26.OOPs

June 10, 2020

Object Oriented Programming

`class` is collection of `methods` and `object` is collection of `attributes/features/data`

[35]:
```
class Person(object):
    """A simple class.""" # doc-string
    __species = "Homo Sapiens" # class attribute / shared amond all object of␣
↪Person Class
    def __init__(self, name): # special method to initlize attribues in object␣
↪space at creation time
        """This is the initializer. It's a special method called constructor."""
        self.name = name
    def __str__(self): # special method or string representation of object
        return self.name
    def rename(self, new_name): # it's a instance method because it will change␣
↪attributes of object
        self.name = new_name
        print(f"Now my name is {self.name}")
    def get_species(self):
        print(f"You belong to {self.__species} Species.")
```

[36]:
```
p1 = Person('sachin yadav') # __init__ ?
p2 = Person("rajat goyal")
```

[37]:
```
print(p1) # __str__ ?
```

```
sachin yadav
```

[38]:
```
print(p2) # p2 -> Person.__str__(p2) -> p2 -> self
```

```
rajat goyal
```

[40]:
```
p1.get_species()
```

```
You belong to Homo Sapiens Species.
```

[41]:
```
p2.get_species()
```

```
You belong to Homo Sapiens Species.
```

```
[42]: Person.__species # data hiding
```

```
                                                                          ␣
↪---------------------------------------------------------------------------

        AttributeError                            Traceback (most recent call␣
↪last)

        <ipython-input-42-a9c8f62a744c> in <module>
    ----> 1 Person.__species # data hiding


        AttributeError: type object 'Person' has no attribute '__species'
```

```
[10]: p1.rename("Sachin")
```

```
Now my name is Sachin
```

```
[11]: p2.rename("Rajat")
```

```
Now my name is Rajat
```

```
[12]: p1.name
```

```
[12]: 'Sachin'
```

```
[13]: p2.name
```

```
[13]: 'Rajat'
```

```
[14]: p1.name = 'yahoo'
```

```
[15]: p2.name = 'google'
```

```
[16]: print(p1)
```

```
yahoo
```

```
[17]: print(p2)
```

```
google
```

```
[18]: p1.species
```

```
[18]: 'Homo Sapiens'
```

[32]:
```
p1.species = 'a selfish kind' # we are creating a new instance varaible
```

[33]:
```
p1.species
```

[33]:
```
'a selfish kind'
```

[34]:
```
p2.species
```

[34]:
```
'Homo Sapiens'
```

[ ]:
```

```

[ ]:
```
#p1.name = 'some new value'-> variable

#p1.rename = 'overide' # setter property

#p1.rename('new value') # setter method
```

[27]:
```
class Product:
    inventory = {
        'iphone': 120000,
        'oneplus': 65000,
        'samsung': 80000
    }
    discount = {
        'male': .1,
        'female': .25,
    }
    def __init__(self, cusname, gender):
        self.name = cusname
        self.gender = gender.lower()
    def get_product_price(self, product):
        if product in Product.inventory:
            price = Product.inventory[product]
            final_price = price - (price*Product.discount[self.gender])
            print("Price is: ", final_price)
        else:
            print("Not Available")
```

[28]:
```
c1 = Product('sachin', 'male')
c2 = Product('tanvi', 'female')
```

[29]:
```
c1.get_product_price('oneplus')
```

```
Price is:  58500.0
```

```
[30]: c2.get_product_price('oneplus')
```

```
Price is:  48750.0
```

```
[44]: class A:
          hello = 'hi' # class variable
```

```
[46]: a = A()
      b = A()
```

```
[47]: a.hello
```

```
[47]: 'hi'
```

```
[48]: b.hello
```

```
[48]: 'hi'
```

```
[49]: A.hello
```

```
[49]: 'hi'
```

```
[50]: A.hello = 'bye bye'
```

```
[51]: a.hello
```

```
[51]: 'bye bye'
```

```
[52]: b.hello
```

```
[52]: 'bye bye'
```

```
[53]: import inspect
```

```
[61]: class A:
          def hello(self):
              print(id(A.hello), 'hello world', id(self.hello))
```

```
[62]: a = A()
```

```
[63]: A.hello(a)
```

```
2075953663288 hello world 2075917411912
```

```
[64]: a.hello()
```

```
2075953663288 hello world 2075917344520
```

```
[65]: a.hello() # creating a new bounded method with a object
```

```
2075953663288 hello world 2075917343944
```

```
[66]: a.hello = a.hello
```

```
[67]: a.hello()
```

```
2075953663288 hello world 2075917343240
```

```
[68]: a.hello()
```

```
2075953663288 hello world 2075917343240
```

```
[69]: inspect.isfunction(A.hello)
```

```
[69]: True
```

```
[70]: inspect.ismethod(A.hello)
```

```
[70]: False
```

```
[71]: inspect.isfunction(a.hello)
```

```
[71]: False
```

```
[72]: inspect.ismethod(a.hello) # method
```

```
[72]: True
```

