

# Programmation en Python - Cours 1 : Premier contact

MPSI - Lycée Thiers

2014/2015

## Introduction

Le langage python

## Utilisation en mode console

Lancement en mode console

Calculatrice

Le module `math`

Le module `fractions`

Définition d'une fonction

Variables

## Annexes

Annexe : Architecture des ordinateurs

Annexe : Ecriture d'un nombre en base 2

# Le langage python, kesako ?

Python est un langage de programmation impérative, structurée, orientée objet, de haut niveau.

Il présente les avantages suivants :

- Sa syntaxe est très simple et concise : "on code ce que l'on pense". Donc facile à apprendre. Proche du 'langage algorithmique'.
- Moderne. Très largement répandu dans l'industrie, l'enseignement et la recherche, notamment pour ses applications scientifiques. Une large communauté participe à son développement.
- Puissant, muni de nombreuses bibliothèques de fonctions. Dont de très bonnes bibliothèques scientifiques.
- Pratique pour travailler sur des objets mathématiques. Assez proche du langage mathématique.
- Gratuit, disponible sur la plupart des plateformes (Windows, Mac, Linux, ...).

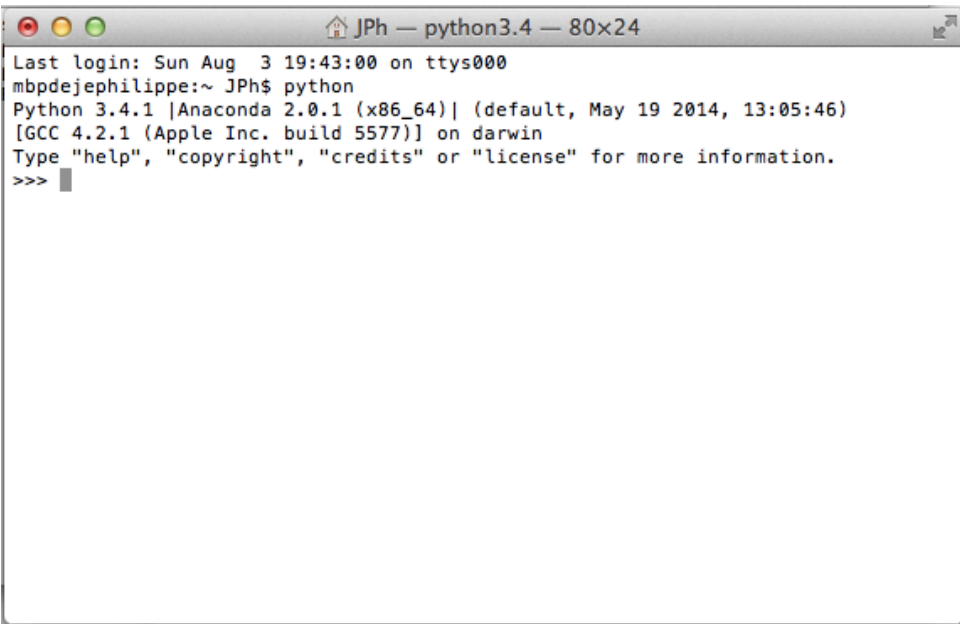
# Python utilisé en mode console

Python est un langage interprété qui peut être utilisé en mode console. Lancer une fenêtre de console (terminal, console ou fenêtre de commande selon le Système d'exploitation) et simplement taper à l'invite 'python' suivi de la touche entrée.

>>' is visible." data-bbox="258 438 730 848"/>

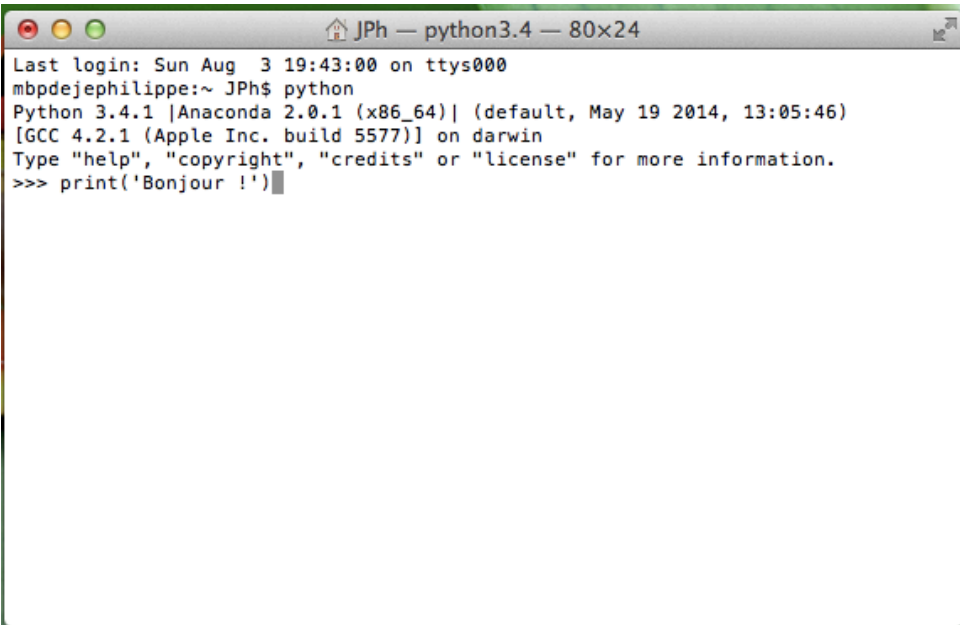
```
JPh — python3.4 — 80x24
Last login: Sun Aug  3 19:43:00 on ttys000
mbpdejeffphilippe:~ JPh$ python
Python 3.4.1 [Anaconda 2.0.1 (x86_64)] (default, May 19 2014, 13:05:46)
[GCC 4.2.1 (Apple Inc. build 5577)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> █
```

Un prompt, symbolisé ici par : >>>, apparaît. On peut y saisir tout type de commandes python.

A screenshot of a macOS terminal window. The title bar at the top shows three colored window control buttons (red, yellow, green) on the left, a home icon followed by the text 'JPh — python3.4 — 80x24' in the center, and a close button on the right. The terminal content shows the following text: 'Last login: Sun Aug 3 19:43:00 on ttys000', 'mbpdejephilippe:~ JPh\$ python', 'Python 3.4.1 |Anaconda 2.0.1 (x86\_64)| (default, May 19 2014, 13:05:46)', '[GCC 4.2.1 (Apple Inc. build 5577)] on darwin', 'Type "help", "copyright", "credits" or "license" for more information.', and '>>>' followed by a cursor. The terminal has a white background and a thin gray border.

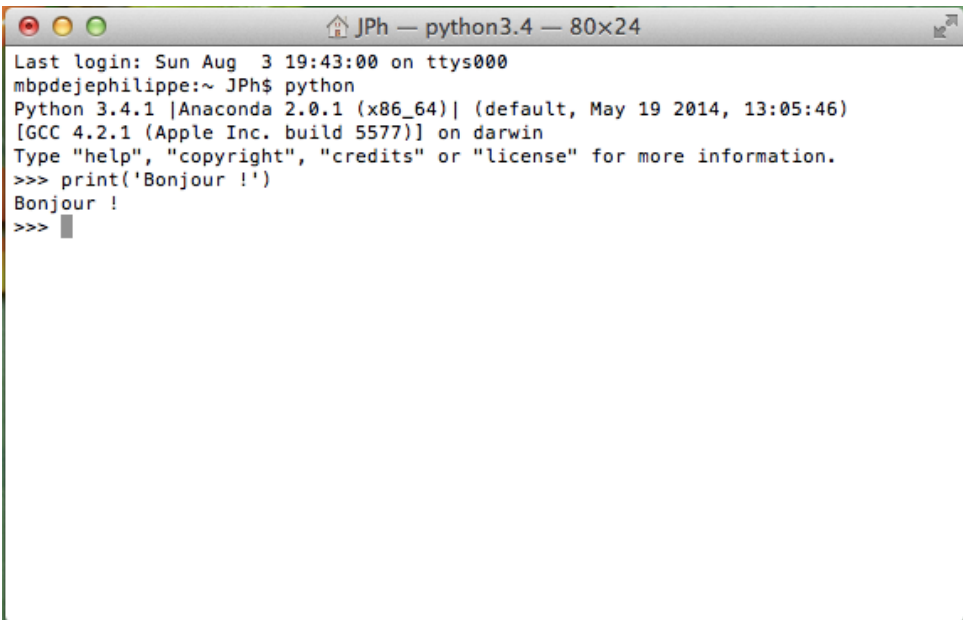
```
Last login: Sun Aug 3 19:43:00 on ttys000
mbpdejephilippe:~ JPh$ python
Python 3.4.1 |Anaconda 2.0.1 (x86_64)| (default, May 19 2014, 13:05:46)
[GCC 4.2.1 (Apple Inc. build 5577)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> █
```

Une console au lancement de python.



