Dayananda Sagar Academy of Technology & Management

(Affiliated to Visvesvaraya Technological University,Belagavi & Approved by AICTE,New Delhi)
Opp. Art of Living, Udayapura, Kanakapura Road,Bangalore – 560082

Department of Artificial Intelligence and Machine Learning
Accredited by NBA, New Delhi

2024-2025

Python Programming Laboratory Manual

23AIML33

Compiled by

Ruchita Singhania (Assistant Professor)

VISION OF THE INSTITUTE

To strive at creating the institution a center of highest caliber of learning, so as to create an overall intellectual atmosphere with each deriving strength from the other to be the best of engineers, scientists with management & design skills.

MISSION OF THE INSTITUTE

- To serve its region, state, the nation and globally by preparing students to make meaningful contributions in an increasing complex global society challenge.
- To encourage, reflection on and evaluation of emerging needs and priorities with state of art infrastructure at institution.
- To support research and services establishing enhancements in technical, economic, human and cultural development.
- To establish inter disciplinary center of excellence, supporting/ promoting student's implementation.
- To increase the number of Doctorate holders to promote research culture on campus.
- To establish IIPC, IPR, EDC, innovation cells with functional MOU's supporting student's quality growth.

QUALITY POLICY

Dayananda Sagar Academy of Technology and Management aims at achieving academic excellence through continuous improvement in all spheres of Technical and Management education. In pursuit of excellence cutting-edge and contemporary skills are imparted to the utmost satisfaction of the students and the concerned stakeholders

OBJECTIVES & GOALS

- Creating an academic environment to nurture and develop competent entrepreneurs, leaders and professionals who are socially sensitive and environmentally conscious.
- Integration of Outcome Based Education and cognitive teaching and learning strategies to enhance learning effectiveness.
- Developing necessary infrastructure to cater to the changing needs of Business and Society.
- Optimum utilization of the infrastructure and resources to achieve excellence in all areas of relevance.
- Adopting learning beyond curriculum through outbound activities and creative assignments.
- Imparting contemporary and emerging techno-managerial skills to keep pace with the changing global trends.
- Facilitating greater Industry-Institute Interaction for skill development and employability enhancement.
- Establishing systems and processes to facilitate research, innovation and entrepreneurship for holistic development of students.
- Implementation of Quality Assurance System in all Institutional processes.

VISION OF THE DEPARTMENT

To develop high quality engineers with technical knowledge, skills and ethics in the area of Artificial Intelligence and Machine Learning to meet industrial and societal needs.

MISSION OF THE DEPARTMENT

M1: To provide high quality technical education with up-to-date infrastructure and trained human resources to deliver the curriculum effectively in order to impart technical knowledge and skills.

M2: To collaborate with academic institutions to elevate innovative research.

M3: To train the students with entrepreneurship qualities, multidisciplinary knowledge and latest skill sets as required for industry and research activities.

M4: To Produce creative and technically strong engineers and to research pioneering solutions to global challenges.

M5: To inculcate knowledge in lifelong learning.

Program Educational Objectives (PEOs)

PEO1: Graduates will have the ability to adapt, contribute and innovate new technologies and systems in the key domains of Artificial Intelligence and Machine Learning.

PEO2: Graduates will be able to successfully pursue higher education in reputed institutions with AI Specialization.

PEO3: Graduates will have the ability to explore research areas and produce outstanding contribution in various areas of Artificial Intelligence and Machine Learning.

PEO4: Graduates will be ethically and socially responsible solution providers and entrepreneurs in the field of Computer Science and Engineering with AI/ML Specialization.

PEO5: Engaged in successful professional practices in their chosen discipline

Program Specific Outcomes (PSOs):

PSO 1: Mathematical Informatics Section: To clarify the process of intelligent activities based on rational reasoning by human beings and to create new methods of problem-solving by implementation with computers.

PSO 2: Architecture Section: To establish the techniques for designing hardware and software systems necessary for the development of the next-generation of intelligent information processing systems.

PSO 3: Application Section: To apply the developed techniques to address challenging problems in various real-life applications from biomedical signal and image analysis, systems biology, climate to social and communication networks.

Program Outcomes (POs)

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis**: Identify, formulate, research literature, and Analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Python Programming Laboratory

Course Code: 23AIML33 **CIE Marks: 50**

Number of Lecture Hours/Week: 3:0:2:0 **SEE Marks: 50**

Total Number of Lecture Hours: 40 T + 20 P

Course Objectives:

1. Develop a strong foundation in Python programming, covering basic concepts like data types, variables, input/output operations, flow control, and functions
2. Use Python data structures (lists, dictionaries, strings, and tuples) to store and manipulate data.
3. Perform file operations in Python, including reading, writing, and managing files, while handling errors and debugging.
4. Use NumPy and Pandas for numerical computations and data analysis, handling and analyzing large datasets.
5. Create and customize data visualizations with Matplotlib to interpret and present data insights.

CONTENTS

Sl.No.	List of Programs
1	a) Develop a program to read the student details like Name, USN, and Marks in three subjects. Display the student details, total marks and percentage with suitable messages. b) Develop a program to read the name and year of birth of a person. Display whether the person is a senior citizen or not.
2	a) Develop a program to generate Fibonacci sequence of length (N). Read N from the console. b) Write a function to calculate factorial of a number. Develop a program to compute binomial coefficient (Given N and R).
3	Read N numbers from the console and create a list. Develop a program to print mean, variance and standard deviation with suitable messages.
4	Read a multi-digit number (as chars) from the console. Develop a program to print the frequency of each digit with suitable message.
5	Develop a program to print 10 most frequently appearing words in a text file. [Hint: Use dictionary with distinct words and their frequency of occurrences. Sort the dictionary in the reverse order of frequency and display dictionary slice of first 10 items]
6	Develop a program to sort the contents of a text file and write the sorted contents into a separate text file. [Hint: Use string methods strip(), len(), list methods sort(), append(), and file methods open(), readlines(), and write()].
7	Develop a program to backing Up a given Folder (Folder in a current working directory) into a ZIP File by using relevant modules and suitable methods.
8	Develop a program to read a CSV file containing daily temperature data with columns like 'Date' and 'Temperature'. Perform data analysis to compute the average temperature for each month. Visualize the average monthly temperatures using a line chart.
9	Develop a program to generate a NumPy array of random integers. Perform basic array operations such as finding the mean, median, variance, and standard deviation. Visualize the array values using a histogram.
10	Design a basic calculator that can perform addition, subtraction, multiplication, and division. Extend it to handle more complex operations like square roots, exponents, and trigonometric functions.
11	Develop a to-do list application where users can add, delete, and view their tasks. Extend it by adding features like due dates, priority levels, and the ability to mark tasks as completed.
12	Develop a hangman game where users try to guess a hidden word by suggesting letters within a certain number of attempts.

DO's & DON'TS

DOs	**DON'TS**
Be regular to the Lab	Do not come late to the Lab
Avoid unnecessary talking while executing the program.	Do not panic if you do not get the output
Arrange your chairs and tables before leaving the laboratory.	
Maintain discipline in the laboratory	

Laboratory Preparation: Each student is responsible for maintaining his/her own Laboratory Observation Notebook. Each student is required to perform pre-lab work and enter it into his/her notebook.

Lab Work: Each student must be checked by the faculty. Check-out will be used to confirm that the actual lab work as recorded in the lab notebook has been completed. The faculty will initial and date all the data acquired during the lab period. All pages is to be signed by the allotted faculty on the same date in the lab itself.

Lab Completion: Each program should be completed during the lab period. If a student is unable to complete the lab program, they may complete it in the Break time, if granted permission by the instructor. The work must be checked to verify that all laboratory exercises are complete.

In case a student is **ABSENT** for a particular lab session, he/she has to compulsorily finish the program before the next lab session and get the observation signed All lab work should be completed before the next laboratory period.

Program 1:

a) **Develop a program to read the student details like Name, USN, and Marks in three subjects. Display the student details, total marks and percentage with suitable messages.**
b) **Develop a program to read the name and year of birth of a person. Display whether the person is a senior citizen or not.**

a)

```python
# Function to input student details
def input_student_details():
    name = input("Enter student's name: ")
    usn = input("Enter student's USN: ")
    marks = []

    # Input marks for three subjects
    for i in range(1, 4):
        mark = float(input(f"Enter marks for subject {i}: "))
        marks.append(mark)

    return name, usn, marks

# Function to calculate total marks and percentage
def calculate_results(marks):
    total_marks = sum(marks)
    percentage = (total_marks / 300) * 100  # Assuming each subject is out of 100 marks
    return total_marks, percentage

# Function to display the student details, total marks, and percentage
def display_student_details(name, usn, total_marks, percentage):
    print("\n--- Student Details ---")
    print(f"Name: {name}")
    print(f"USN: {usn}")
    print(f"Total Marks: {total_marks}")
    print(f"Percentage: {percentage:.2f}%")


# Input student details
name, usn, marks = input_student_details()

# Calculate total marks and percentage
total_marks, percentage = calculate_results(marks)

# Display the student details and results
display_student_details(name, usn, total_marks, percentage)
```

OUTPUT:

```
Enter student's name: Ravi
Enter student's USN: 004
Enter marks for subject 1: 78
Enter marks for subject 2: 87
Enter marks for subject 3: 89

--- Student Details ---
Name: Ravi
USN: 004
Total Marks: 254.0
Percentage: 84.67%
```

b)

```python
from datetime import datetime

# Function to input person details
def input_person_details():
    name = input("Enter the person's name: ")
    year_of_birth = int(input("Enter the year of birth: "))
    return name, year_of_birth

# Function to determine if the person is a senior citizen
def is_senior_citizen(year_of_birth):
    current_year = datetime.now().year
    age = current_year - year_of_birth
    return age >= 60

# Function to display the result
def display_result(name, is_senior):
    print("\n--- Result ---")
    if is_senior:
        print(f"{name} is a senior citizen.")
    else:
        print(f"{name} is not a senior citizen.")


# Input person details
name, year_of_birth = input_person_details()

# Determine if the person is a senior citizen
is_senior = is_senior_citizen(year_of_birth)

# Display the result
display_result(name, is_senior)
```

