

# Знайомство з NumPy

23 вересня 2023 р.

## Імпорт модулів

```
In [1]: import numpy as np
```

1. Створіть одновимірний масив (вектор) з першими 10-ма натуральними числами та виведіть його значення.

Натуральні числа — це числа, що виникають природним чином при лічбі. Це числа: 1, 2, 3, 4, ... Множину натуральних чисел прийнято позначати  $\mathbb{N}$ .

```
In [2]: vector = np.arange(1, 11)
print(vector)
```

```
Out[2]: [ 1  2  3  4  5  6  7  8  9 10]
```

2. Створіть двовимірний масив (матрицю) розміром 3x3, заповніть його нулями та виведіть його значення.

```
In [3]: matrix = np.zeros((3, 3))
print(matrix)
```

```
Out[3]: [[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
```

3. Створіть масив розміром 5x5, заповніть його випадковими цілими числами в діапазоні від 1 до 10 та виведіть його значення

```
In [4]: random_matrix = np.random.randint(1, 11, (5, 5))
print(random_matrix)
```

```
Out[4]: [[ 7 10  4  6  7]
 [ 1 10  1  1 10]
 [10  9 10  6  3]
 [10  7  9  6  6]
 [ 9  2  4  4  4]]
```

4. Створіть масив розміром 4x4, заповніть його випадковими дійсними числами в діапазоні від 0 до 1 та виведіть його значення.

```
In [5]: random_matrix = np.random.rand(4, 4)
print(random_matrix)
```

```
Out[5]: [[0.34716231 0.38666569 0.60847232 0.44947206]
 [0.28441679 0.24889235 0.69068513 0.44390396]
 [0.45757561 0.30179135 0.30334876 0.14319334]
 [0.19297175 0.18552924 0.3236188 0.56707376]]
```

5. Створіть два одновимірних масиви розміром 5, заповніть випадковими цілими числами в діапазоні від 1 до 10 та виконайте на них поелементні операції додавання, віднімання та множення.

#### 5.1. Створення масивів

```
In [6]: array1 = np.random.randint(1, 11, 5)
array2 = np.random.randint(1, 11, 5)

print("Масив 1:", array1)
print("Масив 2:", array2)
```

```
Out[6]: Масив 1: [1 5 1 9 9]
Масив 2: [8 2 4 6 7]
```

#### 5.2. Поелементні операції

```
In [7]: addition_result = array1 + array2
print("Результат додавання:", addition_result)

# Виконаємо поелементну операцію віднімання
subtraction_result = array1 - array2
print("Результат віднімання:", subtraction_result)

# Виконаємо поелементну операцію множення
multiplication_result = array1 * array2
print("Результат множення:", multiplication_result)
```

```
Out[7]: Результат додавання: [ 9  7  5 15 16]
Результат віднімання: [-7  3 -3  3  2]
Результат множення: [ 8 10  4 54 63]
```

6. Створіть два вектори розміром 7, заповніть довільними числами та знайдіть їх скалярний добуток

Скалярний добуток (scalar or dot product) можна обчислити за наступною формулою

$$\vec{v} \cdot \vec{u} = v^1 u_1 + v^2 u_2 + v^3 u_3 + \dots + v^n u_n$$

```
In [8]: vector1 = np.random.random(7)
vector2 = np.random.random(7)

scalar_product = np.dot(vector1, vector2)

print("Скалярний добуток векторів:", scalar_product)
```

Out[8]: Скалярний добуток векторів: 1.8680632685208063

**7. Створіть дві матриці розміром 2x2 та 2x3, заповніть їх випадковими цілими числами в діапазоні від 1 до 10 та перемножте їх між собою.**

```
In [9]: matrix1 = np.random.randint(1, 11, (2, 2))
matrix2 = np.random.randint(1, 11, (2, 3))

print("Матриця 1:")
print(matrix1)
print("Матриця 2:")
print(matrix2)

result = np.dot(matrix1, matrix2)

print("Результат множення:")
print(result)
```

Out[9]: Матриця 1:  
[[ 8 6]  
 [10 4]]  
Матриця 2:  
[[ 7 10 4]  
 [ 6 10 7]]  
Результат множення:  
[[ 92 140 74]  
 [ 94 140 68]]

**8. Створіть матрицю розміром 3x3, заповніть її випадковими цілими числами в діапазоні від 1 до 10 та знайдіть її обернену матрицю.**

Обчислення оберненої матриці здійснюється за формулою:

$$A^{-1} = \frac{1}{\det(A)} \cdot \text{adj}(A),$$

де  $\det(A)$  — визначник матриці  $A$ ,  $\text{adj}(A)$  — доповнена матриця.

