LISTES

I) Présentation des listes

En informatique, une liste est une structure ordonnée permettant de regrouper des données placées les unes à la suite des autres de manière à pouvoir y accéder librement. On utilise préférablement des listes contenant des éléments de même type (int, float, string, etc).

a) Création d'une liste

En Python, pour créer une liste vide, on peut écrire :

```
liste1 = [] #première possibilité
liste2 = list()   #seconde possibilité
```

Pour créer une liste avec des valeurs, on écrira :

```
liste = [15, 23, -9 ,2]
```

On peut également créer des listes « automatiquement » :

```
liste1 = range(6)       #la liste contient [0,1,2,3,4,5]
liste2 = range(5,12)    #la liste contient [5,6,7,8,9,10,11]
liste3 = range(5,12,2)  #la liste contient [5,7,9,11]
liste4 = [i for i in range(0,4)]    #la liste contient [0,1,2,3]
```

b) Accès à une liste

Pour accéder (lire/modifier) à un élément dans la liste, il faut connaître sa position (**indice**), la numérotation commençant à 0 : on indique l'indice de l'élément en le plaçant entre crochets […]

```
liste = [15, 23, -9 ,2]
liste[1] = 22                 #on modifie le 2ème élément de la liste
variableIndice1 = liste[1]    #on accède au 2ème élément de la liste
```

c) Longueur d'une liste

La fonction len() permet de connaître la longueur d'une liste.

```
liste = [15, 23, -9 ,2]
longueur = len(liste)   #longueur vaut 4
```

d) Parcours d'une liste

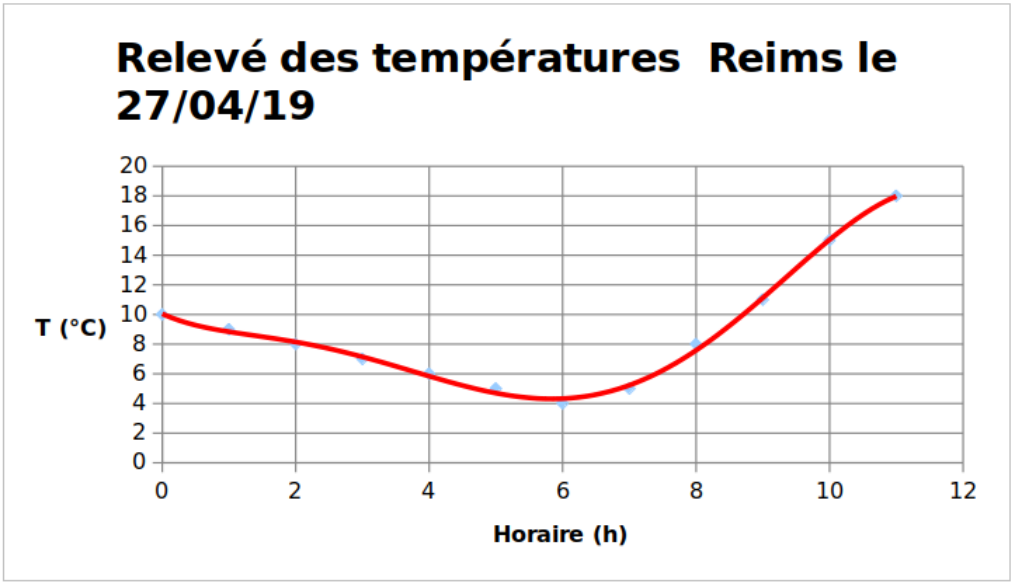
On utilise souvent une boucle for pour parcourir une liste.

```
liste = [15, 23, -9 , 2]
for i in range(0,len(liste)):
    print(liste[i])
```

Une autre syntaxe peut être utilisée.

```
liste = [15, 23, -9 , 2]
for element in liste:
    print(element)
```

Ci-dessus un graphique permettant de visualiser la température à Reims le 27/04/19.



Afin de disposer des valeurs de température pour chaque heure, il serait intéressant de relever celles-ci et les disposer dans le tableau suivant :

Horaire (h)	0	1	2	3	4	5	6	7	8	9	10	11
Température (°C)	10	9	8	7	6	5	4	5	8	11	15	18

Exercice n°1
- Créer un fichier « Listes.py ».
- Créer la liste correspondant au tableau de mesures et afficher cette liste.
- Afficher le premier élément de cette liste.
- Afficher le dernier élément de cette liste (on demande d'utiliser la fonction « len » de façon à ce que l'affichage s'adapte à la longueur de la liste).
- Demander à l'utilisateur de saisir une heure et afficher la température correspondante.

