

# 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

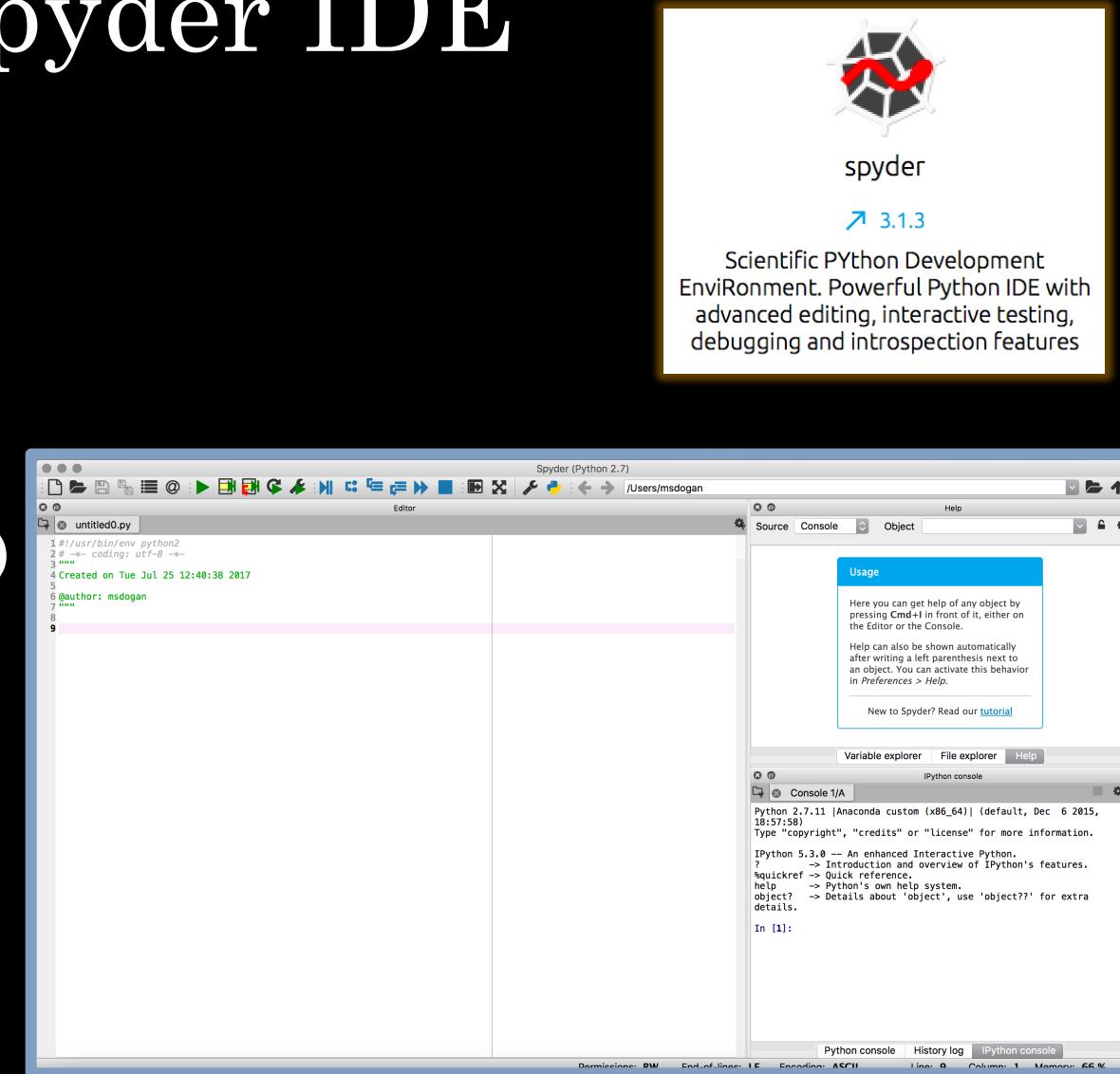
- Anaconda distribution includes almost all packages you will need to run Python
- Link to software (or just google Anaconda)

<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 for "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 and availability. Another paragraph discusses the interpreter and standard library. A sidebar on the right contains a "Back" link and a detailed description of the tutorial's purpose and scope, mentioning the standard library, language reference, and API manual. The footer of the page includes a copyright notice for Python Software Foundation.

