Cours sur les fondamentaux

python™

➔Simplicité (syntaxe)

➔Puissance (Robustesse)

…*Multiplateformes (Variété de supports)*

…*Multi paradigmes (Transversalité)*

➔Communauté (utilisation)

Gratuit : Python est placé sous Général Public License. Il est facilement téléchargeable sur www.python.org

Documentation utile: https://python-django.dev

Prof. Papa DIOP @Sciences Et Technologies -Université de Thiès /SENEGAL

Domaines d'application…

En faisant précéder par des articles au besoin, compléter le texte ci-dessous avec : **Business intelligence, Intelligence artificielle, Cloud Computing, Bases de données, internet des objets**.

*Les enjeux dans la gestion de l'information et la facilitation des traitements de problèmes de tout ordre appellent à la formation de citoyens du monde en mouvement. Ces citoyens devront être au fait des disciplines technologiques qui révolutionnent le monde dans le lequel ils se doivent de moduler réactivité et proactivité pour y trouver et occuper pleinement leur place. Ce monde est celui dans lequel ……**(1)**…… , ensemble structuré de données accessibles (via ordinateur, Smartphones et autres) à plusieurs utilisateurs en même temps, constituent fortement un vivrier de ……**(2)**…….. à travers laquelle les entreprises très motivées par la recherche du profit trouvent par-là un instrument d'aide à la prise de décision, pour booster leur activité, fidéliser et atteindre le maximum de cibles (clientèle ou partenaires). Dans ce monde, ……**(3)**…… permet à des drones conçus, réalisés et connectés d'arroser de larges surfaces agricoles. Et ……**(4)**…….. permettra d'installer un dispositif de reconnaissance faciale au niveau des aéroports pour signaler et déjouer les comportements suspects. Toutes ces masses d'informations sont managées à bien des égards sans encombrement via les services ……**(5)**……*

Pour la petite histoire…

En 1989, **Guido Van Rossum (Pays-Bas)** commença à travailler sur Python qui n'était alors qu'un projet lui servant d'occupation durant les vacances de Noël pendant lesquelles son bureau était fermé.

Le but de ***Guido Van Rossum*** était d'inventer un successeur au langage ***ABC***, un langage d'apprentissage peu apprécié dans le milieu académique.

Pour cela, il fit appel directement à des utilisateurs ***Unix*** habitués au langage ***C***. Il voulut que ***Python*** soit facilement utilisable dans d'autres langages et environnement contrairement à ***ABC***. Il y réussit globalement…

Le nom ***Python*** donné au langage ***provient d'une série anglaise*** appelée ***« Monty Python Flying Circus »*** dont il était fan (amateur).

Pour la petite histoire…

Son succès fait que ***Guido Van Rossum*** *s'est déclaré depuis le* ***12 juillet 2018 Benevolent Dictator For Life (BDFL).***

Plan sommaire

1. Instructions et Expressions
2. Variables
3. Affectations
4. Saisie de données au clavier
5. Erreur d'exécution ou de logique
6. Types de données élémentaires
7. Gestion de la mémoire
8. Structures de contrôle
9. Listes et opérations
10. Objets et Fonctions
11. Graphisme avec Tkinter

1. Les instructions et les expressions

- Python offre deux outils essentiels : les instructions et les expressions (fonctions, équations, etc.).

les instructions :

Des commandes adressées à l'interpréteur impliquant l'emploi de concepts-clés.

```
>>> print ("Ceci est mon premier programme PYTHON")
Ceci est mon premier programme PYTHON
>>>
```

L'instruction **print** permet d'afficher la donnée fournie, en l'occurrence une chaîne de caractères.

Les instructions peuvent ou non déboucher sur un résultat affiché.

Les symboles >>> et le curseur indiquent que l'interpréteur attend la prochaine instruction Python.

Règles et symboles à connaître concernant les instructions en Python

Le signe dièse (#)

Les commentaires débutent toujours par un signe dièse (#).

Un commentaire peut débuter n'importe où sur une ligne.

Tous les caractères qui suivent le # sont ignorés par l'interpréteur, jusqu'à la fin de la ligne.

Les commentaires servent à documenter les programmes et améliorer leur lisibilité. Même si Python est un langage facile à apprendre, cela ne dispense pas le programmeur d'utiliser des commentaires de manière adéquate dans son code.

```
>>> # Place aux commentaires.
>>> print (1 / 3).   # Division réelle.
0.333333333333
>>>
```

Le caractère de fin de ligne (\n)

Il s'agit du retour à la ligne suivante  ce qui signifie normalement une instruction par ligne.

La barre oblique inverse (\) (ou antislash).

Cela annonce que l'instruction n'est pas terminée et qu'elle se poursuit à la ligne suivante.

```
>>> print (10 + 5 \
        * 4 \
        - 7)
23
>>>
```

Il existe 2 cas particuliers où une instruction peut s'étaler sur plusieurs lignes sans avoir besoin de barres obliques inverses :

- Lorsqu'elle utilise des opérateurs comme les parenthèses, les crochets ou les accolades. Voir plus tard pour cet usage.
- Lorsque le caractère de retour à la ligne est inséré dans une chaîne entourée de guillemets triples.

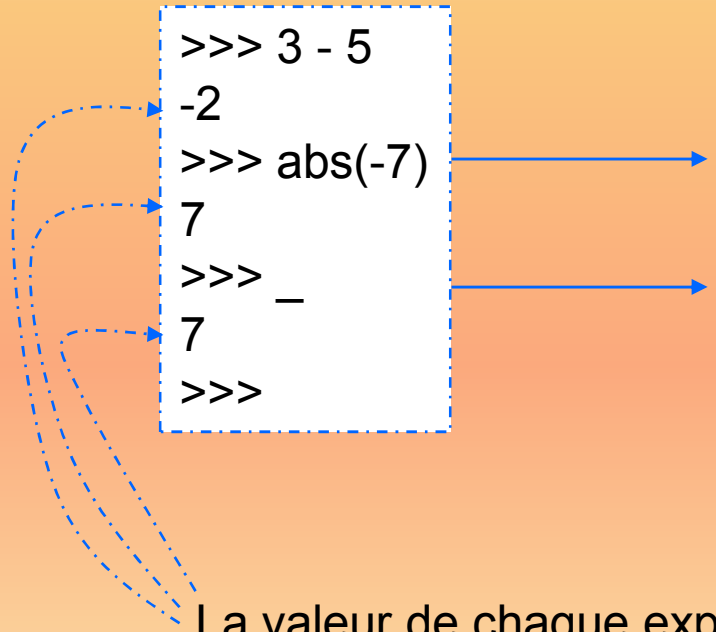
```
>>> print ('''Place
aux guillemets triples.'''‘)
Place
aux guillemets triples.
>>>
```

```
>>> print ("""Place
aux guillemets triples.""")
Place
aux guillemets triples.
>>>
```

Les expressions :

Elles n'utilisent pas de mots-clés.

Il peut s'agir de simples équations, qu'on utilise avec des opérateurs arithmétiques, ou de fonctions, qui sont appelées avec des parenthèses. Les fonctions peuvent ou non accepter une entrée et retourner ou non une valeur.



Utilisation de l'interpréteur comme simple calculatrice

```
>>> -5 + 3
-2
>>> 8 + 4 * 6
32
>>> 22 / 3
7
>>> 2 * (32 – 26)
12
>>> 22. / 3
7.333333333333333
>>>
```

Division entière

Division réelle (l'entier 3 est converti en réel) à cause de la présence du point décimal.

Python dispose de 2 opérateurs de division :

/ Si les opérandes sont tous deux des entiers, la partie entière du résultat de la division sera retenue.

Autrement, il s'agira d'une véritable division réelle avec comme résultat une valeur réelle.

Utilisation de l'interpréteur comme simple calculatrice

// La partie entière du résultat de la division, c'est-à-dire le plus grand entier inférieur ou égal au résultat indépendamment du type des opérandes.

```
>>> 11 // 3
3
>>> 23 // 8
2
>>> -6 // 5
-2
>>> 12.8 // 1.1
11.0
>>>
```

Puisque les opérandes sont réelles, le résultat est réel.

% La partie fractionnaire du résultat de la division, c'est-à-dire le résultat de la division moins sa partie entière.

```
>>> 13 % 3
1
>>> 12.8 % 1.1
0.6999999999999973
>>> -6 % 5
4
```

Si les opérandes sont entières, le résultat l'est.

Si les opérandes sont réelles, le résultat l'est.

** l'opérateur d'exponentiation.

```
>>> 3.1 ** 2
9.610000000000012
>>> 3.1 ** 2.0
9.610000000000012
>>> 3 ** -1
0.3333333333333331
>>> (-3)** 2
9
>>> 2 ** 0.5
1.4142135623730951
>>> --5
5
```

Erreurs de précision

Erreurs de précision

Opérateurs unaires

```
>>> 3.1 * 3.1
9.610000000000012
```

L'opérateur d'exponentiation a une règle de priorité particulière lorsqu'on le combine avec d'autres opérateurs : il est exécuté avant les opérateurs unaires placés à sa gauche, mais après les opérateurs unaires placés à sa droite.

```
>>> 8 ** 2
64
>>> -8 ** 2
-64
>>> 8 ** -2
0.015625
>>> (-8)**2
64
```

Priorité des opérateurs :

+ et -	*, /, //, %	+ et – unaires	**

Bas de la hiérarchie → Haut de la hiérarchie

```
>>> -3 + 5**2 -7 / 2
19
```

```
>>> -3 + 5**2 + -7 / 2
18
```

On peut utiliser les parenthèses pour clarifier la signification d'une expression ou pour outrepasser l'ordre de priorité des opérateurs : (5 – 3) * 2 + 4.

Remarque

- Le point-virgule (;) permet de regrouper 2 instructions sur la même ligne.

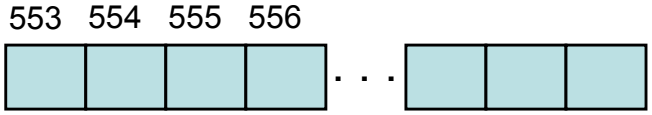
```
>>> -2 + 5;          print (5 – 2)
3
3
>>>
```

Python ne tient pas compte de la présentation : espaces et sauts de lignes.

Un programme pourrait s'écrire en quelques lignes même si ce n'est pas conseillé (attention à la mise en page : présenter un programme de façon lisible).

2. Qu'est-ce qu'une variable ?

- Les programmes doivent mémoriser les données qu'ils utilisent.
- Pour cela, les variables nous fournissent plusieurs représentations et méthodes de stockage des informations.
- Une variable est un emplacement en mémoire principale destiné à recevoir une donnée. Cette zone reçoit une valeur qui peut ensuite être réutilisée.
- La mémoire de votre ordinateur est comparable à des cases alignées une à une. Ces emplacements sont numérotés séquentiellement; il s'agit d'adresses en mémoire.



- Une variable peut occuper une ou plusieurs cases. Ex. : la case d'adresse 556.

Ex.: - une donnée numérique avec une précision plus ou moins grande,

- une chaîne de caractères plus ou moins longue.

Taille des variables

- Chaque emplacement en mémoire a la taille d'un octet, i.e. 8 chiffres binaires 0 ou 1 (8 bits ou *BInary digiTS*). La taille d'une variable dépend de son type.
- L'intérêt de la base 2 est qu'elle représente exactement ce que l'ordinateur reconnaît car les ordinateurs ne connaissent pas les lettres, les chiffres, les instructions ou les programmes.

Note : Conversion d'un nombre binaire en base 10.

1010011 en base 2 (i.e. 1010011_2) équivaut en base 10 au nombre suivant :

$$1 \times 2^0 + 1 \times 2^1 + 0 \times 2^2 + 0 \times 2^3 + 1 \times 2^4 + 0 \times 2^5 + 1 \times 2^6$$

ou encore

$$1 + 2 + 0 + 0 + 16 + 0 + 64$$

ce qui donne 83 en base 10 (i.e. 83_{10}).

- Par conséquent, un octet peut prendre les valeurs comprises entre 0 et 255 car $11111111_2 \equiv 255_{10} = 2^8 - 1$.

Conversion d'un nombre décimal en base 2

Exemple : Convertir 99 en base 2.

Puissance de 2 :	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Valeur :	128	64	32	16	8	4	2	1

Calculer la plus grande puissance de 2 plus petit ou égal à 99 i.e. 2^6.

$99 \geq 2^6 = 64$ $\Rightarrow$ écrire 1 et soustraire 64 de 99 ce qui donne 35.
$35 \geq 2^5 = 32$ $\Rightarrow$ écrire 1 et soustraire 32 de 35 ce qui donne 3.
$3 < 2^4 = 16$ $\Rightarrow$ écrire 0.
$3 < 2^3 = 8$ $\Rightarrow$ écrire 0.
$3 < 2^2 = 4$ $\Rightarrow$ écrire 0.
$3 \geq 2^1 = 2$ $\Rightarrow$ écrire 1 et soustraire 2 de 3 ce qui donne 1.
$1 \geq 2^0 = 1$ $\Rightarrow$ écrire 1 et soustraire 1 de 1 ce qui donne 0.

Résultat : 1100011_2.

Identificateur de variable

- Pour identifier une variable, on utilise un identificateur pour désigner le nom de cette variable.
- L'identificateur doit indiquer le rôle joué par cette variable; il faut éviter d'utiliser des noms qui n'évoquent rien. Ex. : Rayon_du_cercle au lieu de x.
- Cela vous épargnera de devoir connaître l'adresse réelle en mémoire de celle-ci.

Règles à respecter pour les noms de variables

- Une séquence de lettres (a → z, A → Z) et de chiffres (0 à 9) qui doit toujours commencer par une lettre. Le symbole _ est considéré comme une lettre.
- Aucune lettre accentuée, cédille, espace, caractère spécial à l'exception du caractère souligné _.
- Les minuscules et les majuscules sont des lettres différentes.

Exemple :

Variable_entiere, *entier1*, *mot_en_francais* sont valides mais *1er_entier*, *nom.2*, *nom de variable*, *deuxième_entier* et *a-b* ne le sont pas.

Éviter d'utiliser le symbole _ comme 1er caractère car il peut être utilisé pour définir des entités spéciales pour Python.

- Les 28 mots réservés ci-dessous ne peuvent être utilisés comme nom de variable :

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

Attention :

- Vous devez saisir les majuscules et les minuscules exactement telles qu'elles apparaissent.

  Main, main et MAIN sont distincts l'un de l'autre.

  Python distingue les majuscules et les minuscules :

  Nom_de_variable est différent de nom_de_variable.

- Bien que la longueur des identificateurs ne soit plus un problème dans les langages de programmation d'aujourd'hui, utilisez des noms de taille raisonnable ayant une signification.

