WINTER – 2022 EXAMINATION

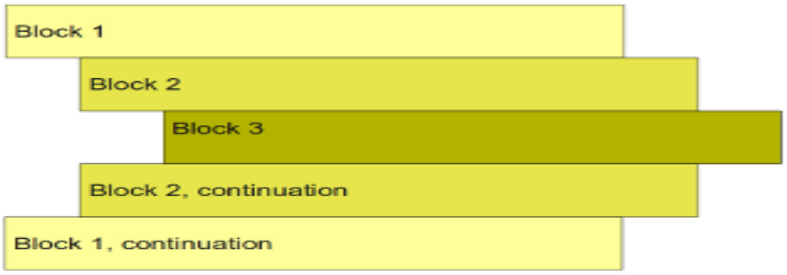
Subject Name: Programming with Python **Model Answer** **Subject Code:** 22616

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.

Q. No.	Sub Q. N.	Answer	Marking Scheme
1		**Attempt any FIVE of the following:**	**10 M**
	a)	**List Python features. (Any four)**	**2 M**
	Ans	• Easy to Learn and Use • Interactive Mode • Expressive Language • Interpreted Language • Cross-platform Language • Portable • Free and Open Source • Object-Oriented Language • Extensible • Large Standard Library • GUI Programming Support • Integrated • Databases • Scalable	2M (1/2 M each) Any Four
	b)	**List comparision operators in Python.**	**2 M**
	Ans	Comparison operators in Python are Operator | Meaning == | Equal to	2M (1M each)

			!= Not Equal to < Less than > Greater than <= Less than and Equal to >= Greater than and Equal to	
	c)	**Describe Tuples in Python.**		**2 M**
	Ans	A tuple is a collection of items which is ordered and **unchangeable**. Tuples are the sequence or series values of different types separated by commas (,). Example: tup1=(10,20,30)		2M for Definition.
	d)	**Write use of lambda function in python.**		**2 M**
	Ans	The lambda function, which is also called anonymous function. A lambda function can take any number of arguments, but can only have one expression. Syntax: lambda arguments : expression Example: x= lambda a,b : a*b Print(x(10,5) Output: 50		2M for use
	e)	**Write syntax of defining class in Python.**		**2 M**
	Ans	class <ClassName>: <statement1> <statement2> . . <statementN>		2M for syntax
	f)	**List file operations in Python.**		**2 M**
	Ans	• Opening file (using open() function) • Reading file (using read() function) • Writing file (using write() function) • Copy files • Delete files (using remove() function) • Closing file (Using close() function)		2M
	g)	**Describe indentation in Python.**		**2 M**
	Ans	Indentation refers to the spaces at the beginning of a code line. Python indentation refers to adding white space before a statement to a particular block of code. In another word, all the statements with the same space to the right, belong to the same code block.		2M



2. **Attempt any THREE of the following:** **12 M**

a) **Describe bitwise operators in Python with example.** **4 M**

Ans Bitwise operators acts on bits and performs bit by bit operation. Assume a=10 (1010) and b=4 (0100) — 4M (for any four, 1M each)

Operator	Meaning	Description	Example
&	Binary AND	This operation performs AND operation between operands. Operator copies a bit, to the result, if it exists in both operands	a &b = 1010 & 0100 = 0000 =0
|	Binary OR	This operation performs OR operation between operands. It copies a bit, if it exists in either operand.	a|b = 1010 | 0100 = 1110 = 14
^	Binary XOR	This operation performs XOR operations between operands. It copies the bit, if it is set in one operand but not both.	a^b=1010 ^ 0100 = 1110 =14
~	Binary Ones Complement	It is unary operator and has the effect of 'flipping' bits i.e. opposite the bits of operand.	~a= ~ 1010 = 0101
<<	Binary Left	The left operand's	a<<2 =

				Shift	value is moved left by the number of bits specified by the right operand.	1010 << 2 =101000 = 40		
			>>	Binary Right Shift	The left operand's value is moved right by the number of bits specified by the right operand.	a>>2 = 1010 >> 2 =0010 = 2		
	b)	**Write any four methods of dictionary.**						**4 M**
	Ans							4M (any four, 1M each)

