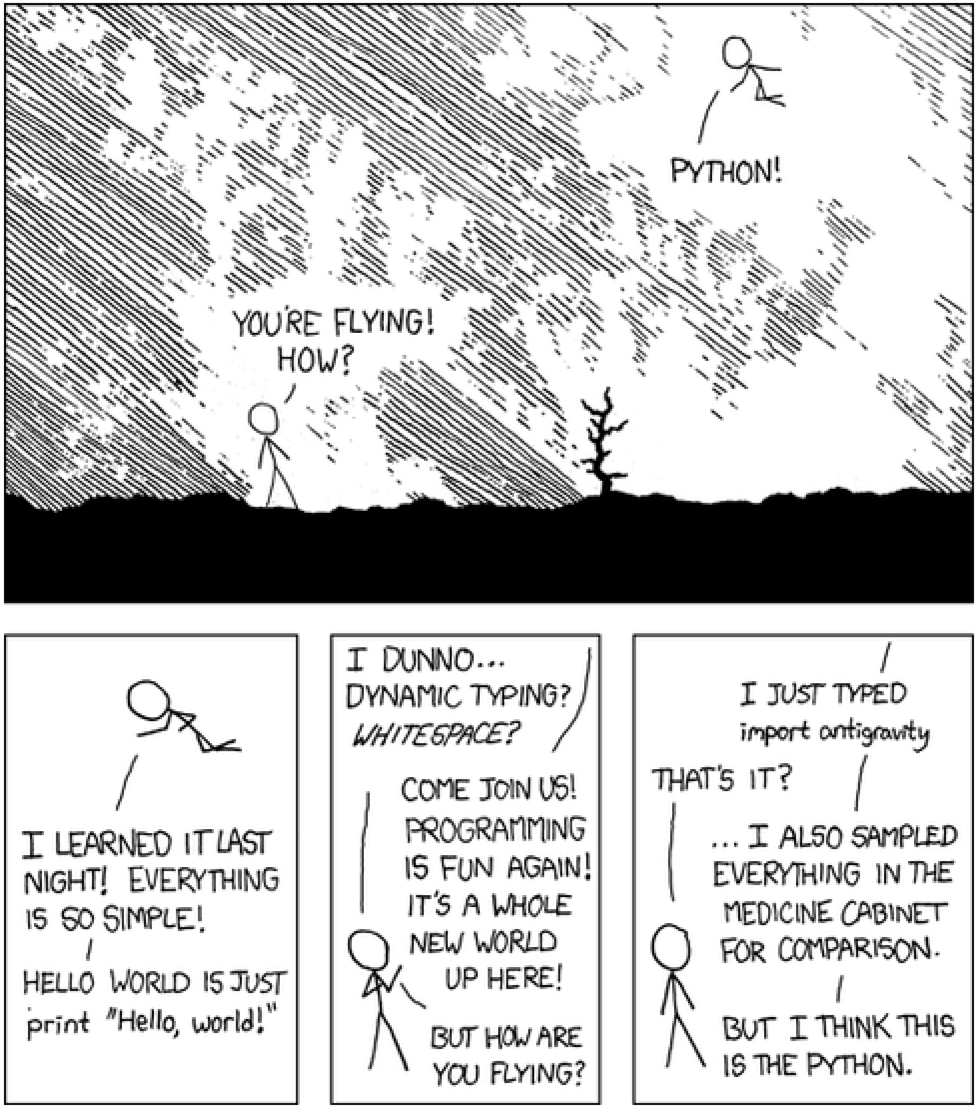
Python for Financial Engineering

금융공학 프로그래밍 I

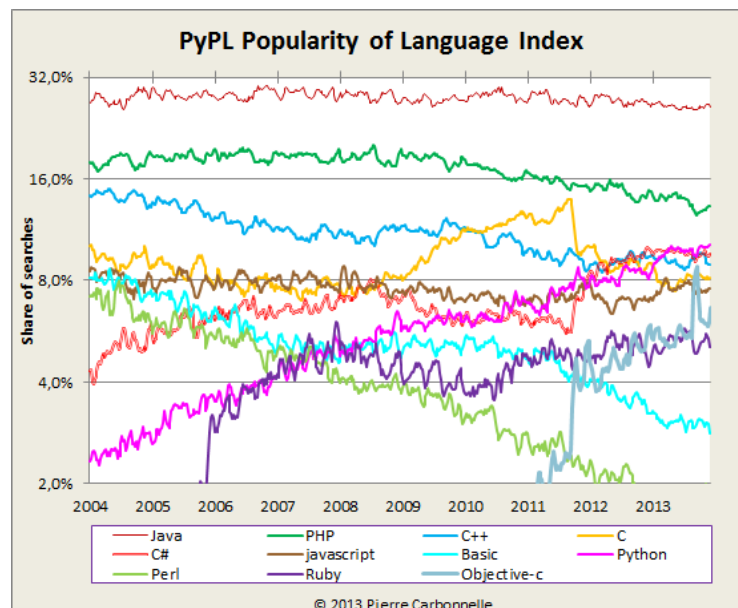
Spring 2017

About Python



About Python

❖ Python is
- a general purpose programming language conceived in 1989 by Dutch programmer Guido van Rossum
- free and open source
- supported by a vast collection of standard and external software libraries
- now one of the most popular programming languages

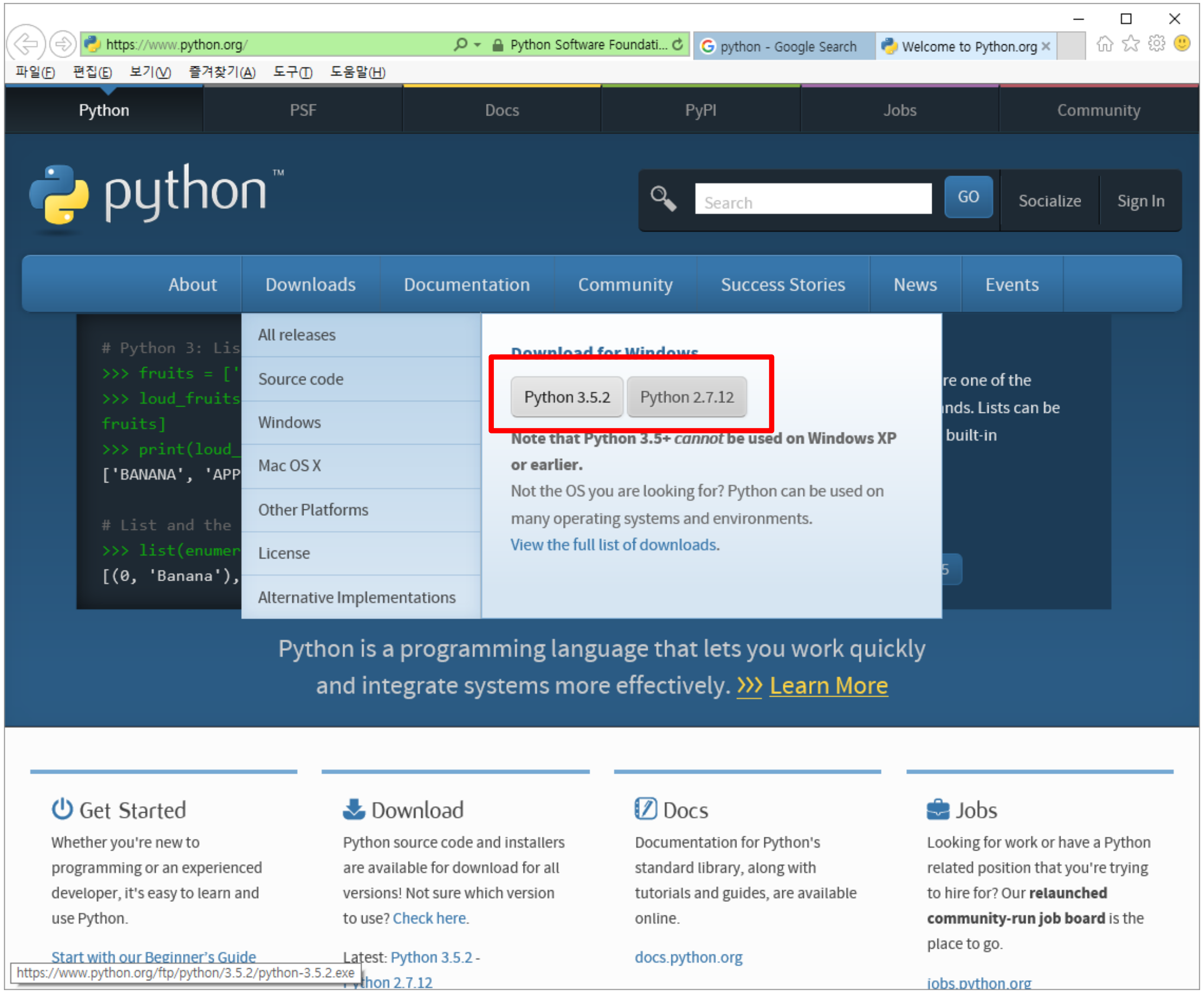


About Python

- Python Ecosystem
  - NumPy
    - Matrix & Array
  - SciPy
    - Linear algebra
    - Integration
    - Interpolation
    - Optimization
    - Distribution & random number generation
    - Signal processing
  - Matplotlib: figures and graphs
  - Pandas, SymPy, statsmodels, scikit-learn …

