Numpy

with his little helper matplotlib

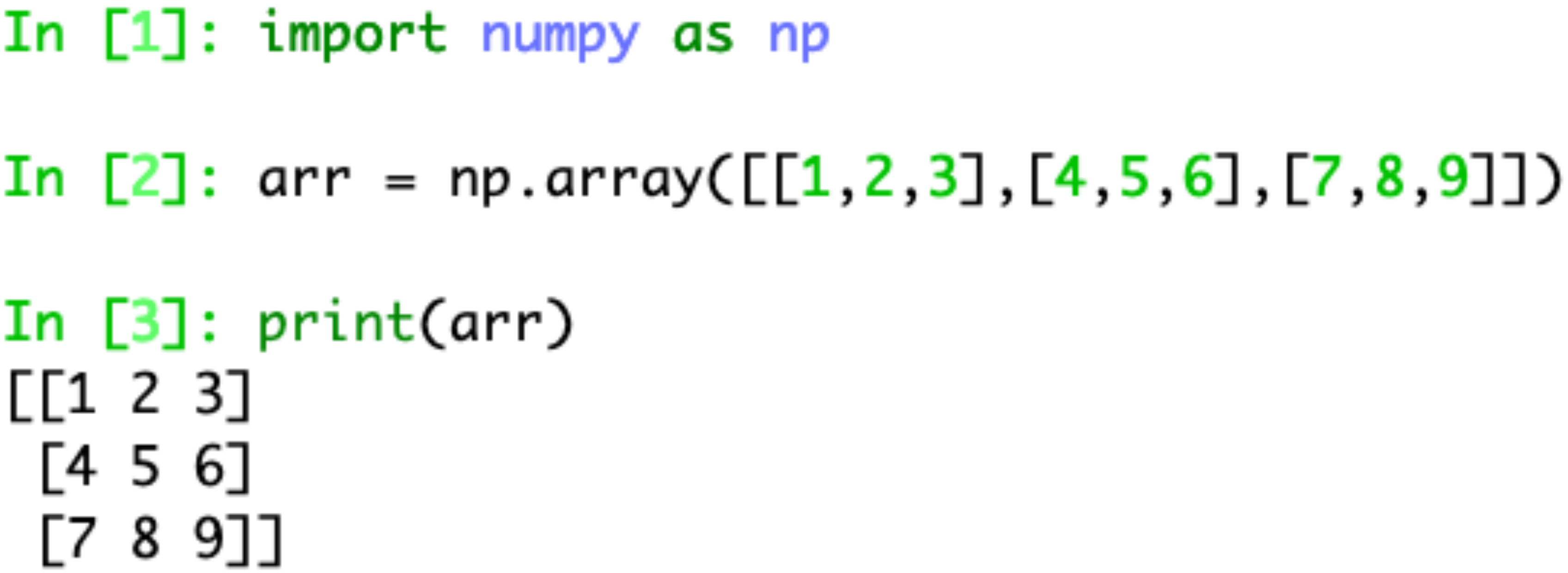
Numpy

A fundamental science computing package.

A manipulator for high-dimensional data: *ndarray*

NDArray

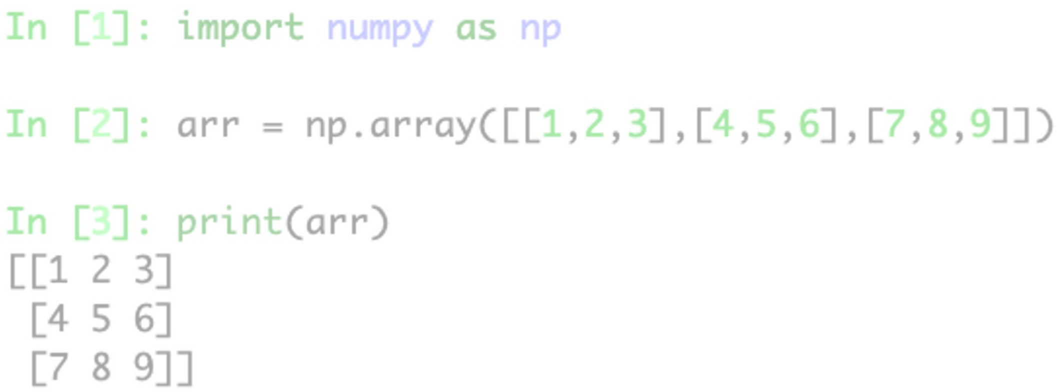
ndarray: An array with arbitrary dimension and size.



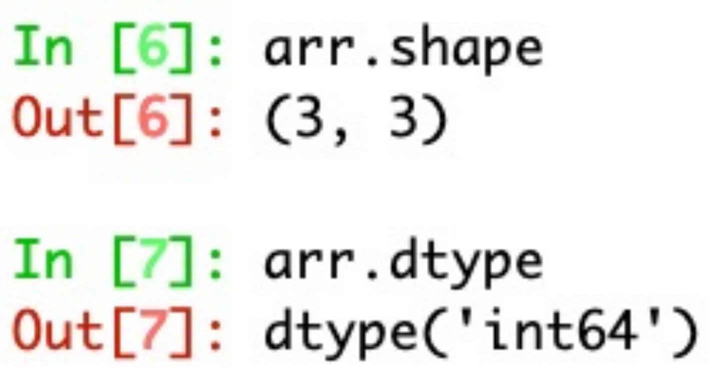
```
In [1]: import numpy as np

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

In [3]: print(arr)
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

NDArray – Data Types & Shapes

Numpy *ndarray* are associated with **shapes** and **data types (dtype)**



```
In [1]: import numpy as np

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

In [3]: print(arr)
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```



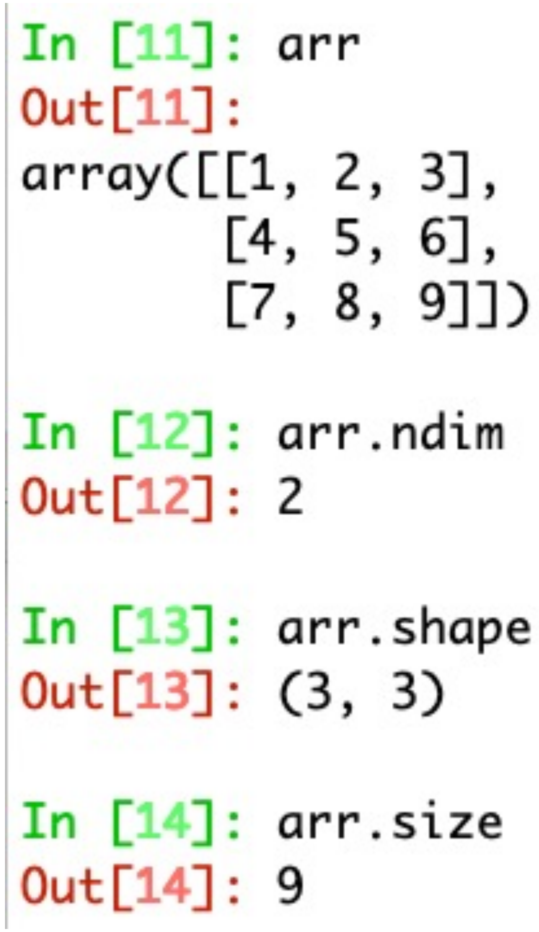
```
In [6]: arr.shape
Out[6]: (3, 3)

In [7]: arr.dtype
Out[7]: dtype('int64')
```

Unlike Python list:

- the shape (size) of the *ndarray* is fixed
- all items in *ndarray* should be of the same type (int / float / …)

NDArray – Data Types & Shapes

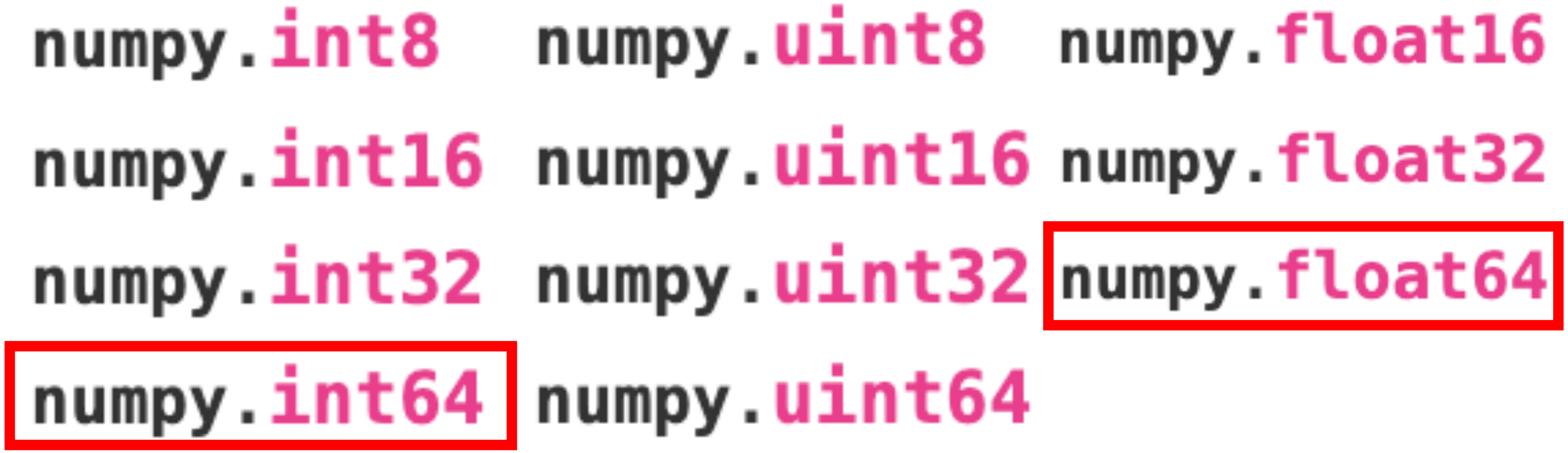
- `numpy.shape`
  - Shape of *ndarray*
- `numpy.size`
  - Number of elements in *ndarray*
- `numpy.ndim`
  - Shape of shape of *ndarray*



```
In [11]: arr
Out[11]:
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])

In [12]: arr.ndim
Out[12]: 2

In [13]: arr.shape
Out[13]: (3, 3)

In [14]: arr.size
Out[14]: 9
```

NDArray – Data Types & Shapes

(Suggested) Supported data types:



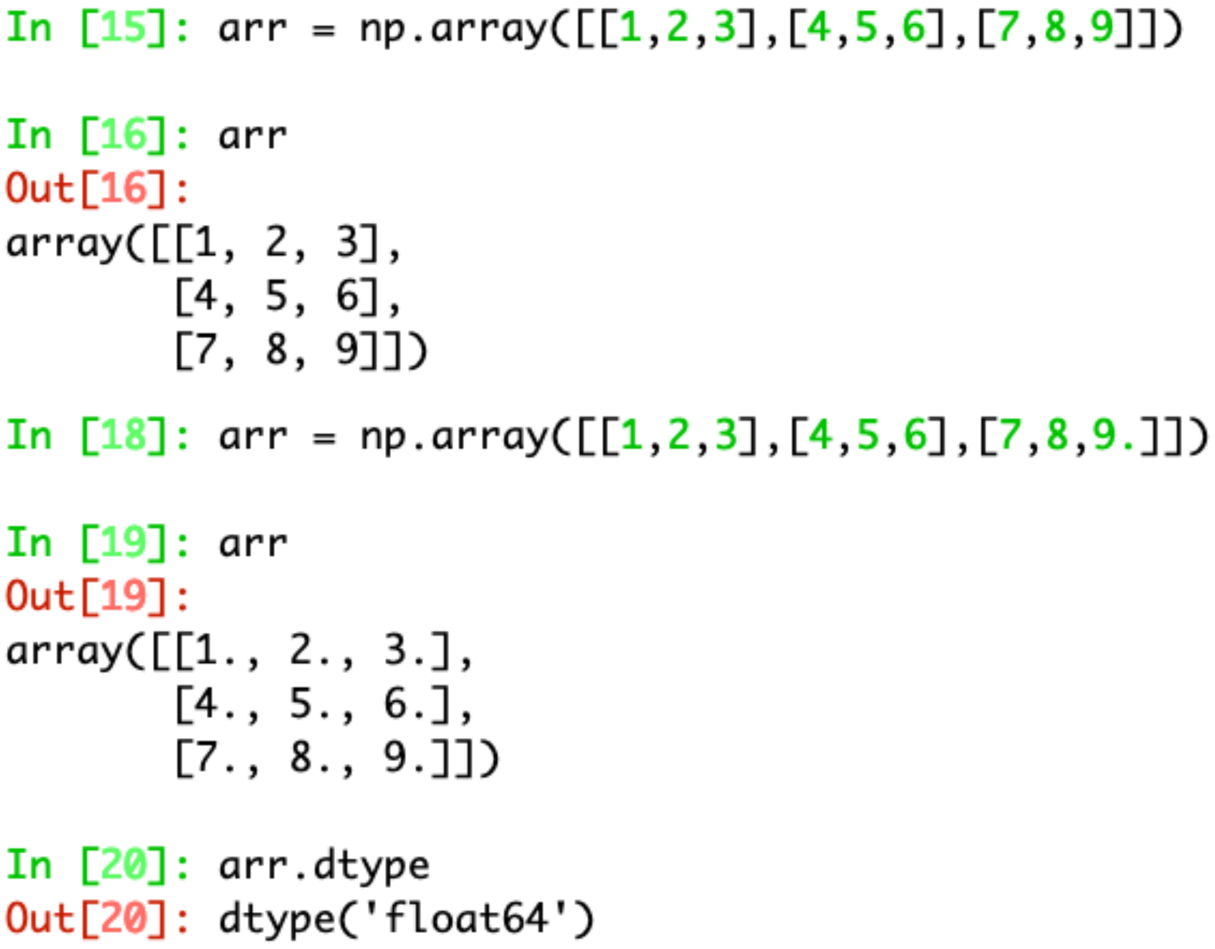
```
numpy.int8   numpy.uint8  numpy.float16
numpy.int16  numpy.uint16 numpy.float32
numpy.int32  numpy.uint32 numpy.float64
numpy.int64  numpy.uint64
```

