

# Python tutorial: Installation & basic syntax

Research Group Meeting

07/25/2017

Mustafa Dogan



# Tutorial outline

- Software installation
- Basic syntax & arithmetic operations (+,-,\*,/)
- Strings
- List, set, dictionary, array, and data frame
- If statements
- Loops: for, while, enumerate
- Load and save data (`csv`, `text`)
- Plotting & data visualization

# Python programming language

- Widely used high-level programming language for general-purpose programming
- a design philosophy which emphasizes code readability
- a syntax which allows programmers to express concepts in fewer lines of code
- constructs intended to enable writing clear programs

# The core philosophy of the language import this

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Flat is better than nested.
- Sparse is better than dense.
- Readability counts.
- Special cases aren't special enough to break the rules.
- Although practicality beats purity.
- Errors should never pass silently.
- Unless explicitly silenced.
- In the face of ambiguity, refuse the temptation to guess.
- There should be one—and preferably only one—obvious way to do it.
- Although that way may not be obvious at first unless you're Dutch.
- Now is better than never.
- Although never is often better than right now.
- If the implementation is hard to explain, it's a bad idea.
- If the implementation is easy to explain, it may be a good idea.
- Namespaces are one honking great idea—let's do more of those!

# Downloading Software

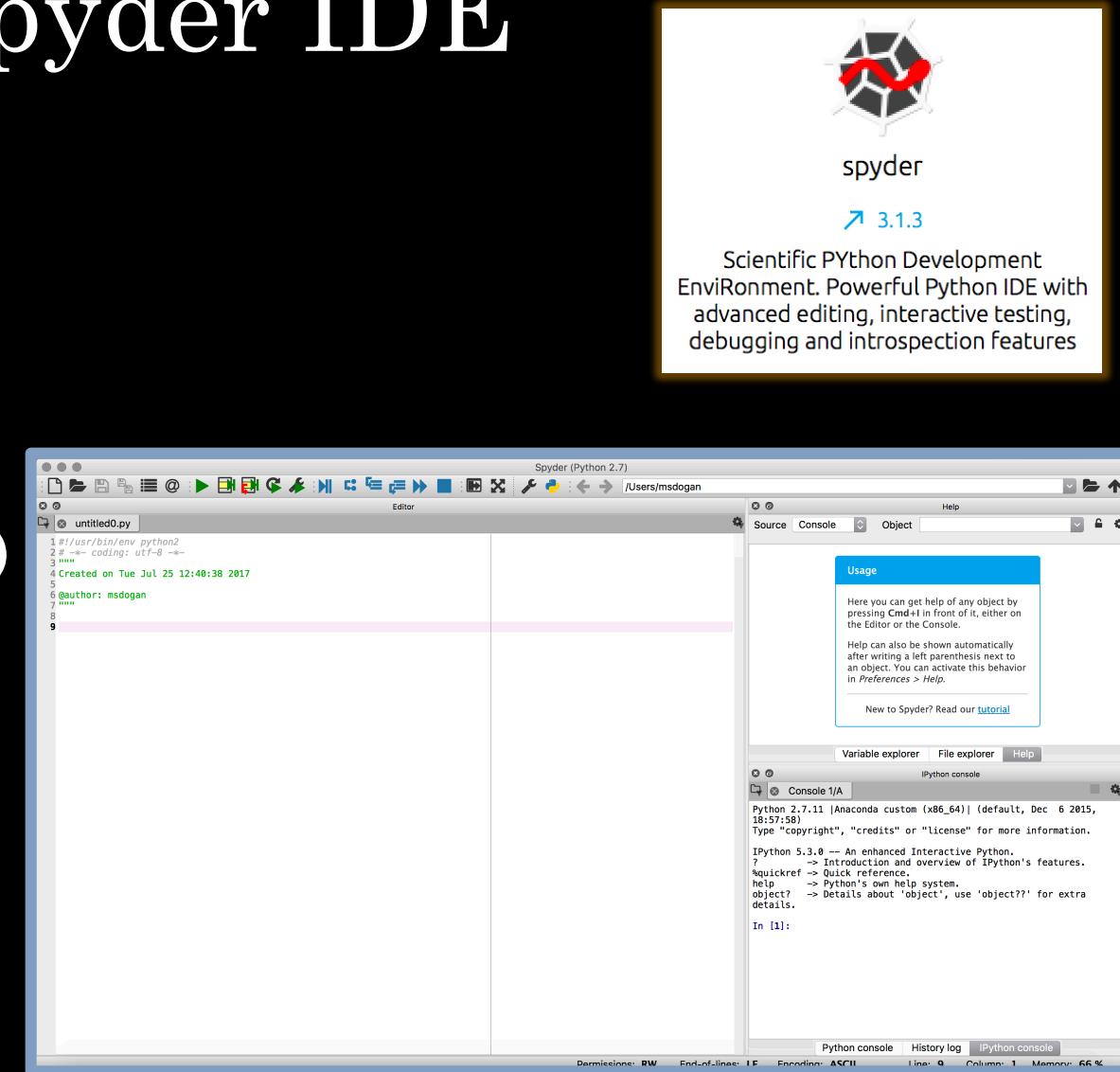
- Anaconda distribution includes almost all packages you will need to run Python
- Link to software (or just google Anaconda)

