Мова програмування Python

NumPy в Python

Пакет NumPy - це свого роду робоча конячка для аналізу даних, машинного навчання та наукових обчислень в екосистемі Python. Цей пакет значно спрощує роботу з векторами і матрицями. Багато популярних пакетів Python (наприклад, scikit-learn, SciPy, pandas і tensorflow) включають NumPy в свою інфраструктуру в якості основного елемента. Крім можливості формування поздовжніх і поперечних зрізів даних пакет NumPy дозволяє налагоджувати і запускати більш складні сценарії використання цих бібліотек.

Функції, які надає NumPy:

→ функції створення масивів;
→ функції для виконання операцій над масивами;
→ функції введення і виведення даних;
→ базові математичні функції;
→ лінійна алгебра;
→ робота з поліномами;
→ статистика;
→ генерація випадкових чисел і розподілів;
→ дискретне перетворення Фур'є;
→ базові фінансові функції;
→ функції для роботи з датою і часом;
→ логічні функції;
→ функції для роботи з рядками;
→ функції для роботи з множинами;
→ функції пошуку, сортування та підрахунку елементів;
→ віконні функції.

- Стандартний метод це - використовувати простий вислів

```
>>> import numpy
```

- Цей вислів дозволяє нам отримувати доступ до numpy об'єктів використовуючи np.X замість numpy.X. Також можна імпортувати numpy прямо в використовуваний простір імен, щоб взагалі не використовувати функції через точку, а викликати їх безпосередньо

```
>>> import numpy as np
```

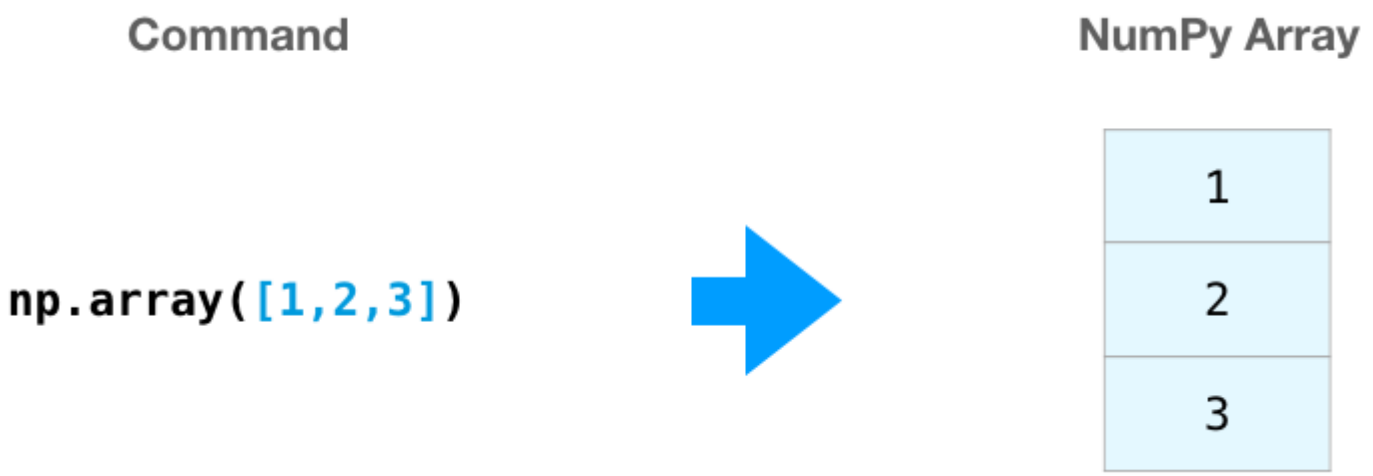
- для великої кількості викликів функцій numpy, стає утомливо писати numpy.X знову і знову. Замість цього набагато легше зробити це так

```
>>> from numpy import *
```

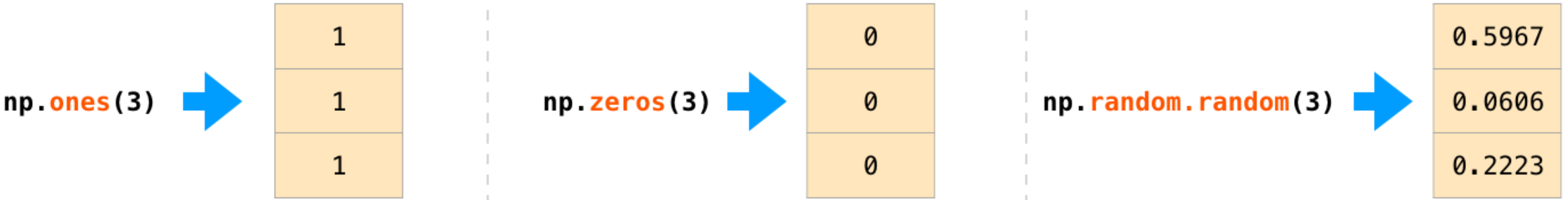
Однак, цей варіант не вітається в програмуванні на python, так як прибирає деякі корисні структури, які модуль надає. До кінця цієї теми ми будемо використовувати другий варіант імпорту (import numpy as np)

Створення масивів

Масиви в NumPy (ndarray) створюються за допомогою передачі в NumPy списку Python в функцію np.array(). У нашому випадку Python створює масив, показаний праворуч:



Часто буває потрібно, щоб NumPy сам ініціалізував значення масиву. У NumPy для таких випадків передбачені особливі методи, наприклад **ones()**, **zeros()** і **random.random()**. Потрібно просто повідомити цим методам кількість елементів, яке необхідно згенерувати:



Після створення масивів можна починати з ними працювати.

Арифметичні операції над масивами даних

Створимо два масиви NumPy і на їх прикладі покажемо переваги пакета.
Назвемо масиви data і ones:

```
data = np.array([1,2])
```

data

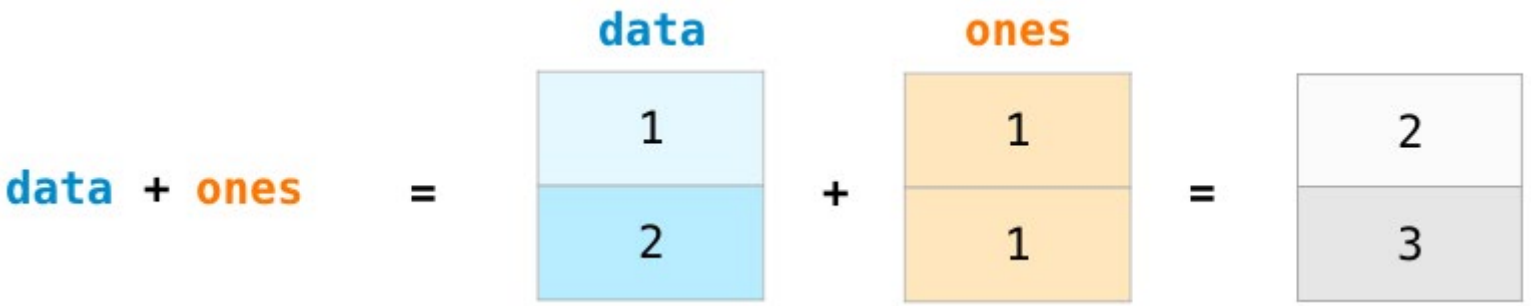
1
2

```
ones = np.ones(2)
```

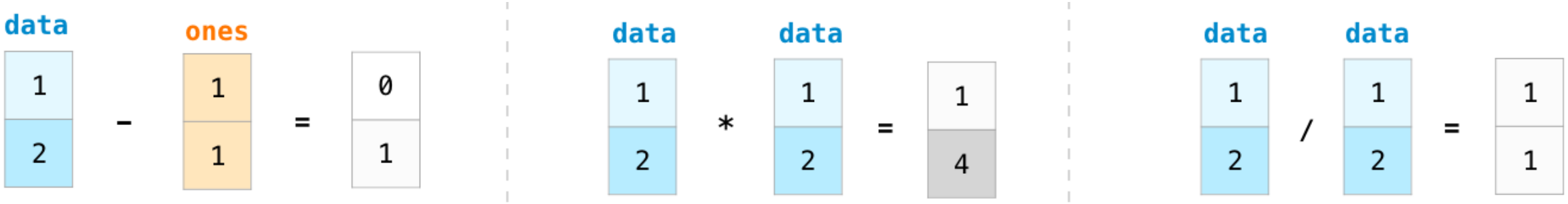
ones

1
1

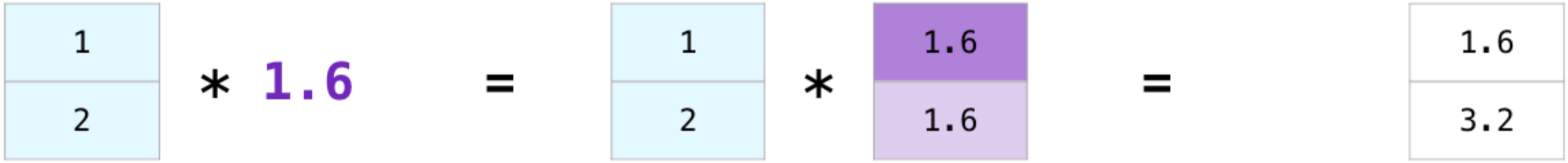
Підсумувати їх по позиціях (тобто підсумувати значення кожного рядка) дуже просто: треба ввести команду data + ones:



Чим хороші такі інструменти?
Тим, що така абстракція дозволяє позбутися від стомлюючого програмування циклів. Реалізований підхід настільки чудовий, що вивільняє розум для роздумів про проблеми на більш високому рівні.
Таким же чином можна виконувати не тільки операції додавання:



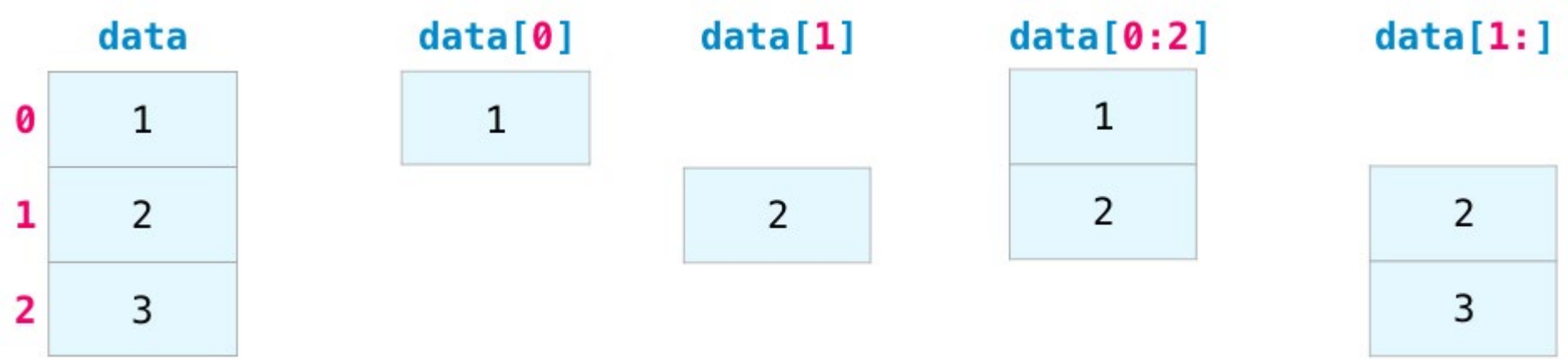
Часто виникають випадки, коли потрібно виконати арифметичну дію між усім масивом і одним числом (таку операцію назвемо векторно-скалярної).
Припустимо, що в масиві представлені дані про відстані в милях, а ми хочемо перетворити ці дані в кілометри.
Просто вводимо команду data * 1.6:



NumPy сам зрозумів, що помножити на вказане число потрібно кожен елемент масиву!
Така концепція називається *транслюванням* , і вона надзвичайно зручна.

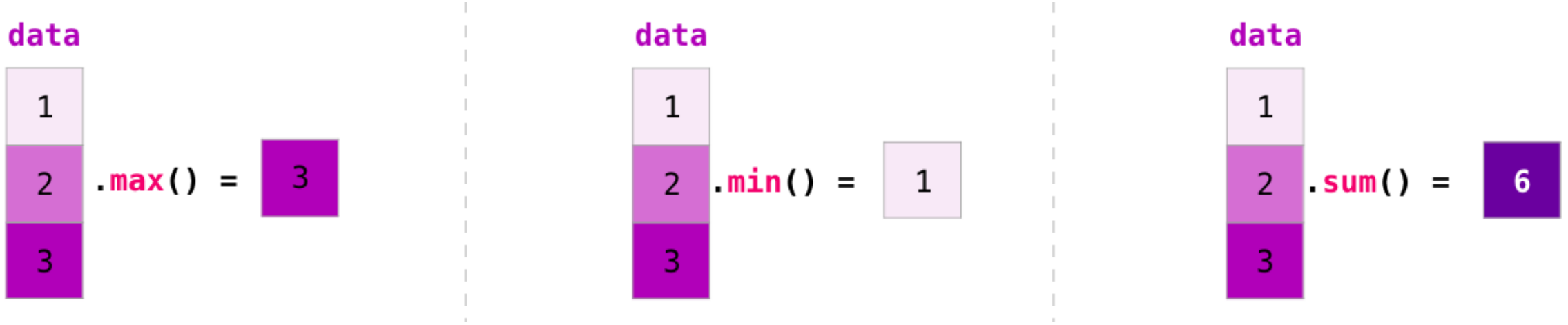
Індексування

У NumPy індексувати і нарізати [більш формально кажуть "робити зріз", а фахівці в Python називають зріз слайсами] масиви можна всіма способами, якими нарізуються списки Python:



Агрегування

Реалізована ще одна корисна функція - агрегування:



Крім функцій обчислення мінімального, максимального значення і суми (min, max і sum) можна скористатися такими чудовими функціями, як mean (для отримання середнього значення), prod (для перемноження всіх елементів), std (для обчислення стандартного відхилення), і безліччю інших

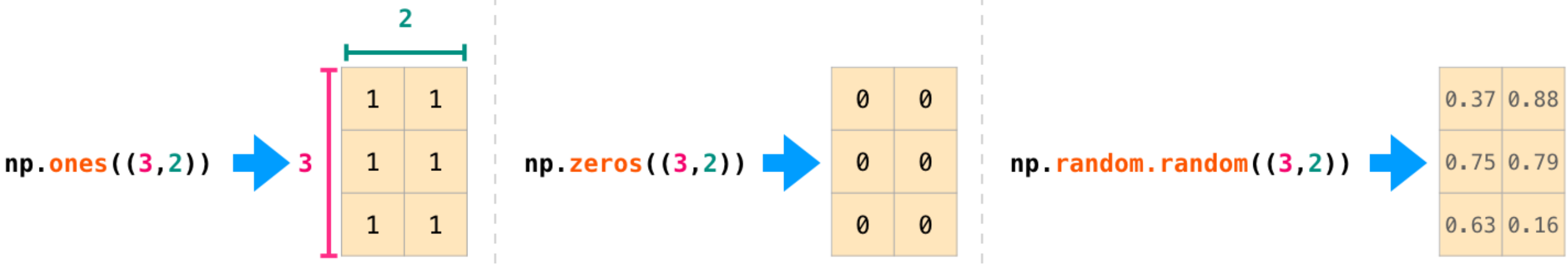
У всіх наведених вище прикладах використовувалися одномірні вектори. Але головна принада пакета NumPy полягає в його здатності застосовувати всі описані вище операції до будь-якої кількості розмірностей.

