

## PYTHON DEVELOPEMENT

### Basic:

Task1:simple calculator

Description: Develop a basic calculator that can perform four primary arithmetic operations: addition, subtraction, multiplication, and division.

objective:

- Create functions for each operation.
- Take two inputs from the user and allow them to select the desired operation.
- Handle division by zero with appropriate error messages.

code:

```
def add(a, b):
```

```
    return a + b
```

```
def subtract(a, b):
```

```
    return a - b
```

```
def multiply(a, b):
```

```
    return a * b
```

```
def divide(a, b):
```

```
    if b == 0:
```

```
        return "Error: Division by zero is not allowed"
```

```
    else:
```

```
        return a / b
```

```
num1 = float(input("Enter first number: "))
```

```
num2 = float(input("Enter second number: "))
```

```
print("\nSelect Operation:")  
  
print("1. Addition")  
  
print("2. Subtraction")  
  
print("3. Multiplication")  
  
print("4. Division")  
  
choice = input("Enter choice (1/2/3/4): ")  
  
if choice == '1':  
    print("Result:", add(num1, num2))  
  
elif choice == '2':  
    print("Result:", subtract(num1, num2))  
  
elif choice == '3':  
    print("Result:", multiply(num1, num2))  
  
elif choice == '4':  
    print("Result:", divide(num1, num2))  
  
else:  
    print("Invalid choice")
```

## Task 2: Number Guessing Game.

Description: Write a program that randomly generates a number between 1 and 100. The user has to guess the number, and the program will give feedback if the guess is too high or too low.

objective :

- Use the random module to generate a random number.
- Give the user multiple attempts to guess the number.
- Provide appropriate feedback (e.g., "Too high" or "Too low").

-Exit the game if the user guesses correctly or after a maximum number of attempts.

code:

```
import random

secret_number = random.randint(1, 100)

max_attempts = 7

attempts = 0

print("🎯 Number Guessing Game")

print("Guess a number between 1 and 100")

print("You have", max_attempts, "attempts\n")

while attempts < max_attempts:

    guess = int(input("Enter your guess: "))

    attempts += 1

    if guess > secret_number:

        print("Too high!\n")

    elif guess < secret_number:

        print("Too low!\n")

    else:

        print(f"🎉 Congratulations! You guessed the number in {attempts} attempts.")

        break

if attempts == max_attempts and guess != secret_number:

    print("❌ Game Over!")

    print("The correct number was:", secret_number)
```

Task 3: Word Counter Read the content of a file.

Description: Create a Python program that reads a text file and counts the number of words in it.

## Objective

- Split the content into words and count them.
- Handle exceptions,such a file not found please give a random file to execute that program.

code:

```
def word_counter(filename):  
    try:  
        with open(filename, "r") as file:  
            content = file.read()  
            words = content.split()  
            print("Total number of words:", len(words))  
    except FileNotFoundError:  
        print("Error: File not found. Please check the file name.")  
    except Exception as e:  
        print("An unexpected error occurred:", e)  
  
# Call the function  
word_counter("python.txt")
```

## Intermediate:

### Task1:To-Do List Application

Description:Build a simple command-line to-do list application. Users should be able to add, delete, mark as done, and list tasks.

## Objective:

- Implement the ability to add,view and delete task.
- Store the tasks in a file (either CSV or JSON format).

-Mark tasks as complete.

-Implement basic error handling(e.g.Trying to delete a task that does not exists)

code:

```
import json
```

```
import os
```

```
FILE_NAME = "tasks.json"
```

```
# Load tasks from file
```

```
def load_tasks():
```

```
    if not os.path.exists(FILE_NAME):
```

```
        return []
```

```
    try:
```

```
        with open(FILE_NAME, "r") as file:
```

```
            return json.load(file)
```

```
    except json.JSONDecodeError:
```

```
        return []
```

```
# Save tasks to file
```

```
def save_tasks(tasks):
```

```
    with open(FILE_NAME, "w") as file:
```

```
        json.dump(tasks, file, indent=4)
```

```
# Add a new task
```

```
def add_task(title):
```

```
    tasks = load_tasks()
```

```

tasks.append({"title": title, "completed": False})

save_tasks(tasks)

print("✅ Task added successfully!")

# View all tasks

def view_tasks():

    tasks = load_tasks()

    if not tasks:

        print("🚫 No tasks available.")

        return

    print("\n📋 To-Do List:")

    for index, task in enumerate(tasks, start=1):

        status = "✅ Done" if task["completed"] else "❌ Not Done"

        print(f"{index}. {task['title']} - {status}")

# Delete a task

def delete_task(task_number):

    tasks = load_tasks()

    try:

        removed = tasks.pop(task_number - 1)

        save_tasks(tasks)

        print(f"🗑️ Task '{removed['title']}' deleted.")

    except IndexError:

        print("❌ Error: Task does not exist.")

# Mark task as completed

def mark_completed(task_number):

