2 - Les éléments de base

1. ***Livre: Apprendre à programmer avec Python3 de Gérard Swinnen***
2. ***Le Mooc: Apprendre à coder en Python:***

https://www.fun-mooc.fr/courses/course-v1:ulb+44013+session03/about

Plan

- Les variables
- L’affectation
- Les opérateurs
- Les entrées - Sorties
- Les structures conditionnelles

Les noms de variables

- ***Les noms de variables sont des noms qu'on choisit assez librement de préférence assez courts, mais aussi explicites que possible, pour exprimer clairement ce que la variable est censée contenir :***

- **Quelques règles pour les noms de variables sous Python :**

1. Un nom de variable est une séquence de lettres (a à z , A à Z) et de chiffres (0 à 9), qui doit toujours commencer par une lettre.
2. Seules les lettres ordinaires sont autorisées. Les lettres accentuées, les cédilles, les espaces, les caractères spéciaux tels que $, #, @, etc. sont interdits, à l'exception du caractère _ (souligné).
3. **La casse est significative** (les caractères majuscules et minuscules sont distingués). ***Attention :*** *prix, PRIX, Prix sont donc des variables différentes.*
4. Prenez l'habitude d'écrire l'essentiel des noms de variables en caractères minuscules (y compris la première lettre). Il s'agit d'une simple convention, mais elle est largement respectée. N'utilisez les majuscules qu'à l'intérieur même du nom, pour en augmenter éventuellement la lisibilité, comme dans *TableDesMatieres*.

Noms de variables et mots réservés

En plus de ces règles, il faut encore ajouter que vous ne pouvez pas utiliser comme noms de variables les 29 « mots réservés » au langage ci-dessous :

and	**assert**	**break**	**class**	**continue**	**def**
del	**elif**	**else**	**except**	**exec**	**finally**
for	**from**	**global**	**if**	**import**	**in**
is	**lambda**	**not**	**or**	**pass**	**print**
raise	**return**	**try**	**while**	**yield**	

