



# Initiation à la POO

Classe



Objet

## P rogrammation O rientée O bjet

La programmation orientée objet ou POO permet de mieux structurer les programmes. La réalité est modélisée en concevant des modèles d'objets, les classes.

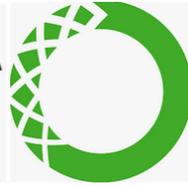
Ces classes permettent de construire des objets interactifs entre eux. Les objets sont créés indépendamment les uns des autres.

Mais un objet fils peut hériter des propriétés de son parent ce qui permet une programmation efficace et bien structurée.



### Sommaire

<b>1</b>	<b>Les différentes façons de produire un programme.....</b>	<b>2</b>
1.1	<i>Quelques techniques .....</i>	2
1.2	<i>Méthodologie procédurale, méthodologie objet .....</i>	2
1.3	<i>Exemple de programmation procédurale .....</i>	3
1.4	<i>Résumé de l'organisation d'un programme python type.....</i>	4
<b>2</b>	<b>La programmation orientée objet .....</b>	<b>5</b>
2.1	<i>Généralités .....</i>	5
2.2	<i>Résumé : définition et utilisation de classes (Swinnen).....</i>	6
2.3	<i>Autre exemple un peu de chimie .....</i>	7
<b>3</b>	<b>Mise en œuvre de Classes .....</b>	<b>10</b>
3.1	<i>Un exercice gestion de comptes bancaires .....</i>	10
3.2	<i>Travail à faire : .....</i>	10
3.3	<i>Un peu de géométrie .....</i>	11
<b>4</b>	<b>Un outil pour tracer des diagrammes UML .....</b>	<b>13</b>
<b>5</b>	<b>Conception POO.....</b>	<b>14</b>
5.1	<i>L'héritage.....</i>	14
5.2	<i>Associations : composition – agrégation .....</i>	14
5.3	<i>Travail à faire : .....</i>	14
<b>6</b>	<b>Résumé .....</b>	<b>15</b>
6.1	<i>Quelques définitions du domaine de la POO.....</i>	15
6.2	<i>Brève historique de la POO .....</i>	15
<b>7</b>	<b>Exercice de récapitulation du cours.....</b>	<b>16</b>
7.1	<i>Questions de compréhension.....</i>	16
7.2	<i>Utilisation des classes .....</i>	16



# 1 Les différentes façons de produire un programme<sup>1</sup>

## 1.1 Quelques techniques

### Deux techniques de production de programme

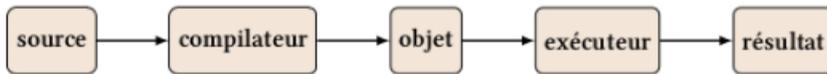


Figure 1.1 - Chaîne de compilation



Figure 1.2 - Chaîne d'interprétation

### Technique utilisée par Python



Figure 1.3 - Interprétation du bytecode compilé

Pour exécuter un programme, Python charge le fichier source .py en mémoire vive, en fait l'analyse (lexicale, syntaxique et sémantique), produit le bytecode et enfin l'exécute.

## 1.2 Méthodologie procédurale, méthodologie objet

Il y a plusieurs méthodes de construction des programmes citons-en deux principales<sup>2</sup> :

- la méthodologie procédurale. On emploie l'analyse descendante (division des problèmes) et remontante (réutilisation d'un maximum de sous-algorithmes). On s'efforce ainsi de décomposer un problème complexe en sous-programmes plus simples. Ce modèle structure d'abord les actions ;
- la méthodologie objet. Centrée sur les données, elle est considérée comme plus stable dans le temps et meilleure dans sa conception. On conçoit des fabriques (classes) qui servent à produire des composants (objets) qui contiennent des données (attributs) et des actions (méthodes). Les classes dérivent (héritage et polymorphisme) de classes de base dans une construction hiérarchique.

- `nom_de_ma_variable` pour les variables ;
- `NOM_DE_MA_CONSTANTE` pour les constantes ;
- `maFonction`, `maMethode` pour les fonctions et les méthodes ; les attributs;
- `MaClasse` pour les classes ;
- `UneExceptionError` pour les exceptions ;
- `nom_de_module` pour les modules et pour tous les autres identificateurs.

Il existe plusieurs règles de nommage, l'important est d'en choisir une et de s'y tenir.

<sup>1</sup> Voir ces liens utilisés pour ce cours <https://python.developpez.com/cours/apprendre-python-3/>  
<https://python.developpez.com/cours/apprendre-python3/>

<sup>2</sup> <https://python.developpez.com/cours/apprendre-python-3/?page=introduction#L1-4-3>



### 1.3 Exemple de programmation procédurale

Exemple avec le calcul du volume d'une sphère :

Pour décrire le code nous pouvons utiliser des algorigrammes ou bien un diagramme de flux de la modélisation UML :

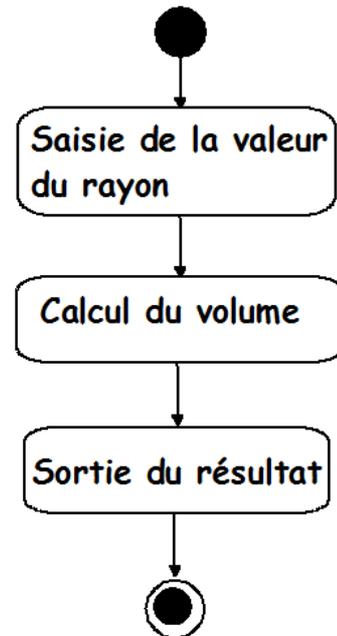
## DIAGRAMME D'ACTIVITÉ UML

### Calcul du volume d'une sphère

```
def cube(n):
    return n**3

def volumeSphere(r):
    return 4 * 3.1416 * cube(r) / 3

r = input('Entrez la valeur du rayon : ')
print('Le volume de cette sphère vaut', volumeSphere(r))
```