  Il peut exister des limites qui peuvent changer d'un interpréteur à l'autre.

3. Opérateur d'affectation =

Syntaxe : identificateur_de_variable = expression

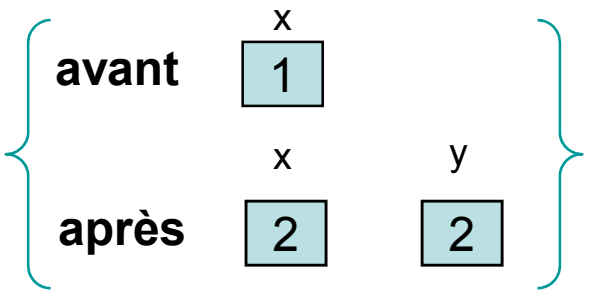
Exemple :

```
>>> entier1 = 8
>>> entier2 = entier1 - 5
>>> entier1 = entier1 + 1
>>> print entier1, entier2
9 3
>>>
```

But : stocker la valeur d'une expression dans une variable.

Note : Les affectations ne sont pas des expressions; elles n'ont pas de valeurs inhérentes mais, il est permis d'enchaîner plusieurs affectations.

```
>>> x = 1
>>> y = x = x + 1
>>> print (x, y)
2 2
>>>
```



On ne peut pas écrire : y = (x = x + 1).

Affectation

```
>>> i = 2
>>> message = "Ceci est un message"
>>> valeur_de_pi = 3.14159
>>>
```

Dans le langage Python, ces instructions d'affectation réalisent les opérations suivantes :

- créer et mémoriser un nom de variable,
- lui attribuer implicitement un type bien déterminé (entier, réel, chaîne de caractères, …)
- lui associer une valeur particulière,
- Établir un lien entre le nom de la variable et l'emplacement mémoire renfermant la valeur associée.

Note : Pour définir le type des variables avant de pouvoir les utiliser, il suffit d'assigner une valeur à un nom de variable pour que celle-ci soit automatiquement créée avec le type qui correspond le mieux à la valeur fournie. Python possède donc un **typage dynamique** et non un typage statique (C++, JAVA).

Affichage de la valeur d'une variable

```
>>> s = "Luc"
>>> entier = 3
>>> reel = 10.2
>>> s
'Luc'
>>> entier, reel
(3, 10.199999999999999)
>>> s = 1
>>> s
1
>>>
```

Affichage de la chaîne de caractères s.

Affichage des variables entier et reel.

Affectation d'une valeur à une variable s. Cela signifie que l'ancienne variable s n'est plus accessible.

Ce mode d'affichage élémentaire est utilisé en mode interactif. Autrement, on opte pour l'instruction print.

```
>>> chaine = "Oh! la! la!"
>>> indice = 5
>>> print (chaine, indice)
Oh! la! la! 5
>>> chaine, indice
('Oh! la! la!', 5)
>>>
```

Les variables sont séparées par des virgules.

Notez les différences entre les modes d'affichage.

Affectations multiples et parallèles

```
>>> centre_x, centre_y, rayon = 1.0, 0.5, 12
>>> print (centre_x, centre_y, rayon)
1.0 0.5 12
>>> centre_x = centre_y = 0
>>> print (centre_x, centre_y, rayon)
0 0 12
>>>
```

→ Affectation parallèle

→ Affectation multiple

Une autre façon de réaliser l'affectation de plusieurs variables à la fois est de placer la liste des variables à gauche de l'opérateur d'affectation et la liste des expressions à droite de l'opérateur d'affectation.

```
>>> x = 5.0
>>> y = 10.0
>>> x, y, z = x + y, x - y, 25
>>> print (x, y, z)
15.0 -5.0 25
>>> u = v = 1.
>>> u, v, w = u + w, v + w, 3.4
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    u, v, w = u + w, v + w, 3.4
NameError: name 'w' is not defined
```

```
>>> x, y = 10, 20
>>> print (x, y)
10 20
>>> x, y = y, x
>>> print (x, y)
20 10
>>>
```

→ Permutation de variables sans utiliser de variable temporaire.

Opérateurs et expressions

```
>>> x, y = 3, 13
>>> x, y = x ** 2, y % x
>>> print (x, y)
9 1
>>> a = 10
>>> a = a + 1
>>> print (a)
11
>>>
```

l'opérateur ** d'exponentiation,
l'opérateur modulo % .

Dans une affectation parallèle avec des expressions, l'évaluation de celles-ci se fait avec la valeur des variables avant l'exécution de l'instruction.

Priorité des opérateurs

- Ordre de priorité : les parenthèses, **, * et /, + et -.
- Si 2 opérateurs ont même priorité, l'évaluation est effectuée de gauche à droite.

```
>>> x = 5
>>> y = 3
>>> print (x - 1 + y ** 2 * 3 / 6)
8
>>>
```

Opérateurs d'affectation +=, -=, *=, /=, %=, **=, //=

Syntaxe : identificateur_de_variable op expression

But : L'évaluation d'une expression et une affectation sont combinées.

Exemple :

```
>>> x = 5
>>> x **= 2          # Équivaut à x = x ** 2.
>>> x
25
>>> x %= 3
>>> x
1
>>> x //= 2
>>> x
0
>>>
```

Note : Contrairement à C++, Python ne renferme pas les opérateurs ++ et --

4. Saisie de données au clavier

La fonction input

- Elle provoque une interruption dans le programme courant où l'utilisateur est invité à entrer des données au clavier et à terminer avec <Enter>. L'exécution du programme se poursuit alors et la fonction fournit en retour les valeurs entrées par l'utilisateur.

- Ces valeurs peuvent alors être stockées dans des variables dont le type correspond à celui des données entrées.

```
>>> print ("Entrez un entier positif :")
>>> n = input()
>>> print ("Deux puissance ", n, " donne comme résultat : "), 2**n
Deux puissance  5  donne comme résultat :  32
```

- La fonction input est soit, sans paramètre ou soit, avec un seul paramètre, une chaîne de caractères, lequel est un message explicatif destiné à l'utilisateur.

```
>>> nom = input("Entrez votre nom (entre guillemets) :")
>>> print (nom)
```

La chaîne de caractères est entrée entre des apostrophes ou des guillemets.

- On peut saisir plusieurs données simultanément.

```
>>> t, u, v, w = input()
"oui", 34, "non", 59
>>> print (t, u, v, w)
oui 34 non 59
```

- On doit fournir exactement le nombre de données voulues à la saisie.

Exemple : Programme qui saisit 2 entiers et affiche leur quotient.

```
>>> #     Ce programme saisit deux valeurs entières au clavier,
>>> #     calcule le quotient et
>>> #     affiche le résultat.
>>>
>>> m, n = input("Entrez 2 valeurs entières au clavier : ")
Entrez 2 valeurs entières au clavier : 34, 6
>>> resultat = m / n
>>> print  ("Le quotient de ", m, " par ", n, " est : ", resultat)
Le quotient de  34  par  6  est :  5
>>>
```

5. Erreur d'exécution ou de logique

En exécutant ce programme, si vous entrez au clavier comme 2ième valeur entière la valeur nulle, le programme terminera anormalement.

Un message sera affiché indiquant que l'on a tenté d'effectuer une division par zéro.

C'est une erreur d'exécution ou de logique.

