untitled23

May 18, 2024

0.0.1 Q1 a. Create a text file named "sample.txt" with the content "Hello, File Handling!". Write a Python program to open the file, read its content, and print it. Now, count the number of words in the file. Print the word count.

[1]:
```python
# Creating the file with the given content
with open("sample.txt", "w") as file:
    file.write("Hello, File Handling!")

# Reading the content of the file and printing it
with open("sample.txt", "r") as file:
    content = file.read()
    print(content)  # Output: Hello, File Handling!

# Counting the number of words in the file
word_count = len(content.split())
print(f"Word count: {word_count}")  # Output: Word count: 3
```

```
Hello, File Handling!
Word count: 3
```

0.0.2 b. Create a new text file named "output.txt" and write three lines of text into the file. Read the lines.

[2]:
```python
# Writing three lines of text to "output.txt"
with open("output.txt", "w") as file:
    file.write("First line of text.\n")
    file.write("Second line of text.\n")
    file.write("Third line of text.\n")

# Reading the lines from "output.txt"
with open("output.txt", "r") as file:
    lines = file.readlines()
    for line in lines:
        print(line.strip())  # Output: Prints each line without additional␣
 ↪newline characters
```

```
First line of text.
Second line of text.
Third line of text.
```

0.0.3 Q2 a. Create a text file named “numbers.txt” with numbers 1 to 5 each on a new line. Write a Python program to open the file, read its content line by line, and print their sum.

[3]:
```python
# Creating the file with numbers 1 to 5
with open("numbers.txt", "w") as file:
    for i in range(1, 6):
        file.write(f"{i}\n")

# Reading the file and calculating the sum
total_sum = 0
with open("numbers.txt", "r") as file:
    for line in file:
        total_sum += int(line.strip())

print(f"Sum: {total_sum}")  # Output: Sum: 15
```

```
Sum: 15
```

0.0.4 b. Write a Python function that opens a file named “example.txt”, reads its content, and prints the number of occurrences of each unique word in the file. Assume that words are separated by spaces, and ignore punctuation.

[6]:
```python
import string

def word_count(filename):
    with open(filename, "r") as file:
        content = file.read()

    # Removing punctuation and converting to lower case
    content = content.translate(str.maketrans("", "", string.punctuation)).
 ↪lower()
    words = content.split()

    word_frequency = {}
    for word in words:
        if word in word_frequency:
            word_frequency[word] += 1
        else:
            word_frequency[word] = 1

    for word, count in word_frequency.items():
        print(f"{word}: {count}")

# Example usage
# Assuming "example.txt" contains some text for testing
word_count("numbers.txt")
```

```
1: 1
2: 1
3: 1
4: 1
5: 1
```

0.0.5 Q3 Write a Python function that takes a file path as input and prints the total number of lines in the file.

[7]:
```python
def count_lines(file_path):
    with open(file_path, "r") as file:
        lines = file.readlines()
        print(f"Total number of lines: {len(lines)}")

# Example usage
# Assuming "example.txt" contains some text for testing
count_lines("sample.txt")
```

```
Total number of lines: 1
```

#EXCEPTIONAL HANDLING

0.0.6 QUESTION 1:

[8]:
```python
class NegativeValueError(Exception):
    pass

class InvalidNumberError(Exception):
    pass

def calculate_average_grades():
    while True:
        try:
            num_students = int(input("Enter the total number of students: "))
            if num_students <= 0:
                raise InvalidNumberError("Number of students must be a positive␣
 ↪integer.")
            break
        except ValueError:
            print("Invalid input. Please enter a numeric value.")
        except InvalidNumberError as e:
            print(e)

    students = []

    for _ in range(num_students):
        while True:
            try:
                name = input("Enter the student's name: ")
                marks = []
                for i in range(3):
                    mark = float(input(f"Enter the mark for subject {i + 1}␣
↪(out of 100): "))
                    if mark < 0:
                        raise NegativeValueError("Marks cannot be negative.")
                    marks.append(mark)
                average = sum(marks) / len(marks)
                students.append((name, average))
                break
            except ValueError:
                print("Invalid input. Please enter numeric values for marks.")
            except NegativeValueError as e:
                print(e)
            except ZeroDivisionError:
                print("Zero Division Error. No marks provided.")

    overall_average = sum([student[1] for student in students]) / len(students)

    print("\nStudent Averages:")
    for name, average in students:
        print(f"{name}: {average:.2f}")

    print(f"\nOverall Class Average: {overall_average:.2f}")

    print("Thank you for using the grade calculator. Goodbye!")

if __name__ == "__main__":
    while True:
        calculate_average_grades()
        try_again = input("Do you want to calculate for another class? (yes/no):
↪ ").strip().lower()
        if try_again != 'yes':
            print("Exiting the program. Have a great day!")
            break
```

```
Enter the total number of students: 5
Enter the student's name: sai
Enter the mark for subject 1 (out of 100): 75
Enter the mark for subject 2 (out of 100): 80
Enter the mark for subject 3 (out of 100): 87
Enter the student's name: krishna
Enter the mark for subject 1 (out of 100): 57
Enter the mark for subject 2 (out of 100): 45
Enter the mark for subject 3 (out of 100): 34
Enter the student's name: ravi
Enter the mark for subject 1 (out of 100): 94
Enter the mark for subject 2 (out of 100): 56
Enter the mark for subject 3 (out of 100): 76
Enter the student's name: gopi
Enter the mark for subject 1 (out of 100): 65
Enter the mark for subject 2 (out of 100): 67
Enter the mark for subject 3 (out of 100): 87
Enter the student's name: david
Enter the mark for subject 1 (out of 100): 65
Enter the mark for subject 2 (out of 100): 76
Enter the mark for subject 3 (out of 100): 89

Student Averages:
sai: 80.67
krishna: 45.33
ravi: 75.33
gopi: 73.00
david: 76.67

Overall Class Average: 70.20
Thank you for using the grade calculator. Goodbye!
Do you want to calculate for another class? (yes/no): no
Exiting the program. Have a great day!
```