Створення матриць

Для того щоб NumPy створив матрицю для представлення списків зі списків Python, ми можемо передати такі списки в формі:

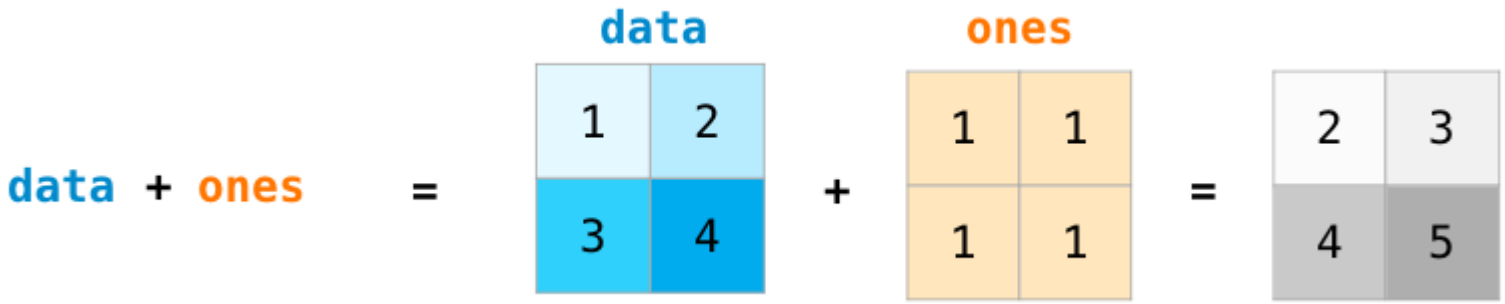
```
np.array([[1,2],[3,4]])
```

А якщо передати в згадані вище методи (ones (), zeros () і random.random ()) кортеж, що описує розмірність створюваної матриці, цими методами можна користуватися точно так же, як для одновимірних даних:

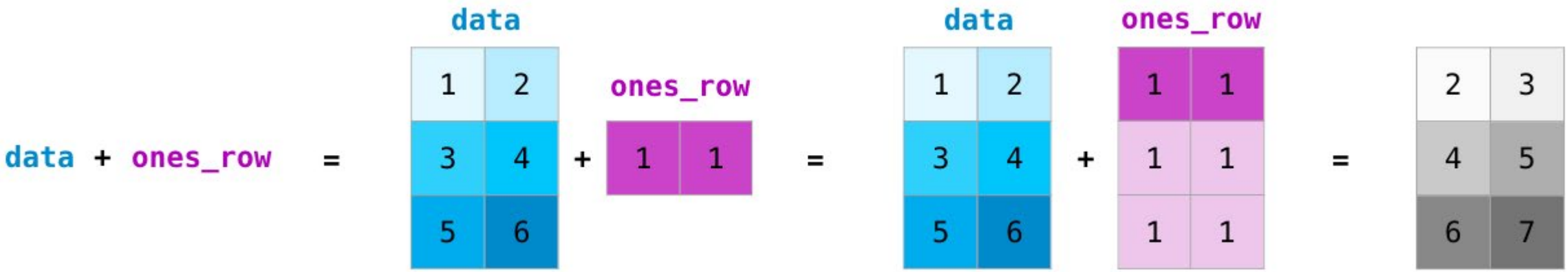


Арифметичні операції над матрицями

Якщо дві матриці мають однакову розмірність, їх можна складати і множити за допомогою арифметичних операторів (+ - * /). NumPy обробляє такі дії як позиційні операції:

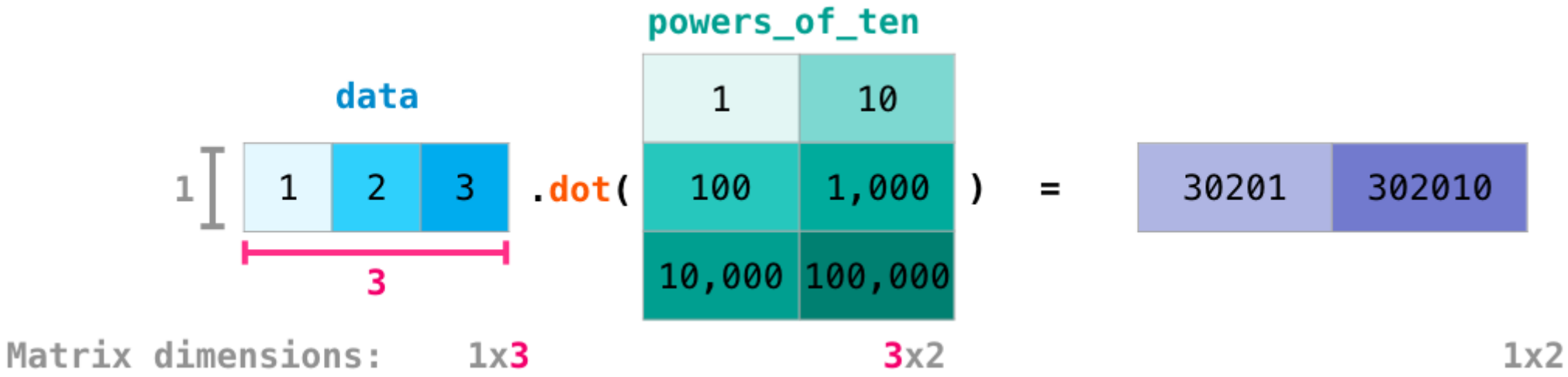


Якщо матриці мають різну розмірність, арифметичні операції до них можна застосовувати, тільки якщо розмірність однієї з матриць дорівнює одиниці (наприклад, матриця має лише один стовпець або один рядок), і в цьому випадку NumPy для даної операції використовує власні правила транслювання:

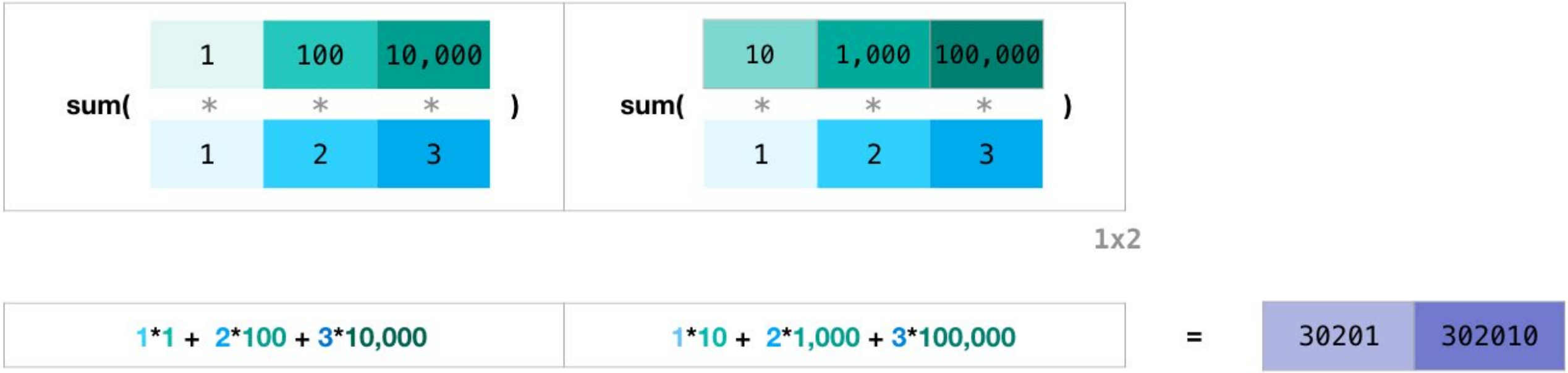


Скалярний добуток

Різновидом арифметичних операцій є операція множення матриць з використанням функції скалярного добутку. Для виконання в NumPy операції скалярного добутка над іншими матрицями до кожної матриці застосовується метод dot ():

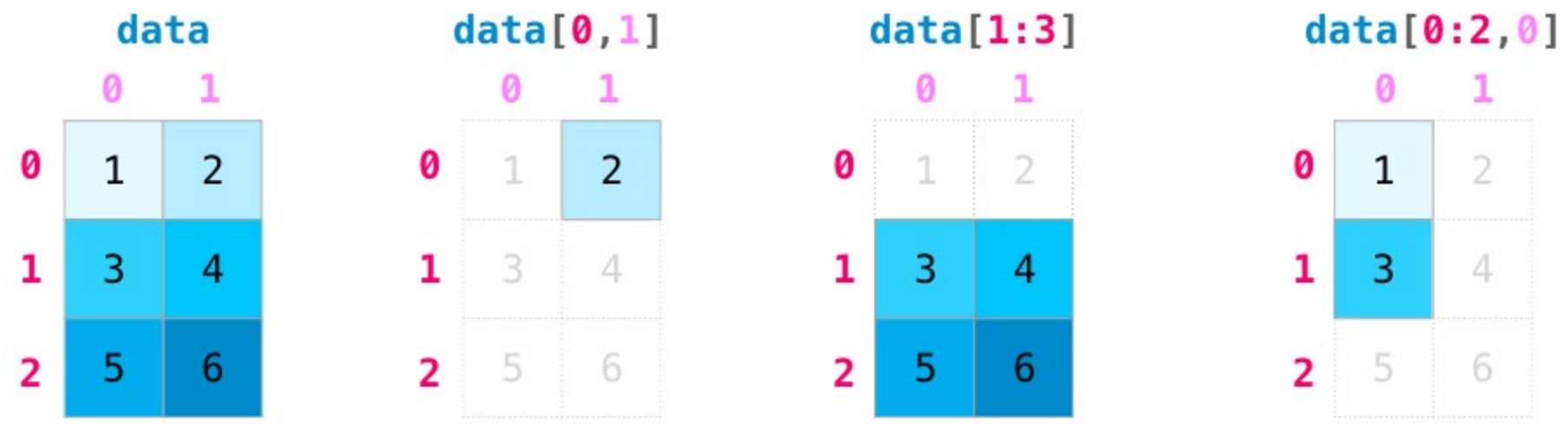


Внизу малюнка вказано розмірності матриць, щоб було зрозуміло, що для виконання операції обидві матриці повинні мати однакову розмірність на сторонах що "примикають" одна до одної. Наочно це можна представити таким чином:



Індексування матриць

При роботі з матрицями операції індексування і зрізи стають ще більш практичними:



Агрегування матриць

Агрегатори - це методи NumPy дозволяють замінювати дані інтегральними характеристиками вздовж деяких осей. Наприклад, можна порахувати середнє значення, максимальне, мінімальне, варіацію або ще якусь характеристику уздовж будь-якої осі або осей і сформувати з цих даних новий масив.
Форма нового масиву буде містити всі осі вихідного масиву, крім тих, уздовж яких підраховується агрегатор.

Для прикладу, сформуємо масив з випадковими значеннями. Потім знайдемо мінімальне, максимальне і середнє значення в його стовпцях:

```
A = np.random.rand(4, 5)

print('A\n', A, '\n')

print('min\n', np.min(A, 0), '\n')
print('max\n', np.max(A, 0), '\n')
print('mean\n', np.mean(A, 0), '\n')
print('average\n', np.average(A, 0), '\n')
```

```
A
 [[0.42142816 0.98752085 0.9354305  0.35546861 0.00730033]
 [0.47836628 0.02699282 0.0659851  0.45678776 0.01926673]
 [0.74195363 0.4795257  0.63953906 0.44356599 0.37541715]
 [0.50830664 0.56138531 0.03986163 0.98910914 0.07657117]]

min
 [0.42142816 0.02699282 0.03986163 0.35546861 0.00730033]

max
 [0.74195363 0.98752085 0.9354305  0.98910914 0.37541715]

mean
 [0.53751368 0.51385617 0.42020407 0.56123287 0.11963885]

average
 [0.53751368 0.51385617 0.42020407 0.56123287 0.11963885]
```

При такому використанні mean і average виглядають синонімами. Але ці функції мають різний набір додаткових параметрів. У них різні можливості по маскуванню і зважуванню усереднених даних.

Можна підрахувати інтегральні характеристики і по декількох осях:

```
A = np.ones((10, 4, 5))

print('sum\n', np.sum(A, (0, 2)), '\n')
print('min\n', np.min(A, (0, 2)), '\n')
print('max\n', np.max(A, (0, 2)), '\n')
print('mean\n', np.mean(A, (0, 2)), '\n')
```

```
sum
 [50. 50. 50. 50.]

min
 [1. 1. 1. 1.]

max
 [1. 1. 1. 1.]

mean
 [1. 1. 1. 1.]
```

У цьому прикладі розглянута ще одна інтегральна характеристика sum - сума.

Список агрегаторів виглядає приблизно так:

→ сума: sum і nansum - варіант що коректно обходиться з nan;

→ добуток: prod і nanprod;

→ середнє і математичне очікування: average і mean (nanmean), nanaverage немає;

→ медіана: median і nanmedian;

→ перцентиль: percentile і nanpercentile;

→ варіація: var і nanvar;

→ стандартне відхилення (квадратний корінь з варіації): std і nanstd;

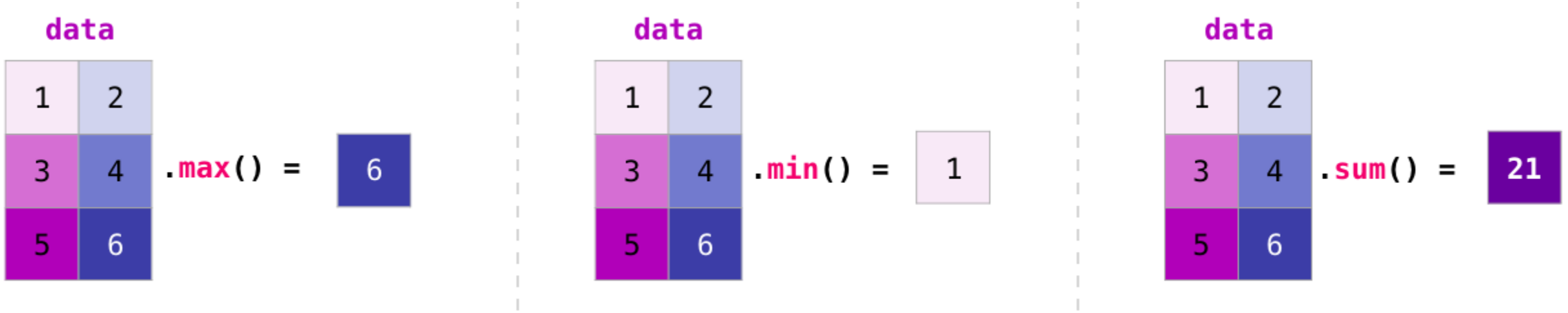
→ мінімальне значення: min і nanmin;

→ максимальне значення: max і nanmax;

→ індекс елемента, що має мінімальне значення: argmin і nanargmin;

→ індекс елемента, що має максимальне значення: argmax і nanargmax.

Агрегування матриць здійснюється точно так же, як агрегування векторів:

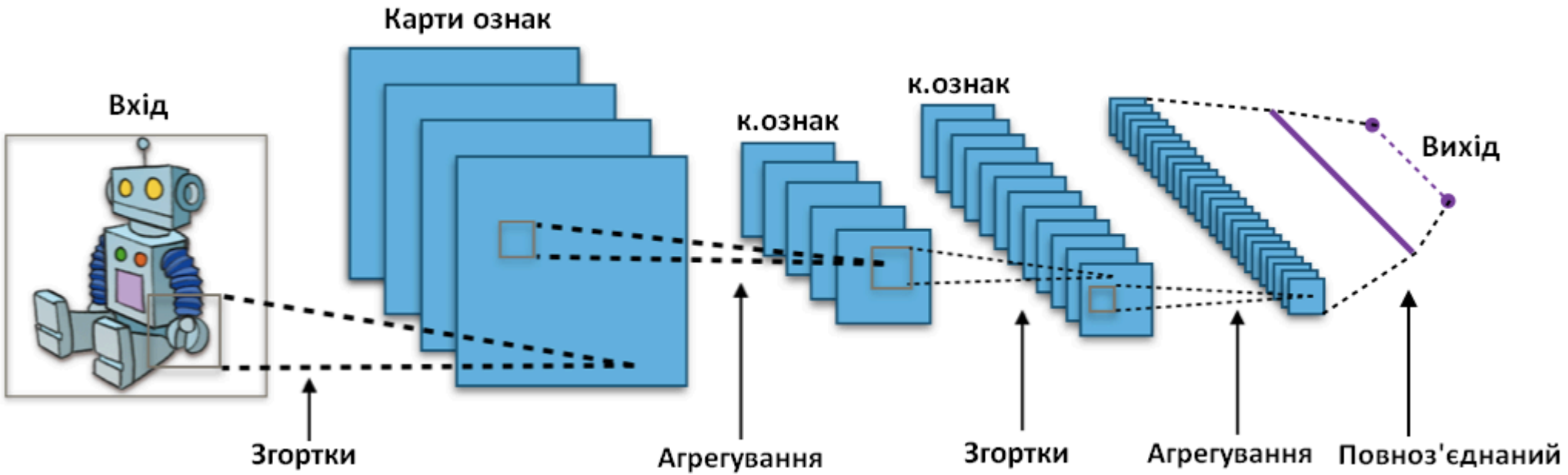


Згорткові нейронні мержі (ЗНМ, англ. convolutional neural network, CNN, ConvNet) в машинному навчанні — це клас глибинних штучних нейронних мереж прямого поширення, який успішно застосовувався до аналізу візуальних зображень.

Згорткові мережі взяли за основу біологічний процес, а саме схему з'єднання нейронів зорової кори тварин.

Вони мають застосування в розпізнаванні зображень та відео, рекомендаційних системах та обробці природної мови.

