Exception Handling in Python

Created by :- The Easylearn Academy
9662512857

Introduction to Exceptions in Python

- **What is an Exception?**
  - An **exception** is a runtime error that may occur during program execution.
  - Exceptions depend on user input or the environment in which the program runs.
  - Example: In a division program, if a user inputs invalid data (e.g., dividing by zero), the program may crash. This is an exception.
  - Python provides built-in exceptions to handle errors gracefully, ensuring programs can recover and complete their tasks.
- **Why Handle Exceptions?**
  - Prevents program crashes.
  - Allows recovery from errors.
  - Enhances user experience by providing meaningful error messages.

The try-except Block

- **Handling Exceptions**
  - Use the try-except block to catch and handle exceptions.
  - Place critical operations in the try block.
  - Handle errors in the except block.

```
try:
    # Code that might raise an exception
except ExceptionType:
    # Code to handle the exception
    print("An error occurred")
else:
    # Code that runs if no exception occurs
    print("Operation successful")
finally:
    # code that will always run regardless of whether an
   exception occurs.
```

  - A single try block can have multiple except clauses to handle different exception types.
  - The else block executes only if no exception is raised.

Basic example

```
try:
    num1 = int(input("Enter numerator: "))
    num2 = int(input("Enter denominator: "))
    result = num1 / num2
except ZeroDivisionError:
    print("Error: Division by zero is not
  allowed.")
except ValueError:
    print("Error: Please enter valid numbers.")
else:
    print(f"Result: {result}")
```

Catching Multiple Exceptions

- **Handling Multiple Exceptions**
- A single except clause can handle multiple exception types.

```
try:
    # Code that might raise an exception
    value = int(input("Enter a number: "))
    result = 100 / value
except (ValueError, ZeroDivisionError):
    print("Error: Invalid input or division by zero.")
else:
    print(f"Result: {result}")
```

Raising Exceptions

- Use the raise keyword to trigger an exception manually.
- Optionally, provide a message to clarify the reason.

```
try:
    age = int(input("Enter your age: "))
    if age < 18:
        raise ValueError("Age must be 18 or older.")
    else:
        print("Age is valid.")
except ValueError as e:
    print(f"Error: {e}")
```

Custom exception

- In Python, we can define custom exceptions by creating a new class that is derived from the built-in Exception class.
- Here's the syntax to define custom exceptions

```
class CustomError(Exception):
    ...
    pass

try:
    ...

except CustomError:
    ...
```

Custom Exceptions example

```
# define Python user-defined exceptions
class InvalidAgeException(Exception):
    pass

# you need to guess this number
voting_age = 18

try:
    age= int(input("Enter your age: "))
    if age< voting_age:
        raise InvalidAgeException # it will execute except block
    else:
        print("Eligible to Vote")

except InvalidAgeException:
    print("Exception occurred: Invalid Age")
```