Exception Handling

September 16, 2025

Exception Handling: Robust and Graceful Error Management

Exception handling is a critical part of writing stable, user-friendly programs. An **exception** is a special type of error that occurs during the execution of your program, but it can be "handled" or "caught" to prevent the program from crashing.

- **The Problem:** Without exception handling, a simple error like a user typing a letter instead of a number will cause your program to stop with a ValueError.
- # This will crash if the user enters a non-integer
- user_age = int(input("Enter your age: "))

- **The Solution: The try...except Block:** The try block contains the code that might cause an error. The except block contains the code that will run if an error occurs. This allows you to gracefully handle the error and give the user a helpful message.

Basic try...except:

```
try:
    result = 10 / 0  # This will cause a ZeroDivisionError
except:
    print("An error occurred.")
```

Output:

```
An error occurred.
```

Handling Specific Exceptions:

```
try:
    num = int(input("Enter a number: "))
except ValueError:
    print("Invalid input. Please enter a valid number.")
```

Output:

Case 1: Valid input
Input:
Enter a number: 25
Output:
(no output, because no error occurred)
num will now hold the value 25.

Case 2: Invalid input (e.g., abc)
Input:
Enter a number: abc
Output:
Invalid input. Please enter a valid number.

The try...except...else Block: The else block runs only if the try block completes successfully without any exceptions.

```
try:
    num = int(input("Enter a number: "))
    print("The number is:", num)
except ValueError:
    print("Invalid input.")
else:
    print("The try block executed successfully!")
```

Output:

If you enter a valid integer:
The number is: 10
The try block executed successfully!

If you enter an invalid value:
Enter a number: abc
Output:
Invalid input.

The try...except...finally Block: The finally block **always** runs, regardless of whether an exception occurred. It's perfect for cleanup tasks, such as closing files or connections.

```
file = None
try:
    file = open("my_file.txt", "r")
    content = file.read()
except FileNotFoundError:
    print("The file was not found.")
finally:
    if file:
        file.close()
        print("File has been closed.")
```

Output:

Case 1: File does not exist
If "my_file.txt" is not present in the same folder:
Output:
The file was not found.
(finally runs, but since file never opened successfully, file is None, so it won't try to close it.)

Case 2: File exists and has some text
Suppose my_file.txt contains:
Hello, Python!
Output:
File has been closed.
And the variable content will store:
Hello, Python!

Comprehensive Example:

```
while True:
    try:
        num1_str = input("Enter a number: ")
        num2_str = input("Enter a second number: ")

        num1 = int(num1_str) # Could raise a ValueError
        num2 = int(num2_str) # Could raise a ValueError

        result = num1 / num2 # Could raise a ZeroDivisionError

    except ValueError:
        print("Invalid input. Please enter a valid number.")

    except ZeroDivisionError:
        print("You cannot divide by zero. Please try again.")

    except Exception as e:
        # This is a generic handler for any other unexpected error
        print(f"An unexpected error occurred: {e}")

    else:
        # The 'else' block runs ONLY if the 'try' block was successful.
        print(f"The result is {result}.")
        break  # Exit the loop on success

    finally:
        # The 'finally' block always runs, no matter what.
        # It's perfect for cleanup, like closing files or connections.
        print("Operation attempt complete.")
```

Output:

Case 1: Invalid number input
Enter a number: abc
Enter a second number: 10
Output:
Invalid input. Please enter a valid number.
Operation attempt complete.

Case 2: Division by zero
Enter a number: 10
Enter a second number: 0
Output:
You cannot divide by zero. Please try again.
Operation attempt complete.

Case 3: Successful division
Enter a number: 20
Enter a second number: 5
Output:
The result is 4.0.
Operation attempt complete.
(The loop breaks here because division was successful.)