Python Projects For Everyone

100 Fun & Practial Coding Exercises

Mohamad Charara

Python Projects for Everyone

Mohamad Charara

Published by Mohamad Charara, 2023.

PYTHON PROJECTS FOR EVERYONE

First edition. March 5, 2023.

Written by Mohamad Charara.

In this book, you will find 100 Python projects ranging from beginner to advanced level. These projects are designed to help you improve your coding skills, learn new programming concepts, and have fun along the way. However, it is important to note that these projects are not meant to be a one-size-fits-all solution. Instead, they are meant to be a starting point for your own experimentation and exploration.

Python Projects for Everyone

100 Fun and Practical Coding Exercises

By

Mohamad Charara

About the Author

Mohamad Charara is an electrical engineer who is passionate about technology and programming. He has always been fascinated by the way things work and the endless possibilities that technology provides.

Over the years, Mohamad has gained experience working with various programming languages, but he found his true passion in Python. He was drawn to the language because of its simplicity, versatility, and power, which allows for the creation of complex applications with relative ease.

In addition to programming, Mohamad enjoys exploring new technologies and learning about emerging trends. He believes that staying up-to-date with the latest technologies is essential to staying ahead in today's fast-paced world.

With this book, Mohamad aims to share his knowledge and passion for Python programming with others. He believes that programming is an essential skill that can be learned and enjoyed by anyone, regardless of their background or experience level.

Mohamad hopes that this book will inspire readers to explore the world of Python programming and to discover the endless possibilities that it provides.

TABLE OF CONTENTS

Project 13: Password Generator: Create a program that generates a random password based on certain criteria, such as length and complexity

Project 14: Currency Converter: Develop an app that can convert one currency to another using real-time exchange rates

Project 15: Web Scraper: Create a program that can extract data from websites and store it in a local file or database

Project 16: File Manager: Create a program that can manage files and folders on a user's computer, allowing them to rename, delete, or move files and folders

Project 17: Simple Game: Use Pygame to create a simple game, such as a space shooter or a platformer

Project 18: Loan Payment Calculator: This calculator can be used to calculate the monthly payments, total interest paid, and total amount paid over the course of a loan

Project 19: Dice Rolling Simulator: Build a program that simulates the rolling of a dice, allowing users to specify the number of dice and sides per die

Project 20: Amortization Schedule Generator: This calculator can generate a detailed amortization schedule for a loan, showing the breakdown of each payment into principal and interest, as well as the remaining balance on the loan after each payment

Project 21: Mad Libs Generator: Develop a game that prompts users to input words to fill in the blanks of a story, resulting in a hilarious, unexpected tale

Project 22: Price prediction model: Select an industry or product that interests you, and build a machine learning model that predicts price changes

Project 23: URL Shortener: Design a web application that can take long URLs and shorten them, allowing users to share them easily

Project 24: Snake Game: Develop a classic arcade-style game where the player navigates a snake around a grid and eats food to grow, avoiding obstacles and the snake's own tail

Project 37: Automatic text summarization tool: Develop a program that summarizes long articles or documents using machine learning algorithms

Project 38: Fraud detection tool: Create a program that uses machine learning algorithms to detect fraud in financial transactions or other types of data

Project 39: Recommendation system with explainable AI: Develop a recommendation system that not only provides recommendations but also explains why a particular recommendation was made

Project 40: BMI calculator: Develop a program that calculates Body Mass Index (BMI) based on a user's height and weight

Project 41: Text editor: Create a simple text editor that allows users to create, edit, and save text files

Project 42: Alarm clock: Develop an application that allows users to set alarms and reminders at specific times

Project 43: Random quote generator: Create a program that generates random quotes or phrases from a database of quotes

Project 44: Calendar app: Develop an application that displays a monthly calendar and allows users to add events and reminders

Project 45: Rock-paper-scissors game: Build a simple rock-paper-scissors game that allows two players to play against each other

Project 46: Countdown timer: Create a program that counts down from a specified time and displays a message when the time is up

Project 47: Contact management system: Develop an application that allows users to store and manage contacts with their name, phone number, and email address

Project 48: Morse code translator: Develop a program that can translate text to Morse code and vice versa

Project 49: Memory game: Create a game that challenges users to memorize a sequence of images or numbers and recall them in the correct order

Project 62: Email automation tool: Create a program that can automate email tasks, such as sending personalized messages or scheduling follow-ups

Project 63: Mind mapping tool: Build a program that can create and visualize mind maps, which are diagrams that represent ideas or concepts

Project 64: Email spam filter: Develop a program that can filter out unwanted or spam emails from a user's inbox

Project 65: Barcode scanner: Build a program that can scan and decode barcodes, which can be used for inventory management or product tracking

Project 66: Workout tracker: Develop a program that can track and analyze workout sessions, including exercises, sets, and repetitions

Project 67: Time zone converter: Build a program that can convert time between different time zones, using geographic location data

Project 68: Stock prediction tool: Develop a program that uses machine learning algorithms to predict the future performance of a stock or portfolio

Project 69: Browser extension: Develop a browser extension that can enhance the functionality of popular web browsers, such as Chrome or Firefox

Project 70: Quantum computing simulations: Develop a program that simulates quantum algorithms and circuits, using Python libraries such as Qiskit and Cirq

Project 71: Traffic simulation: Create a program that simulates traffic flow in a city or highway, using algorithms and statistical models

Project 72: Twitter bot: Build a program that can interact with Twitter users, such as retweeting or responding to tweets

Project 73: Maze generator: Build a program that can generate random mazes of different sizes and complexities

Project 74: Data mining tool: Develop a program that can extract and analyze data from large datasets, using techniques such as clustering and association rule mining

Project 75: Network scanner: Build a program that can scan a network for connected devices and open ports, using network protocols such as ICMP and TCP

Project 76: Code editor: Build a code editor that can support different programming languages, and includes features such as syntax highlighting and code completion.

Project 77: Text classification program: Develop a program that can classify text data into different categories, using natural language processing and machine learning techniques

Project 78: Chess game: Create a program that allows players to play chess against each other, and includes features such as move validation and game history

Project 79: Sudoku generator: Develop a program that can generate random Sudoku puzzles of different difficulties

Project 80: Online quiz game: Create a program that can host online quizzes, and allow users to compete with each other and track their scores

Project 81: Movie ticket booking system: Build a program that can allow users to book movie tickets online, and manage seating arrangements and ticket sales

Project 82: Temperature converter: Build a program that can convert temperatures between Celsius, Fahrenheit, and Kelvin scales

Project 83: Pomodoro timer: Create a program that implements the Pomodoro technique, a time management method that uses timed intervals to improve productivity

Project 84: Distributed computing: Create a program that can distribute computing tasks across multiple computers or servers, using Python libraries such as Dask and PySpark

Project 85: Automated testing tool: Develop a program that can automate software testing processes, using frameworks such as Selenium or PyTest

Project 86: Flight ticket booking system: Develop a program that can allow users to book flight tickets online, with features such as flight search, seat selection, and payment processing

Project 87: Video game emulator: Develop a program that can emulate classic video games, using libraries such as Pygame or PyNES

Project 88: News sentiment analysis: Build a program that can analyze news articles and classify them as positive, negative, or neutral, using natural language processing techniques such as sentiment analysis

Project 89: File transfer tool: Develop a program that can transfer files between different devices or servers, using protocols such as FTP or SFTP

Project 90: Encryption and decryption tool: Develop a program that can encrypt and decrypt messages and files, using cryptographic algorithms such as AES or RSA

Project 91: Automated email responder: Create a program that can respond to emails automatically, using natural language processing techniques such as text classification

Project 92: Movie review sentiment analysis: Develop a program that can analyze movie reviews and classify them as positive or negative, using natural language processing techniques such as sentiment analysis

Project 94: Image style transfer: Develop a program that can transfer the style of one image onto another image, using deep learning algorithms such as neural style transfer

Project 95: Time series forecasting: Create a program that can forecast time series data, such as stock prices or weather patterns, using machine learning algorithms such as autoregressive integrated moving average (ARIMA) and long short-term memory (LSTM) networks

Project 96: Stock price prediction: Create a program that can predict stock prices using machine learning algorithms such as neural networks and decision trees

Project 97: Content-based image retrieval: Build a program that can retrieve images from a large database based on their content, using techniques such as feature extraction and similarity matching

Project 98: Multi-agent systems: Build a program that simulates a multi-agent environment, where agents interact with each other and their environment to achieve a goal, using techniques such as reinforcement learning and game theory

Project 99: Deepfake detection: Build a program that can detect manipulated images and videos, such as deepfake videos, using computer vision techniques such as face detection and analysis

Project 100: Reinforcement learning for robotics: Build a program that uses reinforcement learning techniques to train a robot to perform a task, such as navigating a maze or playing a game

Before We Start

Python has become one of the most important programming languages in recent years. Its popularity is due in part to its simplicity and ease of use, making it an ideal language for beginners to learn. At the same time, its versatility and power have made it a favorite among experienced developers for building complex applications and systems. Python is used extensively in a wide range of industries, including finance, healthcare, gaming, and data science, just to name a few.

In this book, you will find 100 Python projects ranging from beginner to advanced level. These projects are designed to help you improve your coding skills, learn new programming concepts, and have fun along the way. However, it is important to note that these projects are not meant to be a one-size-fits-all solution. Instead, they are meant to be a starting point for your own experimentation and exploration.

In some cases, the code provided may be more like a guidance than a complete code. This is intentional, as the goal is to give you the idea and provide an example of how to implement it in Python. In other cases, the code may be more complex. However, even in these cases, the code should be seen as a starting point for your own exploration.

We hope that you will enjoy working through these projects and that they will help you improve your coding skills and build your confidence as a programmer. Remember to have fun and don't be afraid to experiment and try new things!

PROJECT 1: SIMPLE SOCIAL MEDIA APPLICARION THAT ALLOWS USERS TO CREATE PROFILES, CONNECT WITH OTHER USERS, AND SHARE POSTS

- **Step 1: Create a database to store user information and posts:**

```
import sqlite3

conn = sqlite3.connect('social_media.db')

cursor = conn.cursor()

cursor.execute('''

CREATE TABLE users (

id INTEGER PRIMARY KEY AUTOINCREMENT,

username TEXT,

password TEXT,

name TEXT

)

''')

cursor.execute('''
```

CREATE TABLE posts (

id INTEGER PRIMARY KEY AUTOINCREMENT,

user_id INTEGER,

content TEXT,

timestamp DATETIME DEFAULT CURRENT_TIMESTAMP

)

''')

conn.commit()

conn.close()

- **Step 2: Implement user registration and login functionality:**

import hashlib

def register_user(username, password, name):

hashed_password = hashlib.sha256(password.encode()).hexdigest()

conn = sqlite3.connect('social_media.db')

cursor = conn.cursor()

cursor.execute('''

INSERT INTO users (username, password, name)

VALUES (?, ?, ?)

''', (username, hashed_password, name))

conn.commit()

conn.close()

def login_user(username, password):

hashed_password = hashlib.sha256(password.encode()).hexdigest()

conn = sqlite3.connect('social_media.db')

cursor = conn.cursor()

cursor.execute('''

SELECT * FROM users

WHERE username = ? AND password = ?

''', (username, hashed_password))

user = cursor.fetchone()

conn.close()

if user:

return user

else:

return None

Step 3: Implement functionality to create and retrieve posts:

```
def create_post(user_id, content):

conn = sqlite3.connect('social_media.db')

cursor = conn.cursor()

cursor.execute('''

INSERT INTO posts (user_id, content)

VALUES (?, ?)

''', (user_id, content))

conn.commit()

conn.close()

def get_posts():

conn = sqlite3.connect('social_media.db')

cursor = conn.cursor()

cursor.execute('''

SELECT users.name, posts.content, posts.timestamp

FROM posts

JOIN users ON posts.user_id = users.id

ORDER BY posts.timestamp DESC

''')

posts = cursor.fetchall()
```

conn.close()

return posts

Step 4: Add error handling feature:

Here, we catch ‘sqlite3.Error’ exceptions and print a helpful message to the user indicating that an error occurred. In a real-world application, you might want to do more than just print a message - for example, you could log the error to a file, send an email to the administrator, or display a more user-friendly error message to the user.

import hashlib

import sqlite3

def register_user(username, password, name):

try:

hashed_password = hashlib.sha256(password.encode()).hexdigest()

conn = sqlite3.connect('social_media.db')

cursor = conn.cursor()

cursor.execute('''

INSERT INTO users (username, password, name)

VALUES (?, ?, ?)

''', (username, hashed_password, name))

conn.commit()

except sqlite3.Error as e:

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

finally:

conn.close()

def login_user(username, password):

try:

hashed_password = hashlib.sha256(password.encode()).hexdigest()

conn = sqlite3.connect('social_media.db')

cursor = conn.cursor()

cursor.execute('''

SELECT * FROM users

WHERE username = ? AND password = ?

''', (username, hashed_password))

user = cursor.fetchone()

except sqlite3.Error as e:

print(f'An error occurred: {e}')

finally:

conn.close()

if user:

return user
```

```
else:

return None

def create_post(user_id, content):

try:

conn = sqlite3.connect('social_media.db')

cursor = conn.cursor()

cursor.execute('''

INSERT INTO posts (user_id, content)

VALUES (?, ?)

''', (user_id, content))

conn.commit()

except sqlite3.Error as e:

print(f'An error occurred: {e}')

finally:

conn.close()

def get_posts():

try:

conn = sqlite3.connect('social_media.db')

cursor = conn.cursor()
```

```
cursor.execute('''

SELECT users.name, posts.content, posts.timestamp

FROM posts

JOIN users ON posts.user_id = users.id

ORDER BY posts.timestamp DESC

''')

posts = cursor.fetchall()

except sqlite3.Error as e:

print(f'An error occurred: {e}')

finally:

conn.close()

return posts
```

Note: The ‘try’ block contains the code that may raise an exception, and the ‘except’ block contains the code that will be executed if an exception is raised. The ‘finally’ block contains code that will always be executed, regardless of whether an exception was raised or not.

PROJECT 2: WEATHER APPLICATION THAT RETRIEVES DATA FROM AN API AND DISPLAYS CURRENT WEATHER CONDITIONS AND FORECAST FOR A GIVEN LOCATION

THE FOLLOWING CODE uses the ‘requests’ library to make an API call to OpenWeatherMap's API to retrieve the current weather data for a given location. The API returns a JSON object with the current weather conditions, which the code then parses and prints to the console.

Note that you'll need to sign up for an API key from OpenWeatherMap in order to use their API. Replace ‘your_api_key_here’ in the code with your own API key.

```
import requests

def get_weather(location):

# API Key for OpenWeatherMap

API_KEY = "your_api_key_here"

# Call the API using the location and API key

weather_url = f"http://api.openweathermap.org/data/2.5/weather?q=
{location}&appid={API_KEY}"

weather_data = requests.get(weather_url).json()

# Extract the relevant data from the API response

current_temperature = weather_data["main"]["temp"]
```

```
current_description = weather_data["weather"][0]["description"]

current_humidity = weather_data["main"]["humidity"]

current_pressure = weather_data["main"]["pressure"]

current_wind_speed = weather_data["wind"]["speed"]

# Print the current weather conditions

print(f"Temperature: {current_temperature}")

print(f"Description: {current_description}")

print(f"Humidity: {current_humidity}")

print(f"Pressure: {current_pressure}")

print(f"Wind Speed: {current_wind_speed}")

# Example usage

location = "London"

get_weather(location)
```

PROJECT 3: E-COMMERCE WEBSITE: AN E-COMMERCE WEBSITE THAT ALLOWS USERS TO BROWSE PRODUCTS, ADD ITEMS TO A CART, AND COMPLETE PURCHASES.

START WITH A BASIC outline for the project:

1. Set up a web framework: You can use a popular Python web framework such as Django or Flask to build your e-commerce website.

1. Define the product model: Create a database model that will store information about your products, such as their name, description, price, and image.

1. Create product views: Write views that will allow users to browse your products and view detailed information about each one.

1. Implement a shopping cart: Add the ability for users to add items to a shopping cart and view the contents of their cart.

1. Implement the checkout process: Allow users to complete purchases by entering their shipping and payment information and processing the payment.

1. Secure the application: Implement security measures such as encryption and secure authentication to protect sensitive information like payment details.

1. Deploy the application: Deploy your e-commerce website to a hosting provider or cloud service so that it can be accessed by users over the internet.

This is just a basic outline, and you will likely need to add more features and functionality to meet the needs of your specific e-commerce website. However, with this foundation in place, you will have a solid starting point for building a functional e-commerce platform.

Here are just examples of what the code for your e-commerce website might look like. To build a full-featured e-commerce platform, you would need to implement many more views, templates, and models, as well as add security measures, handle payments, and handle other important tasks. But hopefully, the following code blocks give you a better idea of what the code for an e-commerce website in Django might look like.

- **Step 1: Product model (Django web framework)**

```
from django.db import models

class Product(models.Model):

name = models.CharField(max_length=255)

description = models.TextField()

price = models.DecimalField(max_digits=10, decimal_places=2)

image = models.ImageField(upload_to='products/')

def __str__(self):

return self.name
```

- **Step 2: displays a list of products**

```
from django.shortcuts import render

from .models import Product
```

```
def product_list(request):

products = Product.objects.all()

return render(request, 'products/product_list.html', {'products': products})
```

- **Step 3: Product list view (CSS code)**

```
{% extends "base.html" %}

{% block content %}

<h1>Product List</h1>

<ul>

{% for product in products %}

<li>

<h2>{{ product.name }}</h2>

<p>{{ product.description }}</p>

<p>{{ product.price }}</p>

<img src="{{ product.image.url }}">

</li>

{% endfor %}

</ul>

{% endblock %}
```

- **Step 4: Implement a shopping cart in Django**

```
from django.db import models

from django.contrib.sessions.models import Session

class Cart(models.Model):

session = models.ForeignKey(Session, on_delete=models.CASCADE)

product = models.ForeignKey(Product, on_delete=models.CASCADE)

quantity = models.PositiveIntegerField(default=1)

created_at = models.DateTimeField(auto_now_add=True)

updated_at = models.DateTimeField(auto_now=True)

def __str__(self):

return f"{self.quantity} of {self.product.name}"
```

- **Step 5: Create a view that adds items to the cart:**

```
from django.shortcuts import get_object_or_404, redirect

from .models import Cart, Product

def add_to_cart(request, product_id):

product = get_object_or_404(Product, pk=product_id)

cart, created = Cart.objects.get_or_create(

session=request.session.session_key,
```

```
product=product,

)

if not created:

cart.quantity += 1

cart.save()

return redirect('cart')
```

- **Step 6: Display the contents of the cart**

```
from django.shortcuts import render


def view_cart(request):

cart_items = Cart.objects.filter(session=request.session.session_key)

total = sum(item.quantity * item.product.price for item in cart_items)

return render(request, 'cart.html', {'cart_items': cart_items, 'total': total})
```

- **Step 7: Javascript template that displays the contents of the cart**

```
{% extends "base.html" %}

{% block content %}

<h1>Shopping Cart</h1>

<table>

<thead>
```

```
<tr>

<th>Product</th>

<th>Price</th>

<th>Quantity</th>

<th>Subtotal</th>

</tr>

</thead>

<tbody>

{% for item in cart_items %}

<tr>

<td>{{ item.product.name }}</td>

<td>{{ item.product.price }}</td>

<td>{{ item.quantity }}</td>

<td>{{ item.product.price * item.quantity }}</td>

</tr>

{% endfor %}

</tbody>

<tfoot>

<tr>
```

```
<td colspan="3">Total:</td>

<td>{{ total }}</td>

</tr>

</tfoot>

</table>

{% endblock %}
```

PROJECT 4: CHATBOT: BUILD A CHATBOT THAT CAN ANSWER COMMON QUESTIONS AND ENGAGE IN BASIC CONVERSATION WITH USERS.

THIS IS A VERY BASIC chatbot that just responds to a few specific questions and can engage in simple conversation. You can customize the responses and add more complexity as you see fit.

import random

Define some responses for the chatbot

greetings = ["hello", "hi", "hey", "what's up"]

goodbyes = ["bye", "goodbye", "see you later", "have a nice day"]

questions = {

"what is your name?": "My name is Chatbot!",

"how are you?": "I'm doing well, thanks for asking!",

"what do you like to do?": "I like to chat with people!",

"where are you from?": "I was born and raised in the cloud.",

"what is the meaning of life?": "That's a deep question. What do you think?"

}

Define a function to handle user input and generate responses

```
def chatbot():

# Greet the user

print("Hello! I'm a chatbot. What can I help you with today?")

# Start a loop to keep the chatbot running

while True:

# Get user input

user_input = input("> ").lower()

# Check if the user wants to end the conversation

if user_input in goodbyes:

print("Goodbye!")

break

# Check if the user has a question for the chatbot

elif user_input in questions:

print(questions[user_input])

# If the user doesn't have a question, just chat with them

else:

print(random.choice(greetings))

# Call the chatbot function to start the conversation

chatbot()
```

PROJECT 5: TEXT-BASED ADVENTURE GAME: BUILD A TEXT-BASED ADVENTURE GAME WHERE THE PLAYER CAN EXPLORE DIFFERENT ENVIRONMENTS AND MAKE CHOICES THAT AFFECT THE OUTCOME OF THE GAME

IMPORT TIME

Define some global variables for the game

game_over = False

player_name = ""

player_health = 100

player_inventory = []

Define a function to display the game introduction and get the player's name

def game_intro():

print("Welcome to the Adventure Game!")

print("In this game, you will explore different environments and make choices that affect the outcome.")

print("Let's start by getting your name.")

while True:

```
name = input("What is your name? ")

if name:

global player_name

player_name = name

break

else:

print("Please enter a valid name.")

# Define a function to display the game over screen and end the game

def game_over_screen():

print("Game over.")

print("Thanks for playing!")

global game_over

game_over = True

# Define a function to display the player's status

def player_status():

print(f"Player: {player_name}")

print(f"Health: {player_health}")

print(f"Inventory: {player_inventory}")

# Define a function to simulate a fight with an enemy
```

```
def fight(enemy_name, enemy_health):

print(f"A wild {enemy_name} appears!")

while enemy_health > 0:

print(f"{enemy_name} health: {enemy_health}")

player_attack = input("What will you do? (attack/run) ")

if player_attack == "attack":

damage = random.randint(10, 20)

enemy_health -= damage

print(f"You attack {enemy_name} for {damage} damage.")

time.sleep(1)

if enemy_health <= 0:

print(f"You defeated the {enemy_name}!")

return True

enemy_attack = random.randint(5, 15)

global player_health

player_health -= enemy_attack

print(f"{enemy_name} attacks you for {enemy_attack} damage.")

time.sleep(1)

if player_health <= 0:
```

```
print(f"You were defeated by the {enemy_name}.")

game_over_screen()

return False

elif player_attack == "run":

print("You run away from the battle.")

return False

else:

print("Please enter a valid action.")

# Define a function to simulate a room in the game

def room(room_name):

print(f"You are in the {room_name}.")

while True:

action = input("What do you want to do? (explore/leave) ")

if action == "explore":

print(f"You search the {room_name}.")

time.sleep(1)

if room_name == "forest":

print("You find a healing potion!")

player_inventory.append("healing potion")
```

```
elif room_name == "cave":

if "torch" not in player_inventory:

print("It's too dark to see anything. You need a torch.")

else:

print("You find a treasure chest!")

player_inventory.append("treasure chest")

elif room_name == "castle":

if "key" not in player_inventory:

print("The door is locked. You need a key.")

else:

print("You unlock the door and enter the castle!")

time.sleep(1)

if fight("dragon", 50):

print("You find the princess and rescue her!")

print("You win!")

game_over_screen()

return

else:

print("There's nothing to find here.")
```

```
elif action == "leave":

print(f"You leave the {room_name}.")

return

else:

print("Please enter a valid action.")

Define the game loop

def game_loop():

while not game_over:

player_status()

room("forest")

if game_over:

return

player_status()

room("cave")

if game_over:

return

player_status()

room("castle")

if game_over:
```

return

Call the game_intro function to start the game

game_intro()

Call the game_loop function to run the game

game_loop()

This is a very basic text-based adventure game that simulates exploring different environments and making choices that affect the outcome. You can customize it and add more complexity as you see fit.

PROJECT 6: BUDGET TRACKER: BUILD A BUDGET TRACKER THAT ALLOWS USERS TO KEEP TRACK OF THEIR INCOME AND EXPENSES AND SEE THEIR FINANCIAL STATUS

THIS CODE DEFINES A BudgetTracker class that allows users to keep track of their income and expenses and see their financial status. You can create a new BudgetTracker object with an initial balance, and then use the add_income and add_expense methods to add transactions to the tracker. The get_balance, get_income, get_expenses, and get_expense_total methods allow you to retrieve various financial data, and the get_status method displays a summary of the current financial status. You can customize the class and add more functionality as you see fit.

