Introduction à la programmation, cours 5 : Numpy

I] Fonctions (niveau avancées)

1) Plusieurs valeurs de retour

Une fonction peut avoir plusieurs valeur de retour ;

```
def division(a, b) :        #renvoie le quotient de a par b
        q = a // b
        r = a % b
        return q, r         #deux valeurs de retour
q, r = division(22, 5)
print "quotient = ", q, ", reste = ", r
```

La syntaxe est simple :
return valeur1, valeur2, .., varleurN

Et pour récupérer les valeurs de retour lors d'un appel :
var1, var2, …, varN = nom_fonction(arguments)

2) Docstring

La docstring est une façon de documenter vos fonctions, c'est à dire de donner le moyen à quelqu'un qui doit utiliser votre programme de comprendre rapidement à quoi servent vos fonctions. Il s'écrit en début de fonctions, entre des triplets de guillemets. Exemple :

```
def division(a, b) : #renvoie le qotient de a par b
        """ takes two integers a and b as argument and return the quotient and the rest of the
            euclidean division of a by b """
        q = a // b
        r = a % b
        return (q, r)       #deux valeurs de retour

help(division)    #affiche la docstring de division
```

Dans un projet, il faut mettre une docstring à toutes vos fonctions.
D'ailleurs, les fonctions de Python qu'on utilise ont toutes une docstring :

```
import random as np
help(randint)
```

3) Importer vos fonctions

Créer un fichier division.py contenant la fonction division définie plus haut. Créer un second fichier teste.py contenant:

```
import division as div
q, r = div.division(22, 5)#en la précédent par div. On peut utiliser toutes les fonctions définies dans
                          #division.py
print "qotient = ", q, " reste = ", r
```

Tous les projets sont ainsi, ils contiennent plein de fichiers contenant des groupes de fonctions (appelez modules). Le TP d'aujourd'hui fonctionnera comme ça, on codera chaque fonction dans des fichiers différents, et on aura un fichier à part (qu'on appelle en général *nom_projet_main.py*) dans lequel on importera toutes les autres fonctions et qui contiendra le code principale.

4) Arguments optionnels

La carte du restau MisterPizza comporte des pizzas de différentes saveurs pouvant être commandées en taille normale au prix de 9 euros, ou en taille maxi au prix de 12 euros. La majorité des clients commanden en taille normal.

```
def affiche_pizza(saveur, taille = "normal") :
	""" Affiche saveur, taille et prix de la pizza."""
	print "Pizza", saveur, ", taille : ", taille
	if taille == "normal" :
		prix = 9
	elif taille == "maxi" :
		prix = 12
	print "Prix :", prix, " euros"

affiche_pizza("Anchois")
affiche_pizza("Reine", "maxi")
help(affiche_pizza)
```

L'argument taille est optionnel et a comme valeur par défaut "normal"
Une subtilité apparaît quand il y a plusieurs arguments optionnels :

```
def affiche_pizza(saveur, taille = "normal", afficheFranc = False) :
	""" Affiche saveur, taille et prix de la pizza."""
	print "Pizza", saveur, ", taille : ", taille
	if taille == "normal" :
		prix = 9
	elif taille == "maxi" :
		prix = 12
	if afficheFranc :
		prix_franc = round(prix * 6.55957, 2)
		print "Prix :", prix, " euros (en francs : ", prix_franc, ",  francs)"
	else :
		print "Prix :", prix, " euros"
affiche_pizza("Reine", taille = "maxi", afficheFranc = True)
affiche_pizza("4 fromages", "maxi", True)
affiche_pizza("Reine", True)	#affiche une erreur, il faut faire :
affiche_pizza("Reine", afficheFranc = True)
```

II] Numpy

Numpy est extension de Python (c'est à dire un ensemble de fonctions déjà codées et prêtes à emploie) permettant de gérer un nouveau type de structure de données, les **tableaux** (**array** en anglais).

