

**VPMP Polytechnic, Gandhinagar**  
**Department of Computer Engineering**  
**IMP Questions with answer**

Subject: Scripting Language-Python (4330701)

Semester: 3<sup>rd</sup>

**1. List and explain features of Python.**

- ✓ **Python is object-oriented:** Structure supports such concepts as polymorphism, operation overloading and multiple inheritance.
- ✓ **Indentation:** Indentation is one of the greatest feature in python
- ✓ **It is free (open source):** Downloading python and installing python is free and easy.
- ✓ **It is Powerful:** It provides Dynamic typing, Built-in types and tools, Library utilities, and Automatic memory management
- ✓ **It is Portable:** Python runs virtually every major platform used today
- ✓ **It is easy to use and learn:** It has no intermediate compile. Python Programs are compiled automatically to an intermediate form called byte code, which the interpreter then reads.
- ✓ **Interpreted Language:** Python is processed at runtime by python Interpreter
- ✓ **Interactive Programming Language:** Users can interact with the python interpreter directly for writing the programs
- ✓ **Straight forward syntax:** The formation of python syntax is simple and straight forward which also makes it popular.

**2. Write Application of python.**

- ✓ **Web Applications:** We can use Python to develop web applications. Python provides many useful frameworks, and these are given below:
  - Django and Pyramid framework (Use for heavy applications)
  - Flask and Bottle (Micro-framework)
- ✓ **Desktop GUI Applications:** The GUI stands for the Graphical User Interface, which provides a smooth interaction to any application. Python provides a Tk GUI library to develop a user interface.
- ✓ **Console-based Application:** Console-based applications run from the command-line or shell. These applications are computer program which are used commands to execute.
- ✓ **Software Development:** Python is useful for the software development process. It works as a support language and can be used to build control and management, testing, etc.
- ✓ **Scientific and Numeric:** Python language is the most suitable language for Artificial intelligence or machine learning. It consists of many scientific and mathematical libraries, which makes easy to solve complex calculations.
- ✓ **Business Applications:** E-commerce and ERP are an example of a business application. This kind of application requires extensively, scalability and readability, and Python provides all these features.
- ✓ **Audio or Video-based Applications :** Python is flexible to perform multiple tasks and can be used to create multimedia applications.
- ✓ **Enterprise Applications:** Python can be used to create applications that can be used within an Enterprise or an Organization. Some real-time applications are OpenERP, Tryton, Picalo, etc.

✓ **Image Processing Application:** Python contains many libraries that are used to work with the image. The image can be manipulated according to our requirements. Examples of image processing libraries are: OpenCV, Pillow

### 3. Explain basic structure of Python Program.

Import statements // import statements are used to include library files to the python program
Function definitions //This section include the definitions of various functions written in a Python Program
Program statements // This section include the set of statements for solving the given problem.

There are two modes for using the Python interpreter:

- ✓ Interactive Mode
- ✓ Script Mode

#### Running Python in interactive mode:

✓ Without passing python script file to the interpreter, directly execute code to Python prompt.

✓ Once you're inside the python interpreter, then you can start.

✓ `>>> print("hello world")`

Output: hello world

#### Running Python in script mode:

✓ Alternatively, programmers can store Python script source code in a file with the .py extension, and use the interpreter to execute the contents of the file.

✓ To execute the script by the interpreter, you have to tell the interpreter the name of the file.

### 4. Explain keyword, identifier and variable.

#### Identifiers

✓ Identifier is a name given to various programming elements such as a variable, function, class, module or any other object.

✓ rules to create an identifier.

1. The allowed characters are a-z, A-Z, 0-9 and underscore (\_)
2. It should begin with an alphabet or underscore
3. It should not be a keyword
4. It is case sensitive
5. No blank spaces are allowed.
6. It can be of any size

✓ Valid identifiers examples : si, rate\_of\_interest, student1, ageStudent

✓ Invalid identifier examples : rate of interest, 1student, @age

**Keywords:**Keywords are the identifiers which have a specific meaning in python, there are 33 keywords in python. These may vary from version to version

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

❖ **Variables** : When we create a program, we often need store values so that it can be used in a program. We use variables to store data which can be manipulated by the computer program.

- ✓ Every variable has a name and memory location where it is stored.
  - ✓ It Can be of any size
  - ✓ Variable name Has allowed characters, which are a-z, A-Z, 0-9 and underscore (\_)
  - ✓ Variable name should begin with an alphabet or underscore
  - ✓ Variable name should not be a keyword
  - ✓ Variable name should be meaningful and short
  - ✓ Generally, they are written in lower case letters
  - ✓ A type or datatype which specify the nature of variable (what type of values can be stored in
  - ✓ We can check the type of the variable by using type command
- ```
>>> type(variable_name)
```

### 5. Explain datatype with example.

**Number:** Number data type stores Numerical Values. These are of three different types:

- a) Integer & Long
- b) Float/floating point
- c) Complex

a) Integers are the whole numbers consisting of + or – sign like 100000, -99, 0, 17.  
e.g. age=19  
salary=20000

b) Floating Point: Numbers with fractions or decimal point are called floating point numbers. A floating point number will consist of sign (+,-) and a decimal sign(.)  
e.g. temperature= -21.9,growth\_rate= 0.98333328

c) Complex: Complex number is made up of two floating point values, one each for real and imaginary part. For accessing different parts of a variable x we will use x.real and x.imag. Imaginary part of the number is represented by j instead of i, so 1+0j denotes zero imaginary part.

Example

```
>>> x = 1+0j
>>> print x.real,x.imag
1.0      0.0
```

**Boolean:** Objects of Boolean type may have one of two values, True or False:

```
>>> type(True)
<class 'bool'>
>>> type(False)
<class 'bool'>
```

**Sequence:** A sequence is an ordered collection of items, indexed by positive integers. Three types of sequence data type available in Python are Strings, Lists & Tuples.

a)String: is an ordered sequence of letters/characters. They are enclosed in single quotes (') or double (").

Example

```
>>> a = 'Ami Patel'
>>>a="Ami Patel"
```

b) Lists: List is also a sequence of values of any type. List is enclosed in square brackets.

Example

```
Student = ["Neha", 567, "CS"]
```

c) Tuples: Tuples are a sequence of values of any type, and are indexed by integers. Tuples are enclosed in ().

```
Student = ("Neha", 567, "CS")
```

### Sets

Set is an unordered collection of values, of any type, with no duplicate entry. Sets are immutable.

Example

```
s = set ([1,2,3,4])
```

**Dictionaries:** Dictionaries store a key – value pairs, which are accessed using key. Dictionary is enclosed in curly brackets.

Example

```
d = { 1:'a', 2:'b', 3:'c' }
```

## 6. Explain input and print method of python.

**Print method:**A statement is print with the help of print() method.

syntax : print("message to be printed") Or print('message to be printed') or print(variable\_name)

Example: x=10

```
print(x)
```

Output: 10

**Syntax:** print("message to be printed", variable\_name)

Example:

```
x=10
```

```
print("value of x",x)
```

Output: value of x 10

**Syntax:** print('message to be printed', variable\_name)

Example:

```
x=10
```

```
print('value of x',x)
```

Output: value of x 10

**Input method:**A value can be input from the user with the help of input() method. input method return a string. It can be changed to other datatypes by using type.

