

Stage LIESSE Python

Algorithmique

Matthieu Moy

Ensimag

octobre 2015



Sommaire

- 1 Manipulations et parcours de listes
- 2 Tri de listes (Insertion, Sélection, QuickSort)
- 3 Dessins de fractales
- 4 Recherche de zéro d'une fonction continue



Aperçu de cette partie

- Aucune difficulté algorithmique
- On joue un peu avec des constructions Python
- 99% hors-programme pour les CPGE



Parcourir une liste

- Boucle **for** (Exercice 1.1 : calcul de la moyenne) :

```
for e in l:  
    f(e)
```

- Itération avec **map** (Exercice 1.2 : moyenne avec map) :

```
nouvelle_liste = list(map(f, ancienne_liste))
```

- Exemple :

```
l = [1, 12, 42]  
def plus_un(x):  
    return x + 1  
list(map(plus_un, l))  
# [2, 13, 43]  
list(map(lambda x: x + 1, l))  
# Pareil!
```



Fonctions anonymes

- Fonctions nommées (on connaît !) :

```
def plus_un(x):  
    return x + 1  
plus_un(42) # 43
```

- Fonction anonymes (lambda) :

```
(lambda x: x + 1)(42) # Idem !
```

- Une fonction est une valeur, on peut l'affecter à un variable :

```
ma_fonction = lambda x: x + 1  
ma_fonction(42) # Encore pareil !
```

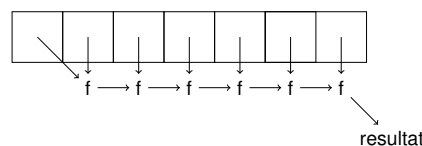
- Limité à une seule expression, **return** implicite.
(Utiliser une fonction nommée pour faire des choses compliquées)



reduce : combiner les éléments d'une liste avec une fonction

```
reduce(f, liste)  
# Applique la fonction f aux éléments de la  
# liste deux à deux
```

```
reduce(f, [a, b, c]) # pareil que f(f(a, b), c)
```



À vous : exercice 1.3 (somme en utilisant reduce)



Variantes autour de if/then/else

- Jusqu'ici : if/then/else pour les effets de bords

```
def signe(x):  
    if x >= 0:  
        resultat = 'positif'  
    else:  
        resultat = 'negatif'  
    return resultat
```

- Autre syntaxe : if/then/else fonctionnel

```
def signe(x):  
    resultat = 'positif' if x >= 0 else 'negatif'  
    return resultat
```

- Utilisation dans une fonction lambda :

```
lambda x: 'positif' if x >= 0 else 'negatif'
```

- À vous : exercice 1.4 (maximum en utilisant reduce)



Compréhensions de listes

- Syntaxe compacte, équivalente à **map** :

```
l = [1, 12, 42]  
list(map(lambda x: x + 1, l)) # [2, 13, 43]  
[x + 1 for x in l] # Idem !
```

- Possibilité de filtrage :

```
[x + 1 for x in l if x > 10] # [13, 43]
```

- Utilisation avec des affectations de tuples :

```
liste_doublets = [(1, 'un'), (2, 'deux')]  
[chaîne for (nombre, chaîne) in liste_doublets]  
# ['un', 'deux']
```

```
[nombre for (nombre, _) in liste_doublets]  
# [1, 2]
```

- À vous : Exercice 1.5 (extraire une sous-liste par compréhension)



C'est beau !



- Retour sur des constructions Python simples, mais des algorithmes plus compliqués
- Explicitement au programme CPGE



Tri par insertion

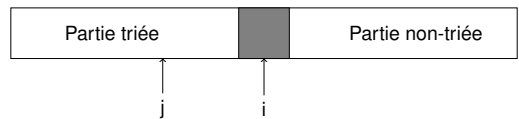
- Tri d'un paquet de cartes :
 - ▶ Partie triée en main gauche
 - ▶ Partie non-triée en main droite
 - ▶ On insère la première carte de la main droite au bon endroit
- Idem avec une liste / un tableau :



- ▶ i = indice de la valeur à insérer
- ▶ j = indice où $l[i]$ doit être inséré



Tri par insertion : corrigé



```
def insertion_sort(lst):
    for i in range(1, len(lst)):
        value = lst[i]
        j = i
        while j > 0 and lst[j - 1] > value:
            lst[j] = lst[j - 1]
            j = j - 1
        lst[j] = value
        # print(lst)
```



```
insertion_sort(lst)
print('after: ', lst)
```

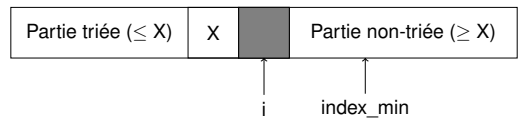
Tri par sélection du minimum



- Recherche du minimum dans la partie non-triée ($index_min$)
- Échange de $lst[index_min]$ et $lst[i]$
- Incrément de i et on recommence !
- À vous ! Exercice 2.1