The same as in C (<stdint.h>)

Default Data Type

NDArray – Creation

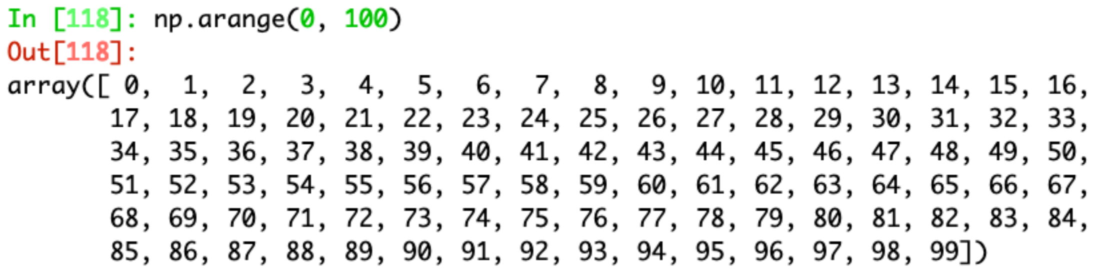
- `numpy.array`



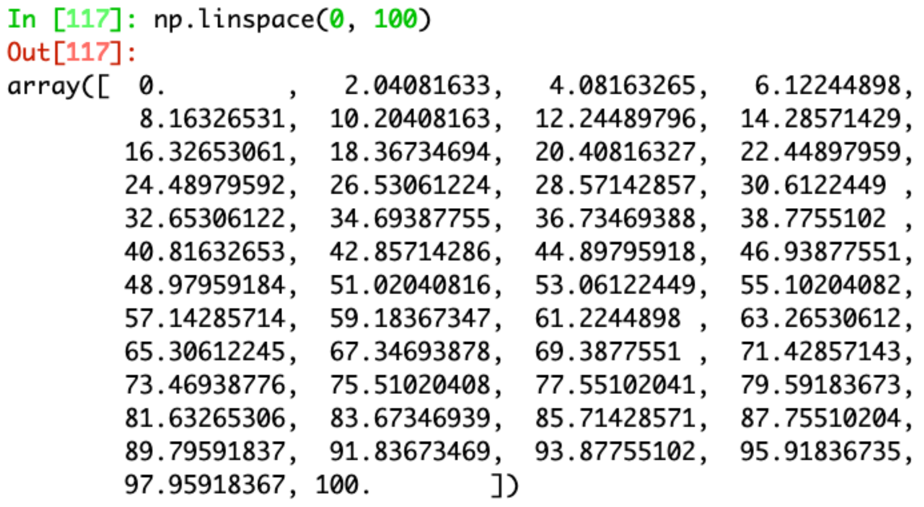
```
In [15]: arr = np.array([[1,2,3],[4,5,6],[7,8,9]])

In [16]: arr
Out[16]:
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])

In [18]: arr = np.array([[1,2,3],[4,5,6],[7,8,9.]])

In [19]: arr
Out[19]:
array([[1., 2., 3.],
       [4., 5., 6.],
       [7., 8., 9.]])

In [20]: arr.dtype
Out[20]: dtype('float64')
```

NDArray – Creation

- `numpy.arange`
  - `numpy.arange([start, ]stop, [step, ]dtype=None, *, like=None)`
  - Default step size is 1
- `numpy.linspace`
  - `numpy.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None, axis=0)`



```
In [118]: np.arange(0, 100)
Out[118]:
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
       34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
       51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67,
       68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84,
       85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99])
```

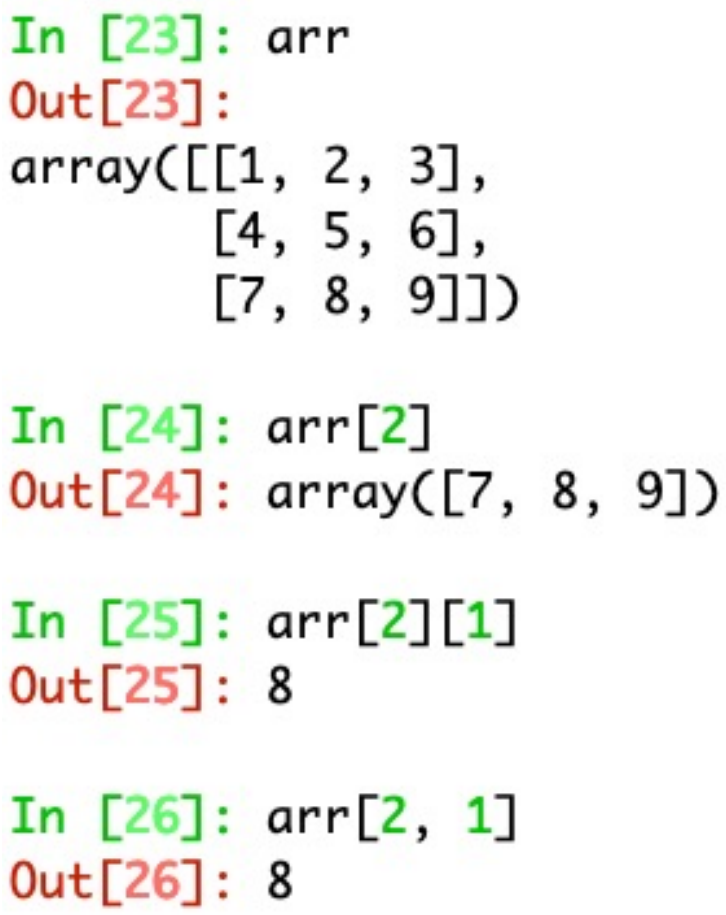


```
In [117]: np.linspace(0, 100)
Out[117]:
array([  0.        ,   2.04081633,   4.08163265,   6.12244898,
         8.16326531,  10.20408163,  12.24489796,  14.28571429,
        16.32653061,  18.36734694,  20.40816327,  22.44897959,
        24.48979592,  26.53061224,  28.57142857,  30.6122449 ,
        32.65306122,  34.69387755,  36.73469388,  38.7755102 ,
        40.81632653,  42.85714286,  44.89795918,  46.93877551,
        48.97959184,  51.02040816,  53.06122449,  55.10204082,
        57.14285714,  59.18367347,  61.2244898 ,  63.26530612,
        65.30612245,  67.34693878,  69.3877551 ,  71.42857143,
        73.46938776,  75.51020408,  77.55102041,  79.59183673,
        81.63265306,  83.67346939,  85.71428571,  87.75510204,
        89.79591837,  91.83673469,  93.87755102,  95.91836735,
        97.95918367, 100.        ])
```

NDArray – Indexing

Single element indexing

C-order indexing

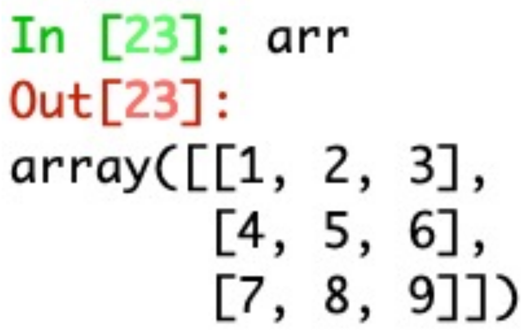


```
In [23]: arr
Out[23]:
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])

In [24]: arr[2]
Out[24]: array([7, 8, 9])

In [25]: arr[2][1]
Out[25]: 8

In [26]: arr[2, 1]
Out[26]: 8
```

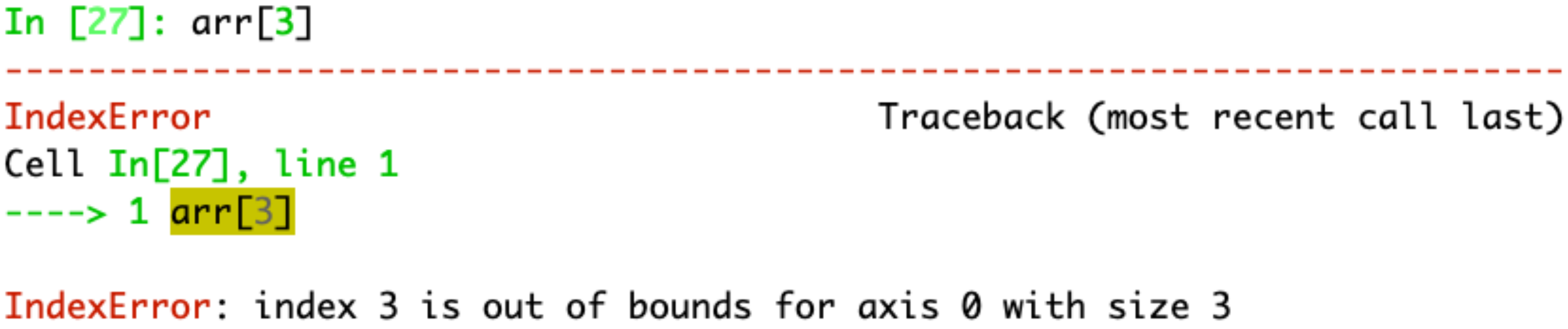
NDArray – Indexing

Single element indexing

Out of bounds



```
In [23]: arr
Out[23]:
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

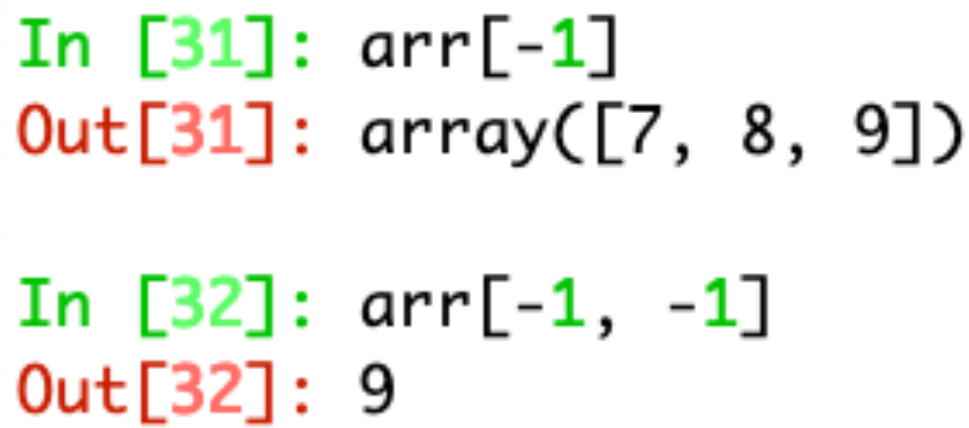


```
In [27]: arr[3]
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[27], line 1
----> 1 arr[3]

IndexError: index 3 is out of bounds for axis 0 with size 3
```

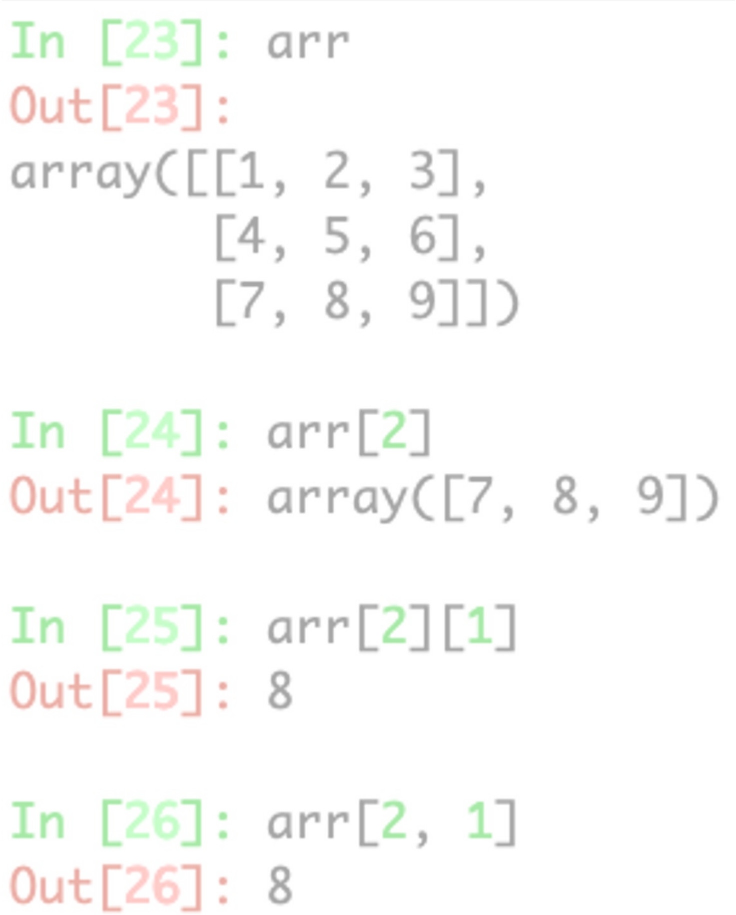
NDArray – Indexing

Single element indexing

Negative indices



```
In [31]: arr[-1]
Out[31]: array([7, 8, 9])

