INSAlgo - Cours 9

...

Sujet du cours : La POO, les classes et comment on peut s'en servir en algo

Cours en Python car la syntaxe est simple et compréhensible pour les débutants ! ٩(◕‿◕｡)۶

Les paradigmes de programmation

Impérative

Fonctionnelle

Orientée Objet

- **Python**: What if everything was a dict?
- **Java**: What if everything was an object?
- **JavaScript**: What if everything was a dict *and* an object?
- **C**: What if everything was a pointer?
- **APL**: What if everything was an array?
- **Tcl**: What if everything was a string?
- **Prolog**: What if everything was a term?
- **LISP**: What if everything was a pair?
- **Scheme**: What if everything was a function?
- **Haskell**: What if everything was a monad?
- **Assembly**: What if everything was a register?
- **Coq**: What if everything was a type/proposition?
- **COBOL**: WHAT IF EVERYTHING WAS UPPERCASE?
- **C#**: What if everything was like Java, but different?
- **Ruby**: What if everything was monkey patched?
- **Pascal**: BEGIN What if everything was structured? END
- **C++**: What if we added everything to the language?
- **C++11**: What if we forgot to stop adding stuff?
- **Rust**: What if garbage collection didn't exist?
- **Go**: What if we tried designing C a second time?
- **Perl**: What if shell, sed, and awk were one language?
- **Perl6**: What if we took the joke too far?
- **PHP**: What if we wanted to make SQL injection easier?
- **VB**: What if we wanted to allow anyone to program?
- **VB.NET**: What if we wanted to stop them again?
- **Forth**: What if everything was a stack?
- **ColorForth**: What if the stack was green?
- **PostScript**: What if everything was printed at 600dpi?
- **XSLT**: What if everything was an XML element?
- **Make**: What if everything was a dependency?
- **m4**: What if everything was incomprehensibly quoted?
- **Scala**: What if Haskell ran on the JVM?
- **Clojure**: What if LISP ran on the JVM?
- **Lua**: What if game developers got tired of C++?
- **Mathematica**: What if Stephen Wolfram invented everything?
- **Malbolge**: What if there is no god?

@nixcraft

Pourquoi la POO ?

Réunir données et traitement

Abstraction des traitements

Code plus générique

Rendre le code plus lisible

Conteneuriser, chaque bloc de code a un but précis

Les briques de base : **l'objet et la classe (😎)**

La classe

C'est ce qu'on définit, c'est là qu' on décrit à quoi ressemble un objet, quels seront ses **attributs** et ses **méthodes**

L'objet

C'est ce qu'on manipule dans le programme, on sait ce qu'il contient et comment l'utiliser grâce à la **classe** qui définit son comportement

Mais alors du coup en code ça donne quoi ?

```python
class Point:
    def __init__(self, px, py):
        self.x = px
        self.y = py

    def afficher(self):
        print(self.x, self.y)


point = Point(1, 2)
print(point.x)   # 1
point.afficher() # 1 2
```

```java
class Point{
    int x;
    int y;
    Point(int px, int py) {
        x = px;
        y = py;
        this.x = val;
        // Equivalent
    }
    void afficher(){
        System.out.print(x);
        System.out.println(" "+y);
    }
}

// Dans le main :
Point point = new Point(1, 2);
System.out.println(point.x)
point.afficher(); // 1 2
```

Qu'est-ce qui se passe donc ?

```
class Point:
    def __init__(self, px, py):
        self.x = px
        self.y = py

    def afficher(self):
        print(self.x, self.y)


point = Point(1, 2)
point.afficher() # 1 2
```

x : attribut
y : attribut
afficher : méthode
__init__ : méthode spéciale (constructeur)

self désigne l'objet depuis lequel la méthode est appelée

point.afficher() -> dans afficher, self est l'objet mon_point.

Point est la classe, mon_point est un objet de la classe Point

Le CONSTRUCTEUR (important)

Le constructeur est une méthode spéciale, elle est appelée à la création d'un objet.

