CDS 230
Modeling and Simulation I

Module 7

NumPy

Dr. Hamdi Kavak
http://www.hamdikavak.com
hkavak@gmu.edu

Outline

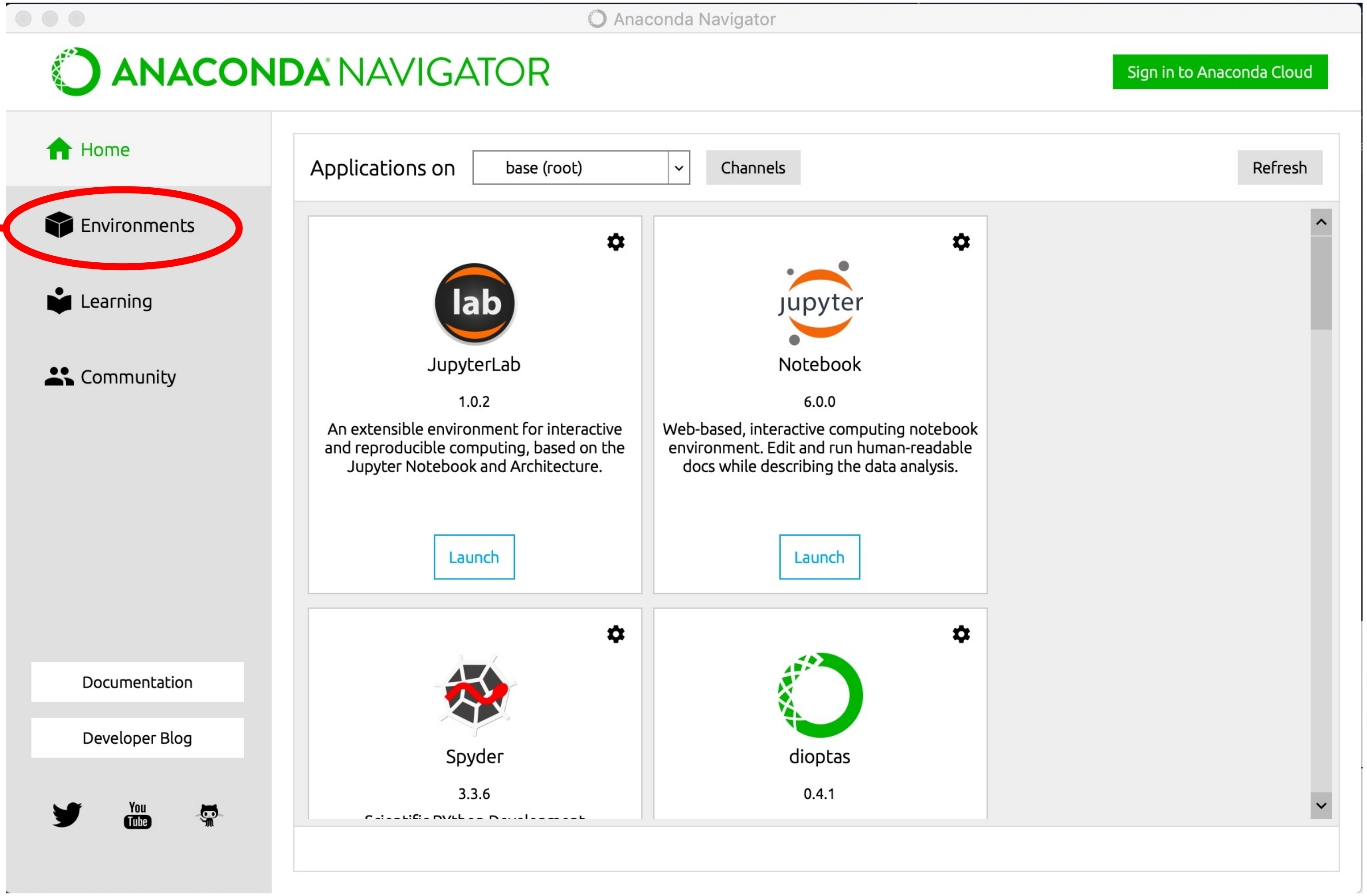
- Installing third-party packages
- NumPy
  - Arrays (creation, initialization, slicing, and random numbers)
  - Matrices (creation, initialization, slicing, and random numbers)
  - Some Linear Algebra concepts
  - Statistics
  - Spaces and ranges

Third party packages

- Python has many great built-in modules and packages
  - Check https://docs.python.org/3/py-modindex.html
- Third party packages extend Python's capabilities
  - NumPy
  - Matplotlib
  - Pandas
  - SciPy
  - scikit-learn
  - …
- When you start a project, better check what open source packages are available.

Anaconda Navigator

Open Anaconda Navigator and click Environments

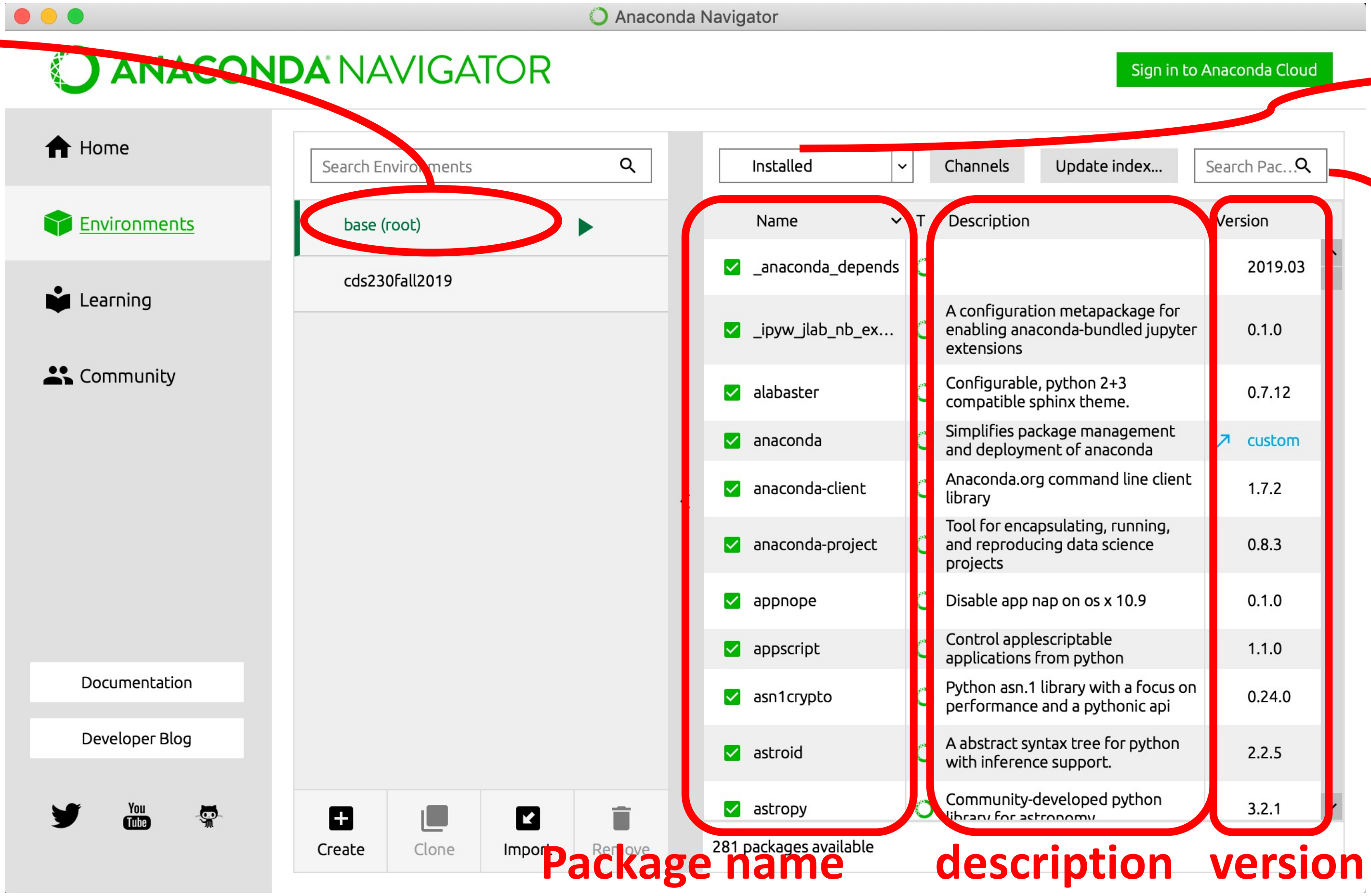


An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

Anaconda Navigator

Make sure base is selected

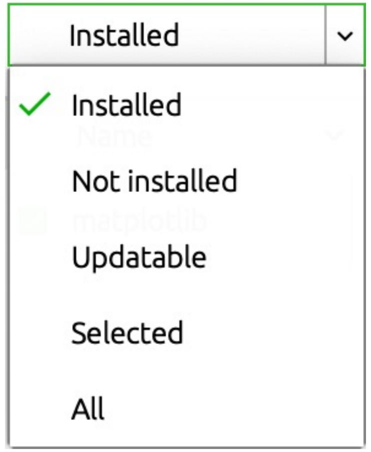


Name	T	Description	Version
☑ _anaconda_depends			2019.03
☑ _ipyw_jlab_nb_ex...		A configuration metapackage for enabling anaconda-bundled jupyter extensions	0.1.0
☑ alabaster		Configurable, python 2+3 compatible sphinx theme.	0.7.12
☑ anaconda		Simplifies package management and deployment of anaconda	↗ custom
☑ anaconda-client		Anaconda.org command line client library	1.7.2
☑ anaconda-project		Tool for encapsulating, running, and reproducing data science projects	0.8.3
☑ appnope		Disable app nap on os x 10.9	0.1.0
☑ appscript		Control applescriptable applications from python	1.1.0
☑ asn1crypto		Python asn.1 library with a focus on performance and a pythonic api	0.24.0
☑ astroid		A abstract syntax tree for python with inference support.	2.2.5
☑ astropy		Community-developed python library for astronomy	3.2.1

281 packages available

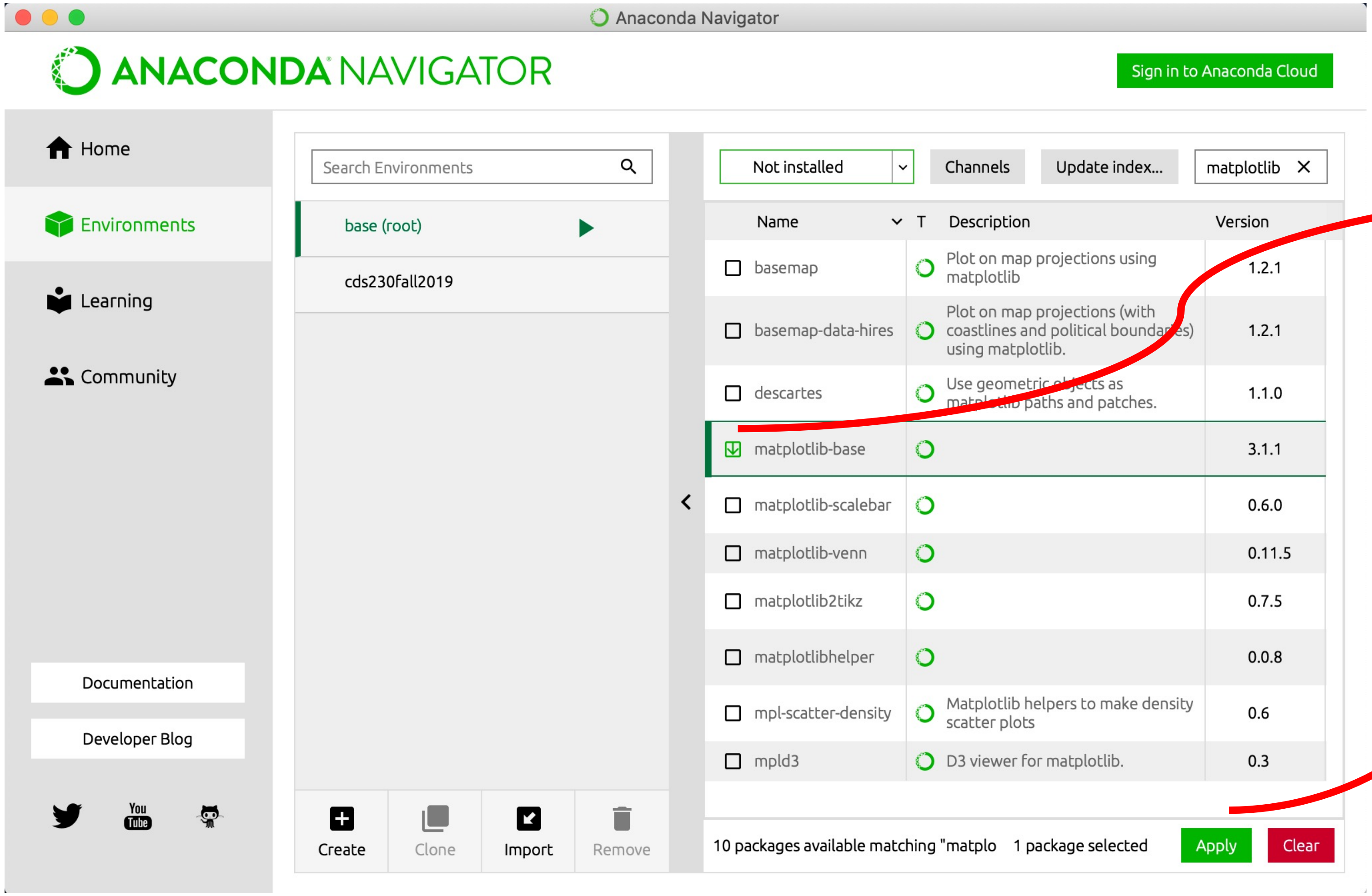
Package name description version

Choose what to display



Search packages

Anaconda Navigator



base (root)

cds230fall2019

Not installed

matplotlib ✕

	Name	T	Description	Version
□	basemap	○	Plot on map projections using matplotlib	1.2.1
□	basemap-data-hires	○	Plot on map projections (with coastlines and political boundaries) using matplotlib.	1.2.1
□	descartes	○	Use geometric objects as matplotlib paths and patches.	1.1.0
☑	matplotlib-base	○		3.1.1
□	matplotlib-scalebar	○		0.6.0
□	matplotlib-venn	○		0.11.5
□	matplotlib2tikz	○		0.7.5
□	matplotlibhelper	○		0.0.8
□	mpl-scatter-density	○	Matplotlib helpers to make density scatter plots	0.6
□	mpld3	○	D3 viewer for matplotlib.	0.3

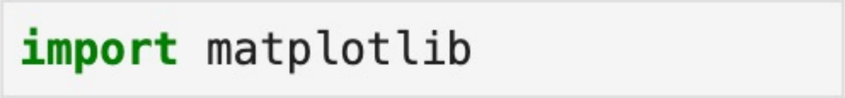
10 packages available matching "matplo 1 package selected

Select the packages you want to install

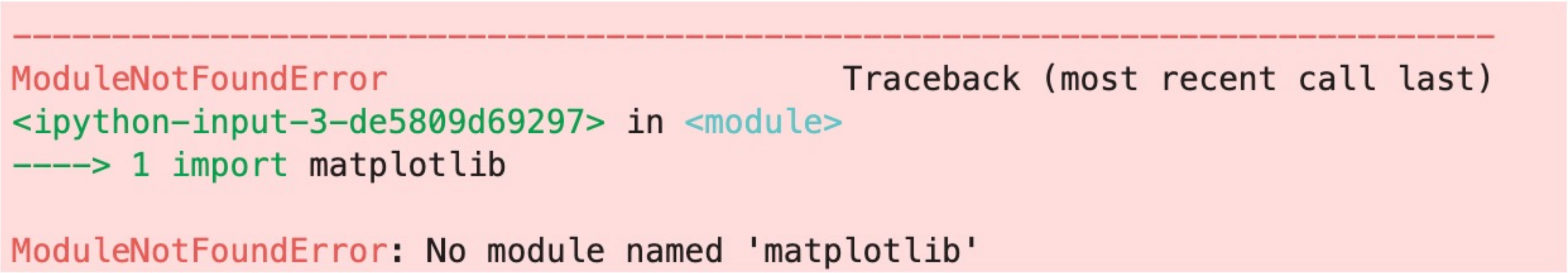
Click Apply

How to know you are missing packages?

- You can list them using `pip` or `conda`.
- List them on Anaconda Navigator.
- Or just try to use them.



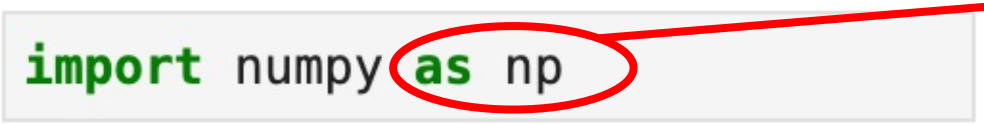
```python
import matplotlib
```



```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-3-de5809d69297> in <module>
----> 1 import matplotlib

ModuleNotFoundError: No module named 'matplotlib'
```

NumPy

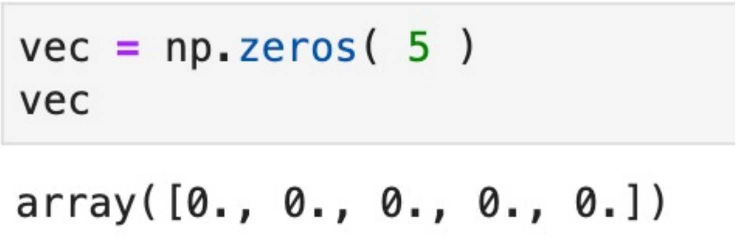
- Provides with an extensive set of mathematical computation capabilities
- Performs computations fast
- Let’s install it first: using command line or Anaconda Navigator
    - Very likely that you have NumPy installed
- How to import?



```
import numpy as np
```

This is optional. Think this like a nickname.

Creating arrays

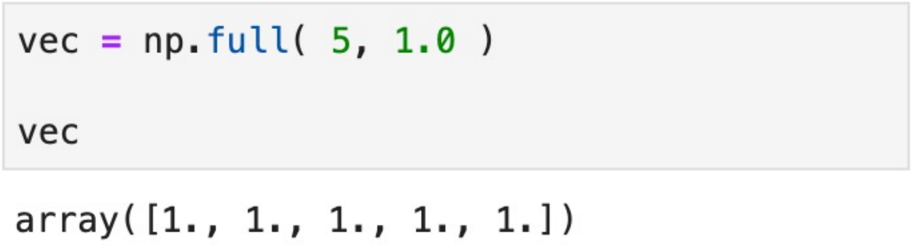


```
vec = np.zeros( 5 )
vec
```

```
array([0., 0., 0., 0., 0.])
```

The **zeros** function creates a vector and each element is 0. The input to the function is the number of elements in the vector.

The **full** function creates a vector of a given length, and all of the elements are what you provide. Note that in both cases, the elements are floating point numbers.

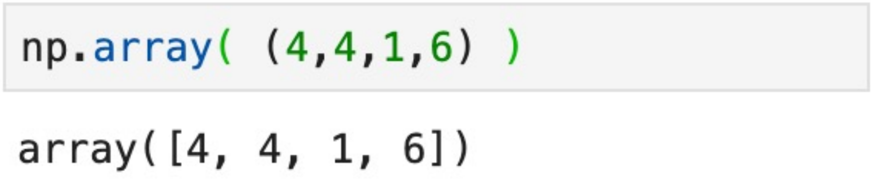


```
vec = np.full( 5, 1.0 )

vec
```

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

NumPy arrays hold the same type of elements. E.g., you can't even mix integers with floats.

Creating arrays with specific values

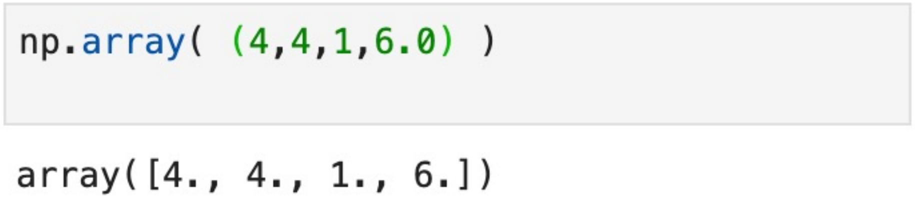


```
np.array( (4,4,1,6) )
```

```
array([4, 4, 1, 6])
```

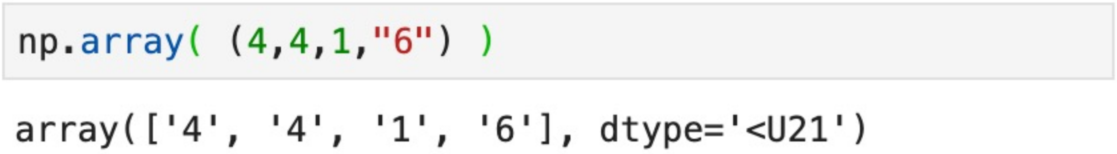
The **array** function receives a tuple or list, and it creates a vector from that input.

Be mindful about the types of your iterable items



```
np.array( (4,4,1,6.0) )
```

```
array([4., 4., 1., 6.])
```

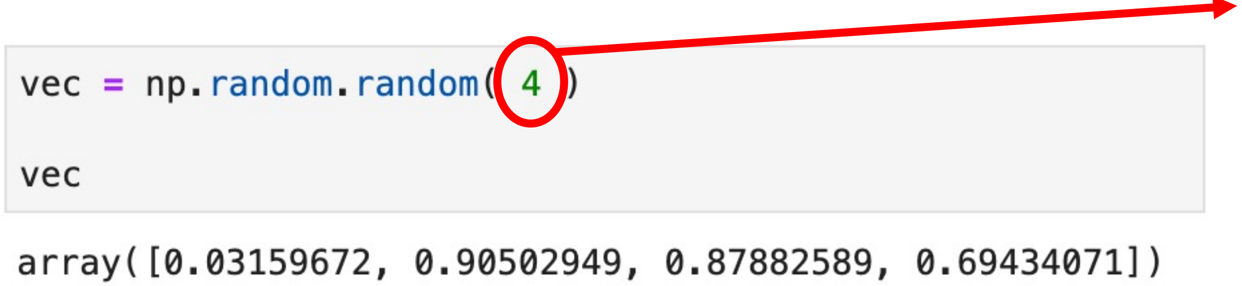


```
np.array( (4,4,1,"6") )
```

```
array(['4', '4', '1', '6'], dtype='<U21')
```

Creating arrays with random values

- The random module within numpy allows generating decimal numbers between 0 and 1



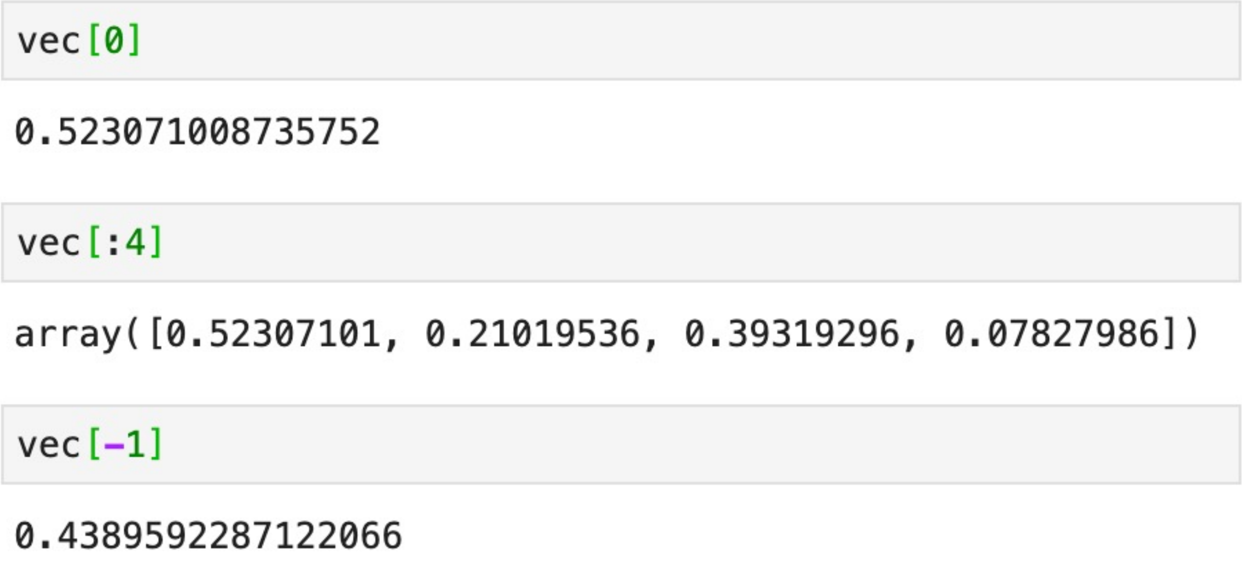
```
vec = np.random.random( 4 )

vec
```

```
array([0.03159672, 0.90502949, 0.87882589, 0.69434071])
```

Number of elements to be generated

Slicing NumPy arrays

- All slicing operations you have learned with lists are the same in NumPy



```
vec[0]
```

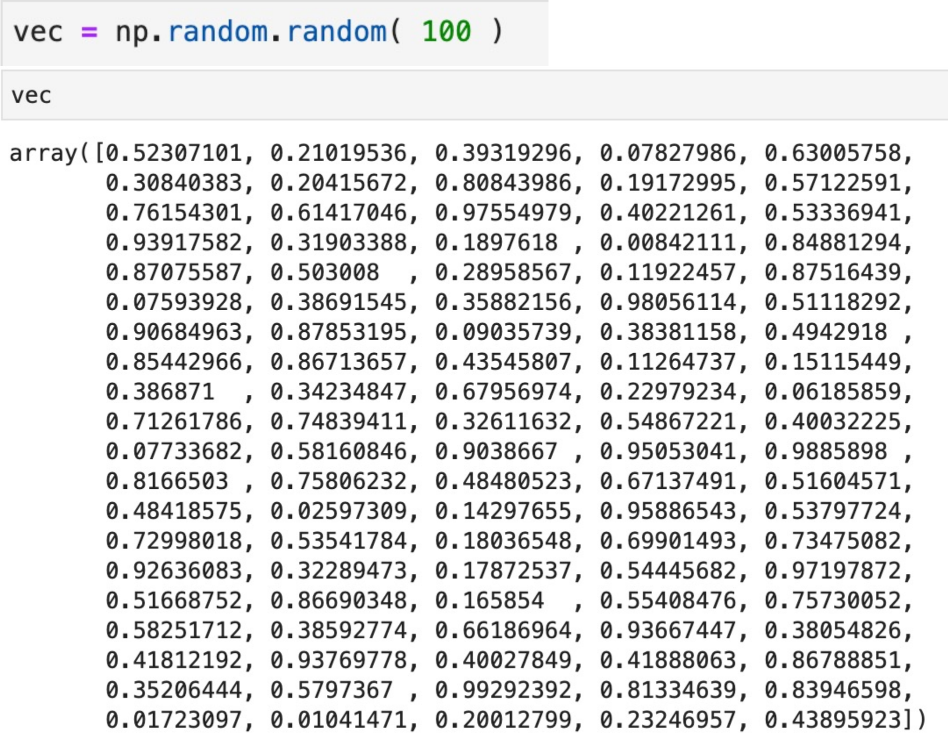
```
0.523071008735752
```

```
vec[:4]
```

```
array([0.52307101, 0.21019536, 0.39319296, 0.07827986])
```

```
vec[-1]
```

```
0.4389592287122066
```



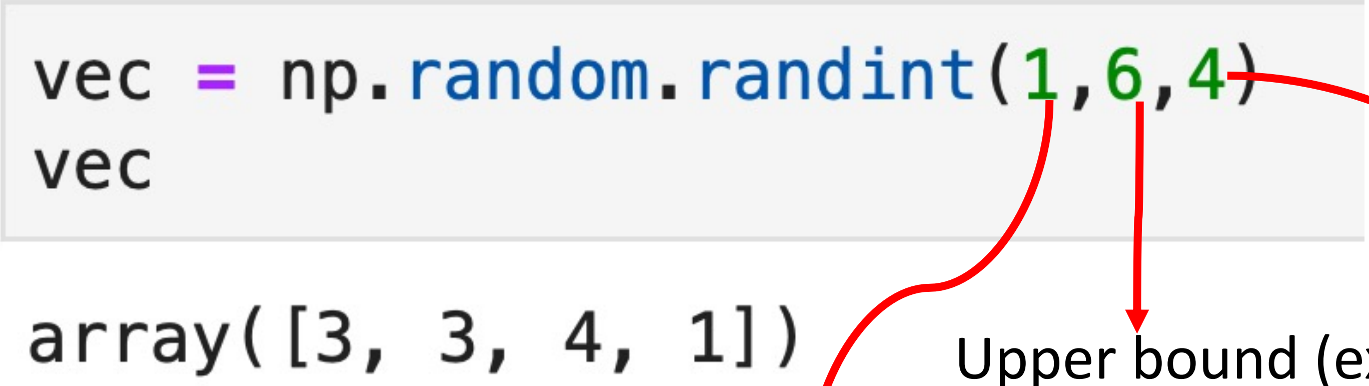
```
vec = np.random.random( 100 )
```

```
vec
```

```
array([0.52307101, 0.21019536, 0.39319296, 0.07827986, 0.63005758,
       0.30840383, 0.20415672, 0.80843986, 0.19172995, 0.57122591,
       0.76154301, 0.61417046, 0.97554979, 0.40221261, 0.53336941,
       0.93917582, 0.31903388, 0.1897618 , 0.00842111, 0.84881294,
       0.87075587, 0.503008  , 0.28958567, 0.11922457, 0.87516439,
       0.07593928, 0.38691545, 0.35882156, 0.98056114, 0.51118292,
       0.90684963, 0.87853195, 0.09035739, 0.38381158, 0.4942918 ,
       0.85442966, 0.86713657, 0.43545807, 0.11264737, 0.15115449,
       0.386871  , 0.34234847, 0.67956974, 0.22979234, 0.06185859,
       0.71261786, 0.74839411, 0.32611632, 0.54867221, 0.40032225,
       0.07733682, 0.58160846, 0.9038667 , 0.95053041, 0.9885898 ,
       0.8166503 , 0.75806232, 0.48480523, 0.67137491, 0.51604571,
       0.48418575, 0.02597309, 0.14297655, 0.95886543, 0.53797724,
       0.72998018, 0.53541784, 0.18036548, 0.69901493, 0.73475082,
       0.92636083, 0.32289473, 0.17872537, 0.54445682, 0.97197872,
       0.51668752, 0.86690348, 0.165854  , 0.55408476, 0.75730052,
       0.58251712, 0.38592774, 0.66186964, 0.93667447, 0.38054826,
       0.41812192, 0.93769778, 0.40027849, 0.41888063, 0.86788851,
       0.35206444, 0.5797367 , 0.99292392, 0.81334639, 0.83946598,
       0.01723097, 0.01041471, 0.20012799, 0.23246957, 0.43895923])
```

Random integers

The `np.random.randint()` function creates random integers.



```
vec = np.random.randint(1,6,4)
vec
```

```
array([3, 3, 4, 1])
```

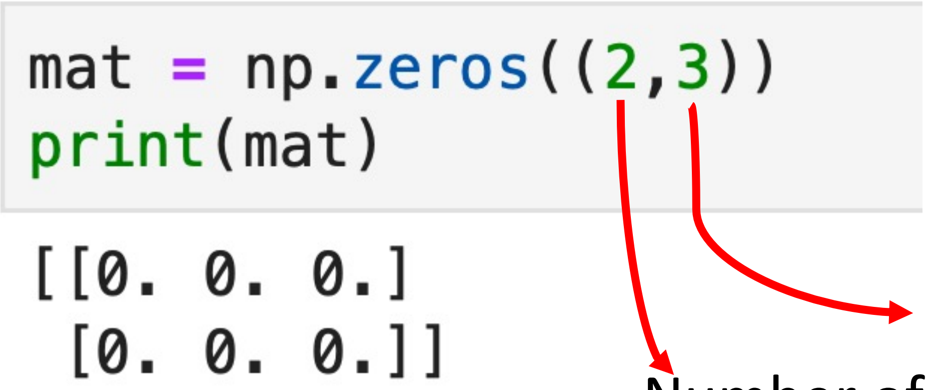
Lower bound (inclusive)

Upper bound (exclusive)

Number of elements to be returned

These numbers will be either 1, 2, 3, 4, or 5.

Matrix



```
mat = np.zeros((2,3))
print(mat)
```

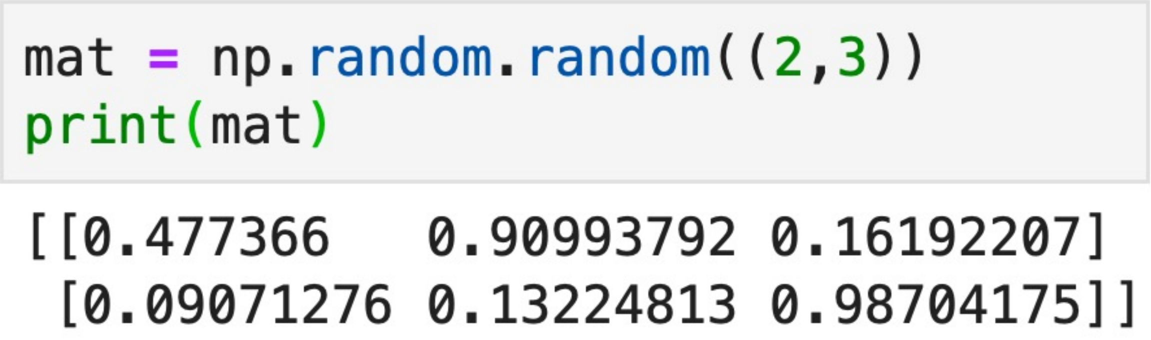
```
[[0. 0. 0.]
 [0. 0. 0.]]
```

To create a matrix, we pass a *tuple* that has the size of the matrix.

Number of columns

Number of rows

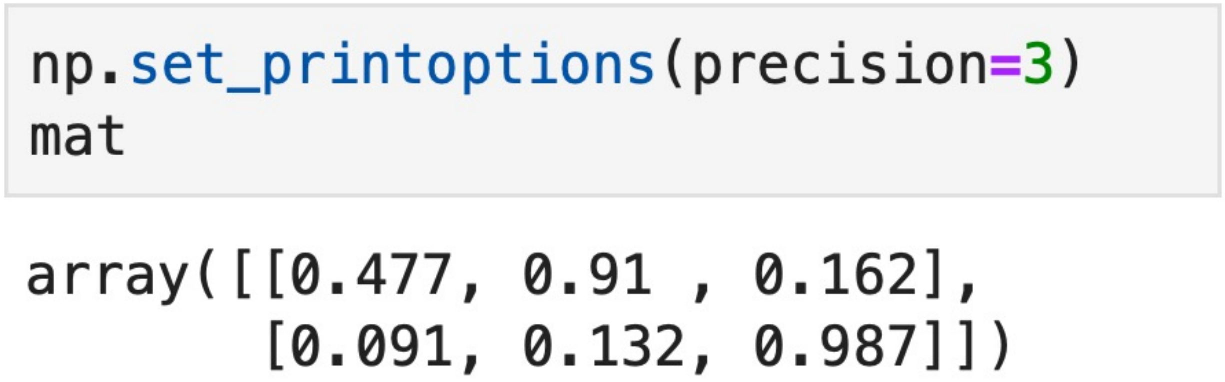
The same size parameter applies to NumPy's random function as well.



```
mat = np.random.random((2,3))
print(mat)
```

```
[[0.477366   0.90993792 0.16192207]
 [0.09071276 0.13224813 0.98704175]]
```

Control printing

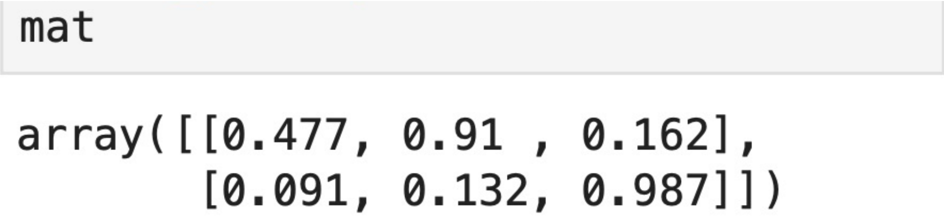


```python
np.set_printoptions(precision=3)
mat
```

```
array([[0.477, 0.91 , 0.162],
       [0.091, 0.132, 0.987]])
```

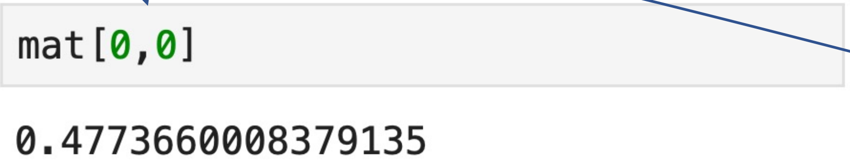
Note: This doesn’t change the actual numbers, just how they’re displayed.

Slicing numpy matrices



```
mat
```

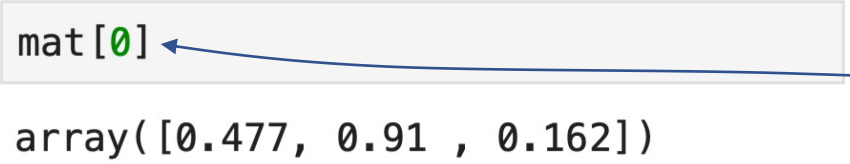
```
array([[0.477, 0.91 , 0.162],
       [0.091, 0.132, 0.987]])
```



```
mat[0,0]
```

```
0.4773660008379135
```

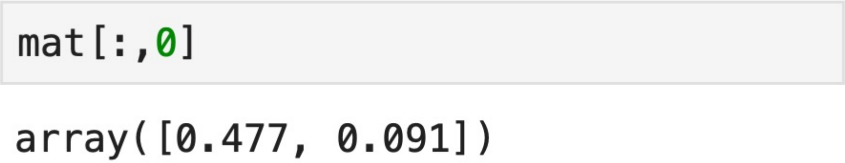
Get the element by [row num, column num].
This is the first element.



```
mat[0]
```

```
array([0.477, 0.91 , 0.162])
```

The first row.

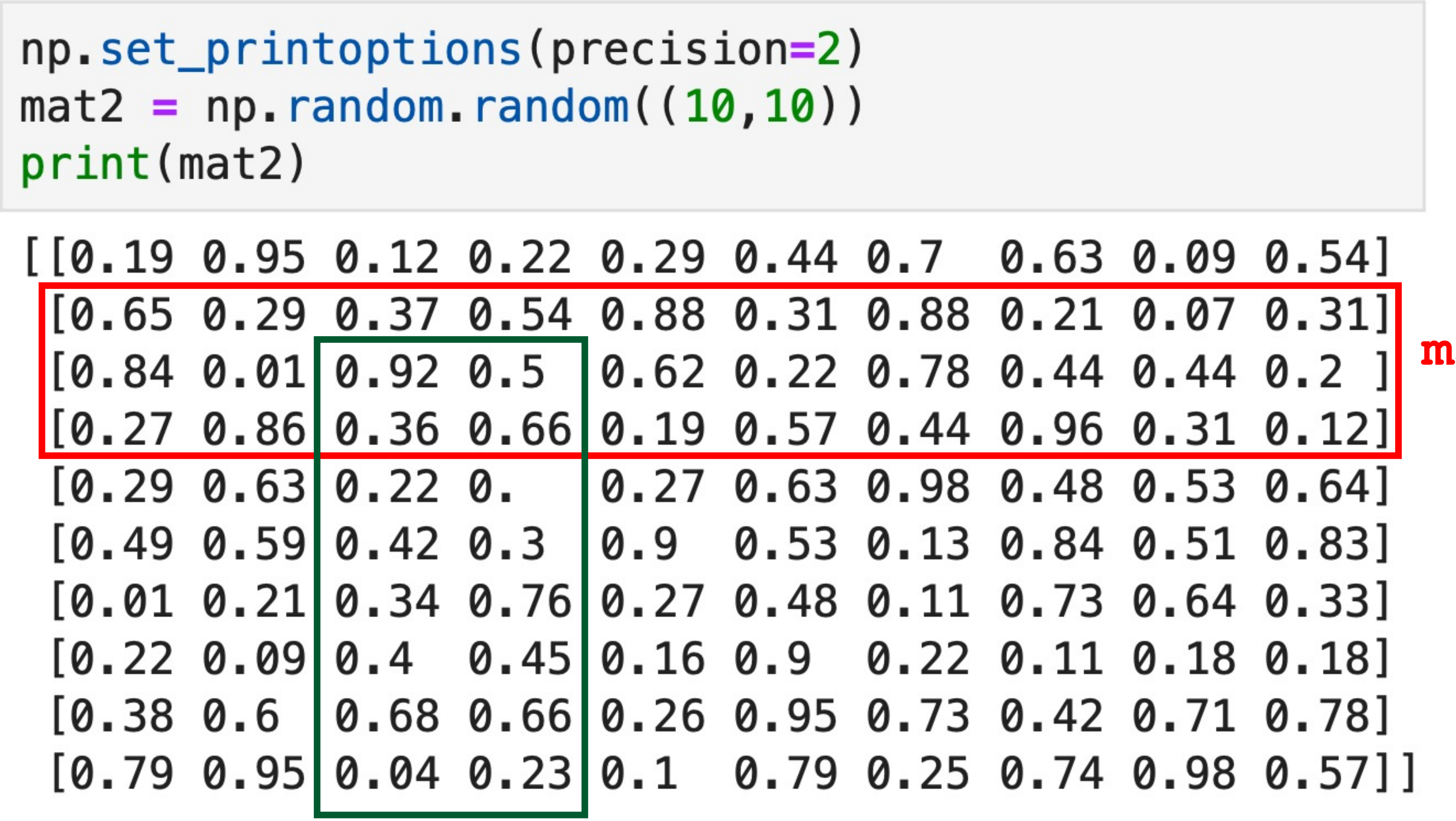


```
mat[:,0]
```

```
array([0.477, 0.091])
```

The colon means to go from the beginning to the end in the vertical dimension.
The 0 indicates that the first column is being accessed.

Slicing numpy matrices



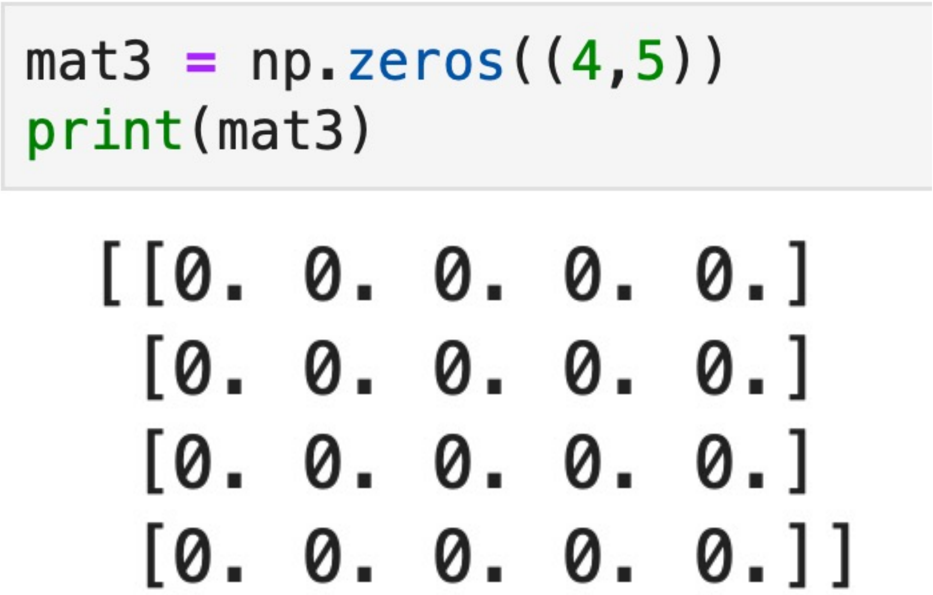
```
np.set_printoptions(precision=2)
mat2 = np.random.random((10,10))
print(mat2)
```

```
[[0.19 0.95 0.12 0.22 0.29 0.44 0.7  0.63 0.09 0.54]
 [0.65 0.29 0.37 0.54 0.88 0.31 0.88 0.21 0.07 0.31]
 [0.84 0.01 0.92 0.5  0.62 0.22 0.78 0.44 0.44 0.2 ]
 [0.27 0.86 0.36 0.66 0.19 0.57 0.44 0.96 0.31 0.12]
 [0.29 0.63 0.22 0.   0.27 0.63 0.98 0.48 0.53 0.64]
 [0.49 0.59 0.42 0.3  0.9  0.53 0.13 0.84 0.51 0.83]
 [0.01 0.21 0.34 0.76 0.27 0.48 0.11 0.73 0.64 0.33]
 [0.22 0.09 0.4  0.45 0.16 0.9  0.22 0.11 0.18 0.18]
 [0.38 0.6  0.68 0.66 0.26 0.95 0.73 0.42 0.71 0.78]
 [0.79 0.95 0.04 0.23 0.1  0.79 0.25 0.74 0.98 0.57]]
```

`mat2[1:4]`

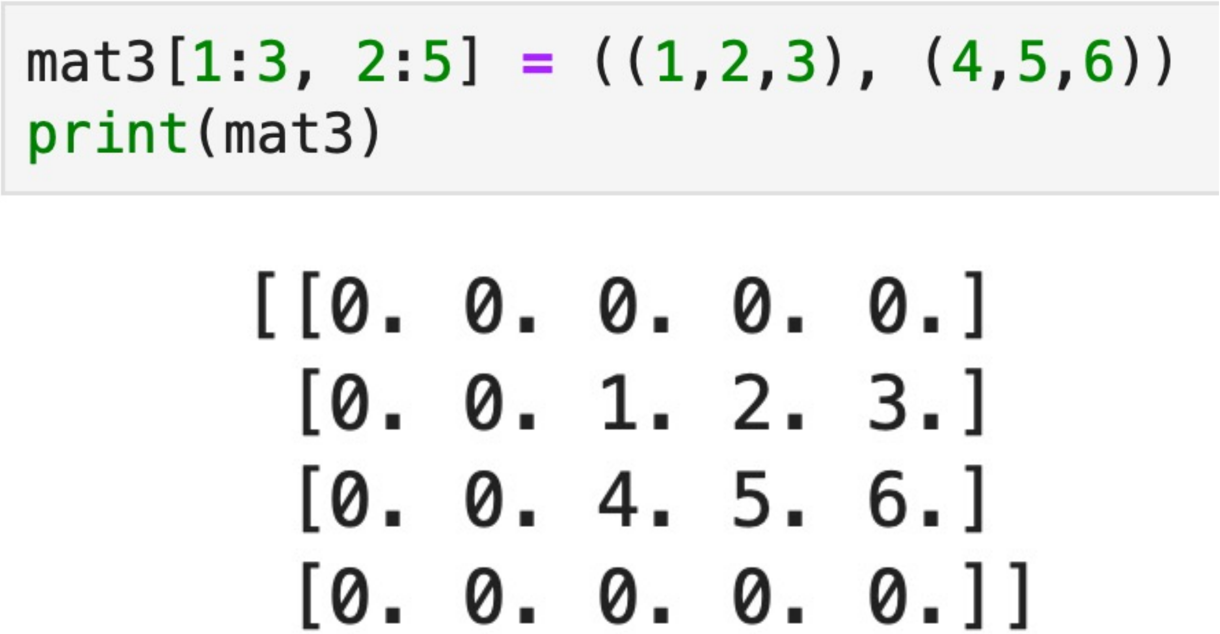
`mat2[2:,2:4]`

Setting multiple values



```python
mat3 = np.zeros((4,5))
print(mat3)
```

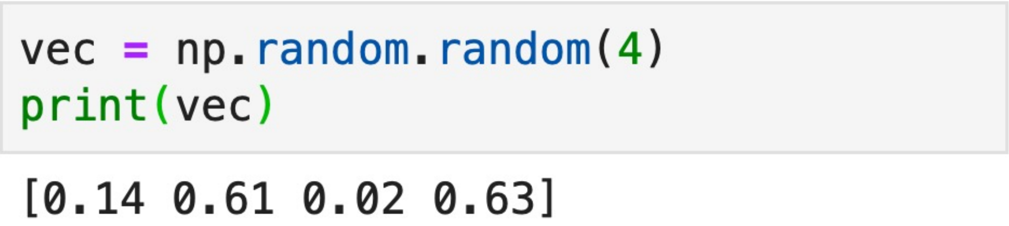
```
[[0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]]
```



```python
mat3[1:3, 2:5] = ((1,2,3), (4,5,6))
print(mat3)
```

```
[[0. 0. 0. 0. 0.]
 [0. 0. 1. 2. 3.]
 [0. 0. 4. 5. 6.]
 [0. 0. 0. 0. 0.]]
```

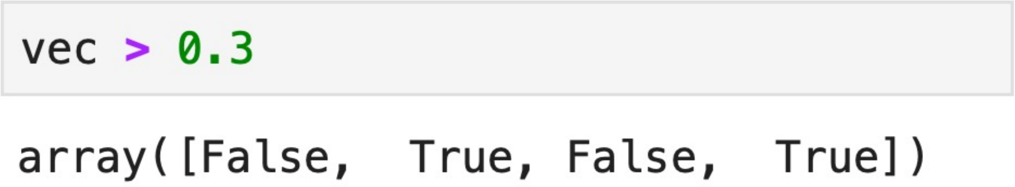
Bulk comparison



```
vec = np.random.random(4)
print(vec)
```

```
[0.14 0.61 0.02 0.63]
```

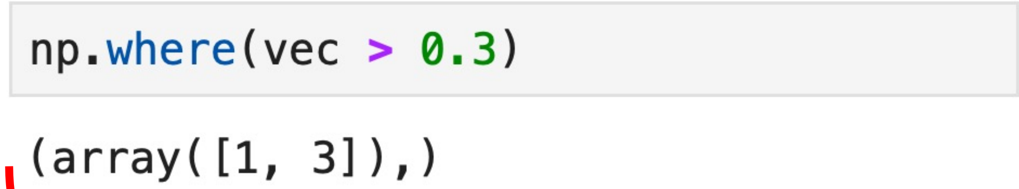
Create a vector of random numbers.



```
vec > 0.3
```

```
array([False,  True, False,  True])
```

Compare it to a value. The result is a Boolean vector.



```
np.where(vec > 0.3)
```

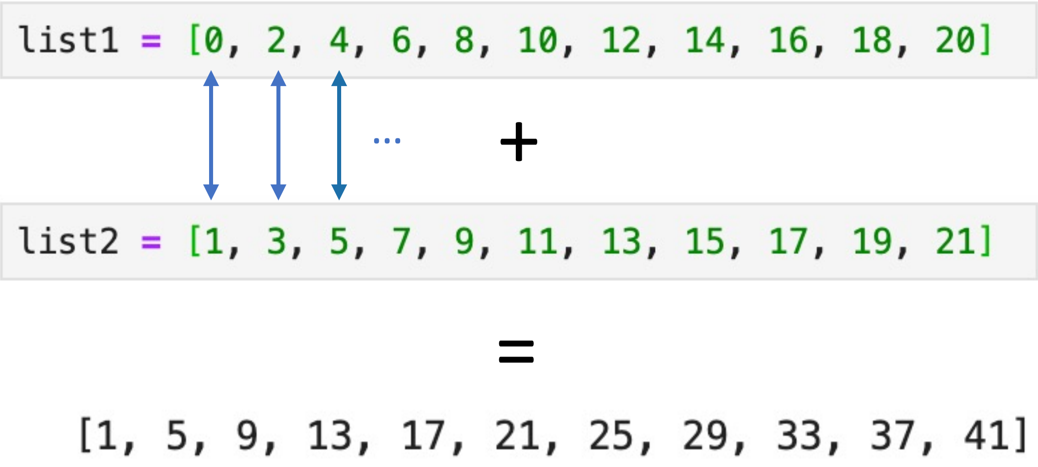
```
(array([1, 3]),)
```

The `np.where()` function will indicate *where* the True values lie. In this case, they True values are in position 1 and position 3.

Note that this is a tuple with two elements. The second element is filled when comparing matrices.

Elementwise operations

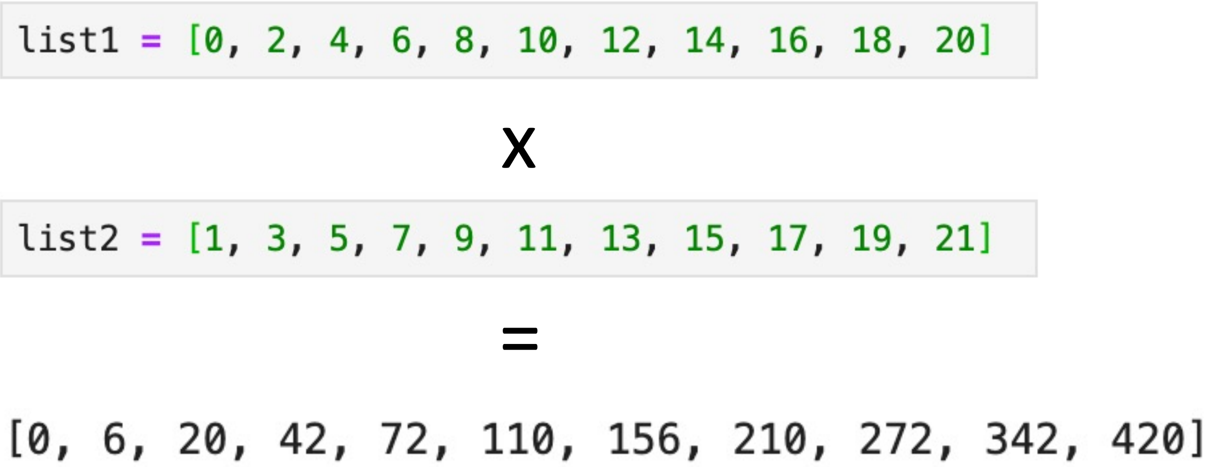
Apply the plus operator to element pairs w/ same index



```
list1 = [0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

```
list2 = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21]
```

```
[1, 5, 9, 13, 17, 21, 25, 29, 33, 37, 41]
```



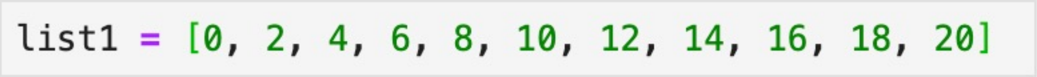
```
list1 = [0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

```
list2 = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21]
```

```
[0, 6, 20, 42, 72, 110, 156, 210, 272, 342, 420]
```

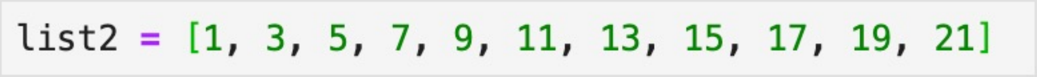
Same applies to **subtraction** and **division**

Loop implementations



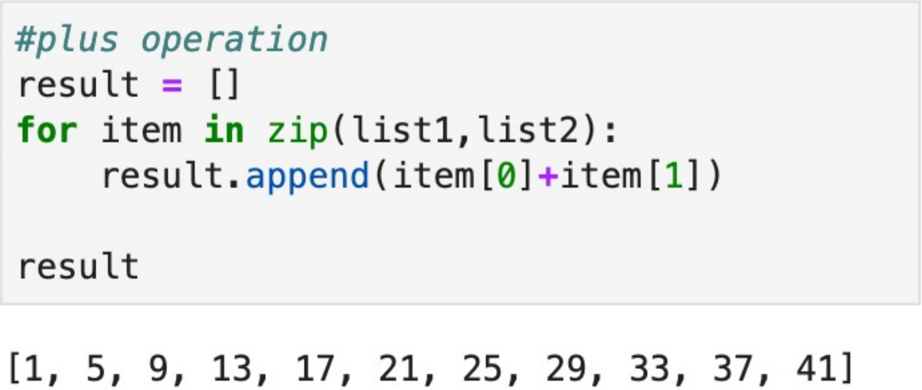
```
list1 = [0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

+



```
list2 = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21]
```

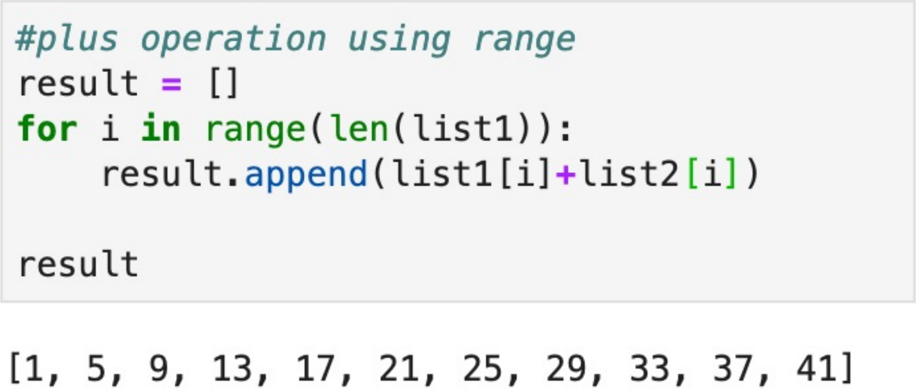
1) Using zip



```
#plus operation
result = []
for item in zip(list1,list2):
    result.append(item[0]+item[1])

result
```

```
[1, 5, 9, 13, 17, 21, 25, 29, 33, 37, 41]
```

2) Using range

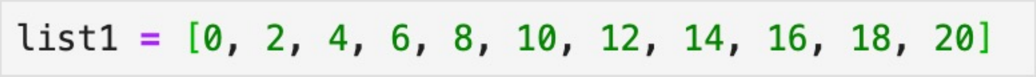


```
#plus operation using range
result = []
for i in range(len(list1)):
    result.append(list1[i]+list2[i])

result
```

```
[1, 5, 9, 13, 17, 21, 25, 29, 33, 37, 41]
```

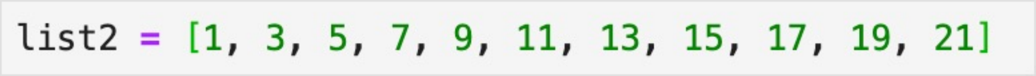
Your code will run slower if you deal w/ large collections

NumPy implementation

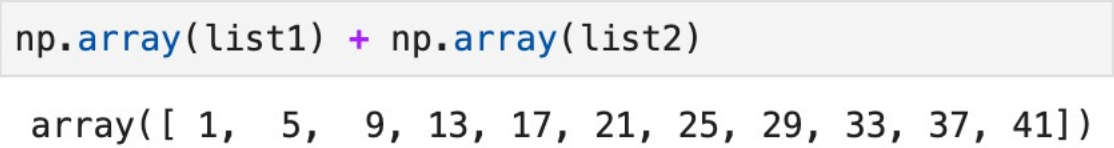


```
list1 = [0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

+

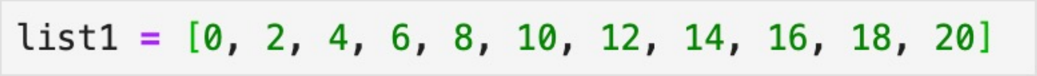


```
list2 = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21]
```



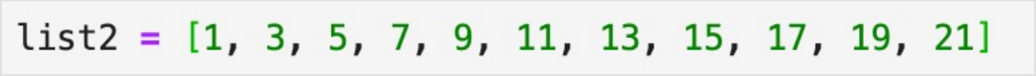
```
np.array(list1) + np.array(list2)
```

```
array([ 1,  5,  9, 13, 17, 21, 25, 29, 33, 37, 41])
```

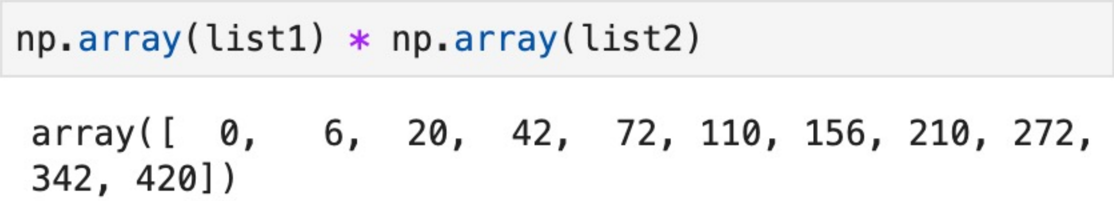


```
list1 = [0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

X



```
list2 = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21]
```

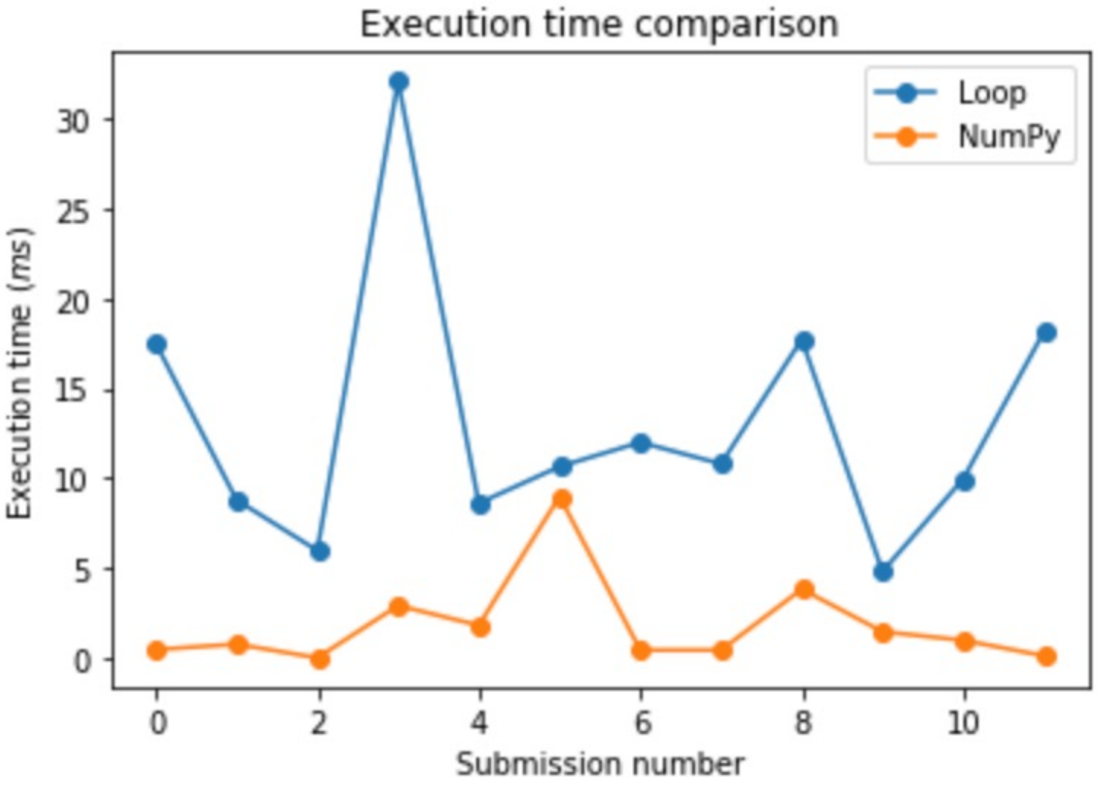


```
np.array(list1) * np.array(list2)
```

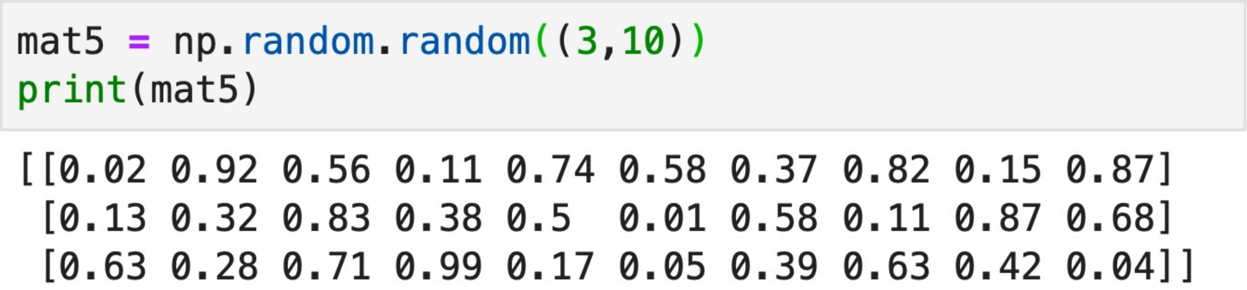
```
array([  0,   6,  20,  42,  72, 110, 156, 210, 272,
342, 420])
```

The same logic works for matrices as well

How fast is NumPy?

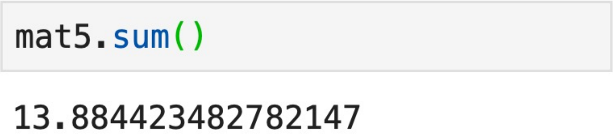


Some statistics and information (1)



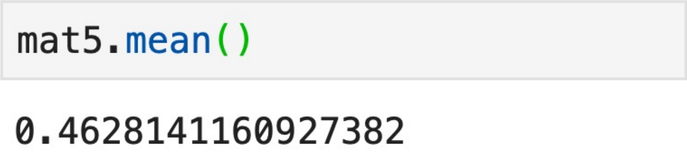
```
mat5 = np.random.random((3,10))
print(mat5)
```

```
[[0.02 0.92 0.56 0.11 0.74 0.58 0.37 0.82 0.15 0.87]
 [0.13 0.32 0.83 0.38 0.5  0.01 0.58 0.11 0.87 0.68]
 [0.63 0.28 0.71 0.99 0.17 0.05 0.39 0.63 0.42 0.04]]
```



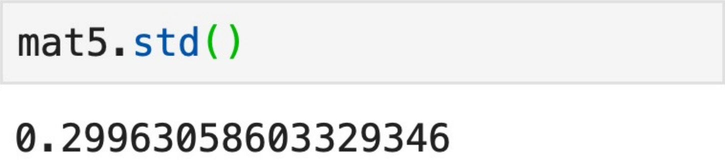
```
mat5.sum()
```

```
13.884423482782147
```



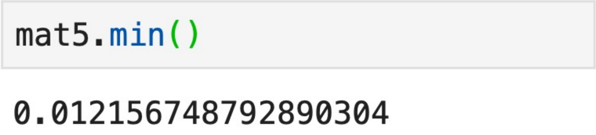
```
mat5.mean()
```

```
0.4628141160927382
```



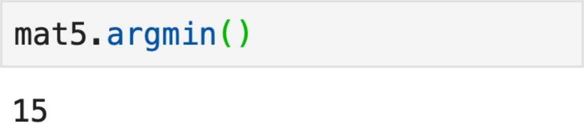
```
mat5.std()
```

```
0.29963058603329346
```



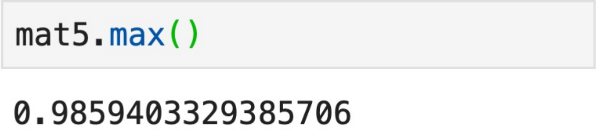
```
mat5.min()
```

```
0.012156748792890304
```



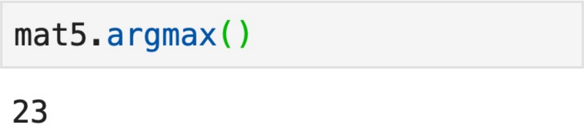
```
mat5.argmin()
```

```
15
```



```
mat5.max()
```

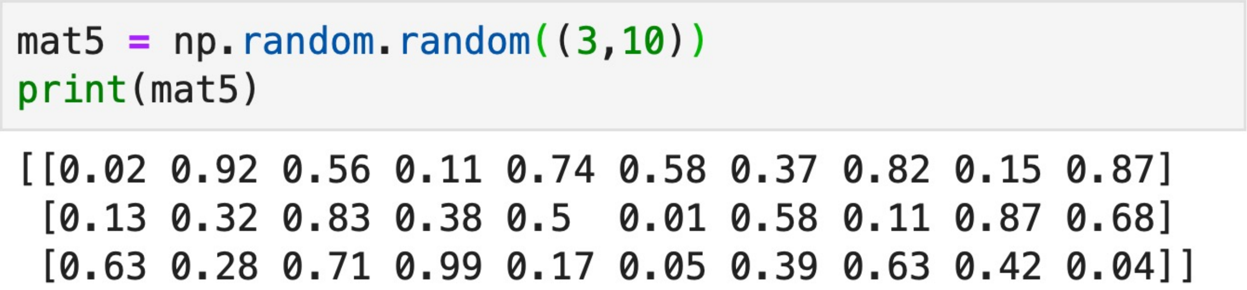
```
0.9859403329385706
```



```
mat5.argmax()
```

```
23
```

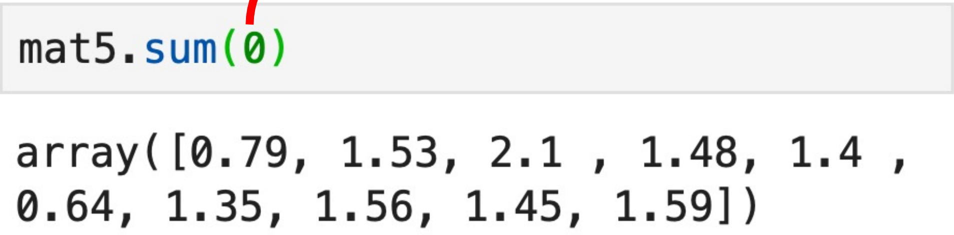
Some statistics and information (2)



```
mat5 = np.random.random((3,10))
print(mat5)
```

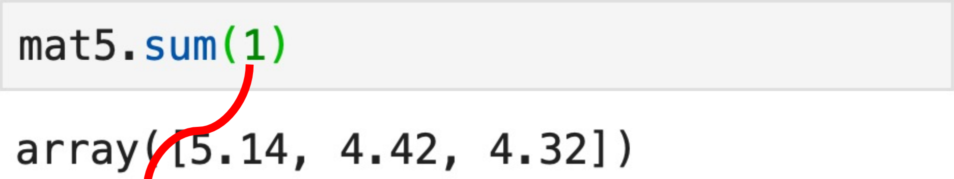
```
[[0.02 0.92 0.56 0.11 0.74 0.58 0.37 0.82 0.15 0.87]
 [0.13 0.32 0.83 0.38 0.5  0.01 0.58 0.11 0.87 0.68]
 [0.63 0.28 0.71 0.99 0.17 0.05 0.39 0.63 0.42 0.04]]
```

Passing 0 means sum by column



```
mat5.sum(0)
```

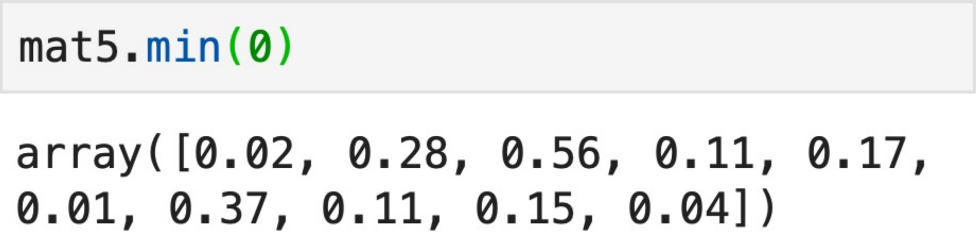
```
array([0.79, 1.53, 2.1 , 1.48, 1.4 ,
0.64, 1.35, 1.56, 1.45, 1.59])
```



```
mat5.sum(1)
```

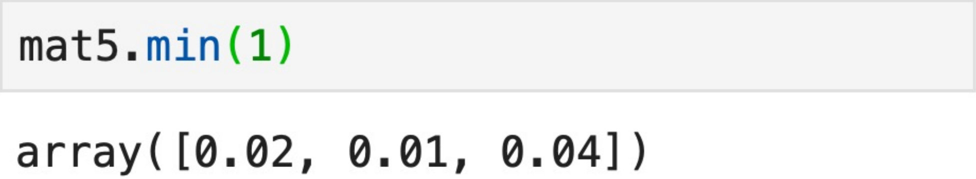
```
array([5.14, 4.42, 4.32])
```

Passing 1 means sum by row



```
mat5.min(0)
```

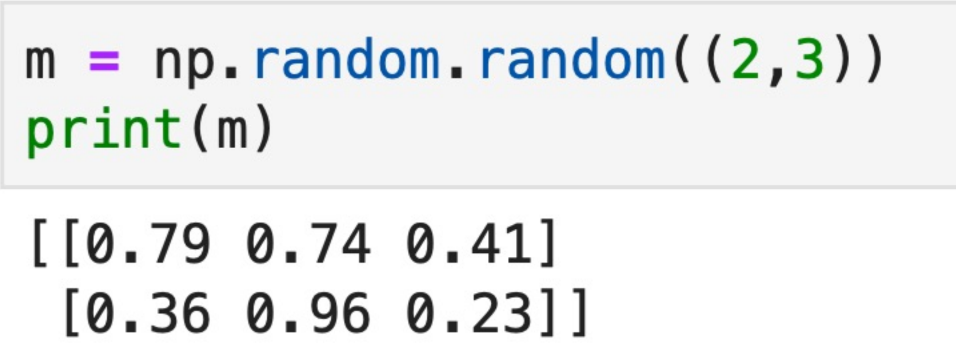
```
array([0.02, 0.28, 0.56, 0.11, 0.17,
0.01, 0.37, 0.11, 0.15, 0.04])
```



```
mat5.min(1)
```

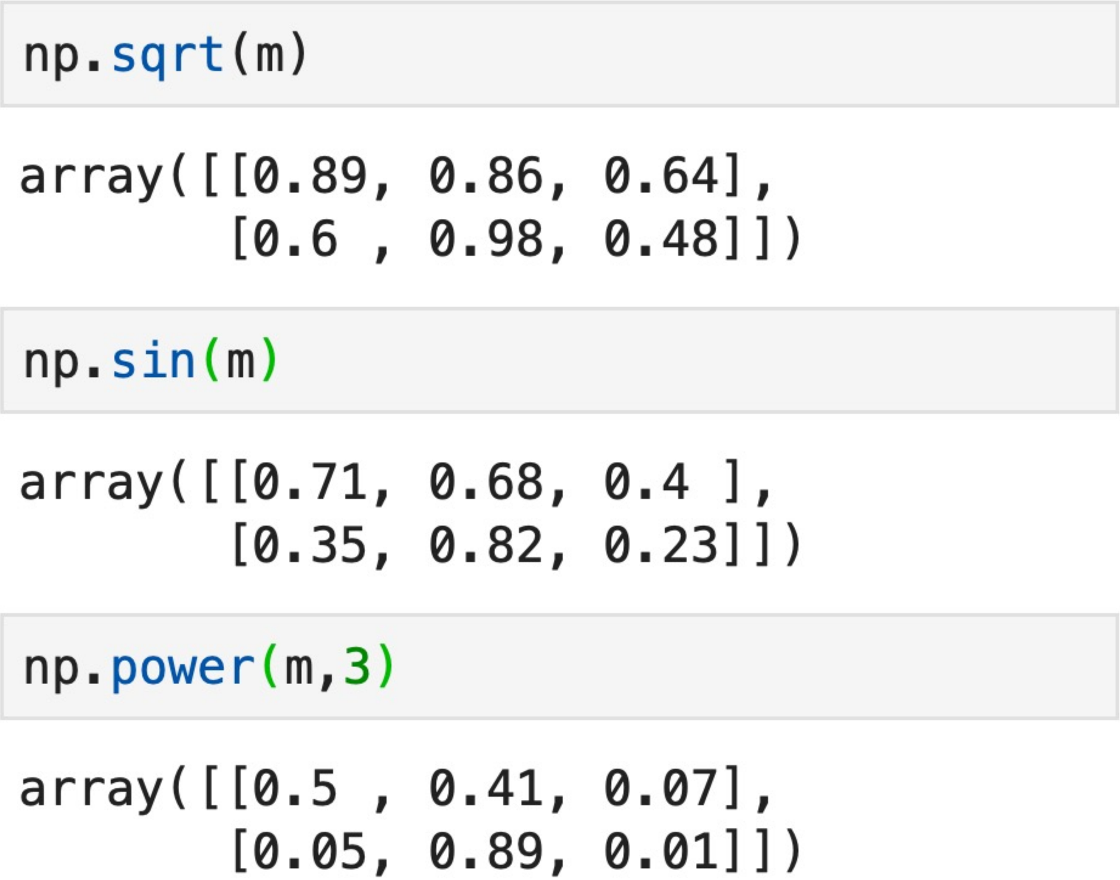
```
array([0.02, 0.01, 0.04])
```

More operations



```
m = np.random.random((2,3))
print(m)
```

```
[[0.79 0.74 0.41]
 [0.36 0.96 0.23]]
```



```
np.sqrt(m)
```

```
array([[0.89, 0.86, 0.64],
       [0.6 , 0.98, 0.48]])
```

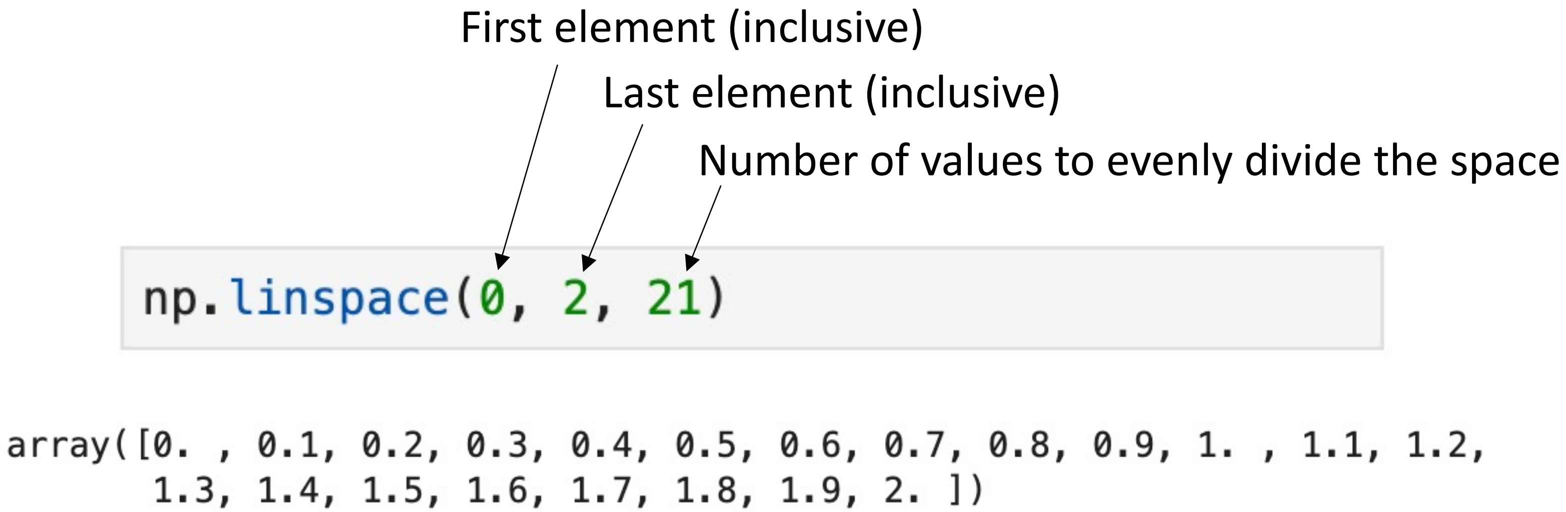
```
np.sin(m)
```

```
array([[0.71, 0.68, 0.4 ],
       [0.35, 0.82, 0.23]])
```

```
np.power(m,3)
```

```
array([[0.5 , 0.41, 0.07],
       [0.05, 0.89, 0.01]])
```

Spaces/Ranges



First element (inclusive)

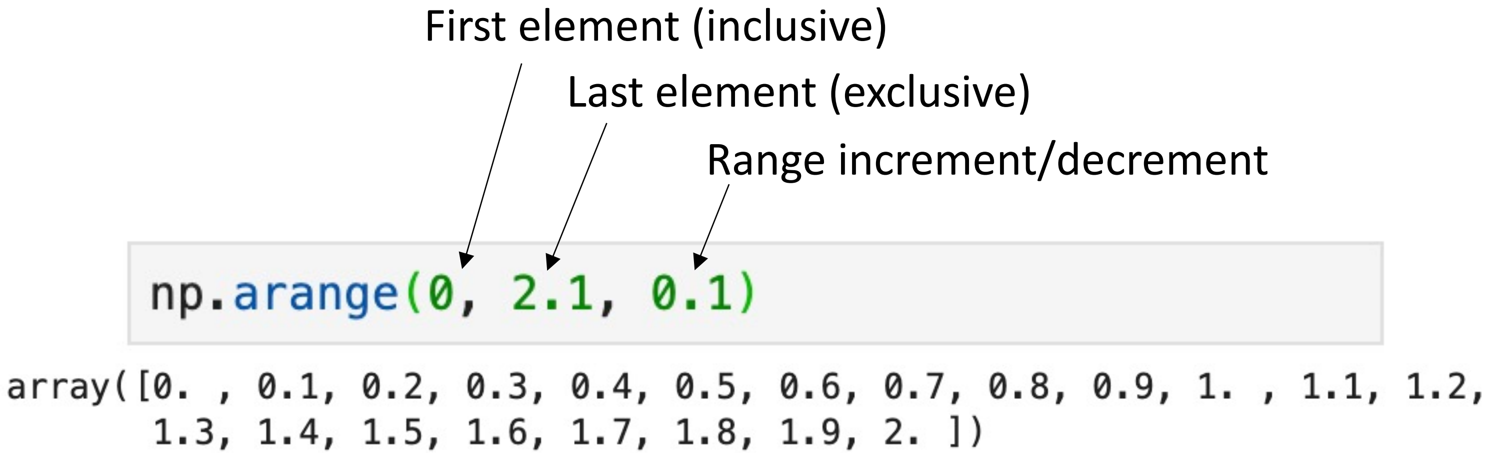
Last element (inclusive)

Number of values to evenly divide the space

```
np.linspace(0, 2, 21)
```

```
array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. , 1.1, 1.2,
       1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2. ])
```

np.arrange is very similar to range but **can handle decimals** and **returns a NumPy array**.



First element (inclusive)

Last element (exclusive)

Range increment/decrement

```
np.arange(0, 2.1, 0.1)
```

```
array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. , 1.1, 1.2,
       1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2. ])
```