



ACCA4020

Python – Error Handling and Optimisation

Dr. Seena Joseph

Errors in Programming

- Errors - mistakes in a program that prevent it from running correctly or producing the expected output.
- Three main types:
 - Syntax Errors
 - Runtime Errors
 - Logical Errors

Errors in Programming

- Syntax Errors

- These occur when the code violates the rules of the programming language.

- a syntax error means that a program is not valid Python code.
 - It doesn't abide by the official rules of the Python language.

- Common syntax errors

- Misplaced, missing, or mismatched punctuation (parentheses, brackets, braces, quotes, commas, colons, etc.);
 - Misspelled, misplaced, or missing Python keywords;
 - Illegal characters in variable names;
 - Incorrect indentation;
 - Incorrect use of the assignment operator (=).

- When we run a program with a syntax error, the computer immediately reports an error in the console. It doesn't even try to execute the first line of the program.

Errors in Programming

- Syntax Error

```
print('Good moring)
```

```
print('Good moring)
      ^
SyntaxError: unterminated string literal (detected at line 1)
```



Errors in Programming

- Logical Errors

- These occur when the code runs without crashing, but the output is incorrect due to faulty logic.
- They are harder to detect since the program doesn't produce an error message.

```
age = 15
if age > 10: # Meant to check if age < 10
    print("You're too old!")
else:
    print("You're young!")
```



Errors in Programming

- Logical Errors

```
# Calculate the sum of numbers from 1 to 5
sum = 0
for i in range(1, 5): # Logical Error: Range should be (1, 6)
    sum += i
print("Sum:", sum)
```

```
# Print even numbers from 1 to 10
for i in range(1, 11):
    💡 if i % 2 == 1: # Logical Error: ???
        print(i)
```

Logical error ?



Errors in Programming

- Runtime Errors

- Errors that occur while the program runs, often due to invalid operations or input.
- Cause: Division by zero, invalid type conversion
- Example

```
a=7  
b=0  
c=a/b  
print(c)
```

```
c=a/b  
~^~  
ZeroDivisionError: division by zero
```

```
num = int(input("Enter a number: ")) # User enters "abc"  
print(num)
```

```
num = int(input("Enter a number: ")) # User enters "abc"  
ValueError: invalid literal for int() with base 10: 'abc'
```

Validation

- **Validation**, in the context of programming, refers to the process of checking whether data (such as user input) meets specific requirements or rules before it is used in a program. The goal is to ensure that the data is correct, appropriate, and safe for processing, thereby preventing errors, crashes, or unexpected behavior.

Validation

- **Key Aspects of Validation**

- **Correctness**: Ensures the data matches the expected format or type (e.g., a number instead of text).
- **Range**: Confirms the data falls within acceptable limits (e.g., an age between 0 and 120).
- **Completeness**: Verifies that required data is provided (e.g., a name isn't empty).
- **Safety**: Protects the program from invalid or harmful input (e.g., avoiding crashes from converting "abc" to an integer).

Validation

- Example

```
# Validating a positive number
num = input("Enter a positive number: ")
if num.isdigit() and int(num) > 0:
    print(f"Valid input: {num}")
else:
    print("Invalid: Must be a positive number!")
```

Here, validation checks if the input is a digit and greater than 0 before proceeding.

Why It Matters

- Prevents **runtime errors** (e.g., crashes from bad data).
- Reduces **logical errors** (e.g., accepting unrealistic values like a 200-year-old age).
- Improves reliability and user experience by catching mistakes early.

In short, validation is like a gatekeeper—it ensures only good data gets through, keeping your program running smoothly.

Validation

- Example

```
age = int(input("Enter your age: "))  
print(f"Your age is {age}")
```

Enter your age: *dd*

Without validation

```
age = int(input("Enter your age: "))  
ValueError: invalid literal for int() with base 10: 'dd'
```

With validation

```
age = input("Enter your age: ")  
if age.isdigit():  
    print(f"Your age is {age}")  
else:  
    print("Error: Must be a number!")
```

Validation

- Example

```
while True:
    try:
        age = int(input("Enter your age (1-120): "))
        if 1 <= age <= 120:
            print("Valid age!")
            break
        else:
            print("Age must be between 1 and 120.")
    except ValueError:
        print("Invalid input. Please enter a number.")
```

With validation

Validation

- Example

```
grade = input("Enter a grade (0-100): ")
if grade.isdigit() and 0 <= int(grade) <= 100:
    print(f"Valid grade: {grade}")
else:
    print("Error: Grade must be 0-100!")
```

Range validation

What is the purpose of this code ?

Validation

- Example

String validation

```
name = input("Enter your name: ")
if name.isalpha():
    print(f"Hello, {name}!")
else:
    print("Error: Name must be letters only and not empty!")
```

What is the purpose of this code ?

Exercise 1

1. Write a Python program that asks the user to create a username. The username must meet the following conditions:
 - It should contain only letters and numbers (no spaces or special characters).
 - It must be at least 5 characters long and no more than 12 characters.
 - If the username is valid, print "Username accepted!"
 - If it is invalid, print an appropriate error message and ask the user to enter a new username.
 - Hint: Use the `.isalnum()` method for validation.

Exercise 2

2. Write a Python program to check if a user's password is valid.

The password must:

Be at least 8 characters long.

Contain at least one alphabet and one number.

Not contain any spaces.

If valid, print "Password accepted!"

If invalid, print the appropriate error message.

Hint: Use `.isalnum()` and `any()` with string methods

Exercise 3

3. Write a Python program that asks for:

A username that should be alphanumeric and between 4 to 12 characters.

An email address that must contain "@" and "." at the correct positions.

If both inputs are valid, print "Registration successful!"

If not, print appropriate error messages.

Hint: Use `.isalnum()` for username validation and string methods like `.count()` and `.find()` for email validation.

Exercise 3

Write a Python program named `generate_even_numbers` that generates all even numbers within a specified range (inclusive).

After writing the program, evaluate how it handles the following test cases by describing the results and explaining how the program processes each case.

Analyse the expected output for each test case.

Explain how the program might handle unusual scenarios like reversed ranges, negative numbers, or identical values.

Provide your observations on how well the program manages edge cases.

Test Cases:

Test Case 1: start = 10 and end = 20

Test Case 2: start = 25 and end = 15

Test Case 3: start = -6 and end = 8

Test Case 4: start = 30 and end = 30

Test Case 5: start = 0 and end = 1

Exercise 4

- Write a Python program with intentional syntax errors.
- Swap programs with your peers to identify the errors in your partner's code.

Optimisation

- Optimisation in programming refers to the process of improving a program's performance, efficiency, or resource usage while maintaining its functionality.
- The goal is to make the code run faster, use less memory, or consume fewer system resources (like CPU or battery power), all without changing what the program does.
- Optimisation in programming is about making your code more efficient—faster, leaner, or easier to manage—without breaking its purpose.

Optimisation

- Example: unoptimised, redundant function call each time

```
total = 0
for i in range(100):
    total += i * len("hello")  # len("hello") recalculated each time
print(total)
```

- Optimised

```
length=len("hello")
total = 0
for i in range(100):
    total += i * length  # len("hello") recalculated each time
print(total)
```

Optimisation

- Example: unoptimised, multiple separate checks

```
num = input("Enter a number: ")
if num.isdigit():
    if int(num) > 0:
        if int(num) <= 100:
            print("Valid!")
```

- Optimised