SANT DNYANESHWAR SHIKSHAN SANSTHA'S
ANNASAHEB DANGE COLLEGE OF ENGINEERING &
TECHNOLOGY, ASHTA

DEPARTMENT OF ARTIFICIAL INTELLIENCE AND DATA SCIENCE

LABORATORY MANUAL

Python For Data Science(1ADVS206)

Mrs. Priyanka Suryawanshi

SECOND YEAR

Academic Year

2023-24

Python for Data Science Experiments List:	
1	Implement basic Python programs to demonstrate fundamental concepts by reading input from console.
2	Implement Python programs to demonstrate decision control and looping statements.
3	Apply Python built-in data types: Strings, List, Tuples, Dictionary, Set and their methods to solve any given problem
4.	Implement OOP concepts like Data hiding and Data Abstraction.
5.	Create user-defined functions with different types of function arguments
6.	Perform File manipulations operations- open, close, read, write, append and copy from one file to another.
7.	Handle Exceptions using Python Built-in Exceptions
8.	Implement various in built functions of NumPy library.
9.	Create Pandas Series and DataFrame from various inputs.
10.	Import any CSV file to Pandas DataFrame and perform the following: (a) Visualize the first and last 10 records (b) Get the shape, index and column details (c) Select/Delete the records(rows)/columns based on conditions. (d) Perform ranking and sorting operations. (e) Do required statistical operations on the given columns. (f) Find the count and uniqueness of the given categorical values. (g) Rename single/multiple columns.
11.	Import any CSV file to Pandas DataFrame and perform the following: (a) Handle missing data by detecting and dropping/ filling missing values. (b) Transform data using apply() and map() method. (c) Detect and filter outliers. (d) Perform Vectorized String operations on Pandas Series. (e) Visualize data using Line Plots, Bar Plots, Histograms, Density Plots and Scatter Plots.

Experiment No : 1

- **Title :** Implement Basic Python Programs To Demonstrate Fundamental Concepts By Reading Input From Console.

- **Aim :** Basic python programs helps to understand the fundamental concepts by reading input from console.

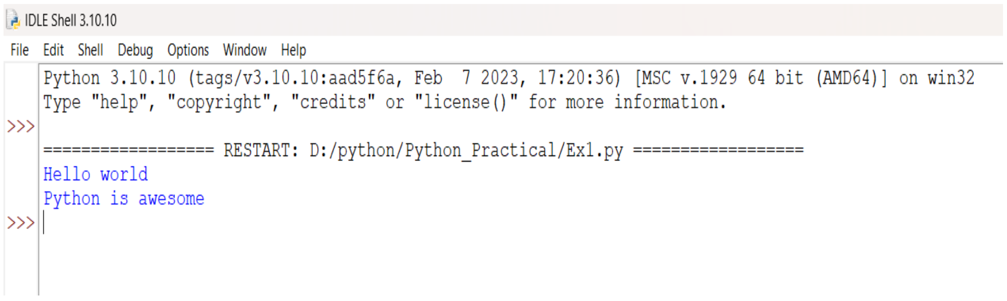
- **Theory :**

  What is Console in Python? Console (also called Shell) is basically a command line interpreter that takes input from the user i.e one command at a time and interprets it. If it is error free then it runs the command and gives required output otherwise shows the error message. Here we write a command and to execute the command just press enter key and your command will be interpreted. For coding in Python, you must know the basics of the console used in Python. The primary prompt of the python console is the three greater than symbols. You are free to write the next command on the shell only when these prompts have appeared after executing the first command. The Python Console accepts commands in Python that you write after the prompt. **Accepting Input from Console** User enters the values in the Console and that value is then used in the program as it was required. To take input from the user we make use of a built-in function *input()*.

- **Program :**
- **#Example No.1**

```
print("Hello world")

x="Python"

y="is"

z="awesome"

print(x,y,z)
```

- **Output :**



```
Python 3.10.10 (tags/v3.10.10:aad5f6a, Feb  7 2023, 17:20:36) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================== RESTART: D:/python/Python_Practical/Ex1.py ==================
Hello world
Python is awesome
>>>|
```

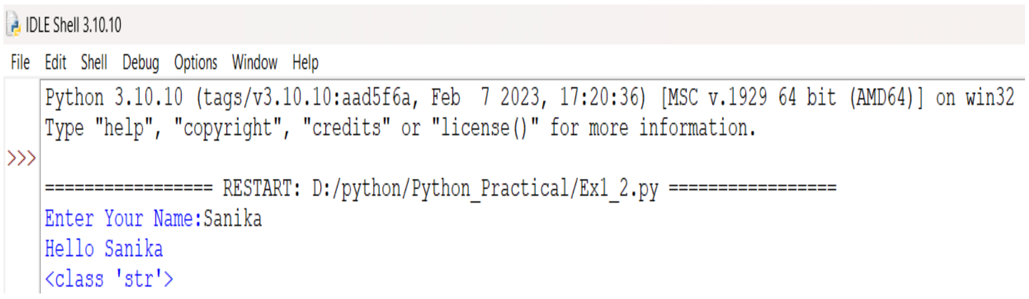
- **#Example No 2**

  name=input("Enter Your Name:")

  print("Hello "+name)

  print(type(name))

- **Output :**



```
Python 3.10.10 (tags/v3.10.10:aad5f6a, Feb  7 2023, 17:20:36) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================= RESTART: D:/python/Python_Practical/Ex1_2.py =================
Enter Your Name:Sanika
Hello Sanika
<class 'str'>
```

- **#Example No 3**

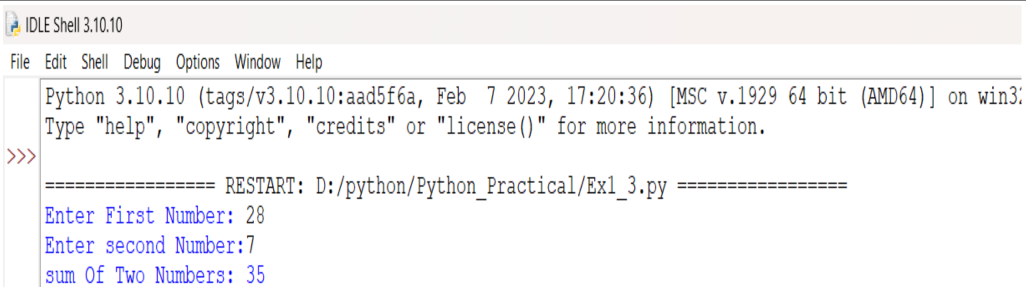
  a=int(input("Enter First Number:"))

  b=int(input("Enter second Number:"))

  c = a + b

  print("sum Of Two Numbers:",c)

- **Output :**

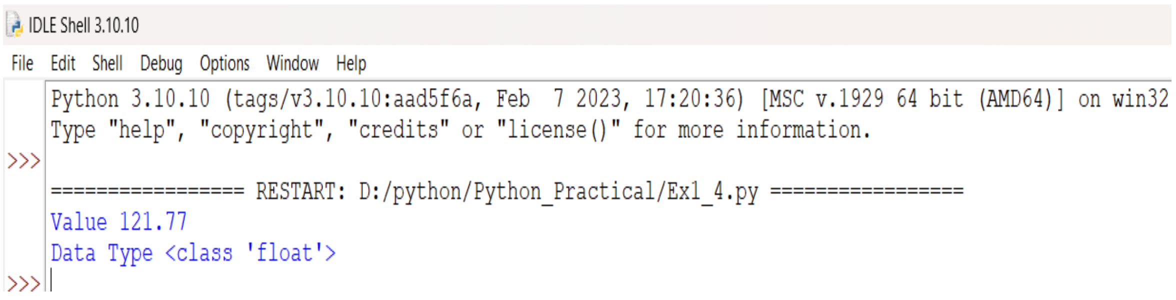


```
Python 3.10.10 (tags/v3.10.10:aad5f6a, Feb  7 2023, 17:20:36) [MSC v.1929 64 bit (AMD64)] on win3
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================= RESTART: D:/python/Python_Practical/Ex1_3.py =================
Enter First Number: 28
Enter second Number:7
sum Of Two Numbers: 35
```

- **#Example No 4**

```
integer_number=123
float_number=1.23
new_number=integer_number-float_number
print("Value",new_number)
print("Data Type",type(new_number))
```

- **Output :**

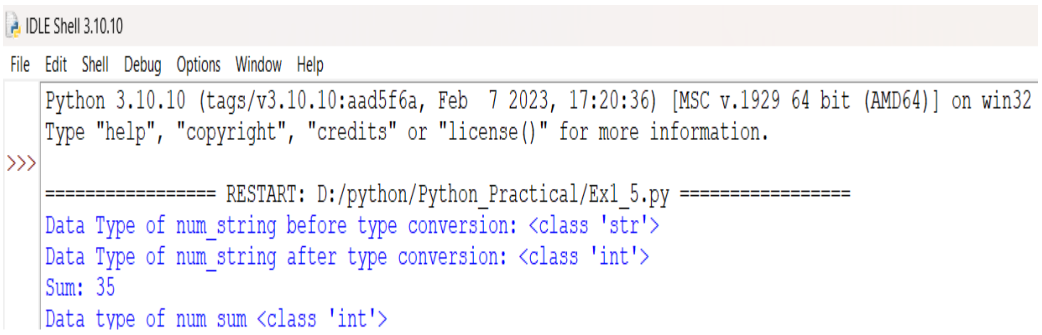


```
Python 3.10.10 (tags/v3.10.10:aad5f6a, Feb  7 2023, 17:20:36) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================= RESTART: D:/python/Python_Practical/Ex1_4.py =================
Value 121.77
Data Type <class 'float'>
>>>|
```

- **#Example No 5**

```
num_string="12"
num_integer=23
print("Data Type of num_string before type
conversion:",type(num_string))
num_string=int(num_string)
print("Data Type of num_string after type conversion:",type(num_string))
num_sum = num_integer + num_string
print("Sum:",num_sum)
print("Data type of num_sum",type(num_sum))
```

- **Output :**

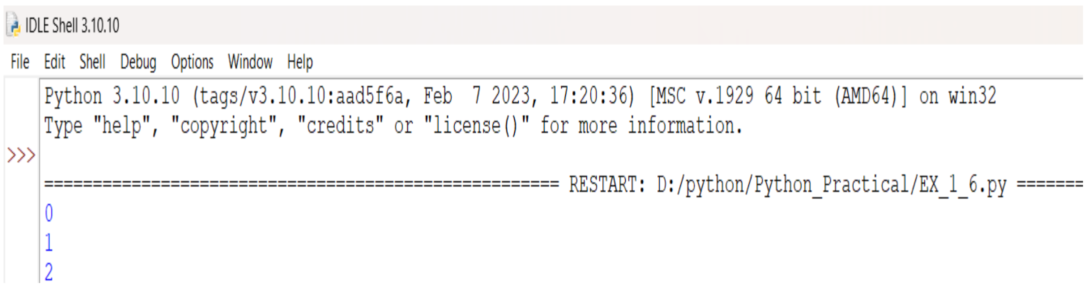


```
Python 3.10.10 (tags/v3.10.10:aad5f6a, Feb  7 2023, 17:20:36) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================= RESTART: D:/python/Python_Practical/Ex1_5.py =================
Data Type of num_string before type conversion: <class 'str'>
Data Type of num_string after type conversion: <class 'int'>
Sum: 35
Data type of num sum <class 'int'>
```

- **#Example No 6**

```
for i in range(10):
    print(i)
    if i == 2:
        break
```

