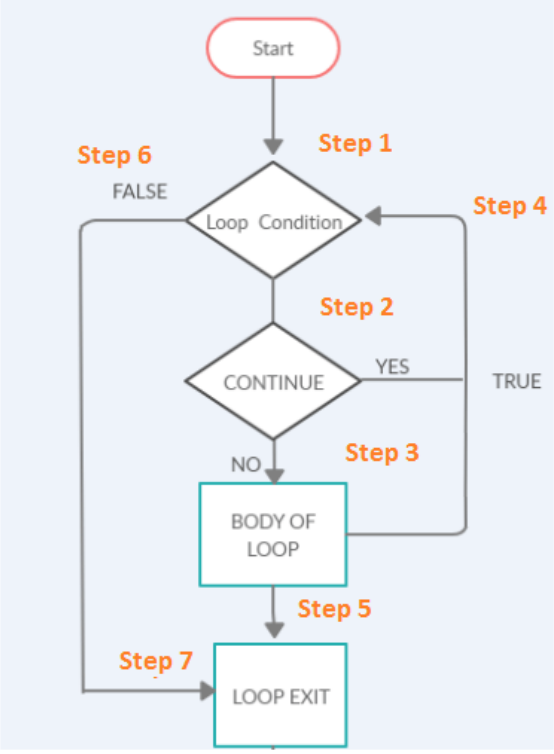
Python Model Paper -2

1. **With Python programming examples to each, explain the syntax and control flow diagrams of break and continue statements.**

   Continue Statement example:

```
for i in range(1, 11):
    if i % 2 == 0:
        continue
    print(i)
```

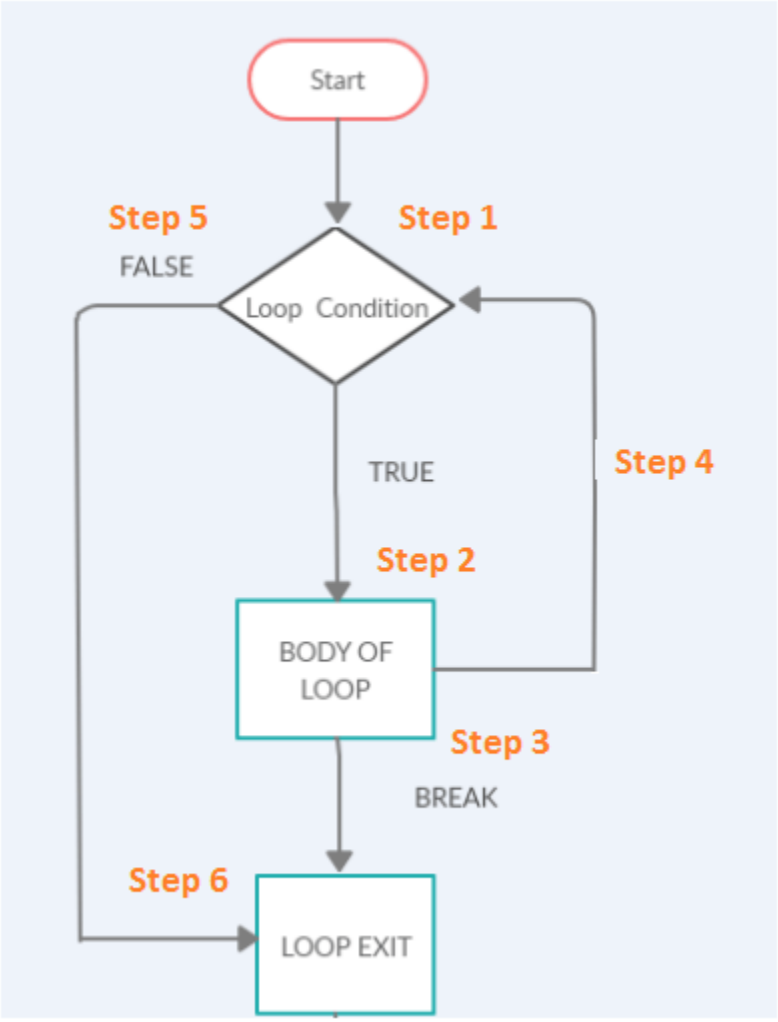
   Continue Statement Flow chart:



Break Statement example:

```
for i in range(1, 11):
    if i == 6:
        break
    print(i)
print("Loop exited prematurely")
```

Break Statement Flowchart:



2. **Explain TWO ways of importing modules into application in Python with syntax and suitable programming examples.**

➢

➢ **import statement:** The import statement is used to import a module or specific attributes and functions from a module. The syntax for importing a module is as follows:

```
import module_name
```

Example:

```python
# importing the math module
import math

# using the pi attribute from math module
print("Value of pi is:", math.pi)

# using the sqrt function from math module
print("Square root of 25 is:", math.sqrt(25))
```

➢ **from...import statement:** The from...import statement is used to import specific attributes or functions from a module. The syntax for using this statement is as follows:

```
from module_name import attribute/function_name
```

```python
# importing the pi attribute and sqrt function from math modul
from math import pi, sqrt

# using the pi attribute directly
print("Value of pi is:", pi)

# using the sqrt function directly
print("Square root of 25 is:", sqrt(25))
```

3. **Write a function to calculate factorial of a number. Develop a program to compute binomialcoefficient (Given N and R).**

   Refer record

4. **Explain looping control statements in Python with a syntax and example to each.**

   Looping control statements in Python are used to control the execution of loops. There are many looping control statements in Python: break, continue etc..

➢ **break statement:** The break statement is used to terminate the loop immediately. When a break statement is encountered inside a loop, the loop is terminated and the program execution continues with the next statement after the loop.

Syntax:

```python
for variable in sequence:
    if condition:
        break
```

Example:

```
# Program to find the first even number in a list

numbers = [1, 3, 5, 2, 6, 7, 8]

for num in numbers:
    if num % 2 == 0:
        print("The first even number is:", num)
        break
```

➢ **continue statement:** The continue statement is used to skip the current iteration of the loop and continue with the next iteration.

Syntax:

```
for variable in sequence:
    if condition:
        continue
```

Example:

```
# Program to print all odd numbers in a list

numbers = [1, 3, 5, 2, 6, 7, 8]

for num in numbers:
    if num % 2 == 0:
        continue
    print(num)
```

5. **Develop a Python program to generate Fibonacci sequence of length (N). Read N from the console.**

   Refer Record

6. **Write a function named DivExp which takes TWO parameters a, b and returns a value c (c=a/b). Write suitable assertion for a>0 in function DivExp and raise an**

exception for when b=0. Develop a Python program which reads two values from the console and calls a function DivExp.

Refer Record

7. **Explain FOUR scope rules of variables in Python.**

   In Python, there are four scope rules for variables, which determine where a variable can be accessed within a program. They are:

- **Local Scope:** A variable defined inside a function or block is considered to have local scope. It can only be accessed within that function or block.
- **Enclosing Scope:** If a variable is not defined inside a function, but is defined in an enclosing function, it can be accessed by the nested function.
- **Global Scope:** A variable defined outside of all functions and blocks is considered to have global scope. It can be accessed from anywhere in the program.
- **Built-in Scope:** This is the widest scope and includes all built-in functions and names. These variables are always available and can be accessed from anywhere in the program.

8. **Explain with a programming example to each: (i) get() (ii) setdefault()**

   (i) get(): The get() method returns the value of a specified key in a dictionary. If the key is not present in the dictionary, it returns the specified default value.

   Syntax:
```
dictionary.get(key, default_value)
```

   Example:

```
# create a dictionary
student = {'name': 'John', 'age': 23, 'gender': 'Male'}

# get the value of the 'name' key
name = student.get('name')
print(name)  # output: John

# get the value of the 'city' key (which is not present in the dictionary)
city = student.get('city', 'Unknown')
print(city)  # output: Unknown
```

(ii) setdefault(): The setdefault() method is used to get the value of a specified key in a dictionary. If the key is not present in the dictionary, it inserts the key with the specified default value. Syntax:

```
dictionary.setdefault(key, default_value)
```

Example:

```
# create a dictionary
student = {'name': 'John', 'age': 23, 'gender': 'Male'}

# get the value of the 'name' key
name = student.setdefault('name', 'Unknown')
print(name)  # output: John

# insert the 'city' key with the value 'Unknown' (as it is not present in th
city = student.setdefault('city', 'Unknown')
print(city)  # output: Unknown

print(student)  # output: {'name': 'John', 'age': 23, 'gender': 'Male', 'cit
```

9. **Develop suitable Python programs with nested lists to explain copy.copy() and copy.deepcopy( ) methods.**

The `copy()` method creates a shallow copy of the original list, i.e., the new list is a separate copy of the original list but the elements in the new list still point to the same objects as the elements in the original list.