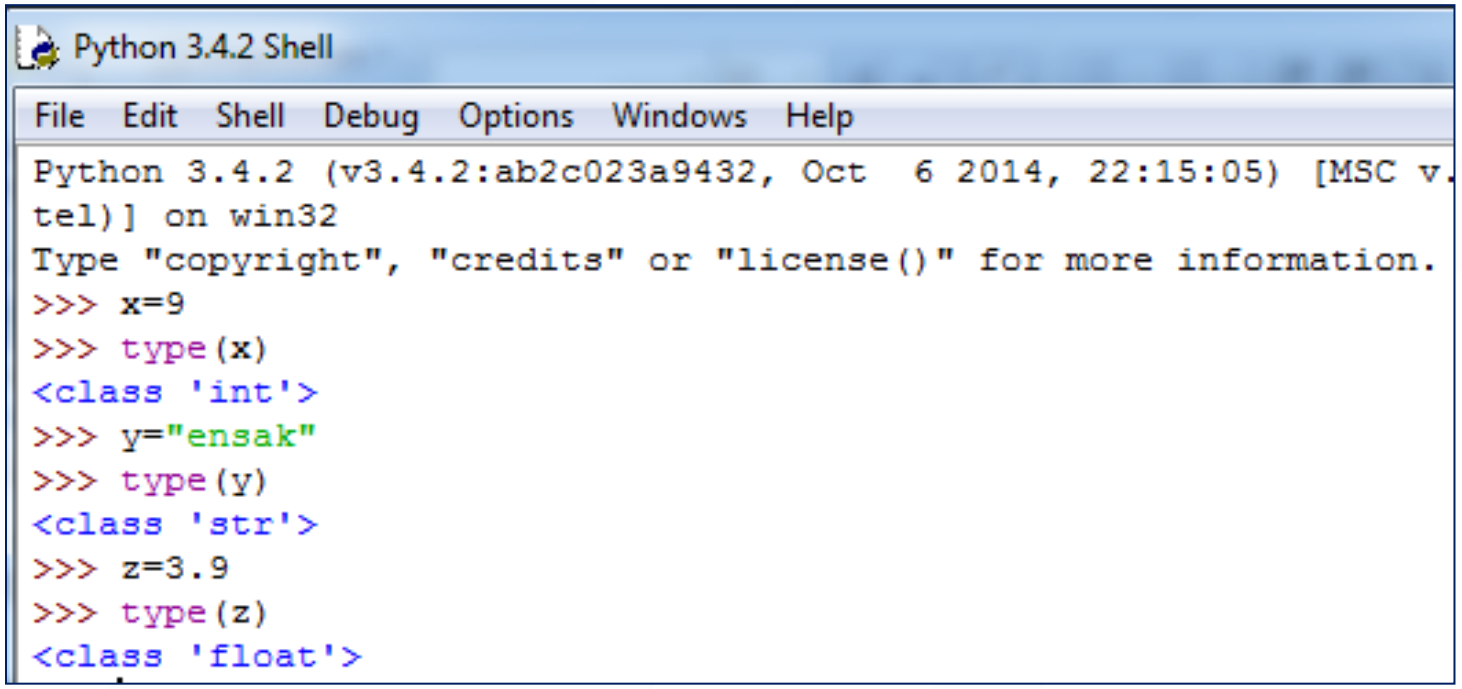
Les types de variables en python

- En python, le type de la variable est déterminé au moment de l'affectation.

Typage dynamique

- Le type d'une variable peut changer au cours du temps.
- On peut connaitre le type courant d'une variable avec la fonction **type**.

Syntaxe: **type(nom_variable)**



Type numériques

- **int et long :** entier et entier long. La taille des entiers n'est limitée que par la mémoire de la machine :
- **Bool:** Deux valeurs possibles : **F**alse, **T**rue.
- ***Float***: Un float est noté avec un point décimal ou en notation exponentielle :

```
2.718
.02
3e8
6.023e23
```

- Les flottants supportent les mêmes opérations que les entiers.

Type numériques

- complex, nombres complexes : 3.2+2.5j
  - Les complexes sont écrits en notation cartésienne formée de deux flottants.
  - La partie imaginaire est suffixée par j :

```
print(1j)           # 1j
print((2+3j) + (4-7j)) # (6-4j)
print((9+5j).real) # 9.0
print((9+5j).imag) # 5.0
print((abs(3+4j)))  # 5.0 : module
```

Fonctions mathématiques: le module math

- Un module est une bibliothèque (un regroupement de fonctions prédéfinies) qui une fois importée permet d'accéder à de nouvelles fonctions. Il en existe plusieurs en python. :
  - le module turtle qui permet de réaliser des dessins géométriques,
  - le module numpy qui permet de faire du calcul scientifique,
  - le module sympy qui permet de faire du calcul formel,
  - le module matplotlib qui permet de faire des graphiques en tout genre.
  - le module math qui permet d'avoir accès aux fonctions mathématiques comme le cosinus (cos), le sinus (sin), la racine carrée (sqrt), le nombre (pi) et bien d'autres

Importer le module math

Méthode 1

```
>>> from math import sqrt
>>> sqrt(25)
5.0
```

```
>>> from math import *
>>> print(pi)
3.141592653589793
```

Méthode 2

```
>>> import math
>>> math.sqrt(25)
5.0
```

Le module math

```
>>> import math
>>> help(math)
```

```
sin(x, /)
    Return the sine of x (measured in radians).

sinh(x, /)
    Return the hyperbolic sine of x.

sqrt(x, /)
    Return the square root of x.

tan(x, /)
    Return the tangent of x (measured in radians).

tanh(x, /)
    Return the hyperbolic tangent of x.

trunc(x, /)
    Truncates the Real x to the nearest Integral toward 0.
```

Type alphanumériques

- **Le type « string » (chaîne de caractères)**

Une donnée de type ***string*** est une suite quelconque de caractères délimitée soit par des apostrophes (*simple quotes*), soit par des guillemets (*double quotes*).

