Exceptional Cases in Python

1. Introduction

In Python, exceptions are errors that occur when the program is running. Instead of letting the program crash, Python lets you handle these errors so your code can continue running or fail gracefully. In this guide, we'll explain what exceptions are, how to handle them, and show some simple examples.

---

2. What Are Exceptions?

- **Definition:**
  Exceptions are errors that happen during runtime. They stop the normal flow of the program and need special handling to resolve or inform you of the problem.
- **Common Examples:**
  - **ZeroDivisionError:** Happens when you try to divide by zero.
  - **FileNotFoundError:** Occurs if you try to open a file that does not exist.
  - **ValueError:** Raised when a function receives an argument that has the right type but an inappropriate value.
  - **TypeError:** Happens when an operation or function is applied to an object of an inappropriate type.

---

3. Basic Exception Handling

3.1 Try and Except Block

Wrap code that might cause an error in a `try` block and handle the error in an `except` block.

Example:

```
try:
    result = 10 / 0
except ZeroDivisionError:
    print("Error: You cannot divide by zero!")
```

3.2 Multiple Except Blocks

You can use more than one `except` block to handle different errors separately.

Example:

```
try:
    x = int("hello")
except ValueError:
    print("Error: Value conversion failed.")
except TypeError:
    print("Error: Wrong type used.")
```

\

3.3 Else Block

The `else` block runs when there is no error in the `try` block.

Example:

```
try:
    number = int("123")
except ValueError:
    print("Error: Conversion failed!")
else:
    print(f"Conversion successful: {number}")
```

3.4 Finally Block

The `finally` block always runs, whether an exception occurs or not. It is useful for cleaning up, like closing files.

Example:

```
try:
    file = open("data.txt", "r")
except FileNotFoundError:
    print("Error: File not found!")
else:
    print("File opened successfully!")
finally:
    print("Execution completed.")
```

---

4. Code Examples for Exceptional Cases

4.1 Handling Files

When working with files, an error may occur if the file does not exist.

Example:

```
try:
    with open("example.txt", "r") as file:
        content = file.read()
except FileNotFoundError:
    print("Error: 'example.txt' not found.")
else:
    print("File content:")
    print(content)
```

4.2 Handling User Input

When taking user input, you may face conversion problems.

Example:

```
try:
    age = int(input("Enter your age: "))
except ValueError:
    print("Please enter a valid number for your age.")
else:
    print(f"Your age is {age}.")
```

4.3 Creating Custom Exceptions

Sometimes you need your own exception types to handle special cases.

Example:

```python
class NegativeNumberError(Exception):
    """Raised when a negative number is encountered."""
    pass

def check_positive(number):
    if number < 0:
        raise NegativeNumberError("Error: Negative numbers are not allowed.")
    return number

try:
    result = check_positive(-5)
except NegativeNumberError as error:
    print(error)
else:
    print(f"Valid number: {result}")
```

---

5. Summary

- **Try Block:**
  Place the code that might cause an error inside a `try` block.
- **Except Block:**
  Handle the errors by catching them with `except` blocks.
- **Else Block:**
  Code under `else` runs only if no error occurs in the `try` block.
- **Finally Block:**
  Code under `finally` runs regardless of whether an error occurred or not.
- **Custom Exceptions:**
  You can create and use your own exception classes for specific error cases.

---

6. Conclusion

Handling exceptional cases in Python is essential to building robust and reliable applications. By using the try-except-else-finally blocks, you can gracefully manage errors in your code. This not only prevents your program from crashing but also makes debugging much easier.
Understanding exceptions will help you write cleaner and more professional Python code.