EXPERIMENT NO: 1

INTRODUCTION TO PYTHON PROGRAMMING

Python is a widely used general-purpose, high level programming language. It was created by Guido van Rossum in 1991 and further developed by the Python Software Foundation. It was designed with an emphasis on code readability, and its syntax allows programmers to express their concepts in fewer lines of code. It also supports Object Oriented programming approach to develop applications.

Python is an Interpreted Language because Python code is executed line by line at a time.

The source code of Python is converted into an immediate form called bytecode.

Python is a dynamically typed language. In other words, in Python, we do not need to declare the data types of the variables which we define.

Python is an open-source programming language and one can download it for free from Python's official website.

Essential features of Python.

Easy to use and Learn: Python has a simple and easy-to-understand syntax.

Expressive Language: It allows programmers to express complex concepts in just a few lines of code or reduces Developer's Time.

Interpreted Language: Python does not require compilation, allowing rapid development and testing. It uses Interpreter instead of Compiler.

Object-Oriented Language: It supports object-oriented programming, making writing reusable and modular code easy.

Open-Source Language: Python is open source and free to use, distribute and modify.

Extensible: Python can be extended with modules written in C, C++, or other languages.

Learn Standard Library: Python's standard library contains many modules and functions that can be used for various tasks, such as string manipulation, web programming, and more.

GUI Programming Support: Python provides several GUI frameworks, such as Tkinter and PyQt, allowing developers to create desktop applications easily.

Integrated: Python can easily integrate with other languages and technologies, such as C/C++, Java, and . NET.

Embeddable: Python code can be embedded into other applications as a scripting language.

Dynamic Memory Allocation: Python automatically manages memory allocation, making it easier for developers to write complex programs without worrying about memory management.

Wide Range of Libraries and Frameworks: Python has a vast collection of libraries and frameworks, such as NumPy, Pandas, Django, and Flask, that can be used to solve a wide range of problems.

Versatility: Python is a universal language in various domains such as web development, machine learning, data analysis, scientific computing, and more.

Large Community: Python has a vast and active community of developers contributing to its development and offering support. This makes it easy for beginners to get help and learn from experienced developers.

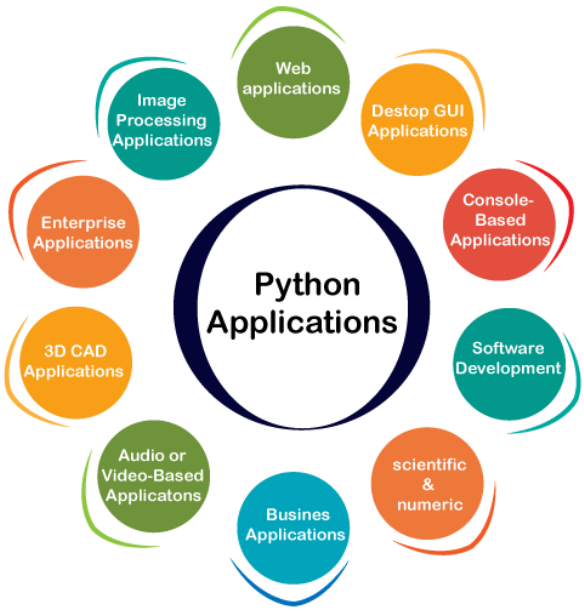
Career Opportunities: Python is a highly popular language in the job market. Learning Python can open up several career opportunities in data science, artificial intelligence, web development, and more.

High Demand: With the growing demand for automation and digital transformation, the need for Python developers is rising. Many industries seek skilled Python developers to help build their digital infrastructure.

Increased Productivity: Python has a simple syntax and powerful libraries that can help developers write code faster and more efficiently. This can increase productivity and save time for developers and organizations.

Big Data and Machine Learning: Python has become the go-to language for big data and machine learning. Python has become popular among data scientists and machine learning engineers with libraries like NumPy, Pandas, Scikit-learn, TensorFlow, and more.

Python Applications



Data Science: Data Science is a vast field, and Python is an important language for this field because of its simplicity, ease of use, and availability of powerful data analysis and visualization libraries like NumPy, Pandas, and Matplotlib.

Desktop Applications: PyQt and Tkinter are useful libraries that can be used in GUI - Graphical User Interface-based Desktop Applications. There are better languages for this field, but it can be used with other languages for making Applications.

Console-based Applications: Python is also commonly used to create command-line or console-based applications because of its ease of use and support for advanced features such as input/output redirection and piping.

Mobile Applications: While Python is not commonly used for creating mobile applications, it can still be combined with frameworks like Kivy or BeeWare to create cross-platform mobile applications.

Software Development: Python is considered one of the best software-making languages. Python is easily compatible with both from Small Scale to Large Scale software.

Artificial Intelligence: AI is an emerging Technology, and Python is a perfect language for artificial intelligence and machine learning because of the availability of powerful libraries such as TensorFlow, Keras, and PyTorch.

Web Applications: Python is commonly used in web development on the backend with frameworks like Django and Flask and on the front end with tools like JavaScript and HTML.

Enterprise Applications: Python can be used to develop large-scale enterprise applications with features such as distributed computing, networking, and parallel processing.

3D CAD Applications: Python can be used for 3D computer-aided design (CAD) applications through libraries such as Blender.

Machine Learning: Python is widely used for machine learning due to its simplicity, ease of use, and availability of powerful machine learning libraries.

Computer Vision or Image Processing Applications: Python can be used for computer vision and image processing applications through powerful libraries such as OpenCV and Scikit-image.

Speech Recognition: Python can be used for speech recognition applications through libraries such as SpeechRecognition and PyAudio.

Scientific computing: Libraries like NumPy, SciPy, and Pandas provide advanced numerical computing capabilities for tasks like data analysis, machine learning, and more.

Education: Python's easy-to-learn syntax and availability of many resources make it an ideal language for teaching programming to beginners.

Testing: Python is used for writing automated tests, providing frameworks like unit tests and pytest that help write test cases and generate reports.

Gaming: Python has libraries like Pygame, which provide a platform for developing games using Python.

IoT: Python is used in IoT for developing scripts and applications for devices like Raspberry Pi, Arduino, and others.

Networking: Python is used in networking for developing scripts and applications for network automation, monitoring, and management.

DevOps: Python is widely used in DevOps for automation and scripting of infrastructure management, configuration management, and deployment processes.

Finance: Python has libraries like Pandas, Scikit-learn, and Statsmodels for financial modeling and analysis.

Audio and Music: Python has libraries like Pyaudio, which is used for audio processing, synthesis, and analysis, and Music21, which is used for music analysis and generation.

Writing scripts: Python is used for writing utility scripts to automate tasks like file operations, web scraping, and data processing.

Python Popular Frameworks and Libraries

Python has wide range of libraries and frameworks widely used in various fields such as machine learning, artificial intelligence, web applications, etc. We define some popular frameworks and libraries of Python as follows.

Web development (Server-side) - Django Flask, Pyramid, CherryPy

GUIs based applications - Tk, PyGTK, PyQt, PyJs, etc.

Machine Learning - TensorFlow, PyTorch, Scikit-learn, Matplotlib, Scipy, etc.

Mathematics - Numpy, Pandas, etc.

