Godavari Institute of Management & Research, Jalgaon
Master of Computer Application (MCA)

Name: _ Roll No: _ _ _ _ _

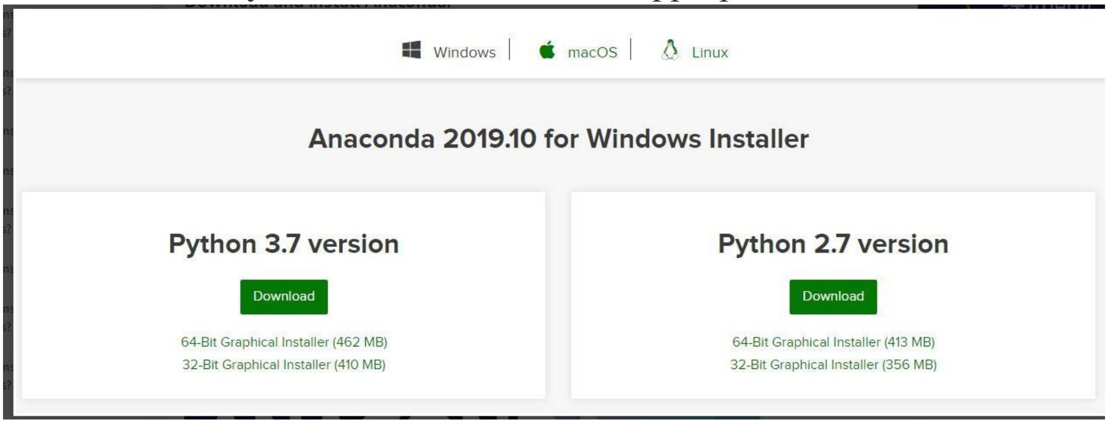
Date of Performance: _ _ /_ _ /20_ _ Batch: _ _ _ _ _ _ _

Class: M.C.A. (I) Practical no:_ _ Sign. of Teacher :_ _ _ _ _ _ _

Subject: 416 Lab on Python Programming

1. Introduction to Python Programming:
a. Python Step-by-step Installation Process.

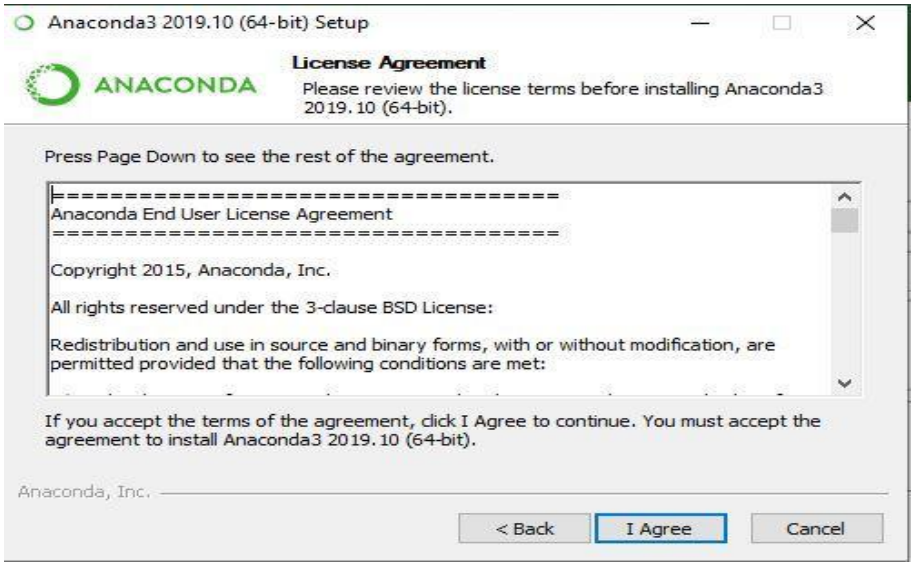
Step 1: Go to anaconda.com and install the latest version of Anaconda. Make sure to download the "Python 3.7 Version" for the appropriate architecture.



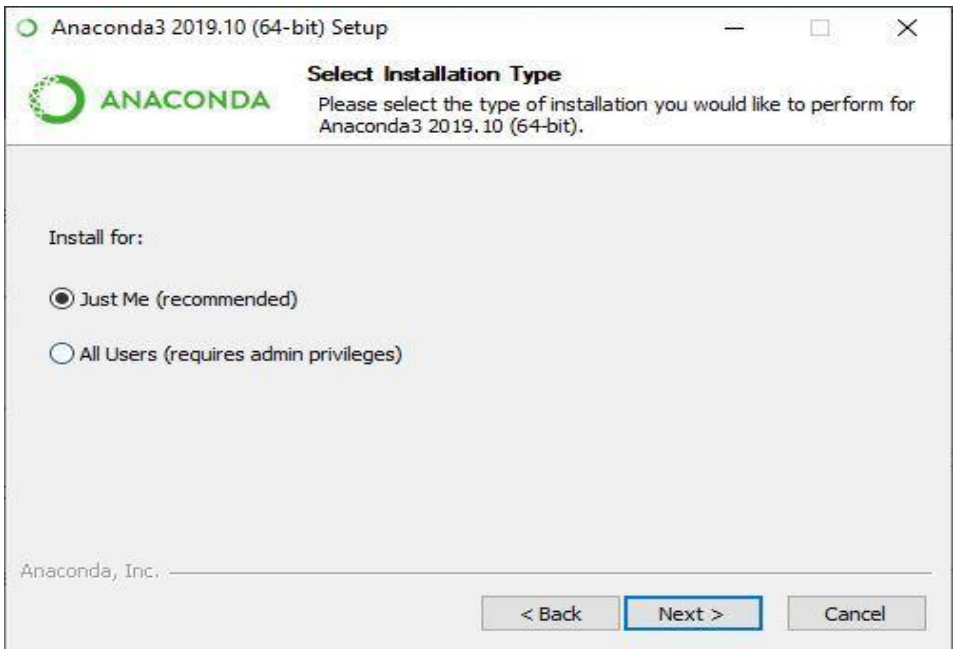
Step 2: Double-click on the executable file.
To get the installation of Anaconda started on your operating system open the executable file in your Download folder.

Begin with the installation process:

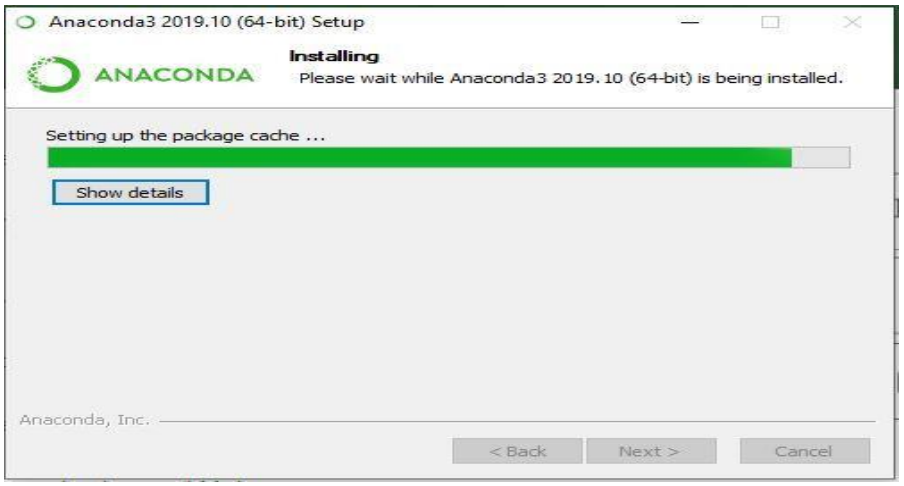
Getting through the License Agreement:



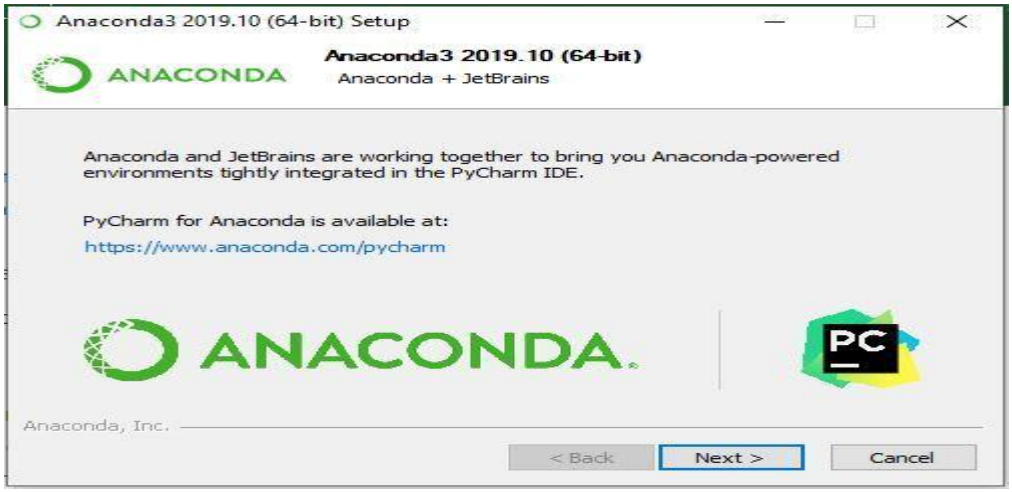
Select Installation Type: Select **Just Me** if you want the software to be used by a single User



