Conditions et boucles en 

1. Structures conditionnelles

1.1. Comparateurs et opérateurs logiques

En Python, comme dans de nombreux langages, on utilise les comparateurs et opérateurs logiques suivants :

- `==` : comparateur d'égalité ;
- `!=` : comparateur de différence;
- `<`, `<=`, `>`, `>=` : comparateurs d'ordre ;
- `in`, `notin` : opérateurs d'inclusion.

Ces opérateurs renvoient une valeur de type **booléen** (donc `True` ou `False`):

Exemples

- Egalité :

```
>>> 5 == 2.0+3.0
True
```

- Inégalités :

```
>>> 4 != 2+3
True
```

- Ordre :

```
>>> 7 < 5
False
>>> 7 >= 7
True
```

- Appartenance :

```
>>> 'to' in 'Toto'
True
>>> 'TO' in 'Toto'
False
>>> 5 not in [0, 1, 2, 3, 4] # On utilise une structure de liste (entre crochets)
True
```

- `and` , `or`, `not` : opérateurs logiques `ET`, `OU` et `NON` ;

1.2. Opérations logiques

Il est souvent nécesaire de vérifier que plusieurs conditions soient vérifiées en même temps, ou bien qu'au moins une condition parmis plusieurs soit vérifiée. On utilisera aussi sovent la négation d'une condition.

Pour combiner ainsi pluseiurs conditions ensembles, on va utiliser la logique booléenne et les opérateurs `ET`, `ÒU` :

- `ET` qui renverra `Vrai` si les deux conditions sont simultanément `Vrai`, comme présenté dans la **table de vérité** ci-dessous :

ET	V	F
V	V	F
F	F	F

* `OU` qui renverra `Vrai` si au moins une des deux conditions est `Vrai`, comme présenté dans la **table de vérité** ci-dessous :

OU	V	F
V	V	V
F	V	F

En Python, il est aussi possible d'enchainer certains de ces opérateurs (mais ce n'et pas le cas dans d'autres langages :

```
>>> 2<7<=7 # qui est très très pratique, et peu présent dans les autres langages que Python...
```

```
True
```

```
3<4 and 5<6
```

```
True
```

```
3<2 and 5<6 #Essayez avec d'autres valeurs
```

```
False
```

```
3<2 or 5<6 #Essayez avec d'autres valeurs
```

```
True
```

```
not(5<6)
```

```
True
```

Python possède en plus un opérateur d'appartenance, qui n'est pas forcément présent dans tous les autres langages informatiques :

```
'TOTO'.lower()=='toto'
```

```
True
```

```
'Premier'<'panier'
```

```
True
```

```
'toto' and 'plage' in 'toto va à la plage'
```

```
True
```

```
'z' in 'aeiouy'
```

```
False
```

Et bien sur on peut combiner ces opérateurs :

```
'z' not in 'aeiouy'
```

```
True
```

1.3. b : Structures conditionnelles

En Python, on utilise pour les structures conditionnelles la syntaxe suivante :

```
if condition1 :#Le : est important !
    #bloc de une ou
    #plusieurs lignes
    #indentées ( avec la touche tabulation)
elif condition2 :
    #encore un autre bloc
elif condition3 :
    #etc
    #etc
    #etc
else :
    #et enfin un dernier bloc si aucune des conditions précédentes n'a été réalisée.
```

Les différents blocs d'instructions doivent être **correctement indentés** (c'est-à-dire correctement décalé s vers la droite). La règle de bonne conduite est d'utiliser la touche **tabulation**, ou à défaut de respecter la norme de 4 espaces par indentation. Il est réellement fondamental de respecter au maximum les indentations, celles-ci étant pour l'interpréteur Python le signal de déclenchement d'un bloc de code indépendant. Par exemple, le code ci-dessous renverra une erreur :

```
nb = int(input("Entrez un nombre entre 1 et 100 :"))
if nb<1 :
    print("Votre nombre est trop petit")
elif nb>100 :
    print("Votre nombre est trop grand")
else :
    print("Merci !")
print("FIN")
```

```
Entrez un nombre entre 1 et 100 :-2
Votre nombre est trop petit
FIN
```

On peut aussi noter que les mots clés `elif` et `else` sont optionnels.

```
if nb<1 :
    print("Votre nombre est trop petit !")
if nb>100 :
    print("Votre nombre est trop grand !")
if 1<nb<100 :
    print("Merci !")
```

```
Votre nombre est trop petit !
```

Les structures conditionnelles peuvent aussi être imbriquées, en indentant à plusieurs reprises :

```
nb=int(input("Entrez un nombre entier, positif ou négatif :"))
if nb>0 :
    print("Votre nombre est positif !")
    if nb%2==0 :
        print("Et c'est un multiple de 2 !")
    else :
        print("Et ce n'est pas un multiple de 2 !")
else :
    print("Votre nombre est négatif !")
```

```
Entrez un nombre entier, positif ou négatif :9
Votre nombre est positif !
Et ce n'est pas un multiple de 2 !
```

2. II : Boucles

2.1. a : Boucle non bornée : while

La boucle `while` (ou boucle *Tant que* en pseudo-code), possède la structure suivante en Python :

```
while condition :#Encore une fois, ne pas oublier le signe :
    #bloc de code
    #indenté
```

Le bloc de code situé sous l'instruction `while` sera exécuté **tant que la condition donnée sera vraie**, comme dans l'exemple donné ci-dessous :

```
nb= int(input("Donnez un nombre entier positif:"))
puissance = 0
while nb//2 >=1  :
    puissance = puissance +1
    nb=nb//2
print("Votre nombre est supérieur ou égal à 2 puissance {p} et inférieur à 2 puissance {q}".format(
    p=puissance,q=puissance+1))#A noter qu'ici le texte étant trop long, on a pu "sauter" à la ligne
                           #le saut de ligne étant compris entre les parenthèses de la méthode format
```

```
Donnez un nombre entier positif:30217
Votre nombre est supérieur ou égal à 2 puissance 14 et inférieur à 2 puissance 15
```

2.2. d : Boucle bornée : `for`

2.3. i : La boucle classique `for`

2.4. ii : la boucle `for` en Python

En Python, la boucle `for` est particulière. Elle est construite de manière à parcourir un **objet itérable**, c'est à dire un objet dont les éléments peuvent être repérés par un index, et donc qu'on peut parcourir du début à la fin. Ainsi, en Python, la boucle `for` peut parcourir : * une série de nombre entiers, grâce à la fonction **built-in** `range()` prenant trois valeurs comme argument, la borne inférieure (incluse),la borne supérieure (exclue) et le pas (optionnel) ; * une chaïne de caractère ; * une liste (à voir dans le prochain cours) ; * un fichier texte (qui sera vu dans le TP crée un site web interactif avec Cherrypy); * un fichier binaire (pour ceux qui veulent s'amuser, un TP sera prévu sur la Stéganographie); * etc.

La syntaxe de la boucle `for` est la suivante :

```
for variable in itérable :#Toujours ce signe ":"
    #Bloc d'instruction
```

On peut utiliser ainsi utiliser la boucle `for` comme sur les exemples suivants :

```
for i in range(1,100) :
    print(i)
```

```
texte ='Un texte quelconque'
for lettre in texte :
    if lettre in 'aeiouy':
        print("Tiens, j'ai vu la voyelle {}.".format(lettre))
```

3. III :Exercices

3.1. a : Exercices de base et logique

1. Ecrivez un programme qui permute et affiche les valeurs de trois variables A,B et C : A va dans B, B va dans C et C dans A

2. Ecrire un programme qui demande un nombre à l'utilisateur, puis affiche le carré de ce nombre.
3. Ecrire un programme qui demande l'heure qu'il est (un nombre pour les heures, un nombre pour les minutes et un pour les secondes). Cet algorithme indiquera en outre si l'heure donnée est valide.
4. Compléter le programme précédent afin que l'algorithme donne l'heure qu'il sera dans 10 minutes.
5. Ecrire un programme qui demande les coefficients a et b d'une fonction affine, et qui donne son nombre de racines et leur valeur éventuelles. Attention aux cas particuliers !
6. Etendre le programme précédent pour la résolution d'équations du type $ax+b=cx+d$, où a, b, c, d sont des entiers saisis par l'utilisateur.
7. Reprendre le problème n°5 pour un trinôme du second degré.

3.2. b : Exercices sur les boucles `for` et `while`

1. Ecrire un programme qui les 20 premiers nombres de la table de multiplication d'un entier choisi par l'utilisateur.
2. Ecrire un programme qui affiche les restes des 200 premiers entiers de la division euclidienne par 7.
3. Ecrire un programme qui affiche les restes des 300 premières puissances de 2 modulo 17. Que constate-t-on ?
4. Ecrire un programme qui affiche une suite de 12 nombres dont chaque terme soit égal au triple du nombre précédent.
5. Ecrire un programme qui demande un nombre entier entre 1 et 10 à l'utilisateur, et qui poursuit cette demande tant que l'utilisateur n'a pas exactement fait ce qui lui était demandé (on appelle ce type de programme **dumbproof**).