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

```
nom_variable = valeur
```

1.3 Types de données

— 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.

II Notion de classe et d'objet

```
>>> 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'.

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 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`.

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.