Типова архітектура ЗНМ



ЗНМ складається з шарів входу та виходу, а також із декількох прихованих шарів.
Приховані шари ЗНМ зазвичай складаються зі згорткових шарів, агрегувальних шарів, повноз'єднаних шарів та шарів нормалізації.
Цей процес описують в нейронних мережах як згортку за домовленістю. З математичної точки зору він є радше взаємною кореляцією, ніж згорткою.
Це має значення лише для індексів у матриці, й відтак які ваги на якому індексі розташовуються.

Згорткові шари
Згорткові шари застосовують до входу операції згортки, передаючи результат до наступного шару. Згортка імітує реакцію окремого нейрону на зоровий стимул
Кожен згортковий нейрон обробляє дані лише для свого рецептивного поля.

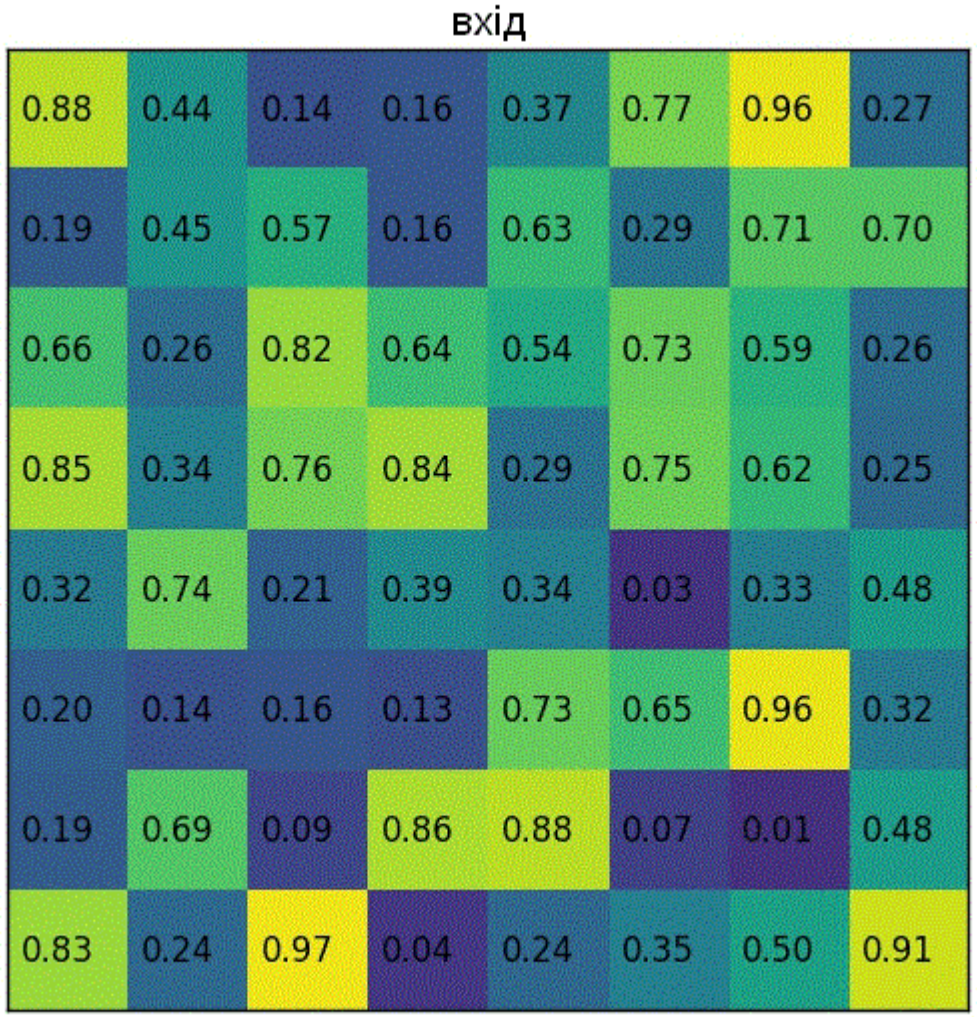
Агрегувальні шари

Згорткові мережі можуть включати шари локального або глобального агрегування, які об'єднують виходи кластерів нейронів одного шару до одного нейрону наступного шару.

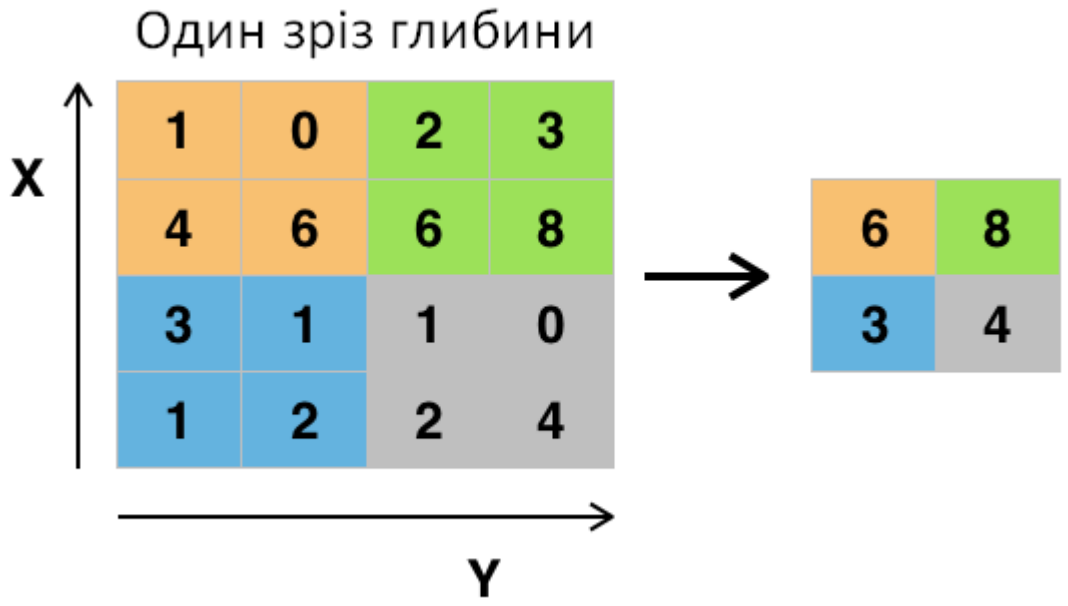
Наприклад, *максимізаційне агрегування* (англ. *max pooling*) використовує максимальне значення з кожного з кластерів нейронів попереднього шару.

Іншим прикладом є *усереднювальне агрегування* (англ. *average pooling*), що використовує усереднене значення з кожного з кластерів нейронів попереднього шару.

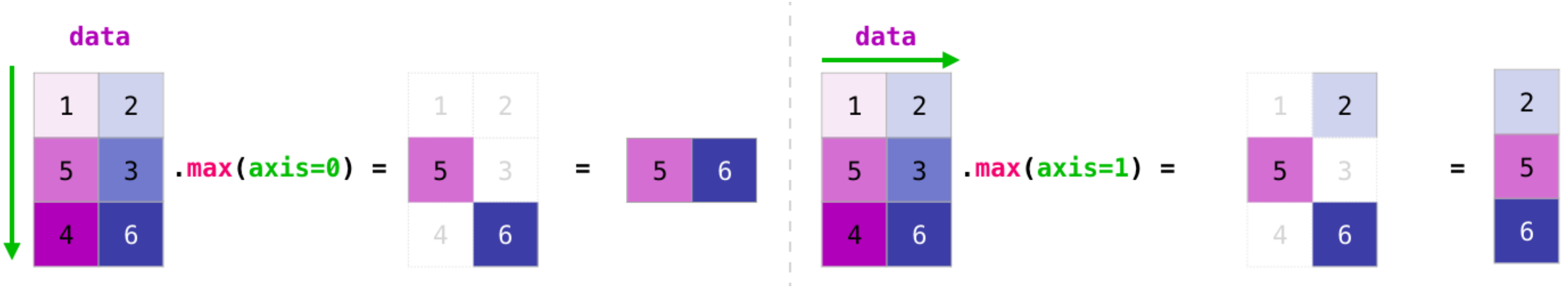
Агрегування областей інтересу (англ. *Region of Interest pooling*, відоме також як англ. *RoI pooling*), — це варіація максимізаційного агрегування, в якій розмір виходу фіксовано, а прямокутник входу є параметром.
Агрегування є важливою складовою згорткових нейронних мереж для виявлення об'єктів, що ґрунтуються на архітектурі швидких згорткових нейронних мереж на основі областей (англ. *Fast R-CNN*).



