INFO-H-600 - Computing foundations of data sciences

Session 5

Introduction to Python

Object Oriented introduction

Based on https://docs.python.org/3/tutorial/classes.html

Université libre de Bruxelles
École polytechnique de Bruxelles

2019-2020

Object Oriented

A class is a model for objects indicating its :

- attributes : variables representing the state of the object
- methods : functions allowing to compute stuff (change the state for instance)

Definition of a Class

```
class MyClass:
  """A simple class"""

  def f(self):
    self.i = 12345
    return 'hello world'
```

Instensiation of an object

```
>>> o = MyClass()
>>> o.i
12345
>>> type(o)
<class '__main__.MyClass'>
>>> o.f()
'hello world'
```

- o.i and o.f are valid attribute references
- return an integer and a function object

Object Oriented : constructor

The function `__init__` is called when an object is created

```
class Complex:
    def __init__(self, realpart, imagpart):
        self.r = realpart
        self.i = imagpart

>>> x = Complex(3.0, -4.5)
>>> x.r, x.i
(3.0, -4.5)
```

Notice the usage of the self keyword which represent the object itself.

Object Oriented : class or object

The attributes *x* initiated in `__init__` with `self.x =...` are proper to each instance while the others belong the the class

```
class Dog:
    # class variable shared by all instances
    kind = 'canine'

    def __init__(self, name):
        # instance variable unique to each instance
        self.name = name

>>> d = Dog('Fido')
>>> e = Dog('Buddy')
>>> d.kind                    # shared by all dogs
'canine'
>>> e.kind                    # shared by all dogs
'canine'
>>> d.name                    # unique to d
'Fido'
>>> e.name                    # unique to e
'Buddy'
```

Object Oriented : class or object

```python
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)

>>> d = Dog('Fido')
>>> e = Dog('Buddy')
>>> d.add_trick('roll over')
>>> e.add_trick('play dead')
>>> d.tricks                # unexpectedly shared by all dogs
['roll over', 'play dead']
```

Object Oriented : class or object

```
class Dog:

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

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

>>> d = Dog('Fido')
>>> e = Dog('Buddy')
>>> d.add_trick('roll over')
>>> e.add_trick('play dead')
>>> d.tricks
['roll over']
>>> e.tricks
['play dead']
```

Object Oriented : Encapsulation

```
class Dog:

    def __init__(self, name):
        self.name = name
        self.hunger = 0

    def eat(self):
        self.hunger = 0

    def not_eat(self):
        self.hunger += 1

>>> d = Dog('Fido')
>>> d.hunger = -10                        # What does this mean ?
```

every one can access the attributes of our objects and this can be problematic

Object Oriented : Encapsulation

Proctect these attributes !

```
class Dog:
    def __init__(self, name):
        self.__name = name
        self.__hunger = 3

    def eat(self):
        self.__hunger = 0

    def not_eat(self):
        self.__hunger += 1

    def get_hunger(self):
        return self.__hunger

d = Dog('Fido')
d.__hunger = -10

print(d.__hunger)       # -10
print(d.get_hunger())   #  3
```

The attribute `__hunger` declared in the class is *hided* for the rest of the code. The line `d.__hunger = -10` creates a new attribute

Object Oriented : Encapsulation

```python
class Dog:
    def __init__(self, name):
        self.__name = name
        self.__hunger = 3

    def eat(self):
        self.__hunger = 0

    def not_eat(self):
        self.__hunger += 1

    def get_hunger(self):
        return self.__hunger
```

Modify the values of the attributes only using methods declared in the class.

Object Oriented : Encapsulation

```python
class Dog:
    def __init__(self, name):
        self.__name = name
        self.__hunger = 3

    def eat(self):
        self.__hunger = 0

    def not_eat(self):
        self.__hunger += 1
```

This is not the most pythonic way to declare attributes. We use it here to keep it simple.

Please visit : `https://www.python-course.eu/python3_properties.php` for more information

Object Oriented : classmethods and staticmethod

```
class Dog:

    quantity = 8

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

    @classmethod
    def get_quantity(cls):
        print(cls.quantity)      # prints 8
        # print(cls.__name)      # doesn't work

    @staticmethod
    def do_sth():
        # print(quantity)     # doesn't work
        # print(__name)       # doesn't work


Dog.get_quantity()
```