OUTPUT:

```
Enter the person's name: Raju
Enter the year of birth: 1967

--- Result ---
Raju is not a senior citizen.


Enter the person's name: Rahul
Enter the year of birth: 147

--- Result ---
Rahul is a senior citizen.
```

Program 2:

a) **Develop a program to generate Fibonacci sequence of length (N). Read N from the console.**

b) **Write a function to calculate factorial of a number. Develop a program to compute binomial coefficient (Given N and R).**

a)

```python
# Function to generate the Fibonacci sequence
def generate_fibonacci(n):
    if n <= 0:
        return []
    elif n == 1:
        return [0]
    elif n == 2:
        return [0, 1]

    fibonacci_sequence = [0, 1]

    for i in range(2, n):
        next_number = fibonacci_sequence[-1] + fibonacci_sequence[-2]
        fibonacci_sequence.append(next_number)

    return fibonacci_sequence


# Read N from the console
n = int(input("Enter the length of the Fibonacci sequence (N): "))

# Generate Fibonacci sequence of length N
fibonacci_sequence = generate_fibonacci(n)

# Display the Fibonacci sequence
print(f"\nFibonacci sequence of length {n}:")
print(fibonacci_sequence)
```

OUTPUT:

```
Enter the length of the Fibonacci sequence (N): 10

Fibonacci sequence of length 10:
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

b)

$$\text{Binomial Coefficient } C(N,R) = \frac{N!}{R! \times (N-R)!}$$

```python
# Function to calculate factorial of a number
def factorial(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n - 1)

# Function to compute the binomial coefficient C(N, R)
def binomial_coefficient(n, r):
    return factorial(n) // (factorial(r) * factorial(n - r))


# Read N and R from the console
n = int(input("Enter the value of N: "))
r = int(input("Enter the value of R: "))

# Compute binomial coefficient C(N, R)
if r > n:
    print("Invalid input: R cannot be greater than N.")
else:
    result = binomial_coefficient(n, r)
    print(f"\nBinomial Coefficient C({n}, {r}) = {result}")
```

OUTPUT:

```
Enter the value of N: 7
Enter the value of R: 4

Binomial Coefficient C(7, 4) = 35
```

Program 3:

Read N numbers from the console and create a list. Develop a program to print mean, variance and standard deviation with suitable messages.

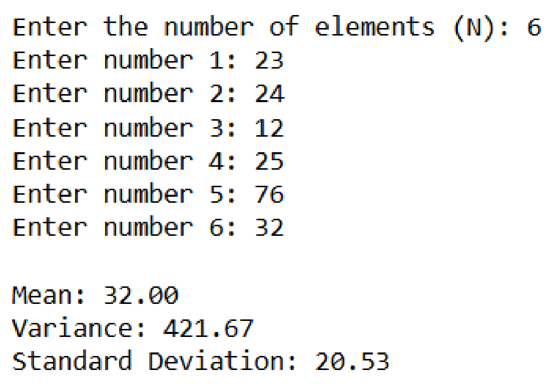
Variance can be calculated using the following equation:

Formula	Explanation
$\sigma^2 = \frac{\sum(X-\mu)^2}{N}$	• σ^2 = population variance • $\sum$ = sum of... • X = each value • μ = population mean • N = number of values in the population

Standard deviation is the square root of the variance.

```python
import math

# Function to calculate the mean
def calculate_mean(numbers):
    return sum(numbers) / len(numbers)

# Function to calculate the variance
def calculate_variance(numbers, mean):
    return sum((x - mean) ** 2 for x in numbers) / len(numbers)

# Function to calculate the standard deviation
def calculate_standard_deviation(variance):
    return math.sqrt(variance)


# Read N from the console
n = int(input("Enter the number of elements (N): "))

# Read N numbers and create a list
numbers = []
for i in range(n):
    number = float(input(f"Enter number {i + 1}: "))
    numbers.append(number)

# Calculate mean, variance, and standard deviation
mean = calculate_mean(numbers)
variance = calculate_variance(numbers, mean)
std_deviation = calculate_standard_deviation(variance)

# Display the results
print(f"\nMean: {mean:.2f}")
print(f"Variance: {variance:.2f}")
print(f"Standard Deviation: {std_deviation:.2f}")
```

OUTPUT:



```
Enter the number of elements (N): 6
Enter number 1: 23
Enter number 2: 24
Enter number 3: 12
Enter number 4: 25
Enter number 5: 76
Enter number 6: 32

Mean: 32.00
Variance: 421.67
Standard Deviation: 20.53
```

Program 4:

Read a multi-digit number (as chars) from the console. Develop a program to print the frequency of each digit with suitable message.

```python
# Function to calculate the frequency of each digit
def calculate_digit_frequency(number):
    frequency = {}

    for digit in number:
        if digit.isdigit():
            frequency[digit] = frequency.get(digit, 0) + 1

    return frequency

# Function to display the frequency of each digit
def display_digit_frequency(frequency):
    print("\nFrequency of each digit:")
    for digit, count in sorted(frequency.items()):
        print(f"Digit {digit}: {count} time(s)")


# Read the multi-digit number from the console
number = input("Enter a multi-digit number: ")

# Calculate the frequency of each digit
frequency = calculate_digit_frequency(number)

# Display the frequency of each digit
display_digit_frequency(frequency)
```

OUTPUT:

```
Enter a multi-digit number: 342346575

Frequency of each digit:
Digit 2: 1 time(s)
Digit 3: 2 time(s)
Digit 4: 2 time(s)
Digit 5: 2 time(s)
Digit 6: 1 time(s)
Digit 7: 1 time(s)
```

Program 5:

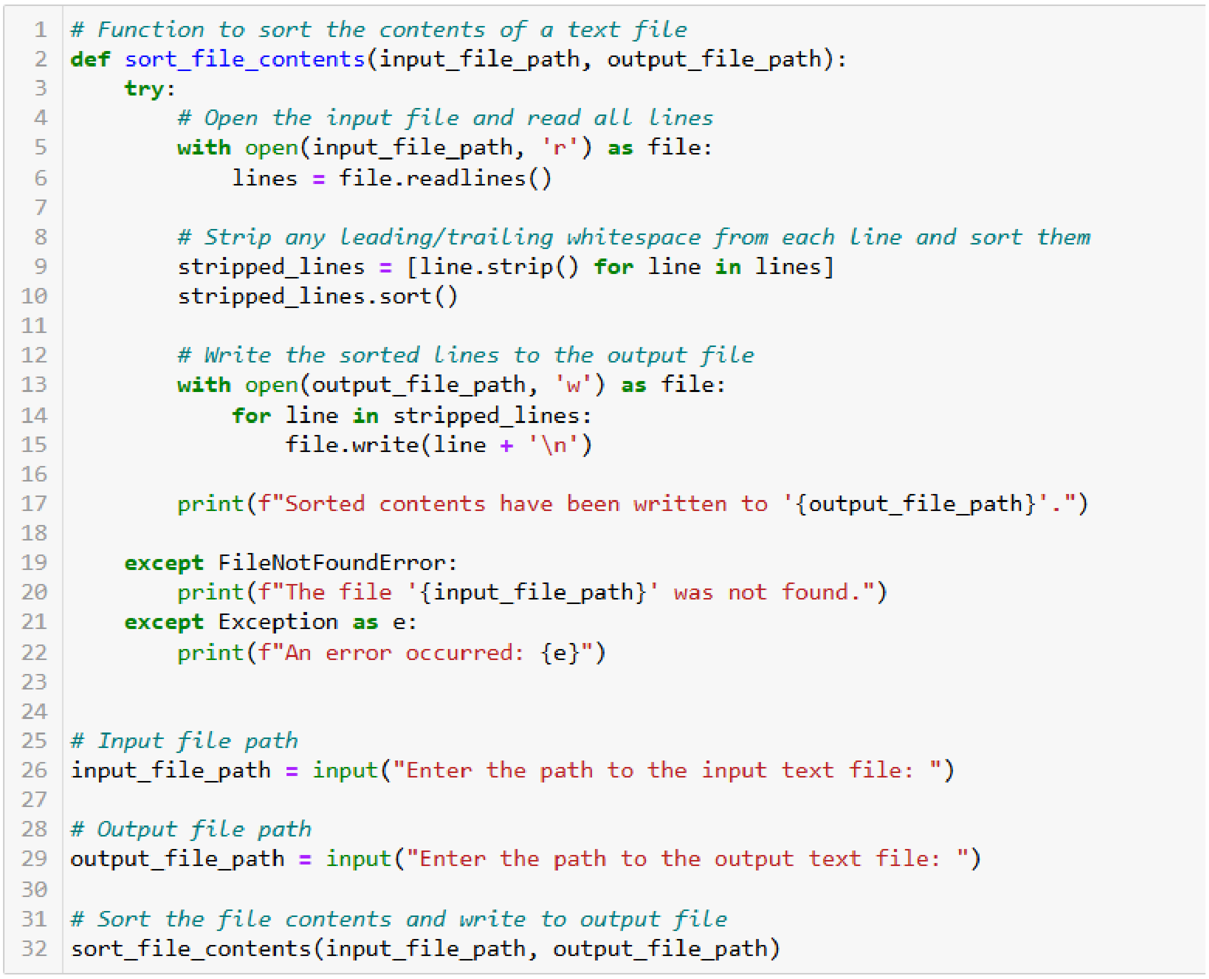
```python
from collections import defaultdict
import string

