université
PARIS-SACLAY
DIU EIL
Bloc 4

# Mise au point de programmes

## 1 Exceptions

Considérons le programme ci-dessous, utilisant version récursive de la fonction `nieme_element` sur les listes chaînées (qui lève une exception en cas d'indice invalide).

```
def ajout_10000(l):
    """rajoute 10000 entiers en tête de liste"""
    c = l
    for i in range(10000):
        c = Cellule(i, c)
    return c

def f():
    l = None
    v = -1
    while v < 0:
        try:
            v = nieme_element(l, 8000)
        except:
            l = ajout_10000(l)
    print(v)
f()
```

Quel est le comportement de ce programme ?

## 2 Annotations de types

1. Rappeler les opérations du type abstrait `Pile` et donner leurs prototypes (i.e. les définitions des fonctions avec leur anotation de type, sans le corps de la fonction). On suppose qu'il existe un type `Pile` et que les opérations sont des fonctions (et non pas des méthodes).
2. Considérons la fonction

   ```
   def f(a, b, c):
       return a * b + c
   ```

   Donner un maximum de types possibles pour cette fonction (au moyen d'annotations). Attention on ne souhaite pas ici avoir une seule annotation complexe, mais plusieurs fois la fonction `f` avec des annotations de types différentes.

## 3 Tests de programme

On souhaite écrire la fonction de `test` d'un tableau trié vue en cours.

1. Écrire une fonction `occurrences(t)` prenant en argument un tableau d'entiers et renvoyant un dictionnaire dont les clés sont les entiers apparaissant dans `t` et les valeurs le nombre d'occurrences de chaque entier dans `t`. Par exemple, pour le tableau `[41,3,3,42,3,3]`, la fonction renvoie le dictionnaire `{ 41: 1, 42: 1, 3 : 4 }`.

2. On rappelle qu'en Python on peut comparer deux dictionnaires avec == et que le test est vrai si les listes triées de leur paires `(clé, valeur)` sont ==. Écrire une fonction `test(f, t)` qui étant donnés une fonction de tri en place `f` et un tableau `t`, vérifie que `tri` est un tri correct sur `t`.