Practise 2

- env
- packages
- numpy

Németh Ernő Nándor

Virtual Environment

- You can create a virtual environment like running OS on VM.
- You can:
  - separate code/package versions
  - publish code with requirement.txt easier
  - reset the environment, if something goes wrong,
  - try other package versions

https://www.geeksforgeeks.org/python-virtual-environment/

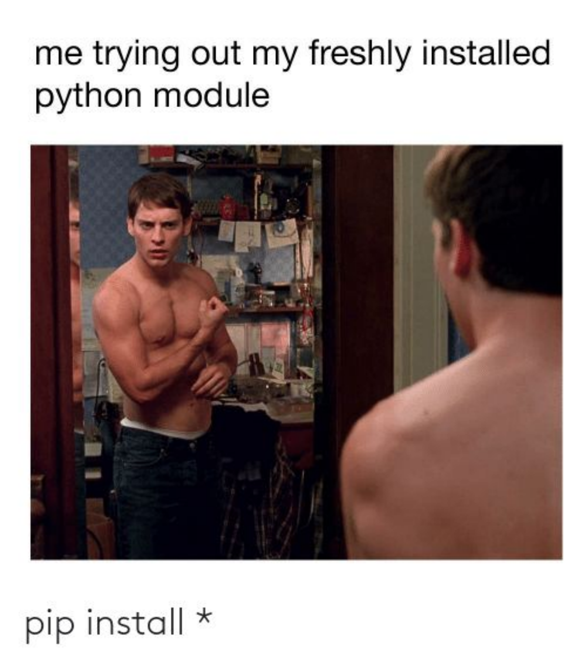
Packages

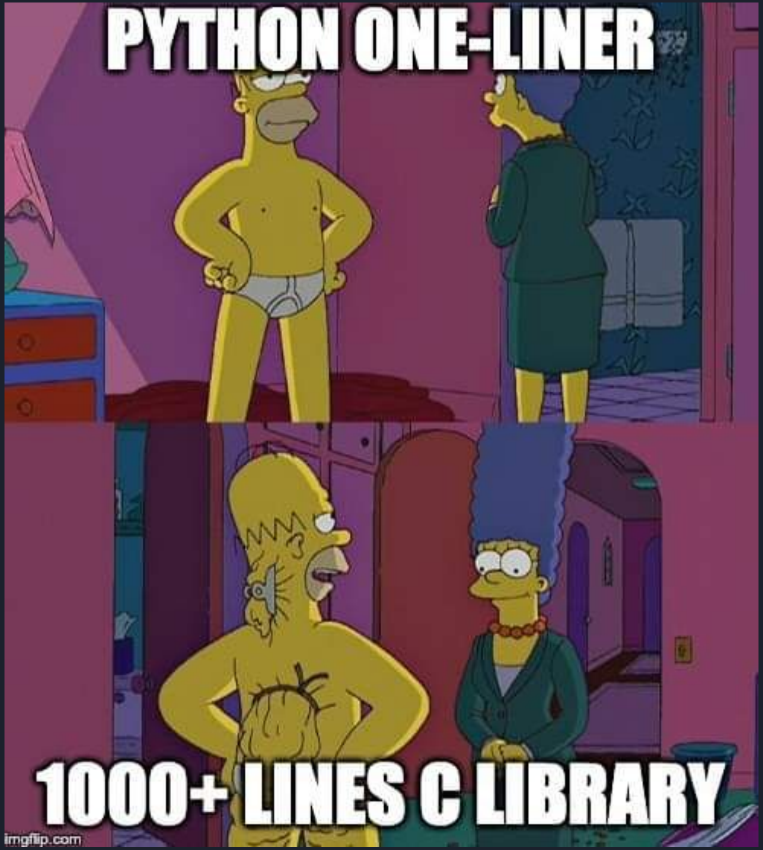
Packages

- from [path] import [function/class/*]

PIP

- Pip - Pip Package Installer o Pip Python Installer
- pip install [name]
- pip uninstall [name]
- pip list




