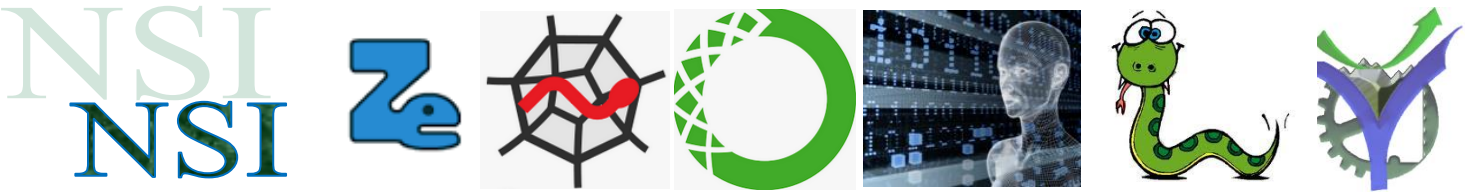
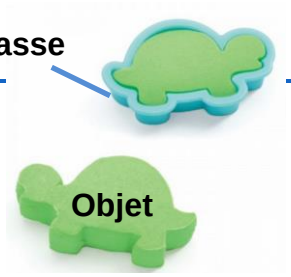


Initiation à la POO



Programmation Orientée Objet

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

1 Les différentes façons de produire un programme[1]

1.1 Quelques techniques

Deux techniques de production de programme

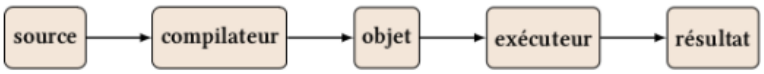


Figure 1.1 - Chaîne de compilation

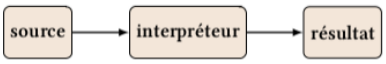


Figure 1.2 - Chaîne d'interprétation

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.

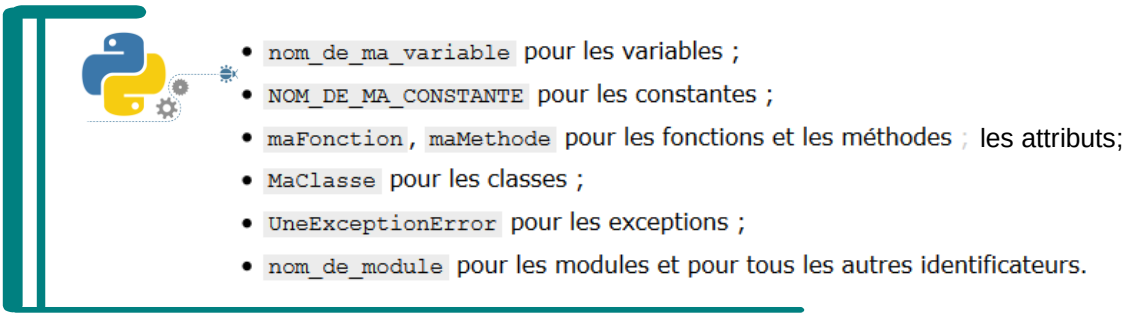
1.2 Différentes méthodologies ou paradigmes de programmation

Il y a plusieurs méthodes de construction des programmes citons-en deux principales[2] :

- la méthodologie procédurale ou impérative. 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 en termes d'actions successives à réaliser par le processeur.

- la méthodologie déclarative. Dans ce cadre on décrit ce que l'on souhaite obtenir comme résultats du traitement de l'information. Cette description est réalisée à l'aide d'une syntaxe appropriée. Exemple le langage d'interrogation de bases de données SQL, ou bien les expressions régulières.

- 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.



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

⇦

[1] Voir ces liens utilisés pour ce cours https://python.developpez.com/cours/apprendre-python-3/
https://python.developpez.com/cours/apprendre-python3/

[2] 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 :

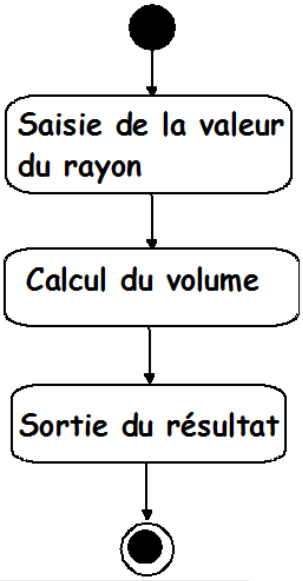


DIAGRAMME D'ACTIVITÉ UML

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(float(r)))
```

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

Swinnen_chapitre7_exo2.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# Swinnen exo 7.2_2010 : mise en oeuvre de fonction

# Fonction qui trace une ligne de n caracteres ca
def ligneCar(n,ca):
    i = 0
    while i < n:
        print(ca, end="")
        i = i + 1

## Programme principal

# Saisie du nombre de caracteres
nombre = input ("Veuillez entrer votre nombre de caracteres : ")
nombre = int(nombre)

# Saisie du caractere désiré
caractere = input("Donner votre caractere : ")

#☻ Appel de la fonction
ligneCar(nombre,caractere)
```