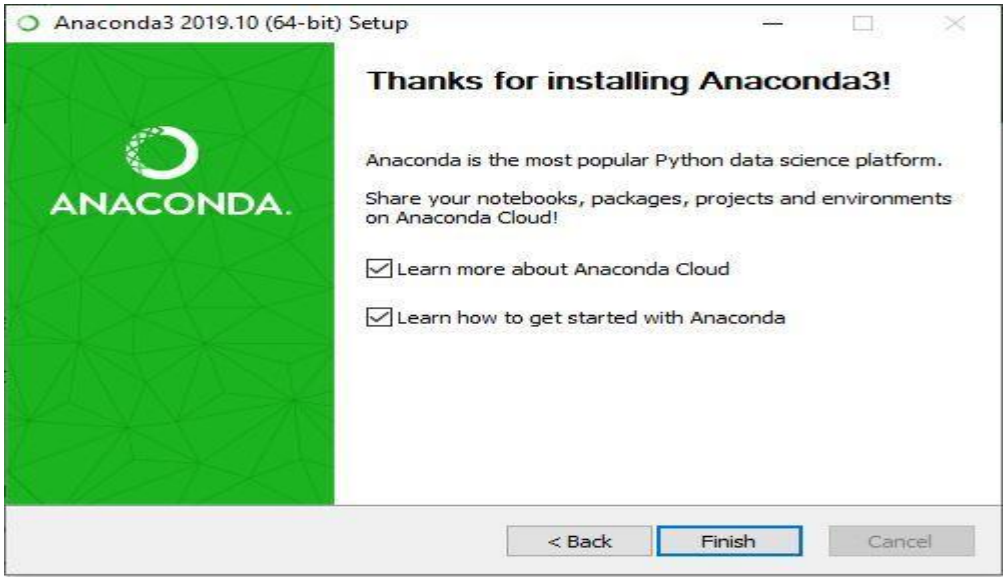
Choose Installation Location:
Getting through the Installation Process:



Recommendation to Install Pycharm:

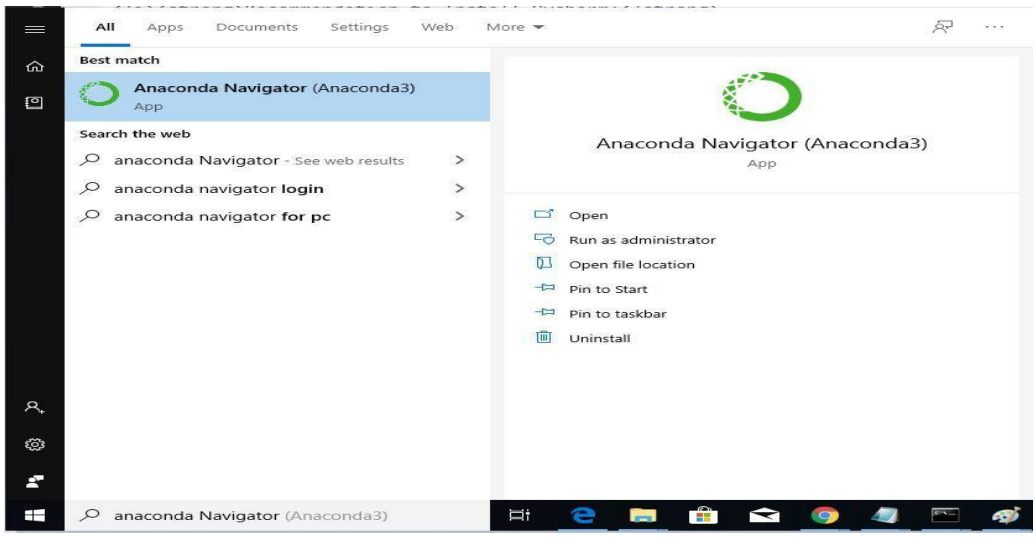


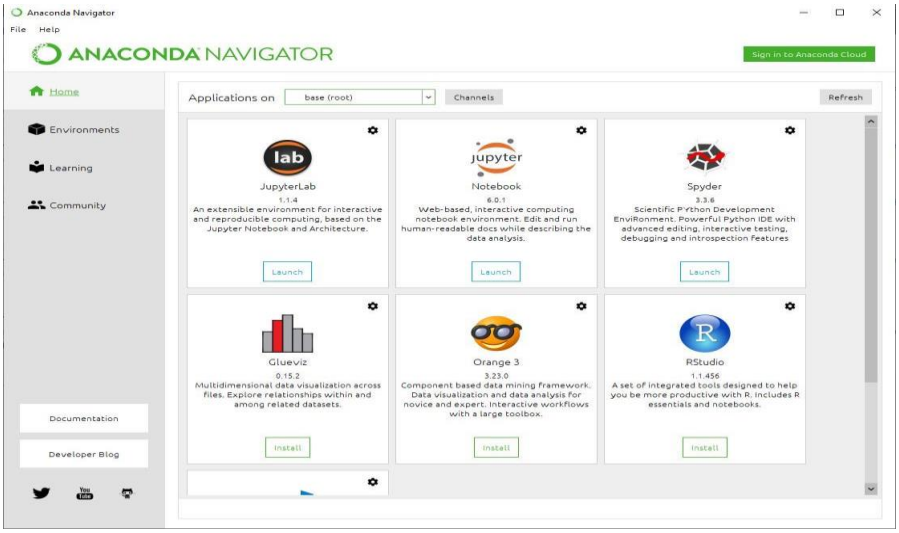
Finishing up the Installation:



Working with Anaconda:

Once the installation process is done, Anaconda can be used to perform multiple operations. To begin using Anaconda, search for Anaconda Navigator from the Start Menu in Windows





To know if you have Python Installed.

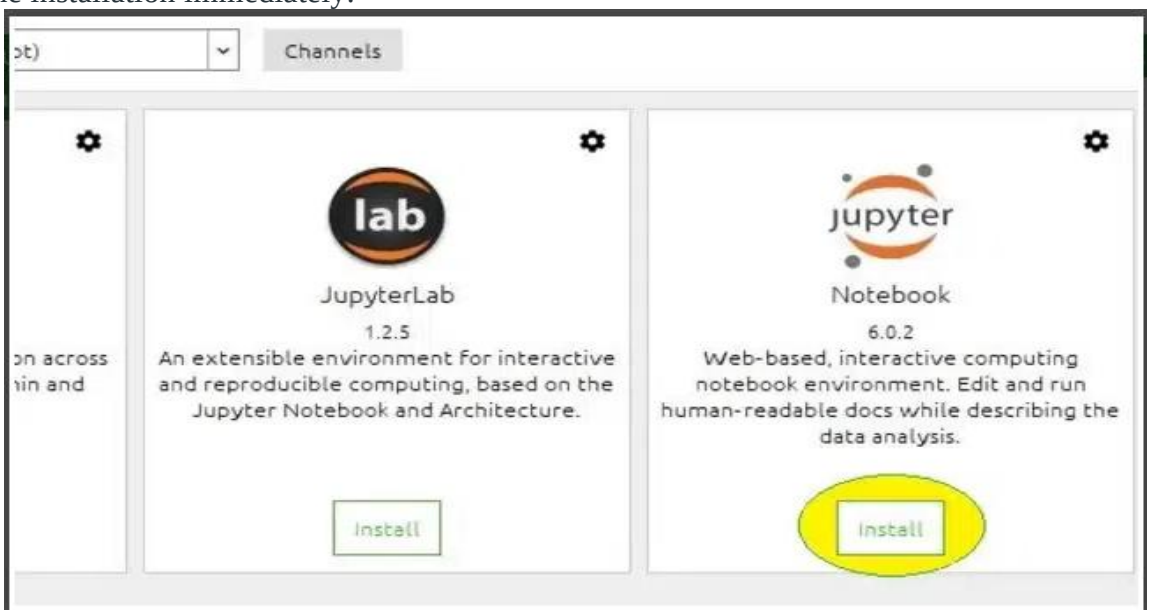
1. Go to Start Menu and type “Command Prompt” to open it.

2. Type the following command and hit the Enter key “python --version”

3. If nothing happens, you don’t have Python installed.

Now install Jupyter Notebook:

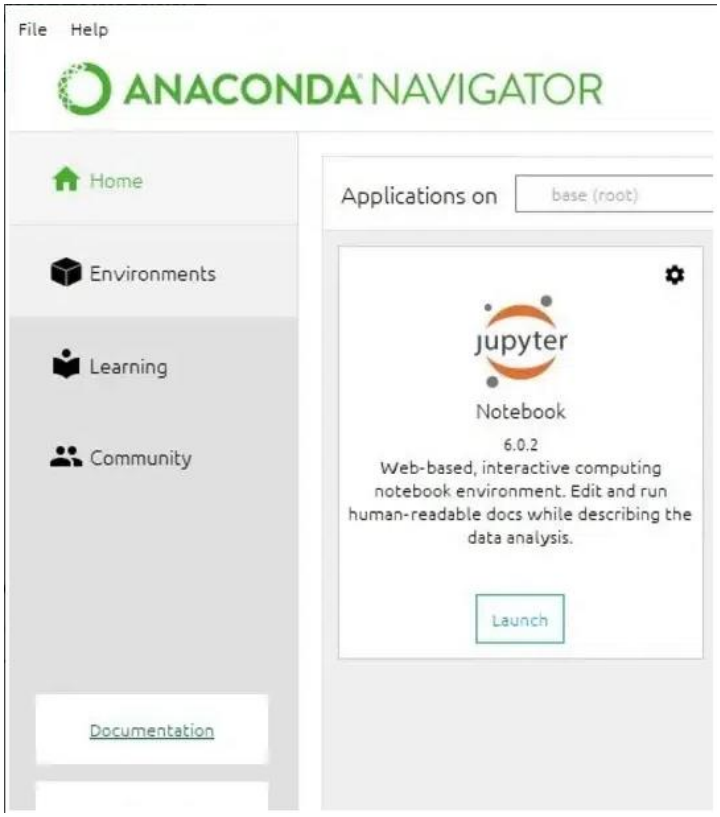
Step 1: Go to Anaconda Navigator
Firstly, Launch anaconda and click on the Install Jupyter Notebook Button.
Step 2: Install Jupyter Notebook
Look up for “Jupyter Notebook”, once yu find it, click on the “Install” button. This will initiate the installation immediately.



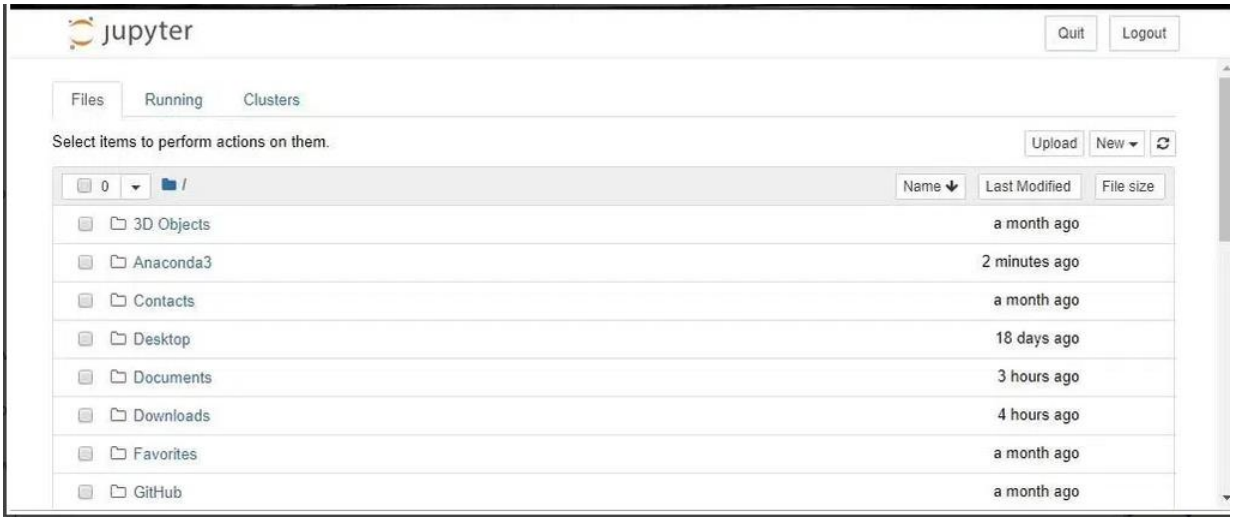
Step 3: Load Packages: Once the installation is complete, it will start loading the packages that comes along with it and click to finish the Installation.

Step 4: Launch Jupyter Notebook

Now, click on Launch button to Launch the Jupyter.



Now, the Jupyter Notebook will launch automatically in your default web browser.



Godavari Institute of Management & Research, Jalgaon
Master of Computer Application (MCA)

Name: _ Roll No: _ _ _ _ _

Date of Performance: _ _ /_ _/20_ _ Batch: _ _ _ _ _ _

Class: M.C.A. (I) Practical no:_ _ Sign. of Teacher :_ _ _ _ _ _

Subject: 416 Lab on Python Programming

---

b. Implement a program based on strings.(slicing, casting, string function)

- **String Slicing:**
  String slicing allows you to extract a portion of a string by specifying a range of indices.
- **String Casting:**
  Casting means converting data of one type to a string
- **String Functions:**

len() Returns the length of the string.

lower() Converts all characters to lowercase.

upper() Converts all characters to uppercase.

strip() Removes leading and trailing whitespace (or characters if specified).

startswith(prefix) Checks if the string starts with the given prefix.

endswith(suffix) Checks if the string ends with the given suffix.

capitalize() Converts the first character to uppercase and the rest to lowercase.

```
string_name=input("Enter a string")
print("string name is: ",string_name)
#string slicing
sliced_string1=string_name[:4]#print string character from index number 0 to 3
print("sliced_string1 :",sliced_string1)
sliced_string2=string_name[1:5]#print string character from index number 1 to 4
print("sliced_string2",sliced_string2)
sliced_string3=string_name[2:4]#print string character from index number 2 to 3
print("sliced_string3",sliced_string3)
```

```
#string casting
num=10
print(num)
print("datatype of num before casting:",type(num))
num_to_str=str(num)#covert int to string
print("datatype of num after casting:",type(num_to_str))
num2=10.15
print(num2)
print("datatype of num2 before casting:",type(num2))
num2_to_str=str(num2)#convert flost to string
print("datatype of num2 after casting:",type(num2_to_str))
#string function
user_string=input("Enter a string")
print("user string is: ",user_string)

#covert string into uppercase
print("uppercase string is:",user_string.upper())
#convert string into lowercase
print("lower case string is",user_string.lower())
#return the length of the string
print("lenght of the string is:",len(user_string))
#remove whitespaces in string
print("after removing whitespcase in string",user_string.strip())
# capitalize the first character of string
print("capitalized string:",user_string.capitalize())
#replace the old character or substring to new character or substring
print("after replacing character in the string",user_string.replace("m","b"))
#returns true if starts with the given character else return false
print(user_string.startswith("a"))
#returns true if starts with the given character else return false
print(user_string.endswith("g"))
```

Output:

```
Enter a stringProgramming
string name is:  Programming
sliced_string1 : Prog
sliced_string2 rogr
sliced_string3 og
10
datatype of num before casting: <class 'int'>
datatype of num after casting: <class 'str'>
10.15
datatype of num2 before casting: <class 'float'>
datatype of num2 after casting: <class 'str'>
Enter a stringprogramming
user string is:  programming
uppercase string is: PROGRAMMING
lower case string is programming

lenght of the string is: 11

after removing whitespcase in string programming
capitalized string: Programming
after replacing character in the string prograbbing
False
True
```

Godavari Institute of Management & Research, Jalgaon
Master of Computer Application (MCA)

Name: _ Roll No: _ _ _ _ _

Date of Performance: _ _ /_ _/20_ _ Batch: _ _ _ _ _ _

Class: M.C.A. (I) Practical no:_ _ Sign. of Teacher :_ _ _ _ _ _

Subject: 416 Lab on Python Programming

---

c. Implement a program based on Simple statements using Variables, Scope of variable, Built-in Data Types, Types of operator, etc.

Variables:

variable in Python is a container for storing data values. Python does not require explicit declaration of the type of variable; it determines the type automatically based on the value assigned.

Scope of Variables:

1.Local Scope:
Variables declared inside a function.
Accessible only within the function.

2.Global Scope:
Variables declared outside all functions.
Accessible throughout the file.

Types of Operators:

Operators are special symbols or keywords used to perform operations on variables and values.

Arithmetic Operators:

Perform mathematical operations.

1. + (Addition): 10+10=20
   - (Subtraction):20-10=10
2. (Multiplication):10*10=100
3. / (Division):10/10=1
4. % (Modulus):get remainder 10%3=1

Comparison Operators:
Compare two values and return True or False.

1. **==(Equal to):** checks if the values of two operands are equal. If they are, it returns True; otherwise, it returns False
2. **!=(Not equal to):**Checks if the values of two operands are not equal. If they are not equal, it returns True; otherwise, it returns False.

3. **>(Greater than):** Checks if the value of the left operand is greater than the value of the right operand. If true, it returns True; otherwise, it returns False.
4. **<(Less than):** Checks if the value of the left operand is less than the value of the right operand. If true, it returns True; otherwise, it returns False.
5. **>=(Greater than or equal to) :**Checks if the value of the left operand is greater than or equal to the value of the right operand. If true, it returns True; otherwise, it returns False.
6. **<=(Less than or equal to):**Checks if the value of the left operand is less than or equal to the value of the right operand. If true, it returns True; otherwise, it returns False.

Logical Operators:

Combine conditional statements.

