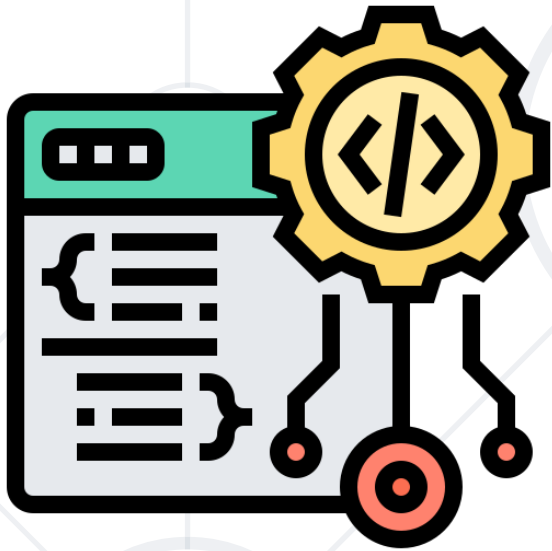


Class and Static Methods



SoftUni Team
Technical Trainers



SoftUni



Software University

<https://softuni.bg>

Table of Contents

1. Methods and Decorators
2. Static Methods
3. Class Methods



Have a Question?

sli.do

#python-advanced



`@staticmethod`

Static Methods

Static Methods



- It **knows nothing** about the **class** or **instance** it is called on
- It **cannot modify** object state or class state
- It could be put outside the class, but it is inside the class where it is **applicable**
- To turn a method into a static, we add a line with **@staticmethod** in front of the method header

Example: Static Methods

- To call a static method, we could use both the **instance**, or the **class**

```
class Person:
    def __init__(self, name):
        self.name = name

    @staticmethod
    def is_adult(age):
        return age >= 18

print(Person.is_adult(5))      # False
girl = Person("Amy")
print(girl.is_adult(20))      # True
```

It does not take a self parameter

Benefits

- Shows that a particular method is **independent** from everything else around it
- Often helps to **avoid accidental modifications** that go against the original design
- Communicates and enforces **developer intent** about the **class design**
- It is much **easier to test** since it is completely independent from the rest of the class



Problem: Calculator

- Follow the instructions in the lab document and create a class called **Calculator** with the following **static** methods
 - **add(*args)**
 - **multiply(*args)**
 - **divide(*args)**
 - **subtract(*args)**

Skeleton: Calculator

```
class Calculator:
    @staticmethod
    def add(*args):
        pass
    @staticmethod
    def multiply(*args):
        pass
    @staticmethod
    def divide(*args):
        pass
    @staticmethod
    def subtract(*args):
        pass
```



@classmethod

Class Methods

Class Methods

- It is **bound to the class** and not the object of the class
- It can **modify a class state** that would apply across all the instances of the class
- To turn a method into a class method, we add a line with **@classmethod** in front of the method header



Example: Class Methods

- We generally use class method to **create factory methods**

```
class Pizza:
    def __init__(self, ingredients):
        self.ingredients = ingredients

    @classmethod
    def pepperoni(cls):
        return cls(["tomato sauce", "parmesan", "pepperoni"])

    @classmethod
    def quattro_formaggi(cls):
        return cls(["mozzarella", "gorgonzola", "fontina", "parmigiano"])

first_pizza = Pizza.peperoni()
second_pizza = Pizza.quattro_formaggi()
```

We could create
different pizzas easily

Benefits

- Simply provide a **shortcut** for creating new instance objects
- Ensures **correct instance creation** of the derived class
- You could **easily follow** the Don't Repeat Yourself (DRY) principle using class methods



- Follow the instructions in the lab document and create a class called **Shop** with the following methods
 - **small_shop**(name: str, type: str)
 - **add_item**(item_name: str)
 - **remove_item**(item_name: str, amount: int)
 - **__repr__**()

```
class Store:
    def __init__(self, name, type, capacity):
        pass
    @classmethod
    def small_shop(cls, name, type):
        pass
    def add_item(self, item_name):
        pass
    def remove_item(self, item_name, count):
        pass
    def __repr__(self):
        pass
```

- Follow the instructions in the lab document and create a class called **Integer** with the following methods
 - **from_float**(value)
 - **from_roman**(value)
 - **from_string**(value)


```
class Integer:
    def __init__(self, value):
        self.value = value
    @classmethod
    def from_float(cls, float_value):
        pass
    @classmethod
    def from_roman(cls, value):
        pass
    @classmethod
    def from_string(cls, value):
        pass
```



