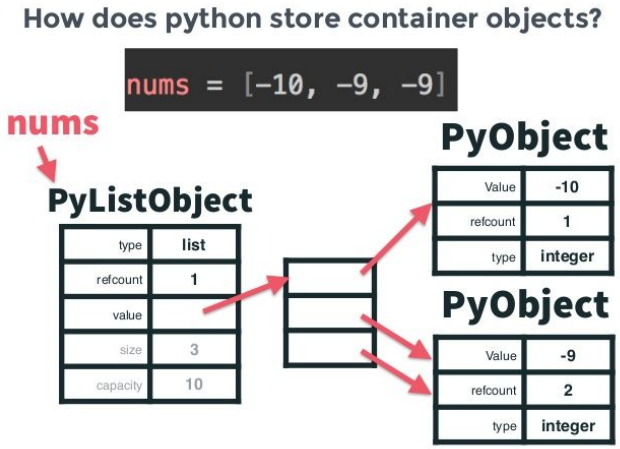
https://dfedorov.spb.ru

```
>>> x = [1, 2, 3, 4]
>>> y = [5, 6, 7, 8]

>>> x * 2

>>> x + 10

>>> x + y
```



Операции с однородными структурами на Python крайне затратны!

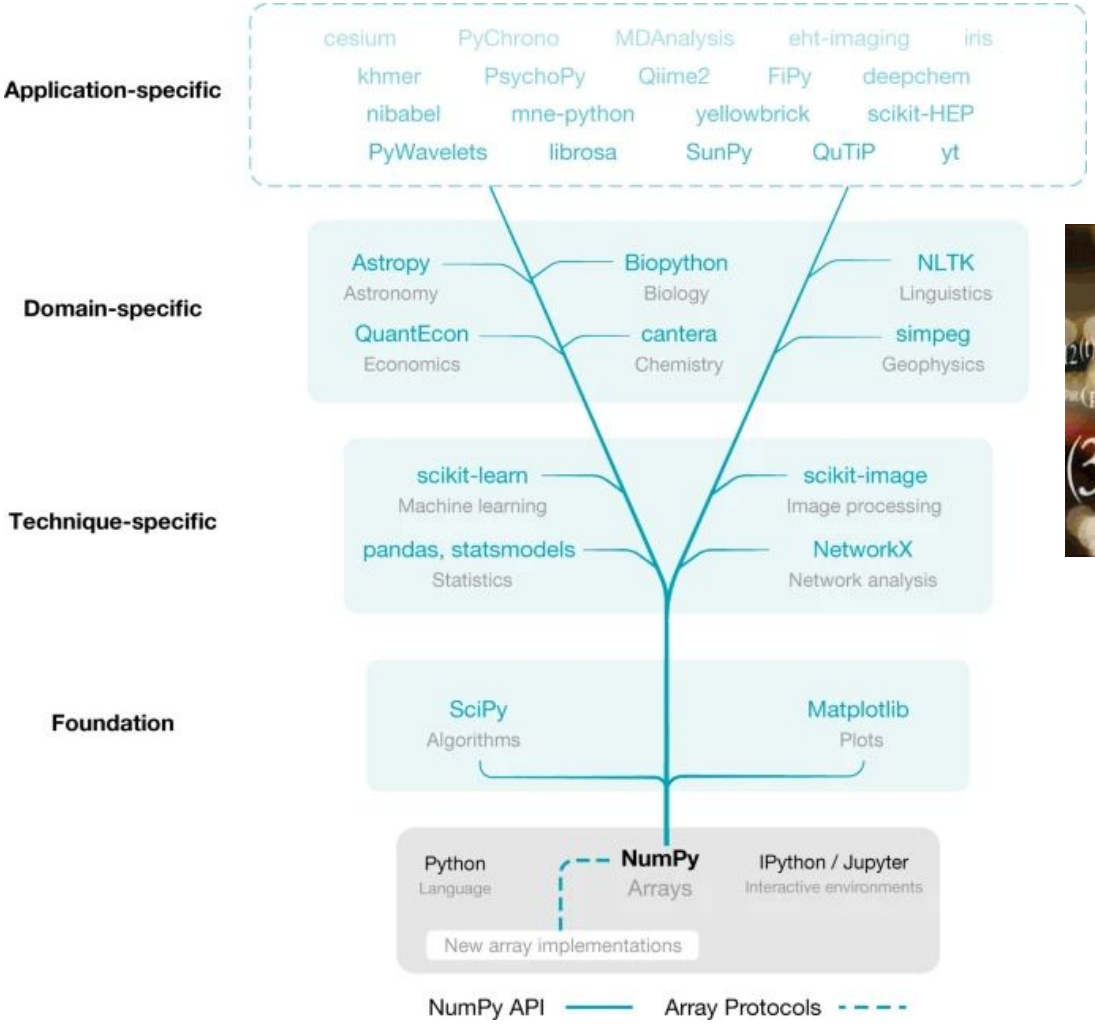
Как поэлементно сложить два списка на Python?