Syntax: variable\_name = input(“Enter any number”)

Example:

```
name = input(“Enter your name”)
age= int(input(“Enter you age”))
```

## 7. List out operator and explain arithmetic,assignment and bitwise operator.

Types of operators:

- ✓ Arithmetic Operator
- ✓ Relational Operator
- ✓ Logical Operator
- ✓ Bitwise Operator
- ✓ Assignment Operator
- ✓ Membership Operator

### Arithmetic Operator

| Symbol | Description          | Example 1                                             | Example 2                             |
|--------|----------------------|-------------------------------------------------------|---------------------------------------|
| +      | Addition             | >>>55+45<br>100                                       | >>> ‘Good’ + ‘Morning’<br>GoodMorning |
| -      | Subtraction          | >>>55-45<br>10                                        | >>>30-80<br>-50                       |
| *      | Multiplication       | >>>55*45<br>2475                                      | >>> ‘Good’* 3<br>GoodGoodGood         |
| /      | Division             | >>>17/5<br>3<br>>>>17/5.0<br>3.4<br>>>> 17.0/5<br>3.4 | >>>28/3<br>9                          |
| %      | Remainder/<br>Modulo | >>>17%5<br>2                                          | >>> 23%2<br>1                         |
| **     | Exponentiation       | >>>2**3<br>8<br>>>>16**.5<br>4.0                      | >>>2**8<br>256                        |
| //     | Integer<br>Division  | >>>7.0//2<br>3.0                                      | >>>3// 2<br>1                         |

**Bitwise Operator:**

| OPERATOR | DESCRIPTION         | SYNTAX |
|----------|---------------------|--------|
| &        | Bitwise AND         | x & y  |
|          | Bitwise OR          | x   y  |
| ~        | Bitwise NOT         | ~x     |
| ^        | Bitwise XOR         | x ^ y  |
| >>       | Bitwise right shift | x>>    |
| <<       | Bitwise left shift  | x<<    |

**Assignment Operator**

|   |                                                           |                              |  |
|---|-----------------------------------------------------------|------------------------------|--|
| = | Assigned values from right side operands to left variable | >>>x=12*<br>>>>y='greetings' |  |
|---|-----------------------------------------------------------|------------------------------|--|

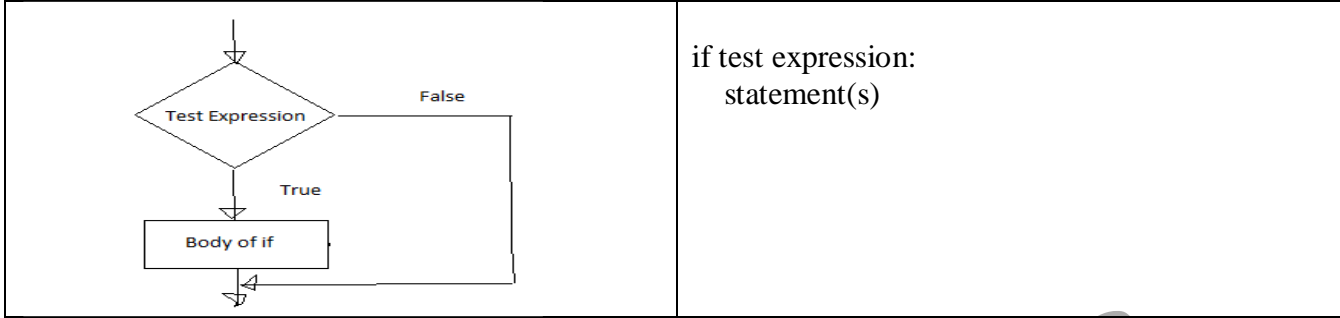
(\*we will use it as initial value of x for following examples)

|     |                                                                                             |          |                                                                                                  |
|-----|---------------------------------------------------------------------------------------------|----------|--------------------------------------------------------------------------------------------------|
| +=  | added and assign back the result to left operand                                            | >>>x+=2  | The operand/ expression/ constant written on RHS of operator is will change the value of x to 14 |
| -=  | subtracted and assign back the result to left operand                                       | x-=2     | x will become 10                                                                                 |
| *=  | multiplied and assign back the result to left operand                                       | x*=2     | x will become 24                                                                                 |
| /=  | divided and assign back the result to left operand                                          | x/=2     | x will become 6                                                                                  |
| %=  | taken modulus using two operands and assign the result to left operand                      | x%=2     | x will become 0                                                                                  |
| **= | performed exponential (power) calculation on operators and assign value to the left operand | x**=2    | x will become 144                                                                                |
| //= | performed floor division on operators and assign value to the left operand                  | x // = 2 | x will become 6                                                                                  |

**8. Explain python selection statements with flowchart, syntax and example.**

❖ **if Statement** : statement execute only if the result of condition is true .

|           |        |
|-----------|--------|
| Flowchart | Syntax |
|-----------|--------|



- ✓ The program checks the test expression. If the test expression is True, then statement(s) will be executed.
- ✓ If the test expression is False, the statement(s) is not executed.
- ✓ In Python, the body of the if statement is indicated by the indentation. The body starts with an indentation and the first unindented line marks the end.

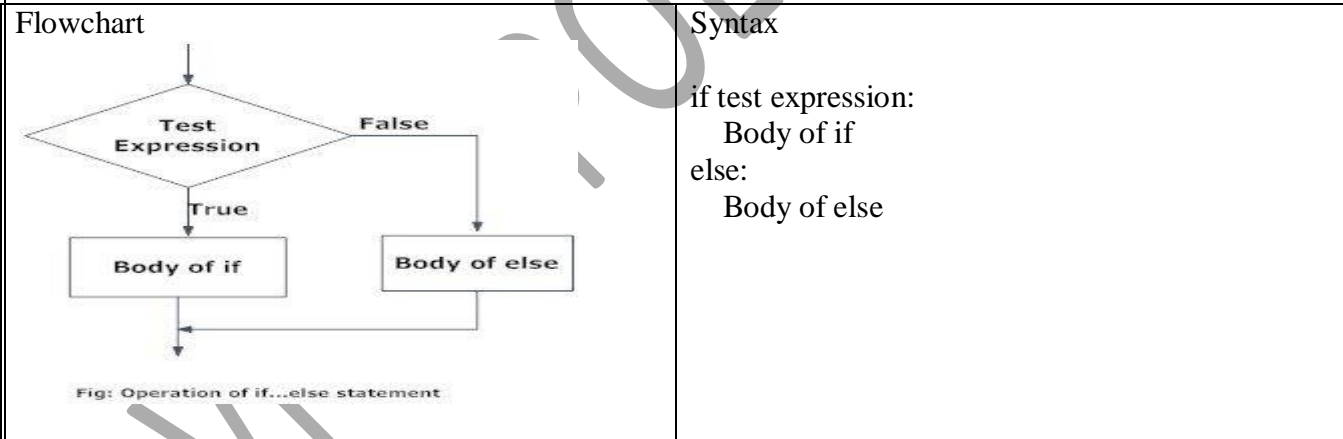
Example:

**Example: Python if Statement**

```
num = 3
if num > 0:
    print("It is a positive number.")
```

Output:  
It is a positive number.

❖ **if else Statement** :When we have to select only one option from the given two option, then we use if ...else.



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

**Example: Python if...else Statement**

```
a = 7
b = 0
if (a > b):
    print("a is greater than b")
else:
```

```
print("b is greater than a")
```

**Output:**

a is greater than b

- ❖ **Nested if-else :** Nested “if-else” statements mean that an “if” statement or “if-else” statement is present inside another if or if-else block.

**Syntax**

```
if ( test condition 1):  
    if ( test condition 2):  
        Statement1  
    else:  
        Statement2  
else:  
    Statement 3
```

- ✓ First test condition1 is checked, if it is true then check condition2.
- ✓ If test condition 2 is True, then statement1 is executed.
- ✓ If test condition 2 is false, then statement2 is executed.
- ✓ If test condition 1 is false, then statement3 is executed

**Example: Python Nested if...else Statement**

```
a=int(input("Enter A: "))  
b=int(input("Enter B: "))  
c=int(input("Enter C: "))  
if a>b:  
    if a>c:  
        max=a  
    else:  
        max=c  
else:  
    if b>c:  
        max=b  
    else:  
        max=c  
  
print("Maximum =",max)
```

