numpy

Pour utiliser la bibliothèque (ou module) **numpy** dans un programme Python, il est nécessaire de l'importer.

• Méthode **array( )**

Dans la console ou l'interpréteur Python de **IDLE**, par exemple : — Taper :

```
>>>   import numpy
>>>   numpy.array([0, 1, 2, 3])
      array([0, 1, 2, 3])
```

L'instruction ci-dessus appelle, dans le module **numpy**, la méthode array( ), laquelle permet de créer un objet de type array contenant la liste de nombres [0, 1, 2, 3].

Pour éviter de taper systématiquement numpy, on importera le module sous le raccourci np.

```
>>>   import numpy as np
>>>   np.array([0, 1, 2, 3])
      array([0, 1, 2, 3])
>>>   N = np.array([0, 1, 2, 3, 4, 5])
>>>   N
      array([0, 1, 2, 3, 4, 5])
```

• Méthode **arange( )**

Pour générer N automatiquement, il existe une méthode plus adaptée que la méthode array( ).

```
>>>   N = np.arange(0, 6, 1)
>>>   N
      array([0, 1, 2, 3, 4, 5])
```

L'instruction np.arange([valeur initiale, valeur finale, pas]) renvoie la liste de nombres commençant à la valeur initiale, en progressant jusqu'à la "valeur finale - 1" avec le pas indiqué.

```
>>>   L = np.arange(0, 20, 3)
>>>   L
      array([0, 3, 6, 9, 12, 15, 18])
```

• Méthode **linspace( )**

Pour générer n nombres entre valeur initiale et valeur finale, on utilisera la méthode linspace(valeur initiale, valeur finale, nombre de valeurs).

```
>>>   x = np.linspace(0, 10, 11)
>>>   x
      array([0., 1., 2., 3., 4., 5., 6., 7., 8., 9., 10.])
```

Cette méthode est particulièrement adaptée pour générer les n abscisses de n points sur un intervalle $[a; b]$.

>>> x = np.linspace(a, b, n). Cette instruction construit une liste de n nombres allant de a à b (inclus).

```
>>>   x = np.linspace(5, 15, 21)
>>>   x
      array([ 5., 5.5, 6., 6.5, 7., 7.5, 8., 8.5, 9., 9.5, 10., 10.5,
11., 11.5, 12., 12.5, 13., 13.5, 14.,      14.5, 15.])
```

• Méthodes **zeros( )**, **ones( )** et **full( )**

Tester les instructions ci-dessous et en déduire leur rôle :

```
>>>   L = np.zeros(5)
>>>   L
      array([0., 0., 0., 0., 0.])

>>>   L = np.ones(4)
>>>   L
      array([1., 1., 1., 1.])

>>>   L = np.full(4, 25)
>>>   L
      array([25, 25, 25, 25])
```

```
>>>   L = np.zeros((2,2))
>>>   L
      array([[0., 0.],
             [0., 0.]])

>>>   L = np.ones((2,3))
>>>   L
      array([[1., 1., 1.],
             [1., 1., 1.]])

>>>   L = np.full((2,2), 25)
>>>   L
      array([[25, 25],
             [25, 25]])
```

• Méthodes de **génération aléatoire**

Tester les deux instructions ci-dessous :

```
>>>   L = np.random.rand(4) # liste 4 valeurs aléatoires entre 0 e
>>>   L
      array([0.87674625, 0.59710785, 0.11285654, 0.16861321])

>>>   L = np.random.randint(1, 6, 10)# simulation dix lancers de dé
>>>   L
      array([5, 2, 1, 2, 3, 1, 2, 3, 5, 2])
```

• Méthodes **mean( )**, **std( )** et **sum( )**

Tester les trois instructions ci-dessous :

```
>>> L = np.array([5, 12, 9, 14, 20, 15, 11, 11, 13, 10]) # notes
    entre 0 et 20 de 10 étudiants
>>> len(L)
    10
>>> np.mean(L) # renvoie la moyenne des notes au format numpy
    np.float64(12.0)
>>> longueur = int(np.mean(L)) # renvoie la moyenne des notes au
    format classique python
>>> longueur
    12
>>> np.std(L) # renvoie l'écart-type
    np.float64(3.7682887362833544)
>>> np.sum(L) # renvoie la somme des notes
    np.int64(120)
```