Exemple:

```
phrase1 = "l'ENSAK ?"
phrase2 = 'vous êtes'
phrase3 = "un étudiant de"
print (phrase2, phrase3, phrase1)
```

Les 3 variables **phrase1**, **phrase2**, **phrase3** sont donc des variables de type ***string***

Remarques :

- La séquence **\n** dans une chaîne provoque un saut à la ligne.
- La séquence **\'** permet d'insérer une apostrophe dans une chaîne délimitée par des apostrophes

```
>>> print(' les étudiants de l\'Ensak ')
 les étudiants de l'Ensak
```

```
>>> print("ilham \n oumaira")
ilham
 oumaira
```

Accès aux caractères individuels d'une chaîne

Python est pourvu de mécanismes qui permettent d'accéder séparément

à chacun des caractères d'une chaîne

Exemple :

```
>>> ch = "ENSAK"
>>> print (ch[0], ch[3]) #affiche EA
```

Les opérateurs

Opérations élémentaires sur les chaînes

concaténation

```
a = 'Nous sommes des'
b = ' débutants en Python'
c = a + b
print (c)
```

```
Nous sommes des débutants en Python
```

len()

```
>>> print (len(c))
35
```

Conversion en nombre d'une chaîne de caractères qui représente un nombre

```
>>> ch = '8647'
>>> print (ch + 45)
==> *** erreur *** on ne peut pas additionner une chaîne et un nombre
>>> n = int(ch)
>>> print (n + 65)
8712 # OK : on peut additionner 2 nombres
```

Opérateurs arithmétiques

Opération	Résultat
-x	Opposé de *x*
x % y	Reste de la division entière de *x* par *y* (*mod*)
x / y	Division de *x* par *y*
x // y	Division entière de *x* par *y* (*div*)
*x * y*	Produit de *x* et *y*
x – y	Différence de *x* et *y*
x + y	Somme de *x* et *y*
*x ** y*	*x* puissance *y*

Opérations sur les numériques

Opération	Résultat
pow(x,y)	*x* puissance *y*
divmod(x,y)	La paire (*x//y,x%y*)
c.conjugate()	Le conjugué de *c* (si *c* ℂ)
float(x)	Conversion de *x* en flottant
long(x)	Conversion de *x* en entier long
int(x)	Conversion de *x* en entier
str(x)	Conversion de *x* en chaîne de caractère

Priorité des opérations
PEMDAS pour le mémoriser !

1. **P** pour ***parenthèses***. Ce sont elles qui ont la plus haute priorité. Elles vous permettent donc de « forcer » l'évaluation d'une expression dans l'ordre que vous voulez. Ainsi **2*(3-1) = 4** , et **(1+1)**(5-2) = 8**.
2. **E** pour ***exposants***. Les exposants sont évalués avant les autres opérations. Ainsi **2**1+1 = 3** (et non 4), et **3*1**10 = 3** (et non 59049 !).
3. **M** et **D** pour ***multiplication*** et ***division***, qui ont la même priorité. Elles sont évaluées avant l'***addition* A** et la ***soustraction* S**, lesquelles sont donc effectuées en dernier lieu. Ainsi **2*3-1 = 5** (plutôt que 4),

- Si deux opérateurs ont la même priorité, l'évaluation est effectuée de gauche à droite. Ainsi dans l'expression **59*100/60**, la multiplication est effectuée en premier, et la machine doit donc ensuite effectuer **5900/60**, ce qui donne **98**. Si la division était effectuée en premier, le résultat serait **59**

Opérateurs de comparaison

Opération	Résultat
x<*y*, *x*<=*y*	*x* inférieur (ou égal) à *y*
x>*y*, *x*>=*y*	*x* supérieur (ou égal) à *y*
x == *y*	*x* égal ? y (valeurs)
x != *y*	x différent de *y*

Opérateurs logiques

Opération	Résultat
x or y, x and y	OU et ET logiques
x | *y*	*x* ou *y*
x & *y*	*x* et *y*
x^ *y*	*x* ou exclusif *y*

