**Модуль 1.2: Елементи лінійної алгебри**

**Елементи лінійної алгебри**

Наступним етапом навчання є ознайомлення з основними поняттями [лінійної алгебри](https://uk.wikipedia.org/wiki/%D0%9B%D1%96%D0%BD%D1%96%D0%B9%D0%BD%D0%B0_%D0%B0%D0%BB%D0%B3%D0%B5%D0%B1%D1%80%D0%B0) (https://uk.wikipedia.org/wiki/Лінійна\_алгебра). Це розділ математики, який вивчає лінійні простори, лінійні перетворення і системи лінійних рівнянь. Вона використовується для розв'язування багатьох проблем, зокрема в науці про обробку сигналів, комп'ютерній графіці, машинному навчанні та багатьох інших галузях науки та техніки.

Основні поняття в лінійній алгебрі включають в себе вектори, матриці, лінійні перетворення, векторні простори, лінійні системи рівнянь, власні значення та вектори, діагоналізацію матриць та багато іншого.

Відразу для закріплення теорії на практиці, ми познайомимось з бібліотекою NumPy(Numeric Python), котра використовується під час роботи з даними, машинним навчанням та наукових досліджень та суттєво полегшує роботу з векторами та матрицями.

Пакети SciPy, Pandas та Tensorflow використовують NumPy як основний елемент їхньої інфраструктури. Тому це є базою, фундаментом, основною для роботи фахівця у сфері Data Science.

**Встановлення бібліотеки NumPy**[​](https://textbook.edu.goit.global/python/data-science-remaster/v1/docs/module-01/main_2" \l "%D0%B2%D1%81%D1%82%D0%B0%D0%BD%D0%BE%D0%B2%D0%BB%D0%B5%D0%BD%D0%BD%D1%8F-%D0%B1%D1%96%D0%B1%D0%BB%D1%96%D0%BE%D1%82%D0%B5%D0%BA%D0%B8-numpy" \t "_blank)

Якщо в системі встановлений Python, ви можете встановити NumPy за допомогою:

pip install numpy

Інший варіант це використання [Anaconda](https://www.anaconda.com/" \t "_blank). Про неї ми згадували в кінці попередньої лекції.

Щоб імпортувати NumPy та отримати доступ до його функцій, імпортуйте його до свого коду наступним чином:

import numpy as np

Ми скорочуємо імпортоване ім'я до np для кращого читання коду за допомогою NumPy. Це широко розповсюджене скорочення, так би мовити домовленість, якій необхідно слідувати, щоб будь-хто, хто працює з нашим кодом, зміг легко його зрозуміти.

**Вектори**

[​](https://textbook.edu.goit.global/python/data-science-remaster/v1/docs/module-01/main_2#%D0%B2%D0%B5%D0%BA%D1%82%D0%BE%D1%80%D0%B8)Ми вказували вище, що основними об'єктами у лінійній алгебрі є вектори та матриці. Тому давайте згадаємо що таке вектор. Як ми пам'ятаємо зі школи, це напрямлений відрізок, та він має якісь координати, наприклад (*x*,*y*). Якщо у нас декартова площина, то координати вектора будуть його кінцем, а початком буде точка (0,0).

А отже ми можемо будь-яку точку нашої координатної площини співвіднести вектору, котрий будемо записувати у вигляді стовпця:

=

А сукупність всіх можливих таких векторів утворить **векторний**, або ж **лінійний** простір R2. Тобто елементами даного лінійного простостору будуть наші вектори, і ми розуміємо що вони співвідносяться точкам на координатній площині. Надалі ми будемо працювати з векторами більших розмірностей, не лише у двовимірній площині. Тому для нас вектор буде просто стовпцем з

набором значень, що відповідають його координатам, наприклад:

=

де  — це вектор, що містить п'ять елементів 4,5,2,4,6

4,5,2,4,6, а тому належить лінійному простору R5.

А вектор-рядок часто позначають так:

=(4 5 2 4 6)*x*=(4 5 2 4 6​)

де — це вектор-рядок, що також містить п'ять елементів 4,5,2,4,6, проте належить лінійному простору R1×5. Згодом ми зрозуміємо чому таке позначення. Надалі коли ми говоримо про вектори, то уявляємо його як просто набір з *n* дійсних чисел, котрий належить векторному простору R*n*, і є стовпцем.

Таким вектором у бібліотеці numpy є об'єкт ndarray — багатовимірний масив даних одного типу. І для нього будуть виконуватись всі правила, як і для векторів у лінійній алгебрі. Оскільки все-таки це масив, то з ним можемо здійснювати й додаткові операції, як сортування, про які ми поговоримо пізніше в лекції.

