

Practical No. 1: Install and configure Python IDE

Installing Python in Windows:

- Open any internet browser. Type <http://www.Python.org/downloads/> in address bar and Enter.
- Home page of Python will have displayed as shown in Fig. 1

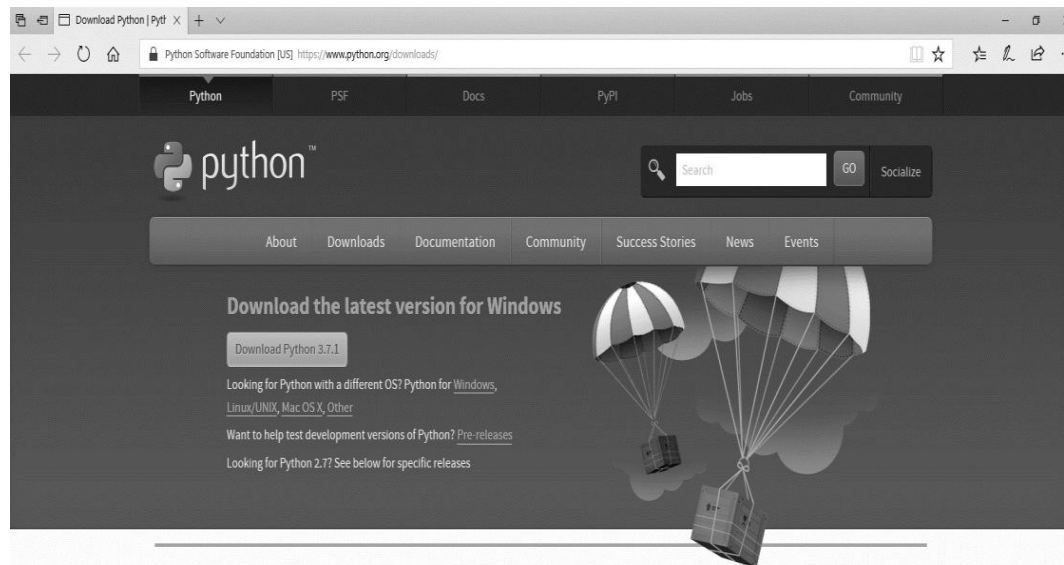


Fig. 1: Home Page

- Click on download the latest version for windows, which shows latest version as shown in Fig. 2

Looking for a specific release?

Python releases by version number:

Release version	Release date	Click for more	
Python 3.7.1	2018-10-20	Download	Release Notes
Python 3.6.7	2018-10-20	Download	Release Notes
Python 3.5.6	2018-08-02	Download	Release Notes
Python 3.4.9	2018-08-02	Download	Release Notes
Python 3.7.0	2018-06-27	Download	Release Notes
Python 3.6.6	2018-06-27	Download	Release Notes
Python 2.7.15	2018-05-01	Download	Release Notes

[View older releases](#)

Fig.2: Python release versions

- Open the Python 3.7.1 version pack and double click on it to start installation and installation windows will be open as shown in Fig. 3.

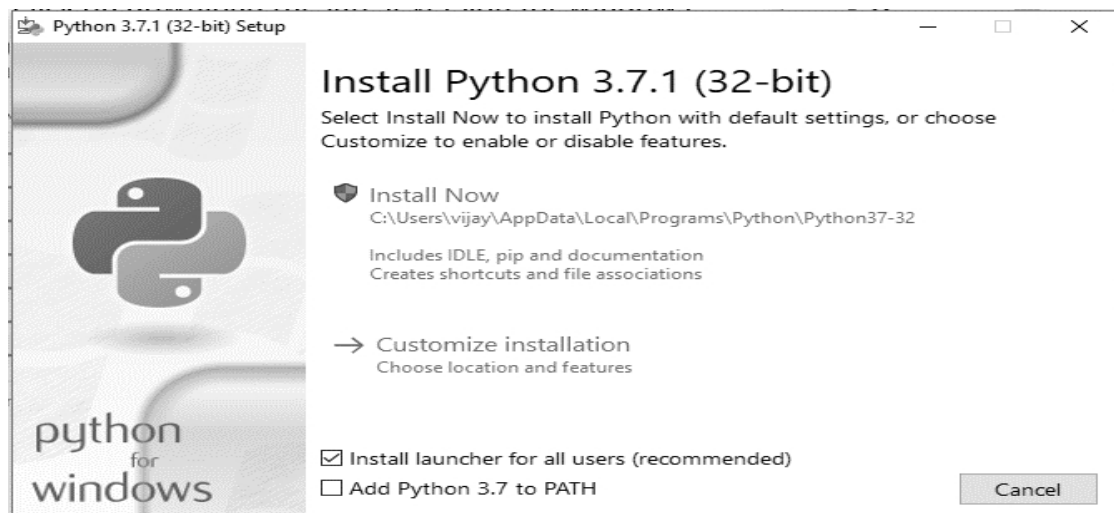
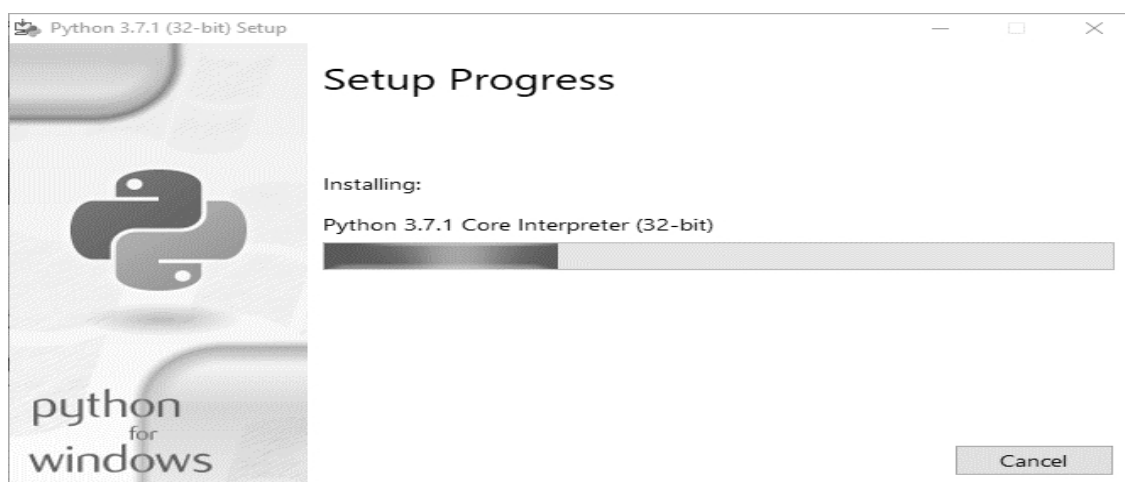


