LETTRES SORBONNE UNIVERSITÉ

Bases de Python

Installation, Notebook, variables et boucles

Crédits : Christian Vincent, Sorbonne Université

Sommaire du TD

1 Installation et exécution

Ceci décrit l'installation pour votre ordinateur personnel, les machines du département ont, en principe, déjà tout ce qu'il faut. **Nous utilisons les versions** $3.x$ **de Python et pas** $2.7.x$**.**

Deux installations vous sont proposées ci-après, la première, celle par défaut qui est légère, une deuxième qui est plus complète sera utile pour la suite de votre année. Vous pouvez enfin utiliser *Python* aussi en ligne sans installation.

1.1 Première installation possible

Si vous n'avez pas beaucoup de place sur votre ordinateur, cette installation est à privilégier. Elle intègre un environnement de développement. Rendez-vous sur : https://www.python.org

Choisissez le menu *Download*puis selon votre système d'exploitation, téléchargez la dernière version 3.6.0 qui est en ce moment disponible. Cette version de *Python* est fournie avec un environnement de développement sous Windows appelé *IDLE* (Environnement de Développement Intégré utilisant Python 3.x).

On choisit ensuite dans le menu installé, *Python 3.6 > IDLE (Python GUI)*. NB : Il faut lancer l'installation **sur un lecteur où vous avez les droits d'écriture**

1.2 Deuxième installation possible (celle recommandée pour nous)

Rendez-vous à l'adresse http://continuum.io/

Par le menu "*Anaconda*", cliquez sur *Download*. Lisez brièvement le contenu de la page puis appuyer sur le bouton suivant (attention, l'interface change souvent et ce qui écrit ici est peut-être obsolète). Entrez une adresse mail pour accéder au téléchargement (même une fausse). Vous recevrez des informations sur les mises à jour, les nouveautés, etc. de façon très épisodique et professionnelle si l'adresse est exacte.

À l'heure actuelle, presque tous les OS sont en 64 bits (allez dans les propriétés de votre système pur le vérifier). Sous Windows, il suffit ensuite de lancer l'exécutable et de désigner un lecteur où vous avez les droits d'écriture Choisissez dernière version **Python 3.6** à la date d'aujourd'hui.

L'installation d'*Anaconda* effectuée, plusieurs interfaces sont disponibles : IDLE, Spyder, IPython et IPython-notebook. C'est cette dernière que nous allons utiliser.

Dans le menu du dossier *Anaconda*, lancez ensuite **IPython Notebook** (ou *Jupiter Notebook*)qui utiliser votre navigateur par défaut. Utiliser de préférence *Firefox* qui est plus stable avec *IPython Notebook*.

1.3 Directement en ligne

Quand vous n'avez pas la possibilité d'utiliser un ordinateur où *Python* est installé, on peut l'utiliser en ligne et ainsi s'exercer. Rendez-vous à l'adresse https://www.python.org/ et sur la page d'accueil de la *Python Software Foundation*, cliquez sur l'icône en jaune (*Launch Interactive Shell*). Ceci permet de faire de petits tests mais pas beaucoup plus.

2 Prise en main de IPython Notebook

2.1 Premiers contacts

2.1.1 Gérer ses "notebook"

Les "notebook" sont vos fichiers de programmes. Ils sont organisés en cellule (partie de code) et portent dans votre système de fichiers l'extension **.ipynb** Il est important de mémoriser l'emplacement de votre travail si vous utilisez un ordinateur de la fac : pour la séance suivante, il est fortement conseillé de stocker vos programmes sur une clé USB votre travail, etc.

2.2 Interprétation directe

Exercice 1:
L'écriture **In[ ]** est le *prompt* de début de ligne qui signale à l'utilisateur qu'il peut commencer à écrire ici. Tout ce qui est écrit sur fond gris est du code à entrer.

Entrez dans la première cellule le texte suivant.

```
2 + 3
```

Rien ne se passe ! Il faut demander à *Python* d'exécuter ce code. Pour cela plusieurs possibilités.