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. On the left is the script editor with a file named `untitled0.py` containing the following 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!')
```

Below the editor is the IPython console window. It displays the following output:

```
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/untitled0.py', wdir='/Users/
msdogan')
Hello python!

In [2]:
```

At the bottom of the console, status information is shown: Permissions: RW, End-of-lines: LF, Encoding: UTF-8, Line: 8, Column: 1, Memory: 73 %.

Script editor

The screenshot shows the Spyder Python 2.7 interface with the console window active. It displays the following command and output:

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

Below the console, status information is shown: Permissions: RW, End-of-lines: LF, Encoding: UTF-8, Line: 8, Column: 1, Memory: 73 %.

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 is the code editor with a file named 'untitled0.py' containing the provided Python code. The code includes a multi-line comment starting with '# this is the first comment'. To the right of the editor is a help panel titled 'Usage' which provides information on how to get help for objects. Below the editor is the IPython console, which shows the output of running the script. The console output includes the creation timestamp, author information, and the printed value '1'. At the bottom of the interface, there are tabs for 'Python console', 'History log', and 'IPython console', along with system status information like permissions, encoding, and memory usage.

```
#!/usr/bin/env python2  
# -*- coding: utf-8 -*-  
"""  
Created on Tue Jul 25 12:40:38 2017  
@author: msdogan  
"""  
print('Hello python!')  
# this is the first comment  
spam = 1 # and this is the second comment  
print(spam)  
1  
In [1]: runfile('/Users/msdogan/untitled0.py', wdir='/Users/msdogan')  
1  
In [2]:
```

# Using Python as a calculator

The screenshot shows the Spyder IDE interface for Python 2.7. On the left, the Editor tab displays the contents of `untitled0.py`. The code includes various print statements and comments demonstrating basic arithmetic operations and division rules in Python 2. On the right, the IPython console tab shows the execution of these commands. A blue box highlights the "Usage" section of the help documentation, which explains how to get help for objects using `Cmd+I`.

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Tue Jul 25 12:40:38 2017
@author: msdogan
"""

# print('Hello python!')

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

# Using python as a calculator
print(2 + 2)
print(50 - 5*6)
print((50 - 5*6) / 4)
print(8 / 5) # division always returns a floating point number in v3+ not v2.7

print(17 / 3) # classic division returns a float in v3+
print(17 // 3) # floor division discards the fractional part
print(17 % 3) # the % operator returns the remainder of the division
print(5 * 3 + 2) # result * divisor + remainder

print(5 ** 2) # 5 squared
print(2 ** 7) # 2 to the power of 7

width = 20
height = 5 * 9
print(width * height)

```

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](#)

```
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]:
```

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, the code editor displays a script named `untitled0.py` with the following content:

```
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
41 word = 'Python'
42 print(len(word)) # number of characters in a string
43 print(word[0]) # character in position 0
44 print(word[5]) # character in position 5
45 print(word[-1]) # last character
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
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
```

In the center, the Help panel has a "Usage" section with instructions on how to get help for objects. At the bottom of the Help panel, it says "New to Spyder? Read our [tutorial](#)".

