Fiche TD1

Ressources utiles :

- Je vous conseille de télécharger le livre de cours en version pdf et de l'avoir enregistré lors des TD pour ceux qui ne l'auraient pas fait voici le lien :

  http://www.xavierdupre.fr/mywiki/InitiationPython

- Le lien pour télécharger les td lors des séances

  http://www.xavierdupre.fr/enseignement/td_python/python_td_minute/index.html

Quelques compléments par rapport à vos questions :

- Comparaison d'un tuple avec un int, l'instruction 1 <(2,1) par exemple suivrait l'ordre lexicographique c'est-à-dire la même logique que pour la comparaison de tuple il comparerait 1 au premier élément du tuple. C'est ce que m'ont dit les autres groupes. Toutefois, dans les versions de python plus récentes (3 et +) la comparaison n'est plus applicable il retourne une erreur.

- Est-il possible de n'exécuter qu'une partie du code dans Scite ?

  L'execution se fait sur tout le code, il est possible pour ne pas avoir trop de résultat dans la fenêtre de rendu de mettre une partie du code en commentaire. Sinon, vous pouvez créer lors du TD plusieurs fichiers .py, l'un, poubelle pour faire des essais et une fois validé, vous l'incorporez proprement à un autre .py.

En bref les notions principales vues lors du TD :

Extrait du livre de cours(il est recommandé d'allez lire de la page **29 à 77**):

- Variables : Ce terme désigne le fait d'attribuer un nom ou identificateur à des informations : en les nommant, on peut manipuler ces informations beaucoup plus facilement et faire aussi varier leur valeur.

  A une variable on lui associe un nom, un type et une valeur.

  Ex :
  x=3 variable nommée x de type int valeur 3
  l=[1,3,2] variable nommée l de type list et dont la valeur est l'ensemble de la liste en elle-même.

- Opérations : On peut exécuter des opérations sur ces variables en faisant attention à ce que ces opérations aient un sens selon les types de variable utilisés. Par exemple, une division de liste n'a pas de signification par défaut.

- Boucles : Les boucles permettent de répéter une séquence d'instructions tant qu'une certaine condition est vérifiée.

```
for x in set :
        instruction 1
```

```
        ...
        instruction n

    while cond :
        instruction 1
        ...
        instruction n
```

Remarquez que les blocs d'instructions doivent être indentés (une tabulation de différence avec le mot clef while ou for de la boucle) pour être exécuter dans la boucle. Exemple :

Cas 1 :

```
while cond :
    instruction 1
    ...
    instruction n
```

les n instructions sont exécutées dans la boucle

Cas 2 :

```
while cond :
    instruction 1
    ...
    Instruction n-1
instruction n
```

les n-1 instructions sont exécutées dans la boucle la dernière est exécutée quoiqu'il arrive après la boucle donc est toujours exécutée !...ce qui est complétement différent ! par exemple si l'instruction n est Resultat= True.

Autre remarque quand dans une boucle on traite une liste, il faut faire attention à ce que l'on balaye la liste en restant dans les bornes inférieures et supérieures. J'ai pu voir par exemple :

```
For i in xrange(0, len(l)) :
    If liste[i-1]>liste[i] :
        …
```

Dans ce cas à la première itération I=0 et liste[-1] non défini.

```
For i in xrange(0, len(l)) :
    If liste[i]>liste[i+1] :
        …
```

Dans ce cas à la dernière itération I=len(l)-1 et liste[len(l)] non défini.

Une habitude possible pour éviter ces pièges est de travailler toujours à partir de i, i+1…généralement on commence à 0 et la borne inférieure ne pose pas de problème ensuite on ne vérifie le comportement que lors de la dernière itération pour voir si on ne va pas trop loin.

La bonne solution :

```
For i in xrange(0, len(l)-1) :
    If liste[i]>liste[i+1] :
        …
```

- Tests : Les tests permettent d'exécuter des instructions différentes selon la valeur d'une condition logique, par exemple :

```
if condition1 :
    instruction1
    instruction2
    ...
else :
    instruction3
    instruction4
    ...
```

Remarquez encore que l'indentation est nécessaire pour que s'exécute les blocs instruction1 instruction2 de chaque cas.

Algorithme de recherche dichotomique :

Nous reprendrons le prochain TP sur cet algorithme que je vous propose de préparer en complétant le programme à trous ci-dessous.

On souhaite définir une fonction rechercheD qui prend en argument un élément et une liste et renvoie la position de l'élément

Fonction : Une fonction est une partie d'un programme - ou sous-programme - qui fonctionne indépendamment du reste du programme. Elle reçoit une liste de paramètres et retourne un résultat. Le corps de la fonction désigne toute instruction du programme qui est exécutée si la fonction est appelée.

Exemple :

```
def somme(a,b) :
    return a+b


print somme(1,2)
k=2
l=3
j=somme(k,l)
print j
```

La fonction somme qui retourne la somme de ces 2 arguments peut être utilisée indépendamment par la suite. Quand un programme devient complexe il est utile et souvent plus facile de diviser le programme en fonction.

Signature de la fonction rechercheD dans le cas ou on travaille avec une liste d'entiers :

rechercheD : int, list ->int

```
def rechercheD(element, liste):
    Trouve = False
    debut = 0
    fin= len(liste)-1
    while not Trouve:
```

```
        index=…
        if liste[index]<element :
            …
        if liste[index]>element :
            …
        if liste[index]==element:
            …
    return ...
print rechercheD(5, [1,2,3,4,5,6])
```

Briques à replacer :

- index
- Trouve=True
- fin=index
- debut=index
- int((debut+fin)/2)

Notre fonction ne gère que le cas ou l'élément est effectivement dans la liste. S'il n'est pas présent la boucle ne se finit pas dans ce cas il nous manque des conditions d'arrêt supplémentaires.