SUMMER-2022 EXAMINATION

Subject Name: Programming with Python Model Answer Subject Code: 22616

Important Instructions to examiners:

1. **The answers should be examined by key words and not as word-to-word as given in the model answer scheme.**
2. **The model answer and the answer written by candidate may vary but the examiner may try to assess the understanding level of the candidate.**
3. **The language errors such as grammatical, spelling errors should not be given more Importance (Not applicable for subject English and Communication Skills.**
4. **While assessing figures, examiner may give credit for principal components indicated in the figure. The figures drawn by candidate and model answer may vary. The examiner may give credit for any equivalent figure drawn.**
5. **Credits may be given step wise for numerical problems. In some cases, the assumed constant values may vary and there may be some difference in the candidate's answers and model answer.**
6. **In case of some questions credit may be given by judgement on part of examiner of relevant answer based on candidate's understanding.**
7. **For programming language papers, credit may be given to any other program based on equivalent concept.**
8. **As per the policy decision of Maharashtra State Government, teaching in English/Marathi and Bilingual (English + Marathi) medium is introduced at first year of AICTE diploma Programme from academic year 2021-2022. Hence if the students in first year (first and second semesters) write answers in Marathi or bilingual language (English +Marathi), the Examiner shall consider the same and assess the answer based on matching of concepts with model answer.**

<table>
<tr><th>Q. No.</th><th>Sub Q. N.</th><th>Answer</th><th>Marking Scheme</th></tr>
<tr><td>1</td><td colspan="2">Attempt Any FIVE of the following</td><td>10</td></tr>
<tr><td></td><td>a)</td><td>Name different modes of Python
Python has two basic modes:
• Script (Normal Mode)
• Interactive Mode</td><td>2M (1m each)</td></tr>
<tr><td></td><td>b)</td><td>List identity operators in python
Identity operators in Python are
• is
• is not</td><td>2M (1m each)</td></tr>
<tr><td></td><td>c)</td><td>Give two differences between list and tuple
<table>
<tr><th>List</th><th>Tuple</th></tr>
<tr><td>Lists are mutable</td><td>Tuples are immutable</td></tr>
<tr><td>Lists consume more memory</td><td>Tuple consume less memory as compared to the list</td></tr>
<tr><td>Lists have several built-in methods</td><td>Tuple does not have many built-in methods.</td></tr>
<tr><td>The unexpected changes and errors are more likely to occur</td><td>In tuple, it is hard to take place.</td></tr>
</table></td><td>2M (1m for each difference, any 2 difference)</td></tr>
</table>

		The List has the variable length	The tuple has the fixed length
		List operations are more error prone.	Tuples operations are safe
		Lists can be used to store homogeneous and heterogeneous elements.	Tuples are used to store only heterogeneous elements.
		List is useful for insertion and deletion operations.	Tuple is useful for readonly operations like accessing elements.
		List iteration is slower and is time consuming.	Tuple iteration is faster.

	d)	**Explain Local and Global variable**
		Local Variables: Local variables are those which are initialized inside a function and belongs only to that particular function. It cannot be accessed anywhere outside the function
		Example:
		`def f():`
		`  # local variable`
		`  s = "I love Python Programming"`
		`  print(s)`
		`# Driver code`
		`f()`
		Output
		`I love Python Programming`
		
		Global Variables: The global variables are those which are defined outside any function and which are accessible throughout the program i.e. inside and outside of every function.
		Example:
		# This function uses global variable s
		def f():
		  print("Inside Function", s)
		
		# Global scope
		s = "I love Python Programming"
		f()
		print("Outside Function", s)
		
		Output:
		Inside Function I love Python Programming
		Outside Function I love Python Programming
			2M (1m each)

	e)	**Define class and object in python** **Class:** A class is a user-defined blueprint or prototype from which objects are created. Classes provide a means of bundling data and functionality together. **Object:** An object is an instance of a class that has some attributes and behavior. Objects can be used to access the attributes of the class.	**2M (Any suitable definition: 1M Each)**

	f)	**How to give single and multiline comment in Python**
		Single line comment: Single-line comments are created simply by beginning a line with the hash (#) character, and they are automatically terminated by the end of line.
		Example:
		# print is a statement
		print('Hello Python')
		
