IFOSUP

Institut de Formation Supérieure
Ville de Wavre

MATHEMATIQUES APPLIQUEES A L'INFORMATIQUE
Rappels sur le langage Python

Gérard Barmarin

2021-2022


PYTHON

Pourquoi Python?

- Python est portable, sur les différentes variantes d'*Unix/Linux*, sur les OS propriétaires: *MacOS*, *BeOS*, *NeXTStep*, *MS-DOS* et sur *Windows*.

- Python est gratuit

- Python convient aussi bien à des petits scripts qu'à des projets complexes

- La syntaxe de Python est simple et, combinée à des types de données évolués (listes, dictionnaires,...), conduit à des programmes à la fois compacts et lisibles.

- Python est dynamiquement typé. Tout objet manipulable par le programmeur possède un type bien défini à l'exécution, qui n'a pas besoin d'être déclaré à l'avance.

- La bibliothèque standard de Python peut être complétée par une multitude de bibliothèques spécialisées (Math, traitement d’images, AI…)

Quelques rappels simples en Python

Pseudo-commentaires pour pouvoir utiliser les caractères accentués

Il est vivement recommandé pour nous francophones d'inclure le pseudo-commentaire suivant au début de tous les scripts Python que l'on écrit (obligatoirement à la 1e ou à la 2e ligne) :

```
# -*- coding:Utf-8 -*-
```

Ce pseudo-commentaire indique à Python que vous utiliserez dans votre script le système de codage mondial sur deux octets appelé Unicode (dont la variante Utf-8 ne code que les caractères « spéciaux » sur deux octets, les caractères du jeu ASCII standard restant codés sur un seul octet). Ce système est devenu très populaire, car il présente l'avantage de permettre la coexistence de caractères de toutes origines dans le même document (caractères grecs, arabes, cyrilliques, japonais, etc.).

Si vous n'indiquez aucun système de codage, vous recevrez des messages d'avertissement de la part de l'interpréteur, et vous éprouverez peut-être même quelques difficultés à éditer correctement vos scripts.

En python, vive l'indentation!

En python, les limites des instructions et des blocs sont définies par la mise en page et c'est donc le retour à la ligne qui termine une instruction et l'indentation qui sert à regrouper les instructions qui font partie d'un même bloc:

```
if embranchement == "vertébrés":                    # 1
    if classe == "mammifères":                      # 2
        if ordre == "carnivores":                   # 3
            if famille == "félins":                 # 4
                print "c'est peut-être un chat"     # 5
        print "c'est en tous cas un mammifère"      # 6
    elif classe == 'oiseaux':                       # 7
        print "c'est peut-être un canari"           # 8
print"la classification des animaux est complexe"   # 9
```

Un commentaire Python commence toujours par le caractère # . Tout ce qui est inclus entre ce caractère et le saut à la ligne suivant est ignoré par le compilateur. En définitive, Python vous force donc à écrire du code lisible!

Ecriture des nombres:

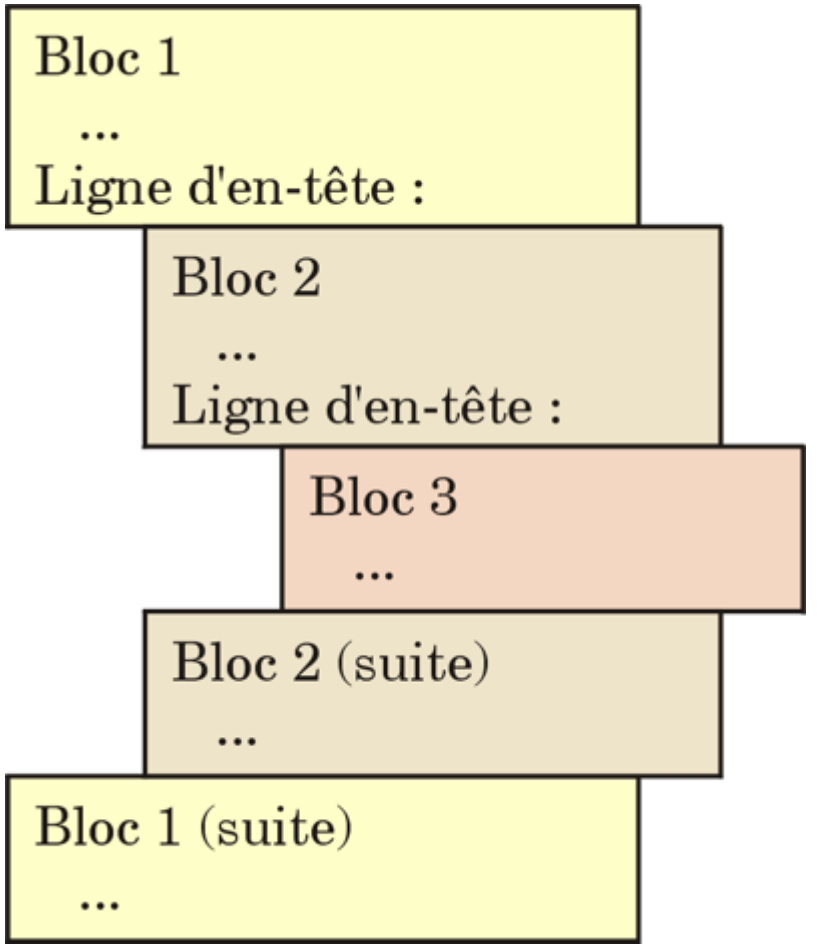
Dans tous les langages de programmation, les conventions mathématiques de base sont celles en vigueur dans les pays anglophones : le séparateur décimal sera donc toujours un point, et non une virgule comme chez nous. Dans le monde de l'informatique, les nombres réels sont souvent désignés comme des nombres "à virgule flottante", ou encore des nombres "de type float".

Le schéma ci-contre en résume ces principes:

- Les blocs d'instructions sont toujours associés à une ligne d'en-tête contenant une instruction bien spécifique (if, elif, else, while, def, ...) se terminant par un double point.

- Les blocs sont délimités par l'indentation : toutes les lignes d'un même bloc doivent être indentées exactement de la même manière (c'est-à-dire décalées vers la droite d'un même nombre d'espaces). Le nombre d'espaces à utiliser pour l'indentation est quelconque, mais la plupart des programmeurs utilisent des multiples de 4.

- Notez que le code du bloc le plus externe (bloc 1) ne peut pas lui-même être écarté de la marge de gauche (Il n'est imbriqué dans rien).



Vous pourriez aussi indenter à l'aide de tabulations, mais attention à ne pas utiliser tantôt des espaces, tantôt des tabulations pour indenter les lignes d'un même bloc. En effet, même si le résultat paraît identique à l'écran, espaces et tabulations sont des codes binaires distincts : Python considérera donc que ces lignes indentées différemment font partie de blocs différents. Il peut en résulter des erreurs difficiles à déboguer. La plupart des programmeurs préfèrent donc se passer des tabulations.

Les noms de variables en python

En Python, les noms de variables doivent obéir à quelques règles simples :

- Un nom de variable est une séquence de lettres (a ® z , A ® Z) et de chiffres (0 ® 9), qui doit toujours commencer par une lettre.
- Seules les lettres ordinaires sont autorisées. Les lettres accentuées, les cédilles, les espaces, les caractères spéciaux ($, #, @, etc.) sont interdits, à l'exception du caractère « _ » (souligné ou underscore).
- La casse est significative (les caractères majuscules et minuscules sont différenciés).
  Donc : Joseph, joseph, et JOSEPH sont des variables différentes.

En plus de ces règles, vous ne pouvez pas utiliser comme noms de variables les 29 « mots réservés » ci-dessous (qui sont utilisés par le langage lui-même) :

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

Conseils (que vous appliquez surement déjà):

Efforcez-vous de bien choisir vos noms de variables: courts mais explicites, de manière à exprimer clairement ce que la variable est censée contenir.
altitude, altit ou alt conviennent mieux que x pour exprimer une altitude!

Prenez l'habitude d'écrire les noms de variables en caractères minuscules (y compris la première lettre), cela vous évitera d’hésiter.
Il s'agit d'une simple convention, mais elle est très largement respectée.
N'utilisez les majuscules qu'à l'intérieur même du nom, pour en augmenter éventuellement la lisibilité, comme dans tableDesMatieres, par exemple.

