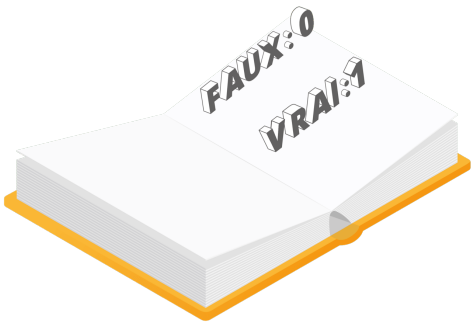
Données structurées en Python

Exemple de données structurées

Il y a bien des siècles, Aristote (384 av. JC - 322 av. JC) constata que la phrase suivante était fausse : "2+2=5 et en plus je suis une fille". Qu'elle soit prononcée par une fille ou non.
Vers 1850, George Boole (1815-1864) rapprocha cette phrase du calcul 0×a=0 sans avoir à tenir compte de la valeur de a. Boole a alors proposé de représenter le *faux* par le nombre 0 et le *vrai* par le nombre 1. Cela lui a notamment permis de ramener les "et" et les "ou" à des multiplications et des additions.
En cela, on peut dire que Boole a proposé une traduction vers des nombres qui permet de fabriquer un dictionnaire donnant la traduction des mots "Vrai" et "Faux" sous forme de nombres. Voici ce dictionnaire, où les mots sont dans l'ordre alphabétique et les définitions (0 et 1) données juste après les mots à définir, précédées de double-points comme dans le Larousse et le Robert.

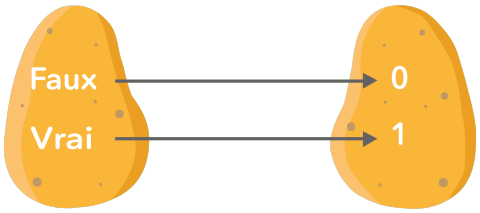


Cela veut tout simplement dire que pour Boole, "Faux" veut dire "0" et "Vrai" veut dire "1". En Python, on écrit ces définitions séparées par une virgule, entre accolades. La variable donnant ceci s'appelle `loi` parce que le titre du livre de Boole est *Les lois de la pensée*. Le type de la variable `loi` est `dict`, abréviation de "dictionnaire". Et pour consulter le dictionnaire, par exemple pour savoir comment Boole traduit "Vrai", on entre `loi["Vrai"]` dans la console :

```
deg                PYTHON
>>> loi={"Faux":0,"Vrai":1}
>>> loi
{'Vrai': 1, 'Faux': 0}
>>> loi["Faux"]
0
>>> loi["Vrai"]
1
>>> type(loi)
<class 'dict'>
>>>
```

Relations

Si on entre `loi.items()` dans la console de Python, on voit des couples du type (clé,valeur). Un dictionnaire Python est un ensemble d'objets comme ('Faux',0) qu'on appelle couple en mathématiques.
On dit qu'on a accouplé le nom "Faux" avec le nombre 0. La notion est dûe à René Descartes qui assimilait les points d'un plan à des couples (x,y) de coordonnées de ces points.
Vers le milieu du XIXe siècle (à l'époque de Boole), le mathématicien anglais Arthur Cayley (1821-1895) a proposé de dessiner un couple par une flèche. Par exemple pour le couple (a,b), Cayley dessine a→b.
L'ensemble de ces dessins de flèches entre divers éléments s'appelle un graphe. Cette notion, dûe à Cayley, est omniprésente en informatique.
Si on veut dessiner le graphe d'un dictionnaire, on constate qu'il est possible de regrouper les clés dans un ensemble dit de départ, les valeurs dans un autre ensemble dit d'arrivée. Comme l'ensemble de départ contient deux éléments `Faux` et `Vrai`, on représente l'ensemble par une sorte de sac (ou de patate) contenant deux éléments, l'un représentant `Vrai`, l'autre représentant `Faux`.



Idem pour l'ensemble d'arrivée, dont les deux éléments représentent respectivement 0 et 1. On a deux flèches, allant de Vrai vers 1 pour représenter le couple (Vrai,1) et l'autre allant de Faux à 0, représentant le couple (Faux,0).
Un ensemble de couples s'appelle une relation. Cette définition est de Gottlob Frege (1848-1925) et son application aux bases de données date de 1970. Elle est dûe à Edgar Codd (1923-2003).
Dans une base de données, une relation n'est pas représentée par des flèches mais par une table comme celle-ci :

Clés	Valeurs
Faux	0
Vrai	1

Le dictionnaire de Boole est quand même une relation bien particulière : chaque clé n'est associée qu'à une valeur. Une telle relation s'appelle application ou fonction.

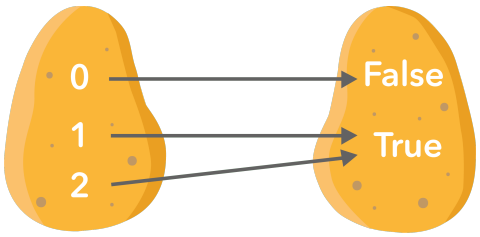
Boole et les fonctions

Python possède une fonction donnant la valeur entière correspondant à un booléen. Cette fonction s'appelle `int()` (comme "integer" en anglais qui veut dire "entier" en français). Mais Boole était anglais, et n'utilisait pas les mots français Vrai et Faux.

1. Taper les instructions suivantes dans la console et noter le résultat :
   - `int(False)`
   - `int(True)`
   - `int(2+2==4)`
   - `int(2+2==5)`

```
>>> int(False)
0
>>> int(True)
1
>>> int(2+2==4)
1
>>> int(2+2==5)
0
```

Python a une autre fonction qui fait le contraire de **`int`** (l'annuaire inversé de tout-à-l'heure) et elle s'appelle **`bool`**, en hommage à George Boole.



La fonction **`bool`** associe la valeur False au nombre 0 et la valeur True au nombre 1 (ou à tout autre nombre différent de 0, par exemple 2).

2. Taper les instructions suivantes dans la console et noter le résultat :
   - `bool(0)`
   - `bool(1)`
   - `bool(0*0)`
   - `bool(0*1)`
   - `bool(1*1)`
   - `bool(0+0)`
   - `bool(0+1)`
   - `bool(1+1)`

```
>>> bool(0)
False
>>> bool(1)
```

```
True
>>> bool(0*0)
False
>>> bool(0*1)
False
>>> bool(1*1)
True
>>> bool(0+0)
False
>>> bool(0+1)
True
>>> bool(1+1)
True
```

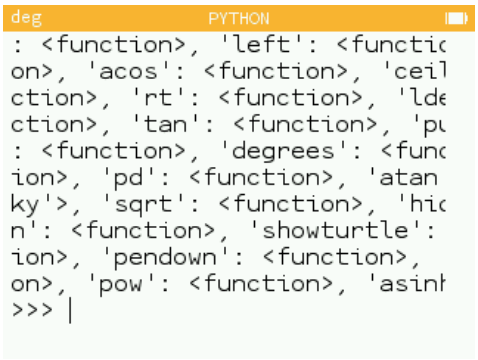
3. Comparer avec le résultat des instructions suivantes :
   - `False and False`
   - `False and True`
   - `True and True`
   - `False or False`
   - `False or True`
   - `True or True`

```
>>> False and False
False
>>> False and True
False
>>> True and True
True
>>> False or False
False
>>> False or True
True
>>> True or True
True
```

La ressemblance entre les deux résultats est à l'origine de l'expression **algèbre de Boole** pour désigner le calcul logique. À l'inverse, les ordinateurs calculent par combinaison d'opérations logiques, en binaire (les opérandes valent tous 0 ou 1).

Un dictionnaire variable : le dictionnaire des variables

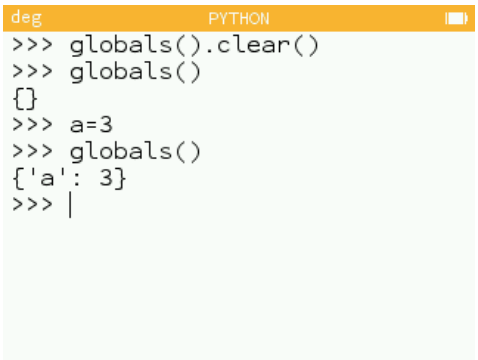
Outre le fait qu'avoir seulement deux entrées dans un dictionnaire soit relativement faible pour des Big Data, le dictionnaire de Boole omet un aspect important des Big Data : le fait qu'elles soient évolutives. Par exemple, lorsqu'un individu demande à placer son numéro de téléphone sur liste rouge, on le retire de l'annuaire.
Chaque exécution d'une instruction a pour effet de modifier au moins une variable, donc l'état de la machine (ordinateur). Ce modèle selon lequel un ordinateur est un dictionnaire donnant à chaque nom de variable une valeur provient de Christopher Strachey (1916-1975) et Dana Scott (1932-).
Il est possible de trouver ce dictionnaire en Python à l'aide de la fonction `globals()` qui donne la liste de toutes les variables (globales) de la session Python courante.



Pour y voir plus clair, on va tout d'abord nettoyer le dictionnaire en le réinitialisant.

```
>>> globals().clear()
>>> globals()
{}
```

Ceci étant fait, l'état de la machine devrait donc être vide, c'est-à-dire qu'il n'y a plus aucune variable globale.
Après l'affectation de la valeur 3 à la variable a (obtenue par `a=3` en Python) le nouvel état de la machine est :



Si on affecte ensuite la valeur 5 à la variable **b**, le dictionnaire `globals()` s'enrichit d'une nouvelle variable.

```
deg                PYTHON
>>> globals().clear()
>>> globals()
{}
>>> a=3
>>> globals()
{'a': 3}
>>> b=5
>>> globals()
{'a': 3, 'b': 5}
>>>
```

Ce nouvel état est représenté par le dictionnaire `'a': 3, 'b': 5` que renvoie `globals()`.

Ce dictionnaire sert à calculer des expressions algébriques :

- Si on écrit `3*8` dans la console, on demande implicitement à Python de calculer 3×8 et d'afficher le résultat : la console est une sorte de calculatrice.
- Mais si on écrit `a*8`, Python ne peut pas effectuer la multiplication avant de savoir quelle est la valeur de a (par quel nombre il doit remplacer la lettre "a"). Pour ce faire, Python va consulter le dictionnaire et trouver que "a" signifie 3, ce qui lui permet d'évaluer `a*8` en 3×8 puis en 24.

Si on veut effectuer quelque chose de répétitif, on utilise une variable **n** servant de compteur (puisqu'on va compter les étapes avec n). Pour cela on fera usage de l'expression n+1 qui donne l'entier suivant n (2 pour n=1, 3 pour n=2, etc).

4. Compléter l'affichage de la session ci-dessous (en console) :

```
>>> n=1
>>> globals()

>>> n+1
2
>>> globals()

>>> n<5

```

```
>>> n=1
>>> globals()
{'n': 1}
>>> n+1
2
>>> globals()
{'n': 1}
>>> n<5
True
```

5. Dans le script précédent, quelle est la valeur finale de n (affichée par `globals()` par exemple) ?

   La valeur finale de n est 1.

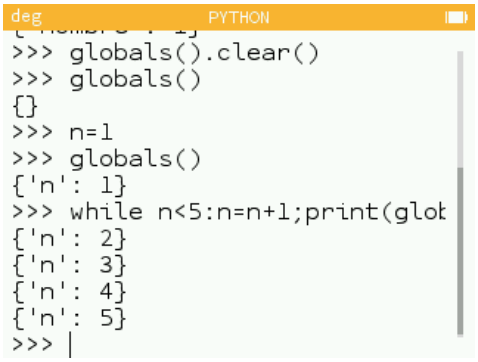
6. Comme on le voit, lorsque Python veut savoir si n est inférieur à 5, il va là encore lire dans le dictionnaire la valeur n, puis comparer ce nombre avec 5.
   Pour que n varie vraiment, il ne faut pas se contenter de calculer n+1, mais il faut ensuite injecter le résultat dans n. Ce qui se fait en écrivant `n=n+1`.
   Compléter les affichages des états successifs de la machine :

```
>>> n = n+1
>>> globals()

>>> n = n+1
>>> globals()

>>> n = n+1
>>> globals()

>>> n = n+1
>>> globals()

>>> n<5

```

```
>>> n = n+1
>>> globals()
{'n': 2}
>>> n = n+1
>>> globals()
{'n': 3}
>>> n = n+1
```

```
>>> globals()
{'n': 4}
>>> n = n+1
>>> globals()
{'n': 5}
>>> n<5
False
```

Pour automatiser le comptage 1, 2, 3, 4, on peut demander à Python de modifier n par **n=n+1** tant que n est inférieur à 5. Cela fait bien 4 itérations :



En résumé, pour compter jusqu'à 4 avec Python, il faut :

- Initialiser la variable n à 1
- Passer de n à l'entier suivant n+1 tant que n est inférieur à 5

7. En langage Python, il est d'usage de commencer à compter à partir de 0, et non de 1. On procède de cette manière :

```
>>>> globals().clear()
>>> globals()
{}
>>> n = 0
>>> while n<4: n=n+1; print(globals())
```

Compléter les étapes suivantes qui donnent les états successifs de la machine :

```
{"n":...........}
{"n":...........}
{"n":...........}
{"n":...........}
```

```
{"n": 1}
{"n": 2}
{"n": 3}
{"n": 4}
```

8. Python peut faire de manière automatique le passage de n à n+1 ainsi que l'initialisation de n et le test de fin de boucle, avec l'instruction for qui traduit "pour n allant de 0 à 4 (non compris)" :

```
>>> globals().clear()
>>> globals()
{}
>>> for n in range(0,4):print(globals())
```

Compléter les étapes suivantes qui donnent les états successifs de la machine :

```
{"n":..........}
{"n":..........}
{"n":..........}
{"n":..........}
```

```
{"n": 0}
{"n": 1}
{"n": 2}
{"n": 3}
```