Exception Handling in Python

Estimated time needed: 10 Minutes

Objectives

1. Understanding Exceptions
2. Distinguishing Errors from Exceptions
3. Familiarity with Common Python Exceptions
4. Managing Exceptions Effectively

What are exceptions?

Exceptions are alerts when something unexpected happens while running a program. It could be a mistake in the code or a situation that was not planned for. Python can raise these alerts automatically, but we can also trigger them on purpose using the raise command. The cool part is that we can prevent our program from crashing by handling exceptions.

Errors vs. Exceptions

Hold on, what is the difference between errors and exceptions? Well, `errors` are usually big problems that come from the computer or the system. They often make the program stop working completely. On the other hand, `exceptions` are more like issues we can control. They happen because of something we did in our code and can usually be fixed, so the program keeps going.

Here is the difference between `Errors and exceptions`:-

Aspect	Errors	Exceptions
Origin	Errors are typically caused by the environment, hardware, or operating system.	Exceptions are usually a result of problematic code execution within the program.
Nature	Errors are often severe and can lead to program crashes or abnormal termination.	Exceptions are generally less severe and can be caught and handled to prevent program termination.
Handling	Errors are not usually caught or handled by the program itself.	Exceptions can be caught using try-except blocks and dealt with gracefully, allowing the program to continue execution.
Examples	Examples include “SyntaxError” due to incorrect syntax or “NameError” when a variable is not defined.	Examples include “ZeroDivisionError” when dividing by zero, or “FileNotFoundError” when attempting to open a non-existent file.
Categorization	Errors are not classified into categories.	Exceptions are categorized into various classes, such as “ArithmeticError,” “IOError,” ValueError,” etc., based on their nature.

Common Exceptions in Python

Here are a few examples of exceptions we often run into and can handle using this tool:

- **ZeroDivisionError:** This error arises when an attempt is made to divide a number by zero. Division by zero is undefined in mathematics, causing an arithmetic error. For instance:
  For example:

```
result = 10 / 0
print(result)
# Raises ZeroDivisionError
```

- **ValueError:** This error occurs when an inappropriate value is used within the code. An example of this is when trying to convert a non-numeric string to an integer:
  For example:

```
num = int("abc")
# Raises ValueError
```

- **FileNotFoundError:** This exception is encountered when an attempt is made to access a file that does not exist.
  For example:

```
with open("nonexistent_file.txt", "r") as file:
        content = file.read()   # Raises FileNotFoundError
```

- **IndexError:** An IndexError occurs when an index is used to access an element in a list that is outside the valid index range.
  For example:

```
my_list = [1, 2, 3]
value = my_list[1]  # No IndexError, within range
missing = my_list[5]  # Raises IndexError
```

- **KeyError:** The KeyError arises when an attempt is made to access a non-existent key in a dictionary.
  For example:

```
my_dict = {"name": "Alice", "age": 30}
value = my_dict.get("city")  # No KeyError, using .get() method
missing = my_dict["city"]  # Raises KeyError
```

- **TypeError:** The TypeError occurs when an object is used in an incompatible manner. An example includes trying to concatenate a string and an integer:
  For example:

```
result = "hello" + 5
# Raises TypeError
```

- **AttributeError:** An AttributeError occurs when an attribute or method is accessed on an object that doesn't possess that specific attribute or method. For instance:
  For example:

```
text = "example"
length = len(text)  # No AttributeError, correct method usage
missing = text.some_method()  # Raises AttributeError
```

- **ImportError:** This error is encountered when an attempt is made to import a module that is unavailable. For example: `import non_existent_module`

> Note: **Please remember, the exceptions you will encounter are not limited to just these. There are many more in Python. However, there is no need to worry. By using the technique provided below and following the correct syntax, you will be able to handle any exceptions that come your way.**

Handling Exceptions:

Python has a handy tool called `try and except` that helps us manage exceptions.

Try and Except : You can use the try and except blocks to prevent your program from crashing due to exceptions.

Here's how they work:

1. The code that may result in an exception is contained in the try block.
2. If an exception occurs, the code directly jumps to except block.
3. In the except block, you can define how to handle the exception gracefully, like displaying an error message or taking alternative actions.
4. After the except block, the program continues executing the remaining code.

Example: Attempting to divide by zero

```
# using Try- except
try:
    # Attempting to divide 10 by 0
    result = 10 / 0
except ZeroDivisionError:
    # Handling the ZeroDivisionError and printing an error message
    print("Error: Cannot divide by zero")
# This line will be executed regardless of whether an exception occurred
print("outside of try and except block")
```

Next Step

As we finish up this reading, you are ready to move on to the next part where you will practice handling errors. For better learning, try out different types of data in the lab. This way, you will encounter various errors and learn how to deal with them effectively. This knowledge will help you write stronger and more reliable code in the future.

Author(s)

Akansha Yadav