The `deepcopy()` method creates a deep copy of the original list, i.e., the new list and its elements are completely independent of the original list and its elements.

Here are the Python programs with nested lists to explain these two methods:

➢ Using `copy()` method:

```
import copy

# Original list
list1 = [[1, 2], [3, 4], [5, 6]]

# Shallow copy using copy() method
list2 = copy.copy(list1)

# Modify the first element of the original list
list1[0][0] = 7

# Print both lists
print("Original list:", list1)
print("New list (shallow copy):", list2)
```

Output:

```
Original list: [[7, 2], [3, 4], [5, 6]]
New list (shallow copy): [[7, 2], [3, 4], [5, 6]]
```

➢ Using `deepcopy()` method:

```
import copy

# Original list
list1 = [[1, 2], [3, 4], [5, 6]]

# Deep copy using deepcopy() method
list2 = copy.deepcopy(list1)

# Modify the first element of the original list
list1[0][0] = 7

# Print both lists
print("Original list:", list1)
print("New list (deep copy):", list2)
```

Output:

```
Original list: [[7, 2], [3, 4], [5, 6]]
New list (deep copy): [[1, 2], [3, 4], [5, 6]]
```

10. Explain append() and index() functions with respect to lists in Python.

The `append()` function is used to add an element to the end of a list.

```
my_list = [1, 2, 3, 4]
my_list.append(5)
print(my_list)  # Output: [1, 2, 3, 4, 5]
```

The `index()` function is used to find the index of the first occurrence of a specified element in a list.

```
my_list = [1, 2, 3, 4, 5]
print(my_list.index(3))  # Output: 2
```

11. Explain different ways to delete an element from a list with suitable Python syntax and programming examples.

➢ **Using the `del` statement:** The `del` statement can be used to delete an element from a list by its index. Here is an example:

```
my_list = [1, 2, 3, 4, 5]
del my_list[2]  # deletes element at index 2 (i.e. 3)
print(my_list)  # output: [1, 2, 4, 5]
```

➢ **Using the `remove()` method**: The `remove()` method can be used to delete an element from a list by its value. Here is an example:

```
my_list = [1, 2, 3, 4, 5]
my_list.remove(3)  # deletes element with value 3
print(my_list)  # output: [1, 2, 4, 5]
```

➢ **Using the `pop()` method:** The `pop()` method can be used to delete an element from a list by its index, and it also returns the deleted element. Here is an example:

```
my_list = [1, 2, 3, 4, 5]
deleted_element = my_list.pop(2)  # deletes element at index 2 (i.e. 3)
print(my_list)  # output: [1, 2, 4, 5]
print(deleted_element)  # output: 3
```

12. **Read a multi-digit number (as chars) from the console. Develop a program to print the frequency of each digit with suitable message.**

    Refer Record

13. **Tuples are immutable. Explain with Python programming example.**

    Tuples are immutable in Python, which means the elements of a tuple cannot be modified once it is created. Here's an example to illustrate this:

```
# Creating a tuple
tup = (1, 2, 3, 4, 5)

# Trying to change the second element of the tuple
tup[1] = 10   # This will result in a TypeError

# Printing the tuple
print(tup)
```

    Output:

```
TypeError: 'tuple' object does not support item assignment
```

14. **Explain Python string handling methods with examples: split(),endswith(),ljust(), center(), lstrip()**

    `split()`: This method is used to split a string into a list of substrings based on a separator. The separator can be a space, a comma, a hyphen, or any other character. Here is an example:

```
my_string = "Hello, World!"
my_list = my_string.split(",")
print(my_list)
```

Output:

```
['Hello', ' World!']
```

`endswith()`: This method is used to check if a string ends with a particular substring. It returns True if the string ends with the specified substring, and False otherwise. Here is an example:

```
my_string = "Hello, World!"
result = my_string.endswith("World!")
print(result)
```

Output:

```
True
```

For ljust center and strip refer IA-2 QB

15. Explain reading and saving python program variables using shelve module with suitable Python program.

The shelve module in Python provides a simple way to store and retrieve Python objects to and from disk. The objects can be stored as key-value pairs, where the keys are strings and the values are the Python objects. In this way, shelve acts like a persistent dictionary.