# Function to read the text file and count word frequencies
def count_word_frequencies(file_path):
    word_frequency = defaultdict(int)

    with open(file_path, 'r') as file:
        for line in file:
            # Remove punctuation and convert to lowercase
            line = line.translate(str.maketrans('', '', string.punctuation)).lower()
            words = line.split()

            for word in words:
                word_frequency[word] += 1

    return word_frequency

# Function to get the 10 most frequently appearing words
def get_top_n_words(word_frequency, n=10):
    # Sort the dictionary by frequency in descending order and take the top `n` items
    sorted_words = sorted(word_frequency.items(), key=lambda item: item[1], reverse=True)
    return sorted_words[:n]


# Specify the path to your text file
file_path = input("Enter the path to the text file: ")

# Count word frequencies in the text file
word_frequency = count_word_frequencies(file_path)

# Get the top 10 most frequently appearing words
top_words = get_top_n_words(word_frequency)

# Display the top 10 words with their frequencies
print("\nThe 10 most frequently appearing words are:")
for word, freq in top_words:
    print(f"{word}: {freq} time(s)")
```

OUTPUT:

```
Enter the path to the text file: C://Users/mahes/Ruchita PESU/Dayanand Sagar/Python Programming/Python lab programs/Sample-text
-file.txt

The 10 most frequently appearing words are:
doe: 21 time(s)
the: 15 time(s)
in: 15 time(s)
john: 14 time(s)
or: 13 time(s)
a: 12 time(s)
used: 10 time(s)
to: 10 time(s)
as: 8 time(s)
is: 8 time(s)
```

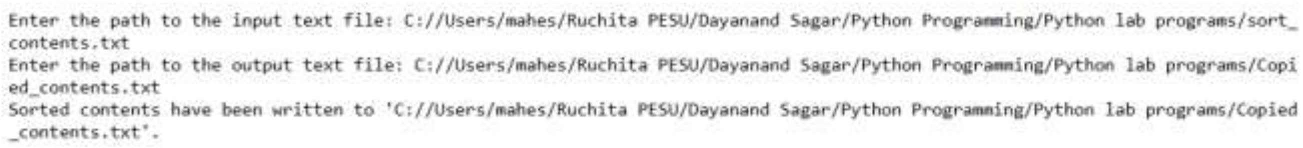
Program 6:

Develop a program to sort the contents of a text file and write the sorted contents into a separate text file. [Hint: Use string methods strip(), len(), list methods sort(), append(), and file methods open(), readlines(), and write()].



```python
# Function to sort the contents of a text file
def sort_file_contents(input_file_path, output_file_path):
    try:
        # Open the input file and read all lines
        with open(input_file_path, 'r') as file:
            lines = file.readlines()

        # Strip any leading/trailing whitespace from each line and sort them
        stripped_lines = [line.strip() for line in lines]
        stripped_lines.sort()

        # Write the sorted lines to the output file
        with open(output_file_path, 'w') as file:
            for line in stripped_lines:
                file.write(line + '\n')

        print(f"Sorted contents have been written to '{output_file_path}'.")

    except FileNotFoundError:
        print(f"The file '{input_file_path}' was not found.")
    except Exception as e:
        print(f"An error occurred: {e}")


# Input file path
input_file_path = input("Enter the path to the input text file: ")

# Output file path
output_file_path = input("Enter the path to the output text file: ")

# Sort the file contents and write to output file
sort_file_contents(input_file_path, output_file_path)
```

OUTPUT:



```
Enter the path to the input text file: C://Users/mahes/Ruchita PESU/Dayanand Sagar/Python Programming/Python lab programs/sort_
contents.txt
Enter the path to the output text file: C://Users/mahes/Ruchita PESU/Dayanand Sagar/Python Programming/Python lab programs/Copi
ed_contents.txt
Sorted contents have been written to 'C://Users/mahes/Ruchita PESU/Dayanand Sagar/Python Programming/Python lab programs/Copied
_contents.txt'.
```

Program 7:

Develop a program to backing Up a given Folder (Folder in a current working directory) into a ZIP File by using relevant modules and suitable methods.

```python
import os
import zipfile
from datetime import datetime

# Function to create a backup of the folder into a ZIP file
def backup_folder_to_zip(folder_name):
    # Get the absolute path of the folder to back up
    folder_path = os.path.abspath(folder_name)

    # Create a unique filename for the ZIP file using the folder name and current timestamp
    timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')
    zip_filename = f"{folder_name}_{timestamp}.zip"

    # Create the ZIP file
    with zipfile.ZipFile(zip_filename, 'w', zipfile.ZIP_DEFLATED) as backup_zip:
        # Walk through the entire folder tree and add files to the ZIP file
        for foldername, subfolders, filenames in os.walk(folder_path):
            # Add the current folder to the ZIP file
            backup_zip.write(foldername, os.path.relpath(foldername, folder_path))

            # Add all the files in this folder to the ZIP file
            for filename in filenames:
                file_path = os.path.join(foldername, filename)
                backup_zip.write(file_path, os.path.relpath(file_path, folder_path))

    print(f"Backup complete! The folder '{folder_name}' has been backed up into '{zip_filename}'.")

# Ask the user for the folder name to back up
folder_name = input("Enter the name of the folder to back up: ")

# Check if the folder exists in the current working directory
if os.path.isdir(folder_name):
    backup_folder_to_zip(folder_name)
else:
    print(f"The folder '{folder_name}' does not exist in the current working directory.")
```

OUTPUT:

```
Enter the name of the folder to back up: Python lab programs
The folder 'Python lab programs' does not exist in the current working directory.
```

```
Enter the name of the folder to back up: dataset
Backup complete! The folder 'dataset' has been backed up into 'dataset_20240817_143554.zip'.
```

Program 8:

Develop a program to read a CSV file containing daily temperature data with columns like 'Date' and 'Temperature'. Perform data analysis to compute the average temperature for each month. Visualize the average monthly temperatures using a line chart.

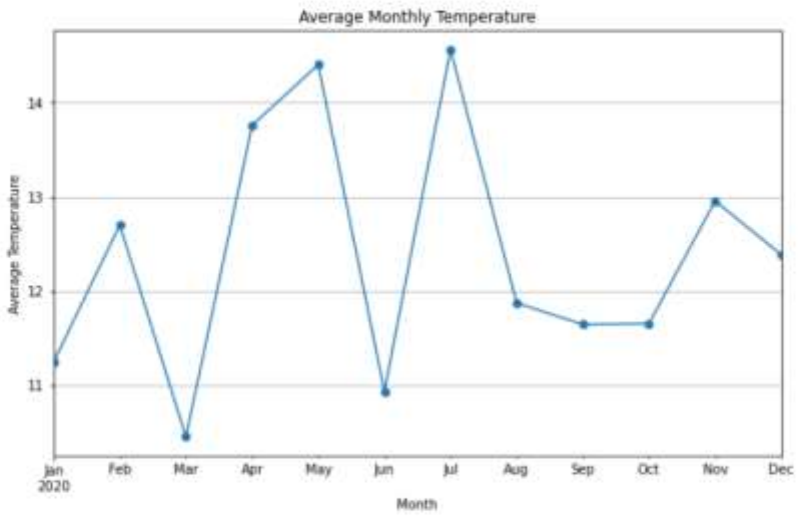
```python
import pandas as pd
import matplotlib.pyplot as plt
```

```python
# Function to compute average monthly temperatures
def compute_average_monthly_temperatures(file_path):
    # Read the CSV file into a DataFrame
    df = pd.read_csv(file_path, parse_dates=['Date'])

    # Ensure 'Date' is the correct datatype
    df['Date'] = pd.to_datetime(df['Date'])

    # Extract the year and month from the 'Date' column
    df['YearMonth'] = df['Date'].dt.to_period('M')

    # Group by 'YearMonth' and calculate the average temperature
    monthly_avg_temp = df.groupby('YearMonth')['Temperature'].mean()

    return monthly_avg_temp

# Function to visualize the average monthly temperatures using a line chart
def visualize_average_monthly_temperatures(monthly_avg_temp):
    # Plot the data
    plt.figure(figsize=(10, 6))
    monthly_avg_temp.plot(kind='line', marker='o')

    # Add titles and labels
    plt.title('Average Monthly Temperature')
    plt.xlabel('Month')
    plt.ylabel('Average Temperature')
    plt.grid(True)

    # Show the plot
    plt.show()
```

```python
# Path to the CSV file
file_path = input("Enter the path to the CSV file containing temperature data: ")

# Compute average monthly temperatures
monthly_avg_temp = compute_average_monthly_temperatures(file_path)

# Visualize the average monthly temperatures
visualize_average_monthly_temperatures(monthly_avg_temp)
```

OUTPUT:

```
Enter the path to the CSV file containing temperature data: C:/Users/mahes/Ruchita PESU/Dayanand Sagar/Python Programming/Pytho
n lab programs/TemperatureData.csv
```



Program 9:

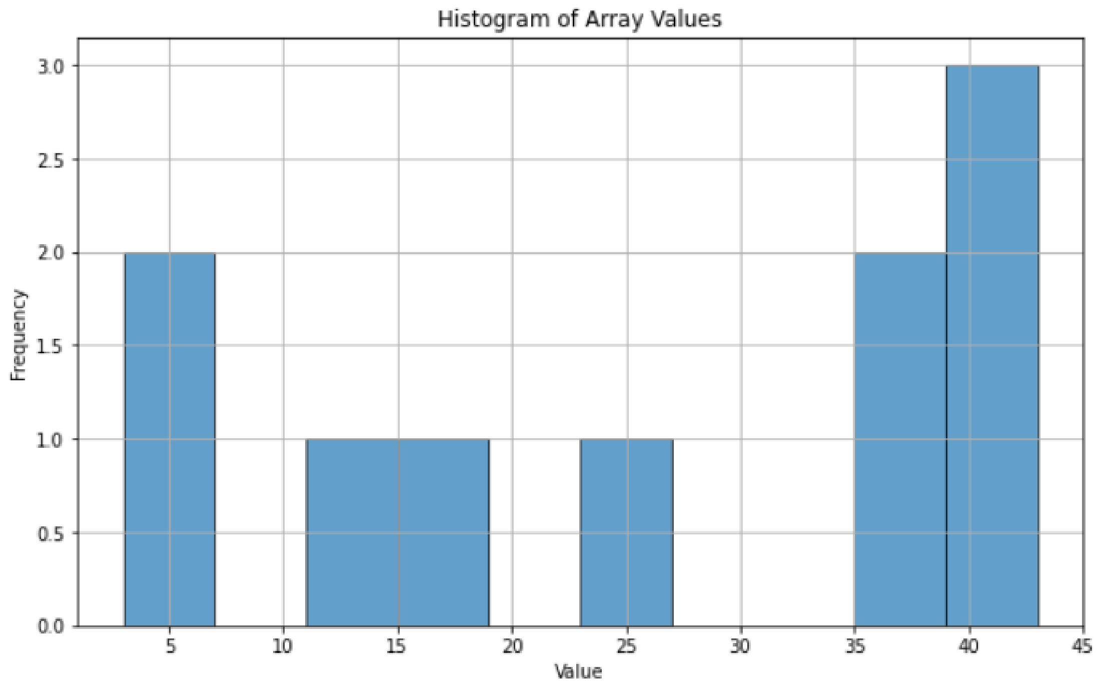
Develop a program to generate a NumPy array of random integers. Perform basic array operations such as finding the mean, median, variance, and standard deviation. Visualize the array values using a histogram.

