blue', 'green']
weights = [1, 2, 3, 4]
```

You start the list with the [(left bracket) which "opens" the list. Then you put each item you want in the list separated by commas, similar to function arguments. Lastly, end the list with a] (right bracket) to indicate that it's over. Python then takes this list and all its contents and assigns them to the variable.

We now will build some lists using some for-loops and print them out:

```
the_count=[1,2,3,4,5]
fruits=['apples','oranges','pears','apricots']
change=[1,'pennies',2,'dimes',3,'quarters']
# this first kind of for-loop goes through a list
fornumberinthe_count:
print"This is count %d"%number
# same as above
forfruitinfruits:
print"A fruit of type: %s"%fruit
# also we can go through mixed lists too
# notice we have to use %r since we don't know what's in it
foriinchange:
print"I got %r"%i
# we can also build lists, first start with an empty one
elements=[]
# then use the range function to do 0 to 5 counts
foriinrange(0,6):
print"Adding %d to the list."%i
# append is a function that lists understand
elements.append(i)
# now we can print them out too
foriinelements:
print"Element was: %d"%i
$ python ex32.py
```

Output:

```
This is count 1
This is count 2
This is count 3
This is count 4
This is count 5
A fruit of type: apples
A fruit of type: oranges
A fruit of type: pears
A fruit of type: apricots
I got 1
I got 'pennies'
I got 2
I got 'dimes'
I got 3
I got 'quarters'
Adding 0 to the list.
Adding 1 to the list.
Adding 2 to the list.
```

Adding 3 to the list.

Adding 4 to the list.

Adding 5 to the list.

Element was: 0

Element was: 1

Element was: 2

Element was: 3

Element was: 4

Element was: 5

3.2.5 Mutability

Q15. Explain about Python List Mutability.

Ans :

A single list in Python may involve various data types like strings, integers, and objects. Furthermore, its alteration is possible even after its creation due to its mutable nature.

Mutable lists in Python have a definite count and they are also ordered. Furthermore, the indexing of a list of elements takes place in accordance with a definite sequence. Moreover, the indexing of a list takes place with 0 being the first index.

Each list's element has its definite place in the list. Furthermore, this allows duplicating of elements that are present in the list. As such, each element in the list has its own distinct credibility and place.

#Creating a list which contains name of Indian cities

cities = ['Delhi', 'Mumbai', 'Kolkata']

# Printing the elements from the list cities, separated by a comma & space

for city in cities:

    print(city, end=', ')

Output [1]: Delhi, Mumbai, Kolkata

#Printing the location of the object created in the memory address in hexadecimal format

print(hex(id(cities)))

Output [2]: 0x1691d7de8c8

#Adding a new city to the list cities

cities.append('Chennai')

#Printing the elements from the list cities, separated by a comma & space

for city in cities:

    print(city, end=', ')

Output [3]: Delhi, Mumbai, Kolkata, Chennai

#Printing the location of the object created in the memory address in hexadecimal format

print(hex(id(cities)))

Output [4]: 0x1691d7de8c8

The above example shows us that we were able to change the internal state of the object 'cities' by adding one more city 'Chennai' to it, yet, the memory address of the object did not change. This confirms that we did not create a new object, rather, the same object was changed or mutated. Hence, we can say that the object which is a type of list with reference variable name 'cities' is a MUTABLE OBJECT.

3.2.6 Aliasing

Q16. Explain Aliasing in lists.

Ans :

In python programming, the second name given to a piece of data is known as an alias. Aliasing happens when the value of one variable is assigned to another variable because variables are just names that store references to actual value.

Consider a following example:

first_variable = "PYTHON"

print("Value of first:", first_variable)

print("Reference of first:", id(first_variable))

print("..................")

second_variable = first_variable # making an alias

print("Value of second:", second_variable)

print("Reference of second:", id(second_variable))

In the example above, first_variable is created first and then string 'PYTHON' is assigned to it. Statement first_variable = second_variable creates an alias of first_variable because first_variable = second_variable copies reference of first_variable to second_variable.

To verify, let's have look at the output of the above program:

Value of first_variable: PYTHON

Reference of first_variable: 2904215383152

....

```
Value of second_variable: PYTHON
Reference of second_variable: 2904215383152
```

From the output of the above program, it is clear that first_variable and second_variable have the same reference id in memory.

So, both variables point to the same string object 'PYTHON'.

And in Python programming, when the value of one variable is assigned to another variable, aliasing occurs in which reference is copied rather than copying the actual value.

3.2.7 Cloning List

Q17. What is list Cloning? Explain list cloning techniques.

Ans :

Cloning or copying a list is simply creating another list that has the same elements as the original list. Elements in a list can be copied into another list by various methods, we will be discussing most of them in this tutorial. For example,

Input: Original list- [3, 6, 12, 14, 78, 24, 56]

Output: After cloning- [3, 6, 12, 14, 78, 24, 56]

Input: Original list- [14, 7, 9, 13, 46, 12]

Output: After cloning- [14, 7, 9, 13, 46, 12]

For cloning a list in Python we can follow these approaches-

1. By following the Slicing technique
2. By using extend() method
3. By using list() method
4. By using list comprehension
5. By using append() method
6. By using copy() method

1. Slicing Technique

We will use list slicing technique to access and copy the list elements in another list. This is the easiest method to modify any list and make a copy of the list along with the reference. This method takes about 0.039 seconds and is the fastest in cloning a list.

Algorithm

Follow the algorithm to understand the approach better.

Step 1- Take input of list from user

Step 2- Declare a new list which will be the copy

Step 3- Copy the elements using slicing operator (:) in the new list

Step 4- Print the new list which will be the copy of the original list

Example 1

Look at the program to understand the implementation of the above-mentioned approach. In this program, we have taken an input of elements of the list and copied the list to a new list by slicing.

```
li=[]
n=int(input("Enter size of list "))
for i inrange(0,n):
e=int(input("Enter element of list "))
 li.append(e)
print("Original list: ",li)
#cloning
list_copy = li[:]
print("After cloning: ",list_copy)
```

```
Enter size of list 5
Enter element of list 3
Enter element of list 6
Enter element of list 1
Enter element of list 2
Enter element of list 3
Original list: [3, 6, 1, 2, 3]
After cloning: [3, 6, 1, 2, 3]
```

2. extend()

A list can be copied into a new list by using the in-built extend() function. This will add each element of the original list to the end of the new list. This method takes around 0.053 seconds to complete.

Algorithm

Follow the algorithm to understand the approach better.

Step 1- Take input of list from user

Step 2- Declare a new list which will be the copy

Step 3 -Copy the elements using extend() in the new list

Step 4- Print the new list which will be the copy of the original list

Example 2

Look at the program to understand the implementation of the above-mentioned approach. In this program, we have input elements of the list from the user and copied it into a new list using the extend() function.

```
li=[]
n=int(input("Enter size of list "))
for i inrange(0,n):
    e=int(input("Enter element of list "))
    li.append(e)
print("Original list: ",li)
#cloning
list_copy =[]
list_copy.extend(li)
print("After cloning: ",list_copy)
```

Output

```
Enter size of list 6
Enter element of list 3
Enter element of list 5
Enter element of list 12
Enter element of list 78
Enter element of list 16
Enter element of list 35
Original list: [3, 5, 12, 78, 16, 35]
After cloning: [3, 5, 12, 78, 16, 35]
```

3. list()

We will simply use the list() function to copy the existing list into another list in this approach. This method takes about 0.075 seconds to complete.

Algorithm

Follow the algorithm to understand the approach better.

Step 1- Take input of list from user

Step 2- Declare a new list which will be the copy of the original list

Step 3- Initialise the new list and pass the original list in the list() function

Step 4- Print the new list which will be the copy of the original list

Example 3

Look at the program to understand the implementation of the above-mentioned approach. In this program, we have to input the list from the user and copy it into a new list by using the list() function.

```
li=[]
n=int(input("Enter size of list "))
for i inrange(0,n):
    e=int(input("Enter element of list "))
    li.append(e)
print("Original list: ",li)
#cloning
list_copy =list(li)
print("After cloning: ",list_copy)
```

Output

```
Enter size of list 4
Enter element of list 2
Enter element of list 7
Enter element of list 9
Enter element of list 10
Original list: [2, 7, 9, 10]
After cloning: [2, 7, 9, 10]
```

4. list comprehension

In this approach, we will use the list comprehension method. List comprehension is a shorter syntax for creating a new list based on the values of an existing list. We can create a copy of the list using list comprehension. The method takes about 0.217 seconds to complete.

Algorithm

Follow the algorithm to understand the approach better.

Step 1- Take input of list from user

Step 2-Declare a new list which will be the copy of the original list

Step 3- Use list comprehension to copy the elements in the list

Step 4- All the values in the original list will be stored in the new list

Step 5- Print the new list which will be the copy of the original list

Ex: 4

Look at the program to understand the implementation of the above-mentioned approach. In this program, we have to take input the list from the user and copied it into a new list by using the list comprehension syntax.

```
li=[]
n=int(input("Enter size of list "))
for i inrange(0,n):
e=int(input("Enter element of list "))
li.append(e)
print("Original list: ",li)
#cloning
list_copy = [ num for num in li ]
print("After cloning: ",list_copy)
```

Output:

```
Enter size of list 7
Enter element of list 12
Enter element of list 13
Enter element of list 2
Enter element of list 24
Enter element of list 6
Enter element of list 35
Enter element of list 8
Original list: [12, 13, 2, 24, 6, 35, 8]
After cloning: [12, 13, 2, 24, 6, 35, 8]
```

5. append()

append() function simply adds the element to the last position of a list. This can be used for copying elements to a new list. This method takes around 0.325 seconds to complete and is the slowest method for cloning a list.

Algorithm

Follow the algorithm to understand the approach better.

Step 1- Take input of list from user

Step 2-Declare a new list which will be the copy of the original list

Step 3- Run a loop to access each element in the list

Step 4- Use append() to add each element in the new list one by one

Step 5- Print the new list which will be the copy of the original list

Ex: 5

Look at the program to understand the implementation of the above-mentioned approach. In this program, we have taken an input of elements of the list from the user and added each element of the list one by one to a new list using append().

```
li=[]
n=int(input("Enter size of list "))
for i inrange(0,n):
  e=int(input("Enter element of list "))
  li.append(e)
print("Original list: ",li)
#cloning
list_copy =[]
for num in li:
  list_copy.append(num)
print("After cloning: ",list_copy)
```

Output

```
Enter size of list 8
Enter element of list 2
Enter element of list 3
Enter element of list 4
Enter element of list 5
Enter element of list 6
Enter element of list 8
Enter element of list 9
Enter element of list 10
Original list: [2, 3, 4, 5, 6, 8, 9, 10]
After cloning: [2, 3, 4, 5, 6, 8, 9, 10]
```

6. copy()

copy() is an in-built method to copy all the elements from one list to another. This method takes around 1.488 seconds to complete and takes the most time out of all the approaches.

Algorithm

Follow the algorithm to understand the approach better.

Step 1- Take input of list from user

Step 2- Declare a new list

Step 3- copy the original list to the new list using the copy() function

Step 4- Print the new list

Ex: 6

Look at the program to understand the implementation of the above-mentioned approach. In this program, we have taken an input of elements of the list from the user and added each element of the list one by one to a new list using append().

```
li=[]
n=int(input("Enter size of list "))
for i inrange(0,n):
  e=int(input("Enter element of list "))
  li.append(e)
print("Original list: ",li)
#cloning
list_copy = li.copy()
print("After cloning: ",list_copy)
```

Output:

```
Enter size of list 5
Enter element of list 12
Enter element of list 13
Enter element of list 14
Enter element of list 25
Enter element of list 26
Original list: [12, 13, 14, 25, 26]
After cloning: [12, 13, 14, 25, 26]
```

3.2.8 List Parameters

Q18. Explain list parameters

Ans :

List () Parameters

The syntax followed to apply the function of the list () Python is as provided below.

list (iterable)

Here, iterable refers to the sequence of collection of data. It can also be an iterator object. Note that it is an optional parameter. Thus, if no parameter is passed in the list () function, then the compiler will not show an error in the code.

The object that is passed as the iterable can be in the form of a sequence such as string or tuple or a collection such as a set or a dictionary.

Return Value from the list ()

If there are no parameters in the list () Python function, then it will return an empty list. However, if any specific and acceptable parameter is passed, then the function will create a list that consists of the items present in the iterable.

Example 1: Create lists from string, tuple, and lists

The program given below will create an empty list, a list through the string, one that has a tuple in its parameter, and the last one that includes a list.

```
print(list())
even = "246810"
print(list(even))
even1 =("2", "4", "6", "8", "10")
```

print(list(even1))

even2 = [“2”, “4”, “6”, “8”, “10”]

print(list(even2))

The output for all the lists that have parameters will be the same. Thus, using the list () function, one can create a list using any type of sequence.

3.3 Tuple

3.3.1 Benefit Of Tuple

Q19. What is tuple in python? What are its advantages

Ans : **(Imp.)**

In Python programming, a tuple is similar to a list. The difference between the two is that we cannot change the elements of a tuple once it is assigned whereas in a list, elements can be changed.

Advantages of Tuple over List

Since, tuples are quite similar to lists, both of them are used in similar situations as well.

However, there are certain advantages of implementing a tuple over a list. Below listed are some of the main advantages:

- We generally use tuple for heterogeneous (different) datatypes and list for homogeneous (similar) datatypes.
- Since tuple are immutable, iterating through tuple is faster than with list. So there is a slight performance boost.
- Tuples that contain immutable elements can be used as key for a dictionary. With list, this is not possible.
- If you have data that doesn't change, implementing it as tuple will guarantee that it remains write-protected.

3.3.2 Operations On Tuple

Q20. Explain the number of operations that can be performed upon tuples

Ans : **(Imp.)**

1. Define Tuples in Two ways

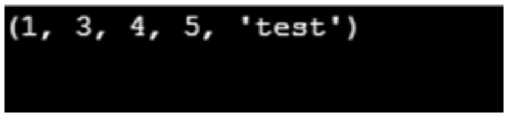
To create a tuple, assign a single variable with multiple values separated by commas in parentheses.

Code:

type1 = (1,3,4,5,'test')

print(type1)

Output:



```
(1, 3, 4, 5, 'test')
```

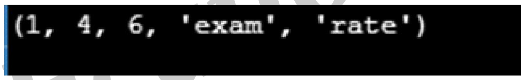
To create a tuple, assign a single variable with multiple values separated by commas without parentheses. Please refer introduction for minor difference.

Code:

type2 = 1,4,6,'exam','rate'

print(type2)

Output:



```
(1, 4, 6, 'exam', 'rate')
```

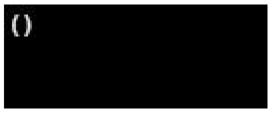
We can define an empty tuple:

Code:

a = ()

print(a)

Output:



```
()
```

2. Accessing Items in a Tuple

One can access the elements of a tuple in multiple ways, such as indexing, negative indexing, range, etc.

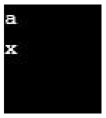
Code:

access_tuple = ('a','b',1,3,[5,'x','y','z'])

print(access_tuple[0])

print(access_tuple[4][1])

Output:



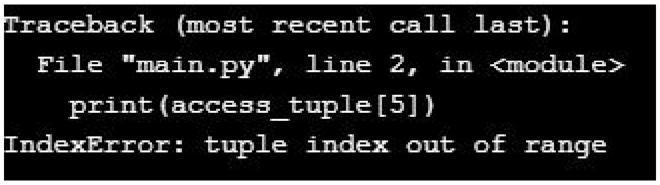
```
a
x
```

In case the index value is out of the scope of tuple, it will through the following error.

Code:

print(access_tuple[5])

Output:



```
Traceback (most recent call last):
  File "main.py", line 2, in <module>
    print(access_tuple[5])
IndexError: tuple index out of range
```

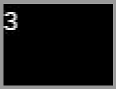
We can find the use of negative indexing on tuples.

Code:

access_tuple = ('a','b',1,3)

print(access_tuple[-1])

Output:



```
3
```

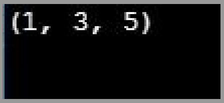
We can find a range of tuples.

Code:

access_tuple = ('a','b',1,3,5,'x','y', 'z')

print(access_tuple[2:5])

Output:



```
(1, 3, 5)
```

3. Concatenation Operation on Tuples

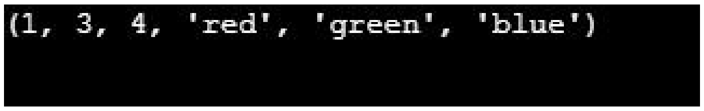
Concatenation simply means linking things together. We can concatenate tuples together. **Code:**

Tuple1 = (1,3,4)

Tuple2 = ('red','green','blue')

print(Tuple1 + Tuple2)

Output:



```
(1, 3, 4, 'red', 'green', 'blue')
```

4. Nesting Operation on Tuples

Nesting simply means the place or store one or more inside the other.

Code:

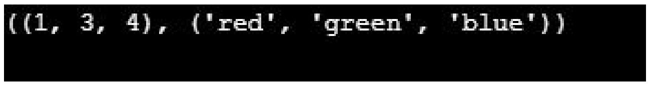
Tuple1 = (1,3,4)

Tuple2 = ('red','green','blue')

Tuple3 = (Tuple1, Tuple2)

print(Tuple3)

Output:



```
((1, 3, 4), ('red', 'green', 'blue'))
```

5. **Slicing Operation on Tuples**

As tuples are immutable, we can take slices of one tuple and place them in another tuple.

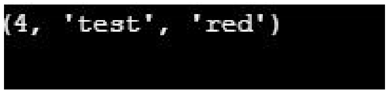
Code:

Tuple1 = (1,3,4,'test','red')

Sliced = (Tuple1[2:])

print(Sliced)

Output:



```
(4, 'test', 'red')
```

6. **Finding length of Tuples**

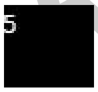
We can find the length of the tuple to see how many values are in there a tuple.

Code:

Tuple1 = (1,3,4,'test','red')

print(len(Tuple1))

Output:



```
5
```

7. **Changing a Tuple**

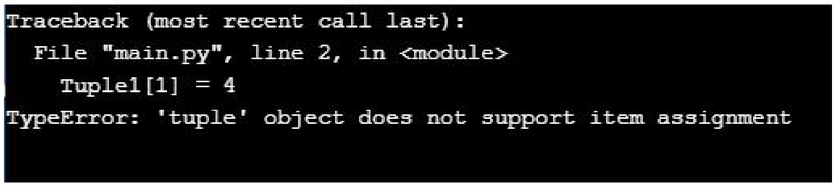
As we know that the tuples are immutable. This means that items defined in a tuple cannot be changed once the tuple has been created.

Code:

Tuple1 = (1,3,4,'test','red')

Tuple1[1] = 4

Output:



```
Traceback (most recent call last):
  File "main.py", line 2, in <module>
    Tuple1[1] = 4
TypeError: 'tuple' object does not support item assignment
```

Here we have one case, if the item in tuple itself is a mutable data type like a list, its nested items can be changed.

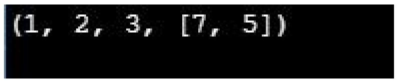
Code:

tuple1 =(1,2,3,[4,5])

tuple1[3][0]=7

print(tuple1)

Output:



```
(1, 2, 3, [7, 5])
```

8. Deleting a Tuple

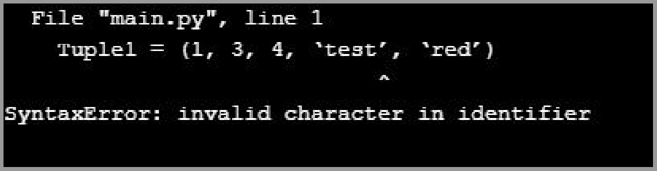
As we have discussed earlier, we cannot change the items in a tuple. which also suggests that we cannot remove items from the tuple.

Code:

Tuple1 =(1,3,4,'test','red')

del(Tuple1[1])

Output:



```
  File "main.py", line 1
    Tuple1 = (1, 3, 4, 'test', 'red')
                                ^
SyntaxError: invalid character in identifier
```

But one can delete a tuple by using the keyword del( ) with a tuple.

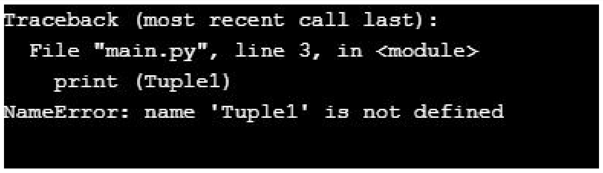
Code:

Tuple1 =(1,3,4,'test','red')

del(Tuple1)

print(Tuple1)

Output:



```
Traceback (most recent call last):
  File "main.py", line 3, in <module>
    print (Tuple1)
NameError: name 'Tuple1' is not defined
```

9. Membership Test on Tuples

This can be tested whether an item exists in a tuple or not; the keyword for this is in.

Code:

Tuple1 =(1,3,4,'test','red')

print(1in Tuple1)

print(5in Tuple1)

Output:

3.3.3 Tuple Methods

Q20. Explain about various Tuple methods.

Ans :

Python Tuples is an immutable collection of that are more like lists. Python Provides a couple of methods to work with tuples. In this article, we will discuss these two methods in detail with the help of some examples.

1. Count() Method

The count() method of Tuple returns the number of times the given element appears in the tuple.

Syntax:

tuple.count(element)

Where the element is the element that is to be counted.

Example 1: Using the Tuple count() method

```
# Creating tuples
Tuple1 = (0, 1, 2, 3, 2, 3, 1, 3, 2)
Tuple2 = ('python', 'geek', 'python','for', 'java', 'python')
# count the appearance of 3
res = Tuple1.count(3)
print('Count of 3 in Tuple1 is:', res)
# count the appearance of python
res = Tuple2.count('python')
print('Count of Python in Tuple2 is:', res)
```

Output:

```
Count of 3 in Tuple1 is: 3
Count of Python in Tuple2 is: 3
```

2. Index() Method

The Index() method returns the first occurrence of the given element from the tuple.

Syntax:

tuple.index(element, start, end)

Parameters:

- **element :** The element to be searched.
- **start (Optional) :** The starting index from where the searching is started
- **end (Optional):** The ending index till where the searching is done

Note: This method raises a ValueError if the element is not found in the tuple.

Example 1: Using Tuple Index() Method

```
# Creating tuples
Tuple = (0, 1, 2, 3, 2, 3, 1, 3, 2)
# getting the index of 3
res = Tuple.index(3)
print('First occurrence of 3 is', res)
 # getting the index of 3 after 4th
# index
res = Tuple.index(3, 4)
print('First occurrence of 3 after 4th index is:', res)
```

Output:

```
First occurrence of 3 is 3
```

3.3.4 Tuple Assignment

Q21. Explain about Tuple assignment.

Ans : **(Imp.)**

An assignment to all of the elements in a tuple using a single assignment statement.

- Python has a very powerful tuple assignment feature that allows a tuple of variables on the left of an assignment to be assigned values from a tuple on the right of the assignment.
- The left side is a tuple of variables; the right side is a tuple of values.
- Each value is assigned to its respective variable.
- All the expressions on the right side are evaluated before any of the assignments. This feature makes tuple assignment quite versatile.
- Naturally, the number of variables on the left and the number of values on the right have to be the same.

