OOP in python

January 18, 2022

1 OOP in Python

```
[1]: mylist = [1,2,3]
```

```
[2]: type(mylist)
```

```
[2]: list
```

```
[3]: myset = set()
```

```
[4]: type(myset)
```

```
[4]: set
```

```
[5]: class Sample():
         pass
```

```
[6]: my_sample = Sample()
```

```
[7]: type(my_sample)
```

```
[7]: __main__.Sample
```

```
[8]: class Dog():
         def __init__(self,breed):
             self.breed = breed
```

```
[9]: my_dog = Dog()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_1396/3226080032.py in <module>
----> 1 my_dog = Dog()

TypeError: __init__() missing 1 required positional argument: 'breed'
```

```
[10]: my_dog = Dog(breed = 'Lab')
```

[11]:
```
type(my_dog)
```

[11]:
```
__main__.Dog
```

[12]:
```
my_dog.breed
```

[12]:
```
'Lab'
```

[13]:
```
class Dog():
    def __init__(self,breed,name,spots):
        self.breed = breed
        self.name = name
        self.spots = spots
```

[14]:
```
my_dog = Dog(breed = 'Lab', name = 'Sammy', spots = False)
```

[16]:
```
my_dog.breed
```

[16]:
```
'Lab'
```

[17]:
```
my_dog.name
```

[17]:
```
'Sammy'
```

[18]:
```
my_dog.spots
```

[18]:
```
False
```

2 Class object attributes and methods:

[19]:
```
class Dog():
    species = 'mammal'
    def __init__(self,breed,name,spots):
        self.breed = breed
        self.name = name
        self.spots = spots
```

[20]:
```
my_dog = Dog(breed = 'Lab',name = 'Sam', spots = False)
```

[21]:
```
type(my_dog)
```

[21]:
```
__main__.Dog
```

[22]:
```
my_dog.species
```

[22]:
```
'mammal'
```

```
[23]: my_dog.name
```

```
[23]: 'Sam'
```

```
[24]: class Dog():
          species = 'mammal'
          def __init__(self,breed,name,spots):
              self.breed = breed
              self.name = name
              self.spots = spots
          def bark(self):
              print('Woof!')
```

```
[25]: my_dog = Dog('lab','sam',False)
```

```
[26]: my_dog.bark()
```

```
Woof!
```

```
[27]: my_dog.name
```

```
[27]: 'sam'
```

```
[35]: class Dog():
          species = 'mammal' #Remains same for any instance of a class
          def __init__(self,breed,name):
              self.breed = breed
              self.name = name
          def bark(self):
              print('Woof! My name is {}'.format(self.name))
```

```
[36]: my_dog = Dog('Lab','Sam')
```

```
[37]: my_dog.bark()
```

```
Woof! My name is Sam
```

```
[38]: class Dog():
          species = 'mammal'
          def __init__(self,breed,name):
              self.breed = breed
              self.name = name
          def bark(self,number):
              print('Woof! My name is {} and my number is {}'.format(self.
       ↪name,number))
```

```
[40]: my_dog = Dog('Lab','Sam')
```

```
[41]: my_dog.bark(10)
```

```
Woof! My name is Sam and my number is 10
```

```
[2]: class Circle():
         pi = 3.14
         def __init__(self,radius):
             self.radius = radius
         def get_circumference(self):
             return self.radius * self.pi * 2
```

```
[3]: mycircle = Circle(10)
```

```
[4]: mycircle.get_circumference()
```

```
[4]: 62.800000000000004
```

```
[6]: mycircle.pi
```

```
[6]: 3.14
```

```
[7]: mycircle.radius
```

```
[7]: 10
```

```
[8]: # An attribute doesn't need to be defined through a parameter call

     class Circle():
         pi = 3.14
         def __init__(self,radius = 10):
             self.radius = radius
         def get_circumference(self):
             return self.radius * self.pi * 2
```

```
[9]: mycircle = Circle()
```

```
[10]: mycircle.get_circumference()
```

```
[10]: 62.800000000000004
```

```
[13]:  # Because pi is a class object attribute can be called with name of the class.
      Circle.pi
```

```
[13]: 3.14
```

3 Inheritance:

```
[14]: # Its a way to create a derived class using the base class.
```

[15]:
```
class Animal():
    def __init__(self):
        print("ANIMAL CREATED")
    def who_am_i(self):
        print("I am animal")
    def eat(self):
        print("I am eating")
```

[16]:
```
myanimal = Animal()
```

```
ANIMAL CREATED
```

[17]:
```
myanimal.who_am_i()
```

```
I am animal
```

[18]:
```
myanimal.eat()
```

```
I am eating
```

[19]:
```
class Dog(Animal):
    def __init__(self):
        Animal.__init__(self)
        print("Dog created")
```

[20]:
```
my_dog = Dog()
```

```
ANIMAL CREATED
Dog created
```

[21]:
```
my_dog.eat() # eventhough eat() is not present in Dog(), it derives from Animal␣
 ↪class
```

```
I am eating
```

[22]:
```
class Dog(Animal):
    def __init__(self):
        Animal.__init__(self)
        print("Dog created")
    def who_am_i(self):
        print("I am a dog") # Modifying the methos present in the base class
```

[23]:
```
my_dog = Dog()
```

```
ANIMAL CREATED
Dog created
```

[24]:
```
my_dog.who_am_i()
```

```
I am a dog
```

4 Polymorphism

[27]:
```
# Here different objects classes shares the same method name
class Dog():
    def __init__(self,name):
        self.name = name
    def speak(self):
        return self.name + ' says woof'
```

[28]:
```
class Cat():
    def __init__(self,name):
        self.name = name
    def speak(self):
        return self.name + ' says meow'
```

[29]:
```
niko = Dog('niko')
felix = Cat('felix')
```

[30]:
```
print(niko.speak())
```

```
niko says woof
```

[31]:
```
print(felix.speak())
```

```
felix says meow
```

[32]:
```
for pet in [niko,felix]:
    print(type(pet))
    print(pet.speak())
```

```
<class '__main__.Dog'>
niko says woof
<class '__main__.Cat'>
felix says meow
```

[36]:
```
def pet_speak(pet):
    print(pet.speak())
```

[37]:
```
pet.speak()
```

[37]:
```
'felix says meow'
```

# Abstract Class

[38]:
```
# never expect abstract class to be instantiated. Basically works as a base␣
↪class.
```

[39]:
```
class Animal():
    def __init__(self,name):
        self.name = name
    def speak(self):
        raise NotImplementedError('Subclass must implement this abstract␣
 ↪method')
```

[40]:
```
myanimal = Animal('fred')
```

[41]:
```
myanimal.speak()
```

```
---------------------------------------------------------------------------
NotImplementedError                       Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_5792/2029186425.py in <module>
----> 1 myanimal.speak()

~\AppData\Local\Temp/ipykernel_5792/3261593226.py in speak(self)
      3         self.name = name
      4     def speak(self):
----> 5         raise NotImplementedError('Subclass must implement this abstract␣
 ↪method')

NotImplementedError: Subclass must implement this abstract method
```

[44]:
```
class Dog():
    def __init__(self,name):
        self.name = name
    def speak(self):
        return self.name + ' says woof'
```

[45]:
```
class Cat():
    def __init__(self,name):
        self.name = name
    def speak(self):
        return self.name + ' says meow'
```

[46]:
```
fido = Dog('fido')
isis = Cat('isis')
```

[47]:
```
print(fido.speak())
print(isis.speak())
```

```
fido says woof
isis says meow
```

[ ]:
```

```