Question 1

```
In [ ]:
# Write a Python program that takes a text file as input and prints a dictiona
# where the keys are the words in the file and the values are the number of
# times each word appears in the file.

def word_count(file_path):
    word_dict = {}
    try:
        with open(file_path, 'r') as file:
            for line in file:
                words = line.split()
                for word in words:
                    word = word.lower().strip('.,!?')
                    if word in word_dict:
                        word_dict[word] += 1
                    else:
                        word_dict[word] = 1
        return word_dict
    except FileNotFoundError:
        print("The file was not found.")
    except IOError:
        print("An error occurred while reading the file.")

# Example usage
file_path = 'example.txt'
print(word_count(file_path))
```

Question 2

```
In [1]:
2# Design a Python program that performs the following tasks:
# Ask the user to input a series of integers, one at a time. Store each valid
# ValueError if the input is not a valid integer.
# AssertionError if the input is not between 1 and 100 (inclusive), using asse
# Once the input process is complete, write the list of integers to a text fil
# Open the text file and read the integers back into a new list. Calculate the
# Write the sum and average to a binary file. Handle any file-related exceptio
# Finally, display the contents of the binary file, including both the sum and

def get_integers():
    integers = []
    while True:
        try:
            user_input = input("Enter an integer (or 'done' to finish): ")
            if user_input.lower() == 'done':
                break
            number = int(user_input)
            assert 1 <= number <= 100, "Number must be between 1 and 100"
            integers.append(number)
        except ValueError:
            print("Invalid input. Please enter a valid integer.")
        except AssertionError as e:
            print(e)
```

```python
    return integers

def write_to_text_file(integers, file_path):
    try:
        with open(file_path, 'w') as file:
            for number in integers:
                file.write(f"{number}\n")
    except IOError:
        print("An error occurred while writing to the file.")

def read_from_text_file(file_path):
    integers = []
    try:
        with open(file_path, 'r') as file:
            for line in file:
                integers.append(int(line.strip()))
    except IOError:
        print("An error occurred while reading the file.")
    return integers

def write_to_binary_file(data, file_path):
    import pickle
    try:
        with open(file_path, 'wb') as file:
            pickle.dump(data, file)
    except IOError:
        print("An error occurred while writing to the binary file.")

def read_from_binary_file(file_path):
    import pickle
    try:
        with open(file_path, 'rb') as file:
            data = pickle.load(file)
            return data
    except IOError:
        print("An error occurred while reading the binary file.")

# Main program
integers = get_integers()
text_file_path = 'integers.txt'
binary_file_path = 'results.bin'

write_to_text_file(integers, text_file_path)
integers_from_file = read_from_text_file(text_file_path)

sum_of_integers = sum(integers_from_file)
average_of_integers = sum_of_integers / len(integers_from_file)

write_to_binary_file({'sum': sum_of_integers, 'average': average_of_integers},
results = read_from_binary_file(binary_file_path)

print(f"Sum: {results['sum']}, Average: {results['average']}")
```

```
Enter an integer (or 'done' to finish): 1
Enter an integer (or 'done' to finish): 2
Enter an integer (or 'done' to finish): 3
Enter an integer (or 'done' to finish): 4
Enter an integer (or 'done' to finish): 5
Enter an integer (or 'done' to finish): r
Invalid input. Please enter a valid integer.
Enter an integer (or 'done' to finish): done
Sum: 15, Average: 3.0
```

Question 3

In [5]:
```python
# Create a custom exception NegativeValueError. Write a program that raises th

import math

class NegativeValueError(Exception):
    pass

def calculate_square_root(number):
    if number < 0:
        raise NegativeValueError("Negative value entered.")
    return math.sqrt(number)

try:
    num = float(input("Enter a number: "))
    result = calculate_square_root(num)
    print(f"The square root of {num} is {result}")
except NegativeValueError as e:
    print(e)
except ValueError:
    print("Invalid input. Please enter a valid number.")
```

```
Enter a number: R
Invalid input. Please enter a valid number.
```

Question 4

In [ ]:
```python
# Implement a program that initializes a list and allows the user to access el

def access_list_element(lst, index):
    try:
        return lst[index]
    except IndexError:
        return "Index out of range."

# Example usage
my_list = [10, 20, 30, 40, 50]
index = int(input("Enter the index you want to access: "))
print(access_list_element(my_list, index))
```