1. **and(Logical AND):** Returns True if **both** conditions are True; otherwise, it returns False.
2. **or(Logical OR):** Returns True if **at least one** condition is True; otherwise, it returns False.
3. **not(Logical NOT):** Reverses the result of a condition. If the condition is True, it becomes False; if it is False, it becomes True.

```
print("Scope of variable")
global_var= 100  #accessible anywhere in program

def scopeExample():
    local_var=20  #accessible within block and function only
    print("global variable in scopeExample() function ",global_var)
    print("local variable in scopeExample() function ",local_var)
def anotherExample():
    print("global variable in anotherExample() function ",global_var)

#calling both function
scopeExample()
anotherExample()

#Built-in datatypes
print("Built-in Datatypes")

int_var=10
print(int_var,type(int_var))# int datatype: Whole numbers without decimals.
float_var=10.50
```

```
print(float_var,type(float_var))# float datatype:Numbers with decimal points
string_var="python"
print(string_var,type(string_var))# string datatype:Text enclosed in quotes

list_var=[1,"two",3.14,5,'A']
print(list_var,type(list_var))#list:Ordered, mutable collection of items
tuple_var=('A',"one",4,7.8)
print(tuple_var,type(tuple_var))#tuple:Ordered, immutable collection of items
set_var = {1, 2, 3, 4, 5}
print(set_var,type(set_var))#set:Unordered collection of unique items
dict_var = {                #dictionary : Key-value pairs for data storage
    "name": "Alice",
    "age": 30,
    "city": "New York"
  }
print(dict_var,type(dict_var))

print("Arithmatic Operators")
#Arithmatic operator
a=10
b=20
addition=a+b
print("addition is:",addition)
substraction=a-b
print("substraction is",substraction)
multiplication=a*b
print("multiplication is:",multiplication)
division=a/b
print("division is:",division)
modulus=a%b # returns the  remainder
print("modulus is",modulus)
```

```
print("Comparision Operator")
#comparision operator
a=10
b=20
print("a is greater b: ",a > b)# greater than
print("a is less than b: ",a < b)#less than
print("Two numbers are equal or not: ",a == b)#eqaul to


print("Two numbers are not equal or not: ",a != b)#not equal to
print("a is greater than or equal to b: ",a >= b)#greater than or equalto
print("a is less than or equal to b: ",a <= b)#less than or equal to

print("Logical Operator")
# Logical Operators AND, OR,NOT
a=True
b=False
print("a and b:",a and b)# logical and :True only if both are True
print("a or b:",a or b)# logical or :True if at least one is True
print("not a:",not a) # logical not: reverse the value of a
print("not b:" ,not b) #logical not: reverse the value of b

print("Assignment operators:")
a=10
b=20
print('a=b:', a==b)
print('a+=b:', a+b)
print('a-=b:', a-b)
print('a*=b:', a*b)
print('a%=b:', a%b)
```

Output:

```
Scope of variable
global variable in scopeExample() function  100
```

local variable in scopeExample() function 20

global variable in anotherExample() function 100

Built-in Datatypes

10 <class 'int'>

10.5 <class 'float'>

python <class 'str'>

[1, 'two', 3.14, 5, 'A'] <class 'list'>

('A', 'one', 4, 7.8) <class 'tuple'>

{1, 2, 3, 4, 5} <class 'set'>

{'name': 'Alice', 'age': 30, 'city': 'New York'} <class 'dict'>

Arithmatic Operators

addition is: 30

substraction is -10

multiplication is: 200

division is: 0.5

modulus is 10

Comparision Operator

a is greater b: False

a is less than b: True

Two numbers are equal or not: False

Two numbers are not equal or not: True

a is greater than or equal to b: False

a is less than or equal to b: True

Logical Operator

a and b: False

a or b: True

not a: False

not b: True

Assignment operators:

a=b: False

a+=b: 30

a-=b: -10

a*=b: 200

a%=b: 10

d. Implement a program based on decision and looping statements.

Decision Statements:

Decision-making statements in Python allow the program to execute certain code based on conditions.

1. **if Statement:**
   Executes a block of code only if a specified condition is True.
2. **if-else Statement**
   Executes one block of code if the condition is True and another block if it is False.
3. **if-elif-else Statement**
   Checks multiple conditions in sequence.

Looping Statements:

Looping statements allow the execution of a block of code multiple times.

1. **for Loop**
   Iterates over a sequence (like a list, tuple, or range).
2. **while Loop**
   Repeats as long as a condition is True.

```
# if statement
print("if statement in python")
a = '10'
b = '5'
if a<b:
    print("a and b are not equal numbers")
```

```
#if-else in python
print("if-else statement in python")
a=5
if a == 1:
   print("a is equal to 1")
elif(a%2) == 0:
   print("a is an even number")
elif a<0:
   print("a is a negative number")
else:
   print("a is a positive odd number other than 1")
#while loop
print("Example of while loop:")
a = 1
# Iterating 8 times and then Printing ints 2-10
while a < 10:
   a += 1
   print(a)

   print("Example of for loop:")
#for loop
# iterating 10 times and then print ints 1-10
for i in range(1, 11):
   print(i)
```

output:

```
if statement in python
a and b are not equal numbers
if-else statement in python
a is a positive odd number other than 1
Example of while loop:
2
3
4
```

5

6

7

8

9

10

Example of for loop:

1

2

3

4

5

6

7

8

9

10

Godavari Institute Of Management & Research, Jalgaon
Master of Computer Application (MCA)

Name: _ Roll No: _ _ _ _ _

Date of Performance: _ _ /_ _/20_ _ Batch: _ _ _ _ _ _

Class: M.C.A. (I) Practical no:_ _ Sign. of Teacher :_ _ _ _ _ _

Subject: 416 Lab on Python Programming

---

e. Implement a program based on Tuples, Lists, Dictionaries and Sets etc.

Tuples:

A **Tuple** is a collection of items that are ordered and **cannot be changed** after creation. Tuples are written inside parentheses (). They can store different types of data and are often used when you want to keep data safe from being modified, like coordinates or fixed data.
Syntax: tuple_name = (item1, item2, item3, ...)

Lists:

A **List** is also an ordered collection, but it is **mutable**, meaning you can change, add, or remove items. Lists are written inside square brackets [] and can store different types of data. Lists are very flexible and are used when the data might change over time
Syntax: list_name = [item1, item2, item3, ...]

Dictionaries:

A **Dictionary** is a collection of **key-value pairs** where each key is unique and maps to a value. Dictionaries are written inside curly braces {}. Unlike lists and tuples, dictionaries are **unordered**. They are useful when you need to associate one piece of data with another, such as storing information about a person where the name is the key and the age is the value.
Syntax: dict_name = {key1: value1, key2: value2, key3: value3}

Sets:

A **Set** is an unordered collection of unique items, meaning it cannot contain duplicates. Sets are written inside curly braces {} and are useful when you only want one of each item, like removing duplicates from a list or performing mathematical set operations like finding common elements between two groups.
Syntax: set_name = {item1, item2, item3, ...}

```
#list
print("Example of list")
my_list=[1,2,3,4.5,"apple"]#list creation
```

```
print(my_list)
print(len(my_list))#return the length of the list
print(my_list[3])
sub_list=my_list[1:4]# accessing sublist from the list
print(sub_list)
my_list[2]="Orange"#update the value of index no.2
print(my_list)
my_list.append("Red")# add the value at the end of list
print(my_list)
my_list.extend([6,7,8])# adding elements at the end of the list
print(my_list)
my_list.insert(3,"Three")#insert the element at specified index no.
print(my_list)
my_list.remove("Orange")#remove specifies elemnts from the list
print(my_list)

#tuple
print("Example of tuple")
my_tuple=(1,2,"apple",'A',4.5,1)#creation of tuple
print(my_tuple)

print("Accessing value of specified index:",my_tuple[3])
print("length of the tuple is :",len(my_tuple))
print("How many times appear the specified value in tuple:",my_tuple.count(1))
print("return the index number of specified value: ",my_tuple.index('A'))

#set
print("Example of set")
my_set={1,2,3,"apple",4.5}
print(my_set)
my_set.add("Orange")
```

```
print(my_set)
my_set.remove(3)
print(my_set)
my_set2={4,5,"python",7.8,2}
print(my_set2)
union_set=my_set.union(my_set2)
print(union_set)
intersection_set=my_set.intersection(my_set2)
print(intersection_set)
diffrence_set=my_set.difference(my_set2)
print(diffrence_set)

#dictionaries
print("Example of Dictionary:")
my_dict={"name":"Python","age":30,"city":"NewYork"}
print(my_dict)
print(my_dict["age"])
my_dict["email"]="123@gmail.com" #adding new key-value pair in dictionary
print(my_dict)
del my_dict["age"]#remove key-value pair from dictionary
print(my_dict)
print(my_dict.keys())
print(my_dict.values())
```

output:

```
Example of list
[1, 2, 3, 4.5, 'apple']
5
4.5
[2, 3, 4.5]
```

[1, 2, 'Orange', 4.5, 'apple']

[1, 2, 'Orange', 4.5, 'apple', 'Red']

[1, 2, 'Orange', 4.5, 'apple', 'Red', 6, 7, 8]

[1, 2, 'Orange', 'Three', 4.5, 'apple', 'Red', 6, 7, 8]

[1, 2, 'Three', 4.5, 'apple', 'Red', 6, 7, 8]

Example of tuple

(1, 2, 'apple', 'A', 4.5, 1)

Accessing value of specified index: A

length of the tuple is : 6

How many times appear the specified value in tuple: 2

return the index number of specified value: 3

Example of set

{1, 2, 3, 4.5, 'apple'}

{1, 2, 3, 4.5, 'apple', 'Orange'}

{1, 2, 4.5, 'apple', 'Orange'}

{2, 4, 5, 7.8, 'python'}

{1, 2, 4.5, 4, 5, 7.8, 'apple', 'Orange', 'python'}

{2}

{'apple', 1, 'Orange', 4.5}

In [16]:

Example of Dictionary:

{'name': 'Python', 'age': 30, 'city': 'NewYork'}

30

{'name': 'Python', 'age': 30, 'city': 'NewYork', 'email': '123@gmail.com'}

{'name': 'Python', 'city': 'NewYork', 'email': '123@gmail.com'}

dict_keys(['name', 'city', 'email'])

dict_values(['Python', 'NewYork', '123@gmail.com'])

Godavari Institute of Management & Research, Jalgaon
Master of Computer Application (MCA)

Name: _ Roll No: _ _ _ _ _

Date of Performance: _ _ /_ _/20_ _ Batch: _ _ _ _ _ _

Class: M.C.A. (I) Practical no:_ _ Sign. of Teacher :_ _ _ _ _ _

Subject: 416 Lab on Python Programming

---

2. Function and Organizing Programs :

a. Implement a program based on Functions, Function with parameters.

