Python Custom Exceptions

programiz.com/python-programming/user-defined-exception

Join our newsletter for the latest updates.

In this tutorial, you will learn how to define custom exceptions depending upon your requirements with the help of examples.

Python has numerous built-in exceptions that force your program to output an error when something in the program goes wrong.

However, sometimes you may need to create your own custom exceptions that serve your purpose.

Creating Custom Exceptions

In Python, users can define custom exceptions by creating a new class. This exception class has to be derived, either directly or indirectly, from the built-in `Exception` class. Most of the built-in exceptions are also derived from this class.

```
>>> class CustomError(Exception):
...     pass
...

>>> raise CustomError
Traceback (most recent call last):
...
__main__.CustomError

>>> raise CustomError("An error occurred")
Traceback (most recent call last):
...
__main__.CustomError: An error occurred
```

Here, we have created a user-defined exception called `CustomError` which inherits from the `Exception` class. This new exception, like other exceptions, can be raised using the `raise` statement with an optional error message.

When we are developing a large Python program, it is a good practice to place all the user-defined exceptions that our program raises in a separate file. Many standard modules do this. They define their exceptions separately as `exceptions.py` or `errors.py` (generally but not always).

User-defined exception class can implement everything a normal class can do, but we generally make them simple and concise. Most implementations declare a custom base class and derive others exception classes from this base class. This concept is made clearer in the following example.

Example: User-Defined Exception in Python

In this example, we will illustrate how user-defined exceptions can be used in a program to raise and catch errors.

This program will ask the user to enter a number until they guess a stored number correctly. To help them figure it out, a hint is provided whether their guess is greater than or less than the stored number.

```
# define Python user-defined exceptions
class Error(Exception):
    """Base class for other exceptions"""
    pass


class ValueTooSmallError(Error):
    """Raised when the input value is too small"""
    pass


class ValueTooLargeError(Error):
    """Raised when the input value is too large"""
    pass


# you need to guess this number
number = 10

# user guesses a number until he/she gets it right
while True:
    try:
        i_num = int(input("Enter a number: "))
        if i_num < number:
            raise ValueTooSmallError
        elif i_num > number:
            raise ValueTooLargeError
        break
    except ValueTooSmallError:
        print("This value is too small, try again!")
        print()
    except ValueTooLargeError:
        print("This value is too large, try again!")
        print()

print("Congratulations! You guessed it correctly.")
```

Here is a sample run of this program.

```
Enter a number: 12
This value is too large, try again!

Enter a number: 0
This value is too small, try again!

Enter a number: 8
This value is too small, try again!

Enter a number: 10
Congratulations! You guessed it correctly.
```

We have defined a base class called `Error`.

The other two exceptions (`ValueTooSmallError` and `ValueTooLargeError`) that are actually raised by our program are derived from this class. This is the standard way to define user-defined exceptions in Python programming, but you are not limited to this way only.

Customizing Exception Classes

We can further customize this class to accept other arguments as per our needs.

To learn about customizing the Exception classes, you need to have the basic knowledge of Object-Oriented programming.

Visit Python Object Oriented Programming to start learning about Object-Oriented programming in Python.

Let's look at one example:

```
class SalaryNotInRangeError(Exception):
    """Exception raised for errors in the input salary.

    Attributes:
        salary -- input salary which caused the error
        message -- explanation of the error
    """

    def __init__(self, salary, message="Salary is not in (5000, 15000) range"):
        self.salary = salary
        self.message = message
        super().__init__(self.message)


salary = int(input("Enter salary amount: "))
if not 5000 < salary < 15000:
    raise SalaryNotInRangeError(salary)
```

Output

```
Enter salary amount: 2000
Traceback (most recent call last):
  File "<string>", line 17, in <module>
    raise SalaryNotInRangeError(salary)
__main__.SalaryNotInRangeError: Salary is not in (5000, 15000) range
```

Here, we have overridden the constructor of the `Exception` class to accept our own custom arguments `salary` and `message`. Then, the constructor of the parent `Exception` class is called manually with the `self.message` argument using `super()`.

The custom `self.salary` attribute is defined to be used later.

The inherited `__str__` method of the `Exception` class is then used to display the corresponding message when `SalaryNotInRangeError` is raised.

We can also customize the `__str__` method itself by overriding it.

```
class SalaryNotInRangeError(Exception):
    """Exception raised for errors in the input salary.

    Attributes:
        salary -- input salary which caused the error
        message -- explanation of the error
    """

    def __init__(self, salary, message="Salary is not in (5000, 15000) range"):
        self.salary = salary
        self.message = message
        super().__init__(self.message)

    def __str__(self):
        return f'{self.salary} -> {self.message}'


salary = int(input("Enter salary amount: "))
if not 5000 < salary < 15000:
    raise SalaryNotInRangeError(salary)
```

Output

```
Enter salary amount: 2000
Traceback (most recent call last):
  File "/home/bsoyuj/Desktop/Untitled-1.py", line 20, in <module>
    raise SalaryNotInRangeError(salary)
__main__.SalaryNotInRangeError: 2000 -> Salary is not in (5000, 15000) range
```

To learn more about how you can handle exceptions in Python, visit Python Exception Handling.