Numpy

Creation

Create array

Array of uninitialized values

```
a=np.empty(5)
a
array([ 0.,  0.,  0.,  0.,  0.])
```

```
a=np.empty((2,2))
a
array([[  1.06545119e-311,   1.06546178e-311],
       [  1.06546172e-311,   1.06546173e-311]])
```

Array of 0s

```
a=np.zeros(10)
a
array([ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.])
```

```
a=np.zeros((10, 2)) # add more numbers to the tuple to get higher dimension
a
array([[ 0.,  0.],
       [ 0.,  0.],
       [ 0.,  0.],
       [ 0.,  0.],
       [ 0.,  0.],
       [ 0.,  0.],
       [ 0.,  0.],
       [ 0.,  0.],
       [ 0.,  0.],
       [ 0.,  0.]])
```

Array of 1s

```
a=np.ones(10)
a
array([ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.])
```

```
a=np.ones((10, 2))
a
```

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

Array from list

```
a=np.array([3,5,7]) #tuple works as well
a
array([3, 5, 7])

a=np.array([[3,5,7], [11, 11, 11]])
a
array([[ 3,  5,  7],
       [11, 11, 11]])
```

Generate array

Array from 0..n

```
a=np.arange(15)
a
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14])
```

Array from n..m

```
a=np.arange(5,15)
a
array([ 5,  6,  7,  8,  9, 10, 11, 12, 13, 14])
```

Array from n..m with custom step increment

```
a=np.arange(5,15, 2)
a
array([ 5,  7,  9, 11, 13])
```

Array from n..m spaced evenly

```
a=np.linspace(5,15,3)
```

```
a
array([  5.,  10.,  15.])
```

```
# Create numpy array
a=np.arange(15).reshape(3,5)
a
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
```

```
# Create numpy array and get number of dimensions
a=np.arange(15).reshape(3,5)
a.ndim
2
```

```
 # Create numpy array and get shape
a=np.arange(15).reshape(3,5) # Create numpy array
a.shape
(3, 5)
```

```
# Create numpy array and get NUMBER OF TOTAL ELEMENTS
a=np.arange(15).reshape(3,5)
a.size
15
```

Operations

Print

Printing an array is just like printing anything else, but by default if you have too many elements numpy will cut out the middle and only show you the corner cases.

```
a=np.arange(10000).reshape(100,100)
a
array([[   0,    1,    2, ...,   97,   98,   99],
       [ 100,  101,  102, ...,  197,  198,  199],
       [ 200,  201,  202, ...,  297,  298,  299],
       ...,
       [9700, 9701, 9702, ..., 9797, 9798, 9799],
       [9800, 9801, 9802, ..., 9897, 9898, 9899],
       [9900, 9901, 9902, ..., 9997, 9998, 9999]])
```

If you want to see the full context, use np.set_printoptions(threshold=np.inf). By default threshold is set to 3.

Access

NOTE: This happens just like normal python lists.

Element by index

```
a=np.array([2,3,5,7,11])
a[0]
2
```

Elements by slice

```
a=np.arange(10)
a[5:7]
array([5, 6])
```

```
a=np.arange(10)
a[:7]
array([0, 1, 2, 3, 4, 5, 6])
```

```
a=np.arange(10)
a[7:]
array([7, 8, 9])
```

Elements by slice (multidimensional)

```
a=np.arange(100).reshape(10, 10)
print('--ORIGINAL--')
print(a)
print('--DISCARDED FIRST AND LAST IN BOTH DIRECTIONS--')
print(a[1:9, 1:9])
--ORIGINAL--
[[ 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]]
```

```
--DISCARDED HEAD AND TAIL IN BOTH DIRECTIONS--
[[11 12 13 14 15 16 17 18]
 [21 22 23 24 25 26 27 28]
 [31 32 33 34 35 36 37 38]
 [41 42 43 44 45 46 47 48]
 [51 52 53 54 55 56 57 58]
 [61 62 63 64 65 66 67 68]
 [71 72 73 74 75 76 77 78]
 [81 82 83 84 85 86 87 88]]
```

Element in loop

```
a=np.arange(10)**2
for i in a:
    print(i)
0
1
4
9
16
25
36
49
64
81
```

Math

IF THERE'S ONE THING YOU TAKE AWAY FROM THIS IT SHOULD BE THIS: Arithmetic operations in numpy are applied element-wise. That means that the operation is applied to each element.

Add and subtract a scalar

```
a=np.array([2,3,5,7,11])
a+10
array([12, 13, 15, 17, 21])
```

```
a=np.array([2,3,5,7,11])
a-10
array([-8, -7, -5, -3,  1])
```

Multiply and divide a scalar

```
a=np.array([2,3,5,7,11])
```

```
a/2
array([ 1. ,  1.5,  2.5,  3.5,  5.5])

a=np.array([2,3,5,7,11])
a*2
array([ 4,  6, 10, 14, 22])
```

Exponents

```
a=np.array([2,3,5,7,11])
a**2
array([  4,   9,  25,  49, 121])
```

Trigonometry

```
a=np.array([2,3,5,7,11])
np.cos(a)
array([-0.41614684, -0.9899925 ,  0.28366219,  0.75390225,  0.0044257 ])
```

Boolean

```
a=np.array([2,3,5,7,11])
a<6
array([ True,  True,  True, False, False], dtype=bool)
```

Aggregation/Grouping

Sum

```
a=np.arange(9).reshape(3, 3)
print(a)
print(a.sum())
[[0 1 2]
 [3 4 5]
 [6 7 8]]
36

a=np.arange(9).reshape(3, 3)
print(a)
print(a.sum(axis=1)) # which axis do you want to sum up? 0 is by column,1 is by row
[[0 1 2]
 [3 4 5]
 [6 7 8]]
[ 3 12 21]
```

Min

```
a=np.arange(9).reshape(3, 3)
print(a)
print(a.min()) # see sum -- axis argument can be applied here to calc by column or row
[[0 1 2]
 [3 4 5]
 [6 7 8]]
0
```

Max

```
a=np.arange(9).reshape(3, 3)
print(a)
print(a.max()) # see sum -- axis argument can be applied here to calc by column or row
[[0 1 2]
 [3 4 5]
 [6 7 8]]
8
```

Average

```
a=np.arange(9).reshape(3, 3)
print(a)
print(np.average(a)) # see sum -- axis argument can be applied here to calc by column or row
[[0 1 2]
 [3 4 5]
 [6 7 8]]
4.0
```

Standard Deviation

```
a=np.arange(9).reshape(3, 3)
print(a)
print(np.std(a)) # see sum -- axis argument can be applied here to calc by column or row
[[0 1 2]
 [3 4 5]
 [6 7 8]]
2.58198889747
```

CSV Loading

If you really want, you can load a CSV using numpy directly. It’s probably better to use pandas for this though.

For a 2 column CSV…

```
x, y = np.loadtxt(‘example.csv’, delimeter=’,’, unpack=True)
```