Correction du TP 3

Exercice 1 — Sommes des premières puissances entières

1. La fonction suivante prend en arguments deux entiers n et p et renvoie la valeur de $\sum_{k=1}^{n} k^p$.

```
def somme_puissances(n, p):
    """
    Entrées : deux entiers naturels n et p.
    Sortie : la somme des puissances p-ièmes des entiers de 1 à n.
    """
    somme = 0 # Au début, la somme est vide.
    for k in range(1, n + 1): # Pour k allant de 1 à n :
        somme += k**p # on ajoute le terme k**p à la somme.
    return somme
```

2. Écrivons une fonction qui prend en argument un entier n et qui renvoie la somme des n premiers entiers naturels impairs.

   - **Première méthode**

     Les n premiers entiers impairs sont tous les entiers de la forme $2k+1$, avec $k \in [\![0; n-1]\!]$.

```
def somme_impairs(n):
    """
    Entrée : un entier naturel n.
    Sortie : la somme des n premiers entiers impairs.
    """
    somme = 0 # Au début, la somme est vide.
    for k in range(n): # k va de 0 à n - 1
                       # donc il y a n tours de boucle :
                       # on sommera bien n entiers.
        somme += 2 * k + 1
    return somme
```

   - **Seconde méthode**

     Pour obtenir les premiers entiers impairs, on peut aussi partir de l'entier 1, puis avancer de deux en deux.

```
def somme_impairs_bis(n):
    """
    Entrée : un entier naturel n.
    Sortie : la somme des n premiers entiers impairs.
    """
    somme = 0 # Au début, la somme est vide.
    for j in range(1, 2 * n, 2):
    # j va de 1 à 2n - 1 par pas de 2,
    # donc j parcourt les n premiers entiers impairs.
        somme += j
    return somme
```

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

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

La fonction suivante prend en argument un entier n et renvoie le terme u_n de la suite.

```
import numpy as np

def suite(n):
    """
    Entrée : un entier n.
    Sortie : le terme u_n.
    """
    terme = np.e
    for k in range(1, n + 1):
    # Pour k allant de 1 à n,
    # la variable terme prend la valeur du terme suivant.
        terme = 2 * np.log(terme) + terme
        # Après le tour où k = 1, terme vaudra u_1,
        # ...
        # après le tour où k = n, terme vaudra u_n.
    return terme
```

Exercice 3 — Factorielle conditionnelle
Écrivons la fonction factorielle à l'aide d'une boucle `while`.

```
def factorielle_while(n):
    """
    Entrée : un entier n.
    Sortie : le nombre n!.
    """
    produit = 1
    k = 2
    while k <= n:
    # On multipliera le produit par chacun des entiers k
    # compris entre 2 et n inclus.
        produit = produit * k
        k = k + 1
    return produit
```

Exercice 4 — Dépassement aléatoire borné
Testons la fonction suivante, 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.
La fonction suivante prend en arguments un seuil compris strictement entre 0 et 1 et un entier `borne`, tire des nombres aléatoires entre 0 et 1, puis 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.

```
import numpy.random as nr

def depassement_aleatoire_borne(seuil, borne):
    """
    Entrées : - un réel seuil dans ]0 ; 1[,
              - un entier borne.
    Tire des nombres aléatoires entre 0 et 1
    jusqu'à dépasser le seuil seuil
    ou jusqu'à avoir effectué borne tirages.
    Sortie : - 0 si on ne dépasse pas le seuil seuil
               en moins de borne tirages,
             - le nombre de tirages nécessaires
               pour dépasser le seuil seuil sinon.
    """
    compteur = 0
    nombre = 0
    while (nombre < seuil) and (compteur < borne):
        nombre = nr.rand()
        compteur += 1
    if nombre >= seuil:
        # Si on sort de la boucle en ayant dépassé le seuil.
        return compteur
    else:
        # On a dépassé le nombre limite de tirages.
        return 0
```

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

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

Commençons par définir une fonction qui prend en argument un entier n et qui renvoie le terme $u_n = \dfrac{\sin(n)}{n}$.

```
import numpy as np

def suite(n):
    """
    Entrée : un entier n non nul.
    Sortie : le terme u_n = sin(n)/n.
    """
    return np.sin(n) / n
```

La fonction suivante prend alors en argument en nombre x non nul et renvoie le plus petit entier naturel n non nul tel que

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

```
def premier_rang(x):
    """
    Entrée : un flottant x non nul.
    Sortie : le plus petit rang n tel que la valeur absolue
             de u_{n + 1} soit inférieure à celle de u_n/x.
    """
    rang = 1
    while np.abs(suite(rang + 1)) > np.abs(suite(rang) / x):
        rang += 1
    return rang
```

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.

Le jeu de devinette décrit se programme ainsi.

```python
import numpy.random as nr

def devinette(N):
    """
    Entrée : un entier N.
    Tire un entier aléatoire entre 1 et N (inclus).
    Sortie : nombre d'essais qu'il faut au joueur ou à la joueuse
             pour deviner ce nombre.
    """
    n = nr.randint(1, N + 1)
    # L'entier n à deviner est compris entre 1 et N.
    proposition = int(input("Proposez un nombre."))
    nombre_essais = 1 # Nombre d'essais.
    while proposition != n:
    # Tant que le nombre n n'a pas été deviné,
    # on rejoue encore un tour.
        if proposition > n:
            proposition = int(input("Trop grand ! Essayez encore."))
        else:
            proposition = int(input("Trop petit ! Essayez encore."))
        nombre_essais += 1
    print("Bien joué !")
    return nombre_essais
```

Exercice 7 — Suite de Fibonacci

Le programme suivant prend en argument un entier n et renvoie le $(n+1)$-ième terme de la suite de Fibonacci $(F_k)_{k\in\mathbb{N}}$ définie par :

$$\begin{cases} F_0 = 0 \\ F_1 = 1 \\ \forall k \in \mathbb{N}, F_{k+2} = F_{k+1} + F_k, \end{cases}$$

c'est-à-dire le terme F_n.

```python
def Fibonacci(n):
    (x, y) = (1, 0)
    # Au départ, x vaut F_1 et y vaut F_0.
    for k in range(n - 1): # k va de 0 à n - 2.
    # Avant l'étape k, x vaut F_{k + 1} et y vaut F_k.
        (x, y) = (x + y, x)
        # Après l'étape k, x vaut F_{k + 2} et y vaut F_{k + 1}.
    # À la fin de la boucle, x vaut F_{(n - 2) + 2} = F_n.
    return x
```