# Class Methods and Static Methods

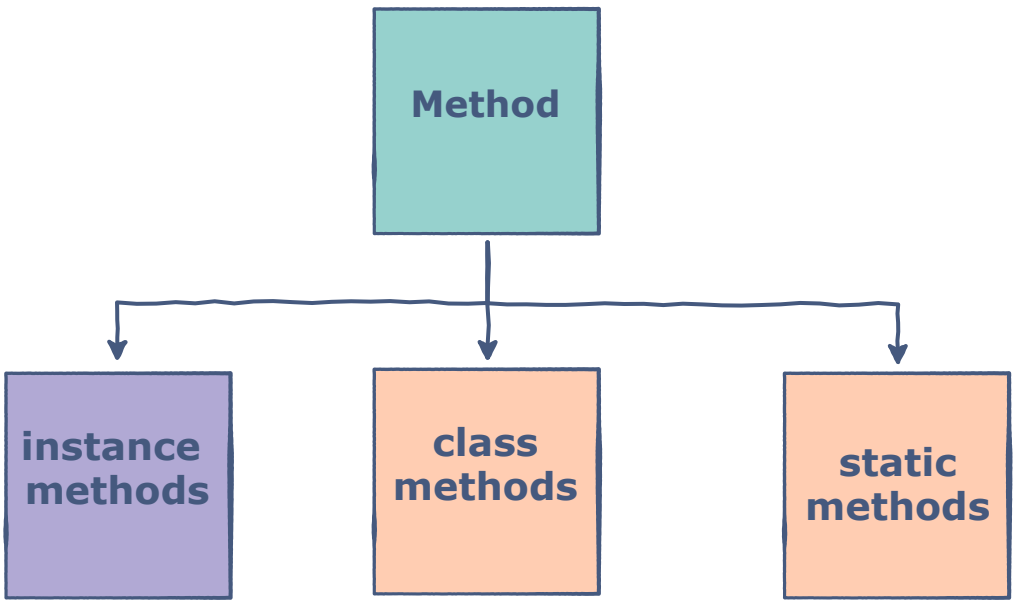
In this lesson, we will be discussing class methods and static methods

WE'LL COVER THE FOLLOWING



## Methods in a Python Class

In Python classes, we have three types of methods: instance methods, class methods, and static methods. In the previous lesson, we discussed instance methods. In this lesson, we will be focusing on **class methods** and **static methods**.




## Class Methods

**Class methods** work with class variables and are accessible using the class

**Class methods** work with class variables and are accessible using the class name rather than its object. Since all the class objects share the class variables, the class methods are used to access and modify class variables.

> Class methods are accessed using the class name and can be accessed without creating a class object.

## Syntax #

To declare a method as a class method, we use the decorator `@classmethod` and `cls` is used to refer to the class just like `self` is used to refer to the object of the class. You can use any other name instead of `cls`, but `cls` is used as per convention, and we will continue to use this convention in our course as well.

> **Note:** Just like instance methods, all class methods have **at least one** argument, `cls`.

```
class MyClass:
    classVariable = 'educative'

    @classmethod
    def demo(cls)
    return cls.classVariable
```

Below is a code implementation for class methods:

```
class Player:
    teamName = 'Liverpool'  # class variables

    def __init__(self, name):
        self.name = name  # creating instance variables

    @classmethod
    def getTeamName(cls):
        return cls.teamName


print(Player.getTeamName())
```

In **line** 7, we have used the `@classmethod` decorator to define `getTeamName` as a class method and in **line 12**, we are calling that method using the class name.

# Static Methods #

**Static methods** are methods that are usually limited to class only and not their objects. They have no direct relation to the class variables or instance variables. They are used as utility functions inside the class or when we do not want the inherited classes, which we will study later, to modify a method definition.

> Static methods can be accessed using the class name or the object name.

## Syntax #

To declare a method as a static method, we use the decorator `@staticmethod`. It does not use a reference to the object or class, so we do not have to use `self` or `cls`. We can pass as many arguments as we want and use this method to perform any function without interfering with the instance or class variables.

```
class MyClass:

    @staticmethod
    def demo()
    print("I am a static method")
```

Below is a code implementation for static methods:

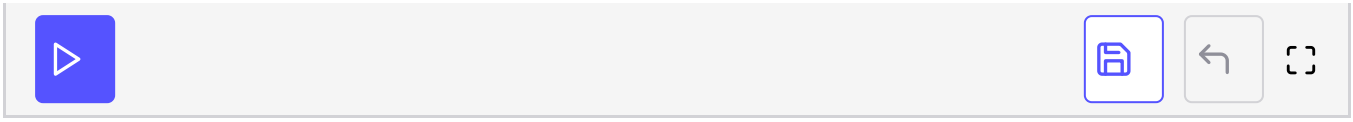
```
class Player:
    teamName = 'Liverpool'  # class variables

    def __init__(self, name):
        self.name = name  # creating instance variables

    @staticmethod
    def demo():
        print("I am a static method.")


p1 = Player('lol')
```

```
p1 = Player( 'lol' )
p1.demo()
Player.demo()
```



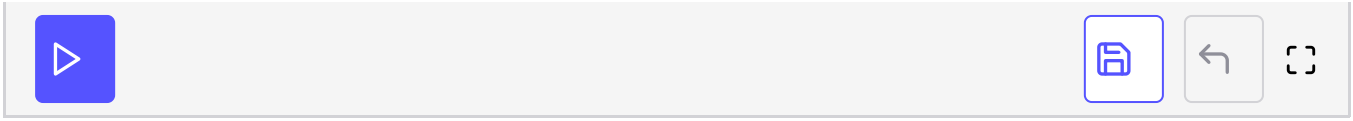
Static methods do not know anything about the state of the class, i.e., they cannot modify class attributes. The purpose of a static method is to use its parameters and produce a useful result.

Let's suppose that there is a class, `BodyInfo`, containing information about the physical attributes of a person. We can make a static method for calculating the BMI of any given weight and height:

```
class BodyInfo:

    @staticmethod
    def bmi(weight, height):
        return weight / (height**2)


weight = 75
height = 1.8
print(BodyInfo.bmi(weight, height))
```



This was all about class and static methods. In the next lesson, we will learn about access modifiers.