Compte rendu du T.P. 3 (1ère partie)

1 Un premier savoir faire essentiel : manipuler les boucles `for`

a)
```
for i in range(100):
    print('python',end=" ") # l'argument optionnel avec le mot clef
    #end permet de définir ce que
    # fait le print après avoir écrit python. Ici on décide d'une espace
    # plutôt que le retour à la ligne standard
```

Seconde méthode

```
print("python "*100)
```

b)
```
for i in range(0,22,2):
    print(i)
## Méthode 2
for i in range(0,22):
    if i%2==0:
        print(i)
```

c) Première méthode en testant la parité à chaque tour ;

```
S=0
for i in range(101):
    if i%2==1:
        S=S+i
```

Seconde méthode avec un pas de 2 dans le range :

```
S=0
for i in range(1,101,2):
    S=S+i
```

d) Faire de même pour le calcul de $\prod_{i=1}^{100} i$.

```
P=1 # attention initialisation différente pour le produit vide.
for i in range(1,101):
    P=P*i
print(P)
```

2 La fabrication des listes : fabriquer une liste à l'aide d'une boucle

Supposons par exemple qu'on veuille fabriquer une liste `L` qui contient tous les i^2 pour $i \in [\![0, 100]\!]$ comment faire ?

a) L'Idée 0 qui ne MARCHE PAS :

On a vu en T.P. qu'on *ne peut pas* fabriquer une liste « à partir de rien » via les commandes $L[0] =$, $L[1] =$ etc...

Par exemple, on ne PEUT PAS définir $L = [2, 3, 1]$, en posant $L[0] = 2$, $L[1] = 3$, $L[2] = 1$. Bien sûr dans ce cas il suffit d'affecter `L=[2,3,1]`. Mais pour la liste L des carrés des entiers de 1 à 100, il faut une autre méthode !

b) L'idée 1 : avec l'*opérateur* + de concaténation :

construire cette liste L à l'aide d'une boucle, par concaténation successive à partir d'une liste vide.

```
L=[]
for i in range(101):
    L=L+[i**2]  # on ajoute à la liste L la liste à un élément [i**2] à chaque tour de boucle
print(L)
```

c) Idée 2 : où l'on rend possible l'idée 0

Il s'agit de *préfabriquer* une liste de longueur n qu'on peut ensuite modifier : Ainsi le code suivant fonctionne :

```
L=[0]*101
for i in range(101):
    L[i]=i**2
```

d) Idée 3 : avec le append : La fonction `append` a une syntaxe bien spécifique qui est celle des *méthodes* qui s'appliquent aux listes :

```
L=[]
for i in range(101):
    L.append(i**2)
print(L)
```

La différence entre le `+` et le `append` sera étudiée plus tard.

e) Idée 4 : En python, on peut définir des listes comme suit (pour cet exemple) :

```
L=[i**2 for i in range(101)]
```

3 Un deuxième savoir faire : écrire une *fonction*

a) Utiliser une fonction : si `L=[1,23,12,5]` en tapant `max(L)` dans le shell, on a la valeur `23` en retour.

c) Fabriquer une fonction :

```
def bissextile(annee):
    test=((annee%4==0) and ((annee%100!=0) or (annee%400==0))
    return test
```

4 Comment fabriquer une liste de nombres aléatoires pour nos tests

a)Savoir importer la fonction et consulter l'aide :

```
from random import randint
help(randint)
Help on method randint in module random:

randint(a, b) method of random.Random instance
    Return random integer in range [a, b], including both end points.
```

On comprend que : `randint(a,b)` fabrique un nombre entier (pseudo)-aléatoire dans $[\![a, b]\!]$.

b) Fabriquer alors une fonction `liste_alea` qui prend un entier `n` comme entrée et qui fabrique une liste de `n` nombres entiers aléatoires :

La méthode est la même qu'au § 2 pour fabriquer une liste de `n` éléments : (si `n` est bien défini) :

```
L=[]
for i in range(n):
    L.append(randint(1,1000))
```

Mais ici on demande en plus de *fabriquer une fonction*, donc on met ce code dans la définition d'une fonction, ce qui donne :

```
def liste_alea(n): # le  n est l'argument de la fonction,
# il sera rentré par l'utilisateur à chaque appel de la fonction
    L=[]
    for i in range(n):
        L.append(randint(1,1000))
    return L
```

5 Maximum dans une liste ou un tuple :

Le but est ici de *comprendre comment trouver le max. d'une liste en parcourant la liste.* C'est notre premier programme de manipulation sur les listes. Il est totalement hors-sujet d'y utiliser des outils plus forts sur les listes comme le tri. En fait dans tout ce qui suit, quand on demande de programmer une opération sur les listes, les seuls outils permis seront les commandes d'extractions $L[i]$ ou $L[i:j]$ et les outils que vous avez déjà développés.

a) Fonction `mon_max` :
Première méthode : en parcourant la liste à l'aide d'une boucle `for` sur les *indices*

```
def mon_max(L):
    "Renvoie le max. d'une liste ou d'un tuple"
    M=L[0]
    for i in range(len(L)):
        if L[i]>M:
            M=L[i]
    return M
```

Deuxième méthode : en parcourant à liste à l'aide d'une boucle `for` sur les *valeurs*

```
def mon_max(L):
    "Renvoie le max. d'une liste ou d'un tuple"
    M=L[0]
    for a in L: # le compteur a parcourt les valeurs dans la liste L
        if a>M:
            M=a
    return M
```

b) Amélioration

```
def max_mieux(L):
    "renvoie la valeur et un indice du max. d'une liste ou d'un tuple"
    M=L[0]
    i_max=0
    for i in range(len(L)):
        if L[i]>M:
            M=L[i]
            i_max=i
    return (M,i_max)
```

Cet algorithme d'obtention de max et de imax est **INCONTOURNABLE** : vous le retrouverez dans la plupart des épreuves écrites de concours sous une forme ou une autre...