Here is an example program that demonstrates how to save variables using the shelve module:

```
import shelve

# create a dictionary to store the variables
data = {
    'name': 'John',
    'age': 30,
    'salary': 50000.0
}

# save the data to a shelve file
with shelve.open('data') as db:
    for key, value in data.items():
        db[key] = value

# retrieve the data from the shelve file
with shelve.open('data') as db:
    print('Name:', db['name'])
    print('Age:', db['age'])
    print('Salary:', db['salary'])
```

16. **Explain Python string handling methods with examples: join(),startswith(),rjust(),strip(),rstrip()**

➢ **join() method:** The join() method is used to concatenate a list of strings into a single string using a specified delimiter.

```
list_of_strings = ['apple', 'banana', 'cherry']
delimiter = '-'
string = delimiter.join(list_of_strings)
print(string)
```

Output:

```
apple-banana-cherry
```

➢ **startswith() method:** The startswith() method is used to check whether a string starts with a specified substring.

➢
```
string = "Hello World!"
substring = "Hello"
if string.startswith(substring):
    print("The string starts with the substring.")
else:
    print("The string does not start with the substring.")
```

Output:
```
The string starts with the substring.
```

Refer IA-2 Qb for r strip r just etc..

17. Explain with suitable Python program segments: (i) os.path.basename() (ii)os.path.join().

(i) `os.path.basename()`**:** This function returns the final component of a path. It essentially extracts the filename from the entire path

```
import os

path = '/Users/JohnDoe/Documents/testfile.txt'
filename = os.path.basename(path)
print(filename)  # output: 'testfile.txt'
```

(ii) `os.path.join()`**:** This function joins one or more path components into a single path. It automatically adds a separator character between the components.

```
import os

path1 = '/Users/JohnDoe/Documents'
path2 = 'testfile.txt'
filepath = os.path.join(path1, path2)
print(filepath)  # output: '/Users/JohnDoe/Documents/testfile.txt'
```

18. Develop a Python program find the total size of all the files in the given directory

Here's an example Python program that uses the `os` module to traverse a directory and its subdirectories and calculates the total size of all the files:

```
import os

def get_directory_size(path):
    total_size = 0
    for root, dirs, files in os.walk(path):
        for file in files:
            file_path = os.path.join(root, file)
            total_size += os.path.getsize(file_path)
    return total_size

directory_path = "/path/to/directory"
total_size = get_directory_size(directory_path)
print("Total size of files in directory:", total_size, "bytes")
```

19. Explain permanent delete and safe delete with a suitable Python programming example to each.

Permanent delete refers to the deletion of a file or directory in a way that it cannot be restored or recovered. Once a file or directory is permanently deleted, it is removed from the file system permanently and cannot be recovered even with specialized software tools. The Python programming example to permanently delete a file is as follows:

```
import os

# Delete a file permanently
os.remove("filename.txt")
```

Safe delete, on the other hand, is a way of deleting files or directories in a manner that allows them to be recovered if necessary. This is done by moving the file or directory to a temporary location, such as the recycle bin or trash folder, instead of deleting it permanently. The Python programming example to safely delete a file by moving it to the recycle bin is as follows:

```
import send2trash

# Delete a file safely by moving it to the recycle bin
send2trash.send2trash("filename.txt")
```

20. Develop a program to backing Up a given Folder (Folder in a current working directory) into a ZIP File by using relevant modules and suitable methods.

