Fiche Mémo - Python

1 Syntaxe élémentaire

- Commentaires :

```
# Ceci est un commentaire
```

- Instructions conditionnelles :

```
if x > 2:
  print("x > 2")
elif x > 1:
  print("x > 1")
else:
  pass
```

> ✖ **Attention !** En Python, les blocs **doivent** être indentés, c'est-à-dire qu'ils doivent être décalés vers la droite d'un nombre d'espaces (ou de tabulations) constant. La fin du bloc n'est pas donnée par un mot-clef mais directement par le retour à un autre alignement.

- Boucle **for** :

```
for i in range(3,9):
  print(i)
```

- Boucle **for** appliqué à un objet que l'on peut parcourir (chaîne de caractères, vecteur, matrice...) :

```
for c in "Test":
  # ord(c) :
  # convertir "c" en code ASCII
  print(ord(c))
```

- Boucle **while** :

```
n = 1
while n < 19:
  print(n)
  n *= 3
```

- Définir une fonction :

```
def add(a,b):
  c = a+b
  return c
```

- Utiliser une bibliothèque :

```
# Importer en utilisant un nom
import numpy as np
# Importer sans utiliser de nom
from numpy import *
```

- Aide en ligne :

```
help(numpy.eye)
```

2 Bibliothèque numpy

Toutes les commandes suivantes font parties de numpy que l'on suppose importé avec **import** numpy as np.

- Créer une matrice numpy :

```
T = np.array([[1,2],[3,4]])
```

- Créer une matrice de zéros ou de uns :

```
A = np.zeros((3,4))
B = np.ones((5,2))
```

- Matrice identité :

```
I = np.eye(5)
```

- 10 nombres de 3 à 5 inclus régulièrement espacés :

```
L = np.linspace(3,5,10)
```

- Équivalent de range dans numpy :

```
R = np.arange(2,9)
```

- Obtenir la taille d'une matrice M :

```
np.shape(M)
```

- Produit matriciel :

```
P = np.dot(M,N)
```

- Transposée :

```
N = np.transpose(M)
```

- Fonctions applicables à une matrice entière ou à une colonne :
  np.sum, **np.min**, **np.max**, np.mean, np.cumsum, np.median, np.var, np.std.
- Fonctions qui s'appliquent à chaque élément d'une matrice séparément :
  np.exp, np.log, np.sin, np.cos, np.sqrt, **np.abs**, np.floor.
- Valeurs utiles :
  np.e, np.pi

3 Bibliothèque `numpy.linalg`

Toutes les commandes suivantes font parties de `numpy.linalg` que l'on suppose importé avec **`import`** `numpy.linalg as al`.

- Inverser une matrice :

```
1 al.inv(M)
```

- Déterminer le rang d'une matrice :

```
1 al.rank(M)
```

- Calculer M^k :

```
1 al.matrix_power(M,k)
```

- Résoudre le système $AX = B$:

```
1 al.solve(A,B)
```

- Déterminer les valeurs et vecteurs propres de M :

```
1 al.eig(M)
```

4 Bibliothèque `numpy.random`

Toutes les commandes suivantes font parties de `numpy.random` que l'on suppose importé avec **`import`** `numpy.random as rd`.
Les commandes suivantes génèrent un tableau de k valeurs aléatoires suivant une loi donnée

- Loi uniforme sur $[0, 1]$:

```
1 rd.random(k)
```

- Loi binomiale de paramètre n et p :

```
1 rd.binomial(n,p,k)
```

- Loi géométrique de paramètre p :

```
1 rd.geometric(p,k)
```

- Loi de Poisson de paramètre `mu` :

```
1 rd.poisson(mu,k)
```

- Loi (à densité) exponentielle de paramètre `mu` :

```
1 rd.exponential(1/mu,k)
```

- Loi normale d'espérance `mu` et d'écart-type `sigma` :

```
1 rd.normal(mu,sigma,k)
```

- Loi γ de paramètres k et `theta` :

```
1 rd.exponential(n,theta,k)
```

- Loi uniforma sur $[\![n, m]\!]$:

```
1 rd.randint(n,m,k)
```

Sans le dernier argument k, les mêmes fonctions donnent une unique valeur tirée aléatoirement selon la même loi.

5 Bibliothèque `scipy.special`

L'import de `scipy.special` est fait avec **`import`** `scipy.special as sp`. On pourra utiliser la fonction Φ (intégrale partielle de la densité Gaussienne) de la manière suivante :

```
1 sp.ndtr(x)
```

6 Graphiques

On importera `matplotlib` avec :

```
1 import matplotlib.pyplot as plt
```

Il y a une fonction à maitriser pour le tracer d'une fonction :

```
1 X = np.arange(1,11)
  Y = np.linspace(2,5,10)
  plt.plot(X,Y)
  plt.show()
```