

L3 Informatique - 2020-2021

Partie POO

Atelier 3 – POO en python



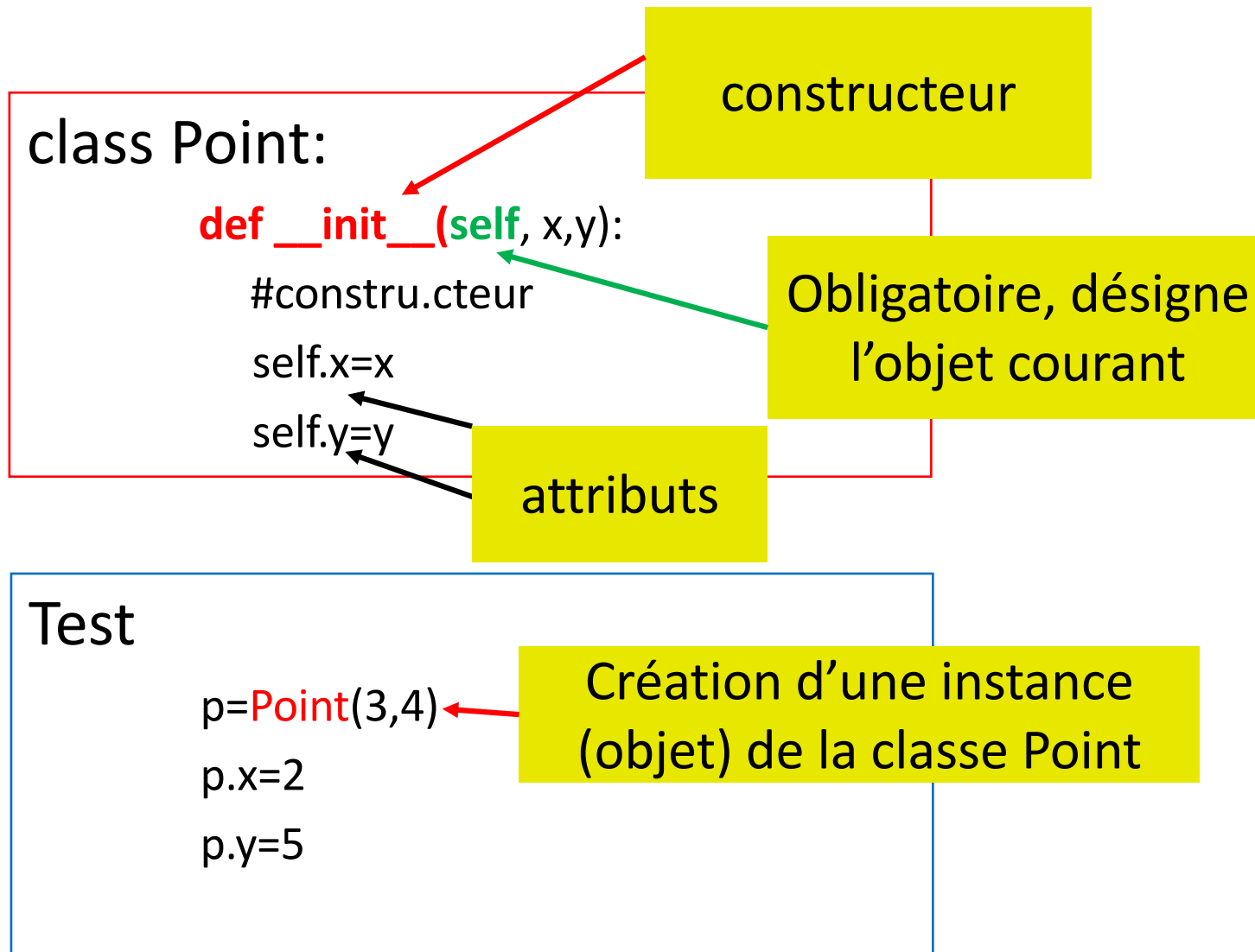
Plan

POO en python (résumé)

- Définition d'une classe
 - Constructeur
 - Attributs
 - Méthodes
 - Encapsulation
- Attributs et méthodes de classe
- Héritage



Constructeur et attributs



Constructeur et attributs (valeurs par défaut)

```
class Point:
```

```
    def __init__(self, x, y, z=0):
```

```
        #constru.cteur
```

```
        self.x=x
```

```
        self.y=y
```

```
        self.z=z
```

Valeur par défaut

A blue arrow points from the yellow box 'Valeur par défaut' to the 'z=0' part of the constructor signature in the code block above.

Test

```
p1=Point(3,4)
```

Création d'un Point 2D

A red arrow points from the yellow box 'Création d'un Point 2D' to the 'Point(3,4)' part of the code line 'p1=Point(3,4)'.

```
p2=Point(3,4,5)
```

Création d'un Point 3D

A red arrow points from the yellow box 'Création d'un Point 3D' to the 'Point(3,4,5)' part of the code line 'p2=Point(3,4,5)'.