Class Method, Static Method, Instance Method

```
[74]: class A:
          msg = 'I am class variable' # class property or class attribute
          def __init__(self): # instance method
              self.name = 'I am instance variable'
          def get_msg(self): # instance method
              print("Message: ", A.msg) # is should be class methods
          def get_name(self): # instance method
              print("Name: ", self.name)
          def info(self): # instance method
              print("Hello This is A normal Class to understand static, class and␣
      ↪instance methods")
              # neigther we need class scope nor object scope in info method so it is␣
      ↪should be static method
```

```
[80]: A.info()
```

```
      ␣
↪---------------------------------------------------------------------------

       TypeError                                 Traceback (most recent call␣
↪last)

       <ipython-input-80-3bd34887fbf7> in <module>
   ----> 1 A.info()


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

[79]:
```
A.get_msg()
```

```
      ␣
↪---------------------------------------------------------------------------

       TypeError                                 Traceback (most recent call␣
↪last)

       <ipython-input-79-91d6b5f15cb0> in <module>
   ----> 1 A.get_msg()


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

[75]:
```
a = A()
```

[76]:
```
a.get_msg()
```

```
Message:  I am class variable
```

[77]:
```
a.get_name()
```

```
Name:  I am instance variable
```

[78]:
```
a.info()
```

```
Hello This is A normal Class to understand static, class and instance methods
```

which `methods` in above class are not using `instance properties` or self or object properties means which can be called without object

[ ]:
```
dec = decrotor(func)
@dec
def func()
```

[81]:
```
classmethod
```

[81]:
```
classmethod
```

[82]:
```
staticmethod
```

[82]:
```
staticmethod
```

[83]:
```
class A:
    msg = 'I am class variable' # class property or class attribute
    def __init__(self): # instance method
        self.name = 'I am instance variable'
    @classmethod
    def get_msg(cls): # class method
        print(cls)
        print("Message: ", cls.msg) # is should be class methods
    def get_name(self): # instance method
        print(self)
        print("Name: ", self.name)
    @staticmethod
    def info(): # static method
        print("Hello This is A normal Class to understand static, class and␣
 ↪instance methods")
```

[84]:
```
a = A()
```

[86]:
```
A.get_msg() # can you create a method which can be called directly by class name
```

```
<class '__main__.A'>
Message:  I am class variable
```

[87]:
```
a.get_msg()
```

```
<class '__main__.A'>
Message:  I am class variable
```

[88]:
```
A.info()
```

```
Hello This is A normal Class to understand static, class and instance methods
```

[89]:
```
a.info()
```

```
Hello This is A normal Class to understand static, class and instance methods
```

```
[90]: a.get_name()
```

```
<__main__.A object at 0x000001E358B400C8>
Name:  I am instance variable
```

```
[91]: class A:
          msg = 'I am class variable' # class property or class attribute
          def __init__(self): # instance method
              self.name = 'I am instance variable'
          @classmethod
          def get_msg(x): # class method
              print(x)
              print("Message: ", x.msg) # is should be class methods
          def get_name(y): # instance method
              print(y)
              print("Name: ", y.name)
          @staticmethod
          def info(): # static method
              print("Hello This is A normal Class to understand static, class and␣
       ↪instance methods")
```

```
[92]: A.get_msg()
```

```
<class '__main__.A'>
Message:  I am class variable
```

```
[93]: a = A()
```

```
[94]: a.get_msg()
```

```
<class '__main__.A'>
Message:  I am class variable
```

```
[95]: a.get_name()
```

```
<__main__.A object at 0x000001E358B27548>
Name:  I am instance variable
```

```
[96]: A.info()
```

```
Hello This is A normal Class to understand static, class and instance methods
```

```
[98]: a.__class__
```

```
[98]: __main__.A
```