Основні операції, котрі ми можемо робити з векторами:

* додавання/віднімання скаляра та вектора
* множити вектор на скаляр (число)
* додавати два вектори
* віднімати два вектори
* шукати скалярний добуток двох векторів

**Додавання/віднімання скаляра та вектора**

[​](https://textbook.edu.goit.global/python/data-science-remaster/v1/docs/module-01/main_2#%D0%B4%D0%BE%D0%B4%D0%B0%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F%D0%B2%D1%96%D0%B4%D0%BD%D1%96%D0%BC%D0%B0%D0%BD%D0%BD%D1%8F-%D1%81%D0%BA%D0%B0%D0%BB%D1%8F%D1%80%D0%B0-%D1%82%D0%B0-%D0%B2%D0%B5%D0%BA%D1%82%D0%BE%D1%80%D0%B0)Якщо ми будемо до вектора додавати звичайне число, це рівносильно що до кожного елемента нашого вектора ми будемо додавати даний скаляр.

Аналогічно і з відніманням. Переконаємось у цьому на прикладі:

import numpy as np

a = np.array([1, 2, 3, 4, 5], dtype=float)

print(a + 2) # [3. 4. 5. 6. 7.]

print(a - 1) # [0. 1. 2. 3. 4.]

А нові вектори, які ми отримали в результаті, також будуть належати векторному простору, що й вектор , тобто лінійному простору .

**Множення на скаляр**

[​](https://textbook.edu.goit.global/python/data-science-remaster/v1/docs/module-01/main_2#%D0%BC%D0%BD%D0%BE%D0%B6%D0%B5%D0%BD%D0%BD%D1%8F-%D0%BD%D0%B0-%D1%81%D0%BA%D0%B0%D0%BB%D1%8F%D1%80)Коли ми множимо на вектор на скаляр, то отримуємо вектор, який належить цьому ж лінійному простору, що й початковий вектор. Перепишемо це ж використанням математичних позначень і кантора приналежності ∈:

*λ* ∗ ∈

А результатом буде те, що кожен елемент вектора, тобто координати, будуть помножені на число *λ*. Перевіримо це за домопогою масиву ndarray. Спочатку створимо його:

import numpy as np

a = np.array([1, 2, 3, 4, 5], dtype=float)

print(a) # [1. 2. 3. 4. 5.]

а тепер помножимо на 5 наш вектор :

b = 5\*a

print(b) *# [ 5. 10. 15. 20. 25.]*

**Додавання та віднімання двох векторів**

[​](https://textbook.edu.goit.global/python/data-science-remaster/v1/docs/module-01/main_2#%D0%B4%D0%BE%D0%B4%D0%B0%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F-%D1%82%D0%B0-%D0%B2%D1%96%D0%B4%D0%BD%D1%96%D0%BC%D0%B0%D0%BD%D0%BD%D1%8F-%D0%B4%D0%B2%D0%BE%D1%85-%D0%B2%D0%B5%D0%BA%D1%82%D0%BE%D1%80%D1%96%D0%B2)Дана операція можлива лише у тому випадку, якщо наші вектори однієї розмірності — тобто у них однакова к-сть елементів. Ми легко можемо дізнатись розмір нашого вектора за допомогою такої команди size:

import numpy as np

a = np.array([1, 2, 3, 4, 5], dtype=float)

print(a.size) # 5

А коли ми додамо, чи віднімемо між собою два вектори, то у нас попарно віднімуться чи просумуються відповідні елементи наших векторів:

import numpy as np

a = np.array([1, 2, 3])

b = np.array([4, 5, 6])

print(a + b) # [5 7 9]

print(a - b) # [-3 -3 -3]

**Скалярний добуток**

[​](https://textbook.edu.goit.global/python/data-science-remaster/v1/docs/module-01/main_2#%D1%81%D0%BA%D0%B0%D0%BB%D1%8F%D1%80%D0%BD%D0%B8%D0%B9-%D0%B4%D0%BE%D0%B1%D1%83%D1%82%D0%BE%D0%BA)Скалярним добутком двох векторів буде сума попарно перемножених відповідних координат цих двох векторів. Для прикладу обчислимо скалярний добуток двох тривимірних векторів **x** і **y** в явному вигляді. Так, часом вектори позначають в літературі просто як жирні друковані літери, і таке позначення зустрічається на рівні зі стрілочкою:

X ⋅ Y=x1y1+ x2y2+ x3y3​

У цьому виразі **x** і **y** - це вектори, а ​ x1, x2, x3​ і y1, y2, y3 ​- їх координати відповідно. Крапка позначає скалярний добуток.

Перевіримо як це працюватиме у нашому інтерпретаторі:

import numpy as np

a = np.array([1, 2, 3])

b = np.array([4, 5, 6])

print(a \* b) # [ 4 10 18]

У даному випадку ми отримали не той результат, на який розраховували, оскільки ми виконали поелементне множення відповідних елементів і отримали новий вектор, а не скалярний добуток. А це тому, що оператор ∗

відповідає поелементному множенню для двох векторів, чи у нашому випадку масивів ndarray, а щоб виконати скалярний добуток ми повинні виконати наступний скрипт:

import numpy as np

a = np.array([1, 2, 3])

b = np.array([4, 5, 6])

print(np.dot(a, b)) # 32

Ось тепер ми справді виконали скалярний добуток і отримали очікуваний результат.

До речі, для масивів ndarray можна виконувати операцію поелементного ділення також:

import numpy as np

a = np.array([1, 2, 3])

b = np.array([4, 5, 6])

print(b / a) # [4. 2.5 2. ]

**Базис векторів**

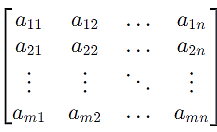
[​](https://textbook.edu.goit.global/python/data-science-remaster/v1/docs/module-01/main_2#%D0%B1%D0%B0%D0%B7%D0%B8%D1%81-%D0%B2%D0%B5%D0%BA%D1%82%D0%BE%D1%80%D1%96%D0%B2)У лінійній алгебрі базис векторів — це набір векторів, що використовуються для опису інших векторів у лінійному просторі. Кожен вектор в просторі можна представити унікальною лінійною комбінацією базисних векторів з деякими коефіцієнтами.

Наприклад, в двовимірному просторі, вектори (1,0)(1,0) та (0,1)(0,1) можуть служити базисом. Будь-який інший вектор у цьому просторі можна представити як [лінійну комбінацію](https://uk.wikipedia.org/wiki/%D0%9B%D1%96%D0%BD%D1%96%D0%B9%D0%BD%D0%B0_%D0%BA%D0%BE%D0%BC%D0%B1%D1%96%D0%BD%D0%B0%D1%86%D1%96%D1%8F#:~:text=%D0%9B%D1%96%D0%BD%D1%96%D0%B9%D0%BD%D0%B0%20%D0%BA%D0%BE%D0%BC%D0%B1%D1%96%D0%BD%D0%B0%D1%86%D1%96%D1%8F%20%E2%80%94%20%D0%B2%D0%B8%D1%80%D0%B0%D0%B7%2C%20%D0%BF%D0%BE%D0%B1%D1%83%D0%B4%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B9%20%D0%BD%D0%B0,%D1%96%20%CE%B2%2D%20%D0%BA%D0%BE%D0%B5%D1%84%D1%96%D1%86%D1%96%D1%94%D0%BD%D1%82%D0%B8) (https://uk.wikipedia.org/wiki/Лінійна\_комбінація) цих базисних векторів зі своїми власними коефіцієнтами. Наприклад, вектор (3,4)(3,4) можна представити як 3∗(1,0)+4∗(0,1)3∗(1,0)+4∗(0,1).

Важливо зазначити, що для одного лінійного простору, ми можемо створити різні набори базисних векторів. У деяких випадках можуть бути використані інші набори векторів, що утворюють базис, залежно від потреби.

**Матриці**

Матрицею розмірності *m*×*n* називатиметься сукупність чисел, які будуть розміщені у таблиці з *m* рядочків та *n* стовпчиків:



Числа *aij*​ називаються елементами матриці. Таким чином, перший індекс *i* елементу відповідає за номер рядка, у якому розміщений даний елемент, а другий індекс *j* — номер стовця. І як ви могли вже здогадатись, така матриця буде належати лінійному простору R*m*×*n*. Тепер ми розуміємо, чому вектор-рядок та вектор-стовпець належать різним лінійним просторам, і коли ми говоримо про вектори-стовпці, для вектора довжиною *n* він належатиме простору R*n*×1 , де ми опускаємо 11 і просто пишемо R*n*.

Оскільки ndarray є масивом, то створивши масив масивів, ми отримаємо матрицю:

m = np.array([[1, 2, 3], [4, 5, 6]], dtype=int)

print(m)

Результат роботи програми:

[[1 2 3]

[4 5 6]]

Розмірність матриці, як і векторів, можна подивитись за допомогою даного атрибуту shape

import numpy as np

a = np.array([1, 2, 3, 4, 5], dtype=float)

print(a.shape) # (5,)

m = np.array([[1, 2, 3], [4, 5, 6]], dtype=int)

print(m.shape) # (2, 3)

Як бачимо, це кортеж. Для вектора це кількість координат вектора, а для матриці перше значення кортежу — кількість рядків, а друге — кількість стовпців матриці. І як ми бачимо, розмірність вектора має вигляд (5,5,), так наче це таблиця з 55-ма рядочками, ну і без стовпців, бо все-таки це вектор, а не матриця. Проте як ми говорили раніше, зазвичай коли ми говоримо про вектор, то маємо на увазі вектор-стовпець, якщо не вказано протилежного.

Операції з матрицями будуть відбуватися таким самим чином, як і з вектором, оскільки це у нас масив масивів, окрім множення. Якщо ми будемо використовувати оператор ∗, то отримаємо поелементне множення всіх відповідних елементів двох матриць і можливо провести таку операцію лише для двох матриць однієї розмірності. Розглянемо це на прикладі:

a = np.array([[2, 3, 4, 5],

[1, 2, 3, 4]])

b = np.array([[2, 2, 2, 2],

[1, 1, 1, 1]])

print(a\*b)

Отримаємо такий результат:

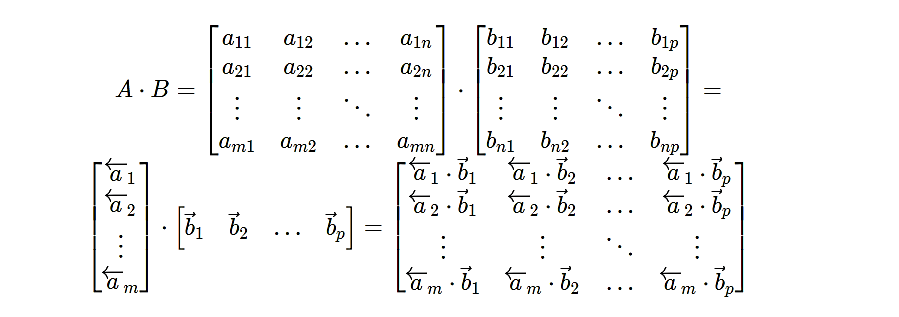
[[ 4 6 8 10]

[ 1 2 3 4]]

Проте з точки зору лінійної алгебри множення двох матриць є дещо іншою операцією.

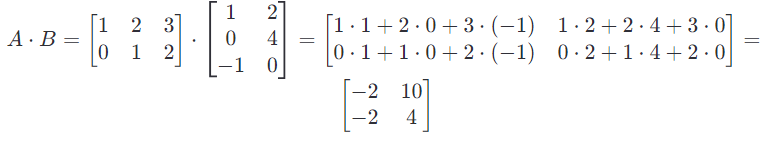
**Множення матриць**

[​](https://textbook.edu.goit.global/python/data-science-remaster/v1/docs/module-01/main_2#%D0%BC%D0%BD%D0%BE%D0%B6%D0%B5%D0%BD%D0%BD%D1%8F-%D0%BC%D0%B0%D1%82%D1%80%D0%B8%D1%86%D1%8C)Множення матриць — це алгебраїчна операція, яка використовується для обчислення нової матриці шляхом множення двох вихідних матриць. Якщо ми маємо дві матриці *A* та *B*, де *A* має розмірність *m*×*n* (тобто має *m* рядків і *n* стовпців), а *B* має розмірність *n*×*p* (має *n* рядків і *p* стовпців), то їх добуток *AB* буде матрицею *C* розмірності *m*×*p*, де кожен елемент нової матриці *cij*​ обчислюється як скалярний добуток *i*-ого рядка матриці *A*, тобто *i*​​, та *j*-ого стовпця матриці *B*​ — *j*​. Звідки ми можемо зрозуміти, що дану операцію можна провести лише у випадку, якщо рядки матриці *A* повинні мати такий же розмір, як і стовпці матриці *B*, тобто к-сть стовпців матриці *A* повинна бути рівна к-сті рядків матриці *B*.

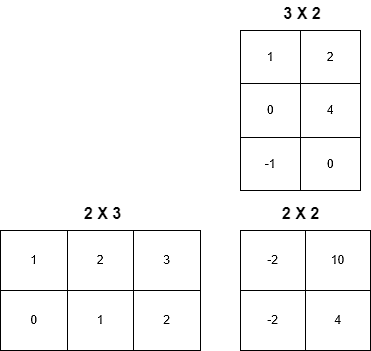


​

І давайте розглянемо конкретний приклад множення двох матриць:



Щоб запам'ятати це правило, використовуйте наступну діаграму множення, представлену на малюнку нижче.



**Схема множення матриць**

Проведемо множення матриць мовою Python. Для цього використаємо знайому нам функцію np.dot з бібліотеки Numpy.

import numpy as np

a = np.array([[1, 2, 3],

[0, 1, 2]])

b = np.array([[ 1, 2],

[ 0, 4],

[-1, 0]])

c = np.dot(a, b)

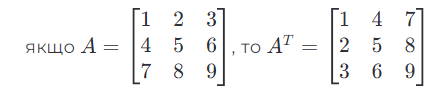
print(c)

[[-2 10]

[-2 4]]

**Транспонування та зміна форми матриць в numpy**

[​](https://textbook.edu.goit.global/python/data-science-remaster/v1/docs/module-01/main_2#%D1%82%D1%80%D0%B0%D0%BD%D1%81%D0%BF%D0%BE%D0%BD%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F-%D1%82%D0%B0-%D0%B7%D0%BC%D1%96%D0%BD%D0%B0-%D1%84%D0%BE%D1%80%D0%BC%D0%B8-%D0%BC%D0%B0%D1%82%D1%80%D0%B8%D1%86%D1%8C-%D0%B2-numpy)Нагадаємо, що процес транспонування матриці — це заміна рядків матриці на її стовпці, а стовпців, відповідно, на рядки. Отримана в результаті матриця називається транспонованою. Символ операції транспонування — літера *T*



import numpy as np

a = np.array([[1, 2, 3],

[4, 5, 6],

[7, 8, 9]])

print(a)

b = a.T

print(b)

[[1 2 3]

[4 5 6]

[7 8 9]]

[[1 4 7]

[2 5 8]

[3 6 9]]

Розмірність вектора або матриці можна також змінювати методом reshape

import numpy as np

a = np.array.([[1, 2, 3],

[4, 5, 6]])

b = a.reshape((3, 2))

print(b)

Ми перетворили матрицю розмірності 2×32×3 на розмірність 3×23×2

[[1 2]

[3 4]

[5 6]]

Також є можливість перетворення матриці в плоский вектор за допомогою методу flatten

import numpy as np

a = np.array([[1, 2, 3], [4, 5, 6]])

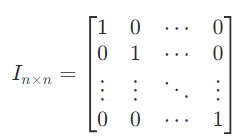
b = a.flatten()

print(b)

[[1 2 3 4 5 6]]

**Одинична матриця**

[​](https://textbook.edu.goit.global/python/data-science-remaster/v1/docs/module-01/main_2#%D0%BE%D0%B4%D0%B8%D0%BD%D0%B8%D1%87%D0%BD%D0%B0-%D0%BC%D0%B0%D1%82%D1%80%D0%B8%D1%86%D1%8F)[Одинична матриця](https://uk.wikipedia.org/wiki/%D0%9E%D0%B4%D0%B8%D0%BD%D0%B8%D1%87%D0%BD%D0%B0_%D0%BC%D0%B0%D1%82%D1%80%D0%B8%D1%86%D1%8F) (<https://uk.wikipedia.org/wiki/Одинична_матриця>) —  це квадратна матриця, у якої на головній діагоналі стоять одиниці, а всі інші елементи дорівнюють нулю. Одиничну матрицю позначають як *I*.



​Одинична матриця виконує роль одиниці для матричного множення:

*Im*​*A*=*AIn*​=*A*

Для створення одиничної матриці можна використати метод identity, але є альтернативний варіант, метод eye, який при цьому дозволяє задати зміщення діагоналі з одиничними значеннями. Зміщення задаємо через ключ значення k

import numpy as np

a = np.eye(4, k=-1, dtype=float)

print(a)

*[[0. 0. 0. 0.]*

*[1. 0. 0. 0.]*

*[0. 1. 0. 0.]*

*[0. 0. 1. 0.]]*

**Обернена матриця**

[​](https://textbook.edu.goit.global/python/data-science-remaster/v1/docs/module-01/main_2#%D0%BE%D0%B1%D0%B5%D1%80%D0%BD%D0%B5%D0%BD%D0%B0-%D0%BC%D0%B0%D1%82%D1%80%D0%B8%D1%86%D1%8F)Ми розібрались із тим, що таке множення матриць. Для звичайного множення у нас є зворотня операція, як ділення, або ж ми можемо помножити на обернене число, тобто число в -1 степені. Для матричного множення також можна зробити зворотню операцію — для цього потрібно помножити на обернену матрицю. Таким чином ми і визначаємо [обернену матрицю](https://uk.wikipedia.org/wiki/%D0%9E%D0%B1%D0%B5%D1%80%D0%BD%D0%B5%D0%BD%D0%B0_%D0%BC%D0%B0%D1%82%D1%80%D0%B8%D1%86%D1%8F) [https://uk.wikipedia.org/ wiki/](https://uk.wikipedia.org/%20wiki/)Обернена\_матриця.

