Week2: Intro to NumPy

Agenda

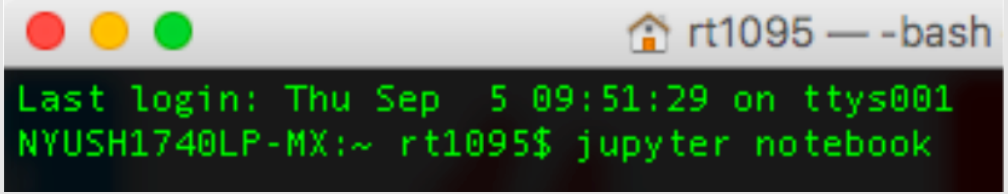
- Jupyter notebook installation
- Intro to NumPy

Jupyter notebook installation

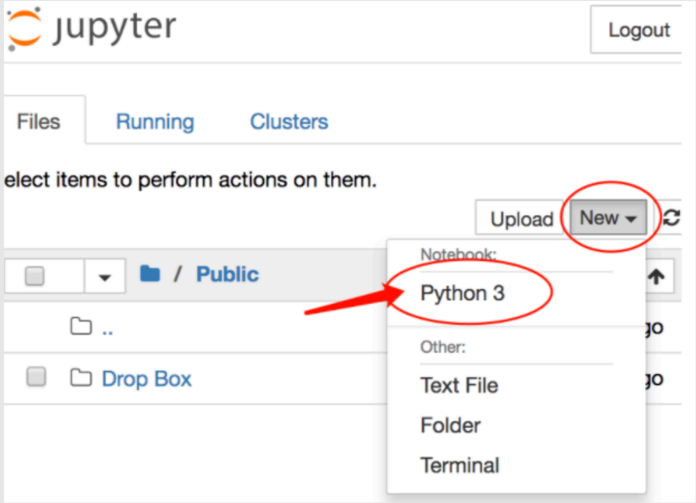
Windows	Mac
Simple way: Download & Install Anaconda In cmd, enter $ anaconda-navigator Hard way: $ pip install jupyter Then configure system Path	Simple way: Download & Install Anaconda Hard way: $ pip install jupyter

Creating a new notebook

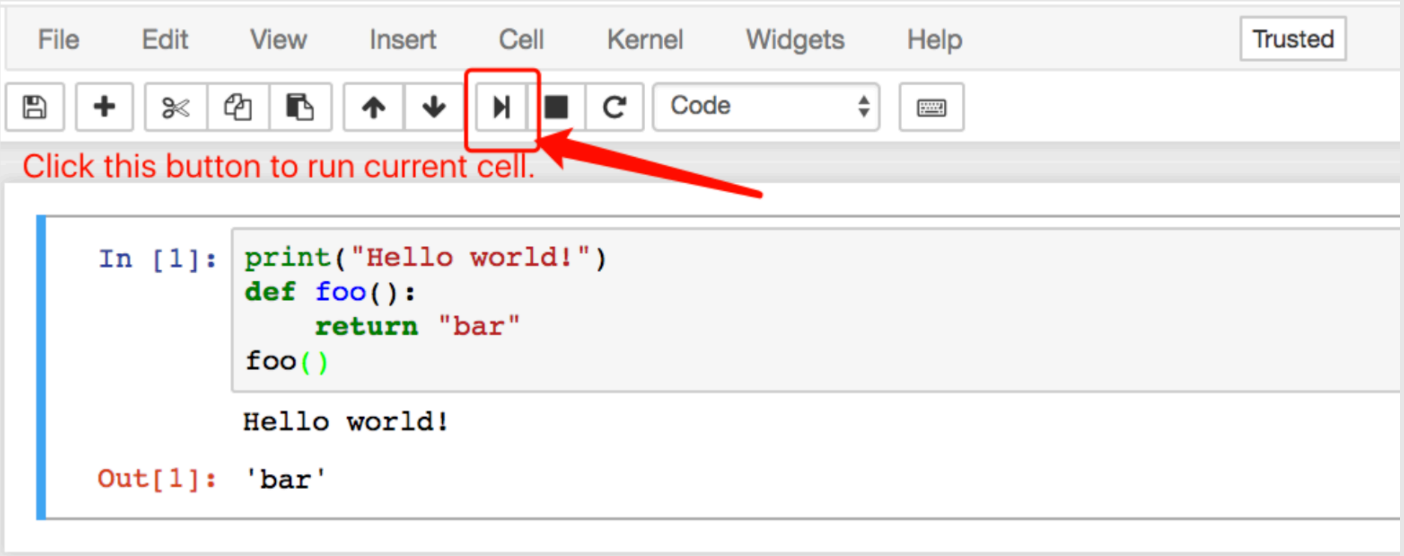
1. Enter "jupyter notebook" in terminal/cmd