- **Output :**

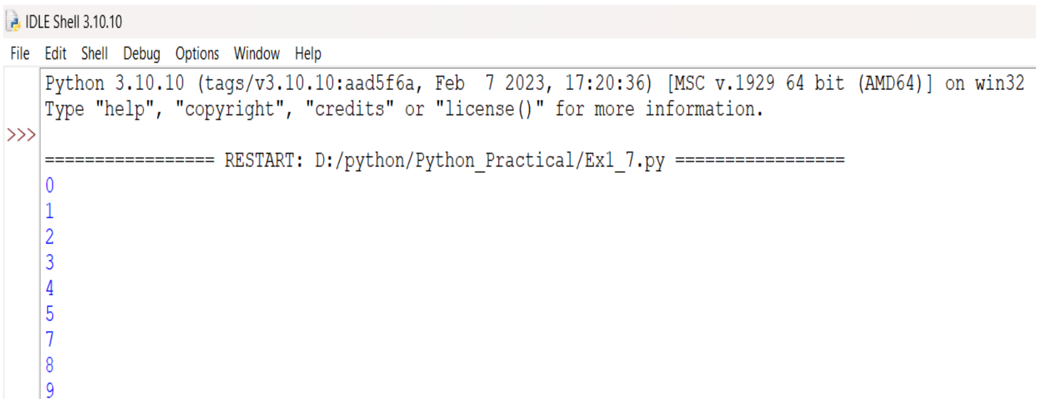


```
Python 3.10.10 (tags/v3.10.10:aad5f6a, Feb  7 2023, 17:20:36) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===================================================== RESTART: D:/python/Python_Practical/EX_1_6.py =======
0
1
2
```

- **#Example No 7**

```
for i in range(10):
    if i == 6:
        continue
    else:
        print(i,end="\n")
```

- **Output :**


IDLE Shell 3.10.10

File Edit Shell Debug Options Window Help

```
Python 3.10.10 (tags/v3.10.10:aad5f6a, Feb  7 2023, 17:20:36) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================= RESTART: D:/python/Python_Practical/Ex1_7.py =================
0
1
2
3
4
5
7
8
9
```

- **Conclusion :** By Studying This Practical ,We Are Able To Implement Python Programs Using Input Function.

Experiment No : 2

- **Title :** Implement Python Programs To Demonstrate Decision Making And Control Statements.

- **Aim :** Basic python programs helps to understand the decision making and control statements.

- **Theory :**

❖ Decision Making Statements :

Decision making statements in python are an essential tool for any programmer. They allow any program to make intelligent and informed decisions by evaluating information before executing that particular code. Decision making statements can direct a program to execute on the based of instructions if a certain condition is True or another if it's False. Different decision making structures can be used such as if statements, else-if statements, and nested if else statements for decision making in python programming language. Each decision structure has its own benefits and applications; however, decision making statements provide an invaluable tool for programming complex conditions in Python.

Types of decision-making in python :

There are three types of decision making statements in python language,

1. If statements in python
2. If else statement in python
3. Nested if else statement in python

❖ Control Statements :

Control statements in Python are used to manage the flow of execution of a program based on certain conditions.Control statements in Python are a powerful tool for managing the flow of execution. They allow developers to make decisions based on specific conditions and modify the normal sequential flow of a program. By using control statements effectively, developers can write more efficient and effective code.

- **Types Of Conditional Statements :**

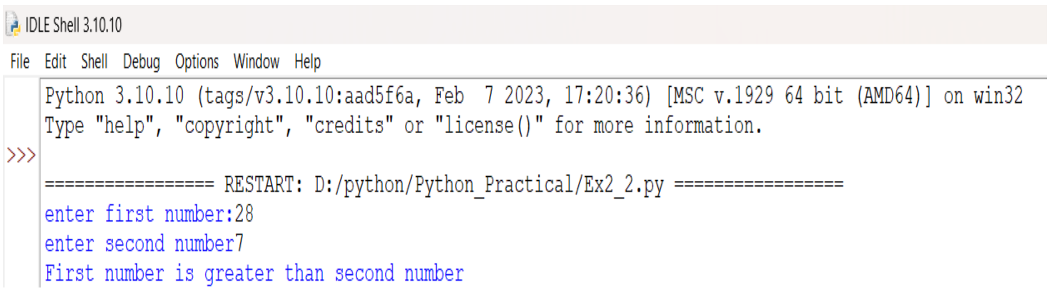
  1.Break Statements.

  2.Continue Statements

- **Program :**
- **#Example No.1**

```
a=input("enter a number:")

b=input("Enter another number")

if a>b:

    print("First number is grater than second number")

elif a==b:

    print("Both number is equal")

else:

    print("Second number is grater than first number")
```

- **Output :**

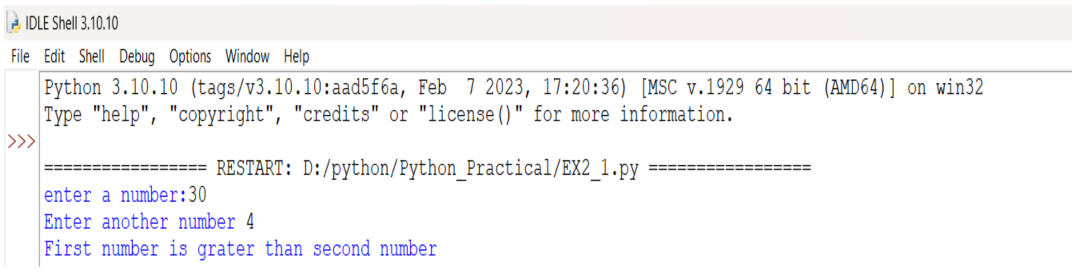


```
Python 3.10.10 (tags/v3.10.10:aad5f6a, Feb  7 2023, 17:20:36) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================= RESTART: D:/python/Python_Practical/Ex2_2.py =================
enter first number:28
enter second number7
First number is greater than second number
```

- **#Example No 2**

```
a=int(input("enter first number:"))
b=int(input("enter second number"))
if a>b:
    print("First number is greater than second number")

else:
    print("Second number is greater than first number")
```

- **Output :**

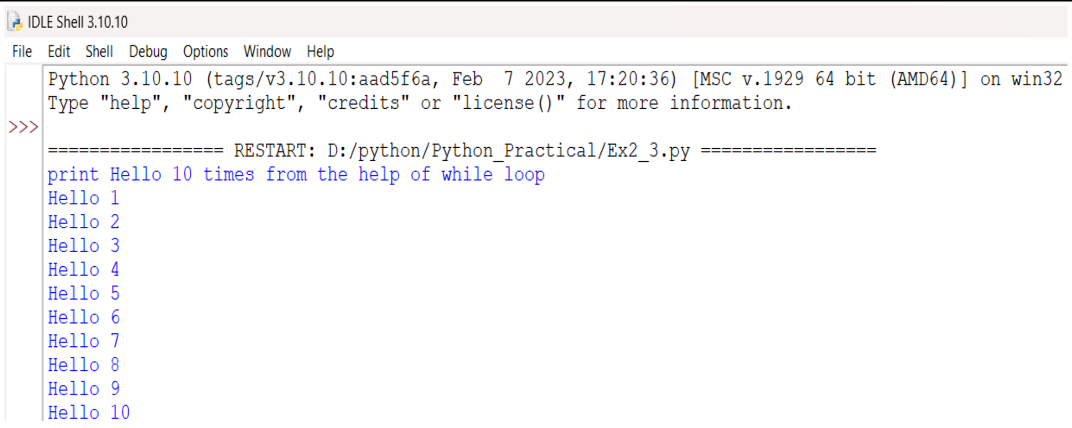


```
Python 3.10.10 (tags/v3.10.10:aad5f6a, Feb  7 2023, 17:20:36) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================= RESTART: D:/python/Python_Practical/EX2_1.py =================
enter a number:30
Enter another number 4
First number is grater than second number
```

- **#Example No 3**

```
print("print Hello 10 times from the help of while loop")
a=int(1)
while a<=10:
    print("Hello",+a)
    a=a+1
```

- **Output :**

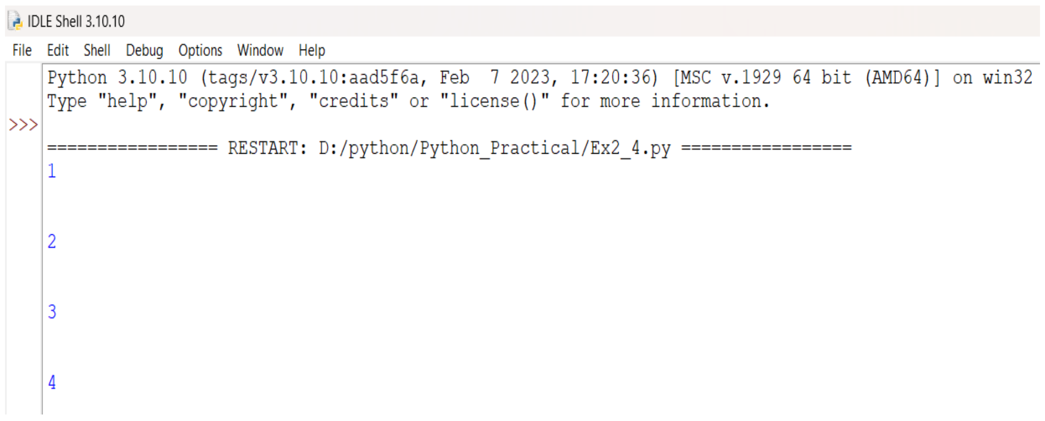


```
Python 3.10.10 (tags/v3.10.10:aad5f6a, Feb  7 2023, 17:20:36) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================= RESTART: D:/python/Python_Practical/Ex2_3.py =================
print Hello 10 times from the help of while loop
Hello 1
Hello 2
Hello 3
Hello 4
Hello 5
Hello 6
Hello 7
Hello 8
Hello 9
Hello 10
```

- **#Example No 4**

```
for i in range(1,5):
    print(i)
    print("\n")
```

- **Output :**

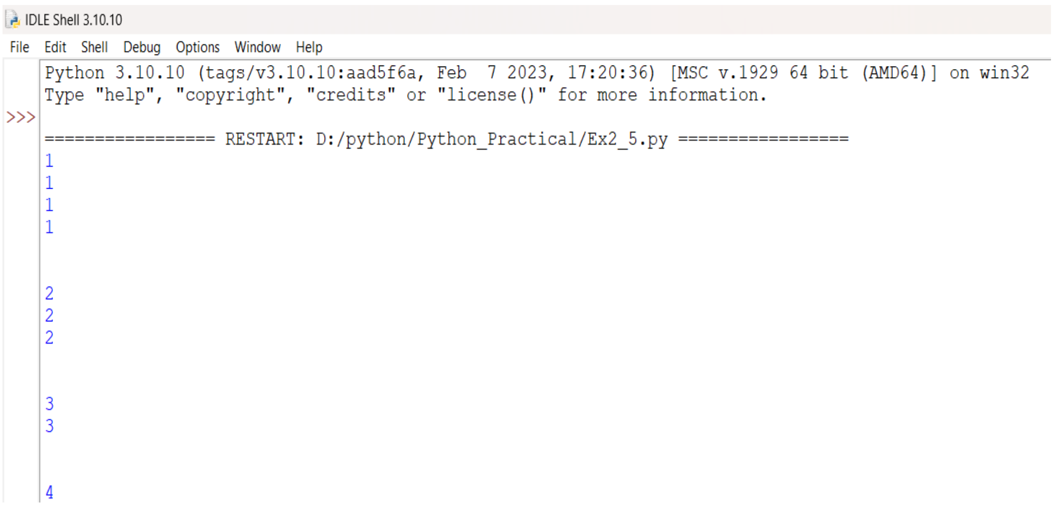


```
Python 3.10.10 (tags/v3.10.10:aad5f6a, Feb  7 2023, 17:20:36) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================= RESTART: D:/python/Python_Practical/Ex2_4.py =================
1


2


3


4
```

- **#Example No 5**

```
for i in range(1,5):
    for j in range(i,5):
        print(i)
print("\n")
```

- **Output :**



```
Python 3.10.10 (tags/v3.10.10:aad5f6a, Feb  7 2023, 17:20:36) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================= RESTART: D:/python/Python_Practical/Ex2_5.py =================
1
1
1
1


2
2
2


3
3


4
```

- **Conclusion :** By Studying This Practical ,We Are Able To Implement Python Program Using Decision Making And Control Statements.