Exemple :

```
>>> m, n = input("Entrez 2 valeurs entières au clavier :")
Entrez 2 valeurs entières au clavier :12, 0
>>> resultat = m / n

Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    resultat = m / n
ZeroDivisionError: integer division or modulo by zero
>>>
```

Exemple :

```
>>> Valeur = 3.14159 * rayon ** 2

Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    Valeur = 3.14159 * rayon ** 2
NameError: name 'rayon' is not defined
>>>
```

Erreurs à l'interprétation

- Lorsque vous écrivez une commande ou une expression en Python avec une erreur syntaxique, l'interpréteur affiche un message d'erreur et rien n'est exécuté. Il faut recommencer.

Exemple :

```
>>> Montant_en_$ = 35.56
SyntaxError: invalid syntax
>>>
```

6. Types de données élémentaires

Type « int »

- Les entiers ordinaires de Python correspondent aux entiers standards.
- La plupart des machines (32 bits) sur lesquelles s'exécute Python permettent de coder tous les entiers entre -2^{31} et $2^{31} - 1$, i.e. -2 147 483 648 et 2 147 483 647.
- Normalement, les entiers sont représentés en format décimal en base 10, mais on peut aussi les spécifier à l'aide de leur représentation en base 8 ou 16.
  Les valeurs octales utilisent le préfixe "0", tandis que les valeurs hexadécimales sont introduites par un préfixe "0x" ou "0X" .

```
>>> x, y, z = 123, -123, 2147483650
>>> print (x, y, z)
123 -123 2147483650
>>> print (x, y, type(z), z)
123 -123 <type 'long'> 2147483650
>>> w = 0x709A
>>> x = -0XEF2
>>> print (w, x)
28826 -3826
```

Aucun débordement de capacité.

La fonction permet de vérifier à chaque itération le type de la variable z.

Types de données élémentaires

Type « long »

Python est capable de traiter des nombres entiers aussi grands que l'on veut. Toutefois, lorsque ceux-ci deviennent très grands, les variables définies implicitement comme étant de type **int** (32 bits) sont maintenant de type **long**.

Ex. :

```
u, v, w = 1, 1, 1
while (w <= 50) :
    if (w >= 40) :
        print (w, " : ", v, type(v))
    u, v, w = v, u + v, w + 1
```

```
40  :  165580141 <type 'int'>
41  :  267914296 <type 'int'>
42  :  433494437 <type 'int'>
43  :  701408733 <type 'int'>
44  :  1134903170 <type 'int'>
45  :  1836311903 <type 'int'>
46  :  2971215073 <type 'long'>
47  :  4807526976 <type 'long'>
48  :  7778742049 <type 'long'>
49  :  12586269025 <type 'long'>
50  :  20365011074 <type 'long'>
```

⇨ Il n'y a pas de débordement de capacité.

Types de données élémentaires

Ex. :

```
>>> r = 10L
>>> s = 12345678901234567890
>>> t = -0XABCDEF0123456789ABCDEF
>>> print (r, s, t)
10 12345678901234567890 -207698809136909011942886895
>>>
```

Les entiers longs sont identifiés par la lettre « L » ou « l », ajoutée à la fin de la valeur numérique. Pour éviter toute confusion, il est préférable d'utiliser L.

```
>>> A= 0XABCDEF01234567890AL
>>> print (A)
3169232317152542296330
>>> A
3169232317152542296330L
>>>
```

```
>>> x = 12345678901234567890
>>> x = x + 1
>>> print (x)
12345678901234567891
>>>
```

Types de données élémentaires

Type « bool »

Les valeurs booléennes True ou False constituent un cas particulier des entiers. Dans un contexte numérique tel qu'une addition avec d'autres nombres, True est traité comme un entier valant 1, et False a la valeur 0.

Type « complex »

Cela représente des nombres complexes de la forme a + b j où a et b sont des réels en virgule flottante, a représente la partie réelle, b la partie imaginaire et $j^2 = -1$.

Ex. :

```
6.25 + 2.3j
-4.37 + 13J
0 + 1j
1 + 0j
```

```
>>> x = 1.2 + 4.5j
>>> print (x)
(1.2+4.5j)
>>> y = 1.2 - 4.5j
>>> print (x * y)
(21.69+0j)
>>> print (x – y)
9j
```

```
>>> Nombre_complexe = 1.57 - 7.9j
>>> Nombre_complexe.real          # partie réelle
1.5700000000000001
>>> Nombre_complexe.imag          # partie imaginaire
-7.900000000000004
>>> Nombre_complexe.conjugate()   # conjugué
(1.5700000000000001+7.900000000000004j)
>>> Nombre_complexe * Nombre_complexe.conjugate()
(64.87489999999997+0j)
>>>
```

Type « float »

Les données ayant un point décimal ou un exposant de 10.

Ex. : 3.14159 -12. .13 2e13 0.3e-11

Cela permet de manipuler des nombres positifs ou négatifs compris entre 10^{-308} et 10^{308} avec une précision de 12 chiffres significatifs.

```
u, v = 1., 1
while (v <= 40) :
    print (v, " : ", u ** (u * u))
    u, v = u + 1, v + 1
```

```
1  :  1.0
2  :  16.0
3  :  19683.0
4  :  4294967296.0
5  :  2.98023223877e+017
6  :  1.03144247985e+028
7  :  2.56923577521e+041
8  :  6.27710173539e+057
9  :  1.96627050476e+077
10  :  1e+100
11  :  1.019799757e+126
12  :  2.52405858453e+155
13  :  1.80478943437e+188
14  :  4.37617814536e+224
15  :  4.17381588439e+264
16  :

Traceback (most recent call last):
  File "E:\essai.py", line 3, in <module>
    print v, " : ", u ** (u * u)
OverflowError: (34, 'Result too large')

>>>
```

En principe, 52 bits sont alloués à la mantisse, 11 à l'exposant et un bit pour le signe. En pratique, cela peut dépendre de la machine utilisée et de l'environnement de programmation.

Conversion de types numériques

- Jusqu'à maintenant, nous avons appliqué les opérateurs précédents à des opérandes de même type. Qu'arrive-t-il lorsque les opérandes sont de types différents ?

```
>>> 3 + 5.4
8.400000000000004
>>>
```

- Il s'agit de convertir l'un des opérandes au type de l'autre opérande avant d'effectuer l'opération.

- Toutes les conversions ne sont pas possibles comme celle d'un réel en entier, ou celle d'un nombre complexe en n'importe quel autre type non complexe.

Règles de conversion :

- Si l'un des arguments est un nombre complexe, l'autre est converti en complexe.
- Sinon, si l'un des arguments est un nombre réel, l'autre est converti en réel.
- Sinon, si l'un des arguments est un long, l'autre est converti en long.
- Sinon, tous deux doivent être des entiers ordinaires et aucune conversion n'est nécessaire.