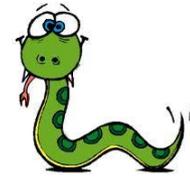
Un autre exemple, notez ici que la fonction ne renvoie aucun résultat au programme appelant :

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  # Swinnen exo 7.2_2010 : mise en oeuvre de fonction
5
6  # Fonction qui trace une ligne de n caracteres ca
7  def ligneCar(n,ca):
8      i = 0
9      while i < n:
10         print(ca, end="")
11         i = i + 1
12
13  ## Programme principal
14
15  # Saisie du nombre de caracteres
16  nombre = input ("Veillez entrer votre nombre de caracteres : ")
17  nombre = int(nombre)
18
19  # Saisie du caractere désiré
20  caractere = input("Donner votre caractere : ")
21
22  # Appel de la fonction
23  ligneCar(nombre,caractere)
```

Swinnen\_chapitre7\_exo2.py

```
>>> (executing lines 1 to 23 of "Swinnen_chapitre7_exo2.py")
Veillez entrer votre nombre de caracteres : 5
Donner votre caractere : e
eeee
```





## 1.4 Résumé de l'organisation d'un programme python <sup>3</sup>

```
# -*- coding:Utf8 -*-

#####
# Programme Python type #
# auteur : G.Swinen, Liège, 2009 #
# licence : GPL #
#####

#####
# Importation de fonctions externes :

from math import sqrt

#####
# Définition locale de fonctions :

def occurrences(car, ch):
    "Cette fonction renvoie le \
    nombre de caractères <car> \
    contenus dans la chaîne <ch>"

    nc = 0
    i = 0
    while i < len(ch):
        if ch[i] == car:
            nc = nc + 1
        i = i + 1
    return nc

#####
# Corps principal du programme :

print("Veuillez entrer un nombre :")
nbr = eval(input())

print("Veuillez entrer une phrase :")
phr = input()
print("Entrez le caractère à compter :")
cch = input()

no = occurrences(cch, phr)
rc = sqrt(nbr**3)

print("La racine carrée du cube", end=' ')
print("du nombre fourni vaut", end=' ')
print(rc)

print("La phrase contient", end=' ')
print(no, "caractères", cch)
```

Un programme Python contient en général les blocs suivants, dans l'ordre :

- Quelques instructions d'initialisation (importation de fonctions et/ou de classes, définition éventuelle de variables globales).
- Les définitions locales de fonctions et/ou de classes.
- Le corps principal du programme.

Le programme peut utiliser un nombre quelconque de fonctions, lesquelles sont définies localement ou importées depuis des modules externes.

Vous pouvez vous-même définir de tels modules.

La définition d'une fonction comporte souvent une liste de PARAMÈTRES.

Ce sont toujours des VARIABLES, qui recevront leur valeur lorsque la fonction sera appelée.

Une boucle de répétition de type 'while' doit toujours inclure au moins quatre éléments :

- l'initialisation d'une variable 'compteur' ;
- l'instruction while proprement dite, dans laquelle on exprime la condition de répétition des instructions qui suivent ;
- le bloc d'instructions à répéter ;
- une instruction d'incréméntation du compteur.

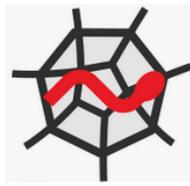
La fonction "renvoie" toujours une valeur bien déterminée au programme appelant.

Si l'instruction 'return' n'est pas utilisée, ou si elle est utilisée sans argument, la fonction renvoie un objet vide : 'None'.

Le programme qui fait appel à une fonction lui transmet d'habitude une série d'ARGUMENTS, lesquels peuvent être des valeurs, des variables, ou même des expressions.



<sup>3</sup> [https://python.developpez.com/cours/apprendre-python3/?page=page\\_9](https://python.developpez.com/cours/apprendre-python3/?page=page_9)



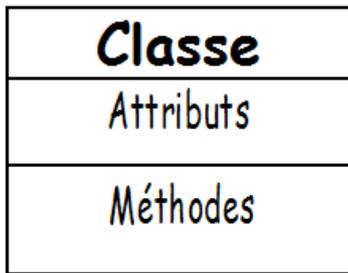
## 2 La programmation orientée objet

### 2.1 Généralités

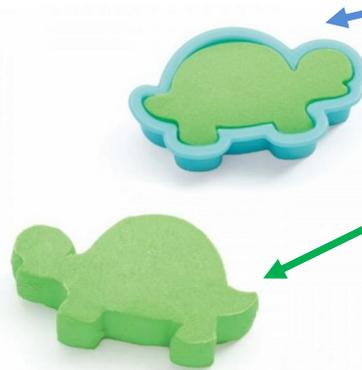
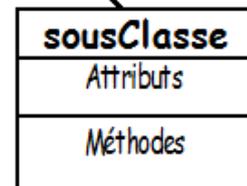
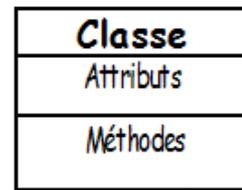
Les classes sont les principaux outils de la programmation orientée objet POO ou Object Oriented Programming OOP.

La classe permet de créer des structures appelées objets.

Ces objets possèdent des attributs, ensemble de valeurs qui les caractérisent et des méthodes qui donnent des comportements aux objets.



La classe (en bleu)



Crée l'objet

Les diagrammes de classe UML<sup>4</sup> permettent de décrire les classes et leurs relations :

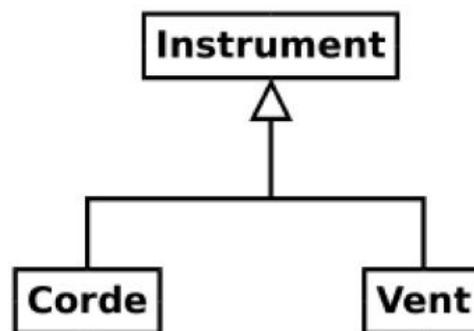
Une classe se compose :

- au moins d'un nom
- d'une zone pour les attributs
- d'une zone pour les méthodes

UneClasse



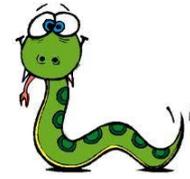
(a) Une classe et une



Le **Langage de Modélisation Unifié**, de l'anglais *Unified Modeling Language (UML)*, est un langage de modélisation graphique à base de pictogrammes conçu pour fournir une méthode normalisée pour visualiser la conception d'un système. Il est couramment utilisé en développement logiciel et en conception orientée objet.



<sup>4</sup> [https://fr.wikipedia.org/wiki/UML\\_\(informatique\)](https://fr.wikipedia.org/wiki/UML_(informatique))



## 2.2 Résumé : définition et utilisation de classes (Swinen)

La classe est un moule servant à produire des objets.  
Chacun d'eux sera une instance de la classe considérée.

