▶ Exercice 1 – Introduction

En informatique une « structure de données » est une façon de ranger et d'ordonner des objets. Pour une structure de données « complexe » il se pose tout un tas de question : comment accéder aux éléments ? Comment ajouter des éléments ? Comment compter le nombre d'éléments ? Etc.

Le type « liste » de Python est extrêmement utilisé. De manière basique, il s'agit de regrouper des éléments, séparés par des virgules et encadrés par des crochets. Il est à noter que les différents éléments d'une liste peuvent être de types différents.

1. Saisir et interpréter (en appuyant sur F5 dans Spyder) le script suivant :

```
L = [5, 2, -1.3, True]
print(L[0])
print(L[1])
print(L[2])
print(L[3])
print(L[4])
print(L)
x = len(L)
print(x)
```

2. Combien y a-t-il d'éléments dans la liste `L` ?

3. Quels sont les éléments de la liste `L` et quels sont leurs types respectifs ?

4. Que fait la fonction `len` ?

5. Quels sont les numéros (index ou indice) de chaque élément dans la liste ? Que remarque-t-on qui est assez étonnant ?

6. Ajouter l'instructions `print(L[4])`. Que se passe-t-il ?

▶ Exercice 2 – Construction progressive d'une liste

1. Saisir et interpréter (en appuyant sur F5 dans Spyder) le script suivant :

```
L = []
print(L)
for i in range(0,10):
    L.append(i)

print(L)
```

Expliquer les deux affichages obtenus.

2. En s'inspirant de l'exemple précédent, écrire un script construisant une liste de taille 10 et contenant les **carrés** des entiers de 1 à 10.

3. Écrire une fonction `carres(n)` prenant en argument un entier naturel non nul n et renvoyant la liste des **carrés** des entiers de 1 à n. Tester (avec un affichage) cette fonction avec différentes valeurs de n, par exemple $n = 10$, $n = 25$ et $n = 100$.

4. On souhaite écrire une fonction `impairs(n)` prenant en argument un entier naturel non nul et renvoyant la liste des entiers impairs entre 1 et n.
Attention – L'entier n lui même peut être pair ou impair. Pour identifier et gérer les deux cas on pourra utiliser `%` et `//` qui permettant d'obtenir le reste et le quotient d'une division euclidienne d'entier.

a) Pour bien comprendre les deux opérateurs `%` et `//`, saisir et interpréter le script suivant :

```
a = 15
r = a % 2
print(r)
q = a//2
print(q)

b = 20
r = b % 2
print(r)
q = b//2
print(q)
```

b) Écrire la fonction `impairs(n)` dans laquelle on traitera deux cas suivant que n est pair ou impair (à l'aide d'une structure `if ... else`), comme vu dans le TP n° 1.

c) Tester votre fonction `impairs(n)` avec quelques valeurs de n, par exemple $n = 10$, $n = 15$ et $n = 20$.

▶ Exercice 3 – Listes en compréhension

Cette manière de définir une liste est très « pythonique » (comprendre spécifique au langage Python). Ce qui est très intéressant est la ressemblance avec ce que l'on peut faire en mathématiques.

1. Saisir, interpréter et comprendre le script suivant :

```
L = [k**2 for k in range(1,11)]
print(L)
L = [k**3 for k in range(1,11)]
print(L)
```

2. Écrire une fonction `doubles(n)` prenant en argument un entier naturel non nul et renvoyant la liste des doubles de tous les entiers entre 1 et n.
Attention – On demande que cette liste soit construite « en compréhension ».
Tester votre fonction avec quelques valeurs de n par exemple $n = 10$, $n = 15$ et $n = 20$.

3. On souhaite écrire une fonction `factorielles(n)` prenant en argument un entier naturel n et renvoyant la liste des factorielles de tous les entiers entre 0 et n.
Rappels – En mathématiques, la factorielle de n est défini par :

$$n! = \begin{cases} 1 & \text{si } n = 0 \\ 1 \times 2 \times 3 \times \cdots \times n & \text{sinon} \end{cases}$$

a) La fonction factorielle ne figure pas dans le noyau central de Python. Mais (heureusement) elle existe dans le **module** (on dit aussi bibliothèque) `math`.
Voici un exemple d'utilisation (saisir, interpréter et comprendre le script suivant) :

```
import math as m

a = 5
b = m.factorial(a)
print(b)
```

b) Écrire la fonction `factorielles(n)` puis tester la avec quelques valeurs de n, par exemple $n = 5$ et $n = 10$.

▶ Exercice 4 – Diverses opérations sur les listes

Il existe de très nombreuses opérations sur les listes, nous donnons ici celles qui semblent les plus utiles. Si nécessaire on complétera par la suite.

1. Opérations linéaires – Saisir, interpréter et comprendre le script suivant :

```
L = [1, 2, 3]
K = [5, 6, 7]
A = L + K
print(A)

n = 3
M = n*L
print(M)
```

2. Extraction d'un élément – Saisir, interpréter et comprendre le script suivant :

```
L = [3, -5, 8, 4, 7]
x = L[2]
print(x)
```

3. Modification d'un élément – Saisir, interpréter et comprendre le script suivant :

```
L = [3, -5, 8, 4, 7]
L[2] = 33
print(L)
```

4. *Slicing* : extraction par tranche – Saisir, interpréter et comprendre le script suivant :

```
L = [2, -4, 10, 5, 3, -6, 7]
K = L[2:5]
print(K)
```

Le *slicing* présenté ci-dessus est très basique. Il existe des manières plus élaborées de faire du *slicing*, nous verrons cela plus tard (si nécessaire).