File handling, Exception Handling, OOPs

October 22, 2024

[107]:
```
# Exercise 1

# create file

open('test.txt', 'x')

# write in file

file = open('test.txt', 'w')
file.write('Hello World!')

# Read File

file = open('test.txt', 'r')
print(file.read())
```

```
Hello World!
```

[119]:
```
# Exercise 2

file_to_read ="test.txt"
write_to_file="WriteData.txt"

# Reading a file
file = open(file_to_read,"r")
data = file.read()
file.close()

# Writing to a file
with open(write_to_file,"a") as file:    # with method auto closes the file object
    file.write(data)
print('completed')
```

```
completed
```

[121]:
```
# Exercise 3

count = 0;
```

```
#Opens a file in read mode
file = open("test.txt", "r")

#Gets each line till end of file is reached
for line in file:
    #Splits each line into words
    words = line.split(" ");
    #Counts each word
    count = count + len(words);

print("Number of words present in given file: " + str(count));
file.close();
```

```
Number of words present in given file: 2
```

[125]:
```
# Exercise 4

# Open the file in read mode
text = open("test.txt", "r")

# Create an empty dictionary
d = dict()

# Loop through each line of the file
for line in text:
    # Remove the leading spaces and newline character
    line = line.strip()

    # Convert the characters in line to
    # lowercase to avoid case mismatch
    line = line.lower()

    # Split the line into words
    words = line.split(" ")


    # Iterate over each word in line
    for word in words:
        # Check if the word is already in dictionary
        if word in d:
            # Increment count of word by 1
            d[word] = d[word] + 1
        else:
            # Add the word to dictionary with count 1
            d[word] = 1
```

```
# Print the contents of dictionary
for key in list(d.keys()):
    print(key, ":", d[key])
```

```
hello : 1
world! : 1
```

[129]:
```
# Exercise 5

def convert_to_integer():
    user_input = input("2250 ")

    try:
        # Try to convert the input to an integer
        number = int(user_input)
        print(f"The integer value is: {number}")

    except ValueError:
        # Handle the case where the input cannot be converted to an integer
        print("Invalid input! Please enter a valid integer.")

# Call the function to test it
convert_to_integer()
```

```
2250  5
```

```
The integer value is: 5
```

[131]:
```
# Exercise 6

class NegativeNumberError(Exception):
    """Custom exception for handling negative numbers."""
    pass

def check_for_negative_numbers(numbers):
    for num in numbers:
        if num < 0:
            raise NegativeNumberError(f"Negative number found: {num}")

def input_integers():
    try:
        # Prompt the user to input a list of integers (separated by spaces)
        user_input = input("Enter a list of integers separated by spaces: ")

        # Convert the input string to a list of integers
        numbers = list(map(int, user_input.split()))

        # Check for any negative numbers
```

```
        check_for_negative_numbers(numbers)

        print("All numbers are non-negative.")

    except ValueError:
        # Handle the case where the input contains non-integer values
        print("Invalid input! Please enter only integers.")

    except NegativeNumberError as e:
        # Handle the case where a negative number is found
        print(e)

# Call the function to test it
input_integers()
```

```
Enter a list of integers separated by spaces:  2 8 11 89 45 62
All numbers are non-negative.
```

[133]:
```
# Exercise 7

def compute_average():
    try:
        # Prompt the user to input a list of integers separated by spaces
        user_input = input("Enter a list of integers separated by spaces: ")

        # Convert the input string to a list of integers
        numbers = list(map(int, user_input.split()))

        if len(numbers) == 0:
            raise ValueError("The list is empty. Cannot compute the average.")

        # Compute the average
        average = sum(numbers) / len(numbers)
        print(f"The average of the entered integers is: {average}")

    except ValueError as e:
        # Handle non-integer inputs or an empty list
        print(f"Error: {e}")

    finally:
        # This block will always run regardless of whether an exception occurred
        print("Program has finished running.")

# Call the function to run the program
compute_average()
```

```
Enter a list of integers separated by spaces:  22 5 8 96 77 52 56 45 68 25 34 12
```

```
The average of the entered integers is: 41.666666666666664
Program has finished running.
```

[135]:
```python
# Exercise 8

def write_to_file():
    try:
        # Prompt the user to input a filename
        file_name = input("Enter the filename: ")

        # Prompt the user to input the string to write to the file
        content = input("Enter the content you want to write to the file: ")

        # Open the file in write mode and write the content
        with open(file_name, 'w') as file:
            file.write(content)

        # If no exceptions occur, print a welcome message
        print("Content written to file successfully! Welcome!")

    except IOError:
        # Handle any file I/O errors (e.g., permission denied or invalid file␣
 ↪name)
        print("An error occurred while trying to write to the file.")

# Call the function to run the program
write_to_file()
```

```
Enter the filename:  test.txt
Enter the content you want to write to the file:  My Name is Renjitha

Content written to file successfully! Welcome!
```

[137]:
```python
# Exercise 9

# Base class for all courses
class Course:
    def __init__(self, course_code, course_name, credit_hours):
        self.course_code = course_code
        self.course_name = course_name
        self.credit_hours = credit_hours

    def __str__(self):
        return f"{self.course_code} - {self.course_name} ({self.credit_hours}␣
 ↪credit hours)"

# Subclass for core courses
class CoreCourse(Course):
```

```
    def __init__(self, course_code, course_name, credit_hours,␣
 ↪required_for_major):
        super().__init__(course_code, course_name, credit_hours)
        self.required_for_major = required_for_major

    def __str__(self):
        requirement = "Required" if self.required_for_major else "Not required"
        return f"{super().__str__()} - {requirement} for the major"

# Subclass for elective courses
class ElectiveCourse(Course):
    def __init__(self, course_code, course_name, credit_hours, elective_type):
        super().__init__(course_code, course_name, credit_hours)
        self.elective_type = elective_type

    def __str__(self):
        return f"{super().__str__()} - Elective Type: {self.elective_type}"

# Example of managing courses
def main():
    # Creating some core courses
    core_course1 = CoreCourse("CS101", "Introduction to Computer Science", 3,␣
 ↪True)
    core_course2 = CoreCourse("MATH201", "Calculus I", 4, False)

    # Creating some elective courses
    elective_course1 = ElectiveCourse("ART101", "Introduction to Art", 3,␣
 ↪"liberal arts")
    elective_course2 = ElectiveCourse("CS301", "Data Structures", 3, "technical")

    # Displaying course information
    print(core_course1)
    print(core_course2)
    print(elective_course1)
    print(elective_course2)

# Run the program
if __name__ == "__main__":
    main()
```

```
CS101 - Introduction to Computer Science (3 credit hours) - Required for the
major
MATH201 - Calculus I (4 credit hours) - Not required for the major
ART101 - Introduction to Art (3 credit hours) - Elective Type: liberal arts
CS301 - Data Structures (3 credit hours) - Elective Type: technical
```

[155]:
```python
# Exercise 10

class Employee:
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary

    def get_name(self):
        return self.name

    def get_salary(self):
        return self.salary

# Create an Employee object
employee1 = Employee("John Doe", 50000)

# Display the employee's name and salary
print(f"Employee Name: {employee1.get_name()}")
print(f"Employee Salary: ${employee1.get_salary()}")
```

```
Employee Name: John Doe
Employee Salary: $50000
```

[ ]: