RÉSUMÉ 3 - LES VARIABLES DANS PYTHON

Lien vers l'activité : Les variables dans Python
www.infoforall.fr - Dernière modif. : 17 09 2020

3.1 - VARIABLES

AFFECTATION D'UN TYPE DE CONTENU

- **Affectation** avec le symbole `=` qui veut dire de stocker le contenu fourni à droite dans la variable fournie à gauche.
- Pour connaître le type d'une variable `a` avec `type(a)`
- Une variable peut contenir un contenu de différents types dont :
  - Des entiers : `int`,
  - Des nombres réels (à virgules flottantes) : `float`,
  - Des chaînes de caractères : `string` ou
  - Des booléens (**True**, **False**) : `bool`.

Voici quelques exemples :

Avec **int**

```
>>> a = 5
>>> a
5
>>> type(a)
<class 'int'>
```

Avec **float**

```
>>> a = 10
>>> b = a + 5.0
>>> b
15.0
>>> type(b)
<class 'float'>
```

- Remarque : Python se charge lui-même de définir le type de façon **dynamique** (cas du int ou float)
- Remarque 2 : attention, souvenez-vous de l'exemple de `3*0.1-0.3` qui ne donne pas 0.

Avec **str**

```
>>> c = "bon" + "jour"
>>> c
'bonjour'
>>> type(c)
<class 'str'>
```

- Remarque : on peut concaténer des strings

Avec **bool**

```
>>> d = "alligator"
>>> e = "poule"
>>> f = d > e
>>> f
False
>>> type(f)
<class 'bool'>
```

- Remarque : Python possède une méthode de comparaison des strings comparable à notre méthode de recherche de mots dans un dictionnaire papier

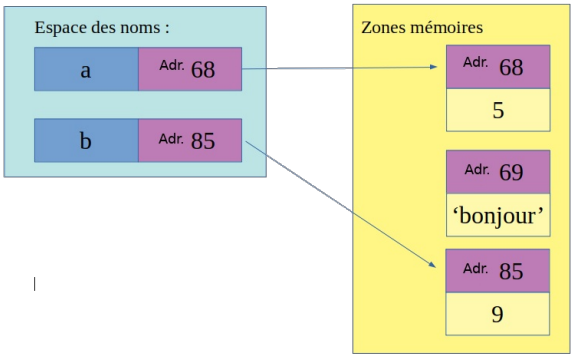
Multiplication de string

```
>>> g = "bon"
>>> g = g * 2
>>> g
'bonbon'
>>> g = "bon"
>>> g = g * 2.0
TypeError # on tente de multiplier un sting avec un 'float'
```

- Remarque : on peut écraser le contenu d'une variable (voir g en ligne 2)
- Remarque 2 : On peut multiplier un string par un int (pas un float comme 2.0 par exemple, attention)

ESPACE DES NOMS

En informatique, une variable est en réalité un **alias** vers une adresse-mémoire.



Lorsqu'on utilise une variable dans un programme, l'interpréteur va donc aller lire ce qui est contenu à l'adresse-mémoire correspondante.
Python ne donne pas directement accès à l'adresse-mémoire réelle mais on peut obtenir un identifiant mémoire à l'aide de la fonction native **id**. Par exemple, on accède à l'identifiant Python d'une variable `a` avec `id(a)`.
Si deux variables ont le même identifiant dans Python, c'est qu'elles font bien référence au même contenu-mémoire.

3.2 - CHOIX DES NOMS DE VARIABLES

Le nom d'une variable :

- doit **commencer** par une **minuscule**
- doit être **explicite** et si possible pas composé d'une seule lettre (et jamais seulement I majuscule ou L minuscule)!
- ne doit pas être l'un des **mots-clés** de Python
- doit utiliser :
  - soit la séparation par **underscores** (_) (`date_interro`)
  - soit la séparation par **majuscules** entre chaque mot (`dateInterro`)

3.3 - LES MOTS-CLÉS POUR INFORMATION

```
False      class      finally    is         return
None       continue   for        lambda     try
True       def        from       nonlocal   while
and        del        global     not        with
as         elif       if         or         yield
assert     else       import     pass
break      except     in         raise
```

3.4 - MODIFIER LE TYPE D'UNE VARIABLE

- Pour tenter de transformer `a` en string, on utilise `str(a)`
- Pour tenter de transformer `a` en integer, on utilise `int(a)`
- Pour tenter de transformer `a` en float, on utilise `float(a)`

3.5 - REMARQUE FINALE TRÈS IMPORTANTE

Si vous voulez modifier le contenu d'une variable contenant l'un des types vus jusqu'ici, il faut impérativement **effectuer une nouvelle affectation qui va écraser l'ancienne variable**.
Exemple **correct** pour augmenter une variable **a** de 1 :

```
1  a = 10
2  a = a + 1
```

Exemple **incorrect** : on demande à Python d'évaluer `a + 1` mais on ne stocke pas le résultat à nouveau dans **a** : **a** n'est donc pas modifiée.

```
1  a = 10
2  a + 1
```