<https://www.continuum.io/downloads>

(Python 3+ is recommended)

# Using Python with Spyder IDE

- After installing Anaconda distribution open spyder
- Spyder is an open source cross-platform integrated development environment (IDE) for scientific programming in the Python language
- Similar to Matlab, there is script editor and console



# Official tutorial website

<https://docs.python.org/3/tutorial/index.html>

The screenshot shows the Python 3.6.2 Documentation homepage. The top navigation bar includes links for "Python", "3.6.2", "Documentation", "Quick search", "Go", and "previous | next | modules | index". On the left, a sidebar provides links to "Previous topic", "Changelog", "Next topic", "1. Whetting Your Appetite", "This Page", "Report a Bug", and "Show Source". The main content area features a large title "The Python Tutorial". Below it, a paragraph explains Python's nature as an easy-to-learn, powerful programming language with efficient data structures and a simple approach to object-oriented programming. It highlights Python's syntax, dynamic typing, and interpreted nature as ideal for scripting and rapid application development. Another paragraph discusses Python's extensibility through C or C++ extensions and its use as an extension language. A sidebar on the right contains a link to the Python Standard Library and a note about the tutorial being self-contained and suitable for off-line reading. The footer of the page includes a copyright notice for Python Software Foundation and a link to the Python License.

Python » 3.6.2 Documentation »

Quick search Go | previous | next | modules | index

Previous topic  
Changelog

Next topic  
1. Whetting Your Appetite

This Page  
Report a Bug  
Show Source

## The Python Tutorial

Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.

The Python interpreter and the extensive standard library are freely available in source or binary form for all major platforms from the Python Web site, <https://www.python.org/>, and may be freely distributed. The same site also contains distributions of and pointers to many free third party Python modules, programs and tools, and additional documentation.

The Python interpreter is easily extended with new functions and data types implemented in C or C++ (or other languages callable from C). Python is also suitable as an extension language for customizable applications.

« This tutorial introduces the reader informally to the basic concepts and features of the Python language and system. It helps to have a Python interpreter handy for hands-on experience, but all examples are self-contained, so the tutorial can be read off-line as well.

For a description of standard objects and modules, see [The Python Standard Library](#). [The Python Language Reference](#) gives a more formal definition of the language. To write extensions in C or C++, read [Extending and Embedding the Python Interpreter](#) and [Python/C API Reference Manual](#). There are also several books covering Python in depth.

This tutorial does not attempt to be comprehensive and cover every single feature, or even every commonly used feature. Instead, it introduces many of Python's most noteworthy features, and will give you a good idea of the language's flavor and style. After reading it, you will be able to read and write Python modules and programs, and you will be ready to learn more about the various Python library modules described in [The Python Standard Library](#).

# First Command: Hello Python!

- Type `print('Hello Python')` to script editor and run (F5)
- Type `print('Hello Python')` to console and hit ‘enter’

The screenshot shows the Spyder Python 2.7 interface. In the top-left pane, there is a code editor window titled "untitled0.py" containing the following Python code:1 #!/usr/bin/env python2
2 # -\*- coding: utf-8 -\*-
3 #
4 Created on Tue Jul 25 12:40:38 2017
5 
6 author: msdogan
7 #
8 
9 print('Hello python!')In the bottom-right pane, there is an IPython console window titled "Console 2/A". It displays the following session:In [1]: runfile('/Users/msdogan/untitled0.py', wdir='/Users/msdogan')
Hello python!

In [2]:A small blue "Usage" help dialog box is overlaid on the IPython console area.

Script editor

The screenshot shows the Spyder Python 2.7 interface. In the bottom-right pane, there is an IPython console window titled "Console 2/A". It displays the following session:In [1]: runfile('/Users/msdogan/untitled0.py', wdir='/Users/msdogan')
Hello python!

In [2]: print('Hello Python')
Hello Python

In [3]: |A small blue "Usage" help dialog box is overlaid on the IPython console area.

Console

# Creating & printing variables & inserting comment (#)

```
# this is the first comment  
spam = 1 # and this is the second comment  
print(spam)
```

The screenshot shows the Spyder Python IDE interface. On the left, the code editor displays a file named `untitled0.py` with the following content:

```
1#!/usr/bin/env python2  
2# -*- coding: utf-8 -*-  
3  
4Created on Tue Jul 25 12:40:38 2017  
5  
6@author: msdogan  
7  
8  
9#print('Hello python!')  
10  
11# this is the first comment  
12spam = 1 # and this is the second comment  
13print(spam)  
14
```

On the right, the IPython console shows the output of running the script:

```
In [1]: runfile('/Users/msdogan/untitled0.py', wdir='/Users/msdogan')  
1  
In [2]:
```

The help panel in the center provides information on using the Spyder interface.