Fig. 3: Installation Type

- Click on next install now for installation and then Setup progress



windows will be opened as shown in Fig. 4.

Fig. 4: Setup Progress

- After complete the installation, Click on close button in the windows as shown in Fig. 5.

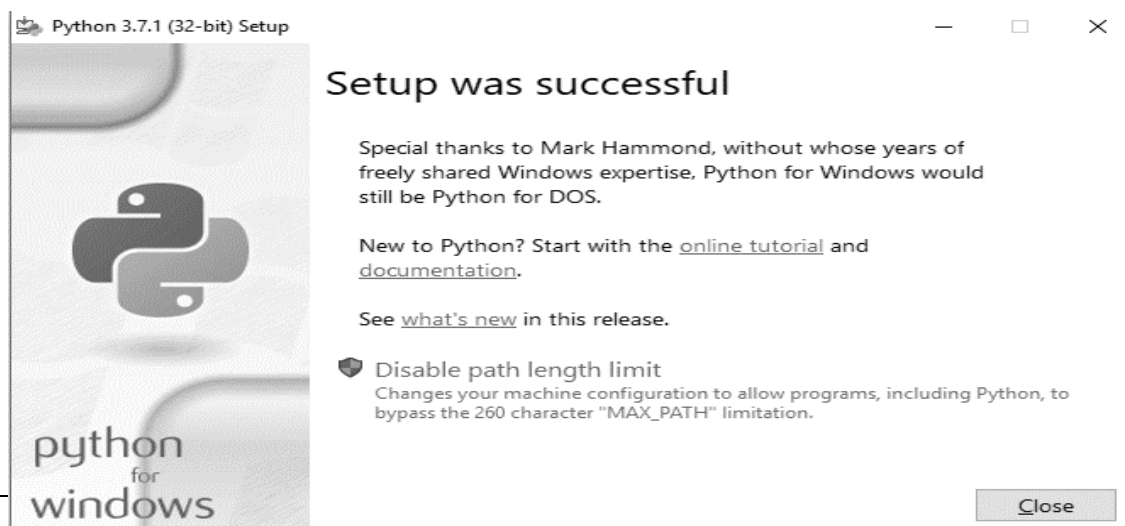


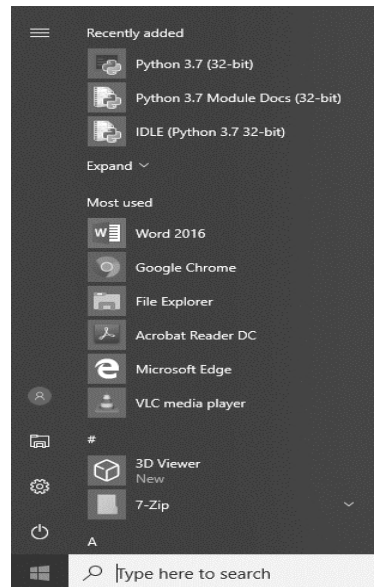
Fig. 5: Setup

Completion Starting Python in different

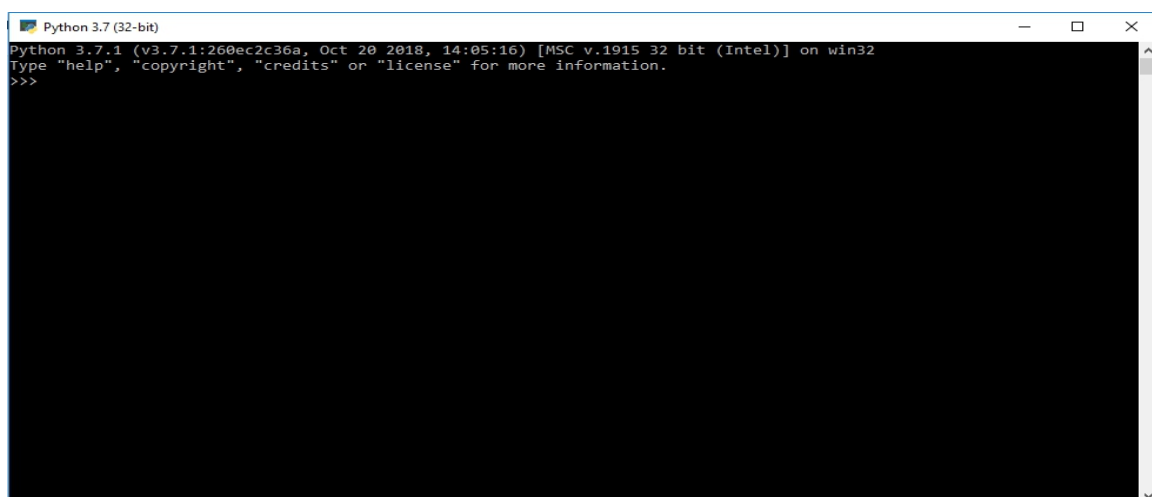
modes:

1) Starting Python (Command Line)

