Python Custom Exceptions

In the previous tutorial, we learned about different built-in exceptions in Python and why it is important to handle exceptions.

However, sometimes we may need to create our own custom exceptions that serve our purpose.

Defining Custom Exceptions

In Python, we can define custom exceptions by creating a new class that is derived from the built-in `Exception` class.

Here's the syntax to define custom exceptions,

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

try:
   ...

except CustomError:
    ...
```

Here, `CustomError` is a user-defined error which inherits from the `Exception` class.

Note:

- 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 define their exceptions separately as `exceptions.py` or `errors.py` (generally but not always).

Example: Python User-Defined Exception

```
# define Python user-defined exceptions
class InvalidAgeException(Exception):
    "Raised when the input value is less than 18"
    pass

# you need to guess this number
number = 18

try:
    input_num = int(input("Enter a number: "))
    if input_num < number:
        raise InvalidAgeException
    else:
        print("Eligible to Vote")

except InvalidAgeException:
    print("Exception occurred: Invalid Age")
```

Output

If the user input *input_num* is greater than **18**,

```
Enter a number: 45
Eligible to Vote
```

If the user input *input_num* is smaller than **18**,

```
Enter a number: 14
Exception occurred: Invalid Age
```

In the above example, we have defined the custom exception `InvalidAgeException` by creating a new class that is derived from the built-in `Exception` class.

Here, when *input_num* is smaller than **18**, this code generates an exception.

When an exception occurs, the rest of the code inside the `try` block is skipped.

The `except` block catches the user-defined `InvalidAgeException` exception and statements inside the `except` block are executed.

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 learn about Object-Oriented programming in Python.

Let's see an 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.

Also Read:

- Python Exception Handling

Table of Contents