SUMMER – 2023 EXAMINATION
Model Answer – Only for the Use of RAC Assessors

Subject Name: Programming with Python **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 features of Python.**	**2 M**
	Ans	Features of Python are listed below: • Easy to Learn and Use • Interpreted Language • Interactive Mode • Free and Open Source • Platform Independence/Cross-platform Language/Portable • Object-Oriented Language • Extensible	Any 2, 1 M each
	b)	**Describe membership operators in python**	**2 M**
	Ans	• 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.	2 M for proper explanation

		• 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.	
	c)	**Write down the output of the following Python code** **>>>indices-['zero','one','two',' three,' four, five']** **i) >>>indices[:4]** **ii) >>>indices[:-2]**	**2 M**
	Ans	**Output as follows:** i) >>>indices[:4] [zero, one, two, three] ii) >>>indices[:-2] [zero, one, two, three]	1 M for each correct output
	d)	**Describe any two data conversion function.**	**2 M**
	Ans	• int(x [,base]): Converts x to an integer. base specifies the base if x is a string. **Example:** x=int('1100',base=2)=12 • long(x [,base]): Converts x to a long integer. base specifies the base if x is a string. **Example:** x=long('123'base=8)=83L • float(x): Converts x to a floating point number. **Example:** x=float('123.45')=123.45 • complex(real[,imag]) : Creates a complex number. **Example:** x=complex(1,2) = (1+2j) • str(x): Converts object x to a string representation. **Example:** x=str(10) = '10' • repr(x): Converts object x to an expression string **Example:** x=repr(3) = 3 • repr(x): Evaluates a string and returns an object. **Example:** x=eval('1+2') = 3 • tuple(s): Converts s to a tuple **Example:** x=tuple('123') = ('1', '2', '3') x=tuple([123]) = (123,) • list(s): Converts s to a list **Example:** x=list('123') = ['1', '2', '3'] x=list(['12'] = ['12'] • set(s): Converts s to a set **Example:** x=set('Python') = {'y', 't', 'o', 'P', 'n', 'h'} • dict(d): Creates a dictionary. d must be a sequence of (key, value) tuples.	Any 2 Conversion function 2 M

		Example: dict={'id':'11','name':'vijay'} print(dict) ={'id': '11', 'name': 'vijay'} • chr(x): Converts an integer to a character. **Example:** x=chr(65) = 'A' • unichr(x): Converts an integer to a Unicode character **Example:** x=unichr(65) =u'A' • ord(x): Converts a single character to its integer value. **Example:** x=ord('A')= 65 • hex(x): Converts an integer to a hexadecimal string. **Example:** x=hex(12) = 0xc • oct(x): Converts an integer to an octal string. **Example:** x=oct(8) = 0o10	
	e)	**With neat example explain default constructor concept in Python.**	**2 M**

	Ans	The default constructor is simple constructor which does not accept any arguments. It's definition has only one argument which is a reference to the instance being constructed.
		
		Example 1: Display Hello message using default constructor.
		
		class Student:
		
		def _ _init_ _(self):
		
		print("This is non parametrized constructor")
		
		def show(self,name):
		
		print("Hello",name)
		
		s1 = Student()
		
		s1.show("Student1")
		
		Output:
		
		This is non parametrized constructor
		Hello Student1
			Explanation 1 M, Example 1 M

	f)	**Describe mkdir() function.**	**2 M**
	Ans	• We can make a new directory using the mkdir() method. • This method takes in the path of the new directory. If the full path is not specified, the new directory is created in the current working directory.	Explanation 2 M

		Syntax: os.mkdir("newdir")
		
		Example:
		
		>>> import os
		
		>>> os.mkdir("testdir")	

	g)	**Describe Multiline comment in python.**	**2 M**
	Ans	• In some situations, multiline documentation is required for a program. If we have comments that extend multiple lines, one way of doing it is to use hash (#) in the beginning of each line. Another way of doing this is to use quotation marks, either ''' or """. • Similarly, when it sees the triple quotation marks ''' it scans for the next ''' and ignores any text in between the triple quotation marks. Example: For multiline comment. '''This is first python program Print is a statement'''	2 M for proper explanation
2.		**Attempt any <u>THREE</u> of the following:**	**12 M**
	a)	**Describe Keyword "continue" with example.**	**4 M**

	Ans	• The continue statement in Python returns the control to the beginning of the while loop.
		• The continue statement rejects all the remaining statements in the current iteration of the loop and moves the control back to the top of the loop.
		