Refer Record

21. Explain the role of Assertions in Python with a suitable program.

Assertions are a means of ensuring that certain conditions are met before proceeding with the execution of the program. They can be used to check the correctness of the program and to ensure that the assumptions made by the programmer are valid. Here's an example of how assertions can be used in Python:

```
def divide(x, y):
    assert y != 0, "Divisor cannot be zero"
    return x / y

print(divide(10, 5)) # Output: 2.0
print(divide(10, 0)) # AssertionError: Divisor cannot be zero
```

22. Explain the functions with examples: (i) shutil.copytree() (ii) shutil.move()

(iii) shutil.rmtree().

(i) `shutil.copytree(src, dst, symlinks=False, ignore=None, copy_function=copy2, ignore_dangling_symlinks=False)`

This function recursively copies an entire directory tree from source path `src` to destination path `dst`. It returns the destination path `dst` after the copy operation is completed. Here is an example of how to use this function:

```
import shutil

# define source and destination paths
src = '/path/to/source/folder'
dst = '/path/to/destination/folder'

# copy directory tree from src to dst
shutil.copytree(src, dst)
```

(ii) `shutil.move(src, dst, copy_function=copy2)`

This function moves a file or directory from source path `src` to destination path `dst`. It returns the destination path `dst` after the move operation is completed. If `src` and `dst` are on the same filesystem, then `os.rename()` is used for efficient move operation, else a copy operation is performed followed by deleting the original file. Here is an example of how to use this function:

```python
import shutil

# define source and destination paths
src = '/path/to/source/file'
dst = '/path/to/destination/file'

# move file from src to dst
shutil.move(src, dst)
```

(iii) `shutil.rmtree(path, ignore_errors=False, onerror=None)`

This function removes an entire directory tree from path `path`, including all its subdirectories and files. It returns None after the deletion is completed. Here is an example of how to use this function:

```python
import shutil

# define directory path to be deleted
path = '/path/to/directory'

# delete entire directory tree
shutil.rmtree(path)
```

23. Develop a Python program to traverse the current directory by listing sub-folders and files.

You can use the `os` module in Python to traverse the current directory and list all the sub-folders and files. Here's an example program:

```python
import os

def traverse_dir(path):
    for item in os.listdir(path):
        item_path = os.path.join(path, item)
        if os.path.isdir(item_path):
            print("Folder:", item_path)
            traverse_dir(item_path)
        else:
            print("File:", item_path)

if __name__ == "__main__":
    path = os.getcwd()
    traverse_dir(path)
```

(In this program, we first import the `os` module. Then we define a `traverse_dir` function that takes a path as input. We use the `os.listdir` function to list all the items in the directory at the given path. We then loop through each item and check if it is a folder or a file using the `os.path.isdir` function. If it is a folder, we print its name and call the `traverse_dir` function recursively to traverse the sub-folder. If it is a file, we print its name.

In the `main` function, we get the current working directory using the `os.getcwd` function and pass it to the `traverse_dir` function to start the traversal.

This program will traverse the current directory and all its sub-folders, printing the name of each folder and file it encounters.)

24. Explain the support for Logging with logging module in Python.

Python provides a built-in module called `logging` which is used to provide a flexible logging system for applications. The `logging` module allows the application to generate log messages for various events, and also allows the developer to define custom logging levels and output destinations.

Here is a simple example of how to use the `logging` module to log messages:

```
import logging

# set up logging
logging.basicConfig(filename='example.log', level=logging.DEBUG)

# log messages
logging.debug('This is a debug message')
logging.info('This is an info message')
logging.warning('This is a warning message')
logging.error('This is an error message')
logging.critical('This is a critical message')
```

(In the above example, we first set up the logging configuration using the `basicConfig()` method. We specify the logging level as `DEBUG` and the output destination as a file called `example.log`. After the logging configuration is set up, we use the various logging methods such as `debug()`, `info()`, `warning()`, `error()`, and `critical()` to log messages with different logging levels.**)** **Output:**

```
DEBUG:root:This is a debug message
INFO:root:This is an info message
WARNING:root:This is a warning message
ERROR:root:This is an error message
CRITICAL:root:This is a critical message
```

25. Explain the methods __init__ and __str__ with suitable code example to each.

The `__init__` and `__str__` methods are two special methods in Python classes. The `__init__` method is called automatically when an object is created, and it initializes the object's attributes. The `__str__` method is called when the object is printed, and it returns a string representation of the object.