Exercice n°2
- A la suite du code précédent, afficher toutes les températures de la liste les unes sous les autres.
- Afficher toutes les températures de la liste sous la forme :
  « A 1 h, la température était de 9°Celsius »
- Afficher toutes les températures de la liste entre 5 heures et 9 heures sous la forme :
  « A 5 h, la température était de 5°Celsius »

Exercice n°3
- A partir de la même liste de températures, rechercher (boucle for) et afficher la température la plus faible et la plus forte.
- Afficher également les heures correspondantes.
- Rechercher et afficher la moyenne des températures.

II) Quelques fonctions supplémentaires pour utiliser les listes

a) min(liste) → retourne le plus petit élément de la liste.

Exemple :

```
liste = [1.5,6.3,8.1,4.0,6,1.3,8.1,2]
minimum = min(liste)
print (minimum)

Affichage : 1.3
```

b) max(liste) → retourne le plus grand élément de la liste.

Exemple :

```
liste = [1.5,6.3,8.1,4.0,6,1.3,8.1,2]
maximum = max(liste)
print(maximum)

Affichage : 8.1
```

c) sorted(liste) → retourne une copie superficielle triée par ordre croissant de la liste .

Exemple :

```
liste = [1.5,6.3,8.1,4.0,6,1.3,8.1,2]
liste=sorted(liste)
print(liste)

Affichage : [1.3, 1.5, 2, 4.0, 6, 6.3, 8.1, 8.1]
```

d) sum(liste) → retourne la somme des éléments de la liste.

Exemple :

```
liste = [1.5,6.3,8.1,4.0,6,1.3,8.1,2]
somme=sum(liste)
print(somme)

Affichage : 37.3
```

III) Quelques méthodes des listes

Les méthodes sont appelées avec l'opérateur . de la liste. On n'en présente ici que quelques unes.

a) liste.append(element) → ajoute l'élément à la fin de la liste

Exemple :

```
liste = [1.5,6.3,8.1,4.0,6,1.3,8.1,2]
liste.append(2.8)
print(liste)

Affichage : [1.5, 6.3, 8.1, 4.0, 6, 1.3, 8.1, 2, 2.8]
```

b) liste.insert(indice,element) → insère l'élément dans la liste à la position indice.

Exemple :

```
liste = [1.5,6.3,8.1,4.0,6,1.3,8.1,2]
liste.insert(3,2.8)
print(liste)

Affichage : [1.5, 6.3, 8.1, 2.8, 4.0, 6, 1.3, 8.1, 2]
```

c) liste.clear() → supprime tous les éléments de la liste.

Exemple :

```
liste = [1.5,6.3,8.1,4.0,6,1.3,8.1,2]
```

```
liste.clear()
print(liste)

Affichage : []
```

d) liste.remove() : supprime l'élément de la liste.
Exemple :

```
liste = [1.5, 6.3, 8.1, 4.0, 6, 1.3, 8.1, 2]
liste.remove(8.1)
print(liste)

Affichage : [1.5, 6.3, 4.0, 6, 1.3, 8.1, 2]
```

e) liste.reverse() : inverse l'ordre des éléments de la liste.
Exemple :

```
liste = [1.5, 6.3, 8.1, 4.0, 6, 1.3, 8.1, 2]
liste.reverse()
print(liste)

Affichage : [2, 8.1, 1.3, 6, 4.0, 8.1, 6.3, 1.5]
```

f) liste.copy(liste) → retourne une copie de la liste dans une autre liste.
Exemple :

```
liste = [1.5, 6.3, 8.1, 4.0, 6, 1.3, 8.1, 2]
liste2 = liste.copy()
liste[0] = 4
print (liste)
print(liste2)

Affichage :   [4, 6.3, 8.1, 4.0, 6, 1.3, 8.1, 2]
              [1.5, 6.3, 8.1, 4.0, 6, 1.3, 8.1, 2]
```

Cette méthode est importante. Dans l'exemple ci-dessus, si on avait écrit `liste2=liste` sans utiliser la méthode copy(), modifier une des 2 listes revient à modifier l'autre (on travaille en fait avec des références sur la même liste). L'affichage final aurait produit le même résultat pour les 2 listes.

Exercice n°4
- Utiliser la même liste que précédemment (températures à Reims le 27/4).
- Afficher cette liste.
- Ajouter une température de 20° à 12 heures.
- Modifier la température de 6h à 2°C.
- Afficher cette liste.
- Afficher la liste dans l'ordre croissant des températures (les heures ne sont pas demandées).
- Afficher la température maximum, la température minimum et la température moyenne.