In [32]: arr[-1, -1]
Out[32]: 9
```

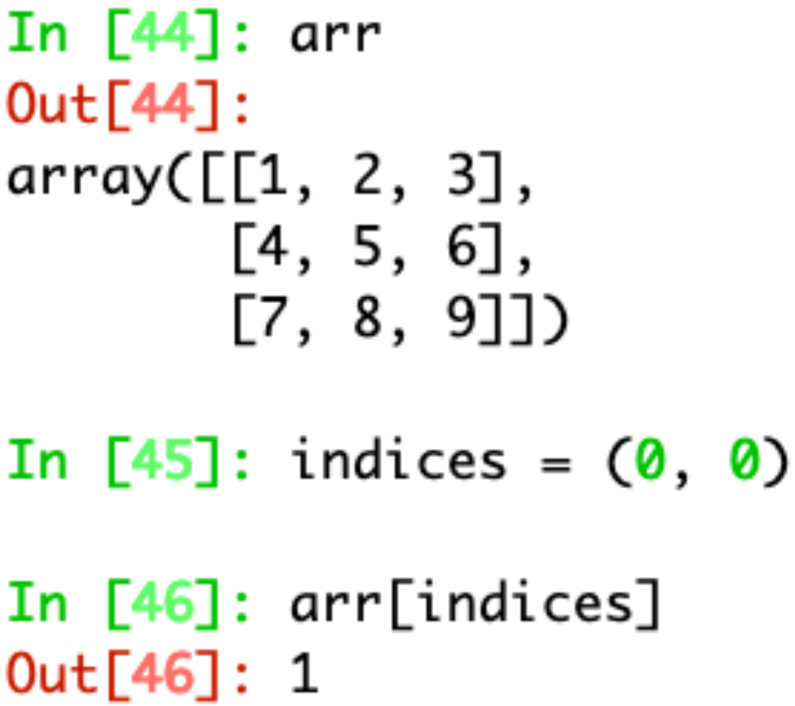


```
In [23]: arr
Out[23]:
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])

In [24]: arr[2]
Out[24]: array([7, 8, 9])

In [25]: arr[2][1]
Out[25]: 8

In [26]: arr[2, 1]
Out[26]: 8
```

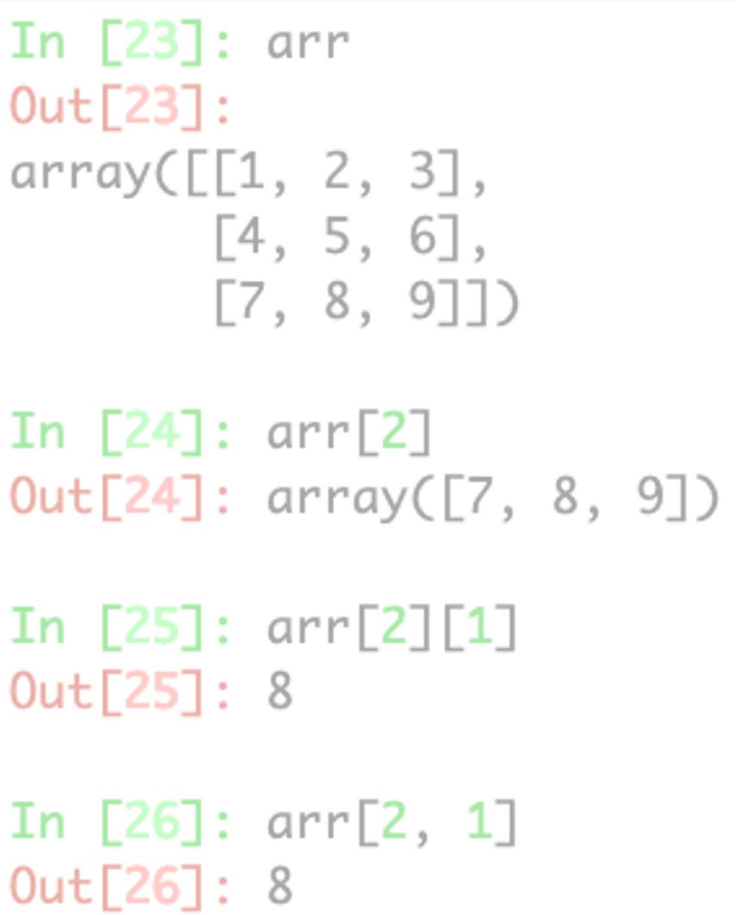
NDArray – Indexing

Single element indexing

With tuple indices



```
In [44]: arr
Out[44]:
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])

In [45]: indices = (0, 0)

In [46]: arr[indices]
Out[46]: 1
```

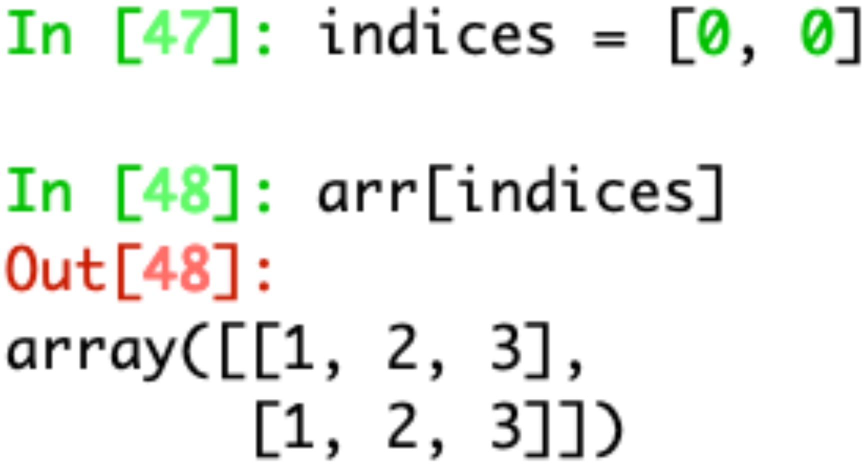


```
In [23]: arr
Out[23]:
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])

In [24]: arr[2]
Out[24]: array([7, 8, 9])

In [25]: arr[2][1]
Out[25]: 8

In [26]: arr[2, 1]
Out[26]: 8
```

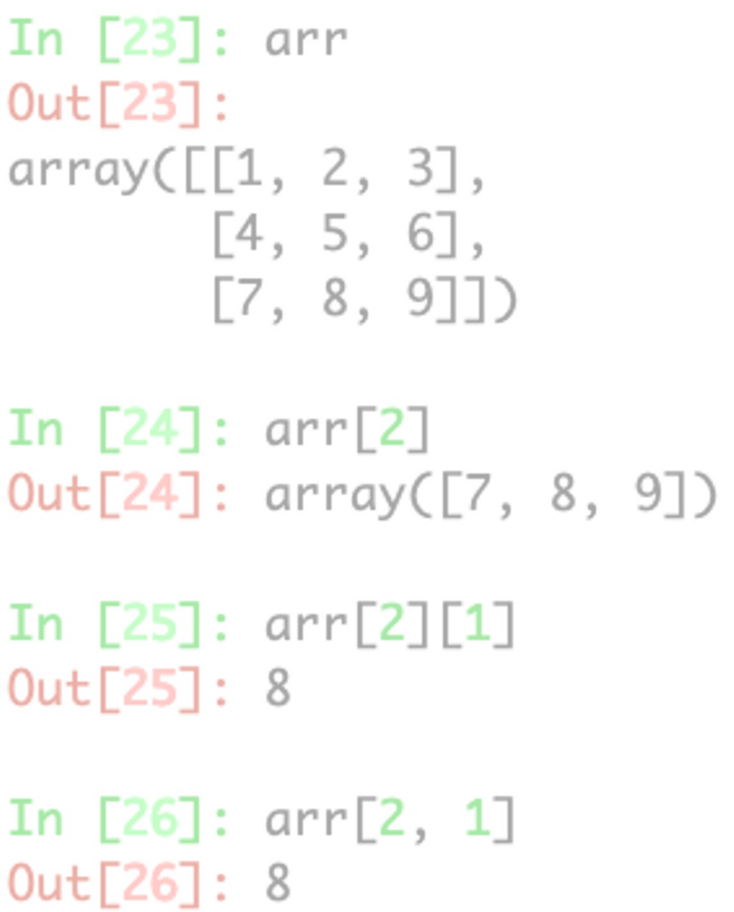
NDArray – Indexing

Single element indexing

With list indices? ☹



```
In [47]: indices = [0, 0]

In [48]: arr[indices]
Out[48]:
array([[1, 2, 3],
       [1, 2, 3]])
```



```
In [23]: arr
Out[23]:
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])

In [24]: arr[2]
Out[24]: array([7, 8, 9])

In [25]: arr[2][1]
Out[25]: 8

In [26]: arr[2, 1]
Out[26]: 8
```

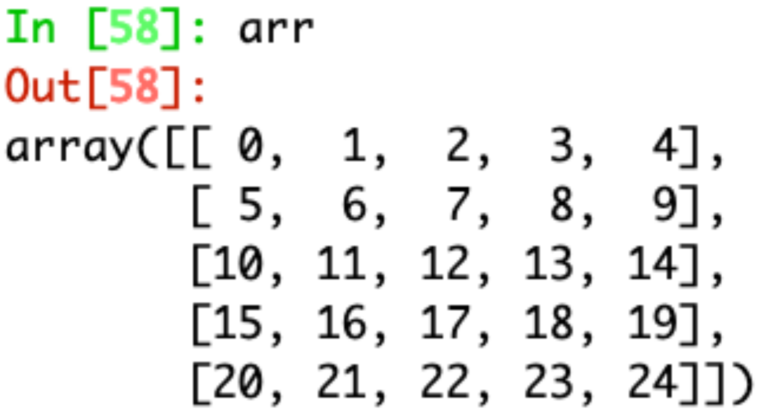
NDArray – Indexing

Slice Indexing

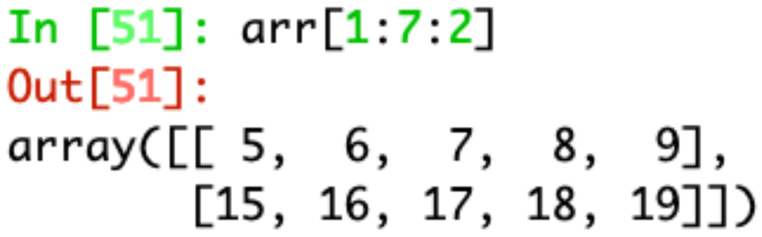
ndarray[start:stop:step]

- start: included
- stop: excluded

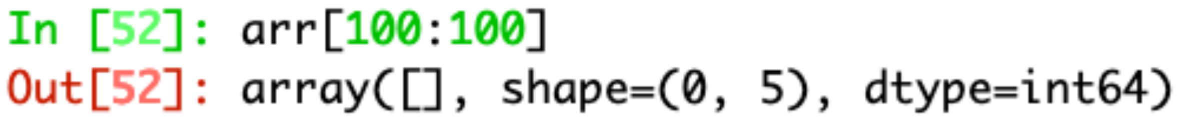
- No bound restrictions ☺



```
In [58]: arr
Out[58]:
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24]])
```



```
In [51]: arr[1:7:2]
Out[51]:
array([[ 5,  6,  7,  8,  9],
       [15, 16, 17, 18, 19]])
```



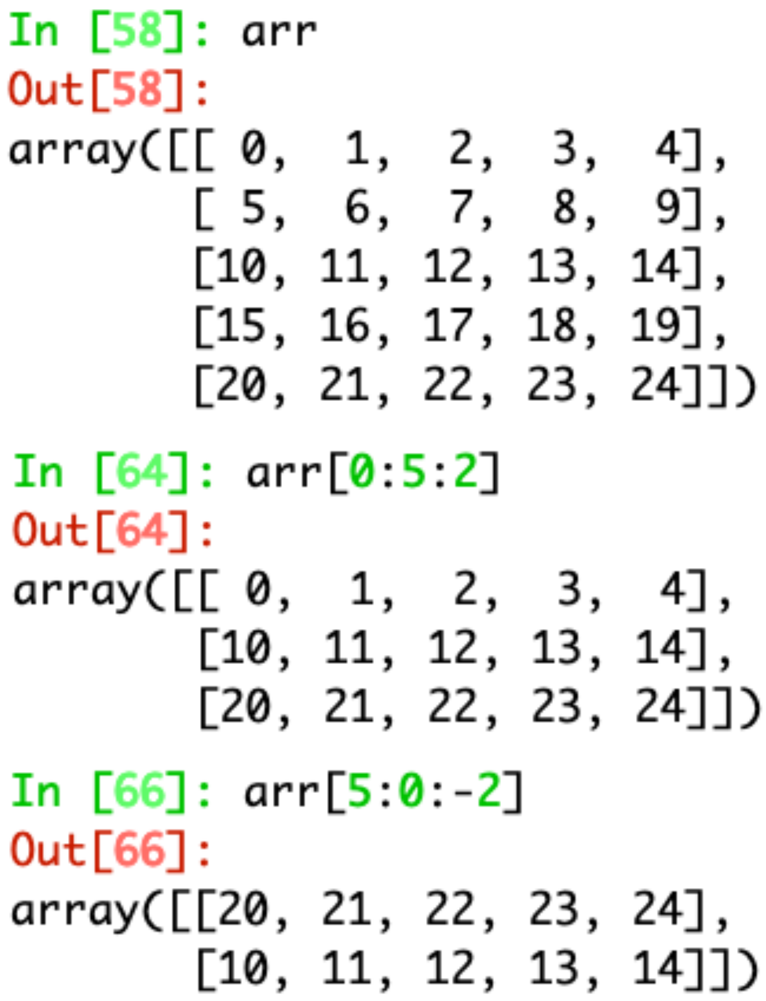
```
In [52]: arr[100:100]
Out[52]: array([], shape=(0, 5), dtype=int64)
```

NDArray – Indexing

Slice Indexing

ndarray`[start:stop:step]`

- `start`: included
- `stop`: excluded
- `step`: enum step & **direction**

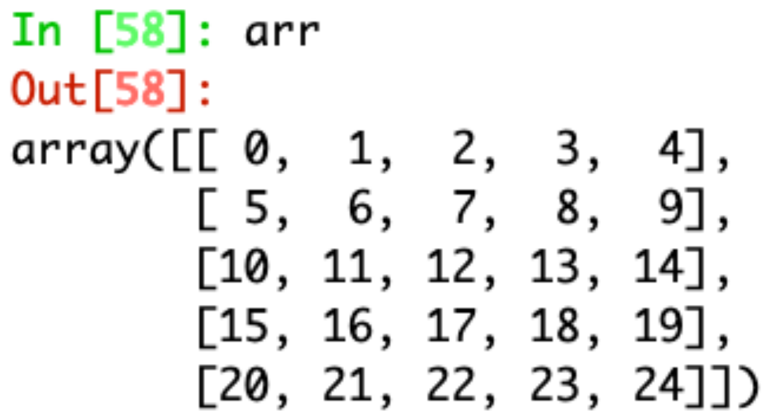