Python 설치

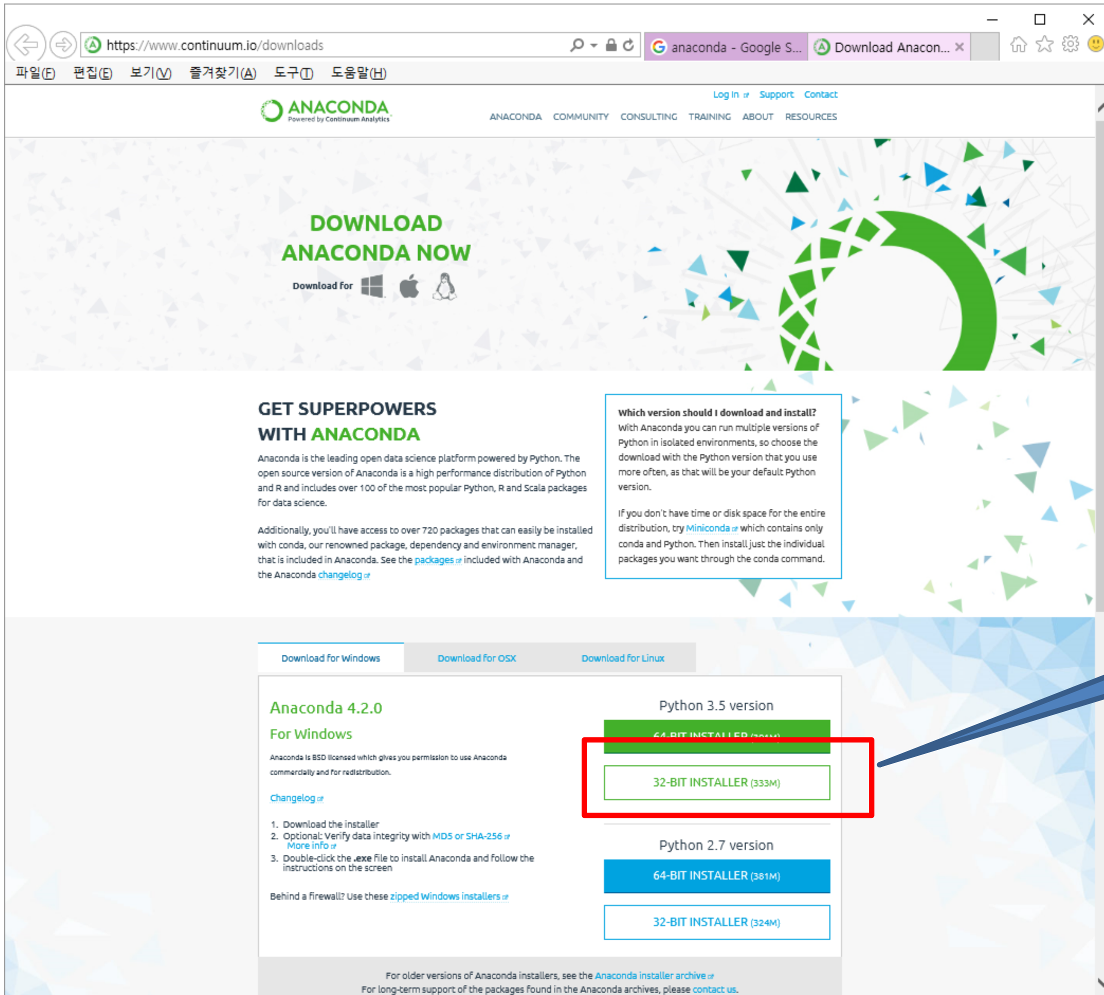
❖ https://www.python.org/



ANACONDA 설치

❖ https://www.continuum.io/downloads

- Python 배포판 –기본 파이썬 라이브러리 외에 유용한 패키지 다수 포함



32-Bit Installer

데이터 타입

Data Types

❖ Boolean (True or False)

```
In [1]: x = True

In [2]: y = 100 < 10

In [3]: y
Out[3]: False

In [4]: type(y)
Out[4]: bool
```

```
In [5]: x + y
Out[5]: 1

In [6]: x * y
Out[6]: 0

In [7]: True + True
Out[7]: 2

In [8]: bools = [True, True, False, True]

In [9]: sum(bools)
Out[9]: 3
```

Data Types

❖ Integer & Floats

```
In [1]: a, b = 1, 2

In [2]: c, d = 2.5, 10.0

In [3]: type(a)
Out[3]: int

In [4]: type(c)
Out[4]: float
```

```
In [5]: 1 / 2          # Integer division in Python 2.x
Out[5]: 0

In [6]: 1.0 / 2.0      # Floating point division
Out[6]: 0.5

In [7]: 1.0 / 2        # Floating point division
Out[7]: 0.5
```

Data Types

❖ String

```
In [23]: t = ‘this is a string object’

In [24]: t.capitalize()
Out[24]: ‘This is a string object’

In [25]: t.split()
Out[25]: [‘this’, ‘is’, ‘a’, ‘string’, ‘object’]

In [26]: t.find(‘string’)
Out[26]: 10

In [27]: t.find(‘Python’)
Out[27]: -1

In [28]: t.replace(‘ ‘, ‘|’)
Out[28]: ‘this|is|a|string|object’
```

Advanced String Formatting

```
r = "{0} {1} {2}".format('GOOG',100,490.10)
r = "{name} {shares} {price}".format(name='GOOG',shares=100,price=490.10)
r = "Hello {0}, your age is {age}".format("Elwood",age=47)
r = "Use {{ and }} to output single curly braces".format()
```

```
name = "Elwood"
r = "{0:<10}".format(name)      # r = 'Elwood    '
r = "{0:>10}".format(name)      # r = '    Elwood'
r = "{0:^10}".format(name)      # r = '  Elwood  '
r = "{0:=^10}".format(name)     # r = '==Elwood=='
```

```
x = 42
r = '{0:10d}'.format(x)         # r = '        42'
r = '{0:10x}'.format(x)         # r = '        2a'
r = '{0:10b}'.format(x)         # r = '    101010'
r = '{0:010b}'.format(x)        # r = '0000101010'

y = 3.1415926
r = '{0:10.2f}'.format(y)       # r = '      3.14'
r = '{0:10.2e}'.format(y)       # r = '  3.14e+00'
r = '{0:+10.2f}'.format(y)      # r = '     +3.14'
r = '{0:+010.2f}'.format(y)     # r = '+000003.14'
r = '{0:+10.2%}'.format(y)      # r = '  +314.16%'
```

Type Casting

❖ Type Casting

```
In [315]: s = '3.14159'

In [316]: fval = float(s)
```

```
In [317]: type(fval)
Out[317]: float
```

```
In [318]: int(fval)
Out[318]: 3
```

```
In [319]: bool(fval)
Out[319]: True
```

```
In [320]: bool(0)
Out[320]: False
```

❖ None

- Python null value type

```
In [321]: a = None
```

```
In [322]: a is None
Out[322]: True
```

```
In [323]: b = 5
```

```
In [324]: b is not None
Out[324]: True
```

Basic Data Structures

❖ Tuples

```
In [37]: t = (1, 2.5, ‘data’)
         type(t)
Out[37]: tuple

In [39]: t[2]
Out[39]: ‘data’
In [40]: type(t[2])
Out[40]: str

In [41]: t.count(‘data’)
Out[41]: 1
In [42]: t.index(1)
Out[42]: 0
```

Tuples (and lists) can be “unpacked” as follows

```
In [21]: integers = (10, 20, 30)

In [22]: x, y, z = integers
```