BeautifulSoup: a library for web scraping and parsing HTML and XML

Requests: a library for making HTTP requests

SQLAlchemy: a library for working with SQL databases

Kivy: a framework for building multi-touch applications

Pygame: a library for game development

Pytest: a testing framework for Python

Django REST framework: a toolkit for building RESTful APIs

FastAPI: a modern, fast web framework for building APIs

Streamlit: a library for building interactive web apps for machine learning and data science

NLTK: a library for natural language processing

EXPERIMENT NO: 2

PYTHON INSTALLATION

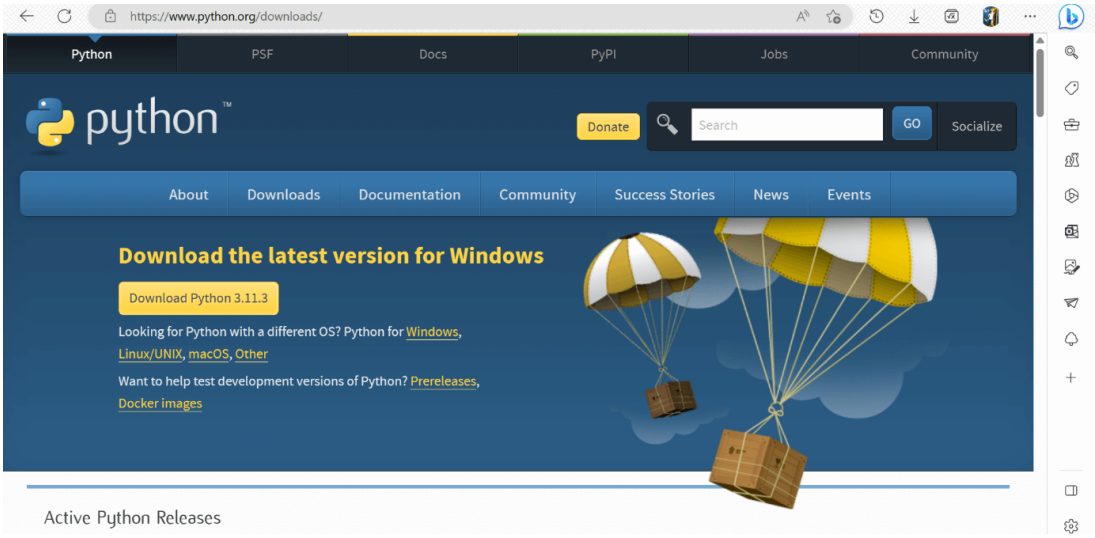
AIM: To install python on windows operating system

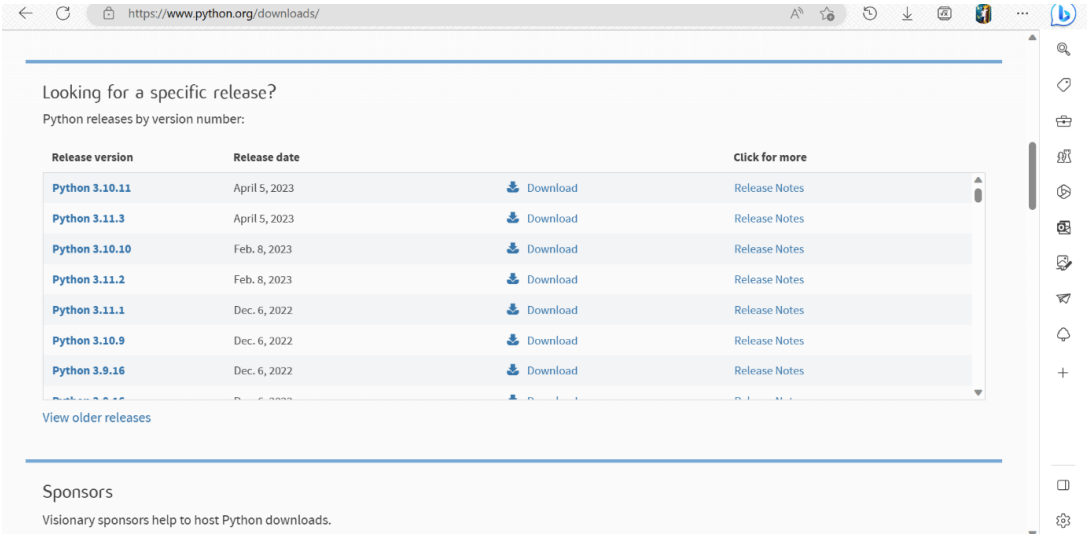
Procedure:

Visit the link https://www.python.org to download the latest release of Python and install on Windows operating system.

Step - 1: Select the Python's version to download.

Click on the download button to download the exe file of Python.

