## **Exception Handling in Python**

### **1. Definition:**

Exception handling is a mechanism in Python (and other programming languages) that allows you to handle runtime errors, also known as exceptions, in a controlled manner. Rather than letting an error cause the program to crash, exception handling allows the program to continue or respond appropriately when an error occurs.

An exception is an event that disrupts the normal flow of a program’s execution. It could be anything from invalid user input to a missing file or network failure. When an exception occurs, Python raises an error, and if not handled properly, the program will terminate.

Exception handling allows you to catch these exceptions and handle them, avoiding program crashes.

### **2. Uses :**

* **Error Prevention:** Catching and handling exceptions prevents program crashes caused by unhandled errors.
* **Graceful Recovery:** By catching exceptions, you can take corrective actions or provide user-friendly error messages, allowing the program to recover and continue running.
* **Debugging Aid:** Exception handling provides useful error messages and logs, which help developers diagnose issues and fix them more easily.

### **3. Syntax:**

try:

# Code that may raise an exception

pass

except SomeExceptionType:

# Code to handle the exception

pass

except AnotherExceptionType:

# Code to handle another type of exception

pass

else:

# Code that runs if no exceptions occur

pass

finally:

# Code that runs no matter what (always runs)

pass

### **Explanation of Each Block:**

1. **try:**
   * The code that might raise an exception is placed inside the try block.
   * If no exception occurs, the rest of the program continues executing as normal.
2. **except:**
   * This block catches specific exceptions raised by the try block.
   * You can specify different except blocks to handle various types of exceptions (e.g., ZeroDivisionError, ValueError).
   * If an exception occurs, the code in the corresponding except block is executed, and the program continues.
3. **else:**
   * The else block is executed if no exceptions are raised in the try block.
   * It’s useful when you want to run code that should execute only if no error occurred in the try block.
4. **finally:**
   * The finally block always runs, no matter whether an exception occurs or not.
   * It is commonly used to clean up resources, such as closing files or database connections.

### **4. Example:**

try:

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

result = 10 / num

except ZeroDivisionError:

print("Cannot divide by zero!")

except ValueError:

print("Invalid input! Enter a number.")

else:

print("Result:", result)

finally:

print("Execution complete.")

### **Explanation of the Example:**

The program prompts the user to input a number and attempts to divide 10 by that number. If the user enters zero, a ZeroDivisionError is caught, and an appropriate message is printed. If the input is not a valid number, a ValueError is caught. If no exception occurs, the division result is printed in the else block. Finally, the finally block ensures that "Execution complete." is printed, regardless of any errors.