- dans le menu "Cell" (cellule) cliquez sur "run"
- dans la barre d'outils, cliquez sur "run cell"
- au clavier, tapez sur Ctrl + entrée (exécute la cellule)

- au clavier, tapez sur Alt + entrée (exécute la cellule et en créé une nouvelle)

Il est recommandé de s'habituer rapidement à ces deux derniers raccourcis.

Entrez puis exécutez les textes suivants, observez et notez à chaque fois les résultats.

```
2 + 3 * 4
```

Si l'on veut entrer maintenant $2+3\times4/2$, pour éviter de taper à nouveau la ligne, placez le curseur dans la ligne précédente (la ligne qui contient $2+3\times4$) et ajoutez /2, puis exécutez à nouveau la ligne.

```
2 + 3 * 4 / 2
```

Découvrez d'autres opérateurs. Essayez ensuite les lignes suivantes et notez les résultats obtenus (la virgule "," pour les nombres est ici un point ".")

```
5 - 2 * 3
```

```
7 / 2
```

```
7 // 2
```

```
7 // 2.0
```

```
7 % 2
```

```
7 ** 2
```

```
2 ** 7
```

Vous avez travaillé avec plusieurs **opérateurs**, un opérateur permet de définir une **opération** entre des **opérandes**. Par exemple quand j'écris $3+2$, "+" est l'opérateur qui définit l'addition entre les opérandes 2 et 3.

Faites plusieurs entrées au clavier pour vérifier que la priorité des opérations est la même qu'en mathématiques. Par exemple, essayez $3+2\times5 = 3+(2\times5)$.

De la même manière que dans une langue naturelle telle que le français, l'ordre des éléments est un des éléments importants de l'analyse syntaxique (que l'on peut voir comme la décomposition et la mise en relation des opérateurs et des opérandes)

```
3 + 2 * 5
```

Justifiez et vérifiez dans la console que :

```
5 * 2 / 3
```

donne 3.333... (donc multiplication et division au même niveau mais exécution de gauche à droite), et que

```
5 / 3 * 2
```

aussi.

Que va donner $(5^2 \times 3)$:

```
5 ** 2 * 3
```

puis $(5 * 2^3)$

```
5 * 2 ** 3
```

Justifiez. Pour terminer, justifiez les résultats obtenus en entrant (2^{3^2})

```
2**3**2
```

et $(\frac{\frac{15}{6}}{2} = 1,25$ et non pas $\frac{15}{\frac{6}{2}} = 7,5)$

```
15 / 6 / 2
```

Astuce : outre les icônes de la barre d'outil, vous pouvez très avantageusement utiliser les raccourcis clavier. Vous en obtenez la liste par le menu "*Help > Keyboard shortcuts*"

2.3 Interprétation différée

Exercice 2:
Parfois une suite de commandes peut être entrée, mais *Notebook* attend votre ordre (Ctrl+Entrée ou Alt+Entrée) pour lancer l'exécution (le calcul)...

Entrez le code suivant en prenant soin d'appuyer simplement sur la touche "*Entrée*" à la fin de chaque ligne.

```
# je suis un commentaire
for lettre in "lapin":
    print(lettre)
```

Vous remarquez plusieurs **points importants** :

1. Sur Notebook, la "sortie" du code (ce que fait le code) est précédée par **Out[xx]** où xx est le numéro attribué à la cellule.

2. Comment réagit *Python* avec ce qui est placé après le symbole # ? il ignore ce qui suit le dièse, on peut donc écrire n'importe quoi après un dièse. On commente son programme.

3. Que se passe-t-il après avoir entré : *for lettre in "lapin":* et appuyé sur *Entrée* ? Notebook provoque une indentation pour ce qui suit. Elle est conservée jusqu'à la sortie de la zone qui débute par *while* (deux fois *Entrée* provoque la sortie)

4. Comment avez-vous fait pour lancer l'exécution des 3 ? on appuie deux fois sur Entrée après la dernière ligne : une fois pour le retour de ligne (que l'on laisse vide) et une fois pour valider l'ensemble

5. Dans la console, au fur et à mesure de la frappe, une coloration des mots entrés apparaît. C'est la ***coloration syntaxique***. A chaque *type* de mot-clé est associée une couleur.