Method	Description	Example

clear()	Removes all the elements from the dictionary	dict={1:'Vijay',2:'Amar',3:'Santosh'}
		>>> dict
		{1: 'Vijay', 2: 'Amar', 3: 'Santosh'}
		>>> dict.clear()
		>>> dict
		{}

items()	Returns a list containing the a tuple for each key value pair	dict {1: 'Vijay', 2: 'Amar', 3: 'Santosh'} >>> for i in dict.items(): print(i) (1, 'Vijay') (2, 'Amar') (3, 'Santosh')
keys()	Returns a list containing the dictionary's keys	dict={1:'Vijay',2:'Amar',3:'Santosh'} >>> dict.keys() dict_keys([1, 2, 3])
pop()	Removes the element with the specified key	dict={1:'Vijay',2:'Amar',3:'Santosh'} >>> print(dict.pop(2)) Amar
popitem()	Removes the last inserted key-value pair	dict={1:'Vijay',2:'Amar',3:'Santosh'} >>> dict.popitem() (3, 'Santosh')

Method	Description	Example

update()	Updates the dictionary with the specified key-value pairs	>>> dict1
		{2: 'Amar', 4: 'Umesh'}
		>>> dict2={1:'Vijay',3:'Santosh'}
		>>> dict2
		{1: 'Vijay', 3: 'Santosh'}
		>>> dict1.update(dict2)
		>>> dict1
		{2: 'Amar', 4: 'Umesh', 1: 'Vijay', 3: 'Santosh'}

values()	Returns a list of all the values in the dictionary	>>> dict
		{1: 'Vijay', 2: 'Amar', 3: 'Santosh'}
		>>> dict.values()
		dict_values(['Vijay', 'Amar', 'Santosh'])

all()	Return True if all keys of the dictionary are true (or if the dictionary is empty).	>>> dict
		{1: 'Vijay', 2: 'Amar', 3: 'Santosh'}
		>>> all(dict)
		True

any()	Return True if any key of the dictionary is true. If the dictionary is empty, return False.	>>> dict={}
		>>> any(dict)
		False

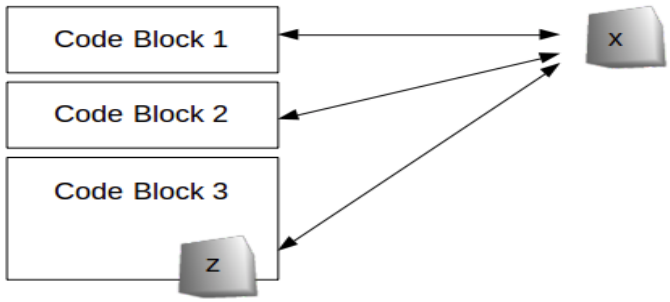
len()	Return the length (the number of items) in the dictionary	>>> dict
		{1: 'Vijay', 2: 'Amar', 3: 'Santosh'}
		>>> len(dict)
		3

c) What is local and global variables? Explain with appropriate example. **4 M**

Ans

- **Global variables:** global variables can be accessed throughout the program body by all functions.
- **Local variables:** local variables can be accessed only inside the function in which they are declared

Concept Diagram:



A global variable (x) can be reached and modified anywhere in the code, local variable (z) exists only in block 3.

Example:

```
g=10                    #global variable g
def test():
   l=20                  #local variable l
   print("local variable=",l)
                         # accessing global variable
   print("Global variable=",g)
```

4M (2M for explanation and 2M for example)

		test() print("global variable=",g) **output:** local variable= 20 Global variable= 10 global variable= 10	
	d)	**Write python program to illustrate if else ladder.**	**4 M**

	Ans	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
		(Similar type of program can consider)
			4M (for correct program and logic)

3.		**Attempt any <u>THREE</u> of the following:**	**12 M**
	a)	**Write basis operations of list.**	**4 M**

	Ans	**1)Accessing values in list:**
		Accessing elements liters from a list in Python is a method to get values that are stared in the list at a particular location or index.
		To access values in lists, use the square brackets for slicing along with the index or indices to obtain value available at that index.
		Example: accessing list values.
		>>> list1 = ["one","two",3,10,"six",20]
		>>> list1[0]
		'one'
		>>> list1[-2]
		'six'
		>>> list1[1:3]
		['two', 3]
		>>> list1[3:]
		[10, 'six', 20]
		>>> list1[:4]
		['one', 'two', 3, 10]
		>>>
			Any two operations:
			
			2 M for each

2) Deleting Values in List

The pop() method in Python is used to remove a particular item/element from the given index in the list. The pop() method removes and returns the last item if index is not provided. This helps us implement lists as stacks (first in, last out data structure).

```
>>> list= [10, 20, 30, 40]
>>> list
[10, 20, 30, 40]
30
>>> list
[10, 20, 40]
>>> list.pop()
40
>>> list
 [10, 30]
```