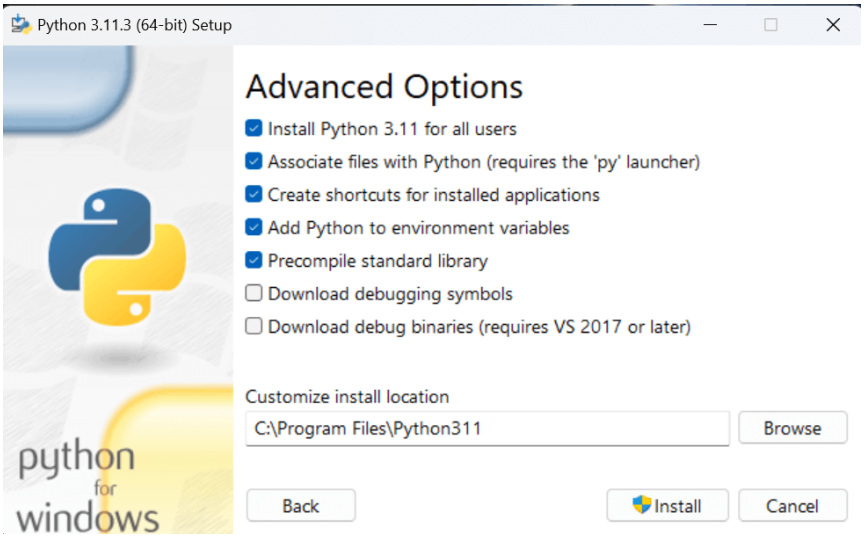


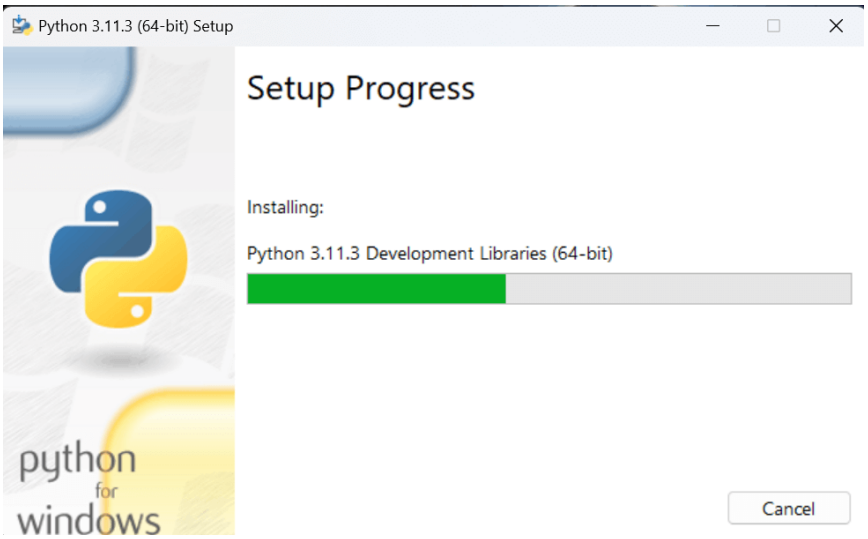


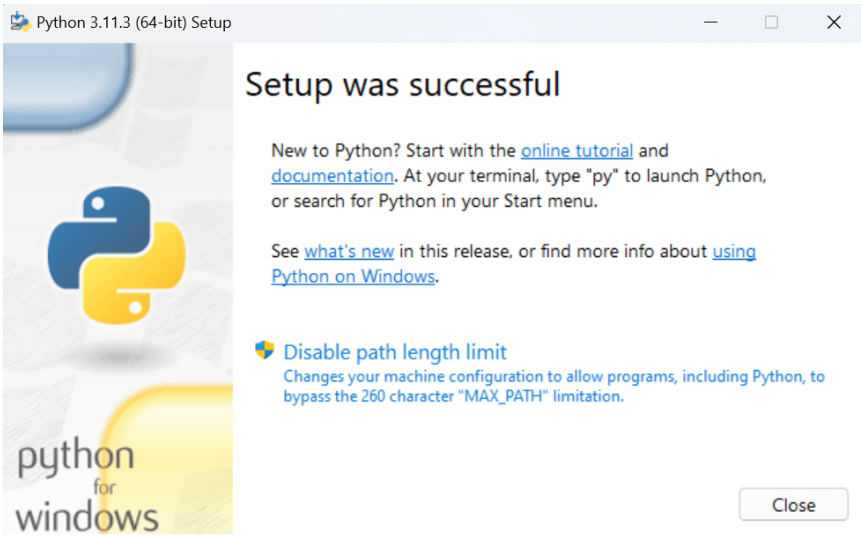
Step - 2: Click on the Install Now

Double-click the executable file, which is downloaded.


Python 3.11.3 (64-bit) Setup
Install Python 3.11.3 (64-bit)
Select Install Now to install Python with default settings, or choose Customize to enable or disable features.
Install Now
C:\Users\bharg\AppData\Local\Programs\Python\Python311
Includes IDLE, pip and documentation
Creates shortcuts and file associations
Customize installation
Choose location and features
python for windows
Use admin privileges when installing py.exe
Add python.exe to PATH
Cancel


Python 3.11.3 (64-bit) Setup
Advanced Options
Install Python 3.11 for all users
Associate files with Python (requires the 'py' launcher)
Create shortcuts for installed applications
Add Python to environment variables
Precompile standard library
Download debugging symbols
Download debug binaries (requires VS 2017 or later)
Customize install location
C:\Program Files\Python311
Browse
python for windows
Back
Install
Cancel


Python 3.11.3 (64-bit) Setup
Setup Progress
Installing:
Python 3.11.3 Development Libraries (64-bit)
python for windows
Cancel



Verifying the Python Installation

To verify whether the python is installed or not in the system:

Go to "Start" button, and search " cmd ".

Then type, " python - - version ".

If python is successfully installed, then we can see the version of the python installed.

Opening idle

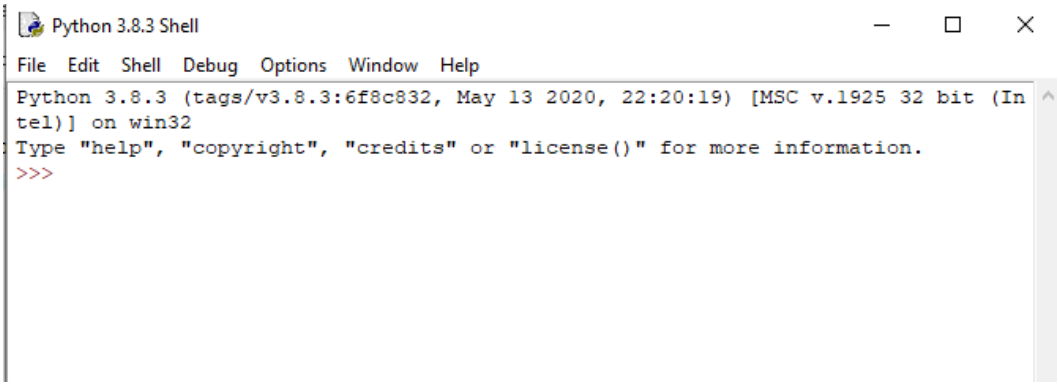
Now, to work on our first python program, we will go the interactive interpreter prompt(idle). To open this, go to "Start" and type idle. Then, click on open to start working on idle.

Interactive interpreter prompt

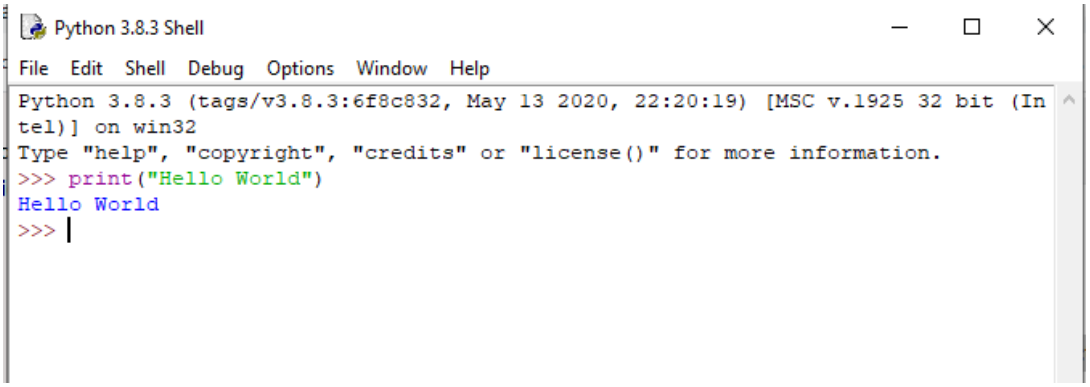
Python provides us the feature to execute the Python statement one by one at the interactive prompt. It is preferable in the case where we are concerned about the output of each line of our Python program.

To open the interactive mode, open the terminal (or command prompt) and type python (python3 in case if you have Python2 and Python3 both installed on your system).

It will open the following prompt where we can execute the Python statement and check their impact on the console.

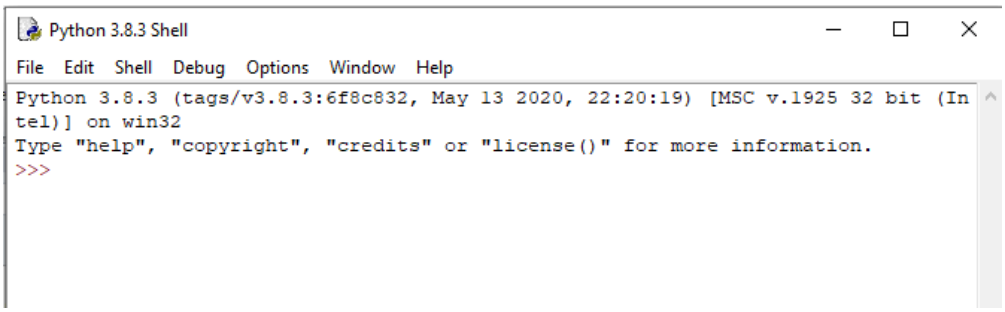


After writing the print statement, press the Enter key.