Affectation d'une valeur à une variable

Le symbole d'affectation d'un contenu à une variable est le signe « = »:

```
msg = "Mon message"
```

Affectation du même contenu à plusieurs variables:

```
x = y = 7
```

Affectations multiples en une ligne:

```
a , b = 4, 8.33  # équiv. à a=4, b=8.33
```

Sous Python, l'affectation multiple permet de programmer l'échange (swap) d'une manière particulièrement élégante :

```
a, b = b, a   # et Voilà on a échangé les valeurs de a et de b!
```

Déclaration du type de variables

En Python, il n'est pas nécessaire de définir dans le code le type des variables avant de pouvoir les utiliser. Il suffit en effet d'assigner une valeur à un nom de variable pour que celle-ci soit ***automatiquement créée avec le type qui correspond au mieux à la valeur fournie*** (nombre entier, chaîne de caractères, nombre à virgule flottante...).

On parle de ***typage dynamique***, par opposition au ***typage statique*** en *C++* ou en *Java*. Dans ces langages, il faut toujours déclarer (définir) par des instructions le nom et le type des variables, et ensuite seulement leur assigner un contenu, qui doit bien sûr être compatible avec le type déclaré. Le typage statique est préférable dans les langages compilés, parce qu'il permet d'optimiser l'opération de compilation (dont le résultat est un code binaire « figé »).

```
print type(mavariable) # renvoie le type de la variable
var2 = float(var1)     # change var1 en float et place le résultat dans var2
```

Le type entier

En python, le type entier (integer) est encodé dans la mémoire de l'ordinateur sous la forme d'un bloc de 4 octets (32 bits). Or la gamme de valeurs décimales qu'il est possible d'encoder sur 4 octets s'étend seulement de -2147483648 à + 2147483647.

Les calculs effectués avec ce type de variable sont très rapides, parce que le processeur de l'ordinateur est capable de traiter directement par lui-même de tels nombres entiers à 32 bits.
En revanche, lorsqu'il est question de traiter des nombres entiers plus grands, ou encore des nombres réels (nombres « à virgule flottante » ou float), l'interpréteur doit effectuer un gros travail de codage/décodage, afin de ne présenter en définitive au processeur que des opérations binaires successives sur des nombres entiers de 32 bits au maximum.

Puisqu'en Python, le type des variables est défini de manière dynamique et qu'il s'agit du type le plus performant (aussi bien en termes de vitesse de calcul qu'en termes d'occupation de place dans la mémoire), Python utilise le type integer par défaut, chaque fois que cela est possible, c'est-à-dire tant que les valeurs traitées sont des entiers compris entre les limites mentionnées plus haut.

Lorsque les valeurs traitées sont des nombres entiers se situant au-delà de ces limites, les variables auxquelles on affecte ces nombres sont alors **automatiquement** définies comme appartenant au type « entier long » (lequel est désigné par « long » dans la terminologie Python). Ce type long permet l'encodage de valeurs entières avec une précision quasi infinie : une valeur définie sous cette forme peut en effet posséder un nombre de chiffres significatifs quelconque, ce nombre n'étant limité que par la taille de la mémoire disponible sur l'ordinateur utilisé !

Le type « float »

Le type « nombre réel », ou « nombre à virgule flottante » ou « float », autorise les calculs sur de très grands ou très petits nombres (données scientifiques, par exemple), avec un degré de précision constant.

Pour qu'une donnée numérique soit considérée par Python comme étant du type float, il suffit qu'elle contienne dans son écriture un élément tel qu'un point décimal ou un exposant de 10.

Par exemple, les données :

3.14 1 . .001 1e100 3.14e-10

sont automatiquement interprétées par Python comme étant du type float.

Le type float utilisé dans notre exemple permet de manipuler des nombres (positifs ou négatifs) compris entre 10^{-308} et 10^{308} avec une précision de 12 chiffres significatifs. Ces nombres sont encodés d'une manière particulière sur 8 octets (64 bits) dans la mémoire de la machine : une partie du code correspond aux 12 chiffres significatifs, et une autre à l'ordre de grandeur (exposant de 10).

Au-delà de ces valeurs, il y a un dépassement de capacité (sans message d'erreur de la part de Python) : les nombres vraiment trop grands sont tout simplement notés « inf » (pour « infini »).