We can delete one or more items from a list using the keyword del. It can even delete the list entirely. But it does not store the value for further use

```
>>> list= [10, 20, 30, 40]
>>> list
[10, 20, 30, 40]
>>> del (list[1]) # del() with index
>>> list
[10, 30, 40]
>>> del list[2] # del with index
>>> list
[10, 30]
```

The remove() method in Python issued to remove a particular element from the list. We use the remove() method if we know the item that we want to remove or delete from the list (but not the index).

```
>>> list=[10,"one",20,"two"]
>>> list.remove(20)
>>> list
[10, 'one', 'two']
>>> list.remove("one")
>>> list
[10, 'two']
>>>
```

3. Updating Lists:

• List are mutable, meaning their elements can be changed or updated unlike string or tuple.

• Mutability is the ability for certain types of data to be changed without entirely recreating it. Using
mutable data types can allow programs to operate quickly and efficiently.

• Multiple values can be added into list. We can use assignment operator (=) to change an item or a
range of items.
• We can update items of the list by simply assigning the value at the particular index position. We
can also remove the items from the list using remove() or pop() or del statement.

```
>>> list1= [10, 20, 30, 40, 50]
>>> list1
[10, 20, 30, 40, 50]
>>> list1[0]=0 # change 0th index element
>>> list1
[0, 20, 30, 40, 50]
>>> list1[-1]=60 # change last index element
>>> list1
[0, 20, 30, 40, 60]
>>> list1[1]=[5,10] # change 1st index element as sublist
>>> list1
[0, [5, 10], 30, 40, 60]
>>> list1[1:1]=[3,4] # add elements to a list at the desired location
>>> list1
[0, 3, 4, [5, 10], 30, 40, 60]
```

4 Indexing

There are various ways in which we can access the elements of a list.

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:

```
>>> list1=[10,20,30,40,50]
>>> list1[0]
10
>>> list1[4]
50
>>> list1[1:3]
[20, 30]
```

5. List Slicing

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]

Example:

```
>>> l1=([10,20,30,40,50])
>>> l1[1:4]
[20, 30, 40]
```

	b)	**Write Python code for finding greatest among four numbers.**	**4 M**
	Ans	list1 = [ ] num = int(input("Enter number of elements in list: ")) for i in range(1, num + 1): element = int(input("Enter elements: ")) list1.append(element) print("Largest element is:", max(list1)) Output: Enter number of elements in list: 4 Enter elements: 10 Enter elements: 20 Enter elements: 45 Enter elements: 20 Largest element is: 45	Any correct logic program 4 M
	c)	**Illustrate with example method over loading.**	**4 M**

	Ans	• 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.
		• Method overloading is a concept in which a method in a class performs operations according to the parameters passed to it.
		Example: With a method to perform different operations using method overloading.
		class operation:
		def add(self,a,b):
		return a+b
		op1=operation()
		# To add two integer numbers
		print("Addition of integer numbers=",op1.add(10,20))
		# To add two floting point numbers
		print("Addition of integer numbers=",op1.add(11.12,12.13))
		# To add two strings
		print("Addition of integer numbers=",op1.add("Hello","Python"))
		Output:
		Addition of integer numbers= 30
		Addition of integer numbers= 23.25
		Addition of integer numbers= HelloPython
		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.
			Explanation 1 M and Example 3 M

		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	

	d)	**Explain how try-catch block is used for exception handling in python.**	**4 M**
	Ans	• 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. • 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. **Syntax:** try: D the operations here	Proper explanation 4 M

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.

```
n=10
m=0
try:
 n/m
except ZeroDivisionError:
 print("Divide by zero error")
else:
 print (n/m)
```

Output:
Divide by zero error

4. **Attempt any THREE of the following:** **12 M**

a) **Compare list and dictionary. (Any 4 points)** **4 M**

Ans

List	**Dictionary**
List is a collection of index values pairs as that of array in c++.	Dictionary is a hashed structure of key and value pairs.
List is created by placing elements in [ ] separated by commas “, “	Dictionary is created by placing elements in { } as “key”:”value”, each key value pair is separated by commas “, “
The indices of list are integers starting from 0.	The keys of dictionary can be of any data type.
The elements are accessed via indices.	The elements are accessed via key-values.
The order of the elements entered are maintained.	There is no guarantee for maintaining order.