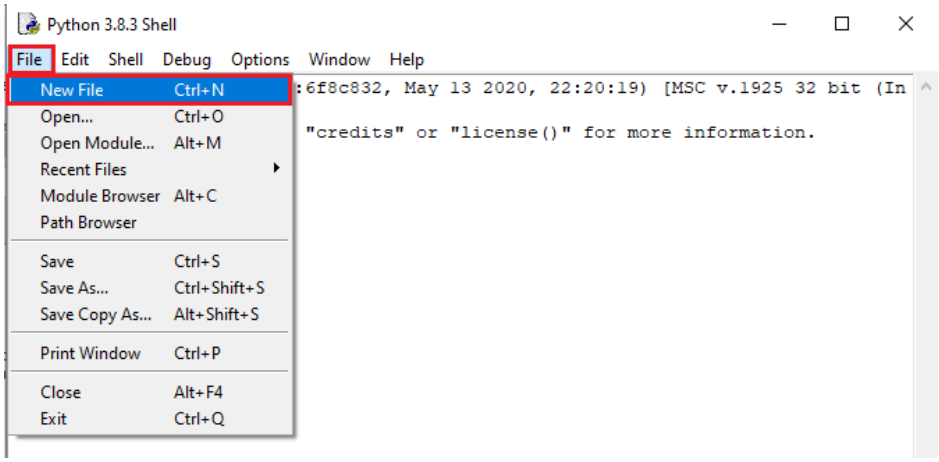


Using a script file (Script Mode Programming)

Using the script mode, we can write multiple lines code into a file which can be executed later. For this purpose, we need to open an editor like notepad, create a file named and save it with .py extension, which stands for "Python". Now, we will implement the above example using the script mode.



Step - 1: Open the Python interactive shell, and click "File" then choose "New", it will open a new blank script in which we can write our code.



Step -2: Now, write the code and press "Ctrl+S" to save the file.

Step - 3: After saving the code, we can run it by clicking "Run" or "Run Module". It will display the output to the shell.

EXPERIMENT NO: 3

DATA TYPES

Question:

Write a program to demonstrate different data types in python

Program:

```
a=3
b=2+3j
c=15.32
d="Hello World"
e= True
print("the data type of a is:",a, "\n its type is:",type(a))
print("the data type of b is:",b, "\nits type is:",type(b))
print("the data type of c is:",c, "\nits type is:",type(c))
print("the data type of  d is:",d, "\nits type is:",type(d))
print("the data type of e is:",e, "\nits type is:",type(e))
```

Result:

The output is verified.

Output:

EXPERIMENT NO: 4
ARITHMETIC OPERATIONS

Question:

Write a program to perform different arithmetic operations on numbers in python

Program:

```
a,b=10,3
print("addition of a:",a,"&b:",b,"is:",a+b)
print("subtraction of a:",a,"&b:",b,"is:",a-b)
print("multiplication of a:",a,"&b:",b,"is:",a*b)
print("division of a:",a,"&b:",b,"is:",a/b)
print("integer division of a:",a,"&b:",b,"is:",a//b)
print("modulus of a:",a,"&b:",b,"is:",a%b)
print("exponent of a:",a,"&b:",b,"is:",a**b)
```

Result:

The output is verified.

Output:

EXPERIMENT NO: 5
STRING OPERATION

Question:

Write a program to create, concatenate and print a string and accessing substring from a given string.

Program:

```
pi=3.14
a= "Hello"
b= "World"
print("the value of a is:",a)
print("the value of b is:",b)
print("after concatenating a and b the string is:",a+b)
text = "The value of pi is " + str(pi)
print(text)
print(a[1:4:1])          #string[start:end:step]
print(a[-1])             #display last character
print(a[-2:])            #display last two characters
```

Result:

The output is verified.

Output:

EXPERIMENT NO: 6
USING FOR LOOP

Question:

Write a program to calculate the factorial of a number using FOR loop

Program:

```
a=int(input("Enter the number: "))
fact=1
for i in range(a,1,-1):
   fact=fact*i
print("Factorial is ", fact)
```

Result:

The output is verified.

Output:

EXPERIMENT NO: 7
USING WHILE LOOP

Question:

Write a program to print Fibonacci series using WHILE loop

Program:

```
l = int(input("Enter the limit: "))

a = 0
b = 1
s = 1
count = 1
print("Fibonacci series is: ", end = "  ")
while (count <= l):
  count += 1
  print(a, end="  ")
  a = b
  b = s
  s = a + b
```

Result:

The output is verified.

Output:

EXPERIMENT NO: 8
USING FUNCTIONS

Question:

Write a program to check the number is Prime or not using function.

Program:

```
num=int(input("Enter the number: "))

def is_prime(n):
   if (n==1):
      return False
   elif (n==2):
      return True;
   else:
      for x in range(2,n):
         if(n % x==0):
            return False
      return True

if (is_prime(num)):
   print(num, "is prime")
else:
   print(num, "is not prime")
```

Result:

The output is verified.

Output:

EXPERIMENT NO: 9
USING LIST

Question:

Write a program to create, append and remove lists in python.

Program:

```
my_list = ["zero", "one", "two", "three"];
print("Elements of the list, my_list are:");
for ml in my_list:
        print(ml)
my_new_list = ["five", "six"];
my_list += my_new_list;
print("List's items after concatenating:");
for l in my_list:
        print(l);
index = input("\nEnter index no:");
index = int(index);
print("Deleting the element present at index number",index);
del my_list[index];
print("\nNow elements of the list, my_list are:");
for ml in my_list:
        print(ml);
```

Result:

The output is verified.

Output:

EXPERIMENT NO: 10
USING TUPLES

Question:

Write a program to demonstrate working with tuples in python.

Program:

```
print("Creating an empty tuple...");
my_tuple = ();
print("An empty tuple, my_tuple is created successfully.");
if not my_tuple:
    print("The tuple, my_tuple, contains no any item.");
print("Inserting some items to the tuple...");
my_tuple = ("a", "bb", "ccc", "dddd");
print("\nPrinting the tuple...");
print(my_tuple);
print("\nNow printing each item in the tuple...");
for item_in_tuple in my_tuple:
    print(item_in_tuple);
```

Result:

The output is verified.

Output:

EXPERIMENT NO: 11
USING DICTIONARIES

Question:

Write a program to demonstrate working with dictionaries in python.

Program:

```
# empty dictionary
my_dict = {}

my_dict = dict([(1,'apple'), (2,'ball')])
my_dict = {'name':'John', 'age': 26}
print(my_dict['name'])
print(my_dict.get('age'))

my_dict['age'] = 27
print(my_dict)

# add item
my_dict['address'] = 'Downtown'
print(my_dict)

# create a dictionary
squares = {1:1, 2:4, 3:9, 4:16, 5:25}

# remove a particular item
print(squares.pop(4))
print(squares)

# remove an arbitrary item
print(squares.popitem())
print(squares)

# delete a particular item
del squares[3]
print(squares)

# remove all items
squares.clear()
print(squares)
```

```
# delete the dictionary itself
del squares
print(squares)
```

Result:

The output is verified.

Output:

Experiment No: 12

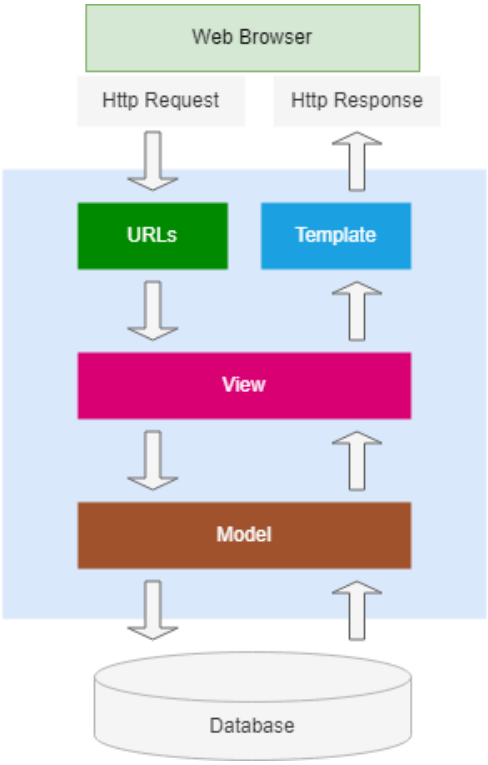
Django Overview

- Django is a Python web framework that includes a set of components for solving common web development problems.
- Django allows you to rapidly develop web applications with less code by taking advantage of its framework.
- Django follows the DRY (don't repeat yourself) principle, which allows you to maximize the code reusability.
- Django uses the MVT (Model-View-Template) pattern, which is slightly similar to MVC (Model-View-Controller) pattern.