```
class BudgetTracker:

def __init__(self, initial_balance):

self.balance = initial_balance

self.income = 0

self.expenses = []

def add_income(self, amount):

self.balance += amount

self.income += amount

def add_expense(self, description, amount):
```

```
self.balance -= amount

self.expenses.append((description, amount))

def get_balance(self):

return self.balance

def get_income(self):

return self.income

def get_expenses(self):

return self.expenses

def get_expense_total(self):

return sum([expense[1] for expense in self.expenses])

def get_status(self):

print(f"Current balance: ${self.get_balance()}")

print(f"Total income: ${self.get_income()}")

print(f"Total expenses: ${self.get_expense_total()}")

print("Expense list:")

for expense in self.get_expenses():

print(f"{expense[0]} - ${expense[1]}")

# Create a new budget tracker with an initial balance of $1000

tracker = BudgetTracker(1000)
```

```python
# Add some income and expenses

tracker.add_income(500)

tracker.add_income(200)

tracker.add_expense("Rent", 800)

tracker.add_expense("Groceries", 100)

tracker.add_expense("Gas", 50)

# Get the current financial status

tracker.get_status()
```

PROJECT 7: PERSONAL ORGANIZER: BUILD A PERSONAL ORGANIZER THAT ALLOWS USERS TO KEEP TRACK OF THEIR SCHEDULE, TASKS, AND CONTACTS

```
CLASS TASK:

def __init__(self, description, due_date):

self.description = description

self.due_date = due_date

self.completed = False

def mark_completed(self):

self.completed = True

class Contact:

def __init__(self, name, phone_number, email):

self.name = name

self.phone_number = phone_number

self.email = email

class PersonalOrganizer:

def __init__(self):
```

self.tasks = []

self.contacts = {}

def add_task(self, task):

self.tasks.append(task)

def complete_task(self, task_index):

self.tasks[task_index].mark_completed()

def add_contact(self, contact):

self.contacts[contact.name] = contact

def remove_contact(self, contact_name):

del self.contacts[contact_name]

def view_tasks(self):

for i, task in enumerate(self.tasks):

print(f"{i+1}. {task.description} (due {task.due_date}) - {'completed' if task.completed else 'incomplete'}")

def view_contacts(self):

for name, contact in self.contacts.items():

print(f"{name} - Phone: {contact.phone_number}, Email: {contact.email}")

Here's an example usage of the PersonalOrganizer class:

organizer = PersonalOrganizer()

```
task1 = Task("Buy groceries", "2023-02-20")

task2 = Task("Finish project", "2023-03-01")

organizer.add_task(task1)

organizer.add_task(task2)

contact1 = Contact("Alice", "555-1234", "alice@example.com")

contact2 = Contact("Bob", "555-5678", "bob@example.com")

organizer.add_contact(contact1)

organizer.add_contact(contact2)

organizer.view_tasks()

# Output:

# 1. Buy groceries (due 2023-02-20) - incomplete

# 2. Finish project (due 2023-03-01) - incomplete

organizer.complete_task(0)

organizer.view_tasks()

# Output:

# 1. Buy groceries (due 2023-02-20) - completed

# 2. Finish project (due 2023-03-01) - incomplete

organizer.view_contacts()

# Output:
```

```
# Alice - Phone: 555-1234, Email: alice@example.com

# Bob - Phone: 555-5678, Email: bob@example.com

organizer.remove_contact("Bob")

organizer.view_contacts()

# Output:

# Alice - Phone: 555-1234, Email: alice@example.com
```

PROJECT 8: RECIPE BOOK: BUILD A RECIPE BOOK THAT ALLOWS USERS TO SEARCH FOR AND SAVE RECIPES, CREATE GROCERY LISTS, AND GET RECOMMENDATIONS FOR MEALS

```
# RECIPE BOOK

import random

class Recipe:

def __init__(self, name, ingredients, instructions):

self.name = name

self.ingredients = ingredients

self.instructions = instructions

def __str__(self):

return self.name

def __repr__(self):

return self.name

class RecipeBook:

def __init__(self):

self.recipes = []
```

self.grocery_list = []

def add_recipe(self, recipe):

self.recipes.append(recipe)

def remove_recipe(self, recipe):

self.recipes.remove(recipe)

def search_recipe(self, query):

return [recipe for recipe in self.recipes if query.lower() in recipe.name.lower()]

def add_to_grocery_list(self, ingredients):

for ingredient in ingredients:

if ingredient not in self.grocery_list:

self.grocery_list.append(ingredient)

def remove_from_grocery_list(self, ingredient):

if ingredient in self.grocery_list:

self.grocery_list.remove(ingredient)

def get_random_recipe(self):

return random.choice(self.recipes)

Example usage

recipe_book = RecipeBook()

Add some recipes

recipe1 = Recipe("Spaghetti Carbonara", ["spaghetti", "eggs", "bacon", "parmesan cheese", "garlic", "olive oil"], "1. Cook spaghetti until al dente. 2. Cook bacon in a large skillet until crispy. 3. In a bowl, whisk together eggs and parmesan cheese. 4. Add garlic to the bacon and cook for 1 minute. 5. Add spaghetti to the skillet and toss with bacon and garlic. 6. Pour the egg mixture over the spaghetti and toss until the eggs are cooked. Serve hot.")

recipe2 = Recipe("Chicken Parmesan", ["chicken breast", "breadcrumbs", "parmesan cheese", "eggs", "marinara sauce", "mozzarella cheese"], "1. Preheat oven to 400°F. 2. Coat chicken breast in beaten eggs, then coat in breadcrumbs mixed with parmesan cheese. 3. Place chicken in a baking dish and bake for 20-25 minutes. 4. Spoon marinara sauce over chicken and top with mozzarella cheese. 5. Bake for an additional 10-15 minutes. Serve hot.")

recipe_book.add_recipe(recipe1)

recipe_book.add_recipe(recipe2)

Search for a recipe

query = "spaghetti"

search_results = recipe_book.search_recipe(query)

print("Search results for '{}':".format(query))

for recipe in search_results:

print(recipe)

Add ingredients to grocery list

```
recipe = recipe_book.get_random_recipe()

print("Adding ingredients for '{}' to grocery list...".format(recipe))

recipe_book.add_to_grocery_list(recipe.ingredients)

print("Grocery list:", recipe_book.grocery_list)

# Remove ingredient from grocery list

ingredient = recipe.ingredients[0]

print("Removing '{}' from grocery list...".format(ingredient))

recipe_book.remove_from_grocery_list(ingredient)

print("Grocery list:", recipe_book.grocery_list)
```

This is just an example, and you can modify the code to fit your needs or add more features, such as user authentication or a GUI interface.

PROJECT 9: STOCK MARKET SIMULATOR: BUILD A STOCK MARKET SIMULATOR THAT ALLOWS USERS TO BUY AND SELL STOCKS AND TRACK THEIR INVESTMENTS OVER TIME

THIS CODE DEFINES TWO classes, Stock and Portfolio, to represent a stock and a portfolio of stocks, respectively. It also defines a stocks dictionary to store the available stocks.

The Portfolio class has methods to buy and sell stocks, as well as to show the portfolio's current contents. The buy method checks if there is enough cash to make the purchase, adds the bought stock to the portfolio, and deducts the cost from the available cash.

```
class Stock:

def __init__(self, name, price):

self.name = name

self.price = price

def update_price(self, new_price):

self.price = new_price

class Portfolio:

def __init__(self, cash):

self.cash = cash
```

```
self.stocks = {}

def buy(self, stock, quantity):

cost = stock.price * quantity

if cost > self.cash:

print("Insufficient funds.")

else:

if stock.name in self.stocks:

self.stocks[stock.name] += quantity

else:

self.stocks[stock.name] = quantity

self.cash -= cost

print("Bought {} shares of {} for ${:.2f}.".format(quantity, stock.name,
cost))

def sell(self, stock, quantity):

if stock.name not in self.stocks:

print("You do not own any shares of {}.".format(stock.name))

elif quantity > self.stocks[stock.name]:

print("You do not have enough shares of {}.".format(stock.name))

else:
```

```
cost = stock.price * quantity

self.stocks[stock.name] -= quantity

self.cash += cost

print("Sold {} shares of {} for ${:.2f}.".format(quantity, stock.name, cost))

def show_portfolio(self):

print("Cash: ${:.2f}".format(self.cash))

print("Stocks:")

for name, quantity in self.stocks.items():

stock_value = quantity * self.get_stock_price(name)

print("- {} shares of {} (current price: ${:.2f}, value:
${:.2f})".format(quantity, name, self.get_stock_price(name), stock_value))

total_value = self.cash + sum([quantity * self.get_stock_price(name) for
name, quantity in self.stocks.items()])

print("Total value: ${:.2f}".format(total_value))

def get_stock_price(self, name):

return stocks[name].price

stocks = {

"AAPL": Stock("AAPL", 135.0),

"GOOG": Stock("GOOG", 2400.0),

"TSLA": Stock("TSLA", 850.0)
```

```
}
portfolio = Portfolio(10000.0)
while True:
action = input("Buy or sell? ")
if action == "buy":
stock_name = input("Enter stock name: ")
quantity = int(input("Enter quantity: "))
stock = stocks.get(stock_name)
if stock is None:
print("Invalid stock name.")
else:
portfolio.buy(stock, quantity)
elif action == "sell":
stock_name = input("Enter stock name: ")
quantity = int(input("Enter quantity: "))
stock = stocks.get(stock_name)
if stock is None:
print("Invalid stock name.")
else:
```

```
portfolio.sell(stock, quantity)

elif action == "show":

portfolio.show_portfolio()

else:

print("Invalid action.")
```

PROJECT 10: TO-DO LIST APPLICATION: CREATE AN APP THAT ALLOWS THE USER TO ADD TASKS TO A LIST AND CHECK THEM OFF WHEN THEY ARE COMPLETED

THIS CODE ALLOWS THE user to add tasks, remove tasks, display the current list of tasks, and mark tasks as complete. It also includes a menu for the user to choose from. You can customize this code as per your requirements.

define an empty list to store the tasks

tasks = []

define a function to add a task to the list

def add_task():

task = input("Enter a new task: ")

tasks.append(task)

print("Task added successfully!")

define a function to remove a task from the list

def remove_task():

task = input("Enter the task to remove: ")

if task in tasks:

tasks.remove(task)

```
print("Task removed successfully!")

else:

print("Task not found in the list.")

# define a function to display the current list of tasks

def show_tasks():

if tasks:

print("List of tasks:")

for i, task in enumerate(tasks):

print(f"{i+1}. {task}")

else:

print("No tasks found.")

# define a function to check off a completed task

def complete_task():

task = input("Enter the task to mark as complete: ")

if task in tasks:

tasks.remove(task)

print("Task completed and removed from the list.")

else:

print("Task not found in the list.")
```

```
# define a function to display the menu options
def show_menu():
print("Menu:")
print("1. Add a task")
print("2. Remove a task")
print("3. Show tasks")
print("4. Mark a task as complete")
print("5. Exit")
# define the main function to run the application
def main():
while True:
show_menu()
choice = input("Enter your choice: ")
if choice == "1":
add_task()
elif choice == "2":
remove_task()
elif choice == "3":
show_tasks()
```

```
elif choice == "4":

complete_task()

elif choice == "5":

print("Goodbye!")

break

else:

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

if __name__ == "__main__":

main()
```

PROJECT 11: CALCULATOR: DEVELOP A SIMPLE CALCULATOR THAT CAN PERFORM BASIC ARITHMETIC OPERATIONS LIKE ADDITION, SUBTRACTION, MULTIPLICATION, AND DIVISION

THIS CODE ALLOWS THE user to perform basic arithmetic operations like addition, subtraction, multiplication, and division. It also includes a menu for the user to choose from. You can customize this code as per your requirements.

```
# define a function for addition

def add(num1, num2):

return num1 + num2

# define a function for subtraction

def subtract(num1, num2):

return num1 - num2

# define a function for multiplication

def multiply(num1, num2):

return num1 * num2

# define a function for division

def divide(num1, num2):
```

```
if num2 == 0:

return "Cannot divide by zero"

else:

return num1 / num2

# define a function to display the menu options

def show_menu():

print("Menu:")

print("1. Addition")

print("2. Subtraction")

print("3. Multiplication")

print("4. Division")

print("5. Exit")

# define the main function to run the calculator

def main():

while True:

show_menu()

choice = input("Enter your choice: ")

if choice in ("1", "2", "3", "4"):

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

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

if choice == "1":

print(num1, "+", num2, "=", add(num1, num2))

elif choice == "2":

print(num1, "-", num2, "=", subtract(num1, num2))

elif choice == "3":

print(num1, "*", num2, "=", multiply(num1, num2))

elif choice == "4":

print(num1, "/", num2, "=", divide(num1, num2))

elif choice == "5":

print("Goodbye!")

break

else:

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

if __name__ == "__main__":

main()
```

PROJECT 12: GUESSING GAME: DEVELOP A GAME THAT GENERATES A RANDOM NUMBER AND ALLOWS THE USER TO GUESS THE NUMBER UNTIL THEY GET IT RIGHT

THIS CODE GENERATES a random number between 1 and 100 and allows the user to guess the number until they get it right. It keeps track of the number of guesses and provides feedback to the user if their guess is too low or too high. Once the user guesses the number correctly, it prints the number of guesses it took to get the right answer. You can customize this code as per your requirements.

```
import random

# define a function to generate a random number

def generate_number():

return random.randint(1, 100)

# define the main function to run the game

def main():

number = generate_number()

guess = None

num_guesses = 0

print("Welcome to the guessing game!")
```

```
while guess != number:

guess = int(input("Guess the number between 1 and 100: "))

num_guesses += 1

if guess < number:

print("Too low. Try again.")

elif guess > number:

print("Too high. Try again.")

print(f"Congratulations! You guessed the number in {num_guesses} tries.")

if __name__ == "__main__":

main()
```

PROJECT 13: PASSWORD GENERATOR: CREATE A PROGRAM THAT GENERATES A RANDOM PASSWORD BASED ON CERTAIN CRITERIA, SUCH AS LENGTH AND COMPLEXITY

THIS CODE GENERATES a random password based on the length and complexity entered by the user. The complexity can be set to "low", "medium", or "high" depending on how complex the password needs to be. The password is generated using the `string` and `random` modules in Python. You can customize this code as per your requirements.

```
import random

import string

# define the function to generate a random password

def generate_password(length, complexity):

# define the characters to be used in the password

if complexity == "low":

chars = string.ascii_lowercase

elif complexity == "medium":

chars = string.ascii_letters + string.digits

elif complexity == "high":

chars = string.ascii_letters + string.digits + string.punctuation
```

generate the password

password = ''.join(random.choice(chars) for i in range(length))

return password

define the main function to run the password generator

def main():

length = int(input("Enter the length of the password: "))

complexity = input("Enter the complexity of the password (low/medium/high): ")

password = generate_password(length, complexity)

print("Your password is:", password)

if __name__ == "__main__":

main()

PROJECT 14: CURRENCY CONVERTER: DEVELOP AN APP THAT CAN CONVERT ONE CURRENCY TO ANOTHER USING REAL-TIME EXCHANGE RATES

THIS CODE USES THE requests module in Python to make an API call to get the real-time exchange rates. It then uses the exchange rate to convert the amount entered by the user from the base currency to the target currency. You can customize this code to add more features, such as displaying a list of available currencies or handling errors.

```
import requests

# define the function to get the exchange rate

def get_exchange_rate(base_currency, target_currency):

url = f"https://api.exchangerate-api.com/v4/latest/{base_currency}"

response = requests.get(url)

if response.status_code == 200:

data = response.json()

exchange_rate = data["rates"][target_currency]

return exchange_rate

else:

return None

# define the function to convert the currency
```

```
def convert_currency(amount, exchange_rate):

return amount * exchange_rate

# define the main function to run the currency converter

def main():

base_currency = input("Enter the base currency: ")

target_currency = input("Enter the target currency: ")

amount = float(input("Enter the amount to convert: "))

exchange_rate = get_exchange_rate(base_currency, target_currency)

if exchange_rate is not None:

converted_amount = convert_currency(amount, exchange_rate)

print(f"{amount} {base_currency} is equal to {converted_amount}
{target_currency}")

else:

print("Unable to get the exchange rate. Please try again later.")

if __name__ == "__main__":

main()
```

PROJECT 15: WEB SCRAPER: CREATE A PROGRAM THAT CAN EXTRACT DATA FROM WEBSITES AND STORE IT IN A LOCAL FILE OR DATABASE

THIS CODE USES THE requests module in Python to make an HTTP request to the website and the BeautifulSoup module to extract the data from the website. It then writes the extracted data to a file named "data.txt". You can customize this code to extract different types of data or store the data in a database instead of a file.

```
import requests

from bs4 import BeautifulSoup

# define the function to scrape the website

def scrape_website(url):

response = requests.get(url)

if response.status_code == 200:

soup = BeautifulSoup(response.text, "html.parser")

# define the tags to be extracted from the website

tags = soup.find_all("h2")

# write the extracted data to a file

with open("data.txt", "w") as f:

for tag in tags:
```

```
f.write(tag.text + "\n")

print("Data has been scraped and stored in data.txt.")

else:

print("Unable to connect to the website. Please try again later.")

# define the main function to run the web scraper

def main():

url = input("Enter the URL of the website to be scraped: ")

scrape_website(url)

if __name__ == "__main__":

main()
```

PROJECT 16: FILE MANAGER: CREATE A PROGRAM THAT CAN MANAGE FILES AND FOLDERS ON A USER'S COMPUTER, ALLOWING THEM TO RENAME, DELETE, OR MOVE FILES AND FOLDERS

THIS CODE USES THE `os` module in Python to rename, delete, or move files and folders on a user's computer. The user is presented with a menu of options to choose from. You can customize this code to add more features, such as creating a new folder or copying a file to a different location.

```
import os

# define the function to rename a file or folder

def rename_file(old_name, new_name):

os.rename(old_name, new_name)

print(f"{old_name} has been renamed to {new_name}.")

# define the function to delete a file or folder

def delete_file(name):

os.remove(name)

print(f"{name} has been deleted.")

# define the function to move a file or folder

def move_file(source, destination):
```

```
os.rename(source, destination)

print(f"{source} has been moved to {destination}.")

# define the main function to run the file manager

def main():

while True:

print("1. Rename file")

print("2. Delete file")

print("3. Move file")

print("4. Quit")

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

if choice == 1:

old_name = input("Enter the old name of the file/folder: ")

new_name = input("Enter the new name of the file/folder: ")

rename_file(old_name, new_name)

elif choice == 2:

name = input("Enter the name of the file/folder to be deleted: ")

delete_file(name)

elif choice == 3:

source = input("Enter the source path of the file/folder: ")
```

```
destination = input("Enter the destination path of the file/folder: ")

move_file(source, destination)

elif choice == 4:

break

else:

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

if __name__ == "__main__":

main()
```

PROJECT 17: SIMPLE GAME: USE PYGAME TO CREATE A SIMPLE GAME, SUCH AS A SPACE SHOOTER OR A PLATFORMER

IMPORT PYGAME

import random

initialize Pygame

pygame.init()

set the screen dimensions

screen_width = 640

screen_height = 480

set the screen title

pygame.display.set_caption("Space Shooter")

create the screen

screen = pygame.display.set_mode((screen_width, screen_height))

define the colors

white = (255, 255, 255)

black = (0, 0, 0)

set the clock

```
clock = pygame.time.Clock()

# load the player image

player_img = pygame.image.load("player.png")

# define the player class

class Player(pygame.sprite.Sprite):

def __init__(self):

super().__init__()

self.image = player_img

self.rect = self.image.get_rect()

self.rect.centerx = screen_width // 2

self.rect.bottom = screen_height - 10

self.speed_x = 0

def update(self):

self.speed_x = 0

keystate = pygame.key.get_pressed()

if keystate[pygame.K_LEFT]:

self.speed_x = -5

if keystate[pygame.K_RIGHT]:

self.speed_x = 5
```

```
self.rect.x += self.speed_x

if self.rect.right > screen_width:

self.rect.right = screen_width

if self.rect.left < 0:

self.rect.left = 0

# load the enemy image

enemy_img = pygame.image.load("enemy.png")

# define the enemy class

class Enemy(pygame.sprite.Sprite):

def __init__(self):

super().__init__()

self.image = enemy_img

self.rect = self.image.get_rect()

self.rect.x = random.randrange(screen_width - self.rect.width)

self.rect.y = random.randrange(-100, -40)

self.speed_y = random.randrange(1, 8)

def update(self):

self.rect.y += self.speed_y

if self.rect.top > screen_height + 10:
```

```
self.rect.x = random.randrange(screen_width - self.rect.width)

self.rect.y = random.randrange(-100, -40)

self.speed_y = random.randrange(1, 8)

# create the player sprite

player = Player()

# create the enemy group

enemies = pygame.sprite.Group()

for i in range(10):

enemy = Enemy()

enemies.add(enemy)

# set the score to 0

score = 0

# set the font

font = pygame.font.SysFont(None, 30)

# define the function to draw the score

def draw_score():

text = font.render(f"Score: {score}", True, white)

screen.blit(text, (10, 10))

# define the main function to run the game
```

```
def main():

global score

running = True

while running:

for event in pygame.event.get():

if event.type == pygame.QUIT:

running = False

# update the player sprite

player.update()

# update the enemy group

enemies.update()

# check for collisions between the player and the enemies

hits = pygame.sprite.spritecollide(player, enemies, False)

if hits:

running = False

# draw the background

screen.fill(black)

# draw the sprites

player_group = pygame.sprite.Group()
```

```
player_group.add(player)

player_group.draw(screen)

enemies.draw(screen)

# draw the score

draw_score()

# update the display

pygame.display.flip()

# set the frame rate

clock.tick(
```

PROJECT 18: LOAN PAYMENT CALCULATOR: THIS CALCULATOR CAN BE USED TO CALCULATE THE MONTHLY PAYMENTS, TOTAL INTEREST PAID, AND TOTAL AMOUNT PAID OVER THE COURSE OF A LOAN

PROMPT THE USER TO input loan details

principal = float(input("Enter the loan amount: "))

interest_rate = float(input("Enter the annual interest rate (as a decimal): "))

loan_term_years = int(input("Enter the loan term (in years): "))

Convert loan term to months and interest rate to monthly rate

loan_term_months = loan_term_years * 12

interest_rate_monthly = interest_rate / 12

Calculate the monthly payment

monthly_payment = (principal * interest_rate_monthly) / (1 - (1 + interest_rate_monthly) ** (-loan_term_months))

Calculate total amount paid and total interest paid

total_amount_paid = monthly_payment * loan_term_months

total_interest_paid = total_amount_paid - principal

Print the results

```
print("Loan amount: ${:,.2f}".format(principal))

print("Interest rate: {:.2%}".format(interest_rate))

print("Loan term: {} years".format(loan_term_years))

print("Monthly payment: ${:,.2f}".format(monthly_payment))

print("Total amount paid: ${:,.2f}".format(total_amount_paid))

print("Total interest paid: ${:,.2f}".format(total_interest_paid))
```

In this code, we first prompt the user to enter the loan amount, interest rate, and loan term in years. We then convert the loan term to months and the interest rate to a monthly rate. Using the formula for the monthly payment of a loan, we calculate the monthly payment. We then use the monthly payment to calculate the total amount paid and the total interest paid over the course of the loan. Finally, we print the results to the console in a formatted manner.

PROJECT 19: DICE ROLLING SIMULATOR: BUILD A PROGRAM THAT SIMULATES THE ROLLING OF A DICE, ALLOWING USERS TO SPECIFY THE NUMBER OF DICE AND SIDES PER DIE

THIS CODE FIRST DEFINES a function roll_dice that takes two arguments, num_dice and num_sides, and rolls num_dice dice with num_sides sides each. The function returns the total sum of the rolls and a list of the individual rolls.

The code then prompts the user to enter the number of dice and sides per die, and calls the roll_dice function with these values. Finally, the code prints the total sum and the individual rolls.

```
import random

# define the function to roll the dice

def roll_dice(num_dice, num_sides):

total = 0

rolls = []

for i in range(num_dice):

roll = random.randint(1, num_sides)

total += roll
```

```
rolls.append(roll)

return total, rolls

# get the input from the user

num_dice = int(input("Enter the number of dice: "))

num_sides = int(input("Enter the number of sides per die: "))

# roll the dice and print the results

total, rolls = roll_dice(num_dice, num_sides)

print(f"You rolled {num_dice} dice with {num_sides} sides each.")

print(f"The total is {total}.")

print(f"The individual rolls are: {rolls}")
```

PROJECT 20: AMORTIZATION SCHEDULE GENERATOR: THIS CALCULATOR CAN GENERATE A DETAILED AMORTIZATION SCHEDULE FOR A LOAN, SHOWING THE BREAKDOWN OF EACH PAYMENT INTO PRINCIPAL AND INTEREST, AS WELL AS THE REMAINING BALANCE ON THE LOAN AFTER EACH PAYMENT

```
# PROMPT THE USER TO input loan details

principal = float(input("Enter the loan amount: "))

interest_rate = float(input("Enter the annual interest rate (as a decimal): "))

loan_term_years = int(input("Enter the loan term (in years): "))

# Convert loan term to months and interest rate to monthly rate

loan_term_months = loan_term_years * 12

interest_rate_monthly = interest_rate / 12

# Calculate the monthly payment

monthly_payment = (principal * interest_rate_monthly) / (1 - (1 +
interest_rate_monthly) ** (-loan_term_months))

# Initialize variables for the amortization schedule

balance = principal
```

```
amortization_schedule = []

# Generate the amortization schedule

for month in range(1, loan_term_months + 1):

interest = balance * interest_rate_monthly

principal_paid = monthly_payment - interest

balance -= principal_paid

amortization_schedule.append((month, round(monthly_payment, 2),
round(principal_paid, 2), round(interest, 2), round(balance, 2)))

# Print the amortization schedule

print("{:<10} {:<15} {:<15} {:<15} {:<15}".format("Month", "Payment",
"Principal", "Interest", "Balance"))

for month, payment, principal, interest, balance in amortization_schedule:

print("{:<10} {:<15} {:<15} {:<15} {:<15}".format(month, payment,
principal, interest, balance))
```

In this code, we first prompt the user to enter the loan amount, interest rate, and loan term in years. We then convert the loan term to months and the interest rate to a monthly rate. Using the formula for the monthly payment of a loan, we calculate the monthly payment. We then initialize a variable for the remaining balance and an empty list to store the amortization schedule.

Using a for loop, we iterate through each month of the loan term and calculate the interest and principal paid for that month. We then subtract the principal paid from the remaining balance to get the new balance. We append the month, payment, principal, interest, and balance to the amortization schedule list.

Finally, we print the amortization schedule to the console in a formatted manner. The amortization schedule includes the month number, the monthly payment, the principal paid, the interest paid, and the remaining balance after each payment.

PROJECT 21: MAD LIBS GENERATOR: DEVELOP A GAME THAT PROMPTS USERS TO INPUT WORDS TO FILL IN THE BLANKS OF A STORY, RESULTING IN A HILARIOUS, UNEXPECTED TALE

THIS CODE FIRST PROMPTS the user to input a series of words, such as adjectives, nouns, verbs, and numbers. The input function is used to capture the user's input and assign it to variables.

The code then uses the user's words to generate a story using string formatting. The story includes a variety of placeholders, such as {adjective1}, {noun2}, and {verb_ing3}, which are replaced with the user's input.

Finally, the code prints the generated story to the console. The resulting story is typically nonsensical and hilarious, based on the user's input.