IV) Utiliser les listes dans des fonctions

a) Rappel

Lorsqu'on passe des paramètres (type primitif comme int ou float) à une fonction, c'est la valeur et non la variable qui est transmise.

Exemple :

```
      def test(a:int,b:int):  #a et b sont des variables différentes
            a=8               #de celles du programme principal
            b=9         #il est souvent preferable de les nommer differemment
            print("a dans la fonction vaut : ",a)
            print("b dans la fonction vaut : ",b)
            return

      a=1
      b=2
      print("a avant l'appel de la fonction vaut : ",a)
      print("b avant l'appel de la fonction vaut : ",b)
      test(a,b)
      print("a apres l'appel de la fonction vaut : ",a)
      print("b apres l'appel de la fonction vaut : ",b)

Affichage :
a avant l'appel de la fonction vaut : 1
b avant l'appel de la fonction vaut : 2
a dans la fonction vaut : 8
b dans la fonction vaut : 9
a apres l'appel de la fonction vaut : 1
b apres l'appel de la fonction vaut : 2
```

b) Référence

Lorsqu'on passe une liste à une fonction, on passe la référence de cette liste. On peut donc modifier le contenu de celle-ci dans la fonction.

Exemple :

```
      def test(lst:list): ->None
            lst[0]=0
            return None

      liste = [1.5,6.3,8.1,4.0,6,1.3,8.1,2]
      test(liste)
      print(liste)

Affichage :[0, 6.3, 8.1, 4.0, 6, 1.3, 8.1, 2]
```

Comme vu précédemment (méthode copy), le signe « = » entre 2 listes ne copie pas les valeurs d'une liste dans une autre mais copie uniquement la référence à cette liste.

Exemple :

```
      def test(lst:list):
            lst[0]=0
            return

      liste = [1.5,6.3,8.1,4.0,6,1.3,8.1,2]
      liste2=liste
      test(liste)
      print (liste)
      print (liste2)    #on obtient le même affichage

Affichage :[0, 6.3, 8.1, 4.0, 6, 1.3, 8.1, 2]
           [0, 6.3, 8.1, 4.0, 6, 1.3, 8.1, 2]
```

c) fonction retournant une liste

Une fonction peut retourner une liste, comme tout autre type.

Exemple :

```
def creation()->list:
      lst=[1.5,6.3,8.1,4.0,6,1.3,8.1,2]
      return lst

liste = creation()
print(liste)
```

```
Affichage :[1.5, 6.3, 8.1, 4.0, 6, 1.3, 8.1, 2]
```

Exercice n°5

- Créer un nouveau fichier « listesEtFonctions »
- Coder la fonction « creationAleatoire » qui retourne une liste d'éléments entiers aléatoires dont le nombre est passé en argument. Utiliser la fonction randint pour générer des valeurs entre 0 et 100.
- Tester avec le nombre 20.

Exercice n°6

- Coder la fonction « affichage » qui affiche tous les éléments d'une liste passée en argument les uns sous les autres.
- Tester en utilisant le code de l'exercice précédent.

Exercice n°7

- Copier la fonction « creationFromCSV » ci-dessous.

```
import csv
def creationFromCSV(nomDeFichier:str)->list:
      lst=[]
      fichier = open(nomDeFichier,"r")    #ouverture du fichier en lecture
      lecteur = csv.reader(fichier) #création d'un lecteur de fichier csv
      for ligne in lecteur:   #pour chaque ligne lue, on a l'indice et la valeur
            val=float(ligne[1])           #ligne[1] correspond à la valeur
            lst.append(val)               #ajout de la valeur dans la liste
      fichier.close()                     #fermeture du fichier
      return lst
```

- Tester cette fonction avec le fichier "relevesTemp_26_4.csv" que vous copierez dans votre répertoire de travail. Ce dernier représente les heures et les températures enregistrées à Troyes le 26 avril.

Exercice n°8

- Coder un programme qui affiche toutes les températures enregistrées à Troyes le 26 avril et affiche la moyenne de la journée.

Exercice n°9 ***(pour les + motivés)***

- Modifier le programme pour permettre à un utilisateur de saisir une heure (par exemple : 8) et de modifier la température relevée correspondante (par exemple : 9.2).
- Afficher ensuite la nouvelle moyenne.

On demande de créer une fonction de saisie de l'heure, une autre de saisie de la nouvelle température et une dernière de modification de la liste.