```python
import numpy as np
import matplotlib.pyplot as plt

# Function to generate a NumPy array of random integers
def generate_random_array(size, low, high):
    return np.random.randint(low, high, size)

# Function to perform basic array operations
def array_operations(arr):
    mean = np.mean(arr)
    median = np.median(arr)
    variance = np.var(arr)
    std_deviation = np.std(arr)

    return mean, median, variance, std_deviation

# Function to visualize the array values using a histogram
def visualize_array_histogram(arr):
    plt.figure(figsize=(10, 6))
    plt.hist(arr, bins=10, edgecolor='black', alpha=0.7)

    plt.title('Histogram of Array Values')
    plt.xlabel('Value')
    plt.ylabel('Frequency')
    plt.grid(True)
    plt.show()
```

```python
# User input for array generation
size = int(input("Enter the size of the array: "))
low = int(input("Enter the lower bound of the random integers: "))
high = int(input("Enter the upper bound of the random integers: "))

# Generate the random array
arr = generate_random_array(size, low, high)
print("\n Random array of integers is: \n", arr)

# Perform basic array operations
mean, median, variance, std_deviation = array_operations(arr)

# Display the results
print(f"\nArray Operations Results:")
print(f"Mean: {mean}")
print(f"Median: {median}")
print(f"Variance: {variance}")
print(f"Standard Deviation: {std_deviation}")

# Visualize the array values using a histogram
visualize_array_histogram(arr)
```

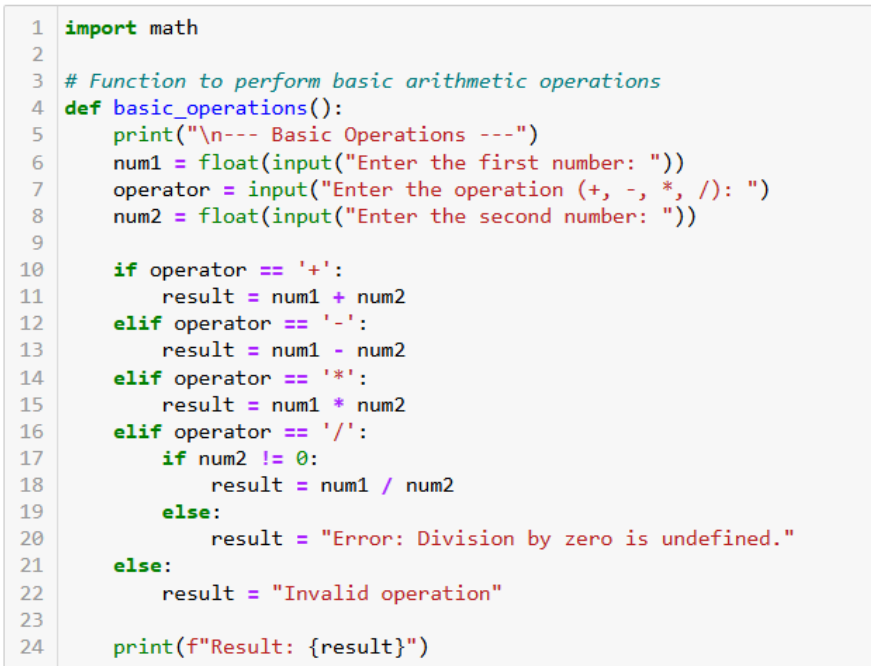
OUTPUT:

```
Enter the size of the array: 10
Enter the lower bound of the random integers: 2
Enter the upper bound of the random integers: 50

 Random array of integers is:
 [42 42 43  6 17 14  3 35 37 25]

Array Operations Results:
Mean: 26.4
Median: 30.0
Variance: 215.64000000000001
Standard Deviation: 14.684685900624501
```

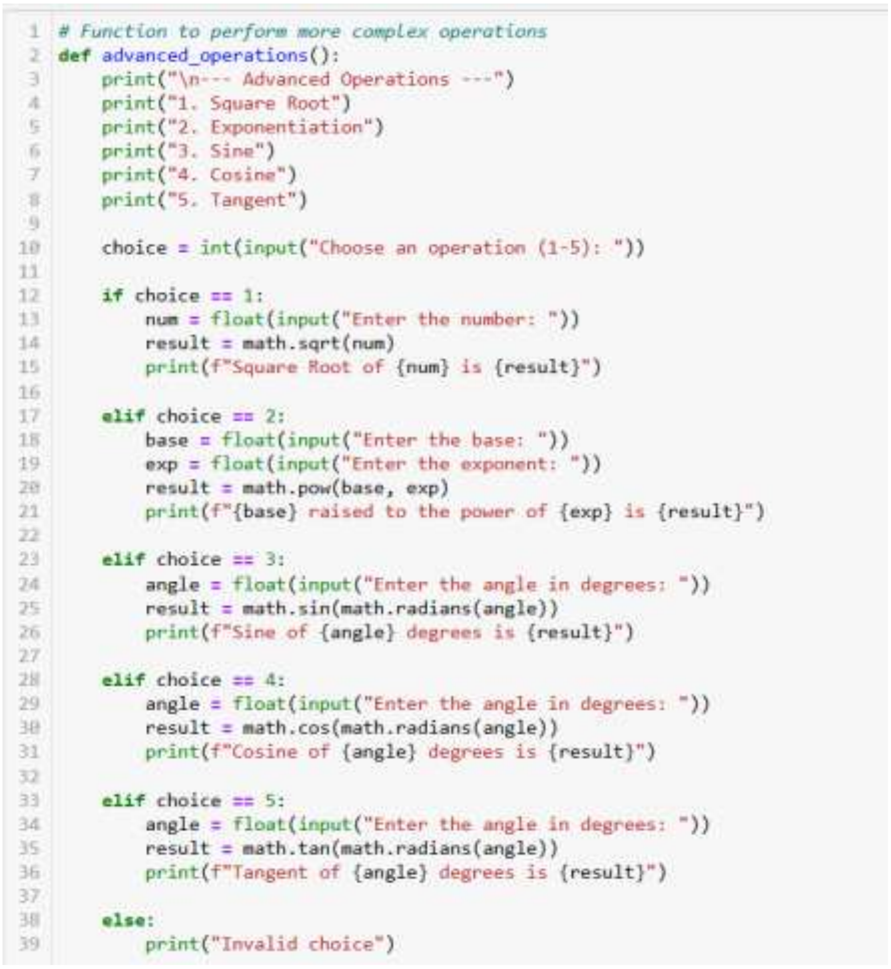


Program 10:

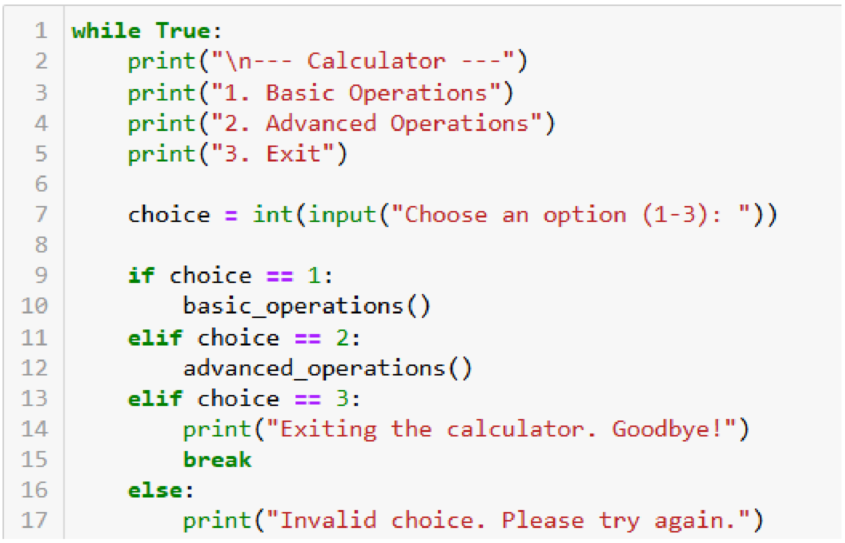
Design a basic calculator that can perform addition, subtraction, multiplication, and division. Extend it to handle more complex operations like square roots, exponents, and trigonometric functions.



