A Tutorial on the Python Programming Language

by Ricardo Aler

The print Statement

- It can be used to print results and variables
- Elements separated by commas print with a space between them
- A comma at the end of the statement (print ‘hello’,) will not print a newline character

```
>>> print 'hello'
hello
>>> print 'hello', 'there'
hello there
```

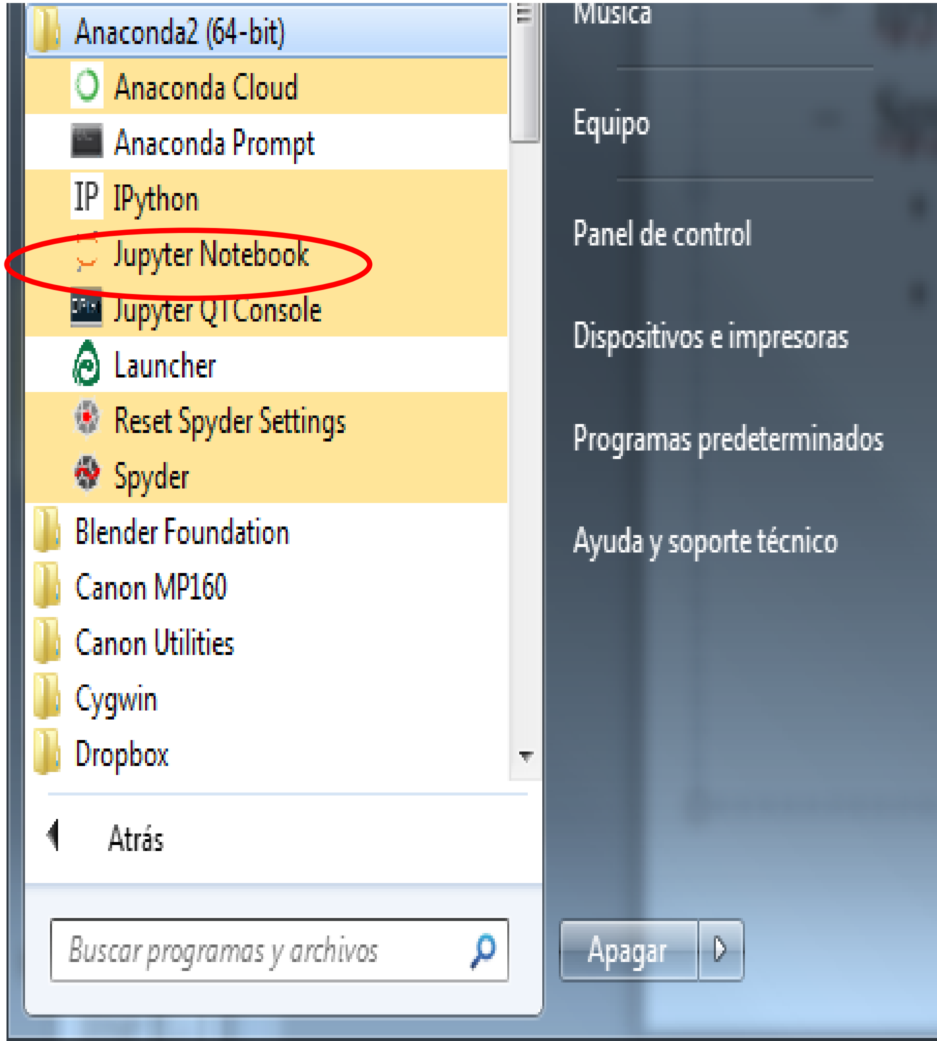
Comments

The ‘#’ starts a line comment

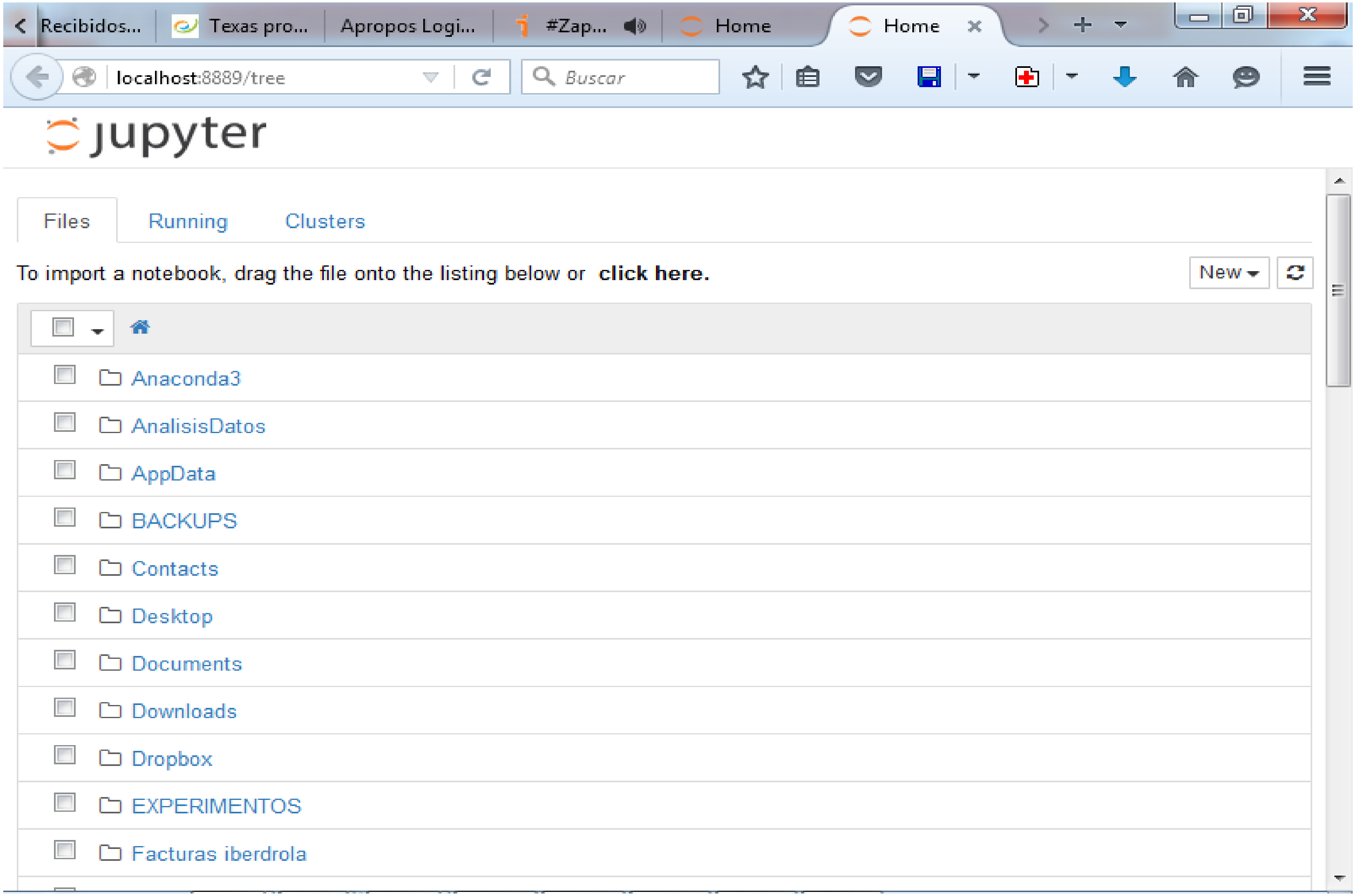
```
>>> 'this will print'
'this will print'
>>> #'this will not'
>>>
```

Using the ipython-notebook

- We already know how to use the qt-console
- The ipython-notebook is similar, but works in the **browser**, and allows to keep a record of the Python session

