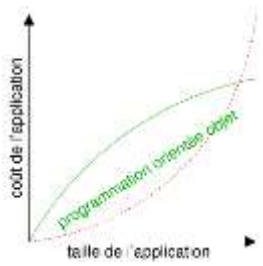
Programmation- Orientée-Objet : POO

1. Contextualisation

2. https://courspython.com/classes-et-objets.html

La programmation « traditionnelle - procédurale » à base de séquences faisant appellent à des fonctions produit des programmes difficiles à maintenir et à faire évoluer. Leurs coûts explosent avec la complexité qui leur est de plus en plus demandé.



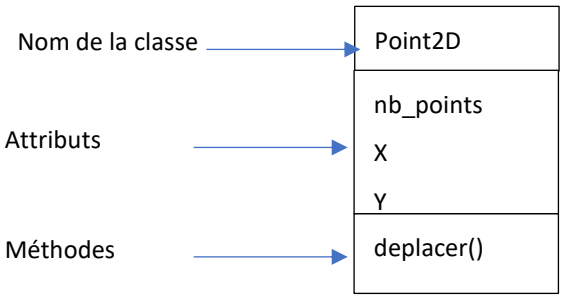
Pour obtenir un code évolutif et facile à maintenir, les informaticiens ont fait évoluer leur paradigme de programmation. Plutôt que de programmer des instructions qui suivent un ordre et une logique déterminée d'avance, ils ont modélisé des objets. Ces objets sont des modèles inspirés du réel qui interagissent en fonction des actions qui leur sont appliquées.

Ces objets peuvent se spécialiser c'est l'héritage ou s'adapter au contexte c'est le polymorphisme. Aussi le code déjà écrit peut-il être réemployé sans avoir besoin de le modifier.

3. Constitution d'un objet

Pour modéliser un objet on définit un cadre c'est ce qu'on appelle **une classe** en informatique. De ce moule on pourra créer autant d'objets. Un objet sera dont une instance de cette classe. **On dit qu'un objet est une instance de telle classe.**

Ex




```
class Point2D():
    nb_points = 0       # attribut de classe
    def __init__(self, abs=0, ord =0):
        self.x = abs    # attribut d'instance
        self.y = ord    # attribut d'instance
        self.__class__.nb_points+=1 # modif attribut de
 classe
    def deplace(self, dx, dy):
        self.x = self.x + dx
        self.y = self.y + dy
```

En python

Le mot réservé pour écrire une classe est class !

Une variable définie au niveau d'une classe (ex nb points) est appelé attribut de classe et est partagée par tous les objets instances de cette classe.

Une variable définie au niveau d'un objet (comme x et y) est appelée attribut d'instance et est liée uniquement à l'objet pour lequel elle est définie.

Self n'est pas un mot réservé en python mais il indique que c'est l'objet lui-même qui est concerné. On pourrait utiliser un autre mot (qui aura la même fonction et dont le nom devra être conservé) mais cela sera difficilement interprétable pour toute la communauté.

Si lors de la création d'un objet nous voulons qu'un certain nombre d'actions soit réalisées (par exemple une initialisation), nous pouvons utiliser un **constructeur**.

Un constructeur n'est rien d'autre qu'une méthode, sans valeur de retour, qui porte un nom imposé par le langage Python : __init__(). Ce nom est constitué de init entourée avant et après par __ (**deux fois** le symbole **underscore** _, qui est le tiret sur la touche 8). Cette méthode sera appelée lors de la création de l'objet. Le constructeur peut disposer d'un nombre quelconque de paramètres, éventuellement aucun.