Basic Data Structures

❖ Lists

```
In [43]: l = [1, 2.5, 'data']
         l[2]
Out[43]: 'data'

In [44]: l = list(t)
         l
Out[44]: [1, 2.5, 'data']
In [45]: type(l)
Out[45]: list
```

Insert & Remove

```
In [46]: l.append([4, 3])  # append list at the end
         l
Out[46]: [1, 2.5, ‘data’, [4, 3]]
In [47]: l.extend([1.0, 1.5, 2.0])  # append elements of list
         l
Out[47]: [1, 2.5, ‘data’, [4, 3], 1.0, 1.5, 2.0]
In [48]: l.insert(1, ‘insert’)  # insert object before index position
         l
Out[48]: [1, ‘insert’, 2.5, ‘data’, [4, 3], 1.0, 1.5, 2.0]
In [49]: l.remove(‘data’)  # remove first occurrence of object
         l
Out[49]: [1, ‘insert’, 2.5, [4, 3], 1.0, 1.5, 2.0]
In [50]: p = l.pop(3)  # removes and returns object at index
         print l, p
Out[50]: [1, ‘insert’, 2.5, 1.0, 1.5, 2.0] [4, 3]
```

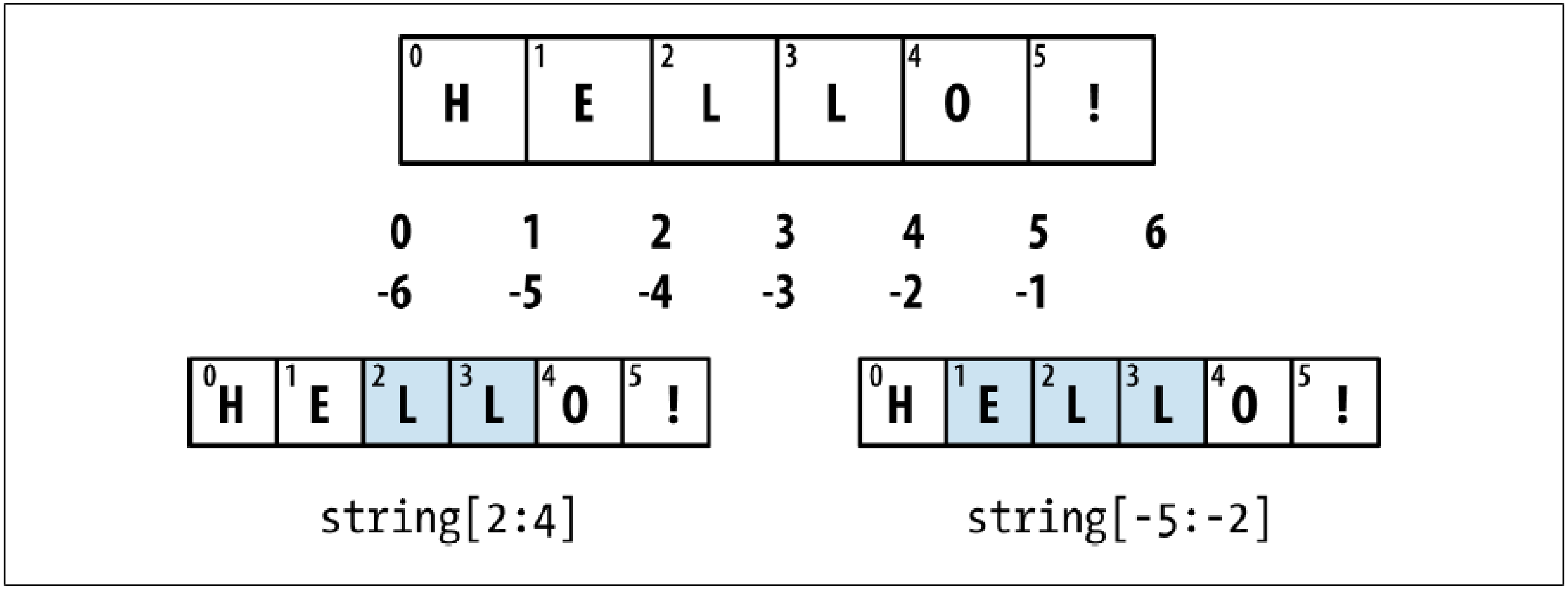
Indexing & Slicing

```
In [14]: a = [2, 4, 6, 8]

In [15]: a[1:]
Out[15]: [4, 6, 8]

In [16]: a[1:3]
Out[16]: [4, 6]
```

```
In [17]: a[-2:]   # Last two elements of the list
Out[17]: [6, 8]
```

```
a = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

b = a[::2]          # b = [0, 2, 4, 6, 8 ]
c = a[::-2]         # c = [9, 7, 5, 3, 1 ]
d = a[0:4:2]        # d = [0,2]
e = a[5:0:-2]       # e = [5,3,1]
f = a[:5:1]         # f = [0,1,2,3,4]
g = a[:5:-1]        # g = [9,8,7,6]
h = a[5::1]         # h = [5,6,7,8,9]
i = a[5::-1]        # i = [5,4,3,2,1,0]
j = a[5:0:-1]       # j = [5,4,3,2,1]
```

Slicing



Basic Data Structures

❖ Dicts

▪ Key-Value

```
In [66]: d = {
              'Name' : 'Angela Merkel',
              'Country' : 'Germany',
              'Profession' : 'Chancelor',
              'Age' : 60
              }
         type(d)
Out[66]: dict
In [67]: print d['Name'], d['Age']
Out[67]: Angela Merkel 60
```

```
In [68]: d.keys()
Out[68]: ['Country', 'Age', 'Profession', 'Name']
In [69]: d.values()
Out[69]: ['Germany', 60, 'Chancelor', 'Angela Merkel']
In [70]: d.items()
Out[70]: [('Country', 'Germany'),
          ('Age', 60),
          ('Profession', 'Chancelor'),
          ('Name', 'Angela Merkel')]
In [71]: birthday = True
         if birthday is True:
             d['Age'] += 1
         print d['Age']
Out[71]: 61
```

```
In [439]: d1
Out[439]: {'a': 'some value', 'b': [1, 2, 3, 4]}

In [440]: d1[7] = 'an integer'

In [441]: d1
Out[441]: {7: 'an integer', 'a': 'some value', 'b': [1, 2, 3, 4]}

In [442]: d1['b']
Out[442]: [1, 2, 3, 4]
```

```
In [443]: 'b' in d1
Out[443]: True
```

```
In [444]: d1[5] = 'some value'

In [445]: d1['dummy'] = 'another value'

In [446]: del d1[5]

In [447]: ret = d1.pop('dummy')
```

```
In [448]: ret
Out[448]: 'another value'
```

```
In [451]: d1.update({'b' : 'foo', 'c' : 12})

In [452]: d1
Out[452]: {7: 'an integer', 'a': 'some value', 'b': 'foo', 'c': 12}
```

Imports

From the start, Python has been designed around the twin principles of

- a small core language
- extra functionality in separate *libraries* or *modules*

```
In [1]: import math

In [2]: math.sqrt(4)
Out[2]: 2.0
```

Or

```
In [3]: from math import *

In [4]: sqrt(4)
Out[4]: 2.0
```

Dates and Times

❖ Built-in Python *datetime* module

```
In [325]: from datetime import datetime, date, time

In [326]: dt = datetime(2011, 10, 29, 20, 30, 21)

In [327]: dt.day        In [328]: dt.minute
Out[327]: 29            Out[328]: 30

In [329]: dt.date()                      In [330]: dt.time()
Out[329]: datetime.date(2011, 10, 29)    Out[330]: datetime.time(20, 30, 21)

In [331]: dt.strftime('%m/%d/%Y %H:%M')
Out[331]: '10/29/2011 20:30'

In [332]: datetime.strptime('20091031', '%Y%m%d')
Out[332]: datetime.datetime(2009, 10, 31, 0, 0)

In [334]: dt2 = datetime(2011, 11, 15, 22, 30)

In [335]: delta = dt2 - dt

In [336]: delta                          In [337]: type(delta)
Out[336]: datetime.timedelta(17, 7179)   Out[337]: datetime.timedelta
```

Operators

Operation	Description
a + b	Add a and b
a - b	Subtract b from a
a * b	Multiply a by b
a / b	Divide a by b
a // b	Floor-divide a by b, dropping any fractional remainder
a ** b	Raise a to the b power
a & b	True if both a and b are True. For integers, take the bitwise AND.
a | b	True if either a or b is True. For integers, take the bitwise OR.
a ^ b	For booleans, True if a or b is True, but not both. For integers, take the bitwise EXCLUSIVE-OR.
a == b	True if a equals b
a != b	True if a is not equal to b
a <= b, a < b	True if a is less than (less than or equal) to b
a > b, a >= b	True if a is greater than (greater than or equal) to b
a is b	True if a and b reference same Python object
a is not b	True if a and b reference different Python objects