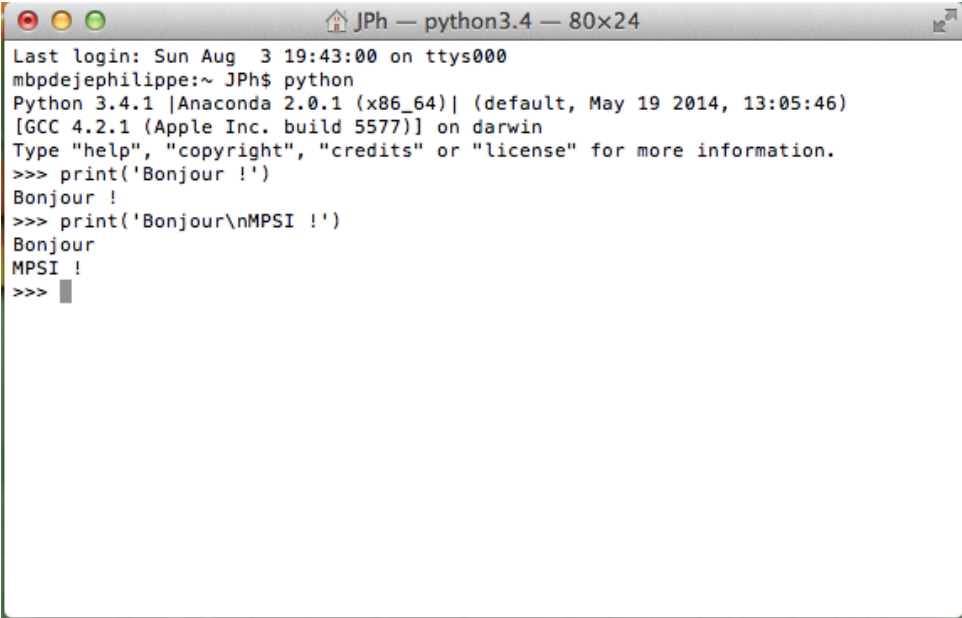
```
Last login: Sun Aug  3 19:43:00 on ttys000
mbpdejephilippe:~ JPh$ python
Python 3.4.1 |Anaconda 2.0.1 (x86_64)| (default, May 19 2014, 13:05:46)
[GCC 4.2.1 (Apple Inc. build 5577)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print('Bonjour !')
```

Une instruction saisie au prompt est lancée en appuyant sur la touche ENTREE.



```
Last login: Sun Aug  3 19:43:00 on ttys000
mbpdejephilippe:~ JPh$ python
Python 3.4.1 |Anaconda 2.0.1 (x86_64)| (default, May 19 2014, 13:05:46)
[GCC 4.2.1 (Apple Inc. build 5577)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print('Bonjour !')
Bonjour !
>>> █
```

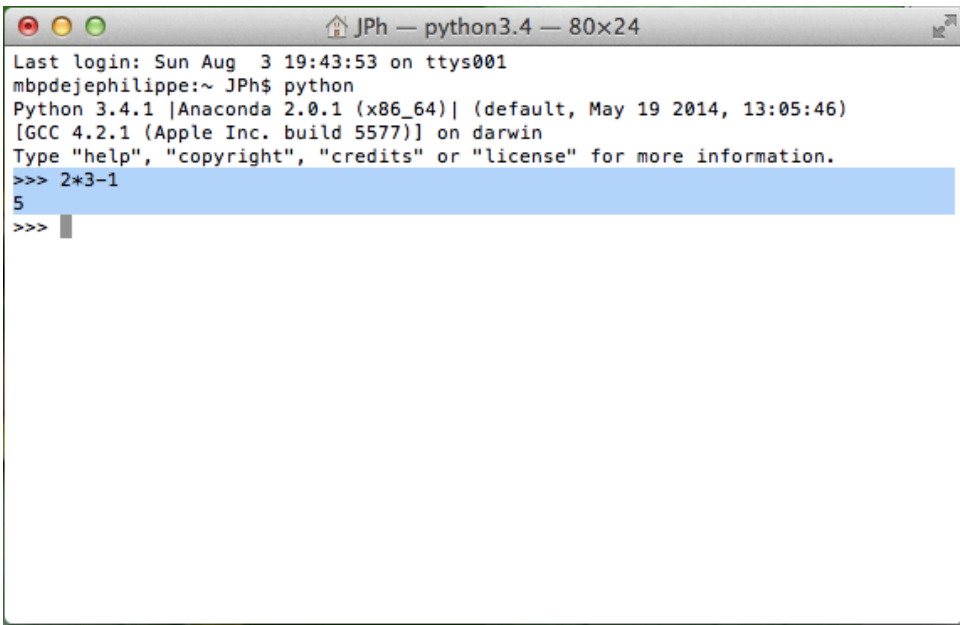
La commande (ou fonction) 'print(.)' écrit à l'écran une chaîne de caractère, c'est à dire une suite finie de caractères entre apostrophes '...' ou double-quotes "...".



```
Last login: Sun Aug  3 19:43:00 on ttys000
mbpdejephilippe:~ JPh$ python
Python 3.4.1 |Anaconda 2.0.1 (x86_64)| (default, May 19 2014, 13:05:46)
[GCC 4.2.1 (Apple Inc. build 5577)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print('Bonjour !')
Bonjour !
>>> print('Bonjour\nMPSI !')
Bonjour
MPSI !
>>> █
```

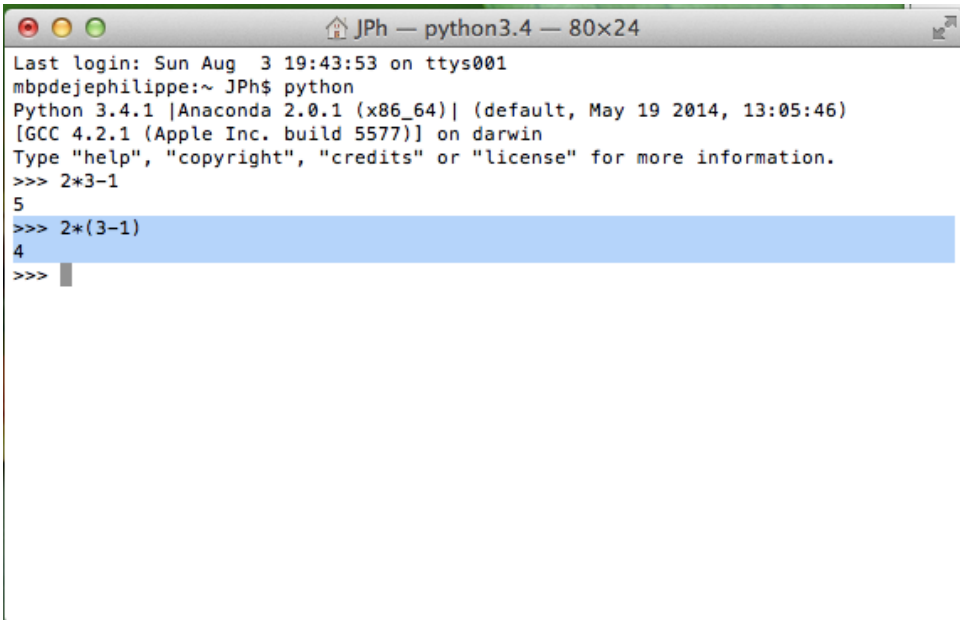
Le caractère spécial `\n` permet un passage à la ligne.





```
Last login: Sun Aug  3 19:43:53 on ttys001
mbpdejephilippe:~ JPh$ python
Python 3.4.1 |Anaconda 2.0.1 (x86_64)| (default, May 19 2014, 13:05:46)
[GCC 4.2.1 (Apple Inc. build 5577)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 2*3-1
5
>>>
```