1) Création d'un tableaux

Pour utiliser numpy, il faut commencer par importer la bibliothèque numpy, il suffit pour cela de taper la ligne de code suivant **en tout début** de fichier : import numpy as np.
A partir de là, quand on voudra utiliser un module de la bibliothèque numpy, il faudra la précéder de **np.module**().
Les tableaux sont très proches des listes, une première façon de créer un tableau et de transformer une liste en tableau à l'aide de la fonction **np.array(nom_liste).** On peut aussi créer directement un tableau :

```
import numpy as np
var_list = [1, 2, 3, 4, 5]		#on crée une liste
var_arr = np.array(var_list)		#on transforme cette liste en tableau
print var_arr
print type(var_arr)
var_arr_2 = np.array([1,2,3,4,5])		#ici on crée directement un tableau, notez l'utilisation des crochets
					#comme pour les listes)
print var_arr_2
```

Quand on travaille sur un fichier, on le transforme d'abord en liste avec **readlines()**, puis si besoin en tableau pour pouvoir le manipuler plus facilement.

Dans l'exemple suivant, on crée un tableau à deux dimensions, et on montre comment accéder à un élément d'un tableau (notez qu'un tableau à deux dimensions n'est rien d'autre qu'un tableau où **chaque élément est un tableau**):

```
import numpy as np
A = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9])      #création d'un tableau à une dimension
B = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])          #un tableau à deux dimensions, c'est en fait un
                                                          # tableau  dont chaque élément est un tableau. Vous
                                                          #pouvez le  visualiser comme ça :          1, 2, 3
                                                                                                     4, 5, 6
                                                                                                     7, 8, 9
print B[1][2]                 #corresponds à l'élément de la première ligne (premier élément), deuxième colonne
                              #(deux élément de ce premier élément. Comme d'habitude, la numérotation
                              #commence à 0.
print B[2][2]
```

2) Parcourir un tableaux

Facile, ça marche comme d'habitude (notez les petites subtilités sur un tableau à deux dimenseions):

```
import numpy as np
A = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9])
for elt in A :
        print elt
B = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
for elt in B :
        print elt
for elt in B :
        for x in elt :
                print elt
```

3) Les méthodes sum() et max()

Pour les tableaux à une dimensions :

```
import numpy as np
A = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9])
var_sum = A.sum()       #A.sum() renvoie la somme des éléments de A
print var_sum
var_max = A.max()       #A.max() renvoie le plus grand élément de A
print var_max
```

Pour les tableaux bidimensionnels, il faut préciser si on parle des colonnes ou des lignes (ceci est généralisable à des tableaux de plus grandes dimensions, mais dans ce cours on aura à priori que besoin de tableaux à une ou deux dimensions).

```
import numpy as np
A = np.array([[1, 5, 3], [4, 2, 9], [7, 8, 6]])          # 1 5 3
                                                            4 2 9
                                                            7 8 6
max_col = A.max(axis = 0)         #max_col est un tableau dont chaque élément est le maximum de chaque
                                  #colonne de A i.e. max_col = [7, 8, 9]
print max_col
max_ligne = A.max(axis = 1)       #pareil pour le max de chaque ligne, i.e. max_ligne = [5, 9, 8]
print max_ligne
```

4)Opération élément par éléments

On peut utiliser les opérations arithmétiques classiques sur les tableaux. En utilisant l'exemple ci-dessous, comprenez par vous-même les résultats :

```
import numpy as np
```

```
A = np.array([2, 4, 6, 8, 10])
B = np.array([1, 3, 5, 7, 9])
C = A + B
D = A – B
E = A * B
F = A / B
A_1 = A.astype(float)    #transforme tous les éléments de A en float
G = A_1 / B
print A.size             #A.size renvoie le nombre d'éléments de A.
```

Vous l'aurez compris, quand, par exemple, on additionne deux tableaux, ça additionne les éléments des tableaux case par case. En particulier, il faut faire bien attention à ce que les deux tableaux aient la même taille (essayez d'additionner deux tableaux de tailles différentes, ça va vous donner un erreur).