On peut lui passer des paramètres comme n'importe quelle méthode.

Permet d'initialiser les attributs de notre objet selon les valeurs passées en paramètre

En python : **__init__**

La surcharge d'opérateurs

Pour utiliser des opérateurs tels que +, *, [],... sur vos objets.

On définit une méthode spéciale. Exemple pour le + :

```
class Point:
     def __init__(self, px, py):
          self.x = px
          self.y = py

     def afficher(self):
          print(self.x, self.y)


     def __add__(self, point2):
          self.x += point2.x
          self.y += point2.y
```

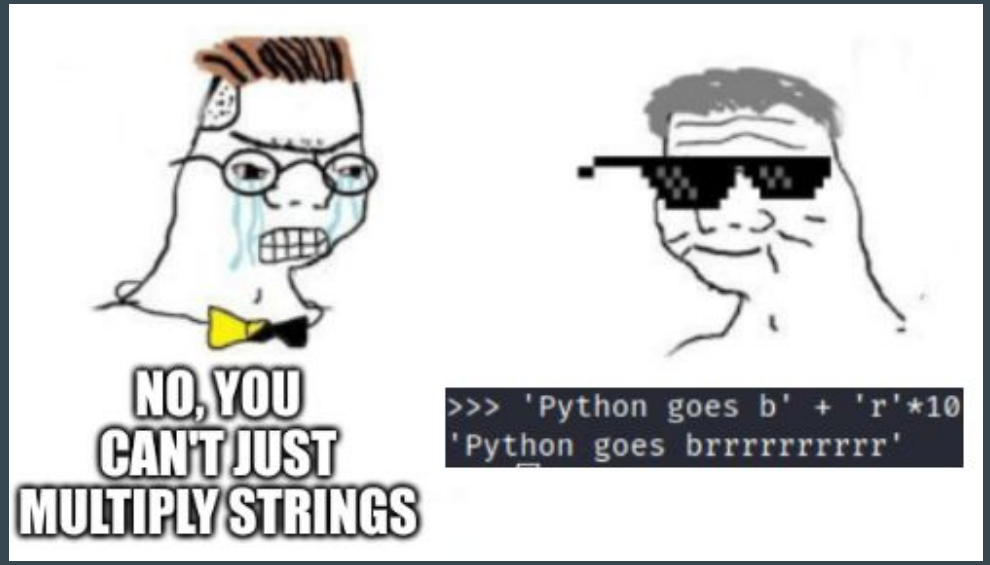
```
p1 = Point(1, 2)
p2 = Point(2, 3)

p1 + p2
// Equivalent : p1.__add__(p2)

p1.afficher() # 3 5
```

La surcharge d'opérateurs

Les classes de python surchargent déjà certains opérateurs :