```
#####
# Programme Python type #
# auteur : G.Swinen, Liège, 2009 #
# licence : GPL #
#####
```

Les instances de la classe Point() seront des objets très simples qui posséderont seulement un attribut 'x' et un attribut 'y'; ils ne seront dotés d'aucune méthode.

```
class Point(object):
    """point géométrique"""
    def __init__(self, x, y):
        self.x = x
        self.y = y
```

Le paramètre SELF désigne toutes les instances qui seront produites à partir de cette classe.

```
class Rectangle(object):
    """rectangle"""
    def __init__(self, ang, lar, hau):
        self.ang = ang
        self.lar = lar
        self.hau = hau
```

Les instances de la classe Rectangle() posséderont trois attributs. Le premier ('ang') doit être lui-même un objet de classe Point(). Il servira à mémoriser les coordonnées de l'angle supérieur gauche du rectangle. Les deux autres contiendront sa largeur et sa hauteur.

```
def trouveCentre(self):
    xc = self.ang.x + self.lar /2
    yc = self.ang.y + self.hau /2
    return Point(xc, yc)
```

La classe Rectangle() comporte une méthode, qui renverra un objet de classe Point() au programme appelant.

```
class Carre(Rectangle):
    """carré = rectangle particulier"""
    def __init__(self, coin, cote):
        Rectangle.__init__(self,
            coin, cote, cote)
        self.cote = cote

    def surface(self):
        return self.cote**2
```

Carre() est une classe dérivée, qui hérite les attributs et méthodes de la classe Rectangle().

Son constructeur doit faire appel au constructeur de la classe parente, en lui transmettant la référence de l'instance en cours de création (self) comme premier argument.

```
#####
## Programme principal : ##
```

La classe Carre() comporte une méthode de plus que sa classe parente.

```
# coord. de 2 coins sup. gauches :
csgR = Point(40,30)
csgC = Point(10,25)
```

Pour créer (ou instancier) un objet, il suffit d'appeler une classe comme on appelle une fonction. La valeur renvoyée est une nouvelle instance de cette classe. Les instructions ci-contre créent donc deux objets de la classe Point() ...

```
# "boîtes" rectangulaire et carrée :
boiteR = Rectangle(csgR, 100, 50)
boiteC = Carre(csgC, 40)
```

... et celles-ci, encore deux autres objets.

```
# Coordonnées du centre pour chacune :
cR = boiteR.trouveCentre()
cC = boiteC.trouveCentre()
```

```
print("centre du rect. :", cR.x, cR.y)
print("centre du carré :", cC.x, cC.y)
```

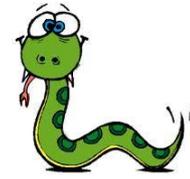
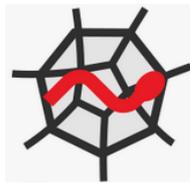
Note : par convention, on donne aux classes des noms commençant par une majuscule.

```
print("surf. du carré :", end=' ')
print(boiteC.surface())
```

La méthode trouveCentre() fonctionne pour les objets des deux types, puisque la classe Carre() a hérité de la classe Rectangle().

La méthode surface(), par contre, ne peut être invoquée que pour les objets Carre().





### 2.3 Autre exemple un peu de chimie

(D'après Swinnen)

Pour bien comprendre le script ci-dessous, il faut cependant d'abord vous rappeler quelques notions élémentaires de chimie.

