Numpy : les bases à connaître et pièges à éviter

```python
import numpy as np
import math

## Différences et lien entre une liste et un np.array ?

# ATTENTION, même si a = [1,2] ressemble à un vecteur mathématique,
# il s'agit d'un objet type "list" et n'autorise pas les opérations
# mathématiques, contrairement à un np.array

a = [1,5] # une liste
# a/2

# Pour passer d'une liste à un np.array, on peut utiliser la fonction de convertion np.array()
v = np.array([1,5])
print(v/2)


## Utilisation des fonctions de la bibliothèque numpy
# Quand on appelle une fonction de la biblio numpy, on doit écrire np.nom_fonction() (comme pour math.cos)

np.cos(2)
math.cos(2)

# Un avantage de la bibliothèque numpy est que les fonctions peuvent prendre en argument
# des "vecteurs" qui sont des objets de type "np.array"

v = np.array([1,5])
np.cos(v)
# math.cos(v) n'est pas autorisé


## Comment définir un np.array ?
# 1) A partir d'une liste
v = np.array([1,5,10,np.pi,1/3])

# 2) On peut aussi définir des np.array directement (sans passer par le type liste)
# avec des fonctions toutes faites de numpy
a = 1
b = 10
n = 10
w = np.linspace(a,b,n) # un np.array rempli de n réels équidistants entre a et b (inclus)
print(w)


## Comment travailler avec un np.array ?
# ATTENTION les indices vont de 0 à (longueur_du_vecteur - 1) !
v = np.array([1,5,10,2,-5])
print(f"v = {v}")
v[2] # 3ème valeur dans v
v[1:4] # un np.array composé des valeurs aux indices de 1 à (4-1) inclus

# ATTENTION si on veut copier une partie d'un np.array, l'égalité suivante
# ne définit pas un nouveau emplacement mémoire donc un changement sur w
# implique un changement sur v
w = v[1:4]
print(f"w = {w}")
w[0] = 0 # modification du premier élément de w
print(f"w = {w}")
print(f"v = {v}")
# pour faire une véritable copie, utiliser np.copy(v)


## Les matrices
# De même que pour les np.array, même si A =  [ [1,2] , [4,7]] ressemble à une matrice,
# on ne peut pas faire d'opérations mathématiques dessus,
# en effet la somme revient à une concaténation

A_liste = [ [1,2] , [3,4]]
B_liste =  [ [10,20] , [-4,7]]
print(A_liste + B_liste)
```

```python

# pour faire des opérations comme sur des matrices mathématiques,
# on convertit avec la même fonction que pour les vecteurs
A_matrice = np.array(A_liste)
B_matrice = np.array(B_liste)
print(A_matrice + B_matrice)

# ATTENTION, on obtient un np.ndarray et non un np.array :
# il faut faire attention car la transposée d'un np.array ne change rien
v = np.array([1,2]) # vecteur 1 colonne, 2 lignes
print(f"taille de v est {v.shape}")
print(f"v={v}")
v_transpose = v.T
print(f"v_transpose={v_transpose}")
v_colonne = v.reshape(1, 2)
print(f"v_colonne={v_colonne}")
print(f"taille de v_ est {v_colonne.shape}")

# ATTENTION, pour le produit matriciel ce n'est pas * (qui fait produit termes à termes) mais np.dot
# print(A_matrice * B_matrice)
# print(np.dot(A_matrice,B_matrice))


## Introduction à matplotlib
import matplotlib.pyplot as plt

# On utilise la bibliothèque matplotlib du module pyplot
# où mat vient de "matlab", plot pour "afficher", lib pour "library" et py pour python.

# La commande plot de cette bibliothèque prend en entrée une liste Python
# ou un np.array d'abscisses et une autre d'ordonnées,
# et trace les lignes brisées entre deux points consécutifs.
# ATTENTION il ne faut pas oublier plt.show() pour afficher le graphique
# ou plt.legend() pour afficher les légendes.
```