MVT pattern

- Model – defines the data or contains the logic that interacts with the data in the database.
- View – communicates with the database via model and transfers data to the template for representing the data.
- Template – defines the template for displaying the data in the web browser.

Django Architecture



- The web browser requests a page by a URL and the web server passes the HTTP request to Django.
- Django matches the URL with URL patterns to find the first match.
- Django calls the view that corresponds to the matched URL.
- The view uses a model to retrieve data from the database.
- The model returns data to the view.
- The view renders a template and returns it as an HTTP response.

How to get Django?

- Django is available open-source under the BSD license
- Command:
    - pip install Django
    - python -m django --version

Creating a virtual environment

- A virtual environment creates an isolated environment that consists of an independent set of Python packages.
- By using virtual environments, you can have projects that use different versions of Django.

Creating a virtual environment

1. Create a new directory:
    - mkdir Django-workarea
    - cd Django-workarea
2. create a new virtual environment using the venv module
    - python -m venv venv
3. Activate the virtual environment in windows:
    - venv\scripts\activate
4. Activate the virtual environment in Ubuntu
    - source venv/bin/activate
5. Setup Django in virtual environment
    - pip install Django

Django commands

- Django comes with a command-line utility program called django-admin that manages administrative tasks such as creating a new project and executing the Django development server.
- Command to create a Project:
    - django-admin startproject django_project
    - cd django_project

- Command to create an App
    - python manage.py startapp app_name

Files in the Django project

- manage.py is a command-line program that you use to interact with the project like starting a development server and making changes to the database.
- django_project is a Python package that consists of the following files:
- __init__.py – is an empty file indicating that the django_project directory is a package.
- settings.py – contains the project settings such as installed applications, database connections, and template directories.
- urls.py – stores a list of routes that map URLs to views.
- wsgi.py – contains the configurations that run the project as a web server gateway interface (wsgi) application with WSGI-compatible web servers.
- asgi.py – contains the configurations that run the project as an asynchronous web server gateway interface (AWSGI) application with AWSGI-compatible web servers.

Running the Django development server

- python manage.py runserver

Django projects and applications

- In the Django framework:
    - A project is a Django installation with some settings.
    - An application is a group of models, views, templates, and URLs.
- A Django project may have one or more applications.
    - Eg., a project is like a website that may consist of several applications such as blogs, users, and wikis.
- A project is the entire Django application and an App is a module inside the project that deals with one specific use case
- An App is basically a web application that is created to perform a specific task
- A project is a collection of these apps
- A single project can consists of n number of apps and a single app can be in multiple projects

How a request is processed in Django?

- A user requests for a resource to the Django, Django works as a controller and check to the available resource in URL
- When Django server is started, the manage.py file searches for settings.py file, which contain information of all the applications installed in the project, middleware used, database connections and path to the main urls config
- Manage.py -> Settings.py -> urls.py -> views.py -> models.py -> templates

- Django first determines which root URL configuration module to be used
- Then that particular Python module urls is loaded and then Django looks for the variable url patterns
- Once that is done, the Django imports and calls the given view
- In case none of the URLs match the requested URL, Django invokes an error-handling view
- If URL maps, a view is called that interact with model and template, it renders a template
- Django responds back to the user and sends a template as a response

Command for Migrations in Django

- Command to create a migration file inside the migration folder:
    - python manage.py makemigrations
- After creating a migration, to reflect changes in the database permanently execute migrate command:
    - python manage.py migrate
- To see all migrations, execute the command:
    - python manage.py showmigrations

Command to create a Superuser in Django

- Command to create a Super User:
    - python manage.py createsuperuser
- Enter your desired username and press enter:
    - Username: admin
    - Email address:
- Final step is to enter your password twice
    - Password: ********
    - Password (again): ********
    - Superuser created successfully

Adding library

- pip install django-widget-tweaks

EXPERIMENT NO: 13

CREATE SIMPLE LOGIN PAGE

Aim: To create a simple login page consists of username and password using Django

Program:

==========
settings.py (project)
==========

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'emp_app',
    'widget_tweaks',
]
```

==========
urls.py (project)
==========

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('emp_app.urls'))
]
```

==========
urls.py (app) (Create this file if it is not there)
==========

```
from django.contrib import admin
from django.urls import path, include
from .import views

urlpatterns = [
    path('', views.index, name='index'),
```

```
    path('userlogin', views.userlogin, name='userlogin'),
]
```

=========

models.py

=========

```
from django.db import models
from django.contrib.auth.models import User

class Employee(models.Model):
    first_name = models.CharField(max_length=100, null=False)
    last_name = models.CharField(max_length=100)
    dept = models.ForeignKey(Department, on_delete=models.CASCADE)
    salary = models.IntegerField(default=0)
    bonus = models.IntegerField(default=0)
    role = models.ForeignKey(Role, on_delete=models.CASCADE)
    phone = models.IntegerField(default=0)
    hire_date = models.DateField()

    def __str__(self):
        return "%s %s %s" %(self.first_name, self.last_name, self.phone)
```

=========

admin.py

=========

```
from django.contrib import admin
from .models import Employee

# Register your models here.
admin.site.register(Employee)
```

=======

apps.py

=======

```
from django.apps import AppConfig
class EmpAppConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'emp_app'
```

========

views.py

========

```
from django.shortcuts import render, HttpResponse
from django.contrib.messages import *
from django.contrib.auth import authenticate, login
from .models import *
from .forms import *
from datetime import datetime
from django.db.models import Q

# Create your views here.
def index(request):
    return render(request, 'index.html')

def userlogin(request):
    if request.method == 'POST':
        # process the request if posted data available
        u = request.POST['username']
        p = request.POST['password']

        #check whether username, passowrd combination is correct
        user=authenticate(username=u,password=p)
        if user is not None:
            #save session as cookie to login the user
            login(request, user)
            #success...
            return render(request, 'index.html', {'login_user':user.get_username})
        else:
            #incorrect credentials
            return render(request, 'userlogin.html',{'error_message':'Incorrect login credentials'})
    else:
        return render(request, 'userlogin.html')
```

==========

userlogin.html

==========

```
<!doctype html>
<html lang="en">
    {% load widget_tweaks %}
  <head>
     <title> project title here </title>
  </head>
  <body>
  {% block content %}
  <div class="container">
          <h1> heading here </h1> <hr>
         <form action="{% url 'userlogin' %}" method="POST">
```

```
        {% csrf_token %}
          {% if error_message %}
            <h4> {{ error_message }} </h4> <br>
          {% endif %}
          <label for="username">User Name : </label>
          <input type="text" id="username" name="username" value="" maxlength="20"
size="20"><br><br>
          <label for="password">Password : </label>
          <input type="password" id="password" name="password" value="" maxlength="20"
size="20"><br><br>
          <input type="submit" value="Login" >  
      </form>
  </div>
  {% endblock %}
  </body>
</html>
```

