cls_static_methods

December 3, 2022

Classmethod & staticmethod

In this section, we will learn about the **static method** and **class method**

[28]:
```
class Employee:
    # define two class variables
    num_empl = 0
    raise_amt = 1.05

    def __init__(self, first, last, pay):

        self.first = first
        self.last = last
        self.pay = int(pay)

        Employee.num_empl += 1

    # in a regular method in class which takes the
    # "self" instance as an argument

    def full_name(self):
        return "{} {}".format(self.first, self.last)

    def apply_raise(self):

        return int(self.pay * self.raise_amt)

# Let's make an instance for our base class
empl = Employee("Sad", "Sikei", 65000)
print(empl.full_name(), empl.apply_raise())
```

```
Sad Sikei 68250
```

Section 2 @Class method

[ ]:
```
By convention, class method uses the decrotor "@classmethod" and takes the "cls"␣
 ↪as
the first argument instead of the instance variable "self" in regular method. It␣
 ↪passes "cls" as the first
```

```
argument and additonal arguments for the class method. Basically the class␣
 ↪method changes the value of the
class variable "raise_amt" defined in the base "Employee" class with the new␣
 ↪argument amount.
```

For clarity regarding the **@classmethod**, we split the classmethod in an another cell from the base-class, in order to call the **classmethod**, which depends on the base class. we have to call **Employee** superclass in cell where we define the **classmethod**. Otherwise, we have attribute issue in the jupyter.

[32]:
```
class Employee(Employee):

    @classmethod
    def set_raise_amt(cls, amount):
        cls.raise_amt = amount


    # Now we add a new functionality to the class that parse
    # names and payment of employees separated by hyphens"-", i.e.␣
 ↪smith-Jahn-60000
    @classmethod
    def split_str(cls, empl_str):
        first, last, pay = empl_str.split("-")
        # the split string can be an imput arguments for your base class
        return cls(first, last, pay)

empl = Employee("Jahn", "Smith", 65000)
Employee.set_raise_amt(1.03)
print(empl.full_name() + " is getting " + str(empl.apply_raise()))
# make an instance of split_str
empl_2 = Employee.split_str("John-Kasi-56000")
print(empl_2.full_name() + "is gettting",  empl_2.apply_raise())
```

```
Jahn Smith is getting 66950
John Kasiis gettting 57680
```

Section 3 @Static method

In this part, we will learn about the **staticmehod**, which *Does* not pass any instance arguments like **self** or **cls** and has a decrotor **@staticmethod**. It behaves just like a regular function, and has its own arguments. But It has some connection with the base class as it is used to extend the functionality of the base class, as will be shown in the following.

[20]:
```

import datetime

class Employee(Employee):

    @staticmethod
```

```
    def is_workday(day):
        if day.weekday == 5 or day.weekday == 6:
            return False
        # if false, then it is weekday
        return True

# make an instance to our staticmethod
empl = Employee("Sad", "Sikei", 65000)
some_date = datetime.date(2012,12,12)
#empl.is_workday(some_date)
#sometimes it is mote direct to call the static method with the base class
#instead of using the instance:
Employee.is_workday(some_date)
```

[20]: True