Introduction à la programmation, Leçon 3: fonctions, fichiers, modules

Réchauffement

1. Quels nombres le programme imprimera-t-il? Pourquoi?

```
i = 10
while i <= 20:
  if i % 2 == 0:
    print '%d is even' % I
    i = i + 1
  if i % 2 == 1:
    print '%d is odd' %i
    i = i + 1
```

2. Quel résultat attendez-vous? Comment nous pouvons corriger le programme ?

```
def sum(a,b)
  return a + b
a = 3
b = 5
print sum
```

Nouveau

3. Considérons à nouveau le problème de la comparaison de trois nombres entrés par un utilisateur à partir d'un clavier. Cette fois, nous aimerions considérer le moins de cas possible. Nous allons d'abord écrire une fonction `swap` qui reçoit deux variables a, b et retourne a, b si a < b et b, a si a >= b. Voici un exemple d'une fonction qui retournes deux variables :

```
def diff_sum(a,b):
  return (a-b, a+b)
a = 3
b = 2
diff, sum = diff_sum(a,b)
print diff
print sum
```

L'objet (a-b, a+b) s'appelle tuple. Ecrire un programme qui ordonne trois variables en utilisant la fonction `swap`. Il suffit de considérer trois cas différents.

4. **Lists**

Comme les chaînes de caractères, parfois nous voulons une chaîne de int ou une chaîne d'autre chose. En Python ces « chaînes » sont des **listes** (de type list), une suite de valeurs quelconques.

```
animals = ['giraffe', 'lion', 'monkey', 'cat']
taille = [5.0, 1.0, 0.7, 2.0]
mix = ['giraffe', 5.0, 'monkey', 2]
```

Il existe une deuxième façon de créer des listes qui est très pratique pour créer une liste en utilisant une boucle.

```
animals = []
animals.append('giraffe')
animals.append('lion')
animals.append('monkey')
animals.append('cat')
```

On peut rappeler ses éléments par leur numéro de position indice. Les indices d'une liste de n éléments commencent à 0 et se terminent à n − 1.

```
print animals[0], animals[1], animals[-1]
```

L'instruction len vous permet de connaître la longueur d'une liste.

```
print len(animals)
```

Il arrive souvent que nous voulons une boucle pour traverser les éléments d'une liste ou chaîne. L'instruction for permet de simplifier.

```
l = ['a', 'b', 'c', 'd']
for element in l:
    # Do something with the element
```

5. Écrire une fonction qui prend une liste en argument et détermine si elle contient un doublon.
6. Vous pouvez considérer un chaîne de caractères comme un liste de caractères. Écrire un programme qui affiche tous les caractères d'une chaîne de caractères en mettant un par ligne.
7. **Files**

```
file_name = "example.txt"
in_file = open(file_name,'r')#Open example.txt for reading
content = in_file.read()
print content
in_file.close()
```

Créez un fichier example.txt. Dans le même dossier, créez un script qui lit example.txt et imprime le contenu de example.txt. Essayez de faire la même chose quand example.txt est dans un dossier différent.

```
in_file = open(file_name,'r')#Open file_name for reading
lines = in_file.readlines()
for line in lines:
  # Do something with the line
in_file.close()
```

Parfois, il est plus pratique de travailler avec des fichiers ligne par ligne. Créez un fichier example.txt, où chaque ligne contient un entier. Dans le même dossier, créez un script qui lit example.txt et imprime tous les entiers du fichier appartenant à l'intervalle [10,20].

```
out_file = open(file_name,'w')#Open file_name for writing
animals = ['elephant', 'cat', 'dog']
for animal in animals:
  out_file.write(animal)
out_file.close()
```

Corrigez ce script afin qu'il écrive le nom de chaque animal sur une ligne distincte. Si le fichier de sortie existe déjà et que vous voulez ajouter quelque chose à la fin, vous devez utiliser le mode 'a' (append) au lieu de 'w'.