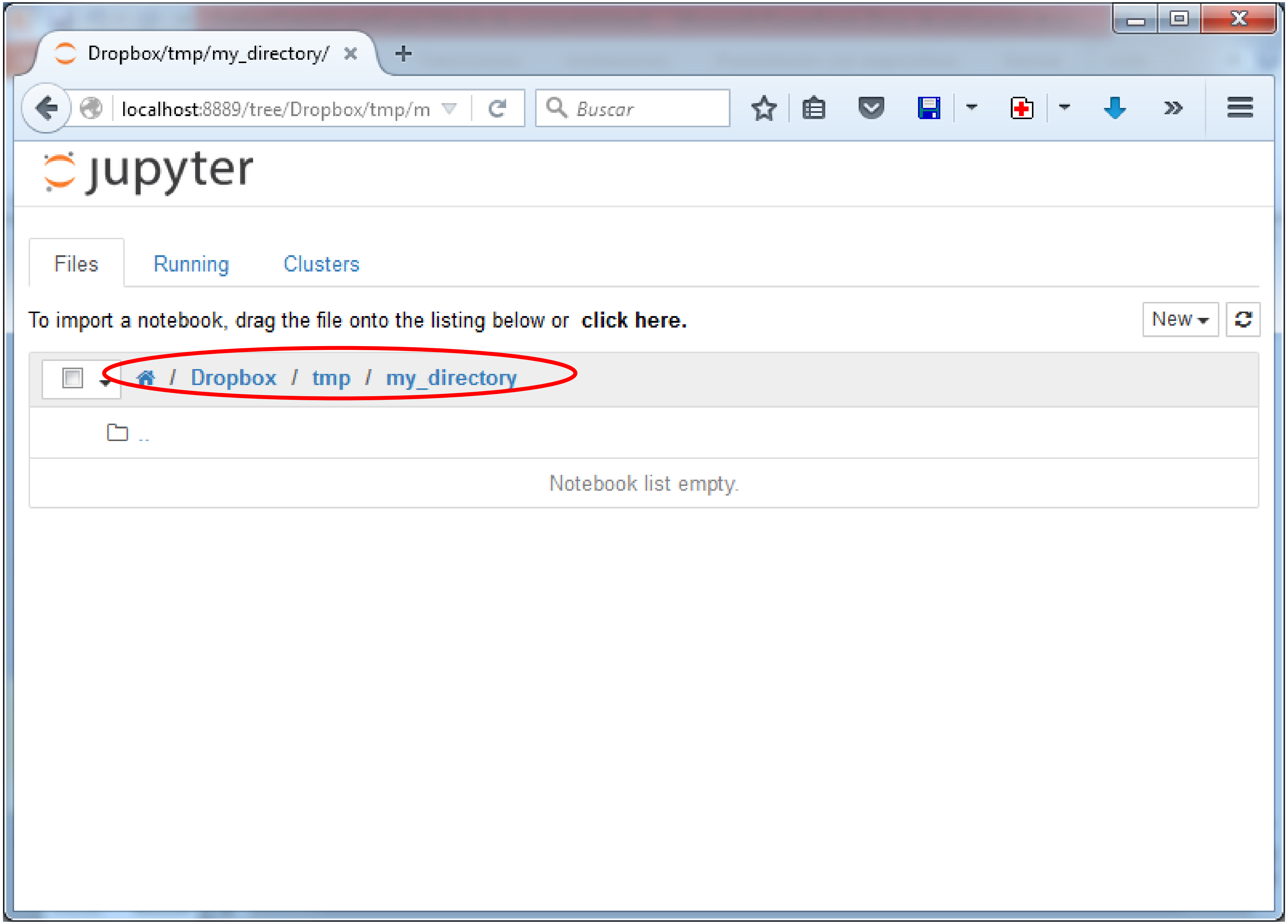


- A new tab will open in your default browser
- Now, you have to go to your directory



To import a notebook, drag the file onto the listing below or **click here.**

- □ Anaconda3
- □ AnalisisDatos
- □ AppData
- □ BACKUPS
- □ Contacts
- □ Desktop
- □ Documents
- □ Downloads
- □ Dropbox
- □ EXPERIMENTOS
- □ Facturas iberdrola