Experiment No : 3

- **Title :** Apply Python Built-In-Data Types : Strings ,List ,Tuples ,Dictionary ,Set And Their Methods To Solve Any Given Problem

- **Aim :** Basic python programs to helps to understand python built-in-data-types and their methods
- **Theory :**

  A variable can contain a variety of values. On the other hand, a person's id must be stored as an integer, while their name must be stored as a string. The storage method for each of the standard data types that Python provides is specified by Python. The following is a list of the Python-defined data types.

1. Numbers
2. Sequence Type
3. Boolean
4. Set
5. Dictionary

 Numbers :

Numeric values are stored in numbers. The whole number, float, and complex qualities have a place with a Python Numbers datatype. Python offers the type() function to determine a variable's data type. The instance () capability is utilized to check whether an item has a place with a specific class.

Python supports three kinds of numerical data.

- **Int:** Whole number worth can be any length, like numbers 10, 2, 29, - 20, - 150, and so on. An integer can be any length you want in Python. Its worth has a place with int.

- **Float:** Float stores drifting point numbers like 1.9, 9.902, 15.2, etc. It can be accurate to within 15 decimal places.
- **Complex:** An intricate number contains an arranged pair, i.e., x + iy, where x and y signify the genuine and non-existent parts separately. The complex numbers like 2.14j, 2.0 + 2.3j, etc.

- **String :**The sequence of characters in the quotation marks can be used to describe the string. A string can be defined in Python using single, double, or triple quotes. String dealing with Python is a direct undertaking since Python gives worked-in capabilities and administrators to perform tasks in the string.

  - When dealing with strings, the operation "hello"+" python" returns "hello python," and the operator + is used to combine two strings.

    List : Lists in Python are like arrays in C, but lists can contain data of different types. The things put away in the rundown are isolated with a comma (,) and encased inside square sections [].

    To gain access to the list's data, we can use slice [:] operators. Like how they worked with strings, the list is handled by the concatenation operator (+) and the repetition operator (*).

  - **Tuple :**

    In many ways, a tuple is like a list. Tuples, like lists, also contain a collection of items from various data types. A parenthetical space () separates the tuple's components from one another.

    Because we cannot alter the size or value of the items in a tuple, it is a read-only data structure.

 Dictionary :

A dictionary is a key-value pair set arranged in any order. It stores a specific value for each key, like an associative array or a hash table. Value is any Python object, while the key can hold any primitive data type.

The comma (,) and the curly braces are used to separate the items in the dictionary.

 Boolean :

True and False are the two default values for the Boolean type. These qualities are utilized to decide the given assertion valid or misleading. The class book indicates this. False can be represented by the 0 or the letter "F," while true can be represented by any value that is not zero.

 Set:

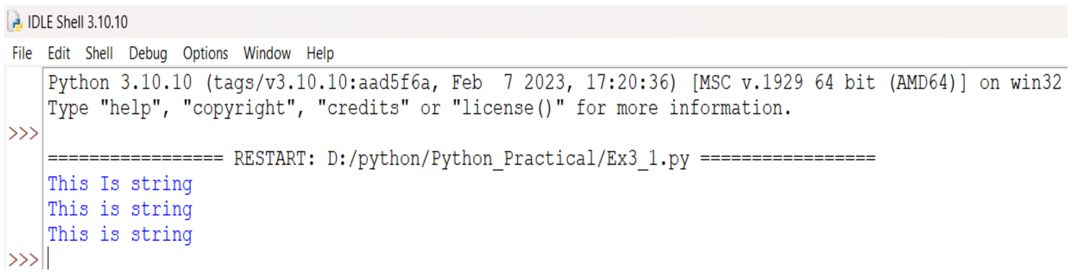
The data type's unordered collection is Python Set. It is iterable, mutable(can change after creation), and has remarkable components. The elements of a set have no set order; It might return the element's altered sequence. Either a sequence of elements is passed through the curly braces and separated by a comma to create the set or the built-in function set() is used to create the set. It can contain different kinds of values.

- **Program :**
- **#Example No.1**

```
a = "This Is string"

print(a)

b="This is string"
```

```
print(b)

c="This is string"

print(c)
```

- **Output :**

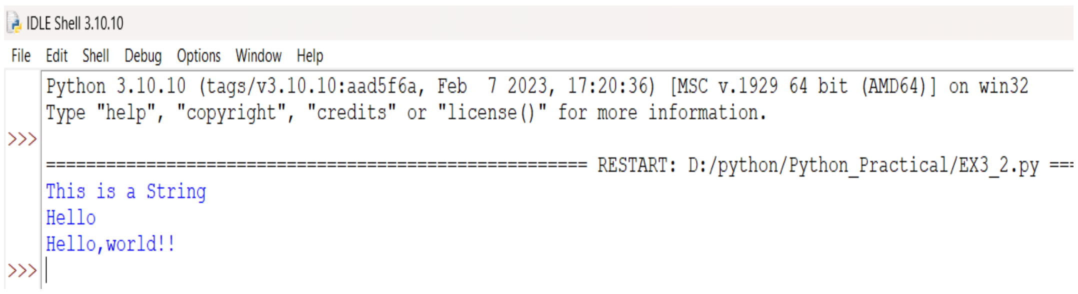


```
Python 3.10.10 (tags/v3.10.10:aad5f6a, Feb  7 2023, 17:20:36) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================= RESTART: D:/python/Python_Practical/Ex3_1.py =================
This Is string
This is string
This is string
>>>
```

- **#Example No 2**

```
a = "This is a String"
print(a)
b = "Hello"
print(b)
c = "Hello,world!!"
print(c)
```

- **Output :**

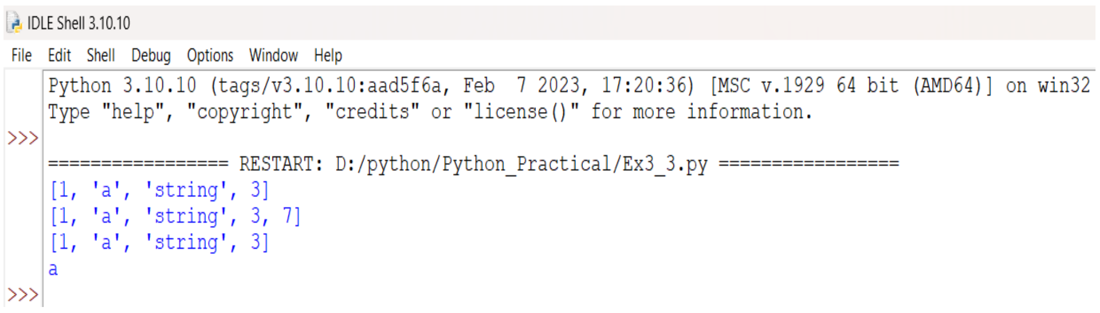


```
Python 3.10.10 (tags/v3.10.10:aad5f6a, Feb  7 2023, 17:20:36) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==================================================== RESTART: D:/python/Python_Practical/EX3_2.py ==
This is a String
Hello
Hello,world!!
>>>
```

- **#Example No 3**

```
l = [1,'a',"string",1+2]
print(l)
l.append(7)
print(l)
l.pop()
print(l)
print(l[1])
```

- **Output :**

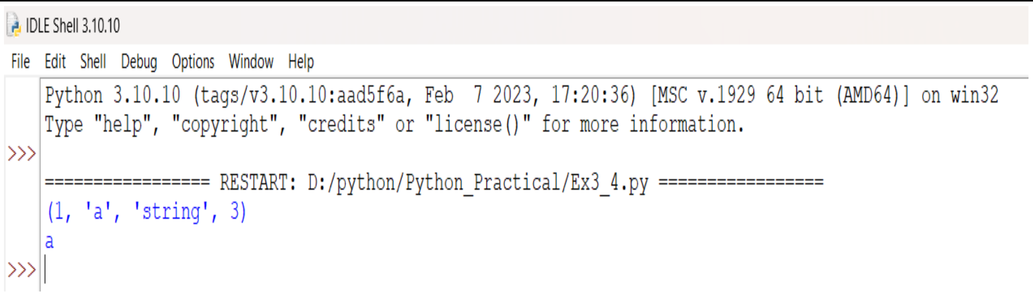


```
Python 3.10.10 (tags/v3.10.10:aad5f6a, Feb  7 2023, 17:20:36) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================= RESTART: D:/python/Python_Practical/Ex3_3.py =================
[1, 'a', 'string', 3]
[1, 'a', 'string', 3, 7]
[1, 'a', 'string', 3]
a
>>>
```

- **#Example No 4**

```
tup = (1,'a','string',1+2)
print(tup)
print(tup[1])
```

- **Output :**



```
Python 3.10.10 (tags/v3.10.10:aad5f6a, Feb  7 2023, 17:20:36) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================= RESTART: D:/python/Python_Practical/Ex3_4.py =================
(1, 'a', 'string', 3)
a
>>>
```

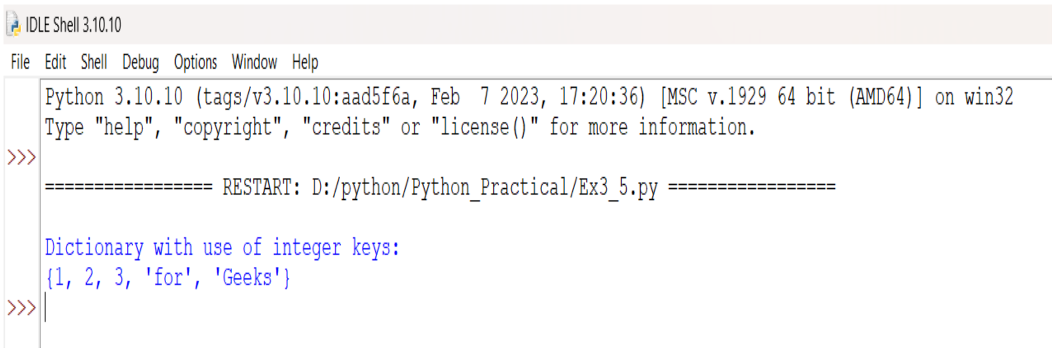
- **#Example No 5**

  dict={1,'Geeks',2,'for',3,'Geeks'}

  print("\nDictionary with use of integer keys:")

  print(dict)

- **Output :**



```
Python 3.10.10 (tags/v3.10.10:aad5f6a, Feb  7 2023, 17:20:36) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================= RESTART: D:/python/Python_Practical/Ex3_5.py =================

Dictionary with use of integer keys:
{1, 2, 3, 'for', 'Geeks'}
>>>
```

- **#Example No 6**

  set1 = set([1,2,3,4,4,5,3,3,6,])

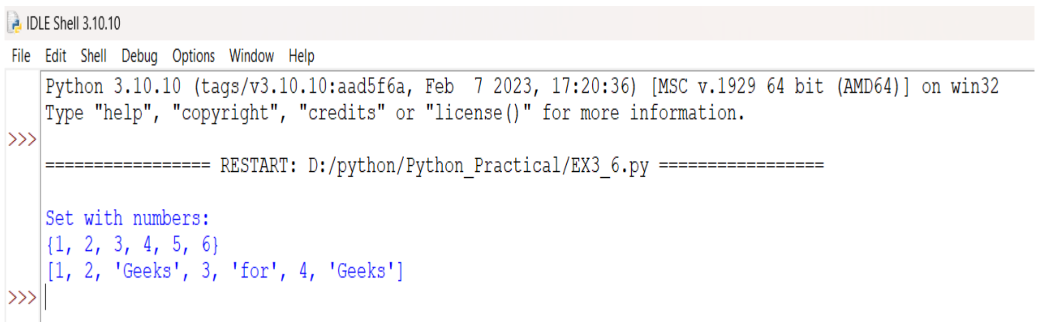
  print("\nSet with numbers: ")

  print(set1)

  set2 = ([1,2,'Geeks',3,'for',4,'Geeks'])

  print(set2)

- **Output :**



```
Python 3.10.10 (tags/v3.10.10:aad5f6a, Feb  7 2023, 17:20:36) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 
================= RESTART: D:/python/Python_Practical/EX3_6.py =================

Set with numbers:
{1, 2, 3, 4, 5, 6}
[1, 2, 'Geeks', 3, 'for', 4, 'Geeks']
>>> 
```

- **Conclusion :** By Studying This Practical ,We Are Able To Implement Python Program Using Built-In-Data-Types And Their Methods.