Fonctions de types numériques

cmp()

Prend en entrée deux expressions a et b de valeurs numériques et retourne

-1	si a < b,
0	si a est égale à b,
+1	si a > b.

```
>>> u = 3.14
>>> v = 2.6
>>> cmp(u**2 - 2, v*3 + 4)
-1
>>> cmp(3, 3)
0
>>> cmp(1, (1. / 3) * 3.)
0
>>> cmp(u, 3)
1
>>>
```

type()

Retourne le type de l'argument.

```
>>> u = 1 + 3.4j
>>> type(u)
<type 'complex'>
>>> type(3 + 4.4)
<type 'float'>
>>>
```

Fonctions de types numériques

bool()

Retourne True si l'argument est différent de 0. False autrement.

```
>>> bool(3.14)
True
>>> bool(0)
False
>>> bool(-3.14)
True
```

int()

Prend en entrée comme argument une expression de valeur numérique ou une chaîne de caractères représentant un entier et retourne le résultat de l'expression où la partie fractionnaire a été omise ou la chaîne de caractères convertie en entier.

int() supprime le point décimal et toutes les décimales qui suivent (le nombre est tronqué).

```
>>> int(3.14)
3
>>> int(-3.14)
-3
>>> int("3")
3
```

Fonctions de types numériques

long()

Prend en entrée comme argument une expression de valeur numérique ou une chaîne de caractères représentant un entier et retourne le résultat de l'expression sous forme d'entier long (partie fractionnaire omise) ou la chaîne de caractères convertie en entier long.

```
>>> long(3.14)
3L
>>> long(0xabc)
2748L
>>> long("3L")
3L
>>> long("3")
3L
```

float()

Prend en entrée comme argument une expression de valeur numérique ou une chaîne de caractères représentant un nombre et retourne le résultat de l'expression sous forme de réel ou la chaîne de caractères convertie en réel.

```
>>> float(3)
3.0
>>> float("3")
3.0
>>> float("3.4")
3.3999999999999999
```

complex()

```
>>> complex(3.2, 7)
(3.2000000000000002+7j)
>>> complex("3.2+7j")
(3.2000000000000002+7j)
>>> complex(3.4)
(3.3999999999999999+0j)
```

Fonctions de types numériques

abs()

Retourne la valeur absolue de l'argument.

```
>>> abs(-1 + 0.25)
0.75
>>> abs(3 -2j)
3.605551275463989
>>> (3 - 2j)*(3 + 2j)
13.0
>>> abs(3 -2j) ** 2
13.000000000000002
```

pow(m, n) ou pow(m, n, p)

Dans les 2 cas, m^n est d'abord calculé; puis, si le troisième argument est fourni,
alors (m ** n) % p est retourné;
sinon, m ** n est retourné.

```
>>> pow(4, 2)
16
>>> pow(3, 2, 5)
4
```

round()

Arrondit un nombre réel à l'entier le plus proche et retourne le résultat comme une valeur réelle. Un 2ième paramètre présent arrondit l'argument au nombre de décimales indiqué.

```
>>> round(3.4999999)
3.0
>>> round(2.8)
3.0
>>> round(253.358901234, 2)
253.36000000000001
>>> round(-3.4), round(-3.5)
(-3.0, -4.0)
```

Fonctions ne s'appliquant qu'à des entiers ordinaires ou longs

Représentation dans une base

- Nous savons que Python gère automatiquement des représentations octales et hexadécimales, en plus de la représentation décimale.

- Python dispose aussi de deux fonctions intégrées, oct() et hex(), qui retournent des chaînes de caractères contenant respectivement la représentation octale ou hexadécimale d'une expression entière quelle qu'en soit la représentation.

```
>>> print (type(hex(255)), hex(255))
<type 'str'> 0xff
>>> hex(1234567890123456789OL)
'0xab54a98ceb1f0ad2L'
>>> oct(8**4)
'010000'
>>>
```

Fonctions ne s'appliquant qu'à des entiers ordinaires ou longs

Conversion ASCII

- Chaque caractère est associé à un nombre unique entre 0 et 255, son indice dans la table ASCII (« American Standard Code for Information Interchange »).

- La table ASCII est la même sur tous les ordinateurs ce qui garantit un comportement identique des programmes sur différents environnements.

Code décimal	Code hex	Caractère	Code décimal	Code hex	Caractère	Code décimal	Code hex	Caractère	Code décimal	Code hex	Caractère
0	00		16	10		32	20	Espace	48	30	0
1	01		17	11		33	21	!	49	31	1
2	02		18	12		34	22	"	50	32	2
3	03		19	13		35	23	#	51	33	3
4	04		20	14		36	24	$	52	34	4
5	05		21	15		37	25	%	53	35	5
6	06		22	16		38	26	&	54	36	6
7	07	\a	23	17		39	27	'	55	37	7
8	08	\b	24	18		40	28	(	56	38	8
9	09	\t	25	19		41	29	)	57	39	9
10	0A	\n	26	1A		42	2A	*	58	3A	:
11	0B	\v	27	1B		43	2B	+	59	3B	;
12	0C	\f	28	1C		44	2C	,	60	3C	<
13	0D	\r	29	1D		45	2D	-	61	3D	=
14	0E		30	1E		46	2E	.	62	3E	>
15	0F		31	1F		47	2F	/	63	3F	?

Fonctions ne s'appliquant qu'à des entiers ordinaires ou longs

- On retrouve ici les 128 premiers caractères qui renferment notamment les majuscules, les minuscules, les chiffres et les signes de ponctuation.
  Des codes étendus sont disponibles.

Code décimal	Code hex	Caractère	Code décimal	Code hex	Caractère	Code décimal	Code hex	Caractère	Code décimal	Code hex	Caractère
64	40	@	80	50	P	96	60	‘	112	70	p
65	41	A	81	51	Q	97	61	a	113	71	q
66	42	B	82	52	R	98	62	b	114	72	r
67	43	C	83	53	S	99	63	c	115	73	s
68	44	D	84	54	T	100	64	d	116	74	t
69	45	E	85	55	U	101	65	e	117	75	u
70	46	F	86	56	V	102	66	f	118	76	v
71	47	G	87	57	W	103	67	g	119	77	w
72	48	H	88	58	X	104	68	h	120	78	x
73	49	I	89	59	Y	105	69	i	121	79	y
74	4A	J	90	5A	Z	106	6A	j	122	7A	z
75	4B	K	91	5B	[	107	6B	k	123	7B	{
76	4C	L	92	5C	\	108	6C	l	124	7C	|
77	4D	M	93	5D	]	109	6D	m	125	7D	}
78	4E	N	94	5E	^	110	6E	n	126	7E	~
79	4F	O	95	5F	_	111	6F	o	127	7F	

Fonctions ne s'appliquant qu'à des entiers ordinaires ou longs

chr() Prend comme argument une expression entière entre 0 et 255 inclusivement et retourne le caractère ASCII sous la forme d'une chaîne de caractères.

ord() Prend comme argument un caractère ASCII sous la forme d'une chaîne de caractères de longueur 1 et retourne le code ASCII correspondant.

```
>>> print ("Le caractère 5 en code ASCII est : ", ord("5"))
Le caractère 5 en code ASCII est :  53
>>> print ("Le code ASCII 36 désigne le caractère : ", chr(36))
Le code ASCII 36 désigne le caractère :  $
>>>
```

Nous verrons plus loin des modules renfermant d'autres fonctions manipulant des expressions numériques.

Type « string »

Une chaîne de caractères délimitée par des apostrophes ou des guillemets.

```
mot1 = "C'est une grosse journée;"
mot2 = 'vous pouvez me croire, la journée est "pesante".'
print (mot1, mot2)
```

C'est une grosse journée; vous pouvez me croire, la journée est "pesante".

L'instruction print insère un espace entre les éléments affichés.

Note : Le caractère spécial « \ » permet d'écrire une commande sur plusieurs lignes.

Il permet d'insérer un certain nombre de caractères spéciaux (saut de ligne, apostrophes, guillemets) à l'intérieur d'une chaîne de caractères.

```
mot = 'C\'est le jour de Pâques.\nBonne \
fin de semaine.'
print mot
```

C'est le jour de Pâques.
Bonne fin de semaine.