# Using Python as a calculator

The screenshot shows the Spyder IDE interface for Python 2.7. The left pane is the Editor, displaying the contents of `untitled0.py`. The right pane is the IPython console, showing a history of calculations.

**Editor Content (`untitled0.py`):**

```
1 #!/usr/bin/env python2
2 # -*- coding: utf-8 -*-
3 """
4 Created on Tue Jul 25 12:40:38 2017
5
6 @author: msdogan
7 """
8
9 #print('Hello python!')
10
11 # this is the first comment
12 #spam = 1 # and this is the second comment
13 #print(spam)
14
15 # Using python as a calculator
16 print(2 + 2)
17 print(50 - 5*6)
18 print((50 - 5*6) / 4)
19 print(8 / 5) # division always returns a floating point number in v3+ not v2.7
20
21 print(17 / 3) # classic division returns a float in v3+
22 print(17 // 3) # floor division discards the fractional part
23 print(17 % 3) # the % operator returns the remainder of the division
24 print(5 * 3 + 2) # result * divisor + remainder
25
26 print(5 ** 2) # 5 squared
27 print(2 ** 7) # 2 to the power of 7
28
29 width = 20
30 height = 5 * 9
31 print(width * height)
32
```

**IPython Console History:**

```
In [1]: 2 + 2
Out[1]: 4

In [2]: 50 - 5*6
Out[2]: 20

In [3]: (50 - 5*6) / 4
Out[3]: 5

In [4]: 8 / 5 # division always returns a floating point number
Out[4]: 1.0

In [5]: 17 // 3
Out[5]: 5

In [6]:
```

At the bottom of the interface, status bars show: Permissions: RW, End-of-lines: LF, Encoding: UTF-8, Line: 32, Column: 1, Memory: 79%.

# Strings

Strings must be enclosed in single ' ' or double " " quotes

The screenshot shows the Spyder Python IDE interface. On the left is the code editor with a file named `untitled0.py` containing Python code demonstrating various string operations. In the center is a help panel titled "Usage" which provides information on how to get help for objects in the editor or console. On the right is the Python console window showing the output of running the script.

```
21 #print(17 / 3) # classic division returns a float in v3+
22 #print(17 // 3) # floor division discards the fractional part
23 #print(17 % 3) # the % operator returns the remainder of the division
24 #print(5 * 3 + 2) # result * divisor + remainder
25 #
26 #print(5 ** 2) # 5 squared
27 #print(2 ** 7) # 2 to the power of 7
28 #
29 #width = 20
30 #height = 5 * 9
31 #print(width * height)
32
33## Strings
34#a = 'orange'
35#b = "peach"
36#c = 'mango'
37#print(a + b + c) # concatenation
38#print(a,b,c)
39
40# Character position
41word = 'Python'
42print(len(word)) # number of characters in a string
43print(word[0]) # character in position 0
44print(word[5]) # character in position 5
45print(word[-1]) # last character
46print(word[-2]) # second-last character
47print(word[-6])
48print(word[0:2]) # characters from position 0 (included) to 2 (excluded)
49print(word[2:5])
50
51print(word[:2]) # character from the beginning to position 2 (excluded)
52print(word[:2] + word[2:])
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
```

Help Panel (Usage):

Here you can get help of any object by pressing Cmd+I in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in Preferences > Help.

New to Spyder? Read our [tutorial](#)

Python console:

```
Python 2.7.11 |Anaconda custom (x86_64)| (default, Dec  6 2015, 18:57:58)
[GCC 4.2.1 (Apple Inc. build 5577)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
Anaconda is brought to you by Continuum Analytics.
Please check out: http://continuum.io/thanks and https://anaconda.org
rg
>>> runfile('/Users/msdogan/untitled0.py', wdir='/Users/msdogan')
6
P
n
n
o
P
Py
tho
Py
Python
>>>
```

Bottom status bar:

Permissions: RW End-of-lines: LF Encoding: UTF-8 Line: 53 Column: 1 Memory: 75 %

# Ways to group, store and recall data

- List: [ ]
- Set: { }
- Dictionary: { }
- Array: `np.array()` (requires Numpy package)
- Dataframe: `pd.DataFrame()` (requires Pandas package)

# Lists

Python knows a number of *compound* data types, used to group together other values. The most versatile is the *list*, which can be written as a list of comma-separated values (items) between square brackets.