Operations on Sequence

Operation	Description
`s + r`	Concatenation
`s * n, n * s`	Makes *n* copies of *s*, where *n* is an integer
`v1, v2..., vn = s`	Variable unpacking
`s[i]`	Indexing
`s[i:j]`	Slicing
`s[i:j:stride]`	Extended slicing
`x in s, x not in s`	Membership
`for x in s:`	Iteration
`all(s)`	Returns `True` if all items in *s* are true.
`any(s)`	Returns `True` if any item in *s* is true.
`len(s)`	Length
`min(s)`	Minimum item in *s*
`max(s)`	Maximum item in *s*
`sum(s [, initial])`	Sum of items with an optional initial value

Copies of a sequence

```
>>> a = [3,4,5]
>>> b = [a]
>>> c = 4*b
>>> c
[[3, 4, 5], [3, 4, 5], [3, 4, 5], [3, 4, 5]]
>>> a[0] = -7
>>> c
[[-7, 4, 5], [-7, 4, 5], [-7, 4, 5], [-7, 4, 5]]
```

조건문과 반복문 & 함수

Comparison

```
In [49]: x = 1     # Assignment

In [50]: x == 2    # Comparison
Out[50]: False
```

```
In [52]: x = 'yes' if 42 else 'no'

In [53]: x
Out[53]: 'yes'

In [54]: x = 'yes' if [] else 'no'

In [55]: x
Out[55]: 'no'
```

```
In [56]: 1 < 2 and 'f' in 'foo'
Out[56]: True

In [57]: 1 < 2 and 'g' in 'foo'
Out[57]: False

In [58]: 1 < 2 or 'g' in 'foo'
Out[58]: True

In [59]: not True
Out[59]: False

In [60]: not not True
Out[60]: True
```

Looping without Indices

```
for x in x_values:
    print(x * x)
```

is preferred to

```
for i in range(len(x_values)):
    print(x_values[i] * x_values[i])
```

❖ range

```
In [352]: range(10)
Out[352]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

In [353]: range(0, 20, 2)
Out[353]: [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

Other loops

❖ While

```
In [57]: total = 0
         while total < 100:
             total += 1
         print total

Out[57]: 100
```

```
x = 256
total = 0
while x > 0:
    if total > 500:
        break
    total += x
    x = x // 2
```

❖ List comprehensions

```
In [58]: m = [i ** 2 for i in range(5)]
         m

Out[58]: [0, 1, 4, 9, 16]
```

continue & break

❖ continue

```
sequence = [1, 2, None, 4, None, 5]
total = 0
for value in sequence:
    if value is None:
        continue
    total += value
```

❖ break

```
sequence = [1, 2, 0, 4, 6, 5, 2, 1]
total_until_5 = 0
for value in sequence:
    if value == 5:
        break
    total_until_5 += value
```

Conditional execution

❖ if / elif / else

```
In [56]: for i in range(1, 10):
             if i % 2 == 0:  # % is for modulo
                 print “%d is even” % i
             elif i % 3 == 0:
                 print “%d is multiple of 3” % i
             else:
                 print “%d is odd” % i

Out[56]: 1 is odd
         2 is even
         3 is multiple of 3
         4 is even
         5 is odd
         6 is even
         7 is odd
         8 is even
         9 is multiple of 3
```

Ternary Expressions

```
value = true-expr if condition else false-expr
```

⟺

```
if condition:
    value = true-expr
else:
    value = false-expr
```

```
In [354]: x = 5

In [355]: 'Non-negative' if x >= 0 else 'Negative'
Out[355]: 'Non-negative'
```

Function definition

❖ Function definition

```
In [59]: def f(x):
             return x ** 2
         f(2)
Out[59]: 4
```

❖ map

```
In [60]: def even(x):
             return x % 2 == 0
         even(3)

Out[60]: False
```

```
In [61]: map(even, range(10))
Out[61]: [True, False, True, False, True, False, True, False, True, False]
```

pass

❖ if statement

```
if x < 0:
    print 'negative!'
elif x == 0:
    # TODO: put something smart here
    pass
else:
    print 'positive!'
```

❖ function

```
def f(x, y, z):
    # TODO: implement this function!
    pass
```

Default argument value

```
def my_function(x, y, z=1.5):
    if z > 1:
        return z * (x + y)
    else:
        return z / (x + y)
```

```
def f(x, coefficients=(1, 1)):
    a, b = coefficients
    return a + b * x
```

```
In [71]: f(2, coefficients=(0, 0))
Out[71]: 0

In [72]: f(2)  # Use default values (1, 1)
Out[72]: 3
```

Returning multiple values

❖ Returning tuple

```
def f():
    a = 5
    b = 6
    c = 7
    return a, b, c

a, b, c = f()
```

❖ Returning dict

```
def f():
    a = 5
    b = 6
    c = 7
    return {'a' : a, 'b' : b, 'c' : c}
```

NumPy

Intro to NumPy

❖ What is NumPy?

- Fast array processing
- Iteration via loops in interpreted languages (including Python) is relatively slow
- Operations are sent in batches to optimized C and Fortran code

❖ NumPy Arrays

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

In [2]: a = np.zeros(3)

In [3]: a
Out[3]: array([ 0.,  0.,  0.])

In [4]: type(a)
Out[4]: numpy.ndarray
```

Shape and Dimension

❖ "flat" array

```
In [11]: z = np.zeros(10)

In [12]: z.shape
Out[12]: (10,)  # Note syntax for tuple with one element
```

❖ Changing shape

```
In [13]: z.shape = (10, 1)

In [14]: z
Out[14]:
array([[ 0.],
       [ 0.],
       [ 0.],
       [ 0.],
       [ 0.],
       [ 0.],
       [ 0.],
       [ 0.],
       [ 0.],
       [ 0.]])
```

```
In [15]: z = np.zeros(4)

In [16]: z.shape = (2, 2)

In [17]: z
Out[17]:
array([[ 0.,  0.],
       [ 0.,  0.]])
```

Creating Arrays

❖ Empty array

```
In [18]: z = np.empty(3)

In [19]: z
Out[19]: array([  8.90030222e-307,   4.94944794e+173,   4.04144187e-262])
```

❖ Grid of evenly spaced numbers

```
In [20]: z = np.linspace(2, 4, 5)  # From 2 to 4, with 5 elements
```

❖ Identity

```
In [21]: z = np.identity(2)

In [22]: z
Out[22]:
array([[ 1.,  0.],
       [ 0.,  1.]])
```

Indexing

```
In [30]: z = np.linspace(1, 2, 5)

In [31]: z
Out[31]: array([ 1.  ,  1.25,  1.5 ,  1.75,  2.  ])

In [32]: z[0]
Out[32]: 1.0

In [33]: z[0:2]  # Slice numbering is left closed, right open
Out[33]: array([ 1.  ,  1.25])

In [34]: z[-1]
Out[34]: 2.0
```

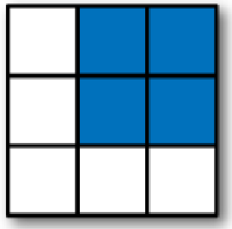
2D

```
In [35]: z = np.array([[1, 2], [3, 4]])

In [36]: z
Out[36]:
array([[1, 2],
       [3, 4]])

In [37]: z[0, 0]
Out[37]: 1

In [38]: z[0, 1]
Out[38]: 2

In [39]: z[0,:]
Out[39]: array([1, 2])

In [40]: z[:,1]
Out[40]: array([2, 4])
```

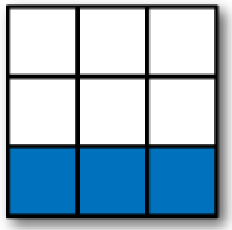
```
In [42]: z
Out[42]: array([ 2. ,  2.5,  3. ,  3.5,  4. ])

In [43]: indices = np.array((0, 2, 3))

In [44]: z[indices]
Out[44]: array([ 2. ,  3. ,  3.5])

In [46]: d = np.array([0, 1, 1, 0, 0], dtype=bool)

In [47]: d
Out[47]: array([False,  True,  True, False, False], dtype=bool)

In [48]: z[d]
Out[48]: array([ 2.5,  3. ])
```

```
In [49]: z = np.empty(3)

In [50]: z
Out[50]: array([ -1.25236750e-041,   0.00000000e+000,   5.45693855e-313])

In [51]: z[:] = 42

In [52]: z
Out[52]: array([ 42.,  42.,  42.])
```

