FACULTE DES SCIENCES AIN CHOCK
UNIVERSITE HASSAN II DE CASABLANCA

Département: Mathématiques & Informatique

Séance 9:
Analyse de données en Python
Module : Numpy

Licence : Physique Chimie

Pr: Youssef Ouassit

Introduction

1. Les bibliothèques Python pour l'analyse de données
2. Lecture, sélection et filtrage des données
3. Manipulation des données:
   - Tri, regroupement, arrangement.
4. Tracer les courbes des fonctions
5. Fonctions de la statistique descriptive

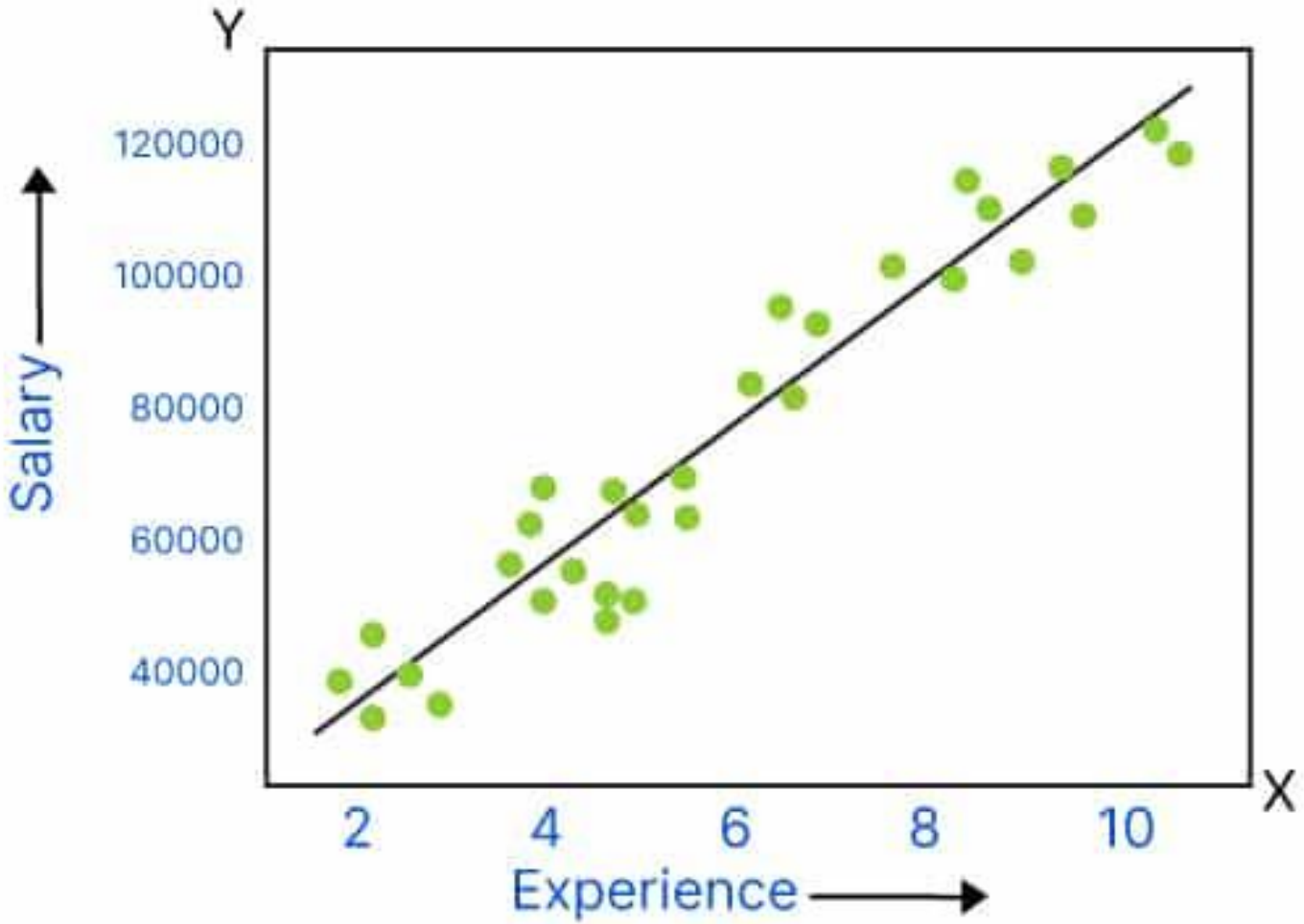
Introduction

L'analyse de données consiste à examiner, nettoyer, transformer et modéliser des données dans le but de découvrir des informations utiles, tirer des conclusions et soutenir la prise de décision.

Python est l'un des langages les plus utilisés pour l'analyse de données en raison de sa simplicité, de ses bibliothèques puissantes et de sa communauté active.

Introduction

Exemple des modèles d'analyse de données: La régression linéaire



Introduction

Bibliothèques essentielles pour l'analyse de données:

Python propose de nombreuses bibliothèques dédiées à l'analyse de données, notamment :

NumPy : Pour les opérations mathématiques et manipulation des tableaux.
Matplotlib : Pour la visualisation des données.
Pandas : Pour la manipulation et l'analyse des structures de données.

Le module numpy

Introduction

NumPy est une bibliothèque fondamentale pour le calcul scientifique en Python. Elle fournit un support pour les tableaux (arrays) multidimensionnels, ainsi que des fonctions mathématiques et logiques pour opérer sur ces tableaux de manière efficace.

NumPy est particulièrement utilisé pour des tâches nécessitant des opérations sur des données numériques, comme la manipulation de matrices, l'algèbre linéaire, le calcul statistique, et bien plus encore.