   Ces couleurs peuvent être différentes selon l'ordinateur car elles peuvent être redéfinies.

6. Ré exécutez le code précédent en écrivant pour la dernière ligne

```
    print(lettre, end =" ")
```

   Que remarquez-vous quant au résultat ? **Attention** le symbole écrit entre les deux " " est là pour dire qu'il faut mettre un espace. C'est juste pour que ce texte de TD soit lisible. Ici par exemple, c'est 2 espaces.

```
   print("  ")
```

   Essayez encore avec cette fois

```
    print(lettre, end ="*")
```

   Quel est le rôle de end ? Place la suite des informations à la suite de ce qui précède avec un espace

7. Vous avez remarqué aussi que les nombres sont entrés "directement" dans les parenthèses du *print* mais une chaîne de caractères (comme lapin), s'entre entre 2 guillemets.

3 Notion de variable, programme et fichier de programme

Vous pouvez donc écrire plusieurs lignes de programme et les exécuter ensuite.

3.1 Les variables

Tapez et exécutez les lignes suivantes

```
a = 2
b = 3
print(a+b, a-b, a*b, a/b, a**b)
```

Vérifiez la syntaxe, puis exécutez ce programme.

Changez plusieurs fois les valeurs de a et b et exécutez le programme (appuyez sur Ctrl + Entrée après chaque changement).

Essayez en particulier le cas où $b = 0$ et remarquez que l'exécution amène à une erreur. ZeroDivisionError

Vous devez bien lire les erreurs pour pouvoir corriger vos programmes. La machine vous signale à quel moment (à quelle ligne généralement) elle ne peut plus exécuter vos instructions. C'est à vous de retrouver l'erreur (parfois plusieurs lignes en arrière).

Dans le programme précédent a et b sont des **variables**.

```
a = 2
```

Signifie : j'affecte à la variable a la valeur 2. Le = est **l'opérateur d'affectation**. On pourrait traduire par a prend la valeur 2

```
a = a + 1
```

Signifie : j'affecte à a la valeur $a + 1$. Le = est **l'opérateur d'affectation**. On pourrait traduire cette instruction par "a prend la valeur connue de a à laquelle j'ajoute 1". On dit encore qu'on a **incrémenté** a de 1.

Pour apprécier l'effet, entrez les lignes suivantes, et affichez ensuite la valeur de a de la façon suivante :

```
a = 2
print("a =", a)
a = a + 1
print("a =", a)
```

L'instruction **print()** signifie que Python doit afficher ce qui se trouve entre les deux parenthèses. Ceci permet, notamment, de contrôler l'évolution du programme. **Une bonne pratique de programmation consiste à afficher régulièrement la valeur des données manipulées pour vérifier que le résultat ets bien conforme aux attentes.** Une fois que le code est sûr, on peut supprimer ou (commenter) ces instructions.

Python permet de raccourcir le programme de début de paragraphe 3.1. Essayez :

```
# Premier programme
a, b, c = 2, 3, "Resultats:"
print(c, a+b, a-b, a*b, a/b, a**b)
```

Cette propriété que permet *Python* s'appelle l'**affectation parallèle**. Elle permet d'attribuer à plusieurs variables simultanément, leurs valeurs respectives.

Par exemple on peut très facilement échanger le contenu de 2 variables. Entrez

```
a, b = 4, 7
print("a=",a,"et b=", b)
a, b = b, a  # a prend la valeur b et inversement
print("a=",a,"et b=", b)
```

Ici, on a réalisé (ou implanté) un **algorithme**, c'est àd dire qu'on a trouvé le moyen de transformer un besoin (exprimé en langue naturelle) en un progarmme (exprimé en langage informatique).

3.2 Les programmes

Un programme est une suite d'instructions, de variables, etc.