**Output:**

```
Enter A: 23  
Enter B: 2  
Enter C: 56  
Maximum = 56
```



**if-elif-else statements**

Flowchart

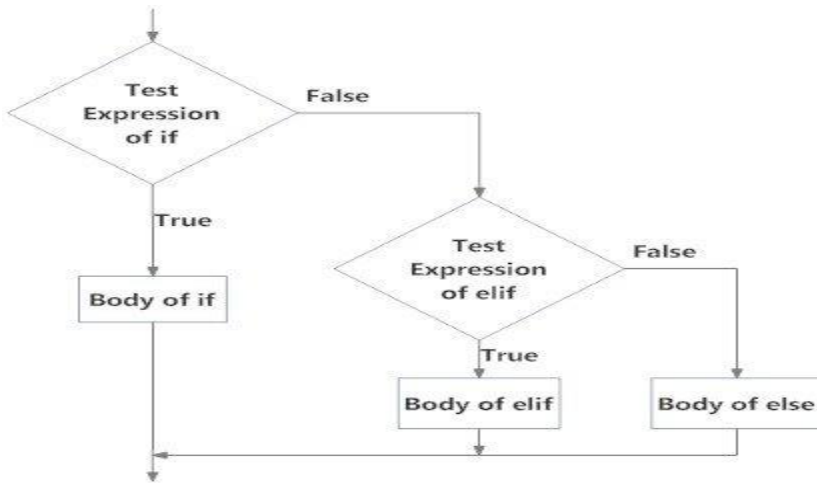


Fig: Operation of if...elif...else statement

Syntax

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

- ✓ The elif is short for else if.
- ✓ It allows us to check for multiple expressions.
- ✓ If the condition for if is False, it checks the condition of the next elif block and so on.
- ✓ If all the conditions are False, the body of else is executed.
- ✓ Only one block among the several if...elif...else blocks is executed according to the condition.

**Example: Python if-elif-else Statement**

```

num = -3
if num > 0:
    print("Positive number")
elif num == 0:
    print("Zero")
else:
    print("Negative number")
  
```

Output:  
 Negative number

```

a,b,c=12,45,3
if a>b and a>c:
    print("a is max")
elif b>a and b>c:
    print("b is max")
else:
    print("c is max")
  
```

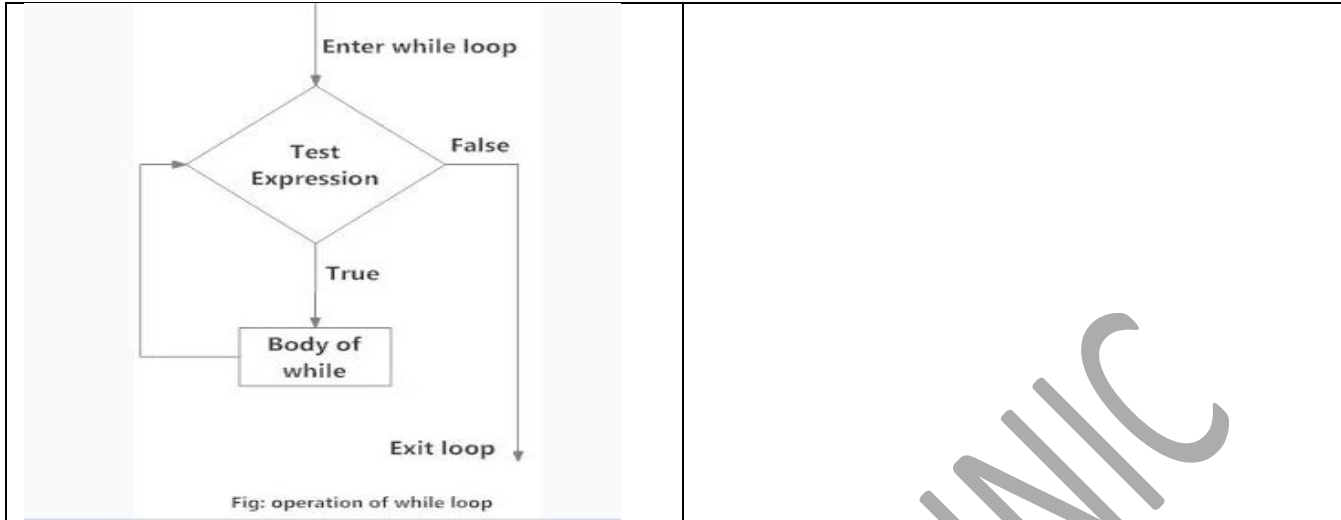
Output:  
 b is max

**9. Explain while loop with flowchart, syntax and example.**

Flowchart

Syntax

while test\_expression:  
 Body of while



- ✓ Loops are either infinite or conditional.
- ✓ In the 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.
- ✓ The statements that are executed inside while can be a single line of code or a block of multiple statements

**Example:**

```

x=1
while(x<=5):
    print(x)
    x+=1
    
```

Output:

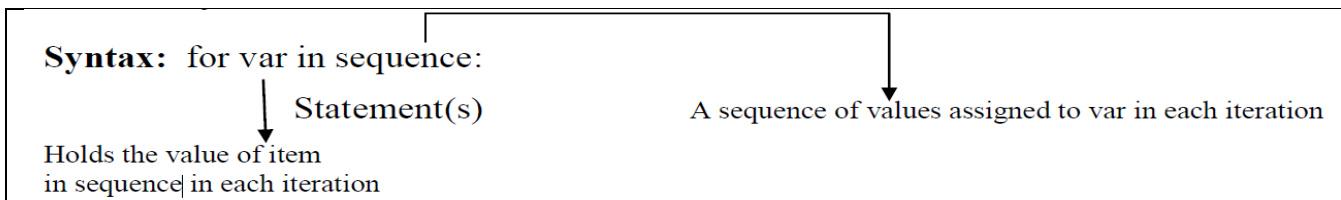
```

1
2
3
4
5
    
```

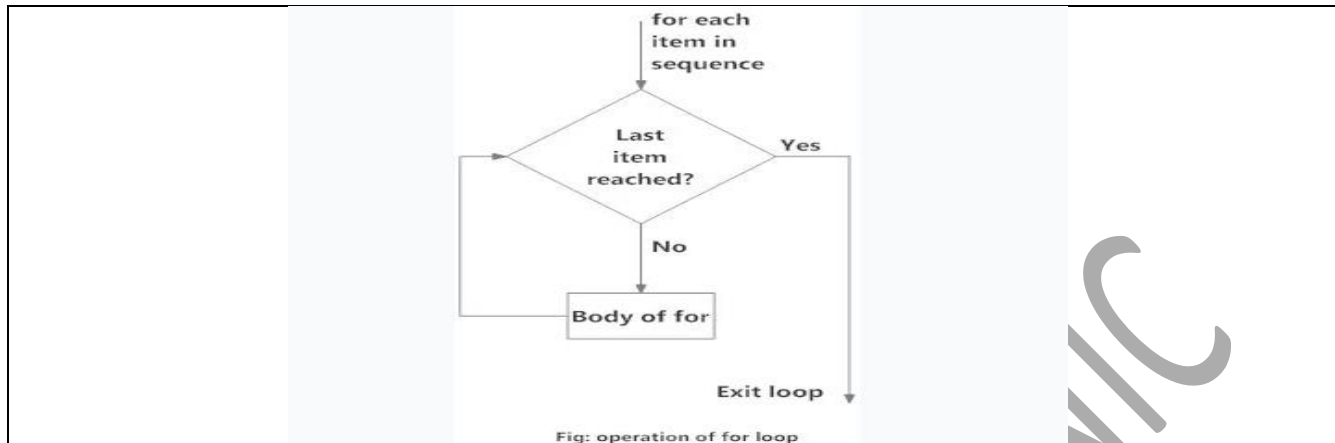
**10. Explain for loop and nested for loop with flowchart, syntax and example.**

**For loop:**

- ✓ Python for loop is used for repeated execution of a group of statements for the desired number of times.
- ✓ It iterates over the items of lists, tuples, strings, the dictionaries and other iterable objects



**Flowchart:**



**Example:**

|                                                                                              |                                                                                                   |
|----------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------|
| <pre>for i in range(1,6):     print(i)</pre> <p>Output:</p> <p>1<br/>2<br/>3<br/>4<br/>5</p> | <pre>a=[10,12,13,14] for i in a:     print(i)</pre> <p>Output:</p> <p>10<br/>12<br/>13<br/>14</p> |
|----------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------|

**Nested for loop:** Nested loops mean loops inside a loop.

For example, while loop inside the for loop, for loop inside the for loop, etc.

