Chapter 9 Exception Handling

An **exception** is an event that disrupts the normal flow of a program's execution. It occurs when an error arises in the program.

Common exceptions: ZeroDivisionError, ValueError, TypeError, IndexError, and more.

Exception Handling is used to handle errors without crashing the program, allowing the program to recover or exit gracefully.

1. try-except Blocks

Syntax:

```
try:
    # Code that may raise an exception
except [SomeException]:
    # Code that runs if exception occurs
```

1.1 Catching General Exceptions

Example 01: Using try-except block.

```
try:
    x = 10 / 0  # This will raise a ZeroDivisionError
except:
    print("Exception occurs")
```

Output:

```
Exception occurs
```

1.2 Catching Specific Exceptions

It is best practice to catch specific exceptions rather than using a general except block.

Example 02: Handle the ZeroDivisionError exception.

```
try:
    x = 10 / 0  # This will raise a ZeroDivisionError

except ZeroDivisionError:
    print("You cannot divide by zero!")
```

Output:

```
You cannot divide by zero!
```

Example 03: Handle the `ValueError` exception.

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

except ValueError:
    print("Invalid input! Please enter an integer.")
```

Output:

```
Enter a number: 10
```

Output:

```
Enter a number: hello
Invalid input! Please enter an integer.
```

1.3 Multiple `except` Blocks

You can have multiple except blocks to handle different types of exceptions.

Example 04: Use multiple `except` blocks

```
try:
    x = int(input("Enter a number: "))
    y = 10 / x
    print("y =", y)

except ValueError:
    print("Please enter a valid number.")

except ZeroDivisionError:
    print("Cannot divide by zero!")

except:
  print("Something else went wrong")
```

Output:

```
Enter a number: 2
y = 5.0
```

Output:

```
Enter a number: abc
Please enter a valid number.
```

Output:

```
Enter a number: 0
Cannot divide by zero!
```

1.3 `else` and `finally` Blocks

The `else` block runs if no exception is raised in the try block.

The `finally` block always runs, regardless of whether an exception was raised or not. It is typically used for cleanup tasks (e.g., closing files or releasing resources).

Example 04: Use `else` and `finally` blocks

```python
try:
    x = int(input("Enter a number: "))
    y = 10 / x
    print("y =", y)

except ValueError:
    print("Please enter a valid number.")

except ZeroDivisionError:
    print("Cannot divide by zero!")

else:
    print("Division successful!")

finally:
    print("Execution completed.")
```

Output:

```
Enter a number: 2
y = 5.0
Division successful!
Execution completed.
```

Output:

```
Enter a number: abc
Please enter a valid number.
Execution completed.
```

Output:

```
Enter a number: 0
Cannot divide by zero!
Execution completed.
```

1.4 Raising Exceptions

You can raise or throw your own exceptions using the `raise` keyword.

Example 05: Raise an exception

```
def check_age(age):
    if age < 18:
        raise ValueError("Age must be at least 18.")
    print("Age is valid.")

# Test program
try:
    check_age(16)

except ValueError as e:
    print(e)
```

Output:

```
Age must be at least 18.
```

Example 06: Raise an exception

```
def check_age(age):
    if age < 18:
        raise Exception("Age must be at least 18.")
    print("Age is valid.")

# Test program
try:
    check_age(16)

except Exception as e:
    print(e)
```

Output:

```
Age must be at least 18.
```

Example 07: Raise an exception

```
def check_age(age):
    if age < 18:
        raise Exception()

    print("Age is valid.")

# Test program
try:
    check_age(16)

except Exception:
    print("Exception occurs")
```

Output:

```
Exception occurs
```

1.5 Custom Exception Classes

You can define your own exceptions by inheriting the built-in Exception class.

Example 08: Custom Exception Classes

```
class AgeTooLowError(Exception):
    pass

def check_age(age):
    if age < 18:
        raise AgeTooLowError("Age must be at least 18.")
    return True

try:
    check_age(16)

except AgeTooLowError as e:
    print(e)
```

Output:

```
Age must be at least 18.
```

Exercises

1. Write a program that asks the user to enter their age. Handle the `ValueError` case where the user enters a non-integer value.

2. Write a program that creates a list with 5 elements and asks the user to input an index. Display the element at that index. Handle:
   - `IndexError` if the index is out of bounds
   - `ValueError` if the entered value is not integer.

3. Write a program that opens a file, and display its content. Use `else` to display "File read successfully!" and also close the file, and use `finally` to display the "Program Ended!". Handle the `FileNotFoundError` if the file does not exist.

4. Create a dictionary with keys as integers and values as strings. Write a program that asks the user for a key and displays the corresponding value.

   Handle:
   - `KeyError` if the key is not in the dictionary.
   - `ValueError` if the user inputs a non-integer key.

5. Write a program that asks the user to input a string and an index. Display the character at the specified index. Handle:
   - `IndexError` if the position is out of range.
   - `ValueError` if the position is not an integer.

6. Define a custom exception `NegativeNumberError`, and write a program that raises this exception if the user enters a negative number.