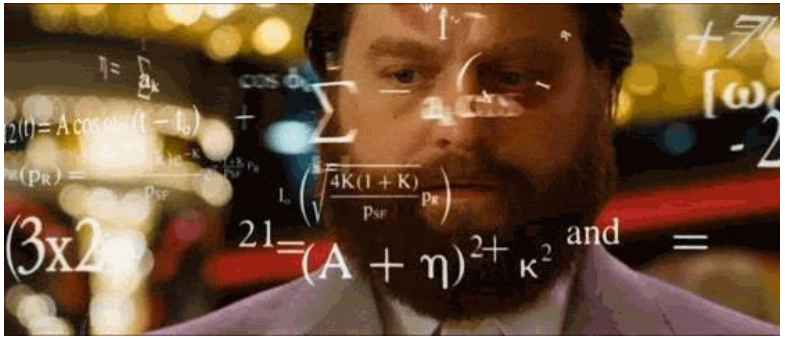
```
>>> list(zip(x, y))

>>> lst = []
for first, second in zip(x, y):
    lst.append(first + second)

>>> lst
```

Пакет NumPy - это рабочая лошадка для анализа данных, машинного обучения и научных вычислений в экосистеме Python. Это значительно упрощает манипулирование векторами и матрицами и их обработку. Некоторые из ведущих пакетов Python полагаются на NumPy как на фундаментальную часть своей инфраструктуры (примеры включают **scikit-learn**, **SciPy**, **pandas** и **tensorflow**).

Вычисления с массивами в Python уходят корнями в 1995 год, когда Джим Хьюганин (Jim Hugunin) создал библиотеку Numeric. В течение следующих десяти лет программирование с массивами распространилось во многих научных сообществах, но в начале 2000-х годов библиотечная экосистема оказалась фрагментированной.

В 2005 году Трэвис Олифант (Travis Oliphant) сумел собрать проект NumPy из существовавших тогда проектов Numeric и Numarray, чтобы объединить сообщество вокруг общей вычислительной инфраструктуры.





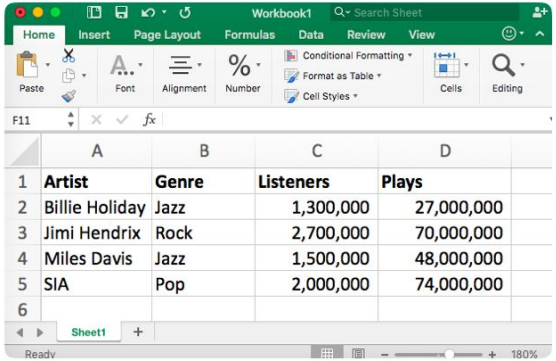
https://www.nature.com/articles/s41586-020-2649-2

Представление данных

Подумайте обо всех типах данных, которые вам понадобятся для обработки и построения моделей (электронные таблицы, изображения, аудио и т. д.). Многие из них идеально подходят для представления в n-мерном массиве:

Электронная таблица или таблица значений - это двухмерная матрица. Каждый лист в электронной таблице может быть собственной переменной. Самая популярная абстракция в Python - кадр данных pandas, который фактически использует NumPy и строит поверх него.

