Cours 1 : Rappels et compléments sur les structures de données

I Données et variables

1.1 Premiers éléments

Un programme d'ordinateur manipule des données que l'on est parvenu à numériser (suite finie de nombres binaires).

Pour accéder à ces données, le programme fait appel à des variables de différents types.

Une variable, c'est :

— un nom de variable, à peu près quelconque qui est une référence désignant une adresse mémoire (un emplacement précis dans la mémoire vive) ;

— une valeur dont le langage de programmation reconnait le type.

En Python, on dit que le typage est dynamique. C'est le langage lui-même qui se charge de reconnaître le type de la variable quand elle est déclarée.

1.2 Affectation des variables

Exemple 1. *Affectation simultanée*

```
x = y = 7
```

Que renvoie `>>> x, y, id(y) == id(x)` *?*

Exemple 2. *Affectation parallèle*

```
a, b, c, d = 4, 4, 8.33, 8.33
```

Que renvoie `>>> a == b, a is b, c == d, c is d` *?*

Exemple 3. *Incrémentation*
Que renvoie `>>> a = a + 1; a += 1; a + 1 = g` *?*

Principe d'affectation

```
nom_variable = valeur
```

1.3 Types de données

Exemple 4. *Données numériques*

```
a = 5; b = float(a)
```

Que renvoie `>>> a, b, a is b` *?*

Exemple 5. *Données alphanumériques*

```
phrase1 = '''C'est un beau roman'''
phrase2 = "C'est une belle histoire"
phrase3 = 'C'est une romance d'aujourd'hui'
couplet = phrase1 + '\n' + phrase2 + '\n' + phrase3
tableau = 'a \tb\t c'
```

Corriger si besoin ces lignes, puis indiquer ce que renvoie la console aux commandes suivantes.

```
print(phrase1[1])
print(phrase2[-1])
print(phrase3[2:5])
print(couplet)
print(tableau)
```

Exemple 6. *Listes*

```
liste_exemple = ['lundi', 4, 8.33, ['mot', 3]]
```

Indiquer ce que renvoie la console aux commandes suivantes.

```
print(liste_exemple[1])
print(liste_exemple[3][0])
```

Notes sur les différents types du langage

- données logiques : 'boolean' `bool`.
  ```
  >>> type(a == a)
  <class 'bool'>
  ```
- données numériques : 'integer' `int`, les nombres entiers relatifs 'floatting number' `float`, les nombres à virgule flottante ou nombre réel arrondi.
- données alphanumériques : 'string' `str`, les chaînes de caractères. Pour accéder à une portion d'une chaîne de caractères, il est possible d'utiliser la technique du 'slicing'.
- listes : list `list` Une liste est une collection d'éléments pouvant être de types variés. C'est un objet mutable, ce qui veut dire qu'il est possible de modifier les éléments de la liste.
- n-uplets : tuple `tuple` Un 'tuple' est comme la liste une collection d'éléments pouvant être de types variés, mais un tuple est un objet non mutable. Il ne peut pas être modifié une fois défini.
  ```
  >>> t = (4, 'a')
  >>> t[0] = 5
  TypeError: 'tuple' object does not support item assignment
  ```
- séquences : ce sont les chaînes de caractères, les listes ou les tuples, des collections d'éléments ordonnés.

Propriétés d'une séquence

Exemple 7. *Que renvoie la console aux commandes suivantes ?*

```
nom = 'Robert'
for lettre in nom:
    print(lettre)
for element in liste_exemple:
    print(element, 'de type', type(element))
3 in liste_exemple, 3 in liste_exemple[3]
```

II Notion de classe et d'objet

2.1 Une notion sous jacente

```
>>> n = 8
>>> type(n)
<class 'int'>
```

n est une *instance* de la classe définissant les entiers relatifs.

On dit aussi que `int` est un *générateur d'objets.*

Ce processus d'*instanciation d'un objet à partir d'une classe* est une opération fondamentale dans les techniques actuelles de programmation ; la *programmation orientée objet* ou OOP, 'Object Oriented Programming'.

La classe est une sorte de moule à partir duquel on demande à la machine de construire un objet informatique particulier.

Elle peut disposer de fonctions intégrées appelées méthodes, d'attributs qui permettent de caractériser l'objet, et même éventuellement d'une redéfinition des opérateurs.

```
>>> phrase = 'Hello world'
>>> texte = (phrase + ' !\n') * 5
```

+ et * ici sont des opérateurs aux propriétés adaptées aux objets de la classe 'string'.

```
>>> mots = phrase.split()
['Hello', 'world']
```

`split()` est une méthode, c'est à dire une fonction définie dans la classe 'string'.

2.2 Création d'une classe

L'objectif ici est de montrer comment définir ses propres classes d'objets. L'application proposée permet de faire un peu de géométrie vectorielle.

