

Variables et fonctions

La base de la base

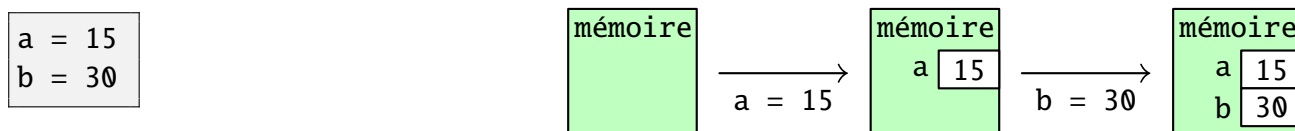
Un **algorithme**, c'est la description d'une méthode pour faire un calcul ou résoudre un problème. On peut par exemple expliquer comment trouver la longueur de l'hypoténuse d'un triangle rectangle à partir des longueurs des deux autres côtés. **Programmer**, c'est traduire un algorithme dans un langage compréhensible par un ordinateur. Cela revient à expliquer à l'ordinateur comment faire le calcul, ou résoudre le problème, dans un langage qu'il peut comprendre.

Il y a de très nombreux langages de programmation, mais ils partagent presque tous un petit nombre d'éléments communs. Au lycée, nous utilisons le langage Python qui n'est ni meilleur, ni plus puissant que les autres, mais qui a l'avantage d'avoir une syntaxe relativement simple. Quand on veut apprendre à programmer, le plus dur c'est d'apprendre un premier langage. Ensuite, apprendre d'autres langages est bien plus simple.

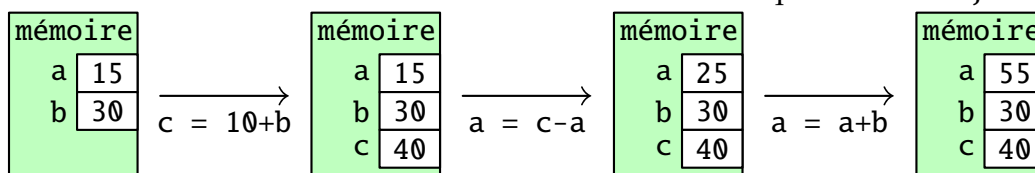
Python est un langage impératif, c'est-à-dire qu'un programme Python est une suite d'instructions à exécuter dans l'ordre. Même s'il existe d'autres styles de programmation (fonctionnels, par contrainte...), on retrouve quasiment toujours les mêmes éléments de base : des affectations, des tests et des boucles. Programmer, c'est juste savoir combiner ces trois éléments.

Affectations et variables

Un ordinateur, ou *computer*, est une machine qui fait des calculs. Les résultats de ces calculs sont stockés dans la mémoire de l'ordinateur. On peut représenter cette mémoire comme des cases contenant des valeurs. Afin de retrouver une valeur parmi les autres, on associe un nom, que l'on appelle **variable**, à chaque case. Cette association est faite au moyen d'une **affectation**. Voici deux exemples d'affectations en Python et une représentation de ce qui se passe en mémoire.



On peut voir que les cases mémoires sont assignées lors de l'affectation. Une fois que la variable `a` a été créée en mémoire, il est possible de l'utiliser dans de nouveaux calculs dont les résultats seront affectés à de nouvelles variables ou à celles qui existent déjà.



Lors de la deuxième affectation, `a = c-a`, on peut voir que le calcul est fait avec la valeur précédente de `a`. Ainsi, avec `a` qui vaut 15 et `c` qui vaut 40, on trouve `c-a` vaut 25. Lors de l'affectation suivante, on fait donc `25+30` et on trouve 55.

Une fois qu'une valeur a été affectée à une variable, les précédentes valeurs ne sont plus disponibles. Il n'y a pas d'historique qui permettrait d'utiliser une ancienne valeur de la variable.

EXERCICE 1 : Déterminer la valeur associée à la variable `a` après la suite d'instructions ci-contre.

```

a = 5
a = 2*a
a = 2*a

```

Arithmétique

Afin d'effectuer les calculs, Python dispose d'un ensemble d'opérateurs arithmétiques présentés ci-contre. Les priorités de calculs sont respectées et il est possible d'utiliser des parenthèses.

Par exemple $\frac{5 \times (7 - 4)}{2}$ s'écrit : `(5 * (7-4)) / 2`

Il faut distinguer la division sur les réels (`x / y`) et la division sur les entiers (`x // y`). On peut également obtenir le reste en faisant `x % y`.