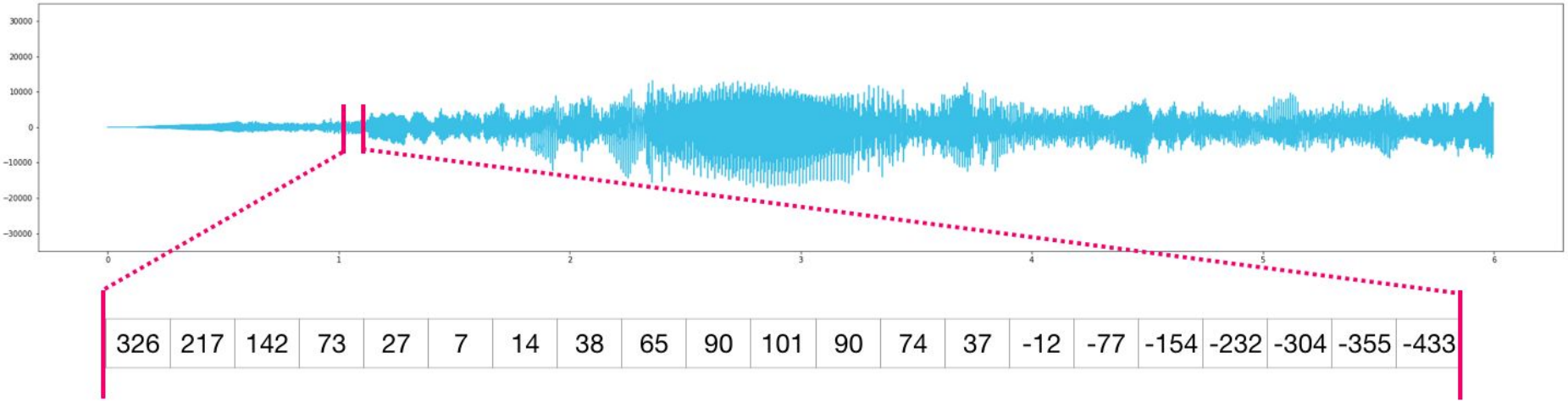
music.csv



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

Аудио и временные ряды

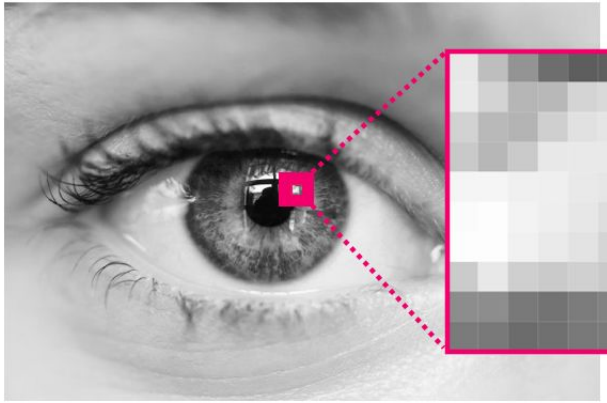


То же самое касается данных временных рядов (например, цены акции с течением времени).

Изображения

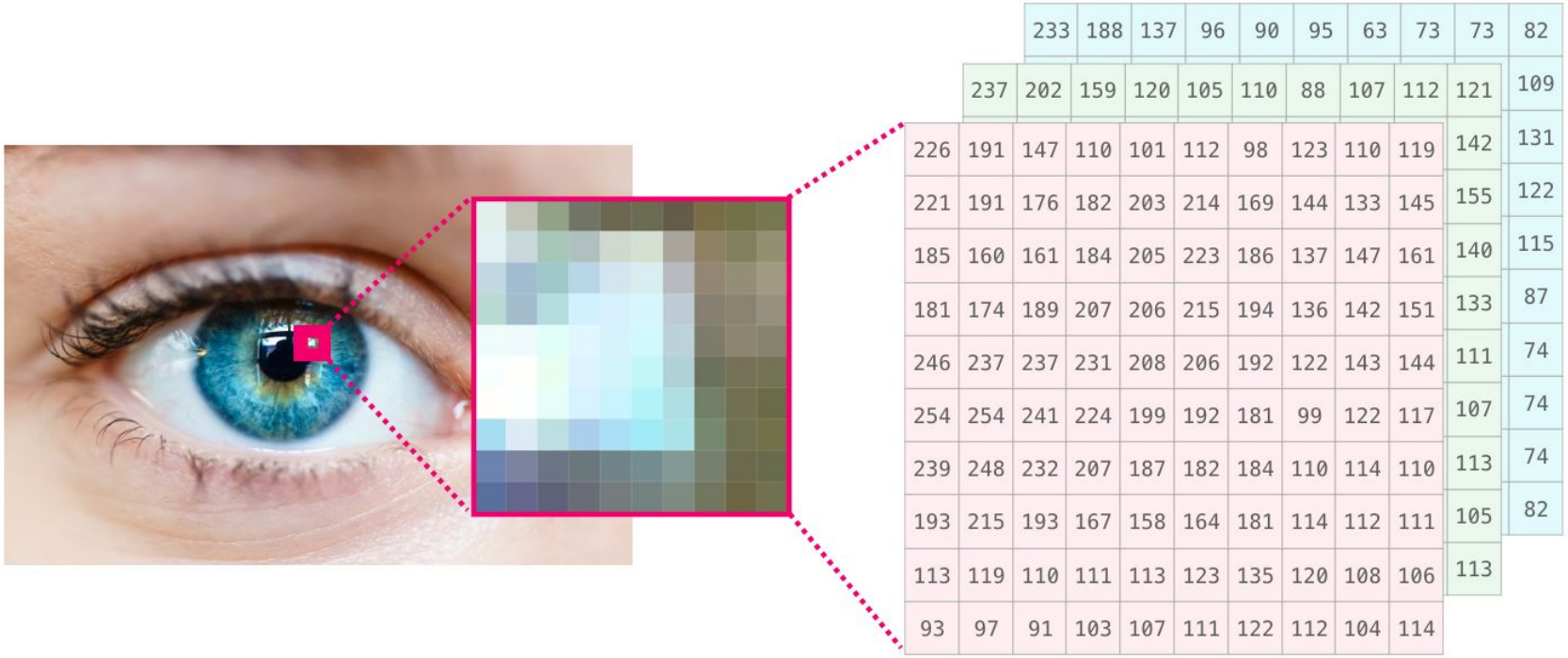
Изображение - это матрица пикселей размером (высота x ширина).

