Handling Exceptions in Python with `try` and `except`

Proper exception handling improves code robustness and ensures user-friendly error messages when things go wrong.

1. Basic Exception Handling with `try` and `except`

The `try` block executes code that might raise exceptions. If an exception occurs, the `except` block catches it and handles it gracefully.

Example:

```
def divide_by(a, b):
    return a / b

try:
    result = divide_by(40, 0)
except ZeroDivisionError:
    print("Something went wrong.")
```

Output:

```
Something went wrong.
```

2. Catching Specific Exceptions

You can handle specific exceptions to provide tailored error messages.

Example:

```
try:
    result = divide_by(40, 0)
except ZeroDivisionError:
    print("Error: We cannot divide by zero.")
```

Output:

```
Error: We cannot divide by zero.
```

3. Accessing Exception Details

Use the `as` keyword to capture the exception object and retrieve additional information.

Example:

```
try:
    result = divide_by(40, 0)
except ZeroDivisionError as e:
    print(f"Error: {e}")
    print(f"Exception Type: {e.__class__}")
```

Output:

```
Error: division by zero
Exception Type: <class 'ZeroDivisionError'>
```

4. Catching Multiple Exceptions

You can handle different exceptions with multiple `except` blocks.

Example:

```
try:
    result = divide_by(40, "10")
except ZeroDivisionError:
    print("Error: Cannot divide by zero.")
except TypeError:
    print("Error: Unsupported types for division.")
```

Output:

```
Error: Unsupported types for division.
```

5. Catching All Exceptions

Use the base class `Exception` to handle any type of exception.

Example:

```
try:
    result = divide_by(40, 0)
except Exception as e:
    print(f"General error: {e}")
```

Output:

```
General error: division by zero
```

6. Combining Specific and Generic Exception Handling

To handle multiple scenarios, start with specific exceptions and fall back to a generic handler.

Example:

```
try:
    result = divide_by(40, 0)
except ZeroDivisionError as e:
    print(f"Math error: {e}")
except Exception as e:
    print(f"General error: {e}")
```

Output:

```
Math error: division by zero
```

Summary

Feature	Explanation
`try` block	Code that might raise an exception.
`except` block	Handles exceptions that occur in the `try` block.
Specific Exceptions	Tailor messages for known exceptions like `ZeroDivisionError`.
Generic Exceptions	Catch all errors using `except Exception`.
Access Details	Use `as e` to capture and print exception details like `e.__class__`.
Chained Exceptions	Use multiple `except` blocks for different exception types.

By mastering `try` and `except`, you can make your Python programs robust, user-friendly, and maintainable.