Le type chaîne de caractères (string)

En Python, une donnée de type string est une suite quelconque de caractères délimitée soit par des apostrophes (simple quotes), soit par des guillemets (double quotes).
Si dans la chaîne, il y a des apostrophes, utilisez les guillemets et à l'inverse s'il y a des guillemets utilisez l'apostrophe pour délimiter la chaîne, vous pouvez aussi utiliser l'antislash à l'intérieur d'une chaîne de caractères pour y insérer un certain nombre de codes spéciaux (sauts à la ligne, apostrophes, guillemets, etc.).

Remarquez aussi que l'instruction print insère un espace entre les éléments affichés.

Le caractère spécial « \ » (antislash) permet également d'écrire sur plusieurs lignes une commande qui serait trop longue pour tenir sur une seule (cela vaut pour n'importe quel type de commande). Exemples :

```
phrase1 = 'les oeufs durs.'
phrase2 = '"Oui", répondit-il,'
phrase3 = "j'aime bien"
print phrase2, phrase3, phrase1 # affichera: "Oui", répondit-il, j'aime bien les oeufs durs.
txt = '"N\'est-ce pas ?" répondit-elle.' # txt3 contient "N'est-ce pas ?" répondit-elle.
salut = "Ceci est une chaîne plutôt longue\n  contenant plusieurs lignes \ de texte (Ceci
fonctionne\n  de la même façon en C/C++. Notez que les blancs en début de ligne sont
significatifs.\n"
print salut # affichera:
Ceci est une chaîne plutôt longue
 contenant plusieurs lignes de texte (Ceci fonctionne
 de la même façon en C/C++. Notez que les blancs en début de ligne sont significatifs.
```

Les listes

En Python, on peut définir une liste comme une collection d'éléments séparés par des virgules, l'ensemble étant enfermé dans des crochets. Exemple, la valeur de la variable jour est une liste:

```
>>> jour = ['lundi', 'mardi', 'mercredi', 1800, 20.357, 'jeudi', 'vendredi']
```

Comme on peut le constater, les éléments qui constituent une liste peuvent être de types variés (ici les trois premiers éléments sont des chaînes de caractères, le quatrième élément est un entier, le cinquième un réel, etc.) On peut même faire une liste de listes. A cet égard, le concept de liste est donc assez différent du concept de « tableau » (array)

On peut accéder à chaque élément de la liste individuellement si l'on connaît son index dans la liste comme c'est aussi le cas pour les caractères dans une chaîne, il faut cependant retenir que la numérotation de ces index commence à partir de zéro, et non à partir de un.

```
>>> print jour[2]
mercredi
>>> print jour[4]
20.357
>>> print jour
['lundi', 'mardi', 'mercredi', 1800, 20.357, 'jeudi', 'vendredi']
>>> jour[3] = jour[3] + 47
>>> print jour
['lundi', 'mardi', 'mercredi', 1847, 20.357, 'jeudi', 'vendredi']
```

Création de listes

Création de listes à une dimension

On peut bien sûr créer la liste de manière exhaustive, mais aussi créer une liste vide puis lui ajouter des éléments ou utiliser un remplissage automatique associant une formule, un range et une condition! :

```
>>> A=[ ]
>>> print(A)
[ ]
>>> A.append(5)
>>> A.append("sept")
>>> print(A)
[5, 'sept']

>>> A=[x**2 for x in range(11)]
>>> A
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
>>> B=[2*x+3 for x in range(11)]
>>> B
[3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23]
>>> C=[x for x in range(25) if x%3==0]
>>> C
[0, 3, 6, 9, 12, 15, 18, 21, 24]
```

Création de listes

Création de listes à deux dimensions

On peut la créer de façon exhaustive :

```
>>> A=[[1,2,3],[4,5,6]]
>>> print(A)
[[1, 2, 3], [4, 5, 6]]
```

Mais on peut aussi utiliser les boucles :

```
>>> B=[[0 for j in range(6)] for i in range(3)]
>>> print(B)
[[0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0]]
```

