Python Exception Handling

Last Updated : 02 Apr, 2025

Python Exception Handling handles errors that occur during the execution of a program. Exception handling allows to respond to the error, instead of crashing the running program. It enables you to catch and manage errors, making your code more robust and user-friendly. Let's look at an example:

Handling a Simple Exception in Python

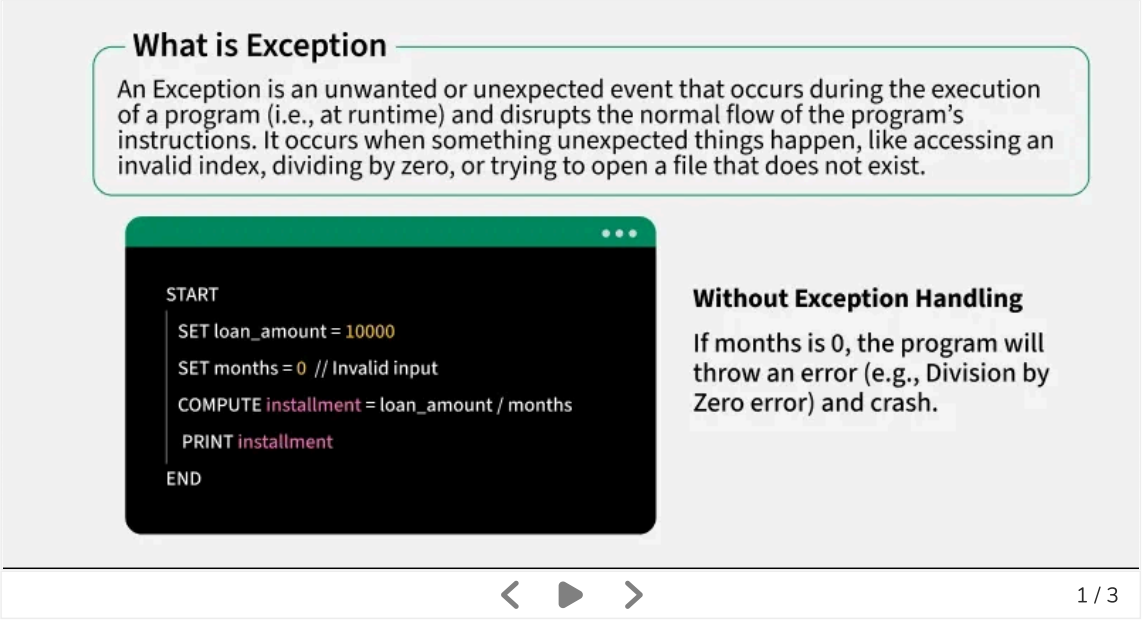
Exception handling helps in preventing crashes due to errors. Here's a basic example demonstrating how to catch an exception and handle it gracefully:

```
# Simple Exception Handling Example
n = 10
try:
    res = n / 0  # This will raise a ZeroDivisionError

except ZeroDivisionError:
    print("Can't be divided by zero!")
```

Output

Explanation: In this example, dividing number by 0 raises a ZeroDivisionError. The try block contains the code that might cause an exception and the except block handles the exception, printing an error message instead of stopping the program.



Difference Between Exception and Error

- **Error**: Errors are serious issues that a program should not try to handle. They are usually problems in the code's logic or configuration and need to be fixed by the programmer. Examples include syntax errors and memory errors.
- **Exception**: Exceptions are less severe than errors and can be handled by the program. They occur due to situations like invalid input, missing files or network issues.

Example:

```
# Syntax Error (Error)
print("Hello world"  # Missing closing parenthesis

# ZeroDivisionError (Exception)
n = 10
res = n / 0
```

Explanation: A syntax error is a coding mistake that prevents the code from running. In contrast, an exception like ZeroDivisionError can be managed during the program's execution using exception handling.

Syntax and Usage

Exception handling in Python is done using the try, except, else and finally blocks.

```
try:
    # Code that might raise an exception
except SomeException:
    # Code to handle the exception
else:
    # Code to run if no exception occurs
finally:
    # Code to run regardless of whether an exception occurs
```

try, except, else and finally Blocks

- **try Block**: try block lets us test a block of code for errors. Python will "try" to execute the code in this block. If an exception occurs, execution will immediately jump to the except block.
- **except Block:** except block enables us to handle the error or exception. If the code inside the try block throws an error, Python jumps to the except block and executes it. We can handle specific exceptions or use a general except to catch all exceptions.
- **else Block:** else block is optional and if included, must follow all except blocks. The else block runs only if no exceptions are raised in the try block. This is useful for code that should execute if the try block succeeds.
- **finally Block:** finally block always runs, regardless of whether an exception occurred or not. It is typically used for cleanup operations (closing files, releasing resources).