Если изображение черно-белое (также известное как оттенки серого), каждый пиксель может быть представлен одним числом (обычно от 0 (черный) до 255 (белый)).

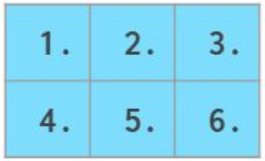


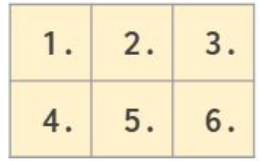
230	194	147	108	90	98	84	96	91	101
237	206	188	195	207	213	163	123	116	128
210	183	180	205	224	234	188	122	134	147
198	189	201	227	229	232	200	125	127	135
249	241	237	244	232	226	202	116	125	126
251	254	241	239	230	217	196	102	103	99
243	255	240	231	227	214	203	116	95	91
204	231	208	200	207	201	200	121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

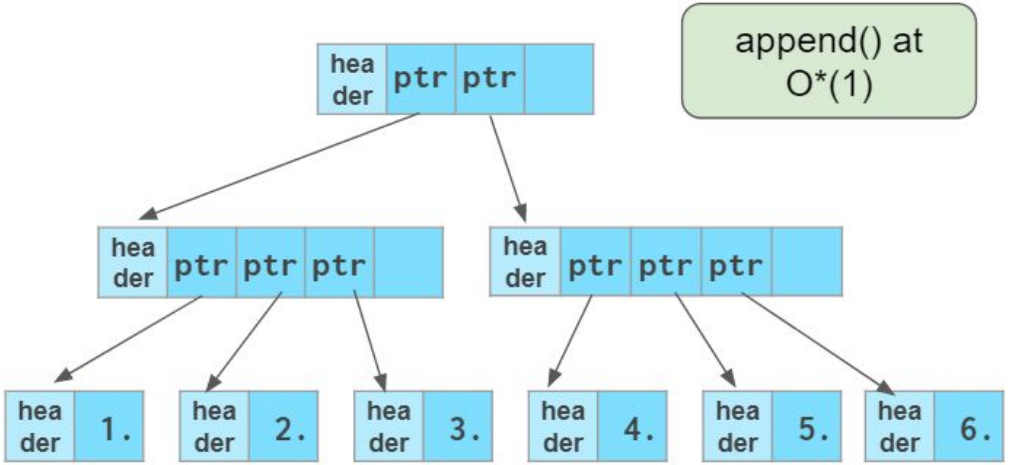
Если изображение цветное, то каждый пиксель представлен тремя числами - значениями для каждого из красного (R), зеленого (G) и синего (B). В этом случае нам нужно 3-е измерение (потому что каждая ячейка может содержать только одно число).