Python peut se comporter comme une calculatrice.



```
Last login: Sun Aug  3 19:43:53 on ttys001
mbpdejephilippe:~ JPh$ python
Python 3.4.1 |Anaconda 2.0.1 (x86_64)| (default, May 19 2014, 13:05:46)
[GCC 4.2.1 (Apple Inc. build 5577)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 2*3-1
5
>>> 2*(3-1)
4
>>>
```

Il respecte l'ordre usuel des opérations.

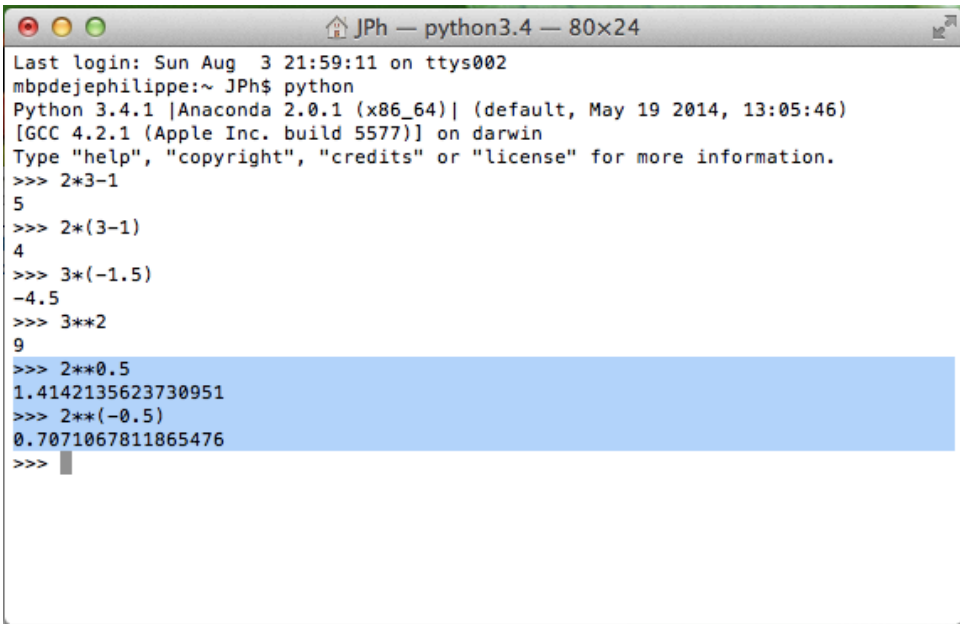
```
JPh — python3.4 — 80x24
Last login: Sun Aug  3 19:43:53 on ttys001
mbpdejephilippe:~ JPh$ python
Python 3.4.1 |Anaconda 2.0.1 (x86_64)| (default, May 19 2014, 13:05:46)
[GCC 4.2.1 (Apple Inc. build 5577)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 2*3-1
5
>>> 2*(3-1)
4
>>> 3*(-1.5)
-4.5
>>>
```

Et comprend les nombres à virgule flottante 'de type float'.

```
JPh — python3.4 — 80x24

Last login: Sun Aug  3 21:41:11 on ttys001
mbpdejephilippe:~ JPh$ python
Python 3.4.1 |Anaconda 2.0.1 (x86_64)| (default, May 19 2014, 13:05:46)
[GCC 4.2.1 (Apple Inc. build 5577)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 2*3-1
5
>>> 2*(3-1)
4
>>> 3*(-1.5)
-4.5
>>> 3**2
9
>>>
```

La mise en puissance s'obtient grâce à l'opérateur \*\*.



```
Last login: Sun Aug 3 21:59:11 on ttys002
mbpdejephilippe:~ JPh$ python
Python 3.4.1 |Anaconda 2.0.1 (x86_64)| (default, May 19 2014, 13:05:46)
[GCC 4.2.1 (Apple Inc. build 5577)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 2*3-1
5
>>> 2*(3-1)
4
>>> 3*(-1.5)
-4.5
>>> 3**2
9
>>> 2**0.5
1.4142135623730951
>>> 2**(-0.5)
0.7071067811865476
>>>
```

L'exposant peut être 'réel'. Ainsi l'opérateur `**0.5` extrait la racine carrée.



```
Last login: Sun Aug  3 21:59:11 on ttys002
mbpdejephilippe:~ JPh$ python
Python 3.4.1 |Anaconda 2.0.1 (x86_64)| (default, May 19 2014, 13:05:46)
[GCC 4.2.1 (Apple Inc. build 5577)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 2*3-1
5
>>> 2*(3-1)
4
>>> 3*(-1.5)
-4.5
>>> 3**2
9
>>> 2**0.5
1.4142135623730951
>>> 2**(-0.5)
0.7071067811865476
>>> (-2)**0.5
(8.659560562354934e-17+1.4142135623730951j)
>>>
```

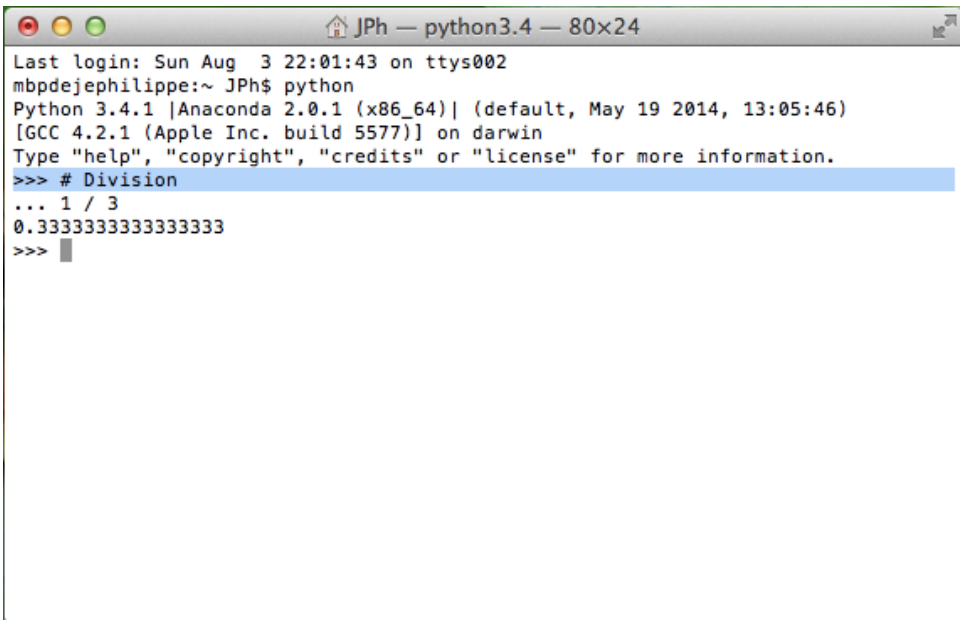
Lorsque  $y$  n'est pas entier, le réel  $x^y$  n'est défini que pour  $x > 0$  : python (version 3) retourne un nombre complexe, valeur approchée ici de  $i\sqrt{2}$ .



```
Last login: Sun Aug  3 21:59:11 on ttys002
mbpdejephilippe:~ JPh$ python
Python 3.4.1 |Anaconda 2.0.1 (x86_64)| (default, May 19 2014, 13:05:46)
[GCC 4.2.1 (Apple Inc. build 5577)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 2*3-1
5
>>> 2*(3-1)
4
>>> 3*(-1.5)
-4.5
>>> 3**2
9
>>> 2**0.5
1.4142135623730951
>>> 2**(-0.5)
0.7071067811865476
>>> (-2)**0.5
(8.659560562354934e-17+1.4142135623730951j)
>>> 1j * 1j
(-1+0j)
>>>
```

$x + i.y$  s'écrit  $x+yj$  (avec  $x,y$  des flottants).