Обернена матриця існує, якщо  $\det(A) \neq 0$ . Матриця, для якої  $\det(A) = 0$  називається сингулярною, і для такої матриці оберненої не існує. Тому в коді ми маємо це перевірити.

```
In [10]: # Створимо матрицю розміром 3x3 з випадковими цілими числами від 1 до 10
matrix = np.random.randint(1, 11, (3, 3))

# Виведемо початкову матрицю
```

```
In [10]: print("Початкова матриця:")
print(matrix)

# Знайдемо обернену матрицю
try:
    inverse_matrix = np.linalg.inv(matrix)
    print("Обернена матриця:")
    print(inverse_matrix)
except np.linalg.LinAlgError:
    print("Матриця не має оберненої матриці.")
```

```
Out[10]: Початкова матриця:
[[ 2  4  6]
 [ 7  1  1]
 [ 5  1 10]]
Обернена матриця:
[[-0.03913043  0.14782609  0.00869565]
 [ 0.2826087  0.04347826 -0.17391304]
 [-0.00869565 -0.07826087  0.11304348]]
```

## 9. Створіть матрицю розміром 4x4, заповніть її випадковими дійсними числами в діапазоні від 0 до 1 та транспонуйте її

Транспонована матриця  $A^T$ , виникає з матриці  $A$  в результаті заміни рядків на стовпчики  $(A^T)_{ji} = A_{ij}$ :

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

Тоді транспонована матриця  $A^T$  буде мати вигляд:

$$A^T = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

```
In [11]: matrix = np.random.rand(4, 4)

# Виведемо початкову матрицю
print("Початкова матриця:")
print(matrix)

# Транспонуємо матрицю
transposed_matrix = np.transpose(matrix)

# Виведемо транспоновану матрицю
print("Транспонована матриця:")
print(transposed_matrix)
```

```
Out[11]: Початкова матриця:
[[0.08402274 0.07052606 0.94668645 0.47278038]
 [0.65529227 0.45497744 0.38751573 0.46614788]
 [0.57033985 0.4134212  0.46341259 0.69720946]
 [0.15384283 0.5854396  0.90123728 0.15706181]]
```

```
Out[11]: Транспонована матриця:
[[0.08402274 0.65529227 0.57033985 0.15384283]
 [0.07052606 0.45497744 0.4134212 0.5854396 ]
 [0.94668645 0.38751573 0.46341259 0.90123728]
 [0.47278038 0.46614788 0.69720946 0.15706181]]
```

10. Створіть матрицю розміром 3x4 та вектор розміром 4, заповніть їх випадковими цілими числами в діапазоні від 1 до 10 та перемножте матрицю на вектор.

$$\mathbf{u} = \mathbf{A} \cdot \mathbf{v} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} a_{11} \cdot v_1 + a_{12} \cdot v_2 + a_{13} \cdot v_3 \\ a_{21} \cdot v_1 + a_{22} \cdot v_2 + a_{23} \cdot v_3 \\ a_{31} \cdot v_1 + a_{32} \cdot v_2 + a_{33} \cdot v_3 \end{bmatrix}$$

```
In [12]: # Створимо матрицю розміром 3x4 з випадковими цілими числами від 1 до 10
matrix = np.random.randint(1, 11, (3, 4))

# Створимо вектор розміром 4 з випадковими цілими числами від 1 до 10
vector = np.random.randint(1, 11, 4)

# Виведемо матрицю і вектор
print("Матриця:")
print(matrix)
print("Вектор:")
print(vector)

# Перемножимо матрицю на вектор
result = np.dot(matrix, vector)

# Виведемо результат
print("Результат перемноження:")
print(result)
```

```
Out[12]: Матриця:
[[5 3 6 2]
 [7 2 7 7]
 [5 9 5 7]]
Вектор:
[ 8  3 10  9]
Результат перемноження:
[127 195 180]
```