Question 5

In [8]:
```python
# Write a Python program that does the following:
# Open a text file in write mode and write the numbers from 1 to 10, each on a
# Reopen the same file in append mode and add the numbers from 11 to 20.
# Finally, open the file in read mode, read all the lines, and print them.

file_path = 'numbers.txt'

# Write numbers 1 to 10
with open(file_path, 'w') as file:
    for i in range(1, 11):
        file.write(f"{i}\n")

# Append numbers 11 to 20
with open(file_path, 'a') as file:
    for i in range(11, 21):
        file.write(f"{i}\n")

# Read and print all lines
with open(file_path, 'r') as file:
    lines = file.readlines()
    for line in lines:
        print(line.strip())
```

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
```

Question 6

In [10]:
```python
# Write a program that writes data to a file in such a way that each character

def process_text(text):
    result = []
    capitalize_next = False
    for char in text:
        if char == '.':
            capitalize_next = True
            result.append(char)
        elif char.isdigit():
            result.append(f"({char})")
        elif capitalize_next and char.isalpha():
            result.append(char.upper())
            capitalize_next = False
        else:
            result.append(char)
    return ''.join(result)

file_path = 'processed_text.txt'
text = "this is a test. it includes numbers like 123 and sentences."

for char in text:
  print(char)


with open(file_path, 'w') as file:
    file.write(process_text(text))

# Example usage
print(process_text(text))
```

this is a test. it includes numbers like 123 and sente

```
n
c
e
s
.
this is a test. It includes numbers like (1)(2)(3) and sentences.
```

Question 7

In [ ]:
```python
# Write a Python program to write a list of strings to a text file, and then a

def write_strings_to_file(file_path, strings):
    try:
        with open(file_path, 'w') as file:
            for string in strings:
                file.write(f"{string}\n")
    except IOError:
        print("An error occurred while writing to the file.")

def append_strings_to_file(file_path, strings):
    try:
        with open(file_path, 'a') as file:
            for string in strings:
                file.write(f"{string}\n")
    except IOError:
        print("An error occurred while appending to the file.")

# Example usage
file_path = 'strings.txt'
initial_strings = ["Hello", "World", "This", "Is", "A", "Test"]
additional_strings = ["Appending", "More", "Strings"]

write_strings_to_file(file_path, initial_strings)
append_strings_to_file(file_path, additional_strings)
```

Question 8

In [ ]:
```python
# Write a program to read a file that contains small case characters. Then wri

def convert_to_uppercase(input_file_path, output_file_path):
    try:
        with open(input_file_path, 'r') as input_file:
            content = input_file.read()
        with open(output_file_path, 'w') as output_file:
            output_file.write(content.upper())
    except IOError:
        print("An error occurred while reading or writing the file.")

# Example usage
input_file_path = 'lowercase.txt'
output_file_path = 'uppercase.txt'
convert_to_uppercase(input_file_path, output_file_path)
```

Question 9

```
In [ ]: # Write a Python program that reads a series of numbers from a file named sour
        # - Sum of the numbers
        # - Average of the numbers
        # - Median of the numbers
        # - Largest number
        # - Smallest number
        # Implement exception handling for:
        # - FileNotFoundError if source.txt is not found.
        # - IOError during reading or writing operations.
        # - If there are any unexpected errors, use a generic exception handler.
        # Ensure that all files are closed properly even if an exception occurs.

        import statistics

        def read_numbers(file_path):
            try:
                with open(file_path, 'r') as file:
                    numbers = [int(line.strip()) for line in file]
                return numbers
            except FileNotFoundError:
                print(f"File {file_path} not found.")
                return []
            except IOError:
                print(f"An error occurred while reading the file {file_path}.")
                return []

        def write_statistics(file_path, stats):
            try:
                with open(file_path, 'w') as file:
                    for key, value in stats.items():
                        file.write(f"{key}: {value}\n")
            except IOError:
                print(f"An error occurred while writing to the file {file_path}.")

        def calculate_statistics(numbers):
            if not numbers:
                return {}
            stats = {
                'Sum': sum(numbers),
                'Average': statistics.mean(numbers),
                'Median': statistics.median(numbers),
                'Largest number': max(numbers),
                'Smallest number': min(numbers)
            }
            return stats

