Programming with Python

Konstantins Tarasjuks

Agenda for Today

Class naming

Class constructors

Class methods

Derived classes

Python naming

1.General

Avoid using names that are too general or too wordy. Strike a good balance between the two.

Bad: data_structure, my_list, info_map, dictionary_for_the_purpose_of_storing_data_representing_word_definitions

Good: user_profile, menu_options, word_definitions

Don't be a jackass and name things "O", "l", or "I"

When using CamelCase names, capitalize all letters of an abbreviation (e.g. HTTPServer)

2. Packages - folders, Modules - file name

Package names should be all lower case

When multiple words are needed, an underscore should separate them

It is usually preferable to stick to 1 word names

3. Classes

Class names should follow the UpperCaseCamelCase convention

Python's built-in classes, however are typically lowercase words

Exception classes should end in "Error"

Python Classes

Python is an object oriented programming language.

Almost everything in Python is an object, with its properties and methods.

A Class is like an object constructor, or a "blueprint" for creating objects.

Create a class named MyClass, with a property named x:

```
class MyClass:
    x = 5
```

Python Classes - __init__ - constructor

Create Object

Now we can use the class named MyClass to create objects:

Example

Create an object named p1, and print the value of x:

```
p1 = MyClass()
print(p1.x)
```

The __init__() Function

The examples above are classes and objects in their simplest form, and are not really useful in real life applications.

To understand the meaning of classes we have to understand the built-in __init__() function.

All classes have a function called __init__(), which is always executed when the class is being initiated.

Use the __init__() function to assign values to object properties, or other operations that are necessary to do when the object is being created:

Example

Create a class named Person, use the __init__() function to assign values for name and age:

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

p1 = Person("John", 36)

print(p1.name)
print(p1.age)
```

Python Classes - __init__ - constructor

Object Methods

Objects can also contain methods. Methods in objects are functions that belong to the object.

Let us create a method in the Person class:

Example

Insert a function that prints a greeting, and execute it on the p1 object:

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

  def myfunc(self):
    print("Hello my name is " + self.name)

p1 = Person("John", 36)
p1.myfunc()
```

Python Classes - self

The self parameter is a reference to the current instance of the class, and is used to access variables that belong to the class.

The self Parameter

The `self` parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class.

It does not have to be named `self` , you can call it whatever you like, but it has to be the first parameter of any function in the class:

Example

Use the words *mysillyobject* and *abc* instead of *self*:

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

  def myfunc(abc):
    print("Hello my name is " + abc.name)

p1 = Person("John", 36)
p1.myfunc()
```

Python Classes

This means to say, since `Person.greet` is a function object (attribute of class), `Person.greet` will be a method object.

```
class Person:
    "This is a person class"
    age = 10

    def greet(self):
        print('Hello')


# create a new object of Person class
harry = Person()

# Output: <function Person.greet>
print(Person.greet)

# Output: <bound method Person.greet of <__main__.Person object>>
print(harry.greet)

# Calling object's greet() method
# Output: Hello
harry.greet()
```

Python classes - when to use then?

When we need to create and work with objects - we create classes core OOP principle.

Most people working with big data are not using classes or OOP at all..

Python are not forcing us using classes like in Java

Types of Methods in Python

There are basically three types of methods in Python:

1. Instance Method
2. Class Method
3. Static Method

Instance methods

The purpose of instance methods is to set or get details about instances (objects), and that is why they're known as instance methods. They are the most common type of methods used in a Python class.

They have one default parameter- **self,** which points to an instance of the class. *Although you don't have to pass that every time.* You can change the name of this parameter but it is better to stick to the convention i.e **self.**

Any method you create inside a class is an instance method unless you specially specify Python otherwise. Let's see how to create an instance method:

```
class My_class:
  def instance_method(self):
    return "This is an instance method."
```

It's as simple as that!

In order to call an instance method, you've to create an object/instance of the class. With the help of this object, you can access any method of the class.