0.0.7 Q2: File Manipulation in Python

```
[9]: def transfer_content(source_file, target_file):
         try:
             with open(source_file, 'r') as src:
                 content = src.read()
             with open(target_file, 'w') as tgt:
                 tgt.write(content)
             print(f"Content transferred from '{source_file}' to '{target_file}'␣
       ↪successfully.")
         except FileNotFoundError:
             print("File not found. Please check the file paths.")

     # Example usage:
     source_path = 'path/to/source_file.txt'
     target_path = 'path/to/target_file.txt'
     transfer_content(source_path, target_path)
```

```
File not found. Please check the file paths.
```

0.0.8 b. Identify and Correct Erroneous Text within a File:

[10]:
```
def correct_errors(file_path):
    try:
        with open(file_path, 'r') as file:
            content = file.read()
            # Implement your error correction logic here
            corrected_content = content.replace('erroneous', 'corrected')
        with open('corrected_file.txt', 'w') as corrected_file:
            corrected_file.write(corrected_content)
        print(f"Corrected content saved in 'corrected_file.txt'.")
    except FileNotFoundError:
        print("File not found. Please check the file path.")

# Example usage:
file_path = 'path/to/your_file.txt'
correct_errors(file_path)
```

```
File not found. Please check the file path.
```

0.0.9 3) Types of errors and exceptional handling in python

Built-in Exceptions in Python: Python provides a set of built-in exceptions that you can handle in your code **ZeroDivisionError:** Raised when dividing by zero. **NameError:** Occurs when a variable is not found. **AssertionError:** Raised when an assert statement fails. **EOFError:** Raised when the input() function reaches the end-of-file condition. **AttributeError:** Raised when an attribute assignment or reference fails. **TabError:** Raised for inconsistent tab or space indentations. **ImportError:** Raised when importing a module fails.

IndexError: Occurs when the index of a sequence is out of range. **Handling Exceptions with try and except:** You can catch exceptions using the try and except blocks. Here's the basic syntax:

[11]:
```
try:
    # Code that may cause an exception
    result = 1 / 0  # Example: ZeroDivisionError
except ZeroDivisionError:
    print("Oops! Division by zero occurred.")
```

```
Oops! Division by zero occurred.
```

Custom Exceptions: You can also create your own custom exceptions by defining a new class that inherits from Exception or its subclasses.

0.0.10 EXCEPTIONAL HANDLING :

The else Clause You can use the else clause along with the try-except block to specify code that should run only if no exceptions are raised.

Example:

try: num = int(input("Enter a number:")) result = 10 / num except ZeroDivisionError: print("Cannot divide by zero!") except ValueError: print("Invalid input. Please enter a valid number.") else: print("Result:", result) In this example, if no exception occurs, the else block is executed and the result is printed.

4.4. The finally Clause The finally block is used to specify code that should run regardless of whether an exception was raised or not. It's often used for cleanup tasks, such as closing files or releasing resources.

Example:

try: file = open("data.txt", "r") content = file.read() print("File content:", content) except FileNotFoundError: print("File not found.") finally: file.close() print("File closed.") In this example, the finally block ensures that the file is closed, whether an exception occurs or not.

4.5. Raising Exceptions You can raise exceptions manually using the raise statement. This can be useful when you want to indicate an error condition based on certain conditions in your code.

Example:

def validate_age(age): if age < 0: raise ValueError("Age cannot be negative") return age

try: user_age = int(input("Enter your age:")) validated_age = validate_age(user_age) print("Your age:", validated_age) except ValueError as ve: print("Error:", ve) In this example, the validate_age function raises

a ValueError if the age is negative. The try block captures the exception and displays the error message.

5. Examples of Exception Handling 5.1. Division by Zero try: numerator = int(input("Enter the numerator:")) denominator = int(input("Enter the denominator:")) result = numerator / denominator print("Result:", result) except ZeroDivisionError: print("Cannot divide by zero!") except ValueError: print("Invalid input. Please enter valid integers.")