The screenshot shows the Spyder Python IDE interface. On the left, the code editor displays a file named 'tutorial.py' containing Python code demonstrating various list operations. In the center, the IPython console shows the execution of the code, displaying the output of each print statement. On the right, there is a 'Usage' panel providing help information for the 'print' function, and a 'Variable explorer' and 'File explorer' panel at the bottom.

```
46 #print(word[-2]) # second-last character
47 #print(word[-6])
48 #print(word[0:2]) # characters from position 0 (included) to 2 (excluded)
49 #print(word[2:5])
50 #
51 #print(word[:2]) # character from the beginning to position 2 (excluded)
52 #print(word[:2] + word[2:])
53
54 # Lists
55 squares = [1, 4, 9, 16, 25]
56 print(squares)
57 print(squares[0]) # indexing returns the item
58 print(squares[-1]) # slicing returns a new list
59 print(squares[3:-1]) # slicing returns a new list
60
61 cubes = [1, 8, 27, 65, 125] # something's wrong here
62 cubes[3] = 64 # replace the wrong value
63 print(cubes)
64
65 cubes.append(216) # add the cube of 6
66 cubes.append(7 ** 3) # and the cube of 7
67 print(cubes)
68
69 letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g']
70 # replace some values
71 letters[2:5] = ['C', 'D', 'E']
72 print(letters)
73 print(len(letters))
74
75 # It is possible to nest lists (create lists containing other lists)
76 a = ['a', 'b', 'c']
77 n = [1, 2, 3]
78 x = [a, n]
79 print(x)
80 print(x[0])
81 print(x[0][1])
82
83
84
85
86
87
88
89
90
91
92
93
```

```
In [1]: runfile('/Users/msdogan/Desktop/tutorial.py', wdir='/Users/msdogan/Desktop')
[1, 4, 9, 16, 25]
1
25
[9, 16, 25]
[1, 8, 27, 64, 125]
[1, 8, 27, 64, 125, 216, 343]
['a', 'b', 'C', 'D', 'E', 'f', 'g']
7
[['a', 'b', 'c'], [1, 2, 3]]
['a', 'b', 'c']
b

In [2]:
```

Permissions: RW End-of-lines: LF Encoding: UTF-8 Line: 82 Column: 1 Memory: 74 %

# Sets

- A set is an unordered collection with no duplicate elements
- Sets support mathematical operations like union, intersection, difference, and symmetric difference

The screenshot shows the Spyder Python IDE interface. The code editor window displays a script named `tutorial.py` with the following content:

```
57 #print(squares[0]) # indexing returns the item
58 #print(squares[-1])
59 #print(squares[-3:]) # slicing returns a new list
60 #
61 #cubes = [1, 8, 27, 65, 125] # something's wrong here
62 #cubes[3] = 64 # replace the wrong value
63 #print(cubes)
64 #
65 #cubes.append(216) # add the cube of 6
66 #cubes.append(7 ** 3) # and the cube of 7
67 #print(cubes)
68 #
69 #letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g']
70 ## replace some values
71 #letters[2:5] = ['C', 'D', 'E']
72 #print(letters)
73 #print(len(letters))
74 #
75 ## It is possible to nest lists (create lists containing other lists)
76 #x = ['a', 'b', 'c']
77 #n = [1, 2, 3]
78 #x = [a, n]
79 #print(x)
80 #print(x[0])
81 #print(x[0][1])
82 #
83 # basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}
84 print(basket) # show that duplicates have been removed
85 print('orange' in basket) # fast membership testing
86 print('crabgrass' in basket)
87 #
88 # set operations
89 a = set('abracadabra')
90 b = set('atacamaz')
91 #
92 print(a) # unique letters in a
93 print(a - b) # letters in a but not in b
94 print(a | b) # letters in a or b or both
95 print(a & b) # letters in both a and b
96 print(a ^ b) # letters in a or b but not both
97 #
98
99
100
101
102
103
104
```

The IPython console window shows the following session:

```
In [1]: runfile('/Users/msdogan/Desktop/tutorial.py', wdir='/Users/msdogan/Desktop')
set(['orange', 'pear', 'banana', 'apple'])
True
False
set(['a', 'r', 'b', 'c', 'd'])
set(['r', 'b', 'd'])
set(['a', 'c', 'b', 'd', 'm', 'l', 'r', 'z'])
set(['a', 'c'])
set(['b', 'd', 'm', 'l', 'r', 'z'])

In [2]:
```

Other visible panels include the Source, Console, and Object tabs in the top right, and a Usage help panel on the right side.

# Dictionaries

- Dictionaries are indexed by *keys*
- The main operations on a dictionary are storing a value with some key and extracting the value given the key

The screenshot shows the Spyder Python IDE interface. On the left, the code editor displays a file named `tutorial.py` with Python code demonstrating various dictionary operations. In the center, a help panel titled "Usage" provides information on how to get help for objects. On the right, the IPython console shows the execution of the script, outputting the created dictionaries and their contents.

```
85 #print(basket) # show that duplicates have been removed
86 #print('orange' in basket) # fast membership testing
87 #print('crabgrass' in basket)
88 #
89 ## set operations
90 #a = set('abracadabra')
91 #b = set('alacazam')
92 #print(a) # unique letters in a
93 #print(a - b) # letters in a but not in b
94 #print(a | b) # letters in a or b or both
95 #print(a & b) # letters in both a and b
96 #print(a ^ b) # letters in a or b but not both
97
98 # Dictionaries
99 tel = {'jack': 4098, 'sape': 4139}
100 tel['guido'] = 4127 # assing a new element
101 print(tel)
102 print(tel['jack'])
103 del tel['sape'] # delete an element
104 tel['irv'] = 4127
105 print(tel)
106 print(tel.keys()) # print dictionary keys
107 print('guido' in tel)
108 print('jack' not in tel)
109
110 # building a dictionary directly from sequences of key-value pairs:
111 d1 = dict([('sape', 4139), ('guido', 4127), ('jack', 4098)])
112 print(d1)
113 # building a dictionary from pairs using keyword arguments
114 d2 = dict(sape=4139, guido=4127, jack=4098)
115 print(d2)
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
```

Here you can get help of any object by pressing Cmd+I in front of it, either on the Editor or the Console.  
Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in Preferences > Help.