        # Main program
        source_file_path = 'source.txt'
        destination_file_path = 'destination.txt'

        numbers = read_numbers(source_file_path)
```

```
stats = calculate_statistics(numbers)
write_statistics(destination_file_path, stats)

# Example usage
print(f"Statistics written to {destination_file_path}")
```

Question 10

In [ ]:
```
# Write a Python program that reads numbers from a file named source.txt and w
# 1. Reading from a File:
# - The program should read numbers line by line from the file source.txt.
# 2. Error Handling:
# - If the file source.txt is not found, raise a FileNotFoundError exception a
# - If any of the numbers read from the file are even, raise a custom exceptio
# 3. Writing to a File:
# - If there are no even numbers, write all the numbers to the file destinatio
# 4. Closing Files:
# - Ensure that both files (source.txt and destination.txt) are properly close

class EvenNumberError(Exception):
    pass

def read_and_write_numbers(source_file, destination_file):
    try:
        with open(source_file, 'r') as src:
            numbers = [int(line.strip()) for line in src]
            for number in numbers:
                if number % 2 == 0:
                    raise EvenNumberError(f"Even number {number} encountered."
        with open(destination_file, 'w') as dest:
            for number in numbers:
                dest.write(f"{number}\n")
    except FileNotFoundError:
        print(f"File {source_file} not found.")
    except EvenNumberError as e:
        print(e)
    except IOError:
        print("An error occurred during file operations.")
    finally:
        try:
            src.close()
            dest.close()
        except NameError:
            pass

# Example usage
source_file_path = 'source.txt'
destination_file_path = 'destination.txt'
read_and_write_numbers(source_file_path, destination_file_path)
```

Question 11

In [ ]:
```
# Write a program to compare two files and write the report of comparison in a

def compare_files(file1_path, file2_path, report_file_path):
    try:
        with open(file1_path, 'r') as file1, open(file2_path, 'r') as file2:
            file1_lines = file1.readlines()
            file2_lines = file2.readlines()

        differences = []
        for i, (line1, line2) in enumerate(zip(file1_lines, file2_lines), star
            if line1 != line2:
                differences.append(f"Line {i}:\nFile1: {line1}\nFile2: {line2}

        with open(report_file_path, 'w') as report_file:
            if differences:
                report_file.writelines(differences)
            else:
                report_file.write("The files are identical.")

    except FileNotFoundError as e:
        print(f"File not found: {e.filename}")
    except IOError as e:
        print(f"An error occurred: {e}")

# Example usage
file1_path = 'file1.txt'
file2_path = 'file2.txt'
report_file_path = 'comparison_report.txt'
compare_files(file1_path, file2_path, report_file_path)
```

Question 12

In [ ]:
```
# Write a program having a function with a logical error, such as using the wr

def find_maximum(a, b, c):
    if a > b and a > c:
        return a
    elif b > a and b > c:
        return b
    else:
        return c

# Example usage
print(find_maximum(10, 20, 30))  # Logical error: This will always return the
```

Question 13

In [ ]:
```
# Write a python program to raise an exception ‘FileNotFoundError’ while writi

def write_binary_file(file_path, data):
    try:
        with open(file_path, 'wb') as file:
            file.write(data)
    except FileNotFoundError:
        print(f"File not found: {file_path}")
    except IOError as e:
        print(f"An error occurred: {e}")

# Example usage
file_path = 'non_existent_directory/file.bin'
data = b'This is binary data.'
write_binary_file(file_path, data)
```

Question 14

In [ ]:
```
# Write a program that exchanges the contents of two files, file_A and file_B.
# - FileNotFoundError if file_A.txt is not found.
# - IOError during reading or writing operations.
# - If there are any unexpected errors, use a generic exception handler.

def exchange_file_contents(file_A_path, file_B_path):
    try:
        with open(file_A_path, 'r') as file_A, open(file_B_path, 'r') as file_
            content_A = file_A.read()
            content_B = file_B.read()

        with open(file_A_path, 'w') as file_A, open(file_B_path, 'w') as file_
            file_A.write(content_B)
            file_B.write(content_A)

    except FileNotFoundError as e:
        print(f"File not found: {e.filename}")
    except IOError as e:
        print(f"An error occurred: {e}")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")