```
obj = My_class()
obj.instance_method()
```

```
'This is an instance method.'
```

Instance methods

When the instance method is called, Python replaces the **self** argument with the instance object, **obj**. That is why we should add one default parameter while defining the instance methods. Notice that when **instance_method()** is called, you don't have to pass self. Python does this for you.

Along with the default parameter self, you can add other parameters of your choice as well:

```
class My_class:

  def instance_method(self, a):
    return f"This is an instance method with a parameter a = {a}."
```

We have an additional parameter "a" here. Now let's create the object of the class and call this instance method:

```
obj = My_class()
obj.instance_method(10)
```

```
'This is an instance method with a parameter a = 10.'
```

Again you can see we have not passed 'self' as an argument, Python does that for us. But have to mention other arguments, in this case, it is just one. So we have passed 10 as the value of "a".

Instance methods

You can use "self" inside an instance method for accessing the other attributes and methods of the same class:

```
class My_class:

  def __init__(self, a, b):
    self.a = a
    self.b = b

  def instance_method(self):
    return f"This is the instance method and it can access the variables a = {self.a} and\
 b = {self.b} with the help of self."
```

Note that the __init__() method is a special type of method known as a **constructor**. This method is called when an object is created from the class and it allows the class to initialize the attributes of a class.

```
obj = My_class(2,4)
obj.instance_method()
```

```
'This is the instance method and it can access the variables a = 2 and b = 4 with the help of self.'
```

Class methods

The purpose of the class methods is to set or get the details (status) of the class. That is why they are known as class methods. They can't access or modify specific instance data. They are bound to the class instead of their objects. Two important things about class methods:

- In order to define a class method, you have to specify that it is a class method with the help of the @classmethod decorator
- Class methods also take one default parameter- **cls,** which points to the class. Again, this not mandatory to name the default parameter "cls". But it is always better to go with the conventions

Now let's look at how to create class methods:

```
class My_class:

  @classmethod
  def class_method(cls):
    return "This is a class method."
```

As simple as that!

As I said earlier, with the help of the instance of the class, you can access any method. So we'll create the instance of this My_class as well and try calling this class_method():

```
obj = My_class()
obj.class_method()
```

```
'This is a class method.'
```

Class methods

This works too! We can access the class methods with the help of a class instance/object. But we can access the class methods directly without creating an instance or object of the class. Let's see how:

```
My_class.class_method()
```

```
'This is a class method.'
```

Without creating an instance of the class, you can call the class method with – **Class_name.Method_name().**

But this is not possible with instance methods where we have to create an instance of the class in order to call instance methods. Let's see what happens when we try to call the instance method directly:

```
My_class.instance_method()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-7-ea37e805ec52> in <module>()
----> 1 My_class.instance_method()

TypeError: instance_method() missing 1 required positional argument: 'self'
```

We got an error stating missing one positional argument – "self". And it is obvious because instance methods accept an instance of the class as the default parameter. And you are not providing any instance as an argument. Though this can be bypassing the object name as the argument:

```
My_class.instance_method(obj)
```

```
'This is an instance method.'
```

Static methods

Static methods cannot access the class data. In other words, they do not need to access the class data. They are self-sufficient and can work on their own. Since they are not attached to any class attribute, they cannot get or set the instance state or class state.

In order to define a static method, we can use the @staticmethod decorator (in a similar way we used @classmethod decorator). Unlike instance methods and class methods, we do not need to pass any special or default parameters. Let's look at the implementation:

```
class My_class:

  @staticmethod
  def static_method():
    return "This is a static method."
```

And done!

Notice that we do not have any default parameter in this case. Now how do we call static methods? Again, we can call them using object/instance of the class as:

```
obj = My_class()
obj.static_method()
```

```
'This is a static method.'
```

And we can call static methods directly, without creating an object/instance of the class:

```
My_class.static_method()
```

```
'This is a static method.'
```

You can notice the output is the same using both ways of calling static methods.

When to Use Which Python Method?

The instance methods are the most commonly used methods. Even then there is difficulty in knowing when you can use class methods or static methods. The following explanation will satiate your curiosity:

Class Method – The most common use of the class methods is for creating **factory methods**. Factory methods are those methods that return a class object (like a constructor) for different use cases. Let's understand this using the given example:

```
from datetime import date

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

# a class method to create a Dog object with birth year.
  @classmethod
  def Year(cls, name, year):
    return cls(name, date.today().year - year)
```

Here as you can see, the class method is modifying the status of the class. If we have the year of birth of any dog instead of the age, then with the help of this method we can modify the attributes of the class.

When to Use Which Python Method?

Here as you can see, the class method is modifying the status of the class. If we have the year of birth of any dog instead of the age, then with the help of this method we can modify the attributes of the class.

Let's check this:

```
Dog1 = Dog('Bruno', 1)
Dog1.name , Dog1.age
```

```
('Bruno', 1)
```

Here we have constructed an object of the class. This takes two parameters name and age. On printing the attributes of the class we get Bruno and 1 as output, which was the values we provided.

```
Dog2 = Dog.Year('Dobby', 2017)
Dog2.name , Dog2.age
```

```
('Dobby', 3)
```

Here, we have constructed another object of the class Dog using the class method **Year()**. The Year() method takes Person class as the first parameter *cls* and returns the constructor by calling **cls(name, date.today().year – year)**, which is equivalent to **Dog(name, date.today().year – year).**
As you can see in printing the attributes name and age we got the name that we provided and the age converted from the year of birth using the class method.

When to Use Which Python Method?

Static Method – They are used for creating utility functions. For accomplishing routine programming tasks we use utility functions. A simple example could be:

```
class Calculator:

  @staticmethod
  def add(x, y):
    return x + y

print('The sum is:', Calculator.add(15, 10))
```

```
The sum is: 25
```

You can see that the use-case of this static method is very clear, adding the two numbers given as parameters. Whenever you have to add two numbers, you can call this method directly without worrying about object construction.

Python Inheritance

Inheritance allows us to define a class that inherits all the methods and properties from another class.

Parent class is the class being inherited from, also called base class.

Child class is the class that inherits from another class, also called derived class.

Python Inheritance Syntax

```
class BaseClass:
  Body of base class
class DerivedClass(BaseClass):
  Body of derived class
```

Derived class inherits features from the base class where new features can be added to it.

This results in re-usability of code.

Python Multiple Inheritance

A class can be derived from more than one base class in Python, similar to C++. This is called multiple inheritance. In multiple inheritance, the features of all the base classes are inherited into the derived class. The syntax for multiple inheritance is similar to single inheritance.

Example

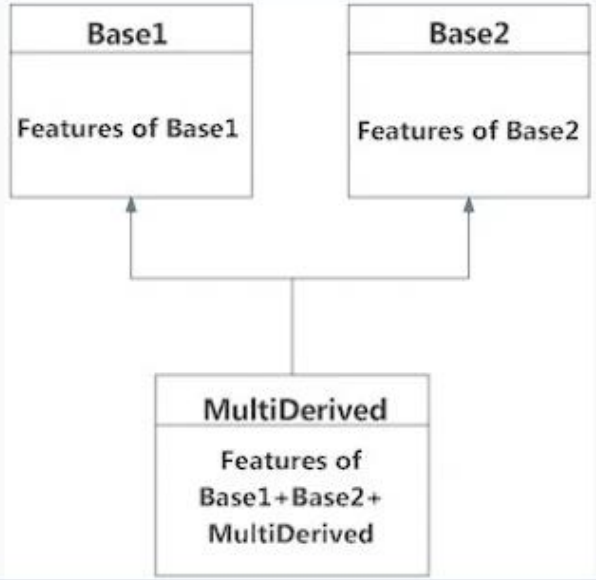
```
class Base1:
    pass