*A*⋅*A*−1=*A*−1⋅*A*=*I*

де *A*−1 є оберненою матрицею для матриці *A*. Обернена матриця існує лише для квадратної матриці — матриці, у якої к-сть рядків рівна к-сті стовпців.

Для отримання оберненої матриці необхідно використати функцію inv із пакету linalg.

Знайдемо *A*−1, для матриці *A* =

​

import numpy as np

a = np.array([[ 1, 0, 3],

[-1,-1, 2],

[ 4, 7, 2]])

a\_inv = np.linalg.inv(a)

print(a\_inv)

[[ 0.64 -0.84 -0.12]

[-0.4 0.4 0.2 ]

[ 0.12 0.28 0.04]]

**Властивості операцій з матрицями**[​](https://textbook.edu.goit.global/python/data-science-remaster/v1/docs/module-01/main_2#%D0%B2%D0%BB%D0%B0%D1%81%D1%82%D0%B8%D0%B2%D0%BE%D1%81%D1%82%D1%96-%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D1%96%D0%B9-%D0%B7-%D0%BC%D0%B0%D1%82%D1%80%D0%B8%D1%86%D1%8F%D0%BC%D0%B8)

Як ми переконались, що для множення матриць, повинні бути певні умови по їхніх розмірностях. І якщо ми можемо помножити матрицю *A* на матрицю *B*—*AB*, то для цих же матриць може не існувати добутку *BA*. Тому давайте запишемо таблицю основних властивостей операцій з матрицями.

A+B=B+A

*AB≠BA*

*A*+(*B*+*C*)=(*A*+*B*)+*C*

*A*(*BC*)=(*AB*)*C*

*A*(*B*+*C*)=*AB*+*AC*

(*A*+*B*)*T*=*AT*+*BT*

(*AB*)*T*=*BTAT*

(*λA*)*T*=*λAT*

(*AT*)*T*=*A*

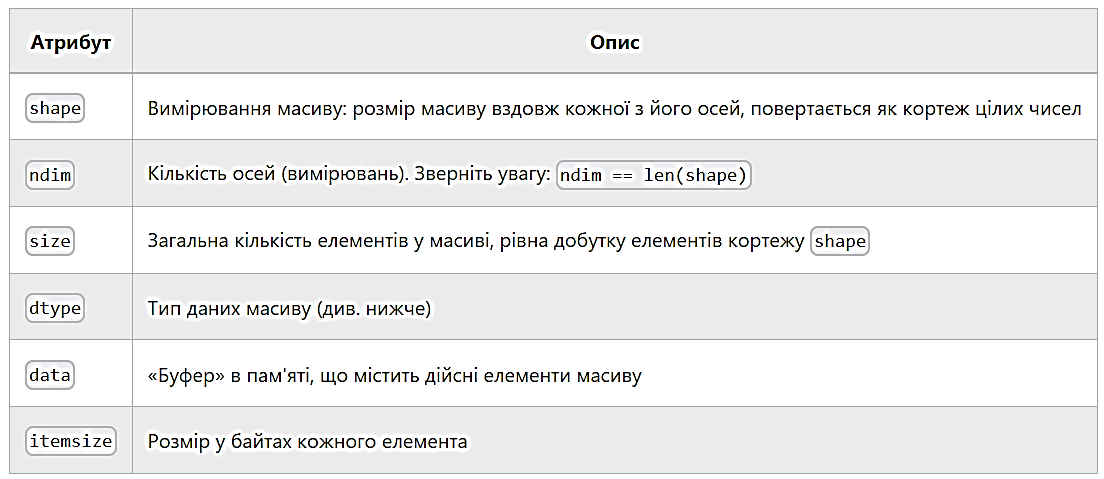
(*A*−1)−1=*A*

(*AB*)−1=*B*−1*A*−1

*I*−1=*I*

**Інші можливості в NumPy**

[**​**](https://textbook.edu.goit.global/python/data-science-remaster/v1/docs/module-01/main_2#%D1%96%D0%BD%D1%88%D1%96-%D0%BC%D0%BE%D0%B6%D0%BB%D0%B8%D0%B2%D0%BE%D1%81%D1%82%D1%96-%D0%B2-numpy)Усі атрибути масиву ndarray для інспекції зведені в таблицi



Щоб створити вектор або матрицю, які складаються з одиниць, використовують метод ones

import numpy as np

a = np.ones((5,), dtype=float)

print(a)

m = np.ones((2, 3), dtype=int)

print(m)

Виведення консолі:

[1. 1. 1. 1. 1.]