Максимізаційне агрегування (англ. *max pooling*) із фільтром 2×2 та кроком = 2

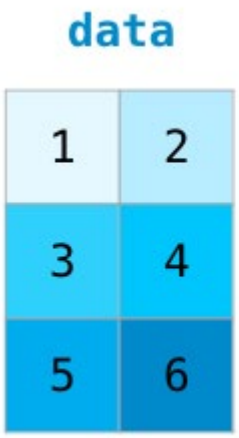


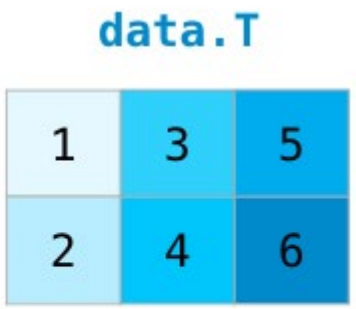
Агрегувати можна не тільки все значення в матриці, але і значення рядків або стовпців за допомогою параметра axis:



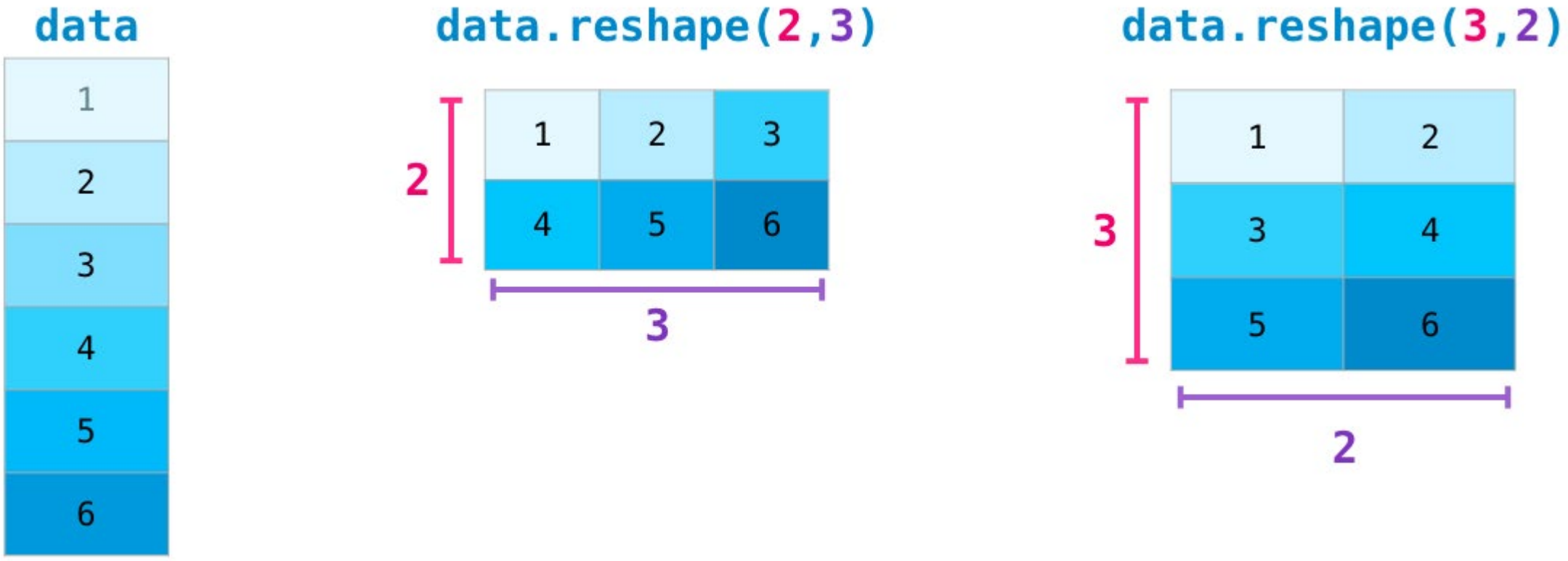
Транспонування і зміна форми матриць

При роботі з матрицями часто виникає необхідність їх повороту. Такий поворот (транспонування) часто необхідний, коли потрібно взяти скалярний добуток двох матриць і для цього привести їх до загальної розмірності. Для транспонування матриці в масивах NumPy передбачено зручна властивість T:





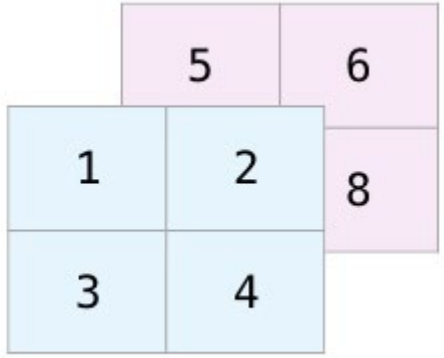
У більш складних випадках може знадобитися перемикання розмірностей певної матриці. Необхідність в цьому часто виникає в додатках машинного навчання, коли певна модель очікує на вході певну форму даних, але на вхід надходить інша форма. Для таких випадків в NumPy передбачений метод **reshape()**. У цей метод потрібно просто передати нові розмірності матриці. Якщо вказати розмірність -1, NumPy на основі вашої матриці зможе визначити правильну розмірність:



А якщо розмірності ще більш високі?

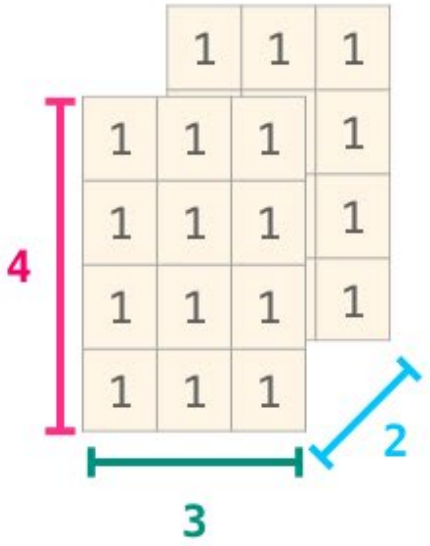
NumPy може виконувати всі вищезгадані операції з матрицями будь-якої розмірності. Не дарма ж основна структура даних NumPy називається n-мірним масивом (ndarray, N-Dimensional Array).

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

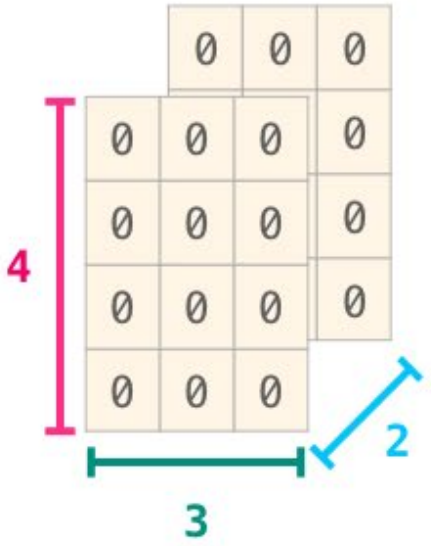


Часто ввести нову розмірність можна простим додаванням коми і числа до параметрів функції NumPy:

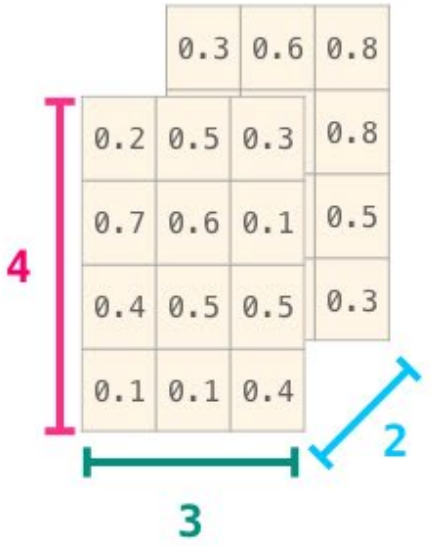
```
np.ones((4,3,2))
```




```
np.zeros((4,3,2))
```




```
np.random.random((4,3,2))
```



Примітка: слід зазначити, що при роздруківці тривимірного масиву NumPy виведений текст відображається трохи інакше. n-мірні масиви в NumPy роздруковуються таким чином, що в першу чергу здійснюється прохід по елементам останньої координати матриці, а в останню чергу - по першій. Іншими словами, np.ones ((4,3,2)) на роздруківці буде виглядати так:

```
array([[[1., 1.],
        [1., 1.],
        [1., 1.]],

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

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

       [[1., 1.],
        [1., 1.],
        [1., 1.]]])
```

Застосування на практиці

Отже, що нам дає цей пакет? Ось кілька корисних прикладів застосування NumPy.