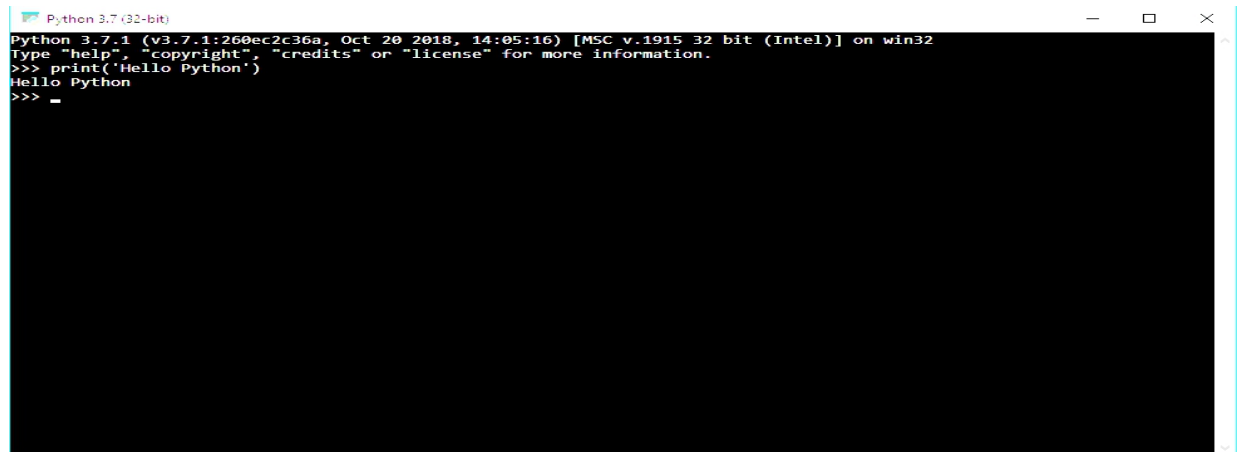
- Press start button



- Click on all programs and then click on Python 3.7 (32 bit). You will see the Python interactive prompt in Python command line.



- Python command prompt contains an opening message >>> called command prompt. The cursor at command prompt waits for to enter Python command. A complete command is called a statement. For example check first command to print message.

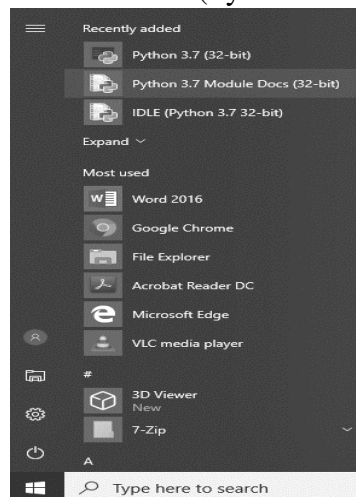


```
Python 3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 14:05:16) [MSC v.1915 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> print('Hello Python')
Hello Python
>>> _
```

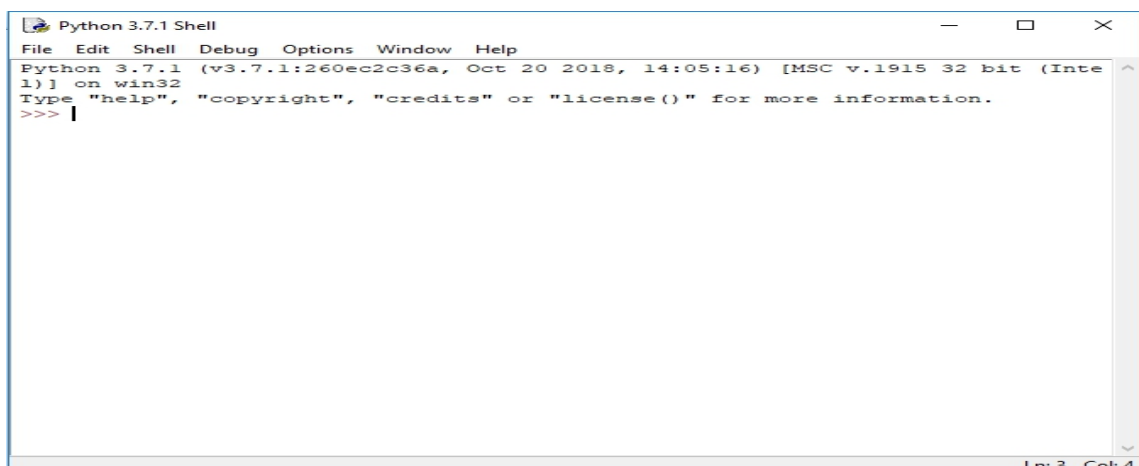
- To exit from the command line of Python, press Ctrl+z followed by Enter or Enter exit() or quit() and Enter.

2) Starting Python IDLE

- Press start button and click on IDLE (Python 3.7 32 bit) options.

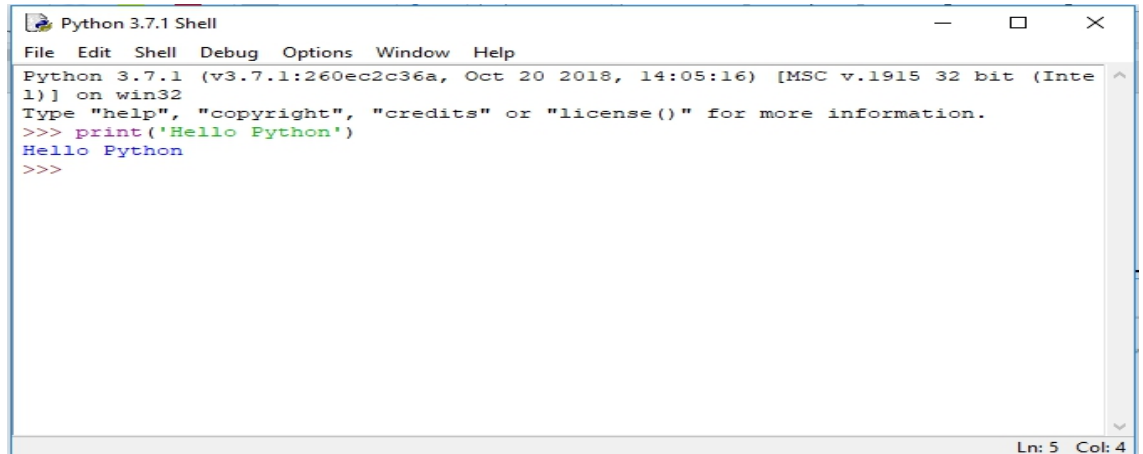


- You will see the Python interactive prompt i.e. interactive shell.



```
Python 3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 14:05:16) [MSC v.1915 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> |
```