|                                                                                                                                            |                                                                                                                                          |
|--------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------|
| <p>Syntax:</p> <pre>for var in sequence:     for var in sequence:         statements(s)     statements(s)</pre>                            | <p>Syntax:</p> <pre>while expression:     while expression:         statement(s)     statement(s)</pre>                                  |
| <p>Example :1</p> <pre>rows = 5 for i in range(1, rows + 1):     for j in range(1, i + 1):         print('*',end=' ')     print(' ')</pre> | <p>Example :2</p> <pre>rows = 5 for i in range(1, rows + 1):     for j in range(1, i + 1):         print(j,end=' ')     print(' ')</pre> |

Output:

```
*
* *
* * *
* * * *
* * * * *
```

Output:

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

**11. Explain switch case statement with flowchart, syntax and example.**

- ✓ Python Switch case is a selection control statement.
- ✓ The switch expression evaluated once.
- ✓ The value of the expression is compared with the value of each case.
- ✓ If there is a match, the associated block of code is executed.
- ✓ Python does not have its inbuilt switch case statement.
- ✓ We have to implement using different methods.
  - Using Function and Lamda Function
  - Using Dictionary Mapping
  - Using Python classes
  - Using if-elif-else

**Using Dictionary Mapping:**

```
def monday():
    return "monday"
def tuesday():
    return "tuesday"
def wednesday():
    return "wednesday"
def thursday():
    return "thursday"
def friday():
    return "friday"
def saturday():
    return "saturday"
def sunday():
    return "sunday"
def default():
    return "Incorrect day"
```

```
Day = {
    1: monday,
    2: tuesday,
    3: wednesday,
    4: thursday,
    5: friday,
    6: saturday,
    7: sunday
}
```

```
def DayName(dayofWeek):
    return Day.get(dayofWeek,default)( )

print(DayName (3))
print(DayName (8))
```

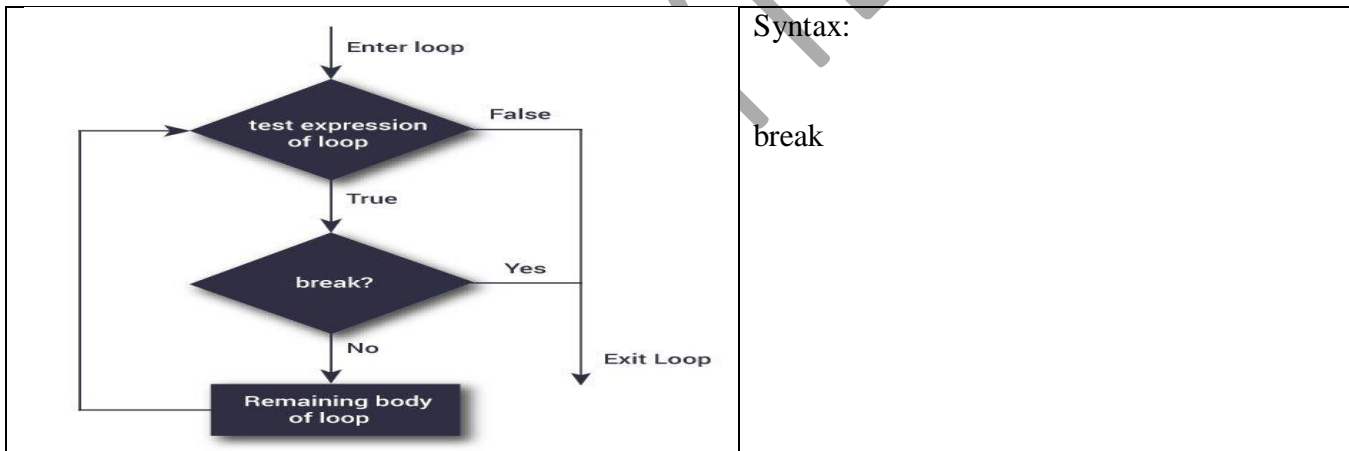
Output:  
Wednesday  
Incorrect day

- ✓ First create different functions which return Day name.
- ✓ We create a dictionary in which stored different key-value pair.
- ✓ After that create main function which takes user input for finding day name.
- ✓ At last, called the function to print the desired output

## 12. Explain break, continue and pass statement with flowchart, syntax and example.

### ❖ 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.
- ✓ The break statement can be used in both while and for loops.



Syntax:

break

### Example:

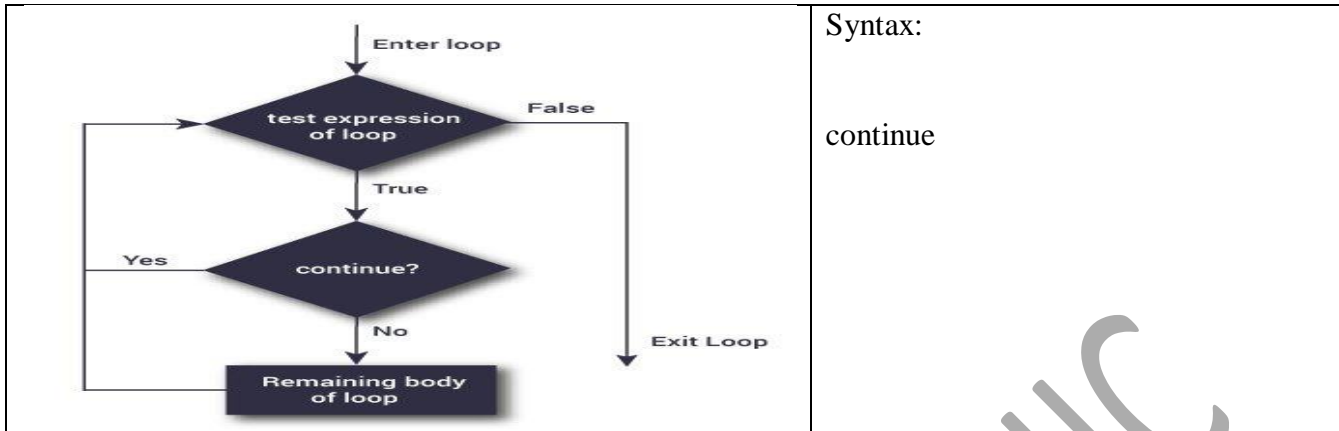
```
for i in range(1,11):
    print(i)
    if i == 2:
        break
```

Output:

1  
2

### ❖ continue statement

- ✓ The continue statement in Python returns the control to the beginning of the while loop.
- ✓ The continue statement can be used in both while and for loops.

**Example:**

```

n=int(input("enter n"))
for i in range(1,n):
    if(i==5):
        continue
    print(i)
  
```

Output:

```

enter n10
1 2 3 4 6 7 8 9
  
```

❖ **Pass Statement**

- ✓ the pass statement is a null statement.
- ✓ Nothing happens when the pass is executed.
- ✓ It results in no operation (NOP).
- ✓ When the user **does not know** what code to write, so user simply places **pass** at that line. Sometimes, **pass** is used when the user **doesn't want any code to execute**.
- ✓ Syntax:
 

```

pass
      
```
- ✓ Example
 

```

a= {'h', 'e', 'l', 'l', 'o'}
for val in a:
    pass
      
```

**13. Define List. Write Characteristics of list.****List:** It is used to store the sequence of various types of data.**Characteristics of List:**

- ✓ List is **ordered**.
- ✓ It allows **duplicate** members.
- ✓ It is a general purpose, most widely used in data structures.
- ✓ It is **mutable** type(We can **modify** its element after it is created)
- ✓ Elements of list can access by index.
- ✓ To use a list, you must declare it first. It is declare using square brackets and separate values with commas.