Overriding Using Class Methods

Overriding Using Methods

```
class Person:
    min_age = 0
    max_age = 150

    def __init__(self, name, age):
        self.name = name
        self.age = age

    @staticmethod
    def __validate_age(value):
        if value < Person.min_age or \
            value > Person.max_age:
            raise ValueError()

    @property
    def age(self):
        return self.__age

    @age.setter
    def age(self, value):
        self.__validate_age(value)
        self.__age = value
```

```
class Employee(Person):
    min_age = 16
    max_age = 150

    def __init__(self, name, age):
        self.name = name
        self.age = age

    @staticmethod
    def __validate_age(value):
        if value < Employee.min_age or \
            value > Employee.max_age:
            raise ValueError()

    @property
    def age(self):
        return self.__age

    @age.setter
    def age(self, value):
        self.__validate_age(value)
        self.__age = value
```

Override all the methods below

Overriding Using a Class Method (1)

- If the methods **do not rely on state** and they **are the same**, they could be optimized using **@classmethod**

```
class Person:
    min_age = 0
    max_age = 150

    def __init__(self, name, age):
        self.name = name
        self.age = age

    @classmethod
    def __validate_age(cls, value):
        raise ValueError(f'{value} must be between '
                        f'{cls.min_age} and {cls.max_age}')

# __validate_age() takes the class attributes of class Person
```

Overriding Using a Class Method (2)

```
@property
def age(self):
    return self.__age

@age.setter
def age(self, value):
    self.__validate_age(value)
    self.__age = value

class Employee(Person):
    min_age = 16
    # __validate_age() takes the class attribute min_age of class Employee

    def __init__(self, name, age, salary):
        super().__init__(name, age) # when checking the age of the Employee
        self.salary = salary
```



