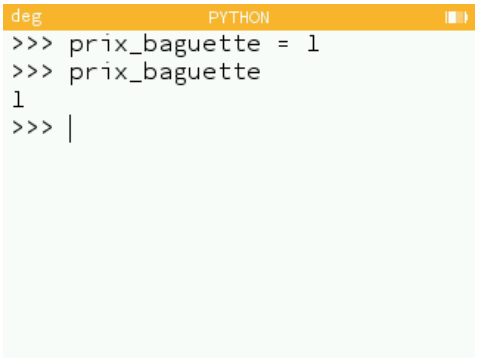
Affectation des variables

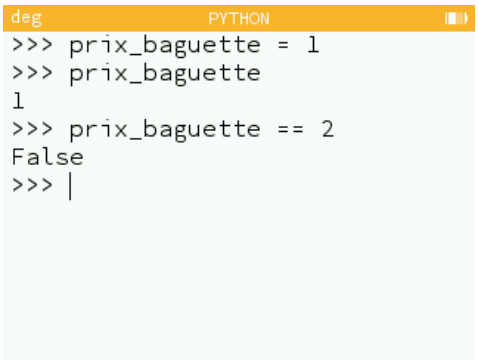
Une variable est un symbole dans lequel il est possible de stocker une valeur. L'affectation de variable est l'action de stocker une valeur dans une variable.
Pour définir le nom d'une variable en Python, vous pouvez utiliser des lettres, minuscules ou majuscules, et des chiffres. Il est aussi possible d'utiliser le caractère underscore : `_`. Cependant, le nom de la variable ne doit pas commencer par un chiffre. Ainsi `variable_1` est accepté par Python. Mais `1ere_variable` ne l'est pas.
L'affectation de variable se fait avec le symbole =. Ainsi, si vous souhaitez stocker la valeur 1 dans `prix_baguette`, vous écrirez l'instruction :

```
prix_baguette = 1
```

Dans la console, vous pouvez demander à Python le contenu d'une variable en écrivant le nom de cette variable et en appuyant sur (EXE), comme sur l'exemple ci-dessous.



Le symbole = étant utilisé pour l'affectation de variable, il n'est plus possible de l'utiliser pour le test d'égalité. Par exemple pour savoir si `prix_baguette` vaut 2, on devra utiliser un double signe ==. Dans la console, on lira le résultat ci-dessous.



On comprendra :
"Est-ce que la variable `prix_baguette` *contient la valeur 2 ?" "Non !"*

Attention : Il ne faut pas confondre variable mathématique et variable informatique.
En mathématiques on peut par exemple avoir une équation : $x - 3 = 5$. L'inconnue de cette équation est la variable x. On lit "x moins 3 égale 5".
En informatique, on peut être amené à écrire des choses comme `x=x+3`.Cette instruction ne se lit plus comme "x égale x plus 3" mais doit être comprise comme : je stocke x plus 3 dans x. Ou bien : j'augmente x de 3.

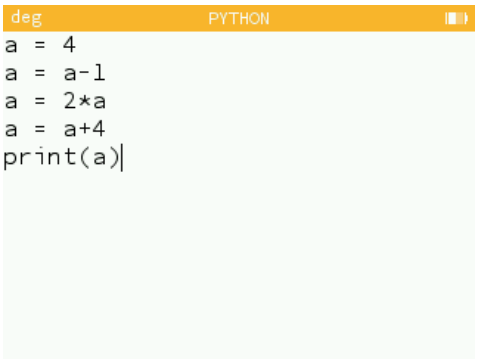
Remarque
Vous pouvez stocker une chaîne de caractères dans une variable. Une chaîne de caractères se délimite avec des guillemets. Par exemple :

```
texte = "Ceci est une chaîne de caractères"
```

Ici nous avons stocké la chaîne de caractères `"Ceci est une chaîne de caractères"` dans la variable `texte`.

Exemple

Je peux écrire la suite d'instructions suivante. Quelle sera la réponse de la console après l'instruction `print(a)` ?



On regarde la première instruction : je diminue **a** de 1. La variable contenait 4, elle contient maintenant 3. La deuxième instruction : je stocke `2*a` dans **a**. La variable **a** contient 3, je stocke donc 6 dans **a**.
La dernière instruction augmente **a** de 4. La variable **a** qui contenait la valeur 6, contient désormais la valeur 10.
La console répond donc 10 !