2. Select New-Python 3



Executing python code



Note on code execution

- Execution order matters
  For example, variable define in `In [1]` will exist in `In [2]`

- Printed values are always displayed

- Last returned values will be displayed as `out [N]`
  Unless a semicolon ‘;’ is at the end of the line

Cell magics

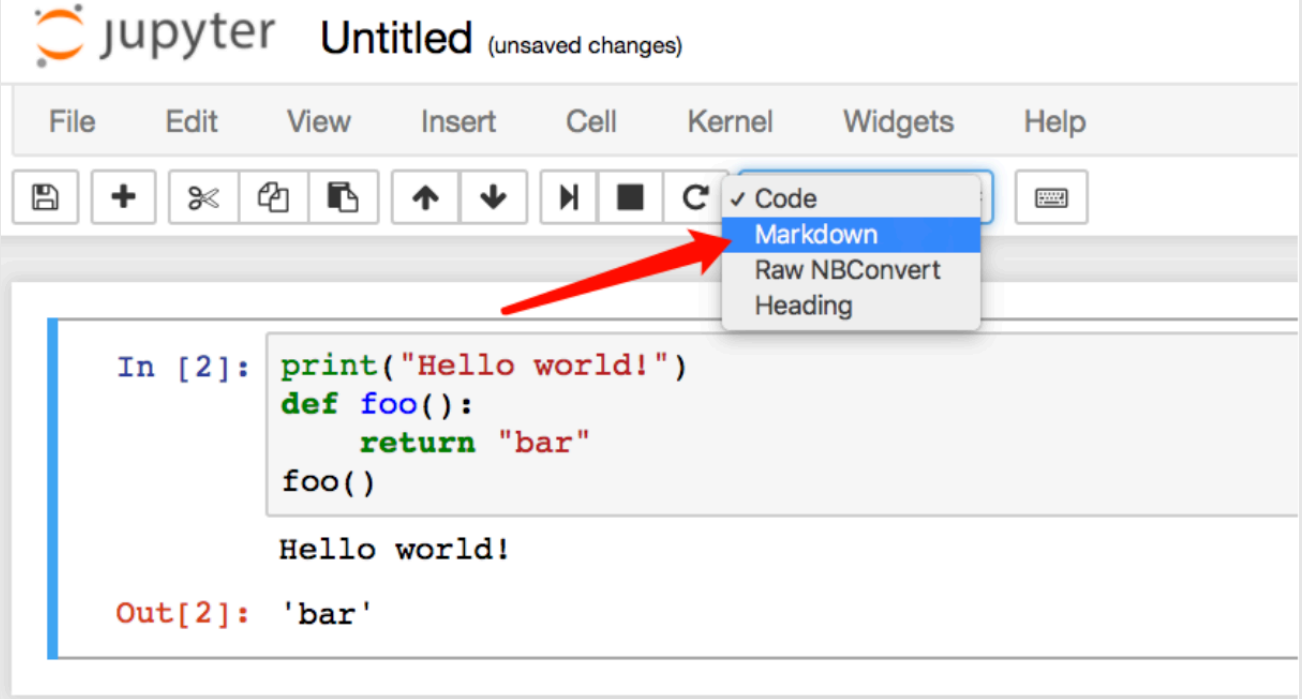
```
In [4]: %matplotlib inline
```

- Cell magics can perform additional features such as showing current working directory, show the time, inserting html, etc.