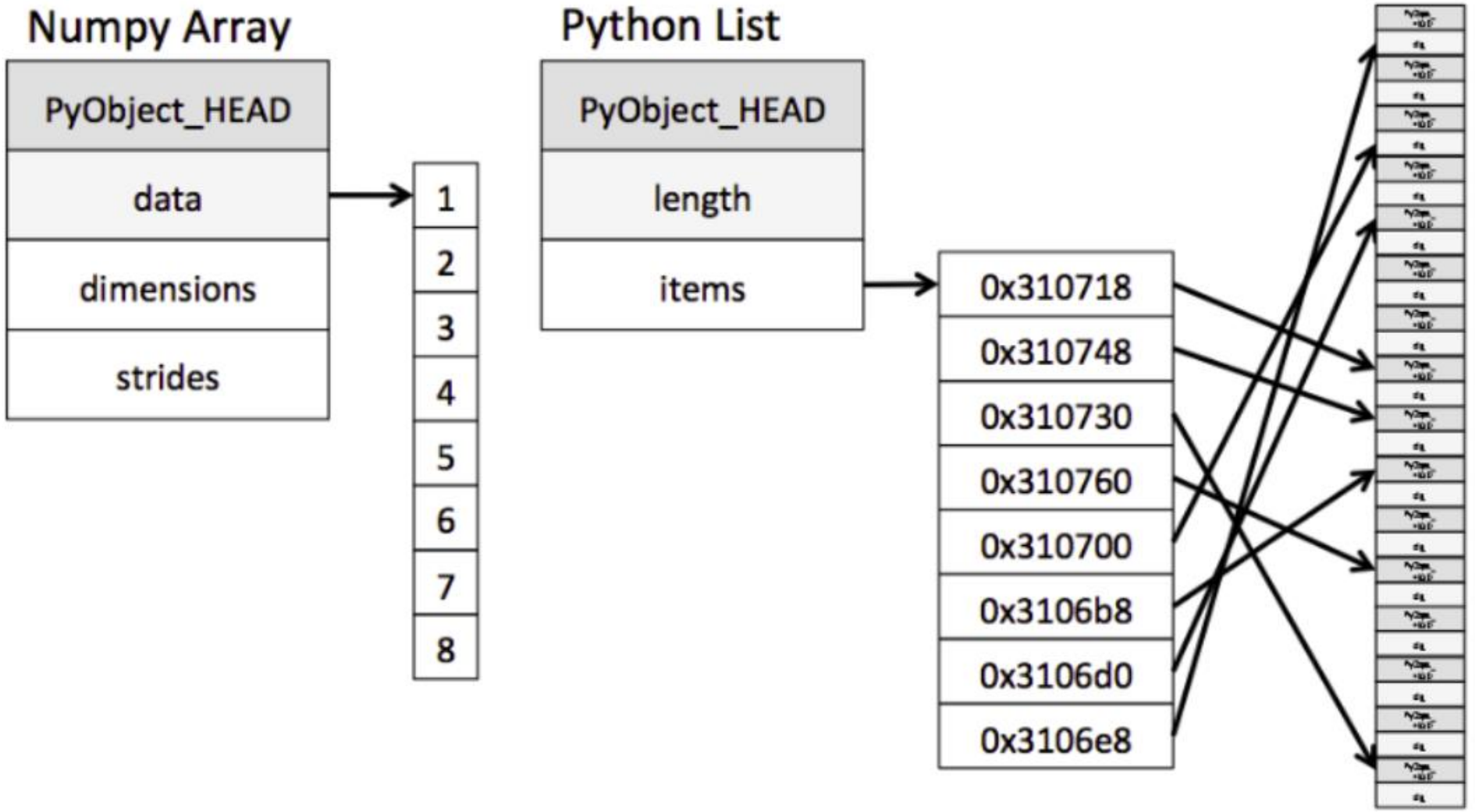
Le module numpy

Introduction

Caractéristiques :

- Tableaux Multidimensionnels (ndarray) :
- Opérations Vectorisées (diffusion de l'opérateur)
- Fonctions Mathématiques et Statistiques
- Manipulation de Tableaux
- Génération de Données
- Indexation Avancée et Slicing
- Compatibilité avec d'autres bibliothèques

Introduction



Le module numpy

Importer le module numpy

Pour installer numpy , il faut exécuter la commande suivante :

```
pip install numpy
```

Cette commande est à exécuter sur le console d'exécution ou l'invite de commande.

Le module numpy

Installer le module numpy

Une fois que **NumPy** est installé, vous pouvez l'importer dans votre code Python de la manière suivante :

```
import numpy
```

Pour appeler une fonction :

```
numpy.sin(x)
```

Installer le module numpy

Une fois que **NumPy** est installé, vous pouvez l'importer dans votre code Python de la manière suivante :

```
import numpy as np
```

np est un alias couramment utilisé pour NumPy. Cela vous permet d'utiliser des fonctions et des objets de NumPy de manière plus concise.

Pour appeler une fonction :
numpy.sin(x)

Le module numpy

Déclaration d'un vecteur