**14. How to create a list. Explain append(),extend(),insert() and delete operations on list.**

**create a list** :In Python, a list is created by placing elements inside square brackets [], separated by commas.

**Example:**

```
x = [1, 2, 3]
print(x)
```

Output:

```
[1, 2, 3]
```

**Append() and extend() :**

- ✓ We can add one item to a list using the append() method or add several items using the extend() method.

```
x = [1, 3, 5]
```

```
x.append(7)
print(x)
```

Output: [1, 3, 5, 7]

```
x.extend([9, 11, 13])
print(x)
```

Output: [1, 3, 5, 7, 9, 11, 13]

**Insert() :**

- ✓ We can insert one item at a desired location by using the method insert() or insert multiple items by squeezing it into an empty slice of a list.

```
x = [1, 9]
x.insert(1,3)
```

```
print(x)
```

Output: [1, 3, 9]

```
x[2:2] = [5, 7]
```

```
print(x)
```

Output: [1, 3, 5, 7, 9]

**Delete**

- ✓ We can delete one or more items from a list using the Python del statement. It can even delete the list entirely.

```
x = [10,20,30,40,50]
```

```
del x[2]
print(x)
```

Output:[19,20,40,50]

```
del x
```

# delete the entire list

```
print(x)
```

# Error: List not defined

**15. Write Characteristics of Tuple. Explain indexing and slicing in tuple.**

**Tuple:** A tuple in Python is similar to a list.

**Characteristics of Tuple:**

- ✓ It is **ordered**.
- ✓ It allows **duplicate** members.
- ✓ It is **immutable** type(We cannot **modify** its element after it is created)

- ✓ Elements of Tuple can access by index.
- ✓ To use a tuple, you must declare it first. It is declare using ( ) brackets and separate values with commas.

**Access tuple items(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 indices from 0 to 5.

Example:

```
>>> x=(10,20,30)
>>> print(x[2])
```

Output: 30

**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.

```
x = (10,20,30,40)
print(x[-1])
print(x[-3])
```

Output:  
40  
20

**Slicing**

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

```
x=(2,4,6,8)
print(x[:])      #Out put: (2, 4, 6, 8)
print(x[1:])     # Out put: (4, 6, 8)
print(x[:2])     #Out put: (2, 4)
print(x[1:3])    #Out put: (4, 6)
```

**16. Write Characteristics of Set. Explain add(),update(),remove() and discard() method of set.**

**A set is an unordered collection of items.**

**Characteristics of Set:**

- ✓ Sets are unordered.
- ✓ Set element is unique. Duplicate elements are not allowed.
- ✓ Set are immutable.(Can not change value)
- ✓ There is no index in set. So they donot support indexing or slicing operator.
- ✓ The set are used for mathematical operation like union,intersection,difference etc.

**Modifying a set in Python (add( ) and update())**

- ✓ We can add a single element using the add() method, and multiple elements using the update() method.

|                        |                   |
|------------------------|-------------------|
| x = {1, 3}<br>print(x) | Output:<br>{1, 3} |
|------------------------|-------------------|



|                                 |              |
|---------------------------------|--------------|
| x.add(2)<br>print(x)            | {1, 2, 3}    |
| x.update([2, 3, 4])<br>print(x) | {1, 2, 3, 4} |

**Removing elements from a set (remove() and discard())**

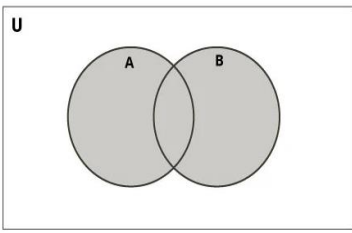
- ✓ A particular item can be removed from a set using the methods discard() and remove().
- ✓ The only difference between the two is that **the discard() function leaves a set unchanged if the element is not present in the set.** On the other hand, the **remove() function will raise an error** in such a condition (if element is not present in the set).

|                                 |                                                                                            |
|---------------------------------|--------------------------------------------------------------------------------------------|
| x = {1, 3, 4, 5, 6}<br>print(x) | Output:<br>{1, 3, 4, 5, 6}                                                                 |
| x.discard(4)<br>print(x)        | {1, 3, 5, 6}                                                                               |
| x.remove(6)<br>print(x)         | {1, 3, 5}                                                                                  |
| x.discard(2)<br>print(x)        | {1, 3, 5}                                                                                  |
| x.remove(2)                     | Traceback (most recent call last):<br>File "<string>", line 28, in <module><br>KeyError: 2 |

**17. Explain mathematical operations on Set(Union,Intersection,difference and symmetric difference)**

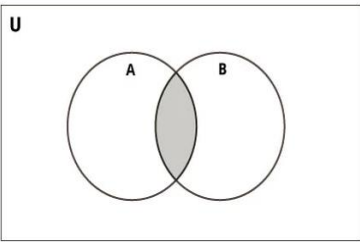
Sets can be used to carry out mathematical set operations like union, intersection, difference and symmetric difference.

**Set Union: Union of A and B is a set of all elements from both sets.** Union is performed using | operator. Same can be accomplished using the union() method.

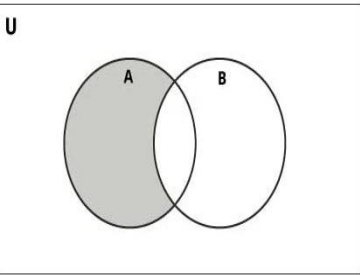
|                                                                                     |                                                            |                                                                                      |
|-------------------------------------------------------------------------------------|------------------------------------------------------------|--------------------------------------------------------------------------------------|
|  | A = {1, 2, 3, 4, 5}<br>B = {4, 5, 6, 7, 8}<br>print(A   B) | A = {1, 2, 3, 4, 5}<br>B = {4, 5, 6, 7, 8}<br>print(A.union(B))<br>print(B.union(A)) |
|                                                                                     | Output:<br>{1, 2, 3, 4, 5, 6, 7, 8}                        | Output:<br>{1, 2, 3, 4, 5, 6, 7, 8}<br>{1, 2, 3, 4, 5, 6, 7, 8}                      |

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

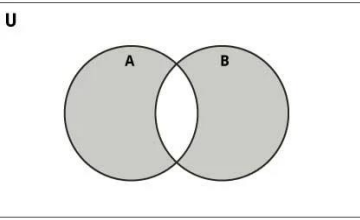
Intersection is performed using & operator. Same can be accomplished using the intersection() method.

|                                                                                   |                                                                                           |                                                                                                                                           |
|-----------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------|
|  | <pre>A = {1, 2, 3, 4, 5} B = {4, 5, 6, 7, 8} print(A &amp; B)</pre> <p>Output: {4, 5}</p> | <pre>A = {1, 2, 3, 4, 5} B = {4, 5, 6, 7, 8} print(A.intersection(B)) print(B.intersection(A))</pre> <p>Output:<br/>{4, 5}<br/>{4, 5}</p> |
|-----------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------|

**Set Difference: Difference of the set B from set A (A - B) is a set of elements that are only in A but not in B.** Similarly, B - A is a set of elements in B but not in A. Difference is performed using - operator. Same can be accomplished using the **difference()** method.

|                                                                                   |                                                                                        |                                                                                                                                         |
|-----------------------------------------------------------------------------------|----------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
|  | <pre>A = {1, 2, 3, 4, 5} B = {4, 5, 6, 7, 8} print(A - B)</pre> <p>Output: {1,2,3}</p> | <pre>A = {1, 2, 3, 4, 5} B = {4, 5, 6, 7, 8} print(A.difference(B)) print(B.difference(A))</pre> <p>Output:<br/>{1,2,3}<br/>{6,7,8}</p> |
|-----------------------------------------------------------------------------------|----------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------|

**Set Symmetric Difference:** Symmetric Difference of A and B is a set of elements in A and B but not in both (excluding the intersection). Symmetric difference is performed using ^ operator. Same can be accomplished using the method **symmetric\_difference()**.

|                                                                                     |                                                                                              |                                                                                                                                                                     |
|-------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  | <pre>A = {1, 2, 3, 4, 5} B = {4, 5, 6, 7, 8} print(A ^ B)</pre> <p>Output: {1,2,3,6,7,8}</p> | <pre>A = {1, 2, 3, 4, 5} B = {4, 5, 6, 7, 8} print(A.symmetric_difference(B)) print(B.symmetric_difference(A))</pre> <p>Output: {1,2,3,6,7,8}<br/>{1,2,3,6,7,8}</p> |
|-------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|

### 18. Write Characteristics of Dictionary. Explain operations on dictionary

- ✓ Python dictionary is an ordered collection of items.
- ✓ Each item of a dictionary has a key/value pair.

