Exercice 1.
Ecrire une fonction `trouve(p, t)` qui reçoit en arguments une fonction `p` (pour «propriété») et un tableau `t` et renvoie le premier élément `x` de `t` tel que `p(x)` vaut `True`. Si aucun élément de `t` ne satisfait `p`, alors la fonction renvoie `None`.

Exercice 2.
Ecrire une fonction `applique(f, t)` qui prend en paramètres une fonction `f` et un tableaut et renvoie un nouveau tableau, de même taille, où la fonction `f` a été appliquée à chaque élément de `t`. Le faire avec et sans la notation par compréhension.

Exercice 3.
La fonction `tri_insertion(t)` ci-dessous prend en paramètre un tableau `t` et trie, en place, ses élément par ordre croissant selon l'algorithme de tri par insertion vu en première.

```
def tri_insertion(t):
    for i in range(1, len(t)):
        v = t[i]
        j = i
        while j>0 and t[j-1]>v:
            t[j] = t[j-1]
            j = j-1
        t[j] = v
```

1. Modifier cette fonction pour qu'elle prenne comme argument supplémentaire une fonction `inf(x,y)` qui retourne `True` si `x <= y` et `False` sinon. Le tri devra se faire respectivement à cette fonction.
2. On souhaite à présent écrire une fonction `tri_insertion` dans le paradigme fonctionnelle. Compléter le programme suivant pour qu'il réalise l'agorithme de tri par insertion.

```
def queue(liste):
    '''la queue de la liste'''
    return ???

def tete(liste):
    '''la tête de la liste'''
    return ???

def insere(v, t):
    '''insere la valeur v dans le tableau trié t'''
    if t==[] or v<tete(t):
        return ???
    else:
        return ???+insere(???)

def tri_insertion(t):
    if ???:
        return []
    else:
        return insere(tete(t),tri_insertion(queue(t)))
```

Exercice 4.
1. Ecrire une fonction `double(f)` qui reçoit une fonction `f` en argument et renvoie une fonction qui applique deux fois de suite la fonction `f` à son argument.
2. Que vaut `double(double(lambda x: x*x))(2)` ? (Trouver le résultat de tête, puis lancer le calcul pour vérifier.

Exercice 5.
Ecrire une fonction `compose(f, g)` qui prend en arguments deux fonctions `f` et `g` et renvoie leur composition, c'est à dre la la fonction `h` définie par `h(x)=f(g(x))`.

Exercice 6.
A l'a l'aide de la notation `lambda`, écrire les fonctions suivantes :

1. une fonction `ajoute_15(x)` qui ajoute 15 à l'entier `x` donné en argument.
2. une fonction `mult(x,y)` qui prend deux entiers `x` et `y` en paramètres et renvoie leur produit.
3. une fonction `mult_fonc(n)` qui prend en argument un entier `n` et renvoie la fonction qui multiplie par `n`.

```
>>> mult_5 = mult_fonc(5)
>>> mult_5(3)
15
```

Exercice 7.
A l'aide de la fonction `map` écrire les fonctions suivantes :

1. une fonction `triple(t)` qui triple tous les éléments de la liste `t` de nombres donnée.
2. une fonction `ajoute_trois_listes(t1, t2, t3)` pour additionner les trois listes de nombres `t1`, `t2`, `t3`.
3. une fonction `maj_min(t)` qui, pour un tableau de charactères `t`, donne la liste des même charactères en majuscule et minuscule.

```
>>> maj_min(['a','F','G','y'])
[('A','a'),('F','f'),('G','g'),('Y','y')]
```

Exercice 8.
A l'aide de la fonction `filter` écrire :

1. une fonction `plus_petit_plus_grand(t, n)` qui partage le tableau d'entiers `t` en deux tableaux, l'un contenant les éléments plus petit que `n` et l'autre contenant les éléments plus grand ou égal à `n`.

```
>>> plus_petit_plus_grand([2,24,3,14,42], 10)
([2,3], [24,14,42])
```

2. une fonction `partition(p, t)` qui partage la liste `t` en deux listes : les éléments de `t` qui vérifient la condition `p` et les autres.
3. une fonction `long_mots(t, n)` qui prend en paramètres une liste de mots `t` et un nombre entier `n` et renvoie la liste des mots dont la longueur est supérieure à `n`.

Exercice 9.
A l'aide de la fonction `reduce` écrire les fonctions suivantes :

1. Une fonction `fact(n)` qui renvoie la valeur de $n! = 1 \times 2 \times 3 \times ... \times n$, c'est à dire le produit des n premiers entiers.

```
>>> fact(5)
120
>>> fact(12)
479001600
```

2. Une fonction `maximum(t)` qui calcul le maximum d'une liste d'entiers `t`, puis une fonction `minimum(t)` qui calcul le minimum d'une liste d'entiers `t`.
   Ecrire ensuite une fonction `max_min(t)` qui donne le maximum et le minimum d'une liste d'entiers `t`.

```
>>> max_min([2,1,5,3])
(1, 5)
```

3. Une fonction `somme_sup(t, n)` qui prend en paramètres un tableau d'entier `t` et un entier `n` renvoie la somme des éléments de `t` supérieur à `n`.

```
>>> somme_sup([4,2,56,-12,1],3)
60
```

4. Une fonction `longueur(t)` qui calcul le nombre d'éléments de la liste `t`.

```
>>> longueur([4,2,56,-12,1])
5
```

5. Une fonction `all_true(t)` qui prend en paramètre un tableau de booléen `t` et renvoie `True` si tous les éléments de `t` sont `True` et `False` sinon.

```
>>> all_true([True,True,True])
True
>>> all_true([True,False,True])
False
```

   Ecrire ensuite une fonction `check_all(p, t)` qui reçoit en argument une fonction `p` et un tableau `t` et revoie `True` si pour tous les éléments `x` de `t`, `p(x)` vaut `True` et `False` sinon.

```
>>> check_all(lambda x: x%2==0, [2,4,12,42])
True
>>> check_all(lambda x: x%2==0, [2,3,12,42])
False
```

6. Une fonction `any_true(t)` qui prend en paramètre un tableau de booléen `t` et renvoie `True` si au moins un des éléments de `t` vaut `True` et `False` sinon.

```
>>> any_true([False,False,False])
False
>>> any_true([True,False,True])
True
```

   Ecrire ensuite une fonction `check_any(p, t)` qui reçoit en argument une fonction `p` et un tableau `t` et revoie `True` s'il existe au moins une valeur `x` de `t` telle que `p(x)` vaut `True` et `False` sinon.

```
>>> check_any(lambda x: x%2==0, [1,3,11,42])
True
>>> check_any(lambda x: x%2==0, [1,3,11,41])
False
```

7. Une fonction `occurence(mot)` qui prend une chaîne de caractère `mot` en paramètre et renvoie le dictionnaire des occurences des lettres formants ce mot.

```
>>> occurence("banane")
{'a': 2, 'b': 1, 'e': 1, 'n': 2}
```

8. A l'aide de la fonction `insere(v, t)` de l'exercice 3, écrire une nouvelle fonction `tri_insertion(t)` qui utilise `reduce`.