Python (M1) – Des fonctions et des listes

TD #1

Rédacteur: Stéphane Vialette

1 Fonction itératives et récursives sur les listes

1.1 Longueur

Écrire (en itératif et récursif) une fonction prenant une liste en argument et retournant le nombre d'éléments la composant (sans utiliser la fonction `len` de python) Exemple d'utilisation :

```
>>> l = range(10)
>>> l
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> import td1.len
>>> td1.len.len_iterative(l)
10
>>> td1.len.len_recursive(l)
10
>>>
```

1.2 Renversement

Écrire une fonction `reverse` qui prend en argument une liste et retourne la liste des mêmes éléments mais dans l'ordre inverse (encore une fois, il ne saurait être question d'utiliser la méthode `reverse` de la classe `list`). Exemple :

```
>>> import td1.reverse
>>> td1.reverse.reverse([])
[]
>>> l = range(10)
>>> l
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> td1.reverse.reverse([])
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
>>>
```

1.3 Maximum

Écrire (en itératif et récursif) une fonction prenant une liste non vide en argument et retournant un de ses éléments maximaux (sans utiliser la fonction `max` de python) Exemple d'utilisation :

```
>>> l = range(10)
>>> l
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> import td1.max
>>> td1.max.max_iterative(l)
9
>>> td1.max.max_recursive(l)
9
>>> import random
>>> random.shuffle(l)
>>> l
[7, 0, 9, 2, 6, 3, 4, 8, 1, 5]
>>> td1.max.max_iterative(l)
9
>>> td1.max.max_recursive(l)
9
>>>
```

1.4 Zip

la fonction `zip` retourne une liste de tuples, où le i-ème tuple contient le i-ème élément de chacune des listes passées en argument. La liste retournée est tronquée à la longueur de la plus courte liste passée en argument. Exemple d'utilisation :

```
>>> x = [1, 2, 3]
>>> y = [4, 5, 6]
>>> zipped = zip(x, y)
>>> zipped
[(1, 4), (2, 5), (3, 6)]
>>> x = [1, 2, 3]
>>> y = [4, 5, 6, 7]
>>> z = [8, 9]
>>> zipped = zip(x, y, z)
>>> zipped
[(1, 4, 8), (2, 5, 9)]
>>>
```

Écrire une implémentation python de la fonction `zip` (vous pourrez dans un premier temps simplifier l'écriture en supposant que votre fonction `zip` ne prend que 2 listes en arguments).

Écrire une implémentation python de la fonction `unzip` effectuant l'opération inverse (vous pourrez une nouvelle fois simplifier l'écriture en supposant dans un premier temps que votre fonction `unzip` prend en argument une liste de paires). Exemple :

```
>>> zipped = [(1, 4), (2, 5), (3, 6)]
>>> x, y = unzip(zipped)
>>> x
[1, 2, 3]
>>> y
[4, 5, 6]
>>> zipped = [(1, 4, 6), (2, 5), (3)]
>>> x, y, z = unzip(zipped)
>>> x
[1, 2, 3]
>>> y
[4, 5]
>>> z
[6]
>>>
```

2 Fonctionnelles

2.1 Implémentation

Écrire une implémentation en python des fonctions `filter`, `map` et `reduce`. Consultez

- `http://docs.python.org/library/functions.html#filter`,
- `http://docs.python.org/library/functions.html#map`, et
- `http://docs.python.org/library/functions.html#reduce`

pour une description complète de ces fonctions.

2.2 Pliage

Considérons la fonction `fold_left` définie de la façon suivante :

```
def fold_left(seq, reduce_fun, init_val):
    if not seq:
         return init_val
    return fold_left(seq[1:], reduce_fun, reduce_fun(init_val, seq[0]))

# some tests
add_fun = lambda total, item: total+item
assert fold_left([1, 2, 3, 4, 5, 6], add_fun, 0) == 21
assert fold_left([], add_fun, 0) == 0
assert fold_left(None, add_fun, 0) == 0
```

Que fait cette fonction ? Pouvez vous en donner une expression python plus concise ?

3 Composition . . .et autre

3.1 `f` ∘ `g` et `g` ∘ `f`

Écrire la fonction `compose(f, g)` qui retourne la composée de `f` et `g`. Par exemple :

```
>>> import td1.compose
>>> succ = lambda x: x+1
>>> double = lambda x: x*x
>>> succ_double = td1.compose.compose(succ, double)
>>> succ_double
<function h at 0x1004d0668>
>>> [succ_double(i) for i in range(5)]
[1, 2, 5, 10, 17]
>>> double_succ = td1.compose.compose(double, succ)
>>> double_succ
<function h at 0x1004d7668>
>>> [double_succ(i) for i in range(5)]
[1, 4, 9, 16, 25]
>>>
```

3.2 fonction itérée

Écrire la fonction `fun_iter(fun, n)` qui retourne la fonction `fun` istérée `n` fois, *i.e.*, `f(f(f(f(f(...)))))`.

3.3 Application partielle

Écrire la fonction `partial(f, x)` qui retourne l'application partielle de `x` à `f` (une fonction prenant exactement 2 arguments). Par exemple :

```
>>> import partial
>>> def addition(x, y):
...     return x+y
...
>>> succ = partial.partial(addition, 1)
>>> succ
<function partial_fun at 0x10f27bed8>
>>> [succ(i) for i in range(10)]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> def multiplication(x, y):
...     return x*y
```

```
...
>>> triple = partial.partial(multiplication, 3)
>>> [triple(i) for i in range(10)]
[0, 3, 6, 9, 12, 15, 18, 21, 24, 27]
>>>
```

Pour aller plus loin, modifiez votre implémentation pour prendre en argument des fonctions `f` d'arité quelconque.

```
>>> import partial
>>> def f(x, y, z):
...     return x + (2*y) + (3*z)
...
>>> g = partial.partial(f, 2)
>>> g(3, 4)
20
>>> h = partial.partial(g, 3)
>>> h(4)
20
>>>
```

4 Crible d'Ératosthène

Le crible d'Ératosthène est un procédé qui permet de trouver tous les nombres premiers inférieurs à un certain entier naturel donné N.

L'algorithme procède par élimination : il s'agit de supprimer d'une table des entiers de 2 à N tous les multiples d'un entier. En supprimant tous les multiples, à la fin il ne restera que les entiers qui ne sont multiples d'aucun entier, et qui sont donc les nombres premiers. On commence par rayer les multiples de 2, puis à chaque fois on raye les multiples du plus petit entier restant. On peut s'arrêter lorsque le carré du plus petit entier restant est supérieur au plus grand entier restant, car dans ce cas, tous les non-premiers ont déjà été rayés précédemment. Á la fin du processus, tous les entiers qui n'ont pas été rayés sont les nombres premiers inférieurs à N.

Écrire la fonction `erastothene(N)` qui retourne la liste des nombre premiers inférieurs à `N` en utilisant la méthode du crible d'Ératosthène.