\n	saut de ligne
\'	permet d'insérer une apostrophe dans une chaîne délimitée par des apostrophes.

Accès aux caractères d'une chaîne

```
mot = "apprendre"            # Le premier caractère est en position 0.
print (mot[3], mot[4], mot[5], mot[6])
```

r e n d

Concaténation de chaînes à l'aide de l'opérateur +

```
mot = "apprend"
mot = mot + "re"
print (mot[3] + mot[4] + mot[5] + mot[6])
```

rend

Répétition de chaînes à l'aide de l'opérateur *

```
mot = "cher" * 2
print (mot)
```

chercher

Longueur d'une chaîne

```
print (len(mot))
```

9

Convertir une chaîne qui représente un nombre en un nombre véritable

```
m = "12.3"
n = '13'
print (float(m) + int(n))
```

25.3

On peut utiliser des apostrophes triples pour protéger des caractères spéciaux.

```
>>> Texte = "Python"
>>> Texte = Texte + ' est un langage '
>>> Texte += '"renfermant les guillemets (").'"
>>> print (Texte)
Python est un langage renfermant les guillemets (").
>>>
```

Convertir un nombre en une chaîne de caractères

- **str()** Convertir un nombre en chaîne de caractères.

```
>>> print ("On peut concaténer une chaîne et un nombre converti : " + str(1.14 + 3))
On peut concaténer une chaîne et un nombre converti : 4.14
>>>
```

Étude complète plus loin

7. Gestion de la mémoire

- En Python, il n'existe pas de déclaration explicite de variables lesquelles sont implicitement déclarées lors de leur première utilisation.

- Cependant, il n'est pas possible d'accéder à une variable avant qu'elle n'ait été créée et initialisée :

```
>>> a

Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    a
NameError: name 'a' is not defined
>>>
```

- En Python, il n'y a pas non plus de spécification explicite de type. Cela se fait implicitement à la première utilisation.

```
X = 3.4
```
→ X est alors une variable de type réel renfermant la valeur 3.4.

- La libération de l'espace mémoire d'une variable est sous la responsabilité de l'interpréteur. Lorsqu'il n'y a plus de références à un espace mémoire, le « ramasse-miettes » se charge de libérer cet espace mémoire.

```
>>> x = 3.4
>>> x = "La variable réelle est perdue."
>>> print (x)
La variable réelle est perdue.
```

- En temps normal, vous ne « supprimez » pas vraiment un nombre : vous cessez simplement de l'utiliser! Si vous souhaitez supprimer une référence, utilisez l'instruction del. Après quoi, on ne peut plus utiliser le nom de variable à moins de l'affecter à une nouvelle valeur.

```
>>> s = 1
>>> print (s)
1
>>> del s
>>> print (s)

Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    print s
NameError: name 's' is not defined
```

```
>>> s = "Ceci est un test."
>>> s
'Ceci est un test.'
>>>
```

8. Structures de contrôle

En Python, il faut comprendre que la syntaxe repose principalement sur l'indentation. Ainsi, il convient de rappeler la non utilisation d'un certains nombres de habituels notamment en C, Java, … De ce fait Python s'avère être un langage informatique performant et puissant avec moins de ponctuation.

- **Syntaxe des structures alternatives (AND, OR opérateurs logiques)**

Cas 1

```
if condition :
        instruction(s)1
else :
        instruction(s)2
```

Cas 2

```
if condition :
        instruction(s)1
elif :
        instruction(s)2
else :
        instruction(s)3
```

Cas 3

```
test = (valeur_si_vrai , valeur_si_faux) [ condition ]
```

Exercices :

1. Écrire un programme qui affiche la mention selon la moyenne au BAC
2. Écrire un programme qui vérifie si une personne sera majeure en 2024. Le programme devra renseigner l'année où ladite personne sera majeure si toutefois qu'elle se révèle mineure.

- **Syntaxe des structures itératives (break** et **continue** sont utilisables**)**

Cas 1

```
while condition :
        instruction(s)

#suite du programme
```

Cas 2

```
for compteur in range (indiceF, indiceL) :
        instruction(s)1

#suite du programe après la bouble
```

Cas 3

```
for element in liste_elements :
        instruction(s)

#suite du programme à la sortie de la bouble
```

Remarque : ***Une liste est un ensemble d'éléments qui peuvent être de types différents et désigné par une identificateur unique. Ce sont des éléments indicés*** *(position commençant par 0).*

EX : ***days = [" lundi " , " mardi ", " mercredi ", " jeudi ", " vendredi ", " samedi " , " dimanche "]***

Exercices :

1. Écrire un programme qui calcule et affiche le nombre de chiffres présents dans un nombre saisi au clavier
2. Écrire un programme qui permet de calculer et d'afficher, si possible le factoriel d'un nombre entier naturel saisie au clavier.

9. Les listes et opérations

Opération	Interprétation
L1=[]	liste vide
L2=[0, 1, 2, 3]	4 élément indicé de 0 à 3
L3=['abc', ['def', 'ghi']]	liste incluses
L2[i], L3[i][j]	indice
L2[i:j]	tranche
len(L2)	longueur
L1+L2	concaténation
L1*3	répétition
for x in L2	parcours
3 in L2	appartenance
L2.append(4)	méthodes : agrandissement
L2.sort()	tri
L2.index()	recherche
L2.reverse()	inversion
del L2[k], L2[i:j]=[]	effacement
L2[i]=1	affectation par indice
L2[i:j]=[4, 5, 6]	affectation par tranche
range(4), xrange(0,4)	création de listes / tuples d'entiers

Les listes (ou list / array) en python sont une variable dans laquelle on peut mettre plusieurs variables.

Créer une liste en python

Pour créer une liste , rien de plus simple:

```
>>> liste = []
```

Vous pouvez voir le contenu de la liste en l'appelant comme ceci:

```
>>> liste
<type 'list'>
```

Ajouter une valeur à une liste python

Vous pouvez ajouter les valeurs que vous voulez lors de la création de la liste python :

```
>>> liste = [1,2,3]
>>> liste
[1, 2, 3]
```

Ou les ajouter après la création de la liste avec la méthode `append` (qui signifie "ajouter" en anglais):

```
>>> liste = []
>>> liste
[]
>>> liste.append(1)
>>> liste
[1]
>>> liste.append("ok")
>>> liste
[1, 'ok']
```

On voit qu'il est possible de mélanger dans une même liste des variables de type différent. On peut d'ailleurs mettre une liste dans une liste.

Afficher un item d'une liste

Pour lire une liste, on peut demander à voir l'index de la valeur qui nous intéresse:

```
>>> liste = ["a","d","m"]
>>> liste[0]
'a'
>>> liste[2]
'm'
```

Le premier item commence toujours avec l'index 0. Pour lire la premier item on utilise la valeur 0, le deuxième on utilise la valeur 1, etc.