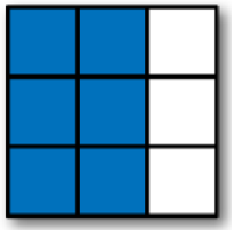
Two-dim slicing



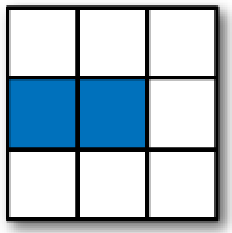
Expression	Shape
arr[:2, 1:]	(2, 2)



Expression	Shape
arr[2]	(3,)
arr[2, :]	(3,)
arr[2:, :]	(1, 3)



Expression	Shape
arr[:, :2]	(3, 2)



Expression	Shape
arr[1, :2]	(2,)
arr[1:2, :2]	(1, 2)

Stacking helpers: r_ & c_

```
In [45]: arr = np.arange(6)

In [46]: arr1 = arr.reshape((3, 2))

In [47]: arr2 = randn(3, 2)

In [48]: np.r_[arr1, arr2]
Out[48]:
array([[ 0.    ,  1.    ],
       [ 2.    ,  3.    ],
       [ 4.    ,  5.    ],
       [ 0.7258, -1.5325],
       [-0.4696, -0.2127],
       [-0.1072,  1.2871]])
```

```
In [49]: np.c_[np.r_[arr1, arr2], arr]
Out[49]:
array([[ 0.    ,  1.    ,  0.    ],
       [ 2.    ,  3.    ,  1.    ],
       [ 4.    ,  5.    ,  2.    ],
       [ 0.7258, -1.5325,  3.    ],
       [-0.4696, -0.2127,  4.    ],
       [-0.1072,  1.2871,  5.    ]])
```

```
In [50]: np.c_[1:6, -10:-5]
Out[50]:
array([[  1, -10],
       [  2,  -9],
       [  3,  -8],
       [  4,  -7],
       [  5,  -6]])
```

Methods

Attribute		
	dtype	ndim
	shape	size
	T	
Method		
	all	any
	argmax	argmin
	max	min
	mean	
	std	var
	prod	sum
	cumprod	cumsum
	dot	transpose
	diagonal	trace
	sort	copy

```
In [53]: A = np.array((4, 3, 2, 1))

In [54]: A
Out[54]: array([4, 3, 2, 1])

In [55]: A.sort()                  # Sorts A in place

In [56]: A
Out[56]: array([1, 2, 3, 4])

In [57]: A.sum()                   # Sum
Out[57]: 10

In [58]: A.mean()                  # Mean
Out[58]: 2.5

In [64]: A.std()
Out[64]: 1.1180339887498949

In [65]: A.shape = (2, 2)

In [66]: A.T
Out[66]:
array([[1, 3],
       [2, 4]])
```

Algebraic Operation

```
In [75]: a = np.array([1, 2, 3, 4])

In [76]: b = np.array([5, 6, 7, 8])

In [77]: a + b
Out[77]: array([ 6,  8, 10, 12])

In [78]: a * b
Out[78]: array([ 5, 12, 21, 32])

In [79]: a + 10
Out[79]: array([11, 12, 13, 14])

In [82]: a * 10
Out[82]: array([10, 20, 30, 40])
```

```
In [86]: A = np.ones((2, 2))

In [87]: B = np.ones((2, 2))

In [88]: A + B
Out[88]:
array([[ 2.,  2.],
       [ 2.,  2.]])

In [89]: A + 10
Out[89]:
array([[ 11.,  11.],
       [ 11.,  11.]])

In [90]: A * B
Out[90]:
array([[ 1.,  1.],
       [ 1.,  1.]])
```

In particular, `A * B` is *not* the matrix product, it is an elementwise product

Matrix Multiplication

```
In [137]: A = np.ones((2, 2))

In [138]: B = np.ones((2, 2))

In [139]: np.dot(A, B)
Out[139]:
array([[ 2.,  2.],
       [ 2.,  2.]])
```

```
In [91]: A = np.array([1, 2])

In [92]: B = np.array([10, 20])

In [93]: np.dot(A, B)    # Returns a scalar in this case
Out[93]: 50
```

Comparisons

❖ Elementwise comparisons

```
In [97]: z = np.array([2, 3])

In [98]: y = np.array([2, 3])

In [99]: z == y
Out[99]: array([ True,  True], dtype=bool)

In [100]: y[0] = 5

In [101]: z == y
Out[101]: array([False,  True], dtype=bool)

In [102]: z != y
Out[102]: array([ True, False], dtype=bool)

In [103]: z = np.linspace(0, 10, 5)

In [104]: z
Out[104]: array([  0. ,   2.5,   5. ,   7.5,  10. ])

In [105]: z > 3
Out[105]: array([False, False,  True,  True,  True], dtype=bool)
```

```
In [109]: z[z > 3]
Out[109]: array([  5. ,   7.5,  10. ])
```

Vectorized Functions

```python
for i in range(n):
    y[i] = np.sin(z[i])
```

```python
In [110]: z = np.array([1, 2, 3])

In [111]: np.sin(z)
Out[111]: array([ 0.84147098,  0.90929743,  0.14112001])

In [113]: (1 / np.sqrt(2 * np.pi)) * np.exp(- 0.5 * z**2)
Out[113]: array([ 0.24197072,  0.05399097,  0.00443185])
```

Universal Functions (ufunc)

❖ Elementwise operations on data in ndarrays

- Unary ufuncs

```
In [120]: arr = np.arange(10)

In [121]: np.sqrt(arr)
Out[121]:
array([ 0.    ,  1.    ,  1.4142,  1.7321,  2.    ,  2.2361,  2.4495,
        2.6458,  2.8284,  3.    ])

In [122]: np.exp(arr)
Out[122]:
array([    1.    ,     2.7183,     7.3891,    20.0855,    54.5982,
         148.4132,   403.4288,  1096.6332,  2980.958 ,  8103.0839])
```

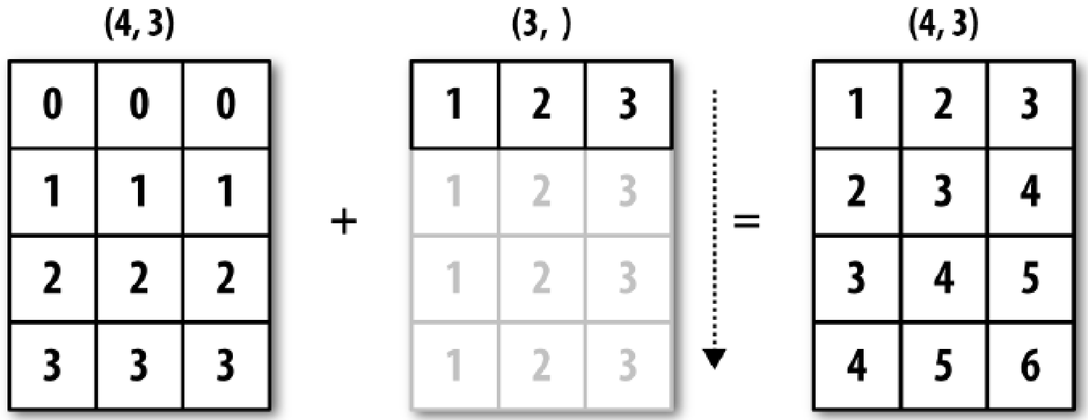
- Binary ufuncs

```
In [127]: np.maximum(x, y) # element-wise maximum
Out[127]:
array([ 0.267 ,  0.0974,  0.2002,  0.6117,  0.4655,  0.9222,  0.446 ,
       -0.7147])
```

Broadcasting

```
In [83]: arr = randn(4, 3)

In [84]: arr.mean(0)
Out[84]: array([ 0.1321,  0.552 ,  0.8571])

In [85]: demeaned = arr - arr.mean(0)

In [86]: demeaned
Out[86]:
array([[ 0.1718, -0.1972, -1.3669],
       [-0.1292,  1.6529, -0.3429],
       [-0.2891, -0.0435,  1.2322],
       [ 0.2465, -1.4122,  0.4776]])
```

```
In [87]: demeaned.mean(0)
Out[87]: array([ 0., -0., -0.])
```