```
>>>    (a, b, c, d) = (1, 2, 3)
ValueError: need more than 3 values to unpack
```

Example:

- It is useful to swap the values of two variables. With conventional assignment statements, we have to use a temporary variable. For example, to swap a and b:

Swap two numbers

```
a=2;b=3
print(a,b)
temp = a
a = b
b = temp
print(a,b)
```

Output:

```
(2, 3)
(3, 2)
>>>
```

- Tuple assignment solves this problem neatly:

  (a, b) = (b, a)

- One way to think of tuple assignment is as tuple packing/unpacking.

  In tuple packing, the values on the left are 'packed' together in a tuple:

  ```
  >>> b = ("George", 25, "20000")
  # tuple packing
  ```

- In tuple unpacking, the values in a tuple on the right are 'unpacked' into the variables/names on the right:

  ```
  >>> b = ("George", 25, "20000")
  # tuple packing
  >>> (name, age, salary) = b
  # tuple unpacking
  >>> name
  'George'
  >>>     age
  25
  >>>     salary
  '20000'
  ```

- The right side can be any kind of sequence (string,list,tuple)

Example:

- To split an email address in to user name and a domain

  ```
  >>> mailid='god@abc.org'
  >>> name,domain=mailid.split('@')
  >>> print name
  god
  print (domain)
  abc.org
  ```

3.3.5 Tuple As Return Value

Q22. Write about Tuple as a return value for the function.

Ans : **(Imp.)**

Functions can return tuples as return values. This is very useful - we often want to know some batsman's highest and lowest score, or we want to find the mean and the standard deviation, or we want to know the year, the month, and the day, or if we're doing some ecological modeling we may want to know the number of rabbits and the number of wolves on an island at a given time. In each case, a function (which can only return a single value), can create a single tuple holding multiple elements.

For example, we could write a function that returns both the area and the circumference of a circle of radius r.

```
def circleInfo(r):
    """ Return (circumference, area) of a circle of radius r """
    c = 2 * 3.14159 * r
    a = 3.14159 * r * r
    return (c, a)
print(circleInfo(10))
```

Example:

The built-in function divmod takes two arguments and returns a tuple of two values, the quotient and remainder. You can store the result as a tuple:

```
>>> t = divmod(7, 3)
>>> t
(2, 1)
```

Or use tuple assignment to store the elements separately:

```
>>> quot, rem = divmod(7, 3)
>>> quot
2
>>> rem
1
```

Here is an example of a function that returns a tuple:

```
def min_max(t):
  return min(t), max(t)
```

max and min are built-in functions that find the largest and smallest elements of a sequence. min_max computes both and returns a tuple of two values.

3.3.6 Tuple As Argument

Q23. Explain about how tuples are used as function arguments.

Ans : **(Imp.)**

Tuples have many applications in all the domains of Python programming. They are immutable and hence are important containers to ensure read-only access, or keeping elements persistent for more time. Usually, they can be used to pass to functions and can have different kinds of behavior. Different cases can arise.

Case 1: fnc(a, b) – Sends a and b as separate elements to fnc.

Case 2: fnc((a, b)) – Sends (a, b), whole tuple as 1 single entity, one element.

Case 3: fnc(*(a, b)) – Sends both, a and b as in Case 1, as separate integers.

The code below demonstrates the working of all cases :

```
# Python3 code to demonstrate working of
# Tuple as function arguments
 # function with default arguments
def fnc(a=None, b=None):
    print("Value of a : " + str(a))
    print("Value of b : " + str(b))
# Driver code
if __name__ == "__main__" :
  # initializing a And b
  a = 4
  b = 7
  # Tuple as function arguments
  # Case 1 - passing as integers
  print("The result of Case 1 : ")
  fnc(a, b)
  # Tuple as function arguments
  # Case 2 - Passing as tuple
  print("The result of Case 2 : ")
  fnc((a, b))
  # Tuple as function arguments
  # Case 3 - passing as pack/unpack
  # operator, as integer
  print("The result of Case 3 : ")
  fnc(*(a, b))
```

Output :

```
The result of Case 1 :
Value of a : 4
Value of b : 7
The result of Case 2 :
Value of a : (4, 7)
Value of b : None
The result of Case 3 :
Value of a : 4
Value of b : 7
```

3.4 Dictionaries

3.4.1 Operations On Dictionary

Q24. What are dictionaries in python? Explain the operations that can be peformed on dictionaries.

Ans :

Dictionaries are know as hash tables in other programming languages, these provide us a mutable associative array type, through two elements, a key and a value, these are totally related. This type of structure is very efficient for data searches.

It is important to know, that Python dictionaries have a restriction where the keys must be an immutable data type, the goal is keeping the dictionary consistent. On the other hand, the values associated to the keys can be any Python data type, giving the possibility to change them after created or defined.

Dictionaries Examples:

x = {1: "one", 2: "two", 3: "three"}y = {'first': 1, ("Python", "Year"): ("Love", 1991)}

Dictionary operations

1. Definition operations

These operations allow us to define or create a dictionary.

i) { }

Creates an empty dictionary or a dictionary with some initial values.

y = {}x = {1: "one", 2: "two", 3: "three"}

2. Mutable operations

These operations allow us to work with dictionaries, but altering or modifying their previous definition.

i) [ ]

Adds a new pair of key and value to the dictionary, but in case that the key already exists in the dictionary, we can update the value.

y = {}y['one'] = 1y['two'] = 2print (y) Output: {'one': 1, 'two': 2}y['two'] = 'dos' print (y) Output:{'one': 1, 'two': 'dos'}

ii) del

Del statement can be used to remove an entry (key-value pair) from a dictionary.

y = {'one': 1, 'two': 2}print(y)Output:{'one': 1, 'two': 2}del y['two']print(y)Output:{'one': 1}

iii) update

This method updates a first dictionary with all the key-value pairs of a second dictionary. Keys that are common to both dictionaries, the values from the second dictionary override those of the first.

x = {'one': 0, 'two': 2}y = {'one': 1, 'three': 3}x.update(y)print(x)Output:{'one': 1, 'two': 2, 'three': 3}

3. Immutable operations

These operations allow us to work with dictionaries without altering or modifying their previous definition.

i) len

Returns the number of entries (key-value pairs) in a dictionary.

x = {'one': 0, 'two': 2}print(len(x))Output:2

ii) keys

This method allows you to get all the keys in the dictionary. It is often used in a "for loop" to iterate over the content of a dictionary.

x = {'one': 1, 'two': 2}print (x.keys()) Output: dict_keys (['one', 'two'])

iii) values

This method allows you to obtain all the values stored in a dictionary.

x = {'one': 1, 'two': 2}print (x.values()) Output: dict_values ([1, 2])

iv) items

Returns all the keys and their associated values as a sequence of tuples.

x = {'one': 1, 'two': 2}print (x.items()) Output: dict_items ([('one', 1), ('two', 2)])

v) in

Attempting to access a key that is not in a dictionary will raise an exception. To handle this exception, you can use the in method that test whether a key exists in a dictionary, returns True if a dictionary has a value stored under the given key and False otherwise.

y = {'one': 1, 'two': 2}del y['three'] Output: Key Error: 'three'y = {'one': 1, 'two': 2}if 'three' in y:

dely['three']

print(y)Output:{'one': 1, 'two': 2}

vi) get

Returns the value associated with a key if the dictionary contains that key, in case that the dictionary does not contain the key, you can specified a second optional argument to return a default value, if the argument is not included get method will return None.

y = {'one': 1, 'two': 2}print (y.get ('one')) print (y.get('three'))print(y.get('three', 'The key does not exist.'))Output:1NoneThe key does not exist.

vii) setdefault

This method is similar to get method, it returns the value associated with a key if the dictionary contains that key, but in case that the dictionary does not contain the key, this method will create a new element in the dictionary (key-value pair), where the first argument in this method is the key, and the second argument is the value. The second argument is optional, but if this is not included, the value will be None.

y = {'one': 1, 'two': 2}print(y.setdefault('three', '3'))print(y.setdefault('two', 'dos')) print(y) Output: 32 {'one': 1, 'two': 2, 'three': '3'}

3.4.2 Methods In Dictionary

Q25. Explain about the methods in dictionary.

Ans :

1. **Copying dictionary**

   You can also copy the entire dictionary to a new dictionary. For example, here we have copied our original dictionary to the new dictionary name "Boys" and "Girls".

   Python 2 Example

   Dict = {'Tim': 18,' Charlie':12,' Tiffany': 22,' Robert':25}

   Boys = {'Tim': 18,'Charlie':12,'Robert':25}

   Girls = {'Tiffany':22}

   student X=Boys.copy()

   student Y=Girls.copy()

   print student X

   print student Y

   Python 3 Example

   Dict = {'Tim': 18,'Charlie': 12,' Tiffany': 22,' Robert': 25}

   Boys = {'Tim': 18,'Charlie':12,'Robert':25}

   Girls = {'Tiffany':22}

   studentX=Boys.copy()

   studentY=Girls.copy()

   print(studentX)

   print(studentY)

➢ We have the original dictionary (Dict) with the name and age of the boys and girls together

➢ But we want boys list separate from girls list, so we defined the element of boys and girls in a separate dictionary name "Boys" and "Girls."

➢ Now again we have created new dictionary name "student X" and "student Y," where all the keys and values of boy dictionary are copied into student X, and the girls will be copied in studentY

➢ So now you don't have to look into the whole list in the main dictionary(Dict) to check who is a boy and who is girl, you just have to print student X if you want boys list and StudentY if you want girls list

➢ So, when you run the student X and studentY dictionary, it will give all the elements present in the dictionary of "boys" and "girls" separately

2. **Updating Dictionary**

   You can also update a dictionary by adding a new entry or a key-value pair to an existing entry or by deleting an existing entry. Here in the example, we will add another name, "Sarah" to our existing dictionary.

Python 2 Example

Dict = {'Tim': 18,' Charlie':12,' Tiffany': 22,' Robert':25}

Dict.update({"Sarah":9})

print Dict

Python 3 Example

Dict = {'Tim': 18,'Charlie':12,' Tiffany': 22,' Robert': 25}

Dict.update({"Sarah":9})

print(Dict)

➢ Our existing dictionary "Dict" does not have the name "Sarah."

➢ We use the method Dict.update to add Sarah to our existing dictionary

➢ Now run the code, it adds Sarah to our existing dictionary

Delete Keys from the dictionary

Python dictionary gives you the liberty to delete any element from the dictionary list. Suppose you don't want the name Charlie in the list, so you can remove the key element by the following code.

Python 2 Example

Dict = {'Tim': 18,'Charlie': 12,' Tiffany': 22,' Robert': 25}

del Dict ['Charlie']

print Dict

Python 3 Example

Dict = {'Tim': 18,'Charlie':12,' Tiffany':22,' Robert':25}

del Dict ['Charlie']

print(Dict)

When you run this code, it should print the dictionary list without Charlie.

- We used the code del Dict
- When code executed, it has deleted the Charlie from the main dictionary

3. Dictionary items() Method

The items() method returns a list of tuple pairs (Keys, Value) in the dictionary.

Python 2 Example

```
Dict = {'Tim': 18,'Charlie':12,' Tiffany':22,' Robert' :25}
print "Students Name: %s" % Dict.items()
```

Python 3 Example

```
Dict = {'Tim': 18,'Charlie':12,' Tiffany':22,' Robert':25}
print("Students Name: %s" % list(Dict.items()))
```

- We use the code items() method for our Dict.
- When code was executed, it returns a list of items (keys and values) from the dictionary

Check if a given key already exists in a dictionary

For a given list, you can also check whether our child dictionary exists in the main dictionary or not. Here we have two sub-dictionaries "Boys" and "Girls", now we want to check whether our dictionary Boys exist in our main "Dict" or not. For that, we use the for loop method with else if method.

Python 2 Example

```
Dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}
Boys = {'Tim': 18,'Charlie':12,'Robert':25}
Girls = {'Tiffany':22}
for key in Boys.keys():
  if key in Dict.keys():
     print True
  else:
     print False
```

3.4.3 Difference Between List Tuple And Dictionary

Q26. Differentiate between list, tuple and dictionary.

Ans : **(Imp.)**

Here are the differences between List, Tuple, and Dictionary in Python:

Parameters	List	Tuple	Dictionary
Basics	A list is basically like a dynamically sized array that gets declared in other languages (Array list in the case of Java, vector in the case of C++).	The tuples refer to the collections of various objects of Python separated by commas between them.	In Python, the dictionary refers to a collection (unordered) of various data types. We use these for storing data values such as maps, and unlike other data types capable of holding only one value in the form of an element, a dictionary can hold the key: value pair.

Homogeneity	A list refers to a data structure of non homogenous type that functions to store various elements in columns, multiple rows, and single rows.	A tuple also refers to a data structure of the non-homogenous type that functions to store various elements in columns, multiple rows, and single rows.	A dictionary also refers to a data structure of the non-homogenous type that functions to store key -value pairs.
Representation	We can represent a List by [ ]	We can represent a Tuple by ( )	We can represent a Dictionary by { }
Duplicate elements	It allows various duplicate elements.	It allows various duplicate elements.	The keys are not at all duplicated.
Nested Among All	It can be utilized in a List.	It can be utilized in a Tuple.	It can be utilized in a Dictionary.
Example	[6, 7, 8, 9, 10]	(6, 7, 8, 9, 10)	{6, 7, 8, 9, 10}
Function for Creation	We can create a list using the list() function.	We can create a tuple using the tuple() function.	We can create a dictionary using the dict() function.
Mutation	It is mutable. It means that a user can make any changes to a list.	It is immutable. It means that a user can't make any changes to a tuple.	It is mutable, but the keys are not at all duplicated.
Order	It is ordered in nature.	It is ordered in nature.	It is ordered in nature.
Empty Elements	If we want to create an empty list, we use:l=[ ]	If we want to create an empty tuple, we use:t=( )	If we want to create an empty dictionary, we use:d={ }

3.5 Advanced List Processing

3.5.1 List Comprehension

Q27. Explain about list comprehension technique.

Ans : **(Imp.)**

A Python list comprehension consists of brackets containing the expression, which is executed for each element along with the for loop to iterate over each element in the Python list.

Advantages of List Comprehension

- More time-efficient and space-efficient than loops.
- Require fewer lines of code.
- Transforms iterative statement into a formula.

Syntax of List Comprehension

```
new List = [ expression(element) for element in oldList if condition ]
```

Example 1: Iteration with List comprehension

```
# Using list comprehension to iterate through loop
List=[character forcharacter in[1, 2, 3]]
# Displaying list
print (List) Output :
[1, 2, 3]
```

Example 2: Even list using list comprehension

```
list=[i fori inrange(11) ifi %2= =0]
print(list)
```

Output :

```
[0, 2, 4, 6, 8, 10]
```

Example 3: Matrix using List comprehension

```
matrix = [[j forj inrange(3)] fori inrange(3)]
print(matrix)
```

Output :

```
[[0, 1, 2], [0, 1, 2], [0, 1, 2]]
```

List Comprehensions vs For Loop

There are various ways to iterate through a list. However, the most common approach is to use the for loop. Let us look at the below example:

```
# Empty list
List = []
 # Traditional approach of iterating
forcharacter in'Geeks 4 Geeks!':
    List.append(character)
 # Display list
print(List)
```

Output:

```
['G', 'e', 'e', 'k', 's', ' ', '4', ' ', 'G', 'e', 'e', 'k', 's', '!']
```

List Comprehensions translate the traditional iteration approach using for loop into a simple formula hence making them easy to use. Below is the approach to iterate through a list, string, tuple, etc. using list comprehension.

```
# Using list comprehension to iterate through loop
List = [character forcharacter in'Geeks 4 Geeks!']
# Displaying list
print(List)
```

Output:

```
['G', 'e', 'e', 'k', 's', ' ', '4', ' ', 'G', 'e', 'e', 'k', 's', '!']
```

Time Analysis in List Comprehensions and Loop

The list comprehensions are more efficient both computationally and in terms of coding space and time than a for a loop. Typically, they are written in a single line of code. The below program depicts the difference between for loops and list comprehension based on performance.

```
# Import required module
importtime
 # define function to implement for loop
deffor_loop(n):
    result = []
    fori inrange(n):
        result.append(i**2)
    returnresult
 # define function to implement list comprehension
deflist_comprehension(n):
    return[i**2fori inrange(n)]
 # Driver Code
 # Calculate time takens by for_loop()
begin = time.time()
for_loop(10**6)
end = time.time()
 # Display time taken by for_loop()
print('Time taken for_loop:', round(end-begin, 2))
 # Calculate time takens by list_comprehension()
begin = time.time()
list_comprehension(10**6)
end = time.time()
# Display time taken by for_loop()
print('Time taken for list_comprehension:', round(end-begin, 2))
```

Output:

```
Time taken for_loop: 0.56
Time taken for list_comprehension: 0.47
```

From the above program, we can see list comprehensions are quite faster than for loop.

Nested List Comprehensions

Nested List Comprehensions are nothing but a list comprehension within another list comprehension which is quite similar to nested for loops. Below is the program which implements nested loop:

```
matrix = []
 fori inrange(3):
    # Append an empty sublist inside the list
  matrix.append([])
   forj inrange(5):
       matrix[i].append(j)
print(matrix)
```

Output

```
[[0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4]]
```

Now by using nested list comprehensions same output can be generated in fewer lines of code.

```
 # Nested list comprehension
 matrix = [[j forj inrange(5)] fori inrange(3)]
print(matrix)
```

Output

```
[[0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4]]
```

3.5.2 Nested List

Q28. Explain about nested lists

Ans : **(Imp.)**

Meaning

A nested list is a *list of lists*, or any list that has another list as an element (a sublist). They can be helpful if you want to create a matrix or need to store a sublist along with other data types.

An example of a nested list

```
    # creating list
nestedList = [1, 2, ['a', 1], 3]
    # indexing list: the sublist has now been accessed
subList = nestedList[2]
    # access the first element inside the inner list:
element = nestedList[2][0]
print("List inside the nested list: ", subList)
print("First element of the sublist: ", element)
```

Creating a matrix

Creating a matrix is one of the most useful applications of nested lists. This is done by creating a nested list that only has other lists of equal length as elements.

- The *number of elements* in the nested lists is equal to the *number of rows* of the matrix.
- The *length of the lists* inside the nested list is equal to the *number of columns*.

Thus, each element of the nested list (matrix) will have two indices: the row and the column.

A matrix of size 3x3

```
# create matrix of size 4 x 3
```

```
matrix  =  [ [0,  1,  2],
              [3,  4,  5],
              [6,  7,  8],
              [9,  10,  11]]
             rows = len(matrix)      #  no  of  rows  is  no  of  sublists  i.e.  len  of  list
             cols = len(matrix[0])   #  no  of  cols  is  len  of  sublist
                                     #  printing  matrix
             print("matrix:")
             for  i  in  range(0,  rows):
             print(matrix[i])
                                     #  accessing  the  element  on  row  2  and  column  1  i.e.,  3
             print("element  on  row  2  and  column  1:",  matrix[1][0])
                                     #  accessing  the  element  on  row  3  and  column  2  i.e.,  7
             print("element  on  row  3  and  column  2:",  matrix[2][1])
```

Short Question and Answers

1. What is string

Ans :

In Python, string is a sequence of Unicode character. Unicode was introduced to include every character in all languages and bring uniformity in encoding.

2. Explain how to do Slicing in Python.

Ans :

You can take subset of string from original string by using [] operator also known as slicing operator.

Syntax: s [start:end]

this will return part of the string starting from index start to index end-1.

Let's take some examples.

```
> > > s = "Welcome"
> > > s[1:3]
El
```

Some more examples

```
> > > s = "Welcome"
> > > s[ : 6]
'Welcom'
> > > s[4 : ]
'ome'
> > > s[1 : -1]
'elcom'
```

Note: start index and end index are optional. If omitted then the default value of start index is 0 and that of end is the last index of the string.

4. Explain various String Manipulation Functions.

Ans :

1. capitalize()

This function returns the copy of the string passed changing the first character of the string to uppercase.

Syntax:

str.capitalize()

Example:

str = "lets test the function";

print "str.capitalize() : ", str.capitalize() // change the first character to uppercase

Output:

str.capitalize() : Lets test the function

2. is lower()

This method checks if the string is in lowercase and returns true if all the characters are in lowercase.

Syntax:

str.islower()

Example:

str = "lets test the function";

print str.islower();// Returns true since all characters are in lower case.

str = "lets Test the function";

print str.islower();

Out put:

True

False

3. isupper()

This method checks if all the characters in the string are in uppercase. If any character is in lower case, it would return false otherwise true.

Syntax:

str.isupper()

Example:

str = "LETS TEST THE FUNCTION";

print str.isupper();//returns true since all characters are capital

str = "LETS TEST THE FUNCTIOn";

print str.isupper(); // Returns false as 'n' is small.

Output

True

False

5. Define array.

Ans : **(Imp.)**

Array is an idea of storing multiple items of the same type together and it makes easier to calculate the position of each element by simply adding an offset to the base value. A combination of the arrays could save a lot of time by reducing the overall size of the code. It is used to store multiple values in single variable. If you have a list of items that are stored in their corresponding variables like this:

car1 = "Lamborghini"

car2 = "Bugatti"

car3 = "Koenigsegg"

If you want to loop through cars and find a specific one, you can use the array.

The array can be handled in Python by a module named array. It is useful when we have to manipulate only specific data values. Following are the terms to understand the concept of an array:

Element - Each item stored in an array is called an element.

Index - The location of an element in an array has a numerical index, which is used to identify the position of the element.

6. How to create a list ?

Ans :

In Python programming, a list is created by placing all the items (elements) inside a square bracket [ ], separated by commas.

It can have any number of items and they may be of different types (integer, float, string etc.).

```
# empty list
my_list = []
# list of integers
my_list = [1,2,3]
# list with mixed datatypes
my_list = [1,"Hello",3.4]
```

Also, a list can even have another list as an item. This is called nested list.

