3. Fonctions (TP)

Exercice 3.5

La liste des fonctions Python existantes (comme par exemple `print()` et `input()` que vous avez déjà utilisées en TP) est disponible à l'adresse suivante :

`https://docs.Python.org/fr/3/library/functions.html`

Dans l'interpréteur de commandes, taper les instructions suivantes :

```
abs(-5)
help(abs)
max(5,12)
help(max)
min(5,12)
help(min)
```

Exercice 3.6 *(Mise en œuvre de l'exercice 3.1)*

Écrire en Python une fonction `maximum()` prenant 2 arguments de type entier et renvoyant le maximum des deux. Tester cette fonction dans un programme Python.

NB : Comme le montre l'exercice précédent, il existe une fonction `max` en Python. Vous devez donc, dans cet exercice, écrire une fonction similaire portant un autre nom que la fonction Python existante.

Exercice 3.7 *(Généralisation de l'exercice 2.11)*

Nous allons généraliser l'exercice 2.11 en calculant la somme de tous les entiers impairs positifs, inférieurs à une valeur `n`.

1. Écrire en Python une fonction qui prend en argument un entier `n` et renvoie la somme de tous les entiers impairs positifs, inférieurs à `n`.
2. Écrire le programme Python qui permet de s'assurer que l'on retrouve le même résultat quand `n` vaut 100.

Exercice 3.8 *Reprise de l'exercice 1.14*

Nous vous rappelons qu'une année est dite bissextile :
— si l'année est divisible par 4 et non divisible par 100 ;
— ou si l'année est divisible par 400.

1. Écrire en Python une fonction qui prend en argument une année et renvoie `True` si elle est bissextile et `False` sinon.
2. Écrire le programme en Python qui demande à l'utilisateur de saisir une année (on supposera que l'utilisateur saisira correctement l'année) et affichera si l'année en question est bissextile ou pas.

Exercice 3.9

1. Écrire une fonction qui prend en argument un entier n et qui renvoie la valeur de la suite U_n définie par :
$$\begin{cases} U_0 &= 10 \\ U_n &= 0.2\,U_{n-1} + 3 \end{cases}$$
2. Écrire le programme Python qui demande à l'utilisateur de saisir un rang et affiche la valeur de la suite associée.

Exercice 3.10 *(Mise en œuvre de l'exercice 3.2)*

Écrire en Python [5] :

1. une fonction `cube()` qui calcule le cube d'un entier n passé en paramètre ;
2. une fonction `volume()` qui calcule le volume d'une sphère de rayon r passé en paramètre en appelant la fonction `cube()` précédente. Pour rappel : le volume d'une sphère se calcule par la formule : $V = (4/3) \times \pi \times r^3$;
3. un programme qui permet de tester la fonction précédente en affichant le volume d'une sphère dont le rayon est saisi par l'utilisateur.

Exercice 3.11 *(Mise en œuvre de l'exercice 3.3)*

Écrire en Python [6] une fonction `tableMulti()` permettant d'afficher une table de multiplication avec 3 arguments : `base`, `debut` et `fin`, la fonction affichant le résultat de la multiplication de `base` par tous les entiers situés entre `debut` et `fin` inclus. Par exemple l'appel de `tableMulti(8, 13, 17)` devra afficher :

```
Fragment de la table de multiplication par 8 :
13 x 8 = 104
14 x 8 = 112
15 x 8 = 120
16 x 8 = 128
17 x 8 = 136
```

Écrire un programme utilisant la fonction précédente pour afficher une table de multiplication dont les paramètres auront été donnés par l'utilisateur.

Exercice 3.12 *(Reprise de l'exercice 2.19)*

On joue à lancer 3 dés : un dé rouge à 6 faces (numérotées de 1 à 6), un dé vert à 8 faces (numérotées de 1 à 8) et un dé bleu à 10 faces (numérotées de 1 à 10). On veut déterminer combien il y a de façons d'obtenir un nombre n donné en additionnant les trois faces des dés. Par exemple, il y a une seule façon d'obtenir 3 qui est de faire 1 avec le dé vert, 1 avec le dé rouge et 1 avec le dé bleu. Il y a 3 façons d'obtenir 4 :

— 1 avec le dé vert, 1 avec le dé rouge et 2 avec le dé bleu, ou bien,
— 1 avec le dé vert, 2 avec le dé rouge et 1 avec le dé bleu, ou bien,
— 2 avec le dé vert, 1 avec le dé rouge et 1 avec le dé bleu.

5. Exercice repris de *Apprendre à programmer avec Python* de Gérard Swinnen, 2009
6. repris de `http://python.developpez.com/cours/TutoSwinnen/?page=page_9`

1. Écrire en Python une fonction qui prend en argument un entier `n` valide et renvoie le nombre de possibilités permettant d'obtenir ce nombre comme somme des dés rouge, vert et bleu.
2. Écrire le programme Python qui demande à l'utilisateur de saisir une valeur valide et appelle la fonction précédente pour afficher le résultat.