[114]:
```
def myclassmethod(method):
    def new_method(self, *args, **kwargs):
        cls = self.__class__
        return method(cls, *args, **kwargs)
    return new_method
```

[115]:
```
def hello(self):
    return self.name
A.hello = hello # instance method
```

[116]:
```
a.hello()
```

[116]:
```
'I am instance variable'
```

[117]:
```
@myclassmethod
def set_msg(self, new_msg):
    self.msg = new_msg
```

[118]:
```
A.set_msg = set_msg
```

[119]:
```
a.set_msg('bouncer!!!double bouncer!!!double bouncer')
```

[120]:
```
A.msg
```

[120]:
```
'bouncer!!!double bouncer!!!double bouncer'
```

[122]:
```
isinstance(a, A)
```

[122]:
```
True
```

[123]:
```
issubclass(A, object)
```

[123]:
```
True
```

[125]:
```
hasattr(a, 'name')
```

[125]:
```
True
```

[126]:
```
hasattr(a, 'age')
```

[126]:
```
False
```

[128]:
```
getattr(a, 'name')
# a.name
```

[128]:
```
'I am instance variable'
```

```
[129]: #a.name = 'some new value'
       setattr(a, 'name', 'some new value')
```

```
[130]: a.name
```

```
[130]: 'some new value'
```

```
[131]: a.age = 24
```

```
[132]: a.age
```

```
[132]: 24
```

```
[133]: def hello():
           print("hello world")
```

```
[134]: a.hello = hello # ? function
```

```
[135]: a.hello()
```

```
hello world
```

```
[141]: @staticmethod
       def get_msg():
           print('ho ho ho')
```

```
[142]: A.abcd = get_msg
```

```
[143]: A.abcd
```

```
[143]: <function __main__.get_msg()>
```

```
[144]: a.abcd()
```

```
ho ho ho
```

0.0.1 Property

```
[145]: class A:
           pass
```

```
[146]: a = A()
```

```
[147]: a.name = 'ha ha ha'
```

```
[149]: a.name
```

[149]: 'ha ha ha'

[158]:
```
class A:
    msg = 'haa haa'
    def set_name(self, name):
        self.name = name
    def get_name(self):
        print(self.name)
```

[159]:
```
a = A()
```

[160]:
```
a.set_name('sachin yadav')
```

[161]:
```
a.get_name()
```

```
sachin yadav
```

[162]:
```
a.name # name getter property of object a
```

[162]: 'sachin yadav'

[163]:
```
a.name = 'i hacked you' # name setter property of object a
```

[168]:
```
a.msg = 'class changed' # setter properties
```

[170]:
```
a.msg  # getter properties
```

[170]: 'class changed'

[165]:
```
A.msg
```

[165]: 'haa haa'

[166]:
```
vars(A)
```

[166]:
```
mappingproxy({'__module__': '__main__',
              'msg': 'haa haa',
              'set_name': <function __main__.A.set_name(self, name)>,
              'get_name': <function __main__.A.get_name(self)>,
              '__dict__': <attribute '__dict__' of 'A' objects>,
              '__weakref__': <attribute '__weakref__' of 'A' objects>,
              '__doc__': None})
```

[167]:
```
vars(a)
```

[167]: {'name': 'i hacked you', 'msg': 'class changed'}

[1]:
```python
class Person:
    __total_object = 0
    def __init__(self, name, country):
        Person.__total_object += 1
        self.__name = name
        self.__country = country
    def __str__(self):
        return self.name
    @property
    def name(self):
        print("Name: ", self.__name)
        print("Country: ", self.__country)

    @classmethod
    def total_objects(cls):
        print("Total Objects are: ", cls.__total_object)
    def __del__(self):
        """destructor will call automatically whenever your object is deleted"""
        print(f"Deleting {self.name} from memory")
        Person.__total_object -= 1
```