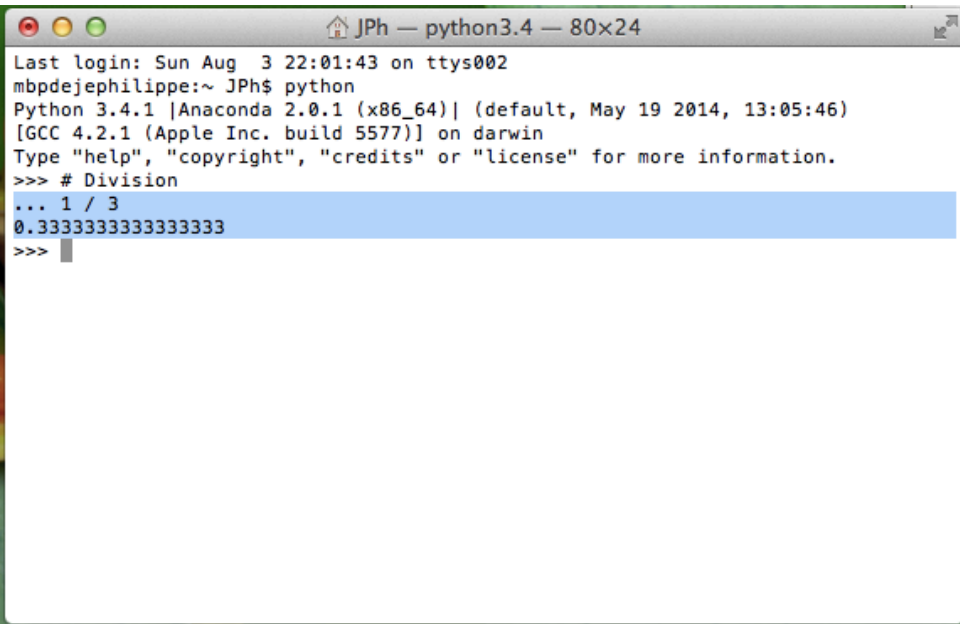
*Remarque* : opérandes et opérations peuvent être séparés d'aucun, un, ou plusieurs espaces.



```
Last login: Sun Aug  3 22:01:43 on ttys002
mbpdejephilippe:~ JPh$ python
Python 3.4.1 |Anaconda 2.0.1 (x86_64)| (default, May 19 2014, 13:05:46)
[GCC 4.2.1 (Apple Inc. build 5577)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> # Division
... 1 / 3
0.3333333333333333
>>>
```

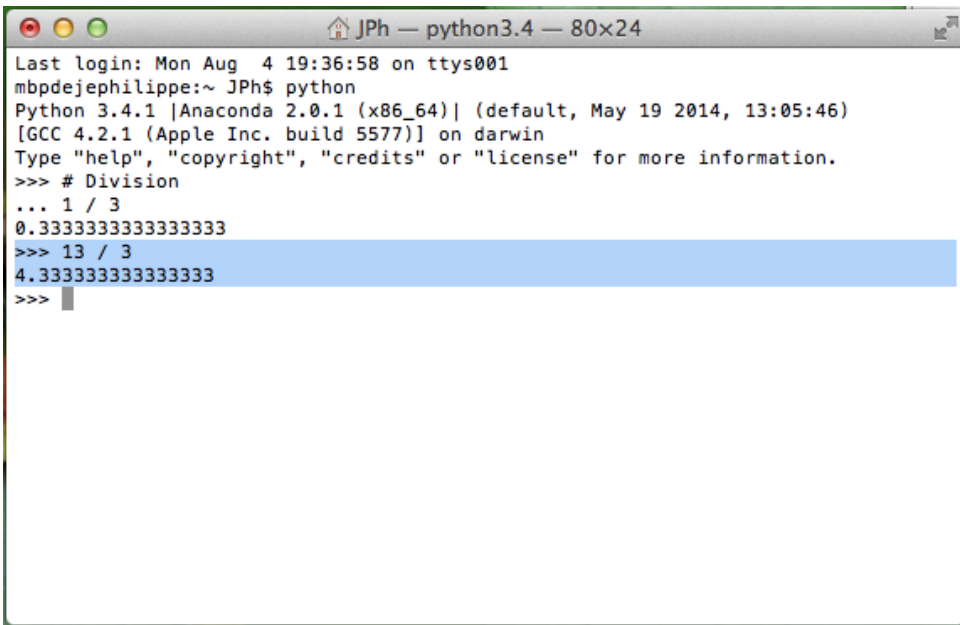
Les commentaires sont placés après un symbole `#`. Tout ce qui est placé après un symbole `#` sur une ligne est ignoré par l'interpréteur. Il est essentiel de les utiliser lors de l'écriture d'un programme.





```
Last login: Sun Aug  3 22:01:43 on ttys002
mbpdejephilippe:~ JPh$ python
Python 3.4.1 |Anaconda 2.0.1 (x86_64)| (default, May 19 2014, 13:05:46)
[GCC 4.2.1 (Apple Inc. build 5577)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> # Division
... 1 / 3
0.3333333333333333
>>>
```

/ est l'opérateur de division.

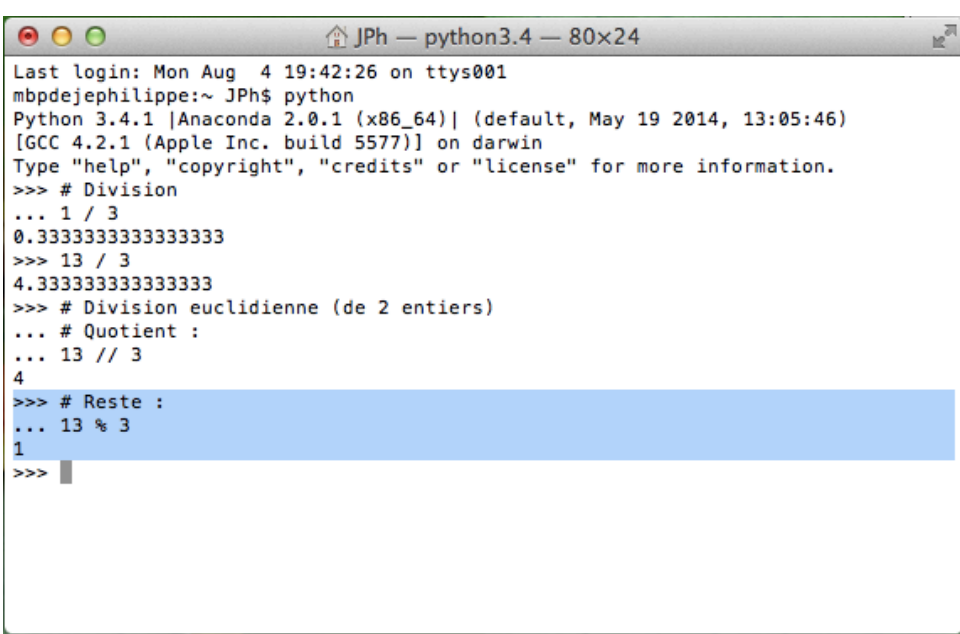


```
Last login: Mon Aug  4 19:36:58 on ttys001
mbpdejephilippe:~ JPh$ python
Python 3.4.1 |Anaconda 2.0.1 (x86_64)| (default, May 19 2014, 13:05:46)
[GCC 4.2.1 (Apple Inc. build 5577)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> # Division
... 1 / 3
0.3333333333333333
>>> 13 / 3
4.333333333333333
>>>
```