Exercice 3.13

Les n premiers nombres de Fibonacci sont définis par :

$$Fib(1) = Fib(2) = 1,\ Fib(i) = Fib(i-1) + Fib(i-2)\ \texttt{si}\ i \geq 3$$

1. Écrire en Python une fonction `Fibo()` prenant en argument un entier `n` et renvoyant le n^{e} nombre de Fibonacci.
2. Écrire le programme Python qui permet de tester votre fonction.

Exercice 3.14 *(Généralisation de l'exercice 2.6)*

Dans l'exercice 2.16, nous avions écrit un programme Python qui affichait :

```
1   2   3   4   5
1   3   5   7   9
1   4   7   10  13
1   5   9   13  17
```

Ce programme ne fonctionne que pour 4 lignes et 5 colonnes. Écrire une fonction qui réalise un affichage similaire mais pour un nombre de lignes et de colonnes passés en paramètre. Écrire un programme qui permet de tester votre fonction en demandant à l'utilisateur de saisir le nombre de lignes et de colonnes voulues.

Exercice 3.15 *(Mise en œuvre de l'exercice 3.4)*

Écrire 3 fonctions qui permettent de calculer le Plus Grand Commun Diviseur (PGCD) de deux nombres a et b (avec $a \geq b$).

1. La première fonction `pgcdParDiviseurs(a,b)`, avec $a \geq b$, calculera les diviseurs communs à chacun des deux nombres et renverra le plus grand diviseur commun (on pourra ne calculer que les diviseurs nécessaires).

   Par exemple, si on appelle `pgcdParDiviseurs(63,42)`, les diviseurs de `63` étant `1;3;7;9;21;63` et ceux de `42` étant `1;2;3;6;7;14;21;42`, on doit pouvoir afficher, à l'aide de la valeur renvoyée :

   ```
   Le plus grand diviseur commun de 63 et 42 est: 21
   ```

2. La deuxième fonction `pgcdParDifferences(a,b)`, avec $a \geq b$, utilisera l'algorithme des différences pour renvoyer le PGCD. Si un nombre est un diviseur de 2 autres nombres a et b, alors il est aussi un diviseur de leur différence $a - b$. Le PGCD de a et b est donc aussi celui de b et $a - b$.

   Par exemple, calculons le PGCD de `60` et `36`.

   `60 - 36 = 24`, donc le PGCD de `60` et `36` est un diviseur de `24`. On recommence en effectuant une nouvelle soustraction entre le résultat obtenu à la soustraction précédente (24) et le plus petit des deux termes de la soustraction précédente (36) : on obtient

`36 - 24 = 12`. Par conséquent, le PGCD de `60` et `36` est un diviseur de `12` et on peut ré-appliquer la soustraction. On arrête l'algorithme dès que la soustraction donne un résultat nul. Le PGCD correspond au résultat obtenu à l'étape précédente.. Voici l'illustration des étapes :

```
60 - 36 = 24
36 - 24 = 12
24 - 12 = 12
12 - 12 = 0
```

Ainsi, l'appel de `pgcdParDifferences(60,36)` doit renvoyer 12.

Vérifier que votre programme rend le bon résultat pour les couples $(60, 32)$; $(6, 2)$; $(36, 15)$.

3. La troisième fonction `pgcdParEuclide(a,b)`, avec $a \geq b$, utilisera l'algorithme d'Euclide pour renvoyer le PGCD. L'algorithme d'Euclide pour calculer le PGCD de deux entiers a et b (avec $a \geq b$) consiste à effectuer une suite de divisions euclidiennes :
   — étape 1 : effectuer la division euclidienne de a par b ; on obtient r le reste ; si r est égal à 0, alors fin et le PGCD est égal à b, sinon aller à l'étape 2 ;
   — étape 2 : a reçoit alors la valeur de b et b reçoit la valeur de r et aller à l'étape 1.

   Cette méthode en général est plus rapide.

   Voici l'illustration des étapes :

```
561 divisé par 357 donne 1 en quotient et 204 en reste
donc 561 = 357 x 1 + 204
357 divisé par 204 donne 1 en quotient et 153 en reste
donc 357 = 204x1 + 153
204 divisé par 153 donne 1 en quotient et 51 en reste
donc 204 = 153x1 + 51
153 divisé par 51 donne 3 en quotient et 0 en reste
donc 153 = 51x3+ 0
Le plus grand diviseur commun de 561 et 357 est : 51
```

   Ainsi, l'appel de `pgcdParEuclide(561,357)` doit renvoyer 51.