```
In [58]: arr
Out[58]:
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24]])

In [64]: arr[0:5:2]
Out[64]:
array([[ 0,  1,  2,  3,  4],
       [10, 11, 12, 13, 14],
       [20, 21, 22, 23, 24]])

In [66]: arr[5:0:-2]
Out[66]:
array([[20, 21, 22, 23, 24],
       [10, 11, 12, 13, 14]])
```

NDArray – Indexing

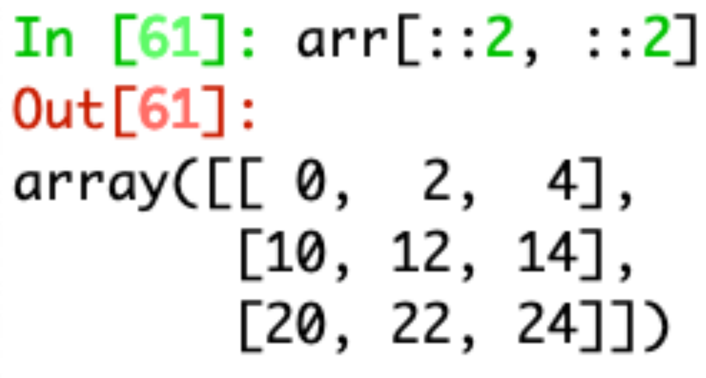
Slice Indexing

ndarray[start:stop:step]



```
In [58]: arr
Out[58]:
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24]])
```

Use with multi-dimension indexing

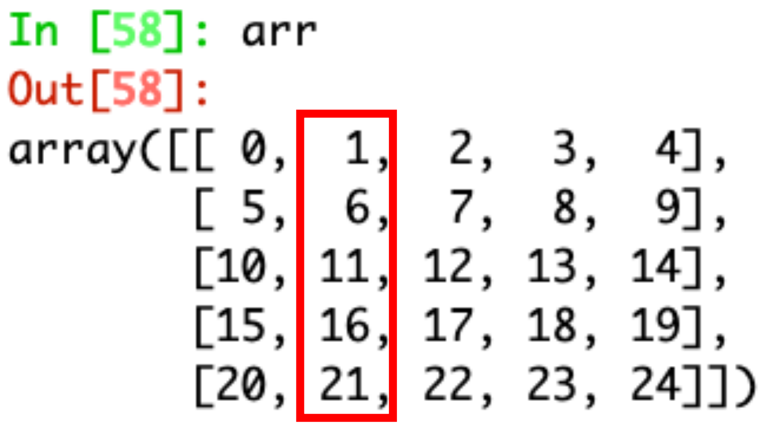


```
In [61]: arr[::2, ::2]
Out[61]:
array([[ 0,  2,  4],
       [10, 12, 14],
       [20, 22, 24]])
```

NDArray – Indexing

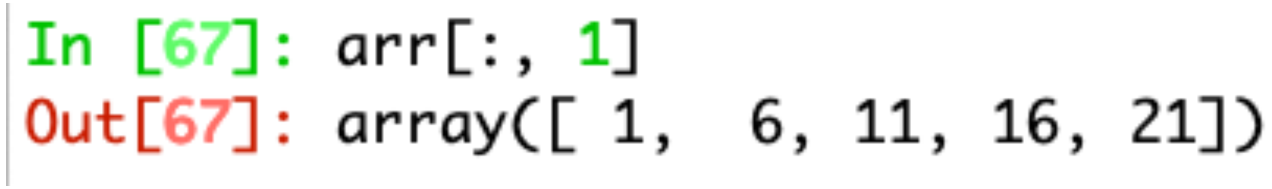
Slice Indexing

ndarray[start:stop:step]



```
In [58]: arr
Out[58]:
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24]])
```

Ignore dimension

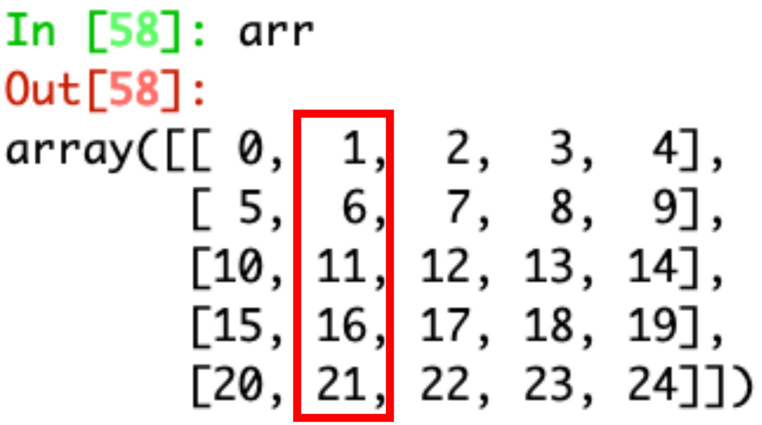


```
In [67]: arr[:, 1]
Out[67]: array([ 1,  6, 11, 16, 21])
```

NDArray – Indexing

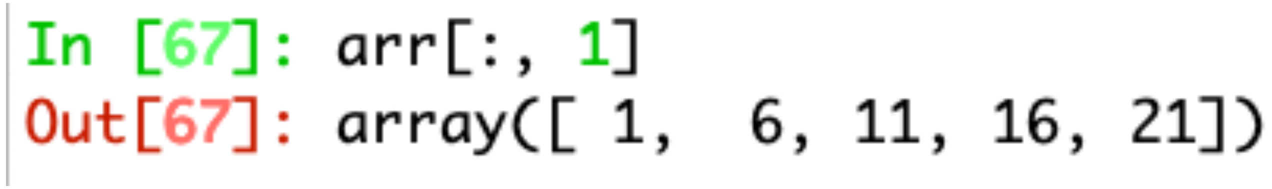
Slice Indexing

`ndarray[start:stop:step]`



```
In [58]: arr
Out[58]:
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24]])
```

Ignore certain dimension



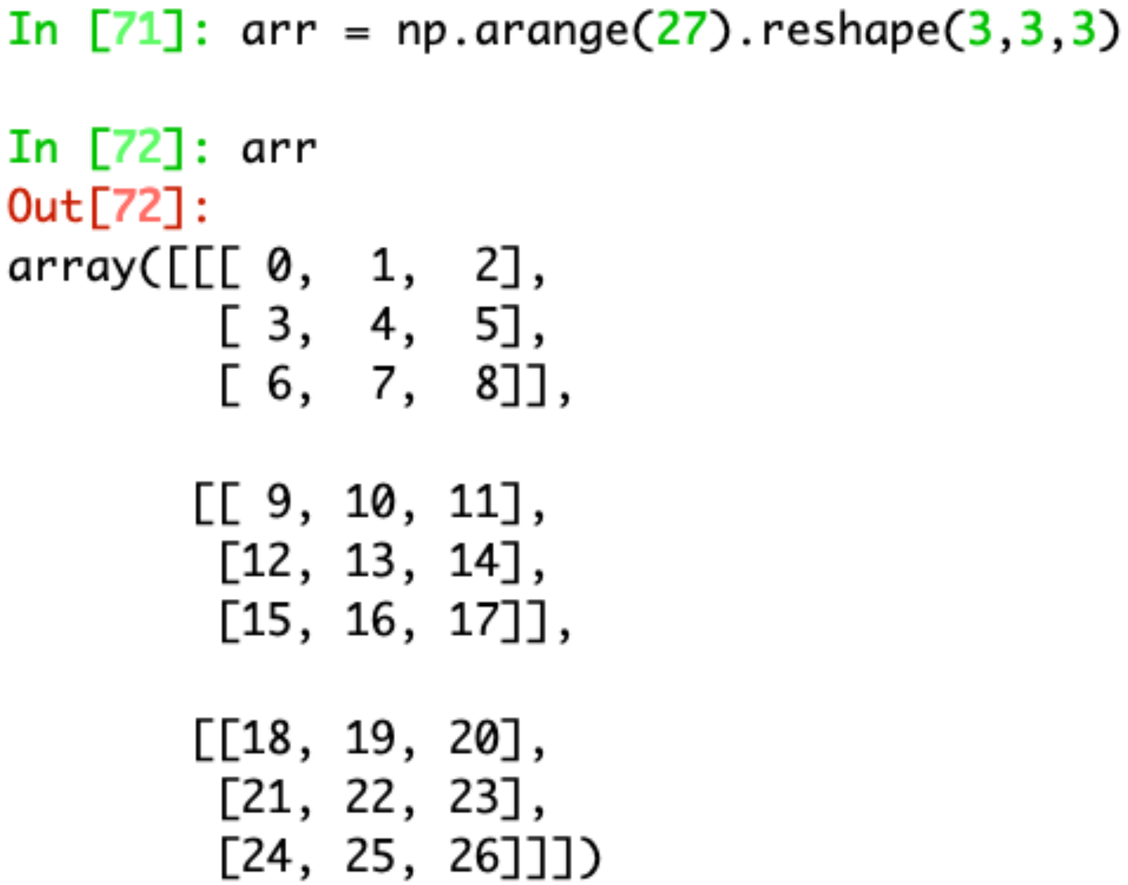
```
In [67]: arr[:, 1]
Out[67]: array([ 1,  6, 11, 16, 21])
```

NDArray – Indexing

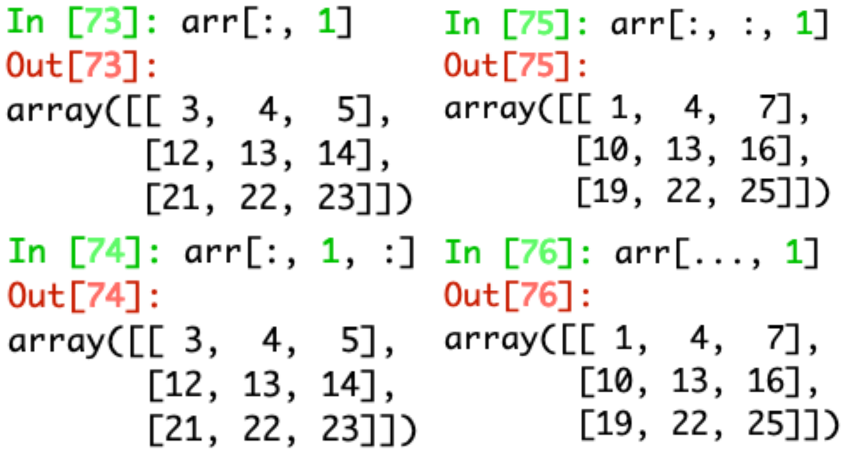
Slice Indexing

ndarray[start:stop:step]

Ignore all dimensions



```
In [71]: arr = np.arange(27).reshape(3,3,3)

In [72]: arr
Out[72]:
array([[[ 0,  1,  2],
        [ 3,  4,  5],
        [ 6,  7,  8]],

       [[ 9, 10, 11],
        [12, 13, 14],
        [15, 16, 17]],

       [[18, 19, 20],
        [21, 22, 23],
        [24, 25, 26]]])
```



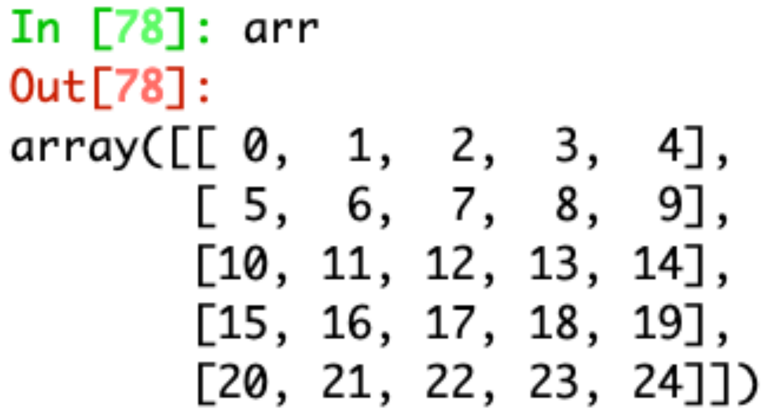
```
In [73]: arr[:, 1]
Out[73]:
array([[ 3,  4,  5],
       [12, 13, 14],
       [21, 22, 23]])

In [74]: arr[:, 1, :]
Out[74]:
array([[ 3,  4,  5],
       [12, 13, 14],
       [21, 22, 23]])
```

```
In [75]: arr[:, :, 1]
Out[75]:
array([[ 1,  4,  7],
       [10, 13, 16],
       [19, 22, 25]])