```
print("Welcome to the Mad Libs Generator!")

print("Please fill in the blanks with the requested words.\n")

# prompt the user for words

adjective1 = input("Enter an adjective: ")

adjective2 = input("Enter another adjective: ")

noun1 = input("Enter a noun: ")

noun2 = input("Enter another noun: ")

plural_noun1 = input("Enter a plural noun: ")
```

```
game = input("Enter the name of a game: ")

verb_ing1 = input("Enter a verb ending in -ing: ")

verb_ing2 = input("Enter another verb ending in -ing: ")

plural_noun2 = input("Enter another plural noun: ")

verb_ing3 = input("Enter another verb ending in -ing: ")

noun3 = input("Enter another noun: ")

plant = input("Enter the name of a plant: ")

body_part = input("Enter a body part: ")

place = input("Enter a place: ")

verb_ing4 = input("Enter another verb ending in -ing: ")

adjective3 = input("Enter another adjective: ")

number = input("Enter a number: ")

# generate the story

print(f"\nIt was a {adjective1}, cold November day. I woke up to the
{adjective2} smell of {noun1} roasting in the {noun2} downstairs. I
quickly got dressed and ran down to the {plural_noun1} to see if I could
help {verb_ing1} the dinner. My mom said, 'See if {game} is ready to be
played.' So I {verb_ing2} down to the {plural_noun2} room and found
{verb_ing3} with my {noun3}. After we played {game}, I was exhausted. I
climbed into bed and pulled the {plant} over my {body_part}. It wasn't
long before I was {verb_ing4} asleep. \n\nThe End\n\nAs I slept, I dreamt
about {adjective3} {plural_noun2} chasing me through {place} and a giant
{noun1} with {number} {adjective1} heads." )
```

PROJECT 22: PRICE PREDICTION MODEL: SELECT AN INDUSTRY OR PRODUCT THAT INTERESTS YOU, AND BUILD A MACHINE LEARNING MODEL THAT PREDICTS PRICE CHANGES

THIS CODE FIRST LOADS historical Apple stock price data from a CSV file using the Pandas library. It then selects relevant features, such as the opening and closing prices, and volume of stocks traded.

The data is then split into a training set and a testing set using the train_test_split function from Scikit-Learn. A linear regression model is created using the training data, and the model is then used to make predictions on the testing data.

Finally, the accuracy of the model is calculated using the score function from Scikit-Learn, which measures how well the model predicts the actual stock prices in the testing data. The accuracy score is printed to the console.

```
import pandas as pd

import numpy as np

from sklearn.linear_model import LinearRegression

from sklearn.model_selection import train_test_split

# load historical Apple stock price data

apple_df = pd.read_csv("AAPL.csv")
```

```
# select relevant features for the prediction model

features = ["Open", "High", "Low", "Volume"]

# split data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(apple_df[features],
apple_df["Close"], test_size=0.2, random_state=42)

# create and train the linear regression model

model = LinearRegression()

model.fit(X_train, y_train)

# use the trained model to make predictions on the testing data

predictions = model.predict(X_test)

# calculate the accuracy of the model

accuracy = model.score(X_test, y_test)

# print the accuracy score to the console

print(f"Accuracy score: {accuracy}")
```

Note that the accuracy of the model will depend on the quality and relevance of the data used to train it, as well as the choice of algorithm and any tuning of the model's hyperparameters. This example code is intended as a starting point for building a price prediction model and should be modified and adapted to fit the specific industry or product being analyzed.

PROJECT 23: URL SHORTENER: DESIGN A WEB APPLICATION THAT CAN TAKE LONG URLS AND SHORTEN THEM, ALLOWING USERS TO SHARE THEM EASILY

THIS CODE FIRST PROMPTS the user to enter a long URL to be shortened. The requests library is then used to send a POST request to the Bitly API's shorten endpoint, along with the user's long URL and an access token.

The response from the API is in JSON format and includes a shortened URL. The code parses the JSON response and extracts the shortened URL using dictionary indexing. Finally, the shortened URL is printed to the console.

Note that in order to use this code, you will need to sign up for a Bitly account and generate an access token. You can do this by following the instructions on the Bitly API documentation website.

```
import requests

import json

# define the endpoint and access token for the Bitly API

endpoint = "https://api-ssl.bitly.com/v4/shorten"

access_token = "<your-access-token>"

# prompt the user to enter a long URL to be shortened
```

```
long_url = input("Enter a long URL to shorten: ")

# define the request headers and body

headers = {

"Authorization": f"Bearer {access_token}",

"Content-Type": "application/json"

}

data = {

"long_url": long_url

}

# send the request to the Bitly API to generate a shortened URL

response = requests.post(endpoint, headers=headers,
data=json.dumps(data))

# parse the JSON response and extract the shortened URL

response_data = json.loads(response.text)

short_url = response_data["link"]

# print the shortened URL to the console

print(f"Your shortened URL is: {short_url}")
```

PROJECT 24: SNAKE GAME: DEVELOP A CLASSIC ARCADE-STYLE GAME WHERE THE PLAYER NAVIGATES A SNAKE AROUND A GRID AND EATS FOOD TO GROW, AVOIDING OBSTACLES AND THE SNAKE'S OWN TAIL

FIRST, WE IMPORT THE pygame library and the random module, which we'll use to generate random positions for the food. Next, we initialize Pygame and set up the game window and clock. We also set up a font for displaying the score and define some colors. Then, we set up the initial position and size of the snake, the initial position and size of the food, the initial direction of the snake, and the initial score. Inside the main game loop, we handle events such as quitting the game or changing the direction of the snake. We then move the snake based on its direction, check for collision with the food, the walls, and the snake's own body, and update the snake's body accordingly. Finally, we draw the background, the food, and the snake on the game window.

```
import pygame

import random

# initialize Pygame

pygame.init()

# set up the game window

window_width = 640

window_height = 480
```

window = pygame.display.set_mode((window_width, window_height))

set up the game clock

clock = pygame.time.Clock()

set up the font for displaying the score

font = pygame.font.SysFont(None, 30)

define some colors

white = (255, 255, 255)

black = (0, 0, 0)

red = (255, 0, 0)

set up the initial position and size of the snake

snake_size = 10

snake_x = window_width / 2

snake_y = window_height / 2

snake_speed = 10

snake_body = [(snake_x, snake_y)]

set up the initial position and size of the food

food_size = 10

food_x = random.randrange(0, window_width - food_size, 10)

food_y = random.randrange(0, window_height - food_size, 10)

```
food_rect = pygame.Rect(food_x, food_y, food_size, food_size)

# set up the initial direction of the snake

direction = "right"

# set up the initial score

score = 0

# main game loop

while True:

# handle events

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

quit()

elif event.type == pygame.KEYDOWN:

if event.key == pygame.K_LEFT and direction != "right":

direction = "left"

elif event.key == pygame.K_RIGHT and direction != "left":

direction = "right"

elif event.key == pygame.K_UP and direction != "down":

direction = "up"
```

```
elif event.key == pygame.K_DOWN and direction != "up":

direction = "down"

# move the snake

if direction == "right":

snake_x += snake_speed

elif direction == "left":

snake_x -= snake_speed

elif direction == "up":

snake_y -= snake_speed

elif direction == "down":

snake_y += snake_speed

# check for collision with the food

if snake_x == food_x and snake_y == food_y:

# increase the score

score += 1

# generate new food

food_x = random.randrange(0, window_width - food_size, 10)

food_y = random.randrange(0, window_height - food_size, 10)

food_rect = pygame.Rect(food_x, food_y, food_size, food_size)
```

```
# add a new segment to the snake

snake_body.append((snake_x, snake_y))

# check for collision with the walls

if snake_x < 0 or snake_x > window_width - snake_size or snake_y < 0 or snake_y > window_height - snake_size:

pygame.quit()

quit()

# check for collision with the snake's own body

for segment in snake_body[:-1]:

if segment == (snake_x, snake_y):

pygame.quit()

quit()

# move the snake's body

snake_body.insert(0, (snake_x, snake_y))

if len(snake_body) > score + 1:

snake_body.pop()

# draw the background

window.fill(black)

# draw the food
```

pygame.draw.rect(window, red, food_rect)

draw the snake

for segment in snake_body:

segment_rect

PROJECT 25: WORD COUNT TOOL: DEVELOP A PROGRAM THAT TAKES IN A TEXT FILE AND COUNTS THE NUMBER OF WORDS IN IT

DEF WORD_COUNT(FILE_name):

with open(file_name, 'r') as file:

text = file.read()

words = text.split()

count = len(words)

return count

This function takes a file name as input and returns the number of words in the file. Here's how it works:

We first open the file using a context manager (with open(file_name, 'r') as file) and read its contents into a string variable text.

We then split the text into a list of words using the split() method. By default, split() splits the text at any whitespace character (space, tab, newline) and returns a list of the resulting "words".

We use the len() function to count the number of items in the words list, which corresponds to the number of words in the text. Finally, we return the word count.

You can call this function like this:

```
count = word_count('my_text_file.txt')

print('Word count:', count)
```

PROJECT 26: SENTIMENT ANALYSIS TOOL: CREATE A PROGRAM THAT ANALYZES TEXT AND DETERMINES THE SENTIMENT BEHIND IT USING NATURAL LANGUAGE PROCESSING TECHNIQUES

IMPORT NLTK

from nltk.sentiment import SentimentIntensityAnalyzer

def analyze_sentiment(text):

sia = SentimentIntensityAnalyzer()

sentiment = sia.polarity_scores(text)

if sentiment['compound'] >= 0.05:

return 'Positive'

elif sentiment['compound'] <= -0.05:

return 'Negative'

else:

return 'Neutral'

This function takes a text string as input and returns the sentiment of the text as a string ('Positive', 'Negative', or 'Neutral'). Here's how it works:

We first import the NLTK library and the SentimentIntensityAnalyzer class, which provides a pre-trained sentiment analysis model.

We create an instance of the SentimentIntensityAnalyzer class using sia = SentimentIntensityAnalyzer().

We pass the text string to the polarity_scores() method of the SentimentIntensityAnalyzer instance, which returns a dictionary containing the scores for positive, negative, and neutral sentiment, as well as a compound score that combines all three scores.

We check the compound score to determine the sentiment of the text. If the compound score is greater than or equal to 0.05, we classify the text as 'Positive'. If the compound score is less than or equal to -0.05, we classify the text as 'Negative'. Otherwise, we classify the text as 'Neutral'.

Here's an example usage of the function:

```
text = "I really enjoyed the movie, it was great!"

sentiment = analyze_sentiment(text)

print(sentiment)  # Output: Positive
```

PROJECT 27: SUDOKU SOLVER: DEVELOP A PROGRAM THAT SOLVES A SUDOKU PUZZLE USING ALGORITHMS LIKE BACKTRACKING

DEF SOLVE_SUDOKU(PUZZLE):

"""

Solves a Sudoku puzzle and returns the solved puzzle.

Input: 9x9 nested list of integers representing the puzzle

Empty cells are represented by 0s

Output: Solved 9x9 nested list of integers

"""

Find an empty cell

empty_cell = find_empty(puzzle)

if not empty_cell:

Puzzle is solved

return puzzle

Try different values for the empty cell

row, col = empty_cell

for num in range(1, 10):

```
if is_valid(puzzle, row, col, num):

puzzle[row][col] = num

# Recursively solve the puzzle

if solve_sudoku(puzzle):

return puzzle

# If the puzzle cannot be solved with the current value,

# backtrack and try the next value

puzzle[row][col] = 0

# Puzzle is unsolvable

return None
```

```
def find_empty(puzzle):

"""

Finds an empty cell (represented by a 0) in the puzzle.

Returns the row and column of the empty cell, or None if there

are no empty cells.

"""

for row in range(9):

for col in range(9):
```

```
if puzzle[row][col] == 0:
return (row, col)
return None
```

```
def is_valid(puzzle, row, col, num):
"""
Checks if the given number can be placed in the given row and column
of the puzzle without violating the Sudoku rules.
"""
# Check row
if num in puzzle[row]:
return False
# Check column
for i in range(9):
if puzzle[i][col] == num:
return False
# Check 3x3 box
box_row = (row // 3) * 3
box_col = (col // 3) * 3
```

for i in range(box_row, box_row + 3):

for j in range(box_col, box_col + 3):

if puzzle[i][j] == num:

return False

The number can be placed in the cell

return True

This code defines three functions:

solve_sudoku(puzzle): Takes a 9x9 nested list of integers representing a Sudoku puzzle as input, where empty cells are represented by 0s. It uses a recursive backtracking algorithm to solve the puzzle and returns the solved puzzle as a 9x9 nested list of integers.

find_empty(puzzle): Takes a 9x9 nested list of integers representing a Sudoku puzzle as input and returns the row and column of an empty cell (represented by a 0). If there are no empty cells, it returns None.

is_valid(puzzle, row, col, num): Takes a 9x9 nested list of integers representing a Sudoku puzzle, a row and column index, and a number as input. It checks if the number can be placed in the given row and column without violating the Sudoku rules.

Here's an example usage of the solve_sudoku() function:

puzzle = [

[0, 0, 0, 0, 0, 0, 0, 1, 0],

[4, 0, 9, 0, 2, 0, 0, 0, 0],

[0, 0, 0, 8,

PROJECT 28: LANGUAGE TRANSLATOR: CREATE A PROGRAM THAT TRANSLATES TEXT FROM ONE LANGUAGE TO ANOTHER USING LIBRARIES LIKE GOOGLE TRANSLATE OR PYTRANSKIT

FROM GOOGLETRANS IMPORT Translator

def translate_text(text, target_language='en'):

"""

Translates a text to the specified target language using Google Translate API.

Input: text (str) - The text to be translated

target_language (str) - The language code of the target language (default is 'en')

Output: The translated text (str)

"""

Initialize the translator

translator = Translator()

Translate the text

translation = translator.translate(text, dest=target_language)

Return the translated text

return translation.text

This code defines a single function translate_text(text, target_language='en') that takes a text and a target language code as input and returns the translated text using the Google Translate API. The function uses the googletrans library, which provides an interface to the Google Translate API.

Here's an example usage of the translate_text() function:

text = "Hello, how are you?"

target_language = 'fr'

translation = translate_text(text, target_language)

print(translation)

This will translate the text "Hello, how are you?" to French and print the result. You can change the target_language parameter to translate the text to a different language. Note that there are some limitations to the usage of the Google Translate API, such as rate limits and usage fees beyond a certain number of requests.

PROJECT 29: NEWS AGGREGATOR: DEVELOP A PROGRAM THAT AGGREGATES NEWS ARTICLES FROM DIFFERENT SOURCES AND DISPLAYS THEM TO THE USER IN A READABLE FORMAT

FROM NEWSAPI IMPORT NewsApiClient

def get_news(api_key, sources, language='en'):

"""

Retrieves news articles from the specified news sources using the NewsAPI.

Input: api_key (str) - The NewsAPI API key

sources (list of str) - The news sources to retrieve articles from

language (str) - The language code of the articles to retrieve (default is 'en')

Output: The news articles (list of dict)

"""

Initialize the NewsAPI client

newsapi = NewsApiClient(api_key=api_key)

Retrieve the news articles

articles = newsapi.get_everything(sources=sources, language=language)['articles']

Return the articles

return articles

This code defines a single function get_news(api_key, sources, language='en') that takes an API key, a list of news sources, and a language code as input and returns a list of news articles. The function uses the newsapi library, which provides an interface to the NewsAPI.

Here's an example usage of the get_news() function:

api_key = 'your_api_key'

sources = ['bbc-news', 'cnn', 'reuters']

articles = get_news(api_key, sources)

for article in articles:

print(article['title'])

print(article['description'])

print(article['url'])

This will retrieve the latest news articles from BBC News, CNN, and Reuters, and print the title, description, and URL of each article. You can change the sources parameter to retrieve articles from different news sources, and the language parameter to retrieve articles in a different language. Note that there are some limitations to the usage of the NewsAPI, such as rate limits and usage fees beyond a certain number of requests.

PROJECT 30: RECOMMENDATION SYSTEM: BUILD A PROGRAM THAT RECOMMENDS PRODUCTS, MOVIES, OR MUSIC BASED ON A USER'S PREFERENCES USING COLLABORATIVE FILTERING OR CONTENT-BASED FILTERING ALGORITHMS

FROM SURPRISE IMPORT Dataset, Reader, SVD

from surprise.model_selection import train_test_split

def get_recommendations(user_id, num_recommendations=10):

"""

Recommends movies to a user based on their past ratings using collaborative filtering.

Input: user_id (int) - The ID of the user to recommend movies to

num_recommendations (int) - The number of movies to recommend (default is 10)

Output: The recommended movies (list of tuple)

"""

Load the MovieLens 100K dataset

reader = Reader(line_format='user item rating timestamp', sep='\t')

data = Dataset.load_from_file('path/to/dataset', reader=reader)

Train the SVD algorithm on the dataset

trainset, testset = train_test_split(data, test_size=0.25)

algo = SVD()

algo.fit(trainset)

Predict the ratings for the user

user_ratings = []

for item_id in range(1, algo.n_items+1):

rating = algo.predict(str(user_id), str(item_id)).est

user_ratings.append((item_id, rating))

Sort the movies by rating and return the top recommendations

user_ratings.sort(key=lambda x: x[1], reverse=True)

recommendations = user_ratings[:num_recommendations]

return recommendations

This code defines a single function get_recommendations(user_id, num_recommendations=10) that takes a user ID and a number of recommendations as input and returns a list of recommended movies. The function uses the MovieLens 100K dataset, trains the SVD algorithm on the dataset, and predicts the ratings for the specified user using collaborative filtering. The recommended movies are then sorted by rating and returned.

Here's an example usage of the get_recommendations() function:

user_id = 1

```
num_recommendations = 10

recommendations = get_recommendations(user_id, num_recommendations)

for recommendation in recommendations:

print(recommendation)
```

This will recommend 10 movies to the user with ID 1 and print the movie ID and predicted rating for each recommendation. You can change the user_id parameter to recommend movies to a different user, and the num_recommendations parameter to recommend a different number of movies. Note that there are other algorithms available in the surprise library and other datasets that can be used for recommendation systems.

PROJECT 31: NEURAL NETWORK CLASSIFIER: BUILD A PROGRAM THAT USES NEURAL NETWORKS TO CLASSIFY IMAGES, TEXT, OR OTHER TYPES OF DATA

```
FROM KERAS.MODELS IMPORT Sequential

from keras.layers import Dense

from keras.datasets import mnist

from keras.utils import to_categorical

# Load the MNIST dataset

(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

# Preprocess the data

train_images = train_images.reshape((60000, 784))

train_images = train_images.astype('float32') / 255

test_images = test_images.reshape((10000, 784))

test_images = test_images.astype('float32') / 255

train_labels = to_categorical(train_labels)

test_labels = to_categorical(test_labels)

# Define the model architecture
```

```
model = Sequential()

model.add(Dense(512, activation='relu', input_shape=(784,)))

model.add(Dense(10, activation='softmax'))

# Compile the model

model.compile(optimizer='rmsprop',

loss='categorical_crossentropy',

metrics=['accuracy'])

# Train the model

model.fit(train_images, train_labels, epochs=5, batch_size=128)

# Evaluate the model on the test data

test_loss, test_acc = model.evaluate(test_images, test_labels)

print('Test accuracy:', test_acc)
```

This code defines a neural network with one input layer, one hidden layer, and one output layer, and trains the network on the MNIST dataset of handwritten digits. The input layer has 784 neurons (one for each pixel in the 28x28 images), the hidden layer has 512 neurons with a ReLU activation function, and the output layer has 10 neurons with a softmax activation function (one for each digit). The model is compiled with the RMSprop optimizer and the categorical cross-entropy loss function, and trained for 5 epochs with a batch size of 128. Finally, the model is evaluated on the test data and the test accuracy is printed.

You can modify this code to work with other datasets and adjust the model architecture and hyperparameters to improve the accuracy of the classifier.

PROJECT 32: OBJECT DETECTION APP: CREATE A PROGRAM THAT USES DEEP LEARNING ALGORITHMS TO DETECT AND LOCATE OBJECTS IN IMAGES OR VIDEOS

```
IMPORT CV2

import numpy as np

import tensorflow as tf

# Load the COCO SSD model

model =
tf.keras.models.load_model('ssd_mobilenet_v2_coco_2018_03_29/saved_
model')

# Define the classes and colors for the objects

classes = ['background', 'person', 'bicycle', 'car', 'motorcycle', 'airplane',
'bus', 'train', 'truck', 'boat', 'traffic light', 'fire hydrant', 'stop sign', 'parking
meter', 'bench', 'bird', 'cat', 'dog', 'horse', 'sheep', 'cow', 'elephant', 'bear',
'zebra', 'giraffe', 'backpack', 'umbrella', 'handbag', 'tie', 'suitcase', 'frisbee',
'skis', 'snowboard', 'sports ball', 'kite', 'baseball bat', 'baseball glove',
'skateboard', 'surfboard', 'tennis racket', 'bottle', 'wine glass', 'cup', 'fork',
'knife', 'spoon', 'bowl', 'banana', 'apple', 'sandwich', 'orange', 'broccoli',
'carrot', 'hot dog', 'pizza', 'donut', 'cake', 'chair', 'couch', 'potted plant', 'bed',
'dining table', 'toilet', 'tv', 'laptop', 'mouse', 'remote', 'keyboard', 'cell phone',
'microwave', 'oven', 'toaster', 'sink', 'refrigerator', 'book', 'clock', 'vase',
'scissors', 'teddy bear', 'hair drier', 'toothbrush']

colors = np.random.uniform(0, 255, size=(len(classes), 3))
```

```
# Load an image

img = cv2.imread('image.jpg')

# Preprocess the image

img = cv2.resize(img, (300, 300))

img = img.astype('float32') / 255

img = np.expand_dims(img, axis=0)

# Detect objects in the image

detections = model.predict(img)

# Process the detections

for i in range(detections.shape[1]):

class_id = int(detections[0, i, 1])

score = detections[0, i, 2]

if score > 0.5:

bbox = detections[0, i, 3:7] * np.array([img.shape[2], img.shape[1],
img.shape[2], img.shape[1]])

x, y, w, h = bbox.astype('int')

label = '{}: {:.2f}'.format(classes[class_id], score)

color = colors[class_id]

cv2.rectangle(img, (x, y), (w, h), color, 2)
```

```
cv2.putText(img, label, (x, y-10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)

# Display the image with the detections

cv2.imshow('Object Detection', img)

cv2.waitKey(0)

cv2.destroyAllWindows()
```

This code loads the pre-trained "COCO SSD" model, which is a neural network that detects and localizes objects in images. The model is compiled and trained on the COCO dataset, which contains 80 object categories. The code loads an image, resizes it to 300x300 pixels, and preprocesses it by normalizing its pixel values.

PROJECT 33: AUTONOMOUS TRADING BOT: CREATE A PROGRAM THAT USES MACHINE LEARNING ALGORITHMS TO ANALYZE FINANCIAL MARKETS AND MAKE AUTONOMOUS TRADES

CREATING AN AUTONOMOUS trading bot involves a complex set of tasks including data collection, cleaning, and analysis, developing and training machine learning models, and integrating the models with a trading platform. Here's a brief overview of how to create an autonomous trading bot in Python:

1. Collect and Clean Data: First, you need to collect the data from various sources such as Yahoo Finance, Alpha Vantage, or any other financial data providers. After that, you need to clean the data by removing missing or irrelevant values and convert it into a format suitable for analysis.
2. Analyze Data: Once the data is clean and in the proper format, you can perform exploratory data analysis to get insights into the data. You can use tools such as pandas, NumPy, and Matplotlib to visualize and analyze the data.
3. Develop Machine Learning Models: After analyzing the data, you can develop machine learning models to predict stock prices or identify trading signals. You can use libraries such as scikit-learn or TensorFlow to develop and train machine learning models.
4. Backtesting: Before deploying the trading bot in the real market, you need to test the performance of the model on historical data. This process is called backtesting, which involves testing the model's performance on historical data to see how it would have performed in the past.
5. Integration with a Trading Platform: Once the model is trained and tested, you can integrate it with a trading platform such as Alpaca, TD

Ameritrade, or Interactive Brokers to execute trades autonomously based on the model's signals.

Here is an example Python code for predicting stock prices using a machine learning model:

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

# Load data

data = pd.read_csv('AAPL.csv')

# Prepare data

X = data.iloc[:, 1:2].values

y = data.iloc[:, 4].values

# Split data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=0)

# Train the model

regressor = LinearRegression()

regressor.fit(X_train, y_train)
```

```
# Make predictions on the testing set

y_pred = regressor.predict(X_test)

# Visualize the results

plt.scatter(X_train, y_train, color='red')

plt.plot(X_train, regressor.predict(X_train), color='blue')

plt.title('AAPL Stock Price Prediction')

plt.xlabel('Date')

plt.ylabel('Price')

plt.show()
```

This code uses a simple linear regression model to predict the stock price of Apple Inc. (AAPL) based on its historical stock price data. The code loads the data from a CSV file and splits it into training and testing sets. It then trains the linear regression model on the training set and makes predictions on the testing set. Finally, it visualizes the results using Matplotlib. This is just a simple example, but it demonstrates the basic process of training and testing a machine learning model for stock price prediction.