Function:

A function is defined using the def keyword, followed by the function name and parentheses (). The function code is written inside a block indented under the function definition.
Syntax: def function_name():

Functions with Parameters:

Functions can accept inputs, called **parameters**. These parameters allow you to pass information into the function, making it more flexible and reusable. When defining a function with parameters, you specify the parameters in the parentheses after the function name.
Syntax: def function_name(parameter1, parameter2):

```
def add_numbers(a, b):  #define a function with parameters
    sum = a + b
    print('Sum:', sum)


def add_float_values(a,b,c): #define a function with parameters
    sum=a+b+c
    print("Addition of float value is: ",sum)


#calling function
add_numbers(2, 3)
add_float_values(2.3,4.5,7.8)
```

Output:

```
Sum: 5
Addition of float value is:  14.6
```

Godavari Institute of Management & Research, Jalgaon
Master of Computer Application (MCA)

Name: _ Roll No: _ _ _ _ _

Date of Performance: _ _ /_ _/20_ _ Batch: _ _ _ _ _ _

Class: M.C.A. (I) Practical no:_ _ Sign. of Teacher :_ _ _ _ _ _

Subject: 416 Lab on Python Programming

---

b. Implement a program based on Functions Inside of Functions.

When you define a function inside another function, it behaves like a normal function. However, it can only be called within the scope of the outer function.

Synatx:

```
def outer_function():
    def inner_function():
        # Code for inner function print("This is the inner function.")
    # Calling the inner function inside the outer function
    inner_function()
# Calling the outer function
outer_function()
```

```
def outer_function(a,b):
   sum=a+b
   print("Addition of both numbers is: ",sum)

   def inner_function(a,b):
      a_square=a*a
      b_square=b*b
      print(a," square is :",a_square)
      print(b," square is: ",b_square)

   inner_function(a,b) #calling inner_function


a=int(input("Enter first number"))
b=int(input("Enter second number"))

outer_function(a,b)  #calling outer function
```

output:

```
Enter first number4
Enter second number5
Addition of both numbers is:  9
4    square is : 16
5    square is:  25
```

c. Implement a program based on built-in functions.

built-in functions:

Python provides a variety of **built-in functions** that allow you to perform common operations such as mathematical calculations, type conversions, and more. These functions are always available in Python without needing to import any external libraries.

1. **abs()**: Returns the absolute value of a number.
2. **max()**: Returns the largest item from an iterable or from two or more arguments.
3. **min()**: Returns the smallest item from an iterable (like a list) or from two or more arguments.
4. **sum()**: Returns the sum of all elements in an iterable.
5. **len()**: Returns the number of items in an object (like a list, string, or tuple).
6. **round()**: Rounds a floating-point number to a specified number of decimal places.
7. **pow()**: Returns the value of a number raised to the power of another number.
8. **sorted()**: Returns a sorted list from an iterable.
9. **reversed()**: Returns an iterator that accesses the elements of the sequence in reverse order.

```
# Function to demonstrate some built-in functions
def example_built_in_functions():
    # Using abs() to return the absolute value of a number
    print("Absolute value of -5:", abs(-5))

    # Using max() to find the largest element in a list
    numbers = [3, 7, 2, 8, 1]
    print("Maximum number in the list:", max(numbers))

    # Using min() to find the smallest element in a list
    print("Minimum number in the list:", min(numbers))
```

```
    # Using sum() to find the sum of elements in a list
    print("Sum of all numbers in the list:", sum(numbers))

    # Using len() to count the number of elements in a list
    print("Number of elements in the list:", len(numbers))

    # Using sorted() to sort elements of the list in ascending order
    print("Sorted list in ascending order:", sorted(numbers))

    # Using reversed() to reverse the list (returns an iterator)
    print("Reversed list:", list(reversed(numbers)))

    # Using round() to round a number to the nearest integer
    pi = 3.14159
    print("Rounded value of pi:", round(pi, 2))

    # Using pow() to raise a number to a power
    print("2 raised to the power of 3:", pow(2, 3))

    # Using all() to check if all elements in a list are true
    bools = [True, True, False]
    print("Are all values True?:", all(bools))

    # Using any() to check if at least one element in a list is true
    print("Is any value True?:", any(bools))

    # Using type() to get the type of an object
    print("Type of 'Hello':", type("Hello"))

# Call the function to see the output
example_built_in_functions()
```

output:

Absolute value of -5: 5

Maximum number in the list: 8

Minimum number in the list: 1

Sum of all numbers in the list: 21

Number of elements in the list: 5

Sorted list in ascending order: [1, 2, 3, 7, 8]

Reversed list: [1, 8, 2, 7, 3]

Rounded value of pi: 3.14

2 raised to the power of 3: 8

Are all values True?: False

Is any value True?: True

Type of 'Hello': <class 'str'>

Godavari Institute of Management & Research, Jalgaon
Master of Computer Application (MCA)

Name: _ Roll No: _ _ _ _ _

Date of Performance: _ _ /_ _/20_ _ Batch: _ _ _ _ _ _

Class: M.C.A. (I) Practical no:_ _ Sign. of Teacher :_ _ _ _ _ _

Subject: 416 Lab on Python Programming

---

d. Implement a program using Anonymous Functions - Lambda and Filter.

anonymous functions: are functions that don't have a name. These are created using the lambda keyword. Lambda functions are useful when you need a short function for a brief task and don't want to define a full function using the def keyword.

Lambda Functions:
A **lambda function** is a simple function that is defined in one line. arguments are the inputs to the function. expression is the operation or result the function returns.

Syntax: lambda arguments: expression

Lambda with filter():
The **filter()** function is used to filter items from a list (or other iterable) based on a condition. It returns only the elements that match the condition. function is the condition or rule you want to apply. iterable is the list or collection of items to check.

Syntax: filter(function, iterable)

```
#addition of numbers using Lambda function
add_numbers=lambda a,b:a+b
result=add_numbers (10,20)
print("Addition of number is: ", result)


#Python Program to find numbers divisible
# by 10 from a list using anonymous function
#Take a list of numbers.
my_list = [12, 65, 54, 39, 102, 339, 221, 50, 70, ]
#use anonymous function to filter and comparing
```

```
#if divisible or not
result = list(filter (lambda a: (a % 10 == 0), my_list))
#printing the result
print(result)
```

output:

```
Addition of number is:  30
[50, 70]
```

Godavari Institute of Management & Research, Jalgaon
Master of Computer Application (MCA)

Name: _ Roll No: _ _ _ _ _

Date of Performance: _ _ /_ _/20_ _ Batch: _ _ _ _ _ _

Class: M.C.A. (I) Practical no:_ _ Sign. of Teacher :_ _ _ _ _ _

Subject: 416 Lab on Python Programming

---

e. Implement a program using Maps, List Comprehension and Dictionaries.

Maps:

The map() function is used to apply a function to each item in an iterable (like a list or tuple). It allows you to transform or modify each element in a collection in a simple way. For example, if you have a list of numbers and you want to square each number, you can use map() to apply the squaring function to each item in the list. You can use lambda (a small anonymous function) with map() to keep the code short and simple. A function that defines the operation to be performed on each item. An iterable (like a list or tuple) whose elements will be passed to the function.

Syntax: map(function, iterable)

List Comprehension:

List comprehension is a quick and easy way to create lists in Python. Instead of using loops to build a list, you can use list comprehension to generate a new list by applying an expression to each item in an existing iterable. It's a cleaner and more readable way to create lists. Additionally, you can add conditions to filter the elements, making it more powerful. For example, you can use list comprehension to create a list of squared numbers from an existing list or a list of even numbers from a range of numbers. expression: operation you want to perform on each item.iterable: The collection of items you want to loop through.condition (optional): A condition to filter elements.

Syntax: [expression for item in iterable if condition]

Dictionaries:

A **dictionary** is a collection of key-value pairs. It is used to store data where each item has a unique key associated with it. This makes it easy to retrieve the value based on the key.

```
# List of numbers
numbers = list(map(int,input("enter numbers:").split()))
# Using map to determine if a number is even or odd
even_or_odd = list(map(lambda x: "even" if x % 2 == 0 else "odd", numbers))
# Using list comprehension to categorize numbers into even and odd
```

```
number_dict = {
    "even": [num for num in numbers if num % 2 == 0],
    "odd": [num for num in numbers if num % 2 != 0]
}

# Display results
print("Even or Odd (using map):", even_or_odd)
print("Number Dictionary (using list comprehension):", number_dict)
```

output:

```
enter numbers:13 45 28 40 25
Even or Odd (using map): ['odd', 'odd', 'even', 'even', 'odd']
Number Dictionary (using list comprehension): {'even': [28, 40], 'odd': [13, 45, 25]}
```

Godavari Institute of Management & Research, Jalgaon
Master of Computer Application (MCA)

Name: _ Roll No: _ _ _ _ _

Date of Performance: _ _ /_ _/20_ _ Batch: _ _ _ _ _ _

Class: M.C.A. (I) Practical no:_ _ Sign. of Teacher :_ _ _ _ _ _

Subject: 416 Lab on Python Programming

---

f. Implement a program to demonstrate Class and Object.

Class:
A class is a blueprint or template for creating objects. It defines the properties (attributes) and behaviors (methods) that the objects created from it will have.You define a class using the class keyword.