		Syntax: continue
		
		Example: For continue statement.
		
		i=0
		
		while i<10:
		
		i=i+1
		
		if i==5:
		
		continue
		
		print("i= ",i)
			Explanation 2M, Example 2M

Output:

```
i=1
 i=2
 i=3
 i=4
 i=6
 i=7
 i=8
i=9
 i=10
```

b) Explain creating Dictionary and accessing Dictionary Elements with example. — **4 M**

Ans

Creating Dictionary

The simplest method to create dictionary is to simply assign the pair of key:values to the dictionary using operator (=).

• There are two ways for creation of dictionary in python.

1. We can create a dictionary by placing a comma-separated list of key:value pairs in curly braces{}. Each key is separated from its associated value by a colon:

Example: For creating a dictionary using { }.

```
>>> dict1={} #Empty dictionary
>>> dict1
{}
>>> dict2={1:"Orange", 2:"Mango", 3:"Banana"} #Dictionary with integer keys
>>> dict2
{1: 'Orange', 2: 'Mango', 3: 'Banana'}
>>> dict3={"name":"vijay", 1:[10,20]}    #Dictionary with mixed keys
>>> dict3
{'name': 'vijay', 1: [10, 20]}
```

2. Python provides a build-in function dict() for creating a dictionary

Example: Creating directory using dict().

```
>>> d1=dict({1:"Orange",2:"Mango",3:"Banana"})
>>> d2=dict([(1,"Red"),(2,"Yellow"),(3,"Green")])
>>> d3=dict(one=1, two=2, three=3)
>>> d1
{1: 'Orange', 2: 'Mango', 3: 'Banana'}
>>> d2
{1: 'Red', 2: 'Yellow', 3: 'Green'}
>>> d3
{'one': 1, 'two': 2, 'three': 3}
```

Creating Dictionary explanation with example 2 M, Accessing Dictionary Element with example 2 M

Accessing Values in a Dictionary

• We can access the items of a dictionary by following ways:

1. Referring to its key name, inside square brackets([]).

Example: For accessing dictionary items [ ] using.

```
>>> dict1={'name':'vijay','age':40}
>>> dict1['name']
'vijay'
>>> dict1['adr']
Traceback (most recent call last):
 File "<pyshell#79>", line 1, in <module>
     dict1['adr']
KeyError: 'adr'
 >>>
```

Here, if we refer to a key that is not in the dictionary, you'll get an exception. This error can be avoided by using get() method.

2. Using get() method returns the value for key if key is in the dictionary, else None, so that this method never raises a KeyError.

Example: For accessing dictionary elements by get().

```
>>> dict1={'name':'vijay','age':40}
>>> dict1.get('name')
'vijay'
```

	c)	**Explain any four Python's Built-in Function with example.**	**4 M**

	Ans	• **len(list)**
		It returns the length of the list.
		Example:
		`>>> list1`
		`[1, 2, 3, 4, 5]`
		`>>> len(list1)`
		`5`
		• **max(list)**
		It returns the item that has the maximum value in a list
		Example:
		`>>> list1`
		`[1, 2, 3, 4, 5]`
		`>>> max(list1)`
		`5`
		• **sum(list)**
		Calculates sum of all the elements of list.
		Example:
		`>>>list1`
		`[1, 2, 3, 4, 5]`
		`>>>sum(list1)`
			Any 4 Built-in function with example 4 M

```
15
```

- **min(list)**
  It returns the item that has the minimum value in a list.
  Example:

```
>>> list1
[1, 2, 3, 4, 5]
>>> min(list1)
1
```

- **list(seq)**
  It converts a tuple into a list.
  Example:

```
>>> list1
[1, 2, 3, 4, 5]
>>> list(list1)
[1, 2, 3, 4, 5]
```

- **abs(*n*)**
  It returns the absolute value of a number.
  Example:

```
>>> abs(10)
10
```

- **all()**
  The all() function returns True if all items in an iterable are true, otherwise it returns False.
  Example:

```
>>> x=[True, True, True]
>>> all(x)
True
```

- **any()**
  The any() function returns True if any item in an iterable are true, otherwise it returns False. If the iterable object is empty, the any() function will return False.
  Example:

```
>>> x=[True, False, True]
>>> any(x)
True
```

- **bin()**
  The bin() function returns the binary version of a specified integer. The result will always start >>> bin(10)
  Example:

		'0b1010'
		with the prefix 0b.
		