Broadcasting

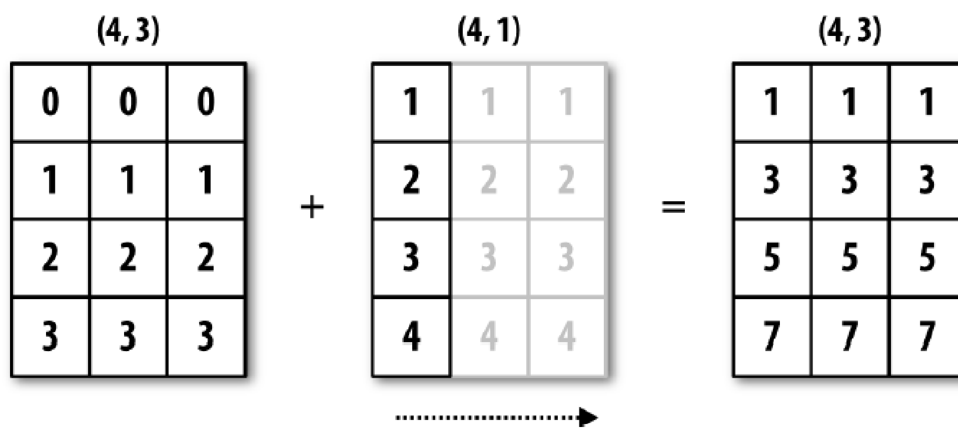
```
In [88]: arr
Out[88]:
array([[ 0.3039,  0.3548, -0.5097],
       [ 0.0029,  2.2049,  0.5142],
       [-0.1571,  0.5085,  2.0893],
       [ 0.3786, -0.8602,  1.3347]])
```

```
In [89]: row_means = arr.mean(1)
```

```
In [90]: row_means.reshape((4, 1))
Out[90]:
array([[ 0.0496],
       [ 0.9073],
       [ 0.8136],
       [ 0.2844]])
```

```
In [91]: demeaned = arr - row_means.reshape((4, 1))
```

```
In [92]: demeaned.mean(1)
Out[92]: array([ 0.,  0.,  0.,  0.])
```



Other NumPy Functions

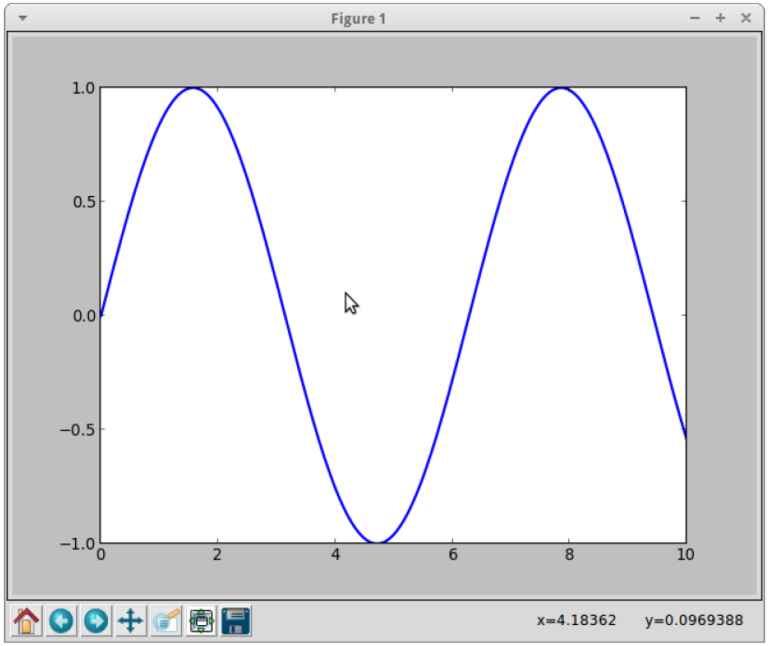
```
In [131]: A = np.array([[1, 2], [3, 4]])

In [132]: np.linalg.det(A)             # Compute the determinant
Out[132]: -2.0000000000000004

In [133]: np.linalg.inv(A)             # Compute the inverse
Out[133]:
array([[-2. ,  1. ],
       [ 1.5, -0.5]])

In [134]: Z = np.random.randn(10000)  # Generate standard normals

In [135]: y = np.random.binomial(10, 0.5, size=1000)
```

MatplotLib

Intro to Matplotlib

- Overview
  - 2D and 3D plots

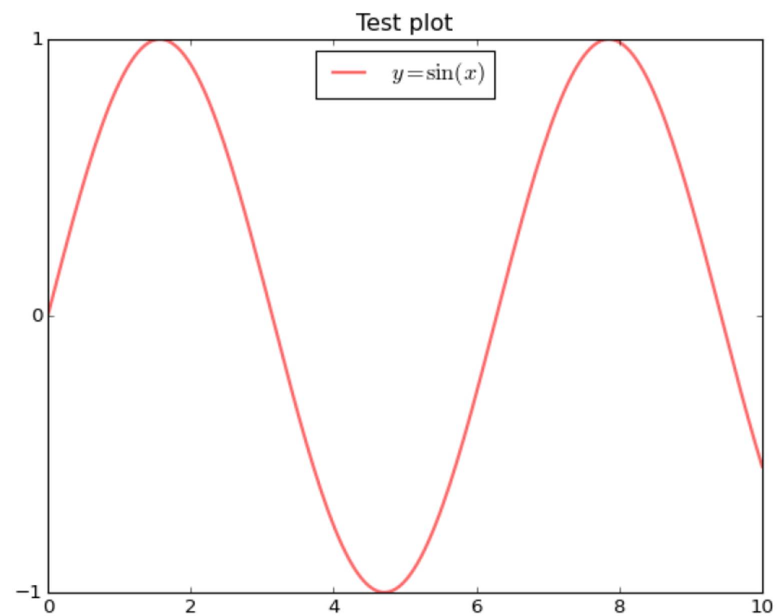
```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0, 10, 200)
y = np.sin(x)
plt.plot(x, y, 'b-', linewidth=2)
plt.show()
```



Something more

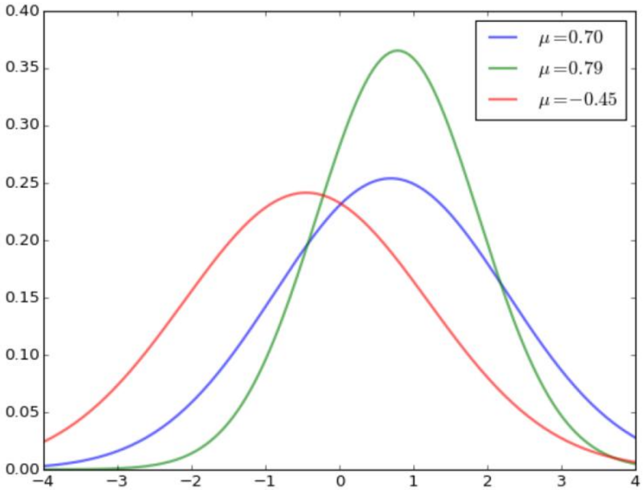
```
import matplotlib.pyplot as plt
import numpy as np
fig, ax = plt.subplots()
x = np.linspace(0, 10, 200)
y = np.sin(x)
ax.plot(x, y, 'r-', linewidth=2, label=r'$y=\sin(x)$', alpha=0.6)
ax.legend(loc='upper center')
ax.set_yticks([-1, 0, 1])
ax.set_title('Test plot')
plt.show()
```

- Title
- linewidth
- label & legend
- LaTeX

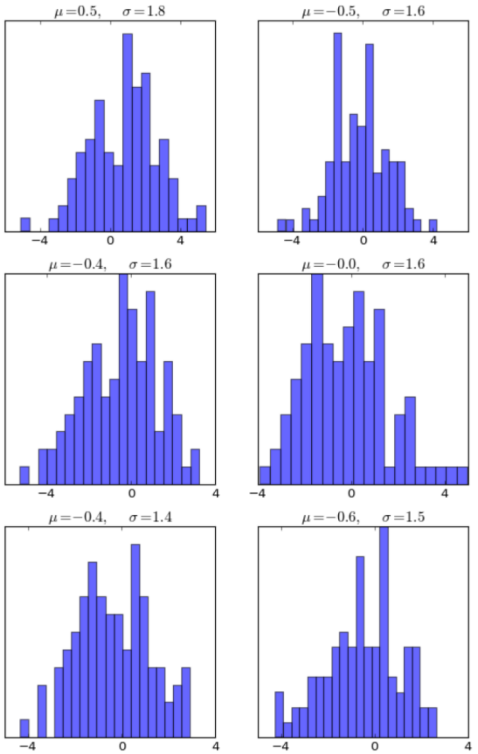


Multiple Plots on One Axis

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import norm
from random import uniform

fig, ax = plt.subplots()
x = np.linspace(-4, 4, 150)
for i in range(3):
    m, s = uniform(-1, 1), uniform(1, 2)
    y = norm.pdf(x, loc=m, scale=s)
    current_label = r'$\mu = {0:.2f}$'.format(m)
    ax.plot(x, y, linewidth=2, alpha=0.6, label=current_label)
ax.legend()
plt.show()
```



Multiple Subplot