New to Spyder? Read our [tutorial](#)

```
In [1]: runfile('/Users/msdogan/Desktop/tutorial.py', wdir='/Users/msdogan/Desktop')
{'sape': 4139, 'jack': 4098, 'guido': 4127}
4098
{'jack': 4098, 'irv': 4127, 'guido': 4127}
['jack', 'irv', 'guido']
True
False
{'sape': 4139, 'jack': 4098, 'guido': 4127}
{'sape': 4139, 'jack': 4098, 'guido': 4127}
```

Permissions: RW End-of-lines: LF Encoding: UTF-8 Line: 116 Column: 1 Memory: 67 %

# Numpy Arrays np.array( )

- `import numpy as np`
- Homogeneous multidimensional array
- Table of elements, all of the same type

The screenshot shows the Spyder Python 2.7 IDE interface. On the left, there are two tabs: 'temp.py' and 'tutorial.py'. The 'temp.py' tab contains a script with various Numpy array creation examples. The 'tutorial.py' tab is currently active. On the right side, there is a 'Usage' panel which provides help for the current object being selected. Below the usage panel is a 'Console 1/A' window showing the output of running the tutorial script. The output includes the definition of arrays 'a' and 'b', their shapes, dtypes, and values, followed by the creation of arrays 'c', 'a2', 'x', 'y', and 'z'. The 'z' array is shown as a 3x3 matrix of floating-point numbers.

```
temp.py
101 tel['guido'] = 4127 # assing a new element
102 print(tel)
103 #print(tel['jack'])
104 #del tel['sape'] # delete an element
105 #tel['irv'] = 4127
106 #print(tel)
107 #print(tel.keys()) # print dictionary keys
108 #print('guido' in tel)
109 #print('jack' not in tel)
110 #
111 ## building a dictionary directly from sequences of key-value pairs:
112 d1 = dict([('sape', 4139), ('guido', 4127), ('jack', 4098)])
113 print(d1)
114 ## building a dictionary from pairs using keyword arguments
115 d2 = dict(sape=4139, guido=4127, jack=4098)
116 #print(d2)
117
118 # Numpy arrays
119 a = np.arange(15).reshape(3, 5) # create a 3x5 array with elements from 0-14
120 print(a.shape)
121 print(a.ndim)
122 print(a.dtype.name)
123 print(a.size)
124 print(type(a))
125
126 a1 = np.array([6, 7, 8]) # create a numpy array from a list
127 print(a1.dtype)
128 b = np.array([1.2, 3.5, 5.1]) # float array
129 print(b.dtype)
130 b = np.array([(1.5,2,3), (4,5,6)]) # 2D array
131 print(b)
132 c = np.array( [ [1,2], [3,4] ], dtype=complex )
133 print(c)
134
135 a2 = np.arange( 10, 30, 5 ) # first item, last item, step size
136 print(a2)
137
138 x = np.zeros(15).reshape(3, 5) # create an array with zeros
139 print(x)
140 x = np.zeros((3,5)) # create an array with zeros
141 print(x)
142 y = np.ones((2,3,4), dtype=np.int16) # create an array with ones
143 print(y)
144 z = np.empty((2,3)) # empty array , output may vary
145 print(z)
146
147
148
149
```

```
In [17]: runfile('/Users/msdogan/Google Drive/Python_tutorial/tutorial.py', wdir='/Users/msdogan/Google Drive/Python_tutorial')
(3, 5)
2
int64
15
<type 'numpy.ndarray'>
int64
float64
[[ 1.5  2.   3. ]
 [ 4.   5.   6. ]]
[[ 1.+0.j  2.+0.j]
 [ 3.+0.j  4.+0.j]]
[10. 15. 20. 25.]
[[ 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. ]]
[[ 1.  1.  1. ]
 [ 1.  1.  1. ]
 [ 1.  1.  1. ]]

[[ 1.  1.  1. ]
 [ 1.  1.  1. ]
 [ 1.  1.  1. ]]

[[ 1.39069238e-309  1.39069238e-309  1.39069238e-309]
 [ 1.39069238e-309  1.39069238e-309  1.39069238e-309]]
```

Permissions: RW End-of-lines: LF Encoding: UTF-8 Line: 146 Column: 1 Memory: 71 %