```
#####
# PREMIERS PAS AVEC LES CLASSES #
# (C) Gérard Swinnen, 2015 CHAPITRE 14 P.158 #
# (C) Gérard Swinnen, 2010 CHAPITRE 11 P.190 #
#####
```

Dans votre cours de chimie, vous avez certainement dû apprendre que les atomes sont des entités, constitués d'un certain nombre de protons (particules chargées d'électricité positive), d'électrons (chargés négativement) et de neutrons (neutres).

Le type d'atome (ou élément) est déterminé par le nombre de protons, que l'on appelle également numéro atomique. Dans son état fondamental, un atome contient autant d'électrons que de protons, et par conséquent il est électriquement neutre. Il possède également un nombre variable de neutrons, mais ceux-ci n'influencent en aucune manière la charge électrique globale.

Dans certaines circonstances, un atome peut gagner ou perdre des électrons. Il acquiert de ce fait une charge électrique globale, et devient alors un ion (il s'agit d'un ion négatif si l'atome a gagné un ou plusieurs électrons, et d'un ion positif s'il en a perdu).

La charge électrique d'un ion est égale à la différence entre le nombre de protons et le nombre d'électrons qu'il contient.

Le script reproduit à la page suivante génère des objets *Atome()* et des objets *Ion()*. Nous avons rappelé ci-dessus qu'un ion est simplement un atome modifié. Dans notre programmation la classe qui définit les objets *Ion()* sera donc une classe dérivée de la classe *Atome()* : elle héritera d'elle tous ses attributs et toutes ses méthodes, en y ajoutant les siennes propres.

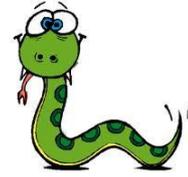
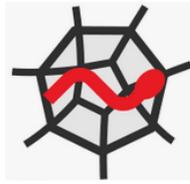
L'une de ces méthodes ajoutée (la méthode *affiche()*) remplace une méthode de même nom héritée de la classe *Atome()*. Les classes *Atome()* et *Ion()* possèdent donc chacune une méthode de même nom, mais qui effectuent un travail différent. On parle dans ce cas de polymorphisme. On pourra dire également que la méthode *affiche()* de la classe *Atome()* a été surchargée.

Il sera évidemment possible d'instancier un nombre quelconque d'atomes et d'ions à partir de ces deux classes.

Or l'une d'entre elles, la classe *Atome()*, doit contenir une version simplifiée du tableau périodique des éléments (tableau de Mendeleïev), de façon à pouvoir attribuer un nom d'élément chimique, ainsi qu'un nombre de neutrons, à chaque objet généré. Comme il n'est pas souhaitable de recopier tout ce tableau dans chacune des instances, nous le placerons dans un attribut de classe.

Ainsi ce tableau n'existera qu'en un seul endroit en mémoire, tout en restant accessible à tous les objets qui seront produits à partir de cette classe.





[Script atomes-ions :](#)

 Swinnen\_Classes\_Atomes\_Ions.py

```
class Atome:
    """atomes simplifiés, choisis parmi les 10 premiers éléments du TP"""
    table = [None, ('hydrogène', 0), ('hélium', 2), ('lithium', 4),
             ('béryllium', 5), ('bore', 6), ('carbone', 6), ('azote', 7),
             ('oxygène', 8), ('fluor', 10), ('néon', 10)]

    def __init__(self, nat):
        """le n° atomique détermine le n. de protons, d'électrons et de neutrons"""
        self.np, self.ne = nat, nat          # nat = numéro atomique
        self.nn = Atome.table[nat][1]

    def affiche(self):
        print()
        print("Nom de l'élément :", Atome.table[self.np][0])
        print("%s protons, %s électrons, %s neutrons" % \
              (self.np, self.ne, self.nn))

class Ion(Atome):
    """les ions sont des atomes qui ont gagné ou perdu des électrons"""

    def __init__(self, nat, charge):
        """le n° atomique et la charge électrique déterminent l'ion"""
        Atome.__init__(self, nat)
        self.ne = self.ne - charge
        self.charge = charge

    def affiche(self):
        Atome.affiche(self)
        print("Particule électrisée. Charge =", self.charge)

### Programme principal : ###

a1 = Atome(5)
a2 = Ion(3, 1)
a3 = Ion(8, -2)
a1.affiche()
a2.affiche()
a3.affiche()
```

Résultats du script :

```
Nom de l'élément : bore
5 protons, 5 électrons, 6 neutrons

Nom de l'élément : lithium
3 protons, 2 électrons, 4 neutrons
Particule électrisée. Charge = 1

