

массивы
numpy

Массивы в Python

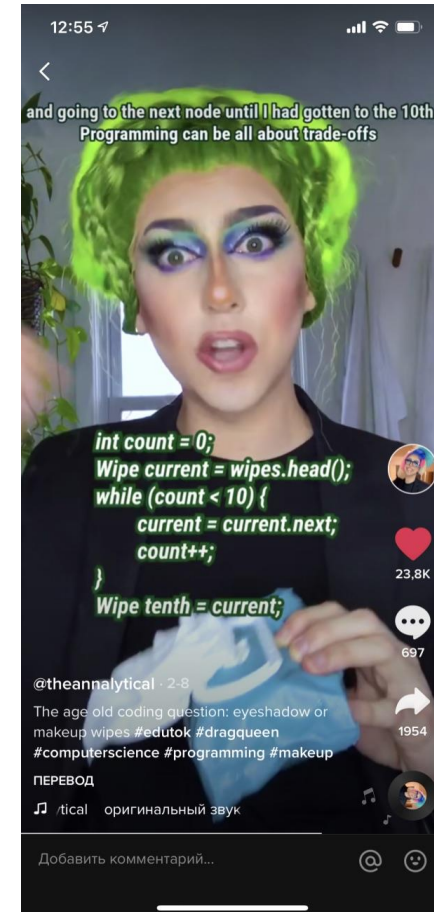
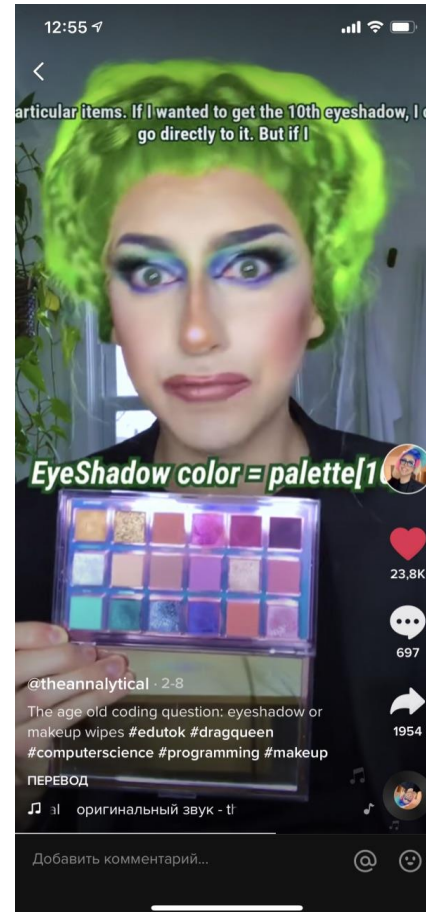
✓ otus.ru › nest › post ▾

[Массивы в Python: как создавать, формат и базовые ...](#)

30 дек. 2018 г. — **Массивы** в Python представляют собой **список** элементов. Значения указываются внутри квадратных скобок, где перечисляются через ...

?!!!

Массивы vs списки




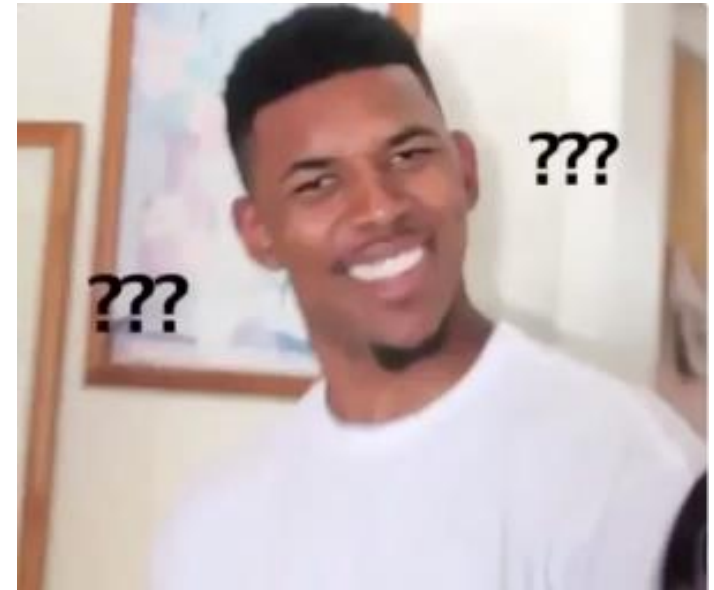
Массивы vs списки

- Массивы – проиндексированная область в памяти
 - быстрый доступ к элементу
 - удобно ассоциируется с элементами лин. алгебры (вектор, матрица, тензор)
 - фикс. размер (м.б. ограничение на размер)
 - может дольше итерироваться
 - все элементы, скорее всего, одного типа данных
- Списки – стр. данных “элемент->хвост (тоже список)”
 - удобно и быстро итерируются
 - динамический размер
 - элементы м.б. разных типов
 - требует больше памяти
 - медленный доступ к i -ому элементу

Массивы в Python

<https://wiki.python.org/moin/TimeComplexity>

Operation	Average Case	 Amortized Worst Case
Copy	$O(n)$	$O(n)$
Append[1]	$O(1)$	$O(1)$
Pop last	$O(1)$	$O(1)$
Pop intermediate[2]	$O(n)$	$O(n)$
Insert	$O(n)$	$O(n)$
Get Item	$O(1)$	$O(1)$
Set Item	$O(1)$	$O(1)$



Массивы в Python

Видны уши C

How are **lists** implemented in CPython?