# Some operations w/ Numpy arrays

The screenshot shows the Spyder Python 2.7 IDE interface. The left pane is the code editor with two tabs: 'temp.py' and 'tutorial.py'. The 'temp.py' tab contains the following code:

```
140 #x = np.zeros((3,5)) # create an array with zeros
141 #print(x)
142 #y = np.ones((2,3,4), dtype=np.int16) # create an array with ones
143 #print(y)
144 #z = np.empty((2,3)) # empty array , output may vary
145 #print(z)
146
147 # Basic operations with Numpy arrays
148 A = np.array( [[1,1],[0,1]] )
149 print(A)
150 B = np.array( [[2,0],[3,4]] )
151 print(B)
152 print(A*B) # elementwise product
153 print(np.dot(A,B)) # matrix product, same as A.dot(B)
154
155 # sum, min, max
156 np.random.seed(101) # if you don't set the seed, you get different results each time
157 a = np.random.random((2,3)) # array with random elements from 0 to 1
158 print(a)
159 print(a.sum())
160 print(a.mean())
161 print(a.min(),a.max())
162
163 b = np.arange(12).reshape(3,4)
164 print(b)
165 print(b.sum(axis=0)) # sum of each column
166 print(b.min(axis=1)) # min of each row
167 print(b.cumsum(axis=1)) # cumulative sum along each row
168 print(np.exp(b)) # exponential of an array
169 print(np.sqrt(b)) # square root of an array
170
```

The right pane has three main sections: 'Help' (with a 'Usage' sub-section), 'Variable explorer', and 'File explorer'. Below these is the 'IPython console' which displays the output of the code execution. The output includes various Numpy array definitions and their properties.

Permissions: RW End-of-lines: LF Encoding: UTF-8 Line: 170 Column: 1 Memory: 75 %

17

# Pandas data frames pd.DataFrame( )

- import pandas as pd
  - Tabular data structure with labeled columns & rows
  - Dictionary-like container for series objects

The screenshot shows a Jupyter Notebook interface with two tabs open: 'temp.py' and 'tutorial.py'. The 'temp.py' tab contains Python code demonstrating various operations like elementwise product, matrix product, sum, min, max, and random number generation. It also shows how to work with Pandas DataFrames, including creating a DataFrame from a dictionary, setting column types, and displaying specific rows and columns.

```
153 #print(A*B) # elementwise product
154 #print(np.dot(A,B)) # matrix product, same as A.dot(B)
155 #
156 ## sum, min, max
157 #np.random.seed(101) # if you don't set the seed, you get different results each time
158 #a = np.random.random((2,3)) # array with random elements from 0 to 1
159 #print(a)
160 #print(a.sum())
161 #print(a.mean())
162 #print(a.min(),a.max())
163 #
164 #b = np.arange(12).reshape(3,4)
165 #print(b)
166 #print(b.sum(axis=0)) # sum of each column
167 #print(b.min(axis=1)) # min of each row
168 #print(b.cumsum(axis=1)) # cumulative sum along each row
169 #print(np.exp(b)) # exponential of an array
170 #print(np.sqrt(b)) # square root of an array
171
172 # Pandas data frames
173 df1 = pd.DataFrame({ 'A' : 1,
174                      'B' : pd.Timestamp('20130102'),
175                      'C' : pd.Series(1,index=list(range(4)),dtype='float32'),
176                      'D' : np.array([3] * 4,dtype='int32'),
177                      'E' : pd.Categorical(["test","train","test","train"]),
178                      'F' : 'foo' })
179 print(df1)
180 #print(df1.dtypes)
181
182 dates = pd.date_range('20130101', periods=6)
183 df = pd.DataFrame(np.random.randn(6,4), index=dates, columns=list('ABCD'))
184 print(df)
185 print(df.keys()) # data frame keys (columns)
186 print(df.index) # row index
187 print(df.columns) # column name
188 print(df['A']) # display column 'A'
189 print(df[0:3]) # display first 3 column
190 print(df.loc['20130102':'20130104,['A','B']]) # display defined rows and columns
191 print(df.iloc[2]) # display 3rd row
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
```

The 'File explorer' panel on the right shows the directory structure of the user's home folder, including 'anaconda', 'AnacondaProjects', 'Applications', 'Desktop', and 'Documents'.