```
# nested list
my_list = ["mouse", [8, 4, 6], ['a']]
```

7. What is list slicing?

Ans :

List Slicing

Python has an amazing feature just for that called *slicing*. Slicing can not only be used for lists, tuples or arrays, but custom data structures as well, with the *slice* object, which will be used later on in this article.

We can access a range of items in a list by using the slicing operator (colon).

8. Aliasing in lists.

Ans :

In python programming, the second name given to a piece of data is known as an alias. Aliasing happens when the value of one variable is assigned to another variable because variables are just names that store references to actual value.

9. What is list Cloning

Ans :

Cloning or copying a list is simply creating another list that has the same elements as the original list. Elements in a list can be copied into another list by various methods, we will be discussing most of them in this tutorial. For example,

Input:	Original list- [3, 6, 12, 14, 78, 24, 56]
Output:	After cloning- [3, 6, 12, 14, 78, 24, 56]
Input:	Original list- [14, 7, 9, 13, 46, 12]
Output:	After cloning- [14, 7, 9, 13, 46, 12]

10. Advantages of Tuple over List

Ans :

Since, tuples are quite similar to lists, both of them are used in similar situations as well.

However, there are certain advantages of implementing a tuple over a list. Below listed are some of the main advantages:

➢ We generally use tuple for heterogeneous (different) datatypes and list for homogeneous (similar) datatypes.

➢ Since tuple are immutable, iterating through tuple is faster than with list. So there is a slight performance boost.

➢ Tuples that contain immutable elements can be used as key for a dictionary. With list, this is not possible.

➢ If you have data that doesn't change, implementing it as tuple will guarantee that it remains write-protected.

11. What are dictionaries in python? Explain the operations that can be peformed on dictionaries.

Ans :

Dictionaries are know as hash tables in other programming languages, these provide us a mutable associative array type, through two elements, a key and a value, these are totally related. This type of structure is very efficient for data searches.

It is important to know, that Python dictionaries have a restriction where the keys must be an immutable data type, the goal is keeping the dictionary consistent. On the other hand, the values associated to the keys can be any Python data type, giving the possibility to change them after created or defined.

Choose the Correct Answers

1. What will be the output of below Python code? [d]

   str1=”poWer”

   str1.upper()

   print(str1)

   (a) POWER (b) Power

   (c) power (d) poWer

2. Which of the following will give “Simon” as output? [c]

   If str1=”John,Simon,Aryan”

   (a) print(str1[-7:-12]) (b) print(str1[-11:-7])

   (c) print(str1[-11:-6]) (d) print(str1[-7:-11])

3. What will be the output of above Python code? [d]

   str1=”6/4"

   print(“str1”)

   (a) 1 (b) 6/4

   (c) 1.5 (d) str1

4. What will be the output of below Python code? [a]

   list1=[8,0,9,5]

   print(list1[::-1])

   (a) [5,9,0,8] (b) [8,0,9]

   (c) [8,0,9,5] (d) [0,9,5]

5. Which of the following commands will create a list? [d]

   (a) list1 = list() (b) list1 = []

   (c) list1 = list([1, 2, 3]) (d) all of the mentioned

6. To add a new element to a list we use which command? [b]

   (a) list1.add(5) (b) list1.append(5)

   (c) list1.addLast(5) (d) list1.add End(5)

8. del statement can delete the following from the List? [d]

   (a) Single Element (b) Multiple Elements

   (c) All elements along with List object (d) All of the above

9. Which of the following is a Python tuple? [d]

   (a) [1, 2, 3] (b) (1, 2, 3)

   (c) {1, 2, 3} (d) { }

10. Which of the following is not a function of the tuple? [d]

   (a) max( ) (b) min()

   (c) count( ) (d) update( )

Fill in the Blanks

1. ______ function in string gives the output by converting only the first character of the string into uppercase and rest characters into lowercase.
2. ______ function is used to return the whole string into uppercase.
3. Elements of lists are stored in ______ memory location is True regarding lists in Python.
4. To shuffle the list(say list1) ______ function do we use?
5. Suppose list1 is [3, 5, 25, 1, 3], what is min(list1) ?
6. Index value in list and string start from ______
7. a tuple can contain both ______ and ______ as its elements
8. Tuples are ______
9. In tuples values are enclosed in ______
10. ______ mathematical operator is used to replicate a tuple?

ANSWERS

1. Capitalize( )
2. upper( )
3. contagious
4. random.shuffle(list1)
5. 1
6. 0
7. integers and strings
8. Immutable
9. Paranthesis
10. Multiplication

UNIT IV

Introduction to Numpy: The basics of numpy array, computation on numpy arrays, aggregations, computations on arrays, comparisons, masks and Boolean logic, fancy indexing, sorting arrays, structured data.

4.1 Introduction to Numpy

4.1.1 The Basics of Numpy Array

Q1. What is NumPy? Explain how to create arrays in python using Numpy.

Ans : **(Imp.)**

NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays. It is the fundamental package for scientific computing with Python. It is open-source software. It contains various features including these important ones:

- A powerful N-dimensional array object
- Sophisticated (broadcasting) functions
- Tools for integrating C/C++ and Fortran code
- Useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined using Numpy which allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

Installation

- Mac and Linux users can install NumPy via pip command:

  pip install numpy

- Windows does not have any package manager analogous to that in linux or mac. Please download the pre-built windows installer for NumPy from here (according to your system configuration and Python version). And then install the packages manually.

Note: All the examples discussed below will not run on an online IDE. 1. Arrays in

NumPy: NumPy's main object is the homogeneous multidimensional array.

- It is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers.
- In NumPy dimensions are called axes. The number of axes is rank.
- NumPy's array class is called ndarray. It is also known by the alias array.

Example :

[[1, 2, 3],

[4, 2, 5]]

Here,

rank = 2 (as it is 2-dimensional or it has 2 axes)

first dimension(axis) length = 2, second dimension has length = 3

overall shape can be expressed as: (2, 3)

```
# Python program to demonstratebasic array characteristics
importnumpy as np
# Creating array object
arr =np.array( [[ 1, 2, 3],
                [ 4, 2, 5]] )
# Printing type of arr object
print("Array is of type: ", type(arr))
# Printing array dimensions (axes)
print("No. of dimensions: ", arr.ndim)
# Printing shape of array
```

```
print("Shape of array: ", arr.shape)
# Printing size (total number of elements) of array
print("Size of array: ", arr.size)
# Printing type of elements in array
print("Array stores elements of type: ", arr.dtype)
```

Output :

```
Array is of type:
No. of dimensions:  2
Shape of array:  (2, 3)
Size of array:  6
Array stores elements of type:  int64
```

Q2. Explain array creation techniques in Numpy with an example program.

Ans : **(Imp.)**

Array creation: There are various ways to create arrays in NumPy

- For example, you can create an array from a regular Python list or tuple using the array function. The type of the resulting array is deduced from the type of the elements in the sequences.
- Often, the elements of an array are originally unknown, but its size is known. Hence, NumPy offers several functions to create arrays with initial placeholder content. These minimize the necessity of growing arrays, an expensive operation. For example: np.zeros, np.ones, np.full, np.empty, etc.
- To create sequences of numbers, NumPy provides a function analogous to range that returns arrays instead of lists.
- **arange:** returns evenly spaced values within a given interval. step size is specified.
- **linspace:** returns evenly spaced values within a given interval. num no. of elements are returned.
- **Reshaping array:** We can use reshape method to reshape an array. Consider an array with shape (a1, a2, a3, …, aN). We can reshape and convert it into another array with shape (b1, b2, b3, …, bM). The only required condition is: a1 x a2 x a3 … x aN = b1 x b2 x b3 … x bM . (i.e original size of array remains unchanged.)
- **Flatten array:** We can use flatten method to get a copy of array collapsed into one dimension. It accepts order argument. Default value is 'C' (for row-major order). Use 'F' for column major order.

Note: Type of array can be explicitly defined while creating array.

```
# Python program to demonstratearray creation techniques
importnumpy as np
# Creating array from list with type float
a =np.array([[1, 2, 4], [5, 8, 7]], dtype ='float')
print("Array created using passed list:\n", a)
# Creating array from tuple
b =np.array((1, 3, 2))
print("\nArray created using passed tuple:\n", b)
```

```
# Creating a 3X4 array with all zeros
c =np.zeros((3, 4))
print("\nAn array initialized with all zeros:\n", c)
# Create a constant value array of complex type
d =np.full((3, 3), 6, dtype ='complex')
print("\nAn array initialized with all 6s."
        "Array type is complex:\n", d)
# Create an array with random values
e =np.random.random((2, 2))
print("\nA random array:\n", e)
# Create a sequence of integers
# from 0 to 30 with steps of 5
f =np.arange(0, 30, 5)
print("\nA sequential array with steps of 5:\n", f)
# Create a sequence of 10 values in range 0 to 5
g =np.linspace(0, 5, 10)
print("\nA sequential array with 10 values between"
                        "0 and 5:\n", g)
# Reshaping 3X4 array to 2X2X3 array
arr =np.array([[1, 2, 3, 4],
            [5, 2, 4, 2],
            [1, 2, 0, 1]])
newarr =arr.reshape(2, 2, 3)
print("\nOriginal array:\n", arr)
print("Reshaped array:\n", newarr)
# Flatten array
arr =np.array([[1, 2, 3], [4, 5, 6]])
flarr =arr.flatten()
print("\nOriginal array:\n", arr)
print("Fattened array:\n", flarr)
```

Output :

```
Array created using passed list:
[[ 1.  2.  4.]
[ 5.  8.  7.]]
Array created using passed tuple:
[1 3 2]
An array initialized with all zeros:
[[ 0.  0.  0.  0.]
[ 0.  0.  0.  0.]
[ 0.  0.  0.  0.]]
An array initialized with all 6s. Array type is complex:
[[ 6.+0.j  6.+0.j  6.+0.j]
[ 6.+0.j  6.+0.j  6.+0.j]
[ 6.+0.j  6.+0.j  6.+0.j]]
A random array:
[[ 0.46829566  0.67079389]
[ 0.09079849  0.95410464]]
A sequential array with steps of 5:
[ 0  5 10 15 20 25]
A sequential array with 10 values between 0 and 5:
[ 0.         0.55555556  1.11111111  1.66666667  2.22222222  2.77777778
3.33333333  3.88888889  4.44444444  5.        ]
Original array:
[[1 2 3 4]
[5 2 4 2]
[1 2 0 1]]
Reshaped array:
[[[1 2 3]
[4 5 2]]
[[4 2 1]
[2 0 1]]]
Original array:
[[1 2 3]
[4 5 6]]
Fattened array:
[1 2 3 4 5 6]
```

Q3. Expalin Array Indexing with an example program.

Ans :

Array Indexing: Knowing the basics of array indexing is important for analysing and manipulating the array object. NumPy offers many ways to do array indexing.

- **Slicing:** Just like lists in python, NumPy arrays can be sliced. As arrays can be multidimensional, you need to specify a slice for each dimension of the array.
- **Integer array indexing:** In this method, lists are passed for indexing for each dimension. One to one mapping of corresponding elements is done to construct a new arbitrary array.
- **Boolean array indexing:** This method is used when we want to pick elements from array which satisfy some condition.

```
# Python program to demonstrate
# indexing in numpy
importnumpy as np
# An exemplar array
arr =np.array([[-1, 2, 0, 4],
               [4, -0.5, 6, 0],
               [2.6, 0, 7, 8],
               [3, -7, 4, 2.0]])

# Slicing array
temp =arr[:2, ::2]
print("Array with first 2 rows and alternate"
                    "columns(0 and 2):\n", temp)
# Integer array indexing example
temp =arr[[0, 1, 2, 3], [3, 2, 1, 0]]
print("\nElements at indices (0, 3), (1, 2), (2, 1),"
                      "(3, 0):\n", temp)
# boolean array indexing example
cond =arr > 0# cond is a boolean array
temp =arr[cond]
print("\nElements greater than 0:\n", temp)
```

Output :

```
Array with first 2 rows and alternatecolumns(0 and 2):
[[-1.  0.]
[ 4.  6.]]
Elements at indices (0, 3), (1, 2), (2, 1),(3, 0):
[ 4.  6.  0.  3.]
Elements greater than 0:
[ 2.   4.   4.   6.   2.6  7.   8.   3.   4.   2. ]
```

4.1.2 Computation on Numpy Arrays

Q4. Explain various operations that can be performed on Numpy Arrays.

Ans : **(Imp.)**

Numpy array is a powerful N-dimensional array object which is in the form of rows and columns. We can initialize NumPy arrays from nested Python lists and access it elements. A Numpy array on a structural level is made up of a combination of:

- The Data pointer indicates the memory address of the first byte in the array.
- The Data type or dtype pointer describes the kind of elements that are contained within the array.
- The shape indicates the shape of the array.
- The strides are the number of bytes that should be skipped in memory to go to the next element.

1. Arithmetic Operations

```
# Python code to perform arithmeticoperations on NumPy array
importnumpy as np
# Initializing the array
arr1 =np.arange(4, dtype =np.float_).reshape(2, 2)
print('First array:')
print(arr1)
print('\nSecond array:')
arr2 =np.array([12, 12])
print(arr2)
print('\nAdding the two arrays:')
print(np.add(arr1, arr2))
print('\nSubtracting the two arrays:')
print(np.subtract(arr1, arr2))
print('\nMultiplying the two arrays:')
print(np.multiply(arr1, arr2))
print('\nDividing the two arrays:')
print(np.divide(arr1, arr2))
```

Output:

```
First array:
[[ 0.  1.]
[ 2.  3.]]
Second array:
[12 12]
Adding the two arrays:
[[ 12.  13.]
[ 14.  15.]]
Subtracting the two arrays:
[[-12. -11.]
[-10.  -9.]]
Multiplying the two arrays:
[[  0.  12.]
[ 24.  36.]]
Dividing the two arrays:
[[ 0.          0.08333333]
[ 0.16666667  0.25      ]]
```

2. numpy.reciprocal() This function returns the reciprocal of argument, element-wise. For elements with absolute values larger than 1, the result is always 0 and for integer 0, overflow warning is issued.

Example:

```
# Python code to perform reciprocal operation on NumPy array
importnumpy as np
arr =np.array([25, 1.33, 1, 1, 100])
print('Our array is:')
print(arr)
print('\nAfter applying reciprocal function:')
print(np.reciprocal(arr))
arr2 =np.array([25], dtype =int)
print('\nThe second array is:')
print(arr2)
print('\nAfter applying reciprocal function:')
print(np.reciprocal(arr2))
```

Output

```
Our array is:
[  25.      1.33    1.      1.    100.  ]
After applying reciprocal function:
[ 0.04       0.7518797  1.          1.          0.01      ]
The second array is:
[25]
After applying reciprocal function:
[0]
```

3. numpy.power() This function treats elements in the first input array as the base and returns it raised to the power of the corresponding element in the second input array.

```
# Python code to perform power operation# on NumPy array
importnumpy as np
arr =np.array([5, 10, 15])
print('First array is:')
print(arr)
print('\nApplying power function:')
print(np.power(arr, 2))
print('\nSecond array is:')
arr1 =np.array([1, 2, 3])
print(arr1)
print('\nApplying power function again:')
print(np.power(arr, arr1))
```

Output:

```
First array is:
[ 5 10 15]
Applying power function:
[ 25 100 225]
Second array is:
[1 2 3]
Applying power function again:
[   5  100 3375]
```

4. **numpy.mod()** This function returns the remainder of division of the corresponding elements in the input array. The function numpy.remainder() also produces the same result.

```
# Python code to perform mod function# on NumPy array
importnumpy as np
arr =np.array([5, 15, 20])
arr1 =np.array([2, 5, 9])
print('First array:')
print(arr)
print('\nSecond array:')
print(arr1)
print('\nApplying mod() function:')
print(np.mod(arr, arr1))
print('\nApplying remainder() function:')
print(np.remainder(arr, arr1))
```

Output:

```
First array:
[ 5 15 20]
Second array:
[2 5 9]
Applying mod() function:
[1 0 2]
Applying remainder() function:
[1 0 2]
```

4.2 Aggregations in Numpy

Q5. Explain the concept of aggregations in Numpy.

Ans : **(Imp.)**

Aggregation is a concept in which an object of one class can own or access another independent object of another class.

- It represents Has-A's relationship.
- It is a unidirectional association i.e. a one-way relationship. For example, a department can have students but vice versa is not possible and thus unidirectional in nature.
- In Aggregation, both the entries can survive individually which means ending one entity will not affect the other entity.

```
# Code to demonstrate Aggregation
# Salary class with the public method
# annual_salary()
classSalary:
    def__init__(self, pay, bonus):
        self.pay = pay
    self.bonus = bonus
    defannual_salary(self):
    return(self.pay*12) + self.bonus
# EmployeeOne class with public method
# total_sal()
classEmployeeOne:
    # Here the salary parameter reflects
    # upon the object of Salary class we
    # will pass as parameter later
    def_init_(self, name, age, sal):
    self.name = name
    self.age = age
        # initializing the sal parameter
    self.agg_salary = sal     # Aggregation
    deftotal_sal(self):
    returnself.agg_salary.annual_salary()
# Here we are creating an object
# of the Salary class
# in which we are passing the
# required parameters
```

```
salary =Salary(10000, 1500)
# Now we are passing the same
# salary object we created
# earlier as a parameter to
# EmployeeOne class
emp =EmployeeOne('Geek', 25, salary)
print(emp.total_sal())
```

Output:

```
121500
```

From the above code, we will get the same output as we got before using the Composition concept. But the difference is that here we are not creating an object of the Salary class inside the EmployeeOne class, rather than that we are creating an object of the Salary class outside and passing it as a parameter of EmployeeOne class which yields the same result.

4.3 Computations on Arrays

Q6. Expalin various arithmetic operation that can be performed on Numpy.

Ans :

The following are the arithmetic opertion tht canbe p

1. Addition

The add() function sums the content of two arrays, and return the results in a new array.

Example

Add the values in arr1 to the values in arr2:

```
import numpy as np
arr1 = np.array([10, 11, 12, 13, 14, 15])
arr2 = np.array([20, 21, 22, 23, 24, 25])
newarr = np.add(arr1, arr2)
print(newarr)
```

The example above will return [30 32 34 36 38 40] which is the sums of 10+20, 11+21, 12+22 etc.

2. Subtraction

The subtract() function subtracts the values from one array with the values from another array, and return the results in a new array.

Example

Subtract the values in arr2 from the values in arr1:

```
import numpy as np
arr1 = np.array([10, 20, 30, 40, 50, 60])
arr2 = np.array([20, 21, 22, 23, 24, 25])
newarr = np.subtract(arr1, arr2)
print(newarr)
```

The example above will return [-10 -1 8 17 26 35] which is the result of 10-20, 20-21, 30-22 etc.

3. Multiplication

The multiply() function multiplies the values from one array with the values from another array, and return the results in a new array.

Example

Multiply the values in arr1 with the values in arr2:

```
import numpy as np
              arr1 = np.array([10, 20, 30, 40, 50, 60])
              arr2 = np.array([20, 21, 22, 23, 24, 25])
           newarr = np.multiply(arr1, arr2)
print(newarr)
```

4. Division

The divide() function divides the values from one array with the values from another array, and return the results in a new array.

Example

Divide the values in arr1 with the values in arr2:

```
import numpy as np
              arr1 = np.array([10, 20, 30, 40, 50, 60])
              arr2 = np.array([3, 5, 10, 8, 2, 33])
           newarr = np.divide(arr1, arr2)
print (newarr)
```

The example above will return [3.33333333 4. 3. 5. 25. 1.81818182] which is the result of 10/3, 20/5, 30/10 etc.

4.4 Comparisons

Q7. Explain the comparisons operations in Numy with examples.

Ans : **(Imp.)**

The Python numpy comparison operators and functions used to compare the array items and returns Boolean True or false. The Python Numpy comparison functions are greater, greater_equal, less, less_equal, equal, and not_equal. Like any other, Python Numpy comparison operators are <, < =, >, > =, = = and ! =

To demonstrate these Python numpy comparison operators and functions, we used the numpy random randint function to generate random two dimensional and three-dimensional integer arrays.

The first array generates a two-dimensional array of size 5 rows and 8 columns, and the values are between 10 and 50.

```
arr1 = np.random.randint(10, 50, size = (5, 8))
```

This second array generates a random three-dimensional array of size 2 * 3 * 6. The generated random values are between 1 and 20.

```
arr2 = np.random.randint(1, 20, size = (2, 3, 6))
```

1. greater function

It is a simple Python Numpy Comparison Operators example to demonstrate the Python Numpy greater function. First, we declared an array of random elements. Next, we are checking whether the elements in an array are greater than 0, greater than 1 and 2. If True, True returned otherwise, False returned.

```
# Python array greater
import numpy as np
x = np.array([0, 2, 3, 0, 1, 6, 5, 2])
print('Original Array = ', x)
print('\nGreater Than 0 = ', np.greater(x, 0))
print()
print('Greater Than 1 = ', np.greater(x, 1))
print()
print('Greater Than 2 = ', np.greater(x, 2))
```

2. greater_equalfunction

The Python Numpy greater_equal function checks whether the elements in a given array (first argument) is greater than or equal to a specified number(second argument). If True, True returned otherwise, False.

The first Numpy statement checks whether items in the area is greater than or equal to 2. The second statement checks the items in a random 2D array is greater than or equal to 25. The third statement checks randomly generated three-dimensional array items that are greater than or equal to 7.

```
import numpy as np
arr = np.array([0, 2, 3, 0, 1, 6, 5, 2])
print('Original Array = ', arr)
print('Greater Than or Equal to 2 = ', np.greater_equal(arr, 2))
arr1 = np.random.randint(10, 50, size = (5, 8))
print('\n–Two Dimensional Random Array——')
print(arr1)
print()
print(np.greater_equal(arr1, 25))
arr2 = np.random.randint(1, 15, size = (2, 3, 6))
print('\n——Three Dimensional Random Array——')
print(arr2)
print()
print(np.greater_equal(arr2, 7))
```

3. less function

The Python Numpy less function checks whether the elements in a given array is less than a specified number or not. If True, boolean True returned otherwise, False. The syntax of this Python Numpy less function is

numpy.less(array_name, integer_value).

Within this example,

- np.less(arr, 4) – check whether items in arr array is less than 4.
- np.less(arr1, 32) – check the items in 2D array arr1 is less than 32.
- np.less(arr2, 15) – check items in randomly generated 3D array are less than 15.

```
import numpy as np
arr = np.array([0, 2, 3, 0, 1, 6, 5, 2])
print('Original Array = ', arr)
print('Less Than 4 = ', np.less(arr, 4))
arr1 = np.random.randint(10, 50, size = (5, 8))
print('\n——Two Dimensional Random Array——')
print(arr1)
print()
print(np.less(arr1, 32))
arr2 = np.random.randint(1, 25, size = (2, 3, 6))
print('\n——Three Dimensional Random Array——')
print(arr2)
print()
print(np.less(arr2, 15))
```

4. less_equalfunction

The Python Numpy less_equal function checks whether each element in a given array is less than or equal to a specified number or not. If True, boolean True returned otherwise, False. The syntax of this Python Numpy less_equal function is.

numpy.less_equal(array_name, integer_value).

Within this example,

- np.less_equal(arr, 3) – check whether items in arr array is less than or equal to 3.
- np.less_equal(arr1, 30) – check the items in 2D array arr1 is less than or equal to 30.
- np.less_equal(arr2, 9) – check whether items in the randomly generated three-dimensional array are less than or equal to 9.

```
import numpy as np
arr = np.array([0, 2, 3, 0, 1, 6, 5, 2])
print('Original Array = ', arr)
print('Less Than or Equal to 3 = ', np.less_equal(arr, 3))
arr1 = np.random.randint(10, 50, size = (5, 8))
print('\n——Two Dimensional Random Array——')
print(arr1)
print()
print(np.less_equal(arr1, 30))
arr2 = np.random.randint(1, 15, size = (2, 3, 6))
print('\n——Three Dimensional Random Array——')
print(arr2)
print()
print(np.less_equal(arr2, 9))
```

5. equalfunction

The Python Numpy equal function checks whether each item in an array is equal to a given number or not. If True, boolean True returned otherwise, False. The syntax of this Python Numpy equal function is

numpy.equal(array_name, integer_value).

Within this example,

- np.equal(arr, 0) – check whether items in arr array is equal to 0.
- np.equal(arr1, 28) – check items in two dimensional array arr1 is equal to 28.
- np.equal(arr2, 8) – check 3D array items are equal to 8.

```
import numpy as np
arr = np.array([0, 2, 3, 0, 1, 6, 0, 2])
print('Original Array = ', arr)
print('Equal to 0 = ', np.equal(arr, 0))
arr1 = np.random.randint(20, 30, size = (5, 8))
print('\n——Two Dimensional Random Array——')
print(arr1)
print()
print(np.equal(arr1, 28))
arr2 = np.random.randint(1, 10, size = (2, 3, 6))
print('\n——Three Dimensional Random Array——')
print(arr2)
print()
print(np.equal(arr2, 8))
```

6. not_equalfunction

The Python Numpy not_equal function checks whether each item in an array is not equals to a given number or not. If True, boolean True returned otherwise, False. The syntax of this Python Numpy not_equal function is

numpy.equal(array_name, integer_value).

Within this example,

- np.not_equal(arr, 0) – check whether items in arr array is not equal to 0.
- np.not_equal(arr1, 25) – check items in two dimensional array arr1 is not equal to 25.
- np.not_equal(arr2, 6) – check 3D array items are not equal to 6.

```
import numpy as np
arr = np.array([0, 2, 3, 0, 1, 6, 0, 2])
print('Original Array = ', arr)
print('Equal to 0 = ', np.not_equal(arr, 0))
arr1 = np.random.randint(20, 30, size = (5, 8))
```

```
print('\n——Two Dimensional Random Array——')
print(arr1)
print()
print(np.not_equal(arr1, 25))
arr2 = np.random.randint(1, 10, size = (2, 3, 6))
print('\n——Three Dimensional Random Array——')
print(arr2)
print()
print(np.not_equal(arr2, 6))
```

7. > Operator

The Python Numpy > Operator is the same as a greater function. Either you can use this to compare each element in an array with a static value, or use this to compare two arrays or matrixes.

```
import numpy as np
x = np.array([0, 2, 3, 0, 1, 6, 5, 2])
print('Original Array = ', x)
print('x Greater Than 2 = ', x > 2)
```

This time, we are using both the > operator and greater function to compare two one dimensional arrays and check whether items in one array is greater than the other.

```
import numpy as np
a = np.array([2, 4])
b = np.array([2, 3])
print('a Greater Than b = ', a > b)
print('a Greater Than b = ', np.greater(a, b))
arr1 = np.random.randint(1, 8, size = (7))
arr2 = np.random.randint(1, 9, size = (7))
print('\n——Random Array——')
print('Values in arr1 = ', arr1)
print('Values in arr2 = ', arr2)
print()
print('Result of arr1 > arr2 = ', arr1 > arr2)
print('greater(arr1, arr2)    = ', np.greater(arr1, arr2))
```

8. >= Operator

The Python Numpy >= Operator is the same as the greater_equal function. You can use >= operator to compare array elements with a static value or find greater than equal values in two arrays or matrixes.

