Q.2Python code execution flow explanation

ANS:

Python Code Execution Flow:

1. Top-Down Execution

Python code is executed from top to bottom, line by line.

2. Sequential Execution

Each line of code is executed in sequence, one after the other.

3. Error Handling

If an error occurs during execution, the program will stop at that point and display an error message.

4. No Implicit Execution

Unlike some other languages, Python does not have implicit execution of code. Every line of code must be explicitly written and executed.

5. Control Flow Statements

Control flow statements such as if, for, while, and try-except blocks can alter the execution flow of the program.

6. Function Calls

When a function is called, the execution flow jumps to the function definition and executes the code inside the function. Once the function completes, the execution flow returns to the point where the function was called.

7. Module Import

When a module is imported, the code in the module is executed at the time of import.

Q.3List partition question

ANS:

```
list=['A1','A2','A3','B1','B2','B3','C1','C2','C3','D1']

l1= len(list) // 3
```

```
partitions = [list[i:i+l1] for i in range(0, len(list), l1)]
if len(partitions[-1])!=l1:
   for i in range(len(partitions[1])-len(partitions[-1])):
      partitions[-1].append(None)
print(partitions)
```

Q.4Use Enum in python

ANS:

```
from enum import Enum
class Animal(Enum):
   DOG = 1
   CAT = 2
   MOUSE = 3
print(Animal.DOG)
print(Animal.DOG.value)
```

OUTPUT:

```
Animal.DOG
1
for animal in Animal:
   print(animal)
```

OUTPUT:

```
Animal.DOG
Animal.CAT
Animal.MOUSE
```

Q.5Prepare notes for itertools functions which are commonly used

ANS:

1.itertools.chain: This function returns an iterator that returns elements from the first iterable until it is exhausted,

then proceeds to the next iterable, until all of the iterables are exhausted.

list(itertools.chain([1, 2, 3], [4, 5, 6], [7, 8, 9]))

Output: [1, 2, 3, 4, 5, 6, 7, 8, 9]

2.itertools.product(*iterables, repeat=1): This function returns an iterator that generates the Cartesian product of input iterables.

list(itertools.product([1, 2], [3, 4]))

Output: [(1, 3), (1, 4), (2, 3), (2, 4)]

3.itertools.permutations(iterable, r=None): This function returns an iterator that generates all possible permutations of length r from the input iterable.

list(itertools.permutations('ABC', 2))

Output: [('A', 'B'), ('A', 'C'), ('B', 'A'), ('B', 'C'), ('C', 'A'), ('C', 'B')]

4.itertools.combinations(iterable, r=None): This function returns an iterator that generates all possible combinations of length r from the input iterable.

list(itertools.combinations('ABC', 2))

Output: [('A', 'B'), ('A', 'C'), ('B', 'C')]

5.itertools.accumulate(iterable[, func[, initial]]): This function returns an iterator that returns accumulated values of the input iterable.

list(itertools.accumulate([1, 2, 3, 4], operator.add))

Output: [1, 3, 6, 10]

6.itertools.groupby(iterable[, key[, reverse]]): This function returns an iterator that groups consecutive elements of the input iterable by a common key.

list(itertools.groupby('AAABBBCCCD'))

Output:

[('A', <itertools._grouper at 0x20634180820>),

('B', <itertools._grouper at 0x20634180640>),

('C', <itertools._grouper at 0x20634180400>),

('D', <itertools._grouper at 0x20634180340>)]

7.itertools.repeat(object[, times]): This function returns an iterator that generates an infinite sequence of object or a sequence of object repeated times times.

list(itertools.repeat('A', 3))

Output: ['A', 'A', 'A']

Q.6Example of pass by reference and pass by value

ANS:

PASS BY REFERENCE:

```
def changeme(mylist):
   "This changes a passed list into this function"
   mylist.append([1,2,3,4])
   print("Values inside the function: ", mylist)
mylist = [10,20,30]
changeme(mylist)
print("Values outside the function: ", mylist)
```

OUTPUT:

Values inside the function: [10, 20, 30, [1, 2, 3, 4]]

Values outside the function: [10, 20, 30, [1, 2, 3, 4]]

PASS BY VALUE:

```
def callbyvalue(n):
   "This changes a passed integer into this function"
   n = 100
```

```
    print("Values inside the function: ", n)
num = 20
callbyvalue(num)
print("Values outside the function: ", num)
```

OUTPUT:

Values inside the function: 100

Values outside the function: 20

Q.7Demo of Union operator using typing library

ANS:

The Union operator in Python, introduced in the typing library, is used to define a type that can be one of several types.

It is represented by the Union class in the typing library.

```
from typing import Union
value1: Union[int, str] = 42
value2: Union[int, str] = "hello"
print(f"The type of value1 is {type(value1)} and its value is {value1}")
print(f"The type of value2 is {type(value2)} and its value is {value2}")
```

OUTPUT:

The type of value1 is <class 'int'> and its value is 42

The type of value2 is <class 'str'> and its value is hello

Q.8How to check if variable is iterable python

ANS:

By using the iter() function in a try-except block

EXAMPLE:

```
def is_iterable(value):
   try:
      iter(value)
      return True
   except TypeError:
      return False
print(is_iterable([1, 2, 3]))
print(is_iterable("hello"))
print(is_iterable(42))
```

OUTPUT:

```
True
True
False
```

Q.9What are various ways to implement string formatting in python.

ANS:

1. format() method:

```
name = 'pappu'
print('Hello, {}!'.format(name))
```

2.f-strings:

```
name = 'pappu'
print(f'Hello, {name}!')
```

3.THE % operator:

```
name = 'rahul'
print('Hello, %s!' % name)
4.Template class from the string module:
from string import Template
name = 'priya'
s = Template('Hello, $name!')
print(s.substitute(name=name))
5.str.format() method: (KEYWORD ARGUMENT)
name = 'Sai'
print('Hello, {name}!'.format(name=name))
6.str.format() method:(POSITIONAL ARGUMENT)
name = 'Rohan'
print('Hello, {}!'.format(name))
```

Q.10Example of any() and all()

```
ANS:
EXAMPLE OF ANY:
numbers1 = [0, 1, 2, 4]
result = any(numbers1)
print(result)
OUTPUT:
TRUE # AS at least one of them is non zero.


EXAMPLE OF ALL:
numbers2 = [1, 2, 3, 4,0]
```

```
result = all(numbers2)

print(result)
```

OUTPUT:

FALSE # AS All are not non zero one of them is zero.