Les variables

Une variable est une donnée de votre programme auquelle est associé une valeur. Elles permettent de stocker des valeurs pour être utilisé plus tard. Les valeurs des variables peuvent changer au cours du temps. On ne peut pas se passer des variables lorsque l'on écrit un programme.

Pour donner une valeur à une variable, il suffit d'écrire :

```
nom_de_la_variable = valeur
```

Par exemple pour stocker mon age il suffit d'écrire :

```
age = 26
```

Cette étape est appelé "affectation de variable", dans cette exemple, on a affecté la valeur `26` à la variable `age`.

Pour votre organisation et pouvoir vous y retrouver dans votre code plus tard, il est conseiller de respecter quelques règles de syntaxe :

- le nom de la variable ne peut etre composé que de lettres, majuscules ou minuscules, de chiffres et du symbole souligné "_"
- le nom de la variable ne peut pas commencer par un chiffre
- Python est sensible à la casse du mot, c'est à dire que la variage `AGE` est différente de la variable `age` qui est différente de la variable `aGe` ...
- le nom de la variable doit être explicite par rapport à son rôle. C'est-à-dire, n'appelez pas vos variables `a`, `b`, `c` et `d`. Lorsque vous aurez besoin de réutiliser votre code, vous ne saurez plus quelle variable correspond à quoi.

Le type de variable

Le type de la variable correspond à la nature de celle-ci. Les principaux utilisés sont les :

- les entiers (int ou integer)
- les décimaux (float)
- les chaines de caractères (string ou str)
- les booléens

ATTENTION Pour que le programme comprenne que vous voulez utiliser une chaine de caractère, il faut bien penser à l'entourer de guillemets (doubles " ou simple ').

Les opérations

Les quatres opérations arithmétiques de base se font de manière simple en utilisant les symboles +-*/. sur les types entiers et décimaux. L'utilisation de parenthèses permet de gérer les priorités des opérations.
Le tableau suivant vous récapitule les opérations de base.

Nom	Désignation	Exemple	Résultat
+	Addition	34+1	35
-	Soustraction	34.0-0.1	33.9
*	Multiplication	300*30	9000
/	Division	1/2	0.5
//	Division entière	1//2	0
**	Exponentiation	4**0.5	2.0
%	Modulo	20%3	2

Les opérations sur les chaines de caractères

On peut utiliser l'addition et la multiplication sur les chaines de caractères. Cependant leur action n'est pas la même que pour les variables numériques.
L'addition concatène les deux chaines de caractères. C'est à dire qu'elle assemble les deux chaines de caractères à la suite l'une de l'autre. Ex:

```
 chaine1 = "salut "
chaine2 = "le monde"
print(chaine1+chaine2)
 >>> salut le monde
```

La multiplication entre une chaine de caractère et un nombre entier duplique la chaine de caractère. Ex:

```
 chaine1 = "salut "
print(chaine1*3)
 >>>salut salut salut
```

Exercice

Créer une variable avec votre age, et afficher la. Puis ajouter 5 à cette variable et afficher la.
Quelle est la nouvelle valeur de votre variable ?
Au lieu d'ajouter l'entier 5, essayer d'ajouter le flottant 5. . Que notez vous sur le résultat ?

In [ ]:

```
```

Les listes

Une liste est une structure de données qui contient une série de valeurs. Ces listes peuvent être composé d'éléments de même type ou de type différents (par exemple chaine de caractère et flottant).
Une liste est déclarée par une série de valeur séparée par des **virgules** et l'intégralité est entouré de **crochets**.
Exemples :

```
 ac = ['dcp1', 'cellmask', '']  quantite = [100, 200, 50]  mixte = ['dcp1', 100,
'cellmask', 200, '', 50]
```

Un des avantages d'une liste, c'est que l'on peut appeler ses éléments par leur positions (= indice de la liste). Cependant, faites attention, le premier élément d'une liste est à l'indice 0.
Exemples :

```
mixte = ['dcp1', 100, 'cellmask', 200, '', 50]
indice =     0     1          2     3       4
```

On a donc notre liste mixte qui contient 5 éléments qui vont de l'indice 0 à 4.

Les listes supportent l'opération + de concaténation et * de duplication. Leurs utilisations est la même que pour les chaines de caractères.