- % means single line magic code

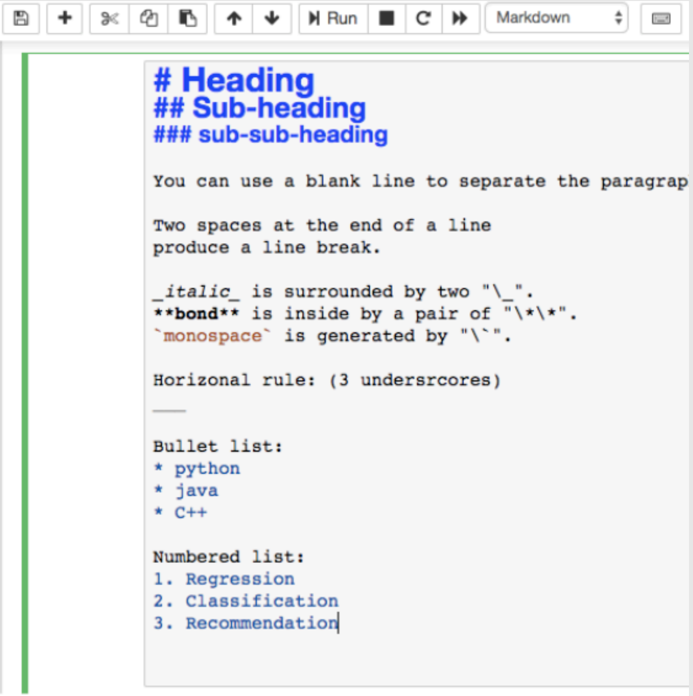
- %% means multiple lines magic code

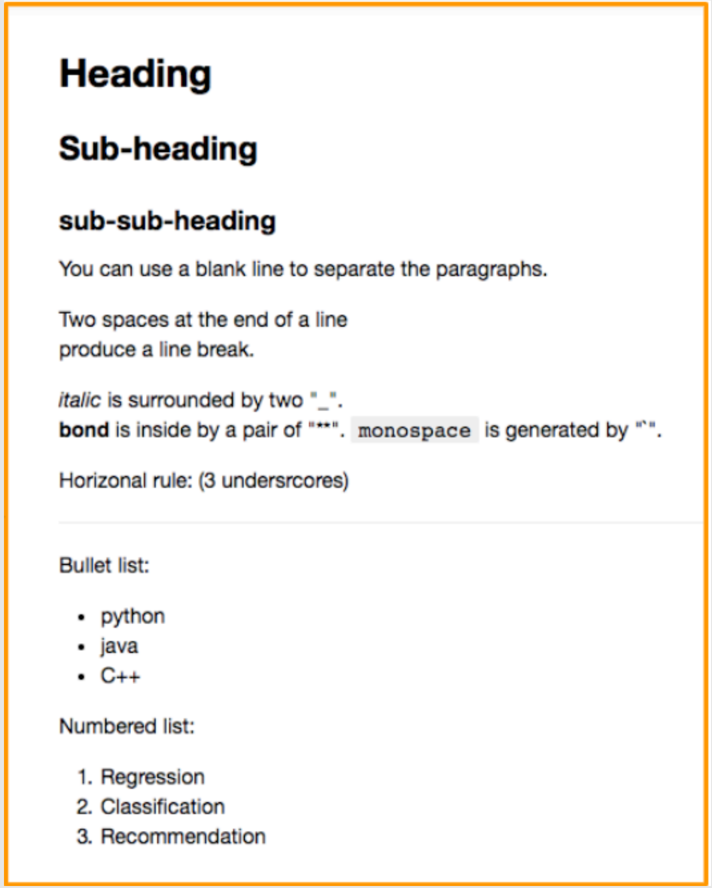
Markdown


jupyter
Untitled (unsaved changes)
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
✓ Code
Markdown
Raw NBConvert
Heading
In [2]:
print("Hello world!")
def foo():
return "bar"
foo()
Hello world!
Out[2]:
'bar'

Markdown

- Lightweight markup language with plain text formatting syntax. Similar to HTML, but more handy





NumPy

NumPy

NumPy is a fundamental package for scientific computing with Python.
It provides a high-performance multidimensional array object, and tools for working with these arrays.