On the right, the Python console window shows the output of running the script:

```
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
>>> runfile('/Users/msdogan/untitled0.py', wdir='/Users/msdogan')
6
P
n
n
o
P
Py
tho
Py
Python
>>>
```

At the bottom of the Spyder window, there are status bars for "Permissions: RW", "End-of-lines: LF", "Encoding: UTF-8", "Line: 53", "Column: 1", and "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. The Editor tab displays the file `tutorial.py` with Python code demonstrating various list operations. The IPython console tab shows the execution of the code, resulting in the output of lists and their elements. The Help panel provides information on using the Spyder interface.

```
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])
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
```

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](#)

Variable explorer File explorer Help

IPython console

Console 1/A

In [1]: runfile('/Users/msdogan/Desktop/tutorial.py', wdir='/Users/msdogan/Desktop')
[1, 4, 9, 16, 25]
1
25
[0, 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 2.7 IDE interface. The left side features a code editor with a file named 'tutorial.py' containing Python code demonstrating set operations. The right side includes a 'Help' panel titled 'Usage' and an 'IPython console' window.

**tutorial.py:**

```
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 # Sets
84 basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}
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('atacazam')
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
```

**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](#)

**IPython console:**

```
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]:
```

Permissions: RW End-of-lines: LF Encoding: UTF-8 Line: 97 Column: 1 Memory: 69 %

# 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` containing Python code related to sets and dictionaries. The IPython console on the right shows the execution of the script, demonstrating the creation of sets and dictionaries, and printing their contents. A help panel titled "Usage" provides information on how to get help in the Spyder environment.

```
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
```

```
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, the 'temp.py' editor tab is open, displaying Python code for creating various Numpy arrays. On the right, the 'Console 1/A' tab shows the execution of the code and its output. A 'Usage' help panel is visible, providing information on how to get help for objects.

```
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. ]  
[ 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 1]  
[ 1 1 1 1]]  
[[ -1.39069238e-309 1.39069238e-309 1.39069238e-309]  
[ -1.39069238e-309 1.39069238e-309 1.39069238e-309]]

In [18]:

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 IDE interface. On the left, the code editor displays a script named `tutorial.py` containing Python code for creating and manipulating NumPy arrays. On the right, the IPython console shows the execution of the script, displaying various array operations and their results.

```
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
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
```

Usage  
Here you can get help of any object by pressing Cmd+I in front of it, either on the Editor or the Console.  
Variable explorer File explorer Help  
In [35]: runfile('/Users/msdogan/Google Drive/Python tutorial/tutorial.py', wdir='/Users/msdogan/Google Drive/Python tutorial')
[[1 1]
 [0 1]
 [[2 0]
 [3 4]]
 [[2 0]
 [0 4]
 [[5 4]
 [3 4]
 [[ 0.51639863 0.57066759 0.02847423]
 [ 0.17152166 0.68527698 0.83389686]]
 2.80623594161
 0.467705990268
 (0.028474226478096942, 0.83389686263607654)
 [[ 0 1 2 3]
 [ 4 5 6 7]
 [ 8 9 10 11]
 [12 15 18 21]
 [0 4 8]
 [[ 0 1 3 6]
 [ 4 9 15 22]
 [ 8 17 27 38]
 [[ 1.0000000e+00 2.71828183e+00 7.38905610e+00 2.00855369e
 +01]
 [ 5.45981500e+01 1.48413159e+02 4.03428793e+02 1.09663316e
 +03]
 [ 2.98095799e+03 8.10308393e+03 2.20264658e+04 5.98741417e
 +04]]
 [[ 0. 1. 1.41421356 1.73205081]
 [ 2. 2.23606798 2.44948974 2.64575131]
 [ 2.82842712 3. 3.16227766 3.31662479]]]

