TP 2 : Fonctions, listes, boucles imbriquées

1 Rappels sur la notion de fonction

— Syntaxe :

```
def f(args):
   [instructions]
```

— Les affectations dans une fonction sont locales, y compris les arguments.
— Si dans un appel `f(args)` une instruction de la forme `return resultat` est rencontrée, `resultat` est évalué, la fonction interrompue et l'évaluation de `f(args)` est celle de `resultat`.
— Si aucun `return` n'est rencontré, `f(args)` s'évalue en `None`.

2 Premières fonctions.

Les exercices qui suivent demandent de manipuler instructions conditionnelles et boucles à l'intérieur de fonctions. Il est important de tester vos codes, dans la console Python par exemple. Vérifiez que vous obtenez comme le texte.

Exercice 1. Sans utiliser le module `math` ou la fonction `abs` :

1. définir une fonction `absolue` prenant en entrée un entier et retournant sa valeur absolue (on ne demande pas de vérifier que le paramètre passé à la fonction est un entier).

```
>>> a=absolue(-4)
>>> b=absolue(2)
>>> a+b
6
```

2. définir une fonction `fact` prenant en entrée un entier positif et retournant sa factorielle définie par $n! = \prod_{i=1}^{n} i$.

```
>>> a=fact(4)
>>> a
24
```

Rappel : `range(k,m)` fournit les entiers de $[\![k, m-1]\!]$.

Exercice 2. *Utilisation d'une fonction dans une autre fonction.*

1. Écrire une fonction `max2(a,b)` prenant en paramètre deux entiers et retournant le maximum des deux. *Remarque :* cette fonction existe déja en Python sous le nom `max`, on ne l'utilisera pas ici, on comparera simplement `a` et `b`.

2. En déduire une fonction `max3(a,b,c)` retournant le maximum de 3 entiers, et utilisant `max2`. *Remarque :* la fonction `max` de Python peut également être utilisée à la place de `max3`.

```
>>> max3(-2, -4, -3)
-2
```

3. Écrire une fonction `max_liste(L)` prenant en entrée une liste non vide et retournant son maximum. *Remarque :* la fonction `max` de Python fonctionne également sur les listes.

```
>>> max_liste([1, 2, 8, 5, 4, 2])
8
```

3 Fonctions basiques sur les listes

Exercice 3. *Création de listes.*

1. Écrire une fonction `carres(n)` qui renvoie la liste des n premiers carrés (de 0^2 à $(n-1)^2$).

2. Écrire une fonction `non_div_3(n)` qui renvoie la liste des entiers de $[\![0, n-1]\!]$ qui ne sont pas divisibles par 3.

3. Écrire une fonction `addition(L1,L2)` qui prend en paramètres deux listes supposées de même taille, et qui renvoie la liste obtenue par addition terme à terme des éléments de `L1` et `L2`. Par exemple :

```
>>> addition([0,2,5],[1,4,8])
[1,6,13]
```

Attention : votre fonction ne doit pas modifier les listes passées en paramètre, mais créer une nouvelle liste.

4 Algorithmes de tri et boucles imbriquées

Dans la suite, on écrit des algorithmes de tri. Pour les tester, vous pouvez générer automatiquement des listes aléatoires de tailles conséquentes avec la fonction `randrange` du module `random` :

```
>>> from random import randrange
>>> L=[randrange(0,100) for i in range(20)]
>>> L
[6, 84, 14, 6, 86, 66, 17, 63, 84, 82, 96, 47, 58, 56, 71, 98, 36, 93, 63, 93]
```

L'appel `randrange(a,b)` renvoie un entier aléatoire entre a inclus et b exclu.

Exercice 4. *Tri par sélection du minimum.* On dit qu'une liste est triée si ses éléments sont dans l'ordre croissant. On donne ci-dessous en pseudo-code un algorithme permettant de trier une liste. Écrire une fonction `tri_selection(L)`

Algorithme 1 : Le tri par sélection du minimum

Entrée : une liste L
Effet : la liste est modifiée, pour être triée dans l'ordre croissant
pour i *variant entre 0 et longueur*$(L)-2$ *inclus* **faire**
  $m \leftarrow$ indice du plus petit élément de la liste situé entre l'indice i et la fin de la liste;
  si $m \neq i$ **alors**
    Échanger les éléments aux positions i et m de L.

permettant de trier la liste `L` :

— on repérera l'indice m avec une boucle `for`.

— pour réaliser l'échange de `L[i]` et `L[m]`, on utilisera une variable intermédiaire.

```
>>> L=[6, 84, 14, 6, 86, 66, 17, 63, 84, 82, 96, 47, 58, 56, 71, 98, 36, 93, 63, 93]
>>> tri_selection(L) #la fonction ne renvoie rien
>>> L
[6, 6, 14, 17, 36, 47, 56, 58, 63, 63, 66, 71, 82, 84, 84, 86, 93, 93, 96, 98]
```

Votre fonction ne renverra rien, mais modifiera la liste passée en paramètre. Remarque : la *méthode* `sort` de Python permet de trier une liste, elle s'utilise comme suit : `L.sort()`

Exercice 5. *Tri à bulles.* ***1.*** Écrire une fonction `passage(L)` faisant un passage de gauche à droite sur la liste `L`, et échangeant deux éléments successifs s'ils ne sont pas dans l'ordre croissant. Par exemple :

```
>>> L=[6, 84, 14, 6, 86, 66, 17, 63, 84, 82, 96, 47, 58, 56, 71, 98, 36, 93, 63, 93]
>>> passage(L) ; L
[6, 14, 6, 84, 66, 17, 63, 84, 82, 86, 47, 58, 56, 71, 96, 36, 93, 63, 93, 98]
```

Attention, si n est la taille de la liste, il y a $n-1$ couples d'indices successifs dans `L`.

2. Écrire une fonction effectuant $n-1$ passages sur la liste, appelée `tri_bulles(L)`. Vérifier qu'elle trie la liste.

```
>>> L = [6, 84, 14, 6, 86, 66, 17, 63, 84, 82, 96, 47, 58, 56, 71, 98, 36, 93, 63, 93]
>>> tri_bulles(L)
>>> L
[6, 6, 14, 17, 36, 47, 56, 58, 63, 63, 66, 71, 82, 84, 84, 86, 93, 93, 96, 98]
```

Remarque : il est possible d'améliorer un peu le tri pour qu'il soit plus efficace.

Exercice 6. La fonction `print` affiche son argument avec, par défaut, un retour à la ligne. On peut modifier ce comportement en écrivant `print(truc, end='')` (où `truc` est une chaîne de caractères et `''` la chaîne de caractères vide), il n'y aura pas de retour à la ligne. En utilisant ce comportement, écrire des fonctions réalisant des carrés, triangles, etc... comme suit, les appels à `print` seront seulement `print('*',end='')` (affichant une étoile), `print('o',end='')` affichant un rond, et `print()` passant à la ligne (le but est d'écrire des doubles boucles) :

```
*****
*****
*****
*****
*****
```

```
*
**
***
****
*****
```

```
*****
****
***
**
*
```

```
*****
*ooo*
*ooo*
*ooo*
*****
```

Ces figures sont le résultat d'appels `carre(5)`, `triangle(5)`, `triangle2(5)` et `carre2(5)`, écrire les fonctions correspondantes.