PYTHON IPE ...

1)Use the file data.csv which contains 169 rows and 4 columns.
1. Convert this file into pandas Data Frame and Display basic statistics like mean, std, quartiles, etc. for this data frame.
2. Print first and last 5 rows. Also print the shape of the dataframe.
3. Create a correlation table for the data frame and comment about what kind of correlation is there between Duration and Calories?
4. Find whether there any null or NA values, drop all such rows if found in the data frame and print the shape of the data frame after dropping.
5. Prepare a scatter matrix for the following data frame.
6. Prepare a parallel coordinates for Duration v/s Pulse, Maxpulse and Calories (all 3 other columns).
7. Prepare a cross-tabulation for Duration v/s Pulse.
8. Do Maxpulse have any outliers? Find using function.

ANS)

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Step 1: Load the CSV file into a pandas DataFrame and display basic
statistics
data = pd.read_csv('data.csv')
statistics = data.describe()
print("Basic Statistics:")
print(statistics)

# Step 2: Print the first and last 5 rows and the shape of the DataFrame
print("\nFirst 5 rows:")
print(data.head())
print("\nLast 5 rows:")
print(data.tail())
print("\nShape of the DataFrame:")
print(data.shape)

# Step 3: Create a correlation table and comment on the correlation
between Duration and Calories
correlation_table = data.corr()
print("\nCorrelation Table:")
print(correlation_table)
correlation_comment = "The correlation between Duration and Calories
appears to be positive, indicating that as the duration of the activity
increases, the number of calories burned also tends to increase."
print("\nCorrelation Comment:")
print(correlation_comment)

# Step 4: Check for and drop rows with null or NA values and print the
shape after dropping
data.dropna(inplace=True)
print("\nShape of the DataFrame after dropping null/NA rows:")
print(data.shape)

# Step 5: Prepare a scatter matrix for the DataFrame
sns.pairplot(data)
plt.show()
```

```
# Step 6: Prepare a parallel coordinates plot for Duration vs. Pulse,
Maxpulse, and Calories
sns.set(style="whitegrid")
parallel_coordinates_data = data[['Duration', 'Pulse', 'Maxpulse',
'Calories']]
plt.figure(figsize=(10, 6))
parallel_coordinates = sns.lineplot(data=parallel_coordinates_data,
palette="tab10", linewidth=2)
plt.title("Parallel Coordinates Plot for Duration vs. Pulse, Maxpulse,
and Calories")
plt.show()

# Step 7: Prepare a cross-tabulation for Duration vs. Pulse
cross_tabulation = pd.crosstab(data['Duration'], data['Pulse'])
print("\nCross-Tabulation for Duration vs. Pulse:")
print(cross_tabulation)

# Step 8: Check for outliers in Maxpulse using a box plot
plt.figure(figsize=(8, 4))
sns.boxplot(x=data['Maxpulse'])
plt.title("Box Plot for Maxpulse (Outlier Detection)")
plt.show()

# You can use the IQR method to identify outliers if needed
Q1 = data['Maxpulse'].quantile(0.25)
Q3 = data['Maxpulse'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
outliers = data[(data['Maxpulse'] < lower_bound) | (data['Maxpulse'] >
upper_bound)]
print("\nOutliers in Maxpulse:")
print(outliers)
```

2)

```
import random
# Sample data for area plot
years = [2010, 2011, 2012, 2013, 2014]
sales = [200, 300, 450, 350, 500]
# Sample data for box plot
category1 = [random.randint(1, 50) for _ in range(50)]
category2 = [random.randint(25, 75) for _ in range(50)]
category3 = [random.randint(50, 100) for _ in range(50)]
# Sample data for scatter plot
x = [random.uniform(0, 10) for _ in range(50)]
y = [random.uniform(0, 10) for _ in range(50)]
# Sample data for heatmap
import numpy as np
data = np.random.rand(5, 5)
# Sample data for regression plot
height = [160, 165, 170, 175, 180, 185]
weight = [60, 65, 70, 75, 80, 85]
```

Use the above code to generate sample data and then create the following:
1. Using the sample data for years and sales, create an area plot to visualize the trend in sales over the years. What insights can you gather from this area plot (answer as a comment)?

2. Utilizing the data in category1, category2, and category3, create a box plot using Matplotlib. How does the box plot reveal the distribution and potential outliers in these three categories (answer as a comment)?
3. Using the generated data for x and y, create a scatter plot with Matplotlib. What patterns or correlations, if any, can you observe between the x and y values in this scatter plot (answer as a comment)?
4. Employ the sample data to create a heatmap using Seaborn. What does the heatmap convey about the relationships between the values in the data matrix (answer as a comment)?
5. With the height and weight data, generate a regression plot using Seaborn. What conclusions can be drawn about the relationship between height and weight from this plot (answer as a comment)?

ANS)

```
import random
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Sample data for area plot
years = [2010, 2011, 2012, 2013, 2014]
sales = [200, 300, 450, 350, 500]

# Sample data for box plot
category1 = [random.randint(1, 50) for _ in range(50)]
category2 = [random.randint(25, 75) for _ in range(50)]
category3 = [random.randint(50, 100) for _ in range(50)]

# Sample data for scatter plot
x = [random.uniform(0, 10) for _ in range(50)]
y = [random.uniform(0, 10) for _ in range(50)]

# Sample data for heatmap
data = np.random.rand(5, 5)

# Sample data for regression plot
height = [160, 165, 170, 175, 180, 185]
weight = [60, 65, 70, 75, 80, 85]

# 1. Area Plot for Sales Over the Years
plt.figure(figsize=(8, 4))
plt.fill_between(years, sales, alpha=0.6, color='b')
plt.plot(years, sales, marker='o', color='b')
plt.xlabel('Years')
plt.ylabel('Sales')
plt.title('Sales Trend Over the Years (Area Plot)')
plt.grid(True)
plt.show()

# Insights: The area plot shows an increasing trend in sales from 2010 to
2014, with some fluctuations. Overall, sales seem to be on an upward
trajectory.

# 2. Box Plot for Categories
```

```
plt.figure(figsize=(8, 6))
data_to_plot = [category1, category2, category3]
plt.boxplot(data_to_plot, labels=['Category 1', 'Category 2', 'Category
3'])
plt.xlabel('Categories')
plt.ylabel('Values')
plt.title('Box Plot of Categories')
plt.grid(True)
plt.show()

# Box plots reveal the distribution of data within each category and help
identify potential outliers. Category 3 has a wider distribution than
Category 1 and Category 2, and it also contains potential outliers.

# 3. Scatter Plot for x and y
plt.figure(figsize=(8, 6))
plt.scatter(x, y, alpha=0.6)
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Scatter Plot of X vs. Y')
plt.grid(True)
plt.show()

# There doesn't seem to be a strong pattern or correlation between the X
and Y values in this scatter plot. The points appear to be scattered
randomly.

# 4. Heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(data, annot=True, cmap='YlGnBu')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Heatmap')
plt.show()

# The heatmap displays the relationships between values in the data
matrix. Darker colors indicate higher values, and lighter colors indicate
lower values. It helps visualize patterns and relationships in the data.