Experiment No : 4

- **Title :** Implement OOP Concepts Like Data Hiding And Data Abstraction.
- **Aim :** Basic python programs to helps to understand the concept of data hiding and data abstraction .
- **Theory :**

- **Abstraction:**

  Abstraction is a process of hiding unnecessary data and showing only relevant data. Out of an ocean of data, we are only maintaining the transparency of some data to the user. This important concept in object-oriented programming will reduce the complexity of the code and increases the readability.

  For example, let’s say we were assigned a task to create an online course enrollment portal, and the data fields available to us are {name, age, current occupation, college name, course name, payment method, sibling’s name, marital status, vehicle number}.

- **Encapsulation :**

  Encapsulation is binding the data members with member variables. This will avoid the direct access of variables, because direct access of variables may violate privacy, and hiding of the implementation will not be possible.

  Encapsulation minimizes your code’s part revealed to the user. The user can be anyone who uses your published code or perhaps your code’s remaining part.

  Encapsulation works like a protective wrapper that conceals the code and data within the class. That data and code will be accessed outside the method/member function and the class that is not the members of that class.

We may have gone through some classic methods in class like set and get, where the set method is used to update or allocate a value to a variable and the get method is used to read or retrieve the value of a variable. Here we can directly access the variables using the object of that class, but if we want to make a variable private then we should use these settings and get methods.

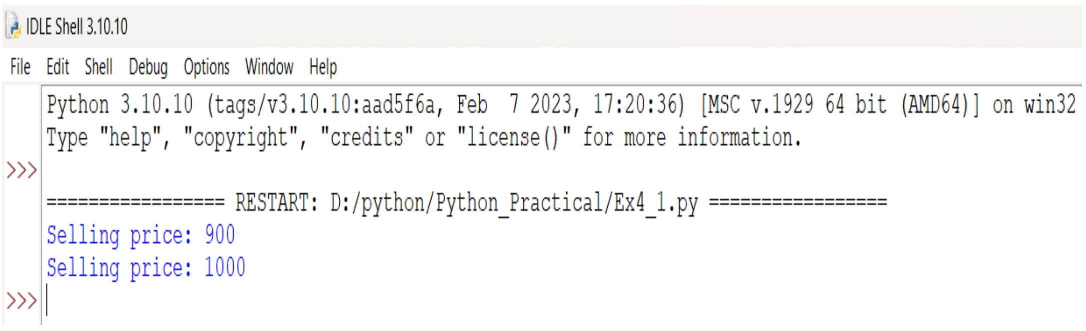
The concept is simple, we’ll make the set and get methods public and the variables are private. So the variables can be accessed only through the public methods outside the class since private objects are not accessible outside class but accessible inside class. This concept of binding or bundling variables with methods is called encapsulation.

- **Program:**
- **#Example No.1**

```
class Computer:
    def __init__(self):
        self._maxprice = 900
    def sell(self):
        print("Selling price: {}".format(self._maxprice))
    def set_max_price(self, price):
        self._maxprice = price


c = Computer()
c.sell()
c.set_max_price(1000)
```

```
c.sell()
```

- **Output :**

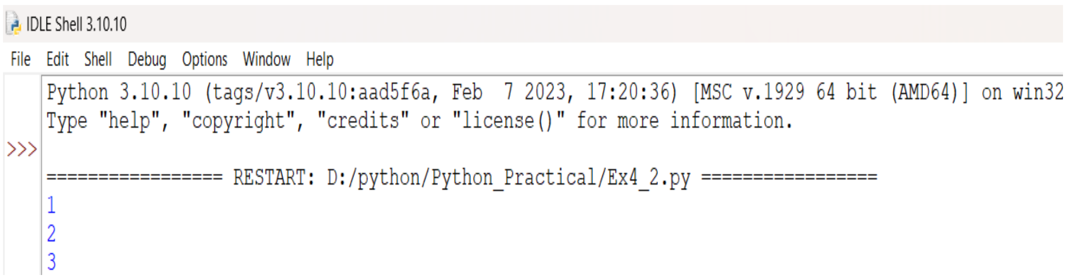


```
Python 3.10.10 (tags/v3.10.10:aad5f6a, Feb  7 2023, 17:20:36) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================= RESTART: D:/python/Python_Practical/Ex4_1.py =================
Selling price: 900
Selling price: 1000
>>>
```

- **#Example No 2**

```
class Solution:
    def __init__(self):
        self.__privatecounter = 0

    def sum(self):
        self.__privatecounter += 1
        print(self.__privatecounter)

count = Solution()
count.sum()
count.sum()
count.sum()
```

- **Output :**

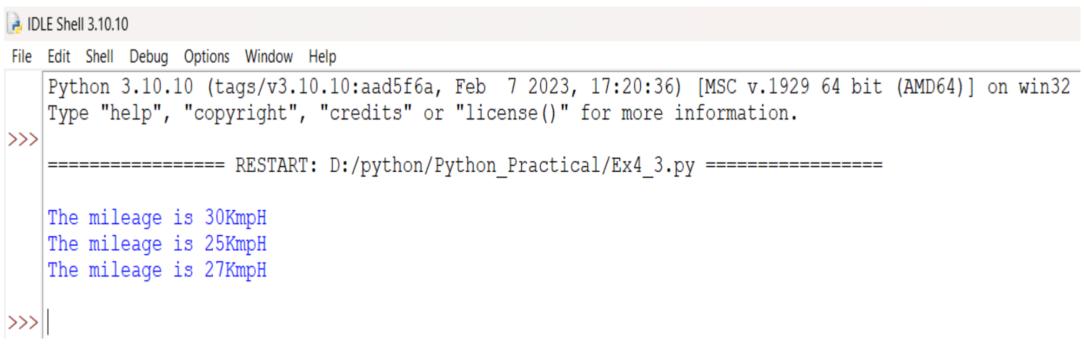


```
Python 3.10.10 (tags/v3.10.10:aad5f6a, Feb  7 2023, 17:20:36) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================= RESTART: D:/python/Python_Practical/Ex4_2.py =================
1
2
3
```

- **#Example No 3**

```
from abc import ABC
class car(ABC):
    def mileage(self):
        pass
class Tesla(car):
    def mileage(self):
        print("\nThe mileage is 30KmpH")
class suzuki(car):
    def mileage(self):
        print("The mileage is 25KmpH")
class renault(car):
    def mileage(self):
        print("The mileage is 27KmpH\n")
t = Tesla()
t.mileage()
s = suzuki()
s.mileage()
r = renault()
r.mileage()
```

- **Output :**



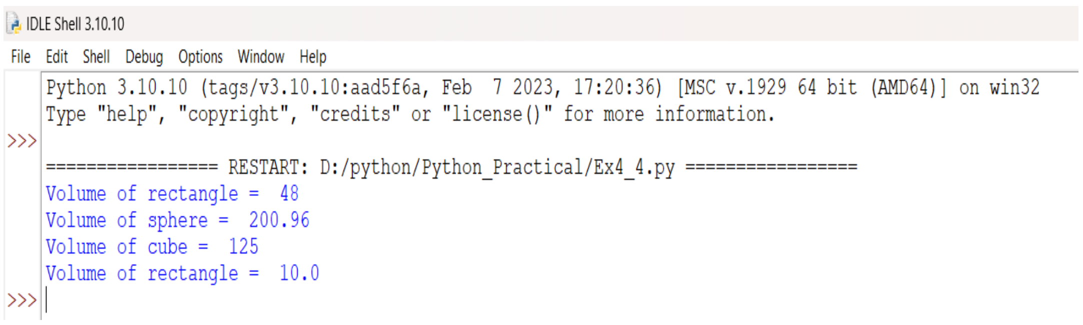
```
Python 3.10.10 (tags/v3.10.10:aad5f6a, Feb  7 2023, 17:20:36) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================ RESTART: D:/python/Python_Practical/Ex4_3.py ================

The mileage is 30KmpH
The mileage is 25KmpH
The mileage is 27KmpH

>>>
```

- **#Example No 4**

```
from abc import ABC
class rect():
    length = 4
    width = 6
    height = 2
    def volume(self):
        return self.length*self.width*self.height
class sphere():
    radius = 8
    def volume(self):
        return 3.14*self.radius*self.radius
class cube():
    edge = 5
    def volume(self):
        return self.edge*self.edge*self.edge
class rectangle_3D():
    length = 5
    width = 4
    def volume(self):
        return 0.5*self.length*self.width
rr = rect()
ss = sphere()
cc = cube()
```

```
tt = rectangle_3D()

print("Volume of rectangle = ",rr.volume())

print("Volume of sphere = ",ss.volume())

print("Volume of cube = ",cc.volume())

print("Volume of rectangle = ",tt.volume())
```

- **Output :**



```
Python 3.10.10 (tags/v3.10.10:aad5f6a, Feb  7 2023, 17:20:36) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================= RESTART: D:/python/Python_Practical/Ex4_4.py =================
Volume of rectangle =  48
Volume of sphere =  200.96
Volume of cube =  125
Volume of rectangle =  10.0
>>>
```

- **Conclusion :** By Studying This Practical ,We Are Able To Implement Python Program Using Concepts Like Data Hiding And Data Abstraction.

Experiment No : 5

- **Title :** Create User-Defined Functions With Different Types Of Function Arguments.
- **Aim :** Basic python programs to helps to understand the user-defined functions with different types of function arguments.
- **Theory :**

□ **Python User-defined functions**

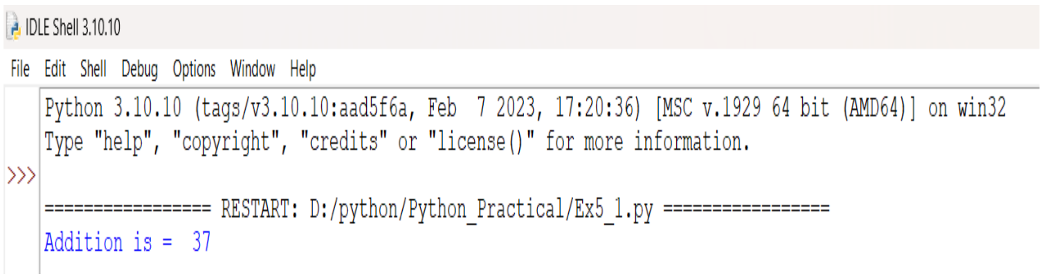
All the functions that are written by any of us come under the category of user-defined functions. Below are the steps for writing user-defined functions in Python.

- In Python, a def keyword is used to declare user-defined functions.
- An indented block of statements follows the function name and arguments which contains the body of the function.