class Base2:
    pass

class MultiDerived(Base1, Base2):
    pass
```

Python Multiple Inheritance

Here, the `MultiDerived` class is derived from `Base1` and `Base2` classes.



Multiple Inheritance in Python

The `MultiDerived` class inherits from both `Base1` and `Base2` classes.

Python Multilevel Inheritance

We can also inherit from a derived class. This is called multilevel inheritance. It can be of any depth in Python. In multilevel inheritance, features of the base class and the derived class are inherited into the new derived class.

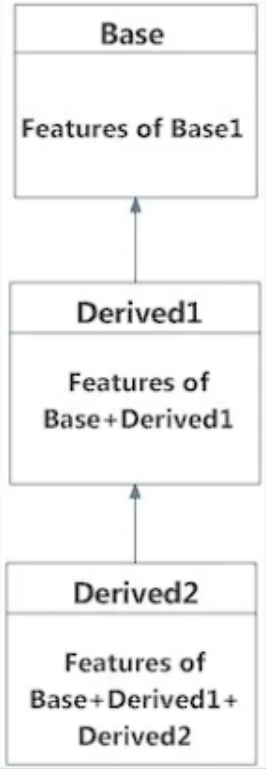
```
class Base:
    pass

class Derived1(Base):
    pass

class Derived2(Derived1):
    pass
```

Python Multilevel Inheritance

Here, the `Derived1` class is derived from the `Base` class, and the `Derived2` class is derived from the `Derived1` class.



Multilevel Inheritance in Python

Python Multilevel Inheritance

We can also inherit from a derived class. This is called multilevel inheritance. It can be of any depth in Python.

In multilevel inheritance, features of the base class and the derived class are inherited into the new derived class.

An example with corresponding visualization is given below.

```
class Base:
    pass

class Derived1(Base):
    pass

class Derived2(Derived1):
    pass
```

Class method vs Static Method

The difference between the Class method and the static method is:

- A class method takes cls as the first parameter while a static method needs no specific parameters.
- A class method can access or modify the class state while a static method can't access or modify it.
- In general, static methods know nothing about the 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 a parameter.
- We use @classmethod decorator in python to create a class method and we use @staticmethod decorator to create a static method in python.

When to use the class or static method?

- We generally use the class method to create factory methods. Factory methods return class objects (similar to a constructor) for different use cases.
- We generally use static methods to create utility functions.

How to define a class method and a static method?

To define a class method in python, we use @classmethod decorator, and to define a static method we use @staticmethod decorator.

Let us look at an example to understand the difference between both of them. Let us say we want to create a class Person. Now, python doesn't support method overloading like C++ or Java so we use class methods to create factory methods. In the below example we use a class method to create a person object from birth year.

As explained above we use static methods to create utility functions. In the below example we use a static method to check if a person is an adult or not.

CLS vs SELF

self

`self` in Python holds the reference of the current working instance. This reference is passed as the first argument to every instance methods of the class by Python itself.

```
class MyClass:
    def what_is_self(self):
        print(self)

MyClass().what_is_self()  #outputs <__main__.MyClass object at 0x7fef4c820d00>
```

cls

`cls` in Python holds the reference of the class. It is passed as the first argument to every class methods (methods with `@classmethod` decorator) by Python itself.

```
class MyClass:
    @classmethod
    def what_is_cls(cls):
        print(cls)

MyClass().what_is_cls()  #outputs <class '__main__.MyClass'>
```

It is important to note that self and cls are not reserved keywords. They are just naming conventions in Python.
It means we can rename them to whatever we want, the underlying significance will remain the same.

Instance is an object that belongs to a class.

cls as an argument indicating that the method points to the class instead of the object instance.

CLS vs SELF

Since `self` refers to the instance and `cls` refers to the class, they differ in terms of scope and accessibilty.

self	cls
self holds the reference of the current working instance.	cls holds the reference of the current working class.
self also holds the reference of the class as `self.__class__`	cls doesn't hold any reference to any instances of the class.
self has access to both instance and class data members of the current working instance.	Using `cls` we can access only the class data members of the current class.
Variables initialized using self get declared in the instance's scope.	Variable initialized using cls get declared in the class's scope.

Class method vs Static Method