L'instruction `len(ma_liste)` permet de connaitre la taille de la liste.

Il est également possible de créer des listes de listes. Exemple:

```
rack1 = ['Sqh-GFP', 'uas Rho1RNAi', 'uas Rok']
rack2 = ['Diap', 'quas-rpr']
rack3 = ['67Gal4', 'tubGal80ts', 'uasSnail'
stock = [rack1, rack2, rack3]
print (stock)
 >>> [['Sqh-GFP', 'uas Rho1RNAi', 'uas Rok'], ['Diap', 'quas-rpr'], ['67Gal4',
'tubGal80ts', 'uasSnail']]
```

Il est possible d'ajouter un élément à la fin du liste avec la méthode `append()` . Par exemple, pour ajouter la lignée 'GC3I' à notre liste rack1, on va écrire : `rack1.append('GC3I')` .
Vous pouvez également supprimer les éléments d'une liste. Par exemple, je veux supprimer 'uas Rho1RNAi' de ma liste rack1. Là, deux méthodes sont possibles :

- supprimer l'élément à la position i. `rack1.del(1)`
- supprimer à partir de la valeur. `rack1.remove('uas Rho1RNAi')` Attention, dans ce cas, si 'uas Rho1RNAi' apparait plusieurs fois dans la liste. Seule la première occurence sera supprimé.

Exercice

Créer une liste des jours de la semaine auquel il manque le dimanche.
Ajouter le dimanche. Supprimer mercredi.

In [ ]:

```
```

Utilisation des librairies scientifiques - Numpy Scipy Pandas

Numpy

La bibliothèque Numpy permet d'effectuer des calculs numériques. De plus elle permet une gestion facilitée des tableaux de nombres. Cette librairie est souvent utilisées avec la librairie Pandas, qui elle permet la gestion de tableau contenant différents types de données (int, string, float...)

Création d'un tableau

Comme on a vu précédemment pour créer un tableau on utilise `[]` . On peut créer un `numpy.array()` de la même manière.
remarque un objet array ne contient que des données homogènes, c'est à dire du même type.
La fonction array() peut créer des tableaux à n'importe quel nombre de dimensions. Toutefois ça devient vite compliqué lorsqu'on dépasse trois dimensions. Retenez qu'un objet array à une dimension peut être considéré comme un vecteur et un array à deux dimensions comme une matrice. Voici quelques attributs intéressants pour décrire un objet array :

- .ndim renvoie le nombre de dimensions (par exemple, 1 pour un vecteur et 2 pour une matrice)
- .shape renvoie les dimensions sous forme d'un tuple
- .size renvoie le nombre total d'éléments contenus dans l'array. 1

In [2]:

```
#Importation de la librarie numpy
import numpy as np

# Ceci est un tableau en une dimension
tableau1d = np.array([1,2,3])
# Ceci est un tableau en deux dimension
tableau2d = np.array([[1,2,3],[4,5,6]])

print(tableau1d)
print(tableau2d)
```

```
[1 2 3]
[[1 2 3]
 [4 5 6]]
```

Recupérer les éléments d'un array

Pour récupérer un ou plusieurs élément(s) d'un objet array, vous pouvez utiliser les indices ou les tranches, de la même manière qu'avec les listes :

```
tableau1d = np.array([1,2,3,4,5,6,7,8,9,10])
print(tableau1d[1])
>> 2
print(tableau1d[7:10])
>> [8 9 10]
```

Dans le cas d'un objet array à deux dimensions, vous pouvez récupérer une ligne complète (d'indice m), une colonne complète (d'indice n) ou bien un seul élément. 1

```
tableau2d = np.array([[1,2,3],[4,5,6]])
print(tableau2d[:,0])
>>[1 4]
print(tableau2d[0,:])
>>[1 2 3]
print(tableau2d[0,1])
>> 2
```

La syntaxe a[m,:] renvoie la ligne m-1, et a[:,n] renvoie la colonne n-1. Les tranches sont évidemment aussi utilisables sur un tableau à deux dimensions.

Pandas

Le module pandas a été conçu pour la manipulation et l'analyse de données. Il est particulièrement puissant pour manipuler des données structurées sous forme de tableau. Pour charger pandas dans la mémoire de Python, on utilise la commande import habituelle : `import pandas as pd` .