PROJECT 34: GENERATIVE ADVERSARIAL NETWORK (GAN) TOOL: CREATE A PROGRAM THAT USES GANS, WHICH ARE DEEP LEARNING ALGORITHMS, TO GENERATE REALISTIC IMAGES OR OTHER TYPES OF DATA

GENERATIVE ADVERSARIAL Networks (GANs) are deep learning models that can generate new data that is similar to a training set. The GAN consists of two neural networks: a generator and a discriminator. The generator creates new data based on random noise, and the discriminator distinguishes the generated data from real data. Here's an example Python code for generating handwritten digits using a GAN:

```
import numpy as np

import matplotlib.pyplot as plt

from keras.datasets import mnist

from keras.layers import Dense, Reshape, Flatten, Input

from keras.layers import Conv2D, Conv2DTranspose

from keras.layers import LeakyReLU, Dropout

from keras.models import Sequential, Model

from keras.optimizers import Adam

# Load and normalize the MNIST dataset

(X_train, _), (_, _) = mnist.load_data()
```

```
X_train = (X_train.astype(np.float32) - 127.5) / 127.5

X_train = np.expand_dims(X_train, axis=3)

# Define the generator model

def build_generator(latent_dim):

model = Sequential()

model.add(Dense(128 * 7 * 7, activation='relu', input_dim=latent_dim))

model.add(Reshape((7, 7, 128)))

model.add(Conv2DTranspose(128, kernel_size=4, strides=2,
padding='same', activation='relu'))

model.add(Conv2DTranspose(64, kernel_size=4, strides=2,
padding='same', activation='relu'))

model.add(Conv2DTranspose(1, kernel_size=7, strides=1, padding='same',
activation='tanh'))

noise = Input(shape=(latent_dim,))

img = model(noise)

return Model(noise, img)

# Define the discriminator model

def build_discriminator(img_shape):

model = Sequential()

model.add(Conv2D(64, kernel_size=3, strides=2, input_shape=img_shape,
padding='same'))
```

```
model.add(LeakyReLU(alpha=0.2))

model.add(Dropout(0.25))

model.add(Conv2D(128, kernel_size=3, strides=2, padding='same'))

model.add(LeakyReLU(alpha=0.2))

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(1, activation='sigmoid'))

img = Input(shape=img_shape)

validity = model(img)

return Model(img, validity)

# Build and compile the GAN model

def build_gan(generator, discriminator):

discriminator.trainable = False

gan_input = Input(shape=(latent_dim,))

img = generator(gan_input)

validity = discriminator(img)

gan = Model(gan_input, validity)

gan.compile(loss='binary_crossentropy', optimizer=Adam(lr=0.0002,
beta_1=0.5))
```

```
return gan

# Set hyperparameters

img_shape = X_train.shape[1:]

latent_dim = 100

batch_size = 128

epochs = 10000

sample_interval = 1000

# Build the generator and discriminator models

generator = build_generator(latent_dim)

discriminator = build_discriminator(img_shape)

# Build the GAN model

gan = build_gan(generator, discriminator)

# Train the GAN

for epoch in range(epochs):

# Train discriminator

idx = np.random.randint(0, X_train.shape[0], batch_size)

imgs = X_train[idx]

noise = np.random.normal(0, 1, (batch_size, latent_dim))

gen_imgs = generator.predict(noise)
```

d_loss_real = discriminator.train

PROJECT 35: BLOCKCHAIN APPLICATION: BUILD A DECENTRALIZED APPLICATION USING BLOCKCHAIN TECHNOLOGY FOR SECURE, TRANSPARENT, AND TAMPER-PROOF DATA STORAGE AND SHARING

```
IMPORT HASHLIB

import json

from time import time

class Blockchain:

def __init__(self):

self.chain = []

self.current_transactions = []

# Create the genesis block

self.new_block(previous_hash=1, proof=100)

def new_block(self, proof, previous_hash=None):

"""

Create a new Block in the Blockchain

:param proof: <int> The proof given by the Proof of Work algorithm
```

```
:param previous_hash: (Optional) <str> Hash of previous Block

:return: <dict> New Block

"""

block = {

'index': len(self.chain) + 1,

'timestamp': time(),

'transactions': self.current_transactions,

'proof': proof,

'previous_hash': previous_hash or self.hash(self.chain[-1]),

}

# Reset the current list of transactions

self.current_transactions = []

self.chain.append(block)

return block

def new_transaction(self, sender, recipient, amount):

"""

Creates a new transaction to go into the next mined Block

:param sender: <str> Address of the Sender

:param recipient: <str> Address of the Recipient
```

```
:param amount: <int> Amount

:return: <int> The index of the Block that will hold this transaction

"""

self.current_transactions.append({

'sender': sender,

'recipient': recipient,

'amount': amount,

})

return self.last_block['index'] + 1

@staticmethod

def hash(block):

"""

Creates a SHA-256 hash of a Block

:param block: <dict> Block

:return: <str>

"""

# We must make sure that the Dictionary is Ordered, or we'll have
inconsistent hashes

block_string = json.dumps(block, sort_keys=True).encode()
```

```
return hashlib.sha256(block_string).hexdigest()

@property

def last_block(self):

return self.chain[-1]

def proof_of_work(self, last_proof):

"""

Simple Proof of Work Algorithm:

- Find a number p' such that hash(pp') contains leading 4 zeroes, where p is the previous p'

- p is the previous proof, and p' is the new proof

:param last_proof: <int>

:return: <int>

"""

proof = 0

while self.valid_proof(last_proof, proof) is False:

proof += 1

return proof

@staticmethod

def valid_proof(last_proof, proof):
```

```
"""

Validates the Proof: Does hash(last_proof, proof) contain 4 leading zeroes?

:param last_proof: <int> Previous Proof

:param proof: <int> Current Proof

:return: <bool> True if correct, False if not.

"""

guess = f'{last_proof}{proof}'.encode()

guess_hash = hashlib.sha256(guess).hexdigest()

return guess_hash[:4] == "0000"
```

This is a basic implementation of a blockchain in Python. It defines a class called Blockchain that has methods for creating new blocks, adding transactions, and validating proof of work. It also has a method for hashing blocks using the SHA-256 algorithm. This implementation uses a simple proof of work algorithm to create a new block. It also includes a method for validating the proof of work.

This implementation is not meant for production use and is only for educational purposes. In a real-world implementation, there would be additional features like consensus algorithms and network communication.

PROJECT 36: PREDICTIVE ANALYTICS TOOL: BUILD A PROGRAM THAT USES STATISTICAL MODELS AND MACHINE LEARNING ALGORITHMS TO PREDICT FUTURE TRENDS OR OUTCOMES

```
IMPORT PANDAS AS PD

from sklearn.linear_model import LinearRegression

# Load the dataset

data = pd.read_csv('house_data.csv')

# Separate the features and the target variable

X = data.drop('price', axis=1)

y = data['price']

# Create and fit the model

model = LinearRegression()

model.fit(X, y)

# Make a prediction for a new house

new_house = [3000, 4, 3, 0, 1, 1, 1]

predicted_price = model.predict([new_house])

print(f"The predicted price of the house is ${predicted_price[0]:,.2f}")
```

This code assumes that you have a CSV file called house_data.csv that contains the features and the target variable of a set of houses, where the price column represents the target variable. The code loads the data into a pandas DataFrame, separates the features (X) from the target variable (y), and fits a linear regression model using the features and the target variable. Finally, the code makes a prediction for a new house with the features [3000, 4, 3, 0, 1, 1, 1] and prints the predicted price.

PROJECT 37: AUTOMATIC TEXT SUMMARIZATION TOOL: DEVELOP A PROGRAM THAT SUMMARIZES LONG ARTICLES OR DOCUMENTS USING MACHINE LEARNING ALGORITHMS

HERE'S AN EXAMPLE CODE using Natural Language Processing (NLP) library called spacy to create a simple automatic text summarization tool in Python:

import spacy

from heapq import nlargest

Load the English language model in spaCy

nlp = spacy.load('en_core_web_sm')

Function to generate the summary

def generate_summary(text):

Parse the text using spaCy

doc = nlp(text)

Calculate the sentence scores based on the number of entities and words

sentence_scores = {}

for sent in doc.sents:

for token in sent:

```
if not token.is_stop and not token.is_punct:

if token.text.lower() in sentence_scores:

sentence_scores[token.text.lower()] += 1

else:

sentence_scores[token.text.lower()] = 1

# Find the top 3 sentences with the highest scores

summary_sentences = nlargest(3, sentence_scores,
key=sentence_scores.get)

# Return the summary as a string

return ' '.join(summary_sentences)

# Example usage

text = 'Natural language processing (NLP) is a subfield of linguistics,
computer science, information engineering, and artificial intelligence
concerned with the interactions between computers and human (natural)
languages. As such, NLP is related to the area of human–computer
interaction. Many challenges in NLP involve natural language
understanding, that is, enabling computers to derive meaning from human
or natural language input, and others involve natural language generation. A
commonly applied method for summarizing text is called textRank.'

summary = generate_summary(text)

print(summary)
```

This code will take in a long text as input, and using spaCy's NLP model, it will extract the sentences with the highest scores based on the number of entities and words in each sentence. The generate_summary function returns a string containing the top 3 sentences with the highest scores, which is then printed to the console. You can adjust the number of sentences returned by changing the argument to the nlargest function.

PROJECT 38: FRAUD DETECTION TOOL: CREATE A PROGRAM THAT USES MACHINE LEARNING ALGORITHMS TO DETECT FRAUD IN FINANCIAL TRANSACTIONS OR OTHER TYPES OF DATA

HERE'S AN EXAMPLE CODE using the scikit-learn library to create a fraud detection tool in Python:

```
import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score, confusion_matrix

# Load the dataset into a pandas dataframe

df = pd.read_csv('fraud_dataset.csv')

# Split the dataset into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(df.drop('label', axis=1),
df['label'], test_size=0.2, random_state=42)

# Create and train the logistic regression model

model = LogisticRegression()
```

```
model.fit(X_train, y_train)

# Test the model on the testing set

y_pred = model.predict(X_test)

# Calculate the accuracy and confusion matrix

accuracy = accuracy_score(y_test, y_pred)

confusion = confusion_matrix(y_test, y_pred)

# Print the results

print('Accuracy:', accuracy)

print('Confusion Matrix:')

print(confusion)
```

In this example, we assume that the fraud data is stored in a CSV file named fraud_dataset.csv. The first column in the CSV file represents the label (i.e., whether a transaction is fraudulent or not), and the rest of the columns represent features of the transactions (e.g., transaction amount, location, etc.).

The code loads the dataset into a pandas dataframe, splits the dataset into training and testing sets, and trains a logistic regression model on the training set. Then, it tests the model on the testing set, calculates the accuracy and confusion matrix, and prints the results to the console.

Note that this is just a simple example, and there are many more sophisticated machine learning algorithms that can be used for fraud detection, such as random forests, decision trees, and neural networks. Additionally, it's important to have a large and diverse dataset to train the model on and to continuously update and improve the model as new data becomes available.

PROJECT 39: RECOMMENDATION SYSTEM WITH EXPLAINABLE AI: DEVELOP A RECOMMENDATION SYSTEM THAT NOT ONLY PROVIDES RECOMMENDATIONS BUT ALSO EXPLAINS WHY A PARTICULAR RECOMMENDATION WAS MADE

HERE'S AN EXAMPLE CODE using the scikit-learn and eli5 libraries to create a recommendation system with explainable AI in Python:

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestRegressor

import eli5

from eli5.sklearn import PermutationImportance

# Load the dataset into a pandas dataframe

df = pd.read_csv('ratings_dataset.csv')

# Split the dataset into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(df.drop('rating', axis=1), df['rating'], test_size=0.2, random_state=42)

# Create and train the random forest regression model

model = RandomForestRegressor()
```

```
model.fit(X_train, y_train)

# Use permutation importance to identify the most important features

perm = PermutationImportance(model).fit(X_test, y_test)

eli5.show_weights(perm, feature_names = X_test.columns.tolist())

# Make a recommendation and explain why it was made

user_id = 1

movie_id = 100

user_data = [user_id, movie_id, 0, 0, 0, 0, 0, 0, 0, 0]

predicted_rating = model.predict([user_data])[0]

explanation = eli5.explain_prediction(model, [user_data],
feature_names=X_test.columns.tolist())

print('The predicted rating for user', user_id, 'on movie', movie_id, 'is',
predicted_rating)

print('Explanation:', explanation)
```

In this example, we assume that the ratings data is stored in a CSV file named ratings_dataset.csv. The CSV file should have columns for the user ID, movie ID, and rating. The other columns can represent additional features such as the genre of the movie, the release year, etc.

The code loads the dataset into a pandas dataframe, splits the dataset into training and testing sets, and trains a random forest regression model on the training set. Then, it uses permutation importance to identify the most important features in the model and visualizes them using the eli5 library.

Finally, it makes a recommendation and explains why the recommendation was made. The code creates a user data array with the user ID, movie ID, and 8 other features set to 0. It then uses the model to predict the rating for the user on the movie and generates an explanation using the eli5 library. The predicted rating and explanation are printed to the console.

Note that this is just a simple example, and there are many more sophisticated recommendation algorithms that can be used, such as collaborative filtering, content-based filtering, and hybrid filtering. Additionally, it's important to have a large and diverse dataset to train the model on and to continuously update and improve the model as new data becomes available.

PROJECT 40: BMI CALCULATOR: DEVELOP A PROGRAM THAT CALCULATES BODY MASS INDEX (BMI) BASED ON A USER'S HEIGHT AND WEIGHT

```
# PROMPT THE USER TO enter their weight and height

weight = float(input("Enter your weight in kilograms: "))

height = float(input("Enter your height in meters: "))

# Calculate the BMI

bmi = weight / (height ** 2)

# Print the result

print("Your BMI is: {:.2f}".format(bmi))
```

In this example, we prompt the user to enter their weight in kilograms and their height in meters. Then, we calculate the BMI using the formula weight / (height ** 2) and round the result to two decimal places using the format function. Finally, we print the result to the console.

Note that BMI is just one measure of body fat based on weight and height. It does not take into account factors such as muscle mass, bone density, or overall body composition. Therefore, it's important to consult a healthcare professional to assess your overall health and fitness.

PROJECT 41: TEXT EDITOR: CREATE A SIMPLE TEXT EDITOR THAT ALLOWS USERS TO CREATE, EDIT, AND SAVE TEXT FILES

IMPORT TKINTER AS TK

from tkinter import filedialog

class TextEditor:

def __init__(self, root):

self.root = root

self.root.title("Text Editor")

Create a menu bar

menu_bar = tk.Menu(self.root)

self.root.config(menu=menu_bar)

Create a file menu

file_menu = tk.Menu(menu_bar, tearoff=0)

file_menu.add_command(label="New", command=self.new_file)

file_menu.add_command(label="Open", command=self.open_file)

file_menu.add_command(label="Save", command=self.save_file)

file_menu.add_separator()

```
file_menu.add_command(label="Exit", command=self.root.quit)

menu_bar.add_cascade(label="File", menu=file_menu)

# Create a text area

self.text_area = tk.Text(self.root)

self.text_area.pack(fill=tk.BOTH, expand=True)

def new_file(self):

self.text_area.delete('1.0', tk.END)

def open_file(self):

file_path = filedialog.askopenfilename()

if file_path:

with open(file_path, 'r') as file:

self.text_area.delete('1.0', tk.END)

self.text_area.insert(tk.END, file.read())

def save_file(self):

file_path = filedialog.asksaveasfilename()

if file_path:

with open(file_path, 'w') as file:

file.write(self.text_area.get('1.0', tk.END))

if __name__ == '__main__':
```

```
root = tk.Tk()

TextEditor(root)

root.mainloop()
```

This code uses the tkinter library to create a GUI for the text editor. The editor has a menu bar with a File menu that allows the user to create a new file, open an existing file, save the current file, and exit the program. The text area is used to display and edit the contents of the file.

The new_file method clears the contents of the text area, while the open_file method uses the filedialog module to prompt the user to select a file to open. If a file is selected, the contents of the file are read and displayed in the text area. The save_file method prompts the user to select a file to save the current contents of the text area to.

Finally, the code creates a new instance of the TextEditor class and starts the main event loop using the mainloop method of the root window.

PROJECT 42: ALARM CLOCK: DEVELOP AN APPLICATION THAT ALLOWS USERS TO SET ALARMS AND REMINDERS AT SPECIFIC TIMES

```
IMPORT DATETIME

import time

import playsound

def set_alarm(alarm_time):

while True:

time.sleep(1)

now = datetime.datetime.now().strftime("%H:%M:%S")

if now == alarm_time:

print("Wake up!")

playsound.playsound("alarm_sound.mp3") # replace with the path of your alarm sound file

break

if __name__ == '__main__':

alarm_time = input("Set the time for the alarm in 'HH:MM:SS' format: ")

set_alarm(alarm_time)
```

This code defines a function set_alarm() that takes an alarm time in the format of 'HH:MM:SS' as input. The function runs an infinite loop that checks the current time every second, and if the current time matches the alarm time, it prints "Wake up!" and plays an alarm sound using the playsound library.

To use the alarm clock, you can simply call set_alarm() and pass in the desired alarm time as a string. For example, set_alarm('07:30:00') would set the alarm to go off at 7:30 AM. Note that the playsound library requires a sound file to be played, so you'll need to replace "alarm_sound.mp3" with the path to an actual sound file on your computer.

PROJECT 43: RANDOM QUOTE GENERATOR: CREATE A PROGRAM THAT GENERATES RANDOM QUOTES OR PHRASES FROM A DATABASE OF QUOTES

IMPORT RANDOM

List of quotes

quotes = [

"The best way to predict the future is to invent it. - Alan Kay",

"The only way to do great work is to love what you do. - Steve Jobs",

"The greatest glory in living lies not in never falling, but in rising every time we fall. - Nelson Mandela",

"If you want to live a happy life, tie it to a goal, not to people or things. - Albert Einstein",

"I have not failed. I've just found 10,000 ways that won't work. - Thomas Edison",

"Believe you can and you're halfway there. - Theodore Roosevelt"

]

Function to generate a random quote

def generate_quote():

quote = random.choice(quotes)

```
return quote

# Main function

if __name__ == '__main__':

print(generate_quote())
```

This code defines a list of quotes and a function generate_quote() that randomly selects a quote from the list using the random module. When the code is run, generate_quote() is called and the resulting quote is printed to the console.

To add more quotes to the generator, simply add them to the quotes list. You can also customize the format of the quotes by modifying the string values in the list.

PROJECT 44: CALENDAR APP: DEVELOP AN APPLICATION THAT DISPLAYS A MONTHLY CALENDAR AND ALLOWS USERS TO ADD EVENTS AND REMINDERS

IMPORT CALENDAR

class Calendar:

def __init__(self, year, month):

self.year = year

self.month = month

def display_calendar(self):

create a calendar object

cal = calendar.monthcalendar(self.year, self.month)

print the calendar header

print(calendar.month_name[self.month] + " " + str(self.year))

print("Mo Tu We Th Fr Sa Su")

print each week of the calendar

for week in cal:

week_str = ""

for day in week:

```
if day == 0:

week_str += "  "

else:

week_str += "{:2d}".format(day) + " "

print(week_str)

def add_event(self, day, event):

# add the event to the calendar

cal = calendar.Calendar()

for d in cal.itermonthdates(self.year, self.month):

if d.day == day:

print("Added event '" + event + "' on " + str(d))

break

if __name__ == '__main__':

# create a calendar object for March 2023

cal = Calendar(2023, 3)

# display the calendar for March 2023

cal.display_calendar()

# add an event to March 15, 2023

cal.add_event(15, "Meeting with clients")
```

This code defines a Calendar class with two methods: display_calendar() and add_event(). The display_calendar() method creates a calendar object using the calendar.monthcalendar() function and prints it to the console. The add_event() method adds an event to the calendar on a specific day, using the calendar.Calendar().itermonthdates() function to iterate over the dates in the month.

To use the calendar app, simply create a Calendar object for the desired year and month, and then call the display_calendar() method to show the calendar. To add an event, call the add_event() method with the day and event name as arguments.

Note that this example code only allows for adding events to specific days; for a more robust calendar app, you may want to consider adding features such as recurring events, reminders, and alerts.

PROJECT 45: ROCK-PAPER-SCISSORS GAME: BUILD A SIMPLE ROCK-PAPER-SCISSORS GAME THAT ALLOWS TWO PLAYERS TO PLAY AGAINST EACH OTHER

```
PRINT("LET'S PLAY ROCK-Paper-Scissors!")

player1 = input("Player 1, please enter your choice (rock/paper/scissors): ")

player2 = input("Player 2, please enter your choice (rock/paper/scissors): ")

if player1 == player2:

print("It's a tie!")

elif player1 == 'rock':

if player2 == 'scissors':

print("Player 1 wins!")

else:

print("Player 2 wins!")

elif player1 == 'paper':

if player2 == 'rock':

print("Player 1 wins!")

else:

print("Player 2 wins!")
```

elif player1 == 'scissors':

if player2 == 'paper':

print("Player 1 wins!")

else:

print("Player 2 wins!")

else:

print("Invalid input! You must enter 'rock', 'paper', or 'scissors'.")

This code prompts both players to enter their choices (rock, paper, or scissors) using the input() function, and then uses a series of conditional statements to determine the winner based on the classic rock-paper-scissors rules.

To play the game, simply run the code and follow the prompts to enter each player's choice. The program will then display the winner or declare a tie. Note that this implementation assumes that both players are human, but you could easily modify it to accept randomized computer choices for a single-player version of the game.

PROJECT 46: COUNTDOWN TIMER: CREATE A PROGRAM THAT COUNTS DOWN FROM A SPECIFIED TIME AND DISPLAYS A MESSAGE WHEN THE TIME IS UP

IMPORT TIME

def countdown_timer(seconds):

print("Countdown timer started!")

for i in range(seconds, 0, -1):

print(i)

time.sleep(1)

print("Time's up!")

Example usage: countdown_timer(10) for a 10-second countdown

This code defines a countdown_timer() function that takes a number of seconds as an argument and counts down from that number to zero, using the range() function and a for loop. The time.sleep() function is used to pause the program for one second between each count, so that the countdown proceeds in real time. Finally, the function prints a message when the countdown is complete.

To use the countdown timer, simply call the countdown_timer() function with the desired number of seconds as an argument. For example, countdown_timer(10) would start a 10-second countdown. You could modify this code to display a message or perform a specific action when the timer reaches zero, such as playing a sound or launching another program.

PROJECT 47: CONTACT MANAGEMENT SYSTEM: DEVELOP AN APPLICATION THAT ALLOWS USERS TO STORE AND MANAGE CONTACTS WITH THEIR NAME, PHONE NUMBER, AND EMAIL ADDRESS

CLASS CONTACT:

def __init__(self, name, phone_number, email):

self.name = name

self.phone_number = phone_number

self.email = email

class ContactManager:

def __init__(self):

self.contacts = []

def add_contact(self, name, phone_number, email):

contact = Contact(name, phone_number, email)

self.contacts.append(contact)

print(f"{name} has been added to your contacts.")

def display_contacts(self):

if not self.contacts:

```
print("You have no contacts.")

else:

print("Your contacts:")

for contact in self.contacts:

print(f"Name: {contact.name}")

print(f"Phone number: {contact.phone_number}")

print(f"Email: {contact.email}")

print("-" * 20)

def search_contact(self, name):

for contact in self.contacts:

if contact.name.lower() == name.lower():

print(f"Name: {contact.name}")

print(f"Phone number: {contact.phone_number}")

print(f"Email: {contact.email}")

return

print(f"{name} not found in your contacts.")

def delete_contact(self, name):

for contact in self.contacts:

if contact.name.lower() == name.lower():
```

self.contacts.remove(contact)

print(f"{name} has been removed from your contacts.")

return

print(f"{name} not found in your contacts.")

Example usage:

contact_manager = ContactManager()

contact_manager.add_contact("John Smith", "123-456-7890", "john.smith@example.com")

contact_manager.add_contact("Jane Doe", "555-555-5555", "jane.doe@example.com")

contact_manager.display_contacts()

contact_manager.search_contact("John Smith")

contact_manager.search_contact("Bob Johnson")

This code defines a Contact class that represents an individual contact, with attributes for name, phone number, and email address. It also defines a ContactManager class that manages a list of contacts and provides methods for adding, displaying, searching, and deleting contacts.

To use the contact management system, you would create an instance of the ContactManager class and then call its methods to add, display, search, and delete contacts. The example usage code above demonstrates how to add two contacts, display all contacts, search for a specific contact by name (both found and not found), delete a contact, and display the updated list of contacts.

You could extend this code to include additional features, such as editing existing contacts, sorting contacts by name or phone number, or exporting contacts to a file.

PROJECT 48: MORSE CODE TRANSLATOR: DEVELOP A PROGRAM THAT CAN TRANSLATE TEXT TO MORSE CODE AND VICE VERSA

MORSE_CODE_DICT = {'A': '.-', 'B': '-...', 'C': '-.-.', 'D': '-..', 'E': '.',

'F': '..-.', 'G': '—.', 'H': '....', 'I': '..', 'J': '.—-',

'K': '-.-', 'L': '.-..', 'M': '—', 'N': '-.', 'O': '—-',

'P': '.—.', 'Q': '—.-', 'R': '.-.', 'S': '...', 'T': '-',

'U': '..-', 'V': '...-', 'W': '.—', 'X': '-..-', 'Y': '-.—',

'Z': '—..', '1': '.——', '2': '..—-', '3': '...—',

'4': '....-', '5': '.....', '6': '-....', '7': '—...',

'8': '—-..', '9': '——.', '0': '——-', ', ': '—..—',

'.': '.-.-.-', '?': '..—..', '/': '-..-.', '-': '-....-',

'(': '-.—.', ')': '-.—.-'}

def text_to_morse(text):

morse_code = ''

for char in text.upper():

if char == ' ':

morse_code += '/ '

```
elif char in MORSE_CODE_DICT:

morse_code += MORSE_CODE_DICT[char] + ' '

return morse_code

def morse_to_text(morse):

text = ''

morse_dict = {v: k for k, v in MORSE_CODE_DICT.items()}

for code in morse.split():

if code == '/':

text += ' '

elif code in morse_dict:

text += morse_dict[code]

return text

# Example usage:

text = 'SOS'

morse = text_to_morse(text)

print(f"{text} in Morse code is: {morse}")

decoded_text = morse_to_text(morse)

print(f"{morse} in text is: {decoded_text}")
```

This code defines a dictionary MORSE_CODE_DICT that maps each letter, number, and punctuation mark to its corresponding Morse code representation. The text_to_morse() function takes a string of text as input and uses the dictionary to translate each character to its Morse code equivalent, separated by spaces. The morse_to_text() function performs the reverse translation, taking a string of Morse code as input and using a reversed version of the dictionary to map each code to its corresponding character.

To use the Morse code translator, simply call either the text_to_morse() or morse_to_text() function with the desired input text or Morse code as an argument. The example usage code above demonstrates how to translate the text "SOS" to Morse code and then back to text. You could modify this code to accept user input, read text from a file, or output the results to a file or display them in a graphical user interface.

PROJECT 49: MEMORY GAME: CREATE A GAME THAT CHALLENGES USERS TO MEMORIZE A SEQUENCE OF IMAGES OR NUMBERS AND RECALL THEM IN THE CORRECT ORDER

```
IMPORT RANDOM

def generate_sequence(length):

"""Generate a random sequence of digits of the specified length"""

digits = [str(random.randint(0, 9)) for _ in range(length)]

return digits

def get_user_input(length):

"""Get the user's input for a sequence of digits of the specified length"""

print(f"Enter the {length} digits in the sequence:")

user_input = input("> ")

# Validate user input

if len(user_input) != length or not user_input.isdigit():

print(f"Invalid input. Please enter {length} digits.")

return get_user_input(length)

return user_input
```

```
def play_game():

"""Play a game of memory"""

sequence_length = 4  # You can adjust this to change the difficulty

sequence = generate_sequence(sequence_length)

print("Memorize the following sequence:")

print(" ".join(sequence))

input("Press Enter to continue...")

# Clear the console screen (may not work on all systems)

print("\033c", end="")

user_input = get_user_input(sequence_length)

if user_input == sequence:

print("Congratulations! You correctly remembered the sequence.")

else:

print(f"Sorry, the sequence was {sequence}. You entered {user_input}.")