Déclarer un tableau un à une seule dimension

Le module numpy

Déclaration d'un vecteur

- Créer un tableau 1D à partir d'une liste:

La fonction np.array() est l'une des façons les plus courantes de créer des tableaux 1D. Elle convertit une liste Python (ou un autre objet itérable) en un tableau NumPy.

```
A = np.array([1, 2, 3, 4, 5])
```

Déclaration d'un vecteur

- Créer un tableau 1D rempli de zéros:

La fonction np.zeros() crée un tableau 1D rempli de zéros. Vous devez spécifier la taille du tableau.

```
A = np.zeros(5)      # donne [ 0.0  0.0  0.0  0.0  0.0]
```

Déclaration d'un vecteur

- Créer un tableau 1D rempli des uns:

La fonction np.zeros() crée un tableau 1D rempli des uns. Vous devez spécifier la taille du tableau.

```
A = np.ones(5)     # donne [ 1.0  1.0  1.0  1.0  1.0]
```

Le module numpy

Déclaration d'un vecteur

- Créer un tableau 1D avec une séquence de nombres:

np.arange() génère un tableau avec une séquence d'entiers ou de nombres à intervalles réguliers, similaire à la fonction range() en Python.

```
# Créer un tableau 1D avec une séquence de nombres de 0 à 9
A = np.arange(10)

# Créer un tableau 1D avec des nombres allant de 0 à 10 (pas de 2)
B = np.arange(0, 10, 2)
```