#### Characteristics of Dictionary:

- ✓ It is used to store data in a key-value pair format.
- ✓ It is mutable(changeable).
- ✓ Duplicate values are not allowed.
- ✓ Key must be a single element.
- ✓ Value can be of any type such as list,tuple,integer etc.

#### Changing and Adding Dictionary elements:

- ✓ We can add new items or change the value of existing items using an **assignment operator**.
- ✓ If the key is already present, then the existing value gets updated.
- ✓ In case the key is not present, a new (key: value) pair is added to the dictionary.

|                                                      |                                                              |
|------------------------------------------------------|--------------------------------------------------------------|
| <pre>x = {'name': 'abc', 'rollno': 1} print(x)</pre> | <p><b>Output</b></p> <pre>{'name': 'abc', 'rollno': 1}</pre> |
|------------------------------------------------------|--------------------------------------------------------------|

|                                  |                |                                                 |
|----------------------------------|----------------|-------------------------------------------------|
| x['rollno'] = 20<br>print(x)     | # update value | {'name': 'abc', 'rollno': 20}                   |
| x['address'] = 'xyz'<br>print(x) | # adding value | {'name': 'abc', 'rollno': 20, 'address': 'xyz'} |

**Removing elements from Dictionary:**

- ✓ We can remove a particular item in a dictionary by using the **pop() method**.
- ✓ The **popitem()** method can be used to remove and return an arbitrary (key, value) item pair from the dictionary.
- ✓ All the items can be removed at once, using the **clear()** method.
- ✓ We can also use the **del keyword** to remove individual items or the entire dictionary itself.

|                                                                                                                                                                    |                                                                                                     |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------|
| <b>Using pop( )</b><br>squares = {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}<br>print(squares.pop(4)) # remove a particular item, returns its value<br>print(squares)         | <b>Output:</b><br>16<br>{1: 1, 2: 4, 3: 9, 5: 25}                                                   |
| <b>Using popitem( )</b><br>squares = {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}<br>print(squares.popitem()) # remove an arbitrary item, return (key,value)<br>print(squares) | <b>Output:</b><br>(5, 25)<br>{1: 1, 2: 4, 3: 9, 4: 16}                                              |
| <b>Using clear( )</b><br>squares = {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}<br>squares.clear()<br>print(squares)                                                           | <b>Output:</b><br>{ }                                                                               |
| <b>Using del keyword</b><br>squares = {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}<br>del squares[2]<br>print(squares)<br><br>del squares<br>print(squares)                    | <b>Output:</b><br><br>{1: 1, 3: 9, 4: 16, 5: 25}<br><br>NameError: name<br>'squares' is not defined |

**19. Define function. Why do we need it?**

- ✓ A function is a block of code which only runs when it is called.
- ✓ In Python, **a function is a group of related statements that performs a specific task.**
- ✓ Functions help break our program into smaller and modular chunks.
- ✓ As our program grows larger and larger, functions make it more organized and manageable.
- ✓ It avoids repetition and makes the code reusable.
- ✓ You can pass data, known as parameters; into a function.
- ✓ A function can return data as a result.

**20. Give difference between User Define Function and Built in Function.**

| Built-in functions                                       | User defined functions                                              |
|----------------------------------------------------------|---------------------------------------------------------------------|
| They are predefined functions which can be used anytime. | They are functions that are defined by the programmer.              |
| Eg: len( ), type( ), etc                                 | Eg:<br><pre>def perimeter (b,l):     p= 2 *(l+b)     print(p)</pre> |

## 21. What is Recursion? Explain with example.

- ✓ A Function calls itself is said to be a recursion.
- ✓ Recursion is the process of defining something in terms of itself.

### Python Recursive Function:

- ✓ Function can call other functions. It is even possible for the function to call itself. These types of construct are termed as recursive functions.
- ✓ The following image shows the workings of a recursive function called recurse.

```
def recurse():
    ...
    recurse()
    ...
recurse()
```

### Example of a recursive function to find the factorial of an integer.

- ✓ Factorial of a number is the product of all the integers from 1 to that number. For example, the factorial of 6 (denoted as 6!) is  $1*2*3*4*5*6 = 720$ .

### Example :

```
def factorial(x):
    if x == 1:
        return 1
    else:
        return (x * factorial(x-1))
```

### calling a function:

```
num = 3
print("The factorial of", num, "is", factorial(num))
```

**Output:**

The factorial of 3 is 6

**22. What are modules in Python? Write Example of user defined module.**

- ✓ Modules is a file containing Python statements and definitions.
- ✓ Modules are pre defined files that contain the python codes which include the basic functionality of class, methods ,variables etc.
- ✓ Modules can be divided in to two parts: User defined and built in
- ✓ Python contains built-in modules that are already programmed into the language and user defined modules are created by user.
- ✓ We can define our most used functions in a module and import it, instead of copying their definitions into different programs.
- ✓ A file containing Python code, for example: example.py, is called a module, and its module name would be **example**.

**Example of user defined module (module name: example):**

```
def add(a, b):
    result = a + b
    print( result)
```

**How to import modules in Python?**

- ✓ We use the import keyword to do this.

```
import example
```

- ✓ Using the module name we can access the function using the dot ( . ) operator. For example:

```
>>> example.add(4,5.5)
9.5
```

**23. List out method of rand module. Explain any four with example.**

random() ,uniform(),randint( ) ,randrange( ),choice( ),choices( ),shuffle()

random( ): It is used to generate random floats between 0 to 1.

**Syntax:** random.random()

**Example:** import random

```
print(random.random())
```

**Output:** 0.371793355562307

uniform( ): It is used to generate random floats between two given parameters.

**Syntax:** random.uniform(First number,Second number)

**Example:** import random

```
print(random.uniform(1,10))
```

**Output:** 7.771509161751196

randint( ): It returns a random integer between the given integers.

**Syntax:** random.randint(First number,Second number)

**Example:** import random

```
print(random.randint(1,10))
```

**Output:** 3

shuffle(): It is used to shuffle a sequence (list). Shuffling means changing the position of the elements of the sequence

**Syntax:** random.shuffle(sequence)

Example: import random

```
a=[10,11,12,13,14]
print('before shuffle',a)
random.shuffle(a)
print('after shuffle',a)
```

Output:

```
before shuffle [10, 11, 12, 13, 14]
after shuffle [10, 12, 14, 11, 13]
```

#### 24. Explain any five mathematical functions of math module.

**ceil():** Rounds a number up to the nearest integer

Example : import math

```
print(math.ceil(8.23))
```

Output : 9

**floor():** Rounds a number down to the nearest integer

Example : import math

```
print(math.floor(8.23))
```

Output : 8

**sqrt():** It returns the square root of the number.

Example: import math

```
print(math.sqrt(25))
```

Output: 5

**pow():** It receives two arguments, raises the first to the second and give result.

Example: import math

```
print(math.pow(2,4))
```

Output: 16.0

**factorial():** It returns the factorial of a given number.

Example: import math

```
print(math.factorial(4))
```

Output : 24

**gcd():** It is used to find the greatest common divisor of two numbers passed as the arguments.

Example :import math  
print(math.gcd(10,35))

Output :5

**fabs():** It returns the absolute value of the number.

Example :import math  
print (math.fabs(-10))

Output : 10.0

**fmod():** It returns the remainder of x/y.

Example : import math  
print (math.fmod(10,3))

Output : 1.0

**exp():** It returns a float number after raising e to the power of a given number. (e\*\*x)

Example : import math  
print (math.exp(2))

Output: 7.38905609893065

## 25. Explain main classes of Date and Time module with one Example.

**date** –Its attributes are year, month and day.

**time** –Its attributes are hour, minute, second, microsecond, and tzinfo.

**datetime** – Its a combination of date and time along with the attributes year, month, day, hour, minute, second, microsecond, and tzinfo.

**timedelta** – A duration expressing the difference between two date, time, or datetime instances to microsecond resolution.