		Multi line comment: Python multi-line comment is a piece of text enclosed in a delimiter (""") Triple quotation marks.
		Example:
		""" Multi-line comment used
		print("Python Comments") """
		or
		To add a multiline comment you could insert a # for each line:
		Example:
		#This is a comment
		#written in
		#more than just one line
		print("Hello, World!")
			2M (1m each)

	g)	**List different modes of opening file in Python** Modes for opening file: • 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. • a: open an existing file for append operation. It won't 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 won't override existing data.	**2M** **(Any 2 names 2M)**
2	**Attempt any THREE of the following**		**12**

	a)	Write a program to print following
		1
		1 2
		1 2 3
		1 2 3 4
		
		for i in range(1,5):
		  for j in range(1,i+1):
		  print(j,end=' ')
		  print()
			4M (for correct program and logic)

	b)	**Explain four Buit-in tuple functions in python with example**	**4M (1M for each function with example)**

Function	Description	Example

cmp(tuple1, tuple2)	Compares elements of both tuples.	>>> tup1=(1,2,3)
		>>> tup2=(1,2,3)
		>>> cmp(tup1,tup2)
		0

len(tuple)	Gives the total length of the tuple.	>>> tup1
		(1, 2, 3)
		>>> len(tup1)
		3

max(tuple)	Returns item from the tuple with max value.	>>> tup1
		(1, 2, 3)
		>>> max(tup1)
		3

min(tuple)	Returns item from the tuple with min value.	>>> tup1
		(1, 2, 3)
		>>> min(tup1)
		1

Count()	Returns the number of times a specified value occurs in a tuple	>>> tup1
		(1, 2, 3, 2, 4)
		>>> tup1.count(2)
		2

Zip(tuple1,tuple2)	It zips elements from two tuples into a list of tuples.	>>> tup1=(1,2,3)
		>>> tup2=('A','B','C')
		>>> tup3=zip(tup1,tup2)
		>>> list(tup3)
		[(1, 'A'), (2, 'B'), (3, 'C')]

Index()	Searches the tuple for a specified value and returns the position of where it was found	>>> tup1
		(1, 2, 3)
		>>> tup1.index(3)
		2

tuple(seq)	Converts a list into tuple.	

c) **Explain how to use user defined function in python with example** — **4M (2m for explanation and 2m for example)**

- In Python, def keyword is used to declare user defined functions.
- The function name with parentheses (), which may or may not include parameters and arguments and a colon:
- An indented block of statements follows the function name and arguments which contains the body of the function.

Syntax:

```
def function_name():
   statements
   .
   .
```

Example:

```
def fun():
     print("User defined function")
fun()
```

output:

User defined function

Parameterized function: The function may take arguments(s) also called parameters as input within the opening and closing parentheses, just after the function name followed by a colon.

Syntax:

```
def function_name(argument1, argument2, ...):
```

		statements
		.
		.
		Example:
		def square(x):
		print("Square=",x*x)
		
		# Driver code
		square(2)
		Output:
		Square= 4	

	d)	**Write a program to create class EMPLOYEE with ID and NAME and display its contents.**
		class employee :
		id=0
		name=""
		def getdata(self,id,name):
		self.id=id
		self.name=name
		def showdata(self):
		print("ID :", self.id)
		print("Name :", self.name)
		
		e = employee()
		e.getdata(11,"Vijay")
		e.showdata()
		
		Output:
		ID : 11
		Name : Vijay
			4M (for correct program and logic)

3	**Attempt any THREE of the following**		**12**
	a)	**List data types used in Python. Explain any two with example** Data types in Python programming includes: • **Numbers:** Represents numeric data to perform mathematical operations. • **String:** Represents text characters, special symbols or alphanumeric data. • **List:** Represents sequential data that the programmer wishes to sort, merge etc. • **Tuple:** Represents sequential data with a little difference from list. • **Dictionary:** Represents a collection of data that associate a unique key with each value. • **Boolean:** Represents truth values (true or false). **1. Integers (int Data Type):** An integer is a whole number that can be positive (+) or negative (−). Integers can be of any length, it is only limited by the memory available.	**4M (2m for list, and 2m for two example)**

Example: For number data types are integers.
>>>a=10
>>>b -10
To determine the type of a variable type() function is used.
>>>type(a)
>>> <class 'int'>