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

⇦

1.4 Résumé de l'organisation d'un programme python type[3]

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

#####################################
# Programme Python type             #
# auteur : G.Swinnen, 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'incrémentation 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.

[3] https://python.developpez.com/cours/apprendre-python3/?page=page_9

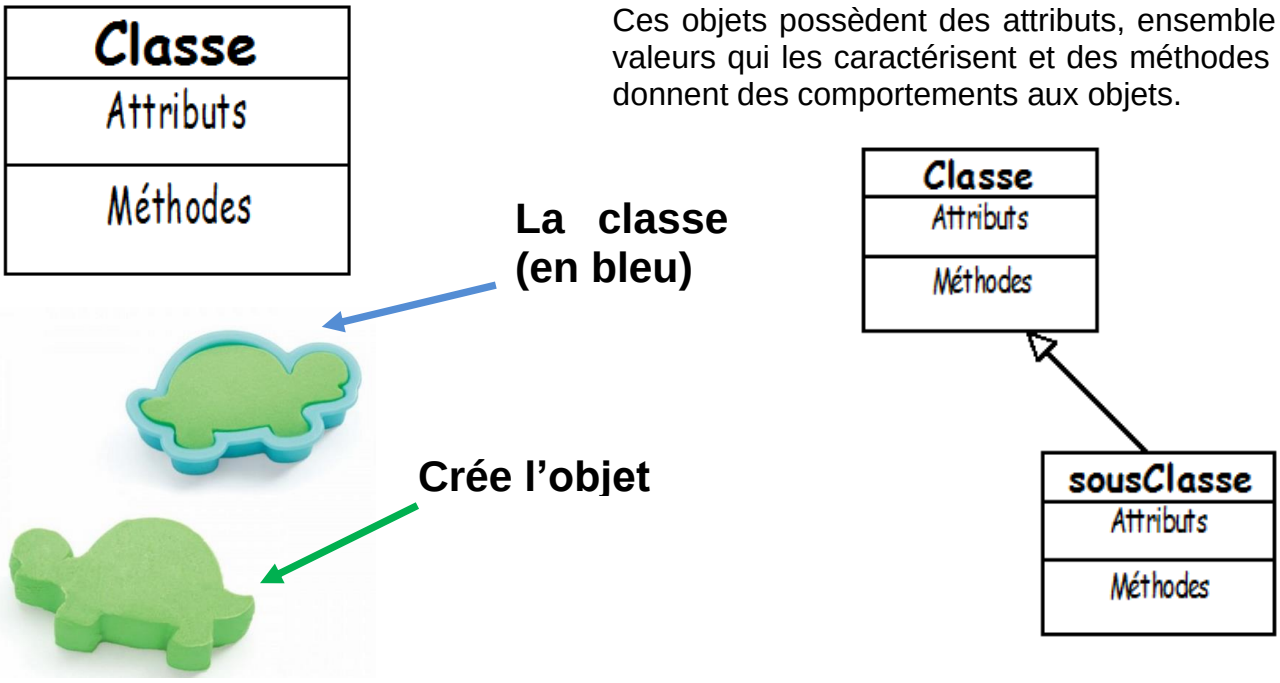