Example:

```
try:
```

```
    n = 0
    res = 100 / n

except ZeroDivisionError:
    print("You can't divide by zero!")

except ValueError:
    print("Enter a valid number!")

else:
    print("Result is", res)

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

Output

```
You can't divide by zero!
Execution complete.
```

Explanation:

- **try block** asks for user input and tries to divide 100 by the input number.
- **except blocks** handle ZeroDivisionError and ValueError.
- **else block** runs if no exception occurs, displaying the result.
- **finally block** runs regardless of the outcome, indicating the completion of execution.

Common Exceptions in Python

Python has many built-in exceptions, each representing a specific error condition. Some common ones include:

Exception Name	Description
BaseException	The base class for all built-in exceptions.
Exception	The base class for all non-exit exceptions.
ArithmeticError	Base class for all errors related to arithmetic operations.
ZeroDivisionError	Raised when a division or modulo operation is performed with zero as the divisor.
OverflowError	Raised when a numerical operation exceeds the maximum limit of a data type.
FloatingPointError	Raised when a floating-point operation fails.
AssertionError	Raised when an assert statement fails.
AttributeError	Raised when an attribute reference or assignment fails.
IndexError	Raised when a sequence subscript is out of range.

Exception Name	Description
KeyError	Raised when a dictionary key is not found.
MemoryError	Raised when an operation runs out of memory.
NameError	Raised when a local or global name is not found.
OSError	Raised when a system-related operation (like file I/O) fails.
TypeError	Raised when an operation or function is applied to an object of inappropriate type.
ValueError	Raised when a function receives an argument of the right type but inappropriate value.
ImportError	Raised when an import statement has issues.
ModuleNotFoundError	Raised when a module cannot be found.

Python Catching Exceptions

When working with exceptions in Python, we can handle errors more efficiently by specifying the types of exceptions we expect. This can make code both safer and easier to debug.

Catching Specific Exceptions

Catching specific exceptions makes code to respond to different exception types differently.

Example:

```
try:
    x = int("str")  # This will cause ValueError

    #inverse
    inv = 1 / x

except ValueError:
    print("Not Valid!")

except ZeroDivisionError:
    print("Zero has no inverse!")
```

Output

```
Not Valid!
```

Explanation:

- The ValueError is caught because the string “str” cannot be converted to an integer.
- If x were 0 and conversion successful, the ZeroDivisionError would be caught when attempting to calculate its inverse.

Catching Multiple Exceptions

We can catch multiple exceptions in a single block if we need to handle them in the same way or we can separate them if different types of exceptions require different handling.

Example:

```
a = ["10", "twenty", 30]  # Mixed list of integers and strings
try:
    total = int(a[0]) + int(a[1])  # 'twenty' cannot be converted to int

except (ValueError, TypeError) as e:
    print("Error", e)

except IndexError:
    print("Index out of range.")
```

Output

```
Error invalid literal for int() with base 10: 'twenty'
```

Explanation:

- The ValueError is caught when trying to convert “twenty” to an integer.
- TypeError might occur if the operation was incorrectly applied to non-integer types, but it’s not triggered in this specific setup.
- IndexError would be caught if an index outside the range of the list was accessed, but in this scenario, it’s under control.

Catch-All Handlers and Their Risks

Here’s a simple calculation that may fail due to various reasons.

Example:

```
try:
    # Simulate risky calculation: incorrect type operation
    res = "100" / 20

except ArithmeticError:
    print("Arithmetic problem.")

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

Output

```
Something went wrong!
```

Explanation:

- An ArithmeticError (more specific like ZeroDivisionError) might be caught if this were a number-to-number division error. However, TypeError is actually triggered here due to attempting to divide a string by a number.

- **catch-all except:** is used to catch the TypeError, demonstrating the risk that the programmer might not realize the actual cause of the error (type mismatch) without more detailed error logging.

Raise an Exception

We raise an exception in Python using the raise keyword followed by an instance of the exception class that we want to trigger. We can choose from built-in exceptions or define our own custom exceptions by inheriting from Python's built-in Exception class.