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

# 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 environment with the following components:

- Editor:** Displays the Python script `temp.py` containing code related to Pandas operations like dot products, array statistics, and DataFrame creation.
- File explorer:** Shows the file structure of the user's home directory.
- Variable explorer:** Lists variables defined in the current session.
- Console 1/A:** Displays the execution of `tutorial.py` and its output.
- Output:** Shows the resulting DataFrame `df1` with columns A, B, C, D, E, F and their corresponding values.

```
In [67]: runfile('/Users/msdoggan/Google Drive/Python_tutorial/tutorial.py', wdir='/Users/msdoggan/Google Drive/Python_tutorial')
A   B   C   D   E   F
0  1.0  2013-01-02  1.0  3  test  foo
1  1.0  2013-01-02  1.0  3  train  foo
2  1.0  2013-01-02  1.0  3  test  foo
3  1.0  2013-01-02  1.0  3  train  foo
4  1.591424 -1.637978  1.753722 -1.566144
5  1.068526 -2.757239  1.748523 -0.702016
6  0.190926 -0.086846 -0.350664  1.547742
7  1.070922 -0.060915  0.471550  0.950493
8  0.306435  0.861754 -0.063417 -0.600040
9  -1.161053  0.339276 -0.897051  0.432775
Index([u'A', u'B', u'C', u'D'], dtype='object')
DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03',
               '2013-01-04', '2013-01-05', '2013-01-06'],
              freq='D', Name: A, dtype: float64
In [68]:
```

