**Lab Exercise 7 – Exception Handling in Python**

**Lab Exercise: Exception Handling in Python**

**Objective:**

To learn how to use Python's exception handling mechanisms (try, except, else, finally), handle multiple exceptions, raise custom exceptions, and work with exception messages.

**Step 1: Understanding Basic Exception Handling**

First, let's explore the basic syntax of try-except in Python. Start by creating a simple function that divides two numbers, and handle the potential exception of division by zero.

# Step 1: Basic Exception Handling

def divide\_numbers(a, b):

try:

result = a / b

print(f"The result of {a} divided by {b} is {result}")

except ZeroDivisionError:

print("Error: Cannot divide by zero!")

# Test with a valid case

divide\_numbers(10, 2)

# Test with division by zero

divide\_numbers(10, 0)

**Explanation:**

* **try block**: The code that might raise an exception is written inside the try block.
* **except block**: This block catches the exception and allows the program to continue running without crashing.

**Step 2: Handling Multiple Exceptions**

Sometimes, you might need to handle multiple types of exceptions. Let’s modify the function to handle both ZeroDivisionError and TypeError.

# Step 2: Handling Multiple Exceptions

def divide\_numbers\_extended(a, b):

try:

result = a / b

print(f"The result of {a} divided by {b} is {result}")

except ZeroDivisionError:

print("Error: Cannot divide by zero!")

except TypeError:

print("Error: Both inputs must be numbers!")

# Test with a TypeError

divide\_numbers\_extended(10, "2") # This will raise a TypeError

# Test with a valid case

divide\_numbers\_extended(10, 2)

# Test with division by zero

divide\_numbers\_extended(10, 0)

**Explanation:**

* We added another except block to handle TypeError, which occurs when non-numeric inputs are provided.

**Step 3: Using else and finally**

Now, let’s explore the use of the else and finally blocks.

* **else**: If no exception occurs, the code inside the else block is executed.
* **finally**: The code inside the finally block is always executed, whether an exception occurs or not. It’s often used for cleanup operations.

# Step 3: Using `else` and `finally`

def divide\_numbers\_with\_finally(a, b):

try:

result = a / b

except ZeroDivisionError:

print("Error: Cannot divide by zero!")

except TypeError:

print("Error: Both inputs must be numbers!")

else:

print(f"The result of {a} divided by {b} is {result}")

finally:

print("Execution of divide\_numbers\_with\_finally is complete.")

# Test cases

divide\_numbers\_with\_finally(10, 2) # No exception

divide\_numbers\_with\_finally(10, 0) # ZeroDivisionError

divide\_numbers\_with\_finally(10, "2") # TypeError

**Explanation:**

* **else block**: Runs if no exceptions are encountered in the try block.
* **finally block**: Executes regardless of whether an exception was raised.

**Step 4: Raising Custom Exceptions**

In some cases, you may want to raise exceptions deliberately. This can be useful for validating user inputs or ensuring that certain conditions are met.

Let’s raise a ValueError if the inputs are negative numbers.

# Step 4: Raising Custom Exceptions

def divide\_with\_custom\_exception(a, b):

try:

if a < 0 or b < 0:

raise ValueError("Negative numbers are not allowed.")

result = a / b

except ZeroDivisionError:

print("Error: Cannot divide by zero!")

except ValueError as ve:

print(f"Error: {ve}")

else:

print(f"The result of {a} divided by {b} is {result}")

# Test cases

divide\_with\_custom\_exception(10, 2) # Valid case

divide\_with\_custom\_exception(-10, 2) # Raise ValueError

divide\_with\_custom\_exception(10, 0) # ZeroDivisionError

**Explanation:**

* **Raising a custom exception**: The raise keyword is used to raise exceptions manually when a condition is met.