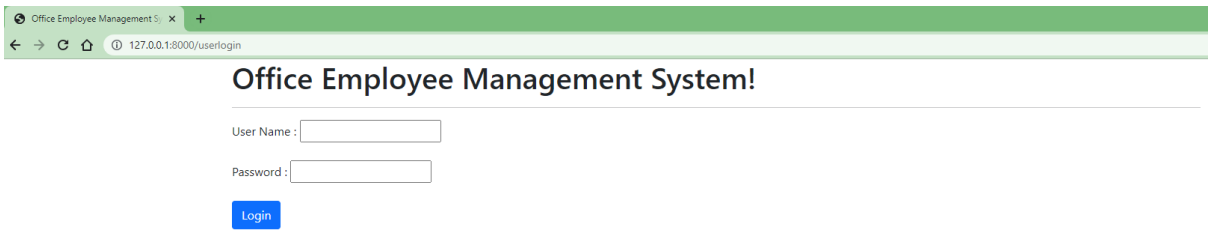
Commands to run the project:

- python manage.py makemigrations app_name
- python manage.py migrate app_name
- python manage.py runserver

Result:

The output is verified

Output:



EXPERIMENT NO: 14

CREATE EMPLOYEE MANAGEMENT APPLICATION

Aim: To create a simple employee management application using Django

Program:

==========
settings.py (project)
==========

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'emp_name',
    'widget_tweaks',
]

ROOT_URLCONF = 'office_emp_proj.urls'
```

==========
urls.py (project)
==========

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include(emp_app.urls'))
]
```

==========
urls.py (app) (Create this file if it is not there)
==========

```
from django.contrib import admin
from django.urls import path, include
from . import views
```

```
urlpatterns = [
    path('', views.index, name='index'),
    path('all_emp', views.all_emp, name='all_emp'),
    path('add_emp', views.add_emp, name='add_emp'),
    path('filter_emp', views.filter_emp, name='filter_emp'),
    path('userlogin', views.userlogin, name='userlogin'),
]
```

=========

models.py

=========

```
from django.db import models
from django.contrib.auth.models import User

class Department(models.Model):
    name = models.CharField(max_length=100,null=False)
    location = models.CharField(max_length=100)

    def __str__(self):
        return self.name

class Role(models.Model):
    name = models.CharField(max_length=100, null=False)

    def __str__(self):
        return self.name

class Employee(models.Model):
    first_name = models.CharField(max_length=100, null=False)
    last_name = models.CharField(max_length=100)
    dept = models.ForeignKey(Department, on_delete=models.CASCADE)
    salary = models.IntegerField(default=0)
    bonus = models.IntegerField(default=0)
    role = models.ForeignKey(Role, on_delete=models.CASCADE)
    phone = models.IntegerField(default=0)
    hire_date = models.DateField()

    def __str__(self):
        return "%s %s %s" %(self.first_name, self.last_name, self.phone)
```

=========

admin.py

```
=========
from django.contrib import admin
from .models import Employee, Role, Department

# Register your models here.
admin.site.register(Employee)
admin.site.register(Role)
admin.site.register(Department)
```

=======

apps.py

=======

```
from django.apps import AppConfig
class EmpAppConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = ' emp_app'
```

========

views.py

========

```
from django.shortcuts import render, HttpResponse
from django.contrib.messages import *
from django.contrib.auth import authenticate, login
from .models import *
from .forms import *
from datetime import datetime
from django.db.models import Q


# Create your views here.
def index(request):
    return render(request, 'index.html')

def userlogin(request):
    if request.method == 'POST':
        # process the request if posted data available
        u = request.POST['username']
        p = request.POST['password']

        #check whether username, passowrd combination is correct
        user=authenticate(username=u,password=p)
        if user is not None:
            #save session as cookie to login the user
            login(request, user)
```

```
            #success...
            return render(request, 'index.html', {'login_user':user.get_username})
        else:
            #incorrect credentials
            return render(request, 'userlogin.html',{'error_message':'Incorrect login credentials'})
    else:
        return render(request, 'userlogin.html')

def all_emp(request):
    emps = Employee.objects.all()
    context = {
        'emps': emps
    }
    print(context)
    return render(request, 'view_all_emp.html', context)


def add_emp(request):
    if request.method == 'POST':
        first_name = request.POST['first_name']
        last_name = request.POST['last_name']
        salary = int(request.POST['salary'])
        bonus = int(request.POST['bonus'])
        phone = int(request.POST['phone'])
        dept = int(request.POST['dept'])
        role = int(request.POST['role'])
        new_emp = Employee(first_name= first_name, last_name=last_name, salary=salary,
bonus=bonus, phone=phone, dept_id = dept, role_id = role, hire_date = datetime.now())
        new_emp.save()
        return HttpResponse('Employee added Successfully')
    elif request.method=='GET':
        return render(request, 'add_emp.html')
    else:
        return HttpResponse("An Exception Occured! Employee Has Not Been Added")


def remove_emp(request, emp_id = 0):
    if emp_id:
        try:
            emp_to_be_removed = Employee.objects.get(id=emp_id)
            emp_to_be_removed.delete()
            return HttpResponse("Employee Removed Successfully")
        except:
            return HttpResponse("Please Enter A Valid EMP ID")
    emps = Employee.objects.all()
    context = {
        'emps': emps
    }
```

```
    return render(request, 'remove_emp.html',context)


def filter_emp(request):
    if request.method == 'POST':
        name = request.POST['name']
        dept = request.POST['dept']
        role = request.POST['role']
        emps = Employee.objects.all()
        if name:
            emps = emps.filter(Q(first_name__icontains = name) | Q(last_name__icontains =
name))
        if dept:
            emps = emps.filter(dept__name__icontains = dept)
        if role:
            emps = emps.filter(role__name__icontains = role)

        context = {
            'emps': emps
        }
        return render(request, 'view_all_emp.html', context)

    elif request.method == 'GET':
        return render(request, 'filter_emp.html')
    else:
        return HttpResponse('An Exception Occurred')
```

==========

index.html

==========

```
<!doctype html>
<html lang="en">
  <head>
    <title>project name</title>
  </head>
  <body>
  <div class="container">
          <h1> heading here </h1>
              <h3> Welcome {{ login_user }} ! </h3>
              <hr>
              <a href="/all_emp" role="button">View All Employee</a>
              <a href="/add_emp" role="button">Add An Employee</a>
              <a href="/remove_emp" role="button">Remove An Employee</a>
              <a href="/filter_emp" role="button">Filter Employee Details</a>
    </div>
  </body></html>
```

==========

userlogin.html

==========

```
<!doctype html>
<html lang="en">
   {% load widget_tweaks %}
  <head>
   <title> project title here </title>
  </head>
  <body>
  {% block content %}
  <div class="container">
        <h1> heading here </h1> <hr>
       <form action="{% url 'userlogin' %}" method="POST">
         {% csrf_token %}
           {% if error_message %}
             <h4> {{ error_message }} </h4> <br>
           {% endif %}
           <label for="username">User Name : </label>
           <input type="text" id="username" name="username" value="" maxlength="20"
size="20"><br><br>
           <label for="password">Password : </label>
           <input type="password" id="password" name="password" value="" maxlength="20"
size="20"><br><br>
           <input type="submit" value="Login" >  
      </form>
  </div>
  {% endblock %}
  </body>
</html>
```