2. Boolean (Bool Data Type: The simplest build-in type in Python is the bool type, it represents the truth values False and True. Internally the true value is represented as 1 and false is 0.
For example
>>>a = 18 > 5
>>>print(a)
True
b=2>3
print(b)
False

3. Floating-Point/Float Numbers (Float Data Type): Floating-point number or Float is a positive or negative number with a fractional part. A floating point number is accurate up to 15 decimal places. Integer and floating points are separated by decimal points. 1 is integer, 1.0 is floating point number.
Example: Floating point number.
x=10.1
type(x)
<class 'float'>

4. Complex Numbers (Complex Data Type): Complex numbers are written in the form, x + yj, where x is the real part and y is the imaginary part.
Example:
Complex number.
>>>x = 3+4j
>>>print(x.real)
3.0
>>>print(x.imag)
4.0

5. String Data Type: String is a collection of group of characters. Strings are identified as a contiguous set of characters enclosed in single quotes (' ') or double quotes (" "). Any letter, a number or a symbol could be a part of the string. Strings are unchangeable (immutable). Once a string is created, it cannot be modified.
Example: For string data type.

```
>>> s1="Hello"   #string in double quotes
>>> s2='Hi'        #string in single quotes
>>> s3="Don't open the door"  #single quote string in double quotes
>>> s4='I said "yipee"'    #double quote string in single quotes
>>>type(s1)
<class 'str'>
```

6. List Data Type: List is an ordered sequence of items. It is one of the most used datatype in Python and is very flexible. List can contain heterogeneous values such as integers, floats, strings, tuples, lists and dictionaries but they are commonly used to store collections of homogeneous objects. The list datatype in Python programming is just like an array that can store a group of elements and we can refer to these elements using a single name. Declaring a list is pretty straight forward. Items separated by commas (,) are enclosed within brackets [ ].

Example: For list.

```
>>> first=[10, 20, 30]           # homogenous values in list
>>> second=["One","Two","Three"] # homogenous values in list
>>> first
[10, 20, 30]
>>> second
['One', 'Two', 'Three']
>>> first + second      # prints the concatenated lists
 [10, 20, 30, 'One', 'Two', 'Three']
```

7. Tuple Data Type: Tuple is an ordered sequence of items same as list. The only difference is that tuples are immutable. Tuples once created cannot be modified. It is defined within parentheses ( ) where items are separated by commas (,).

A tuple data type in python programming is similar to a list data type, which also contains heterogeneous items/elements.

Example: For tuple.

```
>>> a=(10,'abc',1+3j)
>>> a
(10, 'abc', (1+3j))
>>> a[0]
10
>>> a[0]=20
Traceback (most recent call last):
 File "<pyshell#12>", line 1, in <module>
```

8. Dictionary: Dictionary is an unordered collection of key-value pairs. It is the same as the hash table type. The order of elements in a dictionary is undefined, but we can iterate over the following:

o The key
o The value
o The items (key-value pairs) in a dictionary.

When we have the large amount of data, the dictionary data type is used. Items in dictionaries are enclosed in curly braces { } and separated by the comma (,). A colon (:) is used to separate key from value. Values can be assigned and accessed using square braces ([]).

Example: For dictionary data type.

```
>>> dic1={1:"First","Second":2}
>>> dic1
{1: 'First', 'Second': 2}
>>> type(dic1)
<class 'dict'>
>>> dic1[3]="Third"
>>> dic1
{1: 'First', 'Second': 2, 3: 'Third'}
>>> dic1.keys()
dict_keys([1, 'Second', 3])
>>> dic1.values()
dict_values(['First', 2, 'Third'])
>>>
```

b) Explain membership and assignment operators with example

4M: 2m for membership operators and 2m for assignment operators

Membership Operators: The membership operators in Python are used to find the existence of a particular element in the sequence, and used only with sequences like string, tuple, list, dictionary etc. Membership operators are used to check an item or an element that is part of a string, a list or a tuple. A membership operator reduces the effort of searching an element in the list. Python provides 'in' and 'not in' operators which are called membership operators and used to test whether a value or variable is in a sequence.