Формули

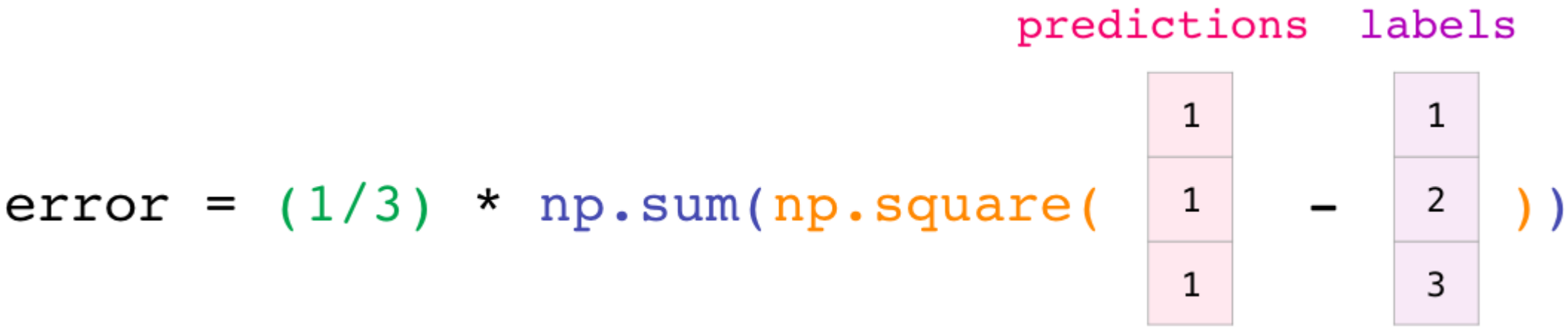
Основною областю застосування NumPy є реалізація математичних формул, які працюють з матрицями і векторами. Саме з цієї причини з NumPy так люблять працювати члени наукового співтовариства Python. Візьмемо, наприклад, головну для моделей машинного навчання формулу для розрахунку середньоквадратичної похибки, яка використовується при вирішенні задач регресії:

$$MeanSquareError = \frac{1}{n}\sum_{i=1}^{n}(Y\_prediction_i - Y_i)^2$$

У NumPy ця формула реалізується дуже просто:

```
error = (1/n) * np.sum(np.square(predictions - labels))
```

Привабливість у тому, що для NumPy байдуже, чи містять вектори predictions і labels одне або тисячу значень (головне, щоб їх розмірності були однаковими). Розглянемо приклад більш уважно, послідовно виконавши чотири операції в цьому рядку коду:



Вектори predictions і labels містять по три значення. Іншими словами, n дорівнює 3.
Після виконання операції віднімання отримуємо такі значення:

```
error = (1/3) * np.sum(np.square(
```

0
-1
-2

```
))
```

Зведемо в квадрат значення в векторі:

```
error = (1/3) * np.sum(
```

0
1
4

```
)
```

І підсумовуємо ці значення:

```
error = (1/3) * 5
```

В результаті отримуємо значення похибки для даного прогнозу і оцінку якості моделі.

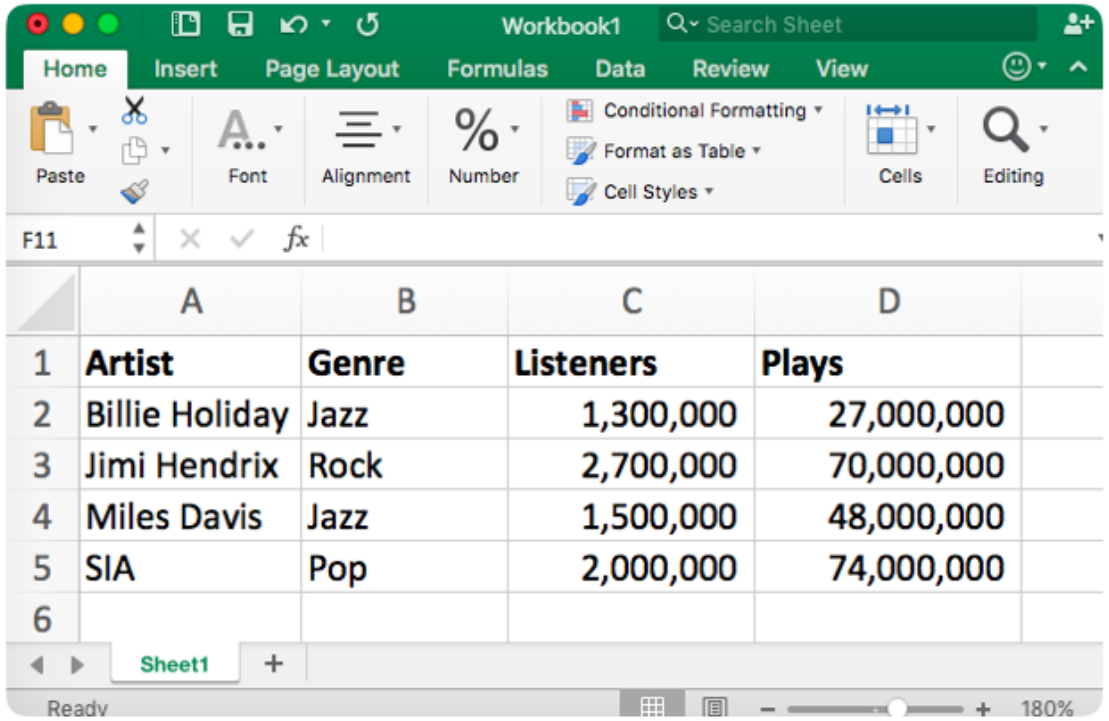
Подання даних

Уявіть, яку безліч типів даних вам, можливо, доведеться обробляти і передавати в моделі (електронні таблиці, зображення, аудіо ... і т. д.). Багато з таких даних ідеально підходять для подання в n-вимірному масиві:

Електронні таблиці

Електронна таблиця, або таблиця значень, нагадує двовимірну матрицю. Кожен лист в електронній таблиці може бути окремою змінною. Найбільш популярною абстракцією в Python для них є об'єкт pandas dataframe , який фактично використовує NumPy і є його надбудовою.

music.csv



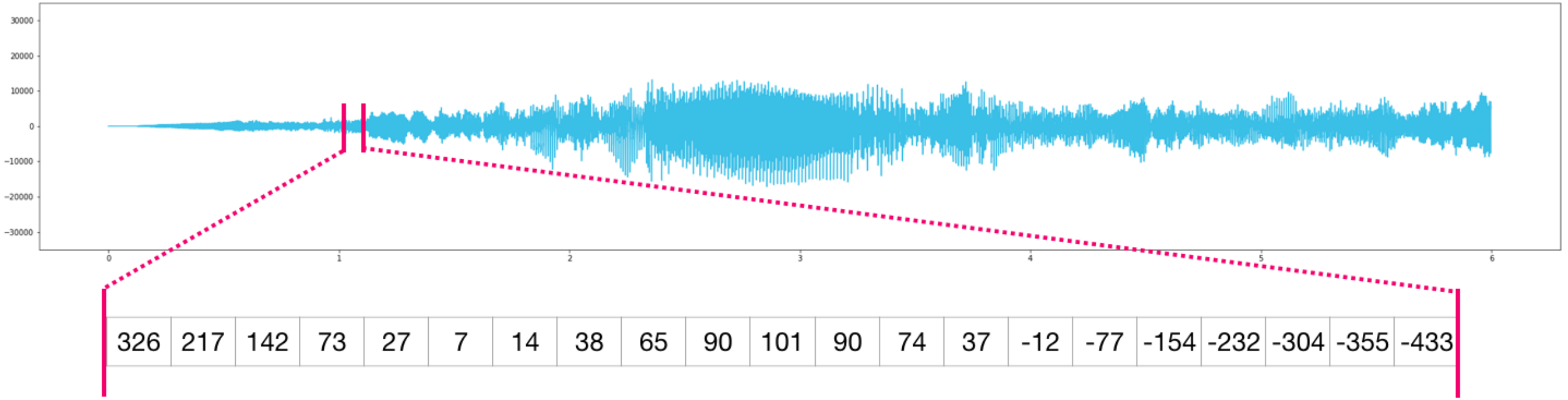
	A	B	C	D
1	**Artist**	**Genre**	**Listeners**	**Plays**
2	Billie Holiday	Jazz	1,300,000	27,000,000
3	Jimi Hendrix	Rock	2,700,000	70,000,000
4	Miles Davis	Jazz	1,500,000	48,000,000
5	SIA	Pop	2,000,000	74,000,000
6				

pandas.read_csv('music.csv')