8. **Modules**

Les développeurs Python ont implémenté de nombreuses fonctions utiles, et vous devriez les utiliser. Les fonctions sont regroupées dans des collections appelées «modules». Par exemple, il y a un module math qui contient différentes fonctions mathématiques (https://docs.python.org/2/library/math.html) , il y a un module re qui vous permet de travailler avec des expressions régulières (https://docs.python.org/2/library/re.html), et ainsi de suite.

```
from math import *
root = sqrt(49)
angle = pi/6
print sin(pi/6)
```

Utilisez le module math pour calculer le logarithme de 27 en base 3.

Exercices plus difficiles

9. **Cent mille milliards de poèmes**

   Cent mille milliards de poèmes, est une œuvre de poésie combinatoire de Raymond Queneau, publiée en 1961. Le livre est composé de dix feuilles, chacune séparée en quatorze bandes horizontales, chaque bande portant sur son recto un vers. Le lecteur peut donc, en tournant les bandes horizontales comme des pages, choisir pour chaque vers une des dix versions proposées par Queneau. Les dix versions de chaque vers ont la même scansion et la même rime, ce qui assure que chaque sonnet ainsi assemblé est régulier dans sa forme. Supposons que l'on dispose d'un fichier qui contient une séquence de 14 groupements de 10 vers séparés par des lignes vides (téléchargez ici : http://bit.ly/2H4OhKw).

   1. Écrire un script Python chargeant le fichier dans une liste associant aux entiers de 0 à 13, la liste des vers de chaque groupement.
   2. Modifier le script pour générer un poème de 14 vers en prenant un vers aléatoirement dans chaque groupement. Utiliser pour cela le module `random` de la bibliothèque standard de Python.

10. **Générateur de texte**

    Écrire une fonction Python qui génère une liste des mots consécutifs d'un texte. (Supposer que "mots" veux dire des chaines de caractères séparés d'espaces.) Sauvegarder cette fonction dans un module. Importer ce module. Écrire un programme Python qui prend le nom d'un fichier en entrée, lit le texte dans ce fichier et affiche les paires de mots consécutifs, une par ligne.

11. **Interpolation linéaire**

    Lorsque nous couvrirons les bibliothèques numériques, nous verrons qu'elles incluent de nombreuses alternatives pour l'interpolation et l'approximation d'une fonction. Néanmoins, écrivons notre propre routine d'approximation de la fonction comme un exercice. En particulier, sans utiliser d'import, écrivez une fonction linapprox qui prend comme arguments

    1. Une fonction `f:` $[a,b] \rightarrow$ ‘
    2. Deux nombres `a` et `b`
    3. Un nombre `n`
    4. Un nombre `x` , `a <= x <= b`

    et retourne l'interpolation linéaire par morceaux de f en x, basée sur n points de grille uniformément espacés `a = point[0] < point[1] < ... < point[n-1] = b.` Plus d'information : http://bit.ly/2FewaW7

12. **(*) Distribution binomiale et portefeuille obligataire**

    La variable aléatoire binomiale Y~Bin (n, p) représente le nombre de succès dans n essais binaires, où chaque essai réussit avec la probabilité p. En n'important que `from numpy.random import uniform`, écrire une fonction binomial_rv telle que

binomial_rv (n, p) génère un tirage de Y. Utilisez la fonction binomial_rv pour modéliser la situation suivante. Imaginons que nous ayons un portefeuille obligataire composé de trois obligations ayant chacune un solde de capital de 1 000 000 $ et ayant chacune une date d'échéance d'un an. Si, à l'échéance, l'obligation n'est pas en défaut, nous recevrons le solde du principal majoré de 15,00% d'intérêt simple. Si, à l'échéance, l'obligation est en défaut, nous recevrons le recouvrement de cette obligation qui devrait représenter 40% du solde du capital. Chaque obligation a une probabilité de défaut annuelle de 0,10 et les valeurs par défaut sont indépendantes. Votre script doit générer la valeur du retour total (qui est une variable aléatoire), dans un an.

Astuce: Si U est uniforme sur (0,1) et pa (0,1), alors l'expression U <p vaut True avec la probabilité p.