CPython's **lists** are really variable-length arrays, not Lisp-style linked **lists**. The implementation uses a contiguous array of references to other objects, and keeps a pointer to this array and the array's length in a list head structure.

This makes indexing a list `a[i]` an operation whose cost is independent of the size of the list or the value of the index.

When items are appended or inserted, the array of references is resized. Some cleverness is applied to improve the performance of appending items repeatedly; when the array must be grown, some extra space is allocated so the next few times don't require an actual resize.

Массивы в Python

Python не создавался как язык для вычислений

В 2005 numpy вырос из библиотек Numeric и её дочки NumArray

Для NumPy и SciPy есть хорошая документация

Python Then



Python Now

- Bro how do i fix this - Try hello world



Чем это закончилось



Массивы numpy

- используют иную реализацию на C, за счёт чего быстрее
- имеют vectorize-синтаксис, позволяющий не писать циклы вручную
- часть операций имеет встроенные “распараллеливатели”, что позволяет не сталкиваться с GIL

just do “import numpy as np”

& do not “from numpy import *”

Массивы numpy

```
import time
import numpy as np

py_list = [i for i in range(10000)]
start = time.process_time()
py_list_2 = [i+1 for i in py_list]
end = time.process_time()
print("list: ", end-start)

py_arr = np.array(py_list)
start = time.process_time()
py_arr += 1
end = time.process_time()

print("NumPy array:", round(end-start, 5))
```

Скорость выше в десятки-сотни раз в зависимости от размера массива

Проверьте время работы скалярного произведения двух списков py_list

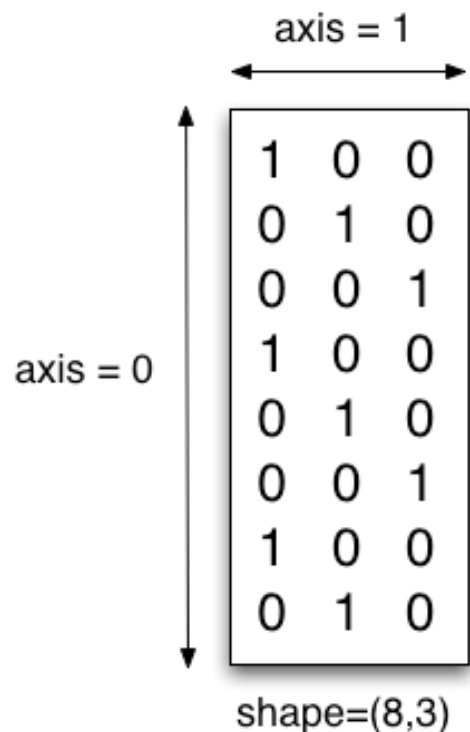
с помощью np.dot(arr1, arr2)

3 человека, скиньте в чат скриншот

Массивы numpy быстрее в

- арифметических операциях над массивами (матрицами)
- поэлементных операциях
- создаётся и удаляется из памяти
- **конкатенация медленная**
- **изменение размера -> нужно создавать новый объект**

Анатомия массива



Linear algebra is
really cool!

= [rows, columns]

элементы должны
быть одного dtype

при изменении dtype
используются float

массивы созданные из
в со-
ответствии типом
будут
названы до-
полнительного" типа

**WARNING: 1-DIMENSIONAL ARRAYS WORK
DIFFERENTLY**

BE CAREFUL WHEN USING AXES WITH 1-D ARRAYS

Массивы numpy команды

- `np.array(object)`
- `arr[i,j]`, `arr[i1:i2,j]`, `arr[i,:]`
- `np.ones((N,M))` / `np.zeros((N,M))` / `np.eye(N)`
- `np.arange(start, stop, step)`, `np.linspace(start, stop, num)`
- `array.shape()`
- `np.sum`

работа с осями и конкатенация:

<https://www.sharpsightlabs.com/blog/numpy-axes-explained/>

мат. операции и линейная алгебра:

https://numpy.org/devdocs/user/absolute_beginners.html

https://pyprog.pro/basic_operations.html

Векторизация

Избавьтесь от циклов в функции с помощью
`np.vectorize(function)`

см. ноутбук

Broadcasting

$$\mathbf{V} * \mathbf{V.T} = \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 2 & 2 & 2 \\ \hline 3 & 3 & 3 \\ \hline \end{array} * \begin{array}{|c|c|c|} \hline 1 & 2 & 3 \\ \hline 1 & 2 & 3 \\ \hline 1 & 2 & 3 \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline 1 & 2 & 3 \\ \hline 2 & 4 & 6 \\ \hline 3 & 6 & 9 \\ \hline \end{array}$$

broadcasting позволяет избегать циклов, быстро “размножая” тензор по отсутствующей оси. Это может ускорить вычисления при больших размерах матрицы

<https://towardsdatascience.com/performing-multidimensional-matrix-operations-using-numpys-broadcasting-cf33e3029170>