1.Write a program which takes 2 digits, X,Y as input and generates a 2-dimensional array. The element value in the i-th row and j-th column of the array should be i\*j.Note: i=0,1.., X-1; j=0,1,...Y-1.

#!/usr/bin/pyhton

input\_str = raw\_input(“enter the input”)

print input\_str

dimensions=[int(x) for x in input\_str.split(',')]

rowNum=dimensions[0]

colNum=dimensions[1]

multilist = [[0 for col in range(colNum)] for row in range(rowNum)]

#print multilist

for row in range(rowNum):

for col in range(colNum):

multilist[row][col]= row\*col

print multilist

o/P : enter the input 2,3

[[0, 0, 0], [0, 1, 2]]

2.Suppose the following input is supplied to the program: 8 Then, the output should be:1. {1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64}

#!/usr/bin/pyhton

num = input(“enter the number”)

num=num+1

list1=[]

list2=[]

for ind in range(1,num):

list1.append(ind)

list2.append(ind\*ind)

dict=dict(zip(list1,list2))

print dict

o/P : enter the number 8

{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64}

3.Write a program which takes 2 digits, X,Y as input and generates a 2-dimensional array. The element value in the i-th row and j-th column of the array should be i\*j. Note: i=0,1.., X-1; j=0,1,¡­Y-1.  Example  Suppose the following inputs are given to the program: 3,5 Then, the output of the program should be: [[0, 0, 0, 0, 0], [0, 1, 2, 3, 4], [0, 2, 4, 6, 8]]

#!/usr/bin/pyhton

input\_str = raw\_input(“enter the input”)

print input\_str

dimensions=[int(x) for x in input\_str.split(',')]

rowNum=dimensions[0]

colNum=dimensions[1]

multilist = [[0 for col in range(colNum)] for row in range(rowNum)]

#print multilist

for row in range(rowNum):

for col in range(colNum):

multilist[row][col]= row\*col

print multilist

o/P : enter the input 3,5

[[0, 0, 0, 0, 0], [0, 1, 2, 3, 4], [0, 2, 4, 6, 8]]

4. How memory is managed in Python?

Memory management in Python involves a private heap containing all Python objects and data structures. Interpreter takes care of Python heap and that the programmer has no access to it.

- The allocation of heap space for Python objects is done by Python memory manager. The core API of Python provides some tools for the programmer to code reliable and more robust program.

- Python also has a build-in garbage collector which recycles all the unused memory. When an object is no longer referenced by the program, the heap space it occupies can be freed. The garbage collector determines objects which are no longer referenced by the sprogram frees the occupied memory and make it available to the heap space.

5.John and Sally should be given bonus on salaries based on their sales, x%. Please write a python code using oops class, callback functions, registering and executing the call back to print the each salary + bonus.

#!/usr/bin/python

class Employee():

def \_\_init\_\_(self,name):

self.name=name

def Sales(self):

self.val=val

def Bonus(self):

if self.val=5000:

self.bonus=(20\*self.val)/100

print "bouns",self.bonus

elif self.val<=4999 and self.val>=3000:

self.bouns=(10\*self.val)/100

print "bouns",self.bouns

else:

print "no bouns"

a=Employee("python")

a.Sales()

a.Bonus()

6.What is Python really? You can (and are encouraged) make comparisons to other technologies in your answer

Here are a few key points:

* Python is an interpreted language. That means that, unlike languages like C and its variants, Python does not need to be compiled before it is run. Other interpreted languages include PHP and Ruby.
* Python is dynamically typed, this means that you don't need to state the types of variables when you declare them or anything like that. You can do things like x=111and then x="I'm a string" without error
* Python is well suited to object orientated programming in that it allows the definition of classes along with composition and inheritance. Python does not have access specifiers (like C++'s public, private), the justification for this point is given as "we are all adults here"
* In Python, functions are first-class objects. This means that they can be assigned to variables, returned from other functions and passed into functions. Classes are also first class objects
* Writing Python code is quick but running it is often slower than compiled languages. Fortunately， Python allows the inclusion of C based extensions so bottlenecks can be optimised away and often are. The numpy package is a good example of this, it's really quite quick because a lot of the number crunching it does isn't actually done by Python
* Python finds use in many spheres - web applications, automation, scientific modelling, big data applications and many more. It's also often used as "glue" code to get other languages and components to play nice.
* Python makes difficult things easy so programmers can focus on overriding algorithms and structures rather than nitty-gritty low level details.

7.Looking at the below code, write down the final values of A0, A1, ...An.

A0 = dict(zip(('a','b','c','d','e'),(1,2,3,4,5)))

A1 = range(10)

A2 = sorted([i for i in A1 if i in A0])

A3 = sorted([A0[s] for s in A0])

A4 = [i for i in A1 if i in A3]

A5 = {i:i\*i for i in A1}

A6 = [[i,i\*i] for i in A1]

O/P : A0 = {'a': 1, 'c': 3, 'b': 2, 'e': 5, 'd': 4}

A1 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

A2 = []

A3 = [1, 2, 3, 4, 5]

A4 = [1, 2, 3, 4, 5]

A5 = {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81}

A6 = [[0, 0], [1, 1], [2, 4], [3, 9], [4, 16], [5, 25], [6, 36], [7, 49], [8, 64], [9, 81]]

8.What does this stuff mean: \*args, \*\*kwargs? And why would we use it?

\*args and \*\*kwargs are mostly used in function definitions. \*args and \*\*kwargs allow you to pass a variable number of arguments to a function. What does variable mean here is that you do not know before hand that how many arguments can be passed to your function by the user so in this case you use these two keywords. \*args is used to send a non-keyworded variable length argument list to the function.

def test\_var\_args(\*argv):

for arg in argv:

print "value :", arg

test\_var\_args(1,’a',2,3)

O/P : value : 1

value : ‘a’

value : 2

value : 3

\*\*kwargs allows you to pass keyworded variable length of arguments to a function. You should use \*\*kwargs if you want to handle named arguments in a function.

def greet\_me(\*\*kwargs):

if kwargs is not None:

for key, value in kwargs.iteritems():

print "%s == %s" %(key,value)

greet\_me(name1="yasoob",name2="roy",name3="jon")

O/P : name1 == yasoob

name2 == roy

name3 == jon

9.Consider the following code, what will it output?

class Node(object):

def \_\_init\_\_(self,sName):

self.\_lChildren = []

self.sName = sName

def \_\_repr\_\_(self):

return "<Node '{}'>".format(self.sName)

def append(self,\*args,\*\*kwargs):

self.\_lChildren.append(\*args,\*\*kwargs)

def print\_all\_1(self):

print(self)

for oChild in self.\_lChildren:

oChild.print\_all\_1()

def print\_all\_2(self):

def gen(o):

lAll = [o,]

while lAll:

oNext = lAll.pop(0)

lAll.extend(oNext.\_lChildren)

yield oNext

for oNode in gen(self):

print(oNode)

oRoot = Node("root")

oChild1 = Node("child1")

oChild2 = Node("child2")

oChild3 = Node("child3")

oChild4 = Node("child4")

oChild5 = Node("child5")

oChild6 = Node("child6")

oChild7 = Node("child7")

oChild8 = Node("child8")

oChild9 = Node("child9")

oChild10 = Node("child10")

oRoot.append(oChild1)

oRoot.append(oChild2)

oRoot.append(oChild3)

oChild1.append(oChild4)

oChild1.append(oChild5)

oChild2.append(oChild6)

oChild4.append(oChild7)

oChild3.append(oChild8)

oChild3.append(oChild9)

oChild6.append(oChild10)

# specify output from here onwards

oRoot.print\_all\_1()

oRoot.print\_all\_2()

O/P :

<Node 'root'>

<Node 'child1'>

<Node 'child4'>

<Node 'child7'>

<Node 'child5'>

<Node 'child2'>

<Node 'child6'>

<Node 'child10'>

<Node 'child3'>

<Node 'child8'>

<Node 'child9'>

<Node 'root'>

<Node 'child1'>

<Node 'child2'>

<Node 'child3'>

<Node 'child4'>

<Node 'child5'>

<Node 'child6'>

<Node 'child8'>

<Node 'child9'>

<Node 'child7'>

<Node 'child10'>

10.What will be the output of the code below? Explain your answer.

def multipliers():

return [lambda x : i \* x for i in range(4)]

print [m(2) for m in multipliers()]

O/P : [6, 6, 6, 6]