Introduction to NumPy

Simple Array Creation:

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

Checking array type

```
>>> type(array)
numpy.ndarray
```

Checking data type

```
>>> array.dtype
dtype('int32')  (or 'int64' if Mac)
```

Checking data dimension

```
>>> array.ndim
1
```

Checking array shape

```
# This returns a tuple
>>> array.shape
(4,)
```

Array indexing

```
>>> array[0]
1
```

Array slicing

```
>>> array[0:3]
array([1, 2, 3])
```

Modify array in place (Mutable)

```
>>> array[0] = 66
>>> array
array([66, 2, 3, 4])
```

Creating a new NumPy array

Simple Array Creation from python list:

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

Using np.arange( )

```
>>> np.arange(0, 2*pi, pi/4)
array([ 0.000, 0.785, 1.571,
2.356, 3.142, 3.927, 4.712,
5.497])
```

Using np.zeros( ), ones( ), empty( )

```
>>> np.ones((2, 3),
...             dtype='float32')
array([[ 1., 1., 1.],
       [ 1., 1., 1.]],
        dtype=float32)
```

Creating an identity matrix

```
>>> array = np.identity(4)
>>> array
array([[ 1., 0., 0., 0.],
       [ 0., 1., 0., 0.],
       [ 0., 0., 1., 0.],
       [ 0., 0., 0., 1.]])
```

And you can also use the preprocessed data from Pandas module.

Array Math

Add, multiply, power, etc:

```
>>> array1 = np.array([1, 2, 3, 4])
>>> array2 = np.array([2, 3, 4, 5])
>>> array1 + array2
array([3, 5, 7, 9])

>>> array1 * array2
array([2, 6, 12, 20])

>>> array1 ** array2
array([1, 8, 81, 1024])
```

Working in float64

```
# Similar to python range( ), but returns a numpy
# array instead.
>>> array = np.arange(10.)
>>> array
array([0., 1., ..., 9.])

>>> array * np.pi
array([0., 3.14159265, ...,
28.27433388])

>>> np.sin(array)
array([0., 0.84147098, ...,
0.41211849])
```

Beware of dtype

This ndarray has dtype int64

```
>>> array1 = np.array([1, 2, 3, 4])
>>> array1.dtype
dtype('int64')

>>> array1[0] = 6.66
>>> array1
array([6, 2, 3, 4])

>>> array1.fill(-6.66)
>>> array1
array([-6, -6, -6, -6])
```

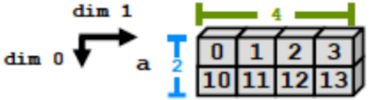
Working in multi – dimension

Creating a 2D array from python list

```
>>> array1 = np.array([[0, 1, 2, 3]
...                    [10, 11, 12, 13]])
>>> array1
array([[0, 1, 2, 3],
       [10, 11, 12, 13]])
```

Checking array shape

```
>>> array1.shape
(2, 4)
```



Checking array number of elements

```
>>> array1.size
8
```

Checking array dimension

```
>>> array1.ndim
2
```

Get/set items

```
>>> array1[1, 3]  #(row, col)
13

>>> array1[1, 3] = -1
>>> array1
array([[0, 1, 2, 3],
       [10, 11, 12, -1]])
```

Selecting a column

```
>>> array1[:, 2]  #(row, col)
array([ 2, 12])
```

Similar slicing syntax to python:

```
Lower : Upper : Step
```

Unlike python list, NumPy slices are references

Changing the slice will change the original array

```
>>> array1 = np.array([0, 1, 2, 3, 4])

# Create a slice
>>> slice = array1[2:4]
>>> slice
array([2, 3])

# Change the slice
>>> slice[0] = 10

# Changing the slice modifies original!
>>> array1
array([0, 1, 10, 3, 4])
```