- Python interactive shell prompt contains opening message >>>, called shell prompt. A cursor is waiting for the command. A complete command is called a statement. When you write a command and press enter, the Python interpreter will immediately display the result.



```
Python 3.7.1 Shell
File Edit Shell Debug Options Window Help
Python 3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 14:05:16) [MSC v.1915 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> print('Hello Python')
Hello Python
>>>
```

Ln: 5 Col: 4

Practical No.2 : To study and execute python program for List, Tuple and Dictionary

THEORY-

An object may be classified into two categories

- 1) Mutable
- 2) Immutable

Mutable are the objects whose value can be altered after assigning a particular value.

Immutable are those objects whose value can not be altered after assigning a value.

Examples of mutable data type are list and dictionary

Example of immutable data type is tuple

LIST

- 1) It is a collection of data in which data is stored in ordered form
- 2) It is mutable
- 3) The square bracket is used to represent list .The item inside the list are separated by comma.
- 4) It supports indexing and slicing
- 5) Eg. a=[10, 22.75, "Sanskrit", 4+5j, true]

TUPLE

- 1) It is a collection of data in which data is stored in ordered form
- 2) It is immutable
- 3) The normal bracket () is used to represent list .The item inside the list are separated by comma.
- 4) It supports indexing and slicing
- 5) Eg. a=(10, 22.75, "Sanskrit", 4+5j, true)

DICTIONARY

- 1) It is a collection of data in which it is not in a sequential order like a list
- 2) It do not support indexing.
- 3) It consists of key and its value.
- 4) It is mutable
- 5) The bracket used for dictionary is curly bracket '{ }'
- 6) Eg. dl={"Name": "Sanskrit", "Age": 20, "roll no": 162}

PROGRAM –**# PROGRAM FOR LIST**

```
a=[10,12.25,"Sanskrit",True, 4+5j, 10]
print(" List of elements in list are ",a)
print(type(a))
print(" Item at index 2 in list is ",a[2])
print(" Index 1 to 4 elements in list are ",a[1:5])
```

```
a.append("IT")
print(" New list with append elements is ",a)
a.insert(2,10)
print(" New list with inserted element is ",a)
a.remove("Sanskrit")
print(" New list with removed element is ",a)
c=a.count(10)
print(" No of 10 in list is ",c)
```

#PROGRAM FOR TUPLE

```
t=(10,12.25,"Sanskrit",True,4+5j)
print(" List of elements in tuple are ",t)
print(type(t))
print(" Item at index 2 in tuple is ",t[2])
print(" Index 1 to 4 elements in tuple are ",t[1:5])
x=t.index(True)
print("Index of True in tuple is = ",x)
y=t.count(10)
print(" No of 10 in list is ",y)
```

#PROGRAM FOR DICTIONARY

```
dl={"Name":"Anand","Age":20,"Roll no":121}
print("Keys available in dictionary d1 are ", dl.keys())
print("Values available in dictionary d1 are ",dl.values())
print("Items available in dictionary d1 are ",dl.items())
print("Dictionry items are ",dl)
print("Printing of value whose key is Name and value = ",dl["Name"])
dl.update({"branch":"IT","college":"PCE"})
print("Updated dictionary items are" ,dl)
del(dl["branch"])
print("Dictionary items after delection of branch key ",dl)
```

Testing Output

List of elements in list are [10, 12.25, 'Sanskrit', True, (4+5j), 10]

<class 'list'>

Item at index 2 in list is Sanskrit

Index 1 to 4 elements in list are [12.25, 'Sanskrit', True, (4+5j)]

New list with append elements is [10, 12.25, 'Sanskrit', True, (4+5j), 10, 'IT']

New list with inserted element is [10, 12.25, 10, 'Sanskrit', True, (4+5j), 10, 'IT']

New list with removed element is [10, 12.25, 10, True, (4+5j), 10, 'IT']

No of 10 in list is 3

List of elements in tuple are (10, 12.25, 'Sanskrit', True, (4+5j))

<class 'tuple'>

Item at index 2 in tuple is Sanskrit

Index 1 to 4 elements in tuple are (12.25, 'Sanskrit', True, (4+5j))

Index of True in tuple is = 3

No of 10 in list is 1

Keys available in dictionary d1 are dict_keys(['Name', 'Age', 'Roll no'])

Values available in dictionary d1 are dict_values(['Anand', 20, 121])

Items available in dictionary d1 are dict_items([('Name', 'Anand'), ('Age', 20), ('Roll no', 121)])

Dictionary items are {'Name': 'Anand', 'Age': 20, 'Roll no': 121}

Printing of value whose key is Name and value = Anand

Updated dictionary items are {'Name': 'Anand', 'Age': 20, 'Roll no': 121, 'branch': 'IT', 'college':

'PCE'}

Dictionary items after deletion of branch key {'Name': 'Anand', 'Age': 20, 'Roll no': 121,

'college': 'PCE'}

Result:

Thus the python program for Tuple and Dictionary was successfully executed.