Les fichiers de programme *Python* portent l'extension .py généralement (.ipynb pour les fichiers notebook). Quand on veut faire exécuter un programme *python* externe dans Notebook, on tape la commande

```
run p1.py
```

Le programme doit être dans ce cas placé dans le répertoire où se situe le fichier notebook *TD1.ipynb* pour que la machine puisse le "trouver". Sinon, indiquez le chemin absolu (chemin complet depuis la racine du disque) ou relatif du fichier après *run*, cf. exemple suivant.

Depuis le dossier TD1 (cf. ci-contre), `run titi.py` marche mais pour l'autre programme je dois taper *run../TD2/toto.py* pour que Python remonte d'un niveau (../) et trouve *TD2*.

Si Python ne trouve pas le fichier recherché vous aurez une erreur de type "Not Found". A vous de bien décrypter les erreurs. Quand un programmeur code (écrit du code), il passe plus de la moitié du temps à corriger ses erreurs. Il est important de comprendre les erreurs faites pour pouvoir les corriger rapidement. On dit souvent que les bugs n'existent pas et qu'il n'existe que des PEBCAK (*Problem Exists Between Chair And Keyboard*). Entrez et exécutez les lignes de code suivantes :

Erreurs de syntaxe : *Python* ne comprend pas ce que vous demandez, ce n'est pas correctement rédigé. Heureusement, il les détecte avant l'exécution du programme lui-même.

```
primt(26)
```

"primt" en est une. la coloration syntaxique n'a d'ailleurs pas eu lieu !

Erreurs de sémantique : ce que vous avez écrit est correct syntaxiquement, mais ne correspond pas à ce que vous voulez. Le programmer "tourne" mais vous n'obtenez pas le résultat escompté.

```
# on veut écrire les entiers pairs entre 0 et 10
a = 1
while a <= 10:
    print(a)                    # on écrit la valeur de a
    a = a + 2                   # on saute de 2 en 2
```

On obtient les entiers impairs ! Il fallait initialiser a à la valeur 0 et pas 1. Corrigez et relancez le programme.

Erreurs à l'exécution : plus difficile à déceler, n'apparaît qu'à l'exécution sans que l'on puisse trouver une erreur des deux types précédents. Entrez er exécutez

```
# Reste de la division de 314
# par les entiers compris entre -4 et 10
a = -4
```

```
while a <= 10:
    print("314 divisé par",a,"reste", 314 % a)
    a = a + 1
```

L'erreur se produit pendant l'exécution alors que tout va bien au départ... Une simple lecture de ce que dit Python à propos de cette erreur indique l'erreur commise...

3.3 Complément sur les variables

3.3.1 Typage des variables

Dans les exemples précédents, *a* et *b* sont des variables.

- Les variables en *Python* sont sensibles à la **casse** (majuscule, minuscule). oui Entrez et exécutez :

```
# variables et casse
L = 20
l = 30
print(L)
print(l)
```

- Le typage des variables est *dynamique*. Entrez et exécutez :

```
# typage des variables
note1 = 12
note2 = "zéro"
print(note1, note2)
```

L'appellation **typage dynamique** provient du fait qu'une variable n'a pas besoin d'être déclarée avant d'être utilisée. *Python* n'a pas besoin de savoir que *note1* est un entier avant de savoir que *note1* vaut 12.

Un des avantages est que le même nom de variable sert pour deux types consécutifs de variables : un entier, puis une chaîne de caractères. C'est la **réaffectation** d'une variable.

```
# réaffectation
var = 12
print(var)
var = "R2D2"
print(var)
```

- Essayez les affectations suivantes pour voir lesquelles sont valides

```
1main = 5
Main = 5
MAIN$ = 5
main_droite = 5
main1 = 5
continue = 5
```

Le nom d'une variable ne peut commencer par un chiffre, peut contenir les caractères a-z et A-Z et le seul symbole autorisé est le souligné _
D'autre part les mots clés du langage comme *for*, *while* ou encore *continue* ne sont pas autorisés

Pour la suite de votre travail en programmation, choisissez des noms de variables éloquents. Mieux vaut écrire

```
age = 16
```

que

```
a = 16
```

a n'est pas "parlant", alors que *age* fait surement référence à un âge. Si l'on fait juste un test rapide ce n'est pas capital, dans les autres cas pensez bien que vous serez amené à relire votre code, dans 24h, dans 1 semaine ...