# 5. Regression Plot for Height vs. Weight
sns.regplot(x=height, y=weight, scatter_kws={"s": 50}, color='b')
plt.xlabel('Height (cm)')
plt.ylabel('Weight (kg)')
plt.title('Regression Plot of Height vs. Weight')
plt.grid(True)
plt.show()

# The regression plot shows a positive linear relationship between height
and weight. As height increases, weight tends to increase as well.
```

3)
1.
Create a program that validates email addresses using regex. It should check if an input string is a valid email address according to common email address rules.
The regex pattern should be common one for basic email address validation. It checks for the following:

-Starts with one or more alphanumeric characters, dots, underscores, percentage signs, plus signs, or hyphens.
-Followed by the "@" symbol.
-Followed by one or more alphanumeric characters or hyphens.
-Followed by a dot (.) and at least two or more alphabetic characters.
2.
Create a program that checks the strength of a password using regex. The program should ensure the password meets certain criteria, such as containing at least one uppercase letter, one lowercase letter, one digit, and one special character.
3.
Create a program that extracts phone numbers from a text using regex. It should find and display all valid phone numbers in the input text.
The regex pattern should account for various formats, including:
+91 1234567890
9876543210
080-12345678
+91-9876543210

ANS)

Email Address Validation:

```
import re

def validate_email(email):
    pattern = r'^[\w.%+-]+@[\w.-]+\.[a-zA-Z]{2,}$'
    if re.match(pattern, email):
        return True
    else:
        return False

# Test the function
email = input("Enter an email address: ")
if validate_email(email):
    print(f"{email} is a valid email address.")
else:
    print(f"{email} is not a valid email address.")
```

Password Strength Checker:

```
import re

def check_password_strength(password):
    # Check for at least one uppercase letter, one lowercase letter, one
digit, and one special character
    pattern = r'^(?=.*[A-Z])(?=.*[a-z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-
z\d@$!%*?&]{8,}$'
    if re.match(pattern, password):
        return True
    else:
        return False

# Test the function
password = input("Enter a password: ")
if check_password_strength(password):
    print(f"{password} is a strong password.")
else:
    print(f"{password} is not a strong password.")
```

Phone Number Extraction:

```
import re

def extract_phone_numbers(text):
    pattern = r'(\+\d{1,2}\s?)?(\d{10}|\d{2}-\d{8}|\d{3}-\d{7})'
    phone_numbers = re.findall(pattern, text)
    return phone_numbers

# Test the function
text = """
Here are some phone numbers:
+91 1234567890
9876543210
080-12345678
+91-9876543210
"""
phone_numbers = extract_phone_numbers(text)
print("Valid phone numbers found in the text:")
for number in phone_numbers:
    print(''.join(number))
```

4)
Create a program that extracts URLs from a text using regex. It should find and display all valid URLs in the input text.
Example of URL : www.google.com
Create a program that validates IP addresses using regex. It should check if an input string is a valid IPv4 address or not.
Example of IPv4 address :192.0.2.146
Create a program that extracts HTML tags from an HTML document using regex. It should find and display all HTML tags in the input text

ANS)
Extract URLs from Text:

```
import re

def extract_urls(text):
    pattern = r'https?://\S+|www\.\S+'
    urls = re.findall(pattern, text)
    return urls

# Test the function
text = """
Here are some URLs:
https://www.google.com
Visit my website at www.example.com
This is not a URL: invalid.url
"""
urls = extract_urls(text)
print("Valid URLs found in the text:")
for url in urls:
    print(url)
```

Validate IPv4 Addresses:

```
import re

def validate_ipv4(ip):
```

```
    pattern = r'^(25[0-5]|2[0-4][0-9]|[0-1]?[0-9][0-9]?)\.(25[0-5]|2[0-
4][0-9]|[0-1]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[0-1]?[0-9][0-
9]?)\.(25[0-5]|2[0-4][0-9]|[0-1]?[0-9][0-9]?)$'
    if re.match(pattern, ip):
        return True
    else:
        return False

# Test the function
ip_address = input("Enter an IPv4 address: ")
if validate_ipv4(ip_address):
    print(f"{ip_address} is a valid IPv4 address.")
else:
    print(f"{ip_address} is not a valid IPv4 address.")
```

Extract HTML Tags from HTML Document:

```
import re

def extract_html_tags(html_text):
    pattern = r'<[^>]*>'
    html_tags = re.findall(pattern, html_text)
    return html_tags

# Test the function
html_text = """
<html>
<head>
<title>Sample HTML</title>
</head>
<body>
<p>This is a <b>sample</b> HTML document.</p>
<a href="https://www.example.com">Visit Example</a>
</body>
</html>
"""
tags = extract_html_tags(html_text)
print("HTML tags found in the text:")
for tag in tags:
    print(tag)
```

5)

pip install beautifulsoup4 pandas

For the given fakepython html file, write a python program using BeautifulSoup library and perform following tasks -
1. Import beautifulsoup library. Attach the given html file
2. Scrape the given html and extract all Python related job titles and print them.
3. Extract all job titles, locations and companies and print them.
4. Create a pandas data frame with the details of python related job titles, locations and companies

ANS)

```
import pandas as pd
from bs4 import BeautifulSoup
```

```
# Load the HTML file
with open('fakepython.html', 'r', encoding='utf-8') as file:
    html_content = file.read()

# Parse the HTML content with BeautifulSoup
soup = BeautifulSoup(html_content, 'html.parser')

# Task 2: Extract and print Python-related job titles
python_job_titles = [job.get_text() for job in soup.find_all('h2',
class_='job-title') if 'python' in job.get_text().lower()]
print("Python-related Job Titles:")
for title in python_job_titles:
    print(title)

# Task 3: Extract all job titles, locations, and companies
job_details = []
for job_info in soup.find_all('div', class_='job-info'):
    title = job_info.find('h2', class_='job-title').get_text()
    location = job_info.find('p', class_='location').get_text()
    company = job_info.find('p', class_='company').get_text()
    job_details.append({
        'Title': title,
        'Location': location,
        'Company': company
    })

print("\nAll Job Titles, Locations, and Companies:")
for job in job_details:
    print(f"Title: {job['Title']}, Location: {job['Location']}, Company:
{job['Company']}")

# Task 4: Create a pandas DataFrame
df = pd.DataFrame(job_details)

# Print the DataFrame
print("\nDataFrame with Python-related Job Titles, Locations, and
Companies:")
print(df)
```

6)
For the given Quotes to Scrape.html file, write a python program using BeautifulSoup library and perform following tasks -
1. Import beautifulsoup library. Attach the given html file
2. Scrape the given html and extract all Quotes.
3. Extract all Quotes and authors and print them.
4. Create a pandas data frame with the details of Quotes and authors.

ANS)

```
pip install beautifulsoup4 pandas

import pandas as pd
from bs4 import BeautifulSoup

# Load the HTML file
with open('Quotes to Scrape.html', 'r', encoding='utf-8') as file:
    html_content = file.read()
```

```
# Parse the HTML content with BeautifulSoup
soup = BeautifulSoup(html_content, 'html.parser')

# Task 2: Extract and print all Quotes
quotes = [quote.get_text() for quote in soup.find_all('span',
class_='text')]
print("All Quotes:")
for quote in quotes:
    print(quote)

# Task 3: Extract all Quotes and authors and print them
quote_author_pairs = []
for quote_info in soup.find_all('div', class_='quote'):
    quote = quote_info.find('span', class_='text').get_text()
    author = quote_info.find('small', class_='author').get_text()
    quote_author_pairs.append({
        'Quote': quote,
        'Author': author
    })

print("\nAll Quotes and Authors:")
for pair in quote_author_pairs:
    print(f"Quote: {pair['Quote']}")
    print(f"Author: {pair['Author']}\n")