11. Створіть матрицю розміром 2x3 та вектор розміром 3, заповніть їх випадковими дійсними числами в діапазоні від 0 до 1 та перемножте матрицю на вектор.

```
In [13]: # Створимо матрицю розміром 2x3 з випадковими дійсними числами від 0 до 1
matrix = np.random.rand(2, 3)

# Створимо вектор розміром 3 з випадковими дійсними числами від 0 до 1
vector = np.random.rand(3)
```

```
In [13]: # Виведемо матрицю і вектор
print("Матриця:")
print(matrix)
print("Вектор:")
print(vector)

# Перемножимо матрицю на вектор
result = np.dot(matrix, vector)

# Виведемо результат
print("Результат перемноження:")
print(result)
```

```
Out[13]: Матриця:
[[0.40142629 0.24681107 0.41057536]
 [0.72516117 0.91335118 0.68146542]]
Вектор:
[0.3660769 0.20024095 0.43328539]
Результат перемноження:
[0.37427088 0.74362408]
```

## 12. Створіть дві матриці розміром 2x2, заповніть їх випадковими цілими числами в діапазоні від 1 до 10 та виконайте їхнє поелементне множення.

Поелементне множення двох матриць визначається так: результатом цієї операції є нова матриця, у якій кожний елемент результуючої матриці визначається, як добуток відповідних елементів вихідних матриць.

Елемент результуючої матриці  $C$  обчислюється як добуток відповідних елементів матриць  $A$  і  $B$ :

$$C_{ij} = A_{ij} \cdot B_{ij}$$

де  $C_{ij}$  — елемент результуючої матриці  $C$  на позиції  $(i, j)$ ,  $A_{ij}$  — елемент матриці  $A$  на позиції  $(i, j)$  і  $B_{ij}$  - елемент матриці  $B$  на позиції  $(i, j)$ .

Поелементне множення відрізняється від стандартного множення матриць, оскільки в останньому випадку обчислюється добуток рядків однієї матриці на стовпці іншої матриці, що веде до іншої результуючої матриці.

```
In [14]: # Створимо першу матрицю 2x2 з випадковими цілими числами в діапазоні від 1 до 10
matrix1 = np.random.randint(1, 11, (2, 2))

# Створимо другу матрицю 2x2 з випадковими цілими числами в діапазоні від 1 до 10
matrix2 = np.random.randint(1, 11, (2, 2))

# Виведемо обидві матриці
print("Перша матриця:")
print(matrix1)
print("Друга матриця:")
print(matrix2)

# Виконаємо поелементне множення матриць
result = np.multiply(matrix1, matrix2)
```

```
In [14]: # Виведемо результат
print("Результат поелементного множення:")
print(result)
```

```
Out[14]: Перша матриця:
[[5 7]
 [1 2]]
Друга матриця:
[[ 7 10]
 [ 3  9]]
Результат поелементного множення:
[[35 70]
 [ 3 18]]
```

**13. Створіть дві матриці розміром 2x2, заповніть їх випадковими цілими числами в діапазоні від 1 до 10 та знайдіть їх добуток.**

Добуток матриць визначається як:

$$C_{ij} = \sum_{r=1}^m A_{ir} B_{rj} \quad (i = 1, 2, \dots, l; j = 1, 2, \dots, n)$$

```
In [15]: # Створимо першу матрицю 2x2 з випадковими цілими числами в діапазоні від 1 до 10
matrix1 = np.random.randint(1, 11, (2, 2))

# Створимо другу матрицю 2x2 з випадковими цілими числами в діапазоні від 1 до 10
matrix2 = np.random.randint(1, 11, (2, 2))

# Виведемо обидві матриці
print("Перша матриця:")
print(matrix1)
print("Друга матриця:")
print(matrix2)

# Знайдемо добуток матриць
result = np.dot(matrix1, matrix2)

# Виведемо результат
print("Результат добутку матриць:")
print(result)
```

```
Out[15]: Перша матриця:
[[1 3]
 [2 5]]
Друга матриця:
[[7 7]
 [2 3]]
Результат добутку матриць:
[[13 16]
 [24 29]]
```

14. Створіть матрицю розміром  $5 \times 5$ , заповніть її випадковими цілими числами в діапазоні від 1 до 100 та знайдіть суму елементів матриці.

```
In [16]: # Створимо матрицю 5x5 з випадковими цілими числами від 1 до 100
matrix = np.random.randint(1, 101, (5, 5))

# Виведемо матрицю
print("Матриця:")
print(matrix)

# Знайдемо суму елементів матриці
sum_of_elements = np.sum(matrix)

# Виведемо суму
print("Сума елементів матриці: ", sum_of_elements)
```

```
Матриця:
[[ 65  54  46 100  89]
 [ 19  41  79  28  70]
 [ 73  27  94  20  94]
 [ 46  35  18  68  27]
 [ 36  63  33  76   7]]
Сума елементів матриці: 1308
```

15. Створіть дві матриці розміром  $4 \times 4$ , заповніть їх випадковими цілими числами в діапазоні від 1 до 10 та знайдіть їхню різницю.