Il est d'ailleurs possible de modifier une valeur avec son index

```
>>> liste = ["a","d","m"]
>>> liste[0]
'a'
>>> liste[2]
'm'
>>> liste[2] = "z"
>>> liste
['a', 'd', 'z']
```

Supprimer une entrée avec un index

Il est parfois nécessaire de supprimer une entrée de la liste. Pour cela vous pouvez utiliser la fonction `del` .

```
>>> liste = ["a", "b", "c"]
>>> del liste[1]
>>> liste
['a', 'c']
```

Supprimer une entrée avec sa valeur

Il est possible de supprimer une entrée d'une liste avec sa valeur avec la méthode `remove` .

```
>>> liste = ["a", "b", "c"]
>>> liste.remove("a")
>>> liste
['b', 'c']
```

Inverser les valeurs d'une liste

Vous pouvez inverser les items d'une liste avec la méthode `reverse` .

```
>>> liste = ["a", "b", "c"]
>>> liste.reverse()
>>> liste
['c', 'b', 'a']
```

Compter le nombre d'items d'une liste

Il est possible de compter le nombre d'items d'une liste avec la fonction `len` .

```
>>> liste = [1,2,3,5,10]
>>> len(liste)
5
```

Compter le nombre d'occurences d'une valeur

Pour connaitre le nombre d'occurences d'une valeur dans une liste, vous pouvez utiliser la méthode `count` .

```
>>> liste = ["a","a","a","b","c","c"]
>>> liste.count("a")
3
>>> liste.count("c")
2
```

Trouver l'index d'une valeur

La méthode `index` vous permet de connaitre la position de l'item cherché.

```
>>> liste = ["a","a","a","b","c","c"]
>>> liste.index("b")
3
```

Manipuler une liste

Voici quelques astuces pour manipuler des listes:

```
>>> liste = [1, 10, 100, 250, 500]
>>> liste[0]
1
>>> liste[-1] # Cherche la dernière occurence
500
>>> liste[-4:] # Affiche les 4 dernières occurrences
[500, 250, 100, 10]
>>> liste[:] # Affiche toutes les occurences
[1, 10, 100, 250, 500]
>>> liste[2:4] = [69, 70]
[1, 10, 69, 70, 500]
>>> liste[:] = [] # vide la liste
[]
```

Boucler sur une liste

Pour afficher les valeurs d'une liste, on peut utiliser une boucle:

```
>>> liste = ["a","d","m"]
>>> for lettre in liste:
...     print lettre
...
a
d
m
```

Si vous voulez en plus récupérer l'index, vous pouvez utiliser la fonction `enumerate` .

```
>>> for lettre in enumerate(liste):
...     print lettre
...
(0, 'a')
(1, 'd')
(2, 'm')
```

Les valeurs retournées par la boucle sont des tuples.

Copier une liste

Beaucoup de débutants font l'erreur de copier une liste de cette manière

```
>>> x = [1,2,3]
>>> y = x
```

Or si vous changez une valeur de la liste `y`, la liste `x` sera elle aussi affectée par cette modification:

```
>>> x = [1,2,3]
>>> y = x
>>> y[0] = 4
>>> x
[4, 2, 3]
```

En fait cette syntaxe permet de travailler sur un même élément nommé différemment

Alors comment copier une liste qui sera indépendante?

```
>>> x = [1,2,3]
>>> y = x[:]
>>> y[0] = 9
>>> x
[1, 2, 3]
>>> y
[9, 2, 3]
```

Pour des données plus complexes, vous pouvez utiliser la fonction `deepcopy` du module `copy`

```
>>> import copy
>>> x = [[1,2], 2]
>>> y = copy.deepcopy(x)
>>> y[1] = [1,2,3]
>>> x
[[1, 2], 2]
>>> y
[[1, 2], [1, 2, 3]]
```

Transformer une string en liste

Parfois il peut être utile de transformer une chaine de caractère en liste. Cela est possible avec la méthode `split` .

```
>>> ma_chaine = "Olivier:ENGEL:Strasbourg"
>>> ma_chaine.split(":")
['Olivier', 'ENGEL', 'Strasbourg']
```

Transformer une liste en string

L'inverse est possible avec la méthode " join ".

```
>>> liste = ["Olivier","ENGEL","Strasbourg"]
>>> ":".join(liste)
'Olivier:ENGEL:Strasbourg'
```

Trouver un item dans une liste

Pour savoir si un élément est dans une liste, vous pouvez utiliser le mot clé in de cette manière:

```
>>> liste = [1,2,3,5,10]
>>> 3 in liste
True
>>> 11 in liste
False
```

La fonction range

La fonction `range` génère une liste composée d'une simple suite arithmétique.

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Agrandir une liste par une liste

Pour mettre bout à bout deux listes, vous pouvez utiliser la méthode `extend`

```
>>> x = [1, 2, 3, 4]
>>> y = [4, 5, 1, 0]
>>> x.extend(y)
>>> print x
[1, 2, 3, 4, 4, 5, 1, 0]
```

Permutations

La permutation d'un ensemble d'éléments est une liste de tous les cas possibles. Si vous avez besoin de cette fonctionnalité, inutile de réinventer la roue, `itertools` s'en occupe pour vous.

```
>>> from itertools import permutations
>>> list(permutations(['a', 'b', 'c']))
[('a', 'b', 'c'), ('a', 'c', 'b'), ('b', 'a', 'c'), ('b', 'c', 'a'), ('c', 'a', 'b'), ('c', 'b', 'a')]
```

Permutation d'une liste de liste

Comment afficher tous les cas possibles d'une liste elle-même composée de liste? Avec l'outil `product` de `itertools` :

```
>>> from itertools import product
>>> list(product(['a', 'b'], ['c', 'd']))
[('a', 'c'), ('a', 'd'), ('b', 'c'), ('b', 'd')]
```

Bonus sur les listes…

Vous pouvez additionner deux listes pour les combiner ensemble en utilisant l'opérateur `+` :

```
>>> x = [1, 2, 3]
>>> y = [4, 5, 6]
>>> x + y
[1, 2, 3, 4, 5, 6]
```

Vous pouvez même mutiplier une liste:

```
>>> x = [1, 2]
>>> x*5
[1, 2, 1, 2, 1, 2, 1, 2, 1, 2]
```

Ce qui peut être utile pour initialiser une liste:

```
>>> [0] * 5
[0, 0, 0, 0, 0]
```

Liste de listes…

Pour construire un tableau à plusieurs dimensions, il existe plusieurs méthodes sous Python.
On peut construire un tableau – une matrice, si le tableau est à 2 dimensions - comme une liste de listes :

Exemple : pour construire $a = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$

```
>>> a=[[1, 2] ,[3, 4] ,[5, 6]]
>>> a
[[1, 2], [3, 4], [5, 6]]
>>> len(a)
3
>>> a[1]
[3, 4]
>>> a[1][0]
3
```

Un tableau de dimension 2 (liste de listes) contiendra des éléments qui devront être appelé par la double indexation `[i][j]` :

- Le premier index renvoie donc à l'index de la ligne.
- Le deuxième index renvoie donc à l'index de la colonne.

La commande *len()* renvoie la longueur de la liste.

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

Encryptage et décryptage d'un message

CRYPTAGE : Action de rendre illisible des informations pour quiconque qui ne possède pas la clé de décryptage.

DECRYPTAGE : Action de rendre lisible des données cryptées grâce à la clé de cryptage.

Luc veut transmettre un message à Pierre sans que personne d'autres n'y ait accès.
Luc veut transmettre à Pierre un caractère dont le code ASCII est s; pour y arriver, il lui transmet plutôt 2 caractères correspondants aux codes entiers d et e suivants :

d = (a * s + b) / 256 (division entière)
e = (a * s + b) % 256 (opération modulo)

où a et b sont 2 constantes entières non nulles plus petites que 256.

Les valeurs a et b sont connues de Luc et de Pierre. Fixons-les à a = 251, b = 247.