3.3.2 Affectation

En écrivant

```
a = 2
```

on indique que la variable *a* va contenir la valeur 2.

De même lorsque l'on écrit

```
a = a + 1
```

Ce n'est pas au sens mathématique (a n'est sans doute pas égal à a+1), on exprime le fait que *a* va désormais contenir l'ancienne valeur de *a* plus 1.

Nous avons vu que nous pouvons effectuer une *affectation parallèle*

```
a, b = 3, 7
```

L'affectation parallèle se passe au niveau de la ligne même et les affectations qui y figurent ont lieu simultanément. C'est-à-dire...

Entrez et exécutez

```
a = 1                  # a prend la valeur 1
a, b = 0, a + 1        # a prend la valeur 0
                       et b prend la valeur a+1
print(a)
print(b)
```

Il existe aussi l'**affectation multiple**, par exemple

```
a = b = c = 7
```

A partir de l'exercice suivant, sauvegardez dans un même fichier texte (dans un éditeur de texte comme Notepad++ sous Windows, Sublime Text sous Mac ou Gedit sous Linux) vos réponses à chacun des exercices. Vous m'enverrez ce fichier pour jeudi 13 septembre (gael.lejeune@sorbonne-universite.fr).

Le format sera le suivant: une ligne avec "Exercice" + le numéro de l'exercice, vos réponses et/ou votre code pour l'exercice puis 2 lignes vides.

NB: le but est de vous faire progresser, ces exercices doivent vous permettre d'atteindre le niveau requis en Python. Ils ne seront pas notés mais serviront à adapter les prochains cours.

Exercice 3:
Ecrivez un programme qui calcule l'aire d'un rectangle connaissant sa largeur $l = 30$, longueur $L = 70$.

(on rappelle que l'aire est : $A = l \times L$). Essayez ensuite avec d'autres valeurs pour l, L.

La fonction ***input*** est faite pour améliorer l'interactivité d'un programme avec l'utilisateur. Elle permet d'affecter une valeur à la volée(récupérer la valeur au cours du programme) et de traiter cette valeur ensuite.

Attention, l'instruction *input()* renvoie une chaîne de caractères. Mais...

int convertit une chaîne de caractères en un entier.

Entrez et exécutez dans l'éditeur le programme suivant (il faut entrer un nombre)

```
    a = int(input())
# a contient le nombre saisi au clavier converti en entier
    print("Le double de", a, "est", 2*a)
```

On peut améliorer l'usage de la fonction *input* à l'aide d'un texte qui précède la saisie

```
a = int(input("Entre un nombre : "))
print("Le double de", a, "est", 2*a)
```

Pourquoi mettre la fonction **int** ? Ce qu'on entre au clavier est une chaîne de caractères, si l'on veut faire des opérations "mathématiques" il faut la convertir en nombre entier. Ceci est fait aussi ci-dessous

```
a = "3"      # c'est du texte comme dans "j'en veux 3 !"
a = int(a)   # a est converti en entier
```

Attention : les noms des variables ne peuvent être des nombres, ou du texte commençant par un chiffre. Les variables ne doivent contenir que des lettres [a - z], ou [A - Z] et des chiffres, le caractère souligné "_" est toléré. Pas non plus d'accent, ou de cédille dans les noms des variables.

Exercice 4:
Écrire un programme qui demande votre année de naissance et affiche votre âge sachant que l'on est en 2018 (à un an près, peu importe le mois).

Exercice 5:
De nombreuses formules mathématiques ont été mises au point pour essayer de déterminer le "poids idéal théorique" d'un individu en fonction de divers paramètres dont le principal est sa taille.

