GIRIJANANDA CHOWDHURY UNIVERSITY, GUWAHATI(ASSAM)

Advanced python programming

LAB MANUAL

SUBJECT CODE: BCS23211P

B.TECH 4TH SEMESTER

Prepared by:-

MRIDUSMITA BARUAH
Laboratory Instructor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEEING

GIRIJANANDA CHOWDHURY UNIVERSITY, GUWAHATI(ASSAM)

GUWAHATI-781017

		L	T	P	C
BCS23211P	**ADVANCED PYTHON PROGRAMMING LAB**	0	0	2	1

Pre-requisite: Basic knowledge of python programming

Course Objectives: By the end of this course, students will be able to

1. Develop Advanced Python Programs.
2. Implement Data Handling and GUI-Based Applications
3. Perform Image Processing and Web Scraping
4. Analyze and Visualize Data

Programs:

1. Write a Python program that uses assertions to ensure a user-provided number is positive.
2. Create a decorator that measures and prints the execution time of functions.
3. Implement a generator to yield Fibonacci numbers up to a specified limit.
4. Create a custom iterator for iterating over a list of prime numbers.
5. Write a program to demonstrate multi-threading where each thread calculates the square of a number from a list.
6. Create a program where threads access shared data safely using threading.Lock.
7. Write a program using SQLite Database to store student records. Insert, retrieve, update, delete records and execute query to calculate and display the average grade of students.
8. Design a GUI with buttons that change the text of a label on a button click.
9. Create a form using Entry widgets to accept user input (e.g., name, age) and display it in a label.
10. Build an application to display a list of cities in a Listbox with a Scrollbar.
11. Design a menu-driven application with options like "About" and "Exit," using tkMessageBox to display messages.
12. Write a program to perform cropping, scaling, rotation, and flipping of an image using the Pillow library. And adjust the brightness, contrast, and apply sharpening filters to an image.
13. Scrape the latest news headlines from a website using the BeautifulSoup library and save them to a text file. Also scrape table data from a webpage and store it in a CSV file.
14. Load a dataset into a Pandas DataFrame, select specific columns, filter rows, and add new columns. Perform group-by operations and calculate aggregated metrics such as sum

and mean for a dataset.

15. Use matplotlib to visualize relationships in a dataset using scatter plots and line plots. Create histograms and box plots to display data distribution and outliers. Build a bar chart with custom titles, labels, legends, and colors.

Text Book

1. Beazley, D., & Jones, B. K. (2013). **Python cookbook: Recipes for mastering Python 3** (3rd ed.). O'Reilly Media.
2. Chollet, F. (2021). **Deep learning with Python** (2nd ed.). Manning Publications.
3. Grinberg, M. (2018). **Flask web development: Developing web applications with Python** (2nd ed.). O'Reilly Media

4. McKinney, W. (2022). **Python for data analysis: Data wrangling with Pandas, NumPy, and Jupyter** (3rd ed.). O'Reilly Media.

5. Ramalho, L. (2022). **Fluent Python: Clear, concise, and effective programming** (2nd ed.). O'Reilly Media

Reference Books

1. Lutz, M. (2013). **Programming Python** (4th ed.). O'Reilly Media.
2. Mitchell, R. (2018). **Web scraping with Python: Collecting data from the modern web** (2nd ed.). O'Reilly Media

3. Slatkin, B. (2020). **Effective Python: 90 specific ways to write better Python** (2nd ed.). Addison-Wesley Professional.

4. Sweigart, A. (2020). **Automate the boring stuff with Python: Practical programming for total beginners** (2nd ed.). No Starch Press.

5. Vincent, W. S. (2022). **Django for professionals: Production websites with Python & Django** (3rd ed.). WelcomeToCode

6. Python Software Foundation. (n.d.). **Python 3 documentation**. Retrieved from https://docs.python.org/3/

GIRIJANANDA CHOWDHURY UNIVERSITY, GUWAHATI(ASSAM)

LIST OF EXPERIMENTS:

1. Write a Python program that uses assertions to ensure a user-provided number is positive.
2. Create a decorator that measures and prints the execution time of functions.
3. Implement a generator to yield Fibonacci numbers up to a specified limit.
4. Create a custom iterator for iterating over a list of prime numbers.
5. Write a program to demonstrate multi-threading where each thread calculates the square of a number from a list.
6. Create a program where threads access shared data safely using threading.Lock.
7. Write a program using SQLite Database to store student records. Insert, retrieve, update, delete records and execute query to calculate and display the average grade of students.
8. Design a GUI with buttons that change the text of a label on a button click.
9. Create a form using Entry widgets to accept user input (e.g., name, age) and display it in a label.
10. Build an application to display a list of cities in a Listbox with a Scrollbar.
11. Design a menu-driven application with options like "About" and "Exit," using tkMessageBox to display messages.
12. Write a program to perform cropping, scaling, rotation, and flipping of an image using the Pillow library. And adjust the brightness, contrast, and apply sharpening filters to an image.
13. Scrape the latest news headlines from a website using the BeautifulSoup library and save them to a text file. Also scrape table data from a webpage and store it in a CSV file.
14. Load a dataset into a Pandas DataFrame, select specific columns, filter rows, and add new columns. Perform group-by operations and calculate aggregated metrics such as sum and mean for a dataset.

15. Use matplotlib to visualize relationships in a dataset using scatter plots and line plots. Create histograms and box plots to display data distribution and outliers. Build a bar chart with custom titles, labels, legends, and colors.

GIRIJANANDA CHOWDHURY UNIVERSITY, GUWAHATI(ASSAM)

1.AIM: Write a Python program that uses assertions to ensure a user-provided number is positive.

PROGRAM:

```
user_input = input("Enter a number: ")
number = float(user_input)
assert number > 0, "The number must be positive!"
print(f"You entered a positive number: {number}")
```

OUTPUT:

For positive number

```
Enter a number: 5
You entered a positive number: 5.0
```

For negative number

```
Enter a number: -5
assert number > 0, "The number must be positive!"
AssertionError: The number must be positive!
```

2.AIM: Create a decorator that measures and prints the execution time of functions.

PROGRAM:

```
import time
def measure_execution_time(func):
def wrapper(*args, **kwargs):
start_time = time.time()
result = func(*args, **kwargs)
end_time = time.time()
execution_time = end_time - start_time
print(f"Function {func.__name__} took {execution_time:.4f} seconds to execute")
return result # Return the result of the original function
return wrapper # Return the wrapper function
@measure_execution_time
def check_even_odd(number):
if number % 2 == 0:
print("Even no")
else:
print("Odd No")
return number
result = check_even_odd(10)
```

OUTPUT:

```
Even no
Function check_even_odd took 0.0010 seconds to execute
```

3.AIM: Implement a generator to yield Fibonacci numbers up to a specified limit.

PROGRAM:

```
def fibonacci():
x, y = 0, 1
while True:
yield x
x, y = y, x + y
n = int(input("Input the number of Fibonacci numbers you want to generate? "))
print("Number of first ",n,"Fibonacci numbers:")
fib = fibonacci()
for _ in range(n):
print(next(fib),end=" ")
```

OUTPUT:

```
Input the number of Fibonacci numbers you want to generate? 5
Number of first  5 Fibonacci numbers:
0 1 1 2 3
```

4.AIM: Create a custom iterator for iterating over a list of prime numbers.

PROGRAM:

```
class PrimeIterator:
def __init__(self, limit):
self.limit = limit
self.num = 2
def __iter__(self):
return self
def __next__(self):
while True:
for i in range(2, int(self.num ** 0.5) + 1):
if self.num % i == 0:
break
else:
prime = self.num
self.num += 1
return prime
self.num += 1
raise StopIteration
prime_iter = PrimeIterator(10)
for _ in range(10):
 print(next(prime_iter))
```

OUTPUT:

```
2
3
5
7
11
13
17
19
23
29
```

5.AIM: Write a program to demonstrate multi-threading where each thread calculates the square of a number from a list.

PROGRAM:

```
import threading
def calculate_square(number):
print(f"Square of {number}: {number ** 2}")
numbers = [2, 4, 6, 8, 10]
threads = []
for num in numbers:
thread = threading.Thread(target=calculate_square, args=(num,))
threads.append(thread)
thread.start()
for thread in threads:
thread.join()
print("All threads have finished execution.")
```

OUTPUT:

```
Square of 2: 4
Square of 4: 16Square of 6: 36
Square of 8: 64
Square of 10: 100
All threads have finished execution.
```

6.AIM: Create a program where threads access shared data safely using threading.Lock.