Use array1.copy( ) for deep copy

```
>>> array2 = array1.copy()
```

Linear algebra operations

Getting the transpose

```
>>> array1 = np.array([[0, 1, 2, 3]
...                    [10, 11, 12, 13]])
>>> array1.T
array([[ 0, 10],
       [ 1, 11],
       [ 2, 12],
       [ 3, 13]])
```

Computing dot product

```
>>> array2 = np.array([1, 2])
>>> array2.dot(array1)
array([20, 23, 26, 29])
```

Compute determinant of a square matrix

```
>>> sq = np.array([[1,5,3],
...                [5,7,6],
...                [3,6,9]])
>>> np.linalg.det(sq)
-80.999999999999996
```

Compute eigenvalues/eigenvectors of a square matrix

```
>>> evals,evecs = np.linalg.eig(sq)
>>> evals
array([16.13076266, -1.84800146,
2.7172388 ])
>>> evecs
array([[-0.34742777, -0.85487543,
-0.38533355],[-0.64037349,  0.51649006,
-0.56847147],[-0.68499324, -0.04925461,
0.72688255]]
```

Array calculation methods

Sum function

```
>>> array1 = np.array([[1, 2, 3]
...                    [4, 5, 6]])
>>> array1.sum()
21
```

Min function

```
>>> array1.min()
1
```

Calculate among an axis

```
>>> array1.sum(axis = 0)## axis 0 is col?
array([5, 7, 9])
```

Other methods

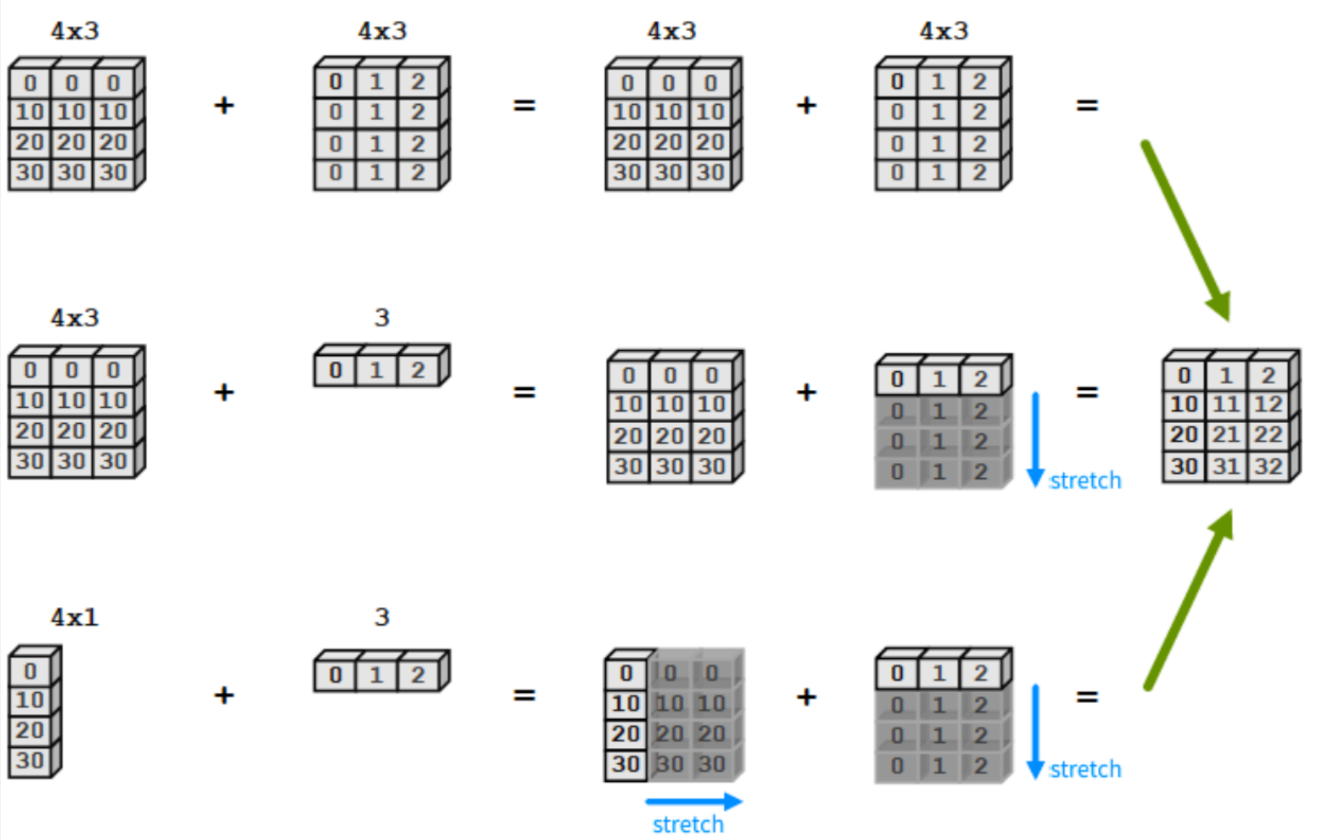
```
sum, prod
min, max, argmin, argmax

mean, std, var

any, all

where
```