2 La programmation orientée objet

2.1 Généralités

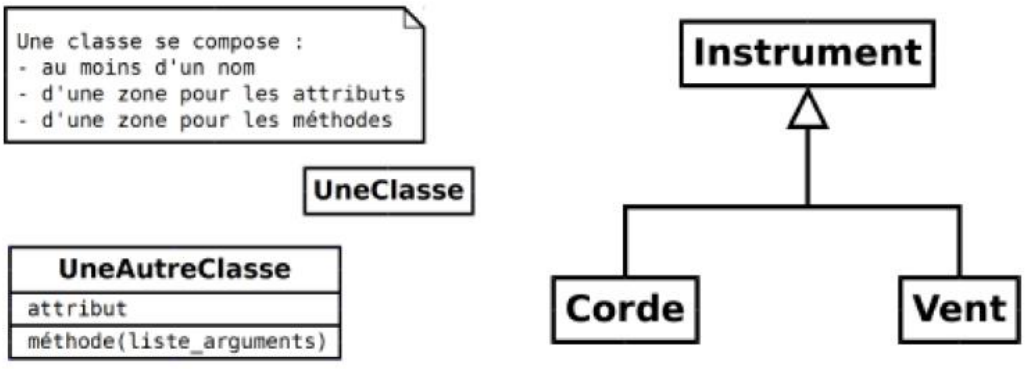
Les classes sont les principaux outils de la programmation orientée objet POO ou Object Oriented Programming OOP.

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.



Les diagrammes de classe UML[4] permettent de décrire les classes et leurs relations :

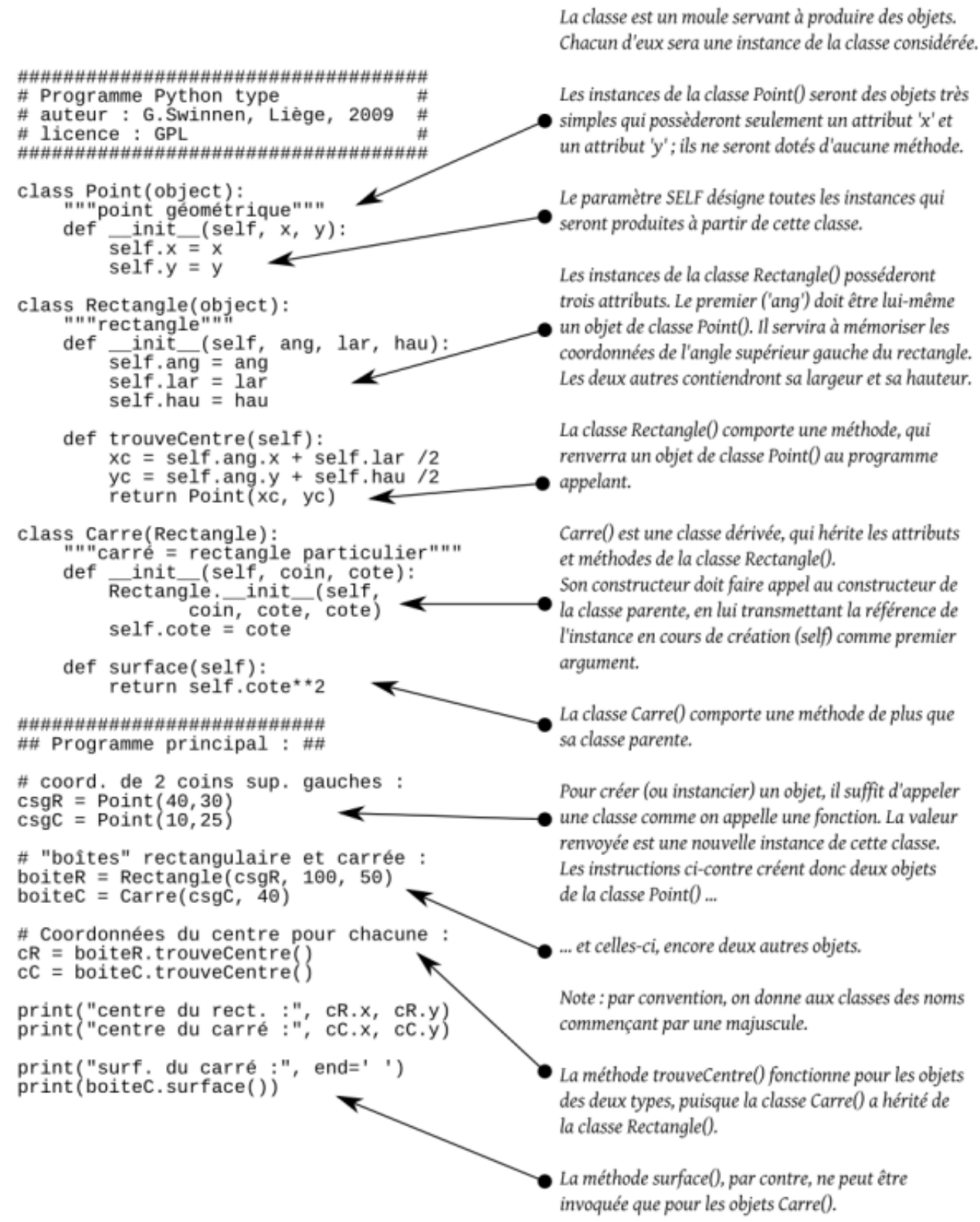


(a) Une classe et une note (b) Un héritage

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.