Définition d'une classe point La première application proposée est de définir une classe point en coordonnées cartésiennes dans l'espace.

```
class Point:
    '''Définition d'un point en coordonnées cartésiennes 3D.'''
```

Maintenant que cette nouvelle classe est définie, on peut instancier un premier objet de ce type.

```
p_1 = Point()
```

On peut alors effectuer quelques opérations élémentaires sur ce nouvel objet.
- `print(p_1.__doc__)` renvoie la documentation de la classe.
- `type(p_1)` renvoie le type de la variable `p_1` : `<class '__main__.Point'>`.
- `print(p_1)` renvoie l'adresse mémoire de l'objet point.

Attributs et méthodes d'un objet Pour l'instant, notre nouvelle classe est une boîte vide. On va l'équiper d'*attributs* ou de *variables d'instance*. Pour cela, il est fait appel à une fonction particulière appelée un *constructeur*. Une fonction définie, ou plutôt, encapsulée dans une classe est appelée une *méthode*. Une méthode constructeur est exécutée automatiquement lorsque l'on instancie un nouvel objet à partir de la classe. On peut y placer tout ce qui semble nécessaire pour initialiser automatiquement l'objet que l'on crée. Sous *Python*, la méthode constructeur doit obligatoirement s'appeler `__init__` encadré par deux caractères "souligné".

```
class Point :
    '''Définition d'un point en coordonnées cartésiennes 3D.'''
    def __init__(self, px = 0, py = 0, pz = 0):
        '''Constructeur du Point.'''
        self.abscisse = px
        self.ordonnee = py
        self.cote = pz
```

À partir de maintenant, l'instanciation d'un nouveau point lui donnera par défaut une abscisse, une ordonnée et une cote nulles.
- `O = Point() ; print(O.abscisse)` renvoie `0`.
- `A = Point(-1, 0, 2); print(A) ; print(A.cote)` renvoie deux lignes ; l'adresse mémoire de A `<__main__.Point object at 0x000001795E9080F0>`, et 2.

Enfin, on va pouvoir adjoindre à cette classe d'autres fonctions ou méthodes qui vont permettre d'agir sur les objets de cette classe. Ajouter cette méthode dans la définition de la classe `Point` à la suite de la méthode constructeur.

```
def affichage(self) :
    '''Affiche le Point sous forme d'une chaîne de caractères.'''
    print ('(', str(self.abscisse), ',', str(self.ordonnee), ',', \
           str(self.cote), ')')
```

On peut alors utiliser cette méthode de la manière suivante.
- `O = Point() ; O.affichage()` renvoie `( 0 , 0 , 0 )`.
- `A = Point(-1, 0, 2); A.affichage()` renvoie `( -1 , 0 , 2 )`.

Exemple 8. *De la même façon, ajouter les méthodes suivantes à la classe Point :*
- `format_tuple(self)` *qui retourne le point sous la forme d'un tuple.*
- `format_list(self)` *qui retourne le point sous la forme d'une liste.*

Exemple 9. *Définir maintenant une nouvelle classe* `Vecteur` *avec les méthodes suivantes :*
- *une méthode constructeur* `__init__` *qui définit le vecteur nul par défaut.*
- `affichage(self)` *qui affiche le vecteur sous forme d'une chaîne de caractères.*
- `format_tuple(self)` *qui retourne le vecteur sous la forme d'un tuple.*
- `format_list(self)` *qui retourne le vecteur sous la forme d'une liste.*

Ajouter enfin la méthode suivante à la classe Point :
- `bipoint(self, p)` *qui retourne le vecteur (Point, p).*

Redéfinition des opérateurs dans une classe La programmation orientée objet permet de redéfinir les opérateurs pour chaque nouvelle classe. Par exemple, il est possible de redéfinir l'addition pour nos objets de la classe `Vecteur`.

```
def __add__(self, other):
    '''Retourne un Vecteur qui est l'addition du Vecteur initial
                              et du vecteur passé en argument.'''
    return(Vecteur(self.abscisse + other.abscisse,\
                   self.ordonnee + other.ordonnee ,\
                   self.cote + other.cote))
```

Pour connaître le nom des opérateurs (ici `__add__`), il suffit de taper sur la console `help(int)` par exemple.

Exemple 10. *De la même manière, redéfinir l'opérateur* `*` *pour qu'il retourne un Vecteur qui est le Vecteur initial multiplié par un réel k. Remarquer le défaut de commutativité de l'opérateur défini.*

III Compléments sur les chaînes de caractères

Une chaîne de caractères est une séquence qui est une classe encapsulant les listes et les tuples qui correspond à une collection ordonnée d'éléments.

Comparaison de deux chaînes

Comparaison de deux chaînes

Exemple 11. *Que renvoie* `>>> 'Robert' < 'Jean', 'Robert' < 'jean'` *?*

Les comparaisons entre chaînes de caractères sont possibles parce que les caractères alphanumériques sont codés sous forme binaire dont la valeur est liée à la place qu'occupent ces caractères dans la table de codage (ASCII, unicode, etc.).