[[1 1 1]

[1 1 1]]

Щоб створити вектор або матрицю, які складаються з нулів, використовують метод zeros

import numpy as np

a = np.zeros(5, dtype=float)

print(a)

m = np.zeros((2, 3), dtype=int)

print(m)

Виведення консолі:

[0. 0. 0. 0. 0.]

[[0 0 0]

[0 0 0]]

Для створення масиву, що містить послідовність чисел, існують два методи: np.arange та np.linspace

Метод np.arange аналогічний методу Python range

import numpy as np

a = np.arange(5)

b = np.arange(1, 3, 0.5)

print(a)

print(b)

Виведення консолі:

[0 1 2 3 4]

[1. 1.5 2. 2.5]

Як і в range, масиви, згенеровані в цих прикладах, не включають останні елементи 5 та 3

Щоб включити останній елемент визначеної послідовності, можна використовувати метод np.linspace

import numpy as np

a = np.linspace(1, 5, num=5)

b = np.linspace(1, 3, num=3)

print(a)

print(b)

Виведення консолі:

[1. 2. 3. 4. 5.]

[1. 2. 3.]

Як бачимо ключовий параметр num визначає кількість елементів у послідовності.

Є можливість створювати масиви та матриці наповнені випадковими значеннями

import numpy as np

a = np.random.random(3)

b = np.random.random((3, 3))

print(a)

print(b)

Виведення консолі (при виконанні прикладу числа будуть іншими, але розмірність має бути тією самою):

[0.98686431 0.75971365 0.13002021]

[[0.78318837 0.18224777 0.99976367]

[0.98347105 0.67166331 0.28229379]

[0.75064336 0.97735688 0.2869088 ]]

**Індексація**

[​](https://textbook.edu.goit.global/python/data-science-remaster/v1/docs/module-01/main_2#%D1%96%D0%BD%D0%B4%D0%B5%D0%BA%D1%81%D0%B0%D1%86%D1%96%D1%8F)Масив NumPy можна розділити на частини та присвоїти їм індекси. Принцип роботи схожий на те, як це відбувається зі списками Python.

import numpy as np

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

print(a[0])

print(a[1:])

print(a[:2])

Виведення консолі:

1

[2 3 4 5]

[1 2]

Індексація багатовимірного масиву NumPy може здійснюватись як і індексація звичайного списку списків у Python: елемент з *i*-ого рядка та *j*-ого стовця можна отримати вказавши ці індекси в окремих квадратних дужках m[i][j], або ж ми можемо використати кортеж цілих чисел m[i, j]:

import numpy as np

a = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

print(a[1, 1])

print(a[1:, 1])

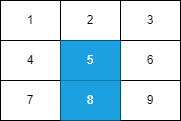
print(a[0, :2])

Виведення консолі:

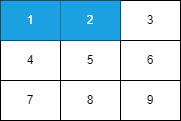
5

[5 8]

[1 2]