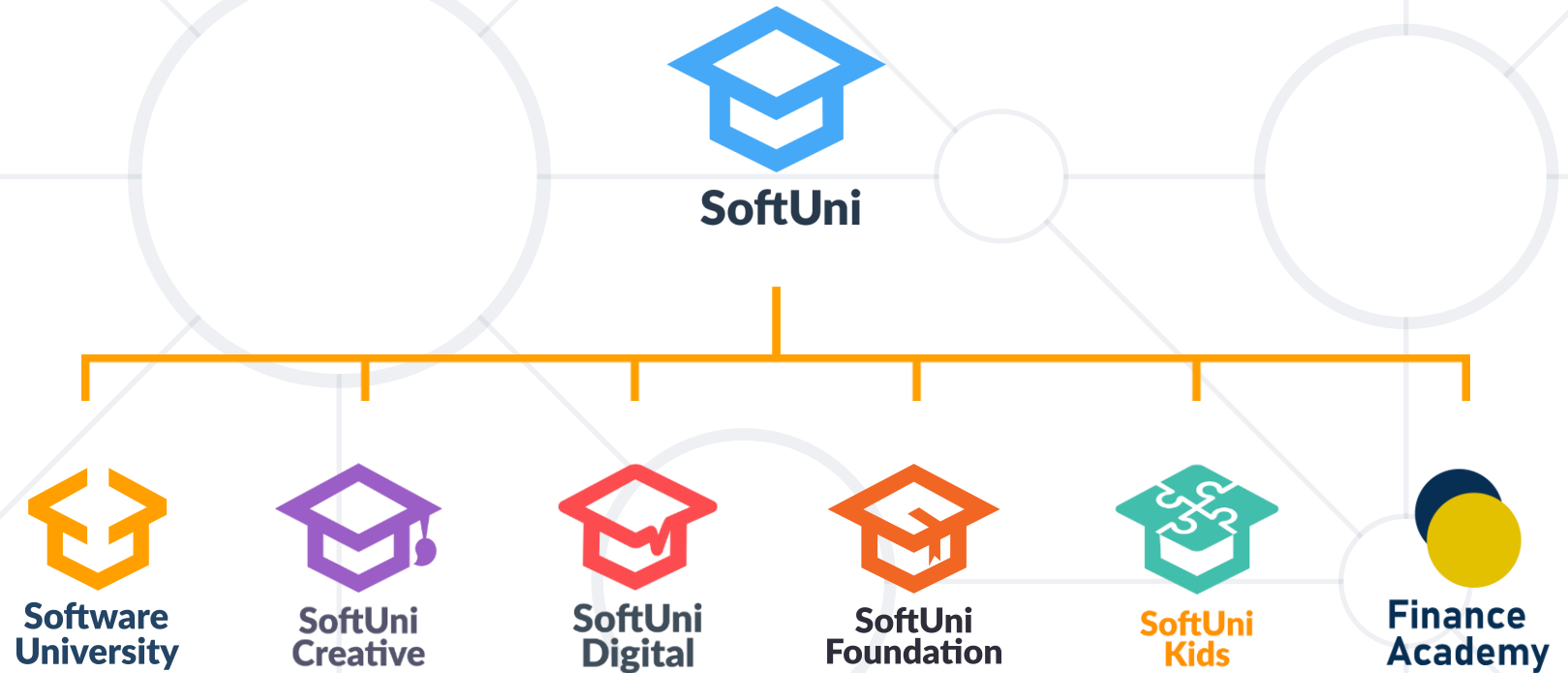
Practice

Live Exercise in Class (Lab)

- A **static method** is a method that **knows nothing** about the **class** or **instance** it is called on
- A **class method**, on the other hand, is **bound to the class** and not the object of the class



Questions?



SoftUni Diamond Partners

**SUPER
HOSTING
.BG**



**Coca-Cola HBC
Bulgaria**



POKERSTARS
POKER | CASINO | SPORTS
a Flutter International brand

INDEAVR
Serving the high achievers



AMBITIONED

 **DRAFT
KINGS**



**SOFTWARE
GROUP**

createX



Postbank

Решения за твоето утре

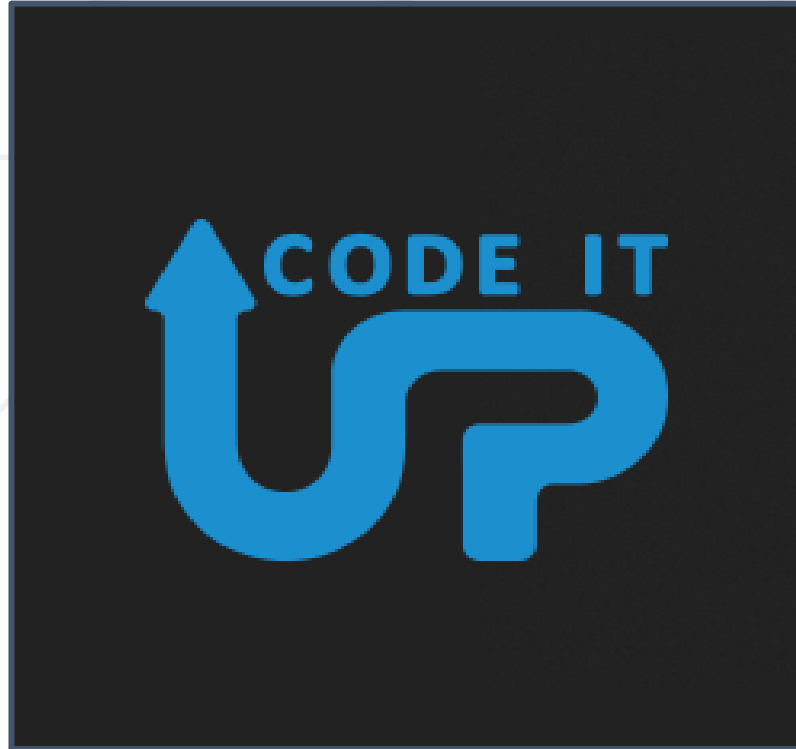


BOSCH

DXC
TECHNOLOGY



SmartIT



- Software University – High-Quality Education, Profession and Job for Software Developers

- softuni.bg, softuni.org

- Software University Foundation

- softuni.foundation

- Software University @ Facebook

- facebook.com/SoftwareUniversity

- Software University Forums

- forum.softuni.bg



- This course (slides, examples, demos, exercises, homework, documents, videos, and other assets) is **copyrighted content**
- Unauthorized copy, reproduction, or use is illegal
- © SoftUni – <https://about.softuni.bg>
- © Software University – <https://softuni.bg>