Array Broadcasting

- Broadcasting is NumPy trying to be smart when you tell it to perform an operation on arrays that are not the same dimension.

```
>>> x = np.random.randn(3,1)

>>> y = np.random.randn(1,3)

>>> x*y

array([[-0.21522849, -0.67781163,  0.33600751],

       [ 0.00836889,  0.02635585, -0.01306523],

       [-0.02365192, -0.07448618,  0.03692459]])
```

Array Broadcasting

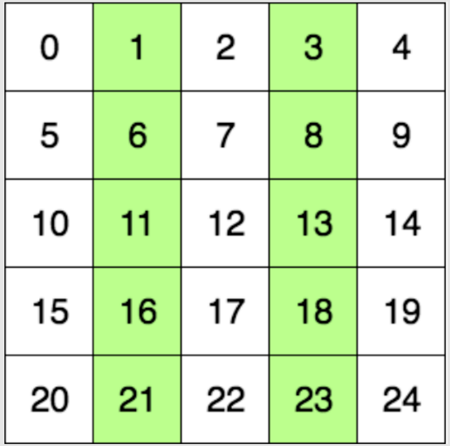


Exercise

Create the array below with the command

```
array = np.arange(25).reshape(5, 5)
```

Extract the slices as color indicated



Exercise

- Create the array below with the command

```
array = np.arange(-15, 15).reshape(5, 6)**2
```

- Compute the row sum of the each row using `np.sum()`
- Divide every number by their corresponding row sum (5, 6) divide with (5,)

```
array([[225, 196, 169, 144, 121, 100],   (Which does not work
       [ 81,  64,  49,  36,  25,  16],    you have to reshape
       [  9,   4,   1,   0,   1,   4],    to broadcast)
       [  9,  16,  25,  36,  49,  64],
       [ 81, 100, 121, 144, 169, 196]])
```

Exercise

Define the following vectors

```
a = np.arange(1, 11)
b = np.arange(10, 0, -1)
c = np.ones(10, dtype=int)
d = 2**a
```

- Write a function `sum_of_squares`
  - Argument: a vector of numeric data x
  - Output: the sum of squares of the elements of the vector $\sum_{i=1}^{n} x_i^2$
  - Calculate the following items

```
sum_of_squares(a)
sum_of_squares(np.concatenate([c, d]))
```

- Write a function `rms_diff`
  - Argument: two vector of numeric data x and y
  - Output: $\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - y_i\ )^2}$
  - If the vectors have different length, the shorter vector will repeat itself until it matches the length of the other vector
  - Calculate the following items

```
rms_diff(a, b)
rms_diff(d, c)
rms_diff(d, [1])
```