**Python and OOP (Theory- Final) Answer Script**

|  |
| --- |
| Question No. 01` |
|  |
| Answer:  data={  'a':[{  'aa':{'aax':5,'aay':6,'aaz':7},  'ab':{'abx':8,'aby':9,'abz':10}  },  {  'aaa':{'aaax':11,'aaay':12,'aaaz':13},  'aab':{'aabx':14,'aaby':15,'aabz':16}  }],  'b':[{  'ba':{'bax':17,'bay':18,'baz':19},  'bb':{'bbx':20,'bby':21,'bbz':22}  },  {  'bba':{'bbax':23,'bbay':24,'bbaz':25},  'bbb':{'bbbx':26,'bbby':27,'bbbz':28}  }],  'c':[{  'ca':{'cax':29,'cay':30,'caz':31},  'cb':{'cbx':32,'cby':33,'cbz':34}  },  {  'cca':{'ccax':35,'ccay':36,'ccaz':37},  'ccb':{'ccbx':38,'ccby':39,'ccbz':40}  }]  }  list1=[]  for key, value in data.items():  list1.append(value)  # print(key, value)  list2=[]  for i in range(len(list1)):  for j in range(len(list1[i])):  # print(list1[i][j])  list2.append(list1[i][j])  # print(type(list1[j]))  list3=[]  for i in range(len(list2)):  # print(type(list2[i]))  for k, v in list2[i].items():  # print(v)  # print(type(v))  for k1,v1 in v.items():  # print(k1, v1)  print(f"Key:{k1} Value: {v1}") |

|  |
| --- |
| Question No. 02 |
|  |
| Answer:  # method 1  lst=[x for x in range(1,6)]  # print(lst)  data={k:[x for x in lst if x!=k] for k in range(1,6)}  print(data)  """  # method 2  dct={k: [item for item in [x for x in range(1,6)] if item!=k] for k in range(1,6) }  print(dct)  """ |

|  |
| --- |
| Question No. 03 |
|  |
| Answer:  from math import e  # mathematical constant e = 2.718281…  print("this is the built in value of e: ", e)  # setting a global value of e  e=2.50  print("this is global value of e: ", e)  def outer\_func():  e=2.123  print("this is local scope of outer\_func, value of e: ", e)  def inner\_func():  e=2.99  print("this is local scope of inner\_func, value of e: ", e)  inner\_func()  outer\_func()  print("global value is not modified in outer\_func", e) |
| There are 4 types of scope in python:  Local, Global, Enclosed or non-local, built in. A python program follows the following sequence, enclosed, local, global and, then built in.  - A variable has a local scope when it is declared in a function, here e=2.123 inside the outer\_func() has a local scope, it works from the beginning to the end of the outer\_func().  - A variable has an enclosed scope when it is defined in a nested function. Here, e=2.99 has an enclosed scope. It works inside the inner\_func().  - In this example, e=2.50 has global scope, because it can be accessed from anywhere in the program. This variable value cannot be modified by outer\_func().  - At last, e is a mathematical constant, which has a built-in value. If the value of e is not modified globally, built-in value works. |

|  |
| --- |
| Question No. 04 |
|  |
| Answer:  UML class diagram: |

|  |
| --- |
| Question No. 05 |
|  |
| Answer:  # modified code  data=[{'a':5,'b':10},{'x':15,'y':20}]  for val in data:  for key,val in val.items():  print(f"Key:{key} Value:{val}") |

|  |
| --- |
| Question No. 06 |
|  |
| Answer:  # function def. to print a row  def print\_row(lst):  for val in lst:  print(val, end=" ")  print()  # main()  n=int(input("enter n: "))  # creat list n by n  lst=[x for x in range(1, n+1)]  # print(lst)  print\_row(lst)  for i in range(n-1):  if i<n-1:  lst[i], lst[i+1]= lst[i+1], lst[i]  print\_row(lst) |

|  |
| --- |
| Question No. 07 |
|  |
| Answer:  lst = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21]  new=lst[2::3]  print(new) |

|  |
| --- |
| Question No. 08 |
|  |
| Answer:  even\_odd=lambda n:"Yes" if (n%2==0) else "No"  print(even\_odd(50)) |

|  |
| --- |
| Question No. 09 |
|  |
| Answer:  from time import sleep, perf\_counter  from threading import Thread  start\_time=perf\_counter()  def f1():  for i in range(5):  print(f"f1 - {i}")  # print("\n")  # sleep(1)    def f2():  for i in range(5):  print(f"f2 - {i}")  # print("\n")  # sleep(1)  def f3():  for i in range(5):  print(f"f3 - {i}")  # print("\n")  # sleep(1)  def f4():  for i in range(5):  print(f"f4 - {i}")  # print("\n")  # sleep(1)  t1= Thread(target=f1)  t2=Thread(target=f2)  t3=Thread(target=f3)  t4=Thread(target=f4)  t1.start()  t2.start()  t3.start()  t4.start()  t1.join()  t2.join()  t3.join()  t4.join()  # f1()  # f2()  # f3()  # f4()  end\_time=perf\_counter()  print(end\_time-start\_time)  # 20 sec  # 5 sec |

|  |
| --- |
| Question No. 10 |
|  |
| Answer: |
| Explanation:  Single inheritance:  Here a child or derived class can inherit only a single parent class, thus all the attributes and methods of the parent class can be accessed from the child class.  In the above example, “Student” is the parent class and “School” is the derived class. Super() method is used to access the parent class.  Multiple inheritance:  In this case, the child class can inherit more than one parent class or base class. Here, “Gmail” and “Gdrive” are separate parent classes and “Google” is the derived class that inherits both the parent classes and all their features. Here, super() method does not work, we have to define it separately.  Multi-level inheritance:  In this inheritance, a child class is again inherited by another class, for example, features of the grandmother are inherited by the mother and from the mother to her daughter.  In the above code example, the “Earth” class is inherited by the “SolarSystem” class, which is further inherited by the “MilkyWay” class. |

Submitted by: Subrata Saha

Email:[Subratabaec@gmail.com](mailto:Subratabaec@gmail.com)

Date: 05.12.2022

THE END