# Task 4: Create a pandas DataFrame
df = pd.DataFrame(quote_author_pairs)

# Print the DataFrame
print("DataFrame with Quotes and Authors:")
print(df)
```

7)
Write a program to create a Model using linear regression to predict the charges of insurance using the csv file provided named "insurance.csv". Do the required process in the data before making a model. Find predicted values, co-efficients, intercept and mean squared erro

ANS)

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score

# Step 1: Load the dataset
data = pd.read_csv('insurance.csv')

# Step 2: Data Preprocessing
# Convert categorical variables (e.g., 'sex', 'smoker', 'region') into
numerical format (one-hot encoding)
data = pd.get_dummies(data, columns=['sex', 'smoker', 'region'],
drop_first=True)

# Step 3: Split the data into input features (X) and target variable (y)
X = data.drop('charges', axis=1)
```

```
y = data['charges']

# Step 4: 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=42)

# Step 5: Create and train the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Step 6: Make predictions on the test set
y_pred = model.predict(X_test)

# Step 7: Get coefficients and intercept of the linear regression model
coefficients = model.coef_
intercept = model.intercept_

# Step 8: Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)

# Step 9: Print results
print("Coefficients:", coefficients)
print("Intercept:", intercept)
print("Mean Squared Error (MSE):", mse)

# Step 10: Calculate R-squared (R2) for model evaluation
r2 = r2_score(y_test, y_pred)
print("R-squared (R2):", r2)
```

8)
Consider variables x and y created from a pandas dataframe "car.csv" . Create new column named "Age_car" (Age_car=2023-year) For multiple linear regression problem, x contains the independent variables (Age_car , Driven_kms , Fuel_Type , Selling_type , Transmission) and y contains the dependent (Selling_Price) variable which is to be predicted.Write a Python program to spilt x and y into training and testing datasets with a 20% split. Then create a multiple linear regression model using the training data and print its coefficients ,intercept and mean squared error.

ANS)

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Step 1: Load the dataset
data = pd.read_csv('car.csv')

# Step 2: Create the 'Age_car' column
data['Age_car'] = 2023 - data['year']

# Step 3: Define independent variables (x) and dependent variable (y)
x = data[['Age_car', 'Driven_kms', 'Fuel_Type', 'Selling_Type',
'Transmission']]
y = data['Selling_Price']
```

```
# Step 4: Split data into training and testing sets (80% train, 20% test)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
random_state=42)

# Step 5: Create and train the multiple linear regression model
model = LinearRegression()
model.fit(x_train, y_train)

# Step 6: Get coefficients and intercept of the linear regression model
coefficients = model.coef_
intercept = model.intercept_

# Step 7: Make predictions on the test set
y_pred = model.predict(x_test)

# Step 8: Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)

# Step 9: Print results
print("Coefficients:", coefficients)
print("Intercept:", intercept)
print("Mean Squared Error (MSE):", mse)
```

10)
Write a Python Program using Memory Matching Game using the SimpleGUICS2Pygame library. The game involves flipping cards and matching pairs of cards with the same number. Below are instructions and explanations for the code:

(1)
Initializing global variables:
The new_game() function initializes several global variables used in the game. These variables are:
deck: A list representing the deck of cards, where each card contains a number from 0 to 7 (duplicated to have a pair of each number).
exposed: A list representing the state of each card. If exposed[i] is True, it means the card at index i is currently face-up, otherwise, it's face-down.
state: An integer representing the game state. It can take three values: 0, 1, or 2.
cIndex1 and cIndex2: Integer variables representing the indices of the two currently flipped cards.

(2)
Create new_game() function:
This function is called to start a new game or reset the current game. It initializes the variables and shuffles the deck to randomize the card positions.

(3) Event Handlers:
mouseclick(pos): This event handler is called whenever the player clicks on a card. The pos parameter represents the position of the mouse click. The function first determines which card was clicked (by dividing the x-coordinate of the click position by 50), and then it applies game logic based on the current state (state).
If the clicked card is face-down (not exposed), it behaves differently depending on the current state.
State 0: Just started - Flip the first card, update the state to 1, and mark the card as exposed.

State 1: One card flipped - Flip the second card, check for a match, update the state to 2.
State 2: Two cards flipped - If the two cards do not match, flip them back (mark them as not exposed). Then, flip the new card, update the state to 1, and mark it as exposed.
If the clicked card is already face-up (exposed), do nothing.
(4)
Creating the GUI:
A frame is created with the title "Memory Game" and a size of 800x100 pixels.
A "Reset" button is added to the frame.
Event handlers are registered to handle mouse clicks (mouseclick) and drawing (draw) on the canvas

ANS)

```
pip install SimpleGUICS2Pygame

import SimpleGUICS2Pygame.simpleguics2pygame as simplegui
import random

# Initialize global variables
deck = []
exposed = []
state = 0
cIndex1 = None
cIndex2 = None

# Create a new game
def new_game():
    global deck, exposed, state, cIndex1, cIndex2
    deck = range(8) * 2
    random.shuffle(deck)
    exposed = [False] * 16
    state = 0
    cIndex1 = None
    cIndex2 = None

# Event handler for mouse clicks
def mouseclick(pos):
    global state, cIndex1, cIndex2
    card_index = pos[0] // 50

    if not exposed[card_index]:
        if state == 0:
            exposed[card_index] = True
            cIndex1 = card_index
            state = 1
        elif state == 1:
            exposed[card_index] = True
            cIndex2 = card_index
            state = 2
        else:
            if deck[cIndex1] != deck[cIndex2]:
                exposed[cIndex1] = False
                exposed[cIndex2] = False
            exposed[card_index] = True
            cIndex1 = card_index
            state = 1
```

```
# Event handler for drawing on the canvas
def draw(canvas):
    for i in range(16):
        if exposed[i]:
            canvas.draw_text(str(deck[i]), (i * 50 + 15, 60), 30,
'White')
        else:
            canvas.draw_polygon([(i * 50, 0), (i * 50, 100), ((i + 1) *
50, 100), ((i + 1) * 50, 0)], 1, 'Black', 'Green')

# Create a frame
frame = simplegui.create_frame("Memory Game", 800, 100)

# Add a Reset button
frame.add_button("Reset", new_game, 100)

# Register event handlers
frame.set_mouseclick_handler(mouseclick)
frame.set_draw_handler(draw)

# Start a new game
new_game()

# Start the frame
frame.start()
```

11)
House Reveal Game Question:
You have been assigned the task of creating a House Reveal Game using the SimpleGUICS2Pygame library in Python. In this game, players can interactively reveal and hide different parts of a house by clicking on buttons. The house consists of three parts: a main body, a roof (triangle), and a circular window.
Instructions:
1. The canvas size should be set to 400x400 pixels.
2. The house is drawn on the canvas using geometric shapes. The main body of the house is a rectangle with a width of 100 pixels and a height of 100 pixels. The roof is a triangle that fits perfectly on top of the main body. The circular window has a radius of 15 pixels.
3. Define three boolean variables house_visible, roof_visible, and window_visible. These variables control the visibility of different parts of the house. When a part is visible, it will appear in the specified color when the corresponding button is clicked.
4. Insert draw_house() function. This function handles the drawing of the house on the canvas, including the main body, roof, and circular window.
5. The program should involve use of three buttons labeled "Reveal House (Red)", "Reveal Roof (Yellow)" and "Reveal Window (Blue)" to allow players to toggle the visibility and color of the corresponding house part.

6. When a button is clicked, the corresponding part of the house is revealed in the
chosen color and also clicking the same button again will hide the corresponding part.
7. First, the main body should be revealed and only then the other parts should be
revealed

ANS)

