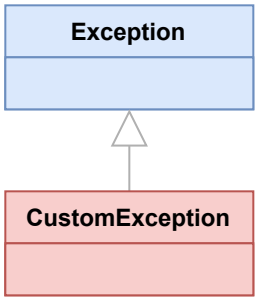
Python Custom Exception

Summary: in this tutorial, you'll learn how to define Python custom exception classes.

Introduction to the Python custom exception

To create a custom exception (https://www.pythontutorial.net/python-oop/python-exceptions/) class, you define a class (https://www.pythontutorial.net/python-oop/python-class/) that inherits (https://www.pythontutorial.net/python-oop/python-inheritance/) from the built-in `Exception` class or one of its subclasses such as `ValueError` class:



The following example defines a `CustomException` class that inherits from the `Exception` class:

```
class CustomException(Exception):
    """ my custom exception class """
```

> Note that the `CustomException` class has a docstring that behaves like a statement. Therefore, you don't need to add the `pass` statement to make the syntax valid.

To raise the CustomException, you use the `raise` (https://www.pythontutorial.net/python-oop/python-raise-exception/) statement. For example, the following uses the `raise` statement to raise the `CustomException`:

```
class CustomException(Exception):
    """ my custom exception class """


try:
    raise CustomException('This is my custom exception')
except CustomException as ex:
    print(ex)
```

Output:

```
This is my custom exception
```

Like standard exception classes, custom exceptions are also classes. Hence, you can add functionality to the custom exception classes like:

- Adding attributes and properties (https://www.pythontutorial.net/python-oop/python-properties/).
- Adding methods (https://www.pythontutorial.net/python-oop/python-methods/) e.g., log the exception, format the output, etc.
- Overriding the `__str__` and `__repr__` methods
- And doing anything else that you can do with regular classes.

In practice, you'll want to keep the custom exceptions organized by creating a custom exception hierarchy. The custom exception hierarchy allows you to catch exceptions at multiple levels, like the standard exception classes.

Python custom exception example

Suppose you need to develop a program that converts a temperature from Fahrenheit to Celsius.

The minimum and maximum values of a temperature in Fahrenheit are 32 and 212. If users enter a value that is not in this range, you want to raise a custom exception e.g., `FahrenheitError`.

Define the FahrenheitError custom exception class

The following defines the `FahrenheitError` exception class:

```
class FahrenheitError(Exception):
    min_f = 32
    max_f = 212

    def __init__(self, f, *args):
        super().__init__(args)
        self.f = f

    def __str__(self):
        return f'The {self.f} is not in a valid range {self.min_f, self.max_f}'
```

How it works.

- First, define the FahrenheitError class that inherits from the `Exception` class.
- Second, add two class attributes `min_f` and `max_f` that represent the minimum and maximum Fahrenheit values.
- Third, define the `__init__` (https://www.pythontutorial.net/python-oop/python-__init__/) method that accepts a Fahrenheit value (`f`) and a number of position arguments (`*args`). In the `__init__` method, call the `__init__` method of the base class. Also, assign the `f` argument to the `f` instance attribute.
- Finally, override the `__str__` (https://www.pythontutorial.net/python-oop/python-__str__/) method to return a custom string representation of the class instance.

Define the fahrenheit_to_celsius function

The following defines the `fahrenheit_to_celsius` function that accepts a temperature in Fahrenheit and returns a temperature in Celcius:

```
def fahrenheit_to_celsius(f: float) -> float:
    if f < FahrenheitError.min_f or f > FahrenheitError.max_f:
        raise FahrenheitError(f)

    return (f - 32) * 5 / 9
```

The `fahrenheit_to_celsius` function raises the `FahrenheitError` excpetion if the input temperature is not in the valid range. Otherwise, it converts the temperature from Fahrenheit to Celcius.

Create the main program

The following main program uses the `fahrenheit_to_celsius` function and the `FahrenheitError` custom exception class:

```
if __name__ == '__main__':
    f = input('Enter a temperature in Fahrenheit:')
    try:
        f = float(f)
    except ValueError as ex:
        print(ex)
    else:
        try:
            c = fahrenheit_to_celsius(float(f))
        except FahrenheitError as ex:
            print(ex)
        else:
            print(f'{f} Fahrenheit = {c:.4f} Celsius')
```

How it works.

First, prompt users for a temperature in Fahrenheit.

```
f = input('Enter a temperature in Fahrenheit:')
```

Second, convert the input value into a float (https://www.pythontutorial.net/advanced-python/python-float/) . If the `float()` cannot convert the input value, the program will raise a `ValueError` exception. In this case, it displays the error message from the `ValueError` exception:

```
try:
    f = float(f)
    # ...
except ValueError as ex:
    print(ex)
```

Third, convert the temperature to Celsius by calling the `fahrenheit_to_celsius` function and print the error message if the input value is not a valid `Fahrenheit` value:

```
try:
    c = fahrenheit_to_celsius(float(f))
except FahrenheitError as ex:
    print(ex)
else:
    print(f'{f} Fahrenheit = {c:.4f} Celsius')
```

Put it all together

```
class FahrenheitError(Exception):
    min_f = 32
    max_f = 212

    def __init__(self, f, *args):
        super().__init__(args)
```

```
        self.f = f

    def __str__(self):
        return f'The {self.f} is not in a valid range {self.min_f, self.max_f}'


def fahrenheit_to_celsius(f: float) -> float:
    if f < FahrenheitError.min_f or f > FahrenheitError.max_f:
        raise FahrenheitError(f)

    return (f - 32) * 5 / 9


if __name__ == '__main__':
    f = input('Enter a temperature in Fahrenheit:')
    try:
        f = float(f)
    except ValueError as ex:
        print(ex)
    else:
        try:
            c = fahrenheit_to_celsius(float(f))
        except FahrenheitError as ex:
            print(ex)
        else:
            print(f'{f} Fahrenheit = {c:.4f} Celsius')
```

Summary

- Subclass the `Exception` class or one of its subclasses to define a custom exception class.
- Create a exception class hierarchy to make the exception classes more organized and catch exceptions at multiple levels.