The 'Variable explorer' and 'File explorer' tabs are visible at the top of the right panel. The 'IPython console' tab is active, showing the output of the code execution. The output includes the creation of the DataFrame 'df1', its structure, and the resulting DataFrame 'df' with six rows and four columns ('A', 'B', 'C', 'D') containing random float values.

```
In [67]: runfile('/Users/msdogan/Google Drive/Python_tutorial/tutorial.py', wdir='/Users/msdogan/Google Drive/Python_tutorial')
   A      B      C      D
0  1.0  2013-01-02  1.0  test
1  1.0  2013-01-02  1.0  train
2  1.0  2013-01-02  1.0  test
3  1.0  2013-01-02  1.0  train
   A      B      C      D
2013-01-01  1.591424 -1.637978  1.753722 -1.566144
2013-01-02  1.068526 -2.757239  1.748523 -0.702016
2013-01-03  0.190926 -0.086846 -0.350664  1.547742
2013-01-04  1.070922 -0.060915  0.471558  0.950493
2013-01-05  -0.306435  0.861754  1.063417 -0.600040
2013-01-06  -1.161053  0.339276 -0.897061  0.432775
Index(['A', 'B', 'C', 'D'], dtype='object')
DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03',
               '2013-01-04', '2013-01-05', '2013-01-06'],
              dtype='datetime64[ns]', freq='D')
Index(['A', 'B', 'C', 'D'], dtype='object')
2013-01-01  1.591424
2013-01-02  1.068526
2013-01-03  0.190926
2013-01-04  1.070922
2013-01-05  -0.306435
2013-01-06  -1.161053
Freq: D, Name: A, dtype: float64
   A      B      C      D
2013-01-01  1.591424 -1.637978  1.753722 -1.566144
2013-01-02  1.068526 -2.757239  1.748523 -0.702016
2013-01-03  0.190926 -0.086846 -0.350664  1.547742
   A      B
2013-01-02  1.068526 -2.757239
2013-01-03  0.190926 -0.086846
2013-01-04  1.070922 -0.060915
A  0.190926
B -0.086846
C -0.350664
D  1.547742
Name: 2013-01-03 00:00:00, dtype: float64
```

The 'IPython console' tab shows the output of the code execution, including the creation of the DataFrame 'df1', its structure, and the resulting DataFrame 'df' with six rows and four columns ('A', 'B', 'C', 'D') containing random float values.

```
In [68]:
```

# Some operations w/ Pandas dataframes

The screenshot shows a Jupyter Notebook interface with the following components:

- Editor:** Contains two files: `temp.py` and `tutorial.py*`. The `tutorial.py` file contains Python code demonstrating various Pandas operations.
- File explorer:** Shows the local directory structure under `/Users/msdogan`.
- Variable explorer:** Shows variables defined in the current session, including `df` (a DataFrame).
- Console 1/A:** Displays the output of running `tutorial.py`. It includes descriptive statistics for the DataFrame `df`, such as mean, std, min, max, and quantiles, along with the raw data for each row.
- Python console:** Shows the command `In [70]: runfile('/Users/msdogan/Google_Drive/Python_tutorial/tutorial.py', wdir='/Users/msdogan/Google_Drive/Python_tutorial')` and its output.
- History log:** Shows previous commands run in the session.
- IPython console:** Shows the command `In [71]:` and its output.

Key code from `tutorial.py`:

```
167 #print(b.min(axis=1)) # min of each row
168 #print(b.cumsum(axis=1)) # cumulative sum along each row
169 #print(np.exp(b)) # exponential of an array
170 #print(np.sqrt(b)) # square root of an array
171
172 # Pandas data frames
173 #df = pd.DataFrame({ 'A' : 1.,
174 #                      'B' : pd.Timestamp('20130102'),
175 #                      'C' : pd.Series(1,index=list(range(4)),dtype='float32'),
176 #                      'D' : np.array([3]*4,dtype='int32'),
177 #                      'E' : pd.Categorical(["test","train","test","train"]),
178 #                      'F' : 'foo' })
179 #print(df)
180 #print(df.dtypes)
181
182 dates = pd.date_range('20130101', periods=6)
183 df = pd.DataFrame(np.random.randn(6,4), index=dates, columns=list('ABCD'))
184 #print(df)
185 #print(df.keys()) # data frame keys (columns)
186 #print(df.index) # row index
187 #print(df.columns) # column name
188 #print(df['A']) # display column 'A'
189 #print(df[0:3]) # display first 3 column
190 #print(df.loc['20130102':'20130104',['A','B']]) # display defined rows and columns
191 #print(df.iloc[2]) # display 3rd row
192
193 # some basic operations with pandas data frames
194 print(df.head(3)) # first 3 rows
195 print(df.tail(2)) # last 2 rows
196 print(df.describe()) # stats
197 print(df.T) # transpose
198 print(df.sort_values(by='B', ascending=True))
199 print(df.mean()) # mean of each column (mean of each row df.mean(1))
200 print(df['A'].sum()) # sum of column A
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
```

# if statements: if , elif , else

- Less than: <
- Less than or equal: <=
- Greater than: >
- Greater than or equal: >=
- Equal: ==
- Not equal: != or <>

The screenshot shows a Jupyter Notebook environment with the following details:

- File Explorer:** Shows the directory structure at `/Users/msdogan`, including `anaconda`, `AnacondaProjects`, `Applications`, `Desktop`, and `Documents`.
- Editor:** Displays the contents of `temp.py` and `tutorial.py*`. The `temp.py` file contains basic Pandas operations like `head`, `tail`, `describe`, and various `print` statements. The `tutorial.py*` file contains Python code demonstrating if statements, including handling negative numbers, zero, one, and more values.
- Console:** Shows the output of the IPython session. It includes the Python version (`Python 2.7.11 |Anaconda custom (x86_64)| (default, Dec 6 2015, 18:57:58)`), copyright information, and help documentation for `object?`. The IPython prompt shows the execution of `In [1]: runfile('/Users/msdogan/Google Drive/Python tutorial/tutorial.py', wdir='/Users/msdogan/Google Drive/Python tutorial')` resulting in `Single 6`. The next cell, `In [2]:`, is currently empty.
- Bottom Status Bar:** Provides system information: Permissions: RW, End-of-lines: LF, Encoding: UTF-8, Line: 217, Column: 1, Memory: 76 %.

