Python Static Methods

Summary: in this tutorial, you'll learn about Python static methods and how to use them to create a utility class.

Introduction to Python static methods

So far, you have learned about instance methods that are bound to a specific instance. It means that instance methods can access and modify the state of the bound object.

Also, you learned about class methods (https://www.pythontutorial.net/python-oop/python-class-methods/) that are bound to a class. The class methods can access and modify the class state.

Unlike instance methods, static methods aren't bound to an object. In other words, static methods cannot access and modify an object state.

In addition, Python doesn't implicitly pass the `cls` parameter (or the `self` parameter) to static methods. Therefore, static methods cannot access and modify the class's state.

In practice, you use static methods to define utility methods or group functions (https://www.pythontutorial.net/python-basics/python-functions/) that have some logical relationships in a class.

To define a static method, you use the `@staticmethod` decorator:

```
class className:
    @staticmethod
    def static_method_name(param_list):
        pass
```

To call a static method, you use this syntax:

```
className.static_method_name()
```

Python static methods vs class methods

Since static methods are quite similar to the class methods (https://www.pythontutorial.net/python-oop/python-class-methods/) , you can use the following to find the differences between them:

Class Methods	**Static Methods**
Python implicitly pass the `cls` argument to class methods.	Python doesn't implicitly pass the `cls` argument to static methods
Class methods **can** access and modify the class state.	Static methods **cannot** access or modify the class state.
Use `@classmethod` decorators to define class methods	Use `@staticmethod` decorators to define static methods.

Python static method examples

The following defines a class called `TemperatureConverter` that has static methods for converting temperatures between Celsius, Fahrenheit, and Kelvin:

```
class TemperatureConverter:
    KEVIN = 'K',
    FAHRENHEIT = 'F'
```

```python
    CELSIUS = 'C'

    @staticmethod
    def celsius_to_fahrenheit(c):
        return 9*c/5 + 32

    @staticmethod
    def fahrenheit_to_celsius(f):
        return 5*(f-32)/9

    @staticmethod
    def celsius_to_kelvin(c):
        return c + 273.15

    @staticmethod
    def kelvin_to_celsius(k):
        return k - 273.15

    @staticmethod
    def fahrenheit_to_kelvin(f):
        return 5*(f+459.67)/9

    @staticmethod
    def kelvin_to_fahrenheit(k):
        return 9*k/5 - 459.67

    @staticmethod
    def format(value, unit):
        symbol = ''
        if unit == TemperatureConverter.FAHRENHEIT:
            symbol = '°F'
        elif unit == TemperatureConverter.CELSIUS:
            symbol = '°C'
        elif unit == TemperatureConverter.KEVIN:
            symbol = '°K'

        return f'{value}{symbol}'
```

And to call the `TemperatureConverter` class, you use the following:

```
f = TemperatureConverter.celsius_to_fahrenheit(35)
print(TemperatureConverter.format(f, TemperatureConverter.FAHRENHEIT))
```

Summary

- Use static methods to define utility methods or group a logically related functions into a class.
- Use the `@staticmethod` decorator to define a static method.