Lycée Albert EINSTEIN

Programmation Orientée Objet Python

N. Liebeaux – Actualisation J. Kergot

Python en 1 diapo

Règles de syntaxe

- langage interprété
- typage dynamique des données
- les « deux points »
- blocs découpés par l'indentation

types de données

- nombres entiers & flottants
- chaînes " "
- listes [ ]

Contrôle de Flux

- `for, while`
- `if, else, elif`

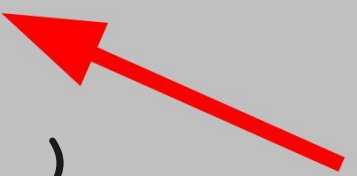
Les Fonctions

- utilisation de **`def`** et « : »
- passage de paramètres
- **`return`** renvoie le résultat

Objet : entité évoluée associée à des données et des fonctions (appelées **méthodes**) qui lui sont propres.
Une **Classe** est le moule qui permet de créer l'objet, ou les objets.
L'opération de création d'un nouvel objet à partir d'une classe est appelée **instanciation**.

```
class Rectangle(object):
   "cette classe définit les rectangles"
```

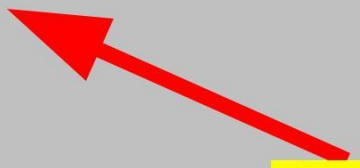
```
>>> rect1 = Rectangle()
>>> print (rect1)
>>> print (rect1.__doc__)
```



piège

```
>>> r2 = r1
>>> print (r2==r1)
>>> print (r1)
>>> print (r2)

>>> r2 = Rectangle()
>>> print (r2==r1)
>>> print (r1)
>>> print (r2)
```



r2 est un alias de r1 portant sur le même objet.

piège bis

```
>>> liste1 = [1,2,3,"a","b","z"]
>>> liste2 = [1,2,3,"a","b","z"]
>>> print liste1,liste2
>>> print liste2[5]="c"
>>> print liste1,liste2
```

constructeur

```
class Rectangle(object):
   "cette classe instancie des rectangles"

   def __init__(self, dim1, dim2):
       self.longueur = dim1
       self.largeur = dim2
```

la fonction __init__ est le **constructeur**. C'est la méthode qui "construit" l'objet lors du premier appel

notez l'usage du mot réservé **`self`**.

la méthode **`__init__`** nécessite ici deux paramètres (et non pas 3), correspondant aux dimensions du rectangle.

exemple

```
>>> r1 = Rectangle()
>>> r1 = Rectangle(2,3)
>>> print (r1.longueur)
>>> print (r1.largeur)
```

encapsulation

un **objet** porte en lui-même ses données et ses méthodes :
c'est **l'encapsulation**.
avec ce concept, l'objet est utilisé via ses méthodes, peu importe pour l'utilisateur de connaître le contenu (le code) des dites méthodes.

syntaxe

objetInstancié.méthode()

encapsulation

```
class Rectangle(object):
   "cette classe instancie des rectangles"

   def __init__(self, dim1, dim2):
        self.longueur = dim1
        self.largeur = dim2
   def calculerAire(self):
        print("Aire :")
        return self.longueur*self.largeur
```

```
class Carre(Rectangle):
   "cette classe instancie des carrés"

   def __init__(self, dim1):

        Rectangle.__init__(self,dim1, dim1)
```

une **classe mère** peut donc dériver plusieurs **classes filles**. C'est la **classe parente**.

idée de **spécialisation** de la **classe fille**.

```
class Carre(Rectangle):
   "cette classe instancie des carrés"

   def __init__(self, dim1):

        rectangle.__init__(self,dim1, dim1)

   def calculerAire(self):
        print("Aire :")

        return self.longueur*self.largeur
```

Polymorphisme = plusieurs méthodes de même nom dans les classes filles.

la méthode **`calculAire`** est définie dans la classe **rectangle**.

la méthode **`calculAire`** est définie dans la classe **carré**. On parle alors de **surcharge**.

exercice corrigé

1/ - proposer une structure de classe point
: coordonnées x et y.
2/ - proposer une structure de classe vecteur.
3/ - proposer une méthode permettant de calculer
le produit scalaire.
4/ - proposer une méthode permettant de vérifier
si deux vecteurs sont orthogonaux.

conclusion

Classes filles peuvent :

étendre les concepts de la classe parente (ex: points vecteurs)
Affiner les concepts de la classe parente (ex: rectangle carré)

exercice

1/ - proposer une structure de classe individu : nom,prénom, date de naissance.

2/ - proposer une structure de classe étudiant : bac, notes par matière.

3/ - proposer une méthode permettant de calculer l'age d'un individu,

4/ - proposer une méthode permettant d'ajouter une note.

5/ - proposer une méthode de calcul de moyenne générale des notes.