```
import SimpleGUICS2Pygame.simpleguics2pygame as simplegui

# Initialize canvas size and variables
canvas_width = 400
canvas_height = 400

house_visible = False
roof_visible = False
window_visible = False

# Function to draw the house
def draw_house(canvas):
    global house_visible, roof_visible, window_visible

    # Draw main body of the house
    if house_visible:
        canvas.draw_polygon([(100, 300), (100, 200), (200, 200), (200,
300)], 2, 'Red', 'Red')

    # Draw the roof
    if roof_visible:
        canvas.draw_polygon([(100, 200), (200, 200), (150, 100)], 2,
'Yellow', 'Yellow')

    # Draw the window
    if window_visible:
        canvas.draw_circle((150, 250), 15, 2, 'Blue', 'Blue')

# Button handlers to toggle visibility
def reveal_house():
    global house_visible
    house_visible = not house_visible

def reveal_roof():
    global roof_visible
    roof_visible = not roof_visible

def reveal_window():
    global window_visible
    window_visible = not window_visible

# Create a frame
frame = simplegui.create_frame("House Reveal Game", canvas_width,
canvas_height)

# Create buttons
frame.add_button("Reveal House (Red)", reveal_house, 150)
frame.add_button("Reveal Roof (Yellow)", reveal_roof, 150)
frame.add_button("Reveal Window (Blue)", reveal_window, 150)
```

```
# Set the draw handler
frame.set_draw_handler(draw_house)

# Start the frame
frame.start()
```

12)
Write a program using SimpleGUICS2Pygame module of Python using the following instructions:
Create three buttons: circles, triangles, and squares.
Upon clicking the circles button, 10 circles (with different colours) should appear on the canvas at random positions. Every one second, their positions should randomly keep changing but they should remain within the canvas.
Upon clicking the triangles button, 10 triangles (with different colours) should appear on the canvas at random positions. Every one second, their positions should randomly keep changing but they should remain within the canvas.
Upon clicking the squares button, 10 squares (with different colours) should appear on the canvas at random positions. Every one second, their positions should randomly keep changing but they should remain within the canvas.
At any given time, if any of the shape's button is clicked again, then that shape should stop appearing on the canvas.
E.g., if the user has clicked circles and circles are visible on the canvas and the user clicks circles again, then circles should disappear from the canvas. So, if circles are not visible then clicking the button should make them visible and if circles are visible then clicking the button should make them disappear.
Displaying multiple shapes at the same time on the screen should also work.
E.g., if the user clicks circles and then clicks triangles, then both, circles and triangles should appear on the canvas

ANS)
```
import SimpleGUICS2Pygame.simpleguics2pygame as simplegui
import random
import math

# Initialize canvas size and variables
canvas_width = 400
canvas_height = 400

shapes = []  # List to store all shapes (circles, triangles, squares)
shapes_visible = {'circles': False, 'triangles': False, 'squares': False}

# Function to create a random color
def random_color():
    return "rgb(" + str(random.randint(0, 255)) + "," +
str(random.randint(0, 255)) + "," + str(random.randint(0, 255)) + ")"

# Function to create random shapes
def create_shapes(shape_type):
    global shapes
    for _ in range(10):
        if shape_type == 'circles':
            radius = random.randint(10, 30)
```

```
            x = random.randint(radius, canvas_width - radius)
            y = random.randint(radius, canvas_height - radius)
            velocity = [random.uniform(-1, 1), random.uniform(-1, 1)]
            color = random_color()
            shapes.append(('circle', (x, y), velocity, radius, color))
        elif shape_type == 'triangles':
            x1 = random.randint(0, canvas_width)
            y1 = random.randint(0, canvas_height)
            x2 = random.randint(0, canvas_width)
            y2 = random.randint(0, canvas_height)
            x3 = random.randint(0, canvas_width)
            y3 = random.randint(0, canvas_height)
            velocity = [random.uniform(-1, 1), random.uniform(-1, 1)]
            color = random_color()
            shapes.append(('triangle', (x1, y1, x2, y2, x3, y3),
velocity, color))
        elif shape_type == 'squares':
            side_length = random.randint(20, 40)
            x = random.randint(0, canvas_width - side_length)
            y = random.randint(0, canvas_height - side_length)
            velocity = [random.uniform(-1, 1), random.uniform(-1, 1)]
            color = random_color()
            shapes.append(('square', (x, y), velocity, side_length,
color))

# Function to update shape positions
def update_shapes():
    for shape in shapes:
        if shape[0] == 'circle':
            x, y = shape[1]
            velocity = shape[2]
            radius = shape[3]
            x += velocity[0]
            y += velocity[1]

            if x < radius or x > canvas_width - radius:
                velocity[0] = -velocity[0]
            if y < radius or y > canvas_height - radius:
                velocity[1] = -velocity[1]

            shape[1] = (x, y)

        elif shape[0] == 'triangle':
            x1, y1, x2, y2, x3, y3 = shape[1]
            velocity = shape[2]
            x1 += velocity[0]
            y1 += velocity[1]
            x2 += velocity[0]
            y2 += velocity[1]
            x3 += velocity[0]
            y3 += velocity[1]

            if x1 < 0 or x1 > canvas_width or y1 < 0 or y1 >
canvas_height:
                velocity[0] = -velocity[0]
                velocity[1] = -velocity[1]
```

```
            if x2 < 0 or x2 > canvas_width or y2 < 0 or y2 >
canvas_height:
                velocity[0] = -velocity[0]
                velocity[1] = -velocity[1]

            if x3 < 0 or x3 > canvas_width or y3 < 0 or y3 >
canvas_height:
                velocity[0] = -velocity[0]
                velocity[1] = -velocity[1]

            shape[1] = (x1, y1, x2, y2, x3, y3)

        elif shape[0] == 'square':
            x, y = shape[1]
            velocity = shape[2]
            side_length = shape[3]
            x += velocity[0]
            y += velocity[1]

            if x < 0 or x > canvas_width - side_length:
                velocity[0] = -velocity[0]
            if y < 0 or y > canvas_height - side_length:
                velocity[1] = -velocity[1]

            shape[1] = (x, y)

# Function to draw shapes on the canvas
def draw_shapes(canvas):
    for shape in shapes:
        if shape[0] == 'circle':
            x, y = shape[1]
            radius = shape[3]
            color = shape[4]
            canvas.draw_circle((x, y), radius, 2, color, color)
        elif shape[0] == 'triangle':
            coords = shape[1]
            color = shape[3]
            canvas.draw_polygon(coords, 2, color, color)
        elif shape[0] == 'square':
            x, y = shape[1]
            side_length = shape[3]
            color = shape[4]
            canvas.draw_polygon([(x, y), (x + side_length, y), (x +
side_length, y + side_length), (x, y + side_length)], 2, color, color)

# Button handlers to toggle shape visibility
def toggle_circles():
    global shapes_visible
    shapes_visible['circles'] = not shapes_visible['circles']
    if shapes_visible['circles']:
        create_shapes('circles')
    else:
        shapes[:] = [shape for shape in shapes if shape[0] != 'circle']

def toggle_triangles():
    global shapes_visible
    shapes_visible['triangles'] = not shapes_visible['triangles']
    if shapes_visible['triangles']:
```

```
        create_shapes('triangles')
    else:
        shapes[:] = [shape for shape in shapes if shape[0] != 'triangle']

def toggle_squares():
    global shapes_visible
    shapes_visible['squares'] = not shapes_visible['squares']
    if shapes_visible['squares']:
        create_shapes('squares')
    else:
        shapes[:] = [shape for shape in shapes if shape[0] != 'square']