```python
# Python program to demonstrate
# use of class method and static method.
from datetime import date


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

    # a class method to create a Person object by birth year.
    @classmethod
    def fromBirthYear(cls, name, year):
        return cls(name, date.today().year - year)

    # a static method to check if a Person is adult or not.
    @staticmethod
    def isAdult(age):
        return age > 18


person1 = Person('mayank', 21)
person2 = Person.fromBirthYear('mayank', 1996)

print(person1.age)
print(person2.age)

# print the result
print(Person.isAdult(22))
```

Output:

```
21

25

True
```

Tabs or Spaces? - PEP8

- **Spaces are the preferred indentation method.**
- *Tabs should be used solely to remain consistent with code that is already indented with tabs.*
- *Python disallows mixing tabs and spaces for indentation.*

Task 1 - Calculator

Create calculator app - without class definition

1. function for +
2. function for -
3. function for *
4. function for /
5. main function where you call previous functions and provide a and b variables

Let’s surround all of this with class - try to run it

Task 2 - Calculator with class

Create another class where we launch the first class

Let’s import calculator class and then launch from another class

Add @staticmethod to our calculator methods

Task 3 - Employer salary and age

Create class employer which will have methods for salary and age calculation

Salary salary rate - variable

__init__ - name (string),age (int) , tax % (int)

Method – check salary – displays current salary – this method should calculate salary and only then print the salary rate

@classmethod - name (string),age (int), salary

Salary method - Current salary rate- taxes (tax % is dependant on employer (as taxes are dependant on employer kids and etc.)

Task 4 - Derived class

```
# parent class
class Person(object):

    # __init__ is known as the constructor
    def __init__(self, name, idnumber):
        self.name = name
        self.idnumber = idnumber

    def display(self):
        print(self.name)
        print(self.idnumber)

# child class


class Employee(Person):
    def __init__(self, name, idnumber, salary, post):
        self.salary = salary
        self.post = post

        # invoking the __init__ of the parent class
        Person.__init__(self, name, idnumber)


# creation of an object variable or an instance
a = Employee('Rahul', 886012, 200000, "Intern")

# calling a function of the class Person using its instance
a.display()
```

What will be printed out?

```
class Dog:
    tricks = []  # mistaken use of a class variable

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

    def add_trick(self, trick):
        self.tricks.append(trick)


if __name__ == '__main__':
    d = Dog('Fido')
    e = Dog('Buddy')
    d.add_trick('roll over')
    e.add_trick('play dead')
    print(e.tricks)
```

What will be printed out?

```
class Dog:

    def __init__(self, name):
        self.name = name
        self.tricks = []    # creates a new empty list for each dog

    def add_trick(self, trick):
        self.tricks.append(trick)


if __name__ == '__main__':
    d = Dog('Fido')
    e = Dog('Buddy')
    d.add_trick('roll over')
    e.add_trick('play dead')
    print(e.tricks)
```

Reference

Static vs Class methods - https://www.geeksforgeeks.org/class-method-vs-static-method-python/

Self vs CLS - https://pencilprogrammer.com/python-self-vs-cls/#:~:text=cls%20in%20Python%20holds%20the,print(cls)%20MyClass().

Python classes - https://docs.python.org/3/tutorial/classes.html#:~:text=Python%20classes%20provide%20all%20the,class%20with%20the%20same%20name.

Derived classes - https://www.programiz.com/python-programming/multiple-inheritance#:~:text=A%20class%20can%20be%20derived,is%20similar%20to%20single%20inheritance.


THANK YOU FOR YOUR ATTENTION!
...YOU ALWAYS HAVE MY ATTENTION.
makeameme.org