PROGRAM:

```
import threading
shared_counter = 0
lock = threading.Lock()
def increment_counter():
global shared_counter
for _ in range(1000):
with lock:
shared_counter += 1
threads = []
for _ in range(5):  # 5 threads
thread = threading.Thread(target=increment_counter)
threads.append(thread)
thread.start()
for thread in threads:
thread.join()
print(f"Final value of shared_counter: {shared_counter}")
```

OUTPUT:

Final value of shared_counter 5000

7.AIM: Write a program using SQLite Database to store student records. Insert, retrieve, update, delete records and execute query to calculate and display the average grade of students.

PROGRAM:

```
import sqlite3

# Connect to SQLite database (or create it if it doesn't exist)
conn = sqlite3.connect('students.db')
cursor = conn.cursor()

# Create table
cursor.execute('''
CREATE TABLE IF NOT EXISTS students (
id INTEGER PRIMARY KEY AUTOINCREMENT,
name TEXT NOT NULL,
grade REAL NOT NULL
)
''')

# Function to insert a student record
def insert_student(name, grade):
cursor.execute('INSERT INTO students (name, grade) VALUES (?, ?)', (name, grade))
conn.commit()

# Function to retrieve all student records
def retrieve_students():
cursor.execute('SELECT * FROM students')
return cursor.fetchall()

# Function to update a student's grade
def update_student(student_id, new_grade):
cursor.execute('UPDATE students SET grade = ? WHERE id = ?', (new_grade,
student_id))
```

```
conn.commit()

# Function to delete a student record
def delete_student(student_id):
cursor.execute('DELETE FROM students WHERE id = ?', (student_id,))
conn.commit()

# Function to calculate and display the average grade
def average_grade():
cursor.execute('SELECT AVG(grade) FROM students')
avg = cursor.fetchone()[0]
return avg
# Sample usage

if _name_ == "_main_":
# Insert sample data
insert_student("A", 85.0)
insert_student("B", 90.5)
insert_student("C", 78.0)

# Retrieve and display students
print("Student Records:")
for student in retrieve_students():
print(student)

# Update a record
update_student(1, 88.5)

# Delete a record
delete_student(3)

# Display updated records
print("\nUpdated Student Records:")
for student in retrieve_students():
print(student)
```

```
# Calculate and display average grade
print("\nAverage Grade:", average_grade())

# Close the connection
conn.close()
```

OUTPUT:

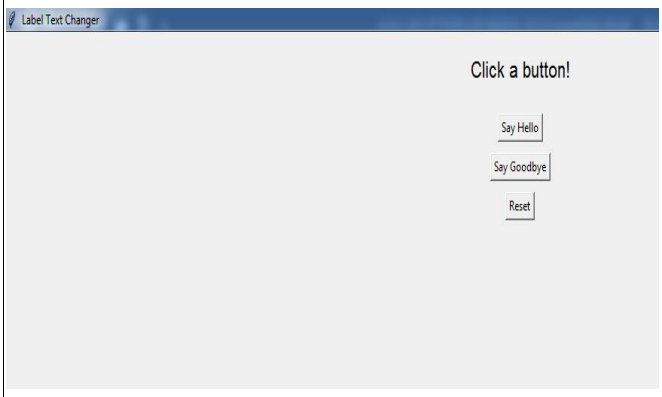
```
All students:
(1, 'A', 85.0)
(2, 'B', 90.5)
(3, 'C', 78.0)

Updated students:
(1, 'A', 88.0)
(2, 'B', 90.5)

Average grade: 89.25
```

8.AIM: Design a GUI with buttons that change the text of a label on a button click.

PROGRAM:

```
import tkinter as tk
def change_text(new_text):
label.config(text=new_text)

# Create main window
root = tk.Tk()
root.title("Label Text Changer")
root.geometry("300x200")

# Create a label
label = tk.Label(root, text="Click a button!", font=("Arial", 16))
label.pack(pady=20)

# Create buttons that change the label's text
button1=tk.Button(root,text="SayHello", command=lambda: change_text("Hello!"))
button1.pack(pady=5)
button2=tk.Button(root,text="SayGoodbye",command=lambda:change_text("Goodbye!
"))
button2.pack(pady=5)
button3=tk.Button(root,text="Reset",command=lambda:change_text("Click  a   button))

button3.pack(pady=5)

# Run the application
root.mainloop()
```

OUTPUT:


Label Text Changer
Click a button!
Say Hello
Say Goodbye
Reset

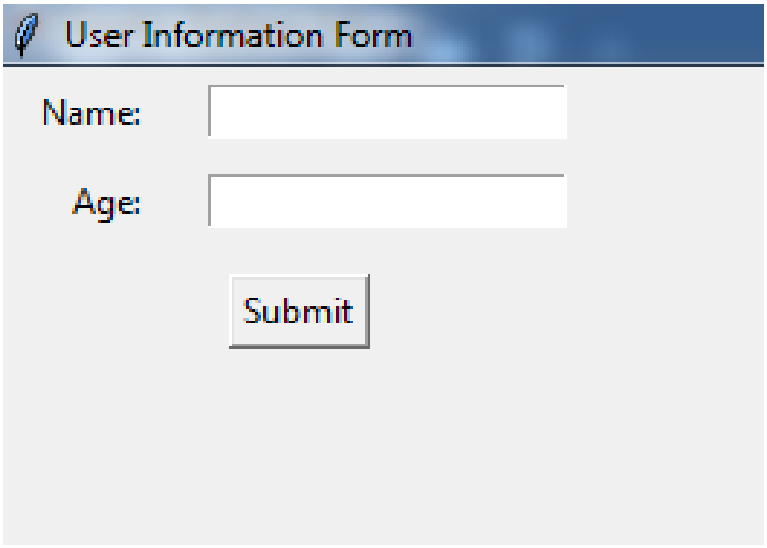

Label Text Changer
Hello!
Say Hello
Say Goodbye
Reset

9.AIM: Create a form using Entry widgets to accept user input (e.g., name, age) and display it in a label.

PROGRAM:

```
# Create main window
root = tk.Tk()
root.title("User Info Form")

# Name label and entry
tk.Label(root, text="Name:").grid(row=0, column=0, padx=10, pady=5, sticky="e")
name_entry = tk.Entry(root)
name_entry.grid(row=0, column=1, padx=10, pady=5)

# Age label and entry
tk.Label(root, text="Age:").grid(row=1, column=0, padx=10, pady=5, sticky="e")
age_entry = tk.Entry(root)
age_entry.grid(row=1, column=1, padx=10, pady=5)

# Submit button
submit_button = tk.Button(root, text="Submit", command=display_info)
submit_button.grid(row=2, column=0, columnspan=2, pady=10)
# Result label
result_label = tk.Label(root, text="")
result_label.grid(row=3, column=0, columnspan=2, pady=10)

# Run the application
root.mainloop()
```

OUTPUT:

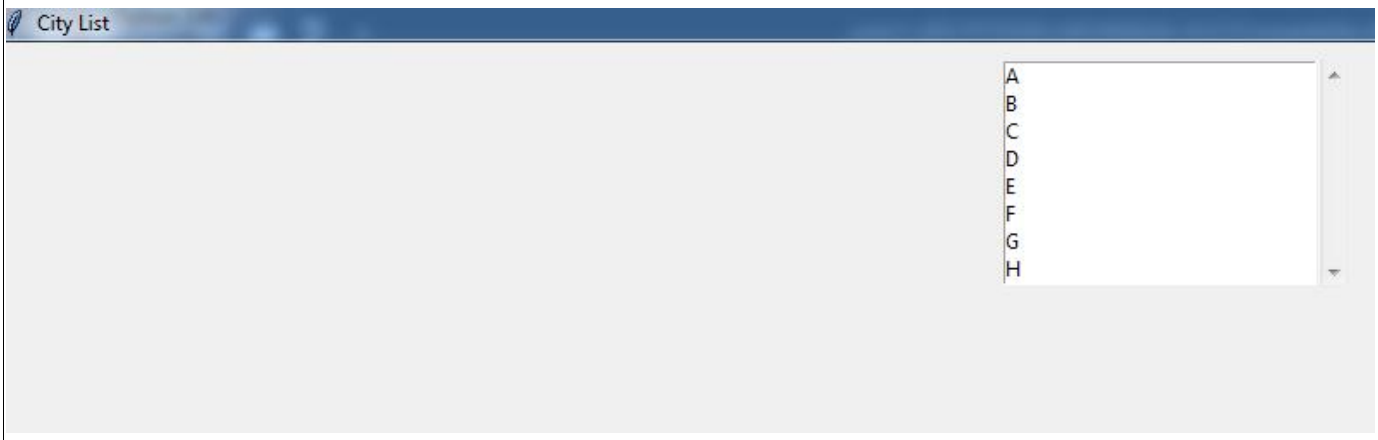


10.AIM: Build an application to display a list of cities in a Listbox with a Scrollbar.