Le module numpy

Déclaration d'un vecteur

- Créer un tableau 1D avec un nombre spécifié d'éléments uniformément espacés:

np.linspace() génère un tableau 1D avec des nombres répartis de manière uniforme entre deux bornes.

```
# Créer un tableau 1D avec 5 nombres uniformément espacés entre 0 et 1
A = np.linspace(0, 1, 5)
```

Le module numpy

Déclaration d'un vecteur

- Créer un tableau 1D avec des nombres aléatoires entre 0 et 1:

np.random.rand() crée un tableau 1D avec des nombres aléatoires uniformément distribués entre 0 et 1.

```
# Créer un tableau 1D de 5 éléments avec des valeurs aléatoires entre 0 et 1
A = np.random.rand(5)
```

Déclaration d'un vecteur

- Créer un tableau 1D avec des entiers aléatoires dans une plage donnée:

np.random.randint() génère un tableau 1D d'entiers aléatoires dans une plage spécifiée..

```
# Créer un tableau 1D de 5 entiers aléatoires entre 0 et 10
A = np.random.randint(0, 10, 5)
```

Déclaration d'un vecteur

- Répéter un tableau 1D:

np.tile() répète un tableau 1D plusieurs fois pour créer un tableau plus grand.

```
# Créer un tableau 1D
A = np.array([1, 2, 3])

# Répéter ce tableau 3 fois
B = np.tile(A, 3)  # Donne [1 2 3 1 2 3 1 2 3]
```

Déclaration d'un vecteur

Déclaration des matrices

Déclaration d'un vecteur

- Créer une matrice à partir d'une liste:

La fonction array() crée une matrice de la forme spécifiée, et tous ses éléments sont initialisés à zéro.

```
# Créer une matrice 2x3
M = np.array( [[1,2,3],[4,5,6]] )

# Créer une matrice 2x3
A = np.array( [[1,2,3],[4,5,6], [0,1,2]] )
```

Déclaration d'un vecteur

- Créer une matrice remplie de zéros:

La fonction np.zeros() crée une matrice de la forme spécifiée, et tous ses éléments sont initialisés à zéro.

```
# Créer une matrice 3x3 remplie de zéros
M = np.zeros((3, 3))
```

Déclaration d'un vecteur

- Créer une matrice remplie des uns:

La fonction np.ones() crée une matrice de la forme spécifiée, et tous ses éléments sont initialisés à un.

```
# Créer une matrice 3x3 remplie des uns
M = np.ones((3, 3))
```

Déclaration d'un vecteur

- Créer une matrice identité:

La fonction np.eye() crée une matrice carrée de taille spécifiée, avec des 1 sur la diagonale principale et des 0 ailleurs.

```
# Créer une matrice identité 4x4
M = np.eye(4)
```

Déclaration d'un vecteur

- Créer une matrice diagonale:

np.diag() permet de créer une matrice diagonale à partir d'un tableau 1D. Les éléments du tableau 1D seront placés sur la diagonale principale de la matrice.

```
# Créer une matrice diagonale à partir d'un tableau
M = np.diag([1, 2, 3])
```

Déclaration d'un vecteur

- Créer une matrice avec des nombres aléatoires uniformément répartis:

np.random.rand() génère une matrice remplie de nombres aléatoires entre 0 et 1, suivant une distribution uniforme.

```
# Créer une matrice 3x3 avec des nombres aléatoires entre 0 et 1
M = np.random.rand(3, 3)
```

Déclaration d'un vecteur

- Créer une matrice avec des entiers aléatoires:

np.random.randint() génère une matrice avec des entiers aléatoires dans une plage spécifiée.

```
# Créer une matrice 3x3 avec des entiers aléatoires entre 0 et 10
M= np.random.randint(0, 10 , (3, 3))
```

Le module numpy

Déclaration d'un vecteur

Fonctions

Le module numpy

Déclaration d'un vecteur

On retrouve les mêmes fonctions déjà vus :

- **len**
- **max**
- **min**
- **sum**
- **sorted**

Déclaration d'un vecteur

Indexation et extraction des valeurs

Le module numpy

Déclaration d'un vecteur

L'indexation nous permet d'accéder aux éléments de la liste. Cependant, il existe d'autres moyens, tels que l'indexation de fantaisie, le découpage et le masquage.