Permissions: RW End-of-lines: LF Encoding: UTF-8 Line: 192 Column: 1 Memory: 72 %

# Some operations w/ Pandas dataframes

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

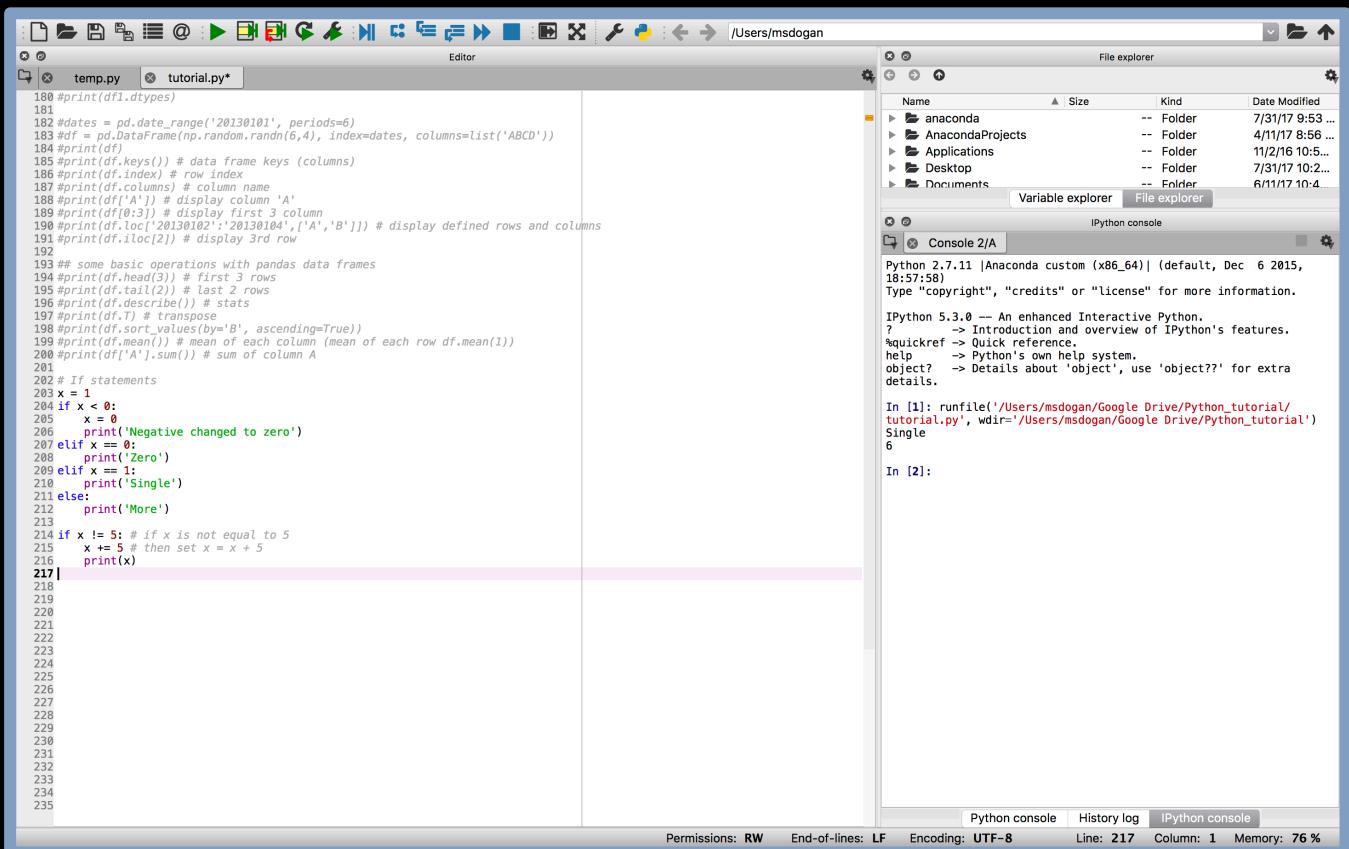
- Editor:** Contains two files: `temp.py` and `tutorial.py*`. The `tutorial.py` file contains Python code demonstrating various Pandas operations.
- File explorer:** Shows the local directory structure under `/Users/msdogan`.
- Variable explorer:** Shows variables defined in the current session, including `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 interface 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 `tutorial.py` and `temp.py`. The `tutorial.py` file contains code demonstrating various pandas operations and an if statement example.
- Variable Explorer:** Shows variables defined in the current session.
- Console 2/A:** Displays the Python environment and help documentation.
- In [1]:** The command run was `runfile('/Users/msdogan/Google Drive/Python tutorial/tutorial.py', wdir='/Users/msdogan/Google Drive/Python tutorial')`. The output is "Single 6".
- In [2]:** The command run was `runfile('/Users/msdogan/Google Drive/Python tutorial/tutorial.py', wdir='/Users/msdogan/Google Drive/Python tutorial')`. The output is "More".
- Bottom Status Bar:** Shows permissions (RW), end-of-lines (LF), encoding (UTF-8), line (217), column (1), and memory usage (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.