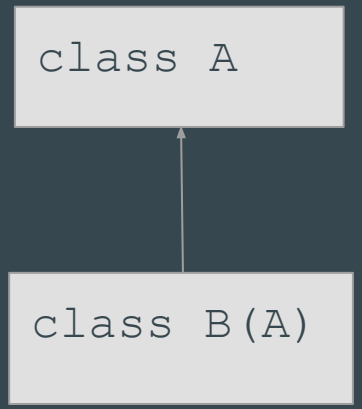


La surcharge d'opérateurs

Magic Method	When it gets invoked (example)	Explanation
`__new__(cls [,...])`	`instance = MyClass(arg1, arg2)`	`__new__` is called on instance creation
`__init__(self [,...])`	`instance = MyClass(arg1, arg2)`	`__init__` is called on instance creation
`__cmp__(self, other)`	`self == other`, `self > other`, etc.	Called for any comparison
`__pos__(self)`	`+self`	Unary plus sign
`__neg__(self)`	`-self`	Unary minus sign
`__invert__(self)`	`~self`	Bitwise inversion
`__index__(self)`	`x[self]`	Conversion when object is used as index
`__nonzero__(self)`	`bool(self)`	Boolean value of the object
`__getattr__(self, name)`	`self.name # name doesn't exist`	Accessing nonexistent attribute
`__setattr__(self, name, val)`	`self.name = val`	Assigning to an attribute
`__delattr__(self, name)`	`del self.name`	Deleting an attribute
`__getattribute__(self, name)`	`self.name`	Accessing any attribute
`__getitem__(self, key)`	`self[key]`	Accessing an item using an index
`__setitem__(self, key, val)`	`self[key] = val`	Assigning to an item using an index
`__delitem__(self, key)`	`del self[key]`	Deleting an item using an index
`__iter__(self)`	`for x in self`	Iteration
`__contains__(self, value)`	`value in self, value not in self`	Membership tests using `in`
`__call__(self [,...])`	`self(args)`	"Calling" an instance
`__enter__(self)`	`with self as x:`	`with` statement context managers
`__exit__(self, exc, val, trace)`	`with self as x:`	`with` statement context managers
`__getstate__(self)`	`pickle.dump(pkl_file, self)`	Pickling
`__setstate__(self)`	`data = pickle.load(pkl_file)`	Pickling

Liste pas forcément à jour et non exhaustives des “méthodes magiques” de Python

L’héritage : étendre les fonctionnalités d’une classe

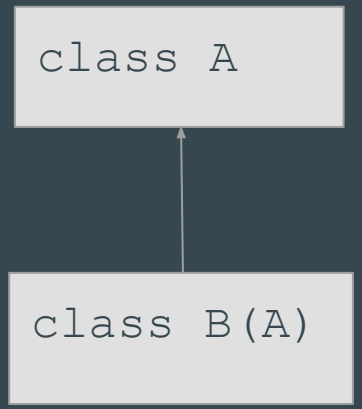


Si la classe A possède des attributs et des méthodes, la classe B les possédera aussi car elle les **hérite** de A.

Ce mécanisme permet d’étendre les fonctionnalités d’une classe, ou de fournir une spécialisation d’une classe plus générale.

Permet de construire votre propre graphe d’héritage avec vos classes pour réduire le code dupliqué.
Ou permet de modifier le comportement d’une classe préexistante dans le langage.

L’héritage : la surcharge de méthodes



Si la classe A possède des attributs et des méthodes, la classe B les possédera aussi car elle les **hérite** de A.

La classe **A** peut implémenter une méthode **afficher**.
Si cette méthode n’est pas redéfinie dans **B**, appeler **afficher** sur un objet de type **B** appellera la méthode **afficher** de **A**.
Si on redéfinit **afficher** dans **B**, appeler **afficher** sur un objet de type **B** va utiliser la méthode la plus “spécialisée”, c’est-à-dire celle de **B**.

Exemple de modification d'une classe :

On veut créer une liste qui affiche tout ce qu'on lui ajoute

```
class ListeAffichage(list):
    def __setitem__(self, key, val):
        print(f'On insère la valeur {val} à l\'emplacement {key}')
        return super().__setitem__(key, val)

liste = ListeAffichage([1, 2, 3, 4, 5])
liste[2] = 0; # On insère la valeur 0 à l'emplacement 2
```

Grâce à l'héritage, et à la surcharge, notre classe se comporte exactement comme une liste sauf à l'insertion

Mais alors en algo ça sert quand ?

Quand on veut manipuler des structures de données plus complexes que juste des nombres