		• **bool()**
		The bool() function returns the boolean value of a specified object.
		Example:
		>>> bool(1)
		True
		
		• **exp()**
		The method exp() returns returns exponential of x: ex.
		x: This is a numeric expression.
		
		Example:
		>>> math.exp(1)
		2.718281828459045
		>>>	

	d)	**Write a Python program to find the factorial of a number provided by the user.**	**4 M**

	Ans	num=int(input("Enter Number:"))
		fact=1
		if num< 0:
		print("Sorry, factorial does not exist for negative numbers")
		elif num == 0:
		print("The factorial of 0 is 1")
		else:
		for i in range(1,num + 1):
		fact=fact*i
		print("The factorial of ",num," is ",fact)
		
		Output:
		Enter Number: 5
		The factorial of 5 is 120
			Correct program
			4 M

3.		**Attempt any <u>THREE</u> of the following:**	**12 M**
	a)	**Write a python program to input any two tuples and interchange the tuple variables.**	**4 M**
	Ans	def interchange_tuples(tup1, tup2):	Any correct programming

		new_tup1 = tup2
		new_tup2 = tup1
		return new_tup1, new_tup2
		# Input two tuples
		tuple1 = tuple(input("Enter the elements of the first tuple (separated by commas): ").split(","))
		tuple2 = tuple(input("Enter the elements of the second tuple (separated by commas): ").split(","))
		# Interchange the tuples
		result_tuple1, result_tuple2 = interchange_tuples(tuple1, tuple2)
		# Display the result
		print("Interchanged tuples:")
		print("Tuple 1:", result_tuple1)
		print("Tuple 2:", result_tuple2)
		output:
		Enter the elements of the first tuple (separated by commas): 10,20
		Enter the elements of the second tuple (separated by commas): 30,40
		Interchanged tuples:
		Tuple 1: ('30', '40')
		Tuple 2: ('10', '20')
			logic 4 M

	b)	**Explain Bitwise operator in Python with appropriate example.**	**4 M**

	Ans	Bitwise operators available in Python:
		1) Bitwise AND (&): Performs a bitwise AND operation on the corresponding bits of two numbers. Each bit of the output is 1 if the corresponding bits of both operands are 1; otherwise, it is 0.
		Example:
		a = 10 # binary: 1010
		b = 6 # binary: 0110
			Any four operator (Each for 1 M)

```
result = a & b

print(result)  # Output: 2 (binary: 0010)
```

2) Bitwise OR (|): Performs a bitwise OR operation on the corresponding bits of two numbers. Each bit of the output is 0 if the corresponding bits of both operands are 0; otherwise, it is 1.

Example:

```
a = 10  # binary: 1010

b = 6   # binary: 0110

result = a | b

print(result)  # Output: 14 (binary: 1110)
```

3) Bitwise XOR (^): Performs a bitwise XOR (exclusive OR) operation on the corresponding bits of two numbers. Each bit of the output is 1 if the corresponding bits of the operands are different; otherwise, it is 0.

Example:

```
a = 10  # binary: 1010

b = 6   # binary: 0110

result = a ^ b

print(result)  # Output: 12 (binary: 1100)
```

4) Bitwise NOT (~): Performs a bitwise NOT operation on a single operand, which inverts all the bits. It returns the complement of the given number.

Example:

```
a = 10  # binary: 1010

result = ~a

print(result)  # Output: -11 (binary: -1011)
```

5) Bitwise left shift (<<): Shifts the bits of the left operand to the left by a specified number of positions. Zeros are shifted in from the right side.

