Error Handling

What is an error / exception?

- Error = something went wrong at runtime (IO failure, invalid input, network down, etc.)..
- Exception = a language mechanism to represent that error and unwind the call stack.
- Exception types include SyntaxError, Runtime exceptions, Custom exceptions.

Why errors happens?

- Users enter data in the wrong format.
- Files or database records may not exist.
- Internet connection might be down.
- APIs may return unexpected responses.
- Hardware fails.
- Logic mistakes happen in code.

Why handle errors?

- Provide meaningful feedback to users.
- Clean up resources (files, DB connections, processes).
- Allow graceful degradation and retries.
- Avoid security leaks and inconsistent state.

The Try/Except Model

```
try:
    # risky code
except <ErrorType>:
    # how we recover
```

Example: Handling Bad Input

```
try:
    age = int(input("Enter age: "))
    print("Your age is", age)
except ValueError:
    print("Invalid number! Please enter digits only.")
```

Else & Finally

```
try:
    f = open('data.txt')
    content = f.read()
except FileNotFoundError:
    print("File missing!")
else:
    print("Read success!")
finally:
    print("Closing resources...")
```

Raising Your Own Errors

```
def set_price(price):
    if price < 0:
        raise ValueError("Price cannot be negative")
```

Custom Exceptions

- Custom exceptions make your error messages clear and specific.

```
class InvalidAgeError(Exception):
    pass


def check_age(age):
    if age < 0:
        raise InvalidAgeError("Age must be positive.")
```

Retry Pattern (Network/Database)

Networks fail frequently. Retrying is a standard engineering technique.
We try up to 3 times because:

- It increases reliability
- It reduces user frustration
- It avoids immediate failure due to temporary issues

```
import random, time

for attempt in range(3):
    try:
        if random.random() < 0.6:
            raise ConnectionError("Network issue.")
        print("Success.")
        break
    except ConnectionError:
        print("Retrying...")
        time.sleep(1)
```

Best Practices

- **Catch** specific exceptions (never use plain except:)
- Keep try-blocks small
- Raise exceptions for invalid data
- Always clean up resources using **finally**
- Use custom exceptions for clarity

Any questions?