L’affectation

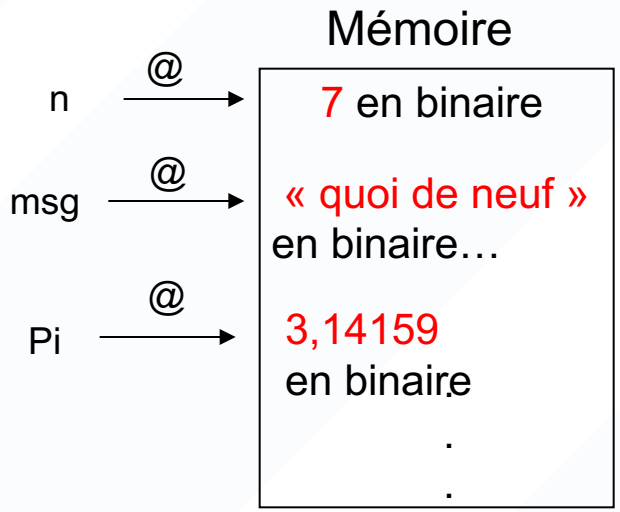
Affectation (ou assignation)

En Python comme dans de nombreux autres langages, l'opération d'affectation est représentée par le signe ***égal*** :

>>> n = 7 # donner à n la valeur 7

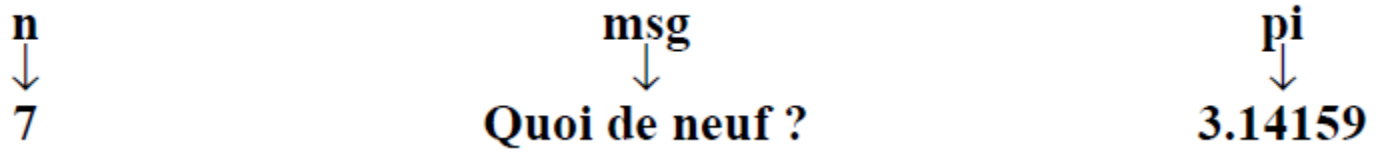
>>> msg = "Quoi de neuf ?" # affecter la valeur "Quoi de neuf ?" à msg

>>> pi = 3.14159 # assigner sa valeur à la variable pi



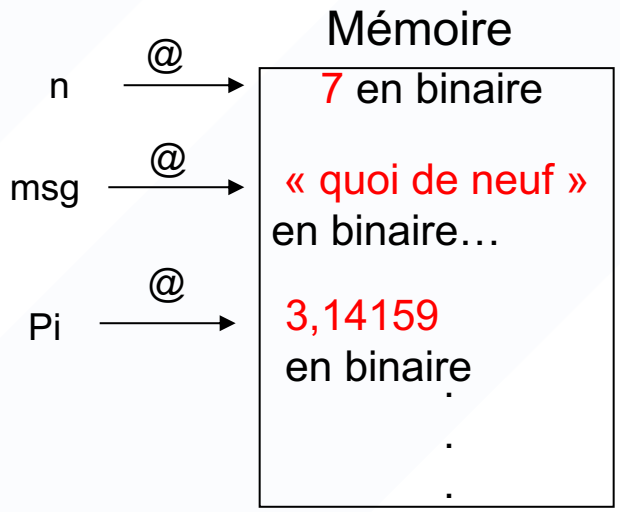
Affectation (ou assignation)

- Les trois instructions d'affectation ci-dessus ont eu pour effet chacune de réaliser plusieurs opérations dans la mémoire de l'ordinateur :
  - créer et mémoriser une valeur particulière ;
  - créer et mémoriser un nom de variable ;
  - lui attribuer un type bien déterminé
  - établir un lien (par un système interne de pointeurs) entre le nom de la variable et l'emplacement mémoire de la valeur correspondante
- On peut mieux présenter tout cela par un diagramme d'état tel que celui-ci :



Affectation (ou assignation)

- Les trois noms de variables sont des ***références, mémorisées dans une zone particulière de la*** mémoire que l'on appelle ***espace de noms, alors que les valeurs correspondantes sont situées*** ailleurs, dans des emplacements parfois fort éloignés les uns des autres.