Example:

```
a = 10  # binary: 1010

result = a << 2

print(result)  # Output: 40 (binary: 101000)
```

6) Bitwise right shift (>>): Shifts the bits of the left operand to the right by a

		specified number of positions. Zeros are shifted in from the left side.
		Example:
		a = 10 # binary: 1010
		result = a >> 2
		print(result) # Output: 2 (binary: 10)	

	c)	**With neat example differentiate between readline () and readlines ( ) functions in file-handling.**	**4 M**

	Ans	readline(): This method reads a single line from the file and returns it as a string. It moves the file pointer to the next line after reading. If called again, it will read the subsequent line.
		# Open the file in read mode
		file = open("example.txt", "r")
		# Read the first line
		line1 = file.readline()
		print(line1)
		
		# Read the second line
		line2 = file.readline()
		print(line2)
		
		# Close the file
		file.close()
		readlines(): This method reads all lines from the file and returns them as a list of strings. Each line is an individual element in the list. It reads the entire file content and stores it in memory.
		Example:
		# Open the file in read mode
		file = open("example.txt", "r")
		
		# Read all lines
			readline() explanation for 1 M and Example for 1 M
			and
			readlines() explanation for 1 M and Example for 1 M

		lines = file.readlines()
		
		# Close the file
		file.close()
		# Print each line
		for line in lines:
		  print(line)	

	d)	**Describe 'Self Parameter with example.**	**4 M**

	Ans	In Python, the self parameter is a convention used in object-oriented programming (OOP) to refer to the instance of a class within the class itself. It allows you to access the attributes and methods of the class from within its own methods. The name self is not a keyword in Python, but it is widely adopted and recommended as a convention.
		Example:
		class Car:
		  def __init__(self, make, model, year):
		  self.make = make
		  self.model = model
		  self.year = year
		
		  def get_info(self):
		  info = f"Make: {self.make}, Model: {self.model}, Year: {self.year}"
		  return info
		
		  def start_engine(self):
		  print("Engine started!")
		
		# Create an instance of the Car class
		my_car = Car("Toyota", "Corolla", 2022)
			Explanation 2 M and example 2 M

```
# Access the attributes using the self parameter
print(my_car.make)   # Output: Toyota
print(my_car.model)  # Output: Corolla
print(my_car.year)   # Output: 2022

# Call the method using the self parameter
car_info = my_car.get_info()
print(car_info)  # Output: Make: Toyota, Model: Corolla, Year: 2022

# Call the method that does not require any additional parameters
my_car.start_engine()  # Output: Engine started!
```

4. **Attempt any <u>THREE</u> of the following:** **12 M**

a) **Differentiate between list and Tuple.** **4 M**

Ans Any 4 correct point 4 M

List	Tuple
Lists are mutable	Tuples are immutable
The implication of iterations is Time-consuming	The implication of iterations is comparatively Faster
The list is better for performing operations, such as insertion and deletion.	A Tuple data type is appropriate for accessing the elements
Lists consume more memory	Tuple consumes less memory as compared to the list
Lists have several built-in methods	Tuple does not have many built-in methods.
Unexpected changes and errors are more likely to occur	In a tuple, it is hard to take place.

b) **Explain any four file modes in Python.** **4 M**

Ans 1 Mode for 1 M

Sr. No.	Mode	Description
1	r	Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.
2	rb	Opens a file for reading only in binary format. The file pointer

				is placed at the beginning of the file. This is the default mode.	
			3	r+	Opens a file for both reading and writing. The file pointer placed at the beginning of the file.
			4	rb+	Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file.
			5	w	Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
			6	wb	Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing
			7	w+	Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
			8	wb+	Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing
			9	a	Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
			10	ab	Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing
			11	a+	Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.
			12	ab+	Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.
			13	t	Opens in text mode (default).
			14	b	Opens in binary mode.
			15	+	Opens a file for updating (reading and writing).
	c)	**Write a program to show user defined exception in Python.**			**4 M**

	Ans	class MyException(Exception):
		def __init__(self, message):
		self.message = message
		
		# Function that raises the custom exception
		def divide_numbers(a, b):
					Any correct logic program 4 M

```
    if b == 0:
        raise MyException("Division by zero is not allowed!")
    return a / b


# Main program
try:
    num1 = int(input("Enter the numerator: "))
    num2 = int(input("Enter the denominator: "))


    result = divide_numbers(num1, num2)
    print("Result:", result)


except MyException as e:
    print("Exception:", e.message)
```

Note: Any correct program of user defined exception can be considered.