Dropbox/tmp/my_directory/

localhost:8889/tree/Dropbox/tmp/m

Buscar

jupyter

Files | Running | Clusters

To import a notebook, drag the file onto the listing below or **click here.**

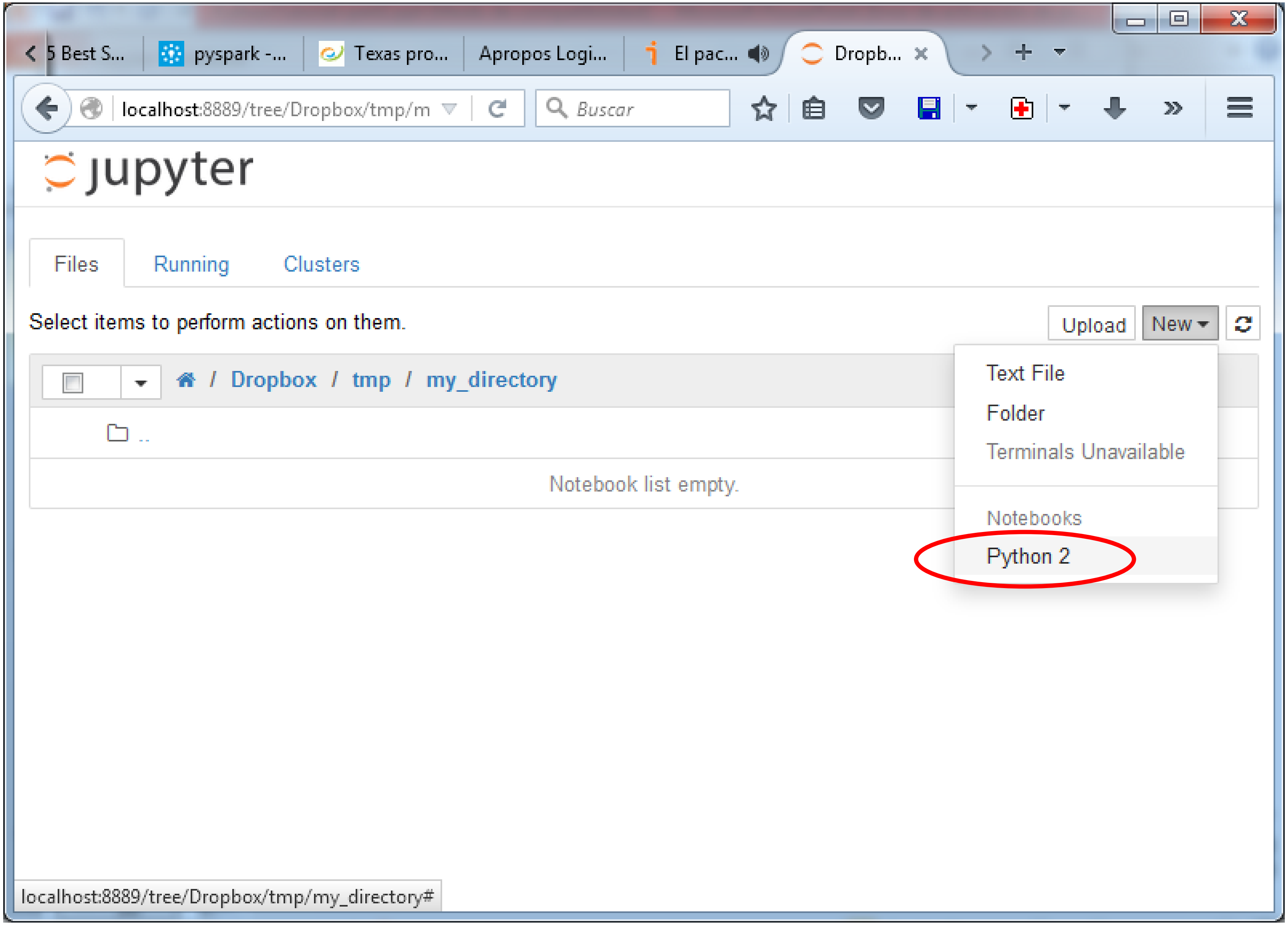
New ▾

□ / Dropbox / tmp / my_directory

..