Sr. No	Operator	Description	Example

1	in	True if value is found in list or in sequence, and false it item is not in list or in sequence	>>> x="Hello World"
			>>> print('H' in x)
			True

2	not in	True if value is not found in list or in sequence, and false	>>> x="Hello World"

		it item is in list or in sequence.	>>> print("Hello" not in x) False

Assignment Operators (Augmented Assignment Operators):

Assignment operators are used in Python programming to assign values to variables. The assignment operator is used to store the value on the right-hand side of the expression on the left-hand side variable in the expression.

For example, a = 5 is a simple assignment operator that assigns the value 5 on the right to the variable a on the left.

There are various compound operators in Python like a += 5 that adds to the variable and later assigns the same. It is equivalent to a = a + 5.

Following table shows assignment operators in Python programming:

Sr. No.	Operator	Description	Example
1	=	Assigns values from right side operands to left side operand.	c = a + b assigns value of a + b into c
2	+=	It adds right operand to the left operand and assign the result to left operand.	c += a is equivalent to c = c + a
3	−=	It subtracts right operand from the left operand and assign the result to left operand.	c −= a is equivalent to c = c − a
4	*=	It multiplies right operand with the left operand and assign the result to left operand.	operand and assign the result to left operand. c *= a is equivalent to c = c * a
5	/=	It divides left operand with the right operand and assign the result to left operand.	c /= a is equivalent to c = c / a
6	%=	It takes modulus using two operands and assign the result to left operand.	c %= a is equivalent to

					c = c % a	
		7	**=	Performs exponential (power) calculation on operators and assign value to the left operand.	c **= a is equivalent to c = c ** a	
		8	//=	Performs exponential (power) calculation on operators and assign value to the left operand.	c //= a is equivalent to c = c // a	

	c)	**Explain indexing and slicing in list with example**
		Indexing: An individual item in the list can be referenced by using an index, which is an integer number that indicates the relative position of the item in the list.
		There are various ways in which we can access the elements of a list some as them are given below:
		1. List Index: We can use the index operator [] to access an item in a list. Index starts from 0. So, a list having 5 elements will have index from 0 to 4.
		Example: For list index in list.
		>>> list1=[10,20,30,40,50]
		>>> list1[0]
		10
		>>> list1[3:] # list[m:] will return elements indexed from mth index to last index
		[40, 50]
		>>>list1[:4] # list[:n] will return elements indexed from first index to n-1th index
		[10, 20, 30, 40]
		>>> list1[1:3] # list[m:n] will return elements indexed from m to n-1.
		[20, 30]
		>>> list1[5]
		Traceback (most recent call last):
		File "<pyshell#71>", line 1, in <module>
		list1[5]
		IndexError: list index out of range
		2. 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.
		Example: For negative indexing in list.
		>>> list2=['p','y','t','h','o','n']
		>>> list2[-1]
		'n'
		>>> list2[-6]
		'p'
		>>> list2[-3:]
				4M: (2m for indexing and 2m for slicing)

		['h', 'o', 'n']
		>>> list2[-7]
		Traceback (most recent call last):
		File "<pyshell#76>", line 1, in <module>
		list2[-7]
		IndexError: list index out of range
		
		List Slicing: Slicing is an operation that allows us to extract elements from units. The slicing feature used by Python to obtain a specific subset or element of the data structure using the colon (:) operator.
		The slicing operator returns a subset of a list called slice by specifying two indices, i.e. start and end.
		Syntax: list_variable[start_index:end_index]
		This will return the subset of the list starting from start_index to one index less than that of the endind
		Example: For slicing list.
		>>> l1=([10,20,30,40,50])
		>>> l1[1:4]
		[20, 30, 40]
		>>>l1[2:5]
		[30,40,50]	

	d)	**Write a program for importing module for addition and subtraction of two numbers**
		calculation.py:
		def add(x,y):
		return (x+y)
		def sub(x,y):
		return (x-y)
		
		operation.py:
		import calculation
		print(calculation.add(1,2))
		print(calculation.sub(4,2))
		Output:
		3
		2
			4M (for correct logic and program)

4	**Attempt any THREE of the following**		**12**
	a)	**Write a program to create dictionary of student the includes their ROLL NO and NAME** **i) Add three students in above dictionary** **ii) Update name='Shreyas' of ROLL NO=2** **iii) Delete information of ROLL NO=1**	**4M (2m for i), 1m for ii) and 1m for iii))**

