Initiation à la programmation, Leçon 2 : Listes et fonctions

Réchauffement : variables, boucles, if

1. Est-ce que ce script affiche « She said, "Hello world" » ?

```
prin("She said, "Hello world"")
```

2. Modifiez ce script pour qu'il affiche les chiffres 0, 2, 4.

```
i = 0
while i < 5:
  i = i + 1
if i % 2 = 0:
  print i
```

3. Est-ce que ce script s'arrête ? Modifiez-le pour qu'il affiche les chiffres 0-4.

```
firstnumber = 0
secondnumber = 2
while firstnumber < 5:
  print firstnumber
  secondnumber = secondnumber + 1
```

4. Quel résultat vous attendez ? Corrigez le script.

```
#V = Voltage in volts, I = Current in amps, R = Resistance in ohms
#Ohm's Law: V = I R
V = 5
I = 2
print "R = ", V/I
```

Nouveau : else, listes, fonctions

1. Une condition est simplement un morceau de code qui peut produire une réponse vraie ou fausse. La meilleure façon de comprendre comment les conditions fonctionnent en Python est d'essayer quelques exemples.

```
print (3 == 5)
print (3 < 5)
print ("banana"  == "banana")
print ("banana"  == "banana" and 3 < 5)
print ("banana"  == "banana" and 3 == 5)
print ("banana"  == "banana" or 3 == 5)
```

2. L'instruction conditionnelle le plus simple est une instruction if. En utilisant if, nous pouvons faire une simple décision oui / non: si l'énoncé est vrai, faites quelque chose. La clause else est étroitement liée à l'instruction if. Souvent, nous avons besoin d'une décision de type soit … ou, où nous avons deux actions possibles à prendre. Pour ce faire, nous pouvons ajouter une clause else après la fin du corps d'une instruction if.

```
expression_level = 125
```

```
if expression_level > 100:
    print("gene is highly expressed")
else :
    print("gene is lowly expressed")
```

3. Écrire un programme qui affiche : « 10 has remainder 1 modulo 3, 11 has remainder 2 modulo 3, 12 has remainder 0 modulo 3, …, 20 has remainder 2 modulo 3 ». Utilisez if/else et while.
4. À partir de trois entiers a, b et c entrés au clavier par l'utilisateur, écrire un programme qui les affiche dans l'ordre croissant. Pour entrer une variable au clavier, vous pouvez utiliser :

```
a = raw_input("Enter a: ")
```

5. Les fonctions sont un moyen pratique de diviser votre code en blocs utiles, ce qui nous permet de commander notre code, de le rendre plus lisible, de le réutiliser et de gagner du temps. Les fonctions sont également un moyen clé pour définir les interfaces afin que les programmeurs puissent partager leur code. Les fonctions sont définies à l'aide du mot-clé « def », suivi du nom de la fonction en tant que nom du bloc. Par exemple:

```
def my_function():
    print("Hello From My Function!")
```

Les fonctions peuvent également recevoir des arguments (variables transmises par l'appelant à la fonction). Par exemple:

```
def my_function_with_args(username, greeting):
    print("Hello, %s , From My Function!, I wish you %s"%(username, greeting))
```

Les fonctions peuvent renvoyer une valeur à l'appelant, en utilisant le mot-clé «retour».

```
def sum_two_numbers(a, b):
    return a + b
```

Pour appeler une fonction, écrivez le nom de la fonction suivi de (), en plaçant tous les arguments nécessaires entre les parenthèses.

```
c = sum_two_numbers(3, 4)
```

6. Ecrire une fonction qui reçoit la tension V et le courant I et renvoie la résistance R. Calculer la résistance pour V = 5, I = 2 ; V = 1, I = 0.2 ; V = 3, I = 0.5 ; V = 0.4, I = 5.
7. 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 quelconque.

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

Une deuxième façon de créer des listes existe 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 indice d'une liste de n éléments commence à 0 et se termine 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
```

8. Écrire une fonction qui prend une liste en argument et détermine si elle contient un doublon.
9. 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.

Exercices plus difficiles

1. Cryptographie

- Créez la chaîne de caractères message contenant la valeur 'ceci est mon message a chiffrer'. A l'aide d'une boucle for sur cette chaîne, chiffrer la par un décalage de 3 (chiffre de César). Par exemple la lettre a sera chiffrée en la lettre d (et la lettre x en la lettre a).

  Vous pouvez convertir une lettre en un nombre en utilisant `ord(letter)` et un nombre en une lettre en utilisant `chr(letter)`. Plus d'information : http://www.asciitable.com/

- Écrire un script Python qui chiffre (ou déchiffre selon le choix de l'utilisateur) une chaîne de caractères entrée au clavier avec une clé (i.e.un décalage) choisi aussi par l'utilisateur. On définira les fonctions chiffrer et dechiffrer. Attention, la chaîne peut contenir des caractères non alphabétiques. Pour verifier si un caractère c est un nombre vous pouvez utiliser `c.isalpha()`.

```
$ python chiffeur.py
Entrer le texte a chiffrer/dechiffrer : ceci est un autre message a chiffrer
Entrer le decalage (0 a 26) : 12
Entrer 'c' pour chiffrer ou 'd' pour dechiffrer : c
oqou qef gz mgfdq yqeemsq m oturrdqd

$ python chiffeur.py
Entrer le texte a chiffrer/dechiffrer : oqou qef gz mgfdq yqeemsq m oturrdqd
Entrer le decalage (0 a 26) : 12
Entrer 'c' pour chiffrer ou 'd' pour dechiffrer : d
ceci est un autre message a chiffrer
```

- Modifier le script précédent pour qu'il utilise la méthode de chiffrement de Vigenère : la clé est désormais une chaîne de caractères et le chiffrement se fait en décalant la i-ème lettre du message grâce à la i-ème lettre de la clé (on reprend au début de la clef quand on a fini de lire celle-ci) suivant la règle naturelle A=1,B=2,...,Z=26(=0).

- Modifier le script précédent pour qu'il chiffre/déchiffre un fichier dont le chemin est entré par l'utilisateur. On pourra utiliser cette nouvelle de Edgar Poe pour tester le script: http://bit.ly/2Cvpfl5. Pour obtenir le contenu d'un fichier a partir de son nom, vous pouvez utiliser :

```
nom_fichier = "Poe.txt"
contenu_fichier = open(nom_fichier).read()
```

2. **Cryptanalyse**

   Comme expliqué dans la nouvelle de E. Poe, les chiffrements par substitution alphabétique peuvent être casser facilement par une analyse fréquentielle. Par exemple, dans un texte écrit en langue française, la lettre la plus fréquente est généralement le "E" et puisqu'un chiffrement de César ne modifie pas les fréquences, la lettre qui apparaît le plus fréquemment dans le texte chiffré correspond vraisemblablement à "E" et si c'est le cas le décalage entre les deux lettres donne la clé et permet de retrouver l'intégralité du message clair.

- Écrire une fonction Python qui prenant en entrée une chaîne de caractères (ou un fichier texte) affiche la fréquence des caractères qui le composent.
- Modifier le programme précédent pour qu'il affiche la lettre qui apparaît le plus de fois.
- Utiliser cette fonction pour créer une fonction qui prend en entrée un texte chiffré avec le chiffrement de César (mais pas la clé) et retourne un texte clair associé. On pourra demander à l'utilisateur de valider que ce texte est correct (et si ce n'est pas le cas proposer un nouveau texte clair probable).