```python
import matplotlib.pyplot as plt
from scipy.stats import norm
from random import uniform
num_rows, num_cols = 3, 2
fig, axes = plt.subplots(num_rows, num_cols, figsize=(8, 12))
for i in range(num_rows):
    for j in range(num_cols):
        m, s = uniform(-1, 1), uniform(1, 2)
        x = norm.rvs(loc=m, scale=s, size=100)
        axes[i, j].hist(x, alpha=0.6, bins=20)
        t = r'$\mu = {0:.1f}, \quad \sigma = {1:.1f}$'.format(m, s)
        axes[i, j].set_title(t)
        axes[i, j].set_xticks([-4, 0, 4])
        axes[i, j].set_yticks([])
plt.show()
```

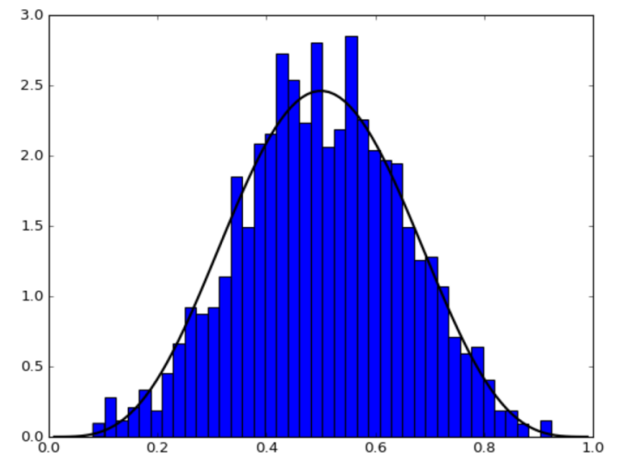
SciPy - Some Useful Functions

Statistics

❖ scipy.stats

- Random variable objects (distributions, random sampling …)
- Estimation procedures
- Statistical tests

```
import numpy as np
from scipy.stats import beta
from matplotlib.pyplot import hist, plot, show
q = beta(5, 5)         # Beta(a, b), with a = b = 5
obs = q.rvs(2000)      # 2000 observations
hist(obs, bins=40, normed=True)
grid = np.linspace(0.01, 0.99, 100)
plot(grid, q.pdf(grid), 'k-', linewidth=2)
show()
```



Statistics

❖ methods

```
In [14]: q.cdf(0.4)        # Cumulative distribution function
Out[14]: 0.26656768000000002

In [15]: q.pdf(0.4)        # Density function
Out[15]: 2.0901888000000004

In [16]: q.ppf(0.8)        # Quantile (inverse cdf) function
Out[16]: 0.63391348346427079

In [17]: q.mean()
Out[17]: 0.5
```

rvs: Random Variates
pdf: Probability Density Function
cdf: Cumulative Distribution Function
sf: Survival Function (1-CDF)
ppf: Percent Point Function (Inverse of CDF)
isf: Inverse Survival Function (Inverse of SF)
moment: non-central moments of the distribution

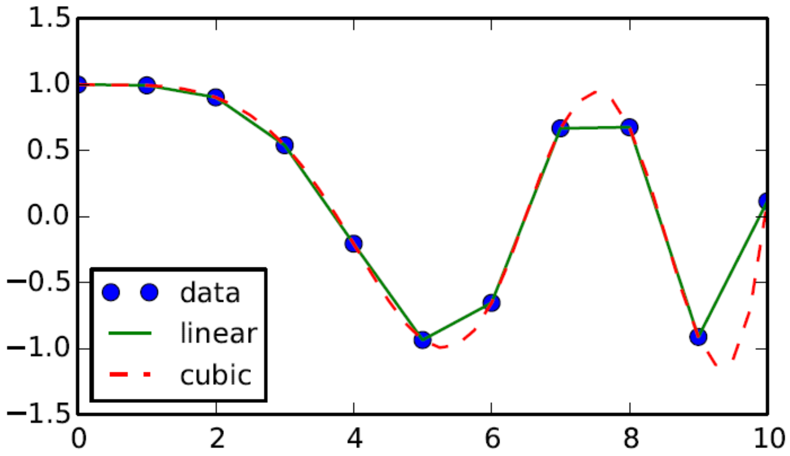
Linear Regression

❖ linregress

```
In [19]: from scipy.stats import linregress

In [20]: x = np.random.randn(200)

In [21]: y = 2 * x + 0.1 * np.random.randn(200)

In [22]: gradient, intercept, r_value, p_value, std_err = linregress(x, y)

In [23]: gradient, intercept
Out[23]: (1.9962554379482236, 0.008172822032671799)
```

1-D Interpolation

```
from scipy.interpolate import interp1d

x = np.linspace(0, 10, num=11, endpoint=True)
y = np.cos(-x**2/9.0)
f = interp1d(x, y)
f2 = interp1d(x, y, kind='cubic')

xnew = np.linspace(0, 10, num=41, endpoint=True)
import matplotlib.pyplot as plt
plt.plot(x, y, 'o', xnew, f(xnew), '-', xnew, f2(xnew), '--')
plt.legend(['data', 'linear', 'cubic'], loc='best')
plt.show()
```



Pandas

Overview

❖ What is pandas?

- 빠르고 효율적인 데이터 처리를 위한 package
- 기본적인 데이터 structures
- 시계열분석과 통계분석

❖ Strengths of pandas

- 편리하고 빠른 데이터 읽기
- 데이터 가공
- 날짜와 시계열 데이터
- 정렬, 그룹화, 머징
- 결측치에 대한 처리

Series

- "column" of data
- Collection of observations on a single variable

```
In [1]: import pandas as pd

In [2]: import numpy as np

In [4]: s = pd.Series(np.random.randn(4), name='daily returns')
```

```
In [5]: s
Out[5]:
0     0.430271
1     0.617328
2    -0.265421
3    -0.836113
Name: daily returns
```

```
In [6]: s * 100
Out[6]:
0     43.027108
1     61.732829
2    -26.542104
3    -83.611339
Name: daily returns
```

```
In [7]: np.abs(s)
Out[7]:
0     0.430271
1     0.617328
2     0.265421
3     0.836113
Name: daily returns
```

More than NumPy Arrays

```
In [9]: s.index = ['AMZN', 'AAPL', 'MSFT', 'GOOG']

In [10]: s
Out[10]:
AMZN    0.430271
AAPL    0.617328
MSFT   -0.265421
GOOG   -0.836113
Name: daily returns

In [8]: s.describe()
Out[8]:
count     4.000000
mean     -0.013484
std       0.667092
min      -0.836113
25%      -0.408094
50%       0.082425
75%       0.477035
max       0.617328
```

```
In [11]: s['AMZN']
Out[11]: 0.43027108469945924

In [12]: s['AMZN'] = 0

In [13]: s
Out[13]:
AMZN    0.000000
AAPL    0.617328
MSFT   -0.265421
GOOG   -0.836113
Name: daily returns

In [14]: 'AAPL' in s
Out[14]: True
```

DataFrame

- DataFrame
  - Object for storing related columns of data
  - Analogous to a Excel spreadsheet

```
In [28]: df = pd.read_csv('data/test_pwt.csv')

In [29]: type(df)
Out[29]: pandas.core.frame.DataFrame

In [30]: df
Out[30]:
        country country isocode  year          POP       XRA
0     Argentina             ARG  2000    37335.653   0.99950
1     Australia             AUS  2000    19053.186   1.72483
2         India             IND  2000  1006300.297  44.94160
3        Israel             ISR  2000     6114.570   4.07733
4        Malawi             MWI  2000    11801.505  59.54380
5  South Africa             ZAF  2000    45064.098   6.93983
6 United States             USA  2000   282171.957   1.00000
7       Uruguay             URY  2000     3219.793  12.09959
```

DataFrame Slicing

```
In [13]: df[2:5]
Out[13]:
  country country isocode  year          POP       XRAT          tc
2   India              IND  2000  1006300.297  44.941600  1728144.374
3  Israel              ISR  2000     6114.570   4.077330   129253.894
4  Malawi              MWI  2000    11801.505  59.543808     5026.221
```