```
import numpy as np
x = np.array([0, 2, 3, 0, 1, 6, 5, 2])
print('Original Array = ', x)
print('x Greater Than or Equal to 3 = \n', x >= 3)
```

9. < Operator

The Python Numpy < Operator is the same as less function. Either you can use this to check whether each element in an array is less than a static value or another array or matrix.

```
import numpy as np
x = np.array([0, 2, 3, 0, 1, 6, 5, 2])
print('Original Array = ', x)
print('x Less Than 2 = ', x < 2)
```

10. < = Operator

The Python Numpy < = Operator is the same as the less_equal function. Use Numpy < = operator to check array items are less than or equal to a number or another array.

```
import numpy as np
x = np.array([0, 2, 3, 0, 1, 6, 5, 2])
print('Original Array = ', x)
print('x Less Than or Equal to 3 = \n', x < = 3)
```

11. = = Operator

The Python Numpy = = Operator is the same as equal function. Use = = operator to check whether array items are equal to a number or another array.

```
import numpy as np
x = np.array([0, 2, 3, 0, 1, 6, 0, 2])
print('Original Array = ', x)
print('x Equal to 0 = ', x = = 0)
```

Python Numpy ! = Operator

The Python Numpy ! = Operator is the same as the not_equal function. Use ! = operator to check whether items in one array are not equal to a number or another array.

```
import numpy as np
x = np.array([0, 2, 3, 0, 1, 6, 0, 2])
print('Original Array = ', x)
print('x Not Equal to 0 = \n', x ! = 0)
```

The Python Numpy ! = Operator is the same as the not_equal function. Use ! = operator to check whether items in one array are not equal to a number or

4.5 Masks and Boolean Logic

Q8. Explain briefly about masks array module in Numpy.

Ans : **(Imp.)**

Numpy's

Using Masking of arrays we can easily handle the missing, invalid, or unwanted entries in our array or dataset/dataframe. Masking is essential works with the list of Boolean values i.e, True or False which when applied to an original array to return the element of interest, here True refers to the value that satisfies the given condition whereas False refers to values that fail to satisfy the condition.

We can mask the array using another by using the following functions :

numpy.ma.masked_where(condition, arr)

numpy.ma.getmask(arr)

numpy.ma.masked_array(arr, mask=)

where,

condition: condition for masking

arr: arr to be masked

mask: result of masked array

Steps

- Import the library.
- Create a function for masking.
- Masking can be done by following two approaches:-
  - **Using masked_where() function:** Pass the two array in the function as a parameter then use numpy.ma.masked_ where() function in which pass the condition for masking and array to be masked. In this we are giving the condition for masking by using one array and masking the another array for that condition.
  - **Using masked_where(), getmask() and masked_array() function:** Pass the two array in the function as a parameter then use numpy.ma.masked _where() function in which pass the condition for masking and array to be masked in this we are using the same array for which we are giving condition for making and the array to be masked and store the result in the variable, then use numpy.ma. getmask() function in which pass the result of marked_where function and store it in the variable named as 'res_mask'. Now mask another array using the created mask, for this, we are using numpy.ma. masked_array() function in which pass the array to be made and the parameter mask='res_mask' for making the array using another array and store it in a variable let be named as 'masked'.
- Then return the masked from the function.
- Now create the main function
- Create two arrays one for masking another.
- Then call the function as we have created above and pass both the arrays in the function as a parameter and store the result in a variable let named 'masked'.
- Now for getting the array as a 1-d array we are using numpy.ma.compressed() which passes the masked as a parameter.
- Then print the Masked array.

Example 1: Masking the first array using the second array

In the above example, we are masking the first array using the second array on the basis of the condition that each element of the first array mod 7 is true, those elements which satisfy the condition at that index elements are masked in the first array.

Since we have the array1 = [1,2,4,5,7,8,9] and array2 = [10,12,14,5,7,0,13], we have given the condition array2%7 so in array2 element 14, 7 and 0 satisfies the condition, and they are present at index 2,4 and 5 so at the same index in array1 elements are masked so the resultant array we have [4 7 8].

```
# importing the library
import numpy as np
```

```
# function to create masked array
def masking(ar1, ar2):
# masking the array1 by using array2
# where array2 mod 7 is true
mask = np.ma.masked_where(ar2%7,ar1)
return mask
# main function
if __name__ = = '__main__':
# creating two arrays
x = np.array([1,2,4,5,7,8,9])
y = np.array([10,12,14,5,7,0,13])
# calling masking function to get
# masked array
masked = masking(x,y)
# getting the values as 1-d array which
# are non masked
masked_array = np.ma.compressed(mask)
# printing the resultant array after masking
print(f'Masked Array is:{masked_array}')
```

Q9. Explain Boolean Arrays in Numpy.

Ans :

Boolean arrays

A boolean array is a numpy array with boolean (True/False) values. Such array can be obtained by applying a logical operator to another numpy array:

importnumpyasnp

```
a=np.reshape(np.arange(16),(4,4))# create a 4x4 array of integers
print(a)
[[ 0  1  2  3]
[ 4  5  6  7]
[ 8  9 10 11]
[12 13 14 15]]
large_values=(a>10)# test which elements of a are greated than 10
print(large_values)
[[False False False False]
[False False False False]
[False False False  True]
[ True  True  True  True]]
even_values=(a%2==0)# test which elements of a are even
print(even_values)
[[ True False  True False]
[ True False  True False]
[ True False  True False]
[ True False  True False]]
```

Logical operations on boolean arrays

Boolean arrays can be combined using logical operators:

operator	meaning
~	negation (logical "not")
&	logical "and"
|	logical "or"

```
b=~(a%3==0)# test which elements of a are not divisible by 3
print('array a:\n{}\n'.format(a))
print('array b:\n{}'.format(b))
array a:
[[ 0  1  2  3]
[ 4  5  6  7]
[ 8  9 10 11]
[12 13 14 15]]
array b:
[[False  True  True False]
[ True  True False  True]
[ True False  True  True]
[False  True  True False]]
c=(a%2==0)|(a%3==0)# test which elements of a are divisible by either 2 or 3
print('array a:\n{}\n'.format(a))
print('array c:\n{}'.format(c))
array a:
```

```
[[ 0  1  2  3]
[ 4  5  6  7]
[ 8  9 10 11]
[12 13 14 15]]
array c:
[[ True False  True  True]
[ True False  True False]
[ True  True  True False]
[ True False  True  True]]
```

d=(a%2==0)&(a%3==0)# test which elements of a are divisible by both 2 and 3

print('array a:\n{}\n'.format(a))

print('array d:\n{}'.format(d))

```
array a:
[[ 0  1  2  3]
[ 4  5  6  7]
[ 8  9 10 11]
[12 13 14 15]]
array d:
[[ True False False False]
[False False  True False]
[False False False False]
[ True False False False]]
```

Indexing with boolean arrays

Boolean arrays can be used to select elements of other numpy arrays. If a is any numpy array and b is a boolean array of the same dimensions then a[b] selects all elements of a for which the corresponding value of b is True.

a=np.reshape(np.arange(16),(4,4))# create a 4x4 array of integers

print(a)

```
[[ 0  1  2  3]
[ 4  5  6  7]
[ 8  9 10 11]
[12 13 14 15]]
```

b=(a%2==0)# test which elements of a are even

print(b)

```
[[ True False  True False]
[ True False  True False]
[ True False  True False]
[ True False  True False]]
```

print(a[b])# select all even elements of the array a

```
[ 0  2  4  6  8 10 12 14]
```

We can use this to modify elements of an array that satisfy a logical condition:

a[a%2==0]=100# set values of all even elements of the array a to 100

print(a)

```
[[100   1 100   3]
[100   5 100   7]
[100   9 100  11]
[100  13 100  15]]
```

In the next example we create two numpy arrays, x and y, and set all values of x that are smaller that the corresponding values of y to -1:

x=np.random.random((3,3))# create a 3x3 array of random numbers

y=np.random.random((3,3))

print('array x:\n{}\n'.format(x))

print('array y:\n{}'.format(y))

```
array x:
[[ 0.76755354 0.39784664 0.60511187]
[ 0.9584705  0.42498244 0.71316056]
[ 0.30123811 0.2202371  0.64291291]]
array y:
[[ 0.58221015 0.09077814 0.26814573]
[ 0.91636671 0.41542893 0.07005894]
[ 0.83128003 0.81483812 0.56582282]]
```

x[x<y]=-1

print(x)

```
[[ 0.76755354 0.39784664 0.60511187]
[ 0.9584705  0.42498244 0.71316056]
[-1.        -1.         0.64291291]]
```

4. 6 Fancy Indexing

Q10. What is fancy indexing in Numpy? Explain.

Ans : **(Imp.)**

Besides using indexing & slicing, NumPy provides you with a convenient way to index an array called fancy indexing.

Fancy indexing allows you to index a numpy array using the following:

- Another numpy array
- A Python list
- A sequence of integers

Let's see the following example:

```
import numpy as np
a = np.arange(1, 10)
print(a)
indices = np.array([2, 3, 4])
print(a[indices])
```

Output:

```
[123456789]
[345]
```

First, use the arange() function to create a numpy array that includes numbers from 1 to 9:

```
[123456789]
```

Second, create a second numpy array for indexing:

```
indices = np.array([2, 3, 4])
```

Third, use the indices array for indexing the a array:

```
print(a[indices])
```

4.7 Sorting Arrays

Q11. Explain about Sorting Arrays in Numpy.

Ans :

Sorting means putting elements in an ordered sequence.

Ordered sequence is any sequence that has an order corresponding to elements, like numeric or alphabetical, ascending or descending.

The NumPy ndarray object has a function called sort(), that will sort a specified array.

Example : Sort the array:

```
import numpy as np
arr = np.array([3, 2, 0, 1])
print(np.sort(arr))
```

You can also sort arrays of strings, or any other data type:

Example : Sort the array alphabetically:

```
import numpy as np

arr = np.array(['banana', 'cherry', 'apple'])
print(np.sort(arr))
```

Example : Sort a boolean array:

```
import numpy as np
arr = np.array([True, False, True])
print(np.sort(arr))
```

Sorting a 2-D Array

If you use the sort() method on a 2-D array, both arrays will be sorted:

Example: Sort a 2-D array:

```
import numpy as np
arr = np.array([[3, 2, 4], [5, 0, 1]])
print(np.sort(arr))
```

4.8 Structured Data

Q12. How to handle structured data in Numpy? Explain.

Ans : **(Imp.)**

Numpy's Structured Array is similar to Struct in C. It is used for grouping data of different types and sizes. Structure array uses data containers called fields. Each data field can contain data of any type and size. Array elements can be accessed with the help of dot notation.

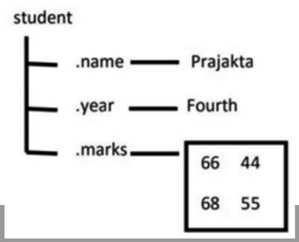
Note

Arrays with named fields that can contain data of various types and sizes.

Properties of Structured array

- All structs in array have same number of fields.
- All structs have same fields names.

For example, consider a structured array of student which has different fields like name, year, marks.



Each record in array student has a structure of class Struct. The array of a structure is referred to as struct as adding any new fields for a new struct in the array, contains the empty array.

Example 1

# Python program to demonstrate # Structured array

import numpy as np

a = np.array([('Sana', 2, 21.0), ('Mansi', 7, 29.0)],

dtype=[('name', (np.str_, 10)), ('age', np.int32), ('weight', np.float64)])

print(a)

Output

[('Sana', 2, 21.0) ('Mansi', 7, 29.0)]

Example 2

The structure array can be sorted by using numpy.sort() method and passing the order as parameter. This parameter takes the value of the field according to which it is needed to be sorted.

# Python program to demonstrate# Structured array

import numpy as np

a = np.array([('Sana', 2, 21.0), ('Mansi', 7, 29.0)],

dtype=[('name', (np.str_, 10)), ('age', np.int32), ('weight', np.float64)])

# Sorting according to the name

b = np.sort(a, order='name')

print('Sorting according to the name', b)

# Sorting according to the age

b = np.sort(a, order='age')

print('\nSorting according to the age', b)

Output

Sorting according to the name [('Mansi', 7, 29.0) ('Sana', 2, 21.0)].

Sorting according to the age [('Sana', 2, 21.0) ('Mansi', 7, 29.0)].

Short Question and Answers

1. What is NumPy?

Ans :

NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays. It is the fundamental package for scientific computing with Python. It is open-source software. It contains various features including these important ones:

- A powerful N-dimensional array object
- Sophisticated (broadcasting) functions
- Tools for integrating C/C++ and Fortran code
- Useful linear algebra, Fourier transform, and random number capabilities

2. Array Indexing

Ans :

Array Indexing

Knowing the basics of array indexing is important for analysing and manipulating the array object. NumPy offers many ways to do array indexing.

- **Slicing:** Just like lists in python, NumPy arrays can be sliced. As arrays can be multidimensional, you need to specify a slice for each dimension of the array.
- **Integer array indexing:** In this method, lists are passed for indexing for each dimension. One to one mapping of corresponding elements is done to construct a new arbitrary array.
- **Boolean array indexing:** This method is used when we want to pick elements from array which satisfy some condition.

3. Aggregations in Numpy

Ans :

Aggregation is a concept in which an object of one class can own or access another independent object of another class.

- It represents Has-A's relationship.
- It is a unidirectional association i.e. a one-way relationship. For example, a department can have students but vice versa is not possible and thus unidirectional in nature.
- In Aggregation, both the entries can survive individually which means ending one entity will not affect the other entity.

4. MaskedArray Module

Ans : **(Imp.)**

Using Masking of arrays we can easily handle the missing, invalid, or unwanted entries in our array or dataset/dataframe. Masking is essential works with the list of Boolean values i.e, True or False which when applied to an original array to return the element of interest, here True refers to the value that satisfies the given condition whereas False refers to values that fail to satisfy the condition.

5. Boolean Arrays

Ans : **(Imp.)**

A boolean array is a numpy array with boolean (True/False) values. Such array can be obtained by applying a logical operator to another numpy array:

importnumpyasnp

```
a=np.reshape(np.arange(16),(4,4))# create a 4x4 array of integers
print(a)
[[ 0  1  2  3]
[ 4  5  6  7]
[ 8  9 10 11]
[12 13 14 15]]
large_values=(a>10)# test which elements of a are greated than 10
print(large_values)
[[False False False False]
[False False False False]
[False False False  True]
[ True  True  True  True]]
even_values=(a%2==0)# test which elements of a are even
print(even_values)
```

```
[[ True False  True False]
 [ True False  True False]
 [ True False  True False]
 [ True False  True False]]
```

6. What is fancy indexing in Numpy? Explain.

Ans : **(Imp.)**

Besides using indexing & slicing, NumPy provides you with a convenient way to index an array called fancy indexing.

Fancy indexing allows you to index a numpy array using the following:

- Another numpy array
- A Python list
- A sequence of integers

Let's see the following example:

```
import numpy as np
a = np.arange(1, 10)
print(a)
indices = np.array([2, 3, 4])
print(a[indices])
```

Output:

```
[123456789]
[345]
```

First, use the arange() function to create a numpy array that includes numbers from 1 to 9:

```
[123456789]
```

Second, create a second numpy array for indexing:

```
indices = np.array([2, 3, 4])
```

Third, use the indices array for indexing the a array:

```
print(a[indices])
```

7. Sorting Arrays

Ans :

Sorting means putting elements in an ordered sequence.

Ordered sequence is any sequence that has an order corresponding to elements, like numeric or alphabetical, ascending or descending.

The NumPy ndarray object has a function called sort(), that will sort a specified array.

Example : Sort the array:

```
import numpy as np
arr = np.array([3, 2, 0, 1])
print(np.sort(arr))
```

You can also sort arrays of strings, or any other data type:

Example : Sort the array alphabetically:

```
import numpy as np
arr = np.array(['banana', 'cherry', 'apple'])
print(np.sort(arr))
```

Example : Sort a boolean array:

```
import numpy as np
arr = np.array([True, False, True])
print(np.sort(arr))
```

Sorting a 2-D Array

If you use the sort() method on a 2-D array, both arrays will be sorted:

Example: Sort a 2-D array:

```
import numpy as np
arr = np.array([[3, 2, 4], [5, 0, 1]])
print(np.sort(arr))
```

8. Structured Data

Ans :

Numpy's Structured Array is similar to Struct in C. It is used for grouping data of different types and sizes. Structure array uses data containers called fields. Each data field can contain data of any type and size. Array elements can be accessed with the help of dot notation.

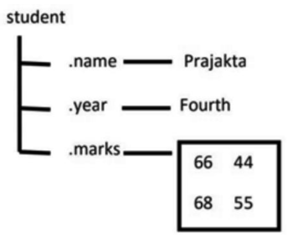
Note

Arrays with named fields that can contain data of various types and sizes.

Properties of Structured array

- All structs in array have same number of fields.
- All structs have same fields names.

For example, consider a structured array of student which has different fields like name, year, marks.



Each record in array student has a structure of class Struct. The array of a structure is referred to as struct as adding any new fields for a new struct in the array, contains the empty array.

Choose the Correct Answers

1. Amongst which Python library is similar to Pandas? [c]
   (a) NPy (b) RPy
   (c) NumPy (d) None of the mentioned above

2. NumPy arrays can be ______. [d]
   (a) Indexed (b) Sliced
   (c) Iterated (d) All of the mentioned above

3. Which of the following Numpy operation are correct? [d]
   (a) Mathematical and logical operations on arrays.
   (b) Fourier transforms and routines for shape manipulation.
   (c) Operations related to linear algebr(a)
   (d) All of the above

4. The basic ndarray is created using? [b]
   (a) Numpy.array(object, dtype = None, copy = True, subok = False, ndmin = 0)
   (b) Numpy.array(object, dtype = None, copy = True, order = None, subok = False, ndmin = 0)
   (c) Numpy_array(object, dtype = None, copy = True, order = None, subok = False, ndmin = 0)
   (d) Numpy.array(object, dtype = None, copy = True, order = None, ndmin = 0)

5. What will be output for the following code? [c]
```
import numpy as np
a = np.array([1,2,3])
print a
```
   (a) [[1, 2, 3]] (b) [1]
   (c) [1, 2, 3] (d) Error

6. Which of the following function stacks 1D arrays as columns into a 2D array? [b]
   (a) row_stack (b) column_stack
   (c) com_stack (d) All of the above

7. Which of the following sets the size of the buffer used in ufuncs? [c]
   (a) bufsize(size) (b) setsize(size)
   (c) setbufsize(size) (d) size(size)

8. Which of the following function stacks 1D arrays as columns into a 2D array? [b]
   (a) row_stack (b) column_stack
   (c) com_stack (d) all of the mentioned

9. Which of the following returns an array of ones with the same shape and type as a given array? [b]
   (a) all_like (b) ones_like
   (c) one_alike (d) all of the mentioned

10. Which of the following function take only single value as input? [a]
   (a) iscomplex (b) minimum
   (c) fmin (d) all of the mentioned

Fill in the Blanks

1. ________ is used when we need to extract a portion of an array from another.
2. We can find the dimension of the array by using ________
3. NumPY stands for ________
4. The most important object defined in NumPy is an N-dimensional array type called ________
5. ndarray is also known as the _______
6. If a dimension is given as ________vin a reshaping operation, the other dimensions are automatically calculated.
7. The ________ function returns its argument with a modified shape, whereas the ________ method modifies the array itself.
8. To create sequences of numbers, NumPy provides a function ________ analogous to range that returns arrays instead of lists.
9. The ________ method makes a complete copy of the array and its data.
10. The ________ function return the element-wise remainder of division.

Answers

1. Slicing
2. Ndim
3. Numerical Python
4. Ndarray
5. Alias array
6. Negetive one
7. Reshape, resize
8. Arrange
9. Copy
10. fmod

UNIT V

Data Manipulation with Pandas: Introducing pandas objects, data indexing and selection, operating on data in pandas, handling missing data, hierarchical indexing, combining datasets, aggregation and grouping

5.1 Data Manipulation With Pandas

5.1.1 Introducing Pandas Objects

Q1. What are panda objects?

Ans :

Meaning

Pandas is an open-source library that is made mainly for working with relational or labeled data both easily and intuitively. It provides various data structures and operations for manipulating numerical data and time series. This library is built on top of the NumPy library. Pandas is fast and it has high performance & productivity for users.

Pandas generally provide two data structures for manipulating data, They are:

- Series
- DataFrame

Q2. Explain, how to create and use series in Pandas.

Ans : **(Imp.)**

Series

The Pandas Series can be defined as a one-dimensional array that is capable of storing various data types. We can easily convert the list, tuple, and dictionary into series using "series' method. The row labels of series are called the index. A Series cannot contain multiple columns. It has the following parameter:

- **data:** It can be any list, dictionary, or scalar value.
- **index:** The value of the index should be unique and hashable. It must be of the same length as data. If we do not pass any index, default np.arrange(n) will be used.
- **dtype:** It refers to the data type of series.
- **copy:** It is used for copying the data.

Creating a Series

We can create a Series in two ways:

1. Create an empty Series
2. Create a Series using inputs.

Create an Empty Series

We can easily create an empty series in Pandas which means it will not have any value.

The syntax that is used for creating an Empty Series:

```
<series object> = pandas.Series()
```

The below example creates an Empty Series type object that has no values and having default datatype, i.e., float64.

Example

```
import pandas as pd
x = pd.Series()
print (x)
```

Output :

```
Series([], dtype: float64)
```

Creating a Series using inputs

We can create Series by using various inputs:

- Array
- Dict
- Scalar value

Creating Series from Array

Before creating a Series, firstly, we have to import the numpy module and then use array() function in the program. If the data is ndarray, then the passed index must be of the same length.

If we do not pass an index, then by default index of range(n) is being passed where n defines the length of an array, i.e., [0,1,2,....range (len(array))-1].

Example

```
import pandas as pd
import numpy as np
info = np.array(['P','a','n','d','a','s'])
a = pd.Series(info)
print(a)
```

Output :

```
0    P
1    a
2    n
3    d
4    a
5    s
dtype: object
```

Create a Series from dict

We can also create a Series from dict. If the dictionary object is being passed as an input and the index is not specified, then the dictionary keys are taken in a sorted order to construct the index.

If index is passed, then values correspond to a particular label in the index will be extracted from the dictionary.

```
#import the pandas library
import pandas as pd
import numpy as np
info = {'x' : 0., 'y' : 1., 'z' : 2.}
a = pd.Series(info)
print (a)
```

Output :

```
x    0.0
y    1.0
z    2.0
dtype: float64
```

Create a Series using Scalar

If we take the scalar values, then the index must be provided. The scalar value will be repeated for matching the length of the index.

```
#import pandas library
import pandas as pd
import numpy as np
x = pd.Series(4, index=[0, 1, 2, 3])
print (x)
```

Output :

```
0    4
1    4
2    4
3    4
dtype: int64
```

Accessing data from series with Position:

Once you create the Series type object, you can access its indexes, data, and even individual elements.

The data in the Series can be accessed similar to that in the ndarray.

```
import pandas as pd
x = pd.Series([1,2,3],index = ['a','b','c'])
#retrieve the first element
print (x[0])
```

Output :

```
1
```

Series object attributes

The Series attribute is defined as any information related to the Series object such as size, datatype. etc. Below are some of the attributes that you can use to get the information about the Series object:

Attributes	Description
Series.index	Defines the index of the Series.
Series.shape	It returns a tuple of shape of the data.
Series.dtype	It returns the data type of the data.
Series.size	It returns the size of the data.
Series.empty	It returns True if Series object is empty, otherwise returns false.
Series.hasnans	It returns True if there are any NaN values, otherwise returns false.
Series.nbytes	It returns the number of bytes in the data.
Series.ndim	It returns the number of dimensions in the data.
Series.itemsize	It returns the size of the datatype of item.

Retrieving Index array and data array of a series object

We can retrieve the index array and data array of an existing Series object by using the attributes index and values.

