université PARIS-SACLAY

M1 BIBS

Langages de scripts : Python

Séance 1 - Affichage

(Correction)

1 Objectifs :

- Savoir utiliser les boucles et les conditions en Python
- Savoir utiliser la fonction `print` pour afficher dans une console ou dans le terminal
- Savoir manipuler les chaînes de caractères
- Comprendre que Python est un langage interprété
- Comprendre que Python fait du passage de variable par référence

2 Mon 1er programme en Python !

1. Hello World ! Écrivez simplement le code suivant dans un fichier avec l'extension .py :

```
print 'Hello World !'
```

   2 solutions pour l'exécuter :
   - Si vous utilisez un IDE dédié (ex : PyCharm, le module pour Eclipse PyDev, DrPython ou encore Jupyter Notebook), il vous suffit de trouver le bouton pour exécuter votre programme.
   - Si vous utilisez un éditeur de texte classique, il vous suffit d'exécuter dans un terminal votre script grâce à la commande `python` : `python monProgramme.py`

   Commentaires :
   Sauvegardez votre code dans des dossiers bien rangés ! Et de ne pas supprimer ce que vous avez fait ! Ca vous servira... Je ne veux pas voir des bouts de fichiers incomplets se baladant dans le même dossier !

   Remarque : En Python, il n'y a pas de "`main`" : chaque fichier peut être exécuté comme étant le script principal. Nous y reviendrons lorsque nous verrons comment séparer un programme Python en plusieurs fichiers.

2. La fonction `print` : cette fonction est bien pratique, parce qu'elle affichera dans la console ou le terminal n'importe quel objet standard du Python (int, float mais aussi liste, dictionnaire, ...), sans avoir à préciser leur type. Pour vous en convaincre, essayez le code suivant :

```
print 'Hello World !' #string
print 1 #char
print 2.3 #float
print [4,5,6] #liste
```

   Attention : Alors que l'on peut simplement concaténer 2 chaînes de caractères avec l'opérateur `+`, si vous essayez de concaténer une chaîne de caractères avec un autre type, vous obtiendrez une erreur. Le code suivant est en effet incorrect :

```
print 'Hello World ' + 2.0
```

   Dans ce cas, utilisez la fonction `str(*)` pour caster un nombre en chaine de caractères.

3. Les commentaires : Si vous voulez commenter une ligne, il suffit de mettre le caractère # devant. Si vous voulez commenter plusieurs lignes, il faudra alors les encadrer par des triples guillemets """.
   Au tout début de votre fichier, écrivez ces 2 lignes de commentaires :

```
#!/usr/bin/env python
#-*- coding: utf8 -*-
```

   Ce n'est pas forcément utile de savoir à quoi elles servent (ce ne sont pas de vrais lignes de commentaires en réalité), mais dites-vous qu'il faut toujours mettre ça au tout début de votre script.

Commentaires :
Le premier, c'est pour trouver votre exécutable python, car des fois, il n'y arrive pas tout seul. Le 2ème, c'est pour préciser l'encodage de caractères utilisés (et l'utf8, c'est LA norme !). C'est important, car même votre code est encodé d'un certaine manière. Attention, parfois syntaxe # coding : utf-8 (sans doute correcte aussi). Pour l'encodage, voir ici.

   Créer ensuite en dessous une zone de commentaire dans laquelle vous indiquerez l'auteur du script, une description rapide du programme et/ou autres informations générales (licence, date, etc.).
   Commencez toujours un script Python de cette manière!!!

4. L'indentation : L'indentation de votre code est FONDAMENTALE en Python!!! C'est elle qui permettra d'interpréter correctement vos scripts. Pour vous en convaincre, essayez les 2 versions suivantes (nous allons revenir rapidement sur la syntaxe d'une boucle `for` en Python, ce n'est pas ce qui est important ici) :

```
for i in range(20):
    print 'a',
    print 'b',
```

   et

```
for i in range(20):
    print 'a',
print 'b',
```

   Remarque : Vu qu'il n'y a pas de séparateurs syntaxiques tels que {}, coder en Python exige de coder un minimum proprement !

3 Les variables et les listes

1. La déclaration de variable est relativement simple en Python : la déclaration se fait en initialisant la variable et il n'y a pas besoin de préciser son type, car c'est reconnu automatiquement. Pour vous en convaincre, essayez ce code :

```
a=15
print a, type(a)
b=2.34
print b, type(b)
c="Hello"
print c, type(c)
```

   Remarque : la fonction `type(*)` permet de donner le type d'une variable ou d'un objet. C'est parfois utile pour s'y retrouver.