# Example usage
file_A_path = 'file_A.txt'
file_B_path = 'file_B.txt'
exchange_file_contents(file_A_path, file_B_path)
```

Question 15

In [ ]:
```
# Write a Python program that reads a list of student names from a text file c

def read_student_names(file_path):
    try:
        with open(file_path, 'r') as file:
            students = file.readlines()
            students = [student.strip() for student in students]  # Remove any
        return students
    except FileNotFoundError:
        print(f"{file_path} does not exist. Creating the file.")
```

```python
        return []

def append_student_names(file_path, new_students):
    with open(file_path, 'a') as file:
        for student in new_students:
            file.write(student + "\n")

def display_student_names(students):
    if students:
        print("\nList of students:")
        for student in students:
            print(student)
    else:
        print("No students found.")

# Main program
file_path = 'students.txt'
# Read existing students from the file
students = read_student_names(file_path)
# Display existing students
display_student_names(students)

# Prompt user to enter new student names
new_students = []
while True:
    new_student = input("Enter a new student name (or type 'done' to finish):
    if new_student.lower() == 'done':
        break
    new_students.append(new_student)

# Append new student names to the file
append_student_names(file_path, new_students)
# Read and display all student names
students = read_student_names(file_path)
display_student_names(students)
```

Question 16

In [ ]:

```python
# Write a program that accepts date of birth along with other personal details

from datetime import datetime

def get_personal_details():
    name = input("Enter your name: ")
    dob = input("Enter your date of birth (dd-mm-yyyy): ")
    try:
        dob_date = datetime.strptime(dob, "%d-%m-%Y")
        print(f"Name: {name}, Date of Birth: {dob_date.strftime('%d-%m-%Y')}")
    except ValueError:
        print("Invalid date format. Please enter the date in dd-mm-yyyy format

# Example usage
get_personal_details()
```

Question 17

In [ ]:
```
# Write a Python program that handles multiple exceptions, such as ValueError,
# Prompt the user to enter two numbers and divide them. Handle invalid input a

def divide_numbers():
    try:
        num1 = float(input("Enter the first number: "))
        num2 = float(input("Enter the second number: "))
        result = num1 / num2
        print(f"The result of division is: {result}")
    except ValueError:
        print("Invalid input. Please enter valid numbers.")
    except ZeroDivisionError:
        print("Division by zero is not allowed.")
    except TypeError:
        print("An error occurred with the data types.")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")

# Example usage
divide_numbers()
```

Question 18

In [ ]:
```
# Write a Python program that reads a file and finds the longest word in the f

def find_longest_word(file_path):
    try:
        with open(file_path, 'r') as file:
            words = file.read().split()
            longest_word = max(words, key=len)
        with open('longest_word.txt', 'w') as output_file:
            output_file.write(longest_word)
        print(f"The longest word is: {longest_word}")
    except FileNotFoundError:
        print(f"File {file_path} not found.")
    except IOError:
        print("An error occurred while reading or writing the file.")

# Example usage
file_path = 'example.txt'
find_longest_word(file_path)
```

Question 19

In [ ]:
```
# Write a Python program that reads a file and finds the longest word in the f

def find_longest_word(file_path):
    try:
        with open(file_path, 'r') as file:
            words = file.read().split()
            longest_word = max(words, key=len)
        with open('longest_word.txt', 'w') as output_file:
            output_file.write(longest_word)
        print(f"The longest word is: {longest_word}")
    except FileNotFoundError:
        print(f"File {file_path} not found.")
    except IOError:
        print("An error occurred while reading or writing the file.")

# Example usage
file_path = 'example.txt'
find_longest_word(file_path)
```

Question 20

```
In [ ]: # Write a program that receives 10 integers and stores them and their cubes in

class NumberTooSmall(Exception):
    pass

class NumberTooBig(Exception):
    pass

def get_integer_input():
    while True:
        try:
            number = int(input("Enter an integer: "))
            if number < 3:
                raise NumberTooSmall("Number is too small.")
            elif number > 30:
                raise NumberTooBig("Number is too big.")
            return number
        except ValueError:
            print("Invalid input. Please enter a valid integer.")
        except NumberTooSmall as e:
            print(e)
        except NumberTooBig as e:
            print(e)

def main():
    numbers_dict = {}
    for _ in range(10):
        number = get_integer_input()
        numbers_dict[number] = number ** 3
    print("Numbers and their cubes:", numbers_dict)

