TP N°1 – Introduction à la programmation Python

1. Prise en main de l'environnement

Vous allez travailler sous Unix. Avant toute chose, il faut vous connecter à la Machine Virtuelle « UFR des Sciences Linux ». A priori, mais cela peut dépendre des installations, les raccourcis des différentes applications dont nous nous servirons dans ce module (LXTerminal, Chrome, FireFox, *etc.*) sont sur le bureau. Si aucun raccourci n'est présent, un menu en bas à gauche (comme dans Windows) est accessible. Vous y trouverez les applications dont vous avez besoin.

Pour commencer, nous ferons connaissance avec votre environnement de travail :

1.1. La fenêtre terminal

Une fenêtre terminal est l'accès non graphique au système Unix via des commandes interprétées par un shell.

1.2. Premiers utilitaires

Après avoir ouvert un terminal, nous allons manipuler quelques utilitaires simples.

Exercice 1 - Utilisez les commandes **date**, puis **cal**. Que font-elles ?

2. Python

De nombreux langages de programmation permettent de développer des applications (ou programmes) bio-informatiques. Nous utiliserons ici le langage Python qui a été conçu par Guido van Rossum en 1991 car il est gratuit, il fonctionne sous Windows, Linux et MacOS, il est assez facile à maîtriser, il peut être utilisé en mode interactif ou non, et il est très répandu en bio-informatique et génomique. C'est pour l'ensemble de ces raisons que nous allons apprendre à nous en servir.

Exercice 2 - Pour lancer le mode interactif de python, il suffit de taper **python** dans un terminal. Pour quitter python dans ce même terminal il faudra taper **Ctrl + d**. Tester les différentes opérations suivantes, que constatez-vous ?

```
5 + 3

2 - 9

34 + 20 * 4

(34 + 20) * 4

20 / 3

 20 % 3

20.0 / 3.0

20 / 3.0
```

4 * 5.7 / 3.0

Exercice 3 - Testez les lignes d'instructions suivantes et répondez aux questions.

Que remarquez-vous sur le contenu de la variable altitude en tapant les lignes suivantes ?

```
>>> altitude = 320
>>> print altitude
>>> altitude = 375
>>> print altitude
```

Que remarquez-vous sur le contenu des variables a et b en tapant les lignes suivantes ?

```
>>> a = 5
>>> b = a
>>> print a, b
>>> b = 2
>>> print a, b
>>> a = 4
>>> print a, b
```

Qu'affichent ces lignes de codes ?

```
a = 40
a = a + 1
print a
a = a - 1
print a
```

Exercice 4 - Affectez les valeurs respectives 3, 5, 7 à trois variables a, b, c. Effectuez l'opération a - b/c. Le résultat est-il mathématiquement correct ? Si ce n'est pas le cas, comment devez-vous procéder pour qu'il le soit ?

Exercice 5 : Ecrivez les lignes de code qui permettent d'échanger les valeurs de 2 variables. Par exemple si a vaut 5 et b vaut 32, on veut obtenir après l'échange : a vaut maintenant 32 et b, 5. Attention, les valeurs 5 et 32, sont au départ affectées aux deux variables, vous devez par la suite effectuer uniquement des manipulations sur les variables afin d'arriver à échanger les 2 valeurs.

Exercice 6 - Testez les lignes d'instructions suivantes. Décrivez ce qu'il se passe :

```
>>> r = 12
```

```
>>> pi = 3.14159

>>> s = pi * r * r

>>> print  s

>>> print type(r), type(pi), type(s)
```

Quel est le rôle de type ?

Exercice 7 - Qu'affichent ces lignes de codes ?

```
ch = "Stephanie"

print type(ch)

print ch[0], ch[3]
```

Puis idem en tapant :

```
a = 'Petit poisson'

b = ' deviendra grand '

c = a + b

print "concatenation = ", c
```

Exercice 8 :- Initialiser les variables seq1 et seq2 au moyen de deux chaînes nucléidiques comme par exemple ci-dessous :

```
seq1 = 'agcgccttgaattcggcaccaggcaaatctcaaggagaagttccggggagaaggtgaaga'

seq2 = 'cggggagtggggagttgagtcgcaagatgagcgagcggatgtccactatgagcgataata'
```

Puis taper les lignes de code suivantes et observez ce qu'il se passe.

```
seq = seq1 + seq2

print seq[1000]

print  seq[0]

print seq[0:3]

print seq[:3]

print seq[3:6]

print seq[3:]

print seq[:]
```

Que pouvez-vous conclure sur l'utilisation des crochets [] ?

Exercice 9 – Taper les lignes de code suivantes et observer les résultats.