# Loops: `for` & `while` statements

- For loops: Iterate over the items of any sequence (a list or a string), in the order that they appear in the sequence
- While loops: continue until exit criteria met

The screenshot shows a Python development environment with the following components:

- Editor:** Displays two files: `temp.py` and `tutorial.py`. `temp.py` contains code demonstrating various loop constructs, including `for` loops over lists and ranges, and a `while` loop. `tutorial.py` contains a Fibonacci series generator function.
- File explorer:** Shows the directory structure at `/Users/msdogan`, including `anaconda`, `AnacondaProjects`, `Applications`, `Desktop`, and `Documents`.
- IPython console:** Displays the Python 2.7.11 environment and the execution of `tutorial.py`. The output shows the execution of the code from line 1 to line 251, printing the results of the loops and the Fibonacci series.

```
temp.py
208 #     print('Zero')
209 elif x == 1:
210 #     print('Single')
211 else:
212 #     print('More')
213 #
214 #if x != 5: # if x is not equal to 5
215 #    x += 5 # then set x = x + 5
216 #    print(x)
217
218 # Loops: for statements
219 words = ['cat', 'window', 'demonstrate']
220 # iterate over items
221 for w in words:
222     print(w, len(w))
223 # iterate over indices
224 for i in range(len(words)):
225     print(i)
226 # iterate over both indices and items
227 for i,w in enumerate(words):
228     print(i,w)
229
230 # double for loop
231 for n in range(2, 10):
232     for x in range(2, n):
233         if n % x == 0:
234             print(n, 'equals', x, '*', n//x)
235             break
236         else:
237             # loop fell through without finding a factor
238             print(n, 'is a prime number')
239
240 # while loop
241 # write a function that contains a while loop
242 def fib(n): # return Fibonacci series up to n
243     # Return a list containing the Fibonacci series up to n.
244     result = []
245     a, b = 0, 1
246     while a < n:
247         result.append(a) # see below
248         a, b = b, a+b
249     return result
250 print(fib(100))
251
252
253
254
255
256
257
258
259
260
261
262
263
264

tutorial.py
Python 2.7.11 |Anaconda custom (x86_64)| (default, Dec  6 2015, 18:57:58)
Type "copyright", "credits" or "license" for more information.

IPython 5.3.0 -- An enhanced Interactive Python.
?           -- Introduction and overview of IPython's features.
%quickref -- Quick reference.
help       -- Python's own help system.
object?   -- Details about 'object', use 'object??' for extra
details.

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

In [2]:
```

# Loading & saving data

- Load / save data from / to csv and txt
- A few different ways: numpy, csv, and pandas

The screenshot shows a Jupyter Notebook environment with several panes:

- Editor:** Displays the Python script `temp.py` containing code for generating Fibonacci numbers, calculating square roots of integers, and saving data to CSV files.
- Variable explorer:** Shows variables `x1` and `x2` defined in the notebook.
- File explorer:** Shows the file structure at `/Users/msdogan`.
- Console 5/A:** Displays the output of running the script, showing the results of the calculations and the saved CSV files.
- In [110]:** Shows the command `runfile('/Users/msdogan/Google Drive/Python_tutorial/tutorial.py', wdir='/Users/msdogan/Google Drive/Python_tutorial')` and its output, which includes the generated CSV files and their contents.
- In [111]:** Shows the command `Index([u'x1_x2'], dtype='object')` and its output, which lists the columns `x1` and `x2` with their respective values.

# Data visualization

- `import matplotlib.pyplot as plt`
- Line plot
- Scatter plot
- Pie chart
- Bar chart
- Histogram
- Bonus: pandas plot

The screenshot shows a Jupyter Notebook environment with the following components:

- Editor:** A code editor window containing a Python script named `tutorial.py`. The script includes imports for `matplotlib.pyplot`, various plotting examples (line, scatter, pie, bar, histogram), and a section for plotting with pandas.
- File explorer:** A sidebar showing the file system structure of the user's home directory (`/Users/msdogan`).
- Console 1/A:** An IPython console window displaying the output of the plotted data. It shows two line plots: one blue line labeled `x1` and one orange line labeled `x2`. Both lines start at (0, 1) and end at (4, 3). The x-axis ranges from 0.0 to 4.0, and the y-axis ranges from 1 to 9.
- Permissions:** RW
- End-of-lines:** LF
- Encoding:** UTF-8
- Line:** 401
- Column:** 1
- Memory:** 69 %