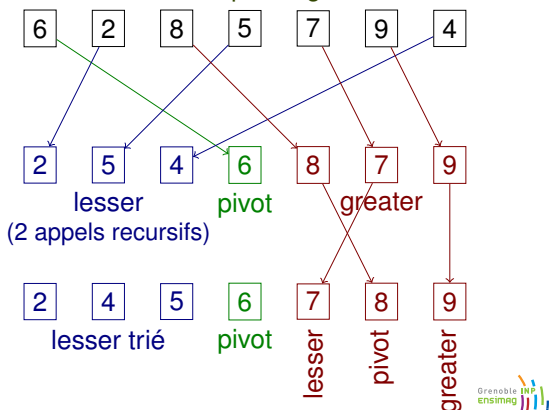
Tri par sélection du minimum : corrigé



```
def selection_sort(lst):
    for i in range(len(lst) - 1):
        min = lst[i]
        i_min = i
        for j in range(i, len(lst)):
            if lst[j] < min:
                min = lst[j]
                i_min = j
        lst[i], lst[i_min] = lst[i_min], lst[i]
```



QuickSort, ou Tri par Segmentation



Principe de l'algorithme

- Choisir un pivot
- Extraire les listes :
 - ▶ lesser des éléments < pivot
 - ▶ greater des éléments > pivot
 - ▶ equal des éléments = pivot
- Trier lesser et greater
- Concaténer lesser, equal et greater



Exercice 2.2 : tests d'algorithmes de tri

- Exécuter `test_sort.py`
- Pour l'instant, rien ne marche sauf `native`, c'est normal.



Exercice 2.3 : Solution impérative (1/2)

```
def qsort(lst):
    if lst == []:
        # cas de base
        return []
    else:
        # recursion
        lesser, equal, greater = split(lst, lst[0])
        l_sorted = qsort(lesser)
        g_sorted = qsort(greater)
        return l_sorted + equal + g_sorted
```

Reste à définir `split(lst, pivot) ...`



Exercice 2.3 : Solution impérative (2/2)

```
def split(lst, pivot):
    lesser = []
    equal = []
    greater = []
    for e in lst:
        if e > pivot:
            greater.append(e)
        elif e < pivot:
            lesser.append(e)
        else:
            equal.append(e)
    return lesser, equal, greater
```

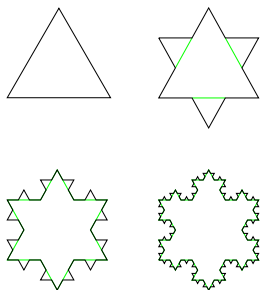


En utilisant les compréhensions de listes (ex 2.4, 2.5)

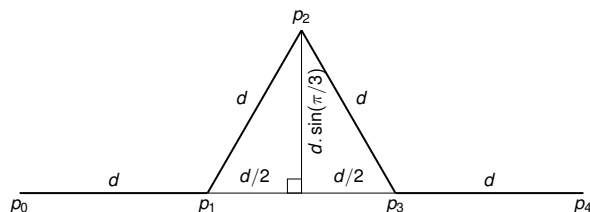
```
def qsort(lst):
    if lst == []:
        # Cas de base
        return []
    else:
        # Recursion
        pivot = lst[0]
        lesser = qsort([x for x in lst if x < pivot])
        equal = [x for x in lst if x == pivot]
        greater = qsort([x for x in lst if x > pivot])
        return lesser + equal + greater
```



Flocon de Koch : vue d'ensemble



Flocon de Koch : une itération



Méthode 1 : turtle

- Le principe de turtle : `square.py` (3.1)
- Version à une tortue : Exercice 3.2 (ou énoncé TD séparé)

```
forward(100)
left(60)
```

- Version « orientée objet » : Exercice 3.3

```
t1 = turtle.RawTurtle(w)
```

```
t1.forward(100)
t1.left(60)
```



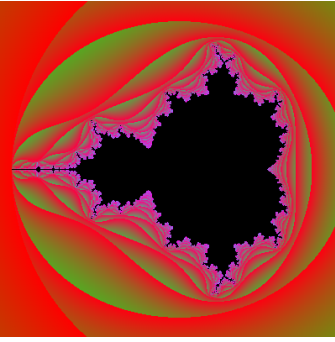
Méthode 2 : coordonnées carthésiennes

Seulement si on veut vraiment avoir mal à la tête !

- Manipulation explicite des coordonnées : Exercice 3.4
- Utilisation des nombres complexes : Exercice 3.5



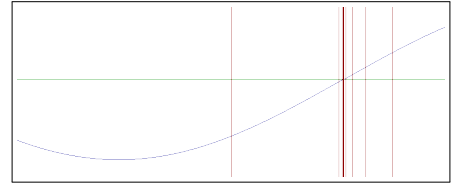
Fractale de Mandelbrot



- Exercice 3.6
- Pour les rapides : un peu d'interface graphique réactive dans l'exercice 3.7



Recherche par dichotomie



- Algorithmique : Exercice 4.1
- Visualisation : Exercice 4.2