```python
import math

# Function to perform basic arithmetic operations
def basic_operations():
    print("\n--- Basic Operations ---")
    num1 = float(input("Enter the first number: "))
    operator = input("Enter the operation (+, -, *, /): ")
    num2 = float(input("Enter the second number: "))

    if operator == '+':
        result = num1 + num2
    elif operator == '-':
        result = num1 - num2
    elif operator == '*':
        result = num1 * num2
    elif operator == '/':
        if num2 != 0:
            result = num1 / num2
        else:
            result = "Error: Division by zero is undefined."
    else:
        result = "Invalid operation"

    print(f"Result: {result}")
```



```python
# Function to perform more complex operations
def advanced_operations():
    print("\n--- Advanced Operations ---")
    print("1. Square Root")
    print("2. Exponentiation")
    print("3. Sine")
    print("4. Cosine")
    print("5. Tangent")

    choice = int(input("Choose an operation (1-5): "))

    if choice == 1:
        num = float(input("Enter the number: "))
        result = math.sqrt(num)
        print(f"Square Root of {num} is {result}")

    elif choice == 2:
        base = float(input("Enter the base: "))
        exp = float(input("Enter the exponent: "))
        result = math.pow(base, exp)
        print(f"{base} raised to the power of {exp} is {result}")

    elif choice == 3:
        angle = float(input("Enter the angle in degrees: "))
        result = math.sin(math.radians(angle))
        print(f"Sine of {angle} degrees is {result}")

    elif choice == 4:
        angle = float(input("Enter the angle in degrees: "))
        result = math.cos(math.radians(angle))
        print(f"Cosine of {angle} degrees is {result}")

    elif choice == 5:
        angle = float(input("Enter the angle in degrees: "))
        result = math.tan(math.radians(angle))
        print(f"Tangent of {angle} degrees is {result}")

    else:
        print("Invalid choice")
```



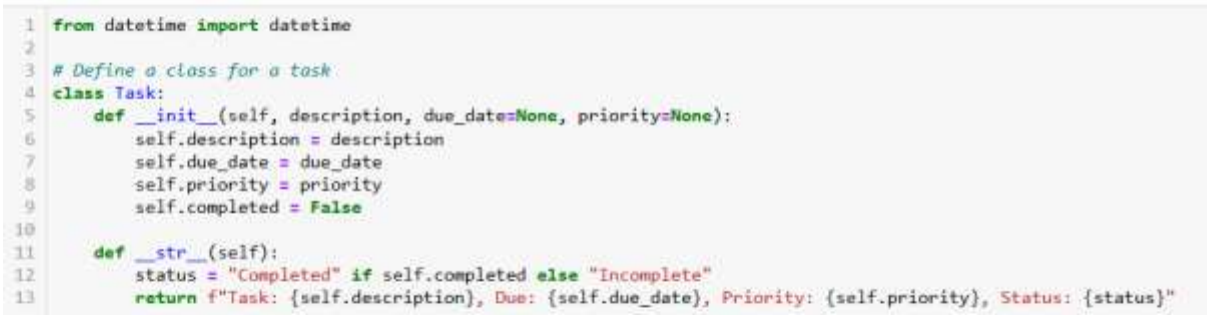
```python
while True:
    print("\n--- Calculator ---")
    print("1. Basic Operations")
    print("2. Advanced Operations")
    print("3. Exit")

    choice = int(input("Choose an option (1-3): "))

    if choice == 1:
        basic_operations()
    elif choice == 2:
        advanced_operations()
    elif choice == 3:
        print("Exiting the calculator. Goodbye!")
        break
    else:
        print("Invalid choice. Please try again.")
```

OUTPUT:

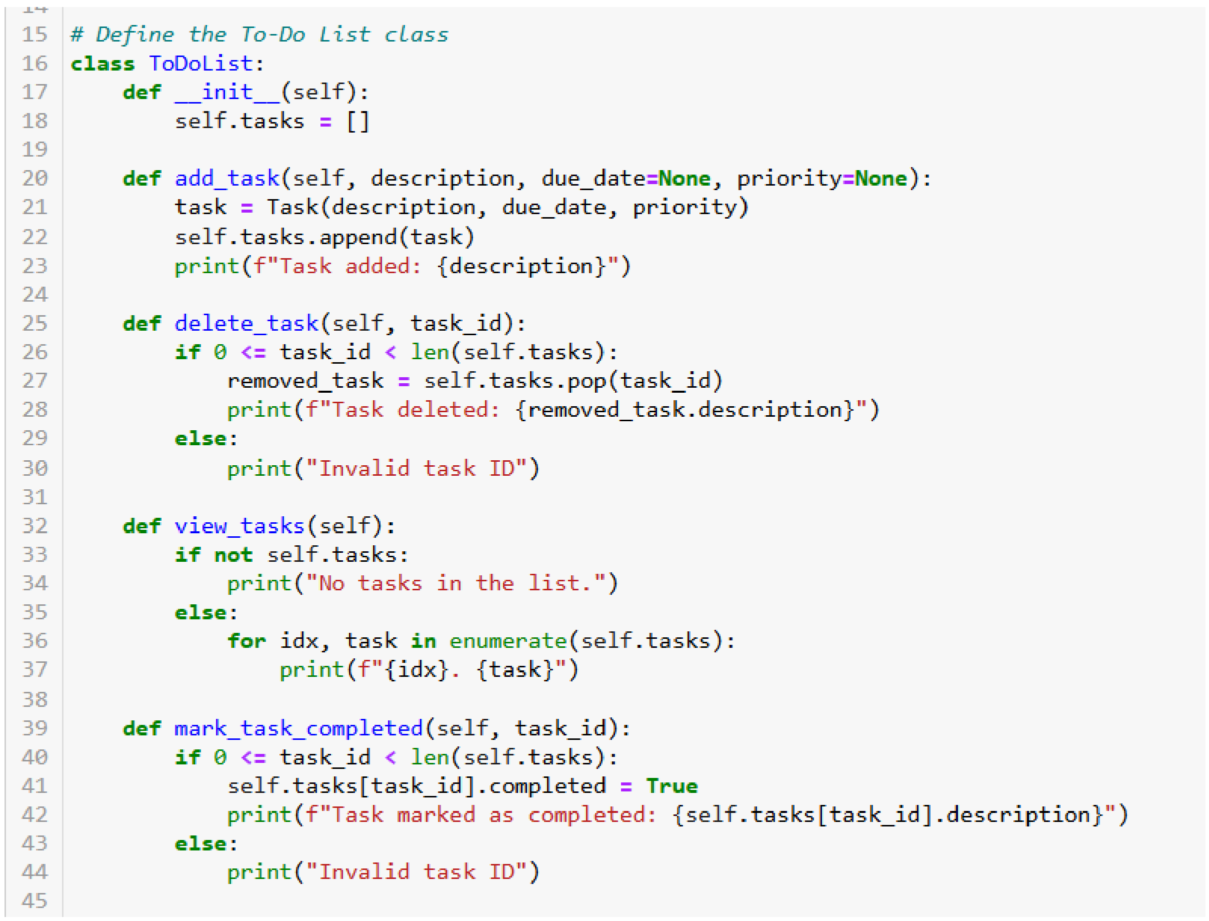
```
--- Calculator ---
1. Basic Operations
2. Advanced Operations
3. Exit
Choose an option (1-3): 1

--- Basic Operations ---
Enter the first number: 5
Enter the operation (+, -, *, /): *
Enter the second number: 8
Result: 40.0

--- Calculator ---
1. Basic Operations
2. Advanced Operations
3. Exit
Choose an option (1-3): 2

--- Advanced Operations ---
1. Square Root
2. Exponentiation
3. Sine
4. Cosine
5. Tangent
Choose an operation (1-5): 2
Enter the base: 6
Enter the exponent: 4
6.0 raised to the power of 4.0 is 1296.0

--- Calculator ---
1. Basic Operations
2. Advanced Operations
3. Exit
Choose an option (1-3): 3
Exiting the calculator. Goodbye!
```

Experiment 11:

Develop a to-do list application where users can add, delete, and view their tasks. Extend it by adding features like due dates, priority levels, and the ability to mark tasks as completed.



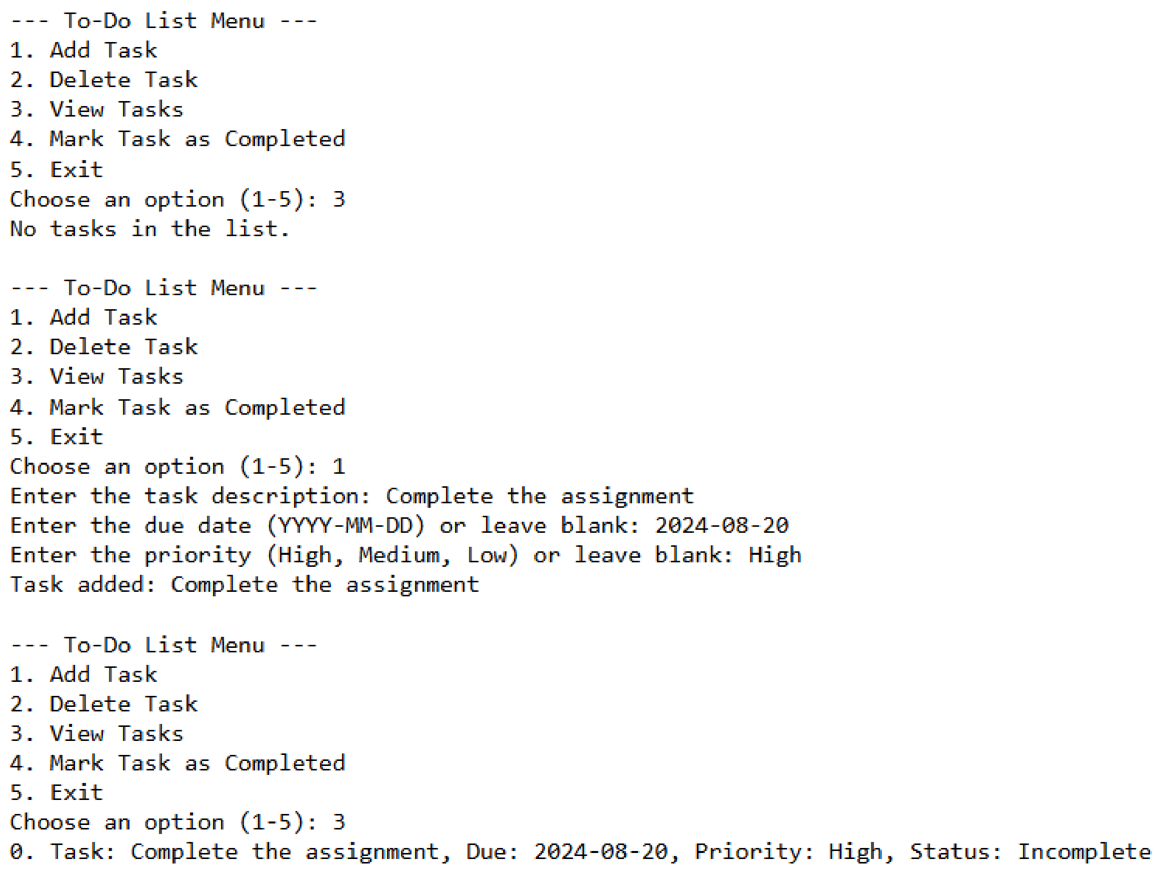
```python
from datetime import datetime

