TOPSTechnologies

Exception Handling

Presented for :

TOPs Technologies

Presented by :

Sumit B Yadav

Que 1

In programming, an exception is an error that occurs during the execution of a program, disrupting its normal flow. Instead of crashing the program, exceptions allow us to handle errors gracefully.

1. Understanding Exceptions

An exception occurs when an operation fails, such as:

- Dividing by zero (ZeroDivisionError)
- Accessing an undefined variable (NameError)
- Using the wrong data type (TypeError)
- File-related errors (FileNotFoundError)

example:

```
x = 5 / 0  # This will raise a ZeroDivisionError
```

2. Handling Exceptions Using try and except

To prevent crashes, we use a try block to test code, and except to handle any exceptions.

syntax;

```
try:
# Code that might raise an exception
except ExceptionType:
# Code to handle the exception
```

3. Handling Multiple Exceptions

You can handle different types of exceptions separately.

basic syntax:

```
try:
num = int(input("Enter a number: "))
result = 10 / num
except ZeroDivisionError:
print("You cannot divide by zero!")
except ValueError:
print("Invalid input! Please enter a number.")
```

4. Using finally for Cleanup

The finally block runs no matter what—whether an exception occurs or not. It is useful for resource cleanup, such as closing files or database connections.

basic syntax:

```
try:
file = open("data.txt", "r")
content = file.read()
except FileNotFoundError:
print("File not found!")
finally:
print("Closing file...")
file.close()  # This runs whether or not an exception occurs.
```

5. Catching Any Exception

If you're unsure about the specific exception, use a generic except:

basic syntax:

```
try:
x = 10 / int("a")
except Exception as e:
print(f"An error occurred: {e}")
```

Que. 2

When developing software, handling different types of errors gracefully improves program stability and user experience. Python allows handling multiple exceptions and even creating custom exceptions for better error management.

1. Handling Multiple Exceptions

Python provides multiple ways to handle different exceptions within a try-except block.

Method 1: Multiple except Blocks

Each except block catches a specific type of exception.

basic syntax:

```
try:
num = int(input("Enter a number: "))
result = 10 / num
except ZeroDivisionError:
print("You cannot divide by zero!")
except ValueError:
print("Invalid input! Please enter a numeric value.")
except Exception as e:
print(f"An unexpected error occurred: {e}")
```

Method 2: Handling Multiple Exceptions in a Single except
If multiple exceptions should have the same handling logic, use a tuple.

basic syntax:

```
try:
num = int(input("Enter a number: "))
result = 10 / num
except (ZeroDivisionError, ValueError) as e:
print(f"An error occurred: {e}")
```

2. Raising Custom Exceptions
Python allows creating custom exceptions to handle specific application logic.
Creating a Custom Exception
Define a custom exception by inheriting from Python’s built-in Exception class.

basic syntax:

```
class NegativeNumberError(Exception): """Exception raised for negative
numbers."""def __init__(self, value): self.value = value
super().__init__(f"Negative numbers are not allowed: {value}")
```

```
try: num = int(input("Enter a positive number: ")) if num < 0: raise NegativeNumberError(num) print(f"Your number is: {num}") except NegativeNumberError as e: print(e)
```

3. Using finally with Multiple Exceptions

The finally block always executes, regardless of whether an exception occurs.

basic ssyntax:

```
try:
 file = open("data.txt", "r")
   content = file.read()
except FileNotFoundError:
  print("File not found!")
finally:
  print("Closing file...")
file.close()  # Cleanup code
```