Ans:

1)

```
>>> dict1={1:"Vijay",2:"Santosh",3:"Yogita"}
>>>print(dict1)
{1: 'Vijay', 2: 'Santosh', 3: 'Yogita'}

ii)
>>>dict1[2]="Shreyas"
>>>print(dict1)
{1: 'Vijay', 2: 'Shreyas', 3: 'Yogita'}

iii)
>>>dict1.pop(1)
 'Vijay'
>>>print(dict1)
{2: 'Shreyas', 3: 'Yogita'}
```

b) Explain decision making statements If-else, if-elif-else with example — **4M (2m for if-else and 2m for if-elif-else)**

The if-else statement: if statements executes when the conditions following if is true and it does nothing when the condition is false. The if-else statement takes care of a true as well as false condition.

Syntax-1:	**Or Syntax-2:**

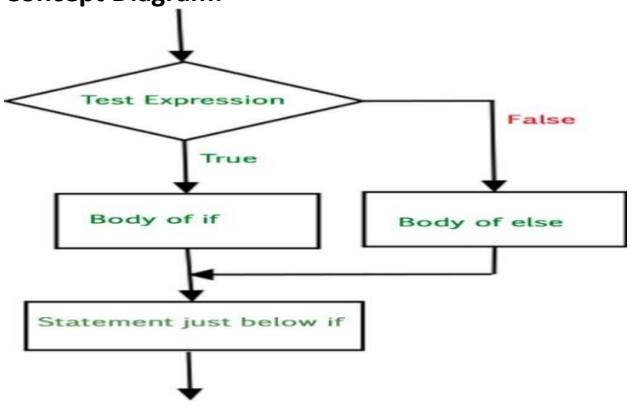
If condition:
  Statement(s)
else:
  Statement(s)
	If condition:
	  If_Block
	else:
	  else_Block

Example:

```
i=20
if(i<15):
   print(" less than 15")
else:
   print("greater than 15")

output:
greater than 15
```

Concept Diagram:



if-elif-else (ladder) statements: Here, a user can decide among multiple options. The if statements are executed from the top down. As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the ladder is bypassed. If none of the conditions is true, then the final else statement will be executed.

Syntax:

```
if (condition-1):
        statement
elif (condition-2):
        statements
.
.
elif(condition-n):
        statements
else:
        statements
```

Example:

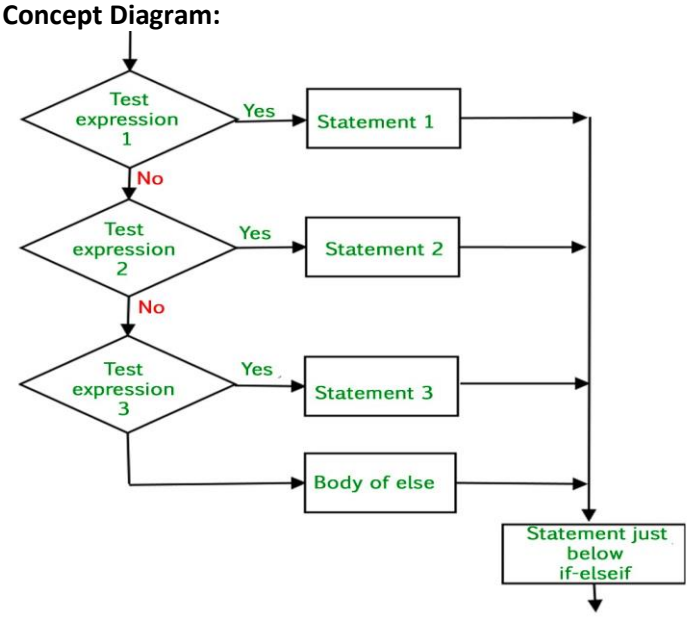
Example:

```
i = 20
if (i == 10):
   print ("i is 10")
elif (i == 15):
   print ("i is 15")
elif (i == 20):
   print ("i is 20")
else:
   print ("i is not present")
```

output:

i is 20

Concept Diagram:



c) **Explain use of format() method with example** — **4M (2m for use and 2m for example)**

The format() method formats the specified value(s) and insert them inside the string's placeholder.
The placeholder is defined using curly brackets: {}.
The format() method returns the formatted string.