[4] https://fr.wikipedia.org/wiki/UML_(informatique)

2.2 Résumé : définition et utilisation de classes (Swinnen)



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

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

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

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

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

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

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

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

# 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)

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

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.

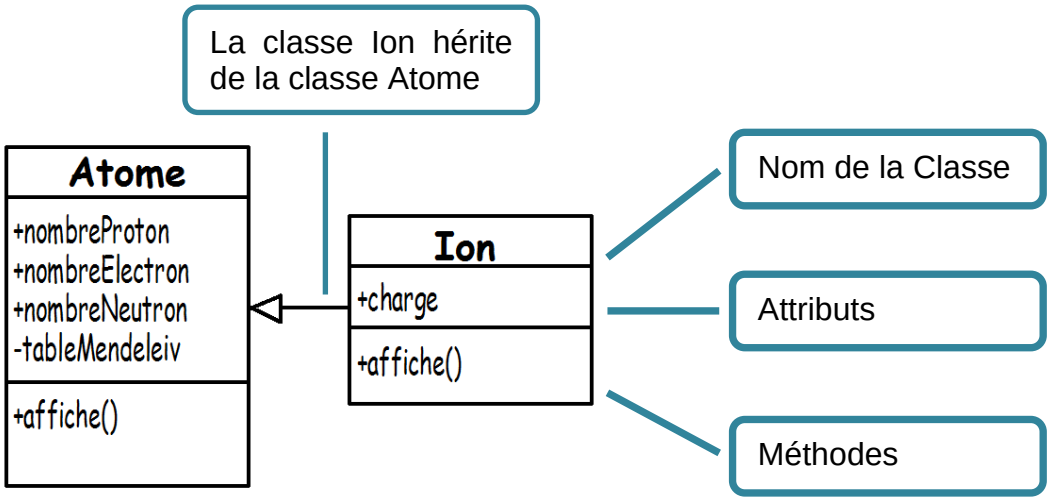
Script atomes-ions : Swinnen_Classes_Atomes_Ions.py

```
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()
```

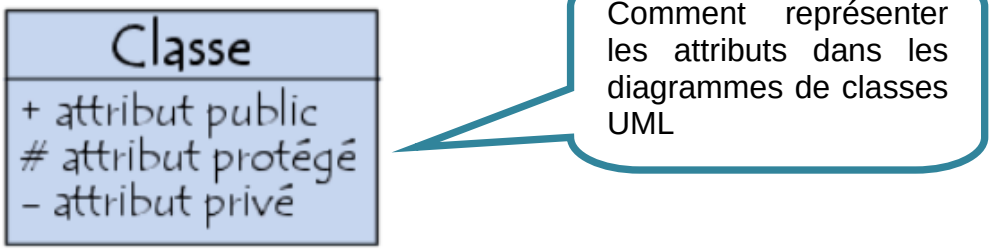
Résultats du script :

```
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
```

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 :



Attribut publique	Attribut protégé	Attribut privé	Attribut de classe
Symbole +	**Symbole #**	**Symbole -**	**Symbole souligné**
Accessible à toutes les autres classes du programme.	Accessible uniquement aux classes dérivées.	Accessible uniquement à la classe.	Attribut commun à tous les objets instanciés par cette classe.

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.

Pour le langage Python il faut garder présent à l'esprit qu'il n'y a pas véritablement d'attribut privé[5][6]

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 indiquer qu'un attribut est privé il faut mettre un double underscore devant son nom.

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

[5] http://sametmax.com/variables-privees-en-python/
[6] Voir le complément de cours NSI_POO_ATTRIBUTS_PRIVES.pdf

3 Mise en œuvre de Classes

3.1 Un exercice gestion de comptes bancaires

Cet exemple issu d'un site bien connu[7] 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

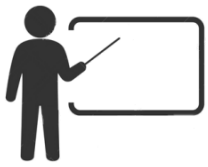
Q2. Préciser les attributs de la classe Compte et en particulier le type public, privé et attribut de classe

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

[7] 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

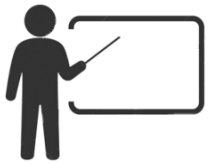
Script_POO_2. Définissez une classe Cercle(). Les objets construits à partir de cette classe seront des cercles de tailles variées. En plus de la méthode constructeur (qui utilisera donc un paramètre rayon), vous définirez une méthode surface(), qui devra renvoyer la surface du cercle.