/ est l'opérateur de division.

```
Last login: Mon Aug  4 19:42:26 on ttys001
mbpdejephilippe:~ JPh$ python
Python 3.4.1 |Anaconda 2.0.1 (x86_64)| (default, May 19 2014, 13:05:46)
[GCC 4.2.1 (Apple Inc. build 5577)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> # Division
... 1 / 3
0.3333333333333333
>>> 13 / 3
4.333333333333333
>>> # Division euclidienne (de 2 entiers)
... # Quotient :
... 13 // 3
4
>>>
```

// est l'opérateur de quotient de 2 entiers  $n$  et  $m \neq 0$  dans la division euclidienne de  $n$  par  $m$ ; il retourne le quotient entier  $n//m = \lfloor \frac{n}{m} \rfloor$ .



```
Last login: Mon Aug 4 19:42:26 on ttys001
mbpdejephilippe:~ JPh$ python
Python 3.4.1 |Anaconda 2.0.1 (x86_64)| (default, May 19 2014, 13:05:46)
[GCC 4.2.1 (Apple Inc. build 5577)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> # Division
... 1 / 3
0.3333333333333333
>>> 13 / 3
4.333333333333333
>>> # Division euclidienne (de 2 entiers)
... # Quotient :
... 13 // 3
4
>>> # Reste :
... 13 % 3
1
>>>
```

L'opérateur % retourne le reste de la division euclidienne :

$$n = (n // m) \times m + (n \% m).$$



```
Last login: Mon Aug  4 19:42:26 on ttys001
mbpdejephilippe:~ JPh$ python
Python 3.4.1 |Anaconda 2.0.1 (x86_64)| (default, May 19 2014, 13:05:46)
[GCC 4.2.1 (Apple Inc. build 5577)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> # Division
... 1 / 3
0.3333333333333333
>>> 13 / 3
4.333333333333333
>>> # Division euclidienne (de 2 entiers)
... # Quotient :
... 13 // 3
4
>>> # Reste :
... 13 % 3
1
>>> # Vérification :
... 3 * 4 + 1
13
>>>
```

L'opérateur % retourne le reste de la division euclidienne :

$$n = (n//m) \times m + (n\%m).$$

# le module math

Par défaut python ne connaît, en dehors des opérateurs arithmétiques, aucune fonction ou constante mathématiques : sans bibliothèque l'appel de `cos(1)` ou `pi` produit une erreur.

```
>>> # Sans bibliothèque python est ignorant en math :  
... cos(1)  
Traceback (most recent call last):  
  File "<stdin>", line 2, in <module>  
NameError: name 'cos' is not defined  
>>> pi  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
NameError: name 'pi' is not defined
```

Il suffit de faire appel à la bibliothèque '`math`' de fonctions mathématiques prédéfinies. C'est un *module* :

```
>>> from math import *           # importation des fonctions de la bibliothèque  
>>> pi  
3.141592653589793  
>>> cos(pi)  
-1.0  
>>> acos(-1)  
3.141592653589793
```

# le module math

```
>>> sqrt(2)                # racine carrée
1.4142135623730951
>>> e
2.718281828459045
>>> log(e)                 # logarithme neperien
1.0
>>> exp(1)                 # exponentielle
2.718281828459045
>>> log(256,2)             # logarithme en base 2
8.0
>>> log(1000,10)           # logarithme en base 10
2.9999999999999996
```

```
>>> # lui préférer :
... log10(1000)            # logarithme base 10 plus précis
3.0
_
```

```
>>> fabs(-3)               # valeur absolue
3.0
>>> floor(pi)              # partie entière
3.0
>>> floor(-pi)
-4.0
```

# Le module fractions

Le module fractions permet le calcul sur les fractions :

Cet exemple se passe de commentaires :

```
>>> # Le module fractions
... 1/3 + 2/5
0.7333333333333334
>>> from fractions import Fraction
>>> Fraction(1,3) + Fraction(2,5)
Fraction(11, 15)
```

Ici l'instruction 'from fractions import Fraction' n'importe que la fonction Fraction de la bibliothèque fractions. On peut aussi importer toute la bibliothèque à l'aide de : 'from fractions import \*'. ('\*' se lit 'all' = 'tout').



# Définition de fonction

L'utilisateur peut définir ses propres *fonctions* :

```
>>> # Definition d'une FONCTION
... def maFonction(x):
...     return x**2-2*x+1
...
>>> maFonction(1)
0
>>> maFonction(0)
1
```

Ici l'appel de `maFonction(x)` retourne  $x^2 - 2x + 1$  grâce à l'instruction `return`.

Une fonction peut aussi ne retourner aucun résultat, c'est une *procédure* :

```
>>> # Définition d'un procédure (pour python c'est aussi une fonction)
... def maProcédure(x):
...     print("L'image par maFonction de", x, "est", maFonction(x))
...
>>> maProcédure(1)
L'image par maFonction de 1 est 0
>>> maProcédure(0)
L'image par maFonction de 0 est 1
>>> maProcédure(10)
L'image par maFonction de 10 est 81
```

Remarquer qu'une fonction peut être appelée dans la définition d'une autre fonction.

# Définition de fonction

L'utilisateur peut définir ses propres *fonctions* :

```
>>> # Définition d'une FONCTION
... def maFonction(x):
...     return x**2-2*x+1
...
>>> maFonction(1)
0
>>> maFonction(0)
1
```

Ici l'appel de `maFonction(x)` retourne  $x^2 - 2x + 1$  grâce à l'instruction `return`.

Une fonction peut aussi ne retourner aucun résultat, c'est une *procédure* :

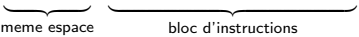
```
>>> # Définition d'une procédure (pour python c'est aussi une fonction)
... def maProcédure(x):
...     print("L'image par maFonction de", x, "est", maFonction(x))
...
>>> maProcédure(1)
L'image par maFonction de 1 est 0
>>> maProcédure(0)
L'image par maFonction de 0 est 1
>>> maProcédure(10)
L'image par maFonction de 10 est 81
```

Remarquer que la fonction prédéfinie `print()` peut prendre plusieurs arguments séparés par des virgules, chaîne de caractère, variable numérique, etc...

# Définition de fonction

Syntaxe pour la définition d'une fonction :

```
def NomdeLaFonction(Paramètres):  
    ..... Instruction1  
    ..... Instruction2  
    .....  
    ..... Dernière instruction
```



L'instruction 'def' permet de définir une fonction. Elle est suivie du nom de la fonction obligatoirement suivi de parenthèses pouvant contenir des noms de variables, séparées si nécessaire de virgules (ce sont ses paramètres). La parenthèse fermante est suivie de deux points ':'. Suit un bloc d'instructions. Elles doivent toutes être décalées du même nombre d'espaces (en général 3 ou 4). Appuyer sur entrée au prompt pour achever la définition.

Le résultat de la fonction, si il y a, est retourné grâce à l'instruction 'return'. Sinon on peut parler de procédure.