LA FORMULE DE BROCA

Poids idéal (en Kg) = Taille (en cm) -100

LA FORMULE DE CREFF

Poids idéal (en Kg) = (Taille (en cm) - 100 + Age (en années) /10) $\times 0,9$

Écrire un programme qui, pour un homme, demande sa taille et son âge et écrit en retour son poids idéal selon Broca et Creff.

Exercice 6:
En utilisant un minimum de lignes, écrire un programme qui traduit ce qui est exprimé ici en langage de réalisation

$a \leftarrow 5$

$b \leftarrow 7$

écrire a, suivi de b

échanger les valeurs contenues par a et b

écrire a, suivi de b

Exercice 7:

Écrivez un programme

qui affecte les valeurs 1, 2 et 3 à respectivement a, b et c

qui incrémente a de 1

qui effectue ensuite une permutation des trois variables ($a \leftarrow b$, $b \leftarrow c$ et $c \leftarrow a$)

qui incrémente a de 1

qui effectue ensuite une permutation des trois variables ($a \leftarrow c$, $b \leftarrow a$ et $c \leftarrow b$)

qui incrémente a de 1

qui affiche les valeurs de a, b et c dans cet ordre, l'une au-dessous de l'autre.

Exercice 8:

L'aire d'un trapèze, comme chacun sait, est $A = \dfrac{(GrandeBase + PetiteBase) \times Hauteur}{2}$ $= \dfrac{(DC + AB) \times H}{2}$. Écrivez un programme qui demande les valeurs des grande et petite bases, la hauteur, et qui affiche l'aire correspondante.

Exercice 9:

Écrivez et testez un programme qui affiche dans l'ordre pour les entiers 12 345, 978 345 et 123 456 789 le quotient entier suivi du reste dans la division par 37 (donc sur 3 lignes) en utilisant des variables adéquates.

Résumé

Opérations			**Usage**
	Addition	$+$	$5+2$
	Soustraction	$-$	$5-2$
	Multiplication	$*$	$5*2$
	Division	$/$	$5/2$
	Division entière	$//$	$5//2$
	Exponentiation	$**$	5**2
	Modulo	$\%$	5%2
Affectation			
	Directe	$=$	$a=5$
	Multiple	$... = ... = ...$	$a=b=3$
	Parallèle	$=$	$a,b,c=3,2,1$
Entrées-Sorties			
	Écrire à l'écran	print()	print(5)
	Entrer au clavier	input	a=input("texte")
Commentaire			
		#	# rien à dire

4 Les boucles, 1ère approche

4.1 Introduction

Les tâches répétitives sont lassantes pour l'humain. L'ordinateur les exécute très bien et facilement au travers de fonctionnalités logicielles ou bien simplement au travers d'un langage comme *Python*.

Dans un des exemples précédents, vous avez utilisé l'instruction *while*, qui signifie *Tant que*.

4.2 Utilisation

La syntaxe d'une boucle *while* est :

```
while condition:
    instruction 1      # les instructions suivantes sont
    instruction 2      # exécutées tant que la condition
    ...                # est vraie.
    instruction N
print("toto")
```

où "condition" est du style : $a < 5$ ou $a! = 5$ ou $2a > a**2$ ou ...

L'ensemble des instructions (1 à N) constituant ce que l'on appelle un **bloc d'instructions**. Cette notion est fondamentale, il s'agit en résumé d'une séquence d'actions ou de calculs (les instructions) qui seront exécutés à la suite les unes des autres.

Le début et la fin de ce bloc sont déterminés par l'indentation (elle est ajoutée automatiquement dans la plupart des éditeurs de code). Ici, toutes les instructions de 1 à N sont exécutées tant que la "condition" est vraie. par contre le print ne sera exécuté qu'après tous les **tours de boucle**.

Par exemple, dans le code ci-dessous

```
i = 0                   # la variable i vaut 0
while i < 5:            # tant que i est plus petit que 5
    print(i, "est plus petit que 5")
    i = i + 1           # on incrémente i
print("Fin")            # quand la boucle est finie, on écrit Fin
```

le texte "i est plus petit que 5" est écrit tant que $i < 5$ et à chaque répétition de la boucle, i est **incrémenté** ($i = i + 1$). Le mot "Fin" n'est écrit seulement quand le programme quitte la boucle et ne sera écrit donc qu'une seule fois car cette ligne ne fait pas partie du bloc de la boucle.