[2]:
```python
p1 = Person('jhon', 'USA')
p2 = Person('natsha', 'India')
```

[3]:
```python
p1.total_objects()
```

```
Total Objects are:  2
```

[4]:
```python
p3 = Person('sachin', 'India')
```

[5]:
```python
Person.total_objects()
```

```
Total Objects are:  3
```

[6]:
```python
p1.name
```

```
Name:  jhon
Country:  USA
```

[7]:
```python
p2.name # property
```

```
Name:  natsha
Country:  India
```

[8]:
```python
p3.name
```

```
Name:  sachin
Country:  India
```

[9]:
```
p1.name = 'not allowed'
```

```
      ␣
↪---------------------------------------------------------------------------

       AttributeError                            Traceback (most recent call␣
↪last)

       <ipython-input-9-202ad6d65469> in <module>
   ----> 1 p1.name = 'not allowed'


       AttributeError: can't set attribute
```

[10]:
```
l = [ 1, 2, 3]
l.name = 'laskdjf'
```

```
      ␣
↪---------------------------------------------------------------------------

       AttributeError                            Traceback (most recent call␣
↪last)

       <ipython-input-10-30fe9b572ab5> in <module>
         1 l = [ 1, 2, 3]
   ----> 2 l.name = 'laskdjf'


       AttributeError: 'list' object has no attribute 'name'
```

[15]:
```
class Person:
    __total_object = 0
    def __init__(self, name, country):
        Person.__total_object += 1
        self.__name = name
        self.__country = country
    def __str__(self):
        return self.name
    @property
    def name(self):
        print("Name: ", self.__name)
        print("Country: ", self.__country)

    @name.setter
    def name(self, tup):
        self.__name = tup[0]
        self.__country = tup[1]

    @classmethod
    def total_objects(cls):
        print("Total Objects are: ", cls.__total_object)
    def __del__(self):
        """destructor will call automatically whenever your object is deleted"""
        print(f"Deleting {self.name} from memory")
        Person.__total_object -= 1
```

[16]:
```
a = Person('sachin', 'india')
```

```
Name:  sachin
Country:  india
Deleting None from memory
```

[17]:
```
a.name
```

```
Name:  sachin
Country:  india
```

[18]:
```
a.name = 'Rajat Goyal', 'USA'
```

[19]:
```
a.name
```

```
Name:  Rajat Goyal
Country:  USA
```

0.0.2 slots

[20]:
```
l = list([1, 2, 3,4])
```

[21]:
```
l.a = 'hi'
```

```
       ␣
↪---------------------------------------------------------------------------

        AttributeError                            Traceback (most recent call␣
↪last)
```

```
    <ipython-input-21-94691a9bf6a3> in <module>
----> 1 l.a = 'hi'


    AttributeError: 'list' object has no attribute 'a'
```

[23]:
```python
class A:
    pass
```

[24]:
```python
a = A()
```

```
Name:  Rajat Goyal
Country:  USA
Deleting None from memory
```

[25]:
```python
a.hi = 'hello' # ?
```

[26]:
```python
a.hi
```

[26]:
```
'hello'
```

[31]:
```python
class A:
    __slots__ = [ 'name', 'age']
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def __str__(self):
        return self.name
    @property
    def data(self):
        s = f"""
            Name = {self.name}
            Age = {self.age}
        """
        return s
```

[32]:
```python
a = A('sachin',24)
```

[34]:
```python
print(a.data)
```

```

            Name = sachin
            Age = 24

```

[35]:
```python
print(a)
```

```
sachin
```

[36]:
```
a.country = 'India'
```

```
        ␣
↪---------------------------------------------------------------------------

        AttributeError                            Traceback (most recent call␣
↪last)

        <ipython-input-36-fa4720cfee86> in <module>
    ----> 1 a.country = 'India'


        AttributeError: 'A' object has no attribute 'country'
```

[37]:
```
a.__dict__
```

```
        ␣
↪---------------------------------------------------------------------------

        AttributeError                            Traceback (most recent call␣
↪last)

        <ipython-input-37-24d50c1843f4> in <module>
    ----> 1 a.__dict__


        AttributeError: 'A' object has no attribute '__dict__'
```

0.0.3 Meta Class

```
Pre-Mature --> Mature
```

MRO

Name Mangling

magic methods

duck typing

Monkey Patching

[38]:
```python
class A:
    pass
```

[39]:
```python
def method(self):
    print("I am monkey patching.")
```

[40]:
```python
A.method = method
```

[41]:
```python
a = A()
```

[42]:
```python
a.method()
```

```
I am monkey patching.
```

```
class composition, aggregation
meta class
abstract class
singleton class
```

[ ]:
```python

```