==========

add_emp.html

==========

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Add Employee</title>
   <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-
1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3"
crossorigin="anonymous">
</head>
  <body>
  <div class="container">
```

```
        <h1>Add An Employee!</h1>
     <hr>

<form action="/add_emp" method="post">
   {% csrf_token %}

   <label for="first_name">First Name</label><br>
   <input type="text" id="first_name" name="first_name" value="{{ first_name }}"><br>
   <label for="last_name">Last Name</label><br>
   <input type="text" id="last_name" name = "last_name" value="{{ last_name }}"><br>
   <label for="salary">Salary</label><br>
   <input type="number" id="salary" name="salary" value="{{ salary }}"><br>
   <label for="dept">Department</label><br>
   <input type="number" id="dept" name="dept" value="{{ dept }}"><br>
   <label for="role">Role</label><br>
   <input type="number" id="role" name="role" value="{{ role }}"><br>
   <label for="bonus">Bonus</label><br>
   <input type="number" id="bonus" name="bonus" value="{{ bonus }}"><br>
   <label for="phone">Phone Number</label><br>
   <input type="number" id="phone" name="phone" value="{{ phone }}"><br>
   <hr>
   <button type="submit" class="btn btn-primary">Submit</button>

</form>

  </div>

   <!-- Optional JavaScript; choose one of the two! -->

   <!-- Option 1: Bootstrap Bundle with Popper -->
   <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"
integrity="sha384-
ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBO0LRn5q+8nbTov4+1p"
crossorigin="anonymous"></script>

   <!-- Option 2: Separate Popper and Bootstrap JS -->
   <!--
   <script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.10.2/dist/umd/popper.min.js"
integrity="sha384-
```

```
7+zCNj/IqJ95wo16oMtfsKbZ9ccEh31eOz1HGyDuCQ6wgnyJNSYdrPa03rtR1zdB"
crossorigin="anonymous"></script>
   <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.min.js"
integrity="sha384-
QJHtvGhmr9XOIpI6YVutG+2QOK9T+ZnN4kzFN1RtK3zEFEIsxhlmWl5/YESvpZ13"
crossorigin="anonymous"></script>
   -->
  </body>
</html>
```

==========

remove_emp.html

==========

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Remove Employee</title>
   <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-
1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3"
crossorigin="anonymous">
</head>
  <body>
  <div class="container">
        <h1>Remove An Employee!</h1>
     <hr>
     <div class="dropdown">
  <button class="btn btn-primary dropdown-toggle" type="button"
id="dropdownMenuButton1" data-bs-toggle="dropdown" aria-expanded="false">
   Choose Employee To Be Removed
  </button>
  <ul class="dropdown-menu" aria-labelledby="dropdownMenuButton1">
     {% for emp in emps %}
   <li><a class="dropdown-item" href="/remove_emp/{{emp.id}}">{{emp.first_name}}
{{emp.last_name}}</a></li>
     {% endfor %}
  </ul>
</div>
```

```
  </div>

    <!-- Optional JavaScript; choose one of the two! -->

    <!-- Option 1: Bootstrap Bundle with Popper -->
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"
integrity="sha384-
ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBO0LRn5q+8nbTov4+1p"
crossorigin="anonymous"></script>

    <!-- Option 2: Separate Popper and Bootstrap JS -->
    <!--
    <script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.10.2/dist/umd/popper.min.js"
integrity="sha384-
7+zCNj/IqJ95wo16oMtfsKbZ9ccEh31eOz1HGyDuCQ6wgnyJNSYdrPa03rtR1zdB"
crossorigin="anonymous"></script>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.min.js"
integrity="sha384-
QJHtvGhmr9XOIpI6YVutG+2QOK9T+ZnN4kzFN1RtK3zEFEIsxhlmWl5/YESvpZ13"
crossorigin="anonymous"></script>
    -->
  </body>
</html>
```

==========

view_all_emp.html

==========

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>View All Employee</title>
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-
1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3"
crossorigin="anonymous">

</head>
  <body>
```

```
<div class="container">
      <h1>All Employee Details!</h1>
    <hr>
   <table class="table table-dark">
<thead>
  <tr>
    <th scope="col">#</th>
    <th scope="col">First Name</th>
    <th scope="col">Last Name</th>
    <th scope="col">Salary</th>
    <th scope="col">Bonus</th>
    <th scope="col">Phone Number</th>
    <th scope="col">Role</th>
    <th scope="col">Department</th>
    <th scope="col">Location</th>
    <th scope="col">Hiredate</th>
  </tr>
</thead>
      {% for emp in emps %}
<tbody>
  <tr>
    <th scope="row">{{emp.id}}</th>
    <td>{{emp.first_name}}</td>
    <td>{{emp.last_name}}</td>
    <td>{{emp.salary}}</td>
    <td>{{emp.bonus}}</td>
    <td>{{emp.phone}}</td>
    <td>{{emp.role.name}}</td>
    <td>{{emp.dept.name}}</td>
    <td>{{emp.dept.location}}</td>
    <td>{{emp.hire_date}}</td>
  </tr>
</tbody>
      {% endfor %}
</table>

</div>

  <!-- Optional JavaScript; choose one of the two! -->

  <!-- Option 1: Bootstrap Bundle with Popper -->
```

```
   <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"
integrity="sha384-
ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBO0LRn5q+8nbTov4+1p"
crossorigin="anonymous"></script>

   <!-- Option 2: Separate Popper and Bootstrap JS -->

<!--    <script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.10.2/dist/umd/popper.min.js"
integrity="sha384-
7+zCNj/IqJ95wo16oMtfsKbZ9ccEh31eOz1HGyDuCQ6wgnyJNSYdrPa03rtR1zdB"
crossorigin="anonymous"></script>-->
<!--    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.min.js"
integrity="sha384-
QJHtvGhmr9XOIpI6YVutG+2QOK9T+ZnN4kzFN1RtK3zEFEIsxhlmWl5/YESvpZ13"
crossorigin="anonymous"></script>-->

  </body>
</html>
```

==========

filter_emp.html

==========

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Filter Employee</title>
   <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-
1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3"
crossorigin="anonymous">
</head>
  <body>
  <div class="container">
        <h1>Filter Employee Details!</h1>
     <hr>
     <form action="/filter_emp" method="post">
   {% csrf_token %}
   <label for="name">Employee First/Last Name</label><br>
```

```
    <input type="text" id="name" name="name" value="{{ name }}"><br>
    <label for="dept">Department</label><br>
    <input type="text" id="dept" name="dept" value="{{ dept }}"><br>
    <label for="role">Role</label><br>
    <input type="text" id="role" name="role" value="{{ role }}"><br>
    <hr>
    <button type="submit" class="btn btn-primary">Submit</button>
</form>


  </div>

    <!-- Optional JavaScript; choose one of the two! -->

    <!-- Option 1: Bootstrap Bundle with Popper -->
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"
integrity="sha384-
ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBO0LRn5q+8nbTov4+1p"
crossorigin="anonymous"></script>

    <!-- Option 2: Separate Popper and Bootstrap JS -->
    <!--
    <script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.10.2/dist/umd/popper.min.js"
integrity="sha384-
7+zCNj/IqJ95wo16oMtfsKbZ9ccEh31eOz1HGyDuCQ6wgnyJNSYdrPa03rtR1zdB"
crossorigin="anonymous"></script>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.min.js"
integrity="sha384-
QJHtvGhmr9XOIpI6YVutG+2QOK9T+ZnN4kzFN1RtK3zEFEIsxhlmWl5/YESvpZ13"
crossorigin="anonymous"></script>
    -->
  </body>
</html>
```