Віднімання матриць  $A - B$  - це операція обчислення матриці  $C$ , усі елементи якої дорівнюють попарній різниці всіх відповідних елементів матриць  $A$  і  $B$ , тобто фактично це поелементне віднімання.

```
In [17]: # Створимо першу матрицю 4x4 з випадковими цілими числами від 1 до 10
matrix1 = np.random.randint(1, 11, (4, 4))

# Створимо другу матрицю 4x4 з випадковими цілими числами від 1 до 10
matrix2 = np.random.randint(1, 11, (4, 4))

# Виведемо обидві матриці
print("Перша матриця:")
print(matrix1)
print("Друга матриця:")
print(matrix2)

# Знайдемо різницю матриць
result = matrix1 - matrix2

# Виведемо результат
print("Різниця матриць:")
print(result)
```

```
Out[17]: Перша матриця:
[[ 6  4 10  4]
 [ 6  3  7 10]
```



```
Out[17]:  [ 5  9  6  4]
          [ 5  9 10  5]]
Друга матриця:
[[6 6 4 2]
 [6 1 2 8]
 [2 9 1 5]
 [5 3 5 5]]
Різниця матриць:
[[ 0 -2  6  2]
 [ 0  2  5  2]
 [ 3  0  5 -1]
 [ 0  6  5  0]]
```

16. Створіть матрицю розміром 3x3, заповніть її випадковими дійсними числами в діапазоні від 0 до 1 та знайдіть вектор-стовпчик, що містить суму елементів кожного рядка матриці.

```
In [18]: # Створимо матрицю 3x3 з випадковими дійсними числами в діапазоні від 0 до 1
matrix = np.random.rand(3, 3)

# Виведемо матрицю
print("Матриця:")
print(matrix)

# Знайдемо суму елементів кожного рядка матриці та створимо вектор-стовпчик
sum_of_rows = np.sum(matrix, axis=1, keepdims=True)

# Виведемо вектор-стовпчик із сумами
print("Вектор-стовпчик із сумами рядків:")
print(sum_of_rows)
```

```
Out[18]: Матриця:
[[0.01297979 0.47161382 0.59052176]
 [0.55785916 0.35450255 0.41946586]
 [0.67527101 0.41623059 0.6897199 ]]
Вектор-стовпчик із сумами рядків:
[[1.07511538]
 [1.33182756]
 [1.78122151]]
```

17. Створіть матрицю розміром 3x4 з довільними цілими числами і створіть матрицю з квадратами цих чисел.

```
In [19]: # Створимо матрицю 3x4 з довільними цілими числами (від -10 до 10)
matrix = np.random.randint(-10, 11, (3, 4))

# Виведемо початкову матрицю
print("Початкова матриця:")
print(matrix)

# Знайдемо матрицю з квадратами чисел
squared_matrix = np.square(matrix)
```

```
In [19]: # Виведемо матрицю з квадратами чисел
print("Матриця з квадратами чисел:")
print(squared_matrix)
```

```
Out[19]: Початкова матриця:
[[-7 -9 -5  3]
 [-2 -8 -6 10]
 [-1  2 -6  9]]
Матриця з квадратами чисел:
[[ 49  81  25   9]
 [  4  64  36 100]
 [  1   4  36  81]]
```

18. Створіть вектор розміром 4, заповніть його випадковими цілими числами в діапазоні від 1 до 50 та знайдіть вектор з квадратними коренями цих чисел

```
In [20]: # Створимо вектор розміром 4 з випадковими цілими числами від 1 до 50
vector = np.random.randint(1, 51, 4)

# Виведемо початковий вектор
print("Початковий вектор:", vector)

# Знайдемо вектор з квадратними коренями чисел
sqrt_vector = np.sqrt(vector)

# Виведемо вектор з квадратними коренями
print("Вектор з квадратними коренями чисел:", sqrt_vector)
```

```
Out[20]: Початковий вектор: [49 12 50 11]
Вектор з квадратними коренями чисел: [7.  3.46410162  7.07106781  3.31662479]
```