Day54_Exception_Handling

July 29, 2025

Python Exception Handling: From Basics to Advanced

What is Exception Handling? Exception Handling in Python is a mechanism that allows you to gracefully respond to errors (called **exceptions**) that occur during program execution. Instead of crashing, your program can catch the error and continue or display a meaningful message.

Why Use It?

- Prevents your program from crashing on runtime errors
- Helps in debugging and maintaining large applications
- Allows clean-up operations (e.g., closing files, releasing resources)

How It Works:

When Python encounters an error during code execution, it:

1. **Raises** an exception.
2. Looks for a **`try` block** to catch it.
3. If found, runs the corresponding **`except` block**.
4. Optionally runs **`else`** (if no error occurred).
5. Always runs **`finally`**, whether there was an exception or not.

Syntax of Exception Handling:

```
try:
    # Code that might cause an exception
except SomeException:
    # Code that runs if exception occurs
else:
    # Runs if no exception occurs
finally:
    # Always runs (clean-up code)
```

Built-in Exception Types:

Exception	Description
`ZeroDivisionError`	Dividing by zero
`ValueError`	Invalid type (e.g., string to int)
`IndexError`	Invalid index for list/tuple
`KeyError`	Accessing invalid dict key
`TypeError`	Wrong type of operation

1 Basic Examples

1.1 Division by Zero

[1]:
```python
try:
    result = 10 / 0
except ZeroDivisionError:
    print("You cannot divide by zero!")
```

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

1.2 String Conversion Error

[2]:
```python
try:
    age = int("twenty")
except ValueError as ve:
    print("Invalid input, please enter a number:", ve)
```

```
Invalid input, please enter a number: invalid literal for int() with base 10:
'twenty'
```

2 else, finally, and Multiple Exceptions

3 Using else

[3]:
```python
try:
    num = int(input("Enter a number: "))
except ValueError:
    print("That's not a valid number!")
else:
    print("Input was successfully processed.")
```

```
Enter a number:  8
```

```
Input was successfully processed.
```

[4]:
```python
try:
    num = int(input("Enter a number: "))
except ValueError:
    print("That's not a valid number!")
else:
    print("Input was successfully processed.")
```

```
Enter a number:  *8*
```

```
That's not a valid number!
```

3.1 Using finally

[5]:
```
try:
    f = open("sample.txt", "r")
    print(f.read())
except FileNotFoundError:
    print("File not found.")
finally:
    print("This runs no matter what!")
```

```
File not found.
This runs no matter what!
```

3.2 Multiple Exception Types

[6]:
```
try:
    nums = [1, 2, 3]
    print(nums[5])
except IndexError:
    print("Index is out of range.")
except Exception as e:
    print("Some other error occurred:", e)
```

```
Index is out of range.
```

4 Raising and Custom Exceptions

4.1 Raising Errors with raise

[7]:
```
age = -5
if age < 0:
    raise ValueError("Age cannot be negative!")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[7], line 3
      1 age = -5
      2 if age < 0:
----> 3     raise ValueError("Age cannot be negative!")

ValueError: Age cannot be negative!
```

4.2 Custom Exception Class

[8]:
```
class WeakPasswordError(Exception):
    pass

def check_password(password):
    if len(password) < 6:
        raise WeakPasswordError("Password is too short!")

try:
    check_password("123")
except WeakPasswordError as wpe:
    print("Custom error:", wpe)
```

```
Custom error: Password is too short!
```

5 Best Practices & Exception Tree

5.1 Avoid Generic except:

[9]:
```
try:
    x = 10 / 2
except Exception as e:
    print("Handled error:", e)
```

5.2 Use Logging Instead of Print

[10]:
```
import logging
logging.basicConfig(level=logging.ERROR)

try:
    result = 10 / 0
except ZeroDivisionError as e:
    logging.error("ZeroDivisionError occurred: %s", e)
```

```
ERROR:root:ZeroDivisionError occurred: division by zero
```

Python Exception Hierarchy - BaseException - Exception - ArithmeticError - LookupError - ValueError - etc.

6 Real-Life Fun Examples

6.1 Pizza Sharing (Divide by Zero)

[11]:
```
people = 0
try:
    print("Each person gets", 8 // people, "slices")
except ZeroDivisionError:
    print("You can't divide pizza by zero people!")
```

```
You can't divide pizza by zero people!
```

6.2 ATM Withdrawal

[12]:
```
balance = 1000
withdraw = int(input("Enter amount to withdraw: "))
try:
    if withdraw > balance:
        raise Exception("Insufficient Balance")
    balance -= withdraw
    print("New balance:", balance)
except Exception as e:
    print(e)
```

```
Enter amount to withdraw:  855
```

```
New balance: 145
```

[13]:
```
balance = 1000
withdraw = int(input("Enter amount to withdraw: "))
try:
    if withdraw > balance:
        raise Exception("Insufficient Balance")
    balance -= withdraw
    print("New balance:", balance)
except Exception as e:
    print(e)
```

```
Enter amount to withdraw:  5000
```

```
Insufficient Balance
```

[14]:
```
balance = 1000
withdraw = int(input("Enter amount to withdraw: "))
try:
    if withdraw > balance:
        raise Exception("Insufficient Balance")
    balance -= withdraw
    print("New balance:", balance)
except Exception as e:
    print(e)
```

```
Enter amount to withdraw:  1000
```

```
New balance: 0
```

6.3 Login System

[15]:
```
def login(username, password):
    if username != "admin" or password != "123":
        raise ValueError("Invalid credentials")
    return "Login Successful"

try:
    print(login("user", "wrong"))
except ValueError as e:
    print(e)
```

```
Invalid credentials
```

7 Practice Challenges (With Solutions)

7.1 Handle input error

[16]:
```
def get_integer():
    try:
        return int(input("Enter a number: "))
    except ValueError:
        print("Not a number!")
        return 0
```

7.2 Custom Exception for Bank

[17]:
```
class InsufficientBalance(Exception):
    pass

def withdraw(balance, amount):
    if amount > balance:
        raise InsufficientBalance("Not enough money!")
    return balance - amount

try:
    print(withdraw(500, 800))
except InsufficientBalance as e:
    print("Banking Error:", e)
```

```
Banking Error: Not enough money!
```

7.3 Marks Validator

[18]:
```
def validate_marks(marks):
    if not 0 <= marks <= 100:
        raise ValueError("Marks must be between 0 and 100")
    return "Valid"
```

```python
print(validate_marks(75))
# print(validate_marks(120))  # Uncomment to test
```

```
Valid
```

Summary

- Always use specific exceptions (`ValueError`, `ZeroDivisionError`, etc.)
- Use `else` for code that should only run if no error occurs
- Use `finally` for clean-up actions
- Raise custom errors for more readable and maintainable code

Exception handling is a **superpower** that makes your code **robust, secure, and production-ready**.

Keep practicing by adding exception handling to every mini project you build!