```
import numpy as np
import pandas as pd
x=pd.Series(data=[2,4,6,8])
y=pd.Series(data=[11.2,18.6,22.5], index=['a','b','c'])
print(x.index)
print(x.values)
print(y.index)
print(y.values)
```

Output :

```
RangeIndex(start=0, stop=4, step=1)
[2 4 6 8]
Index(['a', 'b', 'c'], dtype='object')
[11.2 18.6 22.5]
```

Retrieving Types (dtype) and Size of Type (itemsize)

You can use attribute dtype with Series object as <objectname> dtype for retrieving the data type of an individual element of a series object, you can use the itemsize attribute to show the number of bytes allocated to each data item.

```
import numpy as np
import pandas as pd
a=pd.Series(data=[1,2,3,4])
b=pd.Series(data=[4.9,8.2,5.6],
index=['x','y','z'])
print(a.dtype)
print(a.itemsize)
print(b.dtype)
print(b.itemsize)
```

Output :

```
int64
8
float64
8
```

Retrieving Shape

The shape of the Series object defines total number of elements including missing or empty values(NaN).

```
import numpy as np
import pandas as pd
a=pd.Series(data=[1,2,3,4])
b=pd.Series(data=[4.9,8.2,5.6],index=['x','y','z'])
print(a.shape)
print(b.shape)
```

Output :

```
(4,)
(3,)
```

Retrieving Dimension, Size and Number of bytes:

```
import numpy as np
import pandas as pd
a=pd.Series(data=[1,2,3,4])
b=pd.Series(data=[4.9,8.2,5.6],
index=['x','y','z'])
print(a.ndim, b.ndim)
print(a.size, b.size)
print(a.nbytes, b.nbytes)
```

Output :

```
1 1
4 3
32 24
```

Q3. Write various functions used for series attribute in Pandas.

Ans :

Series Functions

There are some functions used in Series which are as follows:

Functions	Description
Pandas Series.mapQ	Map the values from two series that have a common column.
Pandas Series.stdQ	Calculate the standard deviation of the given set of numbers. DataFrame. column, and rows.
Pandas Series.to_frame()	Convert the series object to the dataframe.
PandasSeries.value_countsO	Returns a Series that contain counts of unique values.

Q4. Expalin , how to use frames in Pandas.

Ans : **(Imp.)**

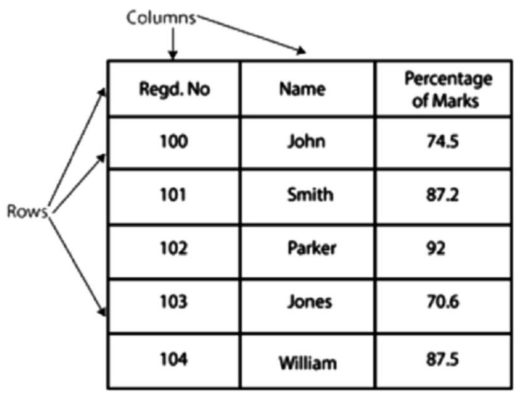
Frames

Pandas DataFrame is a widely used data structure which works with a two-dimensional array with labeled axes (rows and columns). DataFrame is defined as a standard way to store data that has two different indexes, i.e., row index and column index. It consists of the following properties:

- The columns can be heterogeneous types like int, bool, and so on.
- It can be seen as a dictionary of Series structure where both the rows and columns are indexed. It is denoted as "columns" in case of columns and "index" in case of rows.

Parameter & Description

- **data:** It consists of different forms like ndarray, series, map, constants, lists, array.
- **index:** The Default np.arrange(n) index is used for the row labels if no index is passed.
- **columns:** The default syntax is np.arrange(n) for the column labels. It shows only true if no index is passed.
- **dtype:** It refers to the data type of each column.
- **copy():** It is used for copying the data.



Create a DataFrame

We can create a DataFrame using following ways:

- dict
- Lists
- Numpy ndarrrays
- Series

Create an empty DataFrame

The below code shows how to create an empty DataFrame in Pandas:

```
# importing the pandas library
import pandas as pd
df = pd.DataFrame()
print (df)
```

Output :

```
Empty DataFrame
Columns: []
Index: []
```

Create a DataFrame using List

We can easily create a DataFrame in Pandas using list.

```
# importing the pandas library
import pandas as pd
# a list of strings
x = ['Python', 'Pandas']
   # Calling DataFrame constructor on list
df = pd.DataFrame(x)
print(df)
```

Output :

```
0
0  Python
1  Pandas
```

In the above code, we have defined a variable named "x" that consist of string values. The DataFrame constructor is being called for a list to print the values.

Create a DataFrame from Dict of ndarrays/ Lists

```
# importing the pandas library
import pandas as pd
info = {'ID' :[101, 102, 103],'Department' :['B.Sc','B.Tech','M.Tech',]}
```

```
df = pd.DataFrame(info)
print (df)
```

Output :

```
     ID    Department
0    101      B.Sc
1    102      B.Tech
2    103      M.Tech
```

In the above code, we have defined a dictionary named "info" that consist list of ID and Department. For printing the values, we have to call the info dictionary through a variable called df and pass it as an argument in print().

Create a DataFrame from Dict of Series

```
# importing the pandas library
import pandas as pd
info = {'one' : pd.Series([1, 2, 3, 4, 5, 6], index=['a', 'b', 'c', 'd', 'e', 'f']),
    'two' : pd.Series([1, 2, 3, 4, 5, 6, 7, 8], index=['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'])}
d1 = pd.DataFrame(info)
print (d1)
```

Output :

```
      one     two
a     1.0     1
b     2.0     2
c     3.0     3
d     4.0     4
e     5.0     5
f     6.0     6
g     NaN     7
h     NaN     8
```

In the above code, a dictionary named "info" consists of two Series with its respective index. For printing the values, we have to call the info dictionary through a variable called d1 and pass it as an argument in print().

Column Selection

We can select any column from the DataFrame. Here is the code that demonstrates how to select a column from the DataFrame.

```
# importing the pandas library
import pandas as pd
info = {'one' : pd.Series([1, 2, 3, 4, 5, 6], index=['a', 'b', 'c', 'd', 'e', 'f']),
'two' : pd.Series([1, 2, 3, 4, 5, 6, 7, 8], index=['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'])}
d1 = pd.DataFrame(info)
print (d1 ['one'])
```

Output :

```
    a      1.0
    b      2.0
    c      3.0
    d      4.0
    e      5.0
    f      6.0
    g      NaN
    h      NaN
Name: one, dtype: float64
```

In the above code, a dictionary named "info" consists of two Series with its respective index. Later, we have called the info dictionary through a variable d1 and selected the "one" Series from the DataFrame by passing it into the print().

Column Addition

We can also add any new column to an existing DataFrame. The below code demonstrates how to add any new column to an existing DataFrame:

```
# importing the pandas library
import pandas as pd
    info = {'one' : pd.Series([1, 2, 3, 4, 5], index=['a', 'b', 'c', 'd', 'e']),
    'two' : pd.Series([1, 2, 3, 4, 5, 6], index=['a', 'b', 'c', 'd', 'e', 'f'])}
    df = pd.DataFrame(info)
    # Add a new column to an existing DataFrame object
    print ("Add new column by passing series")
df['three']=pd.Series([20,40,60],index=['a','b','c'])
print (df)
    print ("Add new column using existing DataFrame columns")
df['four']=df['one']+df['three']
print (df)
```

Output :

Add new column by passing series

	one	two	three
a	1.0	1	20.0
b	2.0	2	40.0
c	3.0	3	60.0
d	4.0	4	NaN
e	5.0	5	NaN
f	NaN	6	NaN

Add new column using existing DataFrame columns

	one	two	three	four
a	1.0	1	20.0	21.0
b	2.0	2	40.0	42.0
c	3.0	3	60.0	63.0
d	4.0	4	NaN	NaN
e	5.0	5	NaN	NaN
f	NaN	6	NaN	NaN

In the above code, a dictionary named as f consists two Series with its respective index. Later, we have called the info dictionary through a variable df.

To add a new column to an existing DataFrame object, we have passed a new series that contain some values concerning its index and printed its result using print().

We can add the new columns using the existing DataFrame. The "four" column has been added that stores the result of the addition of the two columns, i.e., one and three.

Column Deletion

We can also delete any column from the existing DataFrame. This code helps to demonstrate how the column can be deleted from an existing DataFrame:

```
# importing the pandas library
import pandas as pd
    info = {'one' : pd.Series([1, 2], index= ['a', 'b']),
    'two' : pd.Series([1, 2, 3], index=['a', 'b', 'c'])}
df = pd.DataFrame(info)
    print ("The DataFrame:")
    print (df)
# using del function
print ("Delete the first column:")
del df['one']
```

```
print (df)
# using pop function
print ("Delete the another column:")
df.pop('two')
print (df)
```

Output :

```
The DataFrame:
        one     two
a       1.0     1
b       2.0     2
c       NaN     3
Delete the first column:
        two
a       1
b       2
c       3
Delete the another column:
Empty DataFrame
Columns: []
Index: [a, b, c]
```

In the above code, the df variable is responsible for calling the info dictionary and print the entire values of the dictionary. We can use the delete or pop function to delete the columns from the DataFrame.

In the first case, we have used the delete function to delete the "one" column from the DataFrame whereas in the second case, we have used the pop function to remove the "two" column from the DataFrame.

Row Selection, Addition, and Deletion

Row Selection:

We can easily select, add, or delete any row at anytime. First of all, we will understand the row selection. Let's see how we can select a row using different ways that are as follows:

Selection by Label:

We can select any row by passing the row label to a loc function.

```
# importing the pandas library
import pandas as pd
info = {'one' : pd.Series([1, 2, 3, 4, 5], index=['a', 'b', 'c', 'd', 'e']),
    'two' : pd.Series([1, 2, 3, 4, 5, 6], index=['a', 'b', 'c', 'd', 'e', 'f'])}
    df = pd.DataFrame(info)
print (df.loc['b'])
```

Output :

```
one    2.0
two    2.0
Name: b, dtype: float64
```

In the above code, a dictionary named as info that consists two Series with its respective index.

For selecting a row, we have passed the row label to a loc function.

Selection by integer location:

The rows can also be selected by passing the integer location to an iloc function.

```
# importing the pandas library
import pandas as pd
info = {'one' : pd.Series([1, 2, 3, 4, 5], index=['a', 'b', 'c', 'd', 'e']),
    'two' : pd.Series([1, 2, 3, 4, 5, 6], index=['a', 'b', 'c', 'd', 'e', 'f'])}
df = pd.DataFrame(info)
print (df.iloc[3])
```

Output :

```
    one    4.0
    two    4.0
Name: d, dtype: float64
```

In the above code, we have defined a dictionary named as info that consists two Series with its respective index.

For selecting a row, we have passed the integer location to an iloc function.

Slice Rows

It is another method to select multiple rows using ':' operator.

```
# importing the pandas library
import pandas as pd
info = {'one' : pd.Series([1, 2, 3, 4, 5], index=['a', 'b', 'c', 'd', 'e']),
    'two' : pd.Series([1, 2, 3, 4, 5, 6], index=['a', 'b', 'c', 'd', 'e', 'f'])}
df = pd.DataFrame(info)
print (df[2:5])
```

Output :

```
   one   two
c  3.0   3
d  4.0   4
e  5.0   5
```

In the above code, we have defined a range from 2:5 for the selection of row and then printed its values on the console.

Addition of rows:

We can easily add new rows to the DataFrame using append function. It adds the new rows at the end.

```
# importing the pandas library
import pandas as pd
d = pd.DataFrame([[7, 8], [9, 10]], columns = [‘x’,’y’])
d2 = pd.DataFrame([[11, 12], [13, 14]], columns = [‘x’,’y’])
d = d.append(d2)
print (d)
```

Output :

```
      x    y
0     7    8
1     9    10
0     11   12
1     13   14
```

In the above code, we have defined two separate lists that contains some rows and columns. These columns have been added using the append function and then result is displayed on the console.

Deletion of rows

We can delete or drop any rows from a DataFrame using the index label. If in case, the label is duplicate then multiple rows will be deleted.

```
# importing the pandas library
import pandas as pd
   a_info = pd.DataFrame([[4, 5], [6, 7]], columns = [‘x’,’y’])
b_info = pd.DataFrame([[8, 9], [10, 11]], columns = [‘x’,’y’])
a_info = a_info.append(b_info)
   # Drop rows with label 0
a_info = a_info.drop(0)
```

Output :

```
     x    y
1    6    7
1    10   11
```

Explanation:

In the above code, we have defined two separate lists that contains some rows and columns.

Here, we have defined the index label of a row that needs to be deleted from the list.

5. Write various functions used in Panda data frames

DataFrame Functions

There are lots of functions used in DataFrame which are as follows:

S.No.	Functions	Description
1.	Pandas DataFrame.append	O Add the rows of other dataframe to the end of the given data frame.
2.	Pandas DataFrame.applyO	Allows the user to pass a function and apply it toevery single value of the Pandas series.
3.	Pandas DataFrame.assignQ	Add new column into a dataframe.
4.	Pandas DataFrame.astype()	Cast the Pandas object to a specified dtype.astype()function.
5.	Pandas DataFrame.concat()	Perform concatenation operation along an axis in the DataFrame.
6.	Pandas DataFrame.countO	Count the number of non-NA cells for each columnor row.
7.	Pandas DataFrame.describe()	Calculate some statistical data like percentile, meanand std of the numerical values of the Series orDataFrame.
8.	PandasDataFrame.drop_duplicates()	Remove duplicate values from the DataFrame.
9.	Pandas DataFrame.groupbyO	Split the data into various groups.
10.	Pandas DataFrame.headQ	Returns the first n rows for the object based onposition.
11.	Pandas DataFrame.hist()	Divide the values within a numerical variable into”bins”.
12.	Pandas DataFrame.iterrows()	Iterate over the rows as (index, series) pairs.
13.	Pandas DataFrame.meanO	Return the mean of the values for the requested axis.
14.	Pandas DataFrame.meltO	Unpivots the DataFrame from a wide format to along format.
15.	Pandas DataFrame.merge()	Merge the two datasets together into one.
16.	Pandas DataFrame.pivot_table()	Aggregate data with calculations such as Sum, Count, Average, Max, and Min.
17.	Pandas DataFrame.queryO	Filter the dataframe.
18.	Pandas DataFrame.sample()	Select the rows and columns from the dataframer and omly.
19.	Pandas DataFrame.shift()	Shift column or subtract the column value with the previous row value from the dataframe.
20.	Pandas DataFrame.sort()	Sort the dataframe.
21.	Pandas DataFrame.sum()	Return the sum of the values for the requested axisby the user.
22.	Pandas DataFrame.to_excel()	Export the dataframe to the excel file.
23.	Pandas DataFrame.transpose()	Transpose the index and columns of the dataframe.
24.	Pandas DataFrame.where()	Check the dataframe for one or more conditions.

5.2 Data Indexing and Selection

Q5. Explain about indexing in Pandas.

Ans : **(Imp.)**

Indexing in Pandas

Indexing in pandas means simply selecting particular rows and columns of data from a DataFrame. Indexing could mean selecting all the rows and some of the columns, some of the rows and all of the columns, or some of each of the rows and columns. Indexing can also be known as Subset Selection.

Pandas Indexing using [ ], .loc[], .iloc[ ], .ix[ ]

There are a lot of ways to pull the elements, rows, and columns from a DataFrame. There are some indexing method in Pandas which help in getting an element from a DataFrame. These indexing methods appear very similar but behave very differently. Pandas support four types of Multi-axes indexing they are:

- Dataframe.[ ] ; This function also known as indexing operator
- Dataframe.loc[ ] : This function is used for labels.
- Dataframe.iloc[ ] : This function is used for positions or integer based
- Dataframe.ix[] : This function is used for both label and integer based

Collectively, they are called the indexers. These are by far the most common ways to index data. These are four function which help in getting the elements, rows, and columns from a DataFrame.

Indexing a Dataframe using indexing operator [] :

Indexing operator is used to refer to the square brackets following an object. The .loc and .iloc indexers also use the indexing operator to make selections. In this indexing operator to refer to df[].

Selecting a single columns

In order to select a single column, we simply put the name of the column in-between the brackets

```
# importing pandas package
importpandas as pd
# making data frame from csv file
data =pd.read_csv("nba.csv", index_col =”Name")
# retrieving columns by indexing operator
first =data["Age"]
    print(first)
```

Output :

```
Name
Avery Bradley        25.0
Dae Crowder          25.0
Dohn Holland         27.0
R.D. Hunter          22.0
Donas Derebko        29.0
Amir Dohnson         29.0
Dordan Mickey        21.0
Kelly Olynyk         25.0
Terry Rozier         22.0
Marcus Smart         22.0
Dared Sullinger      24.0
Isaiah Thomas        27.0
Doe Ingles           28.0
Chris Dohnson        26.0
Trey Lyles           20.0
Shelvin Mack         26.0
Raul Neto            24.0
Tibor Pleiss         26.0
Deff Withey          26.0
NaN                  NaN
Name: Age, Length: 458, dtype: float64
```

Selecting multiple columns

In order to select multiple columns, we have to pass a list of columns in an indexing operator.

```
# importing pandas package
importpandas as pd
    # making data frame from csv file
data =pd.read_csv("nba.csv", index_col =”Name")
# retrieving multiple columns by indexing operator
first =data[["Age", "College", "Salary"]]
first
```

Output :

	Age	College	Salary
Name			
Avery Bradley	25.0	Taxes	7730337.0
Jee Crowder	25.0	Marquette	6796117.0
John Holland	27.0	Boston University	NaN
R.J. Hunter	22.0	Georgia State	1148640.0
Jonas Jerebko	29.0	NaN	5000000.0
Amir Johnson	29.0	NaN	12000000.0
Jordan Mickey	21.0	LSU	1170960.0
Kelly Olynyk	25.0	Gonzaga	2165160.0
Terry Rozier	22.0	Louisville	1824360.0
•	•	•	•
•	•	•	•
•	•	•	•
•	•	•	•
•			
•			
Joe ingles	28.0	NaN	2050000.0
Chris Johnson	26.0	Dayton	981348.0
Trey Lyles	20.0	Kentucky	2239800.0
Shelvin Mack	26.0	Butler	2433333.0
Raul Neto	24.0	Nan	900000.0
Tibor Pleiss	26.0	NaN	2900000.0
Jett Withey	26.0	Kansas	947276.0
NaN	NaN	NaN	NaN

458 rows × 3 columns

Indexing a DataFrame using .loc[ ] :

This function selects data by the **label** of the rows and columns. The df.loc indexer selects data in a different way than just the indexing operator. It can select subsets of rows or columns. It can also simultaneously select subsets of rows and columns.

Selecting a single row

In order to select a single row using .loc[], we put a single row label in a .loc function.

```
# importing pandas package
importpandas as pd
    # making data frame from csv file
```

```
data =pd.read_csv("nba.csv", index_col = "Name")
        # retrieving row by loc method
first =data.loc["Avery Bradley"]
second =data.loc["R.J. Hunter"]
print(first, "\n\n\n", second)
```

Output :

As shown in the output image, two series were returned since there was only one parameter both of the times.

```
Team          Boston Celtics
Number                     0
Position                  PG
Age                       25
Height                 6 - 2
Weight                   180
College                Taxes
Salary           7.73θ34e+06
Name : Avery Bradley, dtype : object

Team          Boston Celtics
Number                    28
Position                  5G
Age                       22
Height                 6 - 5
Weight                   185
College        Georgia State
Salary           1.14864e+06
Name : R.J. Hunter, dtype: object
```

Selecting multiple rows

In order to select multiple rows, we put all the row labels in a list and pass that to .loc function.

```
importpandas as pd
# making data frame from csv file
data =pd.read_csv("nba.csv", index_col = "Name")
        # retrieving multiple rows by loc method
first =data.loc[["Avery Bradley", "R.J. Hunter"]]
        print(first)
```

Output :

Name	Team	Number	Position	Age	Height	Weight	College	Salary
Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
R.J. Hunter	Boston Celtics	28.0	SG	22.0	6-5	185.0	Georgia State	1148640.0

Selecting two rows and three columns

In order to select two rows and three columns, we select a two rows which we want to select and three columns and put it in a separate list like this:

```
Dataframe.loc[["row1", "row2"], ["column1", "column2", "column3"]]

importpandas as pd

    # making data frame from csv file

data =pd.read_csv("nba.csv", index_col = "Name")

    # retrieving two rows and three columns by loc method

first =data.loc[["Avery Bradley", "R.J. Hunter"],

            ["Team", "Number", "Position"]]

print(first)
```

Output:

Name	Team	Number	Position
Avery Bradley	Boston Celtics	0.0	PG
R.J. Hunter	Boston Celtics	28.0	SG

Selecting all of the rows and some columns

In order to select all of the rows and some columns, we use single colon **[:]** to select all of rows and list of some columns which we want to select like this:

```
Dataframe.loc[:, ["column1", "column2", "column3"]]

importpandas as pd

# making data frame from csv file

data =pd.read_csv("nba.csv", index_col = "Name")

# retrieving all rows and some columns by loc method

first =data.loc[:, ["Team", "Number", "Position"]]

print(first)
```

Output :

```
                  Team            Number    Position
Name
Avery Bradley     Boston Celtics    0.0       PG
Jae Crowder       Boston Celtics   99.0       SF
John Holland      Boston Celtics   30.0       SG
R.J. Hunter       Boston Celtics   28.0       SG
Jonas Jerebko     Boston Celtics    8.0       PF
Amir Johnson      Boston Celtics   90.0       PF
Jordan Mickey     Boston Celtics   55.0       PF
Kelly Olynyk      Boston Celtics   41.0        C
Terry Rozier      Boston Celtics   12.0       PG
Marcus Smart      Boston Celtics   36.0       PG
Jared Sullinger   Boston Celtics    7.0        C
•                 •                  •         •
•                 •                  •         •
•                 •                  •         •
•                 •                  •         •
•                 •                  •         •
Rudy Gobert       Utah Jazz        27.0        C
Gordon            Utah Jazz        20.0       SF
Rodney            Utah Jazz         5.0       SG
Joe Ingles        Utah Jazz         2.0       SF
Chris Johnson     Utah Jazz        23.0       SF
Trey Lyles        Utah Jazz        41.0       PF
Shelvin Mack      Utah Jazz         8.0       PG
Raul Neto         Utah Jazz        25.0       PG
Tibor pleiss      Utah Jazz        21.0        C
Jeff withey       Utah Jazz        24.0        C
NaN               NaN               NaN      NaN
[458 rows × 3 columns]
```

Indexing a DataFrame using.iloc[ ] :

This function allows us to retrieve rows and columns by position. In order to do that, we'll need to specify the positions of the rows that we want, and the positions of the columns that we want as well. The df.iloc indexer is very similar to df.loc but only uses integer locations to make its selections.

Selecting a single row

In order to select a single row using .iloc[], we can pass a single integer to .iloc[] function.

```
importpandas as pd
```

```
# making data frame from csv file
data =pd.read_csv("nba.csv", index_col = "Name")
# retrieving rows by iloc method
row2 =data.iloc[3]
print(row2)
```

Output :

```
Team                Boston Celtics
Number                          28
Position                        5G
Age                             22
Height                         6-5
Weight                         185
College              Georgia State
Salary                 1.14864e+06
Name : R.3. Hunter, dtype: object
```

Selecting multiple rows

In order to select multiple rows, we can pass a list of integer to .iloc[] function.