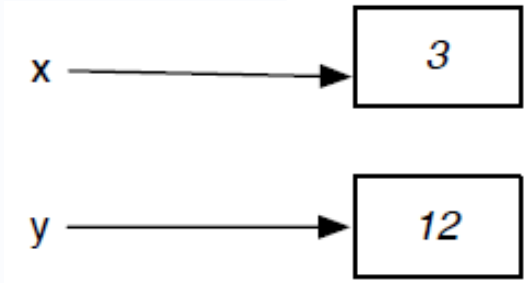


Exemples

- L’instruction x = 3 crée une variable qui référencie un objet de type entier valant 3

- L’instruction y =12 crée une variable qui référencie un objet de type entier valant 12

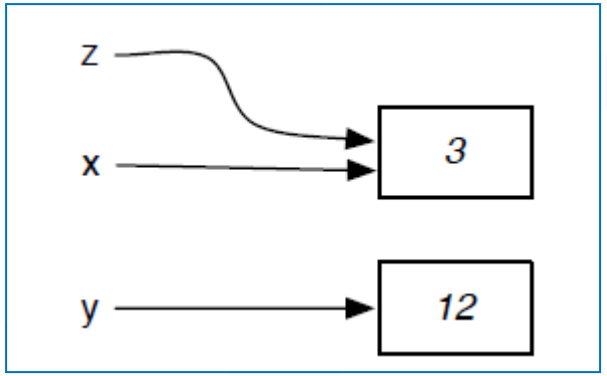
```
x = 3
y = 12
```



Exemples

- L'instruction z = x ne crée aucun nouvel objet ; mais la variable z est créée et reçoit la valeur de x, c'est-à-dire la référence vers l'objet de type entier et valant 3.
- Après ces 3 premières instructions, le diagramme d'état suivant illustre la situation :

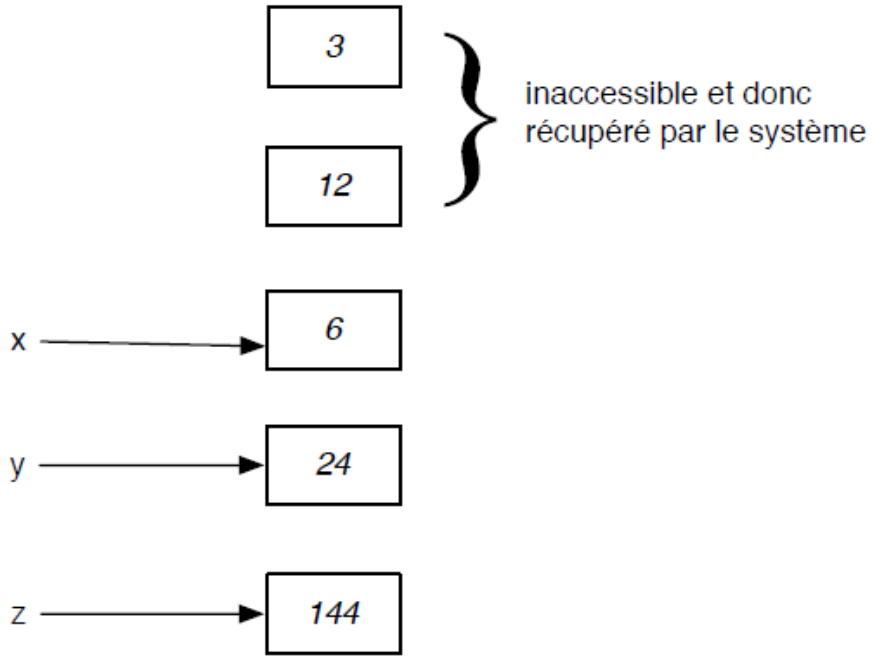
```
x = 3
y = 12
z = x
```



Exemples

- A la fin du programme entier, 5 objets (de type) entiers ont été créés, dont ceux valant 6, 24 et 144 sont référencés respectivement par les variables x, y et z. Les objets valant 3 et 12 ne sont plus accessibles et donc potentiellement récupérés par le système (garbage collector).

```
x = 3
y = 12
z = x
x = 2 * x
y = 2 * y
z = x * y
```



Affectations multiples

Sous Python, on peut assigner une valeur à plusieurs variables simultanément.