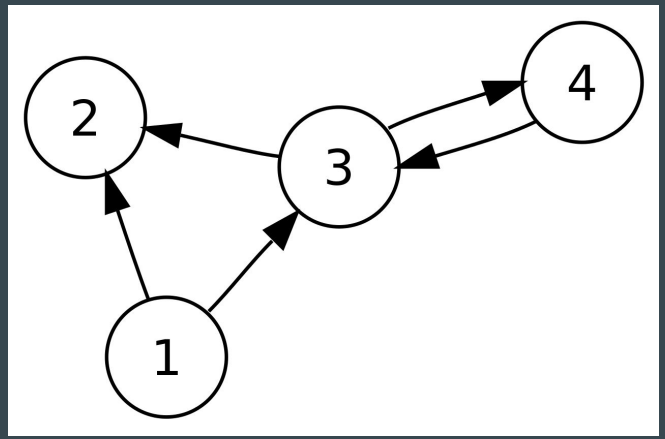
Graphes

Vecteurs

Points

Des singes qui font des maths !?
#AoC11

Les classes dans un graphe



On définit une classe Noeud, avec un attribut nom et un attribut voisins, qui est une liste de Noeud

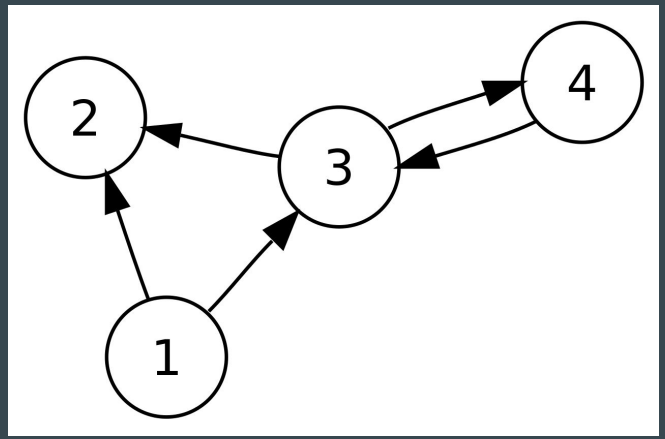
```
class Noeud:
      def __init__(self, nom, voisins = list()):
            self.nom = nom
            self.voisins = set(voisins)

      def ajouter_voisin(self, noeud):
            self.voisins.add(noeud)

      @property
      def nb_voisins(self):
            return len(self.voisins)

      @staticmethod
      def chemin_entre(noeud1, noeud2):
            chemin = ...
            # algo de Dijkstra
            return chemin
```

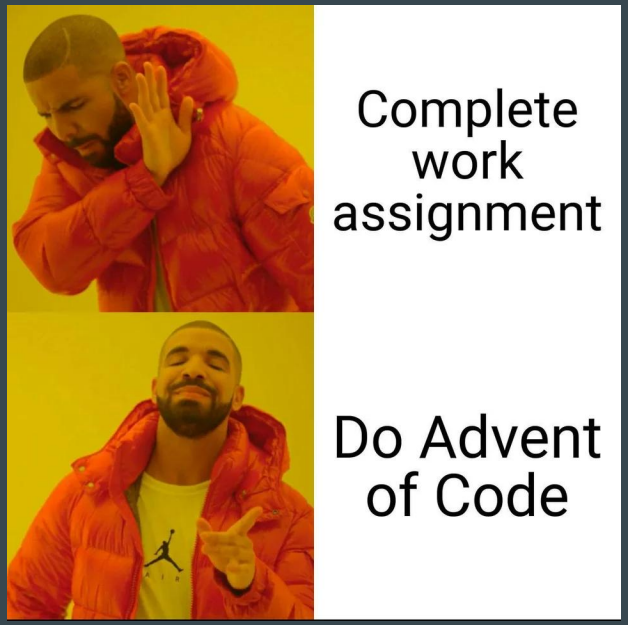
Les classes dans un graphe




```
n1 = Noeud(1)
n2 = Noeud(2)
n3 = Noeud(3)
n4 = Noeud(4, [n3])
n1.ajouter_voisin(n2)
n1.ajouter_voisin(n3)
n3.ajouter_voisin(n2)
n3.ajouter_voisin(n4)
a = n3.nb_voisins # 2
Noeud.chemin_entre(n1, n4)
# retourne [n1, n3, n4]
```

Fin du cours

Place à quelques exos. Comme d'habitude rdv sur le discord pour les voir.



Et n'oubliez pas l'Advent Of Code...