**tzinfo** – It provides time zone information objects.

**timezone** – A class that implements the tzinfo abstract base class as a fixed offset from the UTC

### Example:1 Get current date and time(Using datetime class)

```
import datetime
d=datetime.datetime.now()
print(d)
```

**Output**

2022-11-28 12:40:36.467347

### Example:2 Get current date (Using date class)

```
import datetime
d=datetime.date.today()
print(d)
```

**Output**

2022-11-28

## 26. List out types of plots in Matplotlib. Explain any one with example.

Matplotlib is a library for creating static, animated and interactive visualization in Python.

Types of Plots: Line Plot, Scatter Plot, Area Plot, Bar Graph, Histogram, Pie plot etc.

### 1. Line plot :

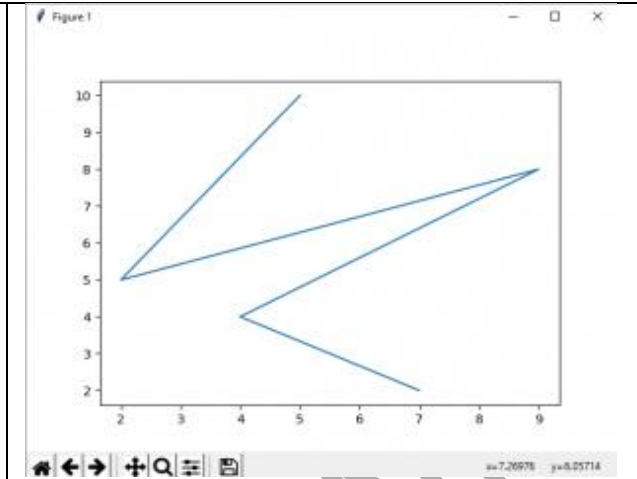
**EXAMPLE:**

**OUTPUT:**

```

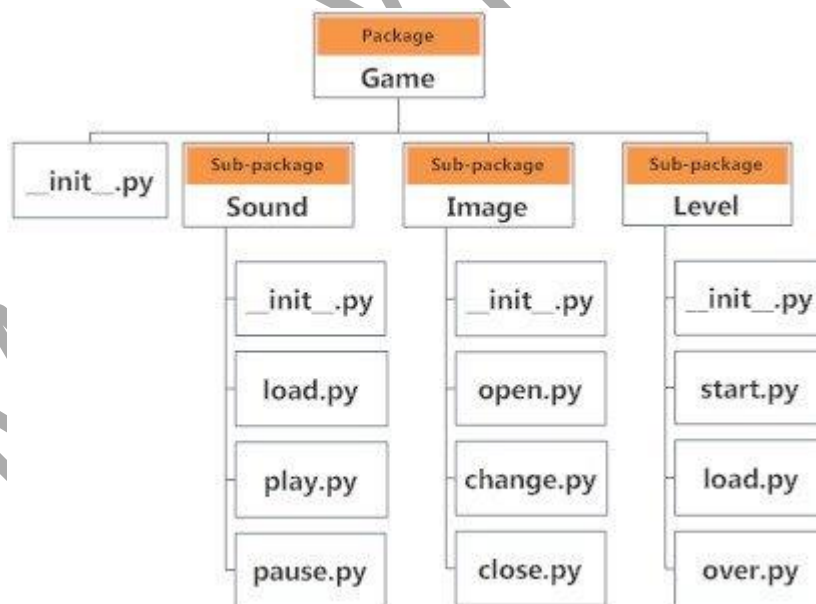
from matplotlib import pyplot as plt
x = [5, 2, 9, 4, 7]
y = [10, 5, 8, 4, 2]
plt.plot(x,y)
plt.show()

```



27. What is package? How to Create and import package. Explain with example.

- ✓ As a directory can contain subdirectories and files, a Python package can have sub-packages and modules.
- ✓ Python has packages for directories and modules for files.
- ✓ A directory must contain a file named `__init__.py` in order for Python to consider it as a package. This file can be left empty
- ✓ Suppose we are developing a **game**. One possible organization of packages and modules could be as shown in the figure below.



#### Importing module from a package

- ✓ We can import modules from packages using the dot (.) operator.
- ✓ For example, if we want to import the start module in the above example, it can be done as follows:

```
import Game.Level.start
```



### ❖ Creating User Defined Package:

- ✓ Create a directory with the name EMP in home directory.
- ✓ Create a Python source file with name p1.py in EMP directory with following code.

```
def getName():
    name=['a','b','c']
    return name
```

- ✓ Now create one more python file with name p2.py in EMP directory with following code.

```
def getSalary():
    salary=[1000,2000,3000]
    return salary
```

- ✓ Now create \_\_init\_\_.py without any code in EMP directory.
- ✓ \_\_init\_\_.py converts Emp directory to Emp package with two modules p1 and p2.

### ❖ Importing a package in Python

- ✓ To use the module defined in Emp package, Create one file Test.py in home directory with code:

```
from Emp import p1
from Emp import p2
print(p1.getName())
print(p2.getSalary())
```

Now run Test.py to see the Output:

```
Output:['a', 'b', 'c']
       [1000, 2000, 3000]
```

### 28. Write down steps for PIP Installation.

- ✓ PIP is the package manager for Python packages.
- ✓ It is a package management system used to install and manage software packages written in Python.
- ✓ We use pip to install packages that do not come with Python.
- ✓ Python pip comes preinstalled on 3.4 or older version of Python.
- ✓ To check whether pip is installed or not, type below command in terminal. It shows the version of pip which is already installed in the system.

```
pip -version
```

- ✓ Pip use PyPI as the default source for packages. So type following command on command prompt for installation of packages.

```
pip install packagename
```

- ✓ Pip will look for that package on PyPI and if found ,download it and install the package on local system.
- ✓ For **uninstalling a package** type following command on command prompt.

```
pip uninstall packagename
```

**Download and Install PIP:**

- ✓ Download the get-pip.py file and store it in the same directory as python is installed.
- ✓ Change the current path of the directory in the command line to the path of the directory where get-pip.py file exists.
- ✓ Type following command in command prompt and wait through the installation process.

```
python get-pip.py
```

- ✓ Type following command to check whether pip is installed successfully or not. If successfully installed then display current version of it.

```
pip -V
```

**29. How to use slicing operator for string. Explain with example.**

- ✓ A segment of a string is called a slice.
- ✓ Selecting a slice is similar to selecting a character:
- ✓ 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.

**Syntax:[Start: stop: steps]**

- ✓ Slicing will start from index and will go up to **stop** in **step** of steps.
- ✓ Default value of start is 0,
- ✓ Stop is last index of list
- ✓ And for step , default is 1

|                                                                                                              |                                                                         |
|--------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------|
| <pre>str = 'Hello World!' print (str ) print(str[0] ) print (str[2:4]) print (str[2:]) print (str * 2)</pre> | <pre>Output Hello World! H ll llo World! Hello World!Hello World!</pre> |
| <pre>str="Hello World" for s in str[0:11:1]:     print(s)</pre>                                              | <pre>Output Hello World</pre>                                           |
| <pre>str="Hello World" for s in str[::2]:     print(s)</pre>                                                 | <pre>Output H l o W r d</pre>                                           |

**30. Explain following function with example: isalnum(), isalpha(), isdigit(), isidentifier(), islower(), isupper(), and isspace()**

**isalnum():** returns True if all the characters in the string is alphanumeric (number or alphabets or both). Otherwise return False

|                                                                         |                                    |
|-------------------------------------------------------------------------|------------------------------------|
| <pre>s="abc" print(s.isalnum()) s='123' print(s.isalnum()) s='+-'</pre> | <pre>Output: True True False</pre> |
|-------------------------------------------------------------------------|------------------------------------|

|                    |  |
|--------------------|--|
| print(s.isalnum()) |  |
|--------------------|--|

**isalpha:** returns True if all the characters in the string are alphabets. Otherwise return False

|                                                                |                          |
|----------------------------------------------------------------|--------------------------|
| s="abc"<br>print(s.isalpha())<br>s='123'<br>print(s.isalpha()) | Output:<br>True<br>False |
|----------------------------------------------------------------|--------------------------|