Practical No.3: To study and execute python program for Function, Scoping and Abstraction

THEORY-

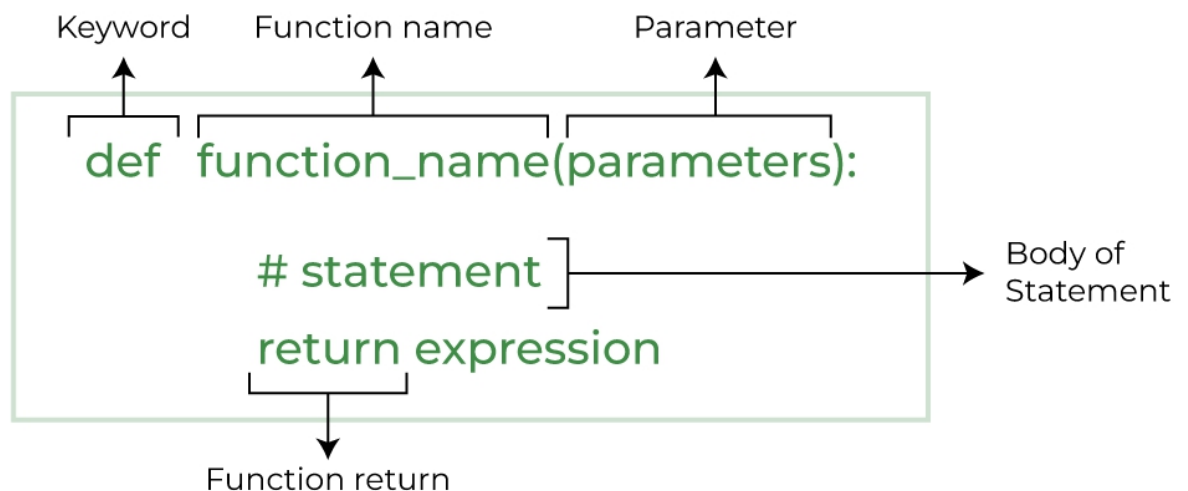
Function

Python Functions is a block of statements that return the specific task. The idea is to put some commonly or repeatedly done tasks together and make a function so that instead of writing the same code again and again for different inputs, we can do the function calls to reuse code contained in it over and over again.

Some Benefits of Using Functions
Increase Code Readability

Python Function Declaration

The syntax to declare a function is:



Types of Functions in Python

Below are the different types of functions in Python:

- **Built-in library function:** These are Standard functions in Python that are available to use.
- **User-defined function:** We can create our own functions based on our requirements.

Creating a Function in Python

We can define a function in Python, using the `def` keyword. We can add any type of functionalities and properties to it as we require.

A simple Python function

```
def fun():  
    print("Welcome to Python Programming")
```

Python Scope variable

The location where we can find a variable and also access it if required is called the **scope of a variable**.

Python Local variable

Local variables are those that are initialized within a function and are unique to that function. It cannot be accessed outside of the function. Let's look at how to make a local variable.

Python Global variables

Global variables are the ones that are defined and declared outside any function and are not specified to any function. They can be used by any part of the program.

Abstraction in Python

Abstraction is used to hide the internal functionality of the function from the users. The users only interact with the basic implementation of the function, but inner working is hidden. User is familiar with that "what function does" but they don't know "how it does."

In simple words, we all use the smartphone and very much familiar with its functions such as camera, voice-recorder, call-dialing, etc., but we don't know how these operations are happening in the background. Let's take another example - When we use the TV remote to increase the volume. We don't know how pressing a key increases the volume of the TV. We only know to press the "+" button to increase the volume.

In Python, an abstraction is used to hide the irrelevant data/class in order to reduce the complexity. It also enhances the application efficiency.

Abstraction classes in Python

In Python, abstraction can be achieved by using abstract classes and interfaces.

A class that consists of one or more abstract method is called the abstract class. Abstract methods do not contain their implementation. Abstract class can be inherited by the subclass and abstract method gets its definition in the subclass. Abstraction classes are meant to be the blueprint of the other class. An abstract class can be useful when we are designing large functions. An abstract class is also helpful to provide the standard interface for different implementations of components. Python provides the abc module to use the abstraction in the Python program. Let's see the following syntax.

Syntax

```
from abc import ABC

class ClassName(ABC):
```

We import the ABC class from the **abc** module.

Interface in Python

Generally, the interface is not part of Python, but we can implement it using the ABC module.

An interface is a design template for creating classes that share common methods. The methods defined in an interface are abstract, meaning they are only outlined and lack implementation. It is unlike classes, which provide a concrete implementation for their methods. Instead, classes that implement an interface will fill in the abstract methods, giving them specific meaning and functionality.

By defining interfaces, developers can create a standard contract that classes can adhere to, regardless of their inheritance hierarchy. It allows for better code organization and reduces the likelihood of code duplication.

PROGRAM –

#Program to check whether the given number is Even or Odd using Function

```
class EvenOrOdd:
    def __init__(self,number):
        self.num = number

    def EvenOrOddd(self):
        if num % 2 == 0:
            print("Entered number is even.")
        else:
            print("Entered number is odd.")

num = int(input("Enter the number: "))
obj = EvenOrOdd(num)
obj.EvenOrOddd()
```

#Program to implement scoping in Python

```
glob = "I am globally available."

def function1():
    local = "I am locally available."
    print(local)
    print(glob)

function1()
print(glob)
```

#Program to implement Abstraction in Python

```
from abc import ABC, abstractmethod

class Shape(ABC):
```

```
@abstractmethod
def calculate_area(self):
    pass

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius
    def calculate_area(self):
        return 3.14 * self.radius *self.radius

class Rectangle(Shape):
    def __init__(self, length, width):
        self.length = length
        self.width = width
    def calculate_area(self):
        return self.length * self.width

circle = Circle(5)
rectangle = Rectangle(4, 6)

print("Area of Circle: ", circle.calculate_area())
print("Area of Rectangle: ", rectangle.calculate_area())
```

OUTPUT:**#Output to check whether the given number is Even or Odd using Python**

Enter the number: 5
Entered number is odd.

#Output of Program to implement scoping in Python

I am locally available.
I am globally available.
I am globally available.