2 > 8

(3 == 3) or (9 > 24)

(9 > 24) and (3 == 3)

3 != 2

Que pouvez-vous dire sur >, ==, !=, and, or ?

Exercice 10 : Passons aux comparaisons de chaînes de caractères :

'atgc' == 'atgc'

'atgc' == 'gcta'

'atgc' >= 'gcta'

'atgc' <= 'gcta'

'atgc' == 'ATGC'

'atgc' <= 'ATGC'

'atgc' >= 'ATGC'

'at' != 'ct'

Que pouvez conclure sur chaque opérateur et sur l'ordre entre majuscules et minuscules ?

Exercice 11 :

seq1 = "atgactagctagctatg"

'n' in seq1

'a' in seq1

'at' in seq1

seq2 = "cta"

seq2 in seq1

Que fait l'instruction **in** ?

Exercice 12 - Un petit peu d'affichage maintenant :

codon1 = 'ATG'

```
codon2 = 'GCC'

print codon1,codon2

print codon1 + codon2
```

Qu'elle est la différence entre les deux affichages de print ?

```
percGC = ((4502.0 + 2567)/15003)*100

print "Le pourcentage de GC est",percGC,"%"
```

Que remarquez-vous sur la syntaxe de print ?

ATTENTION : la fonction **print** comme tout ce qui sera tapé dans python ne tolère ni les accents, ni les caractères spéciaux !!

```
nbG = 4502

nbC = 2567

percGC = ((4502.0 + 2567)/15003)*100

print "Ce genome contient",nbG,"G et ",nbC," C, soit un % de GC de ",percGC ,"%"
```

Apprenez et retenez dorénavant comment utiliser la fonction print de la façon la plus claire possible pour renseigner au mieux l'utilisateur …

Exercice 13 - Approfondissement sur l'utilisation de chaînes de caractères

Il existe en python, un certain nombre de fonctions prédéfinies que l'on peut appliquer à des chaînes de caractères. On va découvrir ici un certain nombre d'entre elles.

1. Commençons par la fonction **len**. Que fait le code suivant ?

```
seq = 'atgctgggctggctatttg'

print len(seq)
```

2. A l'aide de cette nouvelle fonction, afficher les 5 derniers caractères de la séquence. Plusieurs solutions sont possibles. Proposez-en au moins deux.

3. Et si on changeait l'aspect des chaînes de caractères ? Vous essaierez de comprendre ce que font les différents bouts de codes suivants : (faites attention a bien conserver la syntaxe donnée dans les instructions, notamment les parenthèses vides !!!)

```
seq1 = 'atgctg'
seq2 = seq1.upper()
print seq1, seq2
seq1 = 'ATGCTG'
seq2 = seq1.lower()
print seq1, seq2
seq1 = 'atgctgggctGGctatttg'
seq2 = seq1.capitalize()
print seq1, seq2
seq1 = 'atgctGGGggATATAAatttg'
seq2 = seq1.swapcase()
print seq1, seq2
```

4. Et si l'on regardait un peu le contenu de la chaîne ?

```
seq = 'atgctgggctggctatttg'
codon = 'atg'
print seq.find(codon)
seq = 'ttgctgggctggctatgttg'
print seq.find(codon)
seq = 'ttgctgggctggctatttg'
print seq.find(codon)
```

Que fait la fonction **find** ?

5. Nous allons utiliser d'autres fonctions …

```
seq = 'atgctgggctggctatgttg'
codon = 'atg'
print seq.count(codon)
seq = 'ttgctgggctggctatttg'
print seq.count(codon)
```

Que fait la fonction **count** ?

6. Changeons un peu la chaîne maintenant ....

```
seq1 = 'atgcgggctcgattgctaatat'

seq2 = seq1.replace('a','o')

print seq1, seq2

seq2 = seq1.replace('a','OP')

print seq1, seq2
```

Que fait la fonction **replace** ?

Exercice 14 - Nouveaux opérateurs numériques

A quoi correspondent les opérateurs numériques ** et % ? Pour avoir une idée, on saisira dans l'interpréteur Python les instructions suivantes :

```
a = 2**2

print "a = ", a

a = 2**3

print "a = ", a

a = 10 % 5

print "a = ", a

a = 10 % 3

print "a = ", a
```

Exercice 15 - Règles de priorités des opérateurs

1. Quel est le résultat de l'opération suivante ?

```
a = 2.0-4.0**2/2.0/7.0-5.0
```

2. Quel est le résultat de l'opération suivante ?

```
a = (2.0-4.0)**2/(2.0/(7.0-5.0))
```

D'après vous à quoi servent les parenthèses ?