# Create a frame
frame = simplegui.create_frame("Shapes Game", canvas_width,
canvas_height)

# Create buttons
frame.add_button("Circles", toggle_circles, 100)
frame.add_button("Triangles", toggle_triangles, 100)
frame.add_button("Squares", toggle_squares, 100)

# Set the draw handler
frame.set_draw_handler(draw_shapes)

# Timer to update shape positions
timer_interval = 1000  # 1 second
timer = simplegui.create_timer(timer_interval, update_shapes)

# Start the frame and timer
frame.start()
timer.start()
```

13)
Write a program to create Tic Tac Toe game using the SimpleGUICS2Pygame module in Python.
Game Instructions –
1. The game board consists of a 3x3 grid, and two players take turns to place their symbols ('X' or 'O') on the board until one player wins or the game ends in a draw.
2. The player who places three of their symbols in a horizontal, vertical, or diagonal line wins the game.
3. The completed game should display the Tic Tac Toe board, allow players to make moves by clicking on the board, and correctly display the winner on the canvas when the game is over.
4. The game should also have a "New Game" button to reset the board and start a new game.
5. Ensure the characters ('X' and 'O') are centered correctly within each cell of the game board.
Assessment Tasks –
1. Implement the draw_board function to display the Tic Tac Toe board, characters, and lines on the canvas.
2. Implement the mouseclick function to allow players to make moves when they click on an empty cell on the board.
3. Implement the check_winner function to check for a winning combination on the board after each move.
4. Display the winner's symbol ("X" or "O") on the canvas when the game is over.

5. Add functionality to the "New Game" button, so it resets the board and starts a new game when clicked.
6. Ensure the game board and characters are visually appealing and centered correctly.

ANS)

```
import SimpleGUICS2Pygame.simpleguics2pygame as simplegui

# Initialize canvas size and variables
canvas_width = 300
canvas_height = 300
cell_size = canvas_width // 3
board = [['' for _ in range(3)] for _ in range(3)]
current_player = 'X'
winner = None
game_over = False

# Function to draw the Tic Tac Toe board
def draw_board(canvas):
    for row in range(3):
        for col in range(3):
            x = col * cell_size + cell_size / 2
            y = row * cell_size + cell_size / 2
            symbol = board[row][col]
            canvas.draw_text(symbol, (x, y), 48, 'White')

    for i in range(1, 3):
        canvas.draw_line((cell_size * i, 0), (cell_size * i,
canvas_height), 2, 'White')
        canvas.draw_line((0, cell_size * i), (canvas_width, cell_size *
i), 2, 'White')

    if winner:
        canvas.draw_text(f"Player {winner} wins!", (canvas_width // 2 -
80, canvas_height // 2), 24, 'White')
    elif game_over:
        canvas.draw_text("It's a draw!", (canvas_width // 2 - 40,
canvas_height // 2), 24, 'White')

# Function to check for a winning combination
def check_winner():
    global winner, game_over
    # Check rows
    for row in board:
        if row[0] == row[1] == row[2] and row[0] != '':
            winner = row[0]
            game_over = True
            return True
    # Check columns
    for col in range(3):
        if board[0][col] == board[1][col] == board[2][col] and
board[0][col] != '':
            winner = board[0][col]
            game_over = True
            return True
    # Check diagonals
    if board[0][0] == board[1][1] == board[2][2] and board[0][0] != '':
```

```
        winner = board[0][0]
        game_over = True
        return True
    if board[0][2] == board[1][1] == board[2][0] and board[0][2] != '':
        winner = board[0][2]
        game_over = True
        return True
    # Check for a draw
    if all(all(cell != '' for cell in row) for row in board):
        game_over = True
    return False

# Function to handle mouse clicks
def mouseclick(pos):
    global current_player
    if not game_over:
        row = pos[1] // cell_size
        col = pos[0] // cell_size
        if board[row][col] == '':
            board[row][col] = current_player
            if current_player == 'X':
                current_player = 'O'
            else:
                current_player = 'X'
            if check_winner():
                return

# Function to start a new game
def new_game():
    global board, current_player, winner, game_over
    board = [['' for _ in range(3)] for _ in range(3)]
    current_player = 'X'
    winner = None
    game_over = False

# Create a frame
frame = simplegui.create_frame("Tic Tac Toe", canvas_width,
canvas_height)

# Create a button to start a new game
frame.add_button("New Game", new_game, 100)

# Set the draw handler
frame.set_draw_handler(draw_board)

# Set the mouseclick handler
frame.set_mouseclick_handler(mouseclick)

# Start the frame
frame.start()
```

14)
Write a python program for shape shifting by using key down handler. Use SimpleGUICS2Pygame library.
height and width of the frame should be 200.
shapes = ["Square", "Circle", "Triangle"]

if user press d from keyboard then shape will change from left to right direction means shape will change from square to circle and circle to triangle.
if user press s from keyboard then shape will change from right to left direction means shape will change from triangle to circle and circle to square.
Draw shapes in the center of the frame with suitable dimension.
colors = ["DeepPink", "Red", "DarkOrange", "Yellow", "Lime", "Green", "Blue", "Aqua", "Purple", "Magenta"]
if user press v from keyboard then fill color will change from left to right direction.
if user press c from keyboard then fill color will change from right to left direction.
if user press x from keyboard then size of shapes will increase 10.
if user press z from keyboard then size of shapes will decrease 10.
if user press f from the keyboard then color should fill in the shapes and if user press again f from keyboard then fill color should remove from the shape

ANS)

```
import SimpleGUICS2Pygame.simpleguics2pygame as simplegui
import math

# Initialize frame size and variables
frame_width = 200
frame_height = 200
shape_index = 0
color_index = 0
fill_color = False
shape_size = 40

# List of shapes and colors
shapes = ["Square", "Circle", "Triangle"]
colors = ["DeepPink", "Red", "DarkOrange", "Yellow", "Lime", "Green",
"Blue", "Aqua", "Purple", "Magenta"]

# Function to draw the selected shape
def draw_shape(canvas):
    global shape_index, shape_size, fill_color

    canvas_width = canvas.get_canvas_textwidth(shapes[shape_index], 20)
    canvas_height = canvas.get_canvas_textwidth(shapes[shape_index], 20)
    x = frame_width / 2 - canvas_width / 2
    y = frame_height / 2 + canvas_height / 2

    if shapes[shape_index] == "Square":
        if fill_color:
            canvas.draw_polygon([(x, y), (x + shape_size, y), (x +
shape_size, y - shape_size), (x, y - shape_size)], 2, "Black",
colors[color_index])
        else:
            canvas.draw_polygon([(x, y), (x + shape_size, y), (x +
shape_size, y - shape_size), (x, y - shape_size)], 2, "Black")
    elif shapes[shape_index] == "Circle":
        if fill_color:
            canvas.draw_circle((x + shape_size / 2, y - shape_size / 2),
shape_size / 2, 2, "Black", colors[color_index])
        else:
```

```
            canvas.draw_circle((x + shape_size / 2, y - shape_size / 2),
shape_size / 2, 2, "Black")
    elif shapes[shape_index] == "Triangle":
        if fill_color:
            canvas.draw_polygon([(x, y), (x + shape_size, y), (x +
shape_size / 2, y - math.sqrt(3) * shape_size / 2)], 2, "Black",
colors[color_index])
        else:
            canvas.draw_polygon([(x, y), (x + shape_size, y), (x +
shape_size / 2, y - math.sqrt(3) * shape_size / 2)], 2, "Black")

