



Cours d'introduction aux outils Bioinformatiques

Introduction à Python:

- Langage de programmation généraliste vs R focalisé sur la statistique et la data science
- Au départ, créer par un informaticien pour les informaticiens à l'opposé de R qui a été créé par un Statisticien. Ce qui a alimenté l'éternel débat du qui est le plus rapide? R vs Python

R Software Foundation	Python
	

- A servi à coder le **moteur de recherche de Google**.



- Depuis plusieurs décennies en concurrence avec R pour l'analyse de données.
- Classé 4^{ième} et 2^{ième} langage de programmation la plus populaire dans le monde mesuré par différentes métriques (<http://www.developpez.com/actu/93779/Classements-TIOBE-et-PYPL-Python-gagne-en-popularite-mais-Java-demeure-le-leader-incontestable/>) : Classements TIOBE et PYPL
- Nettement plus fourni et utilisé pour la programmation autour du GIS.

Nous allons nous focaliser sur les librairies [numpy](http://www.numpy.org/) (<http://www.numpy.org/>), pandas et Biopython dans ce cours et réaliser quelques applications avec ces librairies là.

Numpy:

Est une librairie python de calcul scientifique très performant en tant de calcul

Pandas:

Est une [librairie](http://pandas.pydata.org/) (<http://pandas.pydata.org/>) de manipulation de données et d'analyse haute performance basée sur numpy et qui utilise une syntaxe similaire à la manipulation de data.frame dans R.

Scikit-Bio et Biopython:

Librairies de référence en Bioinformatique pour les Python users

In [55]:

```
#import io
#import base64
#from IPython.display import HTML
#video = io.open('Introduction.mp4', 'r+b').read()
#encoded = base64.b64encode(video)
#HTML(data='''<video alt="test" controls>
#    <source src="data:video/mp4;base64,{0}" type="video/mp4" />
#    </video>'''.format(encoded.decode('ascii')))
```

Introduction à la programmation sur Python:

Différences clés par rapport à la programmation sur R:

- Indentation: Python est sensible à l'indentation
- Le point (.) sert à appeler une propriété d'un objet (~ fonction associée à un objet)

In [19]:

```
def mafonction(nom):  
    print ("Ma premiere fonction", nom )  
  
mafonction(nom="Hello")  
  
( 'Ma premiere fonction', 'Hello' )
```

Où se trouve le répertoire courant ?

In [21]:

```
import os  
os.getcwd()
```

Out[21]:

```
'/home/herimanitra/Python_BioinfoCourse'
```

In [5]:

```
#Comment changer le répertoire courant:  
os.chdir("/home/herimanitra/Téléchargements")  
os.getcwd()
```

Out[5]:

```
'/home/herimanitra/T\xca9l\xca9chargements'
```

Concaténation de caractères:

In [31]:

```
"Hello" + "World"
```

Out[31]:

```
'HelloWorld'
```

In [1]:

```
1 + 10
```

Out[1]:

```
11
```

In [2]:

```
print(type(1),type("Hello"))  
  
(<type 'int'>, <type 'str'>)
```

D'une chaine de caractères à un tableau...

In [26]:

```
maphrase= "Ceci est un cours de Bioinformatique de base"  
maphrase_tab= maphrase.split()  
maphrase_tab
```

Out[26]:

```
['Ceci', 'est', 'un', 'cours', 'de', 'Bioinformatique', 'de',  
'base']
```

D'une chaine de caractères à un tableau... spécifier l'argument split

In [27]:

```
mon_adn = "A,C,C,G,T,G,G"  
adnSplitted = mon_adn.split(",")  
adnSplitted
```

Out[27]:

```
['A', 'C', 'C', 'G', 'T', 'G', 'G']
```

Et si il n'y avait pas d'espace ?

In [4]:

```
mon_adn = "ACCGTGG"  
mon_adn=list(mon_adn)
```

Compter l'occurrence d'un mot d'une chaine de caractères:

In [1]:

```
mon_adn = "A,C,C,G,T,G,G"  
mon_adn.count("C")
```

Out[1]:

2

[Pour en savoir plus sur la manipulation de chaines de caractères](http://python.developpez.com/cours/apprendre-python3/?page=page_12)

[\(\[http://python.developpez.com/cours/apprendre-python3/?page=page_12\]\(http://python.developpez.com/cours/apprendre-python3/?page=page_12\)\)](http://python.developpez.com/cours/apprendre-python3/?page=page_12)

Introduction à numpy:

Tout d'abord, il va falloir importer la librairie (comme dans R mais avec la commande `import`)

In [2]:

```
import numpy as np
```

Manipulation basique de vecteurs avec numpy:

- Création d'une séquence de nombre:

In [3]:

```
x=np.arange(10)
print(x)
```

```
[0 1 2 3 4 5 6 7 8 9]
```

In [4]:

```
# On peut connaitre le type d'un objet avec "type"
print( type(3), type(3.0), type(x) )
```

```
(<type 'int'>, <type 'float'>, <type 'numpy.ndarray'>)
```

- Création d'une séquence dans R:

In [5]:

```
x=range(0,11) # [0-11[
x
```

Out[5]:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

- Printer le carré des éléments de la séquence:

In [6]:

```
for j in x:  
    print x[j]**2
```

```
0  
1  
4  
9  
16  
25  
36  
49  
64  
81  
100
```

- **For loop avec plusieurs arguments:**

In [7]:

```
montableau1= ['Herimanitra','eric','jean'];  
notes = [15,14,18]  
for name, note in zip(montableau1,notes):  
    print('la note de:'+ name + ' est '+ str(note))
```

```
la note de:Herimanitra est 15  
la note de:eric est 14  
la note de:jean est 18
```

- **For loop avec 02 arguments dont l'un comme index:**

In [18]:

```
montableau1= ['Herimanitra','eric','jean'];  
  
for indice, name in enumerate(montableau1):  
    print ("Name:" + name + " d indice: " + str(indice) )
```

```
Name:Herimanitra d indice: 0  
Name:eric d indice: 1  
Name:jean d indice: 2
```

- Compter le nombre d'occurrence d'un élément d'un vecteur

In [8]:

```
x.count(2)
```

Out[8]:

```
1
```

- Append un élément dans le vecteur

In [9]:

```
x.append(10)
x.count(10)
```

Out[9]:

2

- Slicing de vecteurs avec numpy

Autres structures de controles utiles :

While...

In [10]:

```
j=1
while j<4:
    print j
    j += 1
```

1
2
3

Sélectionne l'élément 4 à 7 du vecteur `x= [0 1 2 3 4 5 6 7 8 9]`

On voit ici que la notation n'est pas très conviviale pour quelqu'un qui à l'habitude d'utiliser R

In [11]:

```
x[3:7]
```

Out[11]:

[3, 4, 5, 6]

sélectionne les 03 premiers éléments du vecteur:

In [12]:

```
x[:3]
```

Out[12]:

```
[0, 1, 2]
```

sélectionne tous les éléments à partir du 5ième élément:

In [13]:

```
x[4:]
```

Out[13]:

```
[4, 5, 6, 7, 8, 9, 10, 10]
```

sélectionne tous les éléments pairs d'un vecteur

In [2]:

```
import numpy as np
x= np.arange(0,101)
x[:101:2]
```

Out[2]:

```
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18, 20,
        22, 24,
        26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46,
        48, 50,
        52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72,
        74, 76,
        78, 80, 82, 84, 86, 88, 90, 92, 94, 96, 98, 1
        00])
```

inversion des éléments du vecteur:

In [15]:

```
x[::-1]
```

Out[15]:

```
array([[100, 99, 98, 97, 96, 95, 94, 93, 92, 91, 90,
      89, 88,
      87, 86, 85, 84, 83, 82, 81, 80, 79, 78, 77,
      76, 75,
      74, 73, 72, 71, 70, 69, 68, 67, 66, 65, 64,
      63, 62,
      61, 60, 59, 58, 57, 56, 55, 54, 53, 52, 51,
      50, 49,
      48, 47, 46, 45, 44, 43, 42, 41, 40, 39, 38,
      37, 36,
      35, 34, 33, 32, 31, 30, 29, 28, 27, 26, 25,
      24, 23,
      22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12,
      11, 10,
      9, 8, 7, 6, 5, 4, 3, 2, 1, 0])
```

Tri d'éléments dans un tableau:

In [16]:

```
x= [0,1,2,1,5,4,12,25,13,18]
print(sorted(x))
```

```
[0, 1, 1, 2, 4, 5, 12, 13, 18, 25]
```

Manipulation de matrices avec numpy:

- Création d'une matrice carré de taille 2x2 et vérification de sa dimension avec shape:

In [17]:

```
mymatrix=np.zeros ((2,2))
mymatrix.shape
```

Out[17]:

```
(2, 2)
```

In [18]:

```
mymatrix
```

Out[18]:

```
array([[ 0.,  0.],
       [ 0.,  0.]])
```

In [19]:

```
#Voici comment numpy représente la matrice visuellement:
print ( mymatrix )
```

```
[[ 0.  0.]
 [ 0.  0.]]
```

- Création de matrice avec tous les éléments égaux à 1

In [20]:

```
mymatrix1= np.ones((10,10))
print (mymatrix1)
```

```
[[ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]]
```

In [4]:

```
#Génère une séquence de 24 éléments et transforme cette séquence en une matrice
b= np.arange(24).reshape(6,4)
b
```

Out[4]:

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15],
       [16, 17, 18, 19],
       [20, 21, 22, 23]])
```

Slicing de matrices:

- **Accéder à la première ligne:**

In [5]:

```
b[0,:] #vous pouvez maintenant monter en ligne 1,2,...
```

Out[5]:

```
array([0, 1, 2, 3])
```

- **Accéder à la première colonne:**

In [6]:

```
b[:,0]
```

Out[6]:

```
array([ 0,  4,  8, 12, 16, 20])
```

- **Donne la transposé de la matrice:**

In [7]:

```
btransp= b.transpose()  
btransp
```

Out[7]:

```
array([[ 0,  4,  8, 12, 16, 20],  
       [ 1,  5,  9, 13, 17, 21],  
       [ 2,  6, 10, 14, 18, 22],  
       [ 3,  7, 11, 15, 19, 23]])
```

In [8]:

```
#On pouvait également utiliser la syntaxe suivante:  
btransp = b.T  
btransp
```

Out[8]:

```
array([[ 0,  4,  8, 12, 16, 20],  
       [ 1,  5,  9, 13, 17, 21],  
       [ 2,  6, 10, 14, 18, 22],  
       [ 3,  7, 11, 15, 19, 23]])
```

On peut faire combiner des matrices avec numpy comme dans R (cbind,rbind)...

In [9]:

```
#Prenons une partie de b (une matrice carré par exemple)  
X=np.resize(b,(4,4))  
X
```

Out[9]:

```
array([[ 0,  1,  2,  3],  
       [ 4,  5,  6,  7],  
       [ 8,  9, 10, 11],  
       [12, 13, 14, 15]])
```

In [10]:

```
# L'équivalent du cbind en python avec hstack
Z1=np.hstack((X,btransp))
```

In [11]:

```
# L'équivalent du rbind en python avec vstack
Z2 = np.vstack((X,X))
Z2
```

Out[11]:

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15],
       [ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

In [12]:

```
#On peut également sommer les diagonales en faisant:
np.trace(X)
```

Out[12]:

30

Comme dans R, le signe multiplication n'est pas synonyme de multiplication MATRICIELLE mais par contre de multiplication élément par élément:

In [13]:

```
print (X, X*X)
```

```
(array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]]), array([[ 0,  1,  4,  9],
       [16, 25, 36, 49],
       [64, 81, 100, 121],
       [144, 169, 196, 225]]))
```

Pour réaliser la multiplication matricielle, On utilise la fonction dot :

In [29]:

```
np.dot(X,X)
```

Out[29]:

```
array([[ 56,  62,  68,  74],
       [152, 174, 196, 218],
       [248, 286, 324, 362],
       [344, 398, 452, 506]])
```

In [30]:

```
y=np.array((10,-1,4,-5)) #vecteur ligne 4x1
np.dot(X,y) #matrice 4x4 multipliée par le vecteur ligne 4x1
```

Out[30]:

```
array([-8, 24, 56, 88])
```

Exemples simples avec hstack & vstack

In [3]:

```
import numpy as np
x1= np.array([1,10,1])
x2= [14,5,10]
prinnp.vstack ( (x1,x2) )
```

Out[3]:

```
array([[ 1, 10,  1],
       [14,  5, 10]])
```

In [4]:

```
np.hstack ( (x1,x2) )
```

Out[4]:

```
array([ 1, 10,  1, 14,  5, 10])
```

Introduction à l'Analyse factorielle:

- Les statistiques descriptives (moyenne, médiane, variance, etc.) s'avère être limitée lorsque l'on cherche à décrire ou à résumer un volume de données important.
- Il est désirable d'avoir une méthode qui permet de résumer les principaux facteurs qui régissent un jeu de données. C'est l'analyse factorielle
- En génomique, on peut avoir des microarray de gènes d'individus de sexe masculin et féminin exposés à différentes conditions (environnementaux, résultats cliniques, prélèvements dans le temps, etc.). On souhaiterait alors connaître les conditions qui conduisent **les expressions de gènes à être différentes (source d'hétérogénéité)**.

- L'analyse factorielle permet à un individu (représenté par une ligne) **et/ou** à une variable (modalité) d'avoir des coordonnées sur le plan (x,y) ==> plan factoriel
- Cependant, il ne suffit pas juste de représenter les individus **et/ou** les variables sur le plan factoriel. Il faudrait que ces individus **et/ou** variables soient les plus dispersées possible de telle sorte à faciliter les interprétations après.
- En termes mathématiques, cela se traduit par **la maximisation de la variance du nuage projeté**:

$$\max((M^t v)^t M^t v) = \max(v^t M M^t v) \\ \text{s/c : } v^t v = 1$$

où M est la matrice des données de départ

- On sait résoudre cela en ayant recours à une méthode issue de l'Algèbre linéaire appelée **décomposition en valeur singulière ou SVD**. Cette méthode dit que toute matrice M dans R^{nk} peut être décomposée en un produit de 03 matrices:

$$M = U S V^t$$

Où

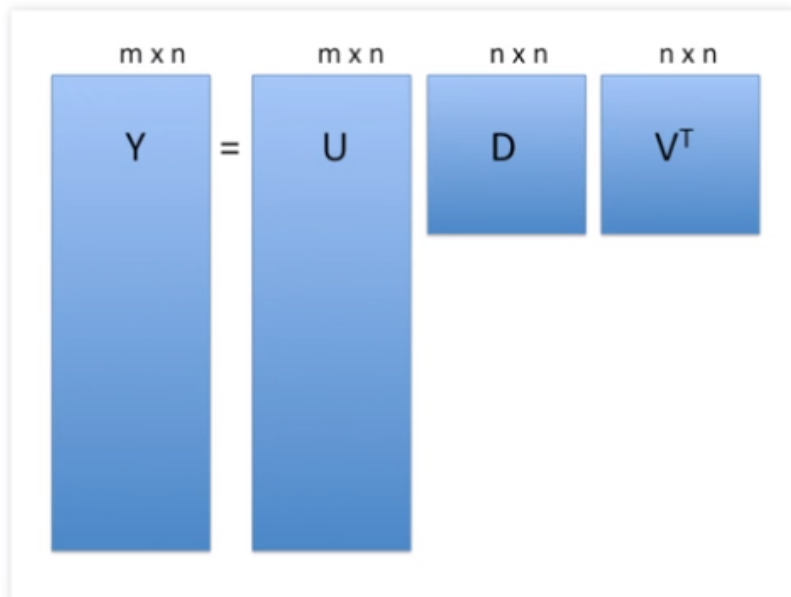
- **M de dimension [n k]**
- **U de dimension [n n] avec certaines propriétés algébriques**
- **S de dimension [n k] matrice DIAGONALE**
- **V^t de dimension [k k] avec certaines propriétés algébriques**
- Sans entrer dans les détails penser juste que comme pour les scalaires (30=2x3x5) , on peut factoriser une matrice avec cette formule

Décomposition d'une matrice (visuellement):

In [31]:

```
from IPython.display import Image  
Image(filename='SVD.png')
```

Out[31]:

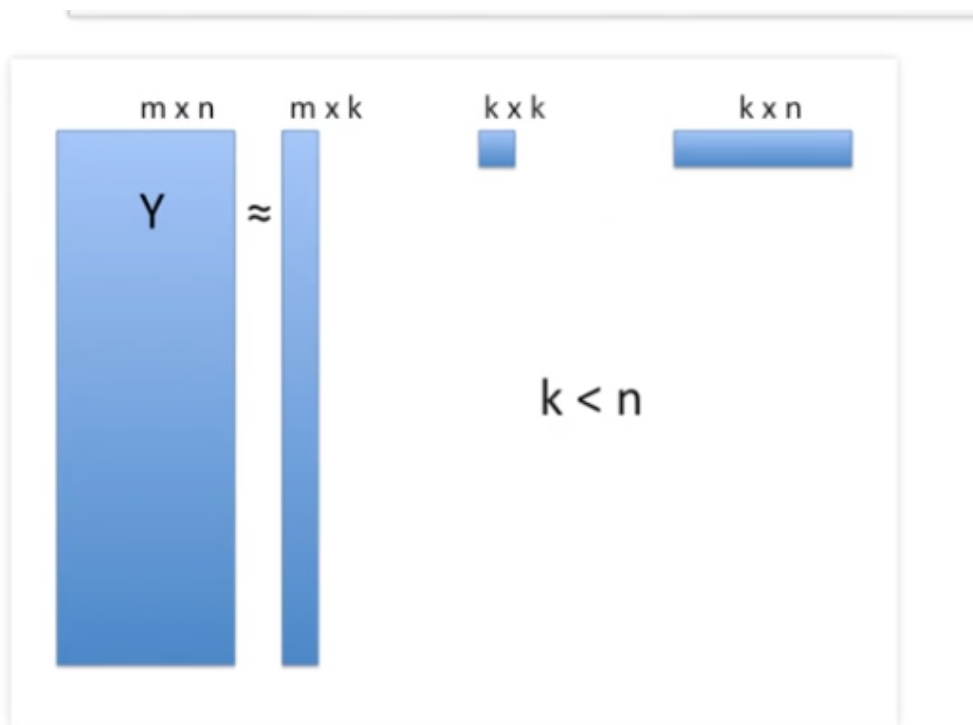


**Moins de variables sans perte d'information
(visuellement)...**

In [32]:

```
from IPython.display import Image
Image(filename='dimredu.png')
```

Out[32]:



Obtenir ces matrices grâce à numpy :

In [33]:

```
from numpy import linalg
```

In [34]:

```
#Construisons une matrice 5x5:
M= np.matrix ([ [0,0,-1,4,4],[2,3,4,5,6],[7,-2,3,8,0],[0,-1,2,9,-13],[-6,-4,-11,11,5]])
M
```

Out[34]:

```
matrix([[ 0.,  0., -1.,  4.,  4.],
        [ 2.,  3.,  4.,  5.,  6.],
        [ 7., -2.,  3.,  8.,  0.],
        [ 0., -1.,  2.,  9., -13.],
        [-6., -4., -11., 11.,  5.]])
```


In [35]:

```
#DECOMPOSITION en valeur singulière:  
U,S,Vt= linalg.svd(M)  
print("La matrice U est:",U)
```