# Example usage:

play_game()
```

When you run this code, it will generate a random sequence of digits (the length of which you can adjust by modifying the sequence_length variable), display the sequence to the player, and then prompt the player to enter the sequence from memory. After the player enters their input, the code will compare it to the original sequence and display a success or failure message.

You can modify this code to use images instead of numbers (by replacing the generate_sequence() function with a function that generates a random sequence of images) or to display the sequence in a graphical user interface. You could also add additional features, such as a scoring system, a timer, or different difficulty levels.

PROJECT 50: MAZE SOLVER: DEVELOP A PROGRAM THAT CAN SOLVE MAZES AUTOMATICALLY BY FINDING THE SHORTEST PATH FROM THE START TO THE END POINT

FROM QUEUE IMPORT QUEUE

Define the maze as a 2D array of integers, where 0 represents a wall and 1 represents a path

maze = [

[1, 1, 0, 1, 1, 1],

[0, 1, 0, 1, 0, 1],

[0, 1, 1, 1, 0, 1],

[0, 0, 0, 1, 1, 1],

[1, 1, 0, 0, 0, 1],

[0, 0, 0, 1, 1, 1],

[1, 1, 1, 1, 0, 1],

]

Define the starting and ending points of the maze

start = (0, 0)

```
end = (len(maze)-1, len(maze[0])-1)

def find_shortest_path(maze, start, end):

"""Find the shortest path from the start point to the end point in the given maze"""

queue = Queue()

queue.put(start)

visited = set()

visited.add(start)

# Use a dictionary to keep track of the path to each point

# The path to the start point is an empty list

paths = {start: []}

while not queue.empty():

current = queue.get()

if current == end:

# Found the end point

return paths[current]

# Check all the neighbors of the current point

row, col = current

for (r, c) in [(row-1, col), (row+1, col), (row, col-1), (row, col+1)]:
```

```
if r < 0 or c < 0 or r >= len(maze) or c >= len(maze[0]):

# Neighbor is outside the maze

continue

if maze[r][c] == 0:

# Neighbor is a wall

continue

neighbor = (r, c)

if neighbor in visited:

# Neighbor has already been visited

continue

visited.add(neighbor)

queue.put(neighbor)

# Add the current point to the path to the neighbor

paths[neighbor] = paths[current] + [current]

# There is no path from start to end

return None

# Find the shortest path from the start to the end of the maze

path = find_shortest_path(maze, start, end)

# Print the path, if it exists
```

```
if path is not None:

print("Shortest path found:")

for point in path + [end]:

print(point)

else:

print("No path found")
```

This code defines a maze as a 2D array of integers, where 0 represents a wall and 1 represents a path. It also defines the starting and ending points of the maze. The find_shortest_path() function uses a breadth-first search algorithm to find the shortest path from the start point to the end point. It returns the path as a list of points.

When you run this code, it will output the shortest path from the start point to the end point, if it exists. If there is no path from the start to the end point, it will print a "No path found" message.

Note that this code assumes that there is only one possible path from the start to the end point.

PROJECT 51: INTERACTIVE MAP: CREATE A MAP APPLICATION THAT ALLOWS USERS TO SEARCH FOR LOCATIONS, GET DIRECTIONS, AND EXPLORE POINTS OF INTEREST

CREATING AN INTERACTIVE map with search functionality, directions, and points of interest requires integration with a map API like Google Maps or Mapbox. Here's an example Python code that uses the Google Maps API to search for a location, get directions, and explore points of interest:

```
import googlemaps

from datetime import datetime

# Replace YOUR_API_KEY with your own API key

gmaps = googlemaps.Client(key='YOUR_API_KEY')

# Search for a location

location = gmaps.geocode('New York City')[0]['geometry']['location']

print("Latitude: {}, Longitude: {}".format(location['lat'], location['lng']))

# Get directions between two locations

directions = gmaps.directions('New York City', 'Washington, DC', mode="driving", departure_time=datetime.now())

for step in directions[0]['legs'][0]['steps']:

print(step['html_instructions'])
```

Search for points of interest near a location

places = gmaps.places_nearby(location, radius=5000, type='restaurant')

for place in places['results']:

print(place['name'], place['vicinity'])

This code uses the googlemaps Python library to interact with the Google Maps API. It first searches for the latitude and longitude of "New York City". It then uses the directions() method to get driving directions from "New York City" to "Washington, DC", and prints the instructions for each step of the directions.

Finally, it uses the places_nearby() method to search for restaurants near the "New York City" location, within a radius of 5000 meters. It then prints the name and address of each restaurant.

Note that you will need to obtain an API key from the map provider and replace YOUR_API_KEY with your own API key in order to use this code. Additionally, the output format may vary depending on the map provider's API.

PROJECT 52: VIDEO DOWNLOADER: CREATE A PROGRAM THAT CAN DOWNLOAD AND SAVE VIDEOS FROM DIFFERENT WEBSITES, SUCH AS YOUTUBE OR VIMEO

HERE'S AN EXAMPLE PYTHON code that uses the pytube library to download videos from YouTube:

```
from pytube import YouTube

# Replace the URL with the YouTube video you want to download

url = "https://www.youtube.com/watch?v=dQw4w9WgXcQ"

# Create a YouTube object

yt = YouTube(url)

# Get the first stream (highest resolution) of the video

stream = yt.streams.first()

# Download the video

stream.download()
```

This code uses the pytube library to interact with YouTube and download a video. It first creates a YouTube object with the URL of the video. It then gets the first stream (which is typically the highest resolution stream) using the first() method, and downloads the video using the download() method.

Note that some websites may have restrictions on downloading their videos, and it may be against their terms of service. Additionally, downloading videos without the owner's permission may be illegal in some cases, so be sure to check local laws and regulations before using this code.

PROJECT 53: CHAT APPLICATION: BUILD A SIMPLE CHAT APPLICATION THAT ALLOWS USERS TO SEND AND RECEIVE MESSAGES IN REAL-TIME

FROM FLASK IMPORT FLASK, render_template

from flask_socketio import SocketIO, emit

app = Flask(__name__)

app.config['SECRET_KEY'] = 'secret!'

socketio = SocketIO(app)

A list to store all the messages

messages = []

A function to handle incoming messages

@socketio.on('message')

def handle_message(message):

messages.append(message)

emit('message', message, broadcast=True)

A route to display the chat interface

@app.route('/')

def index():

```
return render_template('index.html', messages=messages)

if __name__ == '__main__':

socketio.run(app)
```

This code uses the Flask and Socket.IO libraries to create a web server that handles incoming and outgoing messages. It first creates a Flask application and a Socket.IO object. It then defines a function handle_message() that receives incoming messages and appends them to the messages list. The function then broadcasts the message to all connected clients using the emit() method.

It also defines a route / that displays a simple HTML template index.html with a form for sending messages and a list of all the messages stored in the messages list.

To run this code, you will need to install Flask and Socket.IO using pip. You can then run the code with python app.py and access the chat interface by opening a web browser and navigating to http://localhost:5000/.

PROJECT 54: PASSWORD STRENGTH CHECKER: CREATE A PROGRAM THAT CAN EVALUATE THE STRENGTH OF A PASSWORD BASED ON FACTORS LIKE LENGTH, COMPLEXITY, AND UNIQUENESS

IMPORT RE

def password_strength(password):

Check the length of the password

if len(password) < 8:

return "Password is too short."

Check the complexity of the password

pattern = r"(?=.*\d)(?=.*[a-z])(?=.*[A-Z])(?=.*[@#$%^&+=])"

if not re.match(pattern, password):

return "Password is not complex enough."

Check the uniqueness of the password

if password.lower() == password or password.upper() == password:

return "Password is not unique enough."

return "Password is strong."

Example usage

```
print(password_strength("password123"))

print(password_strength("p@ssw0rd"))

print(password_strength("PASSWORD"))

print(password_strength("MyS3cr3tP@ssw0rd"))
```

This code defines a function password_strength() that takes a password as input and checks its strength based on its length, complexity, and uniqueness. It first checks the length of the password and returns an error message if it's too short. It then uses a regular expression pattern to check the complexity of the password, making sure it contains at least one digit, one lowercase letter, one uppercase letter, and one special character. Finally, it checks the uniqueness of the password by comparing it to its lowercase and uppercase versions, returning an error message if it's not unique enough. To use this code, you can call the password_strength() function with a password as input, and it will return a string indicating whether the password is strong, or if it fails any of the strength criteria.

PROJECT 55: FILE COMPRESSION TOOL: BUILD A PROGRAM THAT CAN COMPRESS AND DECOMPRESS FILES OF DIFFERENT FORMATS, SUCH AS ZIP OR RAR FILES

IMPORT ZIPFILE

Function to compress a file

def compress_file(filename):

with zipfile.ZipFile(filename + '.zip', 'w', compression=zipfile.ZIP_DEFLATED) as zipf:

zipf.write(filename)

Function to decompress a file

def decompress_file(filename):

with zipfile.ZipFile(filename, 'r') as zipf:

zipf.extractall()

Example usage

compress_file('file.txt')

decompress_file('file.txt.zip')

This code defines two functions, compress_file() and decompress_file(). The compress_file() function takes a filename as input and creates a new zip file with the same name and the .zip extension. It then opens the zip file using the ZipFile() function and writes the contents of the input file to the zip file using the write() method.

The decompress_file() function takes a zip filename as input and extracts all the files and directories contained in the zip file to the current directory using the extractall() method.

To use this code, you can call the compress_file() function with the name of the file you want to compress, and it will create a new zip file with the same name and the .zip extension. You can then call the decompress_file() function with the name of the zip file you want to decompress, and it will extract all the files and directories contained in the zip file to the current directory.

PROJECT 56: DATA VISUALIZATION TOOL: CREATE A PROGRAM THAT CAN VISUALIZE DATA IN DIFFERENT FORMATS, SUCH AS CHARTS, GRAPHS, AND MAPS

```
IMPORT MATPLOTLIB.PYPLOT as plt

import numpy as np

# Generate some sample data

x = np.random.normal(0, 1, 100)

y = np.random.normal(0, 1, 100)

# Create a scatter plot

plt.scatter(x, y)

plt.title('Sample Scatter Plot')

plt.xlabel('X axis')

plt.ylabel('Y axis')

# Show the plot

plt.show()
```

This code generates some random data using NumPy and then creates a scatter plot using Matplotlib. The scatter() function takes two arrays as input, one for the x-axis data and one for the y-axis data. The title(), xlabel(), and ylabel() functions are used to set the title and labels for the plot. Finally, the show() function is used to display the plot.

You can customize this code to generate different types of visualizations, such as bar charts, line graphs, and heat maps, by using the appropriate functions from the Matplotlib library. Additionally, you can use other libraries like Seaborn and Plotly to create more advanced visualizations like statistical plots and interactive charts.

PROJECT 57: CODE REPOSITORY: DEVELOP A WEB-BASED PLATFORM THAT ALLOWS USERS TO SHARE AND COLLABORATE ON CODE PROJECTS

BUILDING A WEB-BASED platform for code repository and collaboration involves a lot of components such as front-end web development, back-end server setup, and database integration. Here's an example Python code for a basic version of a code repository:

```
from flask import Flask, render_template, request, redirect, url_for

from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)

app.config['SQLALCHEMY_DATABASE_URI'] =
'sqlite:///code_repository.db'

db = SQLAlchemy(app)

class Code(db.Model):

id = db.Column(db.Integer, primary_key=True)

title = db.Column(db.String(100), nullable=False)

code = db.Column(db.Text, nullable=False)

@app.route('/')

def index():

codes = Code.query.all()
```

```
return render_template('index.html', codes=codes)

@app.route('/new_code', methods=['GET', 'POST'])

def new_code():

if request.method == 'POST':

title = request.form['title']

code = request.form['code']

new_code = Code(title=title, code=code)

db.session.add(new_code)

db.session.commit()

return redirect(url_for('index'))

else:

return render_template('new_code.html')

@app.route('/view_code/<int:id>')

def view_code(id):

code = Code.query.filter_by(id=id).first()

return render_template('view_code.html', code=code)

if __name__ == '__main__':

app.run(debug=True)
```

This code uses the Flask web framework and SQLAlchemy to create a simple code repository. It defines a Code class that maps to a database table with columns for id, title, and code. The index() function retrieves all Code objects from the database and displays them on the home page. The new_code() function allows users to create new code entries and add them to the database. The view_code() function retrieves a single Code object by ID and displays it on a separate page.

To use this code, you would need to create the necessary HTML templates for the web pages (index.html, new_code.html, and view_code.html). Additionally, you would need to set up a database to store the Code objects (in this example, the database is SQLite, but you could use a different database system like MySQL or PostgreSQL). This basic code repository can be expanded with additional features such as user authentication, version control, and code collaboration tools.

PROJECT 58: ENCRYPTION TOOL: BUILD A PROGRAM THAT CAN ENCRYPT AND DECRYPT TEXT DATA USING DIFFERENT ENCRYPTION ALGORITHMS

```
DEF CAESAR_ENCRYPT(text, key):

result = ""

for char in text:

if char.isalpha():

shift = key % 26

if char.isupper():

result += chr((ord(char) + shift - 65) % 26 + 65)

else:

result += chr((ord(char) + shift - 97) % 26 + 97)

else:

result += char

return result

def caesar_decrypt(ciphertext, key):

result = ""

for char in ciphertext:
```

```
if char.isalpha():

shift = key % 26

if char.isupper():

result += chr((ord(char) - shift - 65) % 26 + 65)

else:

result += chr((ord(char) - shift - 97) % 26 + 97)

else:

result += char

return result

# Example usage

text = "Hello, world!"

key = 3

encrypted_text = caesar_encrypt(text, key)

print(f"Encrypted text: {encrypted_text}")

decrypted_text = caesar_decrypt(encrypted_text, key)

print(f"Decrypted text: {decrypted_text}")
```

This code defines two functions, caesar_encrypt() and caesar_decrypt(), which implement the Caesar cipher algorithm for encrypting and decrypting text, respectively. The caesar_encrypt() function takes a plaintext string and a key (an integer representing the number of positions to shift each letter) as input, and returns the encrypted text. The caesar_decrypt() function takes a ciphertext string and the same key as input, and returns the decrypted text.

To use the code, you can call the caesar_encrypt() function with a plaintext string and a key, and it will return the encrypted text. You can then call the caesar_decrypt() function with the encrypted text and the same key, and it will return the original plaintext.

PROJECT 59: TIME TRACKING TOOL: A PROGRAM THAT ALLOWS USERS TO TRACK AND ANALYZE TIME SPENT ON DIFFERENT TASKS OR PROJECTS USING AN SQLITE DATABASE, DATETIME LIBRARY, AND PANDAS LIBRARY

```
IMPORT SQLITE3

import pandas as pd

from datetime import datetime

# Connect to the database

conn = sqlite3.connect('timetracker.db')

c = conn.cursor()

# Create the table to store the time logs

c.execute('''CREATE TABLE IF NOT EXISTS timelogs

(id INTEGER PRIMARY KEY AUTOINCREMENT,

task TEXT,

project TEXT,

start_time TEXT,

end_time TEXT)''')
```

```
# Function to insert time logs

def insert_log(task, project, start_time, end_time):

c.execute("INSERT INTO timelogs (task, project, start_time, end_time)
VALUES (?, ?, ?, ?)",

(task, project, start_time, end_time))

conn.commit()

# Function to get time logs as a pandas DataFrame

def get_logs():

df = pd.read_sql_query("SELECT * FROM timelogs", conn)

return df

# Function to calculate the time spent on each task

def calculate_time(df):

df['start_time'] = pd.to_datetime(df['start_time'], format='%Y-%m-%d
%H:%M:%S.%f')

df['end_time'] = pd.to_datetime(df['end_time'], format='%Y-%m-%d
%H:%M:%S.%f')

df['time_spent'] = df['end_time'] - df['start_time']

return df

# Function to display time logs as a pandas DataFrame

def display_logs():
```

```
df = get_logs()

df = calculate_time(df)

print(df)

# Function to filter time logs by task

def filter_by_task(task):

df = get_logs()

df = df[df['task'] == task]

df = calculate_time(df)

print(df)

# Function to filter time logs by project

def filter_by_project(project):

df = get_logs()

df = df[df['project'] == project]

df = calculate_time(df)

print(df)

# Function to generate a report of time usage by project

def generate_report():

df = get_logs()

df = calculate_time(df)
```

```
report = df.groupby(['project'])['time_spent'].sum()

print(report)

# Function to start a new task

def start_task():

task = input("Enter task name: ")

project = input("Enter project name: ")

start_time = datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f')

end_time = ""

insert_log(task, project, start_time, end_time)

print("Started task: ", task)

# Function to end the current task

def end_task():

end_time = datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f')

c.execute("UPDATE timelogs SET end_time = ? WHERE end_time = ''",
(end_time,))

conn.commit()

print("Ended current task")

# Main menu

while True:
```

```
print("1. Start task")

print("2. End task")

print("3. Display time logs")

print("4. Filter by task")

print("5. Filter by project")

print("6. Generate report")

print("7. Quit")

choice = input("Enter choice: ")

if choice == '1':

start_task()

elif choice == '2':

end_task()

elif choice == '3':

display_logs()

elif choice == '4':

task = input("Enter choice: ")
```

PROJECT 60: DIGITAL SIGNATURE TOOL: CREATE A PROGRAM THAT CAN CREATE AND VERIFY DIGITAL SIGNATURES FOR DOCUMENTS AND FILES

THE FOLLOWING CODE that can create and verify digital signatures for documents and files using the hashlib and hmac libraries:

```
import hashlib

import hmac

def create_signature(file_path, key):

"""

Create a digital signature for a given file using a given key.

Args:

file_path (str): The path to the file to be signed.

key (str): The key to use for signing the file.

Returns:

str: The digital signature as a hexadecimal string.

"""

# Read the file contents

with open(file_path, 'rb') as f:
```

file_contents = f.read()

Hash the file contents using SHA-256

hashed_contents = hashlib.sha256(file_contents).digest()

Generate a HMAC-SHA256 signature using the key and hashed file contents

signature = hmac.new(key.encode(), hashed_contents, hashlib.sha256).hexdigest()

return signature

def verify_signature(file_path, key, signature):

"""

Verify the digital signature of a given file using a given key.

Args:

file_path (str): The path to the file to be verified.

key (str): The key used for signing the file.

signature (str): The digital signature to be verified.

Returns:

bool: True if the signature is valid, False otherwise.

"""

Read the file contents

with open(file_path, 'rb') as f:

file_contents = f.read()

Hash the file contents using SHA-256

hashed_contents = hashlib.sha256(file_contents).digest()

Generate a HMAC-SHA256 signature using the key and hashed file contents

generated_signature = hmac.new(key.encode(), hashed_contents, hashlib.sha256).hexdigest()

Compare the generated signature with the provided signature

return hmac.compare_digest(generated_signature, signature)

To create a digital signature for a file, you can call the create_signature function with the file path and the key:

signature = create_signature('file.txt', 'my-secret-key')

This will generate a digital signature for the file file.txt using the key my-secret-key.

To verify the digital signature of a file, you can call the verify_signature function with the file path, the key, and the signature:

is_valid = verify_signature('file.txt', 'my-secret-key', signature)

This will verify the digital signature of the file file.txt using the key my-secret-key and the signature generated earlier. The function will return True if the signature is valid, False otherwise.

PROJECT 61: MACHINE LEARNING MODEL TRAINER: DEVELOP A PROGRAM THAT CAN TRAIN AND OPTIMIZE MACHINE LEARNING MODELS ON DIFFERENT DATASETS

```
FROM SKLEARN.MODEL_selection import GridSearchCV

from sklearn.metrics import accuracy_score

from sklearn import datasets

from sklearn import svm

from sklearn import tree

from sklearn import ensemble

# Load a dataset

dataset = datasets.load_iris()

X = dataset.data

y = dataset.target

# Define the models to be trained

models = [

{

'name': 'Support Vector Machine',
```

```
'model': svm.SVC(),
'params': {
'kernel': ['linear', 'rbf', 'sigmoid'],
'C': [0.1, 1, 10],
'gamma': [0.1, 1, 'scale']
}
},
{
'name': 'Decision Tree',
'model': tree.DecisionTreeClassifier(),
'params': {
'criterion': ['gini', 'entropy'],
'max_depth': [None, 5, 10],
'min_samples_split': [2, 5, 10]
}
},
{
'name': 'Random Forest',
'model': ensemble.RandomForestClassifier(),
```

```
'params': {

'n_estimators': [10, 50, 100],

'max_depth': [None, 5, 10],

'min_samples_split': [2, 5, 10]

}

}

]

# Train and optimize the models

for model in models:

print('Training', model['name'])

clf = GridSearchCV(model['model'], model['params'], cv=5)

clf.fit(X, y)

print('Best parameters:', clf.best_params_)

print('Accuracy:', accuracy_score(y, clf.predict(X)))
```

This code trains and optimizes three different machine learning models (Support Vector Machine, Decision Tree, and Random Forest) on the iris dataset. For each model, the code uses GridSearchCV to perform a grid search over a range of hyperparameters and find the best combination of hyperparameters that maximizes the accuracy of the model.

To use this code with a different dataset, you can simply replace the X and y variables with the features and labels of your dataset. You can also add or remove models to be trained by modifying the models list.

Note that this code assumes that the dataset is already preprocessed and split into training and testing sets. If your dataset is not split, you may want to use train_test_split from sklearn.model_selection to split the dataset before training the models.

PROJECT 62: EMAIL AUTOMATION TOOL: CREATE A PROGRAM THAT CAN AUTOMATE EMAIL TASKS, SUCH AS SENDING PERSONALIZED MESSAGES OR SCHEDULING FOLLOW-UPS

IMPORT SMTPLIB

from email.mime.text import MIMEText

from email.mime.multipart import MIMEMultipart

from email.mime.image import MIMEImage

from datetime import datetime, timedelta

def send_email(sender_email, sender_password, receiver_email, subject, body, image_path=None):

"""

Send an email with an optional image attachment.

Args:

sender_email (str): The email address of the sender.

sender_password (str): The password of the sender's email account.

receiver_email (str): The email address of the receiver.

subject (str): The subject line of the email.

body (str): The body text of the email.

image_path (str, optional): The path to an image file to attach to the email.

Returns:

bool: True if the email was sent successfully, False otherwise.

"""

Create a MIME message

message = MIMEMultipart()

message['From'] = sender_email

message['To'] = receiver_email

message['Subject'] = subject

Add the body text to the message

message.attach(MIMEText(body, 'plain'))

Add an image attachment to the message, if provided

if image_path is not None:

with open(image_path, 'rb') as f:

image = MIMEImage(f.read())

message.attach(image)

Connect to the SMTP server and send the message

try:

server = smtplib.SMTP('smtp.gmail.com', 587)

server.starttls()

server.login(sender_email, sender_password)

server.sendmail(sender_email, receiver_email, message.as_string())

server.quit()

return True

except:

return False

def schedule_email(sender_email, sender_password, receiver_email, subject, body, image_path=None, send_time=None):

"""

Schedule an email to be sent at a specified time.

Args:

sender_email (str): The email address of the sender.

sender_password (str): The password of the sender's email account.

receiver_email (str): The email address of the receiver.

subject (str): The subject line of the email.

body (str): The body text of the email.

image_path (str, optional): The path to an image file to attach to the email.

send_time (datetime, optional): The time at which to send the email.

Returns:

bool: True if the email was scheduled successfully, False otherwise.

"""

Calculate the delay until the specified send time

if send_time is not None:

delay = (send_time - datetime.now()).total_seconds()

if delay < 0:

return False

else:

delay = 0

Schedule the email to be sent after the specified delay

try:

import sched

s = sched.scheduler(datetime.now().timestamp, time.sleep)

s.enter(delay, 1, send_email, (sender_email, sender_password, receiver_email, subject, body, image_path))

s.run()

return True

except:

return False

This code provides two functions: send_email and schedule_email.

The send_email function sends an email with an optional image attachment. You can call this function with the sender's email address and password, the receiver's email address, the subject line of the email, the body text of the email, and the path to an optional image file to attach to the email:

send_email('sender@gmail.com', 'password

PROJECT 63: MIND MAPPING TOOL: BUILD A PROGRAM THAT CAN CREATE AND VISUALIZE MIND MAPS, WHICH ARE DIAGRAMS THAT REPRESENT IDEAS OR CONCEPTS

THE FOLLOWING CODE can create and visualize mind maps using the networkx and matplotlib libraries:

import networkx as nx

import matplotlib.pyplot as plt

def create_mind_map(root, branches):

"""

Create a mind map with the specified root node and branches.

Args:

root (str): The label for the root node of the mind map.

branches (list of tuples): A list of tuples representing the branches of the mind map.

Each tuple should contain two elements: the label for the branch node, and the label

for the parent node of the branch node.

Returns:

networkx.Graph: A graph object representing the mind map.

""""

Create an empty graph for the mind map

mind_map = nx.Graph()

Add the root node to the graph

mind_map.add_node(root)

Add the branch nodes to the graph and connect them to their parent nodes

for branch, parent in branches:

mind_map.add_node(branch)

mind_map.add_edge(parent, branch)

return mind_map

def visualize_mind_map(mind_map):

""""

Visualize a mind map using the networkx and matplotlib libraries.

Args:

mind_map (networkx.Graph): A graph object representing the mind map.

""""

Set the layout of the nodes in the mind map

pos = nx.spring_layout(mind_map, seed=42)

```
# Draw the nodes and edges of the mind map

nx.draw_networkx_nodes(mind_map, pos, node_color='lightblue',
node_size=1500, alpha=0.9)

nx.draw_networkx_edges(mind_map, pos, edge_color='grey', width=2,
alpha=0.7)

# Draw the labels for the nodes in the mind map

nx.draw_networkx_labels(mind_map, pos, font_size=16,
font_family='serif', font_weight='bold')

# Show the mind map in a plot

plt.axis('off')

plt.show()
```

This code provides two functions: create_mind_map and visualize_mind_map.

The create_mind_map function creates a mind map with the specified root node and branches. You can call this function with the label for the root node and a list of tuples representing the branches of the mind map. Each tuple should contain two elements: the label for the branch node, and the label for the parent node of the branch node.

For example, to create a mind map with the root node "Python" and two branches labeled "Data Science" and "Web Development" connected to the root node, you can call the function like this:

```
root = 'Python'

branches = [('Data Science', 'Python'), ('Web Development', 'Python')]
```

mind_map = create_mind_map(root, branches)

The visualize_mind_map function visualizes the mind map using the networkx and matplotlib libraries. You can call this function with the graph object representing the mind map returned by the create_mind_map function:

visualize_mind_map(mind_map)

This will display the mind map in a plot using the specified layout, node colors, edge colors, font sizes, and font families.

PROJECT 64: EMAIL SPAM FILTER: DEVELOP A PROGRAM THAT CAN FILTER OUT UNWANTED OR SPAM EMAILS FROM A USER'S INBOX

```
IMPORT OS

import re

import nltk

from nltk.tokenize import word_tokenize

from nltk.corpus import stopwords

from nltk.stem import PorterStemmer

from sklearn.naive_bayes import MultinomialNB

from sklearn.feature_extraction.text import CountVectorizer

from sklearn.model_selection import train_test_split

nltk.download('stopwords')

nltk.download('punkt')

# Load spam and ham emails

spam_path = 'path_to_spam_folder'

ham_path = 'path_to_ham_folder'

spam_emails = []
```

```
for file in os.listdir(spam_path):

with open(os.path.join(spam_path, file), 'r') as f:

spam_emails.append(f.read())

ham_emails = []

for file in os.listdir(ham_path):

with open(os.path.join(ham_path, file), 'r') as f:

ham_emails.append(f.read())

