Exception Handling

Purpose: Handle runtime errors gracefully without crashing the program

Core Concept: Use `try`, `except`, `else`, `finally` blocks

🧠 What Is an Exception?

- **Error** = Something wrong in code
- **Exception** = Error that happens **while the program is running**

📌 *Examples of Exceptions:*

- Dividing by zero (`ZeroDivisionError`)
- Using a variable that doesn't exist (`NameError`)
- Trying to open a file that doesn't exist (`FileNotFoundError`)

🧱 Basic Structure

```
try:
    # risky code here
except ErrorType:
    # what to do if that error happens
else:
    # if no error happened, do this
finally:
    # always do this (whether error happened or not)
```

```
try:
    # Risky file operation
    with open("file.txt", "r") as file:
        data = file.read()
except FileNotFoundError:
    print("Error: File not found!")
except PermissionError:
    print("Error: No read permissions!")
except Exception as e:  # Catch-all for other errors
    print(f"Unexpected error: {e}")
else:
    print("File read successfully!")
finally:
    print("This runs ALWAYS, success or failure.")
```

🔍 Example 1: Basic Try-Except

```
try:
    result = 10 / 0
except ZeroDivisionError:
    print("You can't divide by zero!")
```

✔️ Prevents crash

✔️ Shows a helpful message instead

🔍 Example 2: Try-Except-Else

```
try:
    num = int(input("Enter a number: "))
except ValueError:
    print("That's not a number!")
else:
    print("Good! You entered:", num)
```

✔️

`else` runs **only if no exception**

Practical Example: Safe File Reading

```
def read_file_safely(file_path):
    try:
        with open(file_path, "r", encoding="utf-8") as file:
            return file.read()
    except FileNotFoundError:
        return "Error: File does not exist."
    except PermissionError:
        return "Error: Access denied."
    except UnicodeDecodeError:
        return "Error: File is not readable as text."
    except Exception as e:
        return f"Unexpected error: {e}"

# Usage
content = read_file_safely("data.txt")
print(content)
```

🔍 Multiple Exceptions

```
try:
    # some code
except ValueError:
    print("Wrong value!")
except ZeroDivisionError:
    print("Can't divide by zero!")
```

✔️ Handle specific errors differently

🔍 Catch All Exceptions (not best practice, but useful)

```
try:
    something_risky()
except Exception as e:
    print("Error happened:", e)
```

✔️Exception catches **all errors**

✔️ `e` holds the error message

When does except `Exception` as ex: get executed?

The except
`Exception as e`: block acts as a general catch-all for any exception that occurs within the try block that is **not specifically caught by the preceding except blocks**.

- If a `ValueError` occurs (e.g., the user types "hello" instead of a number), the `except ValueError:` block is executed.
- If a `ZeroDivisionError` occurs (e.g., the user enters 0), the `except ZeroDivisionError:` block is executed.
- **If *any other type of exception* occurs** in the `try` block (and it's not a `ValueError` or `ZeroDivisionError`), then the `except Exception as ex:` block is executed. The specific error object is assigned to the variable `ex`, which is then printed.

⚠️ When to Use:

Situation	Use Exception Handling?
User input (text vs number)	✅ Yes
File operations	✅ Yes
Network/database operations	✅ Yes
Code you trust fully	❌ Maybe not needed

Key Exception Types for Files

Exception	When It Occurs	Solution
`FileNotFoundError`	File doesn't exist	Check path or prompt user
`PermissionError`	No read/write permissions	Request admin rights or notify user
`IsADirectoryError`	Path is a folder, not a file	Verify path is a file
`UnicodeDecodeError`	File isn't text (e.g., binary)	Use binary mode (`'rb'`) or try different encoding
`IOError`	General disk/network issues	Retry or log the error