OOPs | 03-Methods

Regular, Class and Static methods

Posted on October 18, 2021
Last updated on February 7, 2023

Methods in python

- **Function**
  - Function is block of code that is also called by its name(`independent`)
  - Function does not deal with Class and its instance concept.
- **Methods**
  - Method is called by its name, but it is `associated to an object` (`dependent`)

The following are various types of allowed methods:

1. Instance Methods
2. Class Methods
3. Static Methods

```
class MyClass:
        cat = 'Geometrical'
    # Instance method
    def method(self):
        return 'instance method called', self

    # class method
    @classmethod
    def classmethod(cls):
        return 'class method called', cls

    # static method
    @staticmethod
    def staticmethod():
        return 'static method called'
```

Instance methods

Inside method implementation if we are `using instance variables` then such type of methods are `called instance methods`. Inside instance method declaration,we have to pass self variable.

Here `self` is passed as an argument because `instance is passed as first argument`, without self it throws an error

```
class Vehicle:
    var = 10
    def car_type(self):
        self.var = 'Desire'
        print(self.var)

    def car_type2(self):
        print(self)

if __name__ == '__main__':
    car = Vehicle()
    car.car_type()
    # Alt way - Need to pass 'instance'
    Vehicle.car_type(car)

    # Using self as Variable --> Don't use
    Vehicle.car_type2('Hatch Bag')
```

Class method

Inside method implementation if we are using `only class variables` (static variables), then such type of methods we should declare as class method.

We can declare class method explicitly by using `@classmethod` decorator. For class method we should provide `cls variable` at the time of declaration

- `@classmethoad` and `cls` is used
- classmethod() methods are bound to a class rather than an object.
- Class methods can be called by both class and object

```python
class Pwskills1:
    def __init__(self, name, email):
        self.name = name
        self.email = email

    @classmethod
    def details(cls, name1, email1):
        """
        This internally assign the data to init
        Should have same no of argument as init
        """
        return cls(name1, email1)

    def student_details(self):
        print(self.name, self.email)

if __name__ == '__main__':
    # Pass data without using init but with cls method
    pw1 = Pwskills1.details("mohan" , "mohan@gamil.com")
    pw1.student_details()
    print(pw1.name)
```

- We generally use class method to create factory methods.
- Factory methods return class object (similar to a constructor) for different use cases.
- Good for private variable modification

```python
class Employee:
    # Class Variables
    raise_amount = 1.04
    no_of_emp = 0
    __pvt = 1.1

    def __init__(self, first, pay):
        # Regular variable
        self.first = first
        self.pay = pay

        # Don't use self - it will add as inst var
        # Use class name.
        Employee.no_of_emp += 1

    # Regular method
    def apply_raise(self):
        self.pay = int(self.pay * self.raise_amount)
        return self.pay

    # Class method
    @classmethod
    def set_raise_am(cls, amount):
        cls.raise_amount = amount
        # Good for private class var access
        cls.__pvt = amount

    @classmethod
    def from_str(cls, emp_str):
        first, pay = emp_str.split('-')
        # initialize
        return cls(first, pay)

if __name__ == '__main__':
    emp1 = Employee("Puja", 3000)

    # ClassMethod as --> alternate Constructor
    emp2 = Employee.from_str("puspa-9000")

    print(emp1.__dict__)
    # {'first': 'Puja', 'pay': 3000}
    print(emp2.__dict__)
```

```
# 'first': 'puspa', 'pay': '9000'}

print(Employee.raise_amount, emp1.raise_amount)
# 1.04 1.04

# changing ClassVariable value using ClassMethod
Employee.set_raise_am(2.2)
print(Employee.raise_amount, emp1.raise_amount, emp2.raise_amount
# 2.2 2.2 2.2

print(emp1.apply_raise())
# 6600
```

Static method

In general these methods are `general utility methods`. Inside these methods we won't use any instance or class variables. Here we `won't provide self or cls arguments` at the time of declaration.

- If `'cls' or 'self' is not used` in a method then we can declare it as StaticMethod
- In python there are two ways of defining a static method:
  - Using the `staticmethod()`
  - Using the `@staticmethod`

StaticMethod can be accesed directly by `Class name or Object`

```
class Pwskills:
    @staticmethod
    def static_method(list_mentor):
        print(list_mentor)

if __name__ == '__main__':
    pw1 = Pwskills()
    nlist = ["Amrit" , "Prasad"]
    # Accesed by Object
    pw1.static_method(nlist)
    # Accesed by Class name
    Pwskills.static_method(nlist)
```

- Static method can be accessed from
    - static method
    - instant method
    - class method

```python
class Pwskills:

    @staticmethod
    def static2():
        print('Static 2')

    # static - from static method
    @staticmethod
    def static1():
        print("Static 1")
        Pwskills.static2()

    # static - from cls method
    @classmethod
    def class_method(cls):
        cls.static2()

    # static - from instant method
    def mentor(self):
        self.static1()

if __name__ == '__main__':
    pw1 = Pwskills()
    pw1.mentor()
    Pwskills.class_method()
```

```python
class Employee:
    no_of_emp = 0

    def __init__(self, first):
        self.first = first
        Employee.no_of_emp += 1

    @staticmethod
    def is_workDay(day):
        if day.weekday() == 5 or day.weekday() == 6:
            return False
        return True

    def isAdult(age):
        return age > 18

# StaticMethod can be used without creating instance
import datetime

my_date = datetime.date(2016, 7, 11)
print(Employee.is_workDay(my_date))
print(Employee.isAdult(17))

# Alt way to define static
Employee.isAdult = staticmethod(Employee.isAdult)
print(Employee.isAdult(20))
```

Static variable and methods

- Static variable and methods are used when we want to define some behaviour or property `specific to the class`
- Which is something common for all the class objects.
- If you look closely, for a static method we don't provide the argument self because static methods don't operate on objects.

Class method vs Static Method

- Parameter
  - A class method takes `cls` as first parameter
  - A static method needs no specific parameters
- Modify cls state
  - A class method can access or modify class state

    - A static method can’t access or modify class state.
- In general
    - Static methods know nothing about class state.
        - They are utility type methods that take some parameters and work upon those parameters.
    - On the other hand class methods must have class as parameter.

Inner classes

Sometimes we can declare a class inside another class, such type of classes are called inner classes.

Without existing one type of object if there is no chance of existing another type of object, then we should go for inner classes.

```
class Outer:
    def __init__(self):
        print("outer class object creation")
    class Inner:
        def __init__(self):
            print("inner class object creation")
        def m1(self):
            print("inner class method")
o=Outer()
i=o.Inner()
i.m1()
```

.

.

.

.

.

.