In [76]: arr[..., 1]
Out[76]:
array([[ 1,  4,  7],
       [10, 13, 16],
       [19, 22, 25]])
```

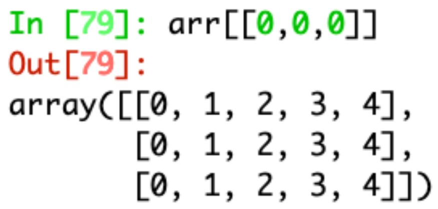
NDArray – Indexing

Slice Indexing

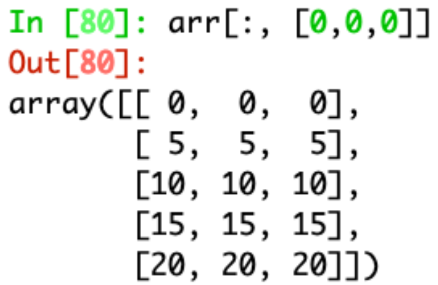
ndarray[start:stop:step]



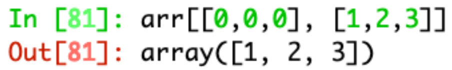
```
In [78]: arr
Out[78]:
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24]])
```



```
In [79]: arr[[0,0,0]]
Out[79]:
array([[0, 1, 2, 3, 4],
       [0, 1, 2, 3, 4],
       [0, 1, 2, 3, 4]])
```



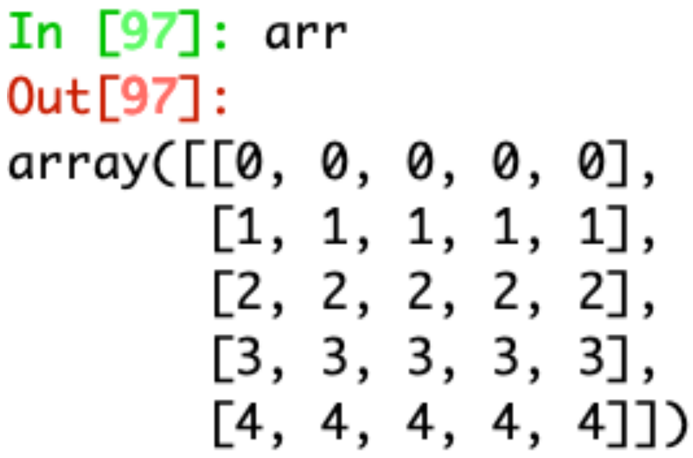
```
In [80]: arr[:, [0,0,0]]
Out[80]:
array([[ 0,  0,  0],
       [ 5,  5,  5],
       [10, 10, 10],
       [15, 15, 15],
       [20, 20, 20]])
```



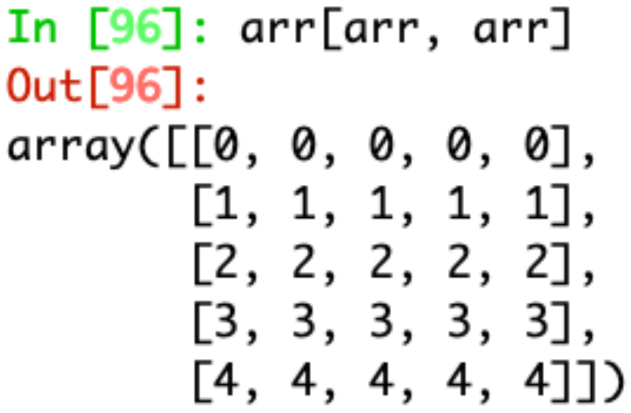
```
In [81]: arr[[0,0,0], [1,2,3]]
Out[81]: array([1, 2, 3])
```

Advanced Indexing

1. Integer array indexing



```
In [97]: arr
Out[97]:
array([[0, 0, 0, 0, 0],
       [1, 1, 1, 1, 1],
       [2, 2, 2, 2, 2],
       [3, 3, 3, 3, 3],
       [4, 4, 4, 4, 4]])
```



```
In [96]: arr[arr, arr]
Out[96]:
array([[0, 0, 0, 0, 0],
       [1, 1, 1, 1, 1],
       [2, 2, 2, 2, 2],
       [3, 3, 3, 3, 3],
       [4, 4, 4, 4, 4]])
```

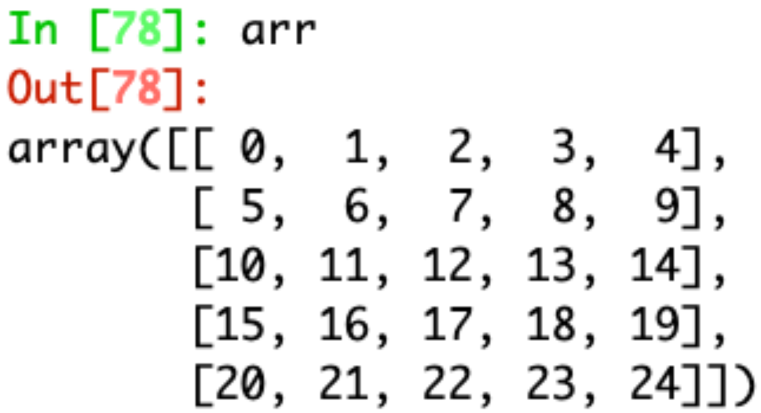
NDArray – Indexing

Slice Indexing

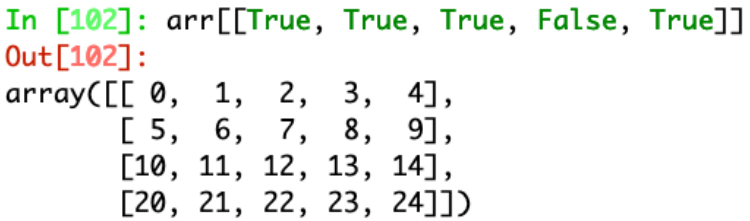
ndarray[start:stop:step]

Advanced Indexing

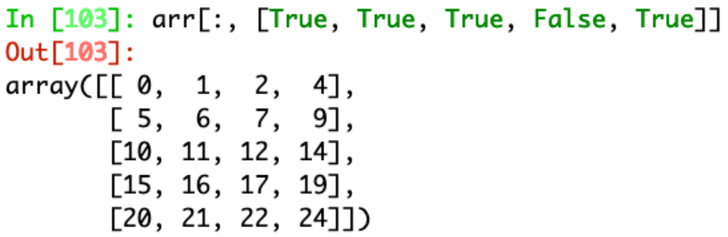
2. Boolean array indexing – Selection



```
In [78]: arr
Out[78]:
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24]])
```



```
In [102]: arr[[True, True, True, False, True]]
Out[102]:
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [20, 21, 22, 23, 24]])
```



```
In [103]: arr[:, [True, True, True, False, True]]
Out[103]:
array([[ 0,  1,  2,  4],
       [ 5,  6,  7,  9],
       [10, 11, 12, 14],
       [15, 16, 17, 19],
       [20, 21, 22, 24]])
```

NDArray – Indexing

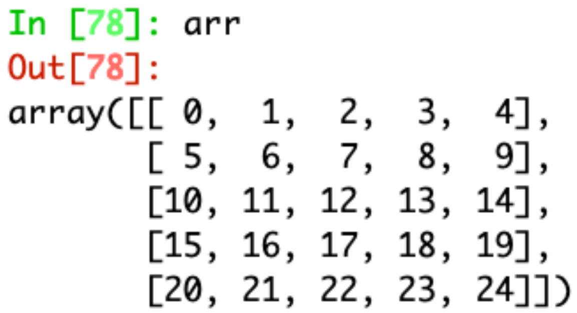
Slice Indexing

```
ndarray[start:stop:step]
```

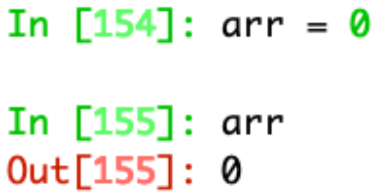
Advanced Indexing creates copies!

NDArray – Editing

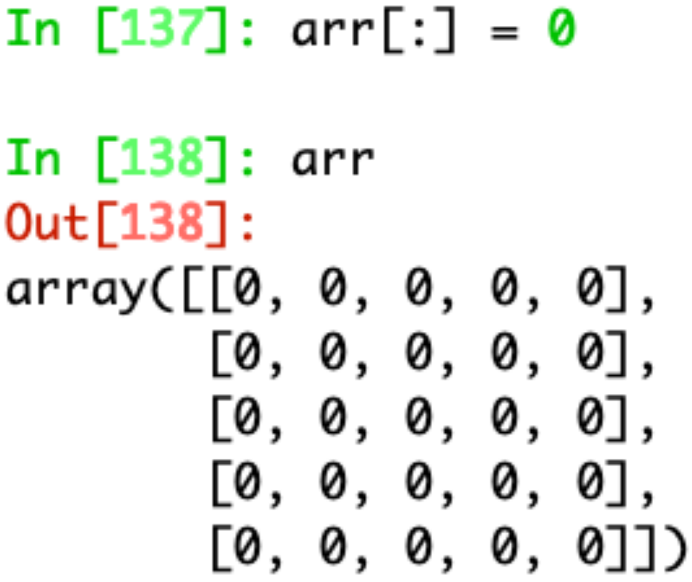
Edit with slice Indexing



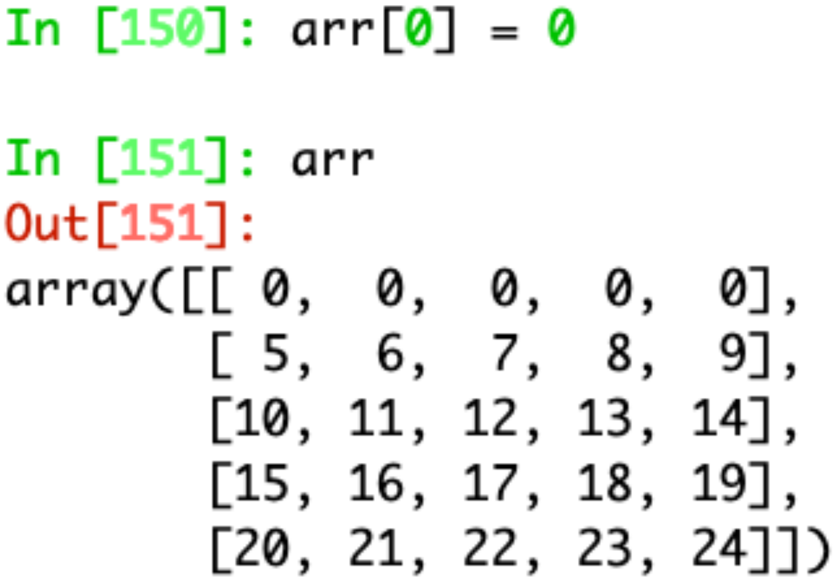
```
In [78]: arr
Out[78]:
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24]])
```



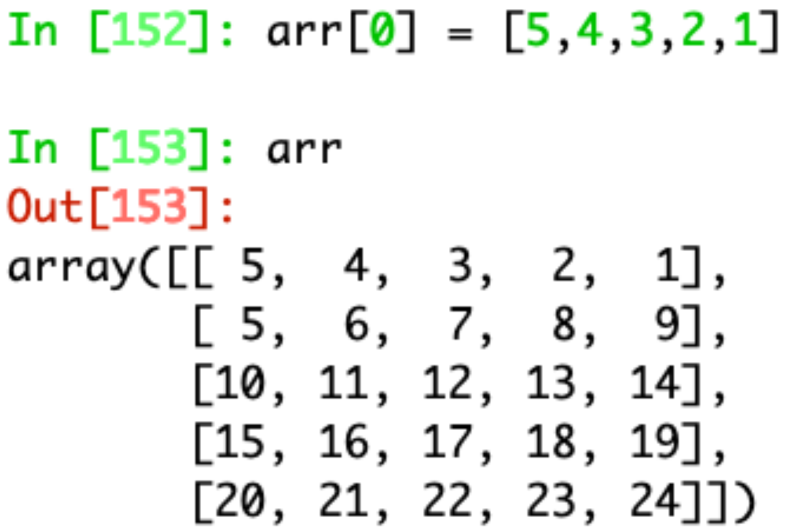
```
In [154]: arr = 0

In [155]: arr
Out[155]: 0
```



```
In [137]: arr[:] = 0

In [138]: arr
Out[138]:
array([[0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0]])
```

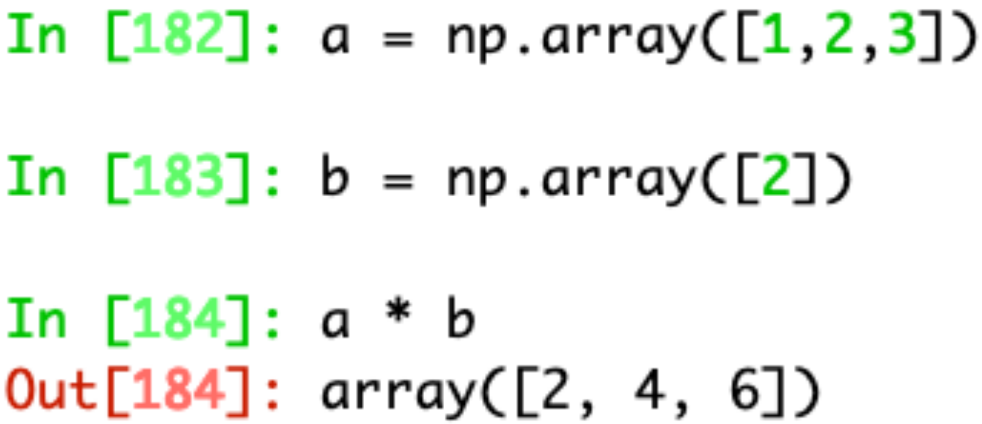