# Variables

- La notion de variable est essentielle en programmation.
- Elle permet de stocker en mémoire des valeurs, et de les utiliser et modifier à volonté au sein d'un programme.
- La valeur d'une variable évolue au cours de l'exécution d'un programme, en fonction du déroulement du programme et selon ses instructions.

```
Python 3.3.5 |Anaconda 2.0.1 (x86_64)| (default, Mar 10 2014, 11:22:25)
[GCC 4.0.1 (Apple Inc. build 5493)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> # Exemples de VARIABLES de différents TYPES
...
>>> nbr = -12
>>> pi = 3.14159
>>> str = "Une chaîne de caractères"
>>>
```

Quelques exemples de variables de types :

- entier, (nbr)
- flottant, (pi)
- et chaîne de caractère (str).

```
Python 3.3.5 |Anaconda 2.0.1 (x86_64)| (default, Mar 10 2014, 11:22:25)
[GCC 4.0.1 (Apple Inc. build 5493)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> # Exemples de VARIABLES de différents TYPES
...
>>> nbr = -12
>>> pi = 3.14159
>>> str = "Une chaîne de caractères"
>>> nbr
-12
>>> print(nbr, pi, str)
-12 3.14159 Une chaîne de caractères
>>> str
'Une chaîne de caractères'
>>> print(str)
Une chaîne de caractères
>>> █
```

Taper leur nom au prompt retourne leur valeur. L'affichage de la valeur est mieux formaté grâce à la fonction `print()`.

```
Python 3.3.5 |Anaconda 2.0.1 (x86_64)| (default, Mar 10 2014, 11:22:25)
[GCC 4.0.1 (Apple Inc. build 5493)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> # Exemples de VARIABLES de différents TYPES
...
>>> nbr = -12
>>> pi = 3.14159
>>> str = "Une chaîne de caractères"
>>> nbr
-12
>>> print(nbr, pi, str)
-12 3.14159 Une chaîne de caractères
>>> str
'Une chaîne de caractères'
>>> print(str)
Une chaîne de caractères
>>> print("La circonférence d'un cercle de rayon 3 est",2*pi*3)
La circonférence d'un cercle de rayon 3 est 18.849539999999998
>>> █
```

On peut effectuer avec une variable de type entier, flottant, chaîne, (...) tout ce que l'on peut faire avec un type respectivement entier, flottant ,chaîne, (...).

```
>>> # Affectation
... nbr = 7
>>> nbr
7
>>> nbr = 2 * nbr + 1
>>> nbr
15
>>> nbr = nbr ** 2
>>> nbr
225
>>> nbr = nbr + 1
>>> nbr
226
>>> nbr += 1
>>> nbr
227
>>> █
```

L'opération principale pour une variable est l'affectation représentée par le symbole =.



```
>>> # Affectation
... nbr = 7
>>> nbr
7
>>> nbr = 2 * nbr + 1
>>> nbr
15
>>> nbr = nbr ** 2
>>> nbr
225
>>> nbr = nbr + 1
>>> nbr
226
>>> nbr += 1
>>> nbr
227
```

Lors d'une affectation l'expression à droite du symbole = est évaluée, avant d'être affecté à la variable dont le nom figure à gauche du symbole =.

Le membre de gauche de = ne peut être que le nom d'une variable. Si elle n'existe pas encore la variable sera créée lors de l'affectation.

Si un même nom de variable figure des 2 côtés de =, la valeur de celle de droite est celle AVANT l'affectation, celle de gauche est celle APRES l'affectation.

L'affectation  $x = x + 1$  n'a rien à voir avec l'équation mathématiques impossible  $x = x + 1$ .

```
>>> # Affectation
... nbr = 7
>>> nbr
7
>>> nbr = 2 * nbr + 1
>>> nbr
15
>>> nbr = nbr ** 2
>>> nbr
225
>>> nbr = nbr + 1
>>> nbr
226
>>> nbr += 1
>>> nbr
227
```

L'instruction :

$$\text{variable } \dagger = \text{expression}$$

est équivalente à l'instruction :

$$\text{variable} = \text{variable } \dagger \text{ expression}$$

où  $\dagger$  désigne n'importe quelle opération : +, -, \*, /, //, %, \*\*, etc....

# Variables

Une **variable** a :

- un **identifiant** : pour nous c'est son nom, qui permet de manipuler la variable au sein d'un programme ou d'une instruction (en mode console). C'est une chaîne de caractère alphanumérique, c.à.d. composée de lettres et de chiffres (et du symbole '\_'), qui ne doit pas débuter par un chiffre. Eviter de le débuter par une lettre majuscule, que l'on réservera aux fonctions. Son nom doit être clair pour faciliter la relecture du programme.
- Un **type** : entier (relatif), flottant (réel...), complexe, chaîne de caractère, etc...
- Un **contenu**, c'est sa valeur. Elle est stockée dans la mémoire sous forme d'un nombre en écriture binaire.

En python la définition (déclaration) d'une variable se fait à l'aide d'une affectation : `variable = valeur` (voir des exemples plus haut).

```
>>> type(nbr)
<class 'int'>
>>> type(pi)
<class 'float'>
>>> type(str)
<class 'str'>
```

La fonction `type(.)` retourne le type d'une variable :

1. `int` pour un type entier,
2. `float` pour un type flottant,
3. `str` pour une chaîne de caractère, etc...

Python pratique le *typage dynamique* : il n'est pas besoin (au contraire d'autres langages) de déclarer le type d'une variable, python s'en charge.

```
>>> a = 1
>>> type(a), a
(<class 'int'>, 1)
>>> a = 1.0
>>> type(a), a
(<class 'float'>, 1.0)
```

Le type d'une variable peut être modifié... C'est cependant à déconseiller !

# Une particularité de l'affectation de variables sous python

Python permet en une seule instruction d'affectation ('=') d'affecter plusieurs variables :

```
>>> # Affectations multiples :  
... varnbr, varstr = 12, "bonjour"  
>>> varnbr  
12  
>>> varstr  
'bonjour'  
>>> █
```

Les variables, à gauche de l'instruction '=' d'affectation, sont séparées par des virgules, de même que les valeurs, à droite de '=', qui doivent être en même nombre, et sont affectées de gauche à droite.

# Une particularité de l'affectation de variables sous python

Exemple : Calcul des premiers termes de la suite de Fibonacci :

$$u_0 = 0, u_1 = 1, \quad \forall n \in \mathbb{N}, u_{n+2} = u_{n+1} + u_n$$

```
>>> # Calcul des premiers termes de la suite de Fibonacci
... u, v = 0, 1 ; print(u,v)
0 1
```

Remarquer l'utilisation du point-virgule pour séparer plusieurs instructions sur une même ligne !

```
>>> u, v = u+v, u+2*v ; print(u,v)
1 2
```

```
>>> u, v = u+v, u+2*v ; print(u,v)
3 5
>>> u, v = u+v, u+2*v ; print(u,v)
8 13
```

La touche ▲ du clavier (déplacement vers le haut) permet de relancer la ligne d'instructions précédentes. Elle permet plus généralement par des appuis répétés de relancer l'une quelconque des lignes précédentes.

# Une particularité de l'affectation de variables sous python

Bien noter que durant une affectation multiple les valeurs (à droite du =) sont celles avant l'appel de l'instruction.

Ainsi l'instruction `a,b = b,a` échange les valeurs de 2 variables a et b :

```
>>> # Echange des valeurs de 2 variables
... a, b = 1, 2
>>> print(a,b)
1 2
>>> a,b = b,a
>>> print(a,b)
2 1
```

Comparer avec l'instruction suivante :

```
>>> a,b = b,a
>>> print(a,b)
2 1
>>> a = b = a
>>> print(a,b)
2 2
```

qui est équivalente à deux affectations simultanées : `b=a` suivie de `a=b` (l'affectation `a=b=a` est effectuée successivement de la droite vers la gauche), ou encore `b=a ; a=b`.