- **Program:**
- **#Example No.1**

```
def add(a,b):
    sum = a+b
    print("Addition is = ",sum)
    print("\n")
add(30,7)
```

- **Output :**

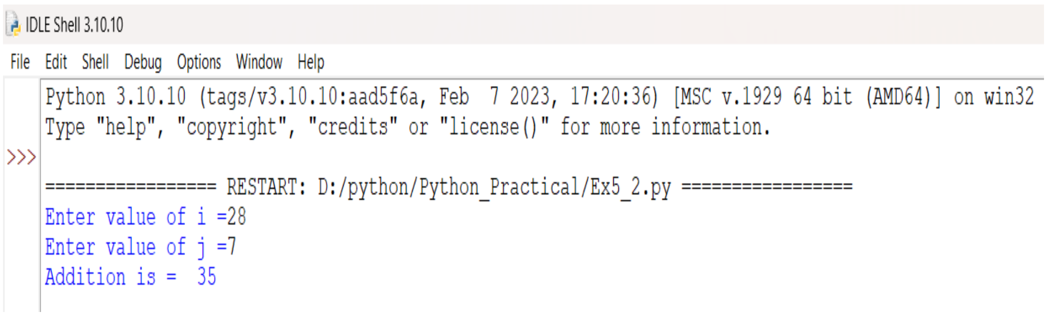


```
Python 3.10.10 (tags/v3.10.10:aad5f6a, Feb  7 2023, 17:20:36) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================= RESTART: D:/python/Python_Practical/Ex5_1.py =================
Addition is =  37
```

- **#Example No 2**

```
def sum(i,j):
    addition = i+j
    print("Addition is = ",addition)
    print("\n")

i = int(input("Enter value of i ="))
j = int(input("Enter value of j ="))
sum(i,j)
```

- **Output :**

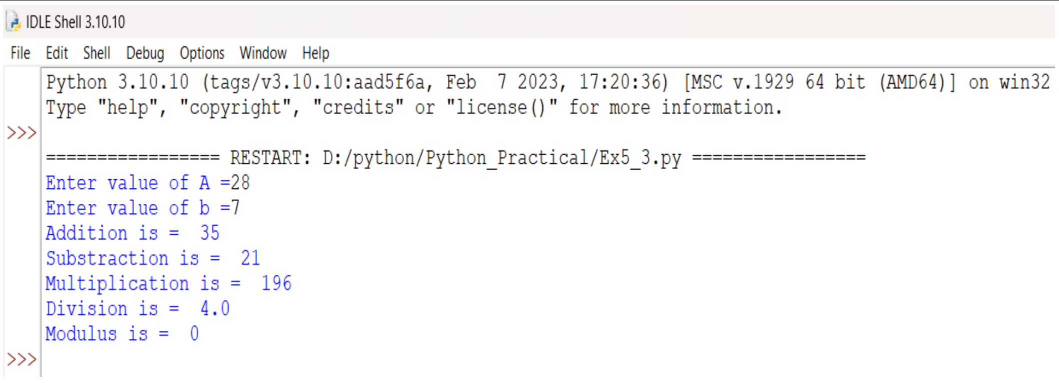


```
Python 3.10.10 (tags/v3.10.10:aad5f6a, Feb  7 2023, 17:20:36) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================= RESTART: D:/python/Python_Practical/Ex5_2.py =================
Enter value of i =28
Enter value of j =7
Addition is =  35
```

- **#Example No 3**

```
def Addition(a,b):
    add = a+b
    print("Addition is = ",add)

def Substraction(a,b):
    sub = a-b
    print("Substraction is = ",sub)

def Multiplication(a,b):
    mul = a*b
    print("Multiplication is = ",mul)

def Division(a,b):
    div = a/b
    print("Division is = ",div)

def Modulus(a,b):
    mod = a%b
    print("Modulus is = ",mod)

a = int(input("Enter value of A ="))
b = int(input("Enter value of b ="))

Addition(a,b)
Substraction(a,b)
Multiplication(a,b)
Division(a,b)
Modulus(a,b)
```

- **Output :**



```
Python 3.10.10 (tags/v3.10.10:aad5f6a, Feb  7 2023, 17:20:36) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================= RESTART: D:/python/Python_Practical/Ex5_3.py =================
Enter value of A =28
Enter value of b =7
Addition is =  35
Substraction is =  21
Multiplication is =  196
Division is =  4.0
Modulus is =  0
>>>
```

- **Conclusion :** By Studying This Practical ,We Are Able To Create User Defined Functions With Different Forms Of Function Arguments

Experiment No : 6

- **Title :** Perform File Manipulation Operations – Open, Read, Write, Append, Copy From One File To Another.

- **Aim :** Basic python programs to helps to understand the file manipulation operations.

- **Theory :**

- **Python File Handling:**

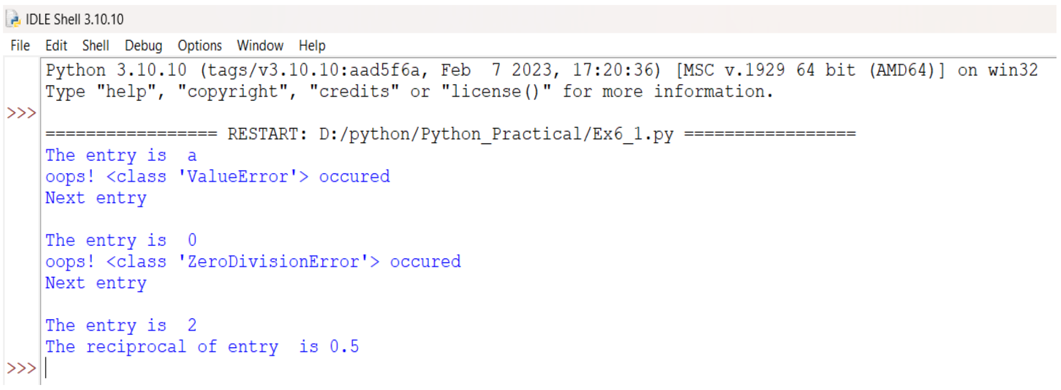
Python too supports file handling and allows users to handle files i.e., to read and write files, along with many other file handling options, to operate on files. The concept of file handling has stretched over various other languages, but the implementation is either complicated or lengthy, but like other concepts of Python, this concept here is also easy and short. Python treats files differently as text or binary and this is important. Each line of code includes a sequence of characters and they form a text file. Each line of a file is terminated with a special character, called the EOL or End of Line characters like comma {,} or newline character. It ends the current line and tells the interpreter a new one has begun. Let's start with the reading and writing files.

- **Program:**

- **#Example No.1**

```
import sys

randomlist= ['a',0,2]

for entry in randomlist:

   try:

      print("The entry is ",entry)

      r=1/int(entry)

      break

   except:
      print("oops!",sys.exc_info()[0],"occured")

      print("Next entry")

      print()

print("The reciprocal of entry  is",r)
```

- **Output :**

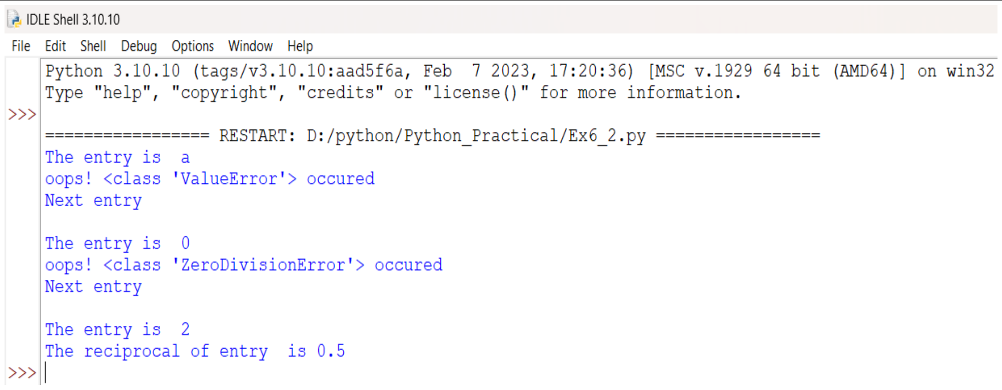

IDLE Shell 3.10.10
File Edit Shell Debug Options Window Help

```
Python 3.10.10 (tags/v3.10.10:aad5f6a, Feb  7 2023, 17:20:36) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================= RESTART: D:/python/Python_Practical/Ex6_1.py =================
The entry is  a
oops! <class 'ValueError'> occured
Next entry

The entry is  0
oops! <class 'ZeroDivisionError'> occured
Next entry

The entry is  2
The reciprocal of entry  is 0.5
>>>
```

- **#Example No 2**

```
import sys
randomlist= ['a',0,2]
for entry in randomlist:
    try:
        print("The entry is ",entry)
        r=1/int(entry)
        break
    except Exception as e:
        print("oops!",e.__class__,"occured")
        print("Next entry")
        print()
print("The reciprocal of entry  is",r)
```

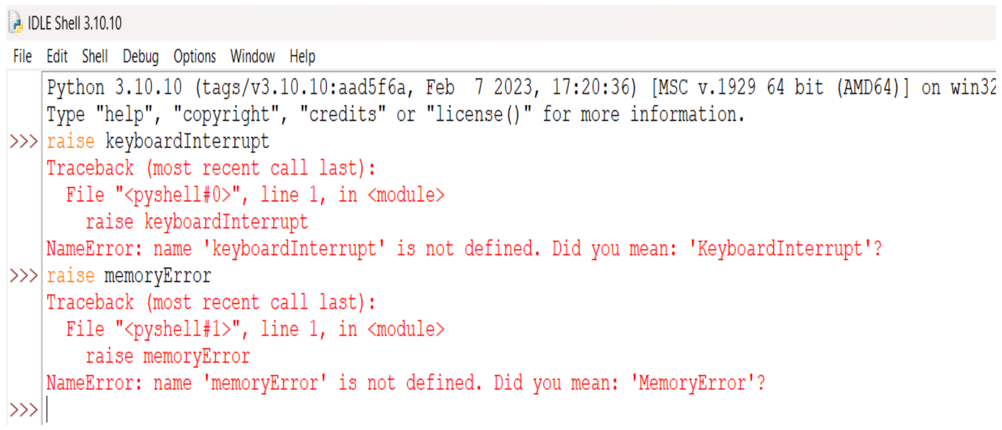
- **Output :**



```
Python 3.10.10 (tags/v3.10.10:aad5f6a, Feb  7 2023, 17:20:36) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================= RESTART: D:/python/Python_Practical/Ex6_2.py =================
The entry is  a
oops! <class 'ValueError'> occured
Next entry

The entry is  0
oops! <class 'ZeroDivisionError'> occured
Next entry

The entry is  2
The reciprocal of entry  is 0.5
>>>
```

- **#Example No 3 :**

- **Output :**

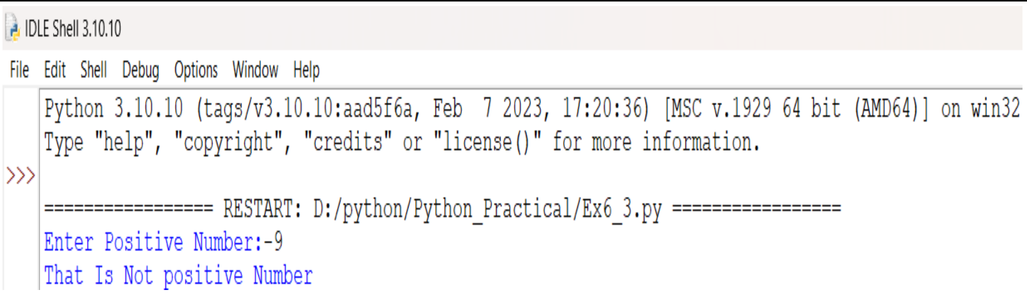


```
Python 3.10.10 (tags/v3.10.10:aad5f6a, Feb  7 2023, 17:20:36) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> raise keyboardInterrupt
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    raise keyboardInterrupt
NameError: name 'keyboardInterrupt' is not defined. Did you mean: 'KeyboardInterrupt'?
>>> raise memoryError
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    raise memoryError
NameError: name 'memoryError' is not defined. Did you mean: 'MemoryError'?
>>>
```

- **#Example No 4 :**

```
try:
    a=int(input("Enter Positive Number:"))
    if a<=0:
        raise ValueError("That Is Not positive Number")
except ValueError as ve:
    print(ve)
```

- **Output :**



```
Python 3.10.10 (tags/v3.10.10:aad5f6a, Feb  7 2023, 17:20:36) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================= RESTART: D:/python/Python_Practical/Ex6_3.py =================
Enter Positive Number:-9
That Is Not positive Number
```

- **Conclusion :** By Studying This Practical ,We Are Able To Perform File Manipulation Operations.