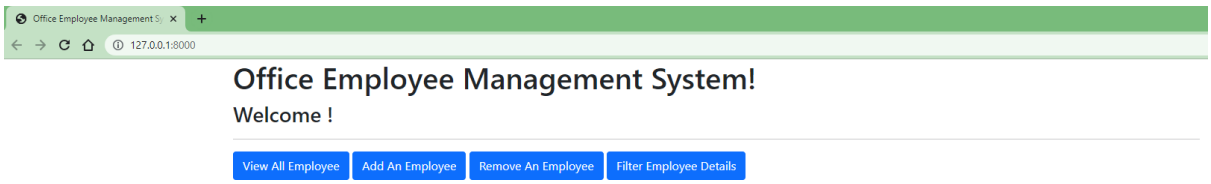
Commands to run the project:

- python manage.py makemigrations emp_app
- python manage.py migrate emp_app
- python manage.py runserver

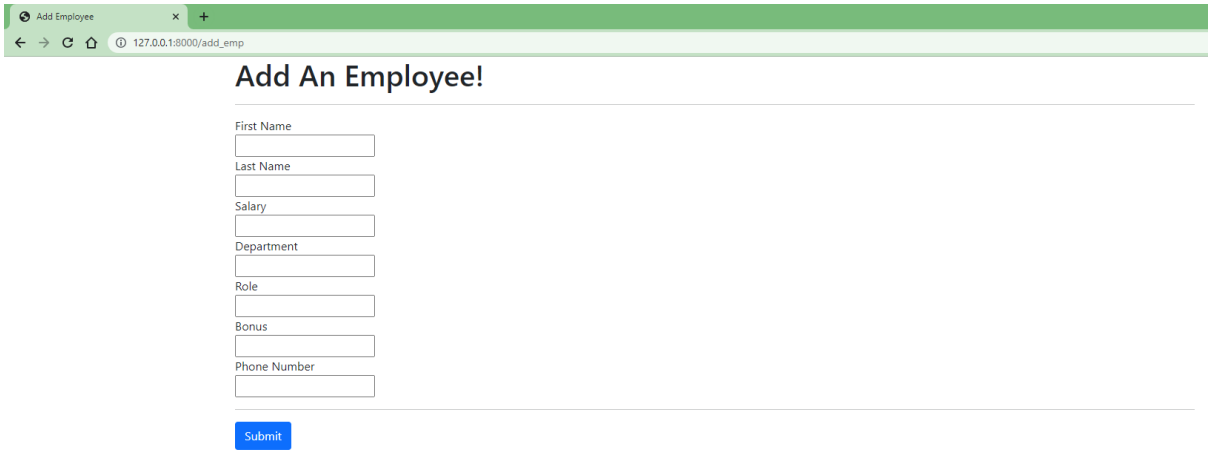
Result:

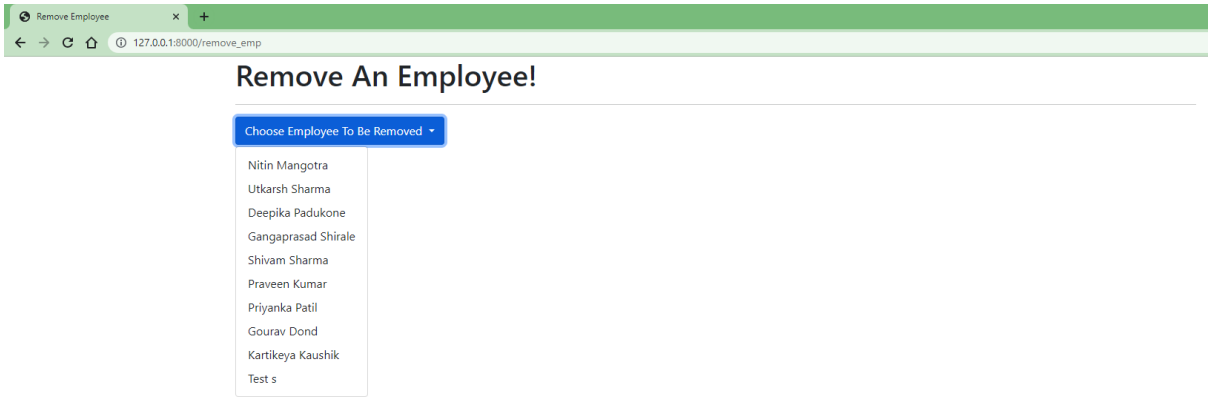
The output is verified

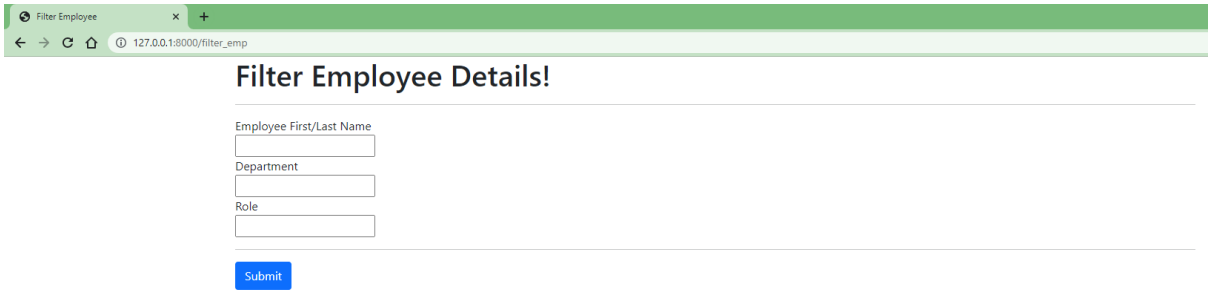
Output:



#	First Name	Last Name	Salary	Bonus	Phone Number	Role	Department	Location	Hiredate
1	Nitin	Mangotra	200000	2000	9876543210	Python Developer	Java Development	Mumbai	Jan. 15, 2022
3	Utkarsh	Sharma	400000	4000	9876543212	Tester	Automation	Delhi	Jan. 15, 2022
4	Deepika	Padukone	600000	4500	9876543213	Senior HR	HR	Pune	Jan. 15, 2022
6	Gangaprasad	Shirale	800000	8000	9876543214	Django Devloper	Python Development	Delhi	Jan. 15, 2022
8	Shivam	Sharma	345000	3450	1234567890	Tester	Manual Testing	Noida	Jan. 15, 2022
9	Praveen	Kumar	800000	8000	1234567654	Java Developer	Java Development	Mumbai	Jan. 15, 2022
10	Priyanka	Patil	660000	6600	987654321	.net Developer	.net Development	Bangalore	Jan. 15, 2022
11	Gourav	Dond	500000	500	3456789098	Project Manager	Automation	Delhi	Jan. 15, 2022
12	Kartikeya	Kaushik	350000	3600	1234567898	.net Developer	Python Development	Delhi	Jan. 15, 2022
16	Test	s	3232	223	8078283451	Java Developer	HR	Pune	April 27, 2023




Remove Employee
127.0.0.1:8000/remove_emp
Remove An Employee!
Choose Employee To Be Removed
Nitin Mangotra
Utkarsh Sharma
Deepika Padukone
Gangaprasad Shirale
Shivam Sharma
Praveen Kumar
Priyanka Patil
Gourav Dond
Kartikeya Kaushik
Test s


Filter Employee
127.0.0.1:8000/filter_emp
Filter Employee Details!
Employee First/Last Name
Department
Role
Submit

****