# Example usage
main()
```

Question 21

```
In [ ]: # Write a Python program to perform the following tasks with binary files and
```

```
# - Create and Write to a Binary File
# - Read and Display Binary File Contents
# - Append Data to the Binary File
# - Handle File Exceptions

import pickle

def create_and_write_binary_file(file_path, data):
    try:
        with open(file_path, 'wb') as file:
            pickle.dump(data, file)
    except IOError:
        print("An error occurred while writing to the binary file.")

def read_and_display_binary_file(file_path):
    try:
        with open(file_path, 'rb') as file:
            data = pickle.load(file)
            print("Binary file contents:", data)
    except FileNotFoundError:
        print(f"File {file_path} not found.")
    except IOError:
        print("An error occurred while reading the binary file.")

def append_to_binary_file(file_path, data):
    try:
        with open(file_path, 'ab') as file:
            pickle.dump(data, file)
    except IOError:
        print("An error occurred while appending to the binary file.")

# Example usage
file_path = 'data.bin'
data_to_write = {'key1': 'value1', 'key2': 'value2'}
create_and_write_binary_file(file_path, data_to_write)
read_and_display_binary_file(file_path)
data_to_append = {'key3': 'value3'}
append_to_binary_file(file_path, data_to_append)
read_and_display_binary_file(file_path)
```

Question 22

In [ ]:
```
# Write a Python program that reads the file to find all the palindrome words

def find_palindromes(file_path):
    try:
        with open(file_path, 'r') as file:
            words = file.read().split()
            palindromes = [word for word in words if word == word[::-1]]
        print("Palindrome words:", palindromes)
    except FileNotFoundError:
        print(f"File {file_path} not found.")
    except IOError:
        print("An error occurred while reading the file.")

# Example usage
file_path = 'example.txt'
find_palindromes(file_path)
```

Question 23

```
In [ ]: # Write a program that reads a file line by line. Each line read from the file

def copy_with_line_numbers(input_file_path, output_file_path):
    try:
        with open(input_file_path, 'r') as input_file, open(output_file_path,
            for line_number, line in enumerate(input_file, start=1):
                output_file.write(f"{line_number}: {line}")
    except FileNotFoundError:
        print(f"File {input_file_path} not found.")
    except IOError:
        print("An error occurred while reading or writing the file.")

# Example usage
input_file_path = 'input.txt'
output_file_path = 'output.txt'
copy_with_line_numbers(input_file_path, output_file_path)
```

Question 24

```
In [ ]: # Develop a Python program that reads an integer from the user. Implement exce

def read_positive_integer():
    try:
        number = int(input("Enter a positive integer: "))
        assert number > 0, "The number must be positive."
        print(f"You entered: {number}")
    except ValueError:
        print("Invalid input. Please enter a valid integer.")
    except AssertionError as e:
        print(e)

# Example usage
read_positive_integer()
```

Question 25

```
In [ ]: # Write python program to search for a string in text file using file handling

def search_string_in_file(file_path, search_string):
    try:
        with open(file_path, 'r') as file:
            lines = file.readlines()
            for line_number, line in enumerate(lines, start=1):
                if search_string in line:
                    print(f"Found '{search_string}' in line {line_number}: {li
    except FileNotFoundError:
        print(f"File {file_path} not found.")
    except IOError:
        print("An error occurred while reading the file.")

# Example usage
file_path = 'example.txt'
search_string = 'search_term'
search_string_in_file(file_path, search_string)
```

Question 26

```
In [ ]: # Write a Python program to accept two numbers and display the quotient. Appro

def divide_numbers():
    try:
        num1 = float(input("Enter the first number: "))
        num2 = float(input("Enter the second number: "))
        if num2 == 0:
            raise ZeroDivisionError("The denominator cannot be zero.")
        result = num1 / num2
        print(f"The result of division is: {result}")
    except ValueError:
        print("Invalid input. Please enter valid numbers.")
    except ZeroDivisionError as e:
        print(e)

# Example usage
divide_numbers()
```

Question 27

```
In [ ]: # Write a Python program to create a file where all letters of English alphabe

def write_alphabet_to_file(file_path, letters_per_line):
    try:
        with open(file_path, 'w') as file:
            alphabet = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
            for i in range(0, len(alphabet), letters_per_line):
                file.write(alphabet[i:i+letters_per_line] + '\n')
    except IOError:
        print("An error occurred while writing to the file.")