```
In [150]: arr[0] = 0

In [151]: arr
Out[151]:
array([[ 0,  0,  0,  0,  0],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24]])
```



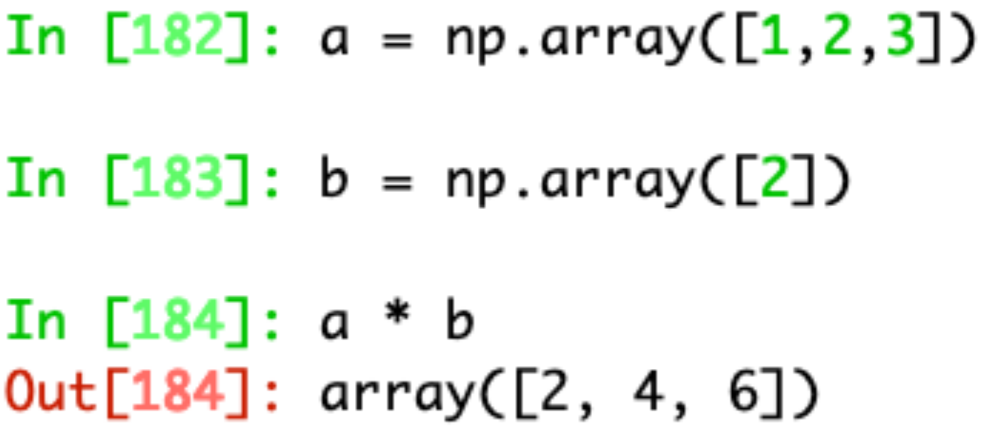
```
In [152]: arr[0] = [5,4,3,2,1]

In [153]: arr
Out[153]:
array([[ 5,  4,  3,  2,  1],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24]])
```

NDArray – Broadcasting

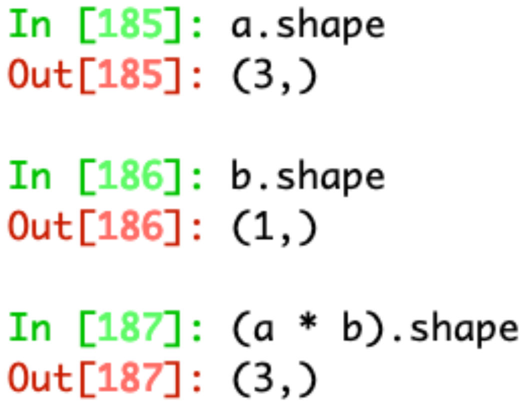


```
In [182]: a = np.array([1,2,3])

In [183]: b = np.array([2])

In [184]: a * b
Out[184]: array([2, 4, 6])
```

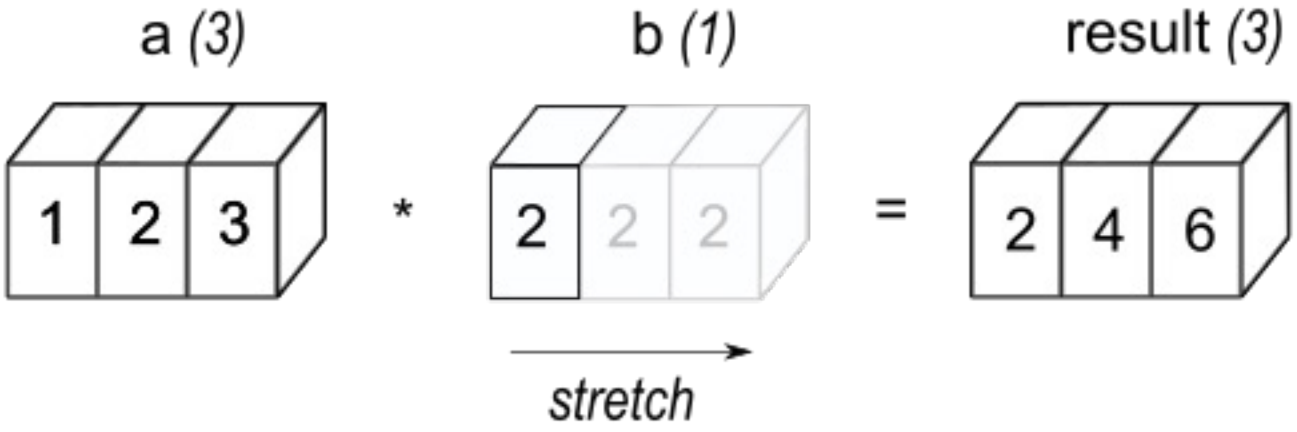
NDArray – Broadcasting



```
In [182]: a = np.array([1,2,3])

In [183]: b = np.array([2])

In [184]: a * b
Out[184]: array([2, 4, 6])
```



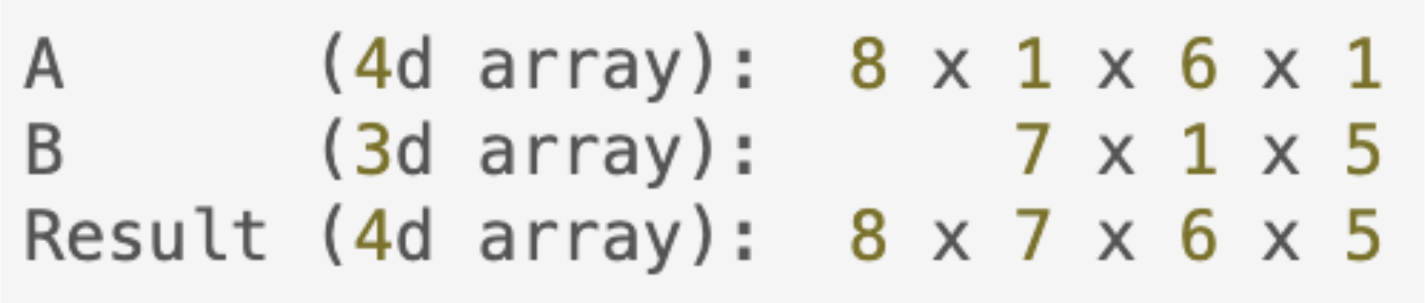
```
In [185]: a.shape
Out[185]: (3,)

In [186]: b.shape
Out[186]: (1,)

In [187]: (a * b).shape
Out[187]: (3,)
```



NDArray – Broadcasting

Can *ndarray*s be broadcast?

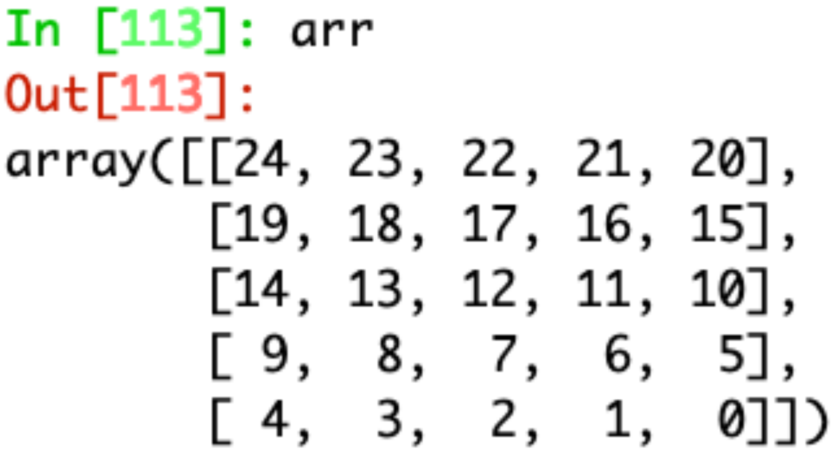
1. Compared the operants from the **rightmost dimension**
2. Two dimensions are compatible when they are **equal**, or **one of them is 1**.



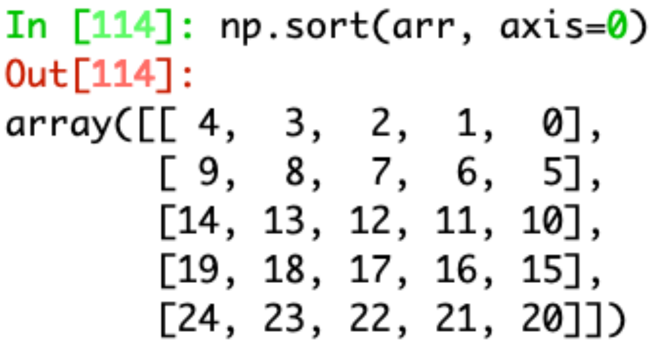
```
A      (4d array):  8 x 1 x 6 x 1
B      (3d array):      7 x 1 x 5
Result (4d array):  8 x 7 x 6 x 5
```

NDArray – Basic Functions

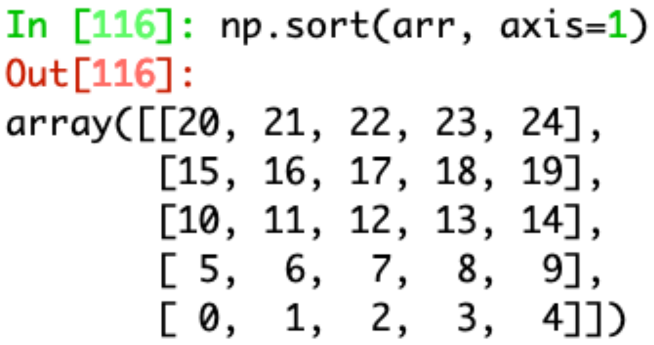
numpy.sort



```
In [113]: arr
Out[113]:
array([[24, 23, 22, 21, 20],
       [19, 18, 17, 16, 15],
       [14, 13, 12, 11, 10],
       [ 9,  8,  7,  6,  5],
       [ 4,  3,  2,  1,  0]])
```



```
In [114]: np.sort(arr, axis=0)
Out[114]:
array([[ 4,  3,  2,  1,  0],
       [ 9,  8,  7,  6,  5],
       [14, 13, 12, 11, 10],
       [19, 18, 17, 16, 15],
       [24, 23, 22, 21, 20]])
```



```
In [116]: np.sort(arr, axis=1)
Out[116]:
array([[20, 21, 22, 23, 24],
       [15, 16, 17, 18, 19],
       [10, 11, 12, 13, 14],
       [ 5,  6,  7,  8,  9],
       [ 0,  1,  2,  3,  4]])
```

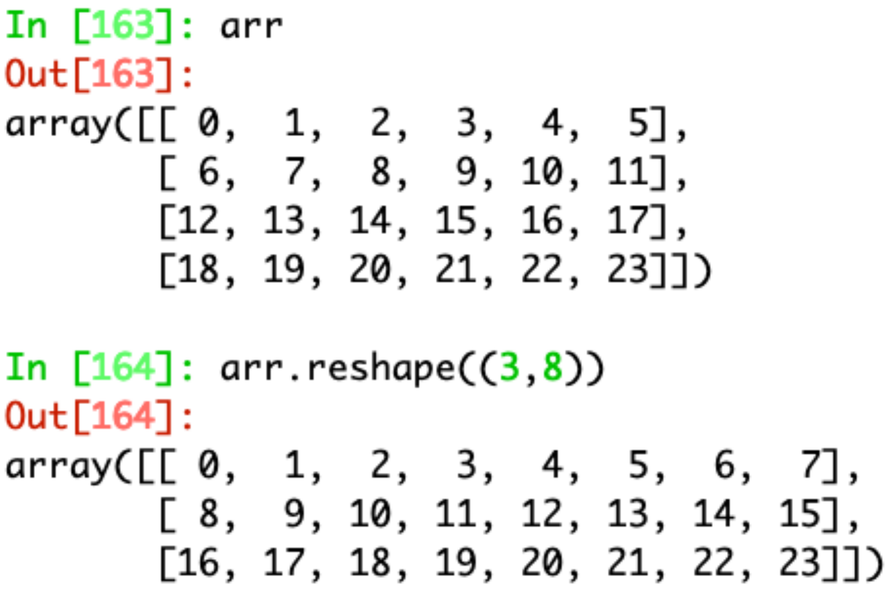
NDArray – Basic Functions

`numpy.reshape (numpy.flatten)`

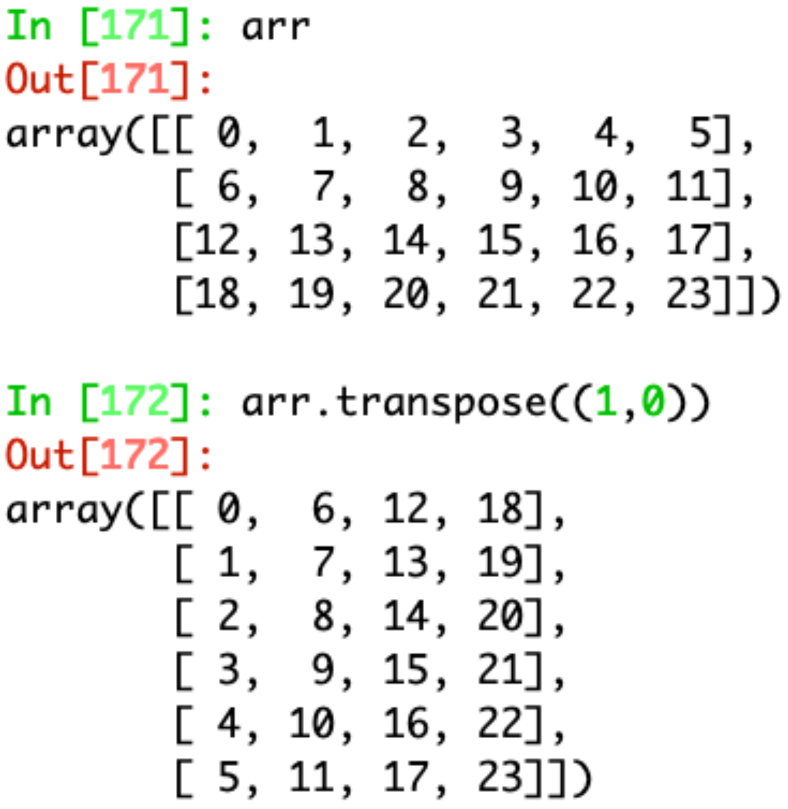
- Reshape ***size***

`numpy.transpose`

- Transpose ***dimension***