# Preprocess emails

def preprocess(text):

text = re.sub(r'[^\w\s]', '', text.lower())

tokens = word_tokenize(text)

tokens = [token for token in tokens if token not in
stopwords.words('english')]

stemmer = PorterStemmer()

tokens = [stemmer.stem(token) for token in tokens]

return ' '.join(tokens)

spam_emails = [preprocess(email) for email in spam_emails]

ham_emails = [preprocess(email) for email in ham_emails]

# Create feature vectors
```

```
vectorizer = CountVectorizer()

X = vectorizer.fit_transform(spam_emails + ham_emails)

y = [1]*len(spam_emails) + [0]*len(ham_emails)

# Train test split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Train Naive Bayes model

clf = MultinomialNB()

clf.fit(X_train, y_train)

# Evaluate model on test set

score = clf.score(X_test, y_test)

print(f"Accuracy: {score*100:.2f}%")

# Test the model on new emails

new_emails = ['Get a free iPhone now!', 'Hello, how are you doing today?']

new_emails = [preprocess(email) for email in new_emails]

new_X = vectorizer.transform(new_emails)

predictions = clf.predict(new_X)

print(predictions)
```

This code first loads spam and ham emails from their respective folders and preprocesses them by removing punctuation, stop words, and applying stemming. Then, the emails are converted into feature vectors using the CountVectorizer class from the scikit-learn library.

After that, the emails are split into training and testing sets using train_test_split function. Then, the Naive Bayes model is trained on the training set and evaluated on the testing set.

Finally, the model is tested on new emails and their predictions are printed.

PROJECT 65: BARCODE SCANNER: BUILD A PROGRAM THAT CAN SCAN AND DECODE BARCODES, WHICH CAN BE USED FOR INVENTORY MANAGEMENT OR PRODUCT TRACKING

```
IMPORT CV2

from pyzbar import pyzbar

# Load image from file

image_path = 'path_to_image'

image = cv2.imread(image_path)

# Convert image to grayscale

gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Find barcodes in the image

barcodes = pyzbar.decode(gray)

# Loop over detected barcodes

for barcode in barcodes:

# Extract barcode data and type

barcode_data = barcode.data.decode("utf-8")

barcode_type = barcode.type
```

```
# Print barcode data and type

print(f"Barcode data: {barcode_data}, Barcode type: {barcode_type}")

# Draw barcode rectangle on the image

(x, y, w, h) = barcode.rect

cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)

# Display image with barcodes

cv2.imshow("Barcode Scanner", image)

cv2.waitKey(0)
```

This code uses the ZBar library to detect and decode barcodes from an image. First, the image is loaded and converted to grayscale. Then, the decode() function of the pyzbar library is used to find barcodes in the image.

The code loops over all the detected barcodes and extracts their data and type. Then, it draws a green rectangle around each barcode on the image. Finally, the image is displayed with the detected barcodes.

Note: You will need to install the pyzbar and opencv-python libraries to run this code. You can install them using pip:

```
pip install pyzbar opencv-python
```

PROJECT 66: WORKOUT TRACKER: DEVELOP A PROGRAM THAT CAN TRACK AND ANALYZE WORKOUT SESSIONS, INCLUDING EXERCISES, SETS, AND REPETITIONS

```
IMPORT CSV

from datetime import datetime

def main():

print("Welcome to Workout Tracker!")

while True:

print("\nWhat would you like to do?")

print("1. Add a new workout")

print("2. View workout history")

print("3. Exit")

choice = input("Enter your choice (1-3): ")

if choice == "1":

add_workout()

elif choice == "2":

view_workout_history()
```

```
elif choice == "3":

break

else:

print("Invalid choice. Try again.")

def add_workout():

exercise = input("\nEnter the exercise name: ")

sets = int(input("Enter the number of sets: "))

reps = int(input("Enter the number of reps per set: "))

weight = float(input("Enter the weight used (in kg): "))

date = datetime.now().strftime("%Y-%m-%d")

time = datetime.now().strftime("%H:%M:%S")

workout = [date, time, exercise, sets, reps, weight]

with open("workout_history.csv", mode="a", newline="") as file:

writer = csv.writer(file)

writer.writerow(workout)

print("Workout added successfully!")

def view_workout_history():

print("\nWorkout History:")

with open("workout_history.csv", mode="r") as file:
```

```
reader = csv.reader(file)

for row in reader:

print(row[0], row[1], row[2], "x", row[3], "x", row[4], " @", row[5], "kg")

if __name__ == "__main__":

main()
```

The program uses the csv module to store workout data in a CSV file. The add_workout() function prompts the user to enter details about their workout and saves them to the CSV file. The view_workout_history() function reads the CSV file and displays a history of all workouts in a readable format.

To use the program, simply run the script in a Python environment and follow the prompts to add or view workout history. The workout data will be stored in a file named workout_history.csv in the same directory as the script.

PROJECT 67: TIME ZONE CONVERTER: BUILD A PROGRAM THAT CAN CONVERT TIME BETWEEN DIFFERENT TIME ZONES, USING GEOGRAPHIC LOCATION DATA

```
IMPORT PYTZ

from tzwhere import tzwhere

from datetime import datetime

# Initialize tzwhere object

tz = tzwhere()

def convert_timezone(datetime_str, from_tz, to_tz):

# Get latitude and longitude from the location data

lat, lng = tz.tzNameAt(float(from_tz.split(',')[0]), float(from_tz.split(',')[1]))

# Convert string to datetime object

datetime_obj = datetime.strptime(datetime_str, '%Y-%m-%d %H:%M:%S')

# Get the timezone object for the original timezone

from_timezone = pytz.timezone(tz.tzNameAt(lat, lng))

# Convert datetime object to the original timezone

from_datetime = from_timezone.localize(datetime_obj)

# Get the timezone object for the target timezone
```

```
to_timezone = pytz.timezone(to_tz)

# Convert datetime object to the target timezone

to_datetime = from_datetime.astimezone(to_timezone)

# Format the output datetime string

output_str = to_datetime.strftime('%Y-%m-%d %H:%M:%S %Z%z')

return output_str

# Example usage

datetime_str = '2023-02-22 09:00:00'

from_tz = '40.7128,-74.0060'  # New York City

to_tz = 'Asia/Tokyo'

converted_time = convert_timezone(datetime_str, from_tz, to_tz)

print("Converted time: ", converted_time)
```

In this code, we use the tzwhere library to determine the latitude and longitude of the source time zone based on geographic location data. Then, we use the pytz library to convert the time from the source time zone to the target time zone.

The convert_timezone function takes three arguments: datetime_str, which is a string representing the date and time in the source time zone; from_tz, which is a string representing the geographic location data for the source time zone in the format "latitude,longitude"; and to_tz, which is a string representing the name of the target time zone in the format "Area/Location".

In the example usage code, we convert a datetime string representing 9:00 AM on February 22, 2023 in New York City to the Japan Standard Time (JST) timezone. The output is formatted as a string and printed to the console.

PROJECT 68: STOCK PREDICTION TOOL: DEVELOP A PROGRAM THAT USES MACHINE LEARNING ALGORITHMS TO PREDICT THE FUTURE PERFORMANCE OF A STOCK OR PORTFOLIO

```
IMPORT PANDAS AS PD

import numpy as np

from sklearn.linear_model import LinearRegression

from sklearn.model_selection import train_test_split

import yfinance as yf

# Download stock data

stock = yf.Ticker("AAPL")

df = stock.history(period="max")

# Define input features and target variable

X = df.drop(['Close'], axis=1)

y = df['Close']

# Split the data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=0)
```

```
# Train a linear regression model

model = LinearRegression()

model.fit(X_train, y_train)

# Predict future stock prices

future_dates = pd.date_range(start='2023-02-22', end='2024-02-22',
freq='B')

future_df = pd.DataFrame(index=future_dates, columns=X.columns)

future_df = future_df.fillna(method='ffill')

future_predictions = model.predict(future_df)

# Print the predicted stock prices

print(future_predictions)
```

In this code, we use the yfinance library to download historical stock data for Apple (AAPL) and the pandas library to manipulate and prepare the data. We define the input features as all columns of the historical data except for the "Close" column, which is used as the target variable. We split the data into training and testing sets using the train_test_split function from the sklearn library.

Next, we train a linear regression model using the training data and the fit method of the LinearRegression class from the sklearn library. Finally, we predict future stock prices using a future_df DataFrame containing future dates and the same input features as the training data, and the predict method of the trained model.

Note that this code is a simple example and may not be suitable for accurate stock predictions. Also, there are many other factors that can influence the performance of a stock, such as company news, economic events, and global trends, which are not taken into account in this code.

PROJECT 69: BROWSER EXTENSION: DEVELOP A BROWSER EXTENSION THAT CAN ENHANCE THE FUNCTIONALITY OF POPULAR WEB BROWSERS, SUCH AS CHROME OR FIREFOX

DEVELOPING A BROWSER extension requires different techniques and APIs for different browsers. Here's an example Python code for a basic Chrome extension that opens a new tab with a customized message when the user clicks on the extension icon:

manifest.json:

```
{

"manifest_version": 2,

"name": "My Extension",

"version": "1.0",

"description": "A simple extension",

"icons": {

"16": "icon16.png",

"48": "icon48.png",

"128": "icon128.png"

},
```

```
"browser_action": {

"default_icon": "icon48.png",

"default_popup": "popup.html"

},

"permissions": [

"activeTab"

]

}
```

popup.html:

```
<!DOCTYPE html>

<html>

<head>

<title>My Extension</title>

<style>

body {

font-family: Arial, sans-serif;

font-size: 14px;
```

```
color: #333;

padding: 10px;

}

</style>

</head>

<body>

<h1>Hello, world!</h1>

<p>This is a simple Chrome extension.</p>

</body>

</html>
```

In this code, the manifest.json file is used to define the metadata and behavior of the extension. The browser_action property specifies that the extension has an icon and a popup window. The popup.html file defines the content and style of the popup window.

To load the extension in Chrome, follow these steps:

1. Save the above files in a directory.
2. Open Chrome and navigate to chrome://extensions.
3. Enable developer mode by clicking the toggle switch in the top-right corner.
4. Click on "Load unpacked" and select the directory where the files are saved.
5. After the extension is loaded, you should see its icon in the toolbar. When you click on the icon, a popup window with the message "Hello, world! This is a simple Chrome extension." should appear.

Note that this code is just a basic example and there are many other features and APIs that can be used to create more complex and useful browser extensions.

PROJECT 70: QUANTUM COMPUTING SIMULATIONS: DEVELOP A PROGRAM THAT SIMULATES QUANTUM ALGORITHMS AND CIRCUITS, USING PYTHON LIBRARIES SUCH AS QISKIT AND CIRQ

TO SIMULATE QUANTUM algorithms and circuits in Python, we can use the Qiskit library. Here is an example of a simple quantum circuit simulation:

```
from qiskit import QuantumCircuit, Aer, execute

# Create a 2-qubit quantum circuit

qc = QuantumCircuit(2)

# Apply Hadamard gates to both qubits

qc.h(0)

qc.h(1)

# Apply a controlled-NOT gate between the qubits

qc.cx(0, 1)

# Measure the qubits

qc.measure_all()

# Simulate the circuit using the qasm simulator
```

```
backend = Aer.get_backend('qasm_simulator')

job = execute(qc, backend)

result = job.result()

# Print the measurement outcomes

counts = result.get_counts(qc)

print(counts)
```

In this code, we create a QuantumCircuit object with two qubits, apply a Hadamard gate to each qubit to put them in a superposition state, and then apply a controlled-NOT gate to entangle the qubits. Finally, we measure the qubits and simulate the circuit using the qasm simulator provided by Qiskit's Aer library.

To run the simulation, we create a backend object that represents the simulation backend (in this case, the qasm simulator), and then execute the circuit on the backend using the execute function. The result of the simulation is returned as a result object, which we can use to extract the measurement outcomes.

Note that this is a very simple example that only scratches the surface of what is possible with quantum computing simulations. To simulate more complex algorithms and circuits, we may need to use additional features of Qiskit, such as custom gates, measurements in different bases, and statevector simulations.

PROJECT 71: TRAFFIC SIMULATION: CREATE A PROGRAM THAT SIMULATES TRAFFIC FLOW IN A CITY OR HIGHWAY, USING ALGORITHMS AND STATISTICAL MODELS

SIMULATING TRAFFIC flow in a city or highway requires a complex and detailed simulation model. Here is an example of a simplified traffic simulation using Python and the Mesa library:

```
import random

from mesa import Model, Agent

from mesa.space import MultiGrid

from mesa.time import SimultaneousActivation

class TrafficAgent(Agent):

def __init__(self, unique_id, model):

super().__init__(unique_id, model)

self.speed = random.randint(1, 5)

def move(self):

x, y = self.pos

dx, dy = self.model.directions[self.heading]
```

```
new_pos = ((x + dx) % self.model.grid.width, (y + dy) %
self.model.grid.height)

if self.model.grid.is_cell_empty(new_pos):

self.model.grid.move_agent(self, new_pos)

def step(self):

self.move()

class TrafficModel(Model):

def __init__(self, width, height, num_agents):

self.grid = MultiGrid(width, height, torus=True)

self.schedule = SimultaneousActivation(self)

self.directions = [(1, 0), (0, 1), (-1, 0), (0, -1)]

for i in range(num_agents):

agent = TrafficAgent(i, self)

x = random.randrange(self.grid.width)

y = random.randrange(self.grid.height)

self.grid.place_agent(agent, (x, y))

self.schedule.add(agent)

def step(self):

self.schedule.step()
```

```
model = TrafficModel(50, 50, 200)

for i in range(100):

model.step()
```

In this code, we define a TrafficAgent class that represents a car in the simulation. Each agent has a speed attribute that determines how many cells it moves each step. The move method updates the agent's position based on its current heading and speed, and wraps around the edges of the grid using toroidal boundary conditions.

The TrafficModel class is the main simulation model, which initializes the grid and adds agents to it. The step method updates the state of all agents in a simultaneous activation order.

To run the simulation, we create an instance of the TrafficModel class with a given grid size and number of agents, and call the step method repeatedly to simulate multiple time steps.

Note that this is a very simplified simulation that does not take into account many real-world factors that affect traffic flow, such as traffic lights, intersections, road layouts, and driver behavior. A more realistic traffic simulation would require much more complex algorithms and data.

PROJECT 72: TWITTER BOT: BUILD A PROGRAM THAT CAN INTERACT WITH TWITTER USERS, SUCH AS RETWEETING OR RESPONDING TO TWEETS

IMPORT TWEEPY

import time

Authenticate with Twitter API

auth = tweepy.OAuthHandler("API_KEY", "API_SECRET_KEY")

auth.set_access_token("ACCESS_TOKEN", "ACCESS_TOKEN_SECRET")

Create the API object

api = tweepy.API(auth)

Define keywords to search for

keywords = ['python', 'coding', 'programming']

Define function to retweet and like tweets with specific keywords

def retweet_and_like():

for keyword in keywords:

tweets = tweepy.Cursor(api.search_tweets, q=keyword, tweet_mode='extended').items(10)

for tweet in tweets:

```
try:

# Retweet the tweet

tweet.retweet()

# Like the tweet

tweet.favorite()

print(f"Retweeted and liked tweet with id {tweet.id}")

time.sleep(5)  # Wait for 5 seconds before next tweet

except tweepy.TweepError as e:

print(e.reason)

# Define function to reply to tweets with specific keywords

def reply_to_tweets():

for keyword in keywords:

tweets = api.search_tweets(q=keyword, tweet_mode='extended', count=10)

for tweet in tweets:

try:

# Reply to the tweet

api.update_status(f"@{tweet.user.screen_name} Thanks for tweeting about
{keyword}!", in_reply_to_status_id=tweet.id)

print(f"Replied to tweet with id {tweet.id}")
```

```
time.sleep(5)  # Wait for 5 seconds before next tweet

except tweepy.TweepError as e:

print(e.reason)

# Call the retweet and like function

retweet_and_like()

# Call the reply to tweets function

reply_to_tweets()
```

In this example, we're using the Tweepy library to authenticate with the Twitter API using our API key, API secret key, access token, and access token secret. We then define a list of keywords to search for, and two functions: one to retweet and like tweets with those keywords, and another to reply to tweets with those keywords.

The retweet_and_like function uses the Tweepy Cursor object to search for tweets with the specified keywords and retweets and likes up to 10 of them. The function waits for 5 seconds between each tweet to avoid hitting the Twitter API rate limit.

The reply_to_tweets function searches for tweets with the specified keywords using the search_tweets method and replies to up to 10 of them with a thank-you message. The function also waits for 5 seconds between each tweet.

To use this Twitter bot, you'll need to replace the placeholders for the API key, API secret key, access token, and access token secret with your own credentials, and modify the keywords and messages to suit your needs.

PROJECT 73: MAZE GENERATOR: BUILD A PROGRAM THAT CAN GENERATE RANDOM MAZES OF DIFFERENT SIZES AND COMPLEXITIES

```
IMPORT RANDOM

class Maze:

def __init__(self, rows, cols):

self.rows = rows

self.cols = cols

self.grid = [['wall' for _ in range(cols)] for _ in range(rows)]

def __str__(self):

output = ""

for row in self.grid:

output += ''.join(row) + '\n'

return output

def carve_passages_from(self, row, col):

directions = ['north', 'south', 'east', 'west']

random.shuffle(directions)

for direction in directions:
```

```
if direction == 'north':

if row > 0 and self.grid[row-1][col] == 'wall':

self.grid[row][col] = 'path'

self.grid[row-1][col] = 'path'

self.carve_passages_from(row-1, col)

elif direction == 'south':

if row < self.rows-1 and self.grid[row+1][col] == 'wall':

self.grid[row][col] = 'path'

self.grid[row+1][col] = 'path'

self.carve_passages_from(row+1, col)

elif direction == 'east':

if col < self.cols-1 and self.grid[row][col+1] == 'wall':

self.grid[row][col] = 'path'

self.grid[row][col+1] = 'path'

self.carve_passages_from(row, col+1)

elif direction == 'west':

if col > 0 and self.grid[row][col-1] == 'wall':

self.grid[row][col] = 'path'

self.grid[row][col-1] = 'path'
```

self.carve_passages_from(row, col-1)

def generate(self, start_row, start_col):

self.grid[start_row][start_col] = 'path'

self.carve_passages_from(start_row, start_col)

maze = Maze(10, 10) # specify number of rows and columns for the maze

maze.generate(0, 0) # specify starting row and column

print(maze) # print the maze

This code will generate a maze of the specified size using the depth-first search algorithm, starting at the specified location. The output will be a string representation of the maze, with walls represented by the character "X" and paths represented by the character " ".

PROJECT 74: DATA MINING TOOL: DEVELOP A PROGRAM THAT CAN EXTRACT AND ANALYZE DATA FROM LARGE DATASETS, USING TECHNIQUES SUCH AS CLUSTERING AND ASSOCIATION RULE MINING

```
IMPORT NUMPY AS NP

from sklearn.cluster import KMeans

# generate random data

data = np.random.rand(100, 2)

# set number of clusters

k = 4

# perform clustering

kmeans = KMeans(n_clusters=k).fit(data)

# get cluster labels

labels = kmeans.labels_

# print cluster centers and labels

print('Cluster centers:')

print(kmeans.cluster_centers_)
```

```
print('\nCluster labels:')

print(labels)
```

This code uses the NumPy library to generate a random dataset with 100 points in 2 dimensions. It then uses the scikit-learn library to perform k-means clustering on the data, with 4 clusters. The resulting cluster labels are printed, along with the cluster centers.

Of course, this is just a basic example, and in a real data mining tool you would likely need to do more preprocessing and analysis on the data, as well as choose the appropriate algorithms and parameters for your specific use case.

PROJECT 75: NETWORK SCANNER: BUILD A PROGRAM THAT CAN SCAN A NETWORK FOR CONNECTED DEVICES AND OPEN PORTS, USING NETWORK PROTOCOLS SUCH AS ICMP AND TCP

IMPORT OS

import platform

import subprocess

def get_os_type():

return platform.system()

def ping_ip_address(ip_address):

os_type = get_os_type()

ping_cmd = ["ping", "-n", "1", "-w", "2"] if os_type == "Windows" else ["ping", "-c", "1", "-W", "2"]

ping_cmd.append(ip_address)

result = subprocess.run(ping_cmd, capture_output=True)

if result.returncode == 0:

return True

else:

return False

def scan_network(subnet):

for i in range(1, 256):

ip_address = f"{subnet}.{i}"

if ping_ip_address(ip_address):

print(f"{ip_address} is alive")

scan_network("192.168.1")

This code defines three functions:

1. get_os_type(): This function returns the operating system type of the machine on which the code is running (e.g. "Windows", "Linux", or "Darwin" for macOS).

1. ping_ip_address(ip_address): This function pings a specified IP address using the ping command, and returns True if the ping is successful, or False otherwise.

1. scan_network(subnet): This function scans a subnet for connected devices by calling ping_ip_address() for each IP address in the subnet. It prints a message for each IP address that is found to be alive.

To use this code, simply call the scan_network() function with the subnet you want to scan (in the example above, it's set to "192.168.1"). The function will scan all IP addresses in the subnet, and print a message for each one that responds to a ping.

PROJECT 76: CODE EDITOR: BUILD A CODE EDITOR THAT CAN SUPPORT DIFFERENT PROGRAMMING LANGUAGES, AND INCLUDES FEATURES SUCH AS SYNTAX HIGHLIGHTING AND CODE COMPLETION.

```
IMPORT TKINTER AS TK

import tkinter.scrolledtext as scrolledtext

from tkinter import filedialog

from tkinter import messagebox

import keyword

# Define list of supported languages

LANGUAGES = {

"Python": ".py",

"JavaScript": ".js",

"HTML": ".html",

"CSS": ".css",

}

# Define syntax highlighting colors for each language
```

```
},

SYNTAX_HIGHLIGHTING = {

"Python": {

"keywords": "orange",

"functions": "blue",

"comments": "gray",

"strings": "green",

},

"JavaScript": {

"keywords": "blue",

"functions": "green",

"comments": "gray",

"strings": "red",

},

"HTML": {

"tags": "blue",

"attributes": "green",

"comments": "gray",

"strings": "red",

},
```

```
},

"CSS": {

"selectors": "blue",

"properties": "green",

"comments": "gray",

"strings": "red",

},

}

# Define code completion suggestions for each language

CODE_COMPLETION = {

"Python": {

"range": "range(start, stop[, step])",

"print": "print(*objects, sep=' ', end='\\n', file=sys.stdout, flush=False)",

"input": "input([prompt])",

},

"JavaScript": {

"console.log": "console.log(object)",

"document.getElementById": "document.getElementById(id)",

"function": "function functionName(parameters) { code to be executed }",

},
```

```
"HTML": {
"<html>": "<html>",
"<head>": "<head>",
"<body>": "<body>",
"<h1>": "<h1>",
"<p>": "<p>",
},
"CSS": {
"color": "color: value;",
"background-color": "background-color: value;",
"font-size": "font-size: value;",
},
}
class CodeEditor:
def __init__(self, master, language="Python"):
self.master = master
self.language = language
self.filename = None
self.keywords = keyword.kwlist
```

```
self.color_tags = []

# Create syntax highlighting tags

for tag, color in SYNTAX_HIGHLIGHTING[self.language].items():

self.master.tag_config(tag, foreground=color)

# Create code completion list

self.code_completion_list = tk.Listbox(self.master)

for suggestion in CODE_COMPLETION[self.language]:

self.code_completion_list.insert(tk.END, suggestion)

self.code_completion_list.bind("<Double-Button-1>",
self.insert_suggestion)

# Create widgets

self.text = scrolledtext.ScrolledText(self.master, wrap=tk.WORD)

self.text.bind("<KeyRelease>", self.highlight_syntax)

self.text.bind("<Control-s>", self.save_file)

self.text.bind("<Control-o>", self.open_file)

self.text.bind("<Control-Space>", self.show_code_completion)

self.text.focus_set()

# Pack widgets

self.text.pack(fill=tk.BOTH, expand=True)
```

```
self.code_completion_list.pack_forget()

def highlight_syntax(self, event):

self.text.tag_remove("keywords", "1.0", tk.END)

self.text.tag_remove("functions", "1.0", tk.END)

self.text.tag_remove("comments", "1.0", tk.END)

self.text.tag_remove("strings", "1.0
```

PROJECT 77: TEXT CLASSIFICATION PROGRAM: DEVELOP A PROGRAM THAT CAN CLASSIFY TEXT DATA INTO DIFFERENT CATEGORIES, USING NATURAL LANGUAGE PROCESSING AND MACHINE LEARNING TECHNIQUES

```
# IMPORT NECESSARY LIBRARIES

import pandas as pd

import nltk

from sklearn.feature_extraction.text import CountVectorizer

from sklearn.naive_bayes import MultinomialNB

from sklearn.metrics import accuracy_score

# Load and preprocess data

data = pd.read_csv('text_data.csv')

data['text'] = data['text'].apply(lambda x: " ".join(x.lower() for x in x.split()))  # Convert to lowercase

data['text'] = data['text'].str.replace('[^\w\s]', '')  # Remove punctuation

stop_words = set(nltk.corpus.stopwords.words('english'))

data['text'] = data['text'].apply(lambda x: " ".join(x for x in x.split() if x not in stop_words))  # Remove stop words
```

```
# Split data into training and testing sets

train_data = data.sample(frac=0.8, random_state=1)

test_data = data.drop(train_data.index)

# Vectorize text data

vectorizer = CountVectorizer()

X_train = vectorizer.fit_transform(train_data['text'])

X_test = vectorizer.transform(test_data['text'])

# Train and fit the classifier

classifier = MultinomialNB()

y_train = train_data['category']

classifier.fit(X_train, y_train)

# Make predictions on test data

y_pred = classifier.predict(X_test)

# Evaluate accuracy

accuracy = accuracy_score(test_data['category'], y_pred)

print("Accuracy: {:.2f}%".format(accuracy * 100))
```

This code assumes that you have a CSV file called "text_data.csv" with two columns: "text" and "category". The "text" column contains the text data to be classified, and the "category" column contains the category or label for each text sample.

The program first preprocesses the text data by converting it to lowercase, removing punctuation, and removing stop words. It then splits the data into training and testing sets, and uses the CountVectorizer class from scikit-learn to vectorize the text data.

Next, the program trains a Multinomial Naive Bayes classifier on the training data, using the vectorized text data as input and the category labels as output. It then uses the trained classifier to make predictions on the test data.

Finally, the program evaluates the accuracy of the classifier by comparing the predicted categories to the actual categories in the test data. The accuracy score is printed to the console.

PROJECT 78: CHESS GAME: CREATE A PROGRAM THAT ALLOWS PLAYERS TO PLAY CHESS AGAINST EACH OTHER, AND INCLUDES FEATURES SUCH AS MOVE VALIDATION AND GAME HISTORY