Output:

```
Enter the numerator: 10
Enter the denominator: 0
Exception: Division by zero is not allowed!
Enter the numerator: 10
Enter the denominator: 5
Result: 2.0
```

	d)	**Explain Module and its use in Python.**	**4 M**
	Ans	Modules are primarily the (.py) files which contain Python programming code defining functions, class, variables, etc. with a suffix .py appended in its file name. • A file containing .py python code is called a module. • If we want to write a longer program, we can use file where we can do editing, correction. This is known as creating a script. As the program gets longer, we may want to split it into several files for easier maintenance. • We may also want to use a function that you've written in several programs without	Explanation 2 M and use 2 M

		copying its definition into each program. • In Python we can put definitions in a file and use them in a script or in an interactive instance of the interpreter. Such a file is called a module. **Use of module in python :** **Code organization:** Modules allow you to organize related code into separate files, making it easier to navigate and maintain large projects. **Code reusability:** Modules can be imported and reused in multiple programs, enabling code reuse and reducing duplication. **Encapsulation:** Modules provide a way to encapsulate code and hide the implementation details, allowing users to focus on the functionality provided by the module. **Name spacing:** Modules help avoid naming conflicts by providing a separate namespace for the names defined within the module.	
5.		**Attempt any <u>TWO</u> of the following:**	**12 M**
	a)	**Write a Python Program to check if a string is palindrome Or not.**	**6 M**

	Ans	def is_palindrome(string):
		# Remove whitespace and convert to lowercase
		string = string.replace(" ", "").lower()
		# Reverse the string
		reversed_string = string[::-1]
		# Check if the original and reversed strings are the same
		if string == reversed_string:
		return True
		else:
		return False
		# Test the function
		input_string = input("Enter a string: ")
		if is_palindrome(input_string):
		print("The string is a palindrome.")
		else:
			Any correct logic program 6M.

		print("The string is not a palindrome.") **output:** Enter a string: madam The string is a palindrome. Enter a string: abc The string is not a palindrome.	
	b)	**Write a Python program to calculate sum of digit of given number using function.**	**6 M**

	Ans	def calculate_digit_sum(number):
		# Convert the number to a string
		num_str = str(number)
		# Initialize a variable to store the sum
		digit_sum = 0
		# Iterate over each character in the string
		for digit in num_str:
		# Convert the character back to an integer and add it to the sum
		digit_sum += int(digit)
		# Return the sum of the digits
		return digit_sum
		# Test the function
		input_number = int(input("Enter a number: "))
		sum_of_digits = calculate_digit_sum(input_number)
		print("Sum of digits:", sum_of_digits)
		output:
		Enter a number: 123
		Sum of digits: 6
			Any correct logic program 6M.

	c)	**Write a Python Program to accept values from user in a list and find the largest number and smallest number in a list.**	**6 M**
	Ans	list = []	Any correct logic

		num = int(input('How many numbers: ')) for n in range(num): numbers = int(input('Enter number ')) list.append(numbers) print("Maximum element in the list is :", max(list), "\nMinimum element in the list is :", min(list)) output: How many numbers: 5 Enter number 10 Enter number 20 Enter number 30 Enter number 40 Enter number 50 Maximum element in the list is : 50 Minimum element in the list is : 10	program 6M
6.		**Attempt any <u>TWO</u> of the following:**	**12 M**
	a)	**Explain any six set function with example.**	**6 M**
	Ans	1) union():Return a new set containing the union of two or more sets Example: set1 = {1, 2, 3} set2 = {3, 4, 5} union_set = set1.union(set2) print(union_set) # Output: {1, 2, 3, 4, 5} 2) Intersection: Intersection operation performed on two sets returns all the elements which are common or in both the sets. Example:	1 function for 1 M each

```
set1 = {1, 2, 3}

set2 = {2, 3, 4}

intersection_set = set1.intersection(set2)

print(intersection_set)  # Output: {2, 3}
```

3) Difference:

Difference operation on two sets set1 and set2 returns all the elements which are present on set1 but not in set2.