Experiment No : 7

- **Title :** Handle Exception Using Python Built-In Exceptions
- **Aim :** Basic python programs to helps to understand the built-in exceptions.
- **Theory :**

 Python Exception Handling :

In Python, there are several built-in Python exceptions that can be raised when an error occurs during the execution of a program. Here are some of the most common types of exceptions in Python:

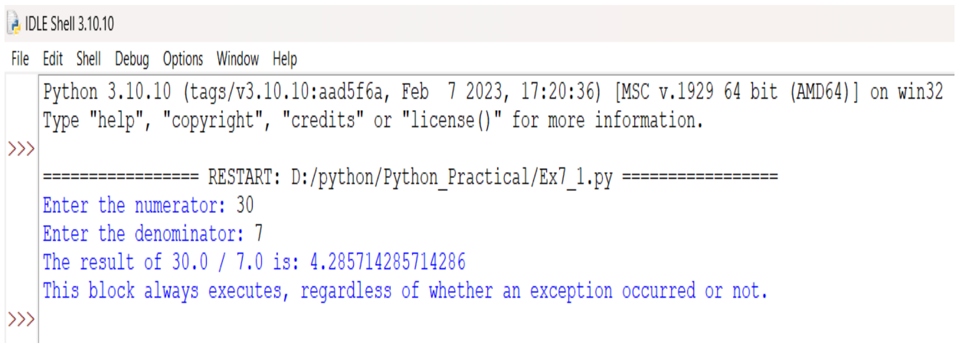
- **SyntaxError:** This exception is raised when the interpreter encounters a syntax error in the code, such as a misspelled keyword, a missing colon, or an unbalanced parenthesis.
- **TypeError**: This exception is raised when an operation or function is applied to an object of the wrong type, such as adding a string to an integer.
- **NameError**: This exception is raised when a variable or function name is not found in the current scope.
- **IndexError**: This exception is raised when an index is out of range for a list, tuple, or other sequence types.
- **KeyError**: This exception is raised when a key is not found in a dictionary.
- **ValueError**: This exception is raised when a function or method is called with an invalid argument or input, such as trying to convert a string to an integer when the string does not represent a valid integer.
- **AttributeError**: This exception is raised when an attribute or method is not found on an object, such as trying to access a non-existent attribute of a class instance.
- **IOError**: This exception is raised when an I/O operation, such as reading or writing a file, fails due to an input/output error.
- **ZeroDivisionError**: This exception is raised when an attempt is made to divide a number by zero.
- **ImportError**: This exception is raised when an import statement fails to find or load a module.

- **Program:**

- **#Example No.1 :**

```
def divide_numbers(x, y):

    try:

        result = x / y

    except ZeroDivisionError:

        print("Error: Division by zero is not allowed.")

    except TypeError:

        print("Error: Please provide valid numeric values.")

    else:

        print(f"The result of {x} / {y} is: {result}")

    finally:

        print("This block always executes, regardless of whether an exception
occurred or not.")



try:

    num1 = float(input("Enter the numerator: "))

    num2 = float(input("Enter the denominator: "))

    divide_numbers(num1, num2)

except ValueError:

    print("Error: Please enter valid numeric values for the numerator and
denominator.")

except Exception as e:

    print(f"An unexpected error occurred: {e}")
```

- **Output :**



```
Python 3.10.10 (tags/v3.10.10:aad5f6a, Feb  7 2023, 17:20:36) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================= RESTART: D:/python/Python_Practical/Ex7_1.py =================
Enter the numerator: 30
Enter the denominator: 7
The result of 30.0 / 7.0 is: 4.285714285714286
This block always executes, regardless of whether an exception occurred or not.
>>>
```

- **Conclusion :** By Studying This Practical ,We Are Able To Perform Exception Handling In Python.

Experiment No : 8

- **Title :** Implement Various In-Built Functions Of Numpy Library
- **Aim :** Basic python programs helps to understand the Numpy library.
- **Theory :**

Numpy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays. It is the fundamental package for scientific computing with Python.
Besides its obvious scientific uses, Numpy can also be used as an efficient multi-dimensional container of generic data.

➢ **Arrays in Numpy**

Array in Numpy is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers. In Numpy, number of dimensions of the array is called rank of the array.A tuple of integers giving the size of the array along each dimension is known as shape of the array. An array class in Numpy is called as **ndarray**. Elements in Numpy arrays are accessed by using square brackets and can be initialized by using nested Python Lists.
Arrays in Numpy can be created by multiple ways, with various number of Ranks, defining the size of the Array. Arrays can also be created with the use of various data types such as lists, tuples, etc. The type of the resultant array is deduced from the type of the elements in the sequences.

- **Program :**
- **#Example No.1**

```
import numpy

arr = numpy.array([1,2,3,4,5])

print(arr)
```

- **Output :**

```
[1 2 3 4 5]
```

- **#Example No.2**

```
import numpy as np

arr = np.array([1,2,3,4,5])

print(arr)
```

- **Output :**

```
[1 2 3 4 5]
```

- **#Example No.3**

```
import numpy as np

print(np.__version__)
```

- **Output :**

```
1.26.2
```

- **#Example No.4**

```
import numpy as np

arr = np.array([1,2,3,4,5])

print(arr)
print(type(arr))
```

- **Output :**

```
[1 2 3 4 5]
<class 'numpy.ndarray'>
```

- **#Example No.5**

```
import numpy as np

arr = np.array(42)

print(arr)
```

- **Output :**

```
42
```

- **#Example No.6**

```
import numpy as np

arr = np.array([1,2,3,4,5,6])

print(arr)
```

- **Output :**

```
[1 2 3 4 5 6]
```

- **#Example No.7**

```
import numpy as np

arr = np.array([[1,2,3],[4,5,6]])

print(arr)
```

- **Output :**

```
[[1 2 3]
        [4 5 6]]
```

- **#Example No.8**

```
import numpy as np

arr = np.array([[[1,2,3],[4,5,6],[1,2,3],[4,5,6]]])
```

```
print(arr)
```

- **Output :**

```
[[[1 2 3]
   [4 5 6]
   [1 2 3]
   [4 5 6]]]
```

- **#Example No.9**

```
import numpy as np

a = np.array(42)

b = np.array([1,2,3,4,5])

c = np.array([[1,2,3],[4,5,6]])

d = np.array([[[1,2,3],[4,5,6]]])

print(a.ndim)

print(b.ndim)

print(c.ndim)

print(d.ndim)
```

- **Output :**

```
0
1
2
3
```

- **#Example No.10**

```
import numpy as np

arr = np.array([1,2,3,4,5])
```

```
print(arr[0])
```

- **Output :**

```
1
```

- **Conclusion :** By Studying This Practical ,We Are Able To Implement Program Using Numpy Library.

Experiment No : 9

- **Title :** Create Pandas Series And DataFrame From Varoius Inputs
- **Aim :** Basic python programs helps to understand the concept of panda series and dataframe
- **Theory :**

Series is a type of list in Pandas that can take integer values, string values, double values, and more. But in Pandas Series we return an object in the form of a list, having an index starting from 0 to n, Where n is the length of values in the series. Later in this article, we will discuss Dataframes in pandas, but we first need to understand the main difference between Series and Dataframe.Series can only contain a single list with an index, whereas Dataframe can be made of more than one series or we can say that a Dataframe is a collection of series that can be used to analyze the data.

Following is a syntax:

Syntax:

```
pandas.Series(data=None, index=None, dtype=None, name=None, copy=False, fastpath=False)
```

Following are the different parameters with description:

Parameter	Description
data	The values of the series are declared here.
index	Mentions the index values for the above-given values, the index values need not be unique. The default range of the index value is (0, 1, 2, …. N). The values of the index need to be the same length of the data being used.
dtype	The data type for the output Series. If not specified, this will be inferred from the *data*.
name	Name of the series
copy	Used to represent whether it needs to be copied or not

- **Program :**
- **#Example No.1**

```
import pandas as pd

Temp_List1 = [2,4,56,7,8,8]

pd_series = pd.Series([])

print(" Created Empty Series",pd_series)
```

```
pd_series = Temp_List1.copy()

print(" Print the entire series",pd_series)

print(" Print the first element of the Series",pd_series[0])
```

- **Output :**

```
Created Empty Series Series([], dtype: object)
Print the entire series [2, 4, 56, 7, 8, 8]
Print the first element of the Series 2
```

- **#Example No.1**

```
import pandas as pd

List = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

index = [101, 102, 103, 104, 105, 106, 107, 108, 109, 110]

dict = { 1:'Test_Value1', 2:'Test_Value2', 3:'Test_Value3'}

pd_series_default_index = pd.Series(List)

print(" Pandas Series with default index: \n",pd_series_default_index)

pd_series_manipulated_index = pd.Series(List,index)

print(" Pandas Series with Manipulated index: \n",pd_series_manipulated_index)

pd_series_manipulated_data_type = pd.Series(List,dtype=str)

print(" Pandas Series with Manipulated index: \n",type(pd_series_manipulated_data_type))

print(" Print type of first element: \n",type(pd_series_manipulated_data_type[0]))

pd_series_dict = pd.Series(dict)
```

```
print(" Pandas series with dicitionary type: \n",pd_series_dict)

print(" Pandas series with dicitionary type: \n",pd_series_dict[3])
```

- **Output :**

```
Pandas Series with default index:
 0     1
1     2
2     3
3     4
4     5
5     6
6     7
7     8
8     9
9    10
dtype: int64
 Pandas Series with Manipulated index:
 101     1
102     2
103     3
104     4
105     5
106     6
107     7
108     8
109     9
110    10
dtype: int64
 Pandas Series with Manipulated index:
 <class 'pandas.core.series.Series'>
 Print type of first element:
 <class 'str'>
 Pandas series with dicitionary type:
 1    Test_Value1
2    Test_Value2
3    Test_Value3
dtype: object
 Pandas series with dicitionary type:
 Test_Value3
```

- **Conclusion :** By Studying This Practical ,We Are Able To Implement Program Using Pandas Series And DataFrame.

Experiment No : 10

- **Title :** Import Any CSV File To Pandas Dataframe And Perform The Following
  a) Visualize The First And Last 10 Records.
  b) Get The Shape, Index, Columns Based On Operation.
  c) Select/Delete The records Based On The Conditions.
  d) Perform Ranking And Sorting Operations.
  e) Do Required Statical Operations On The Given Columns
  f) Find The Count And Uniqueness Of The Given Categorical Values.
  g) Rename Single/Multiple Columns.

- **Aim :** Basic python programs helps to understand the fundamental concepts of importing CSV file

- **Theory :**

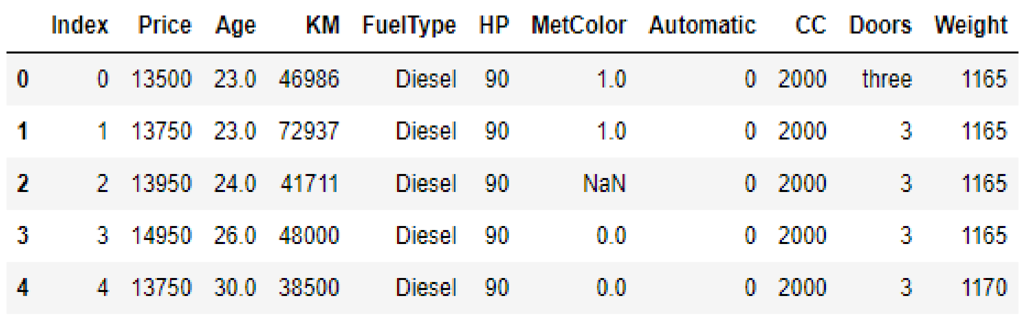
CSV files are the "comma separated values", these values are separated by commas, this file can be viewed as an Excel file. In Python, Pandas is the most important library coming to data science. We need to deal with huge datasets while analyzing the data, which usually can be in CSV file format. Let's see the different ways to import csv files in Pandas.

