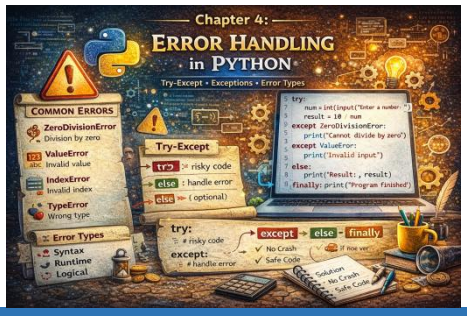
Chapter 4: Error Handling in Python
Gestion des erreurs en Python



1. Introduction to Errors and Exceptions

1.1 What is an Error?

An **error** is a problem that occurs in a program and prevents it from executing correctly.

Errors may occur because of:

- incorrect syntax
- invalid operations
- invalid input from the user
- division by zero
- accessing elements that do not exist

When Python encounters an error, it usually **stops the program and displays an error message**.

Example:

```
x = 10
y = 0
print(x / y)
```

Output

ZeroDivisionError: division by zero

The program stops immediately.

1.2 Types of Errors in Python

Python errors are generally divided into **three categories**.

1. Syntax Errors

These errors occur when the **syntax rules of Python are violated**.

Example

```
if x > 5
   print("Hello")
```

Output

SyntaxError

Reason: Missing : after the condition.

2. Runtime Errors

These errors occur **during program execution**.

Example

```
print(10 / 0)
```

Output

Zero Division Error

3. Logical Errors

Logical errors occur when the program **runs without crashing but produces incorrect results**.

Example

```
a = 10
b = 5
print(a - b)   # supposed to compute addition
```

Output

5

The program runs but the **logic is incorrect**.

2. What is an Exception?

An **exception** is a special type of error that occurs during the execution of a program.

Python provides mechanisms to **handle these exceptions** without stopping the program.

Example of common exceptions:

Exception	Description
ZeroDivisionError	Division by zero
TypeError	Operation between incompatible types
ValueError	Invalid value
IndexError	Invalid index in list
KeyError	Key not found in dictionary
FileNotFoundError	File does not exist

3. The try-except Block

To handle errors in Python, we use the **try-except structure**.

Syntax

```
try:
   # risky code
except:
   # code executed if an error occurs
```

Example 1: Handling division by zero

```
try:
   x = 10
   y = 0
   print(x / y)
except:
   print("Error: Division by zero is not allowed")
```

Output

Error: Division by zero is not allowed

The program **does not stop**.

4. Handling Specific Exceptions

It is better to handle **specific errors**.

Example

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

Explanation:

- ZeroDivisionError → division by zero
- ValueError → invalid number input

5. Using Multiple Exceptions

Example

```
try:
    a = int(input("Enter a number: "))
    print(10 / a)
except ZeroDivisionError:
    print("Division by zero")
except ValueError:
    print("Please enter a valid integer")
```

6. Using a General Exception

Python also allows catching **any type of error**.

Example

```
try:
    x = int("hello")
except Exception as e:
    print("Error occurred:", e)
```

Output

```
Error occurred: invalid literal for int()
```

Here:

e contains the **error message**.

7. The else Block

The else block runs **if no error occurs**.

Syntax

```
try:
    code
except:
    code if error
```

```
else:
    code if no error
```
Example
```
try:
    x = int(input("Enter a number: "))
    print(10 / x)
except ZeroDivisionError:
    print("Division by zero")
else:
    print("Operation successful")
```
8. The finally Block

The finally block **always executes**, whether an error occurs or not.

Example
```
try:
    x = int(input("Enter a number: "))
    print(10 / x)
except:
    print("Error occurred")
finally:
    print("Program finished")
```
Output example
```
Error occurred
Program finished
```
9. Practical Example (Complete Program)
```
try:
    a = int(input("Enter first number: "))
    b = int(input("Enter second number: "))
    result = a / b
except ZeroDivisionError:
    print("Division by zero is not allowed")
except ValueError:
    print("Please enter valid numbers")
else:
    print("Result =", result)
finally:
    print("End of program")
```
10. Advantages of Exception Handling

Using exceptions allows:

- preventing program crashes
- handling user input errors
- writing more robust programs
- improving program reliability

11. Summary

Concept	Description
Error	Problem that stops program execution
Exception	Error that occurs during execution
try	Block containing risky code
except	Handles the error
else	Executes if no error occurs
finally	Always executes

□ Simple Exercise

Task

Write a program that:

1. asks the user for two numbers
2. divides the first by the second
3. handles:
    - division by zero
    - invalid input

Solution

```
try:
    a = int(input("Enter number 1: "))
    b = int(input("Enter number 2: "))
    print(a / b)
except ZeroDivisionError:
    print("Cannot divide by zero")
except ValueError:
    print("Invalid input")
```