PROGRAM:

```
import tkinter as tk
from tkinter import ttk
# Sample list of cities
cities = ["A", "B", "C", "D","E", "F", "G", "H"]

# Create the main window
root = tk.Tk()
root.title("City List")
root.geometry("300x250")

# Create a frame for the Listbox and Scrollbar
frame = ttk.Frame(root)
frame.pack(pady=20, padx=20, fill=tk.BOTH, expand=True)

# Create a scrollbar
scrollbar = tk.Scrollbar(frame, orient=tk.VERTICAL)

# Create a Listbox and link it with the scrollbar
listbox = tk.Listbox(frame, yscrollcommand=scrollbar.set, height=10)
scrollbar.config(command=listbox.yview)

# Pack the Listbox and Scrollbar
listbox.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)
scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

# Insert cities into the Listbox
for city in cities:
listbox.insert(tk.END, city)

# Run the application
root.mainloop()
```

OUTPUT:



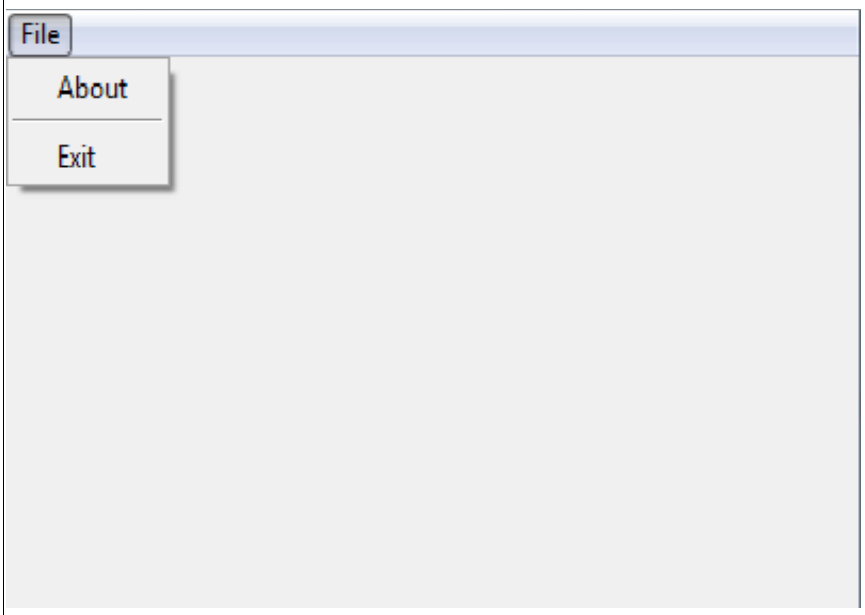
11.AIM: Design a menu-driven application with options like "About" and "Exit," using tkMessageBox to display messages.

PROGRAM:

```
import tkinter as tk
from tkinter import messagebox

# Create main application window
root = tk.Tk()
root.title("Menu-Driven Application")
root.geometry("300x200")

# Define callback functions
def show_about():
    messagebox.showinfo("About", "This is a simple menu-driven application using
    tkinter.")
def exit_app():
result = messagebox.askyesno("Exit", "Are you sure you want to exit?")
if result:
root.destroy()

# Create a menu bar
menubar = tk.Menu(root)

# Add "File" menu
file_menu = tk.Menu(menubar, tearoff=0)
file_menu.add_command(label="About", command=show_about)
file_menu.add_separator()
file_menu.add_command(label="Exit", command=exit_app)

# Add "File" menu to the menu bar
menubar.add_cascade(label="File", menu=file_menu)

# Configure the window to use the menu bar
```

```
root.config(menu=menubar)

# Run the main event loop
root.mainloop()
```

OUTPUT:



12.AIM: Write a program to perform cropping, scaling, rotation, and flipping of an image using the Pillow library. And adjust the brightness, contrast, and apply sharpening filters to an ima

