**Module 5: Exceptions Handling**

Exception handling in Python is a way to gracefully manage errors or unexpected events during the execution of a program, so the program doesn’t crash abruptly.

**Errors**

Types of Errors in Python

1. **SyntaxError**

Raised when Python code has incorrect syntax.

Example

def my\_func()

print("Hello")

# Call the function

my\_func()

**Fix**: Add a colon : at the end of the function definition line.

1. **ImportError / ModuleNotFoundError**

Raised when Python can't find a module or import fails.

Example

import non\_existing\_module

**Output:**

ModuleNotFoundError: No module named 'non\_existing\_module'

1. **AttributeError**

Raised when a module does not have the attribute you're trying to access.

# B\_module\_B.py

def greet(name):

return f"Hello, {name}"

# C\_main\_C.py

import B\_module\_B

print(B\_module\_B.say\_hello("Amit")) # Error: No function called say\_hello

Note:-Correction put “greet” by replacing “say\_hello” in C\_main\_C.py

1. **NameError**

Raised when a variable or function name is not defined.

print(result) # 'result' is not defined

correct program

a = 10

b = 5

result = a + b

# Now we can print it

print("Result:", result)

1. **TypeError**

Raised when a function is used with wrong data type.

print("5" + 10) # Can't add str and int

1. **ValueError**

Raised when a function receives the right type but wrong value.

int("abc") # Cannot convert string to int

the **value** "abc" cannot be converted into a number.

Raises ValueError: invalid literal for int() with base 10: 'abc'

import math

print(math.sqrt(-9))

-9 is an invalid value — you can’t take the square root of a negative number (in real math).

Raises ValueError: math domain error

1. **ZeroDivisionError**

Raised when you divide a number by zero.

result = 10 / 0

a=10

b=0

result=a/b

print(result)

1. **IndentationError**

An IndentationError in Python occurs when there is a problem with the spaces or tabs used to structure your code.

Common Causes of IndentationError

1. Inconsistent use of tabs and spaces
2. Missing indentation after statements like if, for, while, def, class
3. Unexpected indentation (extra space where it’s not required)

Example 1: Missing Indentation

def greet():

print("Hello") # ❌ This will cause an IndentationError

greet()

Corrected

def greet():

print("Hello")

greet()

**Exception Handling in Python**

Use try, except to handle exceptions gracefully.

**try:**

import math

print(math.sqrt(16))

print(10 / 0) # Will raise ZeroDivisionError

except ZeroDivisionError:

print("You cannot divide by zero.")

correct program

import math

print(math.sqrt(16))

try:

print(10 / 0) # Will raise ZeroDivisionError

except ZeroDivisionError:

print("You cannot divide by zero.")

**handling Multiple Exceptions**

1. Multiple except blocks

Each except handles a specific type of exception.

try:

    num = int(input("Enter a number: "))

    result = 10 / num

    print(result)

except ValueError:

    print("Invalid input! Please enter a number.")

except ZeroDivisionError:

    print("Cannot divide by zero.")

1. Single except with multiple exceptions

Use a tuple to handle multiple exception types in one block.

try:

num = int(input("Enter a number: "))

result = 10 / num

except (ValueError, ZeroDivisionError) as e:

print(f"Error: {e}")

1. Catch all exceptions (not recommended unless needed)

This catches any exception, useful for debugging or logging.

try:

    num = int(input("Enter a number: "))

    result = 10 / num

    print(result)

except Exception as e:

    print(f"An unexpected error occurred: {e}")

Optional: else and finally blocks

try:

num = int(input("Enter a number: "))

result = 10 / num

except (ValueError, ZeroDivisionError) as e:

print("Handled error:", e)

else:

print("Result is:", result)

finally:

print("Execution finished.")

**Writing your own Exception**

In Python, you can create your **own custom exceptions** by defining a new class that inherits from the built-in Exception class or any of its subclasses.

Basic Syntax:

class MyCustomError(Exception):

"""Custom Exception"""

Pass

Example with usage:

class AgeTooSmallError(Exception):

"""Raised when the input age is below 18"""

def \_\_init\_\_(self, age):

self.age = age

self.message = f"Age {age} is too small to vote."

super().\_\_init\_\_(self.message)

def check\_voting\_eligibility(age):

if age < 18:

raise AgeTooSmallError(age)

else:

print("Eligible to vote!")

# Test the function

try:

check\_voting\_eligibility(16)

except AgeTooSmallError as e:

print("Custom Exception Caught:", e)