Any four point,
1 M for 1 point

b) **What is command line argument? Write python code to add b) two numbers given as input from command line arguments and print its sum.** **4 M**

	Ans	Python Command line arguments are input parameters passed to the script when executing them. Almost all programming language provide support for command line arguments. Then we also have command line options to set some specific options for the program. There are many options to read python command line arguments. The three most common ones are: Python sys.argv Python getopt module Python argparse module Program: import sys x=int(sys.argv[1]) y=int(sys.argv[2]) sum=x+y print("The addition is :",sum) Output: C:\Python34\python sum.py 6 4 The addition is : 10	1 M for definition and 3 M for program
	c)	**Write python code to count frequency of each characters in a given file.**	**4 M**
	Ans	import collections import pprint file_input = input('File Name: ') with open(file_input, 'r') as info: count = collections.Counter(info.read().upper()) value = pprint.pformat(count) print(value)	Any proper logic program 4 M
	d)	**Write python program to read contents of abc.txt and write same content to pqr.txt.**	**4 M**

	Ans	with open('abs.txt','r') as firstfile, open('prq.txt','w') as secondfile:
		# read content from first file
		for line in firstfile:
		# write content to second file
		secondfile.write(line)
			Any proper logic program for 4 M

5.		**Attempt any <u>TWO</u> of the following:**	**12 M**
	a)	**Write different data types in python with suitable example.**	**6 M**
	Ans	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).

6m for data types

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.
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)	**Example module. How to define module.**	**6 M**
	Ans	A module allows you to logically organize your Python code. Grouping related code into a module makes the code easier to understand and use. A module is a Python object with arbitrarily named attributes that you can bind and reference. Simply, a module is a file consisting of Python code. A module can define functions, classes and variables. A module can also include runnable code. Example The Python code for a module named aname normally resides in a file named aname.py. Here's an example of a simple module, support.py *def print_func(par):* *print "Hello : ", par* *return* To create a module just save the code you want in a file with the file extension .py: Example Save this code in a file named mymodule.py *def greeting(name):* *print("Hello, " + name)* Now we can use the module we just created, by using the import statement:	2 M for module explanation 2 M for creating module 2 M for accessing/using

		Example Import the module named mymodule, and call the greeting function: *import mymodule* *mymodule.greeting("ABC")*	module
	c)	**Write python program to perform following operations on Set (Instead of Tuple)** **i) Create set** **ii) Access set Element** **iii) Update set** **iv) Delete set**	**6 M**

	Ans	# To Create set
		S={10,20,30,40,50}
		
		# To Access Elements from set
		print (S)
		
		#To add element into set using add method
		S.add(60)
		print(S)
		
		#To update set using update method
		S.update(['A','B'])
		print(S)
		
		#To Delete element from Set using discard() method
		S.discard(30)
		print(S)
		
		#To delete element from set using remove() method
		S.remove('A')
		print(S)
		
		#To delete element from set using pop() method
		S.pop()
		print(S)
		
		output:
		{50, 20, 40, 10, 30}
		{50, 20, 40, 10, 60, 30}
		{'B', 50, 20, 'A', 40, 10, 60, 30}
		{'B', 50, 20, 'A', 40, 10, 60}
		{'B', 50, 20, 40, 10, 60}
		{50, 20, 40, 10, 60}
		
		(Any other suitable example can consider)
			6m for any suitable program
			
			(If students attempted with "set" give marks as per marking scheme)
			
			OR
			
			(If students attempted with "Tuple"
			
			Then
			
			2M-create Tuple
			
			2M-Access tuple
			
			2M-delete Tuple)

***If students have attempted by using "Tuple" then**

```
#To create tuple
tuple1=(10,20,30,40,50)
print (tuple1)
#Access tuple values
print (tuple1[1])
print (tuple1[0:3])
# deleting tuple
del tuple1
print (tuple1)

output:
(10, 20, 30, 40, 50)
20
(10, 20, 30)
Traceback (most recent call last):
  File "C:\Users\Vijay Patil\AppData\Local\Programs\Python\Python310\temp.py", line 9, in <module>
    print (tuple1)
NameError: name 'tuple1' is not defined. Did you mean: 'tuple'?
```

6.		**Attempt any <u>TWO</u> of the following:**	**12 M**
	a)	**Explain mutable and immutable data structures.**	**6 M**
	Ans	The data types in Python are divided in two categories: Immutable data types – Values cannot be changed. Immutable data types in Python are 1. Numbers 2. String 3. Tuple Mutable data types – Values can be changed. Mutable data types in Python are: 1. List 2. Dictionaries 3. Sets 1. **Numbers** Python supports integers, floats and complex numbers. An **integer** is a number without decimal point for example 5, 6, 10 etc. A **float** is a number with decimal point for example 6.7, 6.0, 10.99 etc. A **complex number** has a real and imaginary part for example 7+8j, 8+11j etc.	3m for mutable data structure and 3m for immutable data structure

Example:

```
# int
num1 = 10
num2 = 100

# float
a = 10.5
b = 8.9

# complex numbers
x = 3 + 4j
y = 9 + 8j
```

2. **String**

A string is usually a bit of text (sequence of characters). In Python we use ” (double quotes) or ‘ (single quotes) to represent a string.

There are several ways to create strings in Python:

1. We can use ‘ (single quotes), see the string str in the following code.

2. We can use ” (double quotes), see the string str2 in the source code below.

3. Triple double quotes “”” and triple single quotes ”’ are used for creating multi-line strings in Python.

Example:

```
str = 'beginnersbook'
str2 = "Chaitanya"
# multi-line string
str3 = """Welcome to
  Pythonsbook"""

str4 = '''This is a tech
  paper'''
```

3. Tuple

In Python, a tuple is similar to List except that the objects in tuple are immutable which means we cannot change the elements of a tuple once assigned. On the other hand, we can change the elements of a list.

To create a tuple in Python, place all the elements in a () parenthesis, separated by commas. A tuple can have heterogeneous data items, a tuple can have string and list as data items as well.

Example

```
# tuple of strings
my_data = ("hi", "hello", "bye")
```

```
# tuple of int, float, string
my_data2 = (1, 2.8, "Hello World")

# tuple of string and list
my_data3 = ("Book", [1, 2, 3])

# tuples inside another tuple
# nested tuple
my_data4 = ((2, 3, 4), (1, 2, "hi"))
```

4. List

A list is a data type that allows you to store various types data in it. List is a compound data type which means you can have different-2 data types under a list, for example we can have integer, float and string items in a same list.

To create a list all you have to do is to place the items inside a square bracket [] separated by comma ,.`

Example:

```
# list of floats
num_list = [11.22, 9.9, 78.34, 12.0]

# list of int, float and strings
mix_list = [1.13, 2, 5, "beginnersbook", 100, "hi"]

# an empty list
nodata_list = []
```

5. Dictionaries

Dictionary is a mutable data type in Python. A python dictionary is a collection of key and value pairs separated by a colon (:), enclosed in curly braces {}.

Left side of the colon(:) is the key and right side of the : is the value.

```
mydict = {'StuName': 'Ajeet', 'StuAge': 30, 'StuCity': 'Agra'}
```

6. Sets

Set is an unordered and *unindexed* collection of items in Python. Unordered means when we display the elements of a set, it will come out in a random order. Unindexed means, we cannot access the elements of a set using the indexes like we can do in list and tuples.

The elements of a set are defined inside curly brackets and are separated by commas. For example –

```
myset = {1, 2, 3, 4, "hello"}
```

	b)	**Design a class student with data members; Name, roll number address.**	**6 M**

		Create suitable method for reading and printing students details.	

	Ans	class Student:
		def getStudentDetails(self):
		self.rollno=input("Enter Roll Number : ")
		self.name = input("Enter Name : ")
		self.address =input("Enter Address : ")
		def printStudentDetails(self):
		print(self.rollno,self.name, self.address)
		S1=Student()
		S1.getStudentDetails()
		print("Student Details ")
		S1.printStudentDetails ()
		Output:
		Enter Roll Number : 001
		Enter Name : ABC
		Enter Address : New York
		Student Details :
		001 ABC New York
		(Any suitable program can consider)
			2 M for class definition
			2 M to define functions
			2 M to create objects

	c)	**Create a parent class named Animals and a child class Herbivorous which will extend the class Animal. In the child class Herbivorous over side the method feed ( ). Create a object**	**6 M**

	Ans	# parent class
		class Animal:
		# properties
		multicellular = True
		# Eukaryotic means Cells with Nucleus
		eukaryotic = True
		# function breath
		def breathe(self):
		print("I breathe oxygen.")
		# function feed
		def feed(self):
		print("I eat food.")
		# child class
		class Herbivorous(Animal):
			2 M to create parent class
			2 M to define child class
			2 M to create object and call function

```
   # function feed
        def feed(self):
            print("I eat only plants. I am vegetarian.")

herbi = Herbivorous()

herbi.feed()
# calling some other function
herbi.breathe()
```

Output:

```
I eat only plants. I am vegetarian.
I breathe oxygen.
```