Syntax

```
string.format(value1, value2...)
```

Example:

```
#named indexes:
>>>txt1 = ("My name is {fname}, I'm {age}".format(fname =
"abc", age = 36))
>>>print(txt1)
My name is abc, I'm 36

#numbered indexes:
>>>txt2 =( "My name is {0}, I'm {1}".format("xyz",36))
>>>print(txt2)
My name is xyz, I'm 36

#empty placeholders:
>>>txt3 = ("My name is {}, I'm {}".format("pqr",36))
>>>print(txt3)
My name is pqr, I'm 36
```

d) **Explain building blocks of python** — **4M**

Character set: All characters that python can recognize. The below table illustrates the Python character set along with examples.

character Set	Examples
Letters: Upper case and lower case english alphabets	A-Z,a-z
Digits: all digits	0-9
Special symbols	space,+,-,**,*,%,//,/,==,!=,>,<
Whitespaces	Blank space,tabs
Other unicode characters	All ASCII and Unicode characters

Tokens: Tokens in python are building blocks of the Python programming language. The role letters and words play for the English language, Similar to role token play for a python programming language.
Python has the following tokens:
1)keywords
2)identifiers
3)literals
 a)String literals
 b)Numeric literals
 c)Boolean Literals
 d)Special literal None

Tokens	**Example**
Keywords: Words that are already defined and convey a	False,True,if,elif,else,for, while,pass,continue,lambda,

		special meaning to the language compiler/interpreter	return,finally,import,def	
		Identifiers: names given to different parts of program like variables, functions, object, class, names given to different datatypes.	def square,num=20, a_lst=[1,2,3]; here square,num and a_lst are identifiers.	
		Literals/Constants: Data items that have fixed values	String: 'Mayank','abc','anish'; Numeric: 1,1.2,4,-3.95; Boolean: True,False	
		Special literal	None; meaning nothing	

	e)	**Write a program illustrating use of user defined package in python**
		A package is a hierarchical file directory structure that defines a single Python application environment that consists of modules and subpackages and sub-subpackages, and so on.
		Packages allow for a hierarchical structuring of the module namespace using **dot notation**.
		Creating a package is quite straightforward, since it makes use of the operating system's inherent hierarchical file structure.
		Consider the following arrangement:
		 
		
		Here, there is a directory named pkg that contains two modules, `mod1.py` and `mod2.py`. The contents of the modules are:
		mod1.py
		def m1():
		print("first module")
		
		mod2.py
		def m2():
		print("second module")
		
		If the `pkg` directory resides in a location where it can be found, you can refer to the two **modules** with **dot notation**(`pkg.mod1`, `pkg.mod2`) and import them with the syntax:
		
		Syntax-1
		import <module_name>[, <module_name> ...]
		Example:
		>>>import pkg.mod1, pkg.mod2
		>>> pkg.mod1.m1()
		first module
			4M (2m for defining package and 2m for import package in program)

		Syntax-2:
		from <module_name> import <name(s)>
		Example:
		>>> from pkg.mod1 import m1
		>>> m1()
		First module
		>>>
		
		Syntax-3:
		from <module_name> import <name> as <alt_name>
		Example:
		>>> from pkg.mod1 import m1 as module
		>>> module()
		first module
		
		You can import modules with these statements as well:
		from <package_name> import <modules_name>[,
		<module_name> ...]
		from <package_name> import <module_name> as <alt_name>
		Example:
		>>> from pkg import mod1
		>>> mod1.m1()
		First module	

5	**Attempt any TWO of the following**		**12**
	a)	**Write the output of the following** **i) >>> a=[2,5,1,3,6,9,7]** **>>> a[2:6]=[2,4,9,0]** **>>> print(a)** Output: [2, 5, 2, 4, 9, 0, 7] **ii) >>> b=[“Hello”,”Good”]** **>>> b.append(“python”)** **>>>print(b)** Output: ['Hello', 'Good', 'python'] **iii) >>>t1=[3,5,6,7]** **output:** **>>>print(t1[2])** >>>6 **>>>print(t1[-1])** >>>7 **>>>print(t1[2:])** >>>[6, 7] **>>>print(t1[:])** >>>[3, 5, 6, 7]	**6M** **(2m for each)**
	b)	**Explain method overloading in python with example** Method overloading is the ability to define the method with the same name but with a different number of arguments and data types. With this ability one method can perform different tasks, depending on the number of arguments or the types of the arguments given.	**6M (3m for explanation, 3m for example)**

Method overloading is a concept in which a method in a class performs operations according to the parameters passed to it.

