Récapitulatif, premières commandes avec les listes :

si `L` est une liste définie dans Python
- on obtient le premier terme de `L` avec `L[0]`
- `len(L)` permet de connaitre la taille de liste (*length*)
- on peut compléter la liste avec `append` : par exemple `L.append(2)` ajoute un `2` à la fin de la liste
- si `L` contient des valeurs numériques, on peut utiliser les outils `min, max, sum, sort()` (ce dernier pour trier les valeurs).
- on peut obtenir toutes les valeurs de liste jusqu'à une certaine valeur : par exemple `L[:8]` donne toutes les valeurs de la liste jusqu'à `L[7]` (attention `L[8]` est exclu). De manière analogue `L[3:]` donne toutes les valeurs à partir de `L[3]` (inclus cette fois), jusqu'à la fin de la liste.

Liste et `for` : on peut créer une liste en utilisant une formule et en incluant une boucle `for` pour choisir les valeurs pour lesquelles on souhaite obtenir les valeurs. Par exemple les carrés des entiers de 1 à 100 sont obtenus avec : `carres=[i**2 for i in range(1,101)]`

Attention avec cette commande `L[0]` contient 1^2 et plus généralement `L[i]` contient $(i+1)^2$

Echauffement - exploration des listes

1 avec des polypèdes

Tester les commandes suivantes et préciser leur action :

```
animaux = ["girafe", "tigre", "singe", "souris"]
# créée une liste dénommée animaux contenant des objets sous forme de texte : "
    girafe", "tigre", "singe", "souris"
animaux[0]
# renvoie le terme d'indice 0 de la liste, i.e. le premier, soit "girafe" ici
animaux[1]
# renvoie le terme d'indice 1 de la liste, soit "tigre" ici
animaux[4]
# message d'erreur car il n'y pas de terme d'indice 4 dans la liste
animaux*2
# duplique la liste
animaux.append("chat")
# modifie la liste initiale en ajoutant l'élément "chat" à la fin
animaux
# on voit la modification précédente
animaux + ["canari"]
# affiche la liste et l'élément "canari"
animaux
# on comprend que l'ajout précédent était ponctuel, la liste animaux ne contient
    pas "canari"
animaux= animaux + ["canari"]
# modifie la liste animaux en la remplaçant par l'ancienne à laquelle on ajoute "
    canari"
animaux
# pour voir que la liste animaux contient désormaix "canari"
animaux[1:3]
# renvoie tous les éléments de la liste dont l'indice est compris entre 1 et 3 (
    exclus), donc "tigre", "singe"
```

```
animaux[2:]
# renvoie tous les éléments de la liste de l'indice 2 jusqu'à la fin
```

2 avec des nombres

Tester les commandes suivantes et préciser leur action :

```
tailles_bbh = [1.78,1.76,1.76,1.68,1.65,1.72,1.70,1.68,1.78,1.83,
1.83,1.86,1.77,1.83,1.80,1.78,1.78,1.77]
# créée la liste dont le nom est tailles_bbh et contenant les nombres ci-dessus
min(tailles_bbh)
# renvoie le minimum de la liste soit 1.65
max(tailles_bbh)
# renvoie le maximum de la liste soit 1.86
len(tailles_bbh)
# renvoie la longueur de la liste, i.e. le nombre d'éléments, soit 18
sum(tailles_bbh)
# renvoie la somme des nombres de la liste
tailles_bbh.sort()
" modifie la liste initiale en la triant par ordre croissant
```

Ecrire une commande qui calcule la moyenne des tailles ?

Pour accéder aux outils comme `mean` (moyenne), il faut charger une librairie statistique. A défaut, on peut simplement exploiter les commandes vues plus haut permettant d'obtenir la somme des valeurs et la longueur de liste : `sum(tailles_ bbh)/len(tailles_ bbh)`

3 avec la fonction range

Tester la commande suivante et préciser son action :

```
list(range(1,10))
# créée la liste contenant les entiers de 1 à 9
```

De manière analogue, construire la liste des entiers pairs de 0 à 100

Il suffit simplement d'ajouter le paramètre permettant de définir le pas qui vaut alors 2 :

```
list(range(0,101,2))
```

Exercices

Exercice 1 - définir une liste avec *for*

Le but de cet exercice est d'écrire des commandes en utilisant les listes incluant `for`.
Ecrire :

1. la liste de taille 20 qui comporte les 20 premiers entiers non-nuls au carré.

   Comme vu en introduction plus haut, on utilise la syntaxe synthétique :

   ```
   [i**2 for i in range(1,21)]
   ```

2. la liste de taille 10 : $((-1)^1, (-1)^2 \ldots, (-1)^{10})$

   De manière analogue on exécute la commande :

   ```
   [(-1)**i for i in range(1,11)]
   ```

3. une fonction Python qui prend en entrée un entier naturel n et qui renvoie la valeur de $\sum_{k=1}^{n} k^3$

   Prenons l'exemple de $\sum_{k=1}^{10} k^3$: on commence avec créer la liste des cubes des entiers de 1 à 10 avec la commande [i**3 for i in range(1,11)] puis on calcule leur somme avec sum([i**3 for i in range(1,11)]).
   On va maintenant créer une fonction qui permet de choisir la valeur du dernier indice de la somme (attention au $n+1$ pour arrêter la somme à n) :

```
def somme(n):
  return [i**3 for i in range(1,n+1)]
```

4. la liste des nombres $(-1)^n n^2$, pour n allant de 1 à 1000, puis leur somme.

   Comme précédemment, on prend la liste des valeurs pour n variant entre 1 et 1000 et on en fait la somme avec :

```
sum([(-1)**i*i**2 for i in range(1,1001)])
```

   On peut également en faire une fonction.

Exercice 2 - pour s'amuser

1. Constituer une liste semaine contenant les 7 jours de la semaine.

   `semaine=["lundi","mardi","mercredi","jeudi","vendredi","samedi","dimanche"]`

2. A partir de cette liste, récupérer les 5 premiers jours de la semaine d'une part, et ceux du week-end d'autre part ?

   Pour le début on prend tous les éléments de la liste jusqu'au 5ème, i.e. [5] (exclus) `debut_semaine=semaine[:5]`
   pour la fin, à partir du 5ème : `weekend=semaine[5:]`

3. Inverser les jours de la semaine en une commande.

   On peut inverser l'ordre des éléments d'une liste avec la commande `.reverse()`, ici : `semaine.reverse()`
   Cette commande mofidie la liste.