2. Introduction aux listes : En Python, les listes ont un rôle central. Pour faire une boucle `for` par exemple, il sera nécessaire de parcourir une liste. Une liste est une collection ordonnée d'objets/variables. Tous les éléments d'une liste peuvent être d'un type différent. Il existe plusieurs façon de déclarer une liste que vous pouvez tester avec l'exemple ci-dessous :

```
la=list()
lb=[]
lc=[1,2.0,"trois"]
print la,lb,lc,type(la)
```

3. Introduction à la manipulation de listes : Il existe de nombreuses méthodes pour manipuler les listes. En voici quelques-unes à tester :

```
l1=[1,2,3]
print len(l1)
l1.append(5)
print l1
l1[3]=4
print l1

l2=list(l1)#Attention : différent de l2=l1
l3 = range(10)
l4 = range(2,7)
l5 = range(3,40,7)
l6 = l4+l5
print l2,l3,l4,l5,l6
```

Commentaires :
Avec insert(), il y a ça aussi en bonus :
del l3[2]#del permet de supprimer n'importe quelle variable de la mémoire
print l3
l3.remove(5)# Attention, ne retire que la 1ère occurrence trouvée
print l3

4. Le Python est un langage qui passe une partie de ses objets par référence (à la différence du C qui le fait par valeur par exemple). Cela signifie que quand vous faites ceci :

```
liste1=[1,2,3]
```

Vous n'assignez pas une liste à la variable liste1, vous assignez une référence vers la liste, donc une sorte d'adresse qui indique où elle se trouve en mémoire. Pour comprendre cette différence, exécutez le code suivant :

```
a=[1,2,3]
b=a
b.append(4)
print a
```

Remarque : cela est important à garder dans un coin de son esprit pour la manipulation des objets dits "mutables" (listes, dictionnaires, sets, objets personnalisés, etc.), surtout lorsque l'on les utilisera comme arguments dans des fonctions. Pour les objets dits non-mutables, qui sont à peu près équivalents aux variables primitives dans d'autres langages (int, float, strings, tuples, ...), cela ne s'applique pas. Il est toutefois possible de simuler un passage par référence avec ces derniers en utilisant la méthode suivante :

```
c=[13]#Au lieu de manipuler un int, on manipule une liste d'un élément de type int
d=c
print c,d
c[0]=14
print c,d
```

Commentaires :
En savoir plus sur ça : http://sametmax.com/valeurs-et-references-en-python/

4 Boucles et conditions

1. La boucle `for` : Bien que la boucle `while` existe en Python, concentrons-nous sur celle que l'on appelle la boucle `for`. En Python, une boucle `for` permet de parcourir tous les élément d'une séquence (liste ou tuple). Par exemple, on utilise classiquement la fonction `range(*)` pour incrémenter un indice entier. Essayez le code suivant :

```
for i in range(10):
    print i,
```

Ou, autre exemple, avec une liste de chaines de caractères :

```
liste = ["pierre","feuille","ciseaux"]
for w in liste:
    print w,
```

2. Affichage : Afficher une chaîne de caractères 'tata...tata' de 100 caractères de long. Afficher ensuite une chaîne de caractères 'titi...titi' de 50 caractères de long. A partir des variables ta = 'ta' et ti = 'ti', afficher une chaîne de caractères 'tati' de longueur 100.

Correction : *La solution suivante n'est pas vraiment ce qui est demandée (car génère des espaces) :*

```
for i in range(49):
    print "ta",
```

Le mieux est de passer par une variable (ou des fonctions particulières de la librairie sys si on veut vraiment faire compliqué) :

```
ta=''
for i in range(49):
    ta+='ta'
print ta
```

---

3. Listes et chaînes de caractères : Soit la séquence abc = 'abcdefghijklmnopqrstuvwxyz'. Afficher une lettre sur deux. Afficher la séquence dans l'ordre inverse. Soit la liste listeabc = ['a', 'b', 'c',...,'y', 'z']. Inverser puis afficher listeabc. Que remarque-t-on au niveau de l'affichage ? Chercher une méthode sur internet pour l'afficher comme une chaîne de caractères.

Correction :

```
abc = 'abcdefghijklmnopqrstuvwxyz'
result = ''
#Plusieurs solutions, en voici une :
for i in range(len(abc)/2):
    result+=abc[i*2]
print result
\begin{lstlisting}

result = ''
for i in range(len(abc)):
    result+=abc[len(abc)-i-1]
print result

listeabc = ['a', 'b', 'c','y','z']
print listeabc
listeInv=[]
for i in range(len(listeabc)):
    listeInv.append(listeabc[len(listeabc)-i-1])
print listeInv

print "".join(listeInv)
```

Commentaires :
Rappels : Allez chercher sur internet !