Encapsulation : attributs privés

```
class Point:
```

```
    def __init__(self, x,y):
```

```
        #constru.uteur
```

```
        self.__x=x
```

```
        self.__y=y
```

Les attributs sont déclarés
en visibilité privée

__ nom

```
Test
```

```
p=Point(3,4)
```

```
p.x=2
```

```
p.y=5
```

Interdit: il faut définir des
méthodes (modificateurs)

Méthodes

class Point:

```
def __init__(self, x,y):
```

```
    #constructeur
```

```
    self.x=x
```

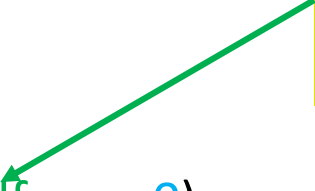
```
    self.y=y
```

```
def deplace (self, x, y=0) :
```

```
    self.x+=x
```

```
    self.y+=y
```

Obligatoire, désigne
l'objet courant



Valeur par défaut



Test

```
p=Point(3,4)
```

```
p.deplacer(2) #le point devient (5,4)
```

```
p.deplacer(2,3) #le point devient (7,7)
```

Méthodes spéciale `__str__`

Idem `toString()` en Java

- `__str__`
 - Renvoie une chaîne de caractère représentant l'objet
 - À utiliser pour afficher des objets avec `print`
- Python offre plusieurs autres méthodes « spéciales » (format : `__methode__`)

Pour plus de détails voir <https://openclassrooms.com/courses/apprenez-a-programmer-en-python/les-methodes-speciales-1>



Attributs de classe

Un attribut de classe en python est une variable définie en dehors de toute méthode.

```
class Point:
```

```
    nb_points=0
```

```
    def __init__(self, x,y):
```

```
        #constructeur
```

```
        self.x=x
```

```
        self.y=y
```

```
        Point.nb_points+=1
```

Variable attachée à la classe et non à l'instance

Méthodes de classe

Une méthode de classe en python est une méthode

```
class Point:
```

```
    nb_points=0
```

```
    def __init__(self, x,y):
```

```
        self.x=x
```

```
        self.y=y
```

```
        nb_points+=1
```

```
    @classmethod
```

```
    def get_nb_points(cls):
```

```
        return Point.nb_points
```

Décorateur indiquant que la méthode est « de classe »

Obligatoire en 1^{er} argument

Héritage

```
class Point:
```

```
    def __init__(self, x,y):
```

```
        #constructeur
```

```
        self.x=x
```

```
        self.y=y
```

```
class Point3D (Point):
```

```
    def __init__(self, x,y,z):
```

```
        #constructeur
```

```
        super().__init__(x,y)
```

```
        self.z=z
```

Invocation du constructeur
de la classe mère (non
obligatoire en python)

Héritage (exemple)

```
>>> class Rectangle:
...     def __init__(self, longueur=30, largeur=15):
...         self.L, self.l = longueur, largeur
...         self.nom = "rectangle"
...     def __str__(self):
...         return "nom : {}".format(self.nom)
...
>>> class Carre(Rectangle): # héritage simple
...     """Sous-classe spécialisée de la super-classe Rectangle."""
...     def __init__(self, cote=20):
...         # appel au constructeur de la super-classe de Carre :
...         super().__init__(cote, cote)
...         self.nom = "carré" # surcharge d'attribut
...
>>> r = Rectangle()
>>> c = Carre()
>>> print(r)
nom : rectangle
>>> print(c)
nom : carré
```

Redéfinition de méthodes

```
class Point:
```

```
....
```

```
def deplace (self, x, y) :
```

```
    self.x+=x
```

```
    self.y+=y
```

```
class Point3D (Point):
```

```
....
```

```
def deplace (self, x,y,z) :
```

```
    Point.deplace(self,x,y)
```

```
    self.z+=z
```

Possibilité d'invocation de
la méthode redéfinie de la
classe mère

Exemple

```
class Point:
    nb_points=0
    def __init__(self, x,y):
        #constructeur
        self.x=x
        self.y=y
        Point.nb_points+=1

    def deplace (self, x, y) :
        self.x+=x
        self.y+=y

    def __str__(self) :
        return str(self.x)+" - "+str(self.y)

    @classmethod
    def get_nb_points(cls):
        return Point.nb_points

class Point3D(Point):
    def __init__(self, x,y,z):
        #constructeur
        super().__init__(x,y)
        self.z=z

    def deplace (self, x, y, z) :
        Point.deplace(self,x,y)
        self.z+=z

    def __str__(self) :
        return Point.__str__(self)+" - "+str(self.z)

p=Point(3,2)
print(p)
q=Point3D(4,5,6)
print(q)
q.deplace(1,1,2)
print(q)
print("Nombre de points créés :"+str(Point.get_nb_points()))
```

3 - 2
4 - 5 - 6
5 - 6 - 8
Nombre de points créés :2