**isdigit:** returns True if all the characters in the string are digits. Otherwise return False.

|                                                                  |                          |
|------------------------------------------------------------------|--------------------------|
| s="abc"<br>print(s.isdigit())<br>s='123'<br>print(s. isdigit ()) | Output:<br>False<br>True |
|------------------------------------------------------------------|--------------------------|

**isidentifier:** returns True if string is an identifier or a keyword Otherwise return False.

|                                                              |                         |
|--------------------------------------------------------------|-------------------------|
| print("hello".isidentifier())<br>print("for".isidentifier()) | Output:<br>True<br>True |
|--------------------------------------------------------------|-------------------------|

**islower:** returns True if all the characters in the string are in lowercase. Otherwise return False.

|                                    |                  |
|------------------------------------|------------------|
| s = 'Python'<br>print(s.islower()) | Output:<br>False |
|------------------------------------|------------------|

**isupper:** returns True if all the characters in the string are in uppercase. Otherwise return False.

|                                    |                  |
|------------------------------------|------------------|
| s = 'Python'<br>print(s.isupper()) | Output:<br>False |
|------------------------------------|------------------|

**isspace :** returns True if all the characters in the string are whitespace characters. Otherwise return False.

|                         |                 |
|-------------------------|-----------------|
| print("\n\t".isspace()) | Output:<br>True |
|-------------------------|-----------------|

### 31. Explain following function with example: endswith(), startswith(), find(), rfind(), count()

**endswith:** Syntax: endswith(suffix, start, end)

- ✓ Returns True when a string ends with the characters specified by suffix.
- ✓ You can limit the check by specifying a starting index using start or an ending index using end.

|                                                                     |                         |
|---------------------------------------------------------------------|-------------------------|
| s="Hello"<br>print(s.endswith('lo'))<br>print(s.endswith('lo',2,5)) | Output:<br>True<br>True |
|---------------------------------------------------------------------|-------------------------|

**startswith:** Syntax:startswith(prefix, start, end):

- ✓ Returns True when a string begins with the characters specified by prefix.
- ✓ You can limit the check by specifying a starting index using start or an ending index using end.

|                                                                         |                         |
|-------------------------------------------------------------------------|-------------------------|
| s="Hello"<br>print(s.startswith('He'))<br>print(s.startswith('He',0,3)) | Output:<br>True<br>True |
|-------------------------------------------------------------------------|-------------------------|

**Find:** Syntax: find(str, start, end):

- ✓ Check whether str occurs in a string and outputs the index of the location.
- ✓ You can limit the search by specifying starting index using start or a ending index using end.

```
s = 'Hello World'
ans = s.find('Hello')
print("Substring found at index:", ans)
```

Output:  
Substring found at index: 0

**rfind:** Syntax: rfind(str, start, end):

- ✓ Provides the same functionality as find(), but searches backward from the end of the string instead of the starting.
- ✓ You can limit the search by specifying starting index using start or a ending index using end.

```
s = 'Hello World'
ans = s.rfind('Hello')
print("Substring found at index:", ans)
```

Output:  
Substring found at index: 0

**count:** Syntax: count(str, start, end):

- ✓ Counts how many times str occurs in a string.
- ✓ You can limit the search by specifying a starting index using start or an ending index using end.

```
s= "Hello Students Hello"
print(s.count("Hello", 0, 5))
```

Output:  
1

### 32. List out Formatting function and explain any two.

**Formatting functions: ljust(), rjust(),center(),format()**

**ljust:** This method left aligns the string according to the width specified and fills the remaining space of the line with blank space if ‘fillchr’ argument is not passed.

Syntax: ljust(len, fillchr)

len: The width of string to expand it.

fillchr (optional): The character to fill in the remaining space.

```
s = "hello"
print(s.ljust(10))
print(s.ljust(10, '-'))
```

Output:  
hello  
hello-----

**rjust:** This method left aligns the string according to the width specified and fills the remaining space of the line with blank space if ‘fillchr’ argument is not passed.

Syntax: rjust(len, fillchr)

len: The width of string to expand it.

fillchr (optional): The character to fill in the remaining space.

```
s = "hello"
print(s.rjust(10))
print(s.rjust(10, '-'))
```

Output:  
hello  
-----hello

**center:** It creates and returns a new string that is padded with the specified character.

Syntax: string.center(length[, fillchar])

```
s="Hello World"
print(s.center(20))
```

Output:  
Hello World

**33. Explain open() and close() for file with example.****open File:**

- ✓ Before performing any operation on the file like reading or writing, first, we have to open that file.
- ✓ Open() is used to open a file.
- ✓ At the time of opening, we have to specify the mode, which represents the purpose of the opening file.

Syntax: file object = open(filename, access mode)

Example: f = open("a.txt", "r")

Where the following mode is supported:

|    |                                                                                                                                                                                 |
|----|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| r  | open an existing file for a read operation.                                                                                                                                     |
| w  | open an existing file for a write operation. If the file already contains some data then it will be overridden but if the file is not present then it creates the file as well. |
| a  | open an existing file for append operation. It will not override existing data.                                                                                                 |
| r+ | To read and write data into the file. The previous data in the file will be overridden                                                                                          |
| w+ | To write and read data. It will override existing data.                                                                                                                         |
| a+ | To append and read data from the file. It will not override existing data.                                                                                                      |

**close File:**

- ✓ close() function closes the file and frees the memory space acquired by that file.
- ✓ It is used at the time when the file is no longer needed or if it is to be opened in a different file mode.

Syntax: File object.close()

Example: f = open("a.txt", "r")  
f.close()

**34. Explain read(),readline() and readlines () for file with example.**

**read() :** Returns the read bytes in form of a string. It reads n bytes, if no n specified, reads the entire file.

Syntax: File object.read([n])

|                                                      |                                           |
|------------------------------------------------------|-------------------------------------------|
| f=open("a.txt","r")<br>print(f.read() )<br>f.close() | Output:<br>hello everyone<br>Good morning |
|------------------------------------------------------|-------------------------------------------|

**readline() :** Reads a line of the file and returns in form of a string. For specified n, reads at most n bytes. It does not reads more than one line.

Syntax: File object.readline([n])

|                                                         |                           |
|---------------------------------------------------------|---------------------------|
| f=open("a.txt","r")<br>print(f.readline())<br>f.close() | Output:<br>hello everyone |
|---------------------------------------------------------|---------------------------|

**Readlines() :** Reads all the lines and return them as each line a string element in a list.

Syntax: File\_object.readlines()

|                                                           |                                                                        |
|-----------------------------------------------------------|------------------------------------------------------------------------|
| f=open("a.txt","r")<br>print(f.readlines() )<br>f.close() | Output:<br>['hello everyone\n', 'Good morning\n', 'Have a nice day\n'] |
|-----------------------------------------------------------|------------------------------------------------------------------------|

**35. Explain write(),append() and writelines () for file with example.**

**write():**The write() method writes a string to a text file.

Syntax: File\_object.write(str1)

Inserts the string str1 in a single line in the text file.

```
f=open("E:/a.txt","w")
f.write("Hello World")
f.close()
```

Output:  
Open a.txt file ..... **Hello World** is display in text file.

**append:** To append to a text file, you need to open the text file for appending mode.

When the file is opened in append mode, the handle is positioned at the end of the file. The data being written will be inserted at the end, after the existing data.

```
f=open("E:/a.txt","a")
f.write("Good Morning")
f.close()
```

Output:  
Open a.txt file ..... **Hello World Good Morning** is display in text file.

**writelines():**The writelines() method write a list of strings to a file at once.

Syntax: File\_object.writelines(L) for L = [str1, str2, str3]

For a list of string elements, each string is inserted in the text file. Used to insert multiple strings at a single time.

```
f=open("E:/a.txt","w")
lines=["Hello\n","good\n","morning"]
f.writelines(lines)
f.close()
```

Output:  
Open a.txt file .....Below content is displayed in text file.

Hello  
good  
morning

**Note: Prepare manual Programs for GTU Exam...**