```
In [14]: df[['country', 'tcgdp']]
Out[14]:
        country          tcgdp
0      Argentina   295072.218690
1      Australia   541804.652100
2          India  1728144.374800
3         Israel   129253.894230
4         Malawi     5026.221784
5   South Africa   227242.369490
6  United States  9898700.000000
7        Uruguay    25255.961693
```

```
In [21]: df.ix[2:5, ['country', 'tcgdp']]
Out[21]:
        country          tcgdp
2          India  1728144.374800
3         Israel   129253.894230
4         Malawi     5026.221784
5   South Africa   227242.369490
```

DataFrame

❖ DF from dict

```
data = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada'],
        'year': [2000, 2001, 2002, 2001, 2002],
        'pop': [1.5, 1.7, 3.6, 2.4, 2.9]}
frame = DataFrame(data)
```

- Specifing a sequence & index

```
In [40]: frame2 = DataFrame(data, columns=['year', 'state', 'pop', 'debt'],
   ....:                    index=['one', 'two', 'three', 'four', 'five'])

In [41]: frame2
Out[41]:
       year   state  pop  debt
one    2000    Ohio  1.5   NaN
two    2001    Ohio  1.7   NaN
three  2002    Ohio  3.6   NaN
four   2001  Nevada  2.4   NaN
five   2002  Nevada  2.9   NaN
```

Descriptive & Summary

Method	Description
count	Number of non-NA values
describe	Compute set of summary statistics for Series or each DataFrame column
min, max	Compute minimum and maximum values
argmin, argmax	Compute index locations (integers) at which minimum or maximum value obtained, respectively
idxmin, idxmax	Compute index values at which minimum or maximum value obtained, respectively
quantile	Compute sample quantile ranging from 0 to 1
sum	Sum of values
mean	Mean of values
median	Arithmetic median (50% quantile) of values
mad	Mean absolute deviation from mean value
var	Sample variance of values
std	Sample standard deviation of values
skew	Sample skewness (3rd moment) of values
kurt	Sample kurtosis (4th moment) of values
cumsum	Cumulative sum of values
cummin, cummax	Cumulative minimum or maximum of values, respectively
cumprod	Cumulative product of values
diff	Compute 1st arithmetic difference (useful for time series)
pct_change	Compute percent changes

Correlation & Covariance

```
In [209]: returns = price.pct_change()

In [210]: returns.tail()

Out[210]:
                AAPL      GOOG       IBM      MSFT
Date
2009-12-24  0.034339  0.011117  0.004420  0.002747
2009-12-28  0.012294  0.007098  0.013282  0.005479
2009-12-29 -0.011861 -0.005571 -0.003474  0.006812
2009-12-30  0.012147  0.005376  0.005468 -0.013532
2009-12-31 -0.004300 -0.004416 -0.012609 -0.015432
```

```
In [213]: returns.corr()
Out[213]:
          AAPL      GOOG       IBM      MSFT
AAPL  1.000000  0.470660  0.410648  0.424550
GOOG  0.470660  1.000000  0.390692  0.443334
IBM   0.410648  0.390692  1.000000  0.496093
MSFT  0.424550  0.443334  0.496093  1.000000

In [214]: returns.cov()
Out[214]:
          AAPL      GOOG       IBM      MSFT
AAPL  0.001028  0.000303  0.000252  0.000309
GOOG  0.000303  0.000580  0.000142  0.000205
IBM   0.000252  0.000142  0.000367  0.000216
MSFT  0.000309  0.000205  0.000216  0.000516

In [215]: returns.corrwith(returns.IBM)
Out[215]:
AAPL    0.410648
GOOG    0.390692
IBM     1.000000
MSFT    0.496093
```

Panel Data

```
In [124]: import pandas_datareader.data as pdd

In [125]: data = pdd.DataReader(["GOOG","MSFT","FB"],"yahoo")

In [126]: data
Out[126]:
<class 'pandas.core.panel.Panel'>
Dimensions: 6 (items) x 1553 (major_axis) x 3 (minor_axis)
Items axis: Open to Adj Close
Major_axis axis: 2010-01-04 00:00:00 to 2016-03-04 00:00:00
Minor_axis axis: FB to MSFT
```

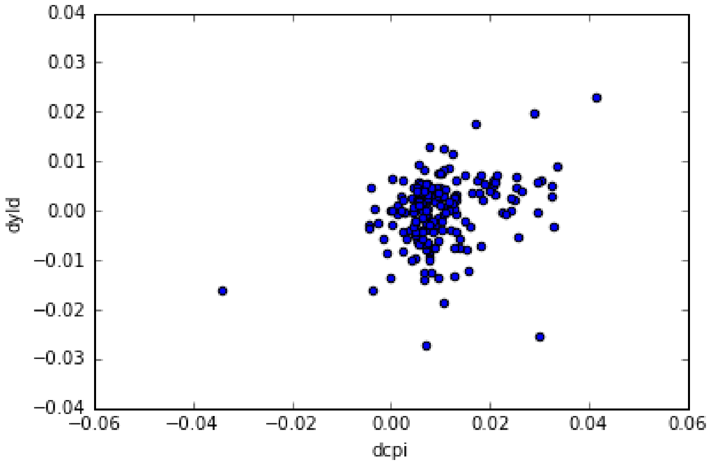
```
In [134]: d = data["Close"]

In [135]: d.tail()
Out[135]:
                    FB        GOOG       MSFT
Date
2016-02-29  106.919998  697.770020  50.880001
2016-03-01  109.820000  718.809998  52.580002
2016-03-02  109.949997  718.849976  52.950001
2016-03-03  109.580002  712.419983  52.349998
2016-03-04  108.389999  710.890015  52.029999
```

```
In [136]: d = data["Close",:3]

In [137]: d
Out[137]:
            FB        GOOG       MSFT
Date
2010-01-04 NaN  626.751061  30.950001
2010-01-05 NaN  623.991055  30.959999
2010-01-06 NaN  608.261023  30.770000

In [138]: d = data["Close",:3,"MSFT"]

In [139]: d
Out[139]:
Date
2010-01-04    30.950001
2010-01-05    30.959999
2010-01-06    30.770000
Name: MSFT, dtype: float64
```

OLS

```
In [43]: import datetime as dt
    ...: import pandas as pd
    ...:
    ...: s = dt.datetime(1970,1,1)
    ...: e = dt.datetime(2016,1,1)
    ...: data = pdd.DataReader(["DGS10","CPIAUCSL"],"fred",start=s,end=e)
    ...: d = data.resample("q", how="last", fill_method='ffill')
    ...: d['dyld'] = d['DGS10'].diff() / 100.0
    ...: d['dcpi'] = d['CPIAUCSL'].pct_change()
    ...: d.plot(kind='scatter',x='dcpi',y='dyld')
    ...: res = pd.ols(x=d['dcpi'],y=d['dyld'])
    ...: print(res)
```

```
-------------------------Summary of Regression Analysis-------------------------

Formula: Y ~ <x> + <intercept>

Number of Observations:         184
Number of Degrees of Freedom:   2

R-squared:         0.1027
Adj R-squared:     0.0977

Rmse:              0.0065

F-stat (1, 182):    20.8221, p-value:     0.0000

Degrees of Freedom: model 1, resid 182
```




```
-----------------------Summary of Estimated Coefficients------------------------
      Variable       Coef    Std Err     t-stat    p-value    CI 2.5%   CI 97.5%
--------------------------------------------------------------------------------
             x     0.2514     0.0551       4.56     0.0000     0.1434     0.3593
     intercept    -0.0028     0.0007      -3.79     0.0002    -0.0042    -0.0013
---------------------------------End of Summary---------------------------------
```

OLS results

```
In [49]: res.beta
Out[49]:
x            0.251371
intercept   -0.002780
dtype: float64

In [50]: type(res.beta)
Out[50]: pandas.core.series.Series

In [51]: res.beta['x']
Out[51]: 0.2513713742169752

In [52]: res.r2_adj
Out[52]: 0.09773163153719055

In [53]: res.x
Out[53]:
                   x  intercept
DATE
1970-06-30  0.013055          1
1970-09-30  0.010309          1
1970-12-31  0.015306          1
1971-03-31  0.005025          1
1971-06-30  0.012500          1
```

```
In [65]: res.summary_as_matrix
Out[65]:
                x  intercept
beta     0.251371  -0.002780
p-value  0.000009   0.000204
std err  0.055088   0.000733
t-stat   4.563128  -3.791296

In [66]: type(res.summary_as_matrix)
Out[66]: pandas.core.frame.DataFrame
```