python list vs numpy array







append() at O*(1)

append() at O(N)

Источник

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

Создание массивов из списков

Мы можем создать массив NumPy (он же ndarray), передав ему список Python и используя **np.array()**. В этом случае Python создает массив, который мы видим здесь справа:

Содержит элементы одного типа
(если int и float, то все приводится к float, т. е. повышающее приведение типов)

Самостоятельно определил тип входных данных (int)

```
>>> import numpy as np

>>> ax = np.array([1, 2, 3, 4])
>>> ay = np.array([5, 6, 7, 8])

>>> ax + ay
```

Список Python

тип данных ndarray

Можем сделать list

```
>>> import numpy as np

>>> ax = np.array([1, 2, 3, 4], dtype='float32')
>>> ay = np.array([5, 6, 7, 8])

>>> ax + ay
```

Явным образом задали тип данных

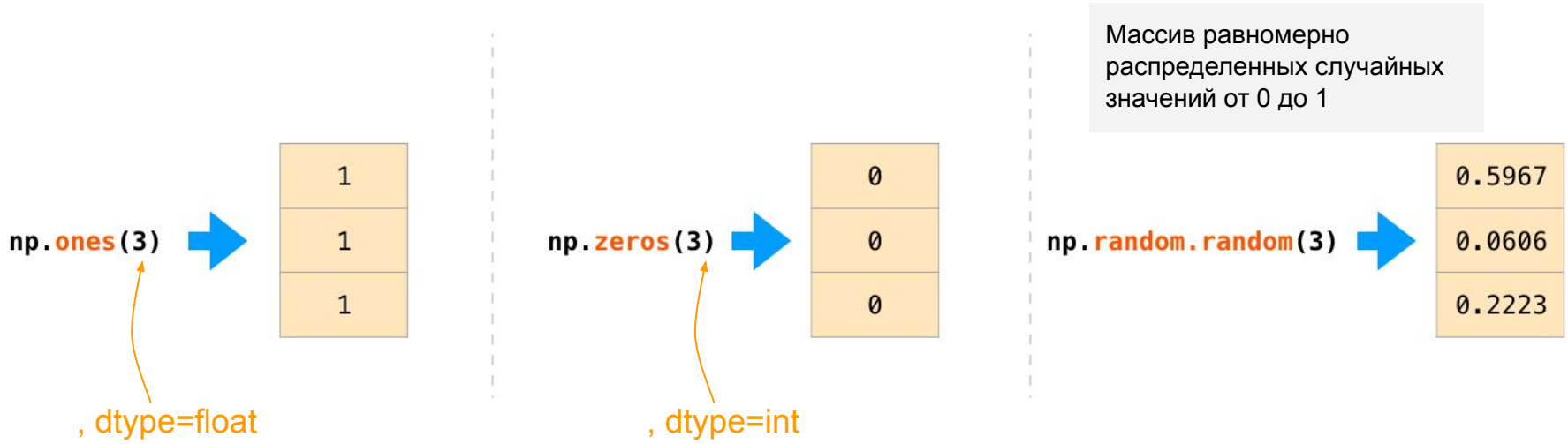
Изначально массив целых чисел

Преобразование типов

```
>>> arr = np.array([1, 2, 3, 4, 5])
>>> arr.dtype

>>> float_arr = arr.astype(np.float64)
>>> float_arr.dtype
```

Преобразование из float в int отбросит дробную чать

Часто бывают случаи, когда мы хотим, чтобы NumPy инициализировал за нас значения массива. NumPy предоставляет такие методы, как **ones()**, **zeros()** и **random.random()** для этих случаев. Мы просто передаем им количество элементов, которое хотим сгенерировать:



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

a

1.	2.	3.

