**Name** : Menaka rowthu

**E mail** : [menakarowthu3@gmail.com](mailto:menakarowthu3@gmail.com)

TASK 1:

Hangman Game:

**Design a text-based Hangman game. The program**

**selects a random word, and the player guesses one**

**letter at a time to uncover the word. You can set a**

**limit on the number of incorrect guesses allowed.**

**CODE :**

Designing a text-based Hangman game is a fun project! Here’s a breakdown of how to approach it:

Steps to build the Hangman game:

1. Word Selection:
   * Predefine a list of words or fetch from a file.
   * Select a random word from the list.
2. Display the Word:
   * Initially, display the word with underscores representing unguessed letters.
3. Take Player Input:
   * Ask the player to guess one letter at a time.
   * If the guessed letter is in the word, reveal its position(s).
   * If the guessed letter is wrong, track the number of incorrect guesses.
4. Track Progress:
   * Display guessed letters and the current state of the word (with guessed letters and underscores).
   * Keep track of incorrect guesses and limit the player to a set number of wrong guesses (e.g., 6).
5. End Conditions:
   * Player wins if they guess the entire word.
   * Player loses if they exceed the maximum number of incorrect guesses.

**CODE:**

import random

def get\_random\_word():

words = ['python', 'developer', 'hangman', 'challenge', 'coding', 'algorithm']

return random.choice(words)

def display\_word(word, guessed\_letters):

return ''.join([letter if letter in guessed\_letters else '\_' for letter in word])

def hangman():

word = get\_random\_word()

guessed\_letters = set()

incorrect\_guesses = 0

max\_incorrect\_guesses = 6

print("Welcome to Hangman!")

while incorrect\_guesses < max\_incorrect\_guesses:

print(f"\nWord: {display\_word(word, guessed\_letters)}")

print(f"Guessed Letters: {', '.join(guessed\_letters)}")

print(f"Incorrect guesses left: {max\_incorrect\_guesses - incorrect\_guesses}")

guess = input("Guess a letter: ").lower()

if guess in guessed\_letters:

print("You've already guessed that letter.")

elif guess in word:

guessed\_letters.add(guess)

print(f"Good guess! {guess} is in the word.")

else:

incorrect\_guesses += 1

guessed\_letters.add(guess)

print(f"Oops! {guess} is not in the word.")

**# Check if the player has guessed the entire word**

if set(word) == guessed\_letters:

print(f"\nCongratulations! You've guessed the word: {word}")

break

else:

print(f"\nYou've run out of guesses! The word was: {word}")

**# Run the game**

hangman()

TASK 2:

Stock Portfolio Tracker:

**Create a stock portfolio tracking tool that allows users**

**to add, remove, and track the performance of their**

**stock investments. Utilize financial APIs for real-time**

**CODE:**

Creating an interactive Stock Portfolio Tracker is a great project! This can involve fetching real-time stock data from financial APIs and allowing users to manage their portfolio by adding, removing, and tracking the performance of their stocks.

Key Features for Stock Portfolio Tracker:

1. Add/Remove Stocks:
   * Allow users to add stocks to their portfolio by specifying the stock ticker symbol, the number of shares, and the purchase price.
   * Allow users to remove stocks from the portfolio.
2. Fetch Real-Time Stock Data:
   * Use a financial API (e.g., Yahoo Finance, Alpha Vantage, or IEX Cloud) to get the latest stock prices.
3. Track Performance:
   * Calculate the total value of the portfolio based on real-time prices.
   * Track individual stock performance (gain/loss) since purchase.
4. Display Portfolio Summary:
   * Display the current value of each stock.
   * Display total portfolio value and gains/losses.
5. Interactivity:
   * Provide options for users to add, remove stocks, and view their portfolio summary.

Sample Structure:

You can use requests to interact with a stock API and fetch real-time data.

High-Level Design:

1. Stock Portfolio Class:
   * A class to manage the portfolio (adding/removing stocks and tracking performance).
2. Real-Time Stock Data:
   * A function to fetch real-time stock prices using an API.
3. User Interface:
   * Provide a text-based menu for user interaction.

**CODE:**

import requests

class StockPortfolio:

def \_\_init\_\_(self):

self.portfolio = {}

def add\_stock(self, symbol, shares, purchase\_price):

self.portfolio[symbol] = {

'shares': shares,

'purchase\_price': purchase\_price,

'current\_price': 0**, # We'll update this with real-time data**

}

def remove\_stock(self, symbol):

if symbol in self.portfolio:

del self.portfolio[symbol]

else:

print(f"Stock {symbol} not found in portfolio.")

def get\_stock\_data(self, symbol):

**# Example: Using a mock API for now (replace with real API call)**

**# For example, Yahoo Finance or Alpha Vantage**

**# Replace the below with a real API request**

return {

'current\_price': 150.00 # Mock price

}

def update\_portfolio(self):

for symbol in self.portfolio:

data = self.get\_stock\_data(symbol)

self.portfolio[symbol]['current\_price'] = data['current\_price']

def view\_portfolio(self):

total\_value = 0

print("\nYour Portfolio:")

for symbol, details in self.portfolio.items():

stock\_value = details['shares'] \* details['current\_price']

total\_value += stock\_value

gain\_loss = (details['current\_price'] - details['purchase\_price']) \* details['shares']

print(f"{symbol}: {details['shares']} shares @ ${details['current\_price']} (Purchase Price: ${details['purchase\_price']}) | Gain/Loss: ${gain\_loss:.2f}")