Entrez et exécutez les lignes suivantes

```
var = 1
lemax = 11
while var < lemax:      # bloc indenté exécuté
    print(var)          # tant que var < lemax
    var = var + 1
```

Remarques :

1. la variable *var* est incrémentée après son affichage (ligne 4 puis 5)
2. la boucle *while* est exécutée tant que *var* est inférieure strictement à *lemax*
3. la boucle sera exécutée $lemax - var = 11 - 1 = 10$ fois

Combien de fois est exécutée cette boucle 5 fois

```
a = 3
b = 23
while a <= b:
    a = a + 5       # piège ! a saute de 5 en 5
    print(a)
```

Attention aux boucles... qui ne s'arrêtent jamais...

A vos risques et périls, exécutez ces lignes (le programme risque de se bloquer !!!)

```
a = 3
b = 11
while a < b:
    print(a)
    print(b)
```

Pourquoi cette boucle est (dite) infinie ?

Astuce : pour mettre fin à ce genre de situation cliquez dans la barre d'outil sur le carré noir. . Cela ne fonctionne pas toujours très bien sous Windows... mais assez correctement sous les autres OS.

4.3 Opérateurs de comparaison

Dans le programme précédent, nous avons utiliser les **opérateurs de comparaison** $<$ et $<=$.

Il en existe d'autres dont voici la liste

Opérateurs	Significations
$<$	Strictement supérieur à
$>$	Strictement inférieur à
$<=$	Inférieur ou égal à
$>=$	Supérieur ou égal à
$==$	Égal à
$!=$	Différent de

Ces opérateurs se combinent avec des parenthèses et *booléens* et servent à traduire presque tous les tests désirés.

Remarquez que le signe $=$ est le l'**opérateur d'affectation**, l'**opérateur d'égalité** étant $==$ et son contraire $!=$.

Exécutez ces lignes

```
# fait la liste des entiers de 1 à n avec leur carre et leur cube
n = int(input("Donne un nombre : "))
compteur = 1
while compteur != n+1:
    print(compteur, compteur**2, compteur**3)
    compteur = compteur + 1
```

Exercice 10:
Modifier une ligne dans le programme précédent en utilisant l'opérateur ">" à la place de "! =" mais dans le but d'obtenir le même résultat.

L'instruction ***break*** permet de sortir d'une boucle. C'est-à-dire, si *Python* rencontre dans une boucle cette instruction, il sort de la boucle et continue immédiatement après la fin du bloc constituant cette boucle.

Exemple

```
a = 1
b = 2
while b == 2:
    print("b=", b)
    break
a = a + 1
print("a=", a)
```

Remarquez qu'à l'exécution, la boucle n'est exécutée qu'une fois, sinon on aurait une boucle infinie (la condition $b == 2$ est toujours vraie). Après le *print b*, l'exécution reprend à *a=a+1*.

4.4 Boucles et algorithmes

Les problèmes que l'on a besoin de résoudre automatiquement sont souvent ceux dont la solution implique des tâches répétitives, d'où l'utilisation de boucles dans de nombeux algorithmes.

Les exemples donnés ici font surtout appel à des opérations mathématiques mais ne vous en faites pas, nous passerons bientôt au traitement de données textuelles.

4.4.1 Algorithmes divers

Exercice 11:
L'instruction ***type(n)*** donne le type du nombre n (sera revu en détail plus tard). Entrez et exécutez le programme suivant

```
a, c = 1, 3.14    # 3.14 s'écrirait en langue naturelle 3,14
print(c, type(c))
while a < 8:
    b = 2**2**a   # 2 puissance 2 puissance a
    print(b, type(b))
    a = a + 1
```

Notez les types manipulés.

Exercice 12:
Écrivez un programme (avec une boucle) qui demande combien vous avez d'amis (nombre entier n) et qui demande l'un après l'autre leur prénom (avec input).