Basic Syntax:

raise ExceptionType("Error message")

Example:

```
def set(age):
    if age < 0:
        raise ValueError("Age cannot be negative.")
    print(f"Age set to {age}")

try:
    set(-5)
except ValueError as e:
    print(e)
```

Output

```
Age cannot be negative.
```

Explanation:

- The function set checks if the age is negative. If so, it raises a ValueError with a message explaining the issue.
- This ensures that the age attribute cannot be set to an invalid state, thus maintaining the integrity of the data.

Advantages of Exception Handling:

- **Improved program reliability**: By handling exceptions properly, you can prevent your program from crashing or producing incorrect results due to unexpected errors or input.
- **Simplified error handling**: Exception handling allows you to separate error handling code from the main program logic, making it easier to read and maintain your code.
- **Cleaner code:** With exception handling, you can avoid using complex conditional statements to check for errors, leading to cleaner and more readable code.
- **Easier debugging**: When an exception is raised, the Python interpreter prints a traceback that shows the exact location where the exception occurred, making it easier to debug your code.

Disadvantages of Exception Handling:

- **Performance overhead:** Exception handling can be slower than using conditional statements to check for errors, as the interpreter has to perform additional work to catch and handle the exception.
- **Increased code complexity**: Exception handling can make your code more complex, especially if you have to handle multiple types of exceptions or implement complex error handling logic.
- **Possible security risks:** Improperly handled exceptions can potentially reveal sensitive information or create security vulnerabilities in your code, so it's important to handle exceptions carefully and avoid exposing too much information about your program.

Next Article

User-defined Exceptions in Python with Examples

Similar Reads

User-defined Exceptions in Python with Examples

In Python, exceptions are used to handle errors that occur during the execution of a program. While Python provides many built-in exceptions, sometimes we may need to crea...

14 min read

Python Built-in Exceptions

In Python, exceptions are events that can alter the flow of control in a program. These errors can arise during program execution and need to be handled appropriately. Pyt...

15+ min read

How to Break a Function in Python?

In Python, breaking a function allows us to exit from loops within the function. With the help of the return statement and the break keyword, we can control the flow of th...

10 min read

Errors and Exceptions in Python

Errors are problems in a program that causes the program to stop its execution. On the other hand, exceptions are raised when some internal events change the program's nor...

15+ min read

Python Try Except

In Python, errors and exceptions can interrupt the execution of program. Python provides try and except blocks to handle situations like this. In case an error occurs in t...

15+ min read

Handling TypeError Exception in Python

TypeError is one among the several standard Python exceptions. TypeError is raised whenever an operation is performed on an incorrect/unsupported object type. For example,...

12 min read

File Handling in Python

File handling refers to the process of performing operations on a file such as creating, opening, reading, writing and closing it, through a programming interface. It invo...

15+ min read

Python OOPs Concepts

Object Oriented Programming is a fundamental concept in Python, empowering developers to build modular, maintainable, and scalable applications. By understanding the core...

15+ min read

Python Modules

Python Module is a file that contains built-in functions, classes,its and variables. There are many Python modules, each with its specific work. In this article, we will c...

15+ min read

Java Exception Handling

Exception handling in Java allows developers to manage runtime errors effectively by using mechanisms like try-catch block, finally block, throwing Exceptions, Custom Exce...

15+ min read

GeeksforGeeks
Sanchhaya Education Private Limited

Corporate & Communications Address:
A-143, 7th Floor, Sovereign Corporate Tower, Sector- 136, Noida, Uttar Pradesh (201305)

Registered Address:
K 061, Tower K, Gulshan Vivante Apartment, Sector 137, Noida, Gautam Buddh Nagar, Uttar Pradesh, 201305

Biology
Social Science
English Grammar

PL/SQL
MongoDB

Preparation Corner

Company-Wise Recruitment Process
Aptitude Preparation
Puzzles
Company-Wise Preparation

More Tutorials

Software Development
Software Testing
Product Management
Project Management
Linux
Excel
All Cheat Sheets

Machine Learning/Data Science

Complete Machine Learning & Data Science Program - [LIVE]
Data Analytics Training using Excel, SQL, Python & PowerBI - [LIVE]
Data Science Training Program - [LIVE]
Data Science Course with IBM Certification

Programming Languages

C Programming with Data Structures
C++ Programming Course
Java Programming Course
Python Full Course

Clouds/Devops

DevOps Engineering
AWS Solutions Architect Certification
Salesforce Certified Administrator Course

GATE 2026

GATE CS Rank Booster
GATE DA Rank Booster
GATE CS & IT Course - 2026
GATE DA Course 2026
GATE Rank Predictor