Pour accéder à ces codes, il suffit d'utiliser les fonctions `ord` et `chr`.

```
>>> ord('R'), ord('r'), ord('J'), ord('j')
(82, 114, 74, 106)
>>> chr(106)
'j'
```

Méthodes associées aux objets chaînes Voici une liste non exhaustive de méthodes utiles.

Méthode	**Commentaire**
`str.split(`*`délimiteur`*`)`	Renvoie une liste contenant les chaînes de caractères comprises entre chaque occurence du délimiteur.
`str.endswith(`*`suffix`*`)`	Renvoie `True` si la chaîne `str` se termine par la chaîne *suffix*.
`str.startswith(`*`prefix`*`)`	Renvoie `True` si la chaîne `str` commence par la chaîne *prefix*.
`str.isalnum()`	Renvoie `True` si la chaîne `str` est formée uniquement de caractères alphabétiques.
`str.isalpha()`	Renvoie `True` si la chaîne `str` est formée uniquement de caractères alphanumériques.
`str.isdigit()`	Renvoie `True` si la chaîne `str` est formée uniquement de chiffres.
`str.lower()`	Renvoie une chaîne obtenue par passage en minuscules.
`str.upper()`	Renvoie une chaîne obtenue par passage en majuscules.
`str.strip()`	Renvoie une chaîne obtenue en supprimant les blancs au début et à la fin, ainsi que les retours à la ligne (`\n`) et les tabulations (`\t`).

`str.replace(`*`old,new[,n]`*`)`	Renvoie une chaîne obtenue en remplaçant *old* par *new*, au plus *n* fois.
`str.find(`*`sub`*`)`	Indice de la première occurence de *sub* (et -1 si non trouvé).
`str.index(`*`sub`*`)`	Comme *find*, mais `ValueError` si non trouvé.
`str.rfind(`*`sub`*`)`	Indice de la dernière occurence de *sub* (et -1 si non trouvé).
`str.ljust(`*`n[,c]`*`)`	Renvoie `str` justifiée à gauche dans une chaîne de longueur `n`, complétée par des `c`.
`str.rjust(`*`n[,c]`*`)`	Renvoie `str` justifiée à droite dans une chaîne de longueur `n`, complétée par des `c`.

Exemple 12. *Á partir de* `couplet` *et de* `nom` *définis précédemment, indiquer ce que renvoie chacune des lignes suivantes.*

```
couplet.split('\n')
phrase1.split()
couplet.find("'")
couplet.find('beau')
couplet.index('y')
couplet.find('y')
couplet.count("'")
couplet.count("C'est")
nom.lower()
nom.upper()
nom.swapcase()
'jean'.capitalize()
'il␣fait␣beau.␣tout␣va␣bien.'.capitalize()
'␣␣␣Mot␣à␣␣mot␣\n␣␣␣'.strip()
```

IV Compléments sur les listes

Attention aux alias de listes.

```
L = [ [ [0] * 3 ] * 3 ]
```

Cela créé semble-t-il une liste de listes de type matrice 3x3. Mais en fait les L[i] sont toutes des alias de la même sous-liste. Donc lorsque l'on va modifier une L[i], on va modifier les trois.

A la fin, quels que soient les modifications apportées, nous aurons une matrice avec 3 lignes identiques.

Méthodes associées aux objets listes Voici une liste non exhaustive de méthodes utiles.

Méthode	**Commentaire**
`list.sort()`	trie la liste fournie dans l'ordre croissant.
`list.append(`*`élément`*`)`	Ajoute l'élément indiqué à la fin de la liste.
`list.reverse()`	Renverse la liste.
`list.index(`*`élément`*`)`	Renvoie l'index où se situe l'élément indiqué.
`list.remove(`*`élément`*`)`	Supprime la première occurence de l'élément indiqué dans la liste.

Exemple 13. *Que renvoie le petit programme suivant ?*

```
noms = ['Robert', 'Jean', 'Lucie']
noms.sort()
```

```
noms.append('Arthur')
noms.reverse()
noms.index('Jean')
noms.remove('Robert')
del(noms[1])
```

Le 'slicing' ou découpage en tranche

Exemple 14. *Que renvoie ce petit programme ?*

```
lettres = [chr(code) for code in range(ord('a'), ord('a') + 5)]
lettres[3:3] = ['\']
lettres[6:6] = ['f', 'g', 'h']
lettres[1:2] = []
lettres[4:6] = ['i']
lettres[4:] = ['c']
groupe = ''
for element in lettres :
    groupe += element.upper()
print(groupe)
```

Copie de listes

Exemple 15. *Indiquer ce que renvoie la console pour les lignes de commandes suivantes.*

```
liste_1 = list(range(1, 4))
liste_2 = liste_1
liste_1[0] = 'a'
liste_2[2] = 'c'
liste_1, liste_2
```