.dtype == np.float64
.shape == (3,)

```
b = np.zeros(3, int)
```

b

0	0	0

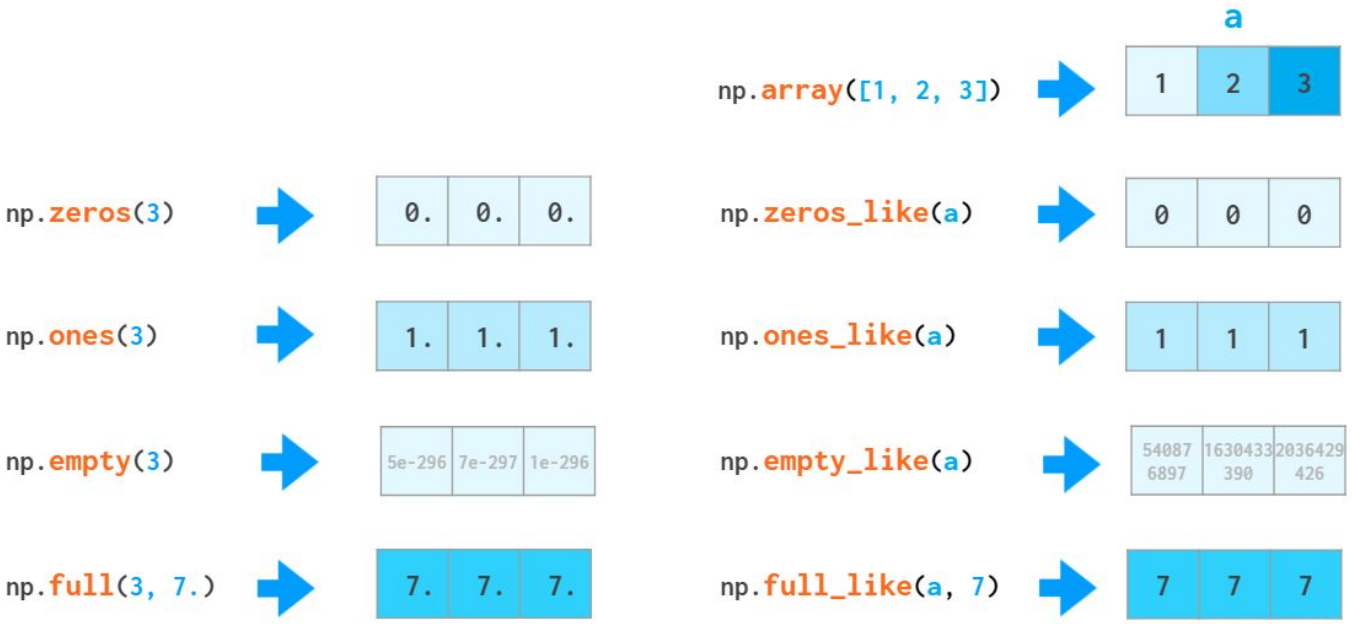
.dtype == np.int32

```
c = np.zeros_like(a)
```

c

0.	0.	0.

.dtype == np.float64
.shape == (3,)


a
np.array([1, 2, 3])
1 2 3
np.zeros(3)
0. 0. 0.
np.zeros_like(a)
0 0 0
np.ones(3)
1. 1. 1.
np.ones_like(a)
1 1 1
np.empty(3)
5e-296 7e-297 1e-296
np.empty_like(a)
540876897 1630433390 2036429426
np.full(3, 7.)
7. 7. 7.
np.full_like(a, 7)
7 7 7

Равномерно распределенные случайные значения от 0 до 1

```
>>> np.random.random(3)

>>> np.random.normal(0, 1, 3)

>>> np.random.randint(0, 10, 3)
```

Нормально распределенные случайные значения с медианой 0 и стандартным отклонением 1

Массив случайных целых чисел в промежутке [0, 10)

`np.random.randint(0, 10, 3)`

uniform, x ∈ [0, 10)

4	3	7

Careful!

`np.random.randint(0, 10)` is [0, 10), but
`random.randint(0, 10)` is [0, 10]

`np.random.rand(3)`

uniform, x ∈ [0, 1)

0.7	0.3	0.8

`np.random.randn(3)`

normal, μ=0, σ=1

0.4	-1.1	0.8

`np.random.uniform(1, 10, 3)`

uniform, x ∈ [1, 10)

5.1	2.7	7.2

`np.random.normal(5, 2, 3)`

normal, μ=5, σ=2

4.5	3.2	6.7

```
>>> import numpy as np
>>>
>>> np.random.seed(0)
>>>
>>> np.random.rand(5)
array([0.5488135 , 0.71518937, 0.60276338, 0.54488318, 0.4236548 ])
>>>
>>> np.random.randint(0, 10, 5)
array([5, 0, 3, 3, 7])
>>>
>>> np.random.beta(2, 3, 5)
array([0.60358975, 0.79868334, 0.31163262, 0.49864896, 0.41528042])
```

```
>>> rng = np.random.RandomState(0)
>>>
>>> rng.rand(5)
array([0.5488135 , 0.71518937, 0.60276338, 0.54488318, 0.4236548 ])
>>>
>>> rng.randint(0, 10, 5)
array([5, 0, 3, 3, 7])
>>>
>>> rng.beta(2, 3, 5)
array([0.60358975, 0.79868334, 0.31163262, 0.49864896, 0.41528042])
```