```
In [163]: arr
Out[163]:
array([[ 0,  1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10, 11],
       [12, 13, 14, 15, 16, 17],
       [18, 19, 20, 21, 22, 23]])

In [164]: arr.reshape((3,8))
Out[164]:
array([[ 0,  1,  2,  3,  4,  5,  6,  7],
       [ 8,  9, 10, 11, 12, 13, 14, 15],
       [16, 17, 18, 19, 20, 21, 22, 23]])
```



```
In [171]: arr
Out[171]:
array([[ 0,  1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10, 11],
       [12, 13, 14, 15, 16, 17],
       [18, 19, 20, 21, 22, 23]])

In [172]: arr.transpose((1,0))
Out[172]:
array([[ 0,  6, 12, 18],
       [ 1,  7, 13, 19],
       [ 2,  8, 14, 20],
       [ 3,  9, 15, 21],
       [ 4, 10, 16, 22],
       [ 5, 11, 17, 23]])
```

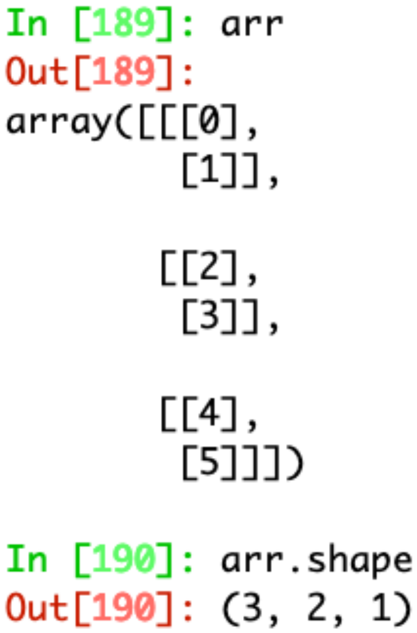
NDArray – Basic Functions

`numpy.squeeze`

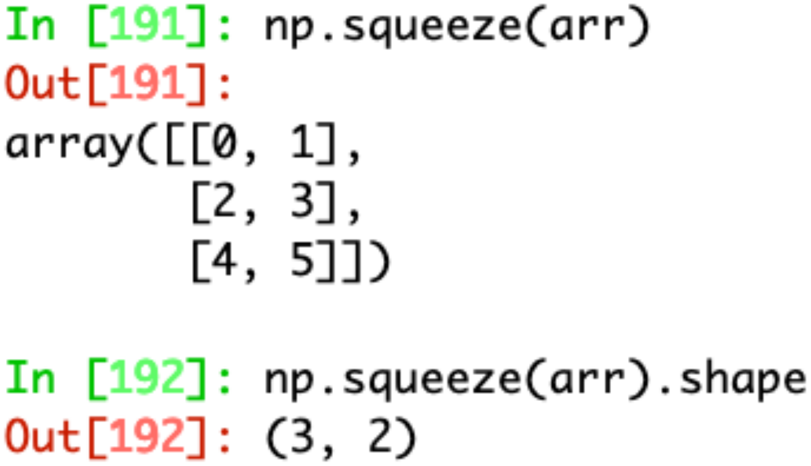
- Remove dimension with size 1

`numpy.newaxis (== None)`

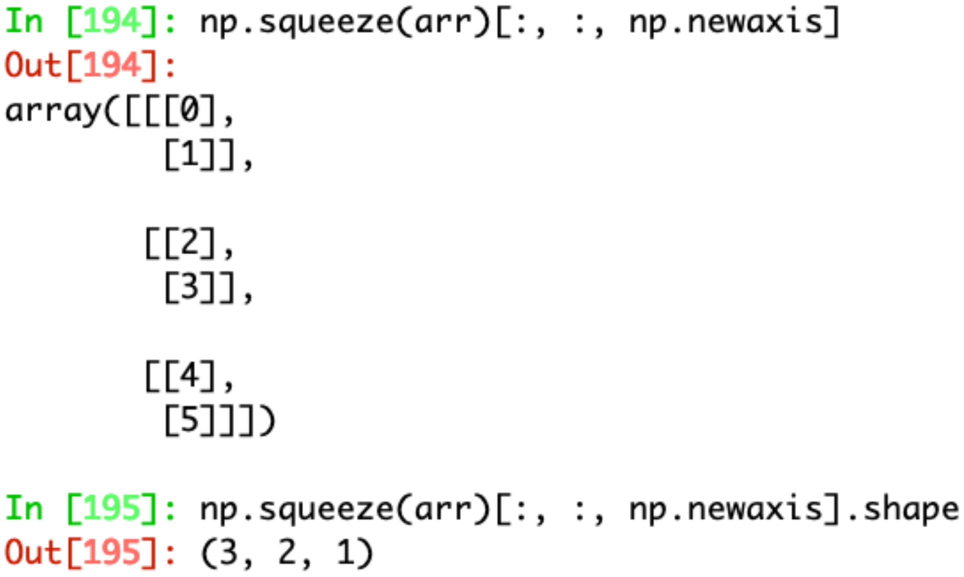
- Add a new dimension with size 1



```
In [189]: arr
Out[189]:
array([[[0],
        [1]],

       [[2],
        [3]],

       [[4],
        [5]]])

In [190]: arr.shape
Out[190]: (3, 2, 1)
```



```
In [191]: np.squeeze(arr)
Out[191]:
array([[0, 1],
       [2, 3],
       [4, 5]])

In [192]: np.squeeze(arr).shape
Out[192]: (3, 2)
```



```
In [194]: np.squeeze(arr)[:, :, np.newaxis]
Out[194]:
array([[[0],
        [1]],

       [[2],
        [3]],

       [[4],
        [5]]])

In [195]: np.squeeze(arr)[:, :, np.newaxis].shape
Out[195]: (3, 2, 1)
```

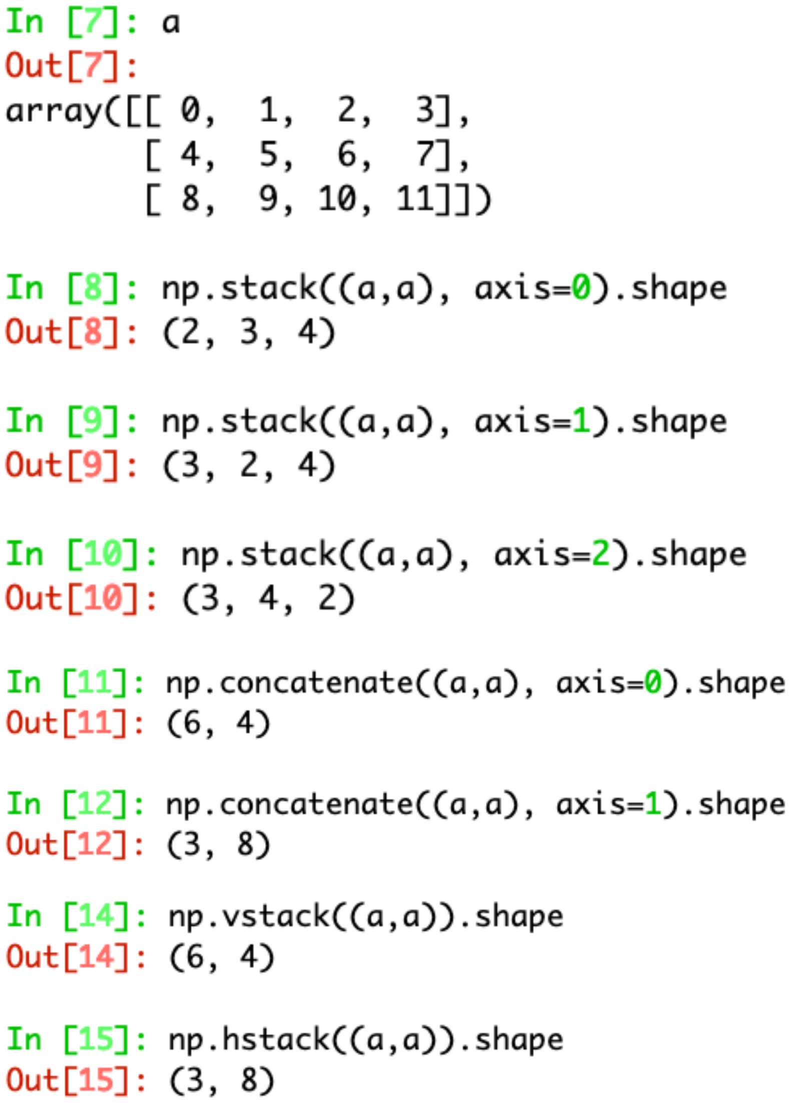
NDArray – Basic Functions

numpy.stack

- Combine list of arrays, creating new dimension

numpy.concatenate

- Combine list of arrays, in existing dimensions
- numpy.hstack: partial(numpy.concatenate, axis=0)
- numpy.vstack:
    - For dimension >1: partial(numpy.concatenate, axis=1)
    - For dimension =1: partial(numpy.stack, axis=1)



```
In [7]: a
Out[7]:
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])

In [8]: np.stack((a,a), axis=0).shape
Out[8]: (2, 3, 4)

In [9]: np.stack((a,a), axis=1).shape
Out[9]: (3, 2, 4)

In [10]: np.stack((a,a), axis=2).shape
Out[10]: (3, 4, 2)

In [11]: np.concatenate((a,a), axis=0).shape
Out[11]: (6, 4)

In [12]: np.concatenate((a,a), axis=1).shape
Out[12]: (3, 8)

In [14]: np.vstack((a,a)).shape
Out[14]: (6, 4)

In [15]: np.hstack((a,a)).shape
Out[15]: (3, 8)
```

NDArray – Basic Functions

```
numpy.sin
numpy.cos
numpy.tan
numpy.sinh
numpy.cosh
numpy.tanh
numpy.arcsin
numpy.arccos
numpy.arctan
numpy.arctan2
numpy.exp
numpy.exp2
numpy.log
numpy.log10
numpy.log2
numpy.add
numpy.substract
numpy.multiply
numpy.divide
numpy.power
numpy.min
numpy.fmin
numpy.max
numpy.fmax
numpy.sqrt
numpy.fabs
```

NDArray – Basic Functions

linalg: Linear algebra

numpy.linalg.dot

numpy.linalg.multi_dot

numpy.linalg.inner

numpy.linalg.outer

numpy.linalg.matmul (a @ b)

numpy.linalg.matrix_power

numpy.linalg.qr

numpy.linalg.svd

numpy.linalg.eigen

numpy.linalg.solve

numpy.linalg.inv

NDArray – Universal Functions (ufunc)

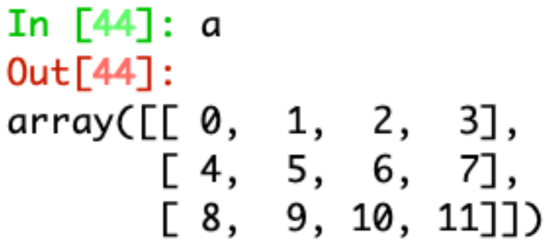
An elegant way to perform batch operations!

numpy.ufunc.reduce
numpy.ufunc.accumulate
numpy.ufunc.outer
numpy.ufunc.at
numpy.ufunc.reduceat

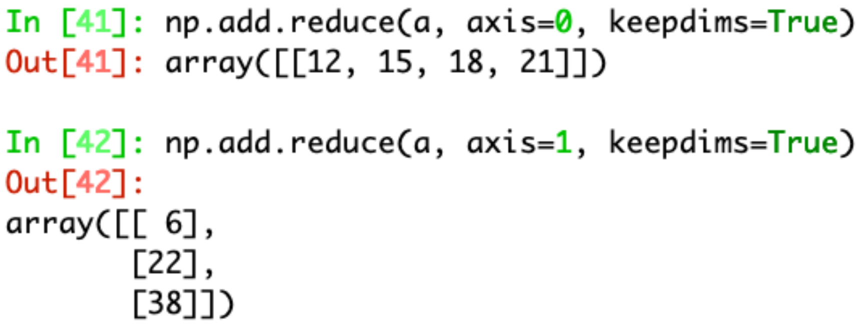
We use numpy.add as example of ufunc!

NDArray – Universal Functions (ufunc)

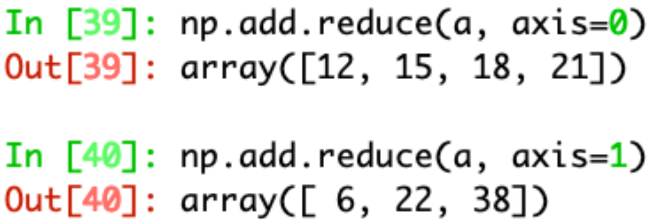
numpy.ufunc.reduce



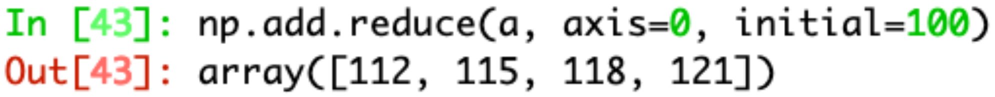
```
In [44]: a
Out[44]:
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```



```
In [41]: np.add.reduce(a, axis=0, keepdims=True)
Out[41]: array([[12, 15, 18, 21]])

In [42]: np.add.reduce(a, axis=1, keepdims=True)
Out[42]:
array([[ 6],
       [22],
       [38]])
```