```
CLASS CHESSGAME:

def __init__(self):

self.board = [

["R", "N", "B", "Q", "K", "B", "N", "R"],

["P", "P", "P", "P", "P", "P", "P", "P"],

[" ", " ", " ", " ", " ", " ", " ", " "],

[" ", " ", " ", " ", " ", " ", " ", " "],

[" ", " ", " ", " ", " ", " ", " ", " "],

[" ", " ", " ", " ", " ", " ", " ", " "],

["p", "p", "p", "p", "p", "p", "p", "p"],

["r", "n", "b", "q", "k", "b", "n", "r"],

]

self.player = 1

self.history = []
```

```
def print_board(self):

print("  A B C D E F G H")

for i in range(8):

print(i + 1, end=" ")

for j in range(8):

print(self.board[i][j], end=" ")

print(i + 1)

print("  A B C D E F G H")

def get_move(self):

while True:

move = input(f"Player {self.player}, enter your move (e.g. e2 e4): ")

if move == "quit":

return None

try:

from_pos, to_pos = move.split()

from_col, from_row = ord(from_pos[0]) - 97, int(from_pos[1]) - 1

to_col, to_row = ord(to_pos[0]) - 97, int(to_pos[1]) - 1

if not (0 <= from_row < 8 and 0 <= from_col < 8 and 0 <= to_row < 8 and
0 <= to_col < 8):
```

```
raise ValueError

if self.board[from_row][from_col].islower() and self.player == 1:

raise ValueError

if self.board[from_row][from_col].isupper() and self.player == 2:

raise ValueError

piece = self.board[from_row][from_col]

if piece == "P" and to_row == 0:

promote = input("Choose promotion (Q, R, B, N): ")

piece = promote.upper() + piece[1:]

elif piece == "p" and to_row == 7:

promote = input("Choose promotion (q, r, b, n): ")

piece = promote.lower() + piece[1:]

moves = self.get_valid_moves((from_row, from_col))

if (to_row, to_col) not in moves:

raise ValueError

return (from_row, from_col), (to_row, to_col), piece

except (ValueError, IndexError):

print("Invalid move. Try again.")

def make_move(self, from_pos, to_pos, piece):
```

```
from_row, from_col = from_pos

to_row, to_col = to_pos

self.board[from_row][from_col] = " "

self.board[to_row][to_col] = piece

self.history.append(((from_row, from_col), (to_row, to_col), piece))

self.player = 3 - self.player

def undo_move(self):

if len(self.history) > 0:
```

PROJECT 79: SUDOKU GENERATOR: DEVELOP A PROGRAM THAT CAN GENERATE RANDOM SUDOKU PUZZLES OF DIFFERENT DIFFICULTIES

IMPORT RANDOM

class SudokuGenerator:

def __init__(self):

self.grid = [[0 for i in range(9)] for j in range(9)]

def print_grid(self):

for i in range(9):

for j in range(9):

print(self.grid[i][j], end=" ")

print()

def fill_diagonal(self):

for i in range(0, 9, 3):

for j in range(0, 9, 3):

self.fill_box(i, j)

def fill_box(self, row, col):

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

```
random.shuffle(nums)

for i in range(row, row+3):

for j in range(col, col+3):

self.grid[i][j] = nums.pop()

def is_valid(self, row, col, num):

for i in range(9):

if self.grid[row][i] == num or self.grid[i][col] == num:

return False

box_row = (row // 3) * 3

box_col = (col // 3) * 3

for i in range(box_row, box_row+3):

for j in range(box_col, box_col+3):

if self.grid[i][j] == num:

return False

return True

def fill_remaining(self, i, j):

if j == 9:

i += 1

j = 0
```

```
if i == 9:
return True
if self.grid[i][j] != 0:
return self.fill_remaining(i, j+1)
nums = [k for k in range(1, 10)]
random.shuffle(nums)
for num in nums:
if self.is_valid(i, j, num):
self.grid[i][j] = num
if self.fill_remaining(i, j+1):
return True
self.grid[i][j] = 0
return False
def remove_k_digits(self, k):
cells = [(i, j) for i in range(9) for j in range(9)]
random.shuffle(cells)
for i, j in cells:
temp = self.grid[i][j]
self.grid[i][j] = 0
```

if not self.is_unique_solution():

self.grid[i][j] = temp

k -= 1

if k == 0:

break

def is_unique_solution(self):

grid_copy = [row[:] for row in self.grid]

solver = SudokuSolver(grid_copy)

return solver.solve()

def generate(self, difficulty):

self.fill_diagonal()

self.fill_remaining(0, 3)

self.remove_k_digits(difficulty)

The SudokuGenerator class has several methods:

1. __init__: initializes the Sudoku grid to all zeroes.
2. print_grid: prints the Sudoku grid to the console.
3. fill_diagonal: fills the diagonal boxes of the grid with random numbers.
4. fill_box: fills a 3x3 box with random numbers, used by fill_diagonal.
5. is_valid: checks if a number can be placed at a given row and column without violating any Sudoku rules.

6. fill_remaining: recursively fills the empty cells of the grid with random numbers, starting from a given cell.
7. remove_k_digits: randomly removes

PROJECT 80: ONLINE QUIZ GAME: CREATE A PROGRAM THAT CAN HOST ONLINE QUIZZES, AND ALLOW USERS TO COMPETE WITH EACH OTHER AND TRACK THEIR SCORES

```
import random

class Quiz:

def __init__(self, questions, answers):

self.questions = questions

self.answers = answers

self.num_questions = len(questions)

self.num_players = 0

self.players = {}

self.scores = {}

def add_player(self, name):

self.num_players += 1

self.players[name] = self.num_players

self.scores[name] = 0
```

```
def ask_question(self, question_num):

print(self.questions[question_num])

options = self.answers[question_num].copy()

random.shuffle(options)

for i, option in enumerate(options):

print(f"{i+1}. {option}")

answer = input("Enter your answer: ")

try:

answer = int(answer)

if answer not in range(1, len(options)+1):

print("Invalid input! Please enter a number between 1 and", len(options))

return self.ask_question(question_num)

if options[answer-1] == self.answers[question_num][0]:

return True

else:

return False

except ValueError:

print("Invalid input! Please enter a number.")

return self.ask_question(question_num)
```

```
def play(self):

for name in self.players:

print(f"Welcome {name}! Get ready to play the quiz.")

for i in range(self.num_questions):

print(f"Question {i+1}:")

if self.ask_question(i):

print("Correct!")

self.scores[name] += 1

else:

print("Incorrect.")

print(f"{name}, your score is {self.scores[name]}/{self.num_questions}")

print("Quiz over. Final scores:")

sorted_scores = sorted(self.scores.items(), key=lambda x: x[1],
reverse=True)

for name, score in sorted_scores:

print(f"{name}: {score}/{self.num_questions}")

# Sample usage

questions = ["What is the capital of France?", "What is the smallest country
in the world?", "What is the largest planet in the solar system?"]
```

answers = [["Paris", "London", "Madrid", "Berlin"], ["Vatican City", "Monaco", "Nauru", "San Marino"], ["Jupiter", "Saturn", "Neptune", "Uranus"]]

quiz = Quiz(questions, answers)

quiz.add_player("Alice")

quiz.add_player("Bob")

quiz.play()

The Quiz class has several methods:

1. __init__: initializes the quiz with a list of questions and a list of answers, where each answer is a list of options.
2. add_player: adds a player to the quiz, with a given name.
3. ask_question: asks a player a given question and waits for their input. If the input is valid and matches the correct answer, returns True; otherwise returns False.
4. play: plays the quiz with all the players, asking each of them all the questions and tracking their scores. At the end, prints the final scores in descending order.

To use the Quiz class, you can create an instance with a list of questions and a list of answers, then add players with add_player, and finally call play to start the quiz. The ask_question method will prompt the user for an answer and validate it, then return True if the answer is correct and False otherwise. At the end of the quiz, the play method will print the final scores for all the players.

PROJECT 81: MOVIE TICKET BOOKING SYSTEM: BUILD A PROGRAM THAT CAN ALLOW USERS TO BOOK MOVIE TICKETS ONLINE, AND MANAGE SEATING ARRANGEMENTS AND TICKET SALES

MOVIES = {'THE LION King': 10, 'Toy Story 4': 8, 'Avengers: Endgame': 5}

seating_arrangement = {

'A': [0, 0, 0, 0, 0, 0, 0, 0],

'B': [0, 0, 0, 0, 0, 0, 0, 0],

'C': [0, 0, 0, 0, 0, 0, 0, 0],

'D': [0, 0, 0, 0, 0, 0, 0, 0],

'E': [0, 0, 0, 0, 0, 0, 0, 0],

'F': [0, 0, 0, 0, 0, 0, 0, 0],

'G': [0, 0, 0, 0, 0, 0, 0, 0],

'H': [0, 0, 0, 0, 0, 0, 0, 0]

}

def print_seating_arrangement():

print(' 1 2 3 4 5 6 7 8')

for row, seats in seating_arrangement.items():

```
print(row + ' ' + ' '.join([str(seat) for seat in seats]))

def purchase_ticket():

movie = input('Enter movie name: ')

if movie in movies:

print(f'Ticket price for {movie} is ${movies[movie]}')

print_seating_arrangement()

row = input('Enter row (A-H): ')

seat_num = int(input('Enter seat number (1-8): '))

if seating_arrangement[row][seat_num - 1] == 0:

seating_arrangement[row][seat_num - 1] = 1

print('Ticket purchased successfully!')

movies[movie] -= 1

else:

print('Sorry, seat is already taken.')

else:

print('Movie not found.')

def show_movies():

print('Available movies:')

for movie in movies:
```

```
print('- ' + movie)

while True:

print('\nWelcome to Movie Booking System')

print('1. Show movies')

print('2. Purchase a ticket')

print('3. Exit')

choice = input('Enter your choice (1-3): ')

if choice == '1':

show_movies()

elif choice == '2':

purchase_ticket()

elif choice == '3':

print('Thank you for using Movie Booking System.')

break

else:

print('Invalid choice.')
```

In this code, we first define a dictionary movie that contains the available movies and their ticket prices. We also define a dictionary seating_arrangement that represents the seating arrangement in the theater. Each key in the seating_arrangement dictionary represents a row, and each value is a list of seats in that row.

- The print_seating_arrangement function is used to print the seating arrangement to the console.
- The purchase_ticket function prompts the user to enter the movie name, row, and seat number.

PROJECT 82: TEMPERATURE CONVERTER: BUILD A PROGRAM THAT CAN CONVERT TEMPERATURES BETWEEN CELSIUS, FAHRENHEIT, AND KELVIN SCALES

```
DEF CELSIUS_TO_FAHRENHEIT(celsius):

"""Convert Celsius to Fahrenheit"""

fahrenheit = (celsius * 9/5) + 32

return fahrenheit

def fahrenheit_to_celsius(fahrenheit):

"""Convert Fahrenheit to Celsius"""

celsius = (fahrenheit - 32) * 5/9

return celsius

def celsius_to_kelvin(celsius):

"""Convert Celsius to Kelvin"""

kelvin = celsius + 273.15

return kelvin

def kelvin_to_celsius(kelvin):

"""Convert Kelvin to Celsius"""
```

```
celsius = kelvin - 273.15

return celsius

def fahrenheit_to_kelvin(fahrenheit):

"""Convert Fahrenheit to Kelvin"""

celsius = fahrenheit_to_celsius(fahrenheit)

kelvin = celsius_to_kelvin(celsius)

return kelvin

def kelvin_to_fahrenheit(kelvin):

"""Convert Kelvin to Fahrenheit"""

celsius = kelvin_to_celsius(kelvin)

fahrenheit = celsius_to_fahrenheit(celsius)

return fahrenheit

# example usage

print(celsius_to_fahrenheit(25))

print(fahrenheit_to_celsius(77))

print(celsius_to_kelvin(25))

print(kelvin_to_celsius(298))

print(fahrenheit_to_kelvin(77))

print(kelvin_to_fahrenheit(298))
```

The functions use the appropriate formulas to perform the conversions. For example, to convert Celsius to Fahrenheit, we use the formula (celsius * 9/5) + 32. To convert Fahrenheit to Celsius, we use the formula (fahrenheit - 32) * 5/9. To convert Celsius to Kelvin, we simply add 273.15 to the Celsius temperature. To convert Kelvin to Celsius, we subtract 273.15 from the Kelvin temperature. To convert Fahrenheit to Kelvin, we first convert the Fahrenheit temperature to Celsius using the fahrenheit_to_celsius function, and then convert the Celsius temperature to Kelvin using the celsius_to_kelvin function. To convert Kelvin to Fahrenheit, we first convert the Kelvin temperature to Celsius using the kelvin_to_celsius function, and then convert the Celsius temperature to Fahrenheit using the celsius_to_fahrenheit function.

We then provide an example usage of the functions by calling each of them with a sample temperature and printing the result.

PROJECT 83: POMODORO TIMER: CREATE A PROGRAM THAT IMPLEMENTS THE POMODORO TECHNIQUE, A TIME MANAGEMENT METHOD THAT USES TIMED INTERVALS TO IMPROVE PRODUCTIVITY

IMPORT TIME

def pomodoro_timer(pomodoro_duration, short_break_duration, long_break_duration, pomodoros_before_long_break):

"""

A function that implements the Pomodoro technique.

pomodoro_duration: duration of a Pomodoro in minutes

short_break_duration: duration of a short break in minutes

long_break_duration: duration of a long break in minutes

pomodoros_before_long_break: number of Pomodoros before a long break

"""

pomodoros_completed = 0

while True:

print(f"Pomodoro {pomodoros_completed + 1}: Work for {pomodoro_duration} minutes.")

```
time.sleep(pomodoro_duration * 60)

pomodoros_completed += 1

if pomodoros_completed % pomodoros_before_long_break == 0:

print(f"Take a long break for {long_break_duration} minutes.")

time.sleep(long_break_duration * 60)

else:

print(f"Take a short break for {short_break_duration} minutes.")

time.sleep(short_break_duration * 60)

# example usage

pomodoro_timer(25, 5, 15, 4)
```

This code defines a function pomodoro_timer that takes four arguments: pomodoro_duration (duration of a Pomodoro in minutes), short_break_duration (duration of a short break in minutes), long_break_duration (duration of a long break in minutes), and pomodoros_before_long_break (number of Pomodoros before a long break).

The function runs an infinite loop that keeps track of the number of Pomodoros completed. For each Pomodoro, it prints a message to work for pomodoro_duration minutes and then waits for pomodoro_duration minutes using the time.sleep() function.

After each Pomodoro, the function checks if the number of Pomodoros completed is a multiple of pomodoros_before_long_break. If so, it prints a message to take a long break for long_break_duration minutes and waits for long_break_duration minutes using the time.sleep() function. Otherwise, it prints a message to take a short break for short_break_duration minutes and waits for short_break_duration minutes using the time.sleep() function.

We provide an example usage of the function by calling pomodoro_timer with a Pomodoro duration of 25 minutes, a short break duration of 5 minutes, a long break duration of 15 minutes, and 4 Pomodoros before a long break.

PROJECT 84: DISTRIBUTED COMPUTING: CREATE A PROGRAM THAT CAN DISTRIBUTE COMPUTING TASKS ACROSS MULTIPLE COMPUTERS OR SERVERS, USING PYTHON LIBRARIES SUCH AS DASK AND PYSPARK

```
IMPORT DASK.ARRAY AS da

import numpy as np

# Create a Dask array

arr = da.random.random((10000, 10000), chunks=(1000, 1000))

# Compute the sum of the array

sum_of_array = arr.sum()

# Print the result

print(sum_of_array.compute())
```

In this example, we first create a Dask array with 10,000 rows and 10,000 columns, and split it into 1000 x 1000 chunks. This allows Dask to distribute the computation of the sum of the array across multiple processors or machines, if available.

We then compute the sum of the array using the sum() method, which returns a Dask object representing the sum of the array. Finally, we call the compute() method on the result to retrieve the actual value of the sum.

Dask can also be used to parallelize more complex computations, such as machine learning algorithms or data processing tasks, by breaking them down into smaller tasks that can be executed independently and in parallel across multiple processors or machines.

PROJECT 85: AUTOMATED TESTING TOOL: DEVELOP A PROGRAM THAT CAN AUTOMATE SOFTWARE TESTING PROCESSES, USING FRAMEWORKS SUCH AS SELENIUM OR PYTEST

```
IMPORT PYTEST

from selenium import webdriver

# Set up the web driver

@pytest.fixture

def driver():

driver = webdriver.Chrome()

yield driver

driver.quit()

# Define the test case

def test_google_search(driver):

driver.get("https://www.google.com")

assert driver.title == "Google"

search_box = driver.find_element_by_name("q")

search_box.send_keys("Python")
```

```
search_box.submit()

assert "Python" in driver.title
```

In this example, we first import the pytest and selenium libraries. We then define a fixture named driver that sets up the web driver, in this case using Chrome. The yield statement is used to define the scope of the fixture, in this case, the driver is only used for one test case.

Next, we define a test case named test_google_search that navigates to Google's homepage, enters the search query "Python", submits the search form, and checks if the word "Python" is in the resulting page title.

Finally, we run the test case using the pytest command in the terminal. PyTest automatically detects and runs all test functions in the current directory and reports the results in a clear and concise format.

This is just a simple example, but PyTest can be used to test complex software systems, with support for features such as fixtures, parameterized tests, and test discovery.

PROJECT 86: FLIGHT TICKET BOOKING SYSTEM: DEVELOP A PROGRAM THAT CAN ALLOW USERS TO BOOK FLIGHT TICKETS ONLINE, WITH FEATURES SUCH AS FLIGHT SEARCH, SEAT SELECTION, AND PAYMENT PROCESSING

CLASS FLIGHT:

def __init__(self, flight_number, source, destination, departure_time, arrival_time, capacity, price):

self.flight_number = flight_number

self.source = source

self.destination = destination

self.departure_time = departure_time

self.arrival_time = arrival_time

self.capacity = capacity

self.price = price

self.seats = [[0 for _ in range(6)] for _ in range(25)]

def search(self, source, destination, date):

Search for available flights matching the source, destination, and date

pass

```
def book(self, seat_row, seat_col):

# Book a seat on the flight

pass

def pay(self, amount):

# Process the payment for the flight

pass
```

```
class User:

def __init__(self, name, email, phone):

self.name = name

self.email = email

self.phone = phone

self.bookings = []

def search_flights(self, source, destination, date):

# Search for flights matching the source, destination, and date

pass

def book_flight(self, flight, seat_row, seat_col):

# Book a flight seat

pass
```

```
def pay_for_booking(self, booking):

# Pay for a flight booking

pass
```

```
class Booking:

def __init__(self, user, flight, seat_row, seat_col):

self.user = user

self.flight = flight

self.seat_row = seat_row

self.seat_col = seat_col

self.amount = flight.price

def pay(self):

self.flight.book(self.seat_row, self.seat_col)

self.flight.pay(self.amount)

self.user.bookings.append(self)
```

```
# Sample usage

flight = Flight("AA123", "SFO", "JFK", "10:00", "18:00", 150, 250.0)

user = User("John Doe", "john.doe@example.com", "555-1234")
```

```
available_flights = user.search_flights("SFO", "JFK", "2023-03-01")

booking = Booking(user, available_flights[0], 5, 2)

booking.pay()
```

In this example, we define three classes: Flight, User, and Booking. The Flight class represents a flight with a unique flight number, source and destination airports, departure and arrival times, capacity, and price. It also has a seats attribute, which represents the available seats on the flight.

The User class represents a user with a name, email, phone number, and a list of flight bookings. It has methods to search for available flights, book a flight seat, and pay for a booking.

The Booking class represents a flight booking made by a user, with references to the user, flight, seat row, seat column, and the booking amount. It also has a pay() method that books the seat on the flight and processes the payment.

In the sample usage, we create a new Flight object and a new User object. We then search for available flights matching the source, destination, and date using the search_flights() method of the User object. We create a new Booking object with the first available flight, and pay for the booking using the pay() method. The booking is added to the user's list of bookings.

PROJECT 87: VIDEO GAME EMULATOR: DEVELOP A PROGRAM THAT CAN EMULATE CLASSIC VIDEO GAMES, USING LIBRARIES SUCH AS PYGAME OR PYNES

```
IMPORT PYGAME

from pygame.locals import *

import nes_file

pygame.init()

SCREEN_WIDTH = 256

SCREEN_HEIGHT = 240

def main():

screen = pygame.display.set_mode((SCREEN_WIDTH,
SCREEN_HEIGHT))

pygame.display.set_caption("NES Emulator")

nes = nes_file.NesFile("rom.nes")  # load NES file

clock = pygame.time.Clock()

while True:

for event in pygame.event.get():

if event.type == QUIT:
```

```
pygame.quit()

return

keys = pygame.key.get_pressed()

# map keyboard keys to NES controller buttons

controller1 = 0x00

if keys[K_UP]:

controller1 |= 0x08

if keys[K_DOWN]:

controller1 |= 0x04

if keys[K_LEFT]:

controller1 |= 0x02

if keys[K_RIGHT]:

controller1 |= 0x01

if keys[K_z]: # A button

controller1 |= 0x80

if keys[K_x]: # B button

controller1 |= 0x40

if keys[K_RETURN]: # Start button

controller1 |= 0x08
```

```
if keys[K_BACKSPACE]: # Select button

controller1 |= 0x04

nes.set_controller_state(0, controller1)

frame = nes.get_frame()

surface = pygame.surfarray.make_surface(frame)

screen.blit(surface, (0, 0))

pygame.display.flip()

clock.tick(60)

if __name__ == "__main__":

main()
```

In this example, we import the pygame library and define constants for the screen width and height. We then define a main() function that sets up the Pygame window, loads a NES file using a custom nes_file module, and runs the game loop.

Inside the game loop, we get the state of the keyboard keys using pygame.key.get_pressed() and map them to the NES controller buttons. We then set the controller state using the nes.set_controller_state() method.

We get the current frame of the game using the nes.get_frame() method and convert it to a Pygame surface using pygame.surfarray.make_surface(). We then blit the surface onto the screen and call pygame.display.flip() to update the display.

Finally, we call pygame.time.Clock().tick(60) to limit the frame rate to 60 frames per second.

This is just a simple example, but a real-world video game emulator would need to support a wide range of games and emulate the hardware accurately.

PROJECT 88: NEWS SENTIMENT ANALYSIS: BUILD A PROGRAM THAT CAN ANALYZE NEWS ARTICLES AND CLASSIFY THEM AS POSITIVE, NEGATIVE, OR NEUTRAL, USING NATURAL LANGUAGE PROCESSING TECHNIQUES SUCH AS SENTIMENT ANALYSIS

```
IMPORT NLTK

from nltk.sentiment import SentimentIntensityAnalyzer

nltk.download('vader_lexicon')  # download the VADER lexicon

def analyze_sentiment(text):

sia = SentimentIntensityAnalyzer()

sentiment = sia.polarity_scores(text)['compound']

if sentiment > 0.05:

return "positive"

elif sentiment < -0.05:

return "negative"

else:

return "neutral"

# Example usage
```

```
news_article = "The stock market soared to new heights today as investors cheered the latest earnings reports. The Dow Jones Industrial Average rose more than 500 points in afternoon trading."

sentiment = analyze_sentiment(news_article)

print(sentiment)  # Output: "positive"
```

In this example, we first download the VADER lexicon using nltk.download(). VADER (Valence Aware Dictionary and sEntiment Reasoner) is a pre-trained sentiment analysis tool included in the nltk library.

We define a function analyze_sentiment() that takes a text input and uses the SentimentIntensityAnalyzer() class from the nltk.sentiment module to compute the sentiment score of the input text. The sentiment score is a value between -1 (most negative) and 1 (most positive).

We then classify the sentiment as positive, negative, or neutral based on the sentiment score. In this example, we use a threshold of +/- 0.05 to determine the sentiment classification.

Finally, we demonstrate the usage of the analyze_sentiment() function by passing in a sample news article and printing out the sentiment classification.

Note that this is just a simple example and more advanced sentiment analysis techniques, such as topic modeling and named entity recognition, can be incorporated to improve the accuracy of the sentiment analysis.

PROJECT 89: FILE TRANSFER TOOL: DEVELOP A PROGRAM THAT CAN TRANSFER FILES BETWEEN DIFFERENT DEVICES OR SERVERS, USING PROTOCOLS SUCH AS FTP OR SFTP

IMPORT OS

import paramiko

Define the hostname, username, and password for the remote server

host = 'example.com'

username = 'username'

password = 'password'

Define the local and remote file paths

local_file = '/path/to/local/file.txt'

remote_file = '/path/to/remote/file.txt'

Create a new SSH client

ssh = paramiko.SSHClient()

Automatically add the server's host key to the local key store

ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())

Connect to the server using the provided credentials

ssh.connect(hostname=host, username=username, password=password)

Create an SFTP client from the SSH connection

sftp = ssh.open_sftp()

Transfer the file from the local machine to the remote server

sftp.put(local_file, remote_file)

Close the SFTP and SSH connections

sftp.close()

ssh.close()

In this example, we first import the os and paramiko libraries.

We then define the hostname, username, and password for the remote server, as well as the local and remote file paths.

We create a new SSH client using paramiko.SSHClient() and set the host key policy to automatically add the server's host key to the local key store.

We connect to the server using the provided credentials using ssh.connect().

We create an SFTP client from the SSH connection using ssh.open_sftp().

We transfer the file from the local machine to the remote server using sftp.put(local_file, remote_file).

Finally, we close the SFTP and SSH connections using sftp.close() and ssh.close().

Note that this is just a simple example and more advanced file transfer techniques, such as using FTP or adding error handling, can be incorporated to improve the functionality and robustness of the file transfer tool.

PROJECT 90: ENCRYPTION AND DECRYPTION TOOL: DEVELOP A PROGRAM THAT CAN ENCRYPT AND DECRYPT MESSAGES AND FILES, USING CRYPTOGRAPHIC ALGORITHMS SUCH AS AES OR RSA

IMPORT OS

from cryptography.fernet import Fernet

generate a key for encryption and decryption

key = Fernet.generate_key()

save the key to a file

with open('key.key', 'wb') as key_file:

key_file.write(key)

load the key from the file

with open('key.key', 'rb') as key_file:

key = key_file.read()

create a Fernet instance using the key

fernet = Fernet(key)

encrypt a file

```
with open('plaintext.txt', 'rb') as plaintext_file:

plaintext = plaintext_file.read()

encrypted_plaintext = fernet.encrypt(plaintext)

with open('encrypted.txt', 'wb') as encrypted_file:

encrypted_file.write(encrypted_plaintext)

# decrypt a file

with open('encrypted.txt', 'rb') as encrypted_file:

encrypted_plaintext = encrypted_file.read()

decrypted_plaintext = fernet.decrypt(encrypted_plaintext)

with open('decrypted.txt', 'wb') as decrypted_file:

decrypted_file.write(decrypted_plaintext)
```

Note that this code assumes that you have a plaintext file named plaintext.txt in the current working directory. It will encrypt the contents of this file and save the encrypted result to a new file named encrypted.txt. It will then read the contents of encrypted.txt, decrypt them, and save the result to a new file named decrypted.txt.

PROJECT 91: AUTOMATED EMAIL RESPONDER: CREATE A PROGRAM THAT CAN RESPOND TO EMAILS AUTOMATICALLY, USING NATURAL LANGUAGE PROCESSING TECHNIQUES SUCH AS TEXT CLASSIFICATION

```
IMPORT IMAPLIB

import email

from textblob import TextBlob

# login to the email account

mail = imaplib.IMAP4_SSL('imap.gmail.com')

mail.login('your_email_address', 'your_email_password')

mail.select('inbox')

# search for unread messages

result, data = mail.search(None, 'UNSEEN')

# loop through each message and classify the content

for num in data[0].split():

result, data = mail.fetch(num, '(RFC822)')

raw_email = data[0][1]
```

```
email_message = email.message_from_bytes(raw_email)

# extract the subject and body of the email

subject = email_message['subject']

body = email_message.get_payload()

# classify the content using TextBlob

blob = TextBlob(body)

if blob.sentiment.polarity < -0.5:

response = "I'm sorry to hear that you're having a bad day. " \

"Is there anything I can do to help?"

elif blob.sentiment.polarity > 0.5:

response = "I'm glad to hear that you're having a good day! " \

"Let me know if there's anything I can do to make it even better."

else:

response = "Thanks for getting in touch. " \

"I'll get back to you as soon as I can."