Позволяет создавать несколько случайных генераторов

```
rng = np.random.default_rng()
```

```
rng.integers(0, 10, 3)
```

uniform, x ∈ [0, 10)

4	3	7

```
rng.integers(0, 10, 3, endpoint=True)
```

uniform, x ∈ [0, 10]

```
rng.random(3)
```

uniform, x ∈ [0, 1)

0.7	0.3	0.8

```
rng.standard_normal(3)
```

normal, μ=0, σ=1

0.4	-1.1	0.8

```
rng.uniform(1, 10, 3)
```

uniform, x ∈ [1, 10)

5.1	2.7	7.2

```
rng.normal(5, 2, 3)
```

normal, μ=5, σ=2

4.5	3.2	6.7

Функция	Описание
`seed`	Задает начальное значение генератора случайных чисел
`permutation`	Возвращает случайную перестановку последовательности или диапазона
`shuffle`	Случайным образом переставляет последовательность на месте
`rand`	Случайная выборка с равномерным распределением
`randint`	Случайная выборка целого числа из заданного диапазона
`randn`	Случайная выборка с нормальным распределением со средним 0 и стандартным отклонением 1 (интерфейс похож на MATLAB)
`binomial`	Случайная выборка с биномиальным распределением
`normal`	Случайная выборка с нормальным (гауссовым) распределением
`beta`	Случайная выборка с бета-распределением
`chisquare`	Случайная выборка с распределением хи-квадрат
`gamma`	Случайная выборка с гамма-распределением
`uniform`	Случайная выборка с равномерным распределением на полуинтервале [0, 1)

Массив из трех элементов, заполненный числами 3.14

```
>>> a = np.full(3, 3.14)

>>> b = np.arange(10, 30, 5)

>>> c = np.arange(10.5, 30.5, 0.5)

>>> c

>>> list(c)
```

Аналог функции **range**, но с произвольным шагом

Построение графика

```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn; seaborn.set()  # стиль графика

import numpy as np

x_coords = np.arange(10.5, 30.5, 0.5)
y_coords = x_coords

# Построить график.
plt.plot(x_coords, y_coords)

# Показать график.
plt.show()
```

Когда **arange** используется с аргументами с плавающей запятой, обычно невозможно предсказать количество полученных элементов из-за конечной точности с плавающей запятой. По этой причине обычно лучше использовать функцию, **linspace** которая получает в качестве аргумента количество элементов, которые мы хотим, вместо шага:

```
>>> from numpy import pi
>>> np.linspace(0, 2, 9)
```

Массив из 9 значений равномерно располагающихся между 0 и 2

```
np.arange(start, stop, step)
```

stop

```
np.arange(6)
```

0	1	2	3	4	5

start,stop

```
np.arange(2, 6)
```

2	3	4	5

start,stop,step

```
np.arange(1, 6, 2)
```

1	3	5

```
np.linspace(start, stop, num) → np.linspace(0, 0.5, 6)
```

0	0.1	0.2	0.3	0.4	0.5

- norm: `start stop step`
  `np.arange(0.4, 0.8, 0.1)` → | 0.4 | 0.5 | 0.6 | 0.7 |
- anomaly: `np.arange(0.5, 0.8, 0.1)` → | 0.5 | 0.6 | 0.7 | 0.8 |
- solution1: `np.arange(0.5, 0.75, 0.1)` → | 0.5 | 0.6 | 0.7 |
- solution2: `start stop num`
  `np.linspace(0.5, 0.7, 3)` → | 0.5 | 0.6 | 0.7 |
- gotcha: `np.linspace(0, 1, ?)` → | 0. | 0.1 | 0.2 | 0.3 | ... | 1. |

Построение графика

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn; seaborn.set()

x_coords = np.linspace(0, 2 * np.pi, 100)
y_coords = np.sin(x_coords)

plt.plot(x_coords, y_coords)

plt.show()
```

https://dfedorov.spb.ru