Exercice 13:
Écrivez un programme qui demande un entier n (n'oubliez pas de convertir la chaîne entrée en entier), qui affiche la table de n pour des multiplicateurs variant de 0 à 10. Par exemple, si on entre 6, le programme affiche exactement 0 6 12 18 24 30 36 42 48 54 60.

Exercice 14:
(plus difficile, pensez à utiliser *break*) On a déjà vu que le code suivant... (à essayer si vous avez oublié)

```
a = "lapin"
for lettre in a:
    print(lettre)
```

...qui permet de parcourir les lettres d'un mot. Écrire un programme qui demande un mot (entré au clavier) et qui affiche une lettre sur deux de ce mot en commençant par la première lettre (indice: vous pouvez utiliser un booléen ou ecnore l'opérateur modulo pour savoir où vous en êtes).

4.4.2 Algorithme et moyenne

Exercice 15:
Écrivez un programme qui demande 5 entiers (positifs) et qui affiche la moyenne des nombres entrés précédemment. Attention, il faut faire une boucle dans laquelle on va trouver l'instruction *input*.

Exercice 16:
On veut effectuer la moyenne de n nombres entiers positifs (il suffit de modifier le programme précédent). Écrivez un programme qui demande combien d'entiers sont concernés par cette moyenne, qui demande un à un ces entiers et qui affiche enfin la moyenne correspondante.

4.4.3 Algorithme et nombre premier

Rappel : un nombre entier naturel est premier s'il a exactement deux diviseurs distincts, 1 *et lui-même. Pour tester si un nombre est premier, il faut donc vérifier qu'il n'a aucun diviseur autres que* 1 *et lui-même.*

Observez, et testez le programme suivant (plus difficile):

```
# n est-il premier ?
n = int(input("Donne un nombre plus grand que 2 : "))
reste = 0
diviseur = 2
while diviseur < n:
    reste = n % diviseur
    while reste == 0:
        print(n,"est divisible par",diviseur)
        diviseur = n
        break
    diviseur = diviseur + 1
while reste != 0:
    print(n,"est premier")
    break
```

1. Combien de fois est exécutée la boucle commençant par **while reste != 0:**
2. Combien vaut *diviseur* à la fin du programme, si on a trouvé un diviseur du nombre entré
3. Combien de fois est exécutée la boucle commençant par **while reste == 0:**

Exercice 17:

Modifiez le programme précédent de telle façon qu'il affiche les diviseurs du nombre entré.

4.4.4 Algorithme et suite de nombres

Exercice 18:

Vous avez vu en cours la suite de *Fibonacci* définie par $u_0 = 1$, $u_1 = 1$ et ensuite $u_n = u_{n-1} + u_{n-2}$.

Ainsi $u_2 = u_1 + u_0 = 1 + 1 = 2$

$u_3 = u_2 + u_1 = 2 + 1 = 3$
$u_4 = u_3 + u_2 = 3 + 2 = 5$ et ainsi de suite...
Écrivez un programme qui affiche, après avoir demandé n, les n premiers termes de cette suite (vu en cours).

Exercice 19:
Conjecture de Syracuse. La suite de Syracuse d'un nombre entier N avec $N > 0$ est définie par récurrence de la façon suivante.

$$u_0 = N \text{ et pour tout } n \text{ entier naturel } u_{n+1} = \begin{cases} \dfrac{u_n}{2} & \text{si } u_n \text{ est pair} \\ 3u_n + 1 & \text{si } u_n \text{ est impair} \end{cases}$$

Pour n'importe quel nombre entier N, la suite des nombres obtenus affiche 1 à un moment donné (personne n'a encore su le démontrer). Écrivez un programme qui demande un nombre N et qui affiche tous les termes de la suite jusqu'à ce que 1 apparaisse.

RAPPEL : envoyez vos réponses aux exercices 3 et suivants pour jeudi 13 septembre (gael.lejeune@sorbonne-universite.fr).
Une deuxième série d'exercices, à rendre pour le lundi 17, vous sera envoyée prochainement.