Exemple :

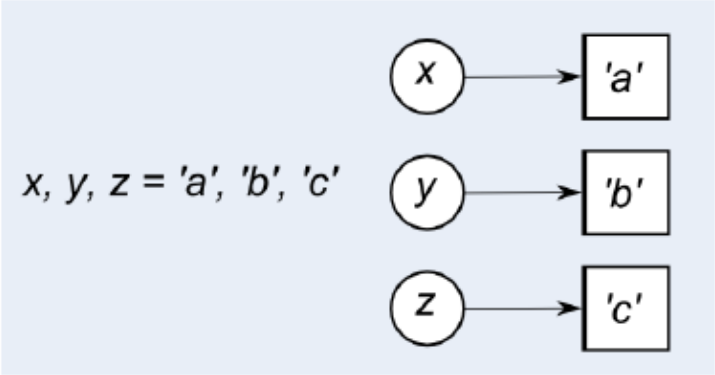
```
>>> x = y = 7
>>> x
7
>>> y
7
```

On peut aussi effectuer des ***affectations parallèles*** à l'aide d'un seul opérateur :

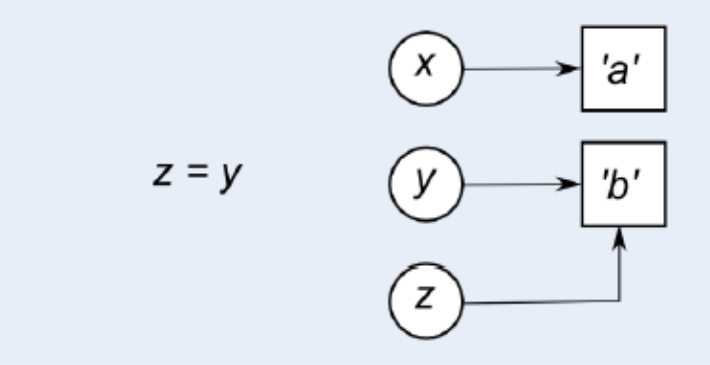
```
>>> a, b = 4, 8.33
>>> a
4
>>> b
8.33
```

Dans cet exemple, les variables **a** et **b** prennent simultanément les nouvelles valeurs 4 et 8,33.

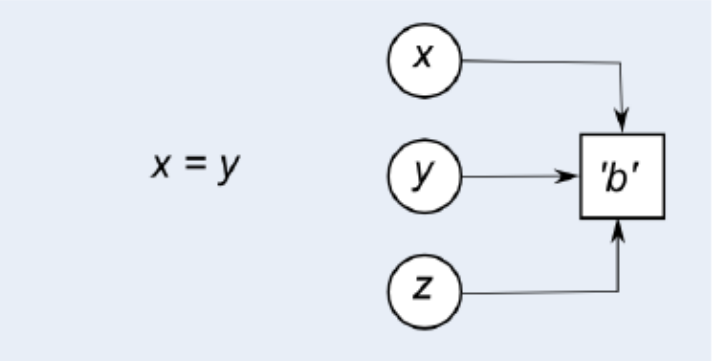
Affectations multiples



(a) Trois affectations



(b) La donnée `'c'` est supprimée



(c) La donnée `'a'` est supprimée

Les entrées sorties

Les entrées

- Il s'agit de réaliser une *saisie à l'écran : la fonction standard* ***input()*** *interrompt le* programme, afficher une éventuelle invite et attend que l'utilisateur entre une donnée et la valide par *Entrée* .
- La fonction standard input() effectue toujours une saisie en *mode texte (la saisie est* une chaîne) dont on peut ensuite changer le type (on dit aussi *transtyper) :*
- *Syntaxe:*

```
Nom_variable = input("un texte qui invite l'utilisateur à entrer une donnée")
```

Ou bien on sépare, comme en algorithmique: d'abord Ecrire ensuite lire

```
Print ("un texte qui invite l'utilisateur à entrer une donnée")
Nom_variable = input()
```

Exemples

```
nb=input("Entrer le nombre d'étudiants : ")
print(type(nb))
```

```
Entrer le nombre d'étudiants : 230
<class 'str'>
```

```
a=float(input('Entrer la valeur de a : '))
b=float(input('Entrer la valeur de b : '))
c=float(input('Entrer la valeur de c : '))
print (a,b,c)
```

```
Entrer la valeur de a : 14
Entrer la valeur de b : 6.5
Entrer la valeur de c : 22
14.0 6.5 22.0
```

