Listes

I/ Le type list

Python dispose d'un type ***list***. Une ***liste*** sert à stocker plusieurs éléments ensembles, qui ne sont pas nécessairement tous du même type. Une liste est délimitée par des crochets, et ses éléments sont séparés par des virgules. Pour connaître la longueur d'une liste, c'est-à-dire le nombre d'éléments que contient cette liste, on utilise la commande `len` .

II/ Créer une liste

1. En explicitant la liste

On peut créer une liste en écrivant explicitement tous ses éléments. La syntaxe est la suivante :

```
L = [ 2 , 3 , True , 10.5 ]
```

2. Par ajouts successifs

On peut créer une liste en créant initialement une liste vide puis en la remplissant progressivement à l'aide d'une boucle for ou while. Par exemple :

```
def creation_liste(n) :
    L = [ ]
    for i in range(n):
        L.append(i)
    return L
```

L'instruction creation_liste(5) renvoie ainsi la liste [0 , 1 , 2 , 3 , 4] .

3. En compréhension

On peut créer une liste ***en compréhension***, c'est-à-dire en utilisant une formule générale pour ses éléments. Par exemple, pour créer la liste L = [2 , 4 , 6 , 8 , 10], on peut utiliser l'instruction :

```
L = [ 2*k for k in range(1,6) ]
```

III/ Elements d'une liste

1. Accès

Les éléments d'une liste L sont numérotées de 0 à len(L) - 1. La position d'un élément est appelée sont ***indice***.

Pour accéder à l'élément d'indice `i` dans la liste L, la commande est : L[i] .

2. Ajout

On peut ajouter un élément à la fin d'une liste avec la commande append .

Pour ajouter l'élément `a` à la fin de la liste L, la commande est : L.append(a) .

3. Suppression

On peut supprimer le dernier élément d'une liste L avec la commande pop .

La commande L.pop() supprime le dernier élément de la liste L et le renvoie en résultat.

4. Exemples

Exemple 1

```
1 def premier(L):
2     return L[0]
```

Exemple 2

Ecrire une fonction Python **dernier** qui prend en argument une liste non vide L et qui renvoie son dernier élément, *sans modifier la liste* L.

Exemple 3

Ecrire une fonction Python **remplace** qui prend en argument une liste non vide L et un réel `a` et qui remplace son premier élément par `a`.

IV/ Parcours de listes

Il est souvent utile de pouvoir parcourir une liste.

1. Parcours sur les indices

```
def ajout(L,a):
    n = len(L)
    for k in range(0,n):
        L[k] = L[k] + a
    return L
```

2. Parcours sur les éléments

```
def presence(L,a):
    for e in L:
        if e == a
            return True
    return False
```

V/ Opérations sur les listes

1. Concaténation

Une opération souvent utile lorsqu'on manipule des listes est la ***concaténation***. On dit qu'on concatène deux listes lorsqu'on les regroupe en une seule. La concaténation s'effectue via le symbole $\boxed{+}$.

Ainsi, si l'on exécute l'instruction :

```
1 L = [1 , 2 , 3 ] + [4 , 5 ]
```

la liste L prend la valeur [1 , 2 , 3 , 4 , 5] .

2. Répétition

On peut également créer une liste en répétant plusieurs fois une autre liste. Pour cela, la syntaxe est :

```
1 liste_depart = [0 , 1 , 2]
2 liste_arrivee = 3*liste_depart
```

Si l'on exécute ces instructions, la liste liste_arrivee prend la valeur [0 , 1 , 2 , 0 , 1 , 2 , 0 , 1 , 2] .

3. Copie

Pour créer une copie d'une liste on utilise la syntaxe suivante :

```
1 liste_copie = liste_origine.copy()
```