Python Programming
Unit 03 – Lecture 02 Notes
Errors vs Exceptions and Exception Handling

Tofik Ali

February 14, 2026

Contents

1 Lecture Overview

When a program meets unexpected input or a missing resource (file, database, network), it should not crash with a confusing traceback for the user. Instead, it should:

- detect the problem,
- display a clear message, and
- recover or exit gracefully.

This lecture introduces exceptions and shows how to handle them using `try/except/else/finally`.

2 Core Concepts

2.1 Syntax Errors vs Exceptions

Syntax errors happen when Python cannot understand your code. Example: missing colon after `if`.

Exceptions are runtime problems. Example: dividing by zero, converting `"abc"` to `int`, missing file, etc.

2.2 What is an Exception?

An exception is an object that represents an error condition. When raised, it interrupts normal flow and searches for a matching `except`. If none is found, the program terminates and prints a traceback.

2.3 Basic `try/except`

```
try:
    n = int(input("Enter n: "))
    print(10 / n)
except ValueError:
    print("Please enter an integer.")
except ZeroDivisionError:
    print("Division by zero is not allowed.")
```

Why catch specific exceptions? Because different problems need different messages and different recovery steps.

2.4 `else` and `finally`

`else` runs only if the `try` block succeeded (no exception). `finally` runs always (cleanup).

```
f = None
try:
    f = open("data.txt", "r", encoding="utf-8")
    data = f.read()
except FileNotFoundError:
    print("File not found.")
else:
    print("Length:", len(data))
finally:
    if f is not None:
        f.close()
```

Note: In real code, prefer `with open(...)` so you avoid manual close.

2.5 Handling Multiple Exceptions

Sometimes multiple error types can happen in the same risky code. Python allows catching a tuple of exceptions:

```
try:
    x = int(input("x: "))
    y = int(input("y: "))
    print(x // y)
except (ValueError, ZeroDivisionError) as e:
    print("Error:", e)
```

2.6 raise: Creating Your Own Errors

You can raise an exception to signal invalid state:

```
age = int(input("Age: "))
if age < 0:
    raise ValueError("Age must be non-negative")
```

This is useful when you want to enforce a rule and stop execution for invalid inputs.

2.7 assert: Debugging Assumptions

`assert condition` is mainly for debugging and internal checks:

```
assert 2 + 2 == 4
```

Warning: assertions can be disabled when Python runs with optimization flags (`-O`). Do not use `assert` as a replacement for user input validation.

3 Demo Walkthrough

File: `demo/exception_handling_demo.py`

What the demo teaches

- Create a safe input function that keeps asking until the user enters a valid integer.
- Use `try/except` for division and file reading.
- Print user-friendly error messages.

4 Interactive Checkpoints (with Solutions)

Checkpoint 1 Solution

Question: When does `else` run in a `try` statement?
Answer: `else` runs only if **no exception** occurs in the `try` block.

Checkpoint 2 Solution

Question: Why prefer catching `ValueError` instead of catching `Exception`?
Answer:

- Catching `Exception` can hide bugs you did not expect (programming mistakes).
- Catching `ValueError` is more precise and documents what you are handling.

5 Practice Exercises (with Solutions)

Exercise 1: Safe Integer Input Function

Task: Write `read_int(prompt)` that keeps asking until user enters a valid integer.
Solution:

```python
def read_int(prompt: str) -> int:
    while True:
        try:
            return int(input(prompt).strip())
        except ValueError:
            print("Invalid integer. Try again.")
```

Exercise 2: Safe Division

Task: Read two integers and print `a/b`. Handle division by zero.
Solution:

```python
try:
    a = int(input("a: "))
    b = int(input("b: "))
    print("a / b =", a / b)
except ZeroDivisionError:
    print("Division by zero is not allowed.")
except ValueError:
    print("Invalid integer input.")
```

Exercise 3: Safe File Read

Task: Read a filename and print its content. If file does not exist, print a message.
Solution:

```python
path = input("File path: ").strip()
try:
    with open(path, "r", encoding="utf-8") as f:
        print(f.read())
except FileNotFoundError:
    print("File not found.")
```

6 Exit Question (with Solution)

Question: Read an integer and print it. If invalid, print `"Invalid integer"`.
Solution:

```python
try:
    n = int(input("Enter n: ").strip())
    print("n =", n)
except ValueError:
    print("Invalid integer")
```