```
('La matrice U est:', matrix([[ 0.23252768, -0.12722154,  0.235  
91025,  0.11670808,  0.92761583],  
[ 0.12021032, -0.07227163,  0.67046145,  0.66643387,  
-0.29440386],  
[ 0.24145219,  0.34503452,  0.61406369, -0.66194132,  
-0.08609007],  
[ 0.27076765,  0.87217399, -0.24101322,  0.32034326,  
0.07273397],  
[ 0.89435232, -0.31441246, -0.2442667 , -0.03819692,  
-0.20038339]]))
```

In [36]:

```
print("La matrice diagonale est:",S)
```

```
('La matrice diagonale est:', array([ 18.74294948,  16.9063934  
,  12.76319501,  4.79332075,  0.02522459]))
```

In [37]:

```
print("La matrice V transpose est:",Vt)
```

```
('La matrice V transpose est:', matrix([[ -0.18329708, -0.211837  
01, -0.44409606,  0.83965249,  0.13888715],  
[ 0.24589356, -0.03084088,  0.35939798,  0.37151985,  
-0.8193847 ],  
[ 0.55667635,  0.15680533,  0.50873107,  0.3410122 ,  
0.53891271],  
[ -0.64079584,  0.65833872,  0.33881648,  0.20161273,  
0.02294621],  
[ 0.43061662,  0.70439771, -0.54772974,  0.00443709,  
-0.13551985]]))
```

Reconstruction de la matrice M d'origine:

In [38]:

```
#Pour économiser de la mémoire numpy stocke la matrice diagonale sous forme de
#Nous allons le transformer sous forme de matrice
Z= np.zeros((5,5))
Z=np.diag(S)
M==np.round(np.dot( np.dot(U,Z),Vt))
```

Out[38]:

```
matrix([[ True,  True,  True,  True,  True],
        [ True,  True,  True,  True,  True],
        [ True,  True,  True,  True,  True],
        [ True,  True,  True,  True,  True],
        [ True,  True,  True,  True,  True]], dtype=bool)
```

On peut ne pas utiliser toutes les colonnes de V et obtenir quand même une approximation de la matrice M de départ:

In [39]:

```
#Sélection des 02 1eres valeurs propres:
Z2=Z[:2,:2]
Vt2= Vt[:2,]
U2=U[:, :2]
np.round(np.dot( np.dot(U2,Z2),Vt2 ),1)
```

Out[39]:

```
array([[ -1.3,  -0.9,  -2.7,   2.9,   2.4],
       [ -0.7,  -0.4,  -1.4,   1.4,   1.3],
       [  0.6,  -1.1,   0.1,   6. ,  -4.2],
       [  2.7,  -1.5,   3. ,   9.7, -11.4],
       [ -4.4,  -3.4,  -9.4,  12.1,   6.7]])
```

Analyse en Composante Principale (ACP):

L'ACP utilise l'analyse factorielle afin de représenter des données multidimensionnelles sur le plan factoriel. L'ACP tente de répondre aux questions suivantes:

- Qui sont les individus ou groupes d'individus qui se ressemblent
- Quelles sont les variables corrélées (+/-)==>LIAISON ENTRE LES VARIABLES
- La résolution de ce problème d'optimisation revient à rechercher la **valeur propre maximale associée à chaque dimension**
- M peut être décomposé en valeur singulière mais aussi MM^t , ce qui donne:

$$MM^t = US^2U^t$$
- Conclusion: on peut piocher la solution du problème d'optimisation de l'ACP dans la décomposition singulière de MM^t

Lire un fichier csv avec python:

In [40]:

```
with open("iris.csv") as f:
    mydata= [line.rstrip().split(';') for line in f] #inline for

#accéder à la première colonne
mydata[0][0]
#for j in mydata[0]:
#    print j
```

Out[40]:

```
'Sepal.Length''
```

Lire un fichier csv avec pandas:

In [3]:

```
#load le data.frame(iris)
import pandas as pds
iris = pds.read_csv("/home/herimanitra/Python_BioinfoCourse/iris.csv", sep=";")
iris.head()
```

Out[3]:

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

- Accéder aux 10 premières lignes de la variable Sepal.Length pour l'espace setosa du data.frame:

In [3]:

```
iris[iris.Species=="setosa"]['Sepal.Length'][:10]
```

Out[3]:

```
0    5.1
1    4.9
2    4.7
3    4.6
4    5.0
5    5.4
6    4.6
7    5.0
8    4.4
9    4.9
```

Name: Sepal.Length, dtype: float64

- **Groupement par espèces et effectifs:**

In [43]:

```
iris.groupby('Species').size()
```

Out[43]:

```
Species
setosa      50
versicolor  50
virginica   50
dtype: int64
```

- **Changement de format avec pandas astype**

In [44]:

```
#iris['Petal.Width']= iris['Petal.Width'].astype('category')
#iris.groupby('Petal.Width').size()
iris['Petal.Width']= iris['Petal.Width'].astype('float')
```

In [45]:

```
#Liste le nombre unique d'espèces:
iris['Species'].unique()
```

Out[45]:

```
array(['setosa', 'versicolor', 'virginica'], dtype=object)
```

- **Visualisation 3D des espèces**

In [4]:

```

import plotly.plotly as py
from plotly.graph_objs import *
py.sign_in("Herimanitra","xkgkjpnn20")

setosa = Scatter3d(
    x=iris[iris.Species=="setosa"]['Sepal.Length'],
    y=iris[iris.Species=="setosa"]['Sepal.Width'],
    z=iris[iris.Species=="setosa"]['Petal.Length'],
    mode='markers',
    marker=dict(
        color='rgb(0, 204, 0)',
        size=2,
        symbol='circle',
        opacity=0.9
    )
)

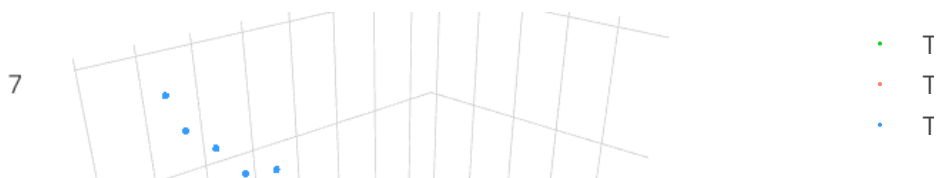
versicolor = Scatter3d(
    x=iris[iris.Species=="versicolor"]['Sepal.Length'],
    y=iris[iris.Species=="versicolor"]['Sepal.Width'],
    z=iris[iris.Species=="versicolor"]['Petal.Length'],
    mode='markers',
    marker=dict(
        color='rgb(255, 99, 71)',
        size=2,
        opacity=0.8
    )
)

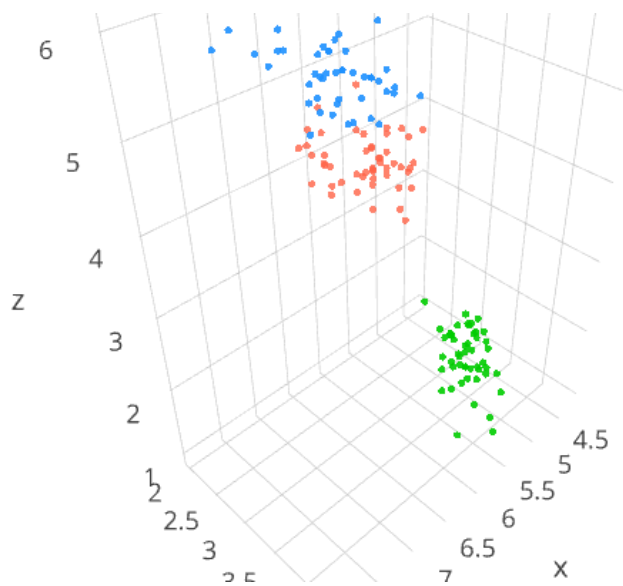
virginica= Scatter3d(
    x=iris[iris.Species=="virginica"]['Sepal.Length'],
    y=iris[iris.Species=="virginica"]['Sepal.Width'],
    z=iris[iris.Species=="virginica"]['Petal.Length'],
    mode='markers',
    marker=dict(
        color='rgb(30, 144, 255)',
        size=2,
        symbol='circle',
        opacity=0.9
    )
)

data = Data([setosa,versicolor,virginica])
py.iplot(data)

```

Out[4]:





Ec

Interprétations pratiques des résultats de l'ACP:

- La décomposition SVD du jeu de données nous donne les valeurs **valeurs propres** classées par ordre décroissant
- La décomposition SVD du jeu de données nous donne également les plans de projection associées à ces valeurs propres
- Dans l'interprétation des résultats on se limite généralement aux n-premiers axes factoriels qui couvrent **70%** de l'information.
- Dans la pratique si vous avez eu du mal à interpréter le premier plan, c'est que vous devriez vous arrêter là car le plan suivant sera plus difficile.

02 principaux Output:

- Le cercle de corrélation servant à visualiser les liaisons entre variables
- Le nuage des individus qui serviront à dégager leur ressemblance

In [47]:

```
%matplotlib inline
#load libraries:
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

#Le premier plan factoriel:
pca = PCA(n_components=2)

#Sélectionne le tableau numérique
X= iris[['Sepal.Length','Sepal.Width','Petal.Length','Petal.Width']]

#Sélectionne les labels des espèces
target_names=np.array(iris[['Species']],dtype="str")

#construit un target libellé numérique:
iris['target'] = 1
iris['target'][iris['Species']=='setosa']=0
iris['target'][iris['Species']=='versicolor']=1
iris['target'][iris['Species']=='virginica']=2
y = np.array(iris['target'])

