1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 1 1 0 1

</ Table of contents

1011 011 01 1011001 10 11011 011 01 110110 110111 1101

Errors and Exceptions in Python

- **What are Errors in Python?**
  Errors are problems in your code that stop it from working correctly.
  - **Syntax Errors.**
  - **Runtime Errors.**
    - ZeroDivision Error.
    - Name Error.
    - Type Error.
    - Index Error. [List]
    - Key Error. {dictionary}
- Why to Handle Errors?
  - The program would crash.
  - Users see scary error messages.
  - Important code might not run.
  - Files might not closed properly.
- How to Handle Errors?
  - We use `try` and `except` blocks to handle errors gracefully.
  - Basic Structure:

```
try:
    ######
except:
    ######
```

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

Errors and Exceptions in Python

- Cont. How to Handle Errors?
  - The Else and Finally:

```
else:
    #######
finally:
    #######
```

  - Raising Exceptions:

```
if age < 0:
    raise ValueError("Age cannot be negative!")
```

  - User Input Validation

- Best Practices:
  1. Be specific about which errors you catch.
  2. Don't use bare except.
  3. Clean up resources in finally.
  4. Give helpful error messages.

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

Try-Except-Else-Finally

- Basic Structure:
  ```
  try:
  except ErrorType:
  else:
  finally:
  ```
- How Each Block Works
  - Try:
    - Contains the code that might raise an error.
    - If an error occurs, Python jumps to except.
    - If no error occurs, Python runs the else block.
  - Except:
    - Catches and handles specific errors.
    - You can have multiple except blocks for different errors.
  - Else:
    - Runs only if no error happens in try.
    - Useful for code that should run only when everything succeeds.
  - Finally:
    - Runs no matter what.
    - Used for cleanup tasks.

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

- When to Use else vs finally:
  - **else:**
    Only if try succeeds.
    Running code that depends on try working.
  - **finally:**
    Always runs.
    Cleanup tasks.

- Best Practices:
  - Be specific in except
  - Use else for code that should run only on success.
  - Use finally for cleanup.
  - Avoid empty except blocks.

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

File Handling in Python

- How Files Work in Python?
  - Files are used to: Save data, Read data, and Log events.
- Opening a File:
  - using open() and close()

```
file = open("example.txt", "r")  # Open in read mode
content = file.read()  # Read the file
file.close()  # Always close the file!
```

    - If you forget close(), the file stays open
  - using with():

```
with open("example.txt", "r") as file:
    content = file.read() # File closes automatically here!
```

    - Always use with (no need to remember close()).
- File Modes.
- Handling File Errors.
- Real-World Examples.

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

Using OOP for File Logs

- Why Use OOP for Files?
  - **Encapsulate** file operations inside a class.
  - **Reuse** the same code across projects.
  - **Add error handling** in one place.
- Basic File Logger Class: class that writes messages to a file.
- Adding Error Handling.
- Example: Game Progress Saver.
- Key Benefits of OOP File Handling.

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

Design a Simple Text-Based Game (OOP)

1011 011 01 1011001 10 11011 011 01 110110 110111 1101

Lab

Create a GameManager class that:

- Takes a player's name
- Launches a GuessGame (from previous exercise)
- Logs all game results
- Handles errors (file issues, bad input, etc.)

Steps:

1. Reuse Existing Classes
2. Create the GameManager Class
3. Main Logic

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1