TP 3
Boucles

1 Cours d'horizon

```
print("Bonjour !")
```

1.1 Boucles `for`

Taper les instructions suivantes dans l'interprète de commandes et observer tout particulièrement le nombre d'éléments des listes données.

```
list(range(15))

list(range(3, 15))

list(range(0, 15))

list(range(3, 15, 2))

list(range(3, 15, 1))

list(range(15, 3))

list(range(15, 3, -2))

list(range(3, 15, 2.0))
```

Taper les instructions suivantes dans l'interprète de commandes ou dans l'éditeur de scripts (pour faciliter l'écriture de l'indentation).

```
for k in range(2, 10):
    print(2**k)

for k in range(2, 10, 3):
    print(2**k)

for k in range(10, 2):
    print(2**k)
```

Combien y a-t-il eu de tours de boucle dans chacun des trois exemples précédents ?

```
def somme_premiers_entiers(n):
    """
    Entrée : un entier naturel n.
    Sortie : somme des n + 1 premiers entiers naturels (de 0 à n),
             sans utiliser la formule.
    """
    somme = 0 # La somme est initialisée à 0 (somme vide).
    for k in range(n + 1): # L'indice k va de 0 à n.
        somme += k # On ajoute l'entier k à la somme.
    return somme
```

Tester la fonction `somme_premiers_entiers` dans l'interprète de commandes.

```
somme_premiers_entiers(100)

somme_premiers_entiers(0)
```

1.2 Boucles `while`

Que fait la fonction suivante ?

```
def fonction(x):
    """ Que fait cette fonction ? """
    while x > 0:
        x = x - 1
    return x
```

Dans l'écriture d'une boucle `while`, on prendra garde à assurer la terminaison du programme, c'est-à-dire l'arrêt de la boucle.

Taper ceci dans l'interprète de commandes

```
while 1 < 2:
    print(1)
```

fera planter l'ordinateur : la condition $1 < 2$ étant toujours vérifiée, la boucle tournera indéfiniment (pour interrompre l'exécution du programme, appuyer sur le carré en haut à droite de la console).

2 Exercices

Exercice 1 — Sommes des premières puissances entières

1. Écrire une fonction qui prend en arguments deux entiers n et p, et qui renvoie la valeur de la somme $\displaystyle\sum_{k=1}^{n} k^p = 1^p + 2^p + \cdots + n^p$.
2. Écrire une fonction qui prend en entrée un entier n et qui renvoie la somme des n premiers entiers naturels impairs.

Exercice 2 — Calcul des termes d'une suite récurrente
Considérons la suite $(u_n)_{n\in\mathbb{N}}$ définie par :

$$\begin{cases} u_0 = e \\ \forall n \in \mathbb{N}, u_{n+1} = 2\ln(u_n) + u_n. \end{cases}$$

Écrire une fonction qui prend en argument un entier n et qui renvoie le terme u_n de la suite.

Exercice 3 — Factorielle conditionnelle
Écrire une fonction qui calcule la factorielle d'un entier à l'aide d'une boucle `while`.

Exercice 4 — Dépassement aléatoire borné
Tester la fonction suivante sur 0.2, 0.7 et 0.99, plusieurs fois.

```
import numpy.random as nr
# numpy.random est une sous-bibliothèque de la bibliothèque numpy,
# qui, comme son nom l'indique, sert à manipuler de l'aléatoire.

def depassement_aleatoire(seuil):
    """
    Entrée : un nombre seuil compris strictement entre 0 et 1.
    Tire des nombres aléatoires entre 0 et 1
    jusqu'à obtenir un nombre qui dépasse le seuil seuil.
    Sortie : le nombre de tirages effectués
             afin de dépasser le seuil seuil.
    """
    compteur = 0
    nombre = 0
    while nombre < seuil:
        nombre = nr.rand() # Nombre aléatoire choisi uniformément
                           # dans l'intervalle ]0 ; 1[.
        compteur += 1
    return compteur
```

On remarque que plus le seuil est grand, plus il est difficile et long de sortir de la boucle.
Pour éviter que le programme ne tourne trop longtemps, il peut être judicieux de fixer une borne sur le nombre d'itérations qu'effectue la boucle.
Écrire alors une fonction qui prend en arguments un seuil compris strictement entre 0 et 1 et un entier `borne`, qui tire des nombres aléatoires entre 0 et 1, et qui renvoie

- ⋆ 0 si on n'a pas dépassé le seuil en moins de `borne` tirages ;
- ⋆ le nombre de tirages nécessaires pour dépasser le seuil sinon.

Exercice 5 — Une suite convergente
Considérons la suite $(u_n)_{n\in\mathbb{N}^*}$ définie par :

$$\forall n \in \mathbb{N}^*, u_n = \frac{\sin(n)}{n}.$$

Écrire une fonction qui prend en argument un nombre x non nul et qui renvoie le plus petit entier naturel n non nul tel que

$$|u_{n+1}| \leqslant \left|\frac{u_n}{x}\right|.$$

Exercice 6 — Devinette

Le but de cet exercice est de programmer le jeu de devinette suivant : étant donné un entier naturel N passé en argument de la fonction de jeu, l'ordinateur choisit un entier aléatoire n dans l'intervalle $[\![1; N]\!]$, que le joueur ou la joueuse va tenter de deviner. À chacune de ses propositions, l'ordinateur affiche le message :

- ⋆ `"Trop grand !"` si l'entier proposé est strictement supérieur à n ;
- ⋆ `"Trop petit !"` si l'entier proposé est strictement inférieur à n ;
- ⋆ `"Bien joué !"` si l'entier proposé est égal au nombre n.

Lorsque le joueur ou la joueuse a trouvé le nombre choisi, le jeu s'arrête et la fonction de jeu renvoie le nombre d'essais qu'il lui a fallu pour trouver le bon nombre.

Pour cela, nous aurons besoin de la fonction `randint` de la bibliothèque `numpy.random`, qui prend en entrées deux entiers a et b, et qui renvoie un entier aléatoire compris entre a inclus et b exclu.

De plus, nous aurons besoin d'interagir avec l'utilisateur ou l'utilisatrice. Pour ce faire, on utilise la commande **`input`** qui fonctionne de la manière suivante.
L'instruction

```
input("Une question")
```

affiche la chaine de caractères [1] `"Une question"` à l'écran et attend que l'utilisateur ou l'utilisatrice entre une réponse au clavier. Sa réponse est lue par Python comme une chaine de caractères. Si cette chaine de caractères ne contient que des entiers, on pourra la convertir en entier en lui appliquant la fonction `int`.

Tester cette fonction.

```
annee_de_naissance = int(input("Quelle est votre année de naissance ?"))
# On affecte la réponse, convertie en entier,
# à la variable annee_de_naissance.

age = 2023 - annee_de_naissance

print("Vous avez eu ou vous aurez", age, "ans cette année.")
```

Programmer alors le jeu de devinette.

Exercice 7 — Suite de Fibonacci

Que fait la fonction suivante ?

```
def Fibonacci(n):
    """ Que fait donc cette fonction ? """
    (x, y) = (1, 0)
    for k in range(n - 1):
        (x, y) = (x + y, x)
    return x
```

1. Une chaine de caractères est une suite de symboles, des lettres en général, mise entre guillemets. Nous étudierons les chaines de caractères en détail dans la prochaine séance.