#Données Centrée réduite
X_r = pca.fit(X).transform(X)
plt.figure()

for c, i, target_name in zip("rgb", [0, 1, 2], target_names):
    plt.scatter(X_r[y == i, 0], X_r[y == i, 1], c=c, label=target_name)
#plt.legend()
plt.title('PCA of IRIS dataset')
plt.show()
```

/usr/local/lib/python2.7/dist-packages/ipykernel/__main__.py:1

9: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: <http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy> (http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy)

/usr/local/lib/python2.7/dist-packages/ipykernel/__main__.py:2

0: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame

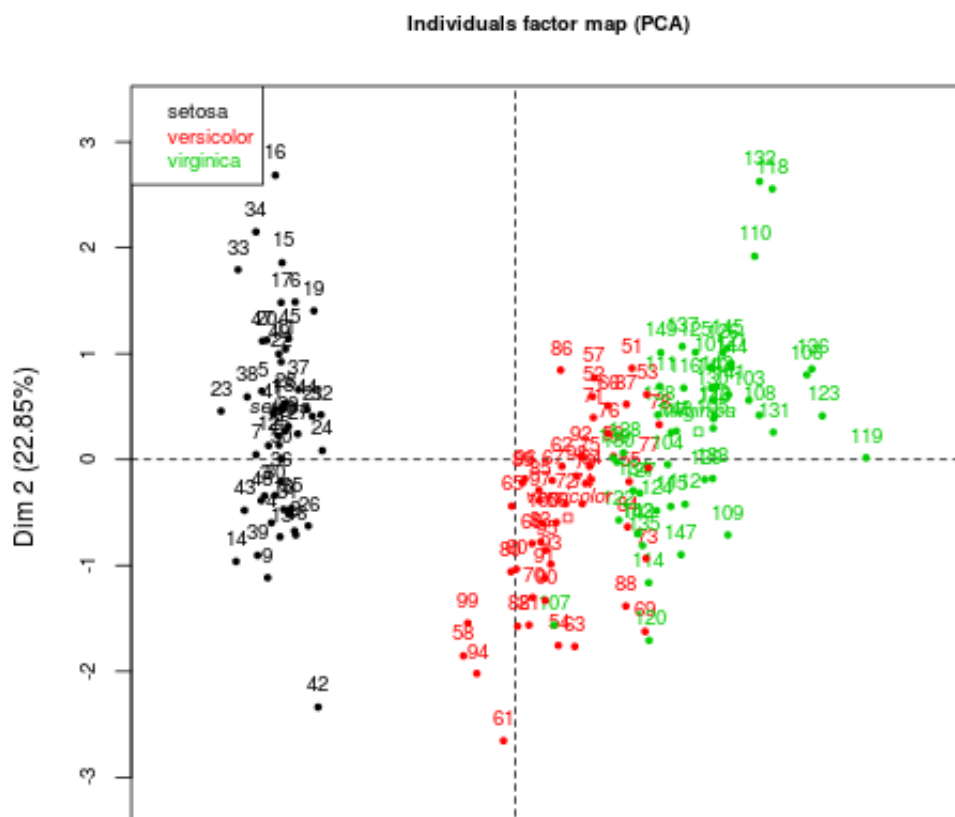
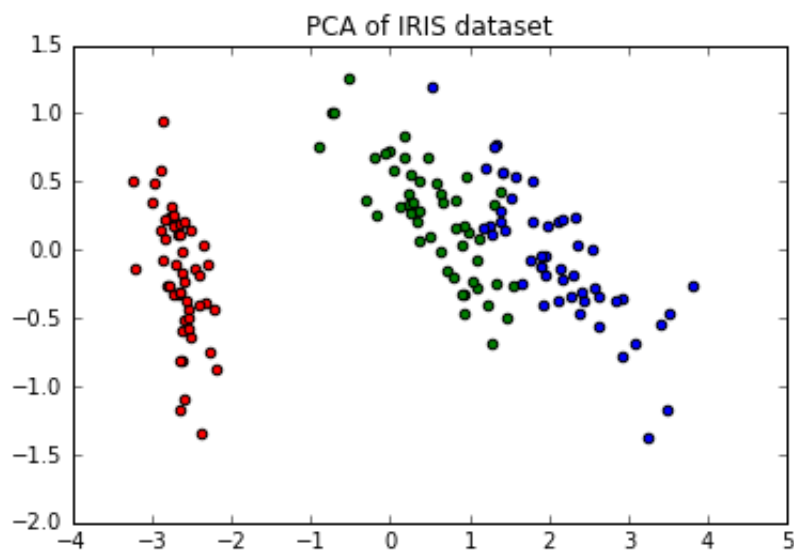
See the caveats in the documentation: <http://pandas.pydata.org>

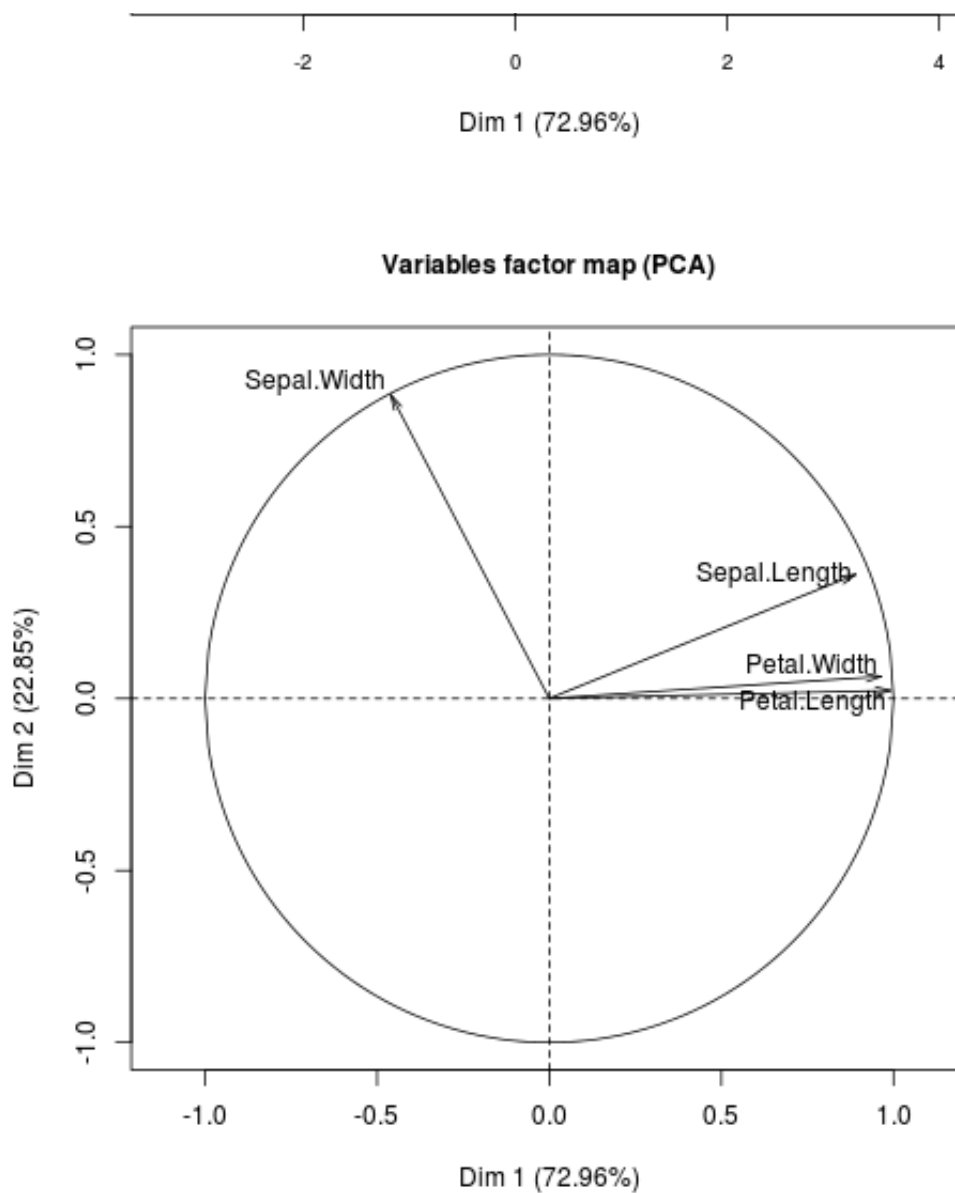
g/pandas-docs/stable/indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy)

/usr/local/lib/python2.7/dist-packages/ipykernel/__main__.py:21: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy)





Application à l'expression de gènes:

Expression de gènes = matrice dont les gènes sont en ligne avec leur différentes expressions à différents checkpoints.

Le but est de classer les gènes en différents groupes de **même comportement**.

Le but est, par exemple, de savoir s'il existe des différences entre les gènes de cellules cancéreuses avec celles de cellules saines (Uri Alon, 1999, a comparé 2000 expressions de gènes de patients atteint d'une tumeur du colon avec des gènes provenant de sujets sains.) ==> POTENTIAL CANCER BIOMARKERS

Python et la Bioinformatique:

Quelques algorithmes d'alignement de séquences

Les activités:

Les activités.

- Comparaison de séquences
- recherche de séquences similaires
- récupération des fonctions associées aux gènes

Idée: on ne va pas prédire la fonction d'un gène, on va interroger une base de donnée existante. En effet, les laboratoires/instituts qui font du séquençage peuvent déposer leur résultat dans une Banque d'ADN (GeneBank, UniProt:Universal Protein Resource,etc.)

Principe des banques...: à une séquence de gène est associée des fonctions (texte, mots-clés~GO,classification enzymatique)

Structure de données dans python ==>Dictionnaire:

Typiquement les résultats issus des banques données sont de la forme:

In [1]:

```
x={'sequence': 'ACCGTTACG', 'organisme': 'E.Coli', 'ec': '3.4.11.4'}
```

In [2]:

```
x['sequence']
```

Out[2]:

```
'ACCGTTACG'
```

Occurence d'un motif:

In [4]:

```
x['sequence'].count('AC')
```

Out[4]:

```
2
```

In [50]:

```
x['organisme']
```

Out[50]:

```
'E.Coli'
```

In [51]:

```
x['ec']
```

Out[51]:

```
'3.4.11.4'
```

In [13]:

```
#tableau à l'intérieur de séquences:  
x={'sequence': ['ACCGTTACG', 'ACTTTTGCC', 'TGGTATGCC'],  
  'organisme': ['E.Coli', 'B.subtilis', 'H.influenzae'],  
  'ec': ['3.4.11.4', '2.3.4', '4.1.1.3']}
```

In [14]:

```
#accéder à des éléments individuels  
for i in x['sequence']:  
    print (i)
```

```
ACCGTTACG  
ACTTTTGCC  
TGGTATGCC
```

In [54]:

```
#les clés ~ nom de variables  
x.keys()
```

Out[54]:

```
['organisme', 'ec', 'sequence']
```

Pourquoi des séquences de génomes sont similaires entre organisme?

- Evolution des espèces va de paire à l'évolution de leur génome==> Les séquences diffèrent au cours du temps (à cause des mutations, etc.) tout en restant **similaire**
- En effet, il peut y avoir **substitutions, délétions, insertions**

Manipulation de séquences avec Python

Le plus long préfixe d'un ADN:

In [55]:

```
def longestpref(s1,s2):  
    i=0 #indicateur pour monitorer la longueur de la séquence  
    while i< len(s1) and i<len(s2) and s1[i]==s2[i]:  
        i = i+1 #incrémente tant qu'il y a un match  
    return s1[:i]  
  
longestpref ('ACCATGT','ACCAGAC')
```

Out[55]:

'ACCA'

In [56]:

```
#Comparaison de deux séquences  
'ACCA' == 'ACCA'
```

Out[56]:

True

Inverse complémentaire d'une séquence d'ADN:

In [57]:

```
complement = {'A':'T','C':'G','G':'C','T':'A'}
```

In [58]:

```
complement['A']
```

Out[58]:

'T'

In [59]:

```
def inverse_sequence_complementaire(s):  
    complement = {'A':'T','C':'G','G':'C','T':'A'}  
    t='' #initialiser avec un caractère vide  
    for base in s:  
        t= complement[base] + t  
    return t  
inverse_sequence_complementaire('ATGC')
```

Out[59]:

'GCAT'

• Hamming Distance:

Etant donné 02 séquences d'ADN, la distance de Hamming est définie comme le nombre minimum de substitution nécessaire pour que les 02 soient égales.

- **Visuellement, cela donne:**

In [60]:

```
from IPython.display import Image
Image(filename='sequences.png')
```

Out[60]:



X: GAGGTAGCGGCGTTTAAC
Y: GTGGTAACGGGGTTTAAC

In [61]:

```
from IPython.display import Image
Image(filename='sequences_edited.png')
```

Out[61]:



X: GAGGTAGCGGCGTTTAAC
Y: GTGGTAACGGGGTTTAAC
Hamming distance = 3

In [8]:

```
#import matplotlib.pyplot as plt
from scipy.spatial.distance import hamming
from skbio import DNA
adn1 = DNA("GAGGTAGTTAACGGTGACCAGGTACCAGAAGGGTACCAGGTAGGACACACGGGGATTAAGTAGG")
adn2 = DNA("GAGGAGGTTAACGGTGACCAGGTACCAGAAGGGTACCAGGTAGGAGACACGGCGATTAAAAAGT")
print(hamming(adn1, adn2))
```

0.109375

Matrice d'incidence & visualisation des alignements

In [9]:

```

#import skbio
import numpy as np
seq1="GAGGTAGTTAACGGTGACCAGG"
seq2="GAGGAGGTAAACCCTGACCAGG"
seq1_list= list(seq1); seq2_list= list(seq2)
mymatrix = np.zeros(shape=(len(seq1),len(seq2)),dtype="int")
p=len(seq1)
for k1,ch1 in enumerate(seq1_list):
    for k2, ch2 in enumerate(seq2_list):
        if ch1==ch2:
            mymatrix[k1,k2]=1
            #mymatrix[k1,k2]=1*p
    p=p-1
print(mymatrix)

```

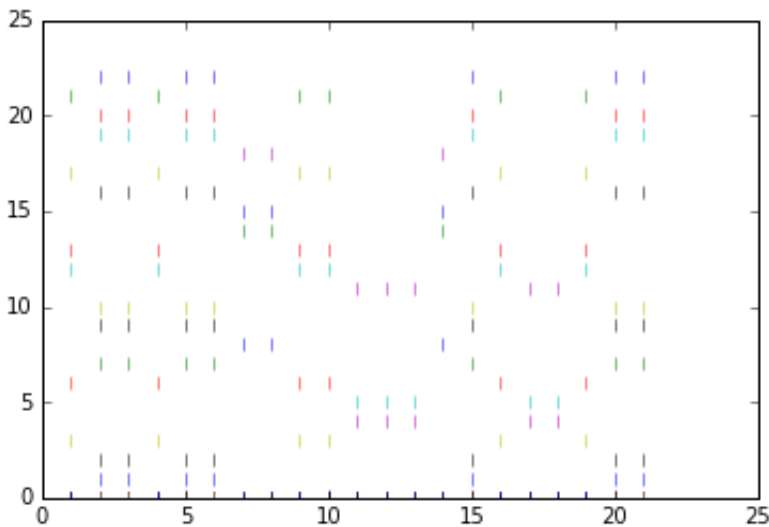
```

[[1 0 1 1 0 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1]
 [0 1 0 0 1 0 0 0 0 1 1 0 0 0 0 0 1 0 0 1 0 0]
 [1 0 1 1 0 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1]
 [1 0 1 1 0 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1]
 [0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0]
 [0 1 0 0 1 0 0 0 0 1 1 0 0 0 0 0 1 0 0 1 0 0]
 [1 0 1 1 0 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1]
 [0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0]
 [0 1 0 0 1 0 0 0 0 1 1 0 0 0 0 0 1 0 0 1 0 0]
 [0 1 0 0 1 0 0 0 0 1 1 0 0 0 0 0 1 0 0 1 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 1 1 0 0]
 [1 0 1 1 0 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1]
 [1 0 1 1 0 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1]
 [0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0]
 [1 0 1 1 0 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1]
 [0 1 0 0 1 0 0 0 0 1 1 0 0 0 0 0 1 0 0 1 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 1 1 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 1 1 0 0]
 [0 1 0 0 1 0 0 0 0 1 1 0 0 0 0 0 1 0 0 1 0 0]
 [1 0 1 1 0 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1]
 [1 0 1 1 0 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1]]

```

In [22]:

```
%matplotlib inline
import matplotlib.pyplot as plt
#for k in range(len(seq1)):
#plt.scatter(x=range(len(seq1)),y=[ row for row in mymatrix] )
mymatrix[0,:] # accéder row wise
for k in range(len(seq1)):
    plt.plot(range(len(seq1)),mymatrix[k,:],'|' )
```

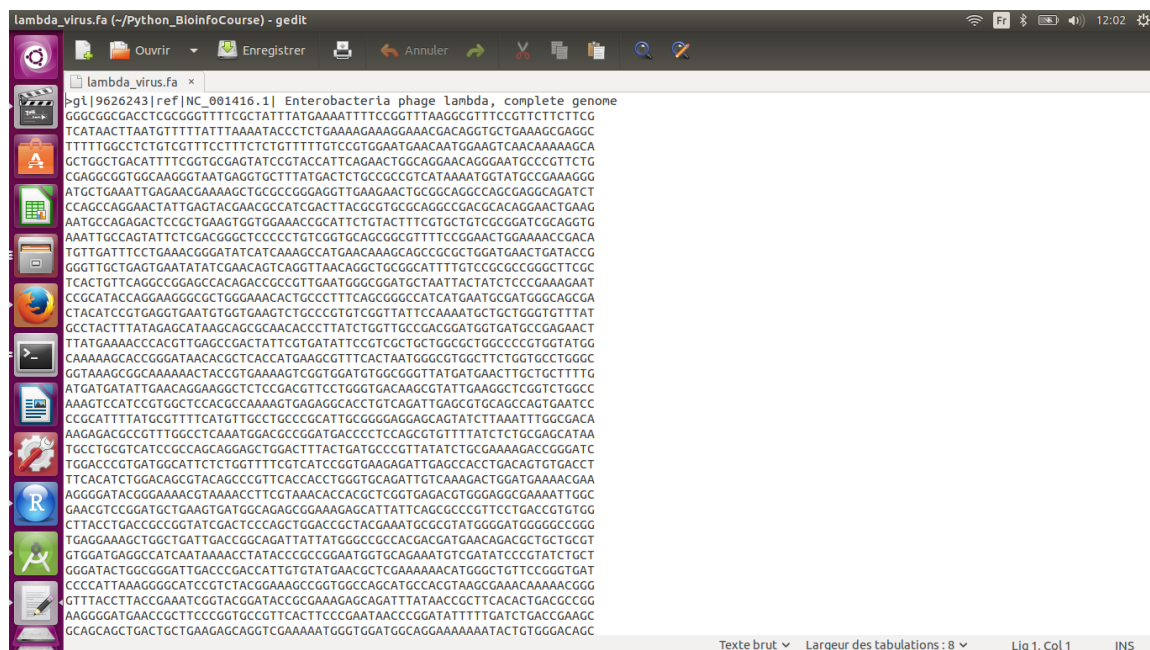


Lecture de fichier fasta avec python:

In [67]:

```
from IPython.display import Image
Image (filename="fichier_fasta.png")
```

Out[67]:



In [36]:

```
def read_genome (filename):
    genome='' #initialisation d'une chaine de caractère
    with open(filename,'r') as f: #connection au fichier
        for line in f:
            if line[0]!='>': #si le 1er caractère d'une ligne donnée ne contient pas
                genome= genome + line.rstrip()
    return genome
X=read_genome("lambda_virus.fa")
```

Compter le nombre d'apparition d'une séquence particulière dans la séquence complète:

In [2]:

```
X.count('ATGCC')
```

Out[2]:

78

Chercher un séquence particulière dans ce génomène complet:

In [41]:

```
disease = 'ATGCC' #je défini mon disease
counter=0 #j initialise un compteur
mylength=len(X) # je récupère la longueur de mon génome complet
mespositions =[] #j initialise un tableau vide pour stocker mes positions
while counter< mylength: #tant que je ne suis pas au dernier caractère de ma séquence
    if X[counter:(counter + len(disease))].find(disease) > -1: # je scanne le
        mespositions.append(counter + X[counter:].find(disease)) #j'ajoute la position
        counter = counter + 1
print (mespositions,len(mespositions))
```

```
([268, 338, 491, 978, 1038, 1539, 1578, 2213, 2896, 3004, 3040,
3477, 3667, 3834, 5243, 5702, 7762, 8954, 8975, 9212, 9929, 100
59, 10211, 10950, 12003, 12873, 13211, 14048, 14536, 15586, 165
48, 16635, 16693, 16804, 16960, 17254, 18064, 18082, 18759, 199
38, 20046, 20487, 21131, 21617, 21857, 22569, 23373, 24372, 269
09, 27864, 27883, 29522, 29865, 30739, 32526, 32592, 32707, 327
57, 33153, 34064, 35088, 35897, 36345, 37776, 39419, 39647, 413
41, 42469, 42768, 43399, 44347, 44454, 45191, 45715, 46083, 464
98, 46995, 48046], 78)
```


In [38]:

```
X[0:(0 + len(disease))]
```

Out[38]:

```
'GGGCG'
```

In [29]:

```
#Quelques stats sur cette séquence:  
import collections  
collections.Counter(X)
```

Out[29]:

```
Counter({'A': 12334, 'C': 11362, 'G': 12820, 'T': 11986})
```

Lecture de fichier fastq avec python:

In [70]:

```
from IPython.display import Image  
Image(filename='reads_fastq.png')
```

Out[70]:

A read in FASTQ format

```
Name @ERR194146.1 HSQ1008:141:D0CC8AC  
Sequence ACATCTGGTTCCTACTTCAGGGCCATAAAGCC  
(ignore) +  
Base qualities ?@@FFBFDDHHBCEAFGEGIIDHGH@GDHHF
```

In [71]:

```
def read_fastq(filename):
    reads= [] ; #déclare un tableau vide
    with open(filename) as f:
        eof=False
        while eof==False:
            f.readline() #je suis dans Name
            seq=f.readline().rstrip() #la sequence proprement dite
            f.readline() #+
            f.readline() #base qualities
            if len(seq)!=0: #si je ne suis pas sur ligne vide j'appende
                reads.append(seq)
            else:
                eof=True

    return reads

reads=read_fastq("ERR037900_1.first1000.fastq")
```

In [72]:

```
#Print les 05 premiers reads
for j in range(5):
    print reads[j]
```

```
TAACCCTAACCCCTAACCCCTAACCCCTAACCCCTAACCCCTAACCCCTAACCCCTAACCCCTAA
CCCNAAACCCTAACCCCTAACCCCTAACCCCTAACCCCTAAC
TAACCCTAACCCCTAACCCCTAACCCCTAACCCCTAACCCCTAACCCCTAACCCCTAACCCCTAA
CCCNAAACCCTAACCCCTAACCCCTAACCCCTAACCCCTAAC
TAACCCTAACCCCTAACCCCTAACCCCTAACCCCTAACCCCTAACCCCTAACCCCTAACCCCTAA
CCCNAAACCCTAACCCCTAACCCCTAACCCCTAACCCCTAAC
TAACCCTAACCCCTAACCCCTAACCCCTAACCCCTAACCCCTAACCCCTAACCCCTAACCCCTAA
CCCNAAACCCTAACCCCTAACCCCTAACCCCTAACCCCTAAC
AACCCTAACCCCTAACCCCTAACCCCTAACCCCTAACCCCTAACCCCTAACCCCTAACCCCTAAC
CCTNACCCTAACCCCTAACCCCTAACCCCTAACCCCTAAC
```

In [73]:

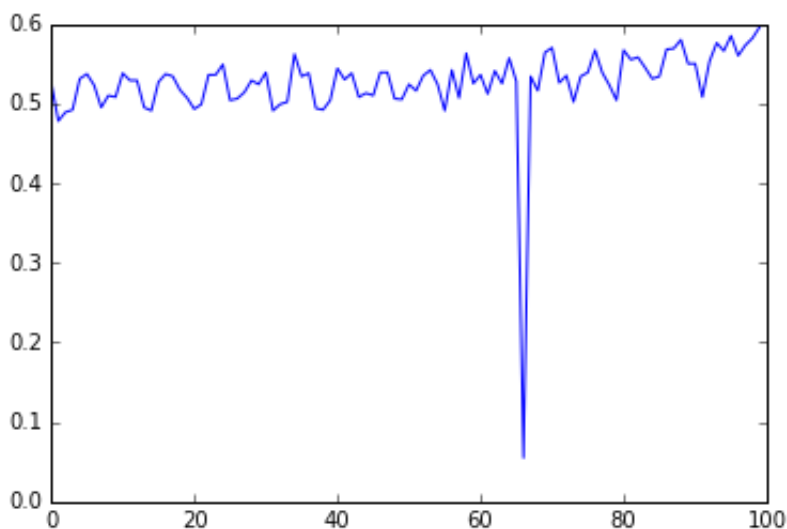
```
##Stats de base sur ces reads:
import collections
count = collections.Counter()
for read in reads:
    count.update(read)
print (count)
```

```
Counter({'C': 29665, 'A': 24057, 'G': 22888, 'T': 22476, 'N': 9
14})
```

In [74]:

```
#Trouver la proportion de 'G' ou de 'C' par reads:
def pos_GC_reads(reads):
    GC=[0.0]*100
    totals= [0.0]*100
    #A chaque fois que je trouve des C ou des G j'incrmente le tableau GC
    for read in reads:
        for i in range(len(read)):
            if read[i]=='C' or read[i]=='G':
                GC[i] += 1
                totals[i] += 1
    #End
    #A la fin du compte, si totals>0 alors fait la division:
    for i in range(len(GC)):
        if totals[i]>0.0:
            GC[i]= GC[i]/totals[i]
    return GC

gc= pos_GC_reads(reads)
import matplotlib.pyplot as plt
%matplotlib inline
plt.plot(range(len(gc)),gc)
plt.show()
```



Algorithmes d'Alignement de séquences:

```
from IPython.display import Image
Image(filename='alignement_seq.png')
```

Alignment

Reference

GATCACAGGCTCTATCACCTTATTAACCACTCACGGGAGCTCTCCATGCATTTGGTATTTT
 CGTCTGGGGGGTATGCACGCGATAGCATTGCGAGACGCTGGAGCCGGAGCACCTATGTC
 GCAGTATCTGCTTTTGATTCTGCTGCTCATCTCTATTATTTATCGCACCTACGTTCAATAT
 ACAGCGCAACTATCTACTAAAGTGTTGTAATTAATTAATGCTGTAGGACATAATAATA
 ACAATTGAATGCTGACACGCGCACTTTCCACACAGATCATAAACAAAAATTTCCACC
 AACCCCCCTCCCCGCTTCTGGCCACAGCACTTCTGCGCAACCCCAAAA
 ACAAAGAACCCTAACACAGCCTAACCACTTCTCAAACTTGGCGGTATGCAC
 TTTTAAACAGTCAACCCCCAACTAACCTTATTTTCCCCCTGCTGTCATCACTACTAAT
 CTCATCAATAACAACCCCGCCATTCTACCCAGACACACACCTTCAACCCGATA
 CCCCCGAACCAACCAACCCCAAACTCACCCCCACAGTTTATGTAACCTCTCTCAA
 GCAATACACTGACCCGCTCAAACTCTGGATTTTGGATCCACCTGCTTGGCCTTAA
 CTAGCTTTCTATTAGCTCTTAGAAGATTACACATGCAAGCACTCCCTCCAGTGAGT
 TCACCTCTAAATCACCAGCATCAAGGAACAAGCATCAAGCACCTTTCAGTGCAGCT
 AAAACGCTTAGCTAGCCACACCTCACCGGGAACAGCAGTGATTAAATTAGCAATAA
 ACGAAAGTTTAACTAAGCTATATCTCCAGGGTTGGTCAATTTCTGTCAGCCACCCG
 GGTCAACAGTAAACCAAGTCAATCAACCGCGGCTAAAGAGTGTATAGATCAACCCC
 TCCCCAATAAGCTAAACCTCACTGTCTGTAATAAACTCCAGTCTCAAAAATAGAC
 TACGAAAGTGGCTTAAACATATCTGAACCTCAATAGCTAAGCTTGGGATTAGA
 TACCCCACTATGCTTAGCCCTAAACCTCAACCTCAACCTCAACCTCAACCTCAACCT
 CACTACGAGCCACAGCTTAAACCTCAAGAGCCTGGCGGTCTTCACTTCAAGAGG
 AGCTGTTCTGTAATCGATAAACCCCGATCAACCTCACCACTCTTGTCTGCTATA
 CCGCCATCTTCAGCAAACTCTGATGAAGGCTACAAAGTAAGCGCAAGTACCTGATAG
 ACGTTAGGTCAGGTTAGCCCATGAGGTGGCAAGAAATGGGCTACATTTCTGATGAT
 AAAACTACGATAGCCCTTATGAACTTAAGGCTCGAAGGTGGATTTAGCAGTAA
 AGTAGAGTCTTAGTTGAACAGGCCCTCGAAGCGCTACACACCGCCGTCACCTT
 AAGTATACTTCAAAGGACATTAACTAAACCCCTACGCATTTATATAGAGGAGACA

