Annexe TP1

Particularité en python

- L'indentation
- Pas de déclaration
- Typage inféré (deviné par python)
- Pas de persistence de type
- = : affectation :
- == : egalité :
- Les opérateurs booléens sont : == ; != ; < ; <= ; > ; >= ;or ; and not
- Opérations entières à noter
    - // : division entière
    - % : pour le modulo

Exemple :

```
x = 5
print(type(x)) # va repondre int
x = "abc" # aucun probleme
print(type(x)) # va repondre str
```

Structures de contrôle

IF

```
if <condition> :
   <code1>
else :
  <code2>
```

Numpy :

Numpy est une bibliothèque très performante (code C optimisé appelable depuis Python) Structure de base = tableau (homogène) multidimensionnel : numpy.array http://www.numpy.org/

1. Construire un tableau NumPy :
   On peut créer un tableau (unidimensionnel : un vecteur) NumPy à partir d'une liste Python ou même d'une liste de listes (il sera alors bi-dimensionel : une matrice) :

```
>>> import numpy
>>> numpy.array([1, 2, 3])
array([1, 2, 3])
>>> numpy.array([[1, 2], [3, 4], [5,6]])
array([[1, 2],
       [3, 4],
       [5, 6]])
>>> a.shape[0]
3
>>> a.shape[1]
2
```

2. NumPy permet de faire des opérations directement sur ces tableaux, sans avoir à réécrire chaque fonction :

```
>>> 2 * numpy.array([1, 2, 3])
              + numpy.array([8, 10, 12])
array([10, 14, 18])
```

3. Accéder aux éléments d'un tableau NumPy :
   - Accès aux éléments comme pour une liste : a[i]
   - Numérotation à partir de 0
   - Matrices : M[ligne][colonne]

   - **Exemple** :

```
>>> b = numpy.array([1.414, 0.5, 3.14, 2.718])
>>> b[2]
3.1400000000000001
>>> a = numpy.array([[ 0.19,  0.14,  0.21],
...                  [ 0.79,  0.43,  0.74],
...                  [ 0.12,  0.40,  0.30]])
>>> a[2, 1]
0.40000000000000002
```

   a. Accès aux lignes : a[numéro de ligne, :]
   b. Accès aux colonnes : a[:,numéro de colonne]
      Exemple :

```
>>> a = numpy.array([[ 0.19,  0.14,  0.21],
...                  [ 0.79,  0.43,  0.74],
...                  [ 0.12,  0.40,  0.30]])

>>> a[:, 1]
array([ 0.14,  0.43,  0.4 ])
>>> a[0, :]
array([ 0.19,  0.14,  0.21])
```

   c. Tranche de tableau :

```
>>> v = numpy.array([-1, -2, -3, -4, -5])
# elements entre les indices 2 (inclus) et 4 (exclu)
>>> v[2:4]
array([-3, -4])
# elements entre les indices 0 (inclus) et 3 (exclu)
>>> v[:3]
array([-1, -2, -3])
# elements entre les indices 2 (inclus)
#                          et fin (inclus)
>>> v[2:]
array([-3, -4, -5])
```

4. Opérations sur les tableaux NumPy :

- Déjà vu : `M1 + M2, scalaire * M`
- Opérations classiques, composante par composante : `-, *, /, abs`

```
>>> u = numpy.array([1, 2, 3, 4])
>>> v = numpy.array([4, 3, 2, 1])
>>> u * v
array([4, 6, 6, 4])
>>> u / v
array([ 0.25,  0.66666667,  1.5,  4.])
>>> abs(u - v)
array([3, 1, 1, 3])
```

5. Produits matriciels, produits scalaires : numpy.dot

```
>>> u = numpy.array([1, 2, 3, 4])
>>> v = numpy.array([4, 3, 2, 1])
>>> numpy.dot(u, v)   # sum_{i=0}^{n-1} u_i · v_i
20
>>> w = u * v
>>> w.sum()
20
>>> a = numpy.array([[1, 2], [3, 4]])
>>> x = numpy.array([1, -1])
>>> b = numpy.dot(a, x)
>>> b
array([-1, -1])
```