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 × 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
```



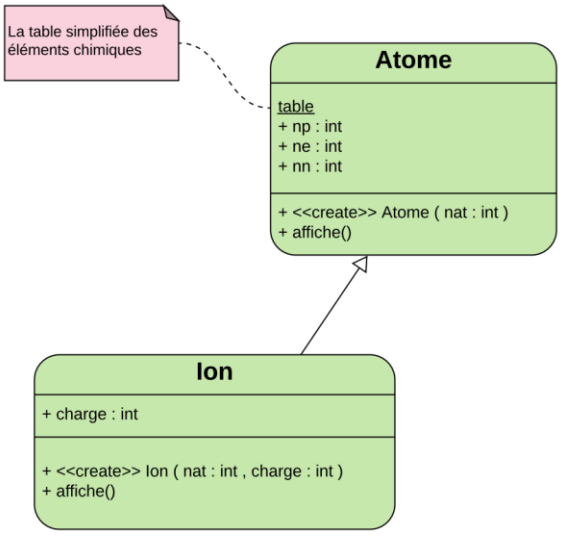
4 Un outil pour tracer des diagrammes UML

https://www.lucidchart.com/documents#docs?folder_id=home&browser=icon&sort=saved-desc

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.

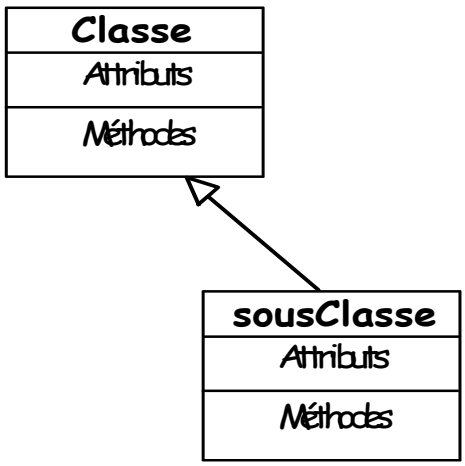
Citons quelques types courants :

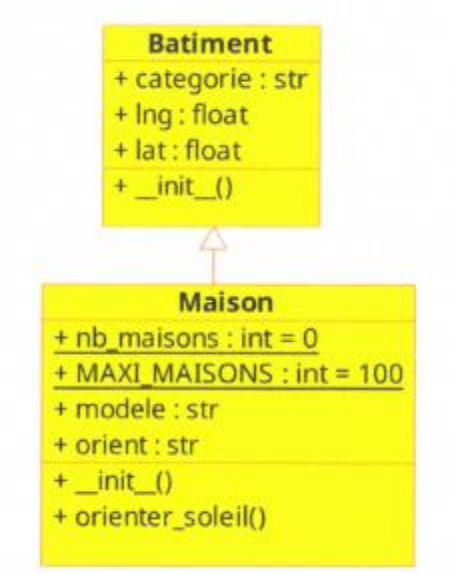
- int : pour un nombre entier
- float : pour un nombre réel
- str : pour une chaine de caractère

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 :



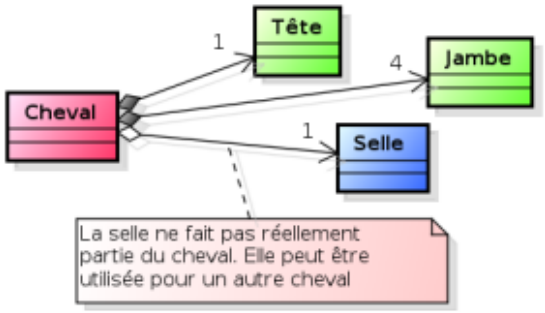


La flèche pointe vers la classe parente.

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'a 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 coté 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.



- **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 coté de l'objet conteneur.

Extrait de l'excellente présentation[8] 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.

[8] http://remy-manu.no-ip.biz/UML/Cours/coursUML3.pdf

6 Résumé

6.1 Quelques définitions du domaine de la POO[9]

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.

[9] D'après https://www.math93.com/images/pdf/NSI/terminale/NSI_Classes.pdf

7 Exercice de récapitulation du cours[10]

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.

[10] Toujours selon https://www.math93.com/images/pdf/NSI/terminale/NSI_Classes.pdf pour 7.2

Les classes Pièce et Appartement

```
#  Exercice_POO_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
    ...
```