#Output of Program to implement Abstraction in Python

Area of Circle: 78.5
Area of Rectangle: 24

Result:

Thus the python program for Function, Scoping and Abstraction is successfully

**Practical No.4: To study and execute python program for classes , objects,
Inheritance and Polymorphism.**

THEORY-

Python Classes/Objects

Python is an object oriented programming language.

Almost everything in Python is an object, with its properties and methods.

A Class is like an object constructor, or a "blueprint" for creating objects.

ALGORITHM-

Step 1- Create a class with name student

Step 2- Create a default method __init__ with three arguments

Step 3- Create a method with name avg for calculation of average marks

Step 4- Create two objects with name s1 and s2

Step 5- Calculate average marks of object s1 and s2 and display the result

PROGRAM –

```
class student:
    def __init__(self,m1,m2,m3):
        self.m1=m1
        self.m2=m2
        self.m3=m3

    def avg(self):
        return (self.m1+self.m2+self.m3)/3

s1=student(40,50,60)
s2=student(70,80,90)
avg1=s1.avg()
avg2=s2.avg()
print("Average marks of 1st student=", avg1)
print("Average marks of 1st student=", avg2)
```

Testing Output

Average marks of 1st student= 50.0

Average marks of 1st student= 80.0

Python Inheritance

Inheritance allows us to define a class that inherits all the methods and properties from another class.

Parent class is the class being inherited from, also called base class.

Child class is the class that inherits from another class, also called derived class.

ALGORITHM- for Inheritance

- Step 1- Create a class with name parent
- Step 2- Create a method with name spouse details
- Step 3- Create a another method with name children details
- Step 4- Create a another class child with inheritance facility
- Step 5- Create a method with name qualification details
- Step 6- Access parent class methods through child class

PROGRAM

```
class parent:
    def spouse(self):
        print(" My wife name is Manda")
    def children(self):
        print(" I have two child, one is boy and another is girl")

class child(parent):
    def qualification(self):
        print(" My first child is Engineer and second is doctor")

c1=child()
c1.spouse()
c1.children()
c1.qualification()
```

Testing Output

```
My wife name is Manda
I have two child, one is boy and another is girl
My first child is Engineer and second is doctor
```

Python Polymorphism

The word "polymorphism" means "many forms", and in programming it refers to methods/functions/operators with the same name that can be executed on many objects or classes.

Class Polymorphism

Polymorphism is often used in Class methods, where we can have multiple classes with the same method name. For example, say we have three classes: Car, Boat, and Plane, and they all have a method called move():

PROGRAM :

```
class Shape:
    def __init__(self,x,y):
        self.x = x
        self.y = y

    def area(self):
        return self.x * self.y

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius
        super().__init__(radius, radius)

    def area(self):
        return 3.14 * super().area()

c = Circle(5)
print("Area of Circle is ", c.area())
```

Output:

Area of Circle is 78.5

Practical No. 5: To execute python programs using Numpy Library**THEORY-**

NumPy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices. NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely. NumPy stands for Numerical Python. In Python we have lists that serve the purpose of arrays, but they are slow to process.

NumPy aims to provide an array object that is up to 50x faster than traditional Python lists. The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy.

NumPy is a Python library and is written partially in Python, but most of the parts that require fast computation are written in C or C++. If you have Python and PIP already installed on a system, then installation of NumPy is very easy.

Install it using this command:

```
C:\Users\Your Name>pip install numpy
```

If this command fails, then use a python distribution that already has NumPy installed like, Anaconda, Spyder etc.

Once NumPy is installed, import it in your applications by adding the import keyword:
`import numpy`

Now NumPy is imported and ready to use.

Example

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

NumPy is usually imported under the np alias.

alias: In Python alias are an alternate name for referring to the same thing.

Create an alias with the as keyword while importing:

```
import numpy as np
```

Now the NumPy package can be referred to as np instead of numpy.

Example

```
import numpy as np
```



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

PROGRAM –

#Program to Create a NumPy ndarray Object using Numpy library

```
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'>
```

Program to use a tuple to create a NumPy array:

```
import numpy as np
arr = np.array((1, 2, 3, 4, 5))
print(arr)
```

Output:

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

Program to create different dimension arrays

```
import numpy as np
arr0 = np.array(42) #0-D Array
arr1 = np.array([1, 2, 3, 4, 5]) #1-D Array
arr2 = np.array([[1, 2, 3], [4, 5, 6]]) #2-D Array
arr3 = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]]) #3-D Array

print("0-D Array: ", arr0)
print("1-D Array: ", arr1)
print("2-D Array: ", arr2)
print("3-D Array: ", arr3)
```

Output:

```
0-D Array: 42
1-D Array: [1 2 3 4 5]
2-D Array: [[1 2 3]
 [4 5 6]]
3-D Array: [[[1 2 3]
 [4 5 6]]
 [[1 2 3]
 [4 5 6]]]
```

Program to Access Array Elements using Numpy library

```
import numpy as np
arr = np.array([1, 2, 3, 4])
print("Second element in array is ", arr[1])
```

Output:

Second element in array is 2

Program to perform NumPy Array Slicing using Numpy library

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print("Slicing of elements from index 1 to index 5 is ", arr[1:5])
print("Slicing of elements from index 4 to the end of the array is", arr[4:])
print("Slicing of elements from the beginning to index 4 (not included) is ", arr[:4])
```

Output:

Slice elements from index 1 to index is 5 [2 3 4 5]

Slice elements from index 4 to the end of the array is [5 6 7]

Slice elements from the beginning to index 4 (not included) is [1 2 3 4]

Result:

Thus the python programs using Numpy Library are successfully executed.

Practical no:6: To execute python programs using Pandas**THEORY:**

Pandas is a Python library used for working with data sets. It has functions for analyzing, cleaning, exploring, and manipulating data. The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008. Pandas allows us to analyze big data and make conclusions based on statistical theories.

Pandas can clean messy data sets, and make them readable and relevant. Relevant data is very important in data science.

If you have Python and PIP already installed on a system, then installation of Pandas is very easy.

Install it using this command:

```
C:\Users\Your Name>pip install pandas
```

If this command fails, then use a python distribution that already has Pandas installed like, Anaconda, Spyder etc.

Once Pandas is installed, import it in your applications by adding the import keyword:

```
import pandas
```

Now Pandas is imported and ready to use.

Pandas is usually imported under the pd alias. Create an alias with the as keyword while importing:

```
import pandas as pd
```

Now the Pandas package can be referred to as pd instead of pandas.

PROGRAM –**#Program to Create a simple Pandas Series from a list.**

```
import pandas as pd
a = [1, 7, 2]
myvar = pd.Series(a)
print(myvar)
```

Output:

```
0    1
1    7
2    2
dtype: int64
```

#Program to Create a simple Pandas Series from a dictionary.