```
importpandas as pd
# making data frame from csv file
data =pd.read_csv("nba.csv", index_col = "Name")
# retrieving multiple rows by iloc method
row2 =data.iloc [[3, 5, 7]]
row2
```

Output:

Name	Team	Number	Position	Age	Height	Weight	College	Salary
R.J. Hunter	Boston Celtics	28.0	SG	22.0	6-5	185.0	Georgia State	1148640.0
Amir Johnson	Boston Celtics	90.0	PF	29.0	6-9	240.0	NaN	12000000.0
Kelly Olynyk	Boston Celtics	41.0	C	25.0	7-0	238.0	Gonzaga	2165160.0

Selecting two rows and two columns

In order to select two rows and two columns, we create a list of 2 integer for rows and list of 2 integer for columns then pass to a .iloc[] function.

```
importpandas as pd

# making data frame from csv file

data =pd.read_csv("nba.csv", index_col = "Name")
```

```
# retrieving two rows and two columns by iloc method
row2 =data.iloc [[3, 4], [1, 2]]
print(row2)
```

Output:

```
                    Number      Position
Name
R.D. Hunter         28.0        SG
Donas Derebko       8.0         PF
```

Selecting all the rows and a some columns

In order to select all rows and some columns, we use single colon **[:]** to select all of rows and for columns we make a list of integer then pass to a .iloc[] function.

```
importpandas as pd
     # making data frame from csv file
data =pd.read_csv("nba.csv", index_col =”Name")
# retrieving all rows and some columns by iloc method
row2 =data.iloc [:, [1, 2]]
print(row2)
```

Output:

Name	Team	Number	Position
Avery Bradley	Boston Celtics	0.0	PG
Jae Crowder	Boston Celtics	99.0	SF
John Holland	Boston Celtics	30.0	SG
R.J. Hunter	Boston Celtics	28.0	SG
Jonas Jerebko	Boston Celtics	8.0	PF
Amir Johnson	Boston Celtics	90.0	PF
Jordan Mickey	Boston Celtics	55.0	PF
Kelly Olynyk	Boston Celtics	41.0	C
Terry Rozier	Boston Celtics	12.0	PG
Marcus Smart	Boston Celtics	36.0	PG
Jared Sullinger	Boston Celtics	7.0	C
•	•	•	•
•	•	•	•
•	•	•	•
•	•	•	•
•	•	•	•
Rudy Gobert	Utah Jazz	27.0	C

Gordon Hayward	Utah Jazz	20.0	SF
Rodney Hood	Utah Jazz	5.0	SG
Joe Ingles	Utah Jazz	2.0	SF
Chris Johnson	Utah Jazz	23.0	SF
Trey Lyles	Utah Jazz	41.0	PF
Shelvin Mack	Utah Jazz	8.0	PG
Raul Neto	Utah Jazz	25.0	PG
Tibor pleiss	Utah Jazz	21.0	C
Jeff withey	Utah Jazz	24.0	C
NaN	NaN	NaN	NaN

[458 rows × 3 columns]

Indexing a DataFrame using .iloc[ ] :

This function allows us to retrieve rows and columns by position. In order to do that, we'll need to specify the positions of the rows that we want, and the positions of the columns that we want as well. The df.iloc indexer is very similar to df.loc but only uses integer locations to make its selections.

Selecting a single row

In order to select a single row using .iloc[], we can pass a single integer to .iloc[] function.

```
importpandas as pd
# making data frame from csv file
data =pd.read_csv("nba.csv", index_col =”Name")
# retrieving rows by iloc method
row2 =data.iloc[3]
print(row2)
```

Output:

```
Team        Boston Celtics
Number                  28
Position                SG
Age                     22
Height                 6-5
Weight                 185
College      Georgia State
Salary         1.14864e+06
```

Selecting multiple rows

In order to select multiple rows, we can pass a list of integer to .iloc[] function.

```
importpandas as pd
```

```
# making data frame from csv file
data =pd.read_csv("nba.csv", index_col = "Name")
# retrieving multiple rows by iloc method
row2 =data.iloc [[3, 5, 7]]
row2
```

Output:

Name	Team	Number	Position	Age	Height	Weight	College	Salary
R.J. Hunter	Boston Celtics	28.0	SG	22.0	6-5	185.0	Georgia State	1148640.0
Amir Johnson	Boston Celtics	90.0	PF	29.0	6-9	240.0	NaN	12000000.0
Kelly Olynyk	Boston Celtics	41.0	C	25.0	7-0	238.0	Gonzaga	2165160.0

Selecting two rows and two columns

In order to select two rows and two columns, we create a list of 2 integer for rows and list of 2 integer for columns then pass to a .iloc[] function.

```
importpandas as pd
# making data frame from csv file
data =pd.read_csv("nba.csv", index_col = "Name")
# retrieving two rows and two columns by iloc method
row2 =data.iloc [[3, 4], [1, 2]]
    print(row2)
```

Output:

```
                  Number     Position
Name
R.D. Hunter       28.0       SG
Donas Derebko     8.0        PF
```

Selecting all the rows and a some columns

In order to select all rows and some columns, we use single colon **[:]** to select all of rows and for columns we make a list of integer then pass to a .iloc[] function.

```
importpandas as pd
    # making data frame from csv file
data =pd.read_csv("nba.csv", index_col = "Name")
# retrieving all rows and some columns by iloc method
    row2 =data.iloc [:, [1, 2]]
print(row2)
```

Output:

```
                    Number    Position
Name
Avery Bradley          0.0          PG
Jae  Crowder          99.0          SF
John Holland          30.0          SG
R.J. Hunter           28.0          SG
Jonas Jerebko          8.0          PF
Amir Johnson          90.0          PF
Jordan Mickey         55.0          PF
Kelly Olynyk          41.0           C
Terry Rozier          12.0          PG
Marcus Smart          36.0          PG
Jared Sullinger        7.0           C
Isaiah Thomas          4.0          PG
Evan Turner           11.0          SG
James Young           13.0          SG
•                        •           •
•                        •           •
•                        •           •
•                        •           •
•                        •           •
Rodney Hood            5.0          SG
Joe Ingles             2.0          SF
Chris Johnson         23.0          SF
Trey Lyles            41.0          PF
Shelvin Mack           8.0          PG
Raul Neto             25.0          PG
Tibor pleiss          21.0           C
Jeff withey           24.0           C
NaN                    NaN         NaN

[458  rows  × 2 columns]
```

Indexing a using Dataframe.ix[ ] :

Early in the development of pandas, there existed another indexer, ix. This indexer was capable of selecting both by label and by integer location. While it was versatile, it caused lots of confusion because it's not explicit. Sometimes integers can also be labels for rows or columns. Thus there were instances where it was ambiguous. Generally, ix is label based and acts just as the .loc indexer. However, .ix also supports integer type selections (as in .iloc) where passed an integer. This only works where the index of the DataFrame is not integer based .ix will accept any of the inputs of .loc and .iloc.

Note: The .ix indexer has been deprecated in recent versions of Pandas.

Selecting a single row using .ix[] as .loc[]

In order to select a single row, we put a single row label in a .ix function. This function act similar as .loc[] if we pass a row label as a argument of a function.

```
# importing pandas package
importpandas as pd
# making data frame from csv file
data =pd.read_csv("nba.csv", index_col =”Name")
# retrieving row by ix method
first =data.ix["Avery Bradley"]
print(first)
```

Output:

```
Team              Boston Celtics
Number                         0
Position                      PG
Age                           25
Height                       6-2
Weight                       180
College                    Taxes
Salary               7.73034e+06
Name :  Avery Bradley, dtype: object
```

Selecting a single row using .ix[] as .iloc[]

In order to select a single row, we can pass a single integer to .ix[] function. This function similar as a iloc[] function if we pass an integer in a .ix[] function.

```
# importing pandas package
importpandas as pd
# making data frame from csv file
data =pd.read_csv("nba.csv", index_col =”Name")
# retrieving row by ix method
first =data.ix[1]
print(first)
```

Output:

```
Team           Boston Celtics
Number                     99
Position                   SF
Age                        25
Height                    6-6
Weight                    235
College             Marquette
Salary           6.79612e +06
Name :  Jae Crowder, dtype: object
```

Q6. Write about the methods in indexing in Data frame.

Ans : **(Imp.)**

Methods for indexing in DataFrame

S.No.	Function	Description
1.	Datafiame.headQ	Return top n tows of a data fiama.
2.	Datafiame.tailO	Return bottom n tows of a data fiama.
3.	Datafiame.at[]	Access a single value for a row column label pair.
4.	Datafiame.iat[]	Access a single value for a row column pair by integer position.
5.	Datafiama.tailO	Purely integer-location based indexing for selection by position.
6.	DataF ramelookupO	Label-based "fancy indexing" function for D3taFrame.
7.	DataF rame.popO	Return item and drop from fiama.
8.	DataF rame.xiO	Returns a cross-section (row(s) or columns)) fiom theDataF rame.
9.	DataFrame.getO	C-et item fiom object for given key {DataFrame column, Panelslice, etc.).
10.	DataFrame.isin{)	Return boolean DataF rame showing whether each element in theDataF rame is con tained in values.
11.	DataF rame.whereQ	Return an object of same shape as self and whose correspondingentries are fiom self where cond is True and other wise are fiomother.
12.	DataFrame.ma*kO	Return an objea of same shape aa self and whose correspondingentries are fiom self where cond is False and other wise are non other.
13.	DataF rame.queryQ	Query the columns of a name with a boolean expression.
14.	DataFrame.insertO	Insert column into D3taFrame at specified location.

5.3 Operating On Data In Pandas

Q7. Explain various operations that can perform on Pandas Data Frames.

Ans : **(Imp.)**

DataFrame is an essential data structure in Pandas and there are many way to operate on it. Arithmetic, logical and bit-wise operations can be done across one or more frames.

Operations specific to data analysis include:

- **Subsetting**: Access a specific row/column, range of rows/columns, or a specific item.
- **Slicing**: A form of subsetting in which Python slicing syntax [:] is used. Rows/columns are numbered from zero. Negative integers imply traversal from the last row/column.
- **Filtering**: Obtain rows that fulfil one or more conditions.
- **Reshaping**: Reorganize data such that the number of rows and columns change.
- **Merging**: A DataFrame is merged with another. This can be a simple concatenation of frames or database-style joins.

➢ **Indexing** is a general term for subsetting, slicing and filtering.

Consider a simple DataFrame with index values 10-12 and columns A-C. A row or column can be accessed by its index value or column label respectively. Column label is used directly in []: df['A']. Index values are used via .loc[]: df.loc[10]. We can also access a row by its position using .iloc[]: df.iloc[0].

In our example, there are many ways to get the last item, a scalar value:

➢ Row first: df.iloc[2]['C'], df.loc[12]['C'], df.loc[12, 'C']

➢ Column first: df['C'][12], df['C'].loc[12], df['C'].iloc[2]

To access multiple rows use df.loc[[10,11]] or df.iloc[0:2]. For multiple columns, use df[['A','B']]. To get last two columns of last two rows, we can write df.loc[[11,12], ['B','C']], df.iloc[1:][['B','C']], df[['B','C']].iloc[1:] or df[['B','C']].loc[[11,12]]. Integer lists df.iloc[[1,2],[1,2]] or slicing df.iloc[1:,1:] or df.iloc[-2:, -2:] can be used.

Positional and label-based indexing can be combined: df.loc[df.index[[0, 2]], 'A'].

Callable can be used for indexing. Function takes a Series or DataFrame and must return a valid index. For example, df.loc[:, lambda df: ['A','B']] or df[lambda df: df.columns[0]].

5.4 Handling Missing Data

Q8. Explain, how to handle the missing data.

Ans : **(Imp.)**

Missing data is always a problem in real life scenarios. Areas like machine learning and data mining face severe issues in the accuracy of their model predictions because of poor quality of data caused by missing values. In these areas, missing value treatment is a major point of focus to make their models more accurate and valid.

When and Why Is Data Missed?

Let us consider an online survey for a product. Many a times, people do not share all the information related to them. Few people share their experience, but not how long they are using the product; few people share how long they are using the product, their experience but not their contact information. Thus, in some or the other way a part of data is always missing, and this is very common in real time.

Let us now see how we can handle missing values (say NA or NaN) using Pandas.

```
# import the pandas library
import pandas as pd
import numpy as np
df = pd.DataFrame(np.random.randn(5,3), index=['a','c','e','f',
'h'],columns=['one','two','three'])
df = df.reindex(['a','b','c','d','e','f','g','h'])
print df
```

Its output is as follows:

	one	two	three
a	0.077988	0.476149	0.965836
b	NaN	NaN	NaN
c	-0.390208	-0.551605	-2.301950
d	NaN	NaN	NaN
e	-2.000303	-0.788201	1.510072
f	-0.930230	-0.670473	1.146615
g	NaN	NaN	NaN
h	0.085100	0.532791	0.887415

Using reindexing, we have created a DataFrame with missing values. In the output, NaN means Not a Number.

Check for Missing Values

To make detecting missing values easier (and across different array dtypes), Pandas provides the isnull() and notnull() functions, which are also methods on Series and DataFrame objects "

Example 1

```
import pandas as pd
import numpy as np
df = pd.DataFrame(np.random.randn(5,3), index=['a','c','e','f',
'h'],columns=['one','two','three'])
df = df.reindex(['a','b','c','d','e','f','g','h'])
print df['one'].isnull()
```

Its output is as follows

```
a    False
b    True
c    False
d    True
e    False
f    False
g    True
h    False
Name: one, dtype: bool
```

Calculations with Missing Data

- When summing data, NA will be treated as Zero
- If the data are all NA, then the result will be NA

Example 1

```
import pandas as pd
import numpy as np
```

```
df = pd.DataFrame(np.random.randn(5,3), index=['a','c','e','f',
'h'],columns=['one','two','three'])
df = df.reindex(['a','b','c','d','e','f','g','h'])
print df['one'].sum()
```

Its output is as follows

```
2.02357685917
```

Cleaning / Filling Missing Data

Pandas provides various methods for cleaning the missing values. The fillna function can "fill in" NA values with non-null data in a couple of ways, which we have illustrated in the following sections.

Replace NaN with a Scalar Value

The following program shows how you can replace "NaN" with "0".

```
import pandas as pd
import numpy as np
df = pd.DataFrame(np.random.randn(3,3), index=['a','c','e'],columns=['one', 'two','three'])
df = df.reindex(['a','b','c'])
print df
print("NaN replaced with '0':")
print df.fillna(0)
```

Its output is as follows

	one	two	three
a	-0.576991	-0.741695	0.553172
b	NaN	NaN	NaN
c	0.744328	-1.735166	1.749580

NaN replaced with '0':

	one	two	three
a	-0.576991	-0.741695	0.553172
b	0.000000	0.000000	0.000000
c	0.744328	-1.735166	1.749580

Here, we are filling with value zero; instead we can also fill with any other value.

Fill NA Forward and Backward

Using the concepts of filling discussed in the ReIndexing Chapter we will fill the missing values.

Example 1

```
import pandas as pd
import numpy as np
df = pd.DataFrame(np.random.randn(5,3), index=['a','c','e','f',
‘h’],columns=[‘one’,’two’,’three’])
df = df.reindex([‘a’,’b’,’c’,’d’,’e’,’f’,’g’,’h’])
print df.fillna(method=’pad’)
```

Its output is as follows

	one	two	three
a	0.077988	0.476149	0.965836
b	0.077988	0.476149	0.965836
c	-0.390208	-0.551605	-2.301950
d	-0.390208	-0.551605	-2.301950
e	-2.000303	-0.788201	1.510072
f	-0.930230	-0.670473	1.146615
g	-0.930230	-0.670473	1.146615
h	0.085100	0.532791	0.887415

Drop Missing Values

If you want to simply exclude the missing values, then use the dropna function along with the axis argument. By default, axis=0, i.e., along row, which means that if any value within a row is NA then the whole row is excluded.

Example 1

```
import pandas as pd
import numpy as np
df = pd.DataFrame(np.random.randn(5,3), index=[‘a’,’c’,’e’,’f’,
‘h’],columns=[‘one’,’two’,’three’])
df = df.reindex([‘a’,’b’,’c’,’d’,’e’,’f’,’g’,’h’])
print df.dropna()
```

Its output is as follows

	one	two	three
a	0.077988	0.476149	0.965836
c	-0.390208	-0.551605	-2.301950
e	-2.000303	-0.788201	1.510072
f	-0.930230	-0.670473	1.146615
h	0.085100	0.532791	0.887415

Replace Missing (or) Generic Values

Many times, we have to replace a generic value with some specific value. We can achieve this by applying the replace method.

Replacing NA with a scalar value is equivalent behavior of the fillna() function.

Example 1

```
import pandas as pd
import numpy as np
df = pd.DataFrame({'one':[10,20,30,40,50,2000],'two':[1000,0,30,40,50,60]})
print df.replace({1000:10,2000:60})
```

Its output is as follows

```
     one   two
0    10    10
1    20    0
2    30    30
3    40    40
4    50    50
5    60    60
```

5.5 Hierarchical Indexing

Q9. Explain about the concept of hierarchical indexing in Pandas.

Ans :

The index is like an address, that's how any data point across the data frame or series can be accessed. Rows and columns both have indexes, rows indices are called index and for columns, it's general column names.

Hierarchical Indexes

Hierarchical Indexes are also known as multi-indexing is setting more than one column name as the index. In this article, we are going to use homelessness.csv file.

```
# importing pandas library as alias pd
importpandas as pd
    # calling the pandas read_csv() function.
# and storing the result in DataFrame df
df =pd.read_csv('homelessness.csv')
    print(df.head())
```

Output:

	region	state	individuals	family_members	state_pop
0	East South Central	Alabama	2570.0	864.0	4887681
1	Pacific	Alaska	1434.0	582.0	735139
2	Mountain	Arizona	7259.0	2606.0	7158024
3	West South Central	Arkanasa	2280.0	432.0	3009733
4	Pacific	California	109008.0	20964.0	39461588

In the following data frame, there is no indexing.

Columns in the Dataframe:

```
# using the pandas columns attribute.
col = df.columns
print(col)
```

Output:

```
Index([‘Unnamed: 0’, ‘region’, ‘state’, ‘individuals’, ‘family_members’,
‘state_pop’],
dtype=’object’)
```

To make the column an index, we use the Set_index() function of pandas. If we want to make one column an index, we can simply pass the name of the column as a string in set_index(). If we want to do multi-indexing or Hierarchical Indexing, we pass the list of column names in the set_index().

Below Code demonstrates Hierarchical Indexing in pandas:

```
# using the pandas set_index() function.
df_ind3 = df.set_index([‘region’, ‘state’, ‘individuals’])
    # we can sort the data by using sort_index()
df_ind3.sort_index()
    print(df_ind3.head(10))
```

Output:

region	state	individuals	family_members	state_pop
East South Central	Alabama	2570.0	864.0	4887681
Pacific	Alaska	1434.0	582	735139
Mountain	Arizona	7259.0	2606.0	7158024
West South Central	Arkansas	2280.0	432.0	3009733
Pacific	California	109008.0	20964.0	39461588
Mountain	Colorado	7607.0	3250.0	5691287
New England	Connecticut	2280.0	1696.0	3571520
South Atlantic	Delaware	708.0	374.0	965479
	District of Columbia	3770.0	3134.0	701547
	Florida	21443.0	9587.0	21244317

Now the dataframe is using Hierarchical Indexing or multi-indexing.

Note that here we have made 3 columns as an index (‘region’, ‘state’, ‘individuals’). The first index ‘region’ is called level(0) index, which is on top of the Hierarchy of indexes, next index ‘state’ is level(1) index which is below the main or level(0) index, and so on. So, the Hierarchy of indexes is formed that’s why this is called Hierarchical indexing.

Selecting Data in a Hierarchical Index or using the Hierarchical Indexing:

For selecting the data from the dataframe using the .loc() method we have to pass the name of the indexes in a list.

```
# selecting the 'Pacific' and 'Mountain'
# region from the dataframe.
    # selecting data using level(0) index or main index.
df_ind3_region = df_ind3.loc[['Pacific', 'Mountain']]
    print(df_ind3_region.head(10))
```

Output:

			family_members	state_pop
region	**state**	**individuals**		
Pacific	Alaska	1434.0	582.0	735139
	California	169008.0	20964.0	39461588
	Hawaii	4131.0	2399.0	1420593
	Oregon	11139.0	3337.0	4181886
	Washington	16424.0	5880.0	7523869
Mountain	Arizona	7259.0	2606.0	7158024
	Colorado	7607.0	3250.0	5691287
	Idaho	1297.0	715.0	1750536
	Montana	983.0	422.0	1060665
	Nevada	7058.0	486.0	3027341

We cannot use only level(1) index for getting data from the dataframe, if we do so it will give an error. We can only use level (1) index or the inner indexes with the level(0) or main index with the help list of tuples.

```
# using the inner index 'state' for getting data.
df_ind3_state = df_ind3.loc[['Alaska', 'California', 'Idaho']]
    print(df_ind3_state.head(10))
```

Output:

```
~/ anaconda3/lib/python3.8/site-packages/pandas/core/indexing. py in getting axis (self, key, axis)
    1097                raise ValueError(*Cannot index with multidimensional key *)
    1098
→   1099             return self, _ getitem_iterable (key, axis = axis)
    1100
    1101         # nested tuple slicing
~ / anaconda3/lib/python 3.8/site-packages/pandas/core/indexing.py in _ getiten_iterable(self, key,  axis)
    1035
```

```
        1036    # A collection of keys
→       1037    keyarr, indexer = self, _get _ listlike_indexer (key, axis, raise_missing=False)
        1038    return self.obj._reindex_with_indexers (
        1039            {axis : [keyarr, indexer]}, copy = True, allow_dups=True
```

Using inner levels indexes with the help of a list of tuples :

Using inner levels indexes with the help of a list of tuples:

Syntax:

df.loc[[(level(0) , level(1) , level(2))]]

selecting data by passing all levels index.