Series

Le premier type de données apporté par pandas est la **series**, qui correspond à un vecteur à une dimension. `s = pd.Series ([10, 20, 30, 40], index = ['a', 'b', 'c', 'd'])` Avec pandas, chaque élément de la série de données possède une étiquette qui permet d'appeler les éléments. Ainsi, pour appeler le premier élément de la série, on peut utiliser son index, comme pour une liste (0 pour le premier élément) ou son étiquette (ici, "a").
Bien sûr, on peut extraire plusieurs éléments, par leurs indices ou leurs étiquettes :

```
s[[1 ,3]]
s[["b","d"]]
```

Les étiquettes permettent de modifier et d'ajouter des éléments :

```
s["c"] = 300
s["z"] = 50
s
```

Enfin, on peut filtrer une partie de la series : `s[s>30]` et même combiner plusieurs critères de sélection : `s[(s>20) & (s <100)]`

Dataframes

Un autre type d'objet particulièrement intéressant introduit par pandas sont les **dataframes**. Ceux-ci correspondent à des tableaux à deux dimensions avec des étiquettes pour nommer les lignes et les colonnes.

Voici comment créer un dataframe avec pandas à partir de données fournies comme liste de lignes :

```
df = pd.DataFrame(columns =["Paris", "Lyon", "Nantes", "Pau"],
    index =[" chat", "singe", "souris"],
    data=[np.arange (10, 14),
    np.arange (20, 24),
    np.arange (30, 34)])
```

Ligne 1. Le dataframe est créé avec la fonction DataFrame() à laquelle on fournit plusieurs arguments. L'argument columns indique le nom des colonnes, sous forme d'une liste.
Ligne 2. L'argument index définit le nom des lignes, sous forme de liste.
Lignes 3-5. L'argument data fournit le contenu du dataframe, sous la forme d'une liste de valeurs correspondantes à des lignes. Ainsi np.arange(10, 14) qui est équivalent à [10, 11, 12, 13] correspond à la première ligne du dataframe.

Quelques propriétés

Les dimensions d'un dataframe sont données par l'attribut **.shape** : df.shape Ici, le dataframe df a 3 lignes et 4 colonnes.

L'attribut **.columns** renvoie le nom des colonnes et permet aussi de renommer les colonnes d'un dataframe : df.columns

La méthode **.head(n)** renvoie les n premières lignes du dataframe (par défaut, n vaut 5) :

Sélection

Les mécanismes de sélection fournis avec pandas sont très puissants.

Sélection de colonnes

On peut sélectionner une colonne par son étiquette : `df["Lyon"]` ou plusieurs colonnes en même temps : `df[[" Lyon", "Pau"]]`
Pour la sélection de plusieurs colonnes, les étiquettes d'intérêt sont rassemblées dans une liste.

Sélection de lignes

Pour sélectionner une ligne, il faut utiliser l'instruction .loc() et l'étiquette de la ligne : `df.loc["singe "]` . Ici aussi, on peut sélectionner plusieurs lignes : `df.loc[[" singe", "chat "]]` . Enfin, on peut aussi sélectionner des lignes avec l'instruction .iloc et l'indice de la ligne (la première ligne ayant l'indice 0): `df.iloc[1]` .
On peut également utiliser les tranches (comme pour les listes) : `df.iloc[0:2]` .

Sélection sur les lignes et les colonnes

On peut bien sûr combiner les deux types de sélection (en ligne et en colonne) :

```
df.loc[" souris", "Pau"] 2 33
df.loc[[" singe", "souris"], ['Nantes ', 'Lyon ']]
```

Notez qu'à partir du moment où on souhaite effectuer une sélection sur des lignes, il faut utiliser loc (ou iloc si on utilise les indices).

Sélection par condition

Sélectionnons maintenant toutes les lignes pour lesquelles les effectifs à Pau sont supérieurs à 15 : `df[ df["Pau"]>15 ]` De cette sélection, on ne souhaite garder que les valeurs pour Lyon : `df[ df["Pau"]>15 ] [" Lyon"]`

On peut aussi combiner plusieurs conditions avec & pour l'opérateur et : `df[ (df["Pau "] >15) & (df["Lyon "]>25) ]` et | pour l'opérateur ou : `df[ (df["Pau "] >15) | (df["Lyon "]>25) ]`