Object:
An object is an instance of a class. When a class is instantiated, an object is created, which can use the attributes and methods defined in the class.You create an object by calling the class like a function.

```
class Student: #class
    def __init__(self, name, roll_no, marks):
        self.name = name
        self.roll_no = roll_no
        self.marks = marks

    def display_details(self):
        print(f"Name: {self.name}, Roll Number: {self.roll_no}, Marks: {self.marks}")

students = []

# Input details for multiple students
while True:
    name = input("Enter Name: ")
```

```
    roll_no = int(input("Enter Roll Number: "))
    marks = float(input("Enter Marks: "))
    students.append(Student(name, roll_no, marks))
    #Student(name, roll_no, marks)=object
    choice = input("Add another student? (yes/no): ").lower()
    if choice != "yes":
        break


# Display details of all students
print("\nStudent Details:")
for student in students:
    student.display_details()
```

output:

```
Enter Name: student1
Enter Roll Number: 11
Enter Marks: 45.78
Add another student? (yes/no): yes
Enter Name: student2
Enter Roll Number: 34
Enter Marks: 89.60
Add another student? (yes/no): yes
Enter Name: student3
Enter Roll Number: 27
Enter Marks: 34.40
Add another student? (yes/no): no

Student Details:
Name: student1, Roll Number: 11, Marks: 45.78
Name: student2, Roll Number: 34, Marks: 89.6
Name: student3, Roll Number: 27, Marks: 34.4
```

Godavari Institute of Management & Research, Jalgaon
Master of Computer Application (MCA)

Name: _ Roll No: _ _ _ _ _

Date of Performance: _ _ /_ _/20_ _ Batch: _ _ _ _ _ _

Class: M.C.A. (I) Practical no:_ _ Sign. of Teacher :_ _ _ _ _ _

Subject: 416 Lab on Python Programming

---

g. Implement a program to demonstrate Array.

An array is a collection of items stored at contiguous memory locations, used to store multiple values of the same data type efficiently. In Python, the **array module** provides a way to create arrays with elements of the same type. To use this module, you need to import it and specify a type code to define the data type of the elements. This makes arrays an effective tool for managing homogeneous data in memory.

```
import array  # Importing the array module


# Create an array of integers
numbers = array.array('i', [])  # 'i' stands for an integer array
# Input elements into the array
n = int(input("Enter the number of elements: "))
for item in range(n):
    element = int(input("Enter an element of array: "))
    numbers.append(element)  # Add the element to the array


# Display the array
print("\nArray elements:")
for num in numbers:
    print(num, end=" ")
```

```
# Perform some operations
print("\n\nOperations on the array:")
print(f"Length of the array: {len(numbers)}")
print(f"First element: {numbers[0]}")
print(f"Last element: {numbers[-1]}")

# Insert an element at a specific position
pos = int(input("\nEnter position to insert a new element (0-based index): "))
new_element = int(input("Enter the new element: "))
numbers.insert(pos, new_element)
print("Array after insertion:", list(numbers))

# Remove an element
element_to_remove = int(input("\nEnter an element to remove: "))
if element_to_remove in numbers:
    numbers.remove(element_to_remove)
    print("Array after removal:", list(numbers))
else:
    print("Element not found in the array.")

# Reverse the array
numbers.reverse()
print("Reversed array:", list(numbers))
```

output:

```
Enter the number of elements: 6
Enter an element of array: 12
Enter an element of array: 45
Enter an element of array: 39
Enter an element of array: 2
Enter an element of array: 10
Enter an element of array: 40
```

Array elements:

12 45 39 2 10 40

Operations on the array:

Length of the array: 6

First element: 12

Last element: 40

Enter position to insert a new element (0-based index): 3

Enter the new element: 30

Array after insertion: [12, 45, 39, 30, 2, 10, 40]

Enter an element to remove: 2

Array after removal: [12, 45, 39, 30, 10, 40]

Reversed array: [40, 10, 30, 39, 45, 12]

Godavari Institute of Management & Research, Jalgaon
Master of Computer Application (MCA)

Name: _ Roll No: _ _ _ _ _

Date of Performance: _ _ /_ _/20_ _ Batch: _ _ _ _ _ _

Class: M.C.A. (I) Practical no:_ _ Sign. of Teacher :_ _ _ _ _ _

Subject: 416 Lab on Python Programming

---

3. File And Directories

a. Implement a program based on Writing Text Files, Appending Text to a File, Reading Text Files.

- **Writing to a Text File:**
  Writing to a file involves creating a new file or overwriting an existing file. This is done using the **open()** function with the mode **'w'**. The mode 'w' creates a new file if it doesn't exist or overwrites the content of an existing file. The **with** statement ensures the file is properly closed after the operation.

- **Appending Text to a File:**
  Appending allows you to add new content to an existing file without deleting its current content. Use the mode **'a'**. The mode 'a' opens the file for appending. If the file doesn't exist, it creates a new one.

- **Reading from a Text File:**
  Reading a file allows you to access its content. Use the mode **'r'**. The **read()** method reads the entire file content as a string.

```
# Open function to open the file "MyFile1.txt"
file1 = open("myfile.txt","r")# read  mode
print("file content:",file1.read())
file1.close()
file1 = open("myfile.txt","w")# write mode
content=input("Enter the content you want to write in file")
file1.write("\n")
file1.write(content)
file1.close()
file1 = open("myfile.txt","r+")#read write mode
print(file1.read())
```

```
file1 = open("myfile.txt", "a")  # append mode
file1.write("\n Today \n")
file1.close()

file1 = open("myfile.txt", "r")# read mode
print("Output of Readlines after appending")
print(file1.readlines())
file1.close()
```

output:

file content:

Enter the content you want to write in filepython programming is easy to learn

python programming is easy to learn

Output of Readlines after appending

['\n', 'python programming is easy to learn\n', ' Today \n']

Godavari Institute of Management & Research, Jalgaon
Master of Computer Application (MCA)

Name: _ Roll No: _ _ _ _ _

Date of Performance: _ _ /_ _/20_ _ Batch: _ _ _ _ _ _

Class: M.C.A. (I) Practical no:_ _ Sign. of Teacher :_ _ _ _ _ _

Subject: 416 Lab on Python Programming

---

b. Implement a program based on paths and directories, file information, naming, moving, copying, and removing files.

- **File Information:**
  use the os module to retrieve information about a file, such as its size, creation time, and modification time.
  - os.path.getsize(): Gets the size of the file in bytes.
  - os.path.getctime(): Retrieves the creation time of the file (on most platforms).
  - os.path.getmtime(): Retrieves the last modification time.
  - os.path.isfile(): Checks if the path is a file.

- **Renaming a File:**
  The os.rename() function is used to rename a file.

- **Moving a File:**
  To move a file, you can use shutil.move().

- **Copying a File:**
  The shutil module also provides functions for copying files, such as shutil.copy() and shutil.copy2()

- **Removing (Deleting) a File:**
  To delete a file, use the os.remove() function.

```
import os
import shutil
# Create a directory "test"
#os.mkdir("test2")
print ("Directory created successfully")
```

```
current_directory = os.getcwd()
print(f"Current working directory: {current_directory}")
contents = os.listdir(current_directory)
print(f"Contents of '{current_directory}':")
for files in contents:
    print(files)

new_directory="C:\\Users\\tejas\\MCA1 415 Practicals\\test1"
os.chdir(new_directory)
print(os.getcwd())
new_directory="C:\\Users\\tejas\\MCA1 415 Practicals"
contents = os.listdir(new_directory)
print(f"Contents of '{new_directory}':")

for files in contents:
    print(files)

# renaming directory
os.rename("Myfile.txt","RenameMyfile.txt")

print("After renaming file")
contents = os.listdir(new_directory)
for files in contents:
    print(files)
shutil.copy("RenameMyfile.txt", "CopyMyfile.txt")
new_directory="C:\\Users\\tejas\\MCA1 415 Practicals"
contents = os.listdir(new_directory)
print(f"Contents of '{new_directory}':")
for files in contents:
    print(files)
```

```
source="C:\\Users\\tejas\\MCA1 415 Practicals\\test1"
destination="C:\\Users\\tejas\\ankita\\test1"
shutil.move(source, destination)
```

output:

```
Directory created successfully
Current working directory: C:\Users\tejas\MCA1 415 Practicals\test1
Contents of 'C:\Users\tejas\MCA1 415 Practicals\test1':
C:\Users\tejas\MCA1 415 Practicals
Contents of 'C:\Users\tejas\MCA1 415 Practicals':
.ipynb_checkpoints
Myfile.txt
test
test1
test2
Untitled.ipynb
Untitled1.ipynb
Untitled2.ipynb
Untitled3.ipynb
After renaming file
.ipynb_checkpoints
RenameMyfile.txt
test
test1
test2
Untitled.ipynb
Untitled1.ipynb
Untitled2.ipynb
Untitled3.ipynb
Contents of 'C:\Users\tejas\MCA1 415 Practicals':
.ipynb_checkpoints
CopyMyfile.txt
```

RenameMyfile.txt

test

test1

test2

Untitled.ipynb

Untitled1.ipynb

Untitled2.ipynb

Untitled3.ipynb

'C:\\Users\\tejas\\ankita\\test1'

Godavari Institute of Management & Research, Jalgaon
Master of Computer Application (MCA)

Name: _ Roll No: _ _ _ _ _

Date of Performance: _ _ /_ _/20_ _ Batch: _ _ _ _ _ _

Class: M.C.A. (I) Practical no:_ _ Sign. of Teacher :_ _ _ _ _ _

Subject: 416 Lab on Python Programming

---

c. Implement a program Based on Regular Expression and its functions.

Regular Expression:

regular expressions (regex) are used to search, match, and manipulate strings based on patterns. The re module in Python provides a wide range of functions for working with regular expressions.