# 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 a list of square root pairs and the contents of the generated CSV files.
- In [111]:** Shows the command `Index([u'x1_x2'], dtype='object')` and its output, which is the same list of square root pairs.
- Permissions:** RW  
End-of-lines: LF  
Encoding: UTF-8  
Line: 293  
Column: 1  
Memory: 78 %

# 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 %

# Linear Programming (LP) with CVXPY

- CVXPY is a Python-embedded modeling language for convex optimization problems
- It allows you to express your problem in a natural way that follows the math
- User's guide: <http://www.cvxpy.org>
- Install CVXPY by running the following command:
  - Windows/Mac OS: `pip install cvxpy`

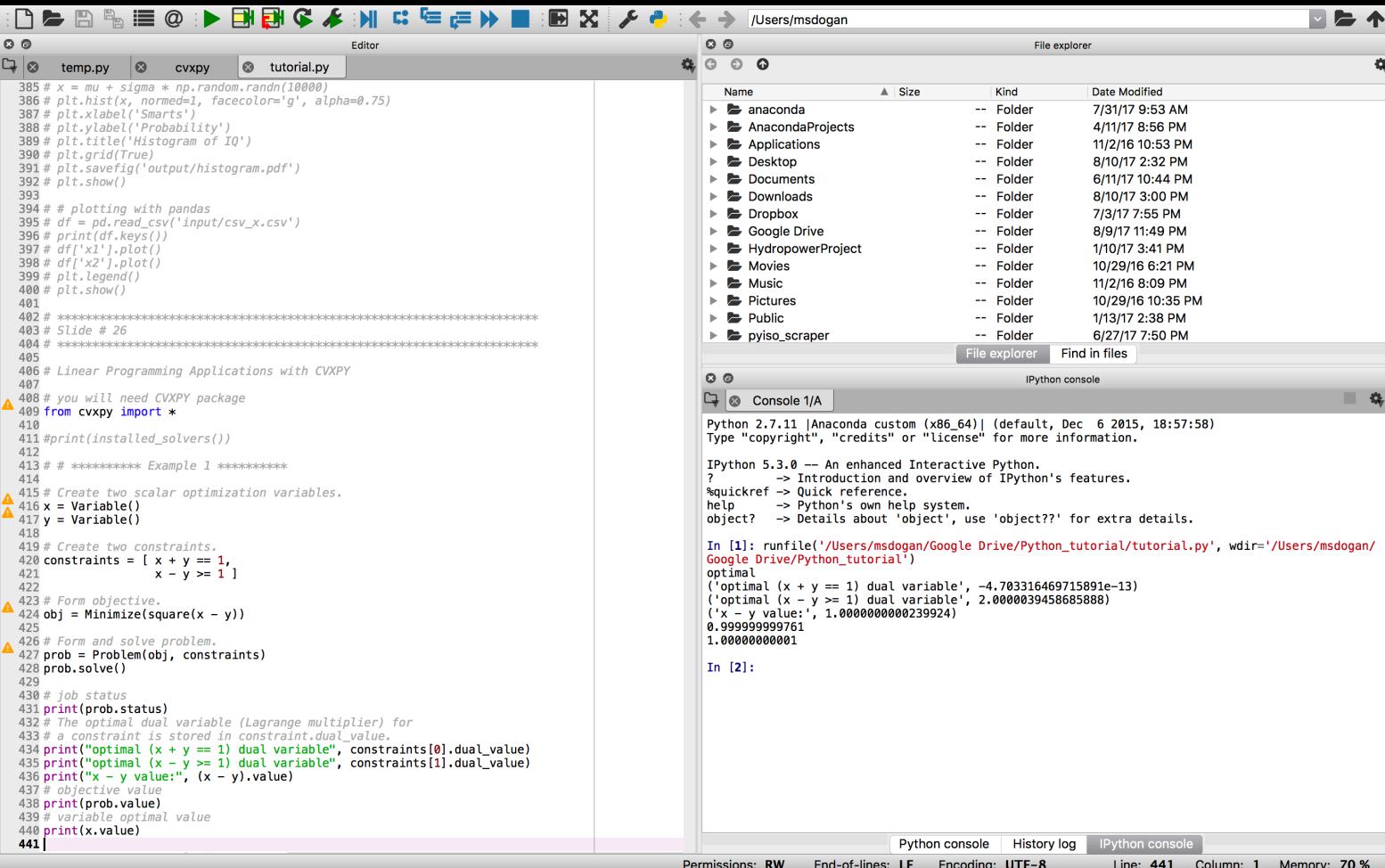
# CVXPY optimization package

- Variables
  - Variables can be scalars, vectors, or matrices.
  - `x = Variable()`, `y = Variable(5)`, or `z = Variable(4, 7)`
- Constraints
  - You can use `==`, `<=`, and `>=`
  - constraints are elementwise
  - `constraints = [ x + y == 1, x - y >= 1 ]`
- Parameters
  - Parameters are symbolic representations of constants
  - `c = Parameter(sign='negative')`, `c.value = 5`
- Objective
  - `obj = Minimize(square(x - y))`, `obj = Maximize(3*car + 5*truck)`

# Example 1

Objective:  
*minimize*  $(x - y)^2$

Subject to:  
 $x + y = 1, x - y \geq 1$



# Example 2

- CVXPY problem with vectors and matrices
- Retrieve:
  - objective value
  - optimal values of variables
  - dual values of constraints