```
phix=read_genome('phix.fa')
```

In [77]:

```
def naive (myread,mysequence):
    occurrence = []
    for i in range( len(mysequence) - len(myread) + 1 ):
        match= True
        for j in range(len(myread)):
            if not mysequence[i+j]==myread[j]:
                match =False
                break
        if match==True:
            occurrence.append(i)
    return occurrence

myread='AG'
mysequence='AGTGAGGGAAG'
naive(myread,mysequence)
```

Out[77]:

[0, 4, 9]

In [78]:

```
mysequence[9:(9+len(myread))] #pour vérification
```

Out[78]:

'AG'

Application pour la recherche du matching exact

In [79]:

```
reads=read_fastq("ERR037900_1.first1000.fastq")
X=read_genome("lambda_virus.fa")
nbmatch = 0
n=0
for r in reads:
    matches = naive(r,X)
    n += 1
    if len(matches)>0:
        nbmatch += 1
print ("Nb match:" + str(nbmatch) + " sur " + str(n))
```

Nb match:0 sur 1000

Introduction aux fonctions built-in d'alignement de séquences avec Biopython:

In [2]:

```
import Bio
from Bio.Blast import NCBIWWW
from Bio.Blast import NCBIXML
```

D'où vient ma séquence d'ADN inconnue?

Le package Biopython est capable d'interfacer **BLAST** (<http://blast.ncbi.nlm.nih.gov/Blast.cgi> (<http://blast.ncbi.nlm.nih.gov/Blast.cgi>)) pour la recherche de séquences similaire.

Lecture de fichier fasta avec Biopython:

In [4]:

```
phix=open('phix.fa').read()
```

In []:

```
help(NCBIWWW)
```

Help on module Bio.Blast.NCBIWWW in Bio.Blast:

NAME

Bio.Blast.NCBIWWW - Code to invoke the NCBI BLAST server over the internet.

FILE

/usr/local/lib/python2.7/dist-packages/biopython-1.66-py2.7-linux-x86_64.egg/Bio/Blast/NCBIWWW.py

DESCRIPTION

This module provides code to work with the WWW version of BLAST provided by the NCBI.
<http://blast.ncbi.nlm.nih.gov/> (<http://blast.ncbi.nlm.nih.gov/>)

FUNCTIONS

StringIO(...)
 StringIO([s]) -- Return a StringIO-like stream for reading or writing

qblast(program, database, sequence, auto_format=None, composition_based_statistics=None, db_genetic_code=None, endpoints=None, entrez_query='(none)', expect=10.0, filter=None, gapcost=None, genetic_code=None, hitlist_size=50, i_thresh=None, layout=None, lcase_mask=None, matrix_name=None, nucl_penalty=None, nucl_reward=None, other_advanced=None, perc_ident=None, phi_pattern=None, query_file=None, query_believe_defline=None, query_from=None, query_to=None, searchsp_eff=None, service=None, threshold=None, ungapped_alignment=None, word_size=None, alignment_s=500, alignment_view=None, descriptions=500, entrez_links_new_window=None, expect_low=None, expect_high=None, format_entrez_query=None, format_object=None, format_type='XML', ncbi_gi=None, results_file=None, show_overview=None, megablast=None)

Do a BLAST search using the QBLAST server at NCBI.

Supports all parameters of the qblast API for Put and Get.

Some useful parameters:

- program	blastn, blastp, blastx, tblastn, or tblastx (lower case)
- database	Which database to search against (e.g. "nr").
- sequence	The sequence to search.
- ncbi_gi	TRUE/FALSE whether to give 'gi' identifier.
- descriptions	Number of descriptions to show. Def 500.
- alignments	Number of alignments to show. Def 500.
- expect	An expect value cutoff. Def 10.0.

```

- matrix_name      Specify an alt. matrix (PAM30, PAM70,
BLOSUM80, BLOSUM45).
- filter           "none" turns off filtering. Default
no filtering
- format_type      "HTML", "Text", "ASN.1", or "XML". D
ef. "XML".
- entrez_query     Entrez query to limit Blast search
- hitlist_size     Number of hits to return. Default 50
- megablast        TRUE/FALSE whether to use MEga BLAST
algorithm (blastn only)
- service          plain, psi, phi, rpsblast, megablast
(lower case)

```

This function does no checking of the validity of the parameters

and passes the values to the server as is. More help is available at:

<http://www.ncbi.nlm.nih.gov/BLAST/Doc/urlapi.html> (<http://www.ncbi.nlm.nih.gov/BLAST/Doc/urlapi.html>)

DATA

```

__docformat__ = 'restructuredtext en'
print_function = _Feature((2, 6, 0, 'alpha', 2), (3, 0, 0,
'alpha', 0))...

```

Outils de requete BLAST

In [4]:

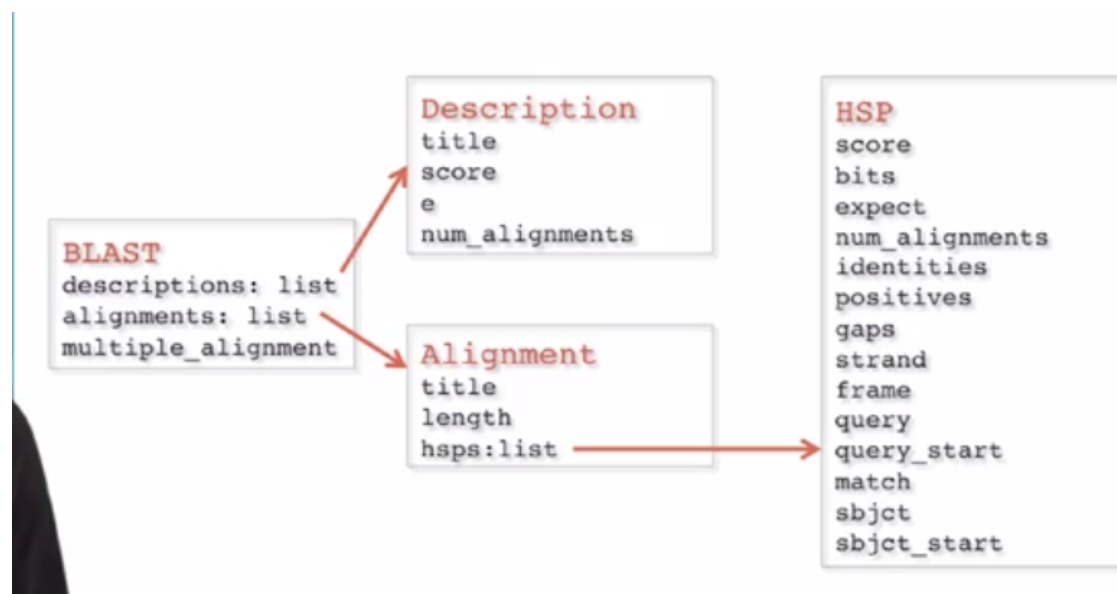
```
output = NCBIWWW.qblast(program="blastn", database="nt", sequence=phix) # l'out
```

Lecture des résultats

In [1]:

```
from IPython.display import Image
Image(filename='query_structure_blast.png')
```

Out[1]:



In [5]:

```
output_record = NCBIXML.read(output)
```

Accéder à la liste des descriptions de l'output:

In [6]:

```
for k in output_record.descriptions:
    print k
```

```
gi|216019|gb|J02482.1|PX1CG Coliphage phi-X174, complete genome
10772.0 0.0
gi|764044028|gb|CP004084.1| Enterobacteria phage phiX174, compl
ete genome 10746.0 0.0
gi|288872851|gb|GU385905.1| Enterobacteria phage phiX174 isolat
e JACSK, complete genome 10746.0 0.0
gi|5815312|gb|AF176034.1| Coliphage phiX174 isolate Anc, comple
te genome 10746.0 0.0
gi|304441978|gb|HM753662.1| Enterobacteria phage phiX174 isolat
e XC+Mad06im6, complete genome 10736.0 0.0
gi|226973390|gb|FJ849058.1| Enterobacteria phage phiX174 isolat
e JACS, complete genome 10736.0 0.0
gi|304441942|gb|HM753659.1| Enterobacteria phage phiX174 isolat
e XC+Mad06ic5, complete genome 10732.0 0.0
gi|304442506|gb|HM753706.1| Enterobacteria phage phiX174 isolat
e XC+mbD10im2, complete genome 10726.0 0.0
gi|304442026|gb|HM753666.1| Enterobacteria phage phiX174 isolat
e XC+Mad10ic6, complete genome 10726.0 0.0
gi|304441990|gb|HM753663.1| Enterobacteria phage phiX174 isolat
e XC+Mad06im7, complete genome 10726.0 0.0
gi|304441954|gb|HM753660.1| Enterobacteria phage phiX174 isolat
e XC+Mad06im1, complete genome >gi|304441966|gb|HM753661.1| Ent
erobacteria phage phiX174 isolate XC+Mad06im3, complete genome
10726.0 0.0
gi|14210821|gb|AF299302.1|AF299302 Coliphage phiX174 isolate C,
complete genome 10726.0 0.0
gi|304442494|gb|HM753705.1| Enterobacteria phage phiX174 isolat
e XC+Mbd10im1, complete genome 10722.0 0.0
gi|304442470|gb|HM753703.1| Enterobacteria phage phiX174 isolat
e XC+Mbd10ic1, complete genome 10722.0 0.0
gi|304442110|gb|HM753673.1| Enterobacteria phage phiX174 isolat
e XC+Mad10im6, complete genome >gi|304442122|gb|HM753674.1| Ent
erobacteria phage phiX174 isolate XC+Mad10im7, complete genome
10722.0 0.0
gi|304442062|gb|HM753669.1| Enterobacteria phage phiX174 isolat
e XC+Mad10im1, complete genome 10722.0 0.0
gi|304441930|gb|HM753658.1| Enterobacteria phage phiX174 isolat
e XC+Mad06ic3, complete genome 10722.0 0.0
gi|304442482|gb|HM753704.1| Enterobacteria phage phiX174 isolat
e XC+Mbd10ic2, complete genome 10716.0 0.0
gi|304442134|gb|HM753675.1| Enterobacteria phage phiX174 isolat
e XC+Mad10im8, complete genome 10716.0 0.0
gi|304442050|gb|HM753668.1| Enterobacteria phage phiX174 isolat
e XC+Mad10ic8, complete genome 10716.0 0.0
gi|304442014|gb|HM753665.1| Enterobacteria phage phiX174 isolat
e XC+Mad10ic4, complete genome 10716.0 0.0
gi|304442002|gb|HM753664.1| Enterobacteria phage phiX174 isolat
e XC+Mad10ic2, complete genome 10716.0 0.0
gi|304269231|gb|HM775306.1| Enterobacteria phage phiX174 strain
gamma, complete genome 10716.0 0.0
gi|125661357|gb|EF380009.1| Enterobacteria phage phiX174 isolat
e AP100, complete genome 10716.0 0.0
```

