I1 - Définir et calculer avec des variables

En programmation, une variable est un conteneur qui permet de stocker une valeur. Elle permet d'utiliser la mémoire de l'ordinateur pour retenir temporairement le résultat d'un calcul pour pouvoir le réutiliser plus tard.

Création d'une variable : affectation

Pour créer une variable il faut utiliser la syntaxe :

```
nom_variable = valeur
```

On parle alors d'affectation de variable. L'ordre est important : la syntaxe `valeur = nom_variable` est incorrecte.

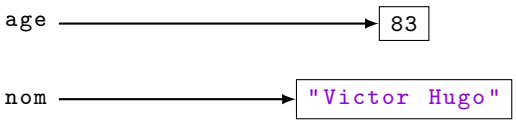
Le nom de la variable doit respecter les conditions suivantes :

- ⋆ il doit commencer par une lettre ;
- ⋆ il doit être constitué que de lettres, de chiffres et du caractère underscore `_` ;
- ⋆ il ne doit pas faire partie des mots-clefs de Python (`for`, `while`, `None`, etc).

Exemple :

```
age = 83
nom = "Victor Hugo"
```

Dans cet exemple, on peut imaginer que deux cases de la mémoire ont été créées : l'une étiquetée `age` contient la valeur `83` et l'autre étiquetée `nom` contient la valeur `"Victor Hugo"`.



En Python, chaque variable possède un type qui correspond au type de valeur qu'elle contient. Il existe plusieurs type dont voici une liste non exhaustive :

- ⋆ `int` : nombres entiers
- ⋆ `float` : nombres à virgule
- ⋆ `complex` : nombres complexes
- ⋆ `str` : chaîne de caractères
- ⋆ `list` : liste
- ⋆ `dict` : dictionnaire

On peut vérifier le type d'une variable en tapant :

```
>>> type(variable)
str
```

Variantes : affectation multiple, affectation simultanée

Pour créer plusieurs variables ayant toutes la même valeur, on peut utiliser la syntaxe :

```
variable_1 = variable_2 = ... = variable_n = valeur_commune
```

Pour créer plusieurs variables **simultanément** avec pour chacune une valeur, on peut utiliser la syntaxe :

```
variable_1, variable_2, ..., variable_n = valeur_1, valeur_2, ..., valeur_n
```

Exemple :

```
x = y = z = 0                # x = 0, y = 0, z = 0
a, b = 3, 5                  # a = 3, b = 5
a, b = a + b, a - b          # a = 8, b = -2 (et non b = 3)
```

On remarque dans la dernière instruction que les valeurs `a + b` et `a - b` sont d'abord évaluées toutes les deux, puis elles sont affectées à `a` et `b` : c'est le sens du mot **simultanées**.

Modification d'une variable

La syntaxe de modification est exactement la même que celle de l'affectation :

```
nom_variable = nouvelle_valeur
```

En Python, une variable peut changer de type ce qui n'est pas le cas de tous les langages de programmation : on dit que Python est un langage faiblement typé.

Fait très important : la nouvelle valeur attribuée à une variable peut être le résultat d'un calcul utilisant l'ancienne valeur ! Dans ce cas, le calcul est effectué avec l'ancienne valeur puis le résultat est affecté à la variable.

Exemple :

```
a = 3
b = 5
a = 3*a + 2*b            # a = 19
b = b - a                # b = -14
```

Echange de valeurs

Un problème récurrent en informatique concerne l'échange de valeurs de deux variables. Supposons qu'on ait déjà défini deux variables `a` et `b`. On souhaite que la variable `b` prenne la valeur de `a` et que la variable `a` prenne la valeur de `b`.

La solution naïve suivante ne fonctionne pas à cause de l'ordre d'exécution des instructions (on parle de flux d'exécution) :

```
a = b
b = a
```

Il y a deux solutions simples à ce problème :

- **Avec une variable auxiliaire :** l'idée étant d'utiliser une troisième variable dont le rôle est de retenir la valeur de `a` pour pouvoir modifier celle-ci avant de modifier `b`.

```
aux = a
a = b
b = aux
```

- **Avec l'affectation simultanée :** elle permet de résoudre le problème en une seule ligne !

```
a, b = b, a
```

Erreurs fréquentes

- **Utiliser la syntaxe d'affectation dans le mauvais sens**

  Par exemple écrire `3 = x` au lieu de `x = 3`.

- **Oublier le symbole * dans un produit**

  Par exemple écrire `3x` au lieu de `3*x` : ici Python pensera que vous essayez d'utiliser une variable qui s'appelle `3x` ce qui est interdit car un nom de variable doit commencer par une lettre.