Exemples de fonctions en *Python*

Exercice 1. *Quelques fonctions simples*

Voici des exemples de fonctions simples en *Python* utilisant des tests (`if`, `else`) et des boucles (`for` et `while`).

1. Calculer le carré d'un nombre

```
def carre(n):
return n*n
```

2. Afficher les k premiers carrés (de 0 à $(k-1)^2$)

```
def premiers_carres(k):
    for i in range(k):
        print carre(i)
```

Ou encore (avec un `while` à la place du `for`)

```
def premiers_carres(k):
    i = 0
    while i < k:
        print carre(i)
        i = i + 1
```

Remarque : Ces fonctions ne renvoient pas de résultat. Elles se contentent d'afficher des valeurs à l'écran mais n'ont pas d'instruction `return`.

Si on veut faire un programme qui demande un entier k à l'utilisateur puis qui affiche les k premiers carrés, on peut écrire

```
k = input("Entrez un nombre : ")
premiers_carres(k)
```

Notez qu'on appelle directement la fonction `premiers_carres`, sans précéder l'instruction de `print` justement parce que quand on exécute la fonction elle affiche déjà des choses à l'écran. Si on écrivait `print premiers_carres(k)` alors après avoir affiché tous les carrés à l'écran, le programme écrirait en plus `None` qui est le résultat de la fonction (comme elle ne renvoie rien, Python considère que son résultat est `None`).

3. Une fonction qui teste si un entier est divisible par 10 mais pas par 3

```
def dix_mais_pas_trois(n):
    if n % 10 == 0:
        if n % 3 == 0:
            return False
        else:
            return True
    else:
        return False
```

Autre solution

```
def dix_mais_pas_trois(n):
    if n % 10 == 0 and n % 3 != 0:
        return True
    else:
        return False
```

4. Une fonction qui compte le nombre de voyelles (minuscules, sans accents) dans une chaîne de caractères `u` passée en argument

```
def nb_voyelles(u):
    voyelles = 0
    for c in u:
        if c == 'a' or c == 'e' or c == 'i' or c == 'o' or c == 'u':
            voyelles = voyelles + 1
    return voyelles
```

La même avec un `while` à la place du `for`

```
def nb_voyelles(u):
    voyelles = 0
    i = 0
    while i < len(u):
        c = u[i]
        if c == 'a' or c == 'e' or c == 'i' or c == 'o' or c == 'u':
            voyelles = voyelles + 1
        i = i + 1
    return voyelles
```

La même en utilisant des raccourcis de syntaxe en *Python* (pas indispensable à comprendre)

```
def nb_voyelles(u):
    voyelles = 0
    for c in u:
        if c in 'aeiou':
            voyelles += 1
    return voyelles
```

Une dernière version, encore plus compacte (encore moins indispensable à comprendre, uniquement pour ceux que ça amuse)

```
def nb_voyelles(u):
    return len([c for c in u if c in 'aeiou'])
```

5. Une fonction qui teste si un mot ne contient que des voyelles (minuscules, sans accents)

```
def que_des_voyelles(u):
    for c in u:
        if c != 'a' and c != 'e' and c != 'i' and c != 'o' and c != 'u':
            return False
    return True
```

Autre solution

```
def que_des_voyelles(u):
    for c in u:
        if not c in 'aeiou':
            return False
    return True
```

6. Une fonction qui teste si deux mots `u` et `v` ont autant de voyelles

```
def autant_de_voyelles(u, v):
    if nb_voyelles(u) == nb_voyelles(v):
        return True
    else:
        return False
```

Exercice 2. *Des fonctions récursives*

Rappel : une fonction est définie récursivement si elle est définie sur des grandes valeurs à partir de sa définition sur des valeurs plus petites (le concept est assez proche d'une preuve par récurrence)

1. La factorielle (vue en TD et TP). On définit directement la valeur de la fonction sur la plus petite valeur (ici 1) puis on dit que la valeur de la fonction pour n est obtenue en fonction de la valeur de la fonction en $(n-1)$ par la relation $f(n) = n \times f(n-1)$.

```
def factorielle(n):
    if n == 1 :
        return 1
    else:
        return n * factorielle(n-1)
```

2. Le PGCD de deux nombres par la méthode d'Euclide (vue en TD), écrite de manière récursive

```
def pgcd(p, q):
    p, q = max(p, q), min(p, q)
    if q == 0:
        return p
    else:
        return pgcd(q, p % q)
```

3. Une fonction qui affiche les termes de la suite de Syracuse (vue de nombreuses fois en TD et TP) à partir d'un entier n jusqu'à atteidre 1 (écrite de manière récursive)

```
def syracuse(n):
    if n == 1:
        print 1 # on affiche 1 et on ne fait rien d'autre
    else:
        print n
        if n % 2 == 0:
            syracuse(n/2)
        else:
            syracuse(3*n + 1)
```