# Example usage
file_path = 'alphabet.txt'
letters_per_line = 5
write_alphabet_to_file(file_path, letters_per_line)
```

Question 28

```
In [ ]: # Write a Python program that reads an existing binary file, converts it to st
```

```python
def read_binary_file(file_path):
    try:
        with open(file_path, 'rb') as file:
            binary_data = file.read()
            string_data = binary_data.decode('utf-8')
            print(string_data)
    except FileNotFoundError:
        print(f"File {file_path} not found.")
    except IOError:
        print("An error occurred while reading the file.")

# Example usage
file_path = 'example.bin'
read_binary_file(file_path)
```

Question 29

In [ ]:

```python
# Design a Python program that integrates following into single program:
# 1. Ask the user for two numbers, divide them, and handle any runtime errors.
# 2. Use assertions to ensure the result of the division is positive.
# 3. Write the result to a text file in append mode.
# 4. Create a binary file and store the division result in it.
# 5. Implement exception handling to manage both file operations and potential

import pickle

def divide_and_store_results():
    try:
        num1 = float(input("Enter the first number: "))
        num2 = float(input("Enter the second number: "))
        result = num1 / num2
        assert result > 0, "The result of the division must be positive."

        # Write result to text file
        with open('results.txt', 'a') as text_file:
            text_file.write(f"Result: {result}\n")

        # Write result to binary file
        with open('results.bin', 'wb') as binary_file:
            pickle.dump(result, binary_file)

        print(f"The result of division is: {result}")
    except ValueError:
        print("Invalid input. Please enter valid numbers.")
    except ZeroDivisionError:
        print("Division by zero is not allowed.")
    except AssertionError as e:
        print(e)
    except IOError:
        print("An error occurred while writing to the file.")

# Example usage
```

```
divide_and_store_results()
```

Question 30

```
In [ ]: # Write a Python program that writes a list of integers to a binary file and t

import pickle

def write_integers_to_binary_file(file_path, integers):
    try:
        with open(file_path, 'wb') as file:
            pickle.dump(integers, file)
    except IOError:
        print("An error occurred while writing to the binary file.")

def read_integers_from_binary_file(file_path):
    try:
        with open(file_path, 'rb') as file:
            integers = pickle.load(file)
            return integers
    except FileNotFoundError:
        print(f"File {file_path} not found.")
    except IOError:
        print("An error occurred while reading the binary file.")
        return []

# Example usage
file_path = 'integers.bin'
integers_to_write = [1, 2, 3, 4, 5]
write_integers_to_binary_file(file_path, integers_to_write)
integers_read = read_integers_from_binary_file(file_path)
print("Integers read from binary file:", integers_read)
```

Question 31

```
In [ ]: # Write a python program that finds prime numbers from 1 to 100 and appends th

def is_prime(number):
    if number <= 1:
        return False
    for i in range(2, int(number ** 0.5) + 1):
        if number % i == 0:
            return False
    return True

def write_primes_to_file(file_path):
    try:
        with open(file_path, 'w') as file:
            prime_count = 0
            for number in range(1, 101):
                if is_prime(number):
                    prime_count += 1
                    file.write(f"The {prime_count} prime number is: {number}\r
    except IOError:
        print("An error occurred while writing to the file.")

# Example usage
file_path = 'primes.txt'
write_primes_to_file(file_path)
```

Question 32

In [ ]:
```
# What are different file opening modes supported in Python Programming Langua

file_modes = {
    'r': 'Read mode - Opens a file for reading. If the file does not exist, ra
    'w': 'Write mode - Opens a file for writing. If the file exists, truncates
    'a': 'Append mode - Opens a file for appending. If the file does not exist
    'r+': 'Read and write mode - Opens a file for both reading and writing. If
    'w+': 'Write and read mode - Opens a file for both writing and reading. If
    'a+': 'Append and read mode - Opens a file for both appending and reading.
    'rb': 'Read binary mode - Opens a file for reading in binary mode. If the
    'wb': 'Write binary mode - Opens a file for writing in binary mode. If the
    'ab': 'Append binary mode - Opens a file for appending in binary mode. If
```