# Define a class for a task
class Task:
    def __init__(self, description, due_date=None, priority=None):
        self.description = description
        self.due_date = due_date
        self.priority = priority
        self.completed = False

    def __str__(self):
        status = "Completed" if self.completed else "Incomplete"
        return f"Task: {self.description}, Due: {self.due_date}, Priority: {self.priority}, Status: {status}"
```



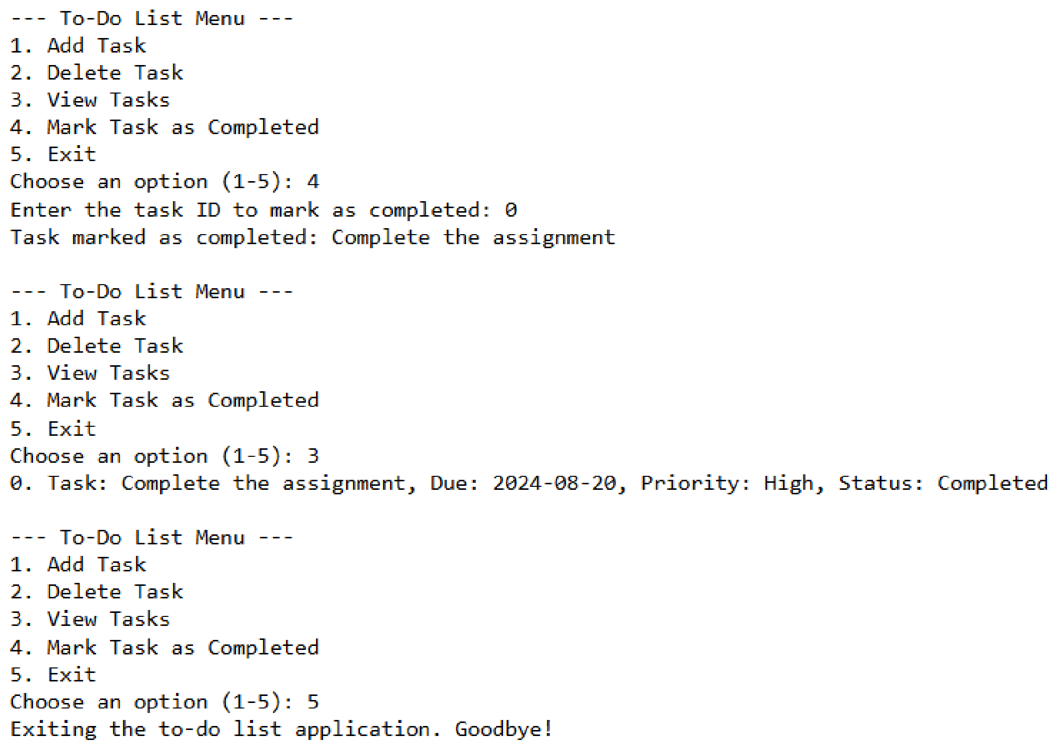
```python
# Define the To-Do List class
class ToDoList:
    def __init__(self):
        self.tasks = []

    def add_task(self, description, due_date=None, priority=None):
        task = Task(description, due_date, priority)
        self.tasks.append(task)
        print(f"Task added: {description}")

    def delete_task(self, task_id):
        if 0 <= task_id < len(self.tasks):
            removed_task = self.tasks.pop(task_id)
            print(f"Task deleted: {removed_task.description}")
        else:
            print("Invalid task ID")

    def view_tasks(self):
        if not self.tasks:
            print("No tasks in the list.")
        else:
            for idx, task in enumerate(self.tasks):
                print(f"{idx}. {task}")

    def mark_task_completed(self, task_id):
        if 0 <= task_id < len(self.tasks):
            self.tasks[task_id].completed = True
            print(f"Task marked as completed: {self.tasks[task_id].description}")
        else:
            print("Invalid task ID")

```

```python
todo_list = ToDoList()

while True:
    print("\n--- To-Do List Menu ---")
    print("1. Add Task")
    print("2. Delete Task")
    print("3. View Tasks")
    print("4. Mark Task as Completed")
    print("5. Exit")

    choice = input("Choose an option (1-5): ")

    if choice == "1":
        description = input("Enter the task description: ")
        due_date_input = input("Enter the due date (YYYY-MM-DD) or leave blank: ")
        due_date = due_date_input if due_date_input else None
        priority = input("Enter the priority (High, Medium, Low) or leave blank: ")
        todo_list.add_task(description, due_date, priority)

    elif choice == "2":
        task_id = int(input("Enter the task ID to delete: "))
        todo_list.delete_task(task_id)

    elif choice == "3":
        todo_list.view_tasks()

    elif choice == "4":
        task_id = int(input("Enter the task ID to mark as completed: "))
        todo_list.mark_task_completed(task_id)

    elif choice == "5":
        print("Exiting the to-do list application. Goodbye!")
        break

    else:
        print("Invalid choice. Please try again.")
```

OUTPUT:



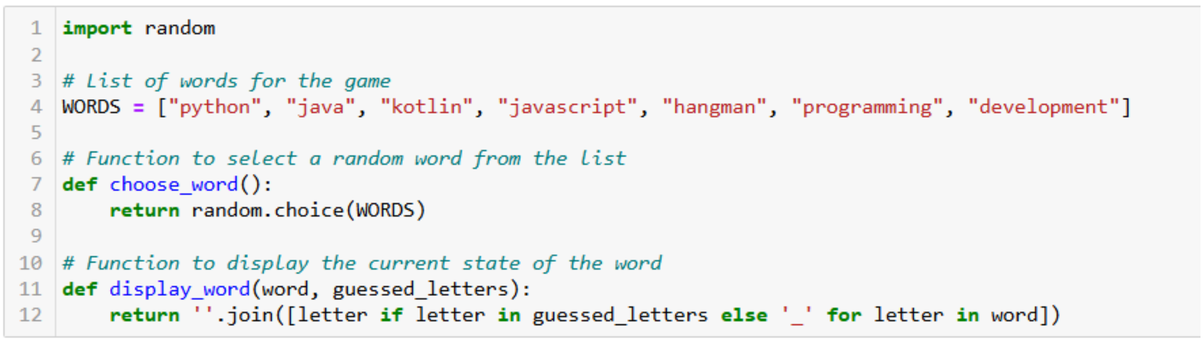
```
--- To-Do List Menu ---
1. Add Task
2. Delete Task
3. View Tasks
4. Mark Task as Completed
5. Exit
Choose an option (1-5): 3
No tasks in the list.

--- To-Do List Menu ---
1. Add Task
2. Delete Task
3. View Tasks
4. Mark Task as Completed
5. Exit
Choose an option (1-5): 1
Enter the task description: Complete the assignment
Enter the due date (YYYY-MM-DD) or leave blank: 2024-08-20
Enter the priority (High, Medium, Low) or leave blank: High
Task added: Complete the assignment