• **Fonctions usuelles**

Tester les instructions ci-dessous :

```
>>> X = np.arange(0, 10, 11)
>>> X
    array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
>>> X**2
    array([  0,   1,   4,   9,  16,  25,  36,  49,  64,  81, 100])
>>> 2*X+1
    array([ 1,  3,  5,  7,  9, 11, 13, 15, 17, 19, 21])
```

```
>>> X = np.arange(0, 91, 15) # renvoie des angles exprimés en
    degrés
>>> X
    array([ 0, 15, 30, 45, 60, 75, 90])
>>> D=np.deg2rad(X) # exprime en radians les angles exprimés en
    degrés dans X
    array([0., 0.26179939, 0.52359878, 0.78539816, 1.04719755,
    1.30899694, 1.57079633])
>>> Y=np.sin(D) # calcul le sinus des angles exprimés en radians
    array([0., 0.25881905, 0.5, 0.70710678, 0.8660254, 0.96592583,
    1.])
>>> Z=np.cos(D) # calcul le cosinus des angles exprimés en radians
    array([1.00000000e+00, 9.65925826e-01, 8.66025404e-01,
    7.07106781e-01,5.00000000e-01, 2.58819045e-01, 6.12323400e-17])
```

```
>>> X = np.arange(1, 6)
>>> X
    array([1, 2, 3, 4, 5])
>>> np.exp(X) # fonction exponentielle
    array([  2.71828183,   7.3890561 ,  20.08553692,  54.59815003,
    148.4131591 ])
>>> np.log(X) # logarithme népérien
    array([0., 0.69314718, 1.09861229, 1.38629436, 1.60943791])
>>> np.sqrt(X) # racine carrée
    array([1., 1.41421356, 1.73205081, 2., 2.23606798])
```

• L'instruction **meshgrid( )**

Tester les instructions ci-dessous :

```
>>>   x = np.arange(-2, 3, 1)
>>>   x
      array([-2, -1,  0,  1,  2])
>>>   y = np.arange(-1, 2, 1)
>>>   y
      array([-1,  0,  1])
>>>   X, Y = np.meshgrid(x,y)
>>>   X
      array([[-2, -1,  0,  1,  2],
                 [-2, -1,  0,  1,  2],
                 [-2, -1,  0,  1,  2]])
>>>   Y
      array([[-1, -1, -1, -1, -1],
                  [ 0,  0,  0,  0,  0],
                  [ 1,  1,  1,  1,  1]])
```

La méthode meshgrid( ) permet de construire le quadrillage complet (x ; y) pour les valeurs de x et de y définies.
Considérons la fonction $f : (x, y) \longmapsto x^2 + y$.
Pour obtenir l'image de chacun des couples (x ; y), il suffit de taper :

```
>>>   Z = X**2 + Y
>>>   Z
      array([[ 3,  0, -1,  0,  3],
                 [ 4,  1,  0,  1,  4],
                 [ 5,  2,  1,  2,  5]])
```

Cette méthode permettra d'afficher des points de coordonnées (x ; y ; z) dans l'espace en calculant leur hauteur z à partir de leurs coordonnées positionnelles (x ; y).

• **Avantages de la bibliothèque numpy**

Tester les instructions ci-dessous :

```
>>>   import numpy as np
>>>   L = list(range(1000)) #génère une liste de 1000 nombres allant
      de 0 à 999.
>>>   [x**2 for x in L] #génère la liste des carrés des nombres
      allant de 0 à 999.
      Squeezed text (95 lines)
>>>   a = np.array(L) #génère un objet numpy de type array contenant
      les nombres de 0 à 999.
>>>   [x**2 for x in a] #gain temps de 30% pour calculer les carrés
      des nombres de 0 à 999.
      Squeezed text (220 lines)
>>>   a**2 #obtention par vectorisation des carrés - temps de calcul
      divisé par 250
      Squeezed text (125 lines)
```