	Artist	Genre	Listeners	Plays
0	Billie Holiday	Jazz	1,300,000	27,000,000
1	Jimi Hendrix	Rock	2,700,000	70,000,000
2	Miles Davis	Jazz	1,500,000	48,000,000
3	SIA	Pop	2,000,000	74,000,000

Аудіо та тимчасові ряди

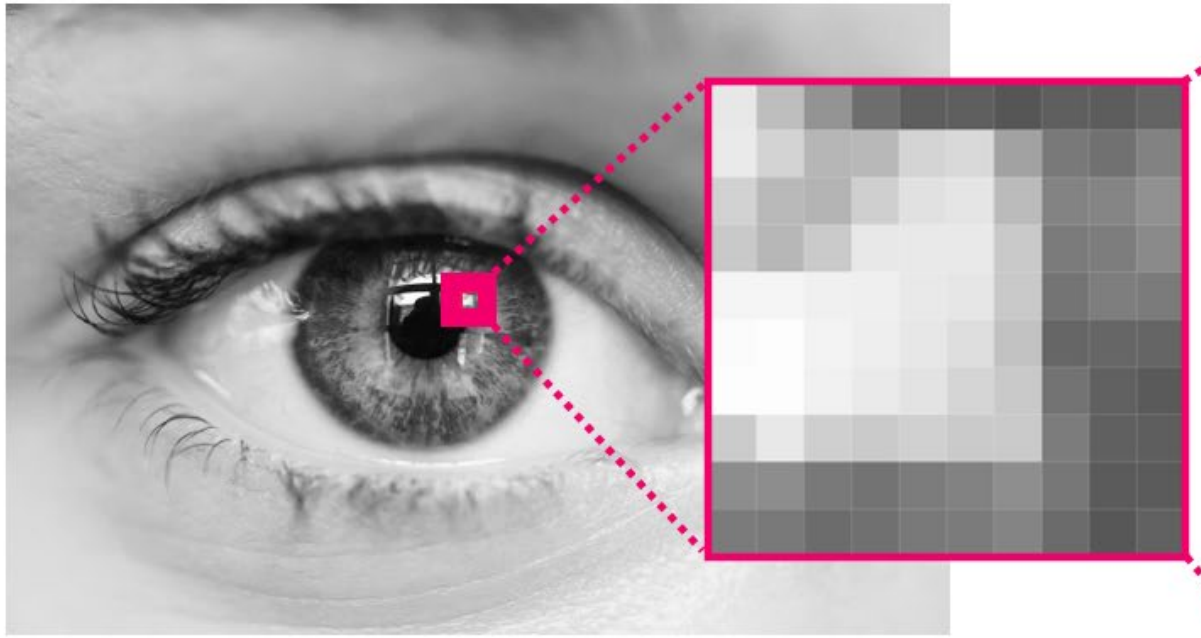
Аудіофайл - це одновимірний масив семплів. Кожен семпл - це число, яке представляє собою крихітний фрагмент аудіосигналу. В аудіо CD-якості можуть міститися до 44 100 семплів в секунду, кожен семпл є цілим числом від -32767 до 32768. Іншими словами, якщо у вас є десятісекундний WAVE-файл CD-якості, його можна завантажити в масив NumPy довжиною 10 * 44100 = 441000 семплів. Хочете отримати першу секунду звуку? Просто завантажте файл в масив NumPy (назвемо його audio) і напишіть audio [: 44100]. Ось так може виглядати фрагмент аудіофайлу:



Аналогічним чином NumPy чинить з даними часових рядів (наприклад, з динамікою зміни цін на акції протягом часу).

Зображення

Зображення - це матриця пікселів розміром (висота x ширина).
Якщо зображення чорно-біле (в так званій градації сірого), кожен піксель може бути представлений одним числом (зазвичай від 0 (чорний) до 255 (білий)). Хочете отримати зріз верхній лівій частині зображення розміром 10 x 10 пікселів? Просто виконайте в NumPy цей код: image [: 10: 10].
Ось так може виглядати фрагмент файлу зображення:



230	194	147	108	90	98	84	96	91	101
237	206	188	195	207	213	163	123	116	128
210	183	180	205	224	234	188	122	134	147
198	189	201	227	229	232	200	125	127	135
249	241	237	244	232	226	202	116	125	126
251	254	241	239	230	217	196	102	103	99
243	255	240	231	227	214	203	116	95	91
204	231	208	200	207	201	200	121	95	95
144	140	120	115	125	127	143	118	92	91
121	121	108	109	122	121	134	106	86	97

Якщо зображення кольорове, кожен піксель представляється трьома числами з червоного, зеленого і синього спектральних кольорів. В даному випадку нам необхідна третя розмірність (так як кожна клітинка може містити тільки одне число). Відповідно, кольорове зображення представляється масивом розмірностей: (висота x ширина x 3).

	233	188	137	96	90	95	63	73	73	82
237	202	159	120	105	110	88	107	112	121	109
226	191	147	110	101	112	98	123	110	119	142	131
221	191	176	182	203	214	169	144	133	145	155	122
185	160	161	184	205	223	186	137	147	161	140	115
181	174	189	207	206	215	194	136	142	151	133	87
246	237	237	231	208	206	192	122	143	144	111	74
254	254	241	224	199	192	181	99	122	117	107	74
239	248	232	207	187	182	184	110	114	110	113	74
193	215	193	167	158	164	181	114	112	111	105	82
113	119	110	111	113	123	135	120	108	106	113	
93	97	91	103	107	111	122	112	104	114		

Текст

При роботі з текстом ситуація дещо інша. Для числового представлення тексту спочатку необхідно створити словник (перелік усіх унікальних слів, які повинна знати модель) і його векторне уявлення (перший етап). Спробуємо уявити в цифровій формі цитату з вірша, перекладену на англійську мову:
"Have the bards who preceded me left any theme unsung?" (Чи залишили барди до мене якийсь із предметів неоспіваним?)

Перед переведенням цього речення в потрібну цифрову форму модель повинна проаналізувати величезну кількість тексту. Відправимо в модель на обробку невеликий набір даних і використовуємо його для створення словника (з 71290 слів):

Model Vocabulary

#	
0	the
1	of
2	and
…	…
71,289	dolophine

Після цього розіб'ємо речення на масив лексем (слів або частин слів, заснованих на загальних правилах):

Потім замінимо кожне слово його ідентифікатором в словникової таблиці:

Однак для роботи моделі такі ідентифікатори не годяться. Тому перед передачею послідовності слів в модель лексеми / слова повинні бути замінені їх векторними уявленнями (в даному випадку використовується 50-мірне векторне уявлення word2vec):

	have	the	bards	who	preceded	me	left	any	theme	unsung
0	-1.621	-0.634	-0.324	-1.357	-0.261	4.632	1.236	-0.046	-1.518	-0.082
1	-1.401	0.683	-0.114	1.891	0.544	-1.487	1.247	-0.408	0.202	-0.022
	…	…	…	…	…	…	…	…	…	…
49	-0.647	0.200	-0.203	1.573	-0.520	-0.355	-1.491	1.325	2.550	-0.010

Очевидно, що цей масив NumPy має наступні розмірності [embedding_dimension x sequence_length]. В реальності все виглядає трохи інакше, проте дане візуальне уявлення більш наочно відображає загальні принципи. З міркувань продуктивності моделі глибокого навчання, як правило, зберігають першу розмірність для пакета (оскільки модель навчається швидше на декількох паралельних прикладах). Саме в таких випадках може виявитися корисною функція **reshape()**. У модель типу BERT , наприклад, дані будуть вводитися в такій формі: [batch_size, sequence_length, embedding_size].

В результаті ми отримали числовий том, з яким може працювати модель. Деякі рядки залишились порожніми, проте їх можна заповнити іншими прикладами, на яких може тренуватися модель (або робити прогнози).