**a[1:, 1]**

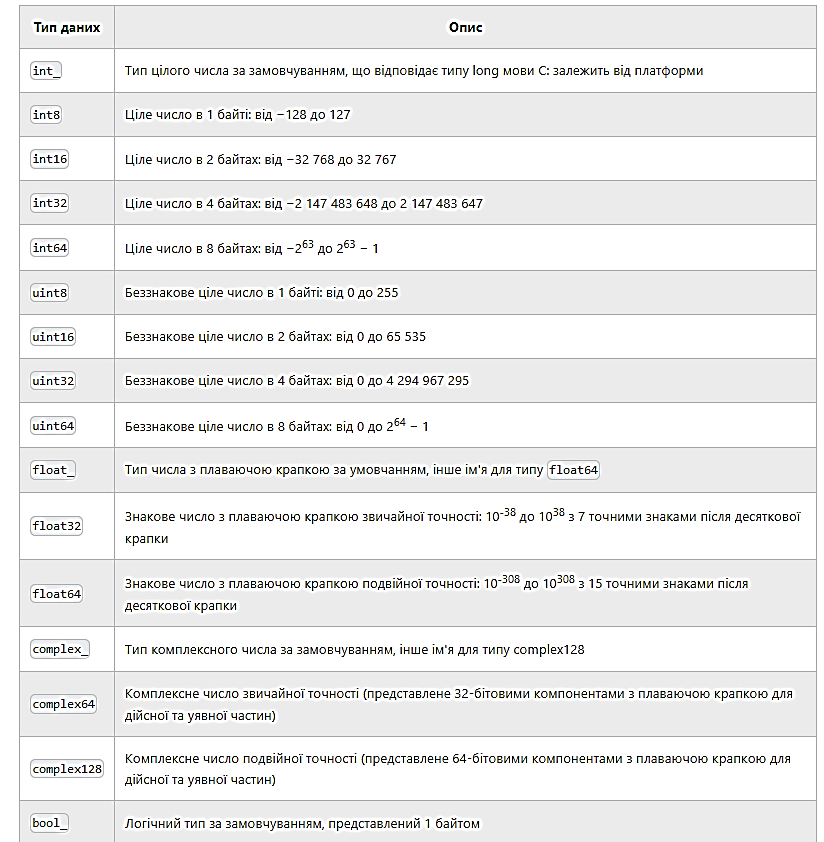


**a[0, :2]**

**Основні типи даних (dtype) NumPy**

​Бібліотека NumPy надає потужний інструмент для визначення типів даних (data type). Це необхідно для організації взаємодії зі скомпільованим кодом на C. Так як більш низького рівня елементи масиву NumPy обов'язково повинні зберігатись у сумісному форматі: тобто кожен елемент представлений у вигляді фіксованої кількості байтів.

Повний [список числових типів даних](https://docs.scipy.org/doc/numpy/user/basics.types.html) ([https://numpy.org/doc/stable/user /basics.types.html](https://numpy.org/doc/stable/user%20/basics.types.html)) наведено в офіційній документації бібліотеки NumPy. Нижче в таблиці перераховані найчастіше використовувані. Усі ці типи існують у пакеті NumPy, тому на них можна посилатися, наприклад, так: np.uint16. До типів даних, які створюються за умовчанням при використання вбудованих числових типів мови Python, додається кінцевий символ підкреслення: np.int\_, np.float\_, np.complex\_ та np.bool\_.



Також, досить часто, для визначення аргументу dtype використовується рядок, що складається з літери, що визначає розширену категорію типу даних (ціле, беззнакове ціле, комплексне число і т.д.), за яким слідує число, відповідне кількості байтів у цьому типі.

Наприклад:

b = np.zeros((3, 3), dtype="u4")

print(b)

Виведення консолі:

[[0 0 0]

[0 0 0]

[0 0 0]]

Ця інструкція створює матрицю розмірності 3 на 3 беззнакових 32-бітових (4-байтових) цілих чисел (рівнозначних типу np.uint32). Список літер, що позначають типи даних, що підтримуються, і їх значення наведені нижче в таблиці.



Приклад:

import numpy as np

a = np.array([u"\u2211", u"\u220F"], dtype="U")

print(a) *# ['∑' '∏']*

**Агрегування в NumPy**

[​](https://textbook.edu.goit.global/python/data-science-remaster/v1/docs/module-01/main_2#%D0%B0%D0%B3%D1%80%D0%B5%D0%B3%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F-%D0%B2-numpy)Пакет NumPy містить досить великий набір [функцій агрегування](https://jakevdp.github.io/PythonDataScienceHandbook/02.04-computation-on-arrays-aggregates.html)

Наприклад:

* min — мінімальне значення
* max — максимальне значення
* sum — сума всіх елементів
* mean — середнє арифметичне
* prod — перемножити всі елементи

Масив

import numpy as np

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

print(a.min())

print(a.max())

print(a.sum())

print(a.mean())

print(a.prod())

Виведення в консоль:

1

5

15

3.0

120

Матриця

import numpy as np

a = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

print(a.min())

print(a.max())

print(a.sum())

print(a.mean())

print(a.prod())

Виведення в консоль:

1

9

45

5.0

362880

**Додаткові джерела**

[​](https://textbook.edu.goit.global/python/data-science-remaster/v1/docs/module-01/main_2#%D0%B4%D0%BE%D0%B4%D0%B0%D1%82%D0%BA%D0%BE%D0%B2%D1%96-%D0%B4%D0%B6%D0%B5%D1%80%D0%B5%D0%BB%D0%B0)

Ми розглянули основні можливості та моменти роботи з пакетом NumPy. Для закріплення матеріалу рекомендуємо прочитати ще наступний цикл статей:

* [NumPy: the absolute basics for beginners](https://numpy.org/doc/stable/user/absolute_beginners.html)
* (https://numpy.org/doc/stable/user/absolute\_beginners.html)
* [Шпаргалка з основними можливостями у NumPy](https://s3.amazonaws.com/assets.datacamp.com/blog_assets/Numpy_Python_Cheat_Sheet.pdf) (https://s3.amazonaws.com/assets.datacamp.com/blog\_assets/Numpy\_Python\_Cheat\_Sheet.pdf)

А також використовуємо [офіційну документацію](https://numpy.org/doc/stable/user/whatisnumpy.html)

(https://numpy.org/doc/stable/user/whatisnumpy.html)