Module 5: File Operations and Error Handling

Introduction

File operations and error handling are essential aspects of programming in Python. In this module, we will explore how to perform various operations on files, such as reading, writing, and manipulating file contents. Additionally, we will cover error handling techniques to gracefully handle exceptions that may occur during program execution.

File Handling

File handling involves opening, reading, writing, and closing files. Python provides built-in functions and methods to perform these operations.

```
# Example 1: Opening and reading a file
file_path = "example.txt"
file = open(file_path, "r") # Open file in read mode
content = file.read() # Read entire file content
print(content)
file.close() # Close the file
```

File Modes

Understanding different file modes is crucial for choosing the appropriate mode based on the intended operation. Common file modes include read mode (`'r'`), write mode (`'w'`), and append mode (`'a'`).

```
# Example 2: Writing to a file
file_path = "output.txt"
with open(file_path, "w") as file: # Open file in write mode
 file.write("Hello, world!\n") # Write data to the file
```

File Navigation

File navigation involves moving the file pointer and understanding its current position within the file. Methods like `seek()` and `tell()` are used for file navigation.

```
# Example 3: Moving the file pointer
file_path = "example.txt"
with open(file_path, "r") as file:
 file.seek(5) # Move file pointer to position 5
 content = file.read() # Read content from current position
 print(content)
 print("Current Position:", file.tell()) # Get current file position
```

Error Handling

Error handling allows us to gracefully handle exceptions that may occur during program execution. Using try-except blocks, we can catch and handle exceptions effectively.

```
# Example 4: Error handling with try-except blocks
try:
 num = int(input("Enter a number: "))
 result = 10 / num
 print("Result:", result)
except ZeroDivisionError:
 print("Error: Cannot divide by zero.")
except ValueError:
 print("Error: Invalid input. Please enter a valid number.")
```

Handling Specific Exceptions

We can handle specific types of exceptions using except clauses. This allows us to catch and handle different types of errors separately.

```
# Example 5: Handling specific exceptions
try:
 num = int(input("Enter a number: "))
 result = 10 / num
 print("Result:", result)
except ZeroDivisionError:
 print("Error: Cannot divide by zero.")
except ValueError:
 print("Error: Invalid input. Please enter a valid number.")
except Exception as e:
 print("Error:", e) # Catching any other exception and printing the error message
```

Conclusion

By mastering file operations and error handling techniques in Python, you'll be equipped to work with files effectively and handle potential errors gracefully. These skills are essential for building robust and reliable software applications that can handle various scenarios encountered during execution. Let's dive into the world of file operations and error handling to enhance your Python programming skills.

Exercises & Answers for File Operations and Error Handling

Exercise 1: Reading and Printing File Contents

Write a Python program that reads the contents of a text file named "data.txt" and prints them to the console.

Exercise 2: Writing to a File

Create a Python program that prompts the user to enter a sentence and then writes that sentence to a file named "output.txt".

Exercise 3: Error Handling - Division

Write a Python program that asks the user to enter two numbers and then divides the first number by the second number. Handle the ZeroDivisionError exception by printing a message indicating that division by zero is not allowed.

Exercise 4: Error Handling - File Not Found

Write a Python program that attempts to open a file named "missing.txt" for reading. Handle the FileNotFoundError exception by printing a message indicating that the file does not exist.

Exercise 5: File Navigation

Write a Python program that reads the contents of a text file named "numbers.txt" and calculates the sum of all the numbers in the file. Use file navigation techniques to ensure that the file pointer is positioned correctly.

Answers:

Exercise 1:

```
try:
 with open("data.txt", "r") as file:
 content = file.read()
 print(content)
except FileNotFoundError:
 print("Error: File not found.")
```

Exercise 2:

```
sentence = input("Enter a sentence: ")
with open("output.txt", "w") as file:
 file.write(sentence)
```

Exercise 3:

```
try:
 num1 = int(input("Enter the first number: "))
 num2 = int(input("Enter the second number: "))
 result = num1 / num2
 print("Result:", result)
except ZeroDivisionError:
 print("Error: Division by zero is not allowed.")
except ValueError:
 print("Error: Invalid input. Please enter valid numbers.")
```

Exercise 4:

```
try:
 with open("missing.txt", "r") as file:
 content = file.read()
 print(content)
except FileNotFoundError:
 print("Error: File not found.")
```

Exercise 5:

```
try:
 with open("numbers.txt", "r") as file:
 total = 0
 for line in file:
 total += int(line.strip())
 print("Sum of numbers:", total)
except FileNotFoundError:
 print("Error: F
```