Example:

```
set1 = {1, 2, 3, 4, 5}

set2 = {3, 4}

difference_set = set1.difference(set2)

print(difference_set)  # Output: {1, 2, 5}
```

4) add(element):

This function adds an element to a set.

Example:

```
fruits = {"apple", "banana", "cherry"}

fruits.add("orange")

print(fruits)  # Output: {'apple', 'banana', 'cherry', 'orange'}
```

5) remove(element):

This function removes an element from a set.

Example:

```
numbers = {1, 2, 3, 4, 5}

numbers.remove(3)

print(numbers)  # Output: {1, 2, 4, 5}
```

6) clear():

This function removes all elements from a set, making it an empty set.

Example:

```
numbers = {1, 2, 3, 4, 5}
```

```
numbers.clear()

print(numbers)  # Output: set()
```

7) isdisjoint():
The isdisjoint() method in Python's set class is used to check whether two sets have any common elements. It returns True if the sets are disjoint (i.e., they have no common elements), and False otherwise.
Example:

```
# Example 1
set1 = {1, 2, 3, 4}
set2 = {5, 6, 7}
set3 = {3, 4, 5}

print(set1.isdisjoint(set2))  # True, no common elements
print(set1.isdisjoint(set3))  # False, both sets have elements 3 and 4


# Example 2
fruits = {"apple", "banana", "orange"}
colors = {"red", "green", "blue"}

print(fruits.isdisjoint(colors))  # True, no common elements


# Example 3
setA = {1, 2, 3}
setB = {4, 5, 6}

print(setA.isdisjoint(setB))  # True, no common elements
```

8) pop():method in Python's set class is used to remove and return an arbitrary element from the set. Since sets are unordered collections, there is no guarantee on which element will be popped.
Example:

```
fruits = {"apple", "banana", "orange", "grape"}

# Remove and return an arbitrary element from the set
popped_element = fruits.pop()

print(popped_element)  # Output: an arbitrary element from the set
print(fruits)  # Output: the modified set after popping an element
```

9) update():The update() method in Python's set class is used to update a set by adding elements from another iterable or set. It modifies the set in place by adding all the elements from the iterable or set specified.
Example:

```
set1 = {1, 2, 3}
set2 = {3, 4, 5}
```

		set1.update(set2) print(set1) # Output: {1, 2, 3, 4, 5}	
	b)	**Design a class student with data members : name, roll no., department, mobile no. Create suitable methods for reading and printing student information.**	**6 M**

	Ans	class Student:
		def __init__(self):
		self.name = ""
		self.roll_no = ""
		self.department = ""
		self.mobile_no = ""
		
		def read_student_info(self):
		self.name = input("Enter student name: ")
		self.roll_no = input("Enter roll number: ")
		self.department = input("Enter department: ")
		self.mobile_no = input("Enter mobile number: ")
		
		def print_student_info(self):
		print("Student Information:")
		print("Name:", self.name)
		print("Roll Number:", self.roll_no)
		print("Department:", self.department)
		print("Mobile Number:", self.mobile_no)
		
		# Create an instance of the Student class
		student = Student()
			Any correct logic program 6 M

		# Read and set student information
		student.read_student_info()
		
		# Print student information
		student.print_student_info()
		output:
		Enter student name: raj
		Enter roll number: 11
		Enter department: computer
		Enter mobile number: 123456
		Student Information:
		Name: raj
		Roll Number: 11
		Department: computer
		Mobile Number: 123456	

	c)	**With suitable example explain inheritance in Python.**	**6 M**

	Ans	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
			Explanation 3 M and Correct example 3 M

Example:

```
# Base class
class Animal:
    def __init__(self, name):
        self.name = name

    def speak(self):
        print("Animal speaks")

# Derived class inheriting from Animal
class Dog(Animal):
    def speak(self):
        print("Dog barks")

# Derived class inheriting from Animal
class Cat(Animal):
    def speak(self):
        print("Cat meows")

# Create instances of derived classes
dog = Dog("Buddy")
cat = Cat("Whiskers")

# Call the speak method of the derived classes
dog.speak()  # Output: Dog barks
cat.speak()  # Output: Cat meows
```