Extraction d'éléments d'une liste

```
>>> nombres = [5, 38, 10, 25]
>>> print nombres[2]            # Affiche le (2+1)ème élément car le premier est noté [0]
10
>>> print nombres[1:3]          # Affiche la liste du (1+1)ème élément au 3ème inclus
[38, 10]
>>> print nombres[2:3]          # Affiche la liste des éléments du (2+1)ème au 3ème (donc le 3ème)
[10]                            # uniquement
>>> print nombres[2:]           # Affiche la liste de tous les éléments du (2+1)ème à la fin
[10, 25]
>>> print nombres[:2]           # Affiche la liste de tous les éléments depuis le début jusqu'au 2ème
[5, 38]
>>> print nombres[-1]           # Affiche le dernier élément de la liste
25
>>> print nombres[-2]           # Affiche l'avant-dernier élément de la liste
10
```

note: les éléments extraits de la liste comme une tranche [x:x] sont eux même des listes même si elles ne contiennent qu'une valeur alors que les éléments extraits avec leur indice redeviennent des variables isolées)
Si l'on souhaite accéder à un élément faisant partie d'une liste, elle-même située dans une autre liste, il suffit d'indiquer les deux index entre crochets successifs :

```
M = [[5, 10, 9, 7], [1, 3, 4, 2], [6, 8, 0, 11]]     #une simulation de matrice 3 x 4
>>> print nombres[2][1]          # Affiche le (1+1)ème élément (2ème colonnes)du (2+1)ème élément
[8]                              # (3ème ligne)
```

Longueur d’une liste

La fonction len() s'applique aux listes et aux chaînes de caractères, et renvoie le nombre d'éléments présents dans la liste ou la chaîne :

```
>>> len(jour)
7
```

Ajouter un élément à une liste

Il est possible d'ajouter un élément à une liste:

```
>>> jour = ['lundi', 'mardi', 'mercredi', 'jeudi', 'vendredi']
>>> jour.append('samedi')
>>> print jour
['lundi', 'mardi', 'mercredi', 'juillet', 'jeudi', 'vendredi', 'samedi']
```

A la deuxième ligne, on applique la méthode append() à l'objet jour , avec l'argument 'samedi'. Si on se rappelle que append signifie « ajouter » en anglais, on peut comprendre que la méthode append() est une sorte de fonction qui est rattachée aux objets du type « liste » et qui ajoute l'argument qui la suit à la **fin** de la liste.

Notons simplement que l'on applique une méthode à un objet en reliant les deux à l'aide d'un point. (D'abord le nom de la variable qui référence l'objet, puis le point, puis le nom de la méthode, cette dernière toujours accompagnée d'une paire de parenthèses même vides le cas échéant).

Autres manipulations sur les listes:

– *append()* ajoute un élément passé en argument en fin de liste.
– *remove()* enlève la première occurrence de la valeur passée en argument.
– *pop()* enlève la valeur correspondant à l'index passé en argument et renvoie cette valeur.
– *extend()* ajoute une liste passée en argument en fin de liste.

```
>>> A=[5,4,3,4,7]
>>> A
[5, 4, 3, 4, 7]
>>> A.append(8)
>>> A
[5, 4, 3, 4, 7, 8]
>>> A.remove(4)
>>> A
[5, 3, 4, 7, 8]
>>> A.pop(1)
3
>>> A
[5, 4, 7, 8]
>>> A.extend([0,0,7])
>>> A
[5, 4, 7, 8, 0, 0, 7]
```

la fonction input()

Dans les scripts en mode texte la méthode la plus simple pour introduire des données consiste à employer la fonction intégrée **input()** qui provoque une interruption dans le programme courant et qui invite l'utilisateur à entrer des caractères au clavier et à terminer avec <Enter>. Lorsque cette touche est enfoncée, l'exécution du programme reprend, et la fonction fournit en retour une valeur correspondant à ce que l'utilisateur a entré. Cette valeur peut alors être assignée à une variable quelconque. L'argument entre parenthèses est le texte qui sera affiché à l'écran pour prévenir l'opérateur que vous attendez une entrée de caractère.

Attention à la confusion avec l'ancien raw_input : **raw_input() n'existe plus en Python 3.x**, seul input() existe . En fait, l'ancien raw_input() a été renommé input() et c'est l'ancien input() qui a disparu. En Python 2 , raw_input() renvoyait toujours une chaîne et input() choisissait le type en fonction des caractères entrés. En Python 3 input() renvoie toujours une chaîne de caractères. Vous pouvez/devez ensuite convertir cette chaîne en nombre à l'aide de **int()** ou de **float()**.

```
prenom = input('Entrez votre prénom (entre guillemets) : ')
Ernest
print 'Bonjour,', prenom
Bonjour, Ernest
a = input('Entrez une donnée : ')     # stockage dans a de la chaîne entrée au clavier
Entrez une donnée : 52.37
type(a)                               # Affichage du type de la variable
<type 'str'>
b = float(a)                          # conversion en valeur numérique
type(b)                               # Afficahge du type de la variable
<type 'float'>
```

Fonction de sortie: print

Dans les scripts en mode texte la méthode la plus simple pour afficher les résultats est la fonction print qui permet d'afficher des chaînes de caractères.