--- To-Do List Menu ---
1. Add Task
2. Delete Task
3. View Tasks
4. Mark Task as Completed
5. Exit
Choose an option (1-5): 3
0. Task: Complete the assignment, Due: 2024-08-20, Priority: High, Status: Incomplete
```



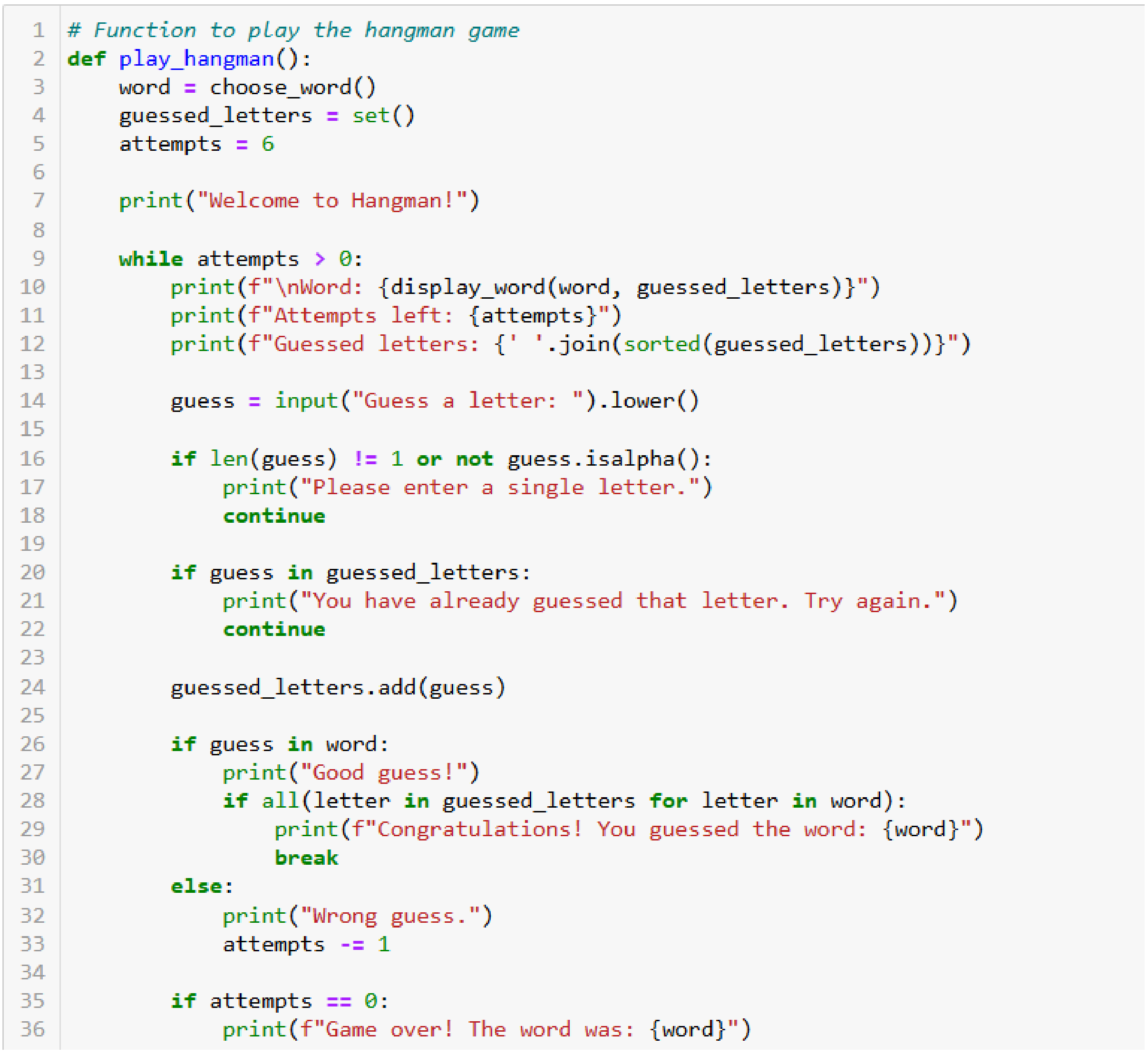
```
--- To-Do List Menu ---
1. Add Task
2. Delete Task
3. View Tasks
4. Mark Task as Completed
5. Exit
Choose an option (1-5): 4
Enter the task ID to mark as completed: 0
Task marked as completed: Complete the assignment

--- To-Do List Menu ---
1. Add Task
2. Delete Task
3. View Tasks
4. Mark Task as Completed
5. Exit
Choose an option (1-5): 3
0. Task: Complete the assignment, Due: 2024-08-20, Priority: High, Status: Completed

--- To-Do List Menu ---
1. Add Task
2. Delete Task
3. View Tasks
4. Mark Task as Completed
5. Exit
Choose an option (1-5): 5
Exiting the to-do list application. Goodbye!
```

Program 12:

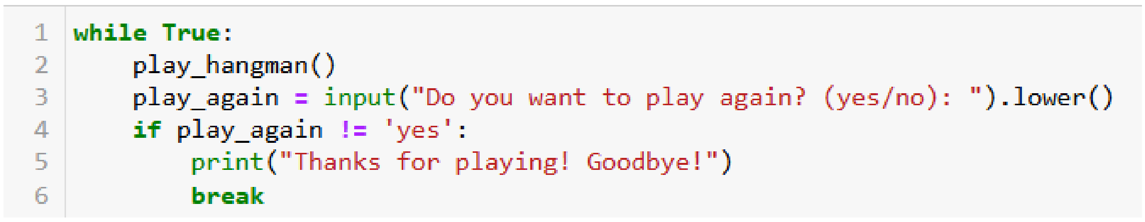
Develop a hangman game where users try to guess a hidden word by suggesting letters within a certain number of attempts.



```python
import random

# List of words for the game
WORDS = ["python", "java", "kotlin", "javascript", "hangman", "programming", "development"]

# Function to select a random word from the list
def choose_word():
    return random.choice(WORDS)

# Function to display the current state of the word
def display_word(word, guessed_letters):
    return ''.join([letter if letter in guessed_letters else '_' for letter in word])
```

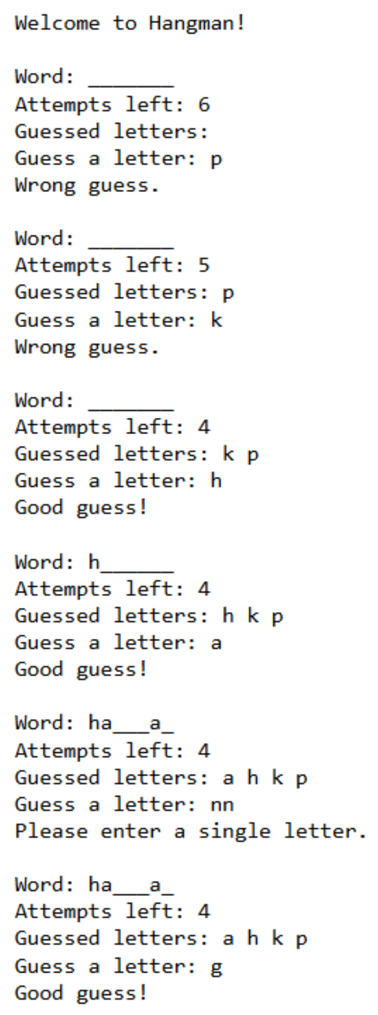


```python
# Function to play the hangman game
def play_hangman():
    word = choose_word()
    guessed_letters = set()
    attempts = 6

    print("Welcome to Hangman!")

    while attempts > 0:
        print(f"\nWord: {display_word(word, guessed_letters)}")
        print(f"Attempts left: {attempts}")
        print(f"Guessed letters: {' '.join(sorted(guessed_letters))}")

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

        if len(guess) != 1 or not guess.isalpha():
            print("Please enter a single letter.")
            continue

        if guess in guessed_letters:
            print("You have already guessed that letter. Try again.")
            continue

        guessed_letters.add(guess)

        if guess in word:
            print("Good guess!")
            if all(letter in guessed_letters for letter in word):
                print(f"Congratulations! You guessed the word: {word}")
                break
        else:
            print("Wrong guess.")
            attempts -= 1

        if attempts == 0:
            print(f"Game over! The word was: {word}")
```

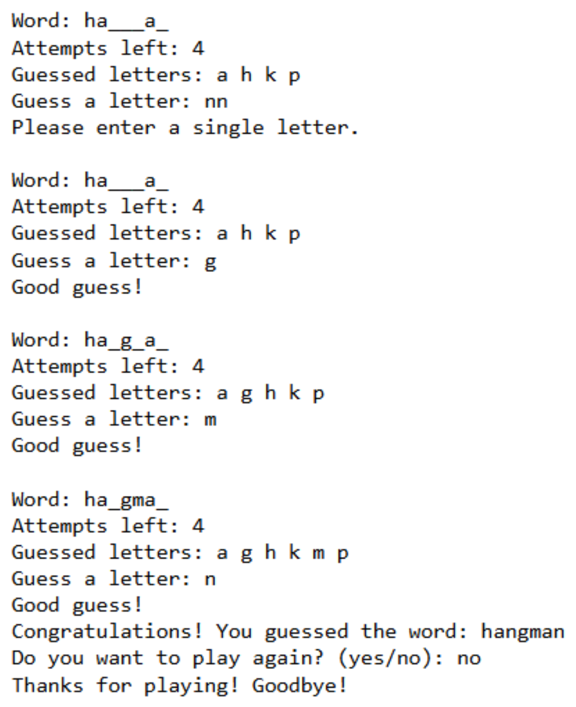


```python
while True:
    play_hangman()
    play_again = input("Do you want to play again? (yes/no): ").lower()
    if play_again != 'yes':
        print("Thanks for playing! Goodbye!")
        break
```

OUTPUT:



```
Welcome to Hangman!

Word: _______
Attempts left: 6
Guessed letters:
Guess a letter: p
Wrong guess.

Word: _______
Attempts left: 5
Guessed letters: p
Guess a letter: k
Wrong guess.

Word: _______
Attempts left: 4
Guessed letters: k p
Guess a letter: h
Good guess!

Word: h______
Attempts left: 4
Guessed letters: h k p
Guess a letter: a
Good guess!

Word: ha___a_
Attempts left: 4
Guessed letters: a h k p
Guess a letter: nn
Please enter a single letter.

Word: ha___a_
Attempts left: 4
Guessed letters: a h k p
Guess a letter: g
Good guess!
```



```
Word: ha___a_
Attempts left: 4
Guessed letters: a h k p
Guess a letter: nn
Please enter a single letter.

Word: ha___a_
Attempts left: 4
Guessed letters: a h k p
Guess a letter: g
Good guess!

Word: ha_g_a_
Attempts left: 4
Guessed letters: a g h k p
Guess a letter: m
Good guess!