- **Ways to Import CSV File in Pandas**
  There are various ways to import CSV files in Pandas, here we are discussing some generally used methods for importing CSV files in pandas.
  - Using read_csv() Method
  - Using csv Module.
  - Using numpy Module

- **Program :**
- **#Example No.1**

```
import pandas as pd
nifty=pd.read_csv("F:\pandas\Toyota.csv")
nifty.head()
```

- **Output :**

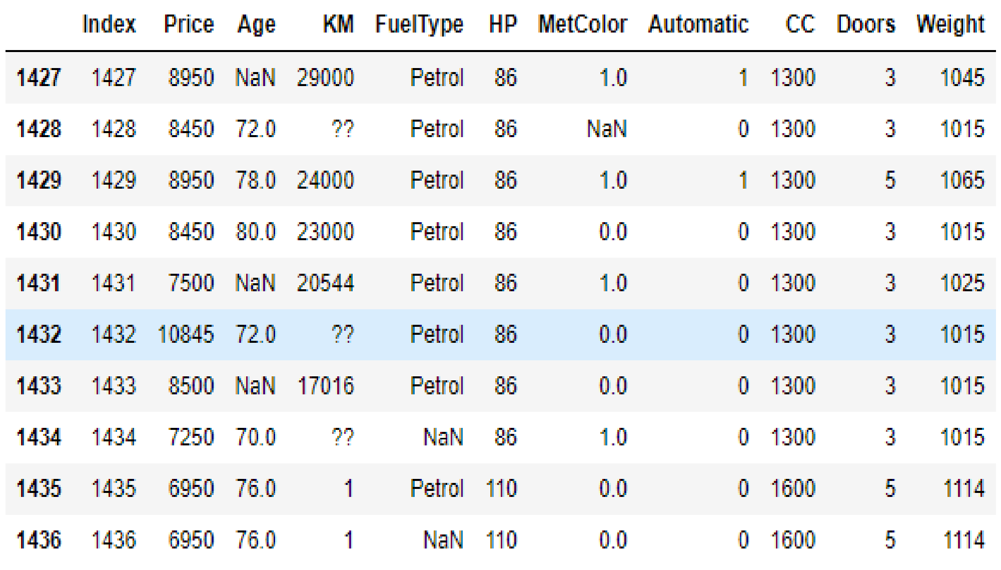


	Index	Price	Age	KM	FuelType	HP	MetColor	Automatic	CC	Doors	Weight
0	0	13500	23.0	46986	Diesel	90	1.0	0	2000	three	1165
1	1	13750	23.0	72937	Diesel	90	1.0	0	2000	3	1165
2	2	13950	24.0	41711	Diesel	90	NaN	0	2000	3	1165
3	3	14950	26.0	48000	Diesel	90	0.0	0	2000	3	1165
4	4	13750	30.0	38500	Diesel	90	0.0	0	2000	3	1170

- **#Example No.2**

  nifty.head(10)
  nifty.tail(10)

- **Output :**



	Index	Price	Age	KM	FuelType	HP	MetColor	Automatic	CC	Doors	Weight
1427	1427	8950	NaN	29000	Petrol	86	1.0	1	1300	3	1045
1428	1428	8450	72.0	??	Petrol	86	NaN	0	1300	3	1015
1429	1429	8950	78.0	24000	Petrol	86	1.0	1	1300	5	1065
1430	1430	8450	80.0	23000	Petrol	86	0.0	0	1300	3	1015
1431	1431	7500	NaN	20544	Petrol	86	1.0	0	1300	3	1025
1432	1432	10845	72.0	??	Petrol	86	0.0	0	1300	3	1015
1433	1433	8500	NaN	17016	Petrol	86	0.0	0	1300	3	1015
1434	1434	7250	70.0	??	NaN	86	1.0	0	1300	3	1015
1435	1435	6950	76.0	1	Petrol	110	0.0	0	1600	5	1114
1436	1436	6950	76.0	1	NaN	110	0.0	0	1600	5	1114

- **#Example No.3**

  nifty=pd.read_csv("F:\pandas\Toyota.csv")

  nifty.shape

  nifty.index

  nifty.columns

- **Output :**

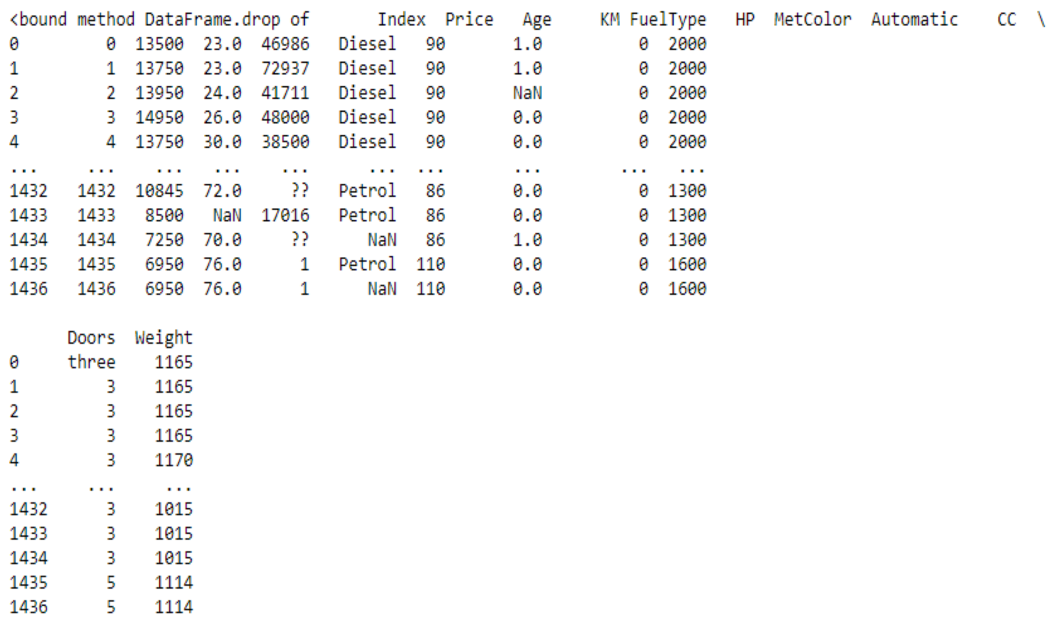
```
Index(['Index', 'Price', 'Age', 'KM', 'FuelType', 'HP', 'MetColor',
       'Automatic', 'CC', 'Doors', 'Weight'],
      dtype='object')
```

- **#Example No.4**

  nifty=pd.read_csv("F:\pandas\Toyota.csv")

  nifty.drop

- **Output :**

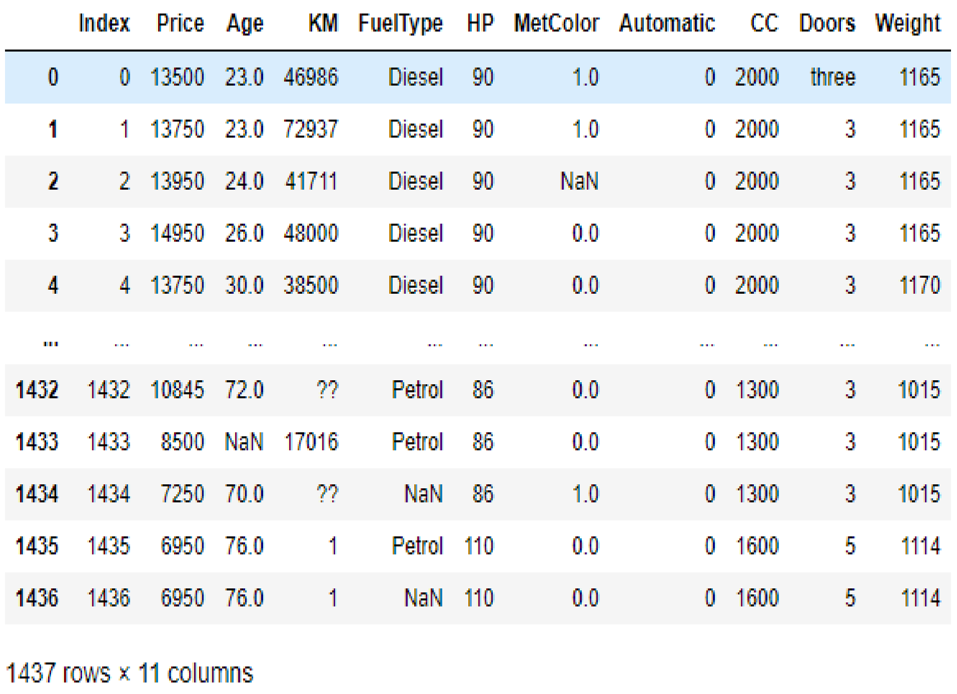


```
<bound method DataFrame.drop of       Index  Price   Age     KM FuelType   HP  MetColor  Automatic    CC  \
0         0  13500  23.0  46986   Diesel   90       1.0          0  2000
1         1  13750  23.0  72937   Diesel   90       1.0          0  2000
2         2  13950  24.0  41711   Diesel   90       NaN          0  2000
3         3  14950  26.0  48000   Diesel   90       0.0          0  2000
4         4  13750  30.0  38500   Diesel   90       0.0          0  2000
...     ...    ...   ...    ...      ...  ...       ...        ...   ...
1432   1432  10845  72.0     ??   Petrol   86       0.0          0  1300
1433   1433   8500   NaN  17016   Petrol   86       0.0          0  1300
1434   1434   7250  70.0     ??      NaN   86       1.0          0  1300
1435   1435   6950  76.0      1   Petrol  110       0.0          0  1600
1436   1436   6950  76.0      1      NaN  110       0.0          0  1600

      Doors  Weight
0     three    1165
1         3    1165
2         3    1165
3         3    1165
4         3    1170
...     ...     ...
1432      3    1015
1433      3    1015
1434      3    1015
1435      5    1114
1436      5    1114
```

- **#Example No.5**

  nifty=pd.read_csv("F:\pandas\Toyota.csv")
  nifty.sort_index()

- **Output :**



	Index	Price	Age	KM	FuelType	HP	MetColor	Automatic	CC	Doors	Weight
0	0	13500	23.0	46986	Diesel	90	1.0	0	2000	three	1165
1	1	13750	23.0	72937	Diesel	90	1.0	0	2000	3	1165
2	2	13950	24.0	41711	Diesel	90	NaN	0	2000	3	1165
3	3	14950	26.0	48000	Diesel	90	0.0	0	2000	3	1165
4	4	13750	30.0	38500	Diesel	90	0.0	0	2000	3	1170
...	...	...	...	...	...	...	...	...	...	...	...
1432	1432	10845	72.0	??	Petrol	86	0.0	0	1300	3	1015
1433	1433	8500	NaN	17016	Petrol	86	0.0	0	1300	3	1015
1434	1434	7250	70.0	??	NaN	86	1.0	0	1300	3	1015
1435	1435	6950	76.0	1	Petrol	110	0.0	0	1600	5	1114
1436	1436	6950	76.0	1	NaN	110	0.0	0	1600	5	1114

1437 rows × 11 columns

- **Conclusion :** By Studying This Practical ,We Are Able To Importing CSV File To Pandas Dataframe.

Experiment No : 11

- **Title :** Import Any CSV File To Pandas Dataframe And Perform Following Operations :
    a) Handle missing data by detecting and dropping/filling missing values.
    b) Transform Data Using Apply() and Map() method.
    c) Detect And Filter Outliers.
    d) Perform Vectorized String Operations On pandas Series.
    e) Visualize Data Using Line Plots, Bar Plots, Histograms, Density Plots And Scatter Plots.

