# Série 1

# Prise en main

## 1. Éléments nécessaires pour la réalisation des exercices

Afin de réaliser les différents activités de cette série de TP, il est indispensable de d'avoir :

— Connaissance de certains éléments du langage Python :
  — Syntaxe de base du langage : indentation, affectations, structures de contrôle (`if` , `while` , `for` , ...), fonctions, etc.
— Des notions générales en algorithmique.
— A noter que la série présente uniquement les fonctions de bases. Elle ne peut, en aucun cas, se substituer de la documentation officielle du langage.

**Eléments à apprendre : (exercice 1) les listes en Python**

→ Une liste est définie par la syntaxe : `[...]`

◇ Exemple : `my_list=[1,2,5,6]` , `z=[]`

◇ Une liste peut contenir n'importe quel type de données (même hétérogènes). Elle peut même contenir des listes : `[1,2,[3,4],4.5]`

→ L'indexation se fait par `list[index]` :

◇ Exemple : `my_list[2]` renvoie 5

◇ Mise à jour : `my_list[1]=9`

→ On peut sélectionner un *slice* (une partie) d'une liste :

◇ Exemple : `my_list[0:2]` renvoie [1,2]

◇ Exemple : `my_list[1:]` renvoie [2,5,6]

◇ Exemple : `my_list[:3]` renvoie [1,2,5]

→ Opérations sur les listes :

◇ Longueur : `len(m_list)` renvoie 4

◇ Le nombre d'occurrence d'un élément : `my_list.count(e)`

◇ Concaténation : `[1,2]+[4,6]` donne [1,2,4,6]

◇ Répéter une liste : `[1,2]*3` donne [1,2,1,2,1,2]

◇ Vérifier si un élément figure dans une liste : `x in my_list` (résultat booléen)

◇ La fonction range : `list(range(0,5))` construit la liste `[0,1,2,3,4]` .

→ Programmer avec les listes :

◇ Parcourir les éléments d'une liste : `for e in my_list:print(e)`

◇ Parcourir les éléments par leurs indices :
```
for i in range(len(my_list)):print(my_list[i])
```

◇ Parcourir les indices et les éléments d'une liste :
```
for i,e in enumerate(my_list):print(i,"=>",e)
```

→ Compréhension : générer une liste d'éléments vérifiant un critère donné :

◇ Générer les nombres paires de my_list : `[ e for e my_list if e % 2==0 ]`

◇ Renvoyer tous les nombres multiples de 5 inférieurs à 100 : `[ e for e in range(101) if e % 5==0 ]`

**Exercice 1**

Écrire les fonctions suivantes (il faut écrire un programme testant les fonctions construites) :

**Activité 1**

Soustraire deux listes (garder les éléments de la première liste qui ne figurent pas dans la deuxième.

**En-tête**

```
def substract(L1,L2)
```

**Test**

```
substract([5,8,12,1,13,17],[6,7,12,5,17])
```

**Activité 2**

Répéter les éléments d'une liste en fonction d'une autre liste (exemple : L1=[6,1] et L2=[2,4], le résultat est : [[6,6],[1,1,1,1]]

**En-tête**

```
def repeat_by_list(L1,L2)
```

**Test**

```
repeat_by_list([6,1],[2,4])
```

**Activité 3**

Réarranger les éléments d'une liste : premier élément, dernier élément, deuxième élément, avant-dernier élément, troisième élément, ...

**En-tête**

```
def symmetric_browse(L)
```

**Test**

```
symmetric_browse([1,3,5,7,9,8,6,4,2])
```

**Eléments à apprendre : (exercice 2) les chaînes de caractères en Python**

→ La plupart des fonctions des listes sont applicables aux chaînes.

→ Une chaîne de caractères est une séquence de caractères (placées entre deux " ou deux '). On peut également avoir une chaînes sur plusieurs lignes avec les délimiteurs """.

◇ `my_str="abcdef"`

→ Une chaîne est une liste immuable

◇ `L'affectation s[2]="v"` provoque une erreur

→ Opérations sur les chaînes :

◇ La fonction `chr(...)` donne le caractère dont le code ASCII ou unicode est passé comme paramètre (exemple `chr(1585)`

◇ La fonction `ord(...)` donne le code ASCII ou unicode du caractère passé comme paramètre (exemple : `ord("a")` )

◇ Longueur : `len(my_str)` renvoie 5

◇ Le nombre d'occurrence d'un caractère : `my_str.count(e)`

◇ Concaténation : `"ab"+"cd"` donne "abcd"

◇ Répéter une chaîne : `"abc"*3` donne "abcabcabc"

◇ Vérifier si un élément figure dans une liste avec le test `x in my_list` (le résultat est booléen). Ceci fonctionne pour un caractère ou pour une chaîne (test de sous-chaînes).

◇ Recherche le premier (resp. le dernier) indice d'une sous-chaîne `my_str.find(sub_str)` (resp. `my_str.rfind(...)` )

◇ Joindre les éléments d'une liste pour en faire une chaîne `my_str.join(my_list)` (il faut que la éléments de la liste soient des chaînes de caractères).

◇ Formater une chaînes selon un format (option 1) : `str_format(a1,a2,...)` . Exemple : `"{}+{}={}".format(3,4,3+4)` .

◇ Formater une chaînes selon un format (option 2, disponible à partir de Python 3.6) : `f"..."` . Exemple : `f" {3}+{4}={3+4} "`.

◇ Diviser une chaînes de caractères par rapport à un séparateur : `my_str.split(sub)`

◇ Remplacer toutes les occurrences d'une chaînes par une autre chaîne : `my_str.replace(sub1,sub2)`

**Exercice 2**

Écrire les fonctions suivantes (il faut écrire un programme testant les fonctions construites) :

**Activité 1**

La fonction crange permettant de générer une chaîne contenant tous les caractères entre deux caractères (par exemple, `crange(["a","c"],["0","2"])` donne `"abc012"` ).

**En-tête**

```
def crange(*cinterv)
```

**Test**

```
crange(["a","z"],["A","Z"],["0","9"])
```

**Activité 2**

Améliorer la fonction replace pour qu'elle prenne deux listes de chaînes de caractères [s1,s2,...], [t1,t2,...]. La fonction change s1 par t1, ensuite s2 par t2, etc.

**En-tête**

```
def replace_many(s,L1,L2)
```

**Test**

```
replace_many("aabc",["a","b","c"],["b","a","d"])
```

**Activité 3**

Implémenter une fonction replace conditionnelle. La chaîne sub1 est remplacée par sub3 uniquement si sub1 est suivie par sub2, sinon elle est remplacée par sub4.

**En-tête**

```
def conditional_replace(s,sub1,sub2,sub3,sub4)
```

**Test**

```
conditional_replace("abc=ded","=","f","=e","=f")
```