Lesson 10: Error Handling with Try and Except in Python

Marcin Kurzyna

Lecture Goals

This lecture introduces:

- What errors and exceptions are in Python.
- Why error handling is important.
- The `try / except` mechanism.
- Catching specific exceptions.
- Using `else` and `finally`.
- Writing safe and user-friendly programs.

1 What Are Errors and Exceptions?

An **error** occurs when Python cannot execute a program correctly.

An **exception** is a runtime error that can be handled.

Example of an Unhandled Exception

```
number = int(input("Enter a number: "))
print(10 / number)
```

If the user enters `0` or a non-numeric value, the program crashes.

2 Why Error Handling Is Important

Without error handling:

- Programs crash unexpectedly.
- Users see confusing error messages.
- Data can be lost.

Goal: Prevent crashes and handle problems gracefully.

3 Basic Try and Except

The `try` block contains code that might fail. The `except` block handles the error.

Basic Syntax

```
try:
    # risky code
except:
    # error handling code
```

Example

```
try:
    number = int(input("Enter a number: "))
    print(10 / number)
except:
    print("Something went wrong!")
```

4 Catching Specific Exceptions

Catching all exceptions is not recommended.

Common Exceptions

- `ValueError`
- `ZeroDivisionError`
- `TypeError`

Example with Specific Exceptions

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

5 Using Else

The `else` block runs only if no exception occurred.

```
try:
    number = int(input("Enter a number: "))
    result = 10 / number
except ZeroDivisionError:
    print("Cannot divide by zero.")
except ValueError:
    print("Invalid input.")
else:
    print("Result:", result)
```

6 Using Finally

The `finally` block always executes.

Typical Use Cases

- Closing files
- Releasing resources
- Cleanup operations

Example

```
try:
    file = open("data.txt", "r")
    content = file.read()
    print(content)
except FileNotFoundError:
    print("File not found.")
finally:
    file.close()
    print("File closed.")
```

7 Raising Exceptions

You can raise your own exceptions using `raise`.

```
def withdraw(balance, amount):
    if amount > balance:
        raise ValueError("Insufficient funds")
    return balance - amount

try:
    withdraw(100, 200)
except ValueError as error:
    print(error)
```

8 Best Practices

- Catch only the exceptions you expect.
- Do not hide errors unnecessarily.
- Use meaningful error messages.
- Avoid empty `except` blocks.

Summary

In this lecture, we covered:

- What exceptions are.
- How `try / except` works.
- Handling specific exceptions.
- Using `else` and `finally`.
- Writing safer Python programs.

9 Exercises

1. Write a program that handles division by zero.
2. Handle invalid user input when converting strings to integers.
3. Modify a file-reading program to handle missing files.
4. Use `else` to display results only when no error occurs.
5. Create a function that raises a custom exception.
6. Challenge: Write a simple ATM simulation using `try / except`.