Gestion du flux d'instructions

I. Ordre des instructions, flux d'instructions

1. Ecrire et exécuter le script :

```
a, b = 3, 7
print('Le contenu des deux variables est :', a, b)
a = b
b = a
print(Le contenu des deux variables est :',a, b)
```

2. Quel est le résultat obtenu ?

3. Inverser les lignes 2 et 3. Quel est le résultat ?

4. L'ordre des instructions a-t-il un sens ? Justifier.

5. Que faut-il faire pour simplement inverser les valeurs de a et de b ? Le tester.

II. Sélection ou exécution conditionnelle avec les mots clés if, elif et else

Le mot clé **if** permet d'ajouter des blocs de codes qui seront exécutés selon les conditions voulues. Les blocs sont repérés par une indentation en python. C'est un décalage de 4 espaces par rapport à l'instruction précédente.

if utilise un opérateur de comparaison suivi de **:**. L'opérateur rend un résultat logique qui est soit **True**, soit **False** (vrai ou faux). S'il est vrai le bloc de code directement sous le if et repéré par une indentation est exécuté.

1. Essayer les trois lignes de code ci-dessous.

```
a = 150
if (a > 100):
    print("a depasse la centaine")
```

2. Essayer avec **a = 20**. Expliquer.

Le mot clé **else** suivie de **:** permet de spécifier un second bloc de code qui sera exécuté si l'opérateur de comparaison rend un résultat faux.

3. Taper le code et interpréter.

```
a = 20
if (a > 100):
    print("a depasse la centaine")
else:
    print("a ne depasse pas cent")
```

Le mot clé **elif** permet d'ajouter de nouveaux tests si le premier rend un résultat faux.

4. Essayer.

```
a = 5
if (a > 0) :
   print("a est positif")
elif (a < 0) :
   print("a est negatif")
else:
   print("a est nul")
```

5. Passer **a** à **-3** puis **0**. Refaire les mêmes essais sous Thonny ou Python Tutor. Expliquer.

6. Modifier le programme pour qu'il demande la valeur de **a** à l'utilisateur.

Tous les opérateurs de comparaison se trouvent à l'adresse :

http://docs.python.org/py3k/library/stdtypes.html#comparisons

III. La répétition (boucle) avec les mots clés for in

Les mots clés **for** et **in** permettent d'exécuter un bloc d'instructions plusieurs fois. On peut traduire **for in** par **pour dans**.

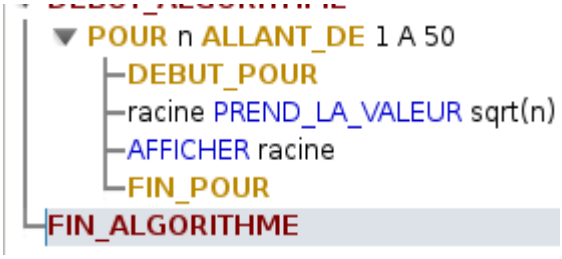
1. Tester cet exemple de boucle for.

```
print('je compte')
for i in range(10):
        print(i)
print("j'ai fini de compter")
```

2. Que peut-on remarquer ?

3. Que fait la fonction **range()** ? Traduire **for i in range(10)**.

4. Coder et exécuter l'algorithme ci-contre. Pour utiliser la fonction racine carrée, taper **import math** en début de code. La fonction est **math.sqrt()**.



Le mot clé **for** permet aussi de balayer des structures de données.

5. Essayer :

```
couleurs = ('violet', 'bleu', 'vert', 'jaune', 'orange', 'rouge')
for element in couleurs:
   print(element)
```

IV. La répétition (boucle) avec le mot clé while

Le mot clé **while** permet de répéter un bloc d'instructions tant que la condition associée est vraie. On peut traduire **while condition** par **tant que la condition est vraie**.

1. Essayer et comprendre les 4 lignes suivantes.

```
a = 0
while (a < 7): # n'oubliez pas le double point
    print(a, a < 7)
    a = a + 1 # n'oubliez pas l'indentation
    print(a, a < 7)
```

2. Changer la valeur de initiale de a par **4**, puis par **8**, puis par **7**, puis par **6**. Commenter les résultats.

Attention à l'opérateur de comparaison. S'il est toujours vrai, la boucle est sans fin, le programme ne s'arrête jamais et ne fait rien d'autre. Dans ce cas il est nécessaire de forcer l'arrêt du programme, par **CTRL + C**, le gestionnaire de tâches ou même parfois d'éteindre l'ordinateur !! Essayer :

```
n = 3
while (n < 5):
    print("hello !")
```

On lance une balle d'une hauteur initiale de 300 cm. On suppose qu'à chaque rebond, la balle perd 10 % de sa hauteur (la hauteur est donc multipliée par 0.9 à chaque rebond). On cherche à savoir le nombre de rebonds nécessaires pour que la hauteur de la balle soit inférieure ou égale à 10 cm.

3. A quelle condition doit être exécutée la boucle tant que de l'algorithme ci-dessous ?

4. Programmer l'algorithme et le tester.

5. Ajouter une ligne de code permettant d'afficher, à chaque étape de la boucle, les phrases du type :

hauteur après rebond n° 11 : 94.14317882700003

6. Conclure sur la condition à utiliser et la solution du problème.

7. Modifier le code pour que les données, hauteur initiale et hauteur finale, soient demandées à l'utilisateur.

8. Permettre à l'utilisateur de pouvoir entrer indéfiniment de nouvelles données et d'obtenir les résultats sans relancer le code. Qu'avez-vous fait ?

9. Permettre à l'utilisateur de pouvoir quitter le programme en tapant "QUIT". Qu'avez-vous fait ?

10. Comment coder une boucle **for** à l'aide de la boucle **while** ? Proposer un exemple et le tester. Détailler son exécution et sa terminaison.

11. Coder l'algorithme du III pour qu'il fonctionne avec une boucle **while**.

12. Ecrire un programme qui affiche un nombre, son carré et son cube, pour les nombres compris entre **0** et **12**. L'affichage se fera ligne par ligne, avec une phrase, chaque ligne contenant le nombre, son carré et son cube.

13. Tester ce code générant la suite qui fait la fierté de votre prof de Maths.

```
print('Voici les 10 premiers termes de la suite de Fibonacci :', end = '\n\n')
a, b, c = 1, 1, 1
while (c < 11):
    print(b, end = " ")
    a, b, c = b, a+b, c+1
print('\n')
print("C'est une suite de nombres dont chaque terme\n",
"est égal à la somme des deux précédents.", sep = "")
```

14. Se renseigner sur cette suite : **http://fr.wikipedia.org/wiki/Suite_de_Fibonacci**

15. Essayer les deux exemples de boucle sans fin ci-dessous. On peut les arrêts avec **CTRL-C**.

```
b = 1 # b est un entier
while (True): # Boucle sans fin
    b = b * 2
    print(b, ' ', type(b))

b = 1. # b est un flottant
while (True): # Boucle sans fin
    b = b * 2
    print(b, ' ', type(b))
```

16. Une manière d'éviter la boucle sans fin dans le cas des flottants est la suivante :

```
b = 1.
while (b != float('inf')):
    b = b * 2
    print(b, ' ', type(b))
```

17. En déduire si le type int et le type float ont une limite ou pas.