```

```
tasks = load_tasks()

try:

    tasks[task_number - 1]["completed"] = True

    save_tasks(tasks)

    print("✓ Task marked as completed!")

except IndexError:

    print("✗ Error: Task does not exist.")
```

# Main menu

```
def main():

    while True:

        print("\n--- To-Do List Menu ---")

        print("1. Add Task")

        print("2. View Tasks")

        print("3. Delete Task")

        print("4. Mark Task as Done")

        print("5. Exit")

        choice = input("Enter your choice (1-5): ")

        if choice == "1":

            title = input("Enter task title: ")

            add_task(title)

        elif choice == "2":

            view_tasks()

        elif choice == "3":
```

```

    view_tasks()

    num = int(input("Enter task number to delete: "))

    delete_task(num)

elif choice == "4":

    view_tasks()

    num = int(input("Enter task number to mark as done: "))

    mark_completed(num)

elif choice == "5":

    print("👋 Exiting To-Do List. Goodbye!")

    break

else:

    print("❌ Invalid choice. Please try again.")

if __name__ == "__main__":

    main()

```

## Task2:Data Scraper

Description: Develop a web scraper to extract specific data from a website (e.g., news headlines, product prices).

Objective:

- Use the requests library to retrieve web page content.
- Parse the HTML using BeautifulSoup.
- Extract specific data, such as article titles or product details
- Saved scraped data into CSV file.

code:

```
import requests
```

```

from bs4 import BeautifulSoup
import csv
URL = "https://quotes.toscrape.com"
def scrape_quotes():
    try:
        response = requests.get(URL)
        response.raise_for_status() # raises error for bad response
        soup = BeautifulSoup(response.text, "html.parser")
        quotes = soup.find_all("div", class_="quote")
        data = []
        for quote in quotes:
            text = quote.find("span", class_="text").text
            author = quote.find("small", class_="author").text
            data.append([text, author])
        save_to_csv(data)
        print("✅ Data scraped and saved successfully!")
    except requests.exceptions.RequestException as e:
        print("❌ Error fetching the webpage:", e)
def save_to_csv(data):
    with open("quotes.csv", "w", newline="", encoding="utf-8") as file:
        writer = csv.writer(file)
        writer.writerow(["Quote", "Author"])
        writer.writerows(data)
if __name__ == "__main__":
    scrape_quotes()

```

### Task 3:API integration

Description: Write a Python script that interacts with an external API to fetch and display data (e.g., weather, cryptocurrency prices).

Objective:

- Use the requests library to make GET requests to an API.
- Parse and Display the data into user-friendly format.
- Handle errors, such as failed requests or invalid responses.

code:

```
import requests
```

```
def fetch_crypto_prices():
```

```
    url = "https://api.coingecko.com/api/v3/simple/price"
```

```
    params = {
```

```
        "ids": "bitcoin,ethereum",
```

```
        "vs_currencies": "usd"
```

```
    }
```

```
    try:
```

```
        response = requests.get(url, params=params, timeout=10)
```

```
        # Check if request was successful
```

```
        if response.status_code == 200:
```

```
            data = response.json()
```

```
            print("📊 Current Cryptocurrency Prices:")
```

```
            print("-" * 35)
```

```
            print(f"Bitcoin (BTC): ${data['bitcoin']['usd']}")
```

```
            print(f"Ethereum (ETH): ${data['ethereum']['usd']}")
```

```
        else:
```

```
            print("❌ Failed to fetch data")
```

```
        print("Status Code:", response.status_code)
except requests.exceptions.RequestException as e:
    print("⚠️ Error occurred while making the request:")
    print(e)
# Run the function
fetch_crypto_prices()
```

## **ADVANCE:**

### Task 1: Django Web Application with Authentication

Description: Build a fully functional web application using Django that includes user authentication (login, registration, and password reset). The application can be a blog, task manager, or e-commerce site.

Objective:

- Implement user registration, login, and logout functionality.
- Secure user passwords using Django's built-in authentication system.
- Create user roles (e.g., admin, regular user) with different permissions.
- Integrate password reset functionality via email.