Nom de l'élément : oxygène
8 protons, 10 électrons, 8 neutrons
Particule électrisée. Charge = -2
```



N'hésitez surtout pas à essayer ces classes !

Le code est fourni.



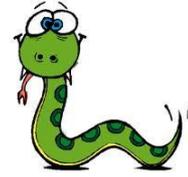
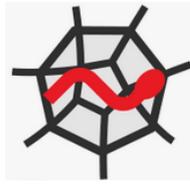
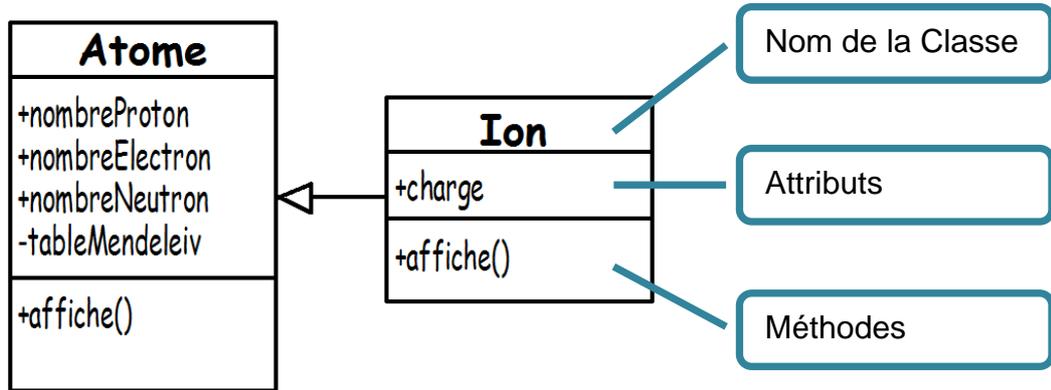
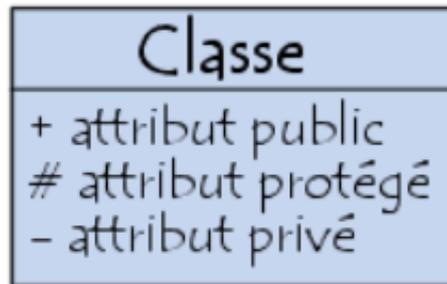


Diagramme de classes en UML du script Atome-Ion :



Dans ces diagrammes il faut préciser le statut des différents attributs. Les conventions en UML sont les suivantes :



Comment représenter les attributs dans les diagrammes de classes UML

Pour le langage Python il faut garder présent à l'esprit qu'il n'y a pas véritablement d'attribut privé<sup>5</sup>



Attention tout est accessible en python, il n'y a pas véritablement d'attribut privé. Il est donc conseillé d'utiliser la convention ci-dessous surtout pour prendre de bonnes habitudes compatibles avec d'autres langages de programmation objet.

Pour rendre privé un attribut il faut mettre un double underscore devant son nom.

```

1 class Monique:
2     def __init__(self):
3         self.__private = "lessons"
    
```

Pour lire la valeur d'un attribut privé on utilisera une méthode appelée accesseur ou getter.  
 Pour modifier la valeur d'un attribut privé on utilisera une méthode appelée mutateur ou setter.



<sup>5</sup> <http://sametmax.com/variables-privées-en-python/>

## 3 Mise en œuvre de Classes

### 3.1 Un exercice gestion de comptes bancaires

Cet exemple issu d'un site bien connu<sup>6</sup> cité en référence propose la gestion de comptes bancaires avec une class Compte présentée ci-dessous :

```
class Compte:
    """gestion d'un compte bancaire"""
    nbreCompte = 0 # Attribut de classe

    # Constructeur de classe
    def __init__(self,nom,numero,valeur):
        self.__nom = nom
        self.__numero = numero
        self.__solde = valeur
        Compte.nbreCompte += 1

    #
    def getSolde(self):
        return self.__solde

    #
    def getNumero(self):
        return self.__numero

    #
    def getNom(self):
        return self.__nom

    #
    def setSolde(self,x):
        if x>=0:
            self.__solde = x
        else:
            print("Transaction refusée")

    #
    def setNom(self,nom):
        self.__nom = nom

    #
    def setNumero(self,num):
        self.__numero = num

    # Méthode définie pour la classe elle même
    # indiquée avec le décorateur @classmethod.
    # Elle reçoit la classe en premier argument
    # qui devient implicite voir les exemples d'appel
    # plus bas.
    @classmethod
    def afficherNbreComptes(cls):
        print("Nombre de comptes : ",cls.nbreCompte)

    #
    def __del__(self):
        Compte.nbreCompte -= 1

    #
    def crediter(self,x):
        self.__solde += x

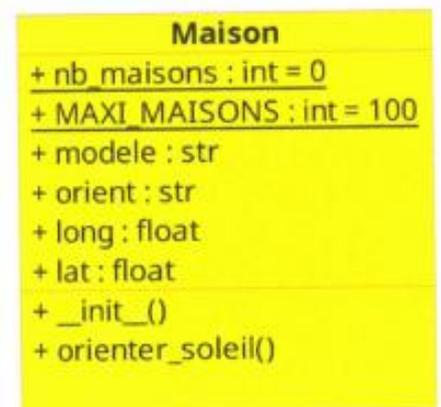
    #
    def afficherSolde(compte):
        print("Le solde vaut : %7.2f €" % (compte.getSolde()))
```

### 3.2 Travail à faire :

 **Q1.** Complétez le code du script python en y ajoutant les commentaires pour toutes les méthodes en précisant en particulier : les mutateurs, les accesseurs, le destructeur, le constructeur, les méthodes simples, les fonctions indépendantes à la définition de la classe.

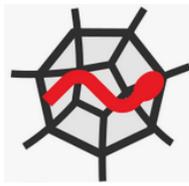
 POO\_compte\_bancaire\_SUPINFO\_eleve.py

**Q2.** Préciser les attributs de la classe Compte et en particulier le type public, privé et attribut de classe



Les attributs de classes sont indiqués en soulignés dans les diagrammes UML.

<sup>6</sup> <https://www.supinfo.com/cours/2OOP/chapitres/02-classes-objets>



**Q3.** Compléter le programme principal pour faire fonctionner cette classe. Obtenir le résultat ci-dessous :

Création du compte A

Création du compte B

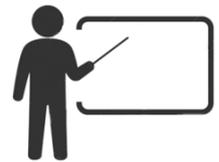
```
Premier compte intitulé : proA
                           numéro : 4500777
                           Solde : 10000
```