Exercice

Écrire une suite d'instructions qui échange le contenu de deux variables.
On pourra par exemple imaginer une enveloppe bleue contenant un billet de 20 euros et une enveloppe rouge contenant un billet de 50 euros. Le problème revient à échanger le contenu des deux enveloppes de telle sorte qu'au début on ait :

```
>>> bleue
20
>>> rouge
50
```

Et à la fin :

```
>>> bleue
50
>>> rouge
20
```

Il est important de bien comprendre la question posée. Pour cela, rien de mieux que de se baser sur un exemple.
On va imaginer deux enveloppes contenant une certaine somme d'argent : une enveloppe bleue contient un billet de 20 euros et une enveloppe rouge contient un billet de 50 euros.
On peut tout de suite définir deux variables indiquant la somme contenue dans l'enveloppe bleue et celle contenue dans l'enveloppe rouge.

```
bleue = 20
rouge = 50
```

Nous avons ainsi stocké la valeur 20 dans la variable **bleue** et la valeur 50 dans **rouge**.
Nous souhaitons maintenant échanger le contenu des deux enveloppes. Le billet de 50 euros se retrouve donc dans l'enveloppe bleue et le billet de 20 euros dans l'enveloppe rouge.
L'idée est d'écrire les instructions Python qui effectuent cette opération.

Solution naïve qui ne fonctionne pas

Nous avons donc deux variables **bleue** et **rouge** dont nous voulons échanger le contenu.
Comme on a vu que l'affectation se fait simplement avec le signe =, il est très tentant d'écrire :

```
rouge = bleue
```

```
bleue = rouge
```

```
deg                PYTHON
>>> bleue = 20
>>> rouge = 50
>>> rouge = bleue
>>> bleue = rouge
>>> bleue
20
>>> rouge
20
>>>
```

Cela n'a pas donné le résultat attendu puisque le contenu de **bleue** est resté à 20. Que s'est-il passé ? Lorsque l'on écrit **rouge=bleue**, le contenu de **bleue**, 20, est stocké dans **rouge**. Le nouveau contenu de **rouge** est donc 20. Nous sommes donc dans l'état suivant :

```
>>> bleue
20
>>> rouge
20
```

Lorsque l'on écrit ensuite **bleue = rouge**, le contenu de **rouge**, qui est maintenant 20, est stocké dans **bleue**. Le nouveau contenu de **bleue** est donc 20. Nous sommes donc dans l'état suivant :

```
>>> bleue
20
>>> rouge
20
```

Cette méthode ne fonctionne donc pas puisque la première instruction a écrasé le contenu de la deuxième variable. L'idée est donc de stocker quelque part le contenu de la deuxième variable avant de l'écraser.

Apparition d'une troisième variable

Nous allons donc, avant toute chose, stocker le contenu de la deuxième variable dans une troisième variable, que nous allons nommer **stock**, pour éviter de perdre l'information. Un peu comme si nous avions une troisième enveloppe vide dans laquelle nous gardions au chaud le billet de 50 euros.

```
stock = rouge
```

Maintenant que l'information est saine est sauve, nous pouvons mettre le contenu de la première variable dans la deuxième variable. Dans notre exemple nous mettons le billet de 20 euros dans l'enveloppe rouge

(maintenant vide).

```
rouge = bleue
```

Il ne reste plus qu'à mettre le billet de 50 euros dans l'enveloppe bleue, c'est-à-dire le contenu de `stock` dans la première variable.

```
bleue = stock
```

```
deg PYTHON
>>> bleue = 20
>>> rouge = 50
>>> stock = rouge
>>> rouge = bleue
>>> bleue = stock
>>> bleue
50
>>> rouge
20
>>>
```

Au terme de ces trois instructions, le contenu des deux variables a bien été échangé.

Pour terminer : une petite astuce propre à Python

En fait, le langage Python permet d'éviter de passer par une autre variable. Il est en effet possible de réaliser l'affectation de deux variables lors de la même instruction.

```
a,b = 2,3
```

Cette instruction permet de stocker 2 dans **a** et 3 dans **b**.

```
deg PYTHON
>>> bleue = 20
>>> rouge = 50
>>> bleue,rouge = rouge,bleue
>>> bleue
50
>>> rouge
20
>>>
```

Pour notre cas d'exemple, il était donc possible d'écrire directement la chose suivante. Comme l'affectation est réalisée lors de la même instruction, le contenu de `rouge` n'est pas écrasé et le contenu des deux variables est échangé.

Un autre exercice

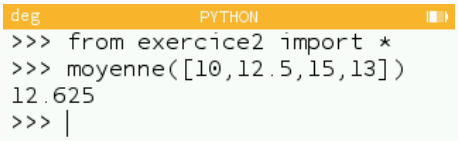
Écrire une fonction `moyenne` qui prend une liste de valeurs en argument et qui renvoie la moyenne arithmétique de ces valeurs.

Cet exercice permet de montrer l'utilisation pratique de variables concrètes à l'intérieur d'une fonction :

```
def moyenne(liste_valeurs):
  effectif=len(liste_valeurs)
  somme=sum(liste_valeurs)
  resultat=somme/effectif
  return resultat
```

Quelques remarques sur le corrigé

Une liste de nombres est une suite de nombres entre crochets séparés par des virgules. Dans la console vous écrirez donc :



La fonction `len` permet de calculer la longueur d'une liste (nombre d'éléments).

```
>>> len([1,4,5,6])
4
```

La fonction `sum` permet de calculer la somme des éléments d'une liste, ici la somme des valeurs entrées.

```
>>> len([1,2,3])
```

```
6
```