- **Aim :** Basic python programs helps to understand the fundamental concepts of importing CSV file

- **Theory :**

CSV files are the "comma separated values", these values are separated by commas, this file can be viewed as an Excel file. In Python, Pandas is the most important library coming to data science. We need to deal with huge datasets while analyzing the data, which usually can be in CSV file format. Let's see the different ways to import csv files in Pandas.

- **Ways to Import CSV File in Pandas**
There are various ways to import CSV files in Pandas, here we are discussing some generally used methods for importing CSV files in pandas.
    - Using read_csv() Method
    - Using csv Module.
    - Using numpy Module
    - Using sklearn module
    - Using seaborn module
    - Using Matplotlib module

- **Program :**

- **#Example No.1**

```
import numpy as np
import pandas as pd
```

```
df = pd.DataFrame({
    "Date": pd.date_range(start="2021-10-01", periods=10, freq="D"),
    "Item": 1014,
    "Measure_1": np.random.randint(1, 10, size=10),
    "Measure_2": np.random.random(10).round(2),
    "Measure_3": np.random.random(10).round(2),
    "Measure_4": np.random.randn(10),
})

print(df)
```

- **Output :**

```
        Date      Item   Measure_1  Measure_ 2  Measure_3  Measure_4
0   2021-10-01  1014         7          0.27        0.93        0.35
1   2021-10-02  1014         1          0.94        0.36       -0.22
2  2021-10-03  1014        3          0.29        0.04        0.11
3  2021-10-04  1014        3          0.64        0.57        0.12
4  2021-10-05  1014        7          0.94        0.94       -0.66
5  2021-10-06  1014        2          0.61        0.79        0.47
6  2021-10-07  1014        2          0.35        0.71       -0.15
7  2021-10-08  1014        4          0.79        0.05       -0.54
8  2021-10-09  1014        6          0.11        0.29       -1.26
9  2021-10-10  1014        5          0.04        0.95        0.21
```

- **#Example No.2**

```
import pandas as pd

df = pd.DataFrame({

    "Name": ['JohnDoe', 'Mary Re', 'Harley Me'],

    "gender": [1, 2, 0],
    "age": [80, 38, 12],

    "height": [101.0, 173.5, 180.5],

    "weight": [62.3, 55.7, 80.0]

})

def custom_sum(row):

    return row.sum()

df['D'] = df.apply(custom_sum, axis=1)

print(df)
```

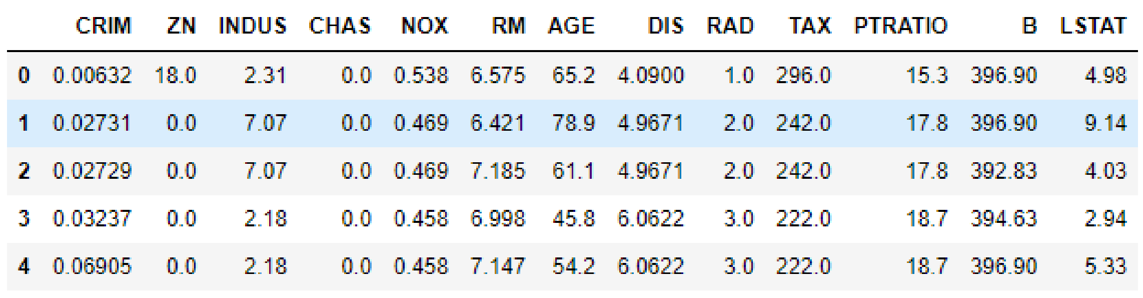
- **Output :**

```
        Name      gender   age  height  weight     D
0    JohnDoe        1       80   101.0    62.3     244.3
1     Mary Re       2       38   173.5    55.7     269.2
2   Harley Me       0       12   180.5    80.0  272.5
```

- **#Example No.3**

```
import sklearn
from sklearn.datasets import load_boston
import pandas as pd
import matplotlib.pyplot as plt

bos_hou = load_boston()
column_names = bos_hou.feature_names
df_boston = pd.DataFrame(bos_hou.data, columns=column_names)
df_boston.head()
```

- **Output :**



	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

- **#Example No.4**

```
import numpy as np
x=np.array([2,3,5,7,11,13])
x*2
```

- **Output :**

```
array([ 4,  6, 10, 14, 22, 26])
```

- **#Example No.5**

```
import pandas as pd
data=['Peter','Paul','Mary','GUIDO']
names=pd.series(data)
names
```

- **Output :**

```
0    Peter
1     Paul
2     Mary
```

```
3    GUIDO

dtype: object
```

- **#Example No.6**

```
import pandas as pd

data=['Peter','Paul','Mary','GUIDO']

[s.capitalize()for s in data]

names=pd.series(data)

names
```

- **Output :**

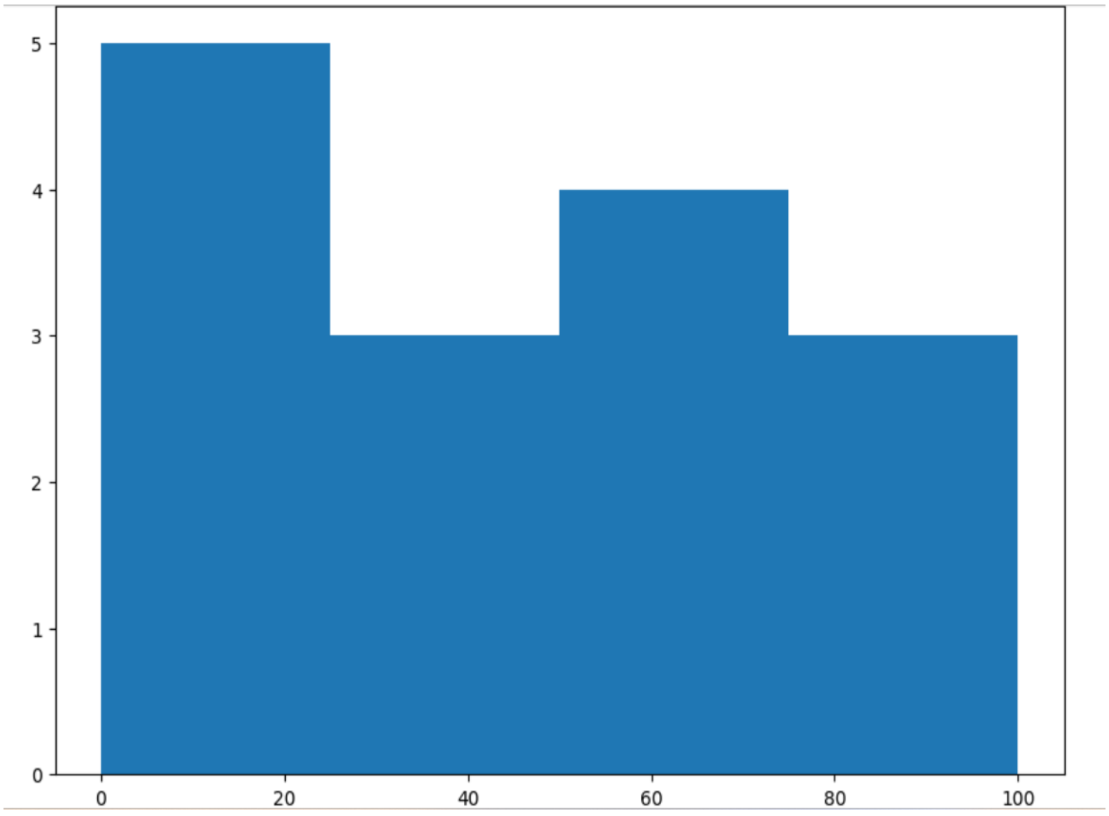
```
0    Peter

1     Paul

2     Mary

3    Guido

dtype: object
```

- **#Example No.6**

```
from matplotlib import pyplot as plt

import numpy as np

a = np.array([22, 87, 5, 43, 56, 93, 55, 54, 11, 20, 51, 5, 79, 39, 27])

fig, ax = plt.subplots(figsize=(10, 7))

ax.hist(a, bins=[0, 25, 50, 75, 100])
```

```
plt.show()
```

- **Output :**

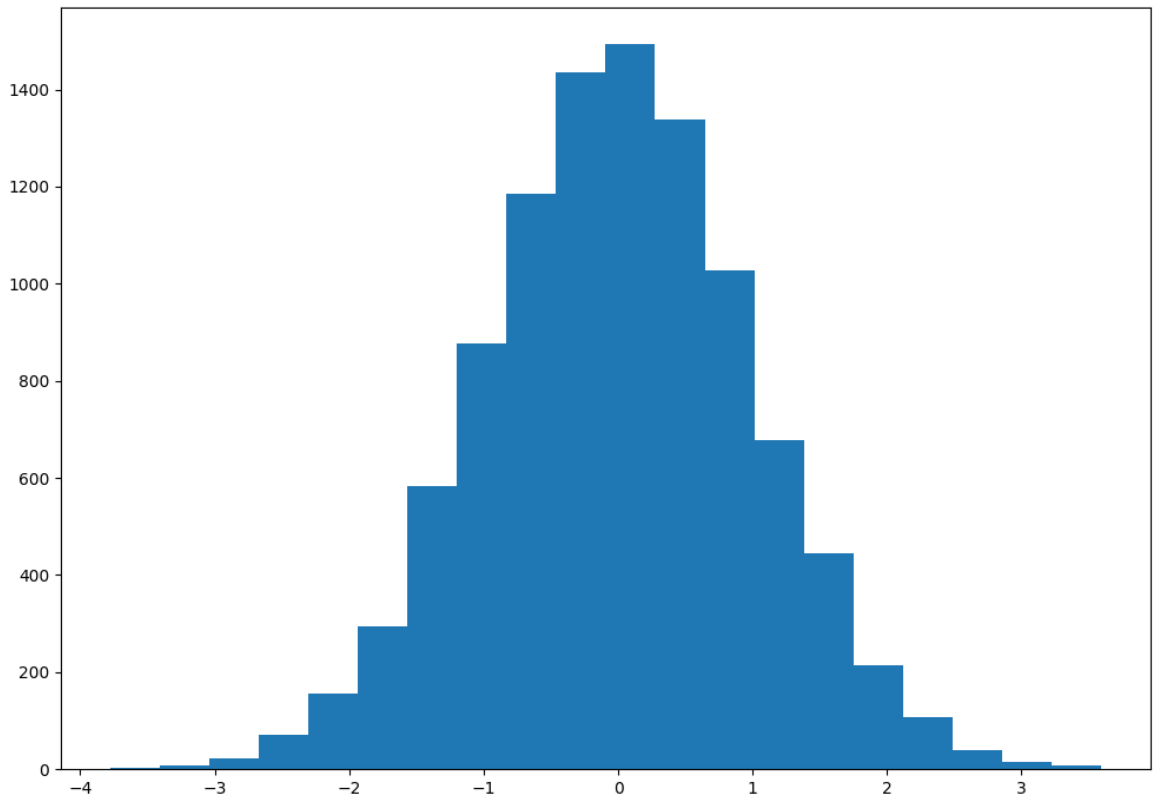


- **#Example No.7**

```
import matplotlib.pyplot as plt

import numpy as np

from matplotlib import colors

from matplotlib.ticker import PercentFormatter

np.random.seed(23685752)

N_Points = 10000

n_bins = 20

x = np.random.randn(N_Points)

y = 8**x + np.random.randn(10000) + 25
```

```
fig, axs = plt.subplots(1, 1, figsize=(10, 7), tight_layout=True)

axs.hist(x, bins=n_bins)

plt.show()
```

- **Output :**



- **Conclusion :** By Studying This Practical ,We Are Able To Importing CSV File To Pandas Dataframe.