```
Premier compte intitulé : proB
                           numéro : 4500802
                           Solde : 5000
```

```
Ajout de 250 € sur le compte B
Le solde vaut : 5250.00 €
Transaction refusée
Le solde vaut : 5250.00 €
```



Script\_POO\_1. Rendre le programme commenté, faites vérifier le fonctionnement par votre professeur.



### 3.3 Un peu de géométrie

#### Une classe Point de départ

 Classe\_Point\_Cours.py

```
#####
# PREMIERS PAS AVEC LES CLASSES #
# (C) Gérard Swinnen, 2015 CHAPITRE 13 P.142 #
# (C) Gérard Swinnen, 2010 CHAPITRE 11 P.191 #
# Exemple codé P.G #
#####

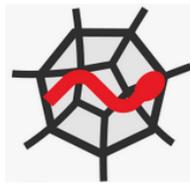
from math import sqrt

def calculDistance(a,b):
    """Calcul de la distance entre deux points"""
    return sqrt((a.x-b.x)**2 + (a.y-b.y)**2)

## DEFINITION D'UNE CLASSE POINT
# La création d'une instance (ou d'un objet) créé dans l'espace de nom
# de l'objet deux attributs d'instance x et y.
# C'est le constructeur __init__ qui s'en charge automatiquement
# une valeur initiale est proposée x=0 et y=0
class Point:
    """Définition d'un point géométrique """

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





Le programme principal donne un exemple d'utilisation de cette classe point :

#### ## PROGRAMME PRINCIPAL SI UTILISE EN DIRECT

```
if __name__=='__main__':
    print("Création d'un point p1 de coordonnées (1,4)")
    p1 = Point(1,4)
    print("Documentation intégrée à l'objet : ",p1.__doc__)
    print("Création d'un point p2 de coordonnées (4,6)")
    p2 = Point(4,6)
    print("Calcul de la distance p1-p2 %7.3f " % (calculDistance(p1,p2)))
    print("Documentation intégrée de la fonction calculDistance : ",calculDistance.__doc__)
    print("Coordonnées du point p1")
    print(p1.x)
    print(p1.y)
    print("Création d'un nouvel objet référencé par p2")
    p2 = Point(8)
    print("Nouvelles coordonnées")
    print(p2.x)
    print(p2.y)
```

Résultat :

```
>>> (executing lines 1 to 46 of "Classe_Point_Cours.py")
Création d'un point p1 de coordonnées (1,4)
Documentation intégrée à l'objet : Définition d'un point géométrique
Création d'un point p2 de coordonnées (4,6)
Calcul de la distance p1-p2 3.606
Documentation intégrée de la fonction calculDistance : Calcul de la distance entre deux points
Coordonnées du point p1
1
4
Création d'un nouvel objet référencé par p2
Nouvelles coordonnées
8
0
```

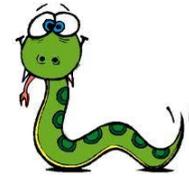
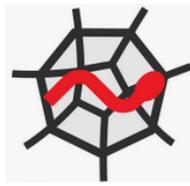
### Ajout d'objets géométriques



Script\_POO\_2. Définissez une classe Cercle(). Les objets construits à partir de cette classe seront des cercles de tailles variées. En plus de la méthode constructeur (qui utilisera donc un paramètre rayon), vous définirez une méthode surface(), qui devra renvoyer la surface du cercle.

Vous pouvez vous inspirer de l'exemple [ici](#) dans ce document.





Script\_POO\_3. Définissez ensuite une classe `Cylindre()` dérivée de la précédente. Le constructeur de cette nouvelle classe comportera les deux paramètres rayon et hauteur. Vous y ajouterez une méthode `volume()` qui devra renvoyer le volume du cylindre (rappel : volume d'un cylindre = surface de section x hauteur).

Exemple de résultats attendus à faire vérifier par votre professeur:

```
>>> cyl = Cylindre(5, 7)
>>> print(cyl.surface())
78.54
>>> print(cyl.volume())
549.78
```



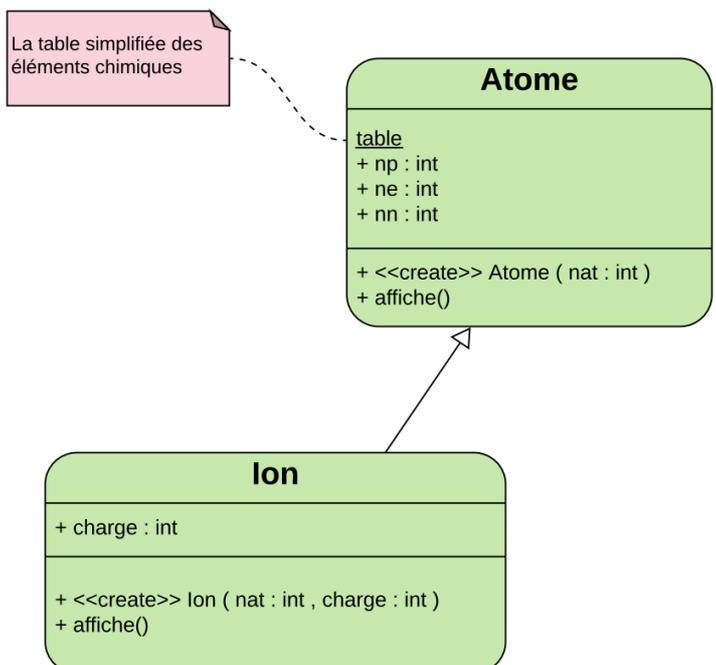
## 4 Un outil pour tracer des diagrammes UML

