I Les types `list`, `str`, `tuple`

- Une liste (type : `list`) est une collection ordonnée et modifiable d'éléments de types variés.

Une liste est définie par des crochets, les différents éléments de la liste sont séparés par des virgules.

Exemple : `L = [7, 1.3, "Bonjour !", True]`
définit la liste `L` comportant 4 éléments : un entier, un flottant, une chaîne, un booléen.

- Une chaîne de caractères (type : `str`) est une suite ordonnée et non modifiable de caractères.

Une chaîne est définie par des guillemets (ou apostrophes).
Exemple : `c = "Ceci est une chaine"`

- Un n-uplet (type : `tuple`) est une succession ordonnée et non modifiable d'objets de types variés, séparés par une virgule. Il est d'usage d'utiliser des parenthèses lors de l'écriture d'un n-uplet.
Un n-uplet ne contenant qu'un seul objet doit finir par une virgule.
Exemple : `t = (2, 3.5, "BCPST")`

II Indexation

Chaque élément d'une liste, d'une chaîne ou d'un n-uplet possède un indice (*index*).
La taille (longueur, nombre d'éléments) est donnée par la fonction `len` (*length*) :

```
>>> len(L)                    Out : 4
>>> len(c)                    Out : 19
>>> len(t)                    Out : 3
```

Les indices des éléments constituant une liste, une chaîne ou d'un n-uplet sont comptés de la gauche vers la droite à partir de 0, ou de la droite vers la gauche en décroissant à partir de -1.
On appelle l'élément d'indice i en faisant suivre le nom de la variable de crochets contenant l'entier i :

```
>>> L[0]                      Out : 7
>>> L[1]                      Out : 1.3
>>> L[-2]                     Out : 'Bonjour !'
```

Les liste, chaîne, tuple vides sont de longueur zéro : `[]`, `""`, `()`.

III Extraction, tranches

On peut extraire une sous-liste, sous-chaîne ou sous-tuple (on parle de *tranches*) :

- `>>> c[p:n]` renvoie la sous-chaîne des indices p (inclus) à n (exclu).
- `>>> c[p:n:h]` fait de même avec un pas de h.
- `>>> c[:]` renvoie une copie de `c`.
- `>>> c[p:]` renvoie la sous-chaîne à partir de l'indice p inclus, jusqu'à la fin.
- `>>> c[:n]` renvoie la sous-chaîne du début jusqu'à l'indice n exclu.
- `>>> c[::h]` renvoie les caractères de h en h à partir du premier.
- `>>> c[p::h]`, `>>> c[:n:h]` et `>>> c[p:n:h]` renvoient les caractères de h en h à partir de celui d'indice p (inclus), ou jusqu'à celui d'indice n (exclu), ou les deux.

IV Concaténation

L'opérateur + concatène (met bout-à-bout) deux listes, chaînes ou tuples :

```
>>> [1,2,3] + [5,6,7]                  Out : [1,2,3,5,6,7]
>>> "Bonjour" + "tout le monde"        Out : "Bonjourtout le monde"
>>> (1,2,3) + (5,6,7)                  Out : (1,2,3,5,6,7)
```

V Réplication

L'opérateur $*$ copie et concatène à elle-même une liste, chaîne ou tuple :

```
>>> [1,2,3] * 2                        Out : [1,2,3,1,2,3]
>>> "Bonjour" * 3                      Out : "BonjourBonjourBonjour"
```

VI Transtypage

Les chaînes, listes et n-uplets peuvent se convertir les un(e)s en les autres.

```
>>> tuple([0, 1, 4])                   Out : (0, 1, 4)
>>> str([0, 1, 4])                     Out : '[0, 1, 4]'
>>> list("abcd")                       Out : ['a', 'b', 'c', 'd']
```

VII Recherche d'un élément

L'opérateur `in` renvoie un booléen si l'élément cherché appartient ou pas à la liste, chaîne ou tuple :

```
>>> 7 in L                                  Out : True
>>> 'aim' in c                              Out : False
```

La méthode `.index` renvoie l'indice du premier élément demandé :

```
>>> L.index(1.3)                            Out : 1
>>> t.index("BCPST")                        Out : 2
```

et un message d'erreur si l'élément est absent : `not in list`, `not in tuple`, `substring not found`.

VIII Comparaisons, tests

Les opérateurs `==`, `<`, `>`, `<=`, `>=` comparent deux listes, chaînes ou tuples en commençant par les premiers éléments (tant que c'est possible), selon le principe de l'ordre lexicographique (du dictionnaire).

```
>>> [1,2,3] < [3,0]                         Out : True
>>> "Pythie" > "Python"                     Out : False
```

Les caractères d'une chaîne sont classés selon leur code ASCII (voir plus loin).

IX Objets itérables

Les listes, chaînes et tuples sont *itérables* : on peut parcourir leurs éléments du premier au dernier.

```
>>> [elt for elt in c if elt > 'n']         Out : ['s', 't', 'u']
```

X Commandes spécifiques aux listes

- Les listes sont modifiables. On peut affecter une valeur à un élément, ou à une tranche.
- La méthode `.append` permet d'ajouter un nouvel élément à la fin d'une liste.
- La méthode `.insert` permet d'insérer un nouvel élément dans une liste, à une position donnée.
- La méthode `.pop` permet de supprimer un élément d'indice donné d'une liste, et renvoie cet élément.

Par défaut, la méthode `.pop` supprime le dernier élément de la liste.

- La méthode `reverse` renverse l'ordre des éléments de la liste.
- La fonction `del` supprime un élément d'indice donné, ou une tranche.
- La méthode `.sort` classe les éléments de la liste `L` du plus petit au plus grand lorsque c'est possible, et renvoie un message d'erreur sinon.

Exemples avec la liste : `L = [7, 1.3, 'Bonjour !', True]`

```
>>> L[0] = 8
>>> L                          Out : [8, 1.3, 'Bonjour !', True]
>>> L[1:3] = [2,5]
>>> L                          Out : [8, 2, 5, True]
>>> L.append(6)
>>> L                          Out : [8, 2, 5, True, 6]
>>> L.insert(2,9)
>>> L                          Out : [8, 2, 9, 5, True, 6]
>>> L.pop(3)                   Out : 5
>>> L                          Out : [8, 2, 9, True, 6]
>>> L.pop()                    Out : 6
>>> L                          Out : [8, 2, 9, True]
>>> del L[1]
>>> L                          Out : [8, 9, True]
>>> L.reverse()
>>> L                          Out : [True, 9, 8]
>>> L.sort()
>>> L                          Out : [True, 8, 9]
```

- On peut définir une liste **'en compréhension'**, en parcourant un objet itérable.

```
>>> L2 = [x**2 for x in range(6)]
>>> L2                                      Out : [0, 1, 4, 9, 16, 25]
>>> L3 = [x**2 for x in range(10) if x%2 == 0]
>>> L3                                      Out : [0, 4, 16, 36, 64]
```

• Les fonctions `min, max, sum` s'appliquent aux listes dont les éléments sont comparables, ou sommables.

```
>>> min([5,3,8,2,7])        Out : 2
>>> sum([5,3,8,2,7])        Out : 25
```

XI Problème du clonage de listes

La commande d'affectation `M = L` définit un "clône" de la liste `L`. Si on modifie `M`, alors la liste `L` risque d'être modifiée en même temps. Pour circonvenir ce problème, utiliser : `M = list(L)` ou `M = L[:]`

```
>>> M = L
>>> M[0] = False
>>> L                                    Out : [False, 8, 9]
```

XII Commandes spécifiques aux chaînes

• Caractères spéciaux : `\n` : retour à la ligne (*new line*), `\t` : tabulation (*tab*), `\\` : antislash (backslash), `\'` : apostrophe, `\ "` : guillemets.

`>>> b = 'L\'altitude'` définit la variable `b` de type `str` égale à `"L'altitude"`

• Chaque caractère d'une chaîne correspond à un code ASCII (American Standard Code for Information Interchange). On peut connaître le code ASCII d'un caractère en utilisant la fonction :

```
>>> ord("T")                             Out : 84
```

et inversement associer à un entier le caractère correspondant :

```
>>> chr(255)                             Out : 'ÿ'
```

• La méthode `replace` remplace des sous-chaînes :

```
>>> c.replace("e", "i")                  Out : Cici ist uni chaini
```

• La fonction `eval` permet d'extraire et d'interpréter le contenu d'une chaîne.

```
>>> eval("print(\"Hello\")")             Out : Hello
```

• La méthode `split` permet de séparer une chaîne de caractères selon les espaces qu'elle contient, et renvoie une liste contenant des sous-chaînes :

```
>>> "Bonjour tout le monde.".split()     Out : ['Bonjour', 'tout', 'le', 'monde.']
```

XIII Travail à réaliser

Exercice 1 : Manipulations de listes

1. Écrire une fonction `somme` qui reçoit une liste `L` de flottants, et renvoie la somme de ses éléments. Si la liste est vide, la fonction doit renvoyer 0. **Ne pas utiliser la fonction native** `sum`.
2. Écrire une fonction `produit` qui reçoit une liste `L` de flottants, et renvoie le produit de ses éléments. Que doit renvoyer cette fonction si la liste est vide ?
3. Écrire une fonction `maxi` qui reçoit une liste `L` de flottants, et qui renvoie le plus grand d'entre eux. **Ne pas utiliser la fonction native** `max`.
4. Écrire une fonction renvoyant l'indice (la place) du maximum de la liste donnée en argument.
5. Écrire une fonction qui reçoit une liste `L` d'entiers, et qui renvoie le nombre d'entiers pairs dans `L`.
6. Écrire une fonction `compte(L,x)` renvoyant le nombre de fois où l'élément `x` apparaît dans la liste `L`.
7. Écrire une fonction `sousListe` qui reçoit une liste `L` et qui renvoie la liste `M` constituée des éléments de la liste `L` pris un sur deux (en prenant le premier).
8. Écrire une fonction `incremente(L)` renvoyant une liste formée par les éléments de `L` augmentés de 1.
9. Écrire une fonction `derive` qui reçoit une liste `L` de flottants, et qui renvoie la liste des différences entre deux termes consécutifs de la liste `L` .
10. Écrire une fonction `positif` qui reçoit une liste `L` de flottants, et qui renvoie la liste obtenue à partir de `L` en ne conservant que les flottants positifs ou nuls.
11. Écrire une fonction renvoyant les deux plus grands éléments d'une liste.

Exercice 2 : Écrire une fonction `asterix` qui reçoit une chaîne de caractères, la recopie en insérant des astérisques entre les caractères, et renvoie le résultat.

Par exemple, `>>> asterix('gaston')` devra renvoyer `'g*a*s*t*o*n'`.

Exercice 3 : Écrire une fonction `palindrome` qui teste si une chaîne de caractères est un palindrome, et renvoie un booléen. Par exemple, `>>> palindrome('kayak')` renverra `True`.

Exercice 4 : Écrire une fonction `compte(mot,lettre)` qui reçoit une chaîne de caractères `mot` et un caractère `lettre`, et renvoie le nombre d'occurences du caractère `lettre` dans la chaîne `mot`.

Exercice 5 : Écrire une fonction `separation` qui reçoit en argument une chaîne de caractères constituée de chiffres et d'un caractère de séparation `&`, et qui renvoie une liste contenant les nombres entiers qu'on peut lire dans cette chaîne entre les caractères de séparation.

Par exemple, `>>> separation("12&8&56")` devra renvoyer la liste `[12,8,56]`.

Exercice 6 : Écrire une fonction `inverse` qui reçoit en arguments un entier naturel n non nul, et un nombre de décimales d ; et qui renvoie la chaîne constituée par les d premières décimales de $\frac{1}{n}$.

Par exemple, `>>> inverse(7,20)` devra renvoyer la chaîne `"14285714285714285714"`.