- **re.search():**
  Searches the entire string for a match to the pattern.Returns a match object if found, otherwise returns None

- **re.findall():**
  Finds all non-overlapping matches of the pattern in the string and returns them as a list

- **re.sub():**
  Replaces occurrences of a pattern with a specified string.

- **re.split():**
  Splits the string by the occurrences of the pattern.

```
import re
string = 'Python is object oriented programming language'
match = re.search(r'object', string)
print('Start Index:', match.start())
print('End Index:', match.end())
pattern = "[a-p]"
result = re.findall(pattern, string)
print(result)
print(re.sub("mm", "bb", string))
```

output:

Start Index: 10

End Index: 16

['h', 'o', 'n', 'i', 'o', 'b', 'j', 'e', 'c', 'o', 'i', 'e', 'n', 'e', 'd', 'p', 'o', 'g', 'a', 'm', 'm', 'i', 'n', 'g', 'l', 'a', 'n', 'g', 'a', 'g', 'e']

Python is object oriented prograbbing language

Godavari Institute of Management & Research, Jalgaon
Master of Computer Application (MCA)

Name: _ Roll No: _ _ _ _ _

Date of Performance: _ _ /_ _/20_ _ Batch: _ _ _ _ _ _

Class: M.C.A. (I) Practical no:_ _ Sign. of Teacher :_ _ _ _ _ _

Subject: 416 Lab on Python Programming

4. Python GUI Programming

a. Implement a program creating Types of GUI Widgets, Resizing, and Configuring Options.

Tkinter:

Tkinter is one of the most popular libraries for creating Graphical User Interfaces (GUIs). It provides a wide range of widgets that you can use to create interactive applications.

- Label: Displays text or an image.
- Button: A clickable button that triggers a function.
- Text: A multi-line text area for user input.
- Checkbutton: A checkbox that the user can check or uncheck.
- Radiobutton: A set of mutually exclusive options (radio buttons).
- Listbox: Displays a list of options that the user can select from.
- Menu: Provides a menu bar with various options like File, Edit, etc.

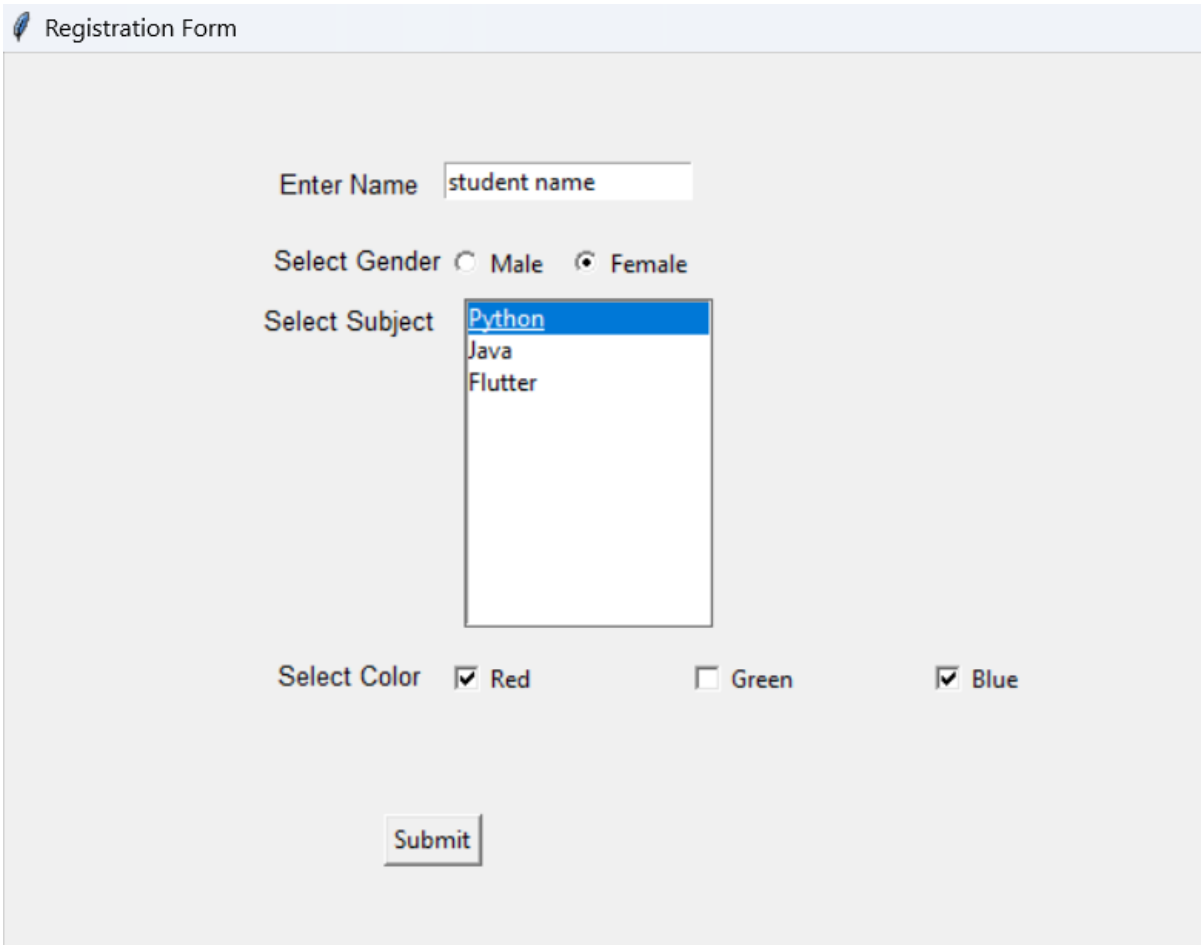
```
from tkinter import *


def registrationForm():
    print("Name =", title1.get())
    i = radio.get()
    if i == 1:
        print("Gender = Male")
```

```
    elif i == 2:
        print("Gender = Female")
    else:
        print("Gender = Not Selected")

    j = checkbox1.get()
    k = checkbox2.get()
    l = checkbox3.get()
    colors = []
    if j == 1:
        colors.append("Red")
    if k == 1:
        colors.append("Green")


    if l == 1:
        colors.append("Blue")
    print("Color =", ", ".join(colors))

    print("Selected Classes:")
    for i in list1.curselection():
        print(list1.get(i))

base = Tk()
base.geometry('600x600')
base.title("Registration Form")

lable1 = Label(base, text="Enter Name", width=20, font=("bold", 10))
lable1.place(x=90, y=55)

title1 = Entry(base)
title1.place(x=220, y=55)
```

```
radio = IntVar()
lable2 = Label(base, text="Select Gender", width=20, font=("bold", 10))
lable2.place(x=94, y=93)

radiobutton1 = Radiobutton(base, text="Male", value=1, variable=radio)
radiobutton1.place(x=220, y=93)

radiobutton2 = Radiobutton(base, text="Female", value=2, variable=radio)
radiobutton2.place(x=280, y=93)

lable3 = Label(base, text="Select Subject", width=20, font=("bold", 10))
lable3.place(x=90, y=123)

list1 = Listbox(base, selectmode=MULTIPLE)
list1.insert(1, "Python")
list1.insert(2, "Java")
list1.insert(3, "Flutter")
list1.place(x=230, y=123)

checkbox1 = IntVar()
checkbox2 = IntVar()
checkbox3 = IntVar()

lable4 = Label(base, text="Select Color", width=20, font=("bold", 10))
lable4.place(x=90, y=300)

checkbutton1 = Checkbutton(base, text="Red", onvalue=1, offvalue=0,
variable=checkbox1)
checkbutton1.place(x=220, y=300)

checkbutton2 = Checkbutton(base, text="Green", onvalue=1, offvalue=0,
variable=checkbox2)
checkbutton2.place(x=340, y=300)
```

```
checkbutton3 = Checkbutton(base, text="Blue", onvalue=1, offvalue=0,
variable=checkbox3)

checkbutton3.place(x=460, y=300)


button1 = Button(base, text="Submit", command=registrationForm)

button1.place(x=190, y=380)


base.mainloop()
```

output:



Name = student name

Gender = Male

Color = Red, Blue

Selected Classes:

Python

Godavari Institute of Management & Research, Jalgaon
Master of Computer Application (MCA)

Name: _ Roll No: _ _ _ _ _

Date of Performance: _ _ /_ _/20_ _ Batch: _ _ _ _ _ _

Class: M.C.A. (I) Practical no:_ _ Sign. of Teacher :_ _ _ _ _ _

Subject: 416 Lab on Python Programming

---

b. Implement a program Creating Layouts, Packing Order, Controlling Widget

```
import tkinter as tk

root = tk.Tk()
root.geometry("300x300")  # Set window size to avoid overlapping elements

# Create buttons with active background and foreground colors
button = tk.Button(root, text="Click Me", activebackground="red", activeforeground="white")
button.place(x=50, y=50)

button1 = tk.Button(root, text="Button1", activebackground="green",
activeforeground="white")
button1.place(x=120, y=50)

button2 = tk.Button(root, text="Button2", activebackground="yellow",
activeforeground="white")
button2.place(x=190, y=50)

# Create labels with background and foreground colors
label = tk.Label(root, text="Python", bg="blue", fg="black")
label.place(x=50, y=100)

label1 = tk.Label(root, text="Java", bg="orange", fg="black")
label1.place(x=120, y=100)

label2 = tk.Label(root, text="Flutter", bg="pink", fg="black")
label2.place(x=190, y=100)

root.mainloop()
```

output:

5. Text Processing

a. Implement a program based on Text Processing, Searching for Files.