Here's an example of how to use the `__init__` method to initialize an object's attributes:

```
class Car:
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year
```

Here's an example of how to use the `__str__` method to return a string representation of the object:

```
class Car:
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year

    def __str__(self):
        return f"{self.year} {self.make} {self.model}"
```

26. Define a function which takes TWO objects representing complex numbers and returns new complex number with a addition of two complex numbers.Define a

suitable class 'Complex' to represent the complex number.Develop a program to read N (N >=2) complex numbers and to compute the addition of N complex numbers.

Refer record

27. Explain the following with syntax and suitable code snippet:

i) Class definition ii) instantiation iii) passing an instance (or objects) as an argument iv) instances as return values.

i) Class definition:

In Python, a class can be defined using the `class` keyword followed by the class name and a colon. Inside the class definition, you can define methods, attributes, and other properties of the class.

```
class ClassName:
    # class definition
```
Syntax

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
```
Snippet

ii) Instantiation:

Instantiation is the process of creating an instance of a class, which can be done using the class name followed by parentheses.

```
instance_name = ClassName(arguments)
```
Syntax

```
person1 = Person("John", 30)
```
Snippet

iii) Passing an instance (or objects) as an argument:

In Python, you can pass an instance (or object) of a class as an argument to a function or method. This allows you to manipulate the instance data in the function or method.

```
def function_name(instance_name):
    # code
```
Syntax

```
def print_person_details(person):
    print("Name:", person.name)
    print("Age:", person.age)

person1 = Person("John", 30)
print_person_details(person1)
```
Snippet

iv) Instances as return values:

In Python, you can return instances of a class from a function or method. This allows you to create and return new instances of a class based on some input.

```
def function_name(arguments):
    # code
    return instance_name
```
Syntax

```
def create_person(name, age):
    person = Person(name, age)
    return person

person2 = create_person("Jane", 25)
```
Snippet

28. **Define pure function and modifier. Explain the role of pure functions and modifiers in application development with suitable python programs.**

In programming, a pure function is a function that always produces the same output for the same input and doesn't have any side effects. On the other hand, a modifier function is a function that modifies the input data in some way and produces a new output.

The role of pure functions is to provide a reliable, predictable way of processing data without the risk of introducing unexpected side effects. Since pure functions only rely on their input parameters to produce output, they are easier to test and debug. Here's an example of a pure function:

```
def multiply(x, y):
    return x * y
```

This function takes two parameters `x` and `y` and returns their product without modifying them in any way. It is a pure function because it always produces the same output for the same input and doesn't have any side effects.

On the other hand, a modifier function modifies the input data in some way and produces a new output. Here's an example of a modifier function:

```
def add_one(numbers):
    for i in range(len(numbers)):
        numbers[i] += 1
    return numbers
```

This function takes a list of numbers as input, increments each number by one, and returns the modified list. It is a modifier function because it modifies the input data in place.

In application development, pure functions are often used in situations where data needs to be processed without altering its original state. Modifier functions, on the other hand, are used when the input data needs to be modified in some way.

For example, suppose we have a list of numbers and we want to compute the sum of the squares of the even numbers in the list. Here's how we could do this using a pure function:

```
def square(x):
    return x * x

def sum_of_squares(numbers):
    even_numbers = filter(lambda x: x % 2 == 0, numbers)
    squares = map(square, even_numbers)
    return sum(squares)
```

NOTE:~ For the above code add these two lines:-

result=square(5) (Here you can pass any number)

print(result)

On the other hand, suppose we have a list of numbers and we want to add a fixed value to each number in the list. Here's how we could do this using a modifier function:

```
def add(numbers, value):
    for i in range(len(numbers)):
        numbers[i] += value

numbers = [1, 2, 3, 4, 5]
add(numbers, 10)
print(numbers)  # prints [11, 12, 13, 14, 15]
```

NOTE:~ 27 AND 28 QUESTIONS ARE 10 MARK QUESTIONS DO NOT SKIP ANYTHING!

ANSWERS MAY CONTAIN SOME ERRORS**