PYTHON ONE-LINER
1000+ LINES C LIBRARY
imgflip.com

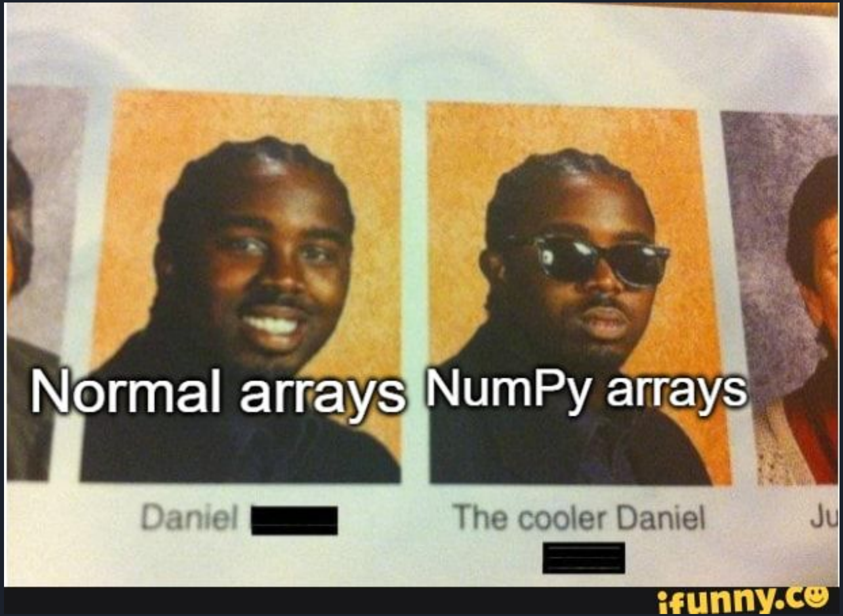
when you try to install
a new library in python
but it decides to upgrade tensorflow,
keras, numpy, pandas, etc ruining
your already stable project
MEMES
ifunny.co

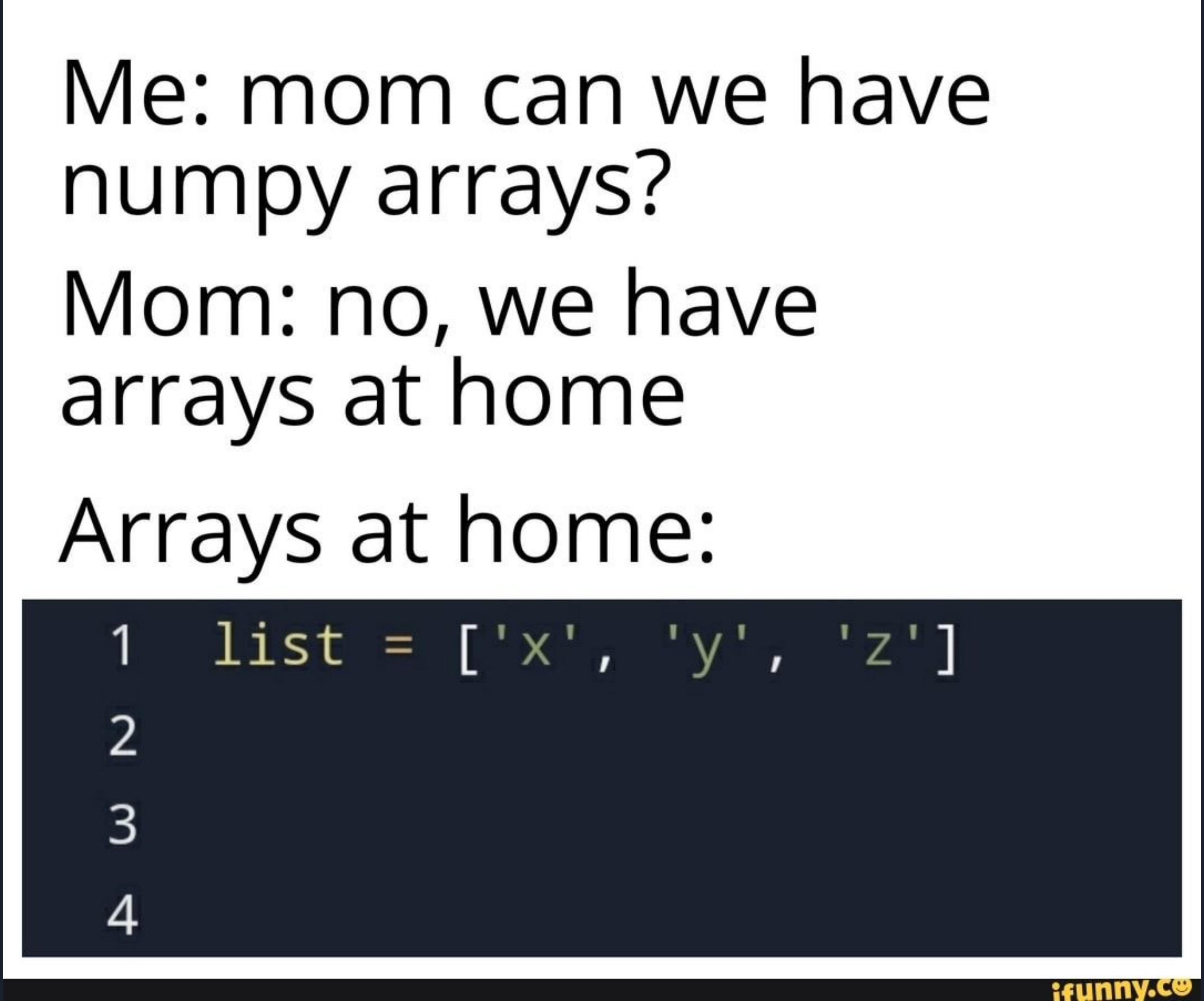
Numpy

- Stands for Numerical Python.
- NumPy is a Python library used for working with arrays.
- Also good for domain of linear algebra, fourier transform, and matrices

- It is written partially in Python, but most of the parts that require fast computation are written in C or C++

https://github.com/numpy/numpy





Me: mom can we have numpy arrays?

Mom: no, we have arrays at home

Arrays at home:

```
list = ['x', 'y', 'z']



```

How to use?

- In terminal: pip install numpy
- import numpy / import numpy as np
- `print(np.__version__)`

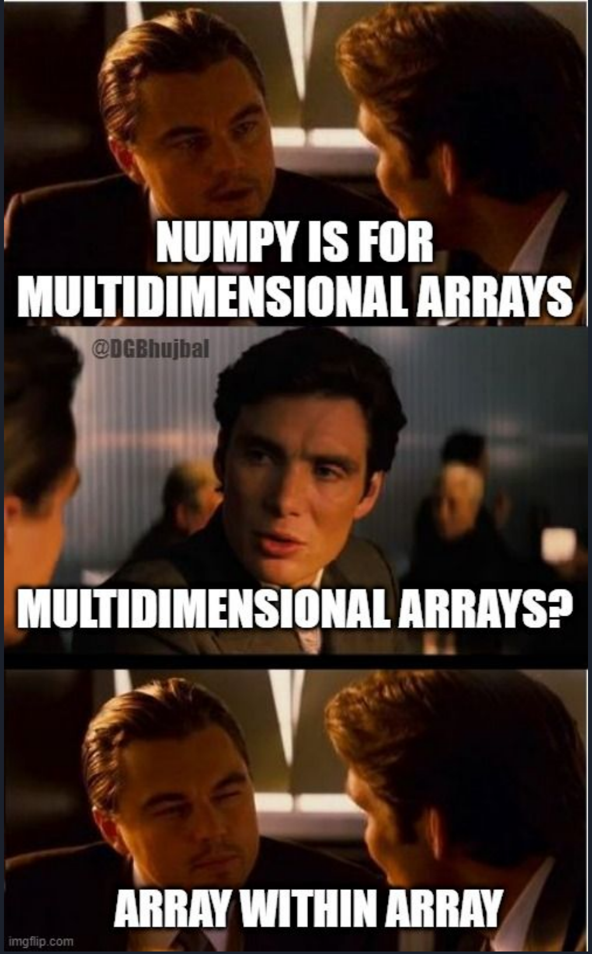
```
import numpy
```

```
import numpy as np
```

when someone asks you why do you call numpy as np

Create arrays

```
0D - arr = np.array(42)

1D - arr = np.array([1, 2, 3, 4, 5])

2D - arr = np.array([[1, 2, 3], [4, 5, 6]])

3D - arr = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

ND - arr = np.array([1, 2, 3, 4], ndmin=N)

GET ARRAY DIMENSION

print(a.ndim)
```


NUMPY IS FOR MULTIDIMENSIONAL ARRAYS
@DGBhujbal
MULTIDIMENSIONAL ARRAYS?
ARRAY WITHIN ARRAY
imgflip.com

Indexing

```
Positive:

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

print(arr[0])
```

```
Negative:

arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])

print('Last element from 2nd dim: ', arr[1, -1])
```

Slicing

```
[start:end:step]

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

print(arr[1:5])

print(arr[4:])

print(arr[-3:-1])

print(arr[-3:-1])
```

Data types

Types in Python:

- string
- integer
- float
- boolean
- complex

Types in numpy:

- i - integer
- b - boolean
- u - unsigned integer
- f - float
- c - complex float
- m - timedelta
- M - datetime
- O - object
- S - string
- U - unicode string
- V - fixed chunk of memory for other type (void)

Data types

Print data type:

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

print(arr.dtype)
```

Change array data type:

```
arr = np.array([1.1, 2.1, 3.1])

newarr = arr.astype(int)
```

Change data type (bool):

```
arr = np.array([1, 0, 3])

newarr = arr.astype(bool)
```

Copy vs View

The main difference between a copy and a view of an array is that the copy is a new array, and the view is just a view of the original array.

The **COPY** owns the data and any changes made to the copy will not affect the original array, and any changes made to the original array will not affect the copy.

The **VIEW** does not own the data and any changes made to the view will affect the original array, and any changes made to the original array will affect the view.

Copy vs View

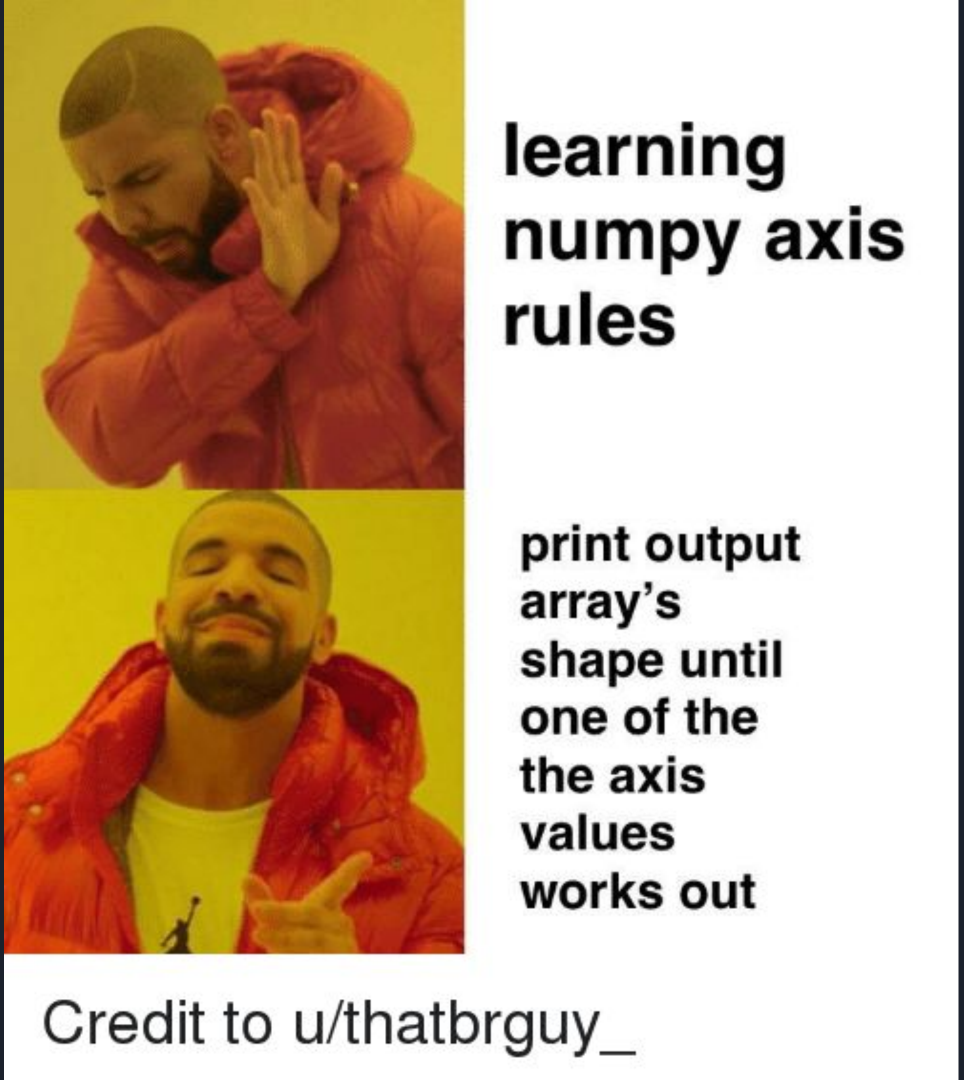
```
Copy:

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

x = arr.copy()

arr[0] = 42


print(arr)

print(x)
```

```
View:

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

x = arr.view()

arr[0] = 42


print(arr)

print(x)
```

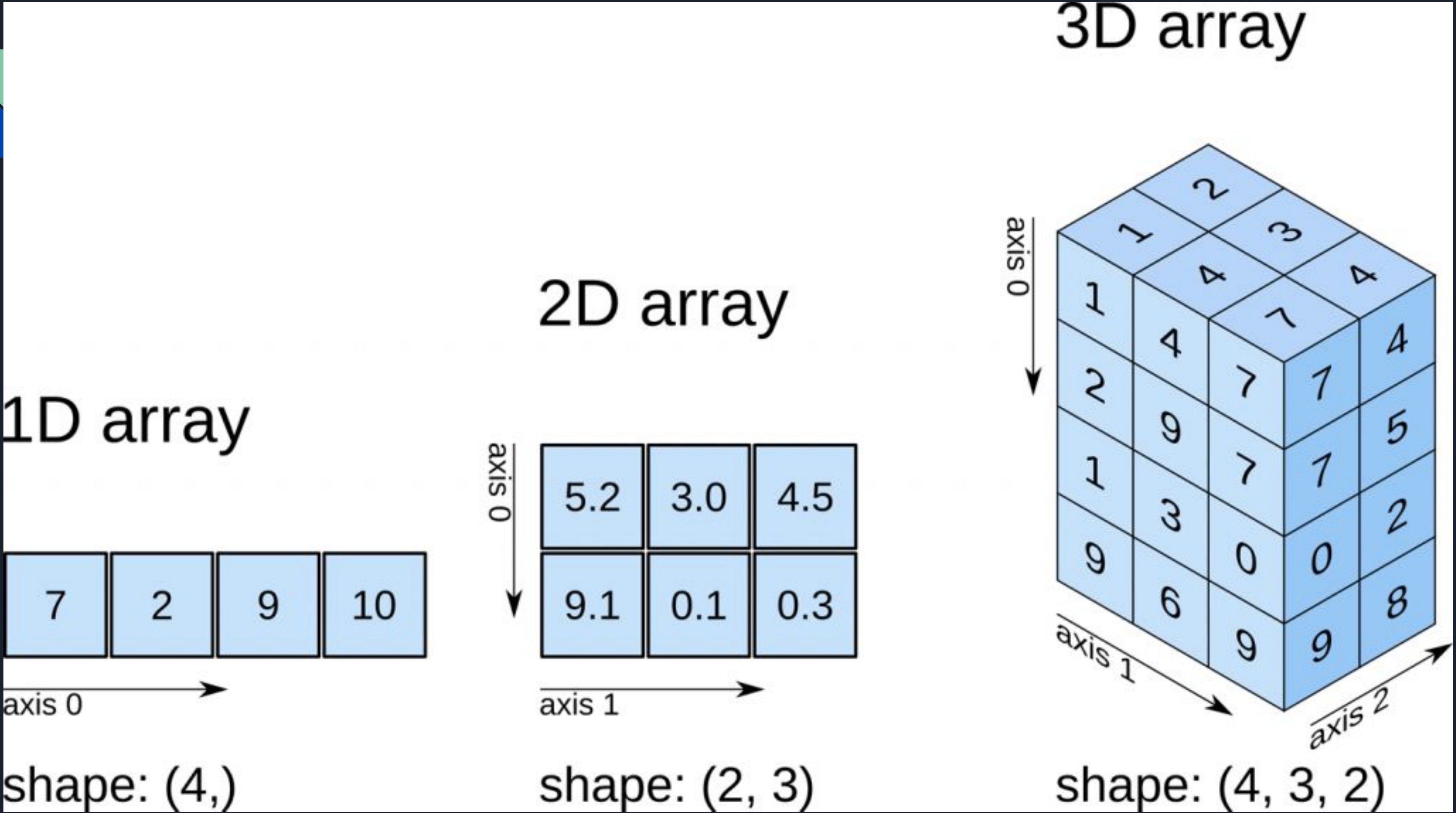
Shape

Get shape of an array:

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

print(arr.shape)
```


learning numpy axis rules
print output array's shape until one of the the axis values works out

Credit to u/thatbrguy_


1D array
7
2
9
10
axis 0
shape: (4,)
2D array
axis 0
5.2
3.0
4.5
9.1
0.1
0.3
axis 1
shape: (2, 3)
3D array
axis 0
1
2
4
3
7
4
1
4
7
2
9
7
1
3
0
9
6
9
7
4
7
5
0
2
9
8
axis 1
axis 2
shape: (4, 3, 2)

Reshape

1D -> 2D:

```
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8,
9, 10, 11, 12])

newarr = arr.reshape(4, 3)
```

1D -> 3D (Unknown dimension):

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

newarr = arr.reshape(2, 2, -1)
```

```
Flattening:

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

newarr = arr.reshape(-1)
```

Iterating

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

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

#arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])


#FIND THE DIFFERENCE!!!

for x in arr:

  print(x)
```

Stacking

```
arr1 = np.array([1, 2, 3])

arr2 = np.array([4, 5, 6])
```

```
arr = np.concatenate((arr1, arr2))

arr = np.concatenate((arr1, arr2),
axis=1)

arr = np.stack((arr1, arr2), axis=1)

arr = np.hstack((arr1, arr2))
```

```
arr = np.vstack((arr1, arr2))

arr = np.dstack((arr1, arr2))
```

Split

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

newarr = np.array_split(arr, 4)
```

Sort

```
arr = np.array([3, 2, 0, 1])

#arr = np.array([[3, 2, 4], [5, 0, 1]])

print(np.sort(arr))
```

Search

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

x = np.where(arr == 4)

x = np.where(arr%2 == 0)
```

```
In sorted array:

arr = np.array([6, 7, 8, 9])

x = np.searchsorted(arr, 7)
```

```
From right side:

arr = np.array([6, 7, 8, 9])

x = np.searchsorted(arr, 7,
side='right')
```

```
Multiple values:

arr = np.array([1, 3, 5, 7])

x = np.searchsorted(arr, [2, 4, 6])
```

Filter

```
arr = np.array([41, 42, 43, 44])

x = [True, False, True, False]

newarr = arr[x]

print(newarr)
```

TASK:

Write a code, which filters the even numbers from the array.

Use this array:

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

Filter - Result

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

filter_arr = arr % 2 == 0

newarr = arr[filter_arr]

print(filter_arr)

print(newarr)
```

Numpy - Random

```
from numpy import random

x = random.randint(100) #int

x = random.randint(100, size=(5)) #rnd array

x = random.rand() #float

x = random.rand(3, 5)

print(x)

x = random.choice([3, 5, 7, 9])
```

What is the difference between random.rand() and random.rand(1)?

Numpy - ufuncs, Vectorization

ufuncs stands for "Universal Functions" and they are NumPy functions that operate on the ndarray object.

What is Vectorization?

Converting iterative statements into a vector based operation is called vectorization.

```
x = [1, 2, 3, 4]
y = [4, 5, 6, 7]
z = []
#.zip is a python in built func.
for i, j in zip(x, y):
  z.append(i + j)
print(z)
#np.add(x, y) is the same in numpy
```

Create a function

```
def myadd(x, y):
  return x+y
```

```
myadd = np.frompyfunc(myadd, 2, 1)
print(myadd([1, 2, 3, 4], [5, 6, 7, 8]))
```

```
#is ufunc?
print(type(np.add))
```

When you replace a for loop with a vectorized numpy function and see the speed improvement

For practise

https://www.w3schools.com/python/numpy/default.asp