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

# Accéder au dernier  élément de A
# Ces deux instructions d'impression génèrent le même résultat
print(A[-1])
print(A[3])
```

Le module numpy

Déclaration d'un vecteur

```
A = np.array([7, 0, 1, 9, 8, 5])

#plage d'indices contigus
print(A[1:3]) # [0  1]

#extrêmes, début à 3 (non-inclus)
print(A[:3]) # [7  0  1]

#extrêmes, 2 à fin
print(v[2:]) # [1  9  8  5]
```

Le module numpy

Déclaration d'un vecteur

```
A = np.array([7, 0, 1, 9, 8, 5])

#indice négatif
print(A[-1]) # 5  , dernier élément

#indices négatifs
print(A[-3:]) # [9  8  5], 3 derniers éléments

#on peut utiliser une condition pour l'extraction
A[ A < 7] # [0  1  5]
```

Déclaration d'un vecteur

La diffusion de l'opérateur

La diffusion de l'opérateur

La **diffusion** (ou **broadcasting**) est une fonctionnalité puissante de **NumPy** qui permet d’effectuer des opérations sur des tableaux de différentes formes (dimensions) sans avoir besoin de les rendre explicitement de même forme.
Cela signifie que **NumPy** peut automatiquement étendre (ou "diffuser") un tableau plus petit pour qu’il corresponde à la forme d'un tableau plus grand, tout en respectant les règles de diffusion.

La diffusion de l'opérateur

Addition d'un scalaire à un tableau:

```
# Un tableau 1D
A = np.array([1, 2, 3])

# Ajouter un scalaire
B  = A + 10   # 10 est diffusé sur chaque élément du tableau  donne [11 12 13]
```

La diffusion de l'opérateur

Multiplication d'un scalaire à un tableau:

```
# Un tableau 1D
A = np.array([1, 2, 3])

# Ajouter un scalaire
B  = A * 10   # 10 est diffusé sur chaque élément du tableau  donne
[10  20  30]
```

La diffusion de l'opérateur

Diffusion entre deux tableaux de formes compatibles:

```
# Un tableau 2D
A = np.array([1, 2, 3])

# Un tableau 1D
B = np.array([10, 20, 30])

# Addition de A à B
C = A + B # A est diffusé sur chaque ligne de B, donne [  11  22  33 ]
```

La diffusion de l'opérateur

Diffusion entre deux tableaux de formes compatibles:

```
# Un tableau 2D
A = np.array([1, 2, 3])

# Un tableau 1D
B = np.array([10, 20, 30])

# Multiplication de A à B
C = A * B # A est diffusé sur chaque ligne de B, donne [  10  40  90 ]
```

La diffusion de l'opérateur

Appliquer des fonctions:

```
# Un tableau 2D
A = np.array([4, 16, 36])

# Racine carré de A
np.sqrt(A) # donne [  2   4   6]
```

La diffusion de l'opérateur

Appliquer des fonctions:

```
# Un tableau 2D
A = np.array([4, 16, 36])

# Que donne ces instructions
R = np.sin(A)**2  + np.cos(A)**2  # [ 1.0  1.0  1.0]
```

Les fonctions statistiques

Les fonctions statistiques

Les fonctions statistiques

Calculer la moyenne d'un tableau:

```
A = np.array([1, 2, 3, 4, 10])
m = np.mean(A)  # donne 4
```

Les fonctions statistiques

Calculer la médiane d'un tableau:

```
A = np.array([1, 2, 3, 4, 6])
m = np.median(A)  # donne 3

A = np.array([1, 2, 3, 4])
m = np.median(A)  # donne 2.5
```

Les fonctions statistiques

Calculer l'écart-type d'un tableau:

```
A = np.array([1, 2, 3, 4])
m = np.std(A)  # donne 1.118
```

Les fonctions statistiques

Trouver l'index du maximum ou du minimum:

```
A = np.array([1, 7, 0, 4])
max_index = np.argmax(A)  # 1 (indice du plus grand élément)
min_index = np.argmin(A)  # 2 (indice du plus petit élément)
```

Opérations sur les matrices

Opérations sur les matrices

Opérations sur les matrices

Somme matricielle :

$$\begin{pmatrix} 1 & 2 \\ 1 & 3 \end{pmatrix} + \begin{pmatrix} 4 & 1 \\ 5 & 3 \end{pmatrix} = \begin{pmatrix} 5 & 3 \\ 6 & 6 \end{pmatrix}$$

```
A = np.array([ [1,2], [1,3]] )
B = np.array([ [4,1], [5,3]] )

S = A + B
```

Opérations sur les matrices

produit matriciel :

$$\mathbf{A} = \begin{bmatrix} 2 & 3 \\ 1 & -1 \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} 9 \\ 2 \end{bmatrix}$$

```
A = np.array([ [2,3], [1,-1]] )
B = np.array([ [9], [2]] )

P = np.dot(A , B)
```

Opérations sur les matrices

Déterminant d'une matrice :

```
A = np.array([ [2,3], [1,-1]] )

P = np.linalg.det(A)
```

QCM

QCM

Le module numpy

QCM

Quel est l'objectif principal de la bibliothèque NumPy ?

a) Créer des interfaces utilisateur graphiques

b) Gérer les opérations mathématiques sur les tableaux multidimensionnels

c) Effectuer des manipulations de chaînes de caractères

d) Analyser des bases de données

Le module numpy

QCM

Comment créer un tableau NumPy à partir d'une liste Python ?

a) np.array()
b) np.list()
c) np.makeArray()
d) np.create()

Le module numpy

QCM

Quelle commande permet de créer un tableau NumPy de zéros de forme (3, 4) ?

a) np.zeros((3, 4))
b) np.ones((3, 4))
c) np.empty((3, 4))
d) np.zeros(3, 4)

Le module numpy

QCM

Quelle fonction est utilisée pour calculer la moyenne d'un tableau NumPy ?

a) np.mean()
b) np.average()
c) np.sum()
d) np.median()

Le module numpy

QCM

Quel est le résultat de l'exécution de cette commande : np.array([1, 2, 3]) * 2 ?

a) [1, 4, 9]
b) [2, 4, 6]
c) [2, 3, 4]
d) [1, 2, 3, 2]

Le module numpy

QCM

Quelle fonction NumPy permet de calculer le produit scalaire de deux vecteurs ?

a) np.dot()
b) np.prod()
c) np.multiply()
d) np.inner()

Le module numpy

QCM

Que fait le code suivant ?

```
import numpy as np
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
c = a + b
```

a) Il crée deux tableaux et les additionne élément par élément.
b) Il crée deux matrices et les additionne de manière matricielle.
c) Il effectue une multiplication élément par élément.
d) Il crée un tableau de a et ajoute un scalaire b.

Le module numpy

QCM

Que fait le code suivant ?

```
import numpy as np
a = np.zeros((3, 3))
b = np.ones((3, 3))
c = a + b
```

a) Il crée un tableau de zéros, un tableau de uns, et additionne les deux.
b) Il crée un tableau de zéros et un tableau de nans, puis les additionne.
c) Il crée un tableau de zéros et un tableau de matrices, puis les additionne.
d) Il multiplie chaque élément de a par b.

Le module numpy

QCM

Que fait le code suivant ?

```
import numpy as np
a = np.array([1, 2, 3, 4])
b = a * 2
```

a) Il effectue une multiplication matricielle entre a et 2.
b) Il multiplie chaque élément de a par 2.
c) Il additionne chaque élément de a à 2.
d) Il crée une matrice de 2x2 à partir de a.