0.1 Fonctions, variables, paramètres

0.1.1 Variable globales et locales

On distingue différents espaces : un espace global dans lequel le programme et ses variables globales sont définies. Une variable globale est donc définie pour l'ensemble du programme. Chaque fonction possède un espace local distinct des autres espaces: il contient les paramètres qui lui sont passés et les variables qui y sont définies. Une fonction ne peut pas modifier par affectation une variable extérieure à son espace (local). La variable locale x n'existe plus après l'appel de la fonction. On pourra modifier une variable globale à l'aide du mot clé `global`. L'utilisation des variables globales si elle est possible est cependant fortement déconseillée.

Exemple 1

```
gravitation = 9.81
def sur_la_lune(masse):
    """
    renvoie le poids sur la lune
    """
    global gravitation
    gravitation = 1.625
    poids= masse * 1.625
    return poids
```

Nous avons bien:

```
print(gravitation)
>>>9.81
```

mais:

```
sur_la_lune(10)
print(gravitation)
>>> 1.625
```

Exemple 2: Que donne la séquence suivante ?

```
x = 1
def incremente(x):
    x = x+1
    return x
incremente(x)
print(x)
```

Notez que si le si le script suivant livre un résultat identique, il est préférable au précedent pour des questions de lisibilité:

```
x = 1
def incremente(t):
    t = t+1
    return t
incremente(x)
```

0.1.2 Cas d'une liste passée en paramètre

Que donne la séquence suivante ?

```
>>> t = [1, 2, 3]
>>> s = t
>>> t[1] = 5
>>> print(t)
[1, 5, 3]
```

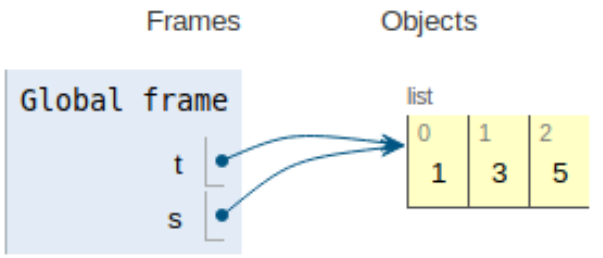
```
>>> print(s)
[1, 5, 3]
```

Le résultat peut paraitre surprenant, cela tient au fait que les listes sont des objets modifiables (ou **mutables** mais c'est anglicisme), si une liste est passée en paramètre, c'est par adresse. Autrement dit, elle va être modifiée par la fonction et rester modifiée en dehors de cette fonction.

Les listes sont des pointeurs ou alias: elles pointent vers une adresse mémoire. Cela permet notamment d'utiliser des objets dont on ne connaît pas la taille.

Note: Lorsqu'une fonction modifie des éléments du programme en dehors de son environnement, cela s'appelle l'**effet de bord**.

```
s est égal à  [1, 3, 5]
t est égal à  [1, 3, 5]
```



Les listes s et t pointent vers la même adresse

On utilisera le module deepcopy ou la syntaxe s = t[:] dans le cas de liste d'objets non mutables pour obtenir une "vraie"copie de t.

```
>>> t = [1, 2, 3]
>>> s = t[:]
>>> t[1] = 5
>>> print(t)
[1, 5, 3]
>>> print(s)
[1, 2, 3]
```

```
>>> from copy import *
>>> t = [1, 2, 3]
>>> s = deepcopy(t)
>>> t[1] = 5
>>> print(t)
[1, 5, 3]
>>> print(s)
[1, 2, 3]
```

Conséquences directes:

Programme principal: Dans une fonction, toute modification sur une liste existante se répercute dans le programme principal.

Valeur de retour: Il est donc inutile de renvoyer la liste.

Variable globale: Il est inutile de préciser dans une fonction qu'une liste est une variable globale.

0.1.3 Exercices

1. Ecrire une fonction `maximum(L)` prenant une liste de nombres en paramètre et retournant le plus grand nombre de la liste.
2. Ecrire une fonction `loto` simulant le tirage du loto: 5 boules parmi les numéros 1 à 49 et un complémentaire entre 1 et 10. Créer une liste de 49 boules, elle sera passée en paramètre à la fonction `loto`.
3. Ecrire une fonction `tableau_aleatoire(n , a, b)` qui retourne un tableau de taille n contenant des entiers tirés au hasard entre a et b.
4. Ecrire une fonction `echange(tab, i, j)` qui échange dans le tableau `tab` les éléments d'indice i et j.
5. Reprendre la fonction `maximum` pour qu'elle retourne un tuple constitué du maximum et de son indice dans la liste.
6. Gérer avec une assertion le cas où la liste est vide (programmation défensive).

0.2 Complément sur les listes

0.2.1 Liste en compréhension

Un ensemble peut se définir:

- en donnant ses éléments E = 2, 4, 6, 8, 10
- en se servant de certaines de ses propriétés: E = $\{x \in \mathbb{N} | x \leqslant 10$ et x est pair$\}$ Cela s'appelle définir en compréhension.

Voici une liste des nombres pairs, définie à partir de la liste des 10 premiers entiers non nuls en compréhension:

```
>>> L = [x for  x in range (1, 11) if x%2 == 0]
>>> L
[2, 4, 6, 8, 10]
```

En voici la syntaxe:
`L = [item for item in liste if condition]`.

1. Ecrire une fonction `multiplier`(liste, k) prenant en paramètre une liste et un entier. Elle retourne une nouvelle liste dont chaque élément est celui de la liste passée en paramètre multiplié par cet entier.
2. Ecrire une fonction `puissance`(liste, k) prenant en paramètre une liste et un entier. Elle retourne une nouvelle liste dont chaque élément est celui de la liste passée en paramètre élevé à la puissance de cet entier.
3. Ecrire une fonction `non_zero`(liste). Elle retourne une nouvelle liste ne contenant aucun zéro.