---

4. Exercice de manipulation de boucles : Afficher les triangles ci-dessous en utilisant la boucle `for` :

Correction :

```
res=""
for i in range(10):
    res+="*"
    print res

#Solution compacte : (le symbole * marche aussi !)
res=""
for i in range(10):
    res=" "*(10-i-1)
    res+="*"*(i+1)
    print res

#idem en inversant simplement "*" et " "

res=""
for i in range(10):
    res=" "*(10-i-1)+"*"*(2*i+1)+" "*(10-i-1)
    print res
```

---

5. Les conditions : En Python, on peut naturellement utiliser des conditions `if/else` comme dans la plupart des langages de programmation. Essayez le code suivant :

```
note=14
if 0<=note<8:
        print "pas tip top"
elif 8<=note<12:
        print "passable"
elif 12<=note<20:
        print "bien"
elif note==20:
        print "félicitations"
elif not(isinstance(note,int)) and not(isinstance(note,float)):
    print "Erreur : ce n'est pas un nombre !"
elif note<0 or note>20:
        print "erreur, note invalide"

if True:
    print "les booléens existent en Python !"
elif False:
    print "Ce print ne sera jamais affiché..."
else:
    print "idem mais vous saurez que le else existe aussi !"

#Une condition typique en Python et très utilisée :
uneListe=[1,2,3]
if 2 in uneListe:
    print "coucou"
```

6. Exercices sur la manipulation de boucles et de conditions : Soit a = [8, 4, 12, 3, 7, 2]. Écrire un script pour identifier la valeur minimale de a sans utiliser la fonction `min()`. Écrire un script pour trier a sans utiliser la fonction sort() (mais vous pouvez utiliser la fonction `min()` ici).
   Soit b = [4, 10, 4, 8, 3, 19, 1, 7, 7, 19, 11, 3, 21, 7]. Écrire un script qui retourne une liste non redondante de b.

Correction :

```
a = [8, 4, 12, 3, 7, 2]
mini=a[0]
for x in a:
    if mini>x:
        mini=x
print mini

res=list()
for i in range(len(a)):
    res.append(min(a))
    a.remove(min(a))
print res

lb = [4, 10, 4, 8, 3, 19, 1, 7, 7, 19, 11, 3, 21, 7]
lc = list()
```

```
for b in lb:
    if b not in lc:
        lc.append(b)
print lc
```

7. Importations : Nous y reviendrons plus en détail dans une prochaine séance, mais si vous voulez importer une des librairies standards en Python, il suffit d'écrire avant votre code :

```
import LibName
```

Une des librairies fréquemment utilisée est la librairie `random`, qui vous permet notamment d'utiliser la fonction `randint`.

Générer une séquence aléatoire de nucléotides de longueur 100 puis retourner la séquence complémentaire.

Générer une séquence aléatoire d'acides aminés de longueur 200 et calculer les proportions de chaque acide aminés.

Correction :

```
# On reverra les importations plus tard...
import random
nucl=["A","T","G","C"]
lnucl=list()

for i in range(100):
    indAlea = random.randint(0,3)
    lnucl.append(nucl[indAlea])
print lnucl

nA = 0
nT = 0
nG = 0
nC = 0
for nu in lnucl:
    if nu=="A":
        nA+=1
    if nu=="T":
        nT+=1
    if nu=="G":
        nG+=1
    if nu=="C":
        nC+=1
print nA,nT,nG,nC
```

5 Jeu de dés et boucle `while`

Écrire un script qui simule le lancement d'un dé tant que la valeur 6 n'est pas obtenue. Retourner le nombre de lancers nécessaires. Idem pour la condition : obtention de deux 6. Idem pour la condition : obtention de deux 6 consécutifs.

Correction :

```
import random
res=0
while random.randint(1,6)!=6:
    res+=1
print res

res=0
mq=0
while mq!=2:
    nb=random.randint(1,6)
    if nb==6:
        mq+=1
    res+=1
print res
```

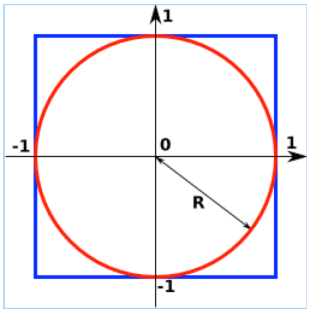
```
res=0
mq=0
while mq!=1:
    nb=random.randint(1,6)
        res+=1

    if nb==6:
        nb2=random.randint(1,6)
                res+=1

        if nb2==6:
            mq=1
print res
```

6 Calcul du nombre π

Nous allons tenter d'estimer la valeur de π géométriquement. Ici le carré a un côté de 2 et son aire est donc de 4. On trace un cercle de rayon 1 à l'intérieur du carré. Que vaut l'aire du cercle et celle du carré ?



En tirant aléatoirement un grand nombre de points à l'intérieur du carré, on peut calculer la proportion de points tombés dans le cercle. Comment en déduire π ? Écrire un programme tirant aléatoirement un grand nombre de points (100, 10000, 100000) afin d'approximer géométriquement la valeur de π.

Correction :

```
import random
from math import sqrt #pour importer une fonction particulière

cmptCercle=0
cmptCarre=0
for i in range(10000):
    x=random.uniform(-1,1)
    y=random.uniform(-1,1)
    hyp=sqrt(x*x + y*y)
    if hyp <=1:
        cmptCercle+=1
    else:
        cmptCarre+=1

print cmptCercle*4.0/(cmptCarre+cmptCercle)
```

Commentaires :
Si on fait AireCercle/AireCarré, c'est égale ici à Pi/4, et ça devrait approcher le taux nbPointInCercle/nbPoint. En passant le 4.0, on devrait donc approcher la valeur de Pi donc.