Errors, Exceptions and Debugging

Exercises

Level - Easy

Exercise 1-1

1. Write a program that asks the user for two numbers and divides the first number by the second number.
2. Use a try-except block to catch a ZeroDivisionError and print an error message if the user tries to divide by zero.

Exercise 1-2

1. Create a program that asks the user to enter an integer.
2. Use a try-except block to catch a ValueError and keep asking the user for an integer until they enter a valid one.

Exercise 1-3

1. Write a program with the following dictionary {'a': 1, 'b': 2, 'c': 3} that asks the user to enter a key and display the value associated with that key.
2. Use a try-except block to catch a KeyError if the user enters a key that isn't in the dictionary.

Exercise 1-4

1. Given the following list [1, 2, 3, 4, 5] write a program that asks the user for an index and prints the element at that index in the list.
2. Use a try-except block to handle an IndexError for invalid indices and a TypeError if the user does not enter an integer.

Exercise 1-5

1. Write a program that asks the user for the name of a file to open.
2. Use a try-except block to handle a situation where the file does not exist (FileNotFoundError). In the except block, inform the user that the file was not found.

Exercise 1-6

1. Write a program that takes an age as input and asserts that the age is a positive integer.
2. Use a try-except block to catch an AssertionError and print an appropriate message if the assertion fails.

Level - Moderate

Exercise 2-1

1. Create a calculator that can perform operations like addition, subtraction, multiplication, division, and modulus.
2. Raise a ValueError if the user enters an invalid operation.
3. Use a try-except-else block, where the else block executes if no exceptions are raised and prints the result of the calculation.
4. Include a finally block that prints a message, regardless of whether an exception was caught or not.

Exercise 2-2

1. Write a program that reads numbers from a file and calculates their sum.
2. The program should handle FileNotFoundError if the file doesn't exist, and ValueError if the file contains non-numeric values.
3. Use try-except blocks for each exception type and a finally block to close the file if it was successfully opened.

Exercise 2-3

1. Write a program with nested try-except blocks, such as reading a number from a file and then performing a calculation with it.
2. The inner block should handle file reading errors, while the outer block handles calculation errors (like ZeroDivisionError).
3. Use raise within the inner except block to propagate the error if necessary.