4. Utilisation des objets

Création

Pour utiliser un objet il faut créer une instance de la classe de cet objet. On prendre le moule et on démoule en donnant un nom au moulage.

```
A = Point2D()
```

Accès aux attributs

En python les attributs sont accessibles par défaut

Code	affiche
print(A.x)	
print(A.y)	
print(A.nb_points)	

```
A.x =5
A.y =5
```

Code	affiche
print(A.x)	
print(A.y)	
print(A.nb_points)	

En donnant des valeurs aux attributs (constructeur paramétré)

```
B = Point2D(2,3)
```

Code	affiche
print(B.x)	
print(B.y)	
print(B.nb_points)	

Manipulation

Pour utiliser les objets on leur applique leurs méthodes : nom_objet.nom_methode(arguments)

Code	affiche
A.deplace(2,3)	
print(A.y)	
print(A.y)	
print(B.nb_points)	

Programmation- Orientée-Objet : POO

5. Représentation visible de l'objet : methode __str__

str(object) retourne une chaîne imprimable ou lisible par l'homme. Si aucun argument n'est donné, cela retourne la chaîne vide, '' .

Ex :

méthode	`def __str__(self) :` `  return(" nb points : {}  abs : {} , ord : {}".format(Point2D.nb_points,self.x,self.y))`
Appel 1	`print(A)  # premier appel`
résultat	

	`A.x = 5`
	`A.y = 5`
	`print(A)`

résultat	

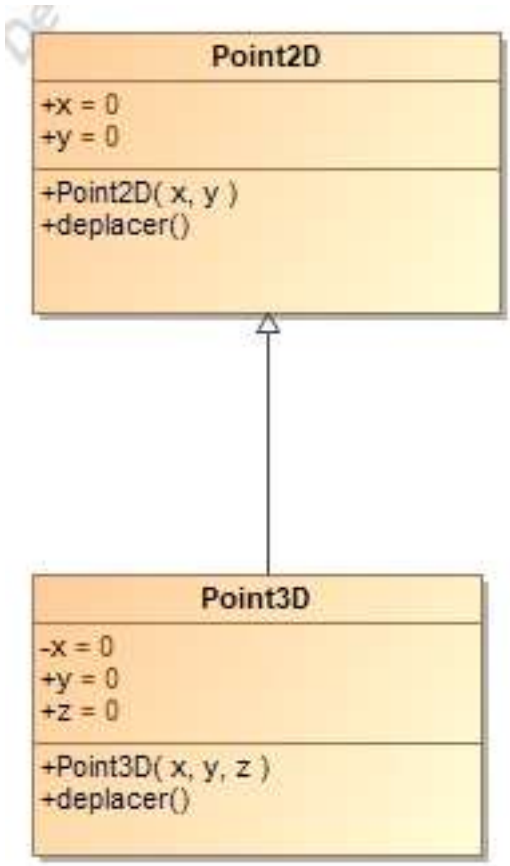
6. Notion de masquage de nom

En python il est possible de créer à tout moment un attribut pour une classe ou une instance de classe (via une méthode ou accès direct à l'instance) peut entrainer des erreurs où par exemple une affectation faite par erreur au niveau d'une instance masque le même nom défini au niveau de la classe.

Ex

Code	Commentaires
`A.nb_points = 366`	
`print(A.nb_points)`	
`print(Point2D.nb_points)`	
`print(B.nb_points)`	

7. Notion d'héritage et polymorphisme (hors programme)




```
class Point2D():
    def __init__(self, x=0, y =0):
        self.x = x    # attribut d'instance
        self.y = y    # attribut d'instance
    def deplace(self, dx, dy):
        self.x = self.x + dx
        self.y = self.y + dy
    def __str__(self) :
        return(" abs : {} , ord : {}".format(self.x,self.y))
```

```
class Point3D(Point2D) :

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

    def deplace(self, dx, dy,dz):
        self.x = self.x + dx
        self.y = self.y + dy
        self.z = self.z + dz
    def __str__(self) :
        return(" abs : {} , ord : {}, hauteur :{}".
format(self.x,self.y,self.z))
```

Appels	Résultats
K=Point2D() print(K) K.deplace(1,1) print(K) G = Point3D() print(G) G.deplace(2,2,2) print(G)	