Nous allons d'abord aider Luc et lui construire un programme Python qui détermine les deux caractères obtenus après l'opération de cryptage.

Nous allons ensuite aider Pierre et lui construire un programme Python qui détermine le caractère obtenu après l'opération de décryptage.

Programme de cryptage

```
>>> #     Il s'agit de saisir au clavier le caractère à encrypter,
>>> #     de déterminer les deux caractères obtenus après l'opération de cryptage,
>>> #     et d'afficher ces deux caractères.
>>> #
>>> a = 251                    # constante de cryptage.
>>> b = 247                    # constante de cryptage.
>>> #
>>> #     Lecture du caractère que Luc veut transmettre à Pierre.
>>> #
>>> c = input("Caractère à transmettre : ")
Caractère à transmettre : 'p'
>>> #
>>> #     Déterminer les codes ASCII des 2 caractères obtenus après cryptage.
>>> #
>>> d = (a * ord(c) + b) / 256
>>> e = (a * ord(c) + b) % 256
>>> #
>>> #     Affichage des 2 caractères obtenus après l'opération de cryptage.
>>> #
>>> print chr(d)+chr(e)
nÇ
>>>
```

Programme de décryptage

```
>>> #     Il s'agit de saisir au clavier les 2 caractères cryptés que Pierre a reçus,
>>> #     de déterminer le code ASCII d et e de ces 2 caractères,
>>> #     calculer 256 * d + e dont la valeur correspond à a * s + b,
>>> #     calculer s le code ASCII du caractère que Luc a transmis à Pierre,
>>> #     et afficher le caractère correspondant au code s.
>>> #
>>> a = 251                    # constante de cryptage.
>>> b = 247                    # constante de cryptage.
>>> #
>>> #     Lecture des 2 caractères cryptés que Pierre a reçus.
>>> #
>>> c1, c2 = input("Caractères cryptés : ")
Caractères cryptés : 'n', 'Ç'
>>> #
>>> #     Déterminer le code ASCII d et e de ces 2 caractères.
>>> #
>>> d = ord(c1)
>>> e = ord(c2)
```

Programme de décryptage

```
>>> #
>>> #     calculer 256 * d + e dont la valeur correspond à a * s + b,
>>> #
>>> r = 256 * d + e
>>> #
>>> #     calculer s le code ASCII du caractère que Luc a transmis à Pierre,
>>> #
>>> s = (r - b)/ a
>>> #
>>> #     Afficher le caractère correspondant au code s.
>>> #
>>> print (chr(s))
p
```

9. Fonctions (1/2)

- Une fonction est un bloc d'instructions qui accomplit un rôle bien précis.

  C'est comme un bouton sur une commande de télévision ou une mallette de jeu

- Les fonctions permettent une meilleure lisibilité du code, une facilité dans la maintenance (correction des erreurs) et des mises-à-jour, une réutilisabilité.
- Une fonction a un nom, retourne ou pas une valeur et dispose ou non d'arguments (paramètres). Lors de son appel, une fonction recevra autant de paramètres effectifs compatibles (en nombre, type et ordre de passage) aux paramètres formels qu'elle aura pris lors de sa définition (déclaration).

- Dans son bloc d'instruction, une fonction peut bel et bien manipuler une variable de portée globale. (**Ex : global variable**)

SYNTAXE : `def nom_de_la_fonction (arguments) :`

bloc d'instructions de la fonction

- **Attention : Un saut de deux lignes est exigible à la fin de l'édition de chaque fonction, soit pour l'appeler ou pour en définir une autre.**

9. Fonctions (2/2)

- Exemple d'une fonction de calcul

```
1        def addition (a, b) :
2
3                resultat = a**b + b//a
4                return resultat
5
6
7        print (addition(-5, 2))
```

?

- Exemple de fonction récursive avec une variable globale

```
1   def addition (a) :
2       global b
2       b = a + b
3       print (b)
4       a = 1 - a
5       if a < 0 :
6          addition (a)
7
8
9   b = 1989
10  addition(5)
11  print(b)
```

?

10. Objets (1/3)

- **Un objet (moule)** est une entité identifiable du monde réel, qu'elle soit concrète ou abstraite

  *De ce fait, un objet répond à un certain nombre de caractéristiques (**propriétés**) appelées **attributs** et qui permettent de l'identifier avec des **valeurs (instanciation)**.*
  A travers ces attributs, des opérations sont possibles sur/avec l'objet.
  *Et ces traitements sur les données sont appelés **méthodes ou encore fonctions**.*

- La structure d'un ensemble représentatif d'objets homogènes, c'est-à-dire, identiques dans la forme et dans le comportement s'appelle **classe** (famille d'objets).

  Exemple : Une voiture a une marque, une puissance, un nombre de places, une couleur . Elle roule, s'arrête, tombe en panne, …
  On pourra ainsi définir la classe voiture par ces attributs et méthodes à travers le mot clé class.

SYNTAXE :
```
class Nom_de_la_classe :

        def __init__ (self, argument1, argument2, …) :
                bloc d'instructions
```

- **ATTENTION : On construit les objets à travers les attributs qui les caractérisent dans leur classe. Ainsi, la fonction `def __init__(self)` prenant les paramètres à injecter (affecter) est appelée `constructeur`.**

  Dans le cadre de l'édition d'une classe, `self` est comme une sorte de base de données qui contient les attributs d'objets de classe, directement manipulables par `self.attribut` et pas besoin de déclaration. Tout objet pourra être instancié par le nom de sa classe et manipulé par les méthodes prévues.

10. Objets (2/3)

- **Exemple :** Classe ***Voiture*** avec constructeur, getters et setters

```
class Voiture :

        def __init__(self, marque, puissance, place, couleur) :
                self.marque = marque
                self.puissance = puissance
                self.place = place
                self.couleur = couleur

        def get_puissance (self) :
                return self.puissance

        def set_couleur (self, couleur) :
                self.couleur = couleur


vehicule = Voiture ("Peugeot301","10 CV",5,"Noire") #instanciation
print(vehicule.get_puissance())
vehicule.set_couleur ("Bleue")
print(("Votre véhicule est maintenant peint en", vehicule.get_puissance())

#L'encapsulation est implicitement gérée par Python
#Elle protège naturellement les données (attributs) d'une classe
#Les accesseurs et mutateurs favorisent ainsi la manipulation des attributs,
#notamment dans d'autres classes.
```

10. Objets (3/3)

- **Héritage :** En POO, l'héritage permet d'établir un lien de parenté et le transfert de propriétés entre des différentes classes, d'où le **polymorphisme**, faculté d'une méthode à être appliquée à des objets de classes différentes. En python, l'héritage multiple existe tout comme l'héritage simple. Et un seul fichier peur comporter plusieurs classes liées même par l'héritage.

```
class Etudiant :

        def __init__(self, prenom, nom) :
                self.prenom = prenom
                self.nom = nom


class Boursier (Etudiant) :

        def __init__(self, prenom, nom, bourse, adresse) :
                super().__init__(prenom, nom)
                self.bourse = bourse
                self.place = adresse
```

NOTEZ BIEN : **Pour le cas de l'héritage multiple, il suffit tout simplement de séparer les classes mères (superclasses) à chaque fois par une virgule.**

`super()` **permet d'invoquer une méthode de classe mère (superclasse).**

- Il existe une méthode nommée `issubclass (fille, superclasse)` pour vérifier si l'héritage a bien eu lieu.