# Une particularité de l'affectation de variables sous python

Dans d'autres langages pour échanger les valeurs de 2 variables il faut faire appel à une fonction prédéfinie (souvent `swap(.,.)`), ou procéder en plusieurs affectations.

A l'aide d'une variable temporaire :

```
>>> a, b = 1, 2
>>> vartemp = b ; b = a ; a = vartemp
>>> print(a,b)
2 1
```

Sans variable temporaire :

```
>>> a, b = 1, 2
>>> a = a+b ; b = a-b ; a = a-b
>>> print(a,b)
2 1
```

il faut tout de même 3 affectations, et le code n'est pas facilement lisible...



## Etat de la mémoire durant l'exécution :

```
>>> a, b = 1, 2  
>>> a = a+b ; b = a-b ; a = a-b  
>>> print(a,b)  
2 1
```

1.  $a, b = 1, 2$

a	Valeur initiale $\alpha$ (=1)
b	Valeur initiale $\beta$ (=2)

2.  $a = a+b$

a	$\alpha + \beta$ (=3)
b	$\beta$ (=2)

3.  $b = a-b$

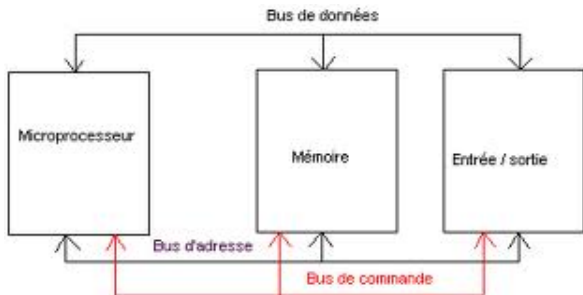
a	$\alpha + \beta$ (=3)
b	$(\alpha + \beta) - \beta = \alpha$ (=1)

4.  $a = a-b$

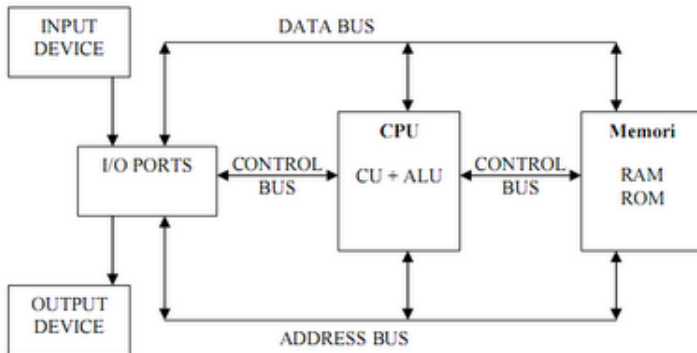
a	$\beta$ (=2)
b	$\alpha$ (=1)

La deuxième ligne a permis d'échanger les valeurs contenues dans les variables a et b.

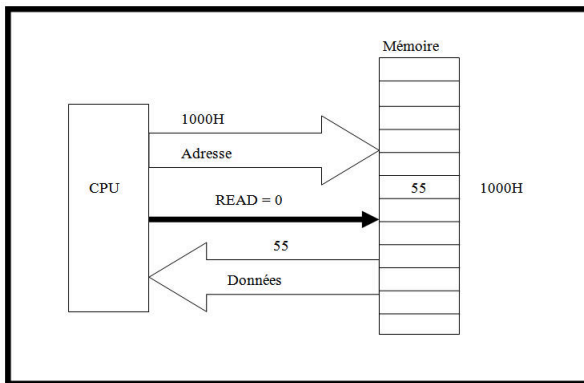
# Architecture des ordinateurs



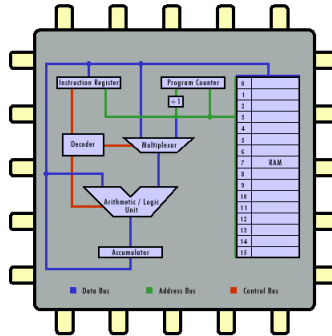
Le microprocesseur est relié à la mémoire centrale et aux périphériques d'entrées-sorties par le bus qui permet le transfert de données.



Le microprocesseur est relié à la mémoire centrale et aux périphériques d'entrées-sorties par le bus qui permet le transfert de données.



Exemple d'accès du CPU à la mémoire centrale (RAM : Random Access Memory). Toutes les données en mémoire ou ailleurs sont codées sous forme de nombres binaires ! Un CPU ne manipule que des nombres écrit en base 2 : des 0 et des 1.



Le CPU est essentiellement composé d'une unité de commande (UC) reliée à une horloge (donnant la fréquence du microprocesseur), d'une unité d'opérations arithmétiques et logiques binaires (ALU) et de mémoire spécifique (les registres où sont chargées les données avant et après calcul).

WinHex - [PW0-X00euro.bin]																
File Edit Search Position View Tools Options File Manager Window Help																
PW0X00euro.bin																
Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00000000	13	16	9A	16	96	16	8F	00	1F	15	67	00	9A	16	5D	15
00000010	65	15	37	01	0C	16	CD	00	9A	16	D1	00	9A	16	9A	16
00000020	16	9A	36	16	9A	16	F2	00	F9	2B	57	2C	33	2C	45	2C
00000030	9B	18	96	2D	B2	2E	B4	2E	08	0E	0E	00	E5	CE	D5	1A
00000040	A2	18	42	55	67	00	01	F5	55	C5	56	0B	CE	0C	C5	06
00000050	2F	C5	07	15	CA	04	C5	07	98	02	52	F2	D5	51	65	52
00000060	ES	CC	A2	08	D5	A2	07	ES	CE	D5	A2	18	42	55	FA	
00000070	C5	56	0B	C9	02	96	01	C5	56	0A	C9	02	86	02	D5	07
00000080	F5	55	52	F2	D5	51	65	52	ES	CC	A2	08	D5	1A	02	57
00000090	20	00	03	23	32	C9	03	32	CC	2E	62	36	00	C5	2C	29
000000A0	CD	26	30	F9	B5	1A	98	40	00	C5	41	98	0B	B5	18	15
000000B0	C5	2E	19	B5	34	98	FF	FF	C5	41	1C	C5	10	1A	77	05
000000C0	D5	13	53	D5	13	C5	10	18	C5	ED	98	47	FF	32	11	29
000000D0	02	E5	CE	D5	1A	A2	18	A4	15	F5	DF	86	01	C6	03	CA
000000E0	0A	DA	42	07	B5	DC	7C	3A	C5	42	0B	E5	CC	A2	08	D5
000000F0	1A	02	E5	1A	55	67	10	00	31	32	EF	30	0F	EB	30	12
00000100	C5	18	0F	C9	0A	C4	2E	08	C4	BA	98	2D	03	CD	03	C4
00000110	2E	18	E5	6A	D5	B0	E5	3A	D5	36	77	01	32	D3	31	00
00000120	C5	24	1C	32	95	29	03	2D	02	67	11	00	D5	1A	B5	04
00000130	98	02	01	57	22	00	01	E5	1A	55	31	ED	C4	BA	98	2D
00000140	C4	20	18	CE	0C	C5	19	0F	C4	18	08	C5	46	08	03	9F
00000150	02	F5	E3	86	01	EF	30	3B	C5	19	0F	CE	13	C4	18	08
00000160	CE	0E	88	D6	03	CE	06	C4	2E	19	C4	1A	18	78	CB	23
00000170	C4	1A	08	C4	EB	98	2D	03	04	C9	17	C4	21	19	CA	09
00000180	C6	08	CA	0B	C4	2E	1D	CB	09	D8	9A	03	E9	9A	F5	C4
00000190	2E	1C	FA	D5	E3	D6	03	D5	E4	F4	9A	86	01	E8	31	30
000001A0	C5	46	08	CE	0E	88	D6	0F	CE	06	C4	2E	1A	C4	1A	19
000001B0	78	CB	1D	C4	1A	09	C4	BC	98	07	C6	10	C9	07	CD	0F
000001C0	D9	21	09	CB	0A	C4	21	09	F5	E4	C9	04	C4	2E	1E	FA
000001D0	DA	9A	D6	0F	CE	15	F5	E7	C9	05	C5	E7	17	CB	09	62
000001E0	1A	02	C2	28	77	01	CD	01	FA	D5	E5	EF	30	03	D8	1A
000001F0	0D	C5	E3	D0	FC	F4	9A	D6	03	C5	E3	E1	D5	E4	E8	31
00000200	03	D9	1A	09	C4	9A	D0	FC	F5	E4	C4	9A	E1	95	EF	30
00000210	03	D8	1A	06	E8	31	06	E9	1A	03	DE	31	01	85	C4	21
00000220	3B	CD	03	C5	FE	1E	EE	2B	04	C4	2E	D0	8F	EC	17	6F
00000230	EA	21	1F	62	99	01	F5	E5	63	F4	9A	C4	02	86	04	D6
00000240	07	C2	C2	CE	5A	F4	98	C2	C2	C9	07	C2	17	CA	03	C2