print(f"Total Portfolio Value: ${total\_value:.2f}")

def main():

portfolio = StockPortfolio()

while True:

print("\n1. Add Stock")

print("2. Remove Stock")

print("3. View Portfolio")

print("4. Update Prices")

print("5. Exit")

choice = input("Choose an option: ")

if choice == '1':

symbol = input("Enter stock symbol: ").upper()

shares = int(input("Enter number of shares: "))

purchase\_price = float(input("Enter purchase price per share: "))

portfolio.add\_stock(symbol, shares, purchase\_price)

elif choice == '2':

symbol = input("Enter stock symbol to remove: ").upper()

portfolio.remove\_stock(symbol)

elif choice == '3':

portfolio.view\_portfolio()

elif choice == '4':

portfolio.update\_portfolio()

elif choice == '5':

print("Exiting...")

break

else:

print("Invalid choice, please try again.")

**# Run the Stock Portfolio Tracker**

main()

TASK 3:

Basic Chatbot:

**Create a text-based chatbot that can have**

**conversations with users. You can use natural**

**language processing libraries like NLTK or spaCy to**

**make your chatbot more conversational**.

**Explanation:**

**1. Personal Portfolio Website**

This project will showcase your skills, projects, and resume, and can serve as an online hub for potential employers.

Steps to Build:

1. Structure with HTML:
   * Create the basic sections of your portfolio:
     + Home: Introduction, personal bio, and your photo.
     + Skills: Highlight your key technical skills.
     + Projects: Showcase your coding projects with descriptions and links to GitHub or live demos.
     + Resume: Include a link to download your resume.
     + Contact: Add a simple contact form or social media links.
2. Style with CSS:
   * Use CSS for styling your site:
     + Add custom fonts and color schemes that match your personal brand.
     + Use grid and flexbox layouts to create a responsive design that looks good on both desktop and mobile.
3. Add Interactivity with JavaScript:
   * You can add simple animations or interactive features like:
     + A scroll-to-top button.
     + Project filters: Allow users to filter your projects by categories like web development, data science, etc.
     + Form validation: Add JavaScript to validate the contact form input before submission.
4. Optional Enhancements:
   * Hosting: Deploy your site on a hosting platform like GitHub Pages, Netlify, or Heroku.
   * SEO: Add meta tags to improve search engine optimization.
   * Version Control: Manage your code with Git and GitHub to show employers your workflow

CODE:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>My Portfolio</title>

<link rel="stylesheet" href="styles.css">

</head>

<body>

<header>

<h1>Welcome to My Portfolio</h1>

<nav>

<ul>

<li><a href="#home">Home</a></li>

<li><a href="#skills">Skills</a></li>

<li><a href="#projects">Projects</a></li>

<li><a href="#contact">Contact</a></li>

</ul>

</nav>

</header>

<section id="home">

<h2>About Me</h2>

<p>Hello, I’m [Your Name], a software developer with a passion for creating efficient solutions.</p>

</section>

<section id="skills">

<h2>Skills</h2>

<ul>

<li>Python</li>

<li>JavaScript</li>

<li>Django</li>

<li>Data Analysis</li>

</ul>

</section>

<section id="projects">

<h2>Projects</h2>

<div class="project">

<h3>Hangman Game</h3>

<p>A text-based game built using Python.</p>

</div>

<div class="project">

<h3>Stock Portfolio Tracker</h3>

<p>A tool to track stock investments using real-time data.</p>

</div>

</section>

<section id="contact">

<h2>Contact Me</h2>

<form>

<input type="text" placeholder="Your Name" required>

<input type="email" placeholder="Your Email" required>

<textarea placeholder="Your Message" required></textarea>

<button type="submit">Send Message</button>

</form>

</section>

<footer>

<p>© 2024 My Portfolio. All rights reserved.</p>

</footer>

<script src="scripts.js"></script>

</body>

</html>

**2. Basic Chatbot**

Creating a basic text-based chatbot can be a fun way to apply Natural Language Processing (NLP) skills. You can build a rule-based chatbot or incorporate NLP libraries to make it more interactive.

**Steps to Build:**

1. **Basic Structure**:
   * Start with a simple loop that allows the user to input text and get responses.
   * Add predefined responses for common phrases like "hello," "how are you," etc.
2. **Using Natural Language Processing (NLP)**:
   * Incorporate libraries like NLTK or spaCy to make your bot understand and process the user’s input more effectively.
   * Use NLP techniques like **tokenization**, **part-of-speech tagging**, and **entity recognition** to enhance the chatbot’s ability to respond naturally.
3. **Enhancing Responses**:
   * Instead of rule-based responses, create dynamic replies based on keyword matching, sentiment analysis, or other techniques.
4. **Optional Enhancements**:
   * Add a **GUI** using Tkinter to make the chatbot more interactive.
   * Integrate an **API** (like OpenAI's GPT API) to make your chatbot more conversational and intelligent.

CODE:

def chatbot():

print("Chatbot: Hi! How can I help you today?")

while True:

user\_input = input("You: ").lower()

if "hello" in user\_input:

print("Chatbot: Hello! How are you?")

elif "bye" in user\_input:

print("Chatbot: Goodbye! Have a great day!")

break

elif "how are you" in user\_input:

print("Chatbot: I'm just a program, but I'm functioning as expected!")

else:

print("Chatbot: Sorry, I don't understand that.")

**# Run the chatbot**

chatbot()