PROGRAM:

```
from PIL import Image, ImageEnhance, ImageFilter

# Load image
image_path = 'your_image.jpg'  # Replace with your image file path
image = Image.open(image_path)

# 1. Cropping (left, upper, right, lower)
cropped_image = image.crop((100, 100, 400, 400))
cropped_image.save('cropped_image.jpg')

# 2. Scaling (Resizing)
scaled_image = image.resize((300, 300))
scaled_image.save('scaled_image.jpg')

# 3. Rotation (rotate 90 degrees)
rotated_image = image.rotate(90, expand=True)
rotated_image.save('rotated_image.jpg')

# 4. Flipping (horizontal and vertical)
flipped_horizontal = image.transpose(Image.FLIP_LEFT_RIGHT)
flipped_horizontal.save('flipped_horizontal.jpg')
flipped_vertical = image.transpose(Image.FLIP_TOP_BOTTOM)
flipped_vertical.save('flipped_vertical.jpg')

# 5. Adjust Brightness
enhancer_brightness = ImageEnhance.Brightness(image)
bright_image = enhancer_brightness.enhance(1.5)  # 1.0 is original, >1 is brighter
bright_image.save('bright_image.jpg')
```

```
# 6. Adjust Contrast
enhancer_contrast = ImageEnhance.Contrast(image)
contrast_image = enhancer_contrast.enhance(1.5)  # 1.0 is original, >1 is more contrast
contrast_image.save('contrast_image.jpg')

# 7. Apply Sharpening Filter
sharpened_image = image.filter(ImageFilter.SHARPEN)
sharpened_image.save('sharpened_image.jpg')
print("Image processing completed. Check the output files.")
```

OUTPUT:

The output of the above program consists of several processed image files saved to disk, each representing a specific transformation or enhancement applied to the original image. Here's what each output file contains:

Output Files and Descriptions:

1. cropped_image.jpg
A rectangular portion of the original image cropped from coordinates (100, 100) to (400, 400).
2. scaled_image.jpg
The original image resized to 300x300 pixels.
3. rotated_image.jpg
The original image rotated 90 degrees clockwise with size expansion to fit the full rotated image.
4. flipped_horizontal.jpg
The original image mirrored horizontally (left-to-right flip).
5. flipped_vertical.jpg
The original image flipped vertically (top-to-bottom flip).
6. bright_image.jpg
A version of the original image with increased brightness by a factor of 1.5.
7. contrast_image.jpg
A version of the original image with increased contrast by a factor of 1.5.
8. sharpened_image.jpg

The original image with a sharpening filter applied to enhance edges and fine details.

FINAL CONSOLE OUTPUT:

Image processing completed. Check

13.AIM: Scrape the latest news headlines from a website using the BeautifulSoup library and save them to a text file. Also scrape table data from a webpage and store it in a CSV fil

PROGRAM:

```
Install the required libraries first:
pip install requests beautifulsoup4 pandas
import requests
from bs4 import BeautifulSoup
import pandas as pd

# Part 1: Scrape News Headlines
def scrape_headlines(url, output_file)
response = requests.get(url)
soup = BeautifulSoup(response.text, 'html.parser')

# Adjust selector as per the site you're scraping
headlines = soup.select('h3')  # Common tag for headlines, update selector if needed
with open(output_file, 'w', encoding='utf-8') as f:
for headline in headlines:
text = headline.get_text(strip=True)
if text:
f.write(text + '\n')
print(f"Headlines saved to {output_file}")

# Part 2: Scrape Table Data
def scrape_table(url, output_file):
response = requests.get(url)
soup = BeautifulSoup(response.text, 'html.parser')
table = soup.find('table')
rows = []
for row in table.find_all('tr'):
cols = [col.get_text(strip=True) for col in row.find_all(['td', 'th'])]
rows.append(cols)
```

```
df = pd.DataFrame(rows[1:], columns=rows[0])  # Assumes first row is header
df.to_csv(output_file, index=False)
print(f"Table data saved to {output_file}")

# Example usage
news_url = 'https://www.bbc.com/news'  # Example; adjust for your use
table_url = 'https://www.w3schools.com/html/html_tables.asp'  # Example; adjust for your use scrape_headlines(news_url, 'headlines.txt')
scrape_table(table_url, 'table_data.csv')
```

OUTPUT:

1. headlines.txt

Contains a list of news headlines scraped from https://www.bbc.com/news.

2. table_data.csv

Contains tabular data scraped from https://www.worldometers.info/world-population/population-by-country/.