-create project named 'auth\_project'

code:

```
python -m django startproject auth_project
```

```
cd auth_project
```

-create app named manage.py

code:

```
python manage.py startapp accounts
```

-setting.py(auth\_project/setting.py)

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

-views.py(accounts/views.py)

```
from django.shortcuts import render, redirect
```

```
from .forms import RegisterForm
```

```
def register(request):  
    if request.method == 'POST':  
        form = RegisterForm(request.POST)  
        if form.is_valid():  
            form.save()  
            return redirect('login')  
    else:  
        form = RegisterForm()  
    return render(request, 'register.html', {'form': form})
```

-urls.py(accounts/urls.py)

code:

```
from django.urls import path
```

```
from .views import register
```

```
from django.contrib.auth import views as auth_views
```

```
urlpatterns = [
```

```
    path('register/', register, name='register'),
```

```
    path('login/', auth_views.LoginView.as_view(template_name='login.html'),  
name='login'),
```

```
    path('logout/', auth_views.LogoutView.as_view(), name='logout'),
```

```
]
```

```
-login.html(templates/login.html)
```

code:

```
<form method="post">
```

```
{% csrf_token %}
```

```
{{ form.as_p }}
```

```
<button type="submit">Login</button>
```

```
</form>
```

```
-register.py(templates/register.py)
```

```
<form method="post">
```

```
{% csrf_token %}
```

```
{{ form.as_p }}
```

```
<button type="submit">Register</button>
```

```
</form>
```

- run the command

```
python manage.py makemigrations
```

```
python manage.py migrate
```

```
python manage.py createsuperuser
```

```
python manage.py runserver
```

Task2:Basic file Encryption/Decryption.

Description:Create a Python script that encrypts anddecrypts text files using a simple encryptionalgorithm (e.g., Caesar cipher or Fernet encryption).

Objective:

- Allow user to input file for encryption/decryption.
- Encrypt the file content and save it as a new file.
- Provide functionality to decrypt the file back to its original form.

code:

```
from cryptography.fernet import Fernet

import os

# Generate and save key (run once)

def generate_key():

    key = Fernet.generate_key()

    with open("secret.key", "wb") as key_file:

        key_file.write(key)


# Load existing key

def load_key():

    return open("secret.key", "rb").read()


# Encrypt file

def encrypt_file(filename):
```

```
key = load_key()
```

```
fernet = Fernet(key)
```

```
with open(filename, "rb") as file:
```

```
    data = file.read()
```

```
encrypted_data = fernet.encrypt(data)
```

```
with open(filename + ".encrypted", "wb") as file:
```

```
    file.write(encrypted_data)
```

```
print("✅ File encrypted successfully!")
```

```
# Decrypt file
```

```
def decrypt_file(filename):
```

```
    key = load_key()
```

```
    fernet = Fernet(key)
```

```
    with open(filename, "rb") as file:
```

```
        encrypted_data = file.read()
```

```
    decrypted_data = fernet.decrypt(encrypted_data)
```

```
    new_filename = filename.replace(".encrypted", ".decrypted.txt")
```

```
with open(new_filename, "wb") as file:
```

```
    file.write(decrypted_data)
```

```
print("✅ File decrypted successfully!")
```

```
# Main Menu
```

```
def main():
```

```
    if not os.path.exists("secret.key"):
```

```
        generate_key()
```

```
    print("1. Encrypt File")
```

```
    print("2. Decrypt File")
```

```
    choice = input("Enter choice (1/2): ")
```

```
    filename = input("Enter file name: ")
```

```
    if choice == "1":
```

```
        encrypt_file(filename)
```

```
    elif choice == "2":
```

```
        decrypt_file(filename)
```

```
    else:
```

```
        print("❌ Invalid choice")
```

```
if __name__ == "__main__":
```

```
    main()
```

Task3:N-QUEEN Problem

Description:Solve the classic N-Queens problemwhere the goal is to place N queens on

an  $N \times N$  chessboard such that no two queens threaten each other.

Objective:

- Represent the chessboard as a 2D-Array.
- Use backtracking to place queens one by one in safe positions.
- Ensure that no two queens are on a same row, column and diagonal.

code:

```
def print_board(board):
```

```
    for row in board:
```

```
        print(" ".join(row))
```

```
    print("\n")
```

```
def is_safe(board, row, col, n):
```

```
    # Check column
```

```
    for i in range(row):
```

```
        if board[i][col] == 'Q':
```

```
            return False
```

```
    # Check upper-left diagonal
```

```
    i, j = row - 1, col - 1
```

```
    while i >= 0 and j >= 0:
```

```
        if board[i][j] == 'Q':
```

```
            return False
```

```
        i -= 1
```

```
        j -= 1
```

```
    # Check upper-right diagonal
```

```

i, j = row - 1, col + 1
while i >= 0 and j < n:
    if board[i][j] == 'Q':
        return False
    i -= 1
    j += 1
return True

def solve_n_queens(board, row, n):
    if row == n:
        print_board(board)
        return True
    for col in range(n):
        if is_safe(board, row, col, n):
            board[row][col] = 'Q'
            if solve_n_queens(board, row + 1, n):
                return True
            board[row][col] = '.' # Backtrack
    return False

def main():
    n = int(input("Enter value of N: "))
    board = [['.' for _ in range(n)] for _ in range(n)]
    if not solve_n_queens(board, 0, n):
        print("No solution exists")

if __name__ == "__main__":
    main()

```