Cependant, comme pour l'instruction input(), si on lui demande d'afficher un entier, il va automatiquement convertir l'entier en chaîne de caractères pour pouvoir l'afficher.

A noter que l'instruction print provoque un saut de ligne (comportement par défaut).

Pour écrire une suite de chaînes de caractères, on pourra utiliser la concaténation de chaînes en utilisant l'opérateur "+".

```
>>> mot1="Romane"
>>> print(mot1)
Romane
>>> mot2="Lucie"
>>> print(mot2)
Lucie
>>> print(mot1 + " et "+ mot2)
Romane et Lucie
>>> print ("{} et {}".format(mot1,mot2) )
Romane et Lucie
```

Fonction de sortie: création d'un fichier de résultat

Dans nos scripts sur les matrices, il pourra être intéressant de récupérer les données dans un fichier.

Voici un exemple de création d'un fichier résultat:

```
# -*- coding:utf-8 -*-
fichier1=open('monfichier.txt','w')
fichier1.write("n" +"\t"+"n^2"+"\n")
for i in range(10):
   fichier1.write(str(i) +"\t"+str(i**2)+"\n")
fichier1.close()
```

La lettre w signifie que le fichier est ouvert en écriture. Cela a deux conséquences capitales à retenir :

- si le fichier monfichier.txt n'existe pas, celui-ci est créé.
- si le fichier monfichier.txt existe déjà, celui-ci est effacé : toutes ses données actuelles seront perdues.

On peut bien sûr rouvrir le fichier en Python, ici pour l'afficher à l'écran:

```
fichier2=open('monfichier.txt','r')
print fichier2.read()
fichier2.close()
```

La lettre r signifie que le fichier est ouvert en lecture. La commande read() peut prendre comme argument un nombre désignant le nombre de caractères à lire à partir de la position courante. On peut aussi préférer une lecture par ligne en utilisant la commande readline().

Autre type de fichier intéressant pour les entrées et les sorties, les fichiers CSV

Les fichiers du type CSV (Coma Separated Value) sont des fichiers à la structure très simple et donc très souvent utilisés (vous pourrez les importer très facilement dans Excel)
Pour se faciliter la vie il faut importer la bibliothèque CSV.

```
import csv
```

Lecture d'un fichier CSV
On crée tout d'abord un objet "reader" (que nous nommerons cr).

```
cr = csv.reader(open("MONFICHIER.csv","rb"))
```

Et là, on obtient chaque rangée (sous forme d'une liste des colonnes) comme ceci:

```
for row in cr:
    print row
```

Priorité des opérations arithmétiques en Python

PEMDAS :

- **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**.

- **E** pour ***exposants***. Les exposants sont évalués ensuite, avant les autres opérations.
  Ainsi **2**1+1 = 3** (et non 4), et **3*1**10 = 3** (et non 59049 !).

- **M** et **D** pour ***multiplication*** et ***division***, qui ont la même priorité.

- **A** et **S** pour l'***addition*** et la ***soustraction S***, lesquelles sont donc effectuées en dernier lieu.
  Ainsi **2*3-1 = 5** (plutôt que 4), et **2/3-1 = -1** (par défaut Python effectue une division ***entière***).

- Si deux opérateurs ont la même priorité, l'évaluation est effectuée de gauche à droite.
  Ainsi dans **59*100/60**, la multiplication est effectuée en premier, et la machine effectue ensuite **5900/60**, ce qui donne **98**. Si la division était effectuée en premier, le résultat serait **59** (rappelez-vous ici encore qu'il s'agit d'une division entière).