# send the response email

response_message = email.message.EmailMessage()

response_message.set_content(response)

response_message['to'] = email_message['from']
```

```
response_message['subject'] = f"Re: {subject}"

mail.send_message(response_message)

# logout of the email account

mail.close()

mail.logout()
```

This code uses the imaplib library to connect to an email account, search for unread messages in the inbox, and retrieve the subject and body of each message. It then uses the TextBlob library to classify the content of the message based on its sentiment polarity. If the polarity is less than -0.5, the program generates a sympathetic response. If the polarity is greater than 0.5, the program generates a positive response. Otherwise, the program generates a neutral response.

The program then creates a new email message with the appropriate response and sends it to the sender of the original message. The new message has the original subject with "Re:" added to the beginning.

Note that this code assumes that you have a Gmail account and that you have enabled IMAP access for your account. You will also need to provide your email address and password in the appropriate places in the code.

PROJECT 92: MOVIE REVIEW SENTIMENT ANALYSIS: DEVELOP A PROGRAM THAT CAN ANALYZE MOVIE REVIEWS AND CLASSIFY THEM AS POSITIVE OR NEGATIVE, USING NATURAL LANGUAGE PROCESSING TECHNIQUES SUCH AS SENTIMENT ANALYSIS

```
IMPORT NLTK

from nltk.corpus import movie_reviews

from nltk.classify import NaiveBayesClassifier

from nltk.tokenize import word_tokenize

# define a function to extract features from the review text

def extract_features(text):

words = word_tokenize(text)

return dict([(word, True) for word in words])

# create a list of positive and negative reviews

positive_reviews = []

for fileid in movie_reviews.fileids('pos'):

words = movie_reviews.words(fileid)

positive_reviews.append((extract_features(words), 'Positive'))
```

```
negative_reviews = []

for fileid in movie_reviews.fileids('neg'):

words = movie_reviews.words(fileid)

negative_reviews.append((extract_features(words), 'Negative'))

# divide the reviews into training and testing sets

positive_cutoff = int(len(positive_reviews) * 0.8)

negative_cutoff = int(len(negative_reviews) * 0.8)

train_features = positive_reviews[:positive_cutoff] +
negative_reviews[:negative_cutoff]

test_features = positive_reviews[positive_cutoff:] +
negative_reviews[negative_cutoff:]

# train a Naive Bayes classifier on the training set

classifier = NaiveBayesClassifier.train(train_features)

# classify the reviews in the testing set and print the accuracy

accuracy = nltk.classify.accuracy(classifier, test_features)

print("Accuracy:", accuracy)

# classify a new review

new_review = "The movie was great! I really enjoyed it."

new_review_features = extract_features(word_tokenize(new_review))

print("Sentiment:", classifier.classify(new_review_features))
```

This code uses the movie_reviews corpus from NLTK to create a list of positive and negative movie reviews. It then defines a function to extract features from the review text by tokenizing the words in the text and creating a dictionary of word features.

The program then divides the reviews into training and testing sets, and trains a Naive Bayes classifier on the training set. It uses the classifier to classify the reviews in the testing set and calculates the accuracy of the classifier.

Finally, the program demonstrates how to classify a new review by creating a feature set from the review text and using the classifier to classify it as positive or negative.

Note that this code uses a simple bag-of-words approach to feature extraction, which may not be the most effective approach for sentiment analysis. More sophisticated techniques, such as using word embeddings or neural networks, may produce better results. Additionally, this code uses a relatively small dataset of movie reviews, and may not generalize well to other types of text data.

Project 93: Restaurant recommendation system: Build a program that can recommend restaurants based on users' preferences, such as cuisine type, price range, and location, using machine learning algorithms such as collaborative filtering

```
import pandas as pd

from sklearn.metrics.pairwise import cosine_similarity

# Load data

restaurants_df = pd.read_csv('restaurants.csv')

reviews_df = pd.read_csv('reviews.csv')

# Merge data

merged_df = pd.merge(restaurants_df, reviews_df, on='restaurant_id')

# Pivot table

pivot_table = pd.pivot_table(merged_df, values='rating', index='user_id', columns='restaurant_id')

# Fill missing values with 0

pivot_table.fillna(0, inplace=True)

# Calculate cosine similarity

cosine_sim = cosine_similarity(pivot_table)
```

Define a function to get restaurant recommendations

def get_recommendations(user_id, cosine_sim, pivot_table, num_recommendations):

Get index of user_id

user_idx = pivot_table.index.get_loc(user_id)

Get similarity scores

sim_scores = list(enumerate(cosine_sim[user_idx]))

Sort by similarity score

sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

Get index of top recommendations

restaurant_indices = [i[0] for i in sim_scores]

Get restaurant names

restaurant_names = restaurants_df.loc[restaurants_df['restaurant_id'].isin(restaurant_indices)]['restaurant_name']

Return top recommendations

return restaurant_names[:num_recommendations]

Example usage

get_recommendations ('user1', cosine_sim, pivot_table, 5)

In this code, we first load two CSV files: one containing restaurant information (such as ID, name, cuisine type, price range, and location) and one containing user reviews (such as user ID, restaurant ID, and rating). We then merge the two data frames and pivot the table so that each row represents a user and each column represents a restaurant, with the ratings as the values. We then calculate the cosine similarity between users and use this to get recommendations for a specific user based on their preferences. The get_recommendations () function takes in a user ID, the cosine similarity matrix, the pivot table, and the number of recommendations to return, and returns the top recommended restaurants based on the user's preferences.

PROJECT 94: IMAGE STYLE TRANSFER: DEVELOP A PROGRAM THAT CAN TRANSFER THE STYLE OF ONE IMAGE ONTO ANOTHER IMAGE, USING DEEP LEARNING ALGORITHMS SUCH AS NEURAL STYLE TRANSFER

THIS IS A BASIC SAMPL code for image style transfer using neural style transfer.

```
import tensorflow as tf

import numpy as np

import PIL.Image

# Load content and style images

content_image = np.array(PIL.Image.open("content_image.jpg").resize((400, 400)))

style_image = np.array(PIL.Image.open("style_image.jpg").resize((400, 400)))

# Convert images to tensors

content_tensor = tf.keras.preprocessing.image.img_to_array(content_image)

style_tensor = tf.keras.preprocessing.image.img_to_array(style_image)

# Preprocess images
```

```
content_tensor =
tf.keras.applications.vgg19.preprocess_input(content_tensor)

style_tensor = tf.keras.applications.vgg19.preprocess_input(style_tensor)

# Define the VGG19 model

vgg19 = tf.keras.applications.vgg19.VGG19(include_top=False,
weights='imagenet')

# Define the content layer and style layers

content_layer = 'block5_conv2'

style_layers = ['block1_conv1', 'block2_conv1', 'block3_conv1',
'block4_conv1', 'block5_conv1']

# Extract content and style features

content_features = vgg19.get_layer(content_layer).output

style_features = [vgg19.get_layer(layer).output for layer in style_layers]

# Create the model for feature extraction

model = tf.keras.models.Model(inputs=vgg19.input, outputs=
[content_features, *style_features])

# Define the content and style targets

content_target = model(content_tensor)[0]

style_targets = [model(style_tensor)[i] for i in range(len(style_layers))]

# Define the Gram matrix function

def gram_matrix(input_tensor):
```

```
channels = int(input_tensor.shape[-1])

a = tf.reshape(input_tensor, [-1, channels])

n = tf.shape(a)[0]

gram = tf.matmul(a, a, transpose_a=True)

return gram / tf.cast(n, tf.float32)

# Define the style loss function

def style_loss(style_target, combination):

style_target_gram = gram_matrix(style_target)

combination_gram = gram_matrix(combination)

channels = style_target_gram.shape[-1]

size = style_target_gram.shape[0] * style_target_gram.shape[1]

return tf.reduce_sum(tf.square(style_target_gram - combination_gram)) /
(4.0 * (channels ** 2) * (size ** 2))

# Define the content loss function

def content_loss(content_target, combination):

return tf.reduce_sum(tf.square(content_target - combination))

# Define the total loss function

def total_loss(style_targets, content_target, combination, style_weights,
content_weight):

style_loss_val = 0
```

```
for i in range(len(style_targets)):

style_loss_val += style_weights[i] * style_loss(style_targets[i],
combination)

content_loss_val = content_weight * content_loss(content_target,
combination)

return style_loss_val + content_loss_val

# Define the optimizer

optimizer = tf.optimizers.Adam(learning_rate=5.0, beta_1=0.99,
epsilon=1e-1)

# Define the training loop

@tf.function()

def train_step(image, style_targets, content_target, style_weights,
content_weight):

with tf.GradientTape() as tape:

outputs = model(image)

combination = outputs[0]

loss = total_loss(style_targets, content_target, combination, style_weights,
content_weight)

grad = tape.gradient(loss, image)

optimizer.apply_gradients([(grad, image)])

image.assign(tf.clip_by_value(image, clip_value_min=0.0,
clip_value_max=255.0))
```

PROJECT 95: TIME SERIES FORECASTING: CREATE A PROGRAM THAT CAN FORECAST TIME SERIES DATA, SUCH AS STOCK PRICES OR WEATHER PATTERNS, USING MACHINE LEARNING ALGORITHMS SUCH AS AUTOREGRESSIVE INTEGRATED MOVING AVERAGE (ARIMA) AND LONG SHORT-TERM MEMORY (LSTM) NETWORKS

THE FOLLOWING SAMPLE Python codes for time series forecasting use ARIMA and LSTM networks:

ARIMA:

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import statsmodels.api as sm

# Load and visualize the time series data

df = pd.read_csv("time_series_data.csv", parse_dates=['Date'],
index_col='Date')

plt.plot(df)

plt.show()
```

```
# Fit an ARIMA model

model = sm.tsa.ARIMA(df, order=(1, 1, 1))

results = model.fit()

# Generate predictions

forecast = results.predict(start="2023-03-01", end="2023-04-01",
dynamic=True)

# Visualize the predictions

plt.plot(df)

plt.plot(forecast)

plt.show()
```

LSTM:

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

from sklearn.preprocessing import MinMaxScaler

# Load and preprocess the time series data

df = pd.read_csv("time_series_data.csv", parse_dates=['Date'],
index_col='Date')

scaler = MinMaxScaler()
```

```
scaled_df = scaler.fit_transform(df)

# Split the data into training and test sets

train_size = int(len(scaled_df) * 0.8)

test_size = len(scaled_df) - train_size

train_data, test_data = scaled_df[0:train_size,:],
scaled_df[train_size:len(scaled_df),:]

# Define the function to create time series datasets

def create_dataset(dataset, time_steps=1):

dataX, dataY = [], []

for i in range(len(dataset)-time_steps-1):

a = dataset[i:(i+time_steps), 0]

dataX.append(a)

dataY.append(dataset[i + time_steps, 0])

return np.array(dataX), np.array(dataY)

# Define the time steps and create the datasets

time_steps = 10

trainX, trainY = create_dataset(train_data, time_steps)

testX, testY = create_dataset(test_data, time_steps)

# Reshape the datasets for LSTM input
```

```
trainX = np.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))

testX = np.reshape(testX, (testX.shape[0], 1, testX.shape[1]))

# Define the LSTM model

model = tf.keras.Sequential([

tf.keras.layers.LSTM(50, input_shape=(1, time_steps)),

tf.keras.layers.Dense(1)

])

# Compile the model

model.compile(loss='mean_squared_error', optimizer='adam')

# Train the model

model.fit(trainX, trainY, epochs=100, batch_size=1, verbose=2)

# Generate predictions

trainPredict = model.predict(trainX)

testPredict = model.predict(testX)

# Invert the predictions to their original scale

trainPredict = scaler.inverse_transform(trainPredict)

trainY = scaler.inverse_transform([trainY])

testPredict = scaler.inverse_transform(testPredict)

testY = scaler.inverse_transform([testY])
```

```
# Visualize the predictions

plt.plot(df)

plt.plot(np.concatenate((trainPredict,testPredict),axis=0))

plt.show()
```

PROJECT 96: STOCK PRICE PREDICTION: CREATE A PROGRAM THAT CAN PREDICT STOCK PRICES USING MACHINE LEARNING ALGORITHMS SUCH AS NEURAL NETWORKS AND DECISION TREES

THE FOLLOWING PYTHON code uses neural networks and decision trees to predict stock price.

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import mean_squared_error

# Load and preprocess the stock price data

df = pd.read_csv("stock_price_data.csv")

scaler = MinMaxScaler()

scaled_df = scaler.fit_transform(df['Close'].values.reshape(-1, 1))

# Split the data into training and test sets

train_size = int(len(scaled_df) * 0.8)
```

```
test_size = len(scaled_df) - train_size

train_data, test_data = scaled_df[0:train_size,:],
scaled_df[train_size:len(scaled_df),:]

# Define the function to create time series datasets

def create_dataset(dataset, time_steps=1):

dataX, dataY = [], []

for i in range(len(dataset)-time_steps-1):

a = dataset[i:(i+time_steps), 0]

dataX.append(a)

dataY.append(dataset[i + time_steps, 0])

return np.array(dataX), np.array(dataY)

# Define the time steps and create the datasets

time_steps = 10

trainX, trainY = create_dataset(train_data, time_steps)

testX, testY = create_dataset(test_data, time_steps)

# Reshape the datasets for LSTM input

trainX = np.reshape(trainX, (trainX.shape[0], trainX.shape[1], 1))

testX = np.reshape(testX, (testX.shape[0], testX.shape[1], 1))

# Define the LSTM model
```

```
model = tf.keras.Sequential([

tf.keras.layers.LSTM(50, return_sequences=True, input_shape=(time_steps,
1)),

tf.keras.layers.Dropout(0.2),

tf.keras.layers.LSTM(50, return_sequences=True),

tf.keras.layers.Dropout(0.2),

tf.keras.layers.LSTM(50),

tf.keras.layers.Dropout(0.2),

tf.keras.layers.Dense(1)

])

# Compile the model

model.compile(loss='mean_squared_error', optimizer='adam')

# Train the model

model.fit(trainX, trainY, epochs=100, batch_size=32, verbose=2)

# Generate predictions

trainPredict = model.predict(trainX)

testPredict = model.predict(testX)

# Invert the predictions to their original scale

trainPredict = scaler.inverse_transform(trainPredict)
```

```
trainY = scaler.inverse_transform([trainY])

testPredict = scaler.inverse_transform(testPredict)

testY = scaler.inverse_transform([testY])

# Calculate the root mean squared error

trainScore = np.sqrt(mean_squared_error(trainY[0], trainPredict[:,0]))

print('Train Score: %.2f RMSE' % (trainScore))

testScore = np.sqrt(mean_squared_error(testY[0], testPredict[:,0]))

print('Test Score: %.2f RMSE' % (testScore))

# Visualize the predictions

plt.plot(df['Close'])

plt.plot(np.concatenate((trainPredict[:,0],testPredict[:,0]),axis=0))

plt.show()
```

Decision Tree:

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.tree import DecisionTreeRegressor

from sklearn.model_selection import train_test_split

from sklearn.metrics import mean_squared_error

# Load and preprocess the stock price data

df = pd.read_csv("stock_price_data.csv")

X = df.drop(['Date', 'Close'], axis=1)

y = df['Close']

X_train, X
```

PROJECT 97: CONTENT-BASED IMAGE RETRIEVAL: BUILD A PROGRAM THAT CAN RETRIEVE IMAGES FROM A LARGE DATABASE BASED ON THEIR CONTENT, USING TECHNIQUES SUCH AS FEATURE EXTRACTION AND SIMILARITY MATCHING

```
IMPORT CV2

import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

from sklearn.neighbors import NearestNeighbors

# Load the images and convert them to grayscale

img1 = cv2.imread('image1.jpg')

img2 = cv2.imread('image2.jpg')

img1_gray = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)

img2_gray = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)

# Extract features from the images using SIFT

sift = cv2.xfeatures2d.SIFT_create()

kp1, des1 = sift.detectAndCompute(img1_gray, None)
```

```
kp2, des2 = sift.detectAndCompute(img2_gray, None)

# Cluster the feature descriptors using KMeans

kmeans = KMeans(n_clusters=10)

kmeans.fit(des1)

labels = kmeans.predict(des1)

# Build a bag of visual words for the first image

histogram = np.zeros(10)

for i in labels:

histogram[i] += 1

# Use the bag of visual words to find similar images in a database

database = [cv2.imread('image3.jpg'), cv2.imread('image4.jpg'),
cv2.imread('image5.jpg')]

database_gray = [cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) for img in
database]

database_des = [sift.detectAndCompute(img, None)[1] for img in
database_gray]

database_hist = []

for des in database_des:

labels = kmeans.predict(des)

histogram = np.zeros(10)
```

```
for i in labels:

histogram[i] += 1

database_hist.append(histogram)

# Use k-nearest neighbors to find the most similar images

nn = NearestNeighbors(n_neighbors=3)

nn.fit(database_hist)

distances, indices = nn.kneighbors([histogram])

print("Most similar images:")

for i in indices[0]:

plt.imshow(database[i])

plt.show()
```

In this example, we use the SIFT feature extraction algorithm to extract features from the images, cluster the feature descriptors using KMeans, and build a bag of visual words for each image. Then, we use k-nearest neighbors to find the most similar images in a database based on their bag of visual words.

PROJECT 98: MULTI-AGENT SYSTEMS: BUILD A PROGRAM THAT SIMULATES A MULTI-AGENT ENVIRONMENT, WHERE AGENTS INTERACT WITH EACH OTHER AND THEIR ENVIRONMENT TO ACHIEVE A GOAL, USING TECHNIQUES SUCH AS REINFORCEMENT LEARNING AND GAME THEORY

IMPORT NUMPY AS NP

import random

Define the environment and agents

num_agents = 2

num_actions = 3

num_states = 4

rewards = np.array([[3, 2, 0], [2, 0, 1]])

actions = np.array([0, 1, 2])

states = np.array([0, 1, 2, 3])

Define the Q-learning algorithm for each agent

def q_learning_agent(q_table, state, epsilon):

if random.uniform(0, 1) < epsilon:

```
# Choose a random action

action = random.choice(actions)

else:

# Choose the best action based on the Q-table

action = np.argmax(q_table[state])

return action

# Define the game between the agents

def game(q_tables, epsilon):

# Choose a random initial state

state = random.choice(states)

# Initialize the rewards and actions

total_rewards = [0] * num_agents

actions = [None] * num_agents

# Iterate through each time step

for t in range(10):

# Choose actions for each agent based on the Q-tables and epsilon

for i in range(num_agents):

actions[i] = q_learning_agent(q_tables[i], state, epsilon)

# Get the rewards for each agent based on their actions
```

for i in range(num_agents):

total_rewards[i] += rewards[i][actions[i]]

Update the state based on the actions

state = actions[0] + actions[1] * 3

return total_rewards

Train the agents using Q-learning

q_tables = [np.zeros((num_states, num_actions)) for i in range(num_agents)]

epsilon = 1.0

decay_rate = 0.99

min_epsilon = 0.01

for episode in range(1000):

Decay the epsilon value over time

epsilon = max(min_epsilon, epsilon * decay_rate)

Play a game and update the Q-tables for each agent

total_rewards = game(q_tables, epsilon)

for i in range(num_agents):

q_tables[i][state][actions[i]] += 0.1 * (total_rewards[i] + 0.9 * np.max(q_tables[i][state]) - q_tables[i][state][actions[i]])

Test the agents by playing a game with a fixed Q-table

```
q_tables_fixed = [np.array([[3, 2, 0], [2, 0, 1], [0, 0, 0], [0, 0, 0]]),
np.array([[0, 0, 0], [2, 0, 1], [3, 2, 0], [0, 0, 0]])]

total_rewards = game(q_tables_fixed, 0)

print("Total rewards:", total_rewards)
```

In this example, we define a simple two-player game where each player can choose one of three actions, and the reward for each player depends on the joint action taken. We use the Q-learning algorithm to train each agent to choose the best action based on the current state and the Q-table, which is updated after each game. We also gradually decay the exploration rate (epsilon) over time to encourage the agents to exploit their learned policies. Finally, we test the agents by playing a game with a fixed Q-table and print out the total rewards for each agent.

PROJECT 99: DEEPFAKE DETECTION: BUILD A PROGRAM THAT CAN DETECT MANIPULATED IMAGES AND VIDEOS, SUCH AS DEEPFAKE VIDEOS, USING COMPUTER VISION TECHNIQUES SUCH AS FACE DETECTION AND ANALYSIS

IN THIS EXAMPLE, WE use the dlib library to detect faces and extract facial landmarks from an image, and then compute the eye aspect ratio.

```
import cv2

import dlib

import numpy as np

import matplotlib.pyplot as plt

# Define the face detector and landmark predictor

detector = dlib.get_frontal_face_detector()

predictor =
dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")

# Define the function for extracting features from an image

def extract_features(image):

# Convert the image to grayscale

gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

```
# Detect the face in the image

face_rect = detector(gray, 1)[0]

# Extract the facial landmarks

landmarks = predictor(gray, face_rect)

# Convert the landmarks to a numpy array

landmarks = np.array([(landmarks.part(i).x, landmarks.part(i).y) for i in range(68)])

# Compute the eye aspect ratio (EAR)

left_eye = landmarks[36:42]

right_eye = landmarks[42:48]

ear_left = (np.linalg.norm(left_eye[1] - left_eye[5]) + np.linalg.norm(left_eye[2] - left_eye[4])) / (2 * np.linalg.norm(left_eye[0] - left_eye[3]))

ear_right = (np.linalg.norm(right_eye[1] - right_eye[5]) + np.linalg.norm(right_eye[2] - right_eye[4])) / (2 * np.linalg.norm(right_eye[0] - right_eye[3]))

# Compute the mouth aspect ratio (MAR)

mouth = landmarks[48:68]

mar = (np.linalg.norm(mouth[12] - mouth[4]) + np.linalg.norm(mouth[13] - mouth[3]) + np.linalg.norm(mouth[14] - mouth[2]) + np.linalg.norm(mouth[15] - mouth[1]) + np.linalg.norm(mouth[16] - mouth[0])) / (5 * np.linalg.norm(mouth[6] - mouth[9]))

# Concatenate the features into a single vector
```

```
features = np.array([ear_left, ear_right, mar])

return features

# Load the real and fake images

real_image = cv2.imread("real_image.jpg")

fake_image = cv2.imread("fake_image.jpg")

# Extract the features from each image

real_features = extract_features(real_image)

fake_features = extract_features(fake_image)

# Compute the distance between the feature vectors

distance = np.linalg.norm(real_features - fake_features)

# Print the distance and label the image as real or fake based on a threshold

threshold = 0.5

if distance < threshold:

print("Real image")

else:

print("Fake image")

# Display the images and feature points

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 5))
```

```
ax1.imshow(cv2.cvtColor(real_image, cv2.COLOR_BGR2RGB))

ax1.scatter(real_features[::2] * real_image.shape[1], real_features[1::2] * real_image.shape[0], color="green", marker=".")

ax1.set_title("Real image")

ax2.imshow(cv2.cvtColor(fake_image, cv2.COLOR_BGR2RGB))

ax2.scatter(fake_features[::2] * fake_image.shape[1], fake_features[1::2] * fake_image.shape[0], color="red", marker=".")

ax2.set_title("Fake image")

plt.show()
```

PROJECT 100: REINFORCEMENT LEARNING FOR ROBOTICS: BUILD A PROGRAM THAT USES REINFORCEMENT LEARNING TECHNIQUES TO TRAIN A ROBOT TO PERFORM A TASK, SUCH AS NAVIGATING A MAZE OR PLAYING A GAME

IMPORT GYM

import numpy as np

Define the Q-learning algorithm

def q_learning(env, num_episodes=10000, alpha=0.1, gamma=0.99, epsilon=0.1):

Initialize the Q-table

q_table = np.zeros((env.observation_space.n, env.action_space.n))

Loop over episodes

for episode in range(num_episodes):

Reset the environment

state = env.reset()

Loop over timesteps in the episode

done = False

```
while not done:

# Choose an action using epsilon-greedy exploration

if np.random.uniform() < epsilon:

action = env.action_space.sample()

else:

action = np.argmax(q_table[state])

# Take a step in the environment

next_state, reward, done, info = env.step(action)

# Update the Q-table using the Q-learning update rule

q_table[state, action] += alpha * (reward + gamma *
np.max(q_table[next_state]) - q_table[state, action])

# Update the state

state = next_state

# Return the learned Q-table

return q_table

# Define the robot navigation environment

env = gym.make('FrozenLake-v0')

# Train the robot using Q-learning

q_table = q_learning(env)
```

Use the learned Q-table to navigate the environment

state = env.reset()

done = False

while not done:

action = np.argmax(q_table[state])

state, reward, done, info = env.step(action)

env.render()

Close the environment

env.close()

In this example, we use the OpenAI Gym environment to define the robot navigation task, which is modeled as a Markov decision process with a finite set of states and actions. We then use the Q-learning algorithm to learn an optimal policy for the task by updating a Q-table with the expected future rewards for each state-action pair. Finally, we use the learned Q-table to navigate the environment by choosing the action with the highest expected reward at each timestep.