Notebook list empty.

- Start a Python 2 notebook



Select items to perform actions on them.

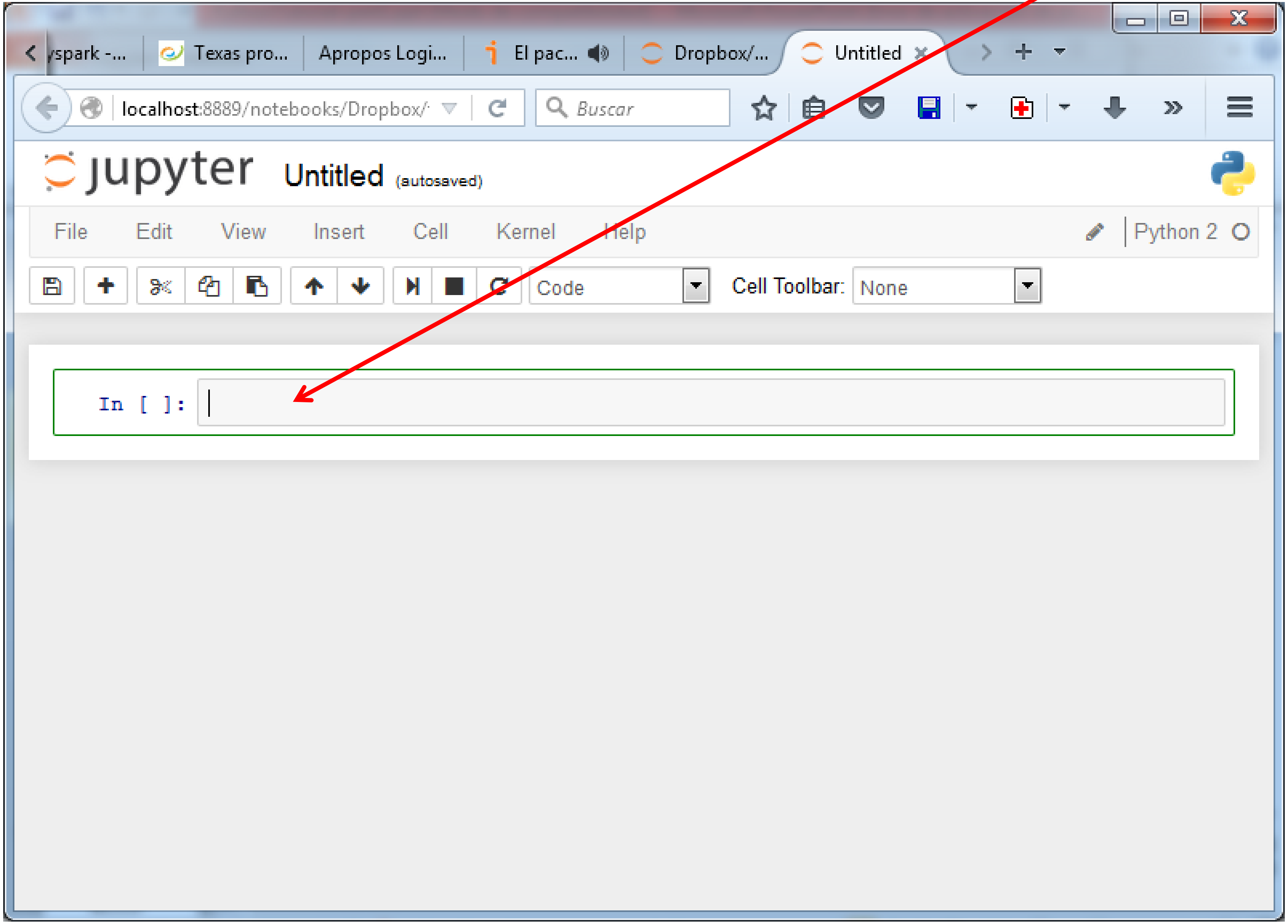
🏠 / Dropbox / tmp / my_directory

..