As in other languages we can write a program having two methods with same name but with different number of arguments or order of arguments but in python if we will try to do the same we will get the following issue with method overloading in Python:

```
# to calculate area of rectangle
def area(length, breadth):
        calc = length * breadth
        print calc
#to calculate area of square
def area(size):
        calc = size * size
         print calc
area(3)
area(4,5)
```

Output:

```
9
TypeError: area() takes exactly 1 argument (2 given)
```

Python does not support method overloading, that is, it is not possible to define more than one method with the same name in a class in Python.

This is because method arguments in python do not have a type. A method accepting one argument can be called with an integer value, a string or a double as shown in next example.

```
class Demo:
def method(self, a):
      print(a)
obj= Demo()
obj.method(50)
obj.method('Meenakshi')
 obj.method(100.2)
```

Output:

```
50
Meenakshi
 100.2
```

Same method works for three different data types. Thus, we cannot define two methods with the same name and same number of arguments but having different type as shown in the above example. They will be treated as the same method.

It is clear that method overloading is not supported in python but that does not mean that we cannot call a method with different number of arguments. There are a couple of

		alternatives available in python that make it possible to call the same method but with different number of arguments.	

	c)	**Write a program to open a file in write mode and append some content at the end of file**
		file1 = open("myfile.txt", "w")
		L = ["This is Delhi \n", "This is Paris \n", "This is London"]
		file1.writelines(L)
		file1.close()
		
		# Append-adds at last
		# append mode
		file1 = open("myfile.txt", "a")
		
		# writing newline character
		file1.write("\n")
		file1.write("Today")
		
		# without newline character
		file1.write("Tomorrow")
		
		file1 = open("myfile.txt", "r")
		print("Output of Readlines after appending")
		print(file1.read())
		print()
		file1.close()
		
		Output:
		Output of Readlines after appending
		This is Delhi
		This is Paris
		This is London
		TodayTomorrow
			6M for any program with suitable logic

6	**Attempt any TWO of the following**		**12**
	a)	**Explain package Numpy with example** • NumPy is the fundamental package for scientific computing with Python. NumPy stands for "Numerical Python". It provides a high-performance multidimensional array object, and tools for working with these arrays. • An array is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers and represented by a single variable. NumPy's array class is called ndarray. It is also known by the alias array. • In NumPy arrays, the individual data items are called elements. All elements of an array should be of the	**6M (3m for explanation and 3m for example)**

same type. Arrays can be made up of any number of dimensions.

- In NumPy, dimensions are called axes. Each dimension of an array has a length which is the total number of elements in that direction.
- The size of an array is the total number of elements contained in an array in all the dimension. The size of NumPy arrays are fixed; once created it cannot be changed again.
- Numpy arrays are great alternatives to Python Lists. Some of the key advantages of Numpy arrays are that they are fast, easy to work with, and give users the opportunity to perform calculations across entire arrays.
- A one dimensional array has one axis indicated by Axis-0. That axis has five elements in it, so we say it has length of five.
- A two dimensional array is made up of rows and columns. All rows are indicated by Axis-0 and all columns are indicated by Axis-1. If Axis-0 in two dimensional array has three elements, so its length it three and Axis-1 has six elements, so its length is six.

Execute Following command to install numpy in window, Linux and MAC OS:

```
python -m pip install numpy
```

To use NumPy you need to import Numpy:

```
import numpy as np # alias np
```

Using NumPy, a developer can perform the following operations:

1. Mathematical and logical operations on arrays.
2. Fourier transforms and routines for shape manipulation.
3. Operations related to linear algebra.
4. NumPy has in-built functions for linear algebra and random number generation

Example:

For NumPy with array object.

```
>>> import numpy as np
>>> a=np.array([1,2,3]) # one dimensional array
>>> print(a)
[1 2 3]
>>> arr=np.array([[1,2,3],[4,5,6]]) # two dimensional array
>>> print(arr)
[[1 2 3]
 [4 5 6]]
```