Structure du IF en python: si (if), sinon (elif) , autrement (else)

Exemple:

```
a = 0
if a > 0 :
... print "a est positif"
elif a < 0 :
... print "a est négatif"
else:
... print "a est nul"
```

Opérateurs utilisables dans la condition testée:

```
x == y                              # x est égal à y , notez bien les deux signes « égale »
x != y                              # x est différent de y
x > y                               # x est plus grand que y
x < y                               # x est plus petit que y
x >= y                              # x est plus grand que, ou égal à y
x <= y                              # x est plus petit que, ou égal à y
(condition1) and (condition2)       # condition1 et condition2 vraies simultanément
(condition1) or (condition2)        # condition1 ou condition2 vraie
a in liste1                         # a inclus dans liste1
```

Boucle while

Le but de la boucle while est de répéter certaines instructions tant qu'une condition est respectée.

On n'est pas donc obligé de savoir au départ le nombre de répétitions à faire.

L'instruction break peut servir à sortir de la boucle while.

```
# -*- coding:utf-8 -*-
print("*** Avant la boucle ***")
entree="bidule"
while entree != "":
    print("------ dans la boucle ------")
    entree=input("Taper un mot : ")
    if entree=="ouistiti" :
        print("J'ai vu un ouistiti !")
        break
print("*** Après la boucle ***")
```

Boucle for

Le but de la boucle for est de répéter certaines instructions pour chaque élément d'une liste.

Contrairement à d'autres langages, l'usage d'une liste est donc nécessaire !

range permet de créer rapidement une liste d'entier, sans devoir les écrire.

Attention, le dernier entier n'est pas pris dans la liste !!

"range(11)" équivaut à "[0,1,2,3,4,5,6,7,8,9,10]".

"range(3,7)" équivaut à "[3,4,5,6]".

```
# -*- coding:utf-8 -*-
print("Tous en choeur !!!")
for i in range(1,4) :
    print("Et "+str(i)+"!")
print("Zéro!")
```

```
# Note: on peut simuler une boucle for avec while
nb_repetitions = 3
i = 1
while i <= nb_repetitions :
    print("Et "+str(i)+"!")
    i = i+1
print("Zéro!")
```

Importation de bibliothèques

Python intègre un certain nombre de fonctions mais il n'est pas possible d'y intégrer toutes les fonctions imaginables!
Il existe donc des bibliothèques regroupant les fonctions utiles dans un domaine particulier.

Le module math, par exemple, contient les définitions de nombreuses fonctions mathématiques telles que sinus, cosinus, tangente, racine carrée, etc. Pour pouvoir utiliser ces fonctions, il vous suffit d'incorporer la ligne suivante au début de votre script python :

```
from math import *
```

Cette ligne indique à Python qu'il lui faut inclure dans le programme courant toutes les fonctions (c'est la signification du symbole *) du module math qui contient une bibliothèque de fonctions mathématiques préprogrammées. Vous pouvez ensuite les utiliser… si vous connaissez leur nom et la structure des paramètres à passer à la fonction dans les parenthèses!

```
x = sin(angle) # pour assigner à la variable x le sinus de angle (en radians!)
```

Exercices sur les matrices en Python

python -m pip install -U scikit-image

Structure de base de vos programmes: `math-ex1-0.py` (en téléchargement sur Teams)

Vous pouvez utiliser au départ ce canevas
pour réaliser vos exercices sur les matrices et tester votre environnement.

Enoncé exercice 1.1 :

Premiers essais sans NumPy: utilisation des listes imbriquées pour manipuler des matrices

Ecrire un pgm en Python sans utiliser Numpy qui :

- affiche la liste de listes suivante: M= [[1,2,3],[4,5,6],[7,8,9]]
- Affiche l'élément m_{23}
- Affiche la 3ème ligne
- Affiche la première colonne si l'on considère la liste de liste comme une matrice

Solution possible Exercice 1.1 : `math-ex1-1.py` (en téléchargement sur Teams)

Enoncé exercice 1.2 :

Premiers essais sans NumPy: utilisation des listes imbriquées pour manipuler des matrices

Ecrire un pgm en Python sans utiliser Numpy qui :

- Affiche les listes A= [[1,2,3],[4,5,6],[7,8,9]] et B= [[9,9,9],[9,9,9],[9,9,9]]
- Calcule A+B et l'affiche (que produit cette addition de liste?)
- Calcule et affiche A+B selon les règles du calculs matriciel

Solution possible Exercice 1.2 : `math-ex1-2.py` (en téléchargement sur Teams)

Source:

Sources:
https://python.developpez.com/cours/TutoSwinnen/

https://wordpress.callac.online/index.php/python/