5)Sous-tableaux

Pour accéder à un sous-tableau, ça marche comme pour les listes :

```
import array as np
A = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
print A[3,6]      #affiche les éléments d'indices 3 à 5
print A[2:]       #affiche tous les éléments à partir de l'indice 2
print A[:4]       #affiche tous les éléments jusqu'à l'indice 3
```

Supposons maintenant qu'on a un tableau contenant les âges d'un groupe de personnes, et qu'on veuille savoir qui a plus de 18 ans. Selon les besoins, on peut faire les choses suivantes (étudier bien ce code, ces techniques sont très utiles, en particulier vous en aurez besoin dans l'exercice suivant):

```
import numpy as np
A = np.array([16, 58, 26, 3, 12, 21])
B = A > 18        #B est un tableau de booléens, les éléments valant True si la case correspondante est >18,
print B           #False sinon. En particulier B à la même taille que A.
C = A[A > 18]     #C est un sous-tableau de A, seul les cases >18 sont gardées
print C
D = A[B]          #B est un tableau de booléen de la même taille que A, D est le sous tableau de A
                  #correspondant aux cases de B valant True, donc D est pareil que C.
```

Exercice sur les tableaux : faire des statistiques

Télécharger le fichier **age_2013.txt** sur moodle. Ce fichier a été généré par l'INSEE est contient des données sur la population de certaines régions de France. La première ligne contient *régions* puis des catégories classées par genres et âges. Chaque item est séparé par une virgule. Les lignes suivantes contiennent le nom d'une région, puis le nombre d'individus par catégorie dans la dite région.
Créer un fichier **stat_fonctions.py** dans lequel vous écrirez toutes les fonctions demandées dans l'exo.
Créez aussi un fichier **stat_main.py** dans lequel vous utiliserez les fonctions codées dans **stat_fonctions.py**.

1) Dans *stat_fonctions.py*, écrire une fonction **read_data()** qui renvoie 4 tableaux : arr_regions, arr_categories, arr_men et arr_women.
arr_regions contient le nom de chaque région (tableau de dimension 1 avec 22 cases).
arr_categories contient le nom de chaque catégorie (tableau de dimension 1 avec 11 cases).
arr_men contient le nombre (type : **int**) d'hommes par catégorie et par région
arr_women contient le nombre (type : **int**) de femmes par catégorie et par région
Ces deux derniers tableaux sont donc de dimension 2, sur 5 colonnes (nombre de catégorie d'âges pour les hommes et pour les femmes) et 22 lignes (nombres de régions).
Nous vous conseillons de travailler sur des listes, puis de les transformer à la fin en tableau.
Voici à quoi doit ressembler votre script :

```
import numpy as np
def read_data() :
        lines = open("age_2013.txt").readlines()
```

```
    regions, categories, men, women = [], [], [], []     #on crée des listes vides qu'on transforme à la fin
                                                          #en tableau.

    # your code goes here

    arr_categories = np.array(categories)      #on transforme les listes en tableaux
    arr_regions = np.array(regions)
    arr_men = np.array(men)
    arr_women = np.array(women)
    return  arr_categories, arr_regions, arr_men, arr_women
```

2) Dans *stat_fonction.py*, écrire une fonction *total_by_regions* prenant en argument deux tableaux contenant respectivement le nombre d'hommes et de femmes par région (ces deux tableaux seront les tableaux *arr_men*, et *arr_women* renvoyés par la fonction *read_data()*) et qui renvoie un tableau *total* contenant le nombre total (type float) d'habitants par région (ce tableau sera donc de dimension 1 avec 22 cases (le nombre de régions). Pour convertir les éléments d'un tableau de int en float utiliser la fonction **astype(float)**.
Indice : en utilisant la fonctions **sum()**, et l'opérateur sur les tableaux +, on peut coder ce qui est demandé en une ligne.

```
def total_by_regions(arr_men, arr_women) :
    #your code goes here
    total = total.astype(float)
    return total
```

3) Dans *stat_fonction.py*, écrire une fonction *percentage_by_regions*, prenant en argument deux tableaux contenant respectivement le nombre d'hommes et de femmes par région (ces deux tableaux seront les tableaux *arr_men*, et *arr_women* renvoyés par la fonction *read_data()*) et renvoie deux tableaux contenant respectivement le pourcentage d'hommes et le pourcentage de femmes par région. Utilisez la fonction *total_by_regions*.

4)Dans *stat_main.py,* écrire un programme qui affiche la région avec le plus grand pourcentage d'hommes, et la région avec le plus grand pourcentage de femmes. Vous devez afficher :
Le plus grand pourcentage d'hommes est : *nombre* en : *nom de la région*
Le plus grand pourcentage de femmes est : *nombre* en : *nom de la région*
Vous devez bien sûr utiliser les fonctions de *stat_fonctions.py* votre programme doit donc commencer par

```
import stat_fonctions as stat
categories, regions, men, women = stat.read_data()
```