Les sorties

- En mode « interactif», Python *lit-évalue-affiche, mais la fonction print() reste indispensable* aux affichages dans les scripts :

```
a, b = 2, 5
print(a, b)                          # 2 5
print("Somme :", a + b)              # Somme : 7
print(a - b, "est la différence")  # -3 est la différence
print("Le produit de", a, "par", b, "vaut :", a * b)
# Le produit de 2 par 5 vaut : 10
print()          # affiche une nouvelle ligne
```

Les paramètres de la fonction print

- Précisons simplement ici qu'elle permet d'afficher n'importe quel nombre de valeurs fournies en arguments (c'est-à-dire entre les parenthèses).
- Par défaut, ces valeurs seront séparées les unes des autres par un espace, et le tout se terminera par un saut à la ligne.
- Vous pouvez remplacer le séparateur par défaut (l'espace) par un autre caractère quelconque (ou même par aucun caractère), grâce à l'argument **« sep »**

```
>>> a="bonjour"
>>> b="à"
>>> c="tous"
>>> print(a,b,c)
bonjour à tous
>>> print(a,b,c,sep ="*")
bonjour*à*tous
```

Les paramètres de la fonction print

vous pouvez remplacer le saut à la ligne terminal avec l'argument « **end** » :

```
>>> i=1
>>> while(i<=5):
        print(i)
        i=i+1



1
2
3
4
5
```

```
>>> i=1
>>> while(i<=5):
        print(i,end=" ")
        i=i+1



1 2 3 4 5
```

```
>>> i=1
>>> while(i<=5):
        print(i,end="*")
        i=i+1



1*2*3*4*5*
```

Les commentaires

- Pour rendre un script plus lisible pour les autres programmeurs qui voudraient comprendre son fonctionnement, on peut rajouter des explications. Pour cela on peut, à la fin de chaque ligne de code, mettre le caractère **#** suivi de texte explicatif. Ce texte s'appelle du *commentaire* et ne sera pas utilisé par l'interpréteur.
- Nous pouvons aussi, n'importe où, rajouter un commentaire multiligne entouré de **3** simples quottes ou double quottes. Nous pouvons mettre une explication plus complète de ce que fait le script dans ce qu'on appel un **docstring** initial :
  - l'auteur
  - la date
  - le but du programme
  - les données reçues (input)
  - les résultats affichés (print)

Les commentaires: Syntaxe

- Ligne de commentaire :

```
# Ceci est un commentaire
```

- En fin de ligne :

```
a = 2 # Ceci est un commentaire
```

- Commentaire sur plusieurs lignes:

```
" " "
```

On écrit le commentaire sur plusieurs lignes

```
" " "
```

Les commentaires: Exemple

commentaire.py - /Users/oumaira/Documents/Python 2019/cours 2019/exemples/commentaire.py (3.7.2)

```
""" Auteur: Oumaira Ilham
    date : Avril 2020
    But du programme :Le programme suivant calcule la surface
    d'un disque dont le rayon est donné en input
    Entrée: rayon : le rayon du disque
    Sortie: la surface du disque
"""
pi=3.14
rayon = float(input("Veuillez donner le rayon : "))  # lecture du rayon de mon disque
S = pi* rayon**2                          # calcul de la circonférence
# Affichage des résultats
print("Surface : ", S)
```

Les structures conditionnelles

Structure conditionnelle

```
if condition :
    instructions
```

1ère forme de Si

```
if condition:
    instructions
else:
    instructions
```

2ème forme de Si

```
if condition:
    instructions
elif condition:
    instructions
else:
    instructions
```

3ème forme de Si

Structure conditionnelle

```
x=float(input("Entrez un nombre : "))
if x==0:
    print("Nul")
elif x>0:
    print("Positif")
else:
    print("Négatif")
```

If imbriqués

```
if a > 0 :
    if b > 0 :
        print("cas 1")
    else :
        print("cas 2")
```

affiche“cas 1” quand a et b sont positifs, “cas 2” quand a est positif et b négatif ou nul

Et N’affiche rien si **a est négatif ou nul**. Le else dépend du second if.

```
if a > 0 :
    if b > 0 :
        print("cas 1")
else :
    print("cas 2")
```

Affiche “cas 1” quand a et b sont positifs, “cas 2” quand a est négatif ou nul et donc rien quand

a est positif et b négatif ou nul. Le else dépend du premier if.