# Keydown handler for shape shifting, color changing, and size
modification
def keydown_handler(key):
    global shape_index, color_index, fill_color, shape_size
    if key == 'd':
        shape_index = (shape_index + 1) % len(shapes)
    elif key == 's':
        shape_index = (shape_index - 1) % len(shapes)
    elif key == 'v':
        color_index = (color_index + 1) % len(colors)
    elif key == 'c':
        color_index = (color_index - 1) % len(colors)
    elif key == 'x':
        shape_size += 10
    elif key == 'z':
        if shape_size > 10:
            shape_size -= 10
    elif key == 'f':
        fill_color = not fill_color

# Create a frame
frame = simplegui.create_frame("Shape Shifting", frame_width,
frame_height)

# Set the draw handler
frame.set_draw_handler(draw_shape)

# Set the keydown handler
frame.set_keydown_handler(keydown_handler)

# Start the frame
frame.start()
```

15)
1. Task: Create a new Django project named "SampleApp."
2. Task: Create a Django app within the project named "sample"
3. Task: Ensure that Django is properly installed and the project can run without errors using the development server.

1. Task: Define three URL patterns in the "sample" app's `urls.py` file:
- '/' should route to the 'home' view.
- '/about/' should route to the 'about' view.
- '/contact/' should route to the 'contact' view.
2. Task: Ensure that each URL pattern is named 'home,' 'about,' and 'contact' respectively.

1. Task: Create three views in the "sample" app's `views.py` file:
- 'home' view should render the 'sample/home.html' template.
- 'about' view should render the 'sample/about.html' template.
- 'contact' view should render the 'sample/contact.html' template.

1. Task: Create HTML templates for the 'home,' 'about,' and 'contact' views in the 'templates/sample' directory.
2. Task: The 'home.html' template should display a welcoming message.
3. Task: The 'about.html' template should contain information about the project or organization.
4. Task: The 'contact.html' template should provide contact information.
- Register the app in the project settings.

1. Task: Include the 'sample' app's URLs in the project's 'urls.py' file.
2. Task: Create a URL pattern that routes the root URL ('/') to the 'home' view.

1. Task: Run migrations to create the necessary database tables.
2. Task: Start the development server and ensure that the project is accessible in a web browser.
3. Task: Verify that the 'home,' 'about,' and 'contact' pages are accessible at the expected URLs.

1. Task: Implement additional functionality, such as creating a '404 Not Found' page and linking it to an invalid URL.
2. Task: Add a navigation menu or links to navigate between the 'home,' 'about,' and 'contact' pages.
D

ANS)
Step 1: Create a new Django project named "SampleApp"
```
  django-admin startproject SampleApp
```

Step 2: Create a Django app within the project named "sample."
```
  cd SampleApp
  python manage.py startapp sample
```

Step 3: Ensure that Django is properly installed and the project can run without errors using the development server.
```
      python manage.py runserver
```

Access the Django development server at http://localhost:8000/ to make sure it runs without errors.

Step 4: Define URL patterns in the "sample" app's urls.py file:

sample/urls.py:

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

urlpatterns = [
    path('', views.home, name='home'),
    path('about/', views.about, name='about'),
    path('contact/', views.contact, name='contact'),
]
```

Step 5: Create three views in the "sample" app's views.py file:

sample/views.py:

```
from django.shortcuts import render

def home(request):
    return render(request, 'sample/home.html')

def about(request):
    return render(request, 'sample/about.html')

def contact(request):
    return render(request, 'sample/contact.html')
```

Step 6: Create HTML templates for the 'home,' 'about,' and 'contact' views in the 'templates/sample' directory.

Create a directory named "sample" inside the "templates" directory if it doesn't exist. Then create the following HTML templates:

templates/sample/home.html:

```
<!DOCTYPE html>
<html>
<head>
    <title>Welcome to SampleApp</title>
</head>
<body>
    <h1>Welcome to SampleApp!</h1>
</body>
</html>
```

templates/sample/about.html:

```
<!DOCTYPE html>
<html>
<head>
    <title>About Us</title>
</head>
<body>
    <h1>About Us</h1>
    <p>Learn more about our project or organization here.</p>
</body>
</html>
```

templates/sample/contact.html:

```
<!DOCTYPE html>
<html>
<head>
    <title>Contact Us</title>
</head>
<body>
    <h1>Contact Us</h1>
    <p>Contact information goes here.</p>
</body>
</html>
```

Step 7: Register the app in the project settings.

In the project's settings.py file, add 'sample' to the INSTALLED_APPS list:

```
INSTALLED_APPS = [
    # ...
    'sample',
    # ...
]
```

Step 8: Include the 'sample' app's URLs in the project's 'urls.py' file.

In the project's urls.py file, include the app's URLs:

SampleApp/urls.py:

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

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

Step 9: Run migrations to create the necessary database tables.

```
python manage.py migrate
```

Step 10: Start the development server and ensure that the project is accessible in a web browser.

```
python manage.py runserver
```

Access the project at http://localhost:8000/ in your web browser.

Step 11: Verify that the 'home,' 'about,' and 'contact' pages are accessible at the expected URLs:

Home: http://localhost:8000/
About: http://localhost:8000/about/
Contact: http://localhost:8000/contact/

Step 12: Implement additional functionality, such as creating a '404 Not Found' page and linking it to an invalid URL, and adding a navigation menu or links to navigate between the 'home,' 'about,' and 'contact' pages. You can do this by editing the templates and views as needed.

16)
a. Create a new Django project named "Bookstore."
b. Set up a Django app named "books."

a. Define a Django model named "Book" with the following fields:
- **Title (CharField)**
- **Author (CharField)**
- **Published Date (DateField)**
- **Price (DecimalField)**
- **ISBN (CharField)**

b. Create and apply the necessary database migrations to create the "Book" model.

a. Register the "Book" model in the Django admin panel.
b. Create a superuser account with the username and a password .

c. Use the admin panel to add at least three sample books with different details.

a. Create a view to display a list of all books in the database. Use a template to render this list.
b. Create a view to display detailed information about a single book, including all its fields.
c. Create templates for both views, ensuring they have appropriate HTML structure.

a. Define URL patterns to route requests to the views you created in Task 4.
b. Implement a homepage that displays a list of all books.
c. Implement URLs for displaying detailed book information.

a. Implement a search functionality that allows users to search for books by title or author.

ANS)

Step 1: Create a new Django project named "Bookstore."

```
django-admin startproject Bookstore
```

Step 2: Set up a Django app named "books."

```
cd Bookstore
python manage.py startapp books
```

Step 3: Define a Django model named "Book" with the specified fields in the "books/models.py" file.

```
from django.db import models

class Book(models.Model):
    title = models.CharField(max_length=100)
    author = models.CharField(max_length=100)
    published_date = models.DateField()
    price = models.DecimalField(max_digits=10, decimal_places=2)
    isbn = models.CharField(max_length=13)

    def __str__(self):
        return self.title
```

Step 4: Create and apply the necessary database migrations to create the "Book" model.

```
python manage.py makemigrations books
python manage.py migrate
```

Step 5: Register the "Book" model in the Django admin panel.
In the "books/admin.py" file, register the "Book" model as follows:

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

admin.site.register(Book)
```

Step 6: Create a superuser account with the following command:

```
python manage.py createsuperuser
```

Follow the prompts to create a superuser account with a username and password.

Step 7: Use the admin panel to add at least three sample books with different details.

Access the Django admin panel at http://localhost:8000/admin/ and log in with the superuser account. Then, use the admin panel to add sample books with their details.

Step 8: Create views and templates for displaying the list of books and detailed book information.

a. Create a view to display a list of all books in the "books/views.py" file:

```
from django.shortcuts import render
from .models import Book

def book_list(request):
    books = Book.objects.all()
    return render(request, 'books/book_list.html', {'books': books})
```

b. Create a view to display detailed information about a single book:

```
from django.shortcuts import render, get_object_or_404
from .models import Book

def book_detail(request, book_id):
    book = get_object_or_404(Book, pk=book_id)
    return render(request, 'books/book_detail.html', {'book': book})
```

c. Create templates for both views in a "templates/books" directory.

Create "templates/books/book_list.html" for the list of books.
Create "templates/books/book_detail.html" for detailed book information.
Step 9: Define URL patterns to route requests to the views created in Task 8.

In the "books/urls.py" file, define URL patterns as follows:

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

urlpatterns = [
    path('', views.book_list, name='book_list'),
    path('<int:book_id>/', views.book_detail, name='book_detail'),
]
```

Step 10: Implement a homepage that displays a list of all books.