Example:

$$\min \sum (Ax - b)$$

$$\text{s.t. } 0 \leq x \leq 1$$

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

- Editor:** Displays the code for a CVXPY problem. The code defines a problem with two constraints ( $x + y = 1$  and  $x - y = 1$ ), prints the dual values, and then constructs a new problem with a sum-squares objective and prints its optimal value.
- File explorer:** Shows the file structure of the user's home directory, including folders like anaconda, Applications, Desktop, Documents, Downloads, Dropbox, Google Drive, HydropowerProject, Movies, Music, Pictures, Public, and pyiso\_scrapers.
- Console 2/A:** An IPython console window showing the Python version (2.7.11), copyright information, and help documentation for various commands like %quickref, %help, and object?. It also shows the execution of two cells:

  - In [1]: `runfile('/Users/msdogan/Google Drive/Python tutorial/tutorial.py', wdir='/Users/msdogan/Google Drive/Python tutorial')`
  - In [2]: `runfile('/Users/msdogan/Google Drive/Python tutorial/tutorial.py', wdir='/Users/msdogan/Google Drive/Python tutorial')`

- Output:** The output of the second cell shows the optimal x values and first constraint dual values.

# ECI 153 Clyde Motor Company

Deterministic Optimization and Design  
UC Davis

Jay R. Lund  
Fall 2015

Today we have a graphical example of a product-mix type problem frequently solved by linear programs.

**Clyde Motor Company Example**  
A company has 3 factories that together produce two products: cars and small trucks.

Each factory specializes in certain components required for both products.  
Plant 1 makes truck bodies.  
Plant 2 makes car bodies.  
Plant 3 makes shared components and assembles trucks and cars.

Each plant has a limited capacity.  
Each product is sold for a fixed price with a fixed profit per unit.

Data:	Product	Capacity Available (units/hour)
Plant	1      2	4
1	1      0	
2	0      2	12
3	3      2	18
Unit Profit	\$3      \$5	--

**Step 1: Define the Problem:**

- ) Max profit
- ) S.T. limited plant capacities
- ) How many cars and trucks should be produced?

**Steps 2 and 3: Express the Problem Mathematically:**

Max  $Z = 3X_1 + 5X_2$  = profit per hour of operation, say.

Subject To:

$$\begin{array}{ll} (1) \quad X_1 & \leq 4 \\ (2) \quad 2X_2 & \leq 12 \\ (3) \quad 3X_1 + 2X_2 & \leq 18 \\ (4) \quad X_1 & \geq 0 \\ (5) \quad X_2 & \geq 0 \end{array}$$

$X_1$  = Number of small trucks produced.  
 $X_2$  = Number of cars produced.

Explanation of constraints:

$$\begin{array}{l} (1) \quad X_1 \leq 4 \\ (2) \quad 2X_2 \leq 12 \\ (3) \quad 3X_1 + 2X_2 \leq 18 \\ (4) \quad X_1 \geq 0 \\ (5) \quad X_2 \geq 0 \end{array} \left. \begin{array}{l} \text{capacity constraints} \\ \text{on production} \\ \text{cannot produce} \\ \text{negative products} \end{array} \right\}$$