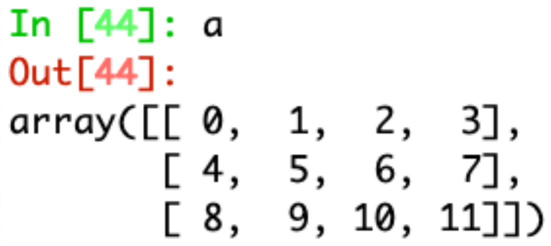
```
In [39]: np.add.reduce(a, axis=0)
Out[39]: array([12, 15, 18, 21])

In [40]: np.add.reduce(a, axis=1)
Out[40]: array([ 6, 22, 38])
```



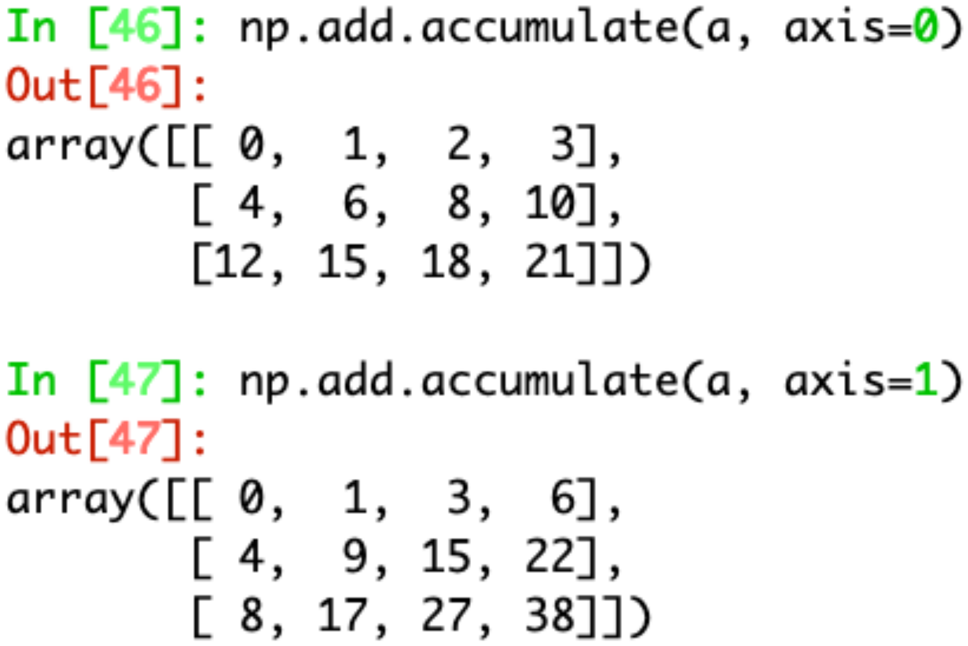
```
In [43]: np.add.reduce(a, axis=0, initial=100)
Out[43]: array([112, 115, 118, 121])
```

NDArray – Universal Functions (ufunc)

numpy.ufunc.accumulate



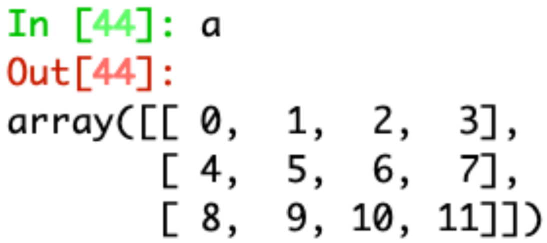
```
In [44]: a
Out[44]:
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```



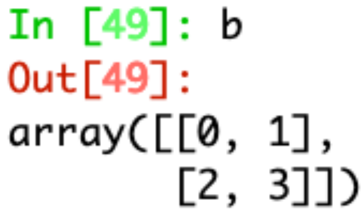
```
In [46]: np.add.accumulate(a, axis=0)
Out[46]:
array([[ 0,  1,  2,  3],
       [ 4,  6,  8, 10],
       [12, 15, 18, 21]])

In [47]: np.add.accumulate(a, axis=1)
Out[47]:
array([[ 0,  1,  3,  6],
       [ 4,  9, 15, 22],
       [ 8, 17, 27, 38]])
```

NDArray – Universal Functions (ufunc)

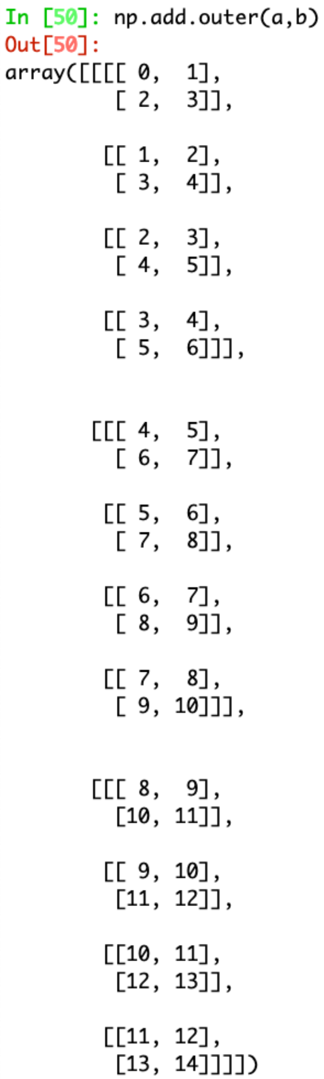
numpy.ufunc.outer



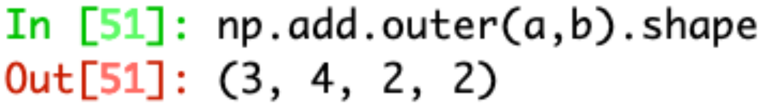
```
In [44]: a
Out[44]:
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```



```
In [49]: b
Out[49]:
array([[0, 1],
       [2, 3]])
```



```
In [50]: np.add.outer(a,b)
Out[50]:
array([[[[ 0,  1],
         [ 2,  3]],

        [[ 1,  2],
         [ 3,  4]],

        [[ 2,  3],
         [ 4,  5]],

        [[ 3,  4],
         [ 5,  6]]],


       [[[ 4,  5],
         [ 6,  7]],

        [[ 5,  6],
         [ 7,  8]],

        [[ 6,  7],
         [ 8,  9]],

        [[ 7,  8],
         [ 9, 10]]],


       [[[ 8,  9],
         [10, 11]],

        [[ 9, 10],
         [11, 12]],

        [[10, 11],
         [12, 13]],

        [[11, 12],
         [13, 14]]]])
```

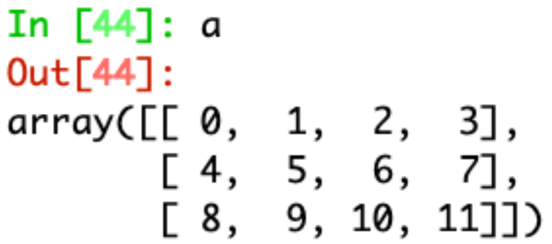


```
In [51]: np.add.outer(a,b).shape
Out[51]: (3, 4, 2, 2)
```

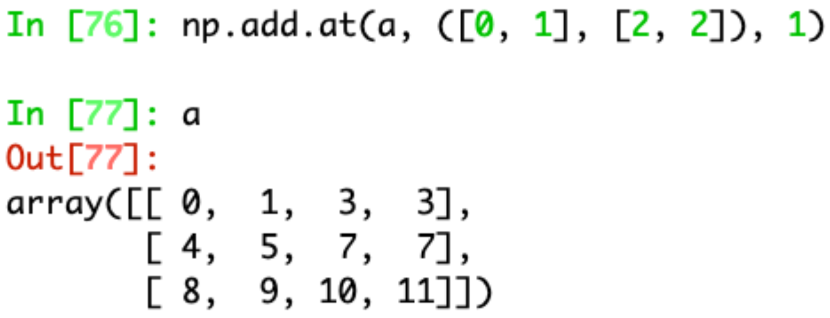
NDArray – Universal Functions (ufunc)

numpy.ufunc.at

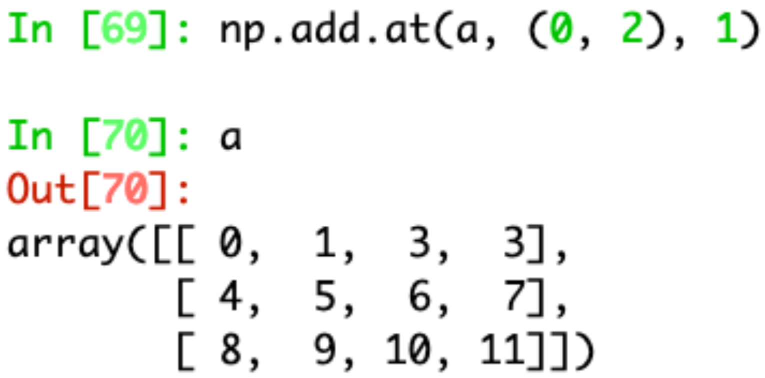
In-place operation!



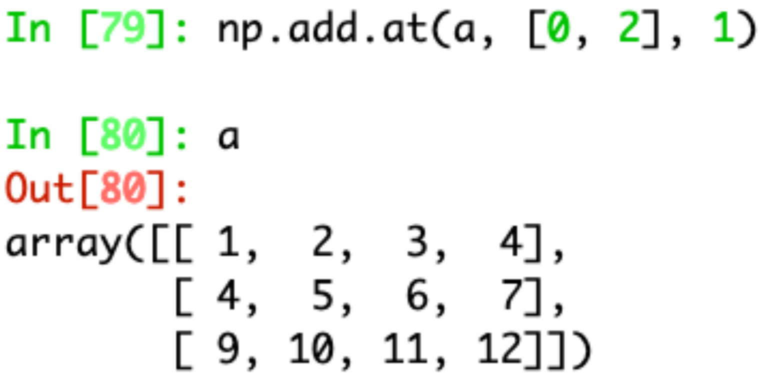
```
In [44]: a
Out[44]:
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```



```
In [76]: np.add.at(a, ([0, 1], [2, 2]), 1)

In [77]: a
Out[77]:
array([[ 0,  1,  3,  3],
       [ 4,  5,  7,  7],
       [ 8,  9, 10, 11]])
```



```
In [69]: np.add.at(a, (0, 2), 1)

In [70]: a
Out[70]:
array([[ 0,  1,  3,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

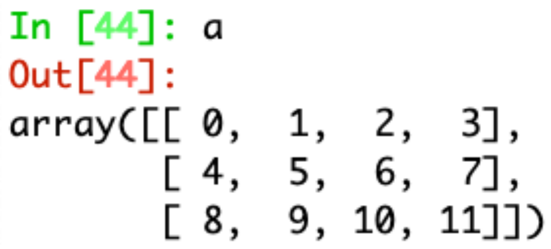


```
In [79]: np.add.at(a, [0, 2], 1)

In [80]: a
Out[80]:
array([[ 1,  2,  3,  4],
       [ 4,  5,  6,  7],
       [ 9, 10, 11, 12]])
```

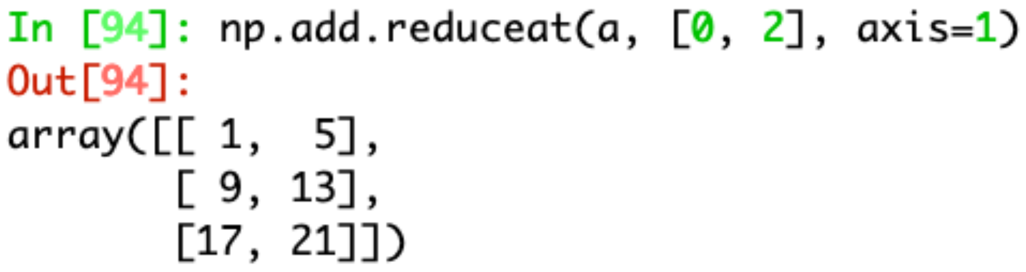
NDArray – Universal Functions (ufunc)

numpy.ufunc.reduceat

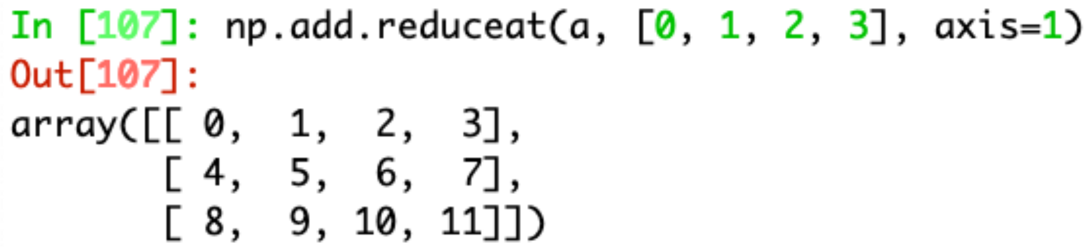
Reduction operation on a single axis, according to the indices



```
In [44]: a
Out[44]:
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```



```
In [94]: np.add.reduceat(a, [0, 2], axis=1)
Out[94]:
array([[ 1,  5],
       [ 9, 13],
       [17, 21]])
```



```
In [107]: np.add.reduceat(a, [0, 1, 2, 3], axis=1)
Out[107]:
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```