[https://www.lucidchart.com/documents#docs?folder\\_id=home&browser=icon&sort=saved-desc](https://www.lucidchart.com/documents#docs?folder_id=home&browser=icon&sort=saved-desc)

L'inscription est gratuite pour le monde de l'éducation.

<https://www.lucidchart.com/fr-FR/users/login>

Exemple de résultat pour la classe Atome Ion :

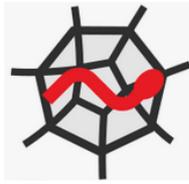


On remarque que le type des attributs est indiqué.

Citons quelques types courants :

- int : pour un nombre entier
- float : pour un nombre réel
- str : pour une chaîne de caractère

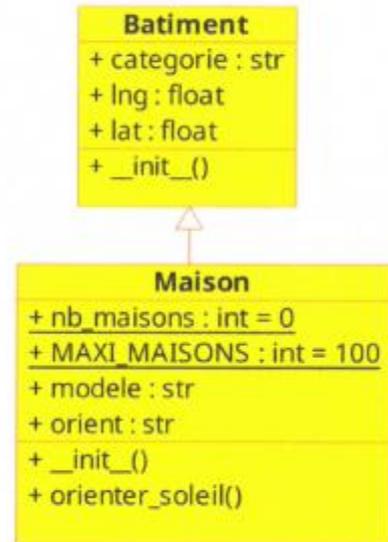
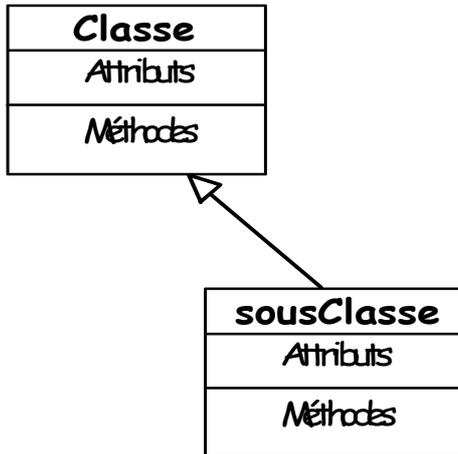




## 5 Conception POO

### 5.1 L'héritage

On peut créer une nouvelle classe à partir d'une classe existante. La relation est indiquée par une flèche vide :

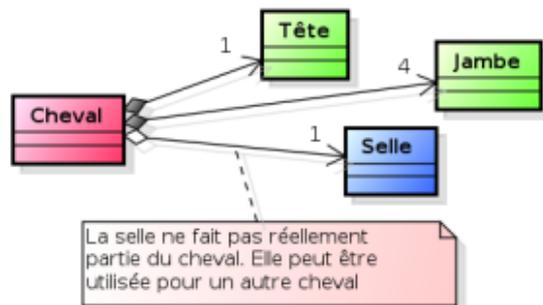


La flèche pointe vers la classe parente.

### 5.2 Associations : composition – agrégation

- **La composition** indique qu'un objet **A** (appelé conteneur) est constitué d'un autre objet **B**. Cet objet **A** n'appartient qu'à l'objet **B** et ne peut pas être partagé avec un autre objet. C'est une relation très forte, si l'objet **A** disparaît, alors l'objet **B** disparaît aussi. **Un cheval possède une tête et 4 jambes. Elle se représente par un losange plein du côté de l'objet conteneur**

La manière habituelle d'implémenter la composition en C++ se fait soit directement au travers d'un attribut normal en utilisant la liste d'initialisation pour la création de l'objet composite, soit à l'aide d'une variable dynamique, auquel cas, nous ne devons pas oublier de rajouter un destructeur pour libérer cette variable dynamique.



La selle ne fait pas réellement partie du cheval. Elle peut être utilisée pour un autre cheval

- **L'agrégation** indique qu'un objet **A** possède un autre objet **B**, mais contrairement à la composition, l'objet **B** peut exister indépendamment de l'objet **A**. La suppression de l'objet **A** n'entraîne pas la suppression de l'objet **B**. L'objet **A** est plutôt à la fois possesseur et utilisateur de l'objet **B**. **Un cheval possède une selle sur son dos. Elle se représente par un losange vide du côté de l'objet conteneur.**

Extrait de l'excellente présentation<sup>7</sup> du langage UML sur <http://remy-manu.no-ip.biz/>

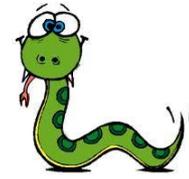
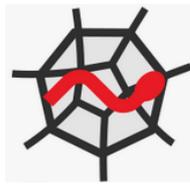
### 5.3 Travail à faire :



- Q4.** Réaliser le diagramme UML de vos classes Cercle et Cylindre. Le fournir au format pdf.