14.AIM: Load a dataset into a Pandas DataFrame, select specific columns, filter rows, and add new columns. Perform group-by operations and calculate aggregated metrics such as sum and mean for a dataset.

PROGRAM:

```
import pandas as pd

# Step 1: Load dataset into DataFrame (example dataset created manually here)
data = {
'Department': ['Sales', 'Sales', 'HR', 'HR', 'IT', 'IT'],
'Employee': ['A', 'B', 'C', 'D', 'E', 'F'],
'Salary': [70000, 80000, 50000, 55000, 90000, 85000],
'Bonus': [5000, 6000, 2000, 2500, 7000, 6500]
}
df = pd.DataFrame(data)

# Step 2: Select specific columns
df_selected = df[['Employee', 'Salary']]

# Step 3: Filter rows (e.g., employees with salary > 60000)
df_filtered = df[df['Salary'] > 60000]

# Step 4: Add a new column (e.g., Total Compensation = Salary + Bonus)
df['TotalCompensation'] = df['Salary'] + df['Bonus']

# Step 5: Group by 'Department'
grouped = df.groupby('Department')

# Step 6: Aggregate metrics (sum and mean of Salary and Total Compensation)
aggregated = grouped[['Salary', 'TotalCompensation']].agg(['sum', 'mean'])

# Display results
print("Filtered DataFrame:\n", df_filtered)
print("\nAggregated Metrics:\n", aggregated)
```

OUTPUT:

Here is the output generated by running the provided pandas code step-by-step:

Filtered DataFrame

(Employees with Salary > 60000)

	Department	Employee	Salary	Bonus
0	Sales	A	70000	5000
1	Sales	B	80000	6000
4	IT	E	90000	7000
5	IT	F	85000	6500

Aggregated Metrics (by Department)

(Sum and mean of Salary and TotalCompensation)

	Salary		TotalCompensation	
	sum	mean	sum	mean
Department				
HR	105000	52500.0	109500	54750.0
IT	175000	87500.0	188500	94250.0
Sales	150000	75000.0	161000	80500.0

15.AIM: Use matplotlib to visualize relationships in a dataset using scatter plots and line plots. Create histograms and box plots to display data distribution and outliers. Build a bar chart with custom titles, labels, legends, and colors.

PROGRAM:

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

# Generate sample dataset
np.random.seed(42)
x = np.random.rand(50)
y = 3 * x + np.random.normal(0, 0.2, 50)
category = np.random.choice(['Group A', 'Group B', 'Group C'], size=50)
values = np.random.randint(10, 100, size=50)

# Create DataFrame
df = pd.DataFrame({
'x': x,
'y': y,
'category': category,
'values': values
})

# Initialize figure
plt.figure(figsize=(15, 10))

# Scatter Plot
plt.subplot(2, 3, 1)
plt.scatter(df['x'], df['y'], color='blue', alpha=0.7)
plt.title('Scatter Plot of X vs Y')
plt.xlabel('X')
plt.ylabel('Y')
```

```
# Line Plot
plt.subplot(2, 3, 2)
sorted_df = df.sort_values('x')
plt.plot(sorted_df['x'], sorted_df['y'], color='green', linestyle='--', marker='o')
plt.title('Line Plot of X vs Y')
plt.xlabel('X')
plt.ylabel('Y')

# Histogram
plt.subplot(2, 3, 3)
plt.hist(df['values'], bins=10, color='purple', edgecolor='black')
plt.title('Histogram of Values')
plt.xlabel('Values')
plt.ylabel('Frequency')

# Box Plot
plt.subplot(2, 3, 4)
plt.boxplot(df['values'])
plt.title('Box Plot of Values')
plt.ylabel('Values')

# Bar Chart
plt.subplot(2, 3, 5)
avg_values = df.groupby('category')['values'].mean()
avg_values.plot(kind='bar', color=['orange', 'cyan', 'limegreen'])
plt.title('Average Values by Category')
plt.xlabel('Category')
plt.ylabel('Average Value')
plt.legend(['Average'])

# Layout and show
plt.tight_layout()
plt.show()
```

OUTPUT:

Top Left: Scatter plot of x vs y showing individual data points.
Top Middle: Line plot connecting sorted x, y pairs to show trends.
Top Right: Histogram showing the frequency distribution of values.
Bottom Left: Box plot revealing data spread and outliers in values.
Bottom Middle: Bar chart displaying average values for each category with distinct colors and a legend.