En math	En Python
$x + y$	<code>x + y</code>
$x - y$	<code>x - y</code>
$x \times y$	<code>x * y</code>
$x : y$	<code>x / y</code>
x^2	<code>x**2</code>
x^3	<code>x**3</code>
x^y	<code>x**y</code>

```
>>> 17 / 5      # Division sur les réels
3.4
>>> 17 // 5     # Quotient de la division euclidienne
3
>>> 17 % 5      # Reste de la division euclidienne
2
```

En Python, il est possible d'utiliser des noms de variables plus parlants. Il faut juste commencer par une lettre ou un "_", et ne contenir que des lettres, chiffres et des "_". Ainsi `une_variable`, `UneAutreVariable`, `_une_3eme` et `_4_fois_3` sont tous valides.

EXERCICE 2 : Déterminer les noms de variables qui sont valides en Python, et ceux qui ne le sont pas.

- 1) `2be3` 2) `_3` 3) `le haut` 4) `b0B_`

Tableau d'exécution

Afin de pouvoir déterminer l'état de la mémoire, on utilise un **tableau d'exécution**. L'exemple de la page précédente peut être résumé par le tableau ci-contre. Pour calculer la valeur des variables après une instruction, on utilise les valeurs de la ligne précédente. On peut laisser vide les cases des variables avant leur affectation.

	a	b	c
<code>a = 15</code>	15		
<code>b = 30</code>	15	30	
<code>c = 10+b</code>	15	30	40
<code>a = c-a</code>	25	30	40
<code>a = a+b</code>	55	30	40

EXERCICE 3 : Compléter les tableaux ci-dessous :

	a	b	c
<code>a = 3</code>			
<code>b = 5</code>			
<code>c = a+b</code>			

	x	y
<code>x = 10</code>		
<code>y = 2*x</code>		
<code>x = y+x</code>		

	Bob	Joe	Al
<code>Bob = 15</code>			
<code>Joe = Bob+8</code>			
<code>Al = Joe//2</code>			

	a
<code>a = 2</code>	
<code>a = a**2</code>	
<code>a = a**2</code>	

	v1	v2	res
<code>v1 = 7</code>			
<code>v2 = 12</code>			
<code>res = v1+v2</code>			
<code>res = res-v2</code>			

	a	b
<code>a = 1</code>		
<code>b = 2</code>		
<code>a = b</code>		
<code>b = a</code>		

Fonctions

Lorsqu'on refait plusieurs fois les mêmes calculs ou qu'on veut structurer un peu son programme, on peut définir des **fonctions**. Une fonction est, comme en mathématiques, un outil qui permet de calculer un résultat à partir de **paramètres**. Par exemple, les quatre fonctions suivantes permettent d'obtenir le même résultat.

```
def f1(x):  
    y = 3*x+5  
    return y
```

```
def f2(x):  
    y = 3*x  
    y = y+5  
    return y
```

```
def f3(x):  
    m = 3*x  
    n = m+5  
    return n
```

```
def f4(x):  
    return 3*x+5
```

Pour utiliser une fonction, il suffit de remplacer les paramètres dans les parenthèses, par les valeurs souhaitées, appelées **arguments**. Il suffit d'écrire `f1(4)`, `f2(10)` ou encore `f4(3)`. La commande **return** permet de définir la valeur renvoyée par la fonction. Cette valeur peut alors être affichée ou affectée à une variable. Lors de l'évaluation, le paramètre reçoit la valeur donnée, le code de la fonction est exécuté et la valeur obtenue correspond à celle après le **return**.

```
a = f1(4) → x = 4  
            y = 3*x+5  
            a = y
```

```
a = f2(4) → x = 4  
            y = 3*x  
            y = y+5  
            a = y
```

```
a = f4(4) → x = 4  
            a = 3*x+5
```

Il est possible de faire des fonctions sans paramètres ou avec plusieurs paramètres.

```
def deux():  
    return 2
```

```
def premier(x, y):  
    return x
```

Une fonction peut ne pas avoir de **return** et dans ce cas, on ne peut pas l'utiliser dans une affectation. Au contraire, s'il y a plusieurs **return**, c'est le premier rencontré qui renvoie la valeur, peu importe les instructions qui peuvent suivre dans la fonction. L'exécution de la fonction s'arrête net et la valeur est renvoyée directement.

EXERCICE 4 :

1) Compléter la fonction `somme(x, y)` pour qu'elle renvoie la somme de `x` et de `y`.

```
def somme(x, y):  
    return ...
```

2) Compléter la fonction `deuxieme(x, y)` pour qu'elle renvoie la valeur du deuxième paramètre.

```
def deuxieme(x, y):  
    return ...
```

3) Compléter les fonctions `g2(a)` et `g3(x)` pour qu'elles calculent les mêmes résultats que `g1(x)`.

```
def g1(x):  
    y = x + 3  
    y = x * y  
    return y
```

```
def g2(a):  
    b = 3 + ...  
    c = ...  
    return c
```

```
def g3(x):  
    return ...
```

4) Déterminer le résultat obtenu par `h(3, 5)`

```
def h(x, y):  
    a = x + y  
    b = x * y  
    c = b - a  
    d = a + b  
    return c  
    return d
```