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

- Editor:** Displays the `tutorial.py` file containing Python code for defining variables, creating constraints, and solving the optimization problem using the ECOS solver.
- File explorer:** Shows the directory structure at `/Users/msdogan`, including folders like anaconda, AnacondaProjects, Applications, Desktop, Documents, Downloads, Dropbox, Google Drive, HydropowerProject, Movies, Music, Pictures, Public, and pyiso\_scraper.
- Console 4/A:** Displays the output of the ECOS solver, showing the optimal values for  $X_1$  and  $X_2$ .
- IPython console:** Shows the command `In [1]: runfile('/Users/msdogan/Google Drive/Python_tutorial/tutorial.py', wdir='/Users/msdogan/Google Drive/Python_tutorial')` and the resulting output from the ECOS solver.

```

472 # *****
473 # SLIDE # 28
474 # *****
475
476 # ***** Example 3 *****
477 ***
478 # A company has 3 factories that together produce two products: cars and small trucks.
479 # Each factory specializes in certain components required for both products.
480 Plant 1 makes truck bodies.
481 Plant 2 makes car bodies.
482 Plant 3 makes shared components and assembles trucks and cars.
483
484 Each plant has limited capacity:
485 Plant 1: less than 4 car per hour (car <= 4)
486 Plant 2: less than 6 trucks per hour (truck <= 6)
487 Plant 3: less than 6 cars or 9 trucks per hour (3*car + 2*truck <= 18)
488
489 Each product is sold for a fixed price with a fixed profit per unit:
490 car: $3, truck: $5
491
492 How many cars and truck should be produced to maximize profit per hour?
493 ***
494 # define variables
495 car = Variable()
496 truck = Variable()
497
498 # create constraints
499 constraints = [ car <= 4, # capacity constraints on production
500                 truck <= 6, # capacity constraints on production
501                 3*car + 2*truck <= 18, # capacity constraints on production
502                 car >= 0, # cannot produce negative products
503                 truck >= 0 ] # cannot produce negative products
504
505 # Form objective.
506 obj = Maximize(3*car + 5*truck)
507
508 # Form and solve problem.
509 prob = Problem(obj, constraints)
510 prob.solve(verbose=True)
511
512 # job status
513 print(prob.status)
514 print('optimal number of cars: ', int(car.value)) # optimal number of cars
515 print('optimal number of trucks: ', int(truck.value)) # optimal number of trucks
516 print('profit ($ per hour: ', round(obj.value,2)) # objective value
517 # dual values (Lagrange multipliers)
518 print('first constraint dual value: ', round(constraints[0].dual_value,2)) # first constraint
519 print('second constraint dual value: ', round(constraints[1].dual_value,2)) # second constraint
520 print('third constraint dual value: ', round(constraints[2].dual_value,2)) # third constraint
521
522 # plot results
523 plt.bar([0],car.value)
524 plt.bar([1],truck.value)
525 plt.xticks([0,1,['car','truck']])
526 plt.ylabel('Production (Unit/hour)')
527 plt.title('Optimal Factory Production')
528 plt.show()

ECOS 2.0.4 - (C) embotech GmbH, Zurich Switzerland, 2012-15. Web: www.embotech.com/ECOS
It pcost dcost gap pres dres k/t mu step sigma IR BT
0 -2.602e+01 -8.200e+01 +4e+01 8e-04 3e-01 1e+00 6e+00 --- --- 1 1 - - 0 0
1 -3.325e+01 -4.223e+01 +6e+00 1e-04 6e-02 2e-01 1e+00 0.8463 3e-02 0 0 0 0 0 0
2 -3.589e+01 -3.628e+01 +3e-01 5e-06 3e-03 1e-02 0.9604 6e-03 0 0 0 0 0 0 0 0
3 -3.600e+01 -3.600e+01 +3e-03 5e-08 3e-05 2e-04 5e-04 0.9890 1e-04 1 0 0 0 0 0 0 0
4 -3.600e+01 -3.600e+01 +3e-05 5e-10 4e-07 2e-06 6e-06 0.9890 1e-04 1 0 0 0 0 0 0 0
5 -3.600e+01 -3.600e+01 +4e-07 7e-12 4e-09 2e-08 7e-08 0.9890 1e-04 1 0 0 0 0 0 0 0
6 -3.600e+01 -3.600e+01 +4e-09 7e-14 4e-11 2e-10 7e-10 0.9890 1e-04 1 0 0 0 0 0 0 0

OPTIMAL (within feastol=4.5e-11, reltol=1.2e-10, abstol=4.2e-09).
Runtime: 0.000413 seconds.

optimal
('optimal number of cars: ', 1)
('optimal number of trucks: ', 5)
('profit ($ per hour: ', 36.0)
('first constraint dual value: ', 0.0)
('second constraint dual value: ', 3.0)
('third constraint dual value: ', 1.0)

Permissions: RW End-of-lines: LF Encoding: UTF-8 Line: 529 Column: 1 Memory: 74 %

```