```
df_ind3_region_state =df_ind3.loc[[("Pacific", "Alaska", 1434),
                                   ("Pacific", "Hawaii", 4131),
                                   ("Mountain", "Arizona", 7259),
                                   ("Mountain", "Idaho", 1297)]]
df_ind3_region_state
```

Output :

region	state	individuals	family_members	state_pop
Pacific	Alaska	1434.0	582.0	735139
	Hawaii	4131.0	2399.0	1420593
Mountain	Arizona	7259.0	2606.0	7158024
	Idaho	1297.0	715.0	1750536

5.6 Combining Datasets

Q10. Explain, how to Combine the data frames in Panda Using Merge() Function.

Ans : **(Imp.)**

Once your data is in a series or frames, you may need to combine the data to prepare for further processing, as some data may be in one frame and some in another. pandas provides functions for merging and concatenating frames as long as you know whether you want to merge or to concatenate.

Merging

Pandas merge() is defined as the process of bringing the two datasets together into one and aligning the rows based on the common attributes or columns. It is an entry point for all standard database join operations between DataFrame objects:

Syntax:

```
pd.merge(left, right, how='inner', on=None, left_on=None, right_on=None,
left_index=False, right_index=False, sort=True)
```

Parameters:

- **right:** DataFrame or named Series

  It is an object which merges with the DataFrame.
- **how:** {'left', 'right', 'outer', 'inner'}, default 'inner'

  Type of merge to be performed.
- **left:** It use only keys from the left frame, similar to a SQL left outer join; preserve key order.
- **right:** It use only keys from the right frame, similar to a SQL right outer join; preserve key order.
- **outer:** It used the union of keys from both frames, similar to a SQL full outer join; sort keys lexicographically.
- **inner:** It use the intersection of keys from both frames, similar to a SQL inner join; preserve the order of the left keys.
- **on:** label or list

  It is a column or index level names to join on. It must be found in both the left and right DataFrames. If on is None and not merging on indexes, then this defaults to the intersection of the columns in both DataFrames.
- **left_on:** label or list, or array-like

  It is a column or index level names from the left DataFrame to use as a key. It can be an array with length equal to the length of the DataFrame.
- **right_on:** label or list, or array-like

  It is a column or index level names from the right DataFrame to use as keys. It can be an array with length equal to the length of the DataFrame.
- **left_index :** bool, default False

  It uses the index from the left DataFrame as the join key(s), If true. In the case of MultiIndex (hierarchical), many keys in the other DataFrame (either the index or some columns) should match the number of levels.
- **right_index :** bool, default False

  It uses the index from the right DataFrame as the join key. It has the same usage as the left_index.
- **sort:** bool, default False

  If True, it sorts the join keys in lexicographical order in the result DataFrame. Otherwise, the order of the join keys depends on the join type (how keyword).
- **suffixes:** tuple of the (str, str), default ('_x', '_y')

  It suffixes to apply to overlap the column names in the left and right DataFrame, respectively. The columns use (False, False) values to raise an exception on overlapping.
- **copy:** bool, default True

  If True, it returns a copy of the DataFrame.

  Otherwise, It can avoid the copy.
- **indicator:** bool or str, default False

  If True, It adds a column to output DataFrame "_merge" with information on the source of each row. If it is a string, a column with information on the source of each row will be added to output DataFrame, and the column will be named value of a string. The information column is defined as a categorical-type and it takes value of:

- **"left_only"** for the observations whose merge key appears only in 'left' of the DataFrame, whereas,
- **"right_only"** is defined for observations in which merge key appears only in 'right' of the DataFrame,
- **"both"** if the observation's merge key is found in both of them.
- **validate:** str, optional

  If it is specified, it checks the merge type that is given below:
- "one_to_one" or "1:1": It checks if merge keys are unique in both the left and right datasets.
- "one_to_many" or "1:m": It checks if merge keys are unique in only the left dataset.
- "many_to_one" or "m:1": It checks if merge keys are unique in only the right dataset.
- "many_to_many" or "m:m": It is allowed, but does not result in checks.

Example1:

Merge two DataFrames on a key

```
# import the pandas library
import pandas as pd
left = pd.DataFrame({
    'id':[1,2,3,4],
    'Name': ['John', 'Parker', 'Smith', 'Parker'],
    'subject_id':['sub1','sub2','sub4','sub6']})
right = pd.DataFrame({
    'id':[1,2,3,4],
    'Name': ['William', 'Albert', 'Tony', 'Allen'],
    'subject_id':['sub2','sub4','sub3','sub6']})
print (left)
print (right)
```

Output

```
id              Name      subject_id
0       1       John      sub1
1       2       Parker    sub2
2       3       Smith     sub4
3       4       Parker    sub6
        id      Name      subject_id
0       1       William   sub2
1       2       Albert    sub4
2       3       Tony      sub3
3       4       Allen     sub6
```

Example 2:

Merge two DataFrames on multiple keys:

```
import pandas as pd
left = pd.DataFrame({
    'id':[1,2,3,4,5],
    'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
    'subject_id':['sub1','sub2','sub4','sub6','sub5']})
right = pd.DataFrame({
    'id':[1,2,3,4,5],
    'Name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
    'subject_id':['sub2','sub4','sub3','sub6','sub5']})
print pd.merge(left,right,on='id')
```

Output

```
      id   Name_x   subject_id_x   Name_y    subject_id_y
0     1    John     sub1           William   sub2
1     2    Parker   sub2           Albert    sub4
2     3    Smith    sub4            Tony     sub3
3     4    Parker   sub6           Allen     sub6
```

Q11. Explain, How to Combine the data frames in Panda Using Concat() Function

Ans : **(Imp.)**

Concatenating

Pandas is capable of combining Series, DataFrame, and Panel objects through different kinds of set logic for the indexes and the relational algebra functionality.

The concat() function is responsible for performing concatenation operation along an axis in the DataFrame.

Syntax:

```
pd.concat(objs,axis=0,join='outer',join_axes=None,
ignore_index=False)
```

Parameters:

- **objs:** It is a sequence or mapping of series or DataFrame objects.

  If we pass a dict in the DataFrame, then the sorted keys will be used as the keys<.strong> argument, and the values will be selected in that case. If any non-objects are present, then it will be dropped unless they are all none, and in this case, a ValueError will be raised.

- **axis:** It is an axis to concatenate along.
- **join:** Responsible for handling indexes on another axis.
- **join_axes:** A list of index objects. Instead of performing the inner or outer set logic, specific indexes use for the other (n-1) axis.

- **ignore_index:** bool, default value False

  It does not use the index values on the concatenation axis, if true. The resulting axis will be labeled as 0, ..., n - 1.

Returns

A series is returned when we concatenate all the Series along the axis (axis=0). In case if objs contains at least one DataFrame, it returns a DataFrame.

Example1:

```
import pandas as pd
a_data = pd.Series([‘p’, ’q’])
b_data = pd.Series([‘r’, ’s’])
pd.concat([a_data, b_data])
```

Output :

```
0    p
1    q
0    r
1    s
```

dtype: object

Example2:

In the above example, we can reset the existing index by using the ignore_index parameter. The below code demonstrates the working of ignore_index.

```
import pandas as pd
a_data = pd.Series([‘p’, ’q’])
b_data = pd.Series([‘r’, ’s’])
pd.concat([a_data, b_data], ignore_index=True)
```

Output

```
0    p
1    q
2    r
3    s
```

dtype: object

Concatenation using append

The append method is defined as a useful shortcut to concatenate the Series and DataFrame.

Example:

```
import pandas as pd
one = pd.DataFrame({
    ’Name’: [‘Parker’, ’Smith’, ’Allen’, ’John’, ’Parker’],
    'subject_id':['sub1','sub2','sub4','sub6','sub5'],
    'Marks_scored':[98,90,87,69,78]},
    index=[1,2,3,4,5])
two = pd.DataFrame({
    'Name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
    'subject_id':['sub2','sub4','sub3','sub6','sub5'],
    'Marks_scored':[89,80,79,97,88]},
    index=[1,2,3,4,5])
print (one.append(two))
```

5.6.1 Aggregation and Grouping

Q12. Explain data aggregation functions in Pandas.

(OR)

Explain groupby() function in Pandas

Ans : **(Imp.)**

Data Aggregation

Data aggregation is a three-step procedure during which data is split, aggregated, and combined:

1. At the split step, the data is split by key or keys into chunks.
2. At the apply step, an aggregation function (such as sum() or count()) is applied to each chunk.
3. At the combine step, the calculated results are combined into a new series or frame.

groupby()

In Pandas, groupby() function allows us to rearrange the data by utilizing them on real-world data sets. Its primary task is to split the data into various groups. These groups are categorized based on some criteria. The objects can be divided from any of their axes.

Syntax:

DataFrame.groupby(by=None, axis=0, level=None, as_index=True, sort=True, group_keys = True, squeeze= False, **kwargs)

This operation consists of the following steps for aggregating/grouping the data:

- Splitting datasets
- Analyzing data
- Aggregating or combining data

Split data into groups

There are multiple ways to split any object into the group which are as follows:

- obj.groupby('key')
- obj.groupby(['key1','key2'])
- obj.groupby(key,axis=1)

We can also add some functionality to each subset. The following operations can be performed on the applied functionality:

- **Aggregation:** Computes summary statistic.
- **Transformation:** It performs some group-specific operation.
- **Filtration:** It filters the data by discarding it with some condition.

Aggregations

It is defined as a function that returns a single aggregated value for each of the groups. We can perform several aggregation operations on the grouped data when the groupby object is created.

Example

```
# import the pandas library
import pandas as pd
import numpy as np
data = {'Name': ['Parker', 'Smith', 'John', 'William'],
'Percentage': [82, 98, 91, 87],
'Course': ['B.Sc','B.Ed','M.Phill','BA']}
df = pd.DataFrame(data)
    grouped = df.groupby('Course')
print(grouped['Percentage'].agg(np.mean))
```

Output

```
Course
B.Ed      98
B.Sc      82
BA        87
M.Phill   91
Name: Percentage, dtype: int64
```

Transformations

It is an operation on a group or column that performs some group-specific computation and returns an object that is indexed with the same size as of the group size.

Example

```
# import the pandas library
import pandas as pd
import numpy as np
    data = {'Name': ['Parker', 'Smith', 'John', 'William'],
         'Percentage': [82, 98, 91, 87],
    'Course': ['B.Sc','B.Ed','M.Phill','BA']}
df = pd.DataFrame(data)
```

Output

```
        Percentage
0       NaN
1       NaN
2       NaN
3       NaN
      BA
```

Parameters of Groupby:

➢ **by:** mapping, function, str, or iterable

Its main task is to determine the groups in the groupby. If we use by as a function, it is called on each value of the object's index. If in case a dict or Series is passed, then the Series or dict VALUES will be used to determine the groups.

If a ndarray is passed, then the values are used as-is determine the groups.

We can also pass the label or list of labels to group by the columns in the self.

➢ **axis:** {0 or 'index', 1 or 'columns'}, default value 0

➢ **level:** int, level name, or sequence of such, default value None.

It is used when the axis is a MultiIndex (hierarchical), so, it will group by a particular level or levels.

➢ **as_index:** bool, default True

It returns the object with group labels as the index for the aggregated output.

➢ **sort:** bool, default True

It is used to sort the group keys. Get better performance by turning this off.

➢ **group_keys:** bool, default value True

When we call it, it adds the group keys to the index for identifying the pieces.

➢ **observed:** bool, default value False

It will be used only if any of the groupers are the Categoricals. If the value is True, then it will show only the observed values for categorical groupers. Otherwise, it will show all of its values.

➢ ****kwargs**

It is an optional parameter that only accepts the keyword argument 'mutated' that is passed to groupby.

Returns

It returns the DataFrameGroupBy or SeriesGroupBy. The return value depends on the calling object that consists of information about the groups.

Example

```
        import pandas as pd
info = pd.DataFrame({'Name': ['Parker', 'Smith',
'John','William'],'Percentage':
[92.,98., 89., 86.]})
        info
```

Output

```
        Name      Percentage
0       Parker    92.0
1       Smith     98.0
2       John      89.0
3       William   86.0
```

Example

```
        # import the pandas library
        import pandas as pd

data = {'Name': ['Parker', 'Smith', 'John', 'William'],
            'Percentage': [82, 98, 91, 87],}
        info = pd.DataFrame(data)
            print (info)
```

Output

```
        Name      Percentage
0       Parker    82
1       Smith     98
2       John      91
3       William   87
```

Short Question and Answers

1. What are panda objects?

Ans :

Meaning

Pandas is an open-source library that is made mainly for working with relational or labeled data both easily and intuitively. It provides various data structures and operations for manipulating numerical data and time series. This library is built on top of the NumPy library. Pandas is fast and it has high performance & productivity for users.

Pandas generally provide two data structures for manipulating data, They are:

- Series
- DataFrame

2. Series

Ans :

The Pandas Series can be defined as a one-dimensional array that is capable of storing various data types. We can easily convert the list, tuple, and dictionary into series using “series’ method. The row labels of series are called the index. A Series cannot contain multiple columns. It has the following parameter:

- **data:** It can be any list, dictionary, or scalar value.
- **index:** The value of the index should be unique and hashable. It must be of the same length as data. If we do not pass any index, default np.arrange(n) will be used.
- **dtype:** It refers to the data type of series.
- **copy:** It is used for copying the data.

3. Frames in Pandas.

Ans :

Pandas DataFrame is a widely used data structure which works with a two-dimensional array with labeled axes (rows and columns). DataFrame is defined as a standard way to store data that has two different indexes, i.e., row index and column index. It consists of the following properties:

- The columns can be heterogeneous types like int, bool, and so on.
- It can be seen as a dictionary of Series structure where both the rows and columns are indexed. It is denoted as “columns” in case of columns and “index” in case of rows.

4. Write various functions used for series attribute in Pandas.

Ans :

Series Functions

There are some functions used in Series which are as follows:

Functions	Description
Pandas Series.mapQ	Map the values from two series that have a common column.
Pandas Series.stdQ	Calculate the standard deviation of the given set of numbers. DataFrame. column, and rows.
Pandas Series.to_frame()	Convert the series object to the dataframe.
PandasSeries.value_countsO	Returns a Series that contain counts of unique values.

5. indexing in Pandas.

Ans :

Indexing in Pandas

Indexing in pandas means simply selecting particular rows and columns of data from a DataFrame. Indexing could mean selecting all the rows and some of the columns, some of the rows and all of the columns, or some of each of the rows and columns. Indexing can also be known as Subset Selection.

Pandas Indexing using [ ], .loc[], .iloc[ ], .ix[ ]

There are a lot of ways to pull the elements, rows, and columns from a DataFrame. There are some indexing method in Pandas which help in getting an element from a DataFrame. These indexing methods appear very similar but behave very differently. Pandas support four types of Multi-axes indexing they are:

- Dataframe.[ ] ; This function also known as indexing operator
- Dataframe.loc[ ] : This function is used for labels.
- Dataframe.iloc[ ] : This function is used for positions or integer based
- Dataframe.ix[] : This function is used for both label and integer based

6. Methods in indexing in Data frame.

Ans :

Methods for indexing in DataFrame

S.No.	Function	Description
1.	Datafiame.headQ	Return top n tows of a data fiama.
2.	Datafiame.tailO	Return bottom n tows of a data fiama.
3.	Datafiame.at[]	Access a single value for a row column label pair.
4.	Datafiame.iat[]	Access a single value for a row column pair by integer position.
5.	Datafiama.tailO	Purely integer-location based indexing for selection by position.
6.	DataF ramelookupO	Label-based “fancy indexing” function for D3taFrame.
7.	DataF rame.popO	Return item and drop from fiama.

7. Hierarchical indexing in Pandas.

Ans :

The index is like an address, that’s how any data point across the data frame or series can be accessed. Rows and columns both have indexes, rows indices are called index and for columns, it’s general column names.

8. Data Aggregation

Ans :

Data aggregation is a three-step procedure during which data is split, aggregated, and combined:

1. At the split step, the data is split by key or keys into chunks.
2. At the apply step, an aggregation function (such as sum() or count()) is applied to each chunk.
3. At the combine step, the calculated results are combined into a new series or frame.

9. Groupby() function in Pandas

Ans :

In Pandas, groupby() function allows us to rearrange the data by utilizing them on real-world data sets. Its primary task is to split the data into various groups. These groups are categorized based on some criteria. The objects can be divided from any of their axes.

Syntax:

DataFrame.groupby(by=None, axis=0, level=None, as_index=True, sort=True, group_keys = True, squeeze= False, **kwargs)

This operation consists of the following steps for aggregating/grouping the data:

- Splitting datasets
- Analyzing data
- Aggregating or combining data

Split data into groups

There are multiple ways to split any object into the group which are as follows:

- obj.groupby('key')
- obj.groupby(['key1','key2'])
- obj.groupby(key,axis=1)

We can also add some functionality to each subset. The following operations can be performed on the applied functionality:

- **Aggregation:** Computes summary statistic.
- **Transformation:** It performs some group-specific operation.
- **Filtration:** It filters the data by discarding it with some condition.

10. Parameters of Groupby.

Ans :

- **by:** mapping, function, str, or iterable

  Its main task is to determine the groups in the groupby. If we use by as a function, it is called on each value of the object's index. If in case a dict or Series is passed, then the Series or dict VALUES will be used to determine the groups.

  If a ndarray is passed, then the values are used as-is determine the groups.

  We can also pass the label or list of labels to group by the columns in the self.
- **axis:** {0 or 'index', 1 or 'columns'}, default value 0
- **level:** int, level name, or sequence of such, default value None.

  It is used when the axis is a MultiIndex (hierarchical), so, it will group by a particular level or levels.
- **as_index:** bool, default True

  It returns the object with group labels as the index for the aggregated output.
- **sort:** bool, default True

  It is used to sort the group keys. Get better performance by turning this off.
- **group_keys:** bool, default value True

  When we call it, it adds the group keys to the index for identifying the pieces.
- **observed:** bool, default value False

  It will be used only if any of the groupers are the Categoricals. If the value is True, then it will show only the observed values for categorical groupers. Otherwise, it will show all of its values.

Choose the Correct Answer

1. Amongst which of the following is / are used to analyze the data in pandas. [c]

   (a) Dataframe (c) Series

   (c) Both A and B (d) None of the mentioned above

2. Amongst which of the following is a correct syntax for panda's dataframe? [a]

   (a) Pandas.DataFrame(data, index, dtype, copy)

   (b) pandas.DataFrame(data, index, columns, dtype, copy)

   (c) pandas.DataFrame(data, index, dtype, copy)

   (d) pandas.DataFrame(data, index, rows, dtype, copy)

3. Amongst which of the following can be used to create various inputs using pandas DataFrame. [d]

   (a) Lists, dict (b) Series

   (c) Numpy ndarrays and Another DataFrame (d) All of the above mentioned

4. Which of the following thing can be data in Pandas? [d]

   (a) a python dict (b) an ndarray

   (c) a scalar value (d) all of the mentioned

5. Which of the following input can be accepted by DataFrame? [d]

   (a) Structured ndarray (b) Series

   (c) DataFrame (d) All of the mentioned

6. Which of the following takes a dict of dicts or a dict of array-like sequences and returns a DataFrame? [a]

   (a) DataFrame.from_items (b) DataFrame.from_records

   (c) DataFrame.from_dict (d) All of the mentioned

7. What will be correct syntax for pandas series? [c]

   (a) pandas_Series(data, index, dtype, copy)

   (b) pandas.Series(data, index, dtype)

   (c) pandas.Series(data, index, dtype, copy)

   (d) pandas_Series(data, index, dtype)

8. To count total number of rows in data frame we use ________. [b]

   (a) Count() (b) Len()

   (c) Values() (d) All the above

9. Which of the following indexing capabilities is used as a concise means of selecting data from a pandas object? [b]

   (a) In (b) ix

   (c) ipy (d) iy

10. What will be output for the following code? [c]

```
import pandas as pd
import numpy as np
s = pd.Series(np.random.randn(2))
print s.size
```

   (a) 0 (b) 1

   (c) 2 (d) 3

Fill in the blanks

1. PANDAS stands for ________.
2. ________ objects are variable in terms of their values, but they are immutable in terms of their sizes.
3. ________ Objects are mutable in terms of their values, but they are not mutable in terms of their sizes.
4. ________ are the minimum number of arguments require to pass in pandas series?
5. ________ is one one-dimensional labeled array that can store any data type like integers, strings, floating-point numbers, Python objects, etc.
6. A ________ is a 2-dimensional labeled data structure with columns that can be of a variety of different kinds.
7. A pandas ________ can be created using various inputs like Lists, dict, Series, Numpy ndarrays, Another DataFrame.
8. ________ is similar to NumPy arrays and is used to retrieve entries inside a series of elements.
9. ________ is used when data is in tabular format.
10. In Panda the ________ format keeps an arrays of all of the locations where the data are not equal to the fill value.

Answers

1. Panel Data Analysis
2. Series Objects
3. Sequence
4. 1
5. Series
6. DataFrame
7. DataFrame
8. Indexing.
9. Panda
10. integer

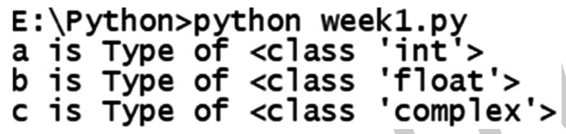
Lab Programming

1. Write a program to demonstrate different numbers data types in python.

Ans :

```
        '''Aim:Write a program to demonstrate different
number data types in Python.'''
        a=10;#Integer Datatype
        b=11.5;#Float Datatype
        c=2.05j;#Complex Number
        print("a is Type of",type(a));#prints type of variable a
        print("b is Type of",type(b));#prints type of variable b
        print("c is Type of",type(c));#prints type of variable c
```

Output:



```
E:\Python>python week1.py
a is Type of <class 'int'>
b is Type of <class 'float'>
c is Type of <class 'complex'>
```

2. Write a python program to design simple calculator using functions.

Ans :

```
# Python Program to Make a Simple Calculator
        def multiplication(num1, num2):
        return num1 * num2
def addition(num1, num2):
        return num1 + num2
        def subtraction(num1, num2):
            return num1 - num2
        def divide(num1, num2):
            return num1 / num2
        value1 = int(input("Enter 1st number: "))
        value2 = int(input("Enter 2nd number: "))
        print("Select operation 1-Division, 2-Multiplication, 3-Addition, 4-Subtraction")
operation = int(input("Choose operation 1/2/3/4: "))
        if operation = = 1:
print(value1, "/", value2, "=", divide(value1, value2))
        elif operation = = 2:
print(value1, "*", value2, "=", multiplication
        (value1, value2))
        elif operation = = 3:
print(value1, "+", value2, "=", addition(value1, value2))
        elif operation = = 4:
print(value1, "-", value2, "=", subtraction
        (value1, value2))
else:
        print("Enter correct operation")
```

Output:

```
        Enter 1st Number: 2
        Enter 2nd Number: 2
        Select operation 1-Division, 2-Multiplication,
3-Addition, 4-Subtraction
        Choose operation 1/2/3/4: 1
        2 / 2 = 1.0
```

3. Write a python program to check whether a given number is Armstrong number or not.

Ans :

```
        # Python program to check if the number is an
        Armstrong number or not
        # take input from the user
        num = int(input("Enter a number: "))
        # initialize sum
        sum = 0
        # find the sum of the cube of each digit
        temp = num
        while temp >0:
        digit = temp % 10
```

```
sum + = digit ** 3
temp //= 10
# display the result
if num = = sum:
print(num,"is an Armstrong number")
else:
print(num,"is not an Armstrong number")
```

Output 1

```
Enter a number: 663
663 is not an Armstrong number
```

4. Write a python program to generate prime numbers between different intervals.

Ans :

```
# Python program to display all the prime numbers within an interval
lower = 900
upper = 1000
print("Prime numbers between", lower, "and", upper, "are:")
for num in range(lower, upper + 1):
# all prime numbers are greater than 1
if num >1:
for i in range(2, num):
if (num % i) = = 0:
break
else:
print(num)
```

Output

```
Prime numbers between 900 and 1000 are:
907
911
919
929
937
941
947
953
967
971
977
983
991
997
```

5. Write a python program to find factorial of a number using recursion.

Ans :

```
# Factorial of a number using recursion
defrecur_factorial(n):
if n = = 1:
return n
else:
return n*recur_factorial(n-1)
num = 7
# check if the number is negative
if num <0:
print("Sorry, factorial does not exist for negative numbers")
elif num = = 0:
print("The factorial of 0 is 1")
else:
print("The factorial of", num, "is", recur _ factorial(num))
```

Output

```
The factorial of 7 is 5040
```

6. Write a python program to check whether a string is palindrome or not.

Ans :

```
# Program to check if a string is palindrome or not
my_str = 'aIbohPhoBiA'
# make it suitable for caseless comparison
my_str = my_str.casefold()
# reverse the string
```

rev_str = reversed(my_str)

# check if the string is equal to its reverse

if list(my_str) = = list(rev_str):

print("The string is a palindrome.")

else:

print("The string is not a palindrome.")

Output

The string is a palindrome.

7. Write a python program to count the number of characters present in a word.

Ans :

# Python Program to Count Total Characters in a String

str1 = input("Please Enter your Own String : ")

total = 0

for i in range(len(str1)):

    total = total + 1

print("Total Number of Characters in this String = ", total)

Please Enter your Own String : Tutorial Gateway

Total Number of Characters in this String = 16

>>>

Please Enter your Own String : Python

Total Number of Characters in this String = 6

8. Write a python program to create, append and remove lists.

Ans :

'''Aim: Write a program to create, append, and remove lists in python. '''

pets =['cat','dog','rat','pig','tiger']

snakes=['python','anaconda','fish','cobra','

mamba']

print("Pets are :",pets)

print("Snakes are :",snakes)

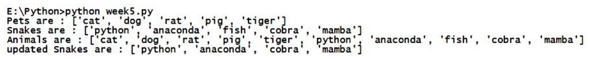
animals=pets+snakes

print("Animals are :",animals)

snakes.remove("fish")

print("updated Snakes are :",snakes)

Output:



```
E:\Python>python week5.py
Pets are : ['cat', 'dog', 'rat', 'pig', 'tiger']
Snakes are : ['python', 'anaconda', 'fish', 'cobra', 'mamba']
Animals are : ['cat', 'dog', 'rat', 'pig', 'tiger', 'python', 'anaconda', 'fish', 'cobra', 'mamba']
updated Snakes are : ['python', 'anaconda', 'cobra', 'mamba']
```

9. Write a program to demonstrate working with tuples in python.

Ans :

'''Write a program to demonstrate working with tuples in python'''

T=("apple","banana","cherry"," mango"," grape","orange")

print("\n Created tuple is :",T)

print("\n Second fruit is :",T[1])

print("\n From 3-6 fruits are :",T[3:6])

print("\n List of all items in Tuple :")

for x in T:

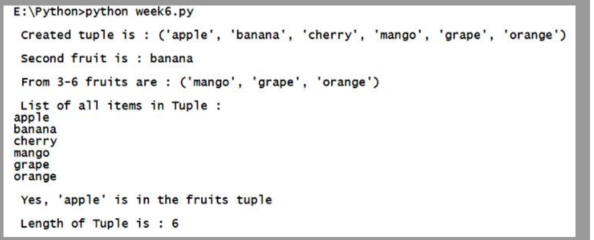
print(x)

if"apple"in T:

print("\n Yes, 'apple' is in the fruits tuple")

print("\n Length of Tuple is :",len(T))

Output:



```
E:\Python>python week6.py
 Created tuple is : ('apple', 'banana', 'cherry', 'mango', 'grape', 'orange')
 Second fruit is : banana
 From 3-6 fruits are : ('mango', 'grape', 'orange')
 List of all items in Tuple :
apple
banana
cherry
mango
grape
orange
 Yes, 'apple' is in the fruits tuple
 Length of Tuple is : 6
```

10. Write a program to demonstrate dictionaries in python.

'''Write a program to demonstrate working with dictionaries in python.'''

Ans :

dict1 ={'StdNo':'532','StuName':'Naveen', 'StuAge': 21,'StuCity':'Hyderabad'}

print("\n Dictionary is :",dict1)

#Accessing specific values

print("\n Student Name is :",dict1['StuName'])

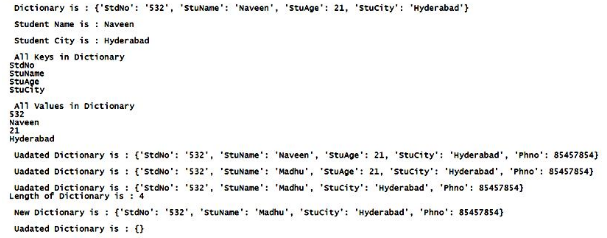
print("\n Student City is :",dict1['StuCity'])

#Display all Keys

print("\n All Keys in Dictionary ")

```
for x in dict1:
print(x)
#Display all values
print("\n All Values in Dictionary ")
for x in dict1:
print(dict1[x])
#Adding items
dict1["Phno"]=85457854
#Updated dictoinary
print("\n Uadated Dictionary is :",dict1)
#Change values
dict1["StuName"]="Madhu"
#Updated dictoinary
print("\n Uadated Dictionary is :",dict1)
#Removing Items
dict1.pop("StuAge");
#Updated dictoinary
print("\n Uadated Dictionary is :",dict1)
#Length of Dictionary
print("Length of Dictionary is :",len(dict1))
#Copy a Dictionary
dict2=dict1.copy()
#New dictoinary
print("\n New Dictionary is :",dict2)
#empties the dictionary
dict1.clear()
print("\n Uadated Dictionary is :",dict1)
```

Output:



```
E:\Python>python week7.py
 Dictionary is : {'StdNo': '532', 'StuName': 'Naveen', 'StuAge': 21, 'StuCity': 'Hyderabad'}
 Student Name is : Naveen
 Student City is : Hyderabad
 All Keys in Dictionary
StdNo
StuName
StuAge
StuCity
 All Values in Dictionary
532
Naveen
21
Hyderabad
 Uadated Dictionary is : {'StdNo': '532', 'StuName': 'Naveen', 'StuAge': 21, 'StuCity': 'Hyderabad', 'Phno': 85457854}
 Uadated Dictionary is : {'StdNo': '532', 'StuName': 'Madhu', 'StuAge': 21, 'StuCity': 'Hyderabad', 'Phno': 85457854}
 Uadated Dictionary is : {'StdNo': '532', 'StuName': 'Madhu', 'StuCity': 'Hyderabad', 'Phno': 85457854}
Length of Dictionary is : 4
 New Dictionary is : {'StdNo': '532', 'StuName': 'Madhu', 'StuCity': 'Hyderabad', 'Phno': 85457854}
 Uadated Dictionary is : {}
```

Numpy

11. Python program to demonstrate basic array characteristics.

Ans :

```
# Python program to demonstrate
# basic array characteristics
import numpy as np
# Creating array object
arr = np.array( [[ 1, 2, 3],
                  [ 4, 2, 5]] )
# Printing type of arr object
print("Array is of type: ", type(arr))
# Printing array dimensions (axes)
print("No. of dimensions: ", arr.ndim)
# Printing shape of array
print("Shape of array: ", arr.shape)
# Printing size (total number of elements) of array
print("Size of array: ", arr.size)
# Printing type of elements in array
print("Array stores elements of type: ", arr.dtype)
```

Output :

```
Array is of type:
No. of dimensions:  2
Shape of array:  (2, 3)
Size of array:  6
Array stores elements of type:  int64
```

12. Python program to demonstrate array creation techniques.

Ans :

```
# Python program to demonstrate
# array creation techniques
import numpy as np
# Creating array from list with type float
a = np.array([[1, 2, 4], [5, 8, 7]], dtype = 'float')
print ("Array created using passed list:\n", a)
# Creating array from tuple
b = np.array((1 , 3, 2))
```

```
print ("\nArray created using passed tuple:\n", b)
# Creating a 3X4 array with all zeros
c = np.zeros((3, 4))
print ("\nAn array initialized with all zeros:\n", c)
# Create a constant value array of complex type
d = np.full((3, 3), 6, dtype = 'complex')
print ("\nAn array initialized with all 6s."
                "Array type is complex:\n", d)
# Create an array with random values
e = np.random.random((2, 2))
print ("\nA random array:\n", e)
# Create a sequence of integers
# from 0 to 30 with steps of 5
f = np.arange(0, 30, 5)
print ("\nA sequential array with steps of 5:\n", f)
# Create a sequence of 10 values in range 0 to 5
g = np.linspace(0, 5, 10)
print ("\nA sequential array with 10 values between"
      "0 and 5:\n", g)
# Reshaping 3X4 array to 2X2X3 array
arr = np.array([[1, 2, 3, 4],
                [5, 2, 4, 2],
                [1, 2, 0, 1]])
newarr = arr.reshape(2, 2, 3)
print ("\nOriginal array:\n", arr)
print ("Reshaped array:\n", newarr)
# Flatten array
arr = np.array([[1, 2, 3], [4, 5, 6]])
flarr = arr.flatten()
print ("\nOriginal array:\n", arr)
print ("Fattened array:\n", flarr)
```

Output :

```
Array created using passed list:
 [[ 1.  2.  4.]
 [ 5.  8.  7.]]
Array created using passed tuple:
 [1 3 2]
An array initialized with all zeros:
 [[ 0.  0.  0.  0.]
 [ 0.  0.  0.  0.]
 [ 0.  0.  0.  0.]]
An array initialized with all 6s. Array type is complex:
 [[ 6.+0.j  6.+0.j  6.+0.j]
 [ 6.+0.j  6.+0.j  6.+0.j]
 [ 6.+0.j  6.+0.j  6.+0.j]]
A random array:
 [[ 0.46829566  0.67079389]
 [ 0.09079849  0.95410464]]
A sequential array with steps of 5:
 [ 0  5 10 15 20 25]
A sequential array with 10 values between 0 and 5:
      [0.0.55555556  1.11111111  1.66666667
2.22222222  2.77777778  3.33333333  3.88888889
4.44444444  5. ]
Original array:
 [[1 2 3 4]
 [5 2 4 2]
 [1 2 0 1]]
Reshaped array:
 [[[1 2 3]
  [4 5 2]]
 [[4 2 1]
  [2 0 1]]]
Original array:
 [[1 2 3]
 [4 5 6]]
Fattened array:
 [1 2 3 4 5 6]
```

13. Python program to demonstrate indexing in numpy

Ans :

```
# Python program to demonstrate
# indexing in numpy
import numpy as np
# An exemplar array
```

```
arr = np.array([[-1, 2, 0, 4],
                [4, -0.5, 6, 0],
                [2.6, 0, 7, 8],
                [3, -7, 4, 2.0]])
# Slicing array
temp = arr[:2, ::2]
print ("Array with first 2 rows and alternate"
                    "columns(0 and 2):\n", temp)
# Integer array indexing example
temp = arr[[0, 1, 2, 3], [3, 2, 1, 0]]
print ("\nElements at indices (0, 3), (1, 2), (2, 1),"
                                    "(3, 0):\n",
    temp)
# boolean array indexing example
cond = arr > 0 # cond is a boolean array
temp = arr[cond]
print ("\nElements greater than 0:\n", temp)
```

Output :

```
Array with first 2 rows and alternatecolumns(0 and 2):
 [[-1.  0.]
 [ 4.  6.]]
Elements at indices (0, 3), (1, 2), (2, 1),(3, 0):
 [ 4.  6.  0.  3.]
Elements greater than 0:
 [ 2.   4.   4.   6.   2.6  7.   8.   3.   4.   2. ]
```

14. Python program to demonstrate basic operations on single array.

Ans :

```
# Python program to demonstrate basic operations on single array
import numpy as np
a = np.array([1, 2, 5, 3])
# add 1 to every element
print ("Adding 1 to every element:", a+1)
# subtract 3 from each element
print ("Subtracting 3 from each element:", a-3)
# multiply each element by 10
print ("Multiplying each element by 10:", a*10)
# square each element
print ("Squaring each element:", a**2)
# modify existing array
a *= 2
print ("Doubled each element of original array:", a)
# transpose of array
a = np.array([[1, 2, 3], [3, 4, 5], [9, 6, 0]])
print ("\nOriginal array:\n", a)
print ("Transpose of array:\n", a.T)
```

Output :

```
Adding 1 to every element: [2 3 6 4]
Subtracting 3 from each element: [-2 -1  2  0]
Multiplying each element by 10: [10 20 50 30]
Squaring each element: [ 1  4 25  9]
Doubled each element of original array: [ 2  4 10  6]
Original array:
[[1 2 3]
[3 4 5]
[9 6 0]]
Transpose of array:
[[1 3 9]
[2 4 6]
[3 5 0]]
```

15. Python program to demonstrate unary operators in numpy

Ans :

```
# Python program to demonstrate unary operators in numpy
import numpy as np
arr = np.array([[1, 5, 6],
                [4, 7, 2],
                [3, 1, 9]])
# maximum element of array
print ("Largest element is:", arr.max())
print ("Row-wise maximum elements:",
                    arr.max(axis = 1))
```

```
# minimum element of array
print ("Column-wise minimum elements:",
                                      arr.min(axis = 0))
# sum of array elements
print ("Sum of all array elements:",
                                          arr.sum())
# cumulative sum along each row
print ("Cumulative sum along each row:\n",
                                      arr.cumsum(axis = 1))
```

Output :

```
      Largest element is: 9
      Row-wise maximum elements: [6 7 9]
      Column-wise minimum elements: [1 1 2]
      Sum of all array elements: 38
      Cumulative sum along each row:
[[ 1  6 12]
 [ 4 11 13]
 [ 3  4 13]]
```

Pandas

16. Python code demonstrate to make a Pandas DataFrame with two-dimensional list

Ans :

```
import pandas as pd
# List1
lst = [['tom', 'reacher', 25], ['krish', 'pete', 30],
        ['nick', 'wilson', 26], ['juli', 'williams', 22]]
df = pd.DataFrame(lst, columns =['FName', 'LName',
      'Age'],
            dtype = float)
print(df)
```

Output:

```
  FName    LName        Age
      0   tom   reacher     25
      1   krish  pete       30
      2   nick   wilson     26
      3  juli    williams   22
```

17. Python code demonstrate creating DataFrame from dictionary of narray and lists

Ans :

```
# Python code demonstrate creatingDataFrame from
      dict narray / lists
# By default addresses.
import pandas as pd
# initialise data of lists.
data = {'Category':['Array', 'Stack', 'Queue'],
          'Marks':[20, 21, 19]}
# Create DataFrame
df = pd.DataFrame(data)
# Print the output.
print(df )
```

Output:

```
Category  Marks
0   Array    20
1   Stack    21
2   Queue    19
```

18. Python code demonstrate creating a Pandas dataframe using list of tuples

Ans :

```
# importing the pandas package
import pandas as pd
# creating a list of tuples
list_of_tuples = [('A',65),('B',66),('C',67)]
# creating DataFrame
      df=pd.DataFrame(list_of_tuples, columns
      =['Char', 'Ord'])
# displaying resultant DataFrame
print(df)
```

Output

```
 Char  Ord
0  A   65
1  B   66
2  C   67
```

19. Python code demonstrate how to iterate over rows in Pandas Dataframe.

Ans :

```
# importing pandas
import pandas as pd
# list of dicts
input_df = [{'name':'Sujeet', 'age':10},
            {'name':'Sameer', 'age':11},
            {'name':'Sumit', 'age':12}]
df = pd.DataFrame(input_df)
print('Original DataFrame: \n', df)
print('\nRows iterated using iterrows() : ')
for index, row in df.iterrows():
    print(row['name'], row['age'])
```

Output:

```
Original DataFrame:
   age   name
0  10  Sujeet
1  11  Sameer
2  12   Sumit
Rows iterated using iterrows() :
Sujeet 10
Sameer 11
Sumit 12
```

20. Python code demonstrate how to get column names in Pandas dataframe

Ans :

```
# Import pandas package
import pandas as pd
# making data frame
data = pd.read_csv("nba.csv")
# iterating the columns
for col in data.columns:
    print(col)
```

Output:

```
Name
Team
Number
Position
Age
Height
Weight
College
Salary
```

FACULTY OF INFORMATICS

BCA II-Year IV-Semester (CBCS) Examination

Model Paper - I

DATA SCIENCE USING PYTHON

Time : 3 Hours] **[Max. Marks : 70**

Note : Answer all questions from Part - A, & any five questions from Part - B Choosing one questions from each unit.

PART - A (10 × 2 = 20 Marks)

			ANSWERS
1.	(a)	Python keywords	**(Unit-I, SQA-7)**
	(b)	What is Data Science?	**(Unit-I, SQA-1)**
	(c)	What is Pass Statement in Python?	**(Unit-II, SQA-10)**
	(d)	User-Defined Functions	**(Unit-II, SQA-4)**
	(e)	What is string	**(Unit-III, SQA-1)**
	(f)	How to create a list ?	**(Unit-III, SQA-6)**
	(g)	What is fancy indexing in Numpy? Explain.	**(Unit-IV, SQA-6)**
	(h)	Structured Data	**(Unit-IV, SQA-8)**
	(i)	Methods in indexing in Data frame.	**(Unit-V, SQA-6)**
	(j)	What are panda objects?	**(Unit-V, SQA-1)**

PART - B (5 × 10 = 50 Marks)

UNIT - I

2.	(a)	What are the main components of Data science?	**(Unit-I, Q.No. 2)**
	(b)	What are the features of python?	**(Unit-I, Q.No. 14)**

OR

3.	(a)	What are Python keywords? Explain.	**(Unit-I, Q.No. 21)**
	(b)	What are the application sof data science? Explain.	**(Unit-I, Q.No. 7)**

UNIT - II

4.	(a)	Explain formatted print function.	**(Unit-II, Q.No. 2)**
	(b)	What is function? How to define and call a function?	**(Unit-II, Q.No. 11)**

OR

5. (a) What are the various types of quotations used in python? **(Unit-II, Q.No. 5)**

   (b) Explain the Flow of Execution in Python. **(Unit-II, Q.No. 14)**

UNIT - III

6. (a) Explain various String Manipulation Functions. **(Unit-III, Q.No. 6)**

   (b) Explain various array methods in Python. **(Unit-III, Q.No. 9)**

OR

7. (a) Write about various methods used in lists. **(Unit-III, Q.No. 13)**

   (b) What is list Cloning? Explain list cloning techniques. **(Unit-III, Q.No. 17)**

UNIT - IV

8. What is NumPy? Explain how to create arrays in python using Numpy. **(Unit-IV, Q.No. 1)**

OR

9. Explain the comparisons operations in Numy with examples. **(Unit-IV, Q.No. 7)**

UNIT - V

10. Explain, how to create and use series in Pandas. **(Unit-V, Q.No. 2)**

OR

11. Explain data aggregation functions in Pandas. **(Unit-V, Q.No. 12)**

FACULTY OF INFORMATICS

BCA II-Year IV-Semester (CBCS) Examination

Model Paper - II

DATA SCIENCE USING PYTHON

Time : 3 Hours] **[Max. Marks : 70**

Note : Answer all questions from Part - A, & any five questions from Part - B Choosing one questions from each unit.

PART - A (10 × 2 = 20 Marks)

ANSWERS

1. (a) What is Data Science? **(Unit-I, SQA-6)**

(b) Python variables **(Unit-I, SQA-9)**

(c) Advantages of user-defined functions **(Unit-II, SQA-5)**

(d) Explain nested if statements with syntax and example. **(Unit-II, SQA-9)**

(e) Explain various String Manipulation Functions. **(Unit-III, SQA-4)**

(f) Define array. **(Unit-III, SQA-5)**

(g) What is NumPy? **(Unit-IV, SQA-1)**

(h) Sorting Arrays **(Unit-IV, SQA-7)**

(i) Frames in Pandas. **(Unit-V, SQA-3)**

(j) Hierarchical indexing in Pandas. **(Unit-V, SQA-7)**

PART - B (5 × 10 = 50 Marks)

UNIT - I

2. (a) Explain various processes of data science, what are used to extract information. **(Unit-I, Q.No. 3)**

(b) Explain about various tools used for data analysis. **(Unit-I, Q.No. 10)**

OR

3. (a) Write about statements in python. **(Unit-I, Q.No. 23)**

(b) What is data analysis? **(Unit-I, Q.No. 9)**

UNIT - II

4. (a) Write about Tuple assignment feature? **(Unit-II, Q.No. 7)**

(b) Write a program to check whether the given number is prime or not. **(Unit-II, Q.No. 21)**

OR

5. (a) What is indentation in python? **(Unit-II, Q.No. 4)**

   (b) Write a program to add two numbers. **(Unit-II, Q.No. 10)**

UNIT - III

6. (a) Explain briefly about Python String Module. **(Unit-III, Q.No. 7)**

   (b) What are lists ? How to create a list ? **(Unit-III, Q.No. 10)**

OR

7. (a) Explain about Tuple assignment. **(Unit-III, Q.No. 21)**

   (b) How to slice lists in Python? **(Unit-III, Q.No. 12)**

UNIT - IV

8. Explain array creation techniques in Numpy with an example program **(Unit-IV, Q.No. 2)**

OR

9. Explain Boolean Arrays in Numpy. **(Unit-IV, Q.No. 9)**

UNIT - V

10. Explain, how to handle the missing data. **(Unit-V, Q.No. 8)**

OR

11. Expalin , how to use frames in Pandas. **(Unit-V, Q.No. 4)**

FACULTY OF INFORMATICS

BCA II-Year IV-Semester (CBCS) Examination

Model Paper - III

DATA SCIENCE USING PYTHON

Time : 3 Hours] [Max. Marks : 70

Note : Answer all questions from Part - A, & any five questions from Part - B Choosing one questions from each unit.

PART - A (10 × 2 = 20 Marks)

ANSWERS

1. (a) Python variables **(Unit-I, SQA-3)**

   (b) Identifiers in python **(Unit-I, SQA-8)**

   (c) Types of Functions **(Unit-II, SQA-3)**

   (d) Flow of Execution in Python. **(Unit-II, SQA-6)**

   (e) What is list slicing? **(Unit-III, SQA-7)**

   (f) Advantages of Tuple over List. **(Unit-III, SQA-10)**

   (g) Aggregations in Numpy **(Unit-IV, SQA-3)**

   (h) Boolean Arrays **(Unit-IV, SQA-5)**

   (i) Write various functions used for series attribute in Pandas. **(Unit-V, SQA-4)**

   (j) Parameters of Groupby. **(Unit-V, SQA-10)**

PART - B (5 × 10 = 50 Marks)

UNIT - I

2. (a) Write the differences between data science with business intelligence. **(Unit-I, Q.No. 6)**

   (b) What are the features of python? **(Unit-I, Q.No. 14)**

OR

3. (a) Explain various modes of Python Interpreter. **(Unit-I, Q.No. 16)**

   (b) What is Data Science? Explain the steps involved in data science processing. **(Unit-I, Q.No. 1)**

UNIT - II

4. (a) What are identifiers in python? **(Unit-II, Q.No. 6)**

   (b) Write a program to calculate a running total in python. **(Unit-II, Q.No. 26)**

OR

5. (a) Write about python expressions. **(Unit-II, Q.No. 1)**

   (b) What are the various types of operators used in python. **(Unit-II, Q.No. 8)**

UNIT - III

6. (a) Define array? Explain about array operations. **(Unit-III, Q.No. 8)**

   (b) How to access elements from a list? **(Unit-III, Q.No. 11)**

OR

7. (a) Explain Aliasing in lists. **(Unit-III, Q.No. 16)**

   (b) What are dictionaries in python? Explain the operations that can be peformed on dictionaries. **(Unit-III, Q.No. 24)**

UNIT - IV

8. Expalin Array Indexing with an example program. **(Unit-IV, Q.No. 3)**

OR

9. Explain about Sorting Arrays in Numpy. **(Unit-IV, Q.No. 11)**

UNIT - V

10. Explain, how to Combine the data frames in Panda Using Merge() Function. **(Unit-V, Q.No. 10)**

OR

11. Explain various operations that can perform on Pandas Data Frames. **(Unit-V, Q.No. 7)**

FACULTY OF INFORMATICS

BCA IV-Semester (CBCS) Examination

February-2023

DATA SCIENCE USING PYTHON

Time : 3 Hours] **[Max. Marks : 70**

Note : I. Answer all questions from Part - A, & answer any five questions from Part - B Choosing one questions from each unit.

II. Missing data, if any, may be suitably assumed.

PART - A (10 × 2 = 20 Marks)

1. (a) Write about challenges of Data science technology.
   (b) What are variables, keywords and Identifiers in python?
   (c) Explain about Types of Functions.
   (d) Write about Return values and variable Scope.
   (e) Explain about Methods of an Array.
   (f) Difference between tuple and Dictionary.
   (g) Briefly write about computation on Numpy Array.
   (h) Write about Fancy indexing sorting array.
   (i) Explain about combining dataset and Grouping.
   (j) Write about Selection operations on Data in Panda.

PART - B (5 × 10 = 50 Marks)

UNIT - I

2. (a) Write about DataScience Job Roles and Tools for Data Science.
   (b) What are the Difference between Data science and BI?

OR

3. (a) What is Data Analysis ? Explain about Data Analysis Tools.
   (b) What is Python? Write about the Features of Python.

UNIT - II

4. (a) Explain the uses and types of Python functions.
   (b) Write about Expressions, I/O, and Tuple Assignment.

OR

5. (a) Write about Flow of Execution, Parameters and Arguments.
   (b) Explain about Return Values, Variable Scope (Local, Global).

UNIT - III

6. (a) Write about Advanced List Processing-List Comprehension and Nested List

   (b) Write about Mutability, aliasing, Cloning List, List Parameters.

OR

7. (a) What is String ? Explain about String Functions and Methods.

   (b) Write about List of an Array and Methods of an Array.

UNIT - IV

8. (a) Write about Computation on numpy arrays, Aggregations and computations on arrays.

   (b) Explain about Sorting and Structured Data.

OR

9. (a) What is Numpy? Explain about basic Numpy array and Boolean logic indexing.

   (b) Write about Comparisons, Fancy Indexing.

UNIT - V

10. (a) Write about pandas object.

    (b) Explain about Data Indexing and Selection.

OR

11. (a) Write about Aggrigation and Grouping of Pandas.

    (b) Explain about Handling Missing data and Combining' data set.