Text Processing:

Text processing in Python involves various techniques and tools for manipulating and analyzing textual data. Python has many powerful libraries and built-in functions that make text processing tasks easy and efficient such as string,re,nltk,os.

- String Operations
- Regular Expressions (Regex)
- Text Cleaning and Transformation
- Text Tokenization

```
# import the necessary libraries
from nltk.tokenize import sent_tokenize,word_tokenize
import string
import re
import os

#convert string into lowercase
user_string = "MCA 415 Python!! Python is Obeject Oriented Programming Language"
text_lowercase=user_string.lower()
print("user string in lowercase:",text_lowercase)

# Remove numbers
remove_num=re.sub(r'\d+', '', user_string)
print("after removing numbers in string:",remove_num)
```

```
# remove punctuation
translator = str.maketrans('', '', string.punctuation)
remove_punctuation=user_string.translate(translator)
print("After rmoving punctuation:",remove_punctuation)

#split user string in sentence
sentence=sent_tokenize(user_string)
print("split user string in one or more sentences:",sentence)

#split user string in word
string_in_word=word_tokenize(user_string)
print("split user string in words:",string_in_word)

#searching for files
def search_files(filename, directory_name):
    result = []

    # Recursively walks through the directory
    for root, dir, files in os.walk(directory_name):
        if filename in files:
            # If the file is found, add its full path to the result list
            result.append(os.path.join(root, filename))
    return result

filename = "RenameMyfile.txt"
directory_name = "C:\\Users\\"

found_files = search_files(filename, directory_name)
```

```
if found_files:
    print(f"File '{filename}' found in the following locations:")
    for file_path in found_files:
        print(f"- {directory_name}")
else:
    print(f"File '{filename}' not found in directory '{directory_name}'.")
```

output:

user string in lowercase: mca 415 python!! python is obeject oriented programming language

after removing numbers in string: MCA Python!! Python is Obeject Oriented Programming Language

After rmoving punctuation: MCA 415 Python Python is Obeject Oriented Programming Language

split user string in one or more sentences: ['MCA 415 Python!!', 'Python is Obeject Oriented Programming Language']

split user string in words: ['MCA', '415', 'Python', '!', '!', 'Python', 'is', 'Obeject', 'Oriented', 'Programming', 'Language']

File 'RenameMyfile.txt' found in the following locations:

-C:\Users\

Godavari Institute of Management & Research, Jalgaon
Master of Computer Application (MCA)

Name: _ Roll No: _ _ _ _ _

Date of Performance: _ _ /_ _/20_ _ Batch: _ _ _ _ _ _

Class: M.C.A. (I) Practical no:_ _ Sign. of Teacher :_ _ _ _ _ _

Subject: 416 Lab on Python Programming

---

c. Implement a program to demonstrate HTML Parsing.

- **HTML Parsing:**
  HTML parsing in Python is the process of extracting data from HTML documents, which is typically done using libraries that can parse HTML content, navigate the DOM (Document Object Model), and extract useful information. There are several libraries available in Python for HTML parsing, with the most popular being BeautifulSoup from the bs4 package and lxml.

- **BeautifulSoup (from bs4 library):**
  BeautifulSoup is one of the most popular Python libraries for parsing HTML and XML documents. It provides easy-to-use methods for navigating and searching the parse tree.

```
from bs4 import BeautifulSoup

# Sample HTML content
html_content = """
<!DOCTYPE html>
<html>
<head>
    <title>Sample HTML Page</title>
</head>
<body>
    <h1>Welcome to HTML Parsing</h1>
    <p>This is a paragraph in the HTML content.</p>
```

```
  <p id="pid">This paragraph contains some <a
href="https://www.w3schools.com/html/">example link</a>.</p>
   <ul>
      <li>Item 1</li>
      <li>Item 2</li>
      <li>Item 3</li>
   </ul>
</body>
</html>
"""

# Create a BeautifulSoup object
soup = BeautifulSoup(html_content, 'html.parser')

# Parsing and extracting data
print("Title of the page:")
print(soup.title.string)

print("\nAll paragraphs:")
for paragraph in soup.find_all('p'):
   print(paragraph.text)

print("\nFirst hyperlink and its URL:")
link = soup.find('a')
print(f"Text: {link.text}")
print(f"URL: {link['href']}")

print("\nList items in the unordered list:")
for li in soup.find_all('li'):
   print(li.text)
```

```
# Find the element to remove by its tag and remove it
element_to_remove = soup.find('p', {'id': 'pid'})
element_to_remove.extract()
print("after rmoving <p id=pid> tag")

# Print the modified HTML
print(soup.prettify())
```

output:

```
Title of the page:
Sample HTML Page

All paragraphs:
This is a paragraph in the HTML content.
This paragraph contains some example link.

First hyperlink and its URL:
Text: example link
URL: https://www.w3schools.com/html/
List items in the unordered list:
Item 1
Item 2
Item 3
after rmoving <p id=pid> tag
<!DOCTYPE html>
<html>
 <head>
  <title>
   Sample HTML Page
  </title>
 </head>
```

```
<body>
 <h1>
  Welcome to HTML Parsing
 </h1>
 <p>
  This is a paragraph in the HTML content.
 </p>
 <ul>
  <li>
   Item 1
  </li>
  <li>
   Item 2
  </li>
  <li>
   Item 3
  </li>
 </ul>
</body>
</html>
```

Godavari Institute of Management & Research, Jalgaon
Master of Computer Application (MCA)

Name: _ Roll No: _ _ _ _ _

Date of Performance: _ _/_ _/20_ _ Batch: _ _ _ _ _ _

Class: M.C.A. (I) Practical no:_ _ Sign. of Teacher :_ _ _ _ _ _

Subject: 416 Lab on Python Programming

6.Accessing Databases

a. Implement a program based on using DBM - Creating and Accessing Persistent Dictionaries.

DBM (Database Manager):

DBM (Database Manager) refers to a simple, file-based database system for storing key-value pairs. It is primarily used for lightweight and fast storage of small amounts of data, offering a dictionary-like interface for storing and retrieving data. Python provides the dbm module (part of the standard library) to interact with DBM databases.

```
import dbm
# Create or open a persistent dictionary
# 'c' mode creates the file if it doesn't exist
def create_persistent_dict():
    with dbm.open('dict_dbm', 'c') as db:
        db['name'] = input("Enter name")
        db['age'] = input("Enter age")
        db['city'] = input("Enter city")
        print("Data has been stored in the persistent dictionary.")

# Access and display data from the persistent dictionary
 # 'r' mode for read-only access
def access_persistent_dict():
    with dbm.open('dict_dbm', 'r') as db:
        print("Stored data:")
```

```
    print(f"Name: {db['name']}")
        print(f"Age: {db['age']}")
        print(f"City: {db['city']}")

create_persistent_dict()
access_persistent_dict()
```

Output:

```
Enter namePython
Enter age34
Enter cityPune
Data has been stored in the persistent dictionary.
Stored data:
Name: b'Python'
Age: b'34'
City: b'Pune'
```

Godavari Institute of Management & Research, Jalgaon
Master of Computer Application (MCA)

Name: _ Roll No: _ _ _ _ _

Date of Performance: _ _/_ _/20_ _ Batch: _ _ _ _ _ _

Class: M.C.A. (I) Practical no:_ _ Sign. of Teacher :_ _ _ _ _ _

Subject: 416 Lab on Python Programming

---

b. Implement a program based on using Relational Database - Writing SQL Statements, Defining Tables, Setting Up a Database.(Transactions and Committing the Results.)

MySQL Database:

MySQL is a widely used relational database management system (RDBMS). You can interact with MySQL databases in Python using the mysql-connector-python library

- **Step 1:** Connect to the MySQL Database
  establish a connection to the MySQL server.
- **Step 2:** Create a Table
  create a table using the CREATE TABLE SQL statement.
- **Step 3:** Insert Data
  insert data into a table using the INSERT INTO SQL statement.
- **Step 4:** Retrieve Data
  retrieve data using a SELECT statement.
- **Step 5:** Update Data
  update records using the UPDATE statement.
- **Step 6:** Delete Data
  delete data using the DELETE SQL statement.
- **Step 7:** Close the Connection

```
pip install mysql-connector-python

import mysql.connector


# Establish the connection to MySQL

db = mysql.connector.connect(

  host="localhost",

  user="username",     # Replace with your MySQL username

  password="userpass"  # Replace with your MySQL password

)
```

```
dbcursor = db.cursor()

# Create database and select it
dbcursor.execute("CREATE DATABASE IF NOT EXISTS my_database")
dbcursor.execute("USE my_database")  # Select the database

# Create table
dbcursor.execute("""
CREATE TABLE IF NOT EXISTS student (

 Roll_no INT(10) ,
   Name VARCHAR(20),
   Address VARCHAR(20)
)
""")

# Insert query
insert_query = "INSERT INTO student (Name, Address) VALUES (%s, %s)"
values = ("Student1", "Pune")
dbcursor.execute(insert_query, values)

# Commit the changes
db.commit()

# Print the number of records inserted
print(dbcursor.rowcount, "record inserted.")

# Fetch and display the data from the student table
dbcursor.execute("SELECT * FROM student")
result = dbcursor.fetchall()
```

```
# Print each row
for x in result:
    print(x)

# Close the connection
dbcursor.close()
db.close()
```

Output:

```
1 record inserted.
(1, 'Student1', 'Pune')
```