Day 10 of 6 weeks Python course:

🕒 Time Breakdown (2 Hours)

Time Slot	Task
0:00 - 0:20	Introduction to Exception Handling
0:20 - 0:40	Handling Specific Exceptions
0:40 - 1:00	The `finally` Block & Raising Exceptions
1:00 - 1:20	Best Practices for Exception Handling
1:20 - 2:00	Mini-Project: Secure Calculator

📌 Step-by-Step Learning Guide 1️⃣ What is Exception Handling? ✅ Definition: Exceptions are runtime errors that can cause a program to crash. ✅ Why Use Exception Handling? 1.Prevents program crashes. 2.Provides user-friendly error messages. ✅ Basic Try-Except Example

In [4]:
```
try:
    num = int(input("Enter a number: "))  # User enters non-integer → Error
    print("You entered:", num)
except:
    print("An error occurred!")
```

```
You entered: 5
```

2️⃣ Handling Specific Exceptions ✅ Handling Different Types of Errors

In [8]:
```
try:
    num = int(input("Enter a number: "))  # User enters a non-integer
    result = 10 / num  # User enters 0 → ZeroDivisionError
except ValueError:
    print("Invalid input! Please enter a number.")
except ZeroDivisionError:
    print("Cannot divide by zero!")
```

3️⃣ Using finally and raise ✅ finally Block (Always Executes)

In [10]:
```
try:
    file = open("sample.txt", "r")
    print(file.read())
except FileNotFoundError:
    print("File not found!")
finally:
    print("This will always run, closing resources if needed.")
```

```
File not found!
This will always run, closing resources if needed.
```

💡 Use finally to release resources (files, databases, etc.). ✅ Raising Custom Exceptions (raise)

In [12]:
```
age = int(input("Enter your age: "))
if age < 0:
    raise ValueError("Age cannot be negative!")
```

💡 Stops execution if a condition is met. 4️⃣ Best Practices for Exception Handling (20 mins) ✅ Best Practices: 1.Use specific exceptions (ValueError, ZeroDivisionError) instead of except:. 2.Avoid suppressing errors (e.g., using except: pass). 3.Use finally for cleanup tasks. Log errors instead of printing messages in production systems. ✅ Example: Logging Errors

In [14]:
```
try:
    num = int(input("Enter a number: "))
except ValueError as e:
    with open("error_log.txt", "a") as log_file:
        log_file.write(f"Error: {e}\n")
    print("Invalid input! Error logged.")
```

🎯 Mini-Project: Secure Calculator 📌 Project Goal 1.Perform basic arithmetic operations. 2.Handle errors gracefully (division by zero, invalid inputs). 3.Ensure a smooth user experience. 🖥️ Code Implementation:

In [17]:
```
def calculator():
    try:
        num1 = float(input("Enter first number: "))
        operator = input("Enter operation (+, -, *, /): ")
        num2 = float(input("Enter second number: "))

        if operator == "+":
            result = num1 + num2
        elif operator == "-":
            result = num1 - num2
        elif operator == "*":
            result = num1 * num2
        elif operator == "/":
            if num2 == 0:
                raise ZeroDivisionError("Cannot divide by zero!")
            result = num1 / num2
        else:
            raise ValueError("Invalid operation!")

        print(f"Result: {result}")

    except ValueError as e:
        print("Invalid input:", e)
    except ZeroDivisionError as e:
        print("Error:", e)
    finally:
        print("Calculation complete.\n")

# Main Program
while True:
    print("\n🧮 Secure Calculator 🧮")
    print("1. Perform Calculation")
    print("2. Exit")

    choice = input("Enter your choice (1-2): ")

    if choice == "1":
        calculator()
    elif choice == "2":
        print("Exiting Calculator. Goodbye!")
        break
    else:
        print("Invalid choice! Please enter 1 or 2.\n")
```

```
🧮 Secure Calculator 🧮
1. Perform Calculation
2. Exit
Result: 3.0
Calculation complete.


🧮 Secure Calculator 🧮
1. Perform Calculation
2. Exit
Exiting Calculator. Goodbye!
```

🔍 Step-by-Step Explanation

📌 Step 1: Taking User Input

python Copy Edit

```python
num1 = float(input("Enter first number: "))
operator = input("Enter operation (+, -, *, /): ")
num2 = float(input("Enter second number: "))
```

- Converts input to **float** for calculations.

📌 Step 2: Performing Arithmetic Operations

python Copy Edit

```python
if operator == "+":
    result = num1 + num2
elif operator == "-":
    result = num1 - num2
elif operator == "*":
    result = num1 * num2
elif operator == "/":
    if num2 == 0:
        raise ZeroDivisionError("Cannot divide by zero!")
    result = num1 / num2
else:
    raise ValueError("Invalid operation!")
```

- Checks for valid operations.
- Raises errors if input is invalid.

📌 Step 3: Handling Errors Gracefully

python Copy Edit

```python
except ValueError as e:
    print("Invalid input:", e)
except ZeroDivisionError as e:
    print("Error:", e)
finally:
    print("Calculation complete.\n")
```

- Catches **specific errors** and provides **user-friendly messages.**

✅ Example Run:

pgsql Copy Edit

```pgsql
🧮 Secure Calculator 🧮
1. Perform Calculation
2. Exit

Enter your choice (1-2): 1
Enter first number: 10
Enter operation (+, -, *, /): /
Enter second number: 0
Error: Cannot divide by zero!
Calculation complete.
```

📌 Summary of Day 10 ✔ Learned Exception Handling (try, except, finally) ✔ Practiced Error Handling in a Secure Calculator ✔ Completed a Mini-Project: Secure Calculator