		>>> type(arr)
		<class 'numpy.ndarray'>
		>>> print("No. of dimension: ", arr.ndim)
		No. of dimension: 2
		>>> print("Shape of array: ", arr.shape)
		Shape of array: (2, 3)
		>> >print("size of array: ", arr.size)
		size of array: 6
		>>> print("Type of elements in array: ", arr.dtype)
		Type of elements in array: int32
		>>> print("No of bytes:", arr.nbytes)
		No of bytes: 24	

	b)	**Write a program to implement the concept of inheritance in python**
		• In inheritance objects of one class procure the properties of objects of another class.
		• Inheritance provide code usability, which means that some of the new features can be added to the code while using the existing code.
		• The mechanism of designing or constructing classes from other classes is called inheritance.
		• The new class is called derived class or child class and the class from which this derived class has been inherited is the base class or parent class.
		• In inheritance, the child class acquires the properties and can access all the data members and functions defined in the parent class. A child class can also provide its specific implementation to the functions of the parent class.
		Syntax:
		class A:
		# properties of class A
		class B(A):
		# class B inheriting property of class A
		# more properties of class B
		
		Example 1: Inheritance without using constructor.
		class Vehicle: #parent class
		name="Maruti"
		def display(self):
		print("Name= ",self.name)
		class Category(Vehicle): #derived class
		price=2000
		def disp_price(self):
		print("Price=$",self.price)
		car1=Category()
		car1.display()
			6M for any suitable example of inheritance

```
car1.disp_price()
```
Output:
Name= Maruti
Price=$ 2000

Example 2: Inheritance using constructor.

```
class Vehicle:                          #parent class
      def __init__(self,name):
            self.name=name
            def display(self):
            print("Name= ",self.name)
class Category(Vehicle):                #derived class
          def __init__(self,name,price):
              Vehicle.__init__(self,name)
 # passing data to base class constructor
              self.price=price
             def disp_price(self):
             print("Price=$ ",self.price)
car1=Category("Maruti",2000)
car1.display()
car1.disp_price()
car2=Category("BMW",5000)
car2.display()
car2.disp_price()
```
Output:
Name= Maruti
Price=$ 2000
Name= BMW
Price=$ 5000

	c)	**Explain Try-except block used in exception handling in python with example** • In Python, exceptions can be handled using a try statement. A try block consisting of one or more statements is used by programmers to partition code that might be affected by an exception. • A critical operation which can raise exception is placed inside the try clause and the code that handles exception is written in except clause. • The associated except blocks are used to handle any resulting exceptions thrown in the try block. That is we want the try block to succeed and if it does not succeed, we want to control to pass to the catch block. • If any statement within the try block throws an exception, control immediately shifts to the catch block. If no exception is thrown in the try block, the catch block is skipped.	**6M (3m for explanation and 3m for program)**

- There can be one or more except blocks. Multiple except blocks with different exception names can be chained together.
- The except blocks are evaluated from top to bottom in the code, but only one except block is executed for each exception that is thrown.
- The first except block that specifies the exact exception name of the thrown exception is executed. If no except block specifies a matching exception name then an except block that does not have an exception name is selected, if one is present in the code.
- For handling exception in Python, the exception handler block needs to be written which consists of set of statements that need to be executed according to raised exception. There are three blocks that are used in the exception handling process, namely, try, except and finally.

1. try Block: A set of statements that may cause error during runtime are to be written in the try block.
2. except Block: It is written to display the execution details to the user when certain exception occurs in the program. The except block executed only when a certain type as exception occurs in the execution of statements written in the try block.
3. finally Block: This is the last block written while writing an exception handler in the program which indicates the set of statements that many use to clean up to resources used by the program.

Syntax:

```
try:
 D the operations here
 ......................
except Exception1:
 If there is Exception1, then execute this block.
except Exception2:
 If there is Exception2, then execute this block.
 ......................
else:
 If there is no exception then execute this block.
```

Example: For try-except clause/statement.

```
try:
 fh = open("testfile", "w")
 fh.write("This is my test file for exception handling!!")
```

		except IOError: print ("Error: can\'t find file or read data") else: print ("Written content in the file successfully") fh.close()	