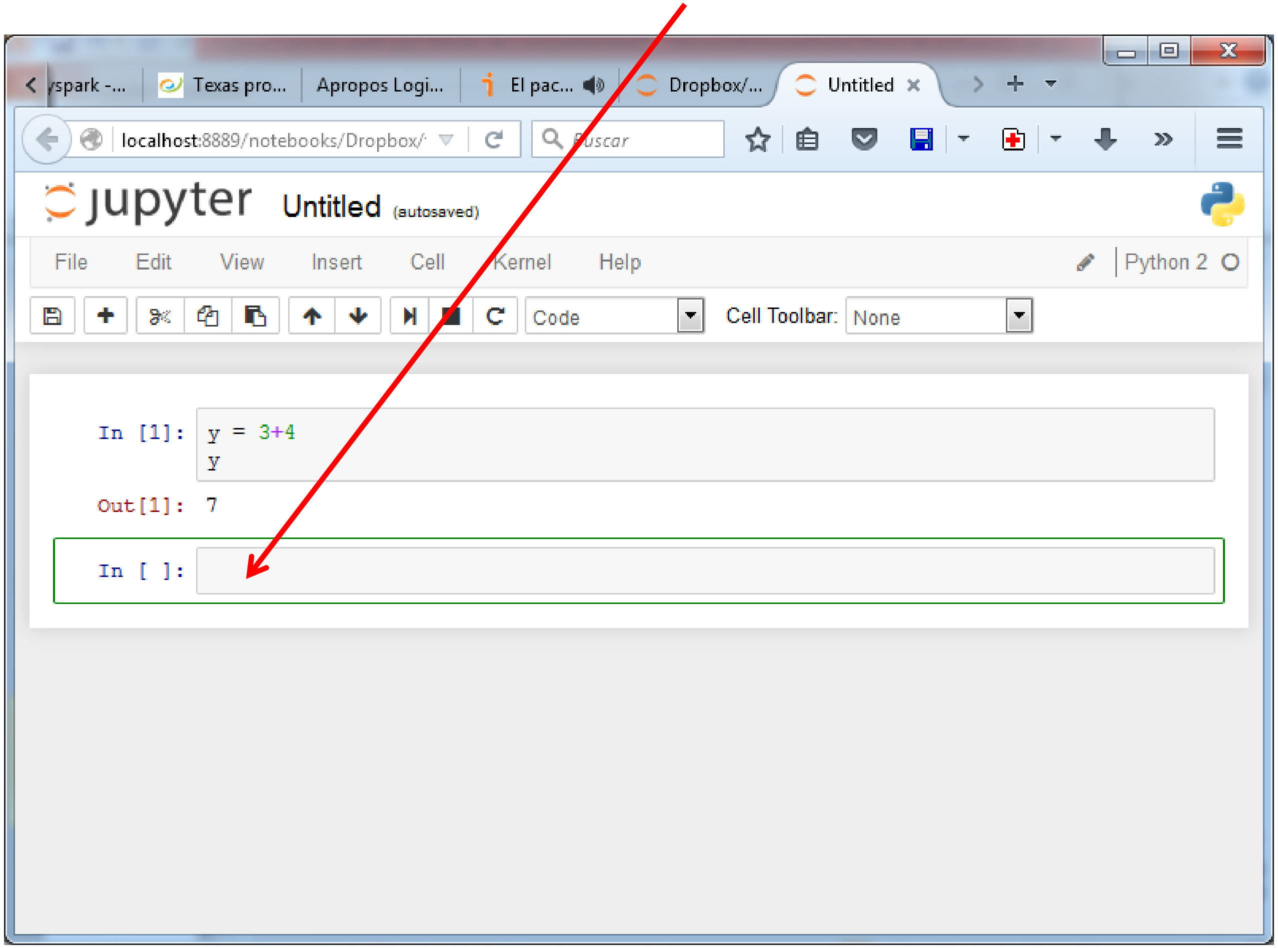
Notebook list empty.

localhost:8889/tree/Dropbox/tmp/my_directory#

- You can type python commands in the cell

