# TP N°9: Les fonctions (2/2)

## Utilisations de fonctions

En Python, une fonction est une valeur comme une autre, c'est-à-dire qu'elle peut être *passée en argument* (1), *renvoyée comme résultat* (2) ou encore *stockée dans une variable* (3).

```
1 def g():
2     return 4
3 def f(fonction):
4     return fonction() + 1
5 print(f(g))
```

Listing 1: fonction passée en argument

```
1 def f():
2     def g():
3         return 4
4     return g
5 print(f()())
```

Listing 2: fonction renvoyée comme résultat

```
1 def g():
2     return 4
3
4 f = g()
5 print(f())
```

Listing 3: fonction stockée dans une variable

**Remarques:**

- On peut assigner une fonction à une variable comme tout autre objet python.
  Exemple: `afficher=print; afficher("bonjour")`

**Exercice 1**

Écrire les fonctions Python suivantes :

- `carre(n)` : calcule et retourne le carré d'un entier $n$ passé en paramètre.
- `somme_fonction(f, n)` : calcule et retourne la somme $\sum_{i=0}^{n} f(i)$ pour une fonction $f$ qui retourne le carré du paramètre $n$=10.

## La fonction lambda()

Pour optimiser le code de l'exercice 1, on peut même éviter de nommer la fonction *carré(n)*, puisqu'elle est réduite à une simple expression, cela nécessiterait d'y incorporer un concept supplémentaire : les *expressions lambda*.
Une expression lambda est une *fonction anonyme* s'écrit: `lambda x: e` qui désigne la fonction $x \to e(x)$ où `e` est une expression pouvant comporter la variable `x`.
Dans L'exercice 1, on peut, sans définir la fonction `carré`, appeler la fonction `somme_fonction` plus simplement comme suit :

```
1 somme_fonction(lambda x : x**x , 10)
```

**Remarques:**

- En Python, une expression lambda est une fonction
  - anonyme (n'a pas de nom),
  - qu'on peut appliquer "à la volée" dans une expression.
  - et dont l'expression retournée ne doit pas contenir d'instructions composées (pas d'affectation, pas de boucle, etc.)
- Les fonctions lambda sont réservées à des situations relativement simples.

**Exemple :**

```
1 somme = lambda x,y : x + y # la somme de ses deux arguments
2 print(somme(3,4))
3 f = lambda  : 1/2  # fonction lambda sans arguments
4 print(f()) # 0.5
```

**Exercice 2**

Écrire les fonctions Python suivantes : ...

Les fonctions anonymes permettent d'appliquer deux commandes intéressantes sur les listes : *map* et *filter*.

## La fonction map()

Une Exemple : ajouter 1 à tous les éléments d'une liste `L`.

```
f = lambda x : x + 1
L = [1,2,3]
L = list(map(f,L)) # L = [2, 3, 4]
```

**Remarques:**

- La fonction `map()` peut être appliquée à plus d'une liste. Les listes doivent avoir la même longueur. Exemple : `map(f, L1, L2, L3)`
- `map()` appliquera sa fonction lambda pour les éléments des listes d'arguments.
- `map()` applique d'abord la fonction `f` aux éléments avec l'indice 0, puis aux éléments d'indice 1 et ainsi de suite jusqu'à ce que le dernier indice est atteint.

Exemple :

```
L1 = [1,2,3]; L2 = [1,2,3]
L = list(map(lambda x,y : x + y, L1,L2)) # L = [2, 4, 6]
```

**Exercice 3**

Écrire les fonctions Python suivantes :

- `fact(n)` : calcul et retourne le factoriel d'un entier $n$ passé en paramètre.

## La fonction filter()

La fonction filter offre un moyen élégant de filtrer tous les éléments d'une séquence.
Syntaxe : `filter(fonction, sequence)`
La fonction `filter` a besoin de deux arguments :

**Une fonction** : comme premier argument, qui retourne une valeur booléenne, soit `True` soit `False`. Cette fonction sera appliquée à tous les éléments de la sequence (deuxième argument de `filter`).

**Une sequence**: c'est le deuxième argument de `filter`. Seulement les éléments inclus dans cette séquence où la fonction renvoie `True` seront produites par le résultat du `filter`.

Exemple : supprimer tous les multiples de 2 d'une liste d'entiers `L`.

```
f = lambda x : x % 2
L = list(range(20))
L = list(filter(f,L)) # L = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
```

**Remarques:**

- La fonction `map()` peut être appliquée à plus d'une liste. Les listes doivent avoir la même longueur. Exemple : `map(f, L1, L2, L3)`

**Exercice 4**

- Filtrer d'abord les nombre impairs puis les nombres pairs des éléments de la séquence des n premiers nombres de Fibonacci.