```
import pandas as pd
calories = {"day1": 420, "day2": 380, "day3": 390}
myvar = pd.Series(calories)
print(myvar)
```

Output:

```
day1    420
day2    380
day3    390
dtype: int64
```

#Program to add a list of names to give each row a name

```
import pandas as pd
data = {
    "calories": [420, 380, 390],
    "duration": [50, 40, 45]
}
df = pd.DataFrame(data, index = ["day1", "day2", "day3"])
print(df)
```

Output:

	calories	duration
day1	420	50
day2	380	40
day3	390	45

Result:

Thus the python programs using Pandas Library are successfully executed.

Practical no:7

Aim: Demonstrate the concept of input output file and write a program to understand the concept of input output file

Theory: Files in Python

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.

● File I/O in Python

Python can be used to perform operations on a file. (read & write data)

Types of all files

1. Text Files : .txt, .docx, .log etc.
2. Binary File : .mp4, .mov, .png, .jpeg etc.

● Open, read & close File

We have to open a file before reading or writing.

```
f = open( "file_name","mode")  
//(file name:sample.txt      /demo.docx  )  
//(mode: r : read mode and w : write mode)  
data = f.read( )  
f.close( )
```

The key function for working with files in Python is the open() function.

The open() function takes two parameters; filename, and mode.

There are four different methods (modes) for opening a file:

"r" - Read - Default value. Opens a file for reading, error if the file does not exist

"a" - Append - Opens a file for appending, creates the file if it does not exist

"w" - Write - Opens a file for writing, creates the file if it does not exist

"x" - Create - Creates the specified file, returns an error if the file exists

In addition you can specify if the file should be handled as binary or text mode

"t" - Text - Default value. Text mode

"b" - Binary - Binary mode (e.g. images)

Syntax

To open a file for reading it is enough to specify the name of the file:

```
f = open("demofile.txt")
```

The code above is the same as:

```
f = open("demofile.txt", "rt")
```

● Reading a file

```
data = f.readline( ) #reads one line at a time
```

```
data = f.read( ) #reads entire file
```

● Writing to a file

```
f.write( "this is a new line" ) #overwrites the entire file
```

```
f = open( "demo.txt", "w")
```

```
f.write( "this is a new line" ) #adds to the file
```

with Syntax

```
with open( "demo.txt", "a") as f:  
    data = f.read( )
```

● Deleting a File

using the os module

Module (like a code library) is a file written by another programmer that generally has a functions we can use.

```
import os
os.remove( filename )
```

Program:

```
# Define the filename
filename = 'example_file.txt'

# Writing to a file
with open(filename, 'w') as file:
    file.write('Hello, this is a test file.\n')
    file.write('This file demonstrates basic file I/O operations in Python.\n')
    file.write('Enjoy coding with Python!')

print(f'Content has been written to {filename}')

# Reading from the file
with open(filename, 'r') as file:
    content = file.read()

print("\nContent read from the file:")
print(content)
```

output:

Content has been written to example_file.txt

Content read from the file:

Hello, this is a test file.

This file demonstrates basic file I/O operations in Python.

Enjoy coding with Python!

Result:

Thus, I have studied the concept of file input output in python and successfully executed program on it.

Practical no. 8

Aim: Demonstrate the concept of web programming in Python and write a program to understand the concept of web programming in Python.

Theory:

● Python web programming

Python can be used to support web programming that falls into one of two general categories
client programming – accesses web sites or web applications
server programming – script executed by the server to provide a web site, perform server-side applications, etc.
The role of web programming in creating dynamic, interactive websites and web applications

● What is a web client?

Any program that retrieves data from a web server using the HTTP protocol

Examples:

web browsers – contact web servers using HTTP protocol and display HTTP responses

web crawlers – traverse the web automatically to gather information

web service clients (service requester) – request and process data from a web service provider; the web service provider responds using some web service protocol such as RSS (Rich Site Summary) or RPC (remote procedure call)

● Why Python for Web Development?

1. Python is easy to learn and use, with a clear and readable syntax.
2. Rich ecosystem of libraries and frameworks.
3. Strong community support and extensive documentation.
4. Versatility for both simple and complex web applications.

● Popular Python Web Frameworks

1. Flask:

- ◆ Lightweight and flexible.
- ◆ Ideal for small to medium-sized applications.
- ◆ Simple to use, great for beginners.

2. Django:

- Full-featured framework with batteries included.
- Excellent for larger applications.
- Built-in admin panel, authentication, and ORM.

3. FastAPI:

- ◆ Modern, fast, and high-performance.
- ◆ Supports asynchronous programming.
- ◆ Great for building APIs.

program:**Step-by-Step Guide to Create a Login Page Using Flask****1. Install Flask**

If you haven't already installed Flask, you can do so using pip:

pip install flask**2. Create Project Structure**

Let's set up a basic project structure for our Flask application:

```
/flask_login_app
  /templates
    login.html
    home.html
  app.py
```

3. Create HTML Templates

Create two HTML templates, login.html and home.html, in the templates directory.

login.html:**Code:**

