PI2TJuin2018_A

August 27, 2018

1 PI2T Juin 2018 - Version A

1.1 programmation fonctionnelle

La fonction `reduce()` est une fonction classique de la programmation fonctionnelle. Elle permet d'effectuer une opération (une fonction, le **réducteur**) sur chaque élément d'une liste et d'accumuler les résultats dans une variable, l'**accumulateur**.

Dans le code qui suit vous trouverez une implémentation de la fonction `reduce()` ainsi que celle d'un réducteur, `add()`, permettant de sommer les éléments d'une liste. Il s'agit là d'un exemple de base de l'utilisation de `reduce()`.

On vous demande d'écrire une fonction nommée `limitMax()` qui **renverra un réducteur**. Ce réducteur permettra de limiter les valeur d'une liste à une valeur maximale reçue en paramètre de `limitMax()`. Si une valeur de la liste est plus grande que la valeur maximale elle sera remplacée par ce maximum.

1.1.1 Exemple:

```
reduce(limitMax(3), [1, 2, 3, 4, 5, 6], [])        # return [1, 2, 3, 3, 3, 3]
```

```
In [ ]: def reduce(fn, L, init):
            acc = init
            for elem in L:
                acc = fn(elem, acc)
            return acc

        def add(elem, acc):
            return elem + acc

        reduce(add, [1, 2, 3], 0)                          # return 6

        def limitMax(maximum):
            # Votre code ici
```

1.2 Fonction récursive

Ecrire une fonction récursive nommée `zero` prenant une fonction (`fn` à valeur numérique et prenant un paramètre numérique, continue) en paramètre ainsi que deux bornes (inférieure et suppérieure) et cherchant une racine de `fn` par la méthode dichotomique.

Voici l'algorithme: 1. Si `fn(inf)` et `fn(sup)` sont de signes différents, il existe une racine entre `inf` et `sup` (sinon on renvoie `None`). 2. On divise l'interval en deux et on applique la même procedure sur le demi-interval qui satisfait à la condition du point 1. 3. On s'arrête quand l'écart entre `inf`et `sup` devient plus petit que la précision recherchée.

On vous demande une précision à 6 décimales.

1.2.1 Exemple:

```
from math import sin
zero(sin, 2, 4)  # renvoie 3.141592025756836
```

```
In [ ]: def zero(fn, inf, sup):
            # Votre code ici
```

1.3 Décorateur

Ecrivez un décorateur paramétrique nommé `password` et prenant un mot de passe en paramètre. La fonction décorée devra prendre ce mot de passe en paramètre pour renvoyer son résultat. Si le mot de passe n'est pas donné, elle devra lever une exception de type `ValueError`. Le decorateur doit fonctionner sur n'importe quelle fonction quelques soient ses paramètres (nommés et positionnels).

1.3.1 Exemple:

```
@password("secret")
def add(a, b):
    return a + b

add("hacker", 1, 2)      # ValueError Exception
add(1, 2)                # ValueError Exception
add("secret", 1, 2)      # 3
```

```
In [ ]: def password(pswd):
            # Votre code ici
```

1.4 Arbres

Ecrire un fonction nommée `treeMax()` qui reçoit un arbre (de la classe `Tree` ci-dessou) et qui renvoie la plus grande valeur de l'arbre.

1.4.1 Exemple:

```
t = Tree(0, Tree(4, Tree(-5), Tree(3, Tree(8), Tree(1))), Tree(2, Tree(4)))

treeMax(t)       # renvoie 8
```

```
In [ ]: class Tree:
            def __init__(self, value, left=None, right=None):
                self.value = value
                self.left = left
                self.right = right
```

```
In [ ]: def treeMax(tree):
            # Votre code ici
```