<sup>7</sup> <http://remy-manu.no-ip.biz/UML/Cours/coursUML3.pdf>



## 6 Résumé

---

### 6.1 Quelques définitions du domaine de la POO<sup>8</sup>



#### Classe, Attributs, Méthodes, Accesseur et mutateurs

- Le type de données avec ses **caractéristiques** et ses **actions** possibles s'appelle **classe**.
- Les **caractéristiques (ou variables)** de la classe s'appellent les **attributs**.
- Les **actions possibles** à effectuer avec la classe s'appellent les **méthodes**.
- La **classe** définit donc les **attributs** et les actions possibles sur ces attributs, les **méthodes**.
- **Constructeur** : la manière « normale » de spécifier l'initialisation d'un objet est d'écrire un constructeur .
- **L'encapsulation** désigne le principe de **regrouper des données brutes** avec un ensemble de **routines (méthodes)** permettant de les lire ou de les manipuler.
- **Accesseur** ou « **getter** » : une fonction qui retourne la valeur d'un attribut de l'objet. Par convention son nom est généralement sous la forme : `getNom_attribut()`.
- Un **Mutateur** ou **setter** : une procédure qui permet de modifier la valeur d'un attribut d'un objet. Son nom est généralement sous la forme : `setNom_attribut()`.

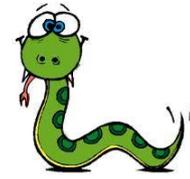
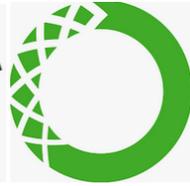
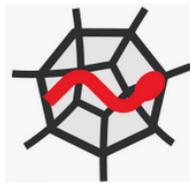
### 6.2 Brève historique de la POO

La programmation orientée objet (POO), ou programmation par objet, est un paradigme de programmation informatique élaboré par les Norvégiens Ole-Johan Dahl et Kristen Nygaard au début des années 1960 et poursuivi par les travaux de l'Américain Alan Kay dans les années 1970. Il consiste en la définition et l'interaction de briques logicielles appelées objets. La programmation orientée objet a été introduite par Alan Kay avec Smalltalk. Toutefois, ses principes n'ont été formalisés que pendant les années 1980 et, surtout, 1990.



---

<sup>8</sup> D'après [https://www.math93.com/images/pdf/NSI/terminale/NSI\\_Classes.pdf](https://www.math93.com/images/pdf/NSI/terminale/NSI_Classes.pdf)



## 7 Exercice de récapitulation du cours<sup>9</sup>

Compléter (sur cahier) le code de définition des deux classes décrites page suivante en répondant aux questions ci-dessous :

### 7.1 Questions de compréhension



- Q5.** Donner les noms et types des attributs de la classe *Piece*.
- Q6.** Donner les noms et types des attributs de la classe *Appartement*.
- Q7.** Donner le code du mutateur de la classe *Piece*.
- Q8.** Donner le code de la méthode *ajouter* de la classe *Appartement*
- Q9.** Donner le code de la méthode *nbPieces* de la classe *Appartement*
- Q10.** Donner le code de la méthode *surfaceTotale* de la classe *Appartement*
- Q11.** Donner le code de la méthode *getListePieces* de la classe *Appartement*

### 7.2 Utilisation des classes

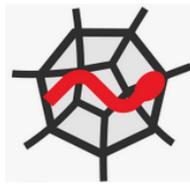
Écrire (sur cahier) un programme principal utilisant ces deux classes qui va :



- Q12.** Créer une pièce « chambre1 », de surface 20 m2 et une pièce « chambre2 », de surface 15 m2.
- Q13.** Créer une pièce « séjour », de surface 25 m2 et une pièce « sdb », de surface 10 m2.
- Q14.** Créer une pièce « cuisine », de surface 12 m2.
- Q15.** Créer un appartement « monappartement » qui contiendra toutes les pièces créées.
- Q16.** Afficher la surface totale de l'appartement créé.
- Q17.** Afficher la liste des pièces et surfaces de l'appartement créé.



<sup>9</sup> Toujours selon [https://www.math93.com/images/pdf/NSI/terminale/NSI\\_Classes.pdf](https://www.math93.com/images/pdf/NSI/terminale/NSI_Classes.pdf) pour 7.2



## Les classes Pièce et Appartement

```
# Exercice_P00_Papier_1.py
class Piece:
    # nom est une string et surface est un float
    def __init__(self,nom,surface):
        self.nom=nom
        self.surface=surface

    # Accesseurs: retournent les attributs d'un objet de cette classe
    def getSurface(self):
        return self.surface

    def getNom(self):
        return self.nom

    # Mutateur
    def setSurface(self,s): # s est un float,
        ...

class Appartement:
    # nom est une string
    def __init__(self,nom):
        # L'objet est une liste de pièces
        # (objets issus de la classe Piece)
        self.listeDePieces=[]
        self.nom=nom

    # Accesseurs:
    def getNom(self):
        return self.nom

    # pour ajouter une pièce de classe Piece
    def ajouter(self,piece):
        ...

    # pour avoir le nombre de pièces de l'appartement
    def nbPieces(self): #
        ...

    # retourne la surface totale de l'appartement (un float)
    def surfaceTotale(self):
        ...

    # retourne la liste des pièces avec les surfaces
    def getListePieces(self): # sous forme d'une liste de tuples
        ...
```