```
<!-- templates/login.html -->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Login Page</title>
</head>
<body>
  <h2>Login Page</h2>
  <form method="POST">
    <label for="username">Username:</label>
    <input type="text" id="username" name="username" required>
    <br><br>
    <label for="password">Password:</label>
    <input type="password" id="password" name="password" required>
    <br><br>
    <button type="submit">Login</button>
  </form>
  {% if error %}
  <p style="color:red;">{{ error }}</p>
  {% endif %}
</body>
</html>
```

home.html:**Code:**

```
<!-- templates/home.html -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Home Page</title>
</head>
<body>
    <h2>Welcome, {{ username }}!</h2>
    <p>You have successfully logged in.</p>
    <a href="/">Logout</a>
</body>
</html>
```

3. Create the Flask Application

Create an app.py file in the flask_login_app directory with the following content:

Code:

```
# app.py
from flask import Flask, render_template, request, redirect, url_for

app = Flask(__name__)

# Hardcoded users dictionary for demonstration
users = {"admin": "password123", "user1": "mypassword"}

@app.route('/', methods=['GET', 'POST'])
def login():
    error = None
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']

        # Simple validation of username and password
        if username in users and users[username] == password:
            return redirect(url_for('home', username=username))
        else:
            error = 'Invalid username or password. Please try again.'
    return render_template('login.html', error=error)

@app.route('/home')
def home():
    username = request.args.get('username')
    return render_template('home.html', username=username)
```

```
if __name__ == '__main__':  
    app.run(debug=True)
```

Output:

In that see the login and home page

Result:

Hence, I have studied and successfully create the web pages using python web programming.

Practical No: 9

Aim: Demonstrate the concept of regular Expression and write a program to understand the concept of regular Expression in Python.

Theory: Introduction to Regular Expressions

A regular expression is a sequence of characters that forms a search pattern. It is used to match strings of text, such as specific characters, words, or patterns. Regular expressions are utilized in various applications, including data validation, text processing, and searching within documents.

● **Metacharacters**

Metacharacters are symbols that have a special meaning in regex patterns:

`.`: Matches any single character except newline characters.

Example: `a.c` matches `abc`, `a1c`, `a c`, etc.

`^`: Anchors a pattern to the start of a string.

Example: `^abc` matches `abc` only at the beginning of the string.

`$`: Anchors a pattern to the end of a string.

Example: `abc$` matches `abc` only at the end of the string.

`*`: Matches zero or more occurrences of the preceding character or group.

Example: `a*` matches ````, `a`, `aa`, `aaa`, etc.

`+`: Matches one or more occurrences of the preceding character or group.

Example: `a+` matches `a`, `aa`, `aaa`, etc., but not ````.

`?`: Matches zero or one occurrence of the preceding character or group.

Example: `a?` matches ```` or `a`.

`|`: Acts as a logical OR.

Example: `alb` matches `a` or `b`.

`()`: Groups patterns together.

Example: `(abc)+` matches `abc`, `abcabc`, etc.

`[]`: Defines a character class, matching any one of the characters inside the brackets.

Example: `[abc]` matches `a`, `b`, or `c`.

`{ }`: Specifies the quantity of the preceding element.

Example: `a{2,4}` matches `aa`, `aaa`, or `aaaa`.

3. Special Sequences

Special sequences are shorthand for common sets of characters:

`\d`: Matches any digit (equivalent to `[0-9]`).

Example: `\d` matches 0, 1, 2, etc.

`\D`: Matches any non-digit character.

`\w`: Matches any word character (alphanumeric plus underscore, equivalent to `[a-zA-Z0-9_]`).

`\W`: Matches any non-word character.

`\s`: Matches any whitespace character (space, tab, newline).

`\S`: Matches any non-whitespace character.

● Regex Syntax and Patterns

Regex patterns can be combined to create complex search criteria. Here are some examples:

Email Validation:

Pattern: `\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b`

Phone Number Validation:

Pattern: `\+\d{1,3}-\d{3}-\d{3}-\d{4}`

Date Extraction:

Pattern: `\d{4}-\d{2}-\d{2}` for YYYY-MM-DD

Pattern: `\d{2}/\d{2}/\d{4}` for DD/MM/YYYY

Program:

```
import re

# Sample text for demonstration
text = """
Contact us at support@example.com or sales@example.org.
Call us at +1-800-555-1234 or (123) 456-7890.
For dates, use formats like 2024-08-27 or 27/08/2024.
"""

# 1. Find all email addresses
email_pattern = r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b'
emails = re.findall(email_pattern, text)
print("Email Addresses Found:")
print(emails)
```

```
# 2. Validate phone number formats
phone_patterns = [
    r'\+\d{1,3}-\d{3}-\d{3}-\d{4}', # International format
    r'\(\d{3}\) \d{3}-\d{4}'        # US format with parentheses
]

print("\nPhone Numbers Found:")
for pattern in phone_patterns:
    phones = re.findall(pattern, text)
    print(f'Pattern '{pattern}':')
    print(phones)

# 3. Extract dates in specific formats
date_patterns = [
    r'\d{4}-\d{2}-\d{2}', # YYYY-MM-DD
    r'\d{2}/\d{2}/\d{4}'  # DD/MM/YYYY
]

print("\nDates Found:")
for pattern in date_patterns:
    dates = re.findall(pattern, text)
    print(f'Pattern '{pattern}':')
    print(dates)
```

Output:

Email Addresses Found:
['support@example.com', 'sales@example.org']

Phone Numbers Found:
Pattern '\+\d{1,3}-\d{3}-\d{3}-\d{4}':
['+1-800-555-1234']
Pattern '\(\d{3}\) \d{3}-\d{4}':
['(123) 456-7890']

Dates Found:
Pattern '\d{4}-\d{2}-\d{2}':
['2024-08-27']
Pattern '\d{2}/\d{2}/\d{4}':
['27/08/2024']

Result:

Thus, I have studied the concept of Regular Expression in python and successfully executed program on regular expression.