Le microprocesseur obéit à des instructions chargées en mémoire centrale et codées en *langage machine* (les instructions sont des nombres, souvent écrits en base 16 ou hexadécimal).

59	I		
60	LECT:	EDU #	;Point d'entree
61 9395 211792	LD	HL,LIT	;Lecture disque
62 9398 CDD4BC	CALL	FIND	;KL FIND COMMAND
63 939B 220C90	LD	(AD),HL	
64 939E 79	LD	A,C	
65 939F 320290	LD	(AD+2),A	
66 93A2 3A0492	LD	A,(DRIVE)	
67 93A5 5F	LD	E,A	;No de drive
68 93A6 3A0592	LD	A,(P:STE)	
69 93A9 57	LD	D,A	;No de piste
70 93AA 3A0692	LD	A,(SECT)	
71 93AD 4F	LD	C,A	;No de secteur
72 93AE 210390	LD	HL,BUFF	
73 93B1 DF	RST	24	;Activ. instruction en RD
74 93B2 0090	DW	AD	
75 93B4 C9	RET		
76	END		

Au plus bas niveau un microprocesseur se programme en *assembleur* qui est une traduction lisible du langage machine. C'est un langage impératif séquentiel (non structuré). Tout langage de programmation, après interprétation ou compilation est transformé dans un langage de bas niveau converti en langage machine. Au lancement du programme ce code est chargé en mémoire centrale avant d'être exécuté.

# Ecriture d'un nombre en base 2

**Rappel : Division euclidienne.**  $\forall n \in \mathbb{N}, \forall m \in \mathbb{N}^* :$

$$\exists! (q, r) \in \mathbb{N}^2, \quad \text{tel que} \quad \begin{cases} n = q \times m + r \\ 0 \leq r < m \end{cases}$$

**Corollaire.** *Pour tout entier  $m \in \mathbb{N} \setminus \{0, 1\}$ , pour tout entier  $n \in \mathbb{N}$ , il existe une unique suite finie  $(u_n)_{n \in \llbracket 0, N \rrbracket}$  d'entiers compris entre 0 et  $m - 1$ , tels que :*

$$n = \sum_{i=0}^N u_i \times m^i$$

et si  $N > 0$ ,  $u_N \neq 0$ . L'écriture en base  $m$  de  $n$  est  $u_N u_{N-1} \cdots u_1 u_0$ .



# Ecriture d'un nombre en base $p \in \mathbb{N}^*$

**Preuve.** Existence : par division euclidienne  $\exists ! (q_0, u_0) \in \mathbb{N}^2$  avec  $n = q_0 \times m + u_0$  et  $0 \leq u_0 < m$ . Si  $q_0 = 0$  la suite  $u_0$  convient puisque  $n = u_0 \times m^0$ . Si  $q_0 \neq 0$  alors par division euclidienne  $\exists ! (q_1, u_1) \in \mathbb{N}^2$  avec  $q_0 = q_1 \times m + u_1$  et  $0 \leq u_1 < m$ . Remarquer que  $n > q_0 > q_1$  puisque  $m > 1$ . Ainsi en poursuivant ce procédé on construit une suite finie  $(u_0, u_1, \dots, u_N)$  avec  $0 \leq u_i < m$ ,  $q_N = 0$  et :

$$\begin{aligned} n &= q_0 \times m + u_0 = (q_1 \times m + u_1) \times m + u_0 \\ &= ((\dots((0 \times m + u_N) \times m + u_{N-1}) \dots) \times m + u_1) \times m + u_0 = \sum_{i=0}^N u_i \times m^i. \end{aligned}$$

Unicité. Elle provient de l'unicité du quotient et du reste dans la division euclidienne : si  $n = \sum_{i=0}^N u_i \times m^i$  alors le procédé précédent produit la suite  $(u_0, u_1, \dots, u_N)$ .

Exemple : Ecriture de 72 en base 2 :

$$72 = 2 \times 36 + 0 \quad u_0 = 0$$

$$36 = 2 \times 18 + 0 \quad u_1 = 0$$

$$18 = 2 \times 9 + 0 \quad u_2 = 0$$

$$9 = 2 \times 4 + 1 \quad u_3 = 1$$

$$4 = 2 \times 2 + 0 \quad u_4 = 0$$

$$2 = 2 \times 1 + 0 \quad u_5 = 0$$

$$1 = 2 \times 0 + 1 \quad u_6 = 1$$

72 s'écrit  $1001000_2$  en base 2 (ou binaire).

En python :

```
>>> a = 72
>>> while a>0:
...     print(a%2)
...     a = a//2
...
0
0
0
1
0
0
1
```

La "boucle" while s'exécute tant que la condition  $a>0$  est satisfaite.

Exemple : Ecriture de 72 en base 16 :

$$72 = 16 \times 4 + 0 \quad u_0 = 8$$

$$4 = 16 \times 0 + 4 \quad u_1 = 4$$

72 s'écrit  $48_{16}$  ou  $0x48$  en base 16 (ou hexadécimal).

```
>>> a = 72
>>> while a>0:
...     print(a%16)
...     a = a//16
...
8
4
```

Pour écrire un nombre en base 16 on utilise les "chiffres" de 0 à 9 ainsi que :

A=10, B=11, C=12, D=13, E=14, F=15.

En 2 chiffres on écrit tous les nombres de 0 à  $FF = 15 + 15 \times 16 = 16^2 - 1 = 255$ .

Autant qu'en binaire avec 8 'bits' (ou chiffre 0,1) car  $11111111 = 2^8 - 1 = 255$ .

Puisque  $2^4 = 16$  on a la conversion binaire/hexadecimal :

$0000_2 = 0$ ,  $0001_2 = 1$ ,  $0010_2 = 2$ ,  $0011_2 = 3$ ,  $0100_2 = 4$ ,  $0101_2 = 5$ ,  
 $0110_2 = 6$ ,  $0111_2 = 7$ ,  $1000_2 = 8$ ,  $1001_2 = 9$ ,  $1010_2 = A$ ,  $1011_2 = B$ ,  
 $1100_2 = C$ ,  $1101_2 = D$ ,  $1110_2 = E$ ,  $1111_2 = F$ .

Ex :  $72_{10} = 01001000_2 = 48_{16}$ .

Voilà pourquoi un ordinateur fonctionne en 8, 16, 32 ou 64 bits  
(multiples de 4... par une puissance de 2)...