```
gi|304442038|gb|HM753667.1| Enterobacteria phage phiX174 isolat
e XC+Mad10ic7, complete genome 10712.0 0.0
gi|304442098|gb|HM753672.1| Enterobacteria phage phiX174 isolat
e XC+Mad10im4, complete genome 10706.0 0.0
gi|304442074|gb|HM753670.1| Enterobacteria phage phiX174 isolat
e XC+Mad10im2, complete genome 10706.0 0.0
gi|304442158|gb|HM753677.1| Enterobacteria phage phiX174 isolat
e XC+Mad14ic2, complete genome 10702.0 0.0
gi|304442086|gb|HM753671.1| Enterobacteria phage phiX174 isolat
e XC+Mad10im3, complete genome 10702.0 0.0
gi|262064129|gb|GQ153915.1| Enterobacteria phage phiX174 isolat
e PhiX_ancestral, complete genome 10702.0 0.0
gi|304442182|gb|HM753679.1| Enterobacteria phage phiX174 isolat
e XC+Mad14im2, complete genome 10696.0 0.0
gi|304442146|gb|HM753676.1| Enterobacteria phage phiX174 isolat
e XC+Mad14ic1, complete genome 10696.0 0.0
gi|304442170|gb|HM753678.1| Enterobacteria phage phiX174 isolat
e XC+Mad14im1, complete genome 10686.0 0.0
gi|304442578|gb|HM753712.1| Enterobacteria phage phiX174 isolat
e XC+MbD22ic7, complete genome 10682.0 0.0
gi|304442290|gb|HM753688.1| Enterobacteria phage phiX174 isolat
e XC+Mad20im6, complete genome 10682.0 0.0
gi|304442230|gb|HM753683.1| Enterobacteria phage phiX174 isolat
e XC+Mad20ic5, complete genome 10680.0 0.0
gi|14210820|gb|AF299301.1|AF299301 Coliphage phiX174 isolate C
M, complete genome 10680.0 0.0
gi|304442518|gb|HM753707.1| Enterobacteria phage phiX174 isolat
e XC+MbD22ic2, complete genome >gi|304442530|gb|HM753708.1| Ent
erobacteria phage phiX174 isolate XC+MbD22ic3, complete genome
>gi|304442542|gb|HM753709.1| Enterobacteria phage phiX174 isola
te XC+MbD22ic4, complete genome >gi|304442554|gb|HM753710.1| En
terobacteria phage phiX174 isolate XC+MbD22ic5, complete genome
>gi|304442590|gb|HM753713.1| Enterobacteria phage phiX174 isola
te XC+MbD22im3, complete genome >gi|304442602|gb|HM753714.1| En
terobacteria phage phiX174 isolate XC+MbD22im4, complete genome
>gi|304442614|gb|HM753715.1| Enterobacteria phage phiX174 isola
te XC+MbD22im5, complete genome 10678.0 0.0
gi|304442638|gb|HM753717.1| Enterobacteria phage phiX174 isolat
e XC+MbD22im7, complete genome 10676.0 0.0
gi|304442218|gb|HM753682.1| Enterobacteria phage phiX174 isolat
e XC+Mad20ic3, complete genome 10674.0 0.0
gi|304442650|gb|HM753718.1| Enterobacteria phage phiX174 isolat
e XC+MbD22im2, complete genome 10672.0 0.0
gi|304442626|gb|HM753716.1| Enterobacteria phage phiX174 isolat
e XC+MbD22im6, complete genome 10672.0 0.0
gi|304442566|gb|HM753711.1| Enterobacteria phage phiX174 isolat
e XC+MbD22ic6, complete genome 10672.0 0.0
gi|304442302|gb|HM753689.1| Enterobacteria phage phiX174 isolat
e XC+Mad20im8, complete genome 10672.0 0.0
gi|125661633|gb|EF380032.1| Enterobacteria phage phiX174 isolat
e DEL4, complete genome 10672.0 0.0
gi|5815305|gb|AF176027.1| Coliphage phiX174 isolate S1, complet
e genome 10672.0 0.0
gi|304442386|gb|HM753696.1| Enterobacteria phage phiX174 isolat
e XC+Mad22im1, complete genome >gi|304442458|gb|HM753702.1| Ent
erobacteria phage phiX174 isolate XC+Mad22im8, complete genome
10670.0 0.0
```

```
gi|304442374|gb|HM753695.1| Enterobacteria phage phiX174 isolat  
e XC+Mad22ic8, complete genome 10670.0 0.0  
gi|304442278|gb|HM753687.1| Enterobacteria phage phiX174 isolat  
e XC+Mad20im2, complete genome 10670.0 0.0  
gi|304442254|gb|HM753685.1| Enterobacteria phage phiX174 isolat  
e XC+Mad20ic7, complete genome 10670.0 0.0
```

Accéder à la liste des alignements:

In [7]:

```
for alignment in output_record.alignments:  
    print alignment
```

```
gi|216019|gb|J02482.1|PX1CG Coliphage phi-X174, complete genom  
e  
    Length = 5386
```

```
gi|764044028|gb|CP004084.1| Enterobacteria phage phiX174, comp  
lete genome  
    Length = 5386
```

```
gi|288872851|gb|GU385905.1| Enterobacteria phage phiX174 isola  
te JACSK, complete genome  
    Length = 5386
```

```
gi|5815312|gb|AF176034.1| Coliphage phiX174 isolate Anc, compl  
ete genome  
    Length = 5386
```

```
gi|304441978|gb|HM753662.1| Enterobacteria phage phiX174 isola  
te XC+Mad06im6, complete genome  
    Length = 5386
```

Accéder aux résultats complets de l'alignement (BEST MATCH):

In [8]:

```
threshold = 0.01
for alignment in output_record.alignments:
    for hsp in alignment.hsps:
        if hsp.expect < threshold:
            print("=====ALIGNEMENT=====")
            print("Sequence:", alignment.title)
            print("Longueur:", alignment.length)
            print("E-value:", hsp.expect)
            #print("Query:", hsp.query)
            #print("Match:", hsp.match)
            #print("Subject:", hsp.sbjct)

=====ALIGNEMENT=====
('Sequence:', u'gi|216019|gb|J02482.1|PX1CG Coliphage phi-X17
4, complete genome')
('Longueur:', 5386)
('E-value:', 0.0)
=====ALIGNEMENT=====
('Sequence:', u'gi|764044028|gb|CP004084.1| Enterobacteria pha
ge phiX174, complete genome')
('Longueur:', 5386)
('E-value:', 0.0)
=====ALIGNEMENT=====
('Sequence:', u'gi|288872851|gb|GU385905.1| Enterobacteria pha
ge phiX174 isolate JACSK, complete genome')
('Longueur:', 5386)
('E-value:', 0.0)
=====ALIGNEMENT=====
('Sequence:', u'gi|5815312|gb|AF176034.1| Coliphage phiX174 is
olate Anc, complete genome')
('Longueur:', 5386)
('E-value:', 0.0)
```

TODO pour la prochaine fois:

- Manipulation intensive de séquences
- Prédiction de la localisation de gènes (annotations, etc.)
- Exécuter des lignes de commandes avec nos logiciels préférés (R, Python)
- Ecrire des programmes exécutables sur la ligne de commande sous python
- Reconstruction de l'arbre phylogénétique en utilisant une matrice de distance entre plusieurs génomes et une classification hiérarchique
- Les alignements multiples
- Les méthodes d'alignement et leurs limites
- Aborder l'alignement de protéines et visualiser leur structure [ProDy \(http://prody.csb.pitt.edu/\)](http://prody.csb.pitt.edu/)...

Références:

- [Installer Jupyter chez vous \(http://jupyter.readthedocs.org/en/latest/install.html\)](http://jupyter.readthedocs.org/en/latest/install.html)
- [Installer R dans Jupyter \(https://github.com/IRkernel/IRkernel\)](https://github.com/IRkernel/IRkernel)
- [Installer numpy \(http://www.scipy.org/scipylib/download.html\)](http://www.scipy.org/scipylib/download.html)

- [Installer pandas \(http://pandas.pydata.org/\)](http://pandas.pydata.org/)
- [Installer Biopython \(http://biopython.org/DIST/docs/install/Installation.html\)](http://biopython.org/DIST/docs/install/Installation.html)
- [Installer Scikit-Bio \(http://scikit-bio.org/\)](http://scikit-bio.org/)

Feedback par rapport au cours ?