Word: ha_gma_
Attempts left: 4
Guessed letters: a g h k m p
Guess a letter: n
Good guess!
Congratulations! You guessed the word: hangman
Do you want to play again? (yes/no): no
Thanks for playing! Goodbye!
```

VIVA QUESTIONS:

1. Explain the difference between integer, floating-point, and string data types in Python.
   Answer: An integer (int) is a whole number without a decimal point (e.g., 5). A floating-point number (float) is a number that includes a decimal point (e.g., 5.0). A string (str) is a sequence of characters enclosed in quotes (e.g., "Hello").

2. How do you perform string concatenation and replication in Python?
   Answer: String concatenation is done using the + operator to join two strings (e.g., "Hello" + "World" results in "HelloWorld"). String replication is achieved using the * operator to repeat a string a specific number of times (e.g., "Hello" * 3 results in "HelloHelloHello").

3. How do you store a value in a variable, and what are the rules for naming variables in Python?
   Answer: To store a value in a variable, use the assignment operator = (e.g., x = 10). Variable names must start with a letter or an underscore, can contain letters, numbers, and underscores, but cannot start with a number. Variable names are case-sensitive.

4. **What are Boolean values in Python, and how are they used in flow control?**
   Answer: Boolean values in Python are True and False. They are used in flow control statements such as if, while, and for loops to determine whether a block of code should execute.

5. **Explain the difference between == and != operators in Python.**
   Answer: The == operator checks if two values are equal (e.g., 5 == 5 returns True). The != operator checks if two values are not equal (e.g., 5 != 3 returns True).

6. **How can you end a Python program early using sys.exit()?**
   Answer: You can use the sys.exit() function to terminate a Python program before it reaches the end. To use this function, you must first import the sys module with import sys.

7. **How do you define a function with parameters in Python, and what is the role of the return statement?**
   Answer: A function is defined using the def keyword followed by the function name and parameters in parentheses.

8. **What is exception handling in Python, and how is it implemented using try and except blocks?**
   Answer: Exception handling allows you to manage runtime errors without crashing the program. It is implemented using try and except blocks

9. **What is a list in Python, and how do you create one?**
   Answer: A list is a mutable, ordered collection of items in Python. You can create a list using square brackets, e.g., my_list = [1, 2, 3].

10. **How do you access and modify elements in a list?**
    Answer: You can access elements using their index, e.g., my_list[0], and modify them by assigning a new value, e.g., my_list[0] = 10.

11. **Explain the difference between append() and extend() methods in lists.**
    Answer: append() adds a single element to the end of the list, e.g., my_list.append(4). extend() adds multiple elements from another iterable, e.g., my_list.extend([4, 5]).

12. **What are augmented assignment operators, and how are they used with lists?**
    Answer: Augmented assignment operators, such as += and *=, modify the list in place. For example, my_list += [4, 5] appends [4, 5] to my_list.

13. **How does the sort() method work in Python, and how is it different from sorted()?**
    Answer: The sort() method sorts the list in place, modifying the original list. sorted() returns a new sorted list, leaving the original list unchanged.

14. **What are the key differences between lists, strings, and tuples in Python?**
    Answer: Lists are mutable, meaning you can modify their elements. Strings and tuples are immutable, so their contents cannot be changed after creation.

15. **How do you use the index() and count() methods with lists?**
    Answer: index() returns the index of the first occurrence of a specified element, e.g., my_list.index(2). count() returns the number of times an element appears in the list, e.g., my_list.count(2).

16. **Explain the concept of references in Python, especially with lists.**
    Answer: In Python, variables hold references to objects. When you assign one list to another, both variables reference the same list, so changes in one variable affect the other.

17. **What is a dictionary in Python, and how is it different from a list?**
    Answer: A dictionary is a collection of key-value pairs, where each key is unique. Unlike lists, dictionaries are unordered, and elements are accessed using keys rather than indexes.

18. **How do you create a dictionary and add or modify elements?**
    Answer: You create a dictionary using curly braces, e.g., my_dict = {'name': 'Alice', 'age': 25}. You can add or modify elements using the assignment operator, e.g., my_dict['age'] = 26.

19. **Explain how to remove elements from a dictionary.**
    Answer: You can remove elements using the del statement, e.g., del my_dict['name'], or the pop() method, e.g., my_dict.pop('name').

20. **What is the pprint module, and how does it help with dictionaries?**
    Answer: The pprint (pretty print) module formats complex data structures, like dictionaries, into a more readable form, especially when printing nested dictionaries.

21. **How can you use dictionaries and lists to model real-world objects?**
    Answer: You can represent real-world objects using dictionaries for attributes and lists for collections of objects. For example, a dictionary can represent a person with attributes like name and age, while a list can hold multiple person dictionaries.

22. **How can you define and manipulate strings in Python?**
    Answer: Strings in Python are sequences of characters defined by enclosing them in quotes, e.g., str = "Hello". You can manipulate strings using indexing, slicing, concatenation (+), and repetition (*).

23. **What is the difference between upper() and lower() string methods?**
    Answer: The upper() method converts all characters in a string to uppercase, e.g., "hello".upper() results in "HELLO". The lower() method converts all characters to lowercase, e.g., "HELLO".lower() results in "hello".

24. **Explain the use of the strip(), lstrip(), and rstrip() methods in Python.**
    Answer: strip() removes whitespace from both ends of a string, lstrip() removes whitespace from the left side, and rstrip() removes whitespace from the right side.

25. **What is the purpose of the join() and split() methods?**
    Answer: The join() method concatenates a list of strings into a single string with a separator, e.g., "-".join(["a", "b", "c"]) results in "a-b-c". The split() method splits a string into a list of substrings based on a delimiter, e.g., "a-b-c".split("-") results in ["a", "b", "c"].

26. **How do you open and read a file in Python?**
    Answer: You can open a file using the open() function, e.g., file = open("example.txt", "r"), and read its contents using methods like read(), readline(), or readlines().

27. **What is the role of the os.path module in file handling?**
    Answer: The os.path module provides functions for manipulating file paths, such as os.path.join() to concatenate paths, os.path.exists() to check if a file exists, and os.path.abspath() to get the absolute path of a file.

28. **Describe the process of writing to a file in Python.**
    Answer: To write to a file, you open it in write mode ("w") or append mode ("a") using open(). Then, you use the write() or writelines() method to add content, e.g., file.write("Hello World"). Finally, close the file using file.close().

29. **What is the shelve module, and how is it used to save variables in Python?**
Answer: The shelve module allows you to store Python objects, such as lists and dictionaries, in a file-like database. You open a shelf file using shelve.open(), store objects using key-value pairs (e.g., shelf["key"] = value), and retrieve them later.

30. **Explain how the print.format() function is used in file writing.**
Answer: The print.format() function allows you to format strings using placeholders, e.g., print("Hello, {}".format(name)). When writing to files, it helps create well-structured output by dynamically inserting values into strings.

31. **How do you handle file paths across different operating systems using os.path.join()?**
Answer: The os.path.join() function automatically handles the appropriate path separators for the operating system. It concatenates directory names and filenames into a single path, ensuring compatibility across different platforms, e.g., os.path.join("folder", "file.txt").

32. **What is the difference between reading a file line by line using readline() and reading the entire file at once using read()?**
Answer: readline() reads one line at a time from the file, allowing you to process it line by line. read() reads the entire file content at once, storing it as a single string. readline() is more memory-efficient for large files.

33. **What is the shutil module, and how is it used in Python?**
Answer: The shutil module provides high-level file operations, such as copying, moving, and deleting files and directories. For example, shutil.copy("source.txt", "destination.txt") copies a file, while shutil.move("source.txt", "destination.txt") moves it.

34. **How can you copy and move files using the shutil module?**
Answer: You can copy files using shutil.copy(source, destination) and move files using shutil.move(source, destination). The copy() function copies the file content and metadata, while move() relocates the file to the specified destination.

35. **What is the purpose of the shutil.rmtree() function?**
Answer: The shutil.rmtree() function is used to delete an entire directory tree, including all files and subdirectories. It is a powerful tool for removing directories and their contents recursively.

36. **Explain the concept of walking a directory tree using os.walk().**
Answer: The os.walk() function generates the file names in a directory tree by walking through it either top-down or bottom-up. It yields a tuple of directory path, directories, and files for each directory in the tree.

37. **What is the shutil.make_archive() function, and how is it used?**
Answer: The shutil.make_archive() function creates an archive file (e.g., ZIP or TAR) from a directory. You specify the base name, format, and root directory, and the function compresses the entire directory into a single archive file.

38. **How does os.walk() differ from shutil.copytree()?**
Answer: os.walk() is used for traversing directories, while shutil.copytree() is used for copying an entire directory tree, including all files and subdirectories. copytree() is more focused on duplication, whereas walk() is for exploration.

39. **What is an exception, and how do you raise one in Python?**
Answer: An exception is an error that occurs during the execution of a program. You can raise an exception using the raise statement, e.g., raise ValueError("Invalid value"), which interrupts the normal flow of the program.

40. **What are assertions in Python, and how are they used for debugging?**
Answer: Assertions are statements that test a condition during execution. If the condition is false, the program raises an AssertionError. They are used to catch bugs early in the development process, e.g., assert x > 0, "x must be positive".

41. **Explain how you can reshape a NumPy array.**
Answer: You can reshape a NumPy array using the reshape() method. For example, a 1D array with 6 elements can be reshaped into a 2D array with 2 rows and 3 columns using array.reshape(2, 3).

42. **What is the difference between flattening and reshaping a NumPy array?**
Answer: Flattening converts a multidimensional array into a 1D array, using methods like array.flatten() or array.ravel(). Reshaping changes the shape of the array but maintains the number of elements, allowing for multi-dimensional structures.

43. **Explain the difference between a DataFrame and a Series in Pandas.**
Answer: A DataFrame is a 2D data structure with rows and columns, while a Series is a 1D data structure that can be thought of as a single column of data. A DataFrame can be considered a collection of Series objects.

44. **How can you compute basic statistics, such as shape, number of columns, and mean values, in Pandas?**
Answer: You can use df.shape to get the shape of the DataFrame, df.columns to get the number of columns, and df.mean() to calculate the mean values of the numerical columns.

45. **How do you manipulate DataFrames in Pandas (e.g., adding or removing columns)?**
Answer: Columns can be added by assigning values to a new column name, e.g., df["new_column"] = values. Columns can be removed using df.drop("column_name", axis=1).

46. **How do you customize plots in Matplotlib by adding titles, labels, and legends?**
Answer: You can customize plots by adding a title using plt.title("Title"), labels using plt.xlabel("X-axis") and plt.ylabel("Y-axis"), and a legend using plt.legend(). These customizations help clarify the data being presented.