In the project's "Bookstore/urls.py" file, include the "books" app's URLs as follows:

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

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

Step 11: Implement URLs for displaying detailed book information.

In the homepage template (e.g., "books/book_list.html"), include links to individual book pages using the {% url 'book_detail' book.id %} template tag.

Step 12: Implement a search functionality that allows users to search for books by title or author.

a. Create a new view for searching books:

```
from django.db.models import Q
from django.shortcuts import render
from .models import Book

def book_search(request):
    query = request.GET.get('q')
    if query:
        books = Book.objects.filter(Q(title__icontains=query) |
Q(author__icontains=query))
    else:
        books = Book.objects.all()
    return render(request, 'books/book_search.html', {'books': books,
'query': query})
```

b. Define a URL pattern for the search view in the "books/urls.py" file:

```
path('search/', views.book_search, name='book_search'),
```

c. Create a template for the search results (e.g., "books/book_search.html") that displays the search form and search results.

With these steps, you'll have a Django project named "Bookstore" with a "books" app that includes a "Book" model, admin panel integration, views for listing and displaying book details, and a search functionality.

17)

• Create python Django project with name 'moviereview'
• Create an app called movie
• Create home.html file in movieapp.
• Code for home.html

```
<body>
<h1>My movie app </h1>
<h3>Enter data </h3>
<form action="" >
<label for="data">Data:</label>
<input type="text" name=" " ><br><br>
<button type="submit" >Search</button>
</form>
</body>
```

• Create model named Movie with attributes Title,Actor,Date of Relaease.
• Create super user with your enrollment number and password will be your name.(it is compulsory)
• Log in to the django admin portal with this user and Enter the following data in Movie table.

Title	Actor	Date of Release
JAWAN	SRK	8-Sept-2023
GADAR-2	SunnyD	25-Aug-2023

OH MY GOD-2 Akshay K 18-Aug-2023
• Make necessary adjustment to your code to let user search for data from this database by Title on home page

ANS)
Step 1: Create a Django project named "moviereview."
```
django-admin startproject moviereview
```

Step 2: Create an app called "movie."
```
cd moviereview
python manage.py startapp movie
```

Step 3: Create a "home.html" file in the "movie" app's "templates/movie" directory.

Here's the code for "home.html":
```
<!DOCTYPE html>
<html>
<head>
    <title>My Movie App</title>
</head>
<body>
    <h1>My Movie App</h1>
    <h3>Enter data</h3>
    <form action="" method="get">
        <label for="title">Title:</label>
        <input type="text" name="title"><br><br>
        <button type="submit">Search</button>
    </form>
</body>
</html>
```

Step 4: Create a model named "Movie" with attributes Title, Actor, and Date of Release in the "movie/models.py" file.
```
from django.db import models

class Movie(models.Model):
    title = models.CharField(max_length=100)
    actor = models.CharField(max_length=100)
    date_of_release = models.DateField()

    def __str__(self):
        return self.title
```

Step 5: Create a superuser with your enrollment number as the username and your name as the password.
```
python manage.py createsuperuser
```

Follow the prompts to create the superuser account.

Step 6: Log in to the Django admin portal with the superuser account and enter the following data in the "Movie" table:

Title: JAWAN
Actor: SRK
Date of Release: 2023-09-08

Title: GADAR-2

```
Actor: SunnyD
Date of Release: 2023-08-25

Title: OH MY GOD-2
Actor: Akshay K
Date of Release: 2023-08-18
```

Step 7: Make necessary adjustments to your code to let users search for data by Title on the home page.

Modify the "movie/views.py" file to handle the search functionality:

```
from django.shortcuts import render
from .models import Movie

def home(request):
    title = request.GET.get('title')
    movies = Movie.objects.filter(title__icontains=title) if title else
[]
    return render(request, 'movie/home.html', {'movies': movies})
```

Update the "movie/urls.py" file to include the URL pattern for the home view:

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

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

18)

Create python Django project with name 'myproject'

- **Create an app called myapp**
- **Create home.html file in myapp.**
- **Code for home.html**

```
<body>
<h1>My app</h1>
<h3>Enter data </h3>
<form action="" >
<label for="data">Data:</label>
<input type="text" name=" " ><br><br>
<button type="submit" >Search</button>
</form>
</body>
```

- **Create model named Mydata with attributes name,branch,roll no.**
- **Create super user with your enrollment number and password will be your name.(it is compulsory)**
- **Log in to the django admin portal with this user and Enter the following data in Mydata table.**

name branch roll no
Yaksh CE 111
Rohan IT 222
Radha CST 333

- **Make necessary adjustment to your code to let user search for data from this database by name on home page.**

ANS)

Step 1: Create a Django project named "myproject."

```
django-admin startproject myproject
```

Step 2: Create an app called "myapp."
cd myproject
python manage.py startapp myapp

Step 3: Create a "home.html" file in the "myapp" app's "templates/myapp" directory.

Here's the code for "home.html":

```
<!DOCTYPE html>
<html>
<head>
    <title>My App</title>
</head>
<body>
    <h1>My App</h1>
    <h3>Enter data</h3>
    <form action="" method="get">
        <label for="name">Name:</label>
        <input type="text" name="name"><br><br>
        <button type="submit">Search</button>
    </form>
</body>
</html>
```

Step 4: Create a model named "Mydata" with attributes name, branch, and roll no. in the "myapp/models.py" file.

```
from django.db import models

class Mydata(models.Model):
    name = models.CharField(max_length=100)
    branch = models.CharField(max_length=100)
    roll_no = models.IntegerField()

    def __str__(self):
        return self.name
```

Step 5: Create a superuser with your enrollment number as the username and your name as the password.

```
python manage.py createsuperuser
```

Follow the prompts to create the superuser account.

Step 6: Log in to the Django admin portal with the superuser account and enter the following data in the "Mydata" table:

Name: Yaksh
Branch: CE
Roll No: 111

Name: Rohan
Branch: IT
Roll No: 222

Name: Radha
Branch: CST
Roll No: 333

Step 7: Make necessary adjustments to your code to let users search for data by name on the home page.

Modify the "myapp/views.py" file to handle the search functionality:

```
from django.shortcuts import render
from .models import Mydata

def home(request):
    name = request.GET.get('name')
    mydata = Mydata.objects.filter(name__icontains=name) if name else []
    return render(request, 'myapp/home.html', {'mydata': mydata})
```

Update the "myapp/urls.py" file to include the URL pattern for the home view

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

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

19)

1. Create a Django Project named "music"
2. Create an App named 'song'
3. Create Home Page by making template home.html in 'song' App.
4. Code for 'home.html' is as below.

```
<body>
<h2> Songs </h2>
<h4> Enter Song Name </h4>
<form action="" >
<label for="search">Search for Song </label>
<input type="text" name="SearchSong" />
<button type="submit" >Search</button>
</form>
</body>
```

5. Create Model with name 'Song' with attributes songname, singers, musicdirector, year.
6. Create a superuser and using the username and password, enter the details of Song from admin panel. Details are given as below.
7. Make Necessary changes to your code to show all the songs on home page ordered by year.
8. Search the particular song using search box should show the particular searched song details after clicking search button.

ANS)

Step 1: Create a Django project named "music."

```
django-admin startproject music
```

Step 2: Create an app named "song."

```
cd music
python manage.py startapp song
```

Step 3: Create a "home.html" file in the "song" app's "templates/song" directory.

Here's the code for "home.html":

```
<!DOCTYPE html>
<html>
<head>
    <title>Songs</title>
</head>
<body>
    <h2>Songs</h2>
    <h4>Enter Song Name</h4>
    <form action="" method="get">
        <label for="SearchSong">Search for Song</label>
        <input type="text" name="SearchSong">
        <button type="submit">Search</button>
    </form>
    <h3>All Songs</h3>
    <ul>
        {% for song in songs %}
            <li>{{ song.songname }} - {{ song.singers }} ({{ song.year
}})</li>
        {% endfor %}
    </ul>
</body>
</html>
```

Step 4: Create a model named "Song" with attributes songname, singers, musicdirector, and year in the "song/models.py" file.

```
from django.db import models

class Song(models.Model):
    songname = models.CharField(max_length=100)
    singers = models.CharField(max_length=100)
    musicdirector = models.CharField(max_length=100)
    year = models.PositiveIntegerField()

    def __str__(self):
        return self.songname
```

Step 5: Create a superuser with the following command:

```
python manage.py createsuperuser
```

Follow the prompts to create the superuser account.

Step 6: Log in to the Django admin panel with the superuser account and enter the details of songs from the admin panel.

Step 7: Make necessary changes to your code to show all the songs on the home page ordered by year.

Modify the "song/views.py" file to retrieve the songs ordered by year

```
from django.shortcuts import render
from .models import Song

def home(request):
    songs = Song.objects.order_by('year')
    return render(request, 'song/home.html', {'songs': songs})
```

Step 8: Implement search functionality to display the particular searched song details.

Modify the "song/views.py" file to handle the search functionality:

```
from django.shortcuts import render, get_object_or_404
from .models import Song

def home(request):
    songs = Song.objects.order_by('year')

    search_song = request.GET.get('SearchSong')
    if search_song:
        searched_song = get_object_or_404(Song,
songname__icontains=search_song)
        return render(request, 'song/home.html', {'songs': songs,
'searched_song': searched_song})

    return render(request, 'song/home.html', {'songs': songs})
```

20)

DJANGO TEMPLATE ENGINE PROJECT

Task 1: Project Setup and Template Configuration

1. Task: Verify project setup and template configuration.

- Description: Confirm that the Django project and app have been created, and that template settings in `settings.py` are correctly configured.

Task 2: Create a Basic Template

2. Task: Create a basic HTML template.

- Description: Develop a simple HTML template named `hello.html` inside the app's `templates` directory, as shown in the project setup.

Task 3: Create a View to Render the Template

3. Task: Develop a view to render the template.

- Description: Create a view function named `hello_view` in the app's `views.py` that renders the `hello.html` template.

Task 4: Define a URL Pattern for the View

4. Task: Define a URL pattern for the `hello_view` in the app's `urls.py`.

- Description: Create a URL pattern that maps to the `hello_view` function, making sure it includes the `/demo/hello/` URL path.

Task 5: Configure Main URLs

5. Task: Verify main URL configuration.

- Description: Confirm that the app's URLs are included in the main project's `urls.py` correctly.

Task 6: Start the Development Server

6. Task: Run the development server.

- Description: Start the Django development server using the command `python manage.py runserver`. Verify that the server runs without errors.

Task 7: Access the Template via URL

7. Task: Access the template via its URL.

- Description: Access the template at `http://localhost:8000/demo/hello/` using a web browser or a tool like `curl`. Ensure that the template is displayed as expected, showing "Hello, Django User!"

Task 8: Modify the Template Context

8. Task: Modify the template context.

- Description: In the `hello_view`, change the value of the `name` variable in the context to a different name (e.g., "John"). Verify that the template updates accordingly.

Task 9: Template Inheritance

9. Task: Implement template inheritance .
- Description: Create a base template that includes common elements like headers and footers. Then, create a child template that extends the base template and adds content unique to the
child template.
Task 10: Template Tags
10. Task: Explore and use additional template tags
- Description: Experiment with Django's template tags (e.g., `for`, `if`, `include`) to enhance the template's functionality or appearance

ANS)
Task 1: Project Setup and Template Configuration
Ensure you have already set up your Django project and configured the template settings in the settings.py file to include the app's templates directory.

Task 2: Create a Basic Template
Create a new HTML template file named hello.html inside the templates directory of your app. This file should contain the basic HTML structure you want to display.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Hello Template</title>
</head>
<body>
    <h1>Hello, Django User!</h1>
</body>
</html>
```

Task 3: Create a View to Render the Template
In your app's views.py, create a view function named hello_view that will render the hello.html template.

```
from django.shortcuts import render

def hello_view(request):
    return render(request, 'hello.html')
```

Task 4: Define a URL Pattern for the View
In your app's urls.py file, define a URL pattern that maps to the hello_view function, ensuring it includes the /demo/hello/ URL path.

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

urlpatterns = [
    path('demo/hello/', views.hello_view, name='hello'),
]
```

Task 5: Configure Main URLs
Make sure that your app's URLs are included in the main project's urls.py file.

Task 6: Start the Development Server
Run the Django development server using the command:

```
python manage.py runserver
```

Verify that the server starts without errors.

Task 7: Access the Template via URL
Access the template in your web browser by navigating to http://localhost:8000/demo/hello/. You should see the "Hello, Django User!" message.

Task 8: Modify the Template Context
In the hello_view function, you can modify the template context to change the displayed name. For example:

```
from django.shortcuts import render

def hello_view(request):
    context = {'name': 'John'}
    return render(request, 'hello.html', context)
```

This will update the template to say "Hello, John!"

Task 9: Template Inheritance
To implement template inheritance, create a base template that includes common elements, such as headers and footers. Then, create a child template that extends the base template and adds unique content.

Here's an example:

base.html (Base Template)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>{% block title %}My Site{% endblock %}</title>
</head>
<body>
    <header>
        <h1>My Website</h1>
    </header>

    <nav>
        <ul>
            <li><a href="/">Home</a></li>
            <li><a href="/about/">About</a></li>
            <li><a href="/contact/">Contact</a></li>
        </ul>
    </nav>

    <main>
        {% block content %}
        {% endblock %}
    </main>

    <footer>
        &copy; 2023 My Website
    </footer>
</body>
</html>
```

child.html (Child Template)

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

{% block title %}About Us{% endblock %}
```

```
{% block content %}
    <h2>About Us</h2>
    <p>We are a company dedicated to creating amazing things.</p>
{% endblock %}

<ul>
    {% for item in items %}
        <li>{{ item }}</li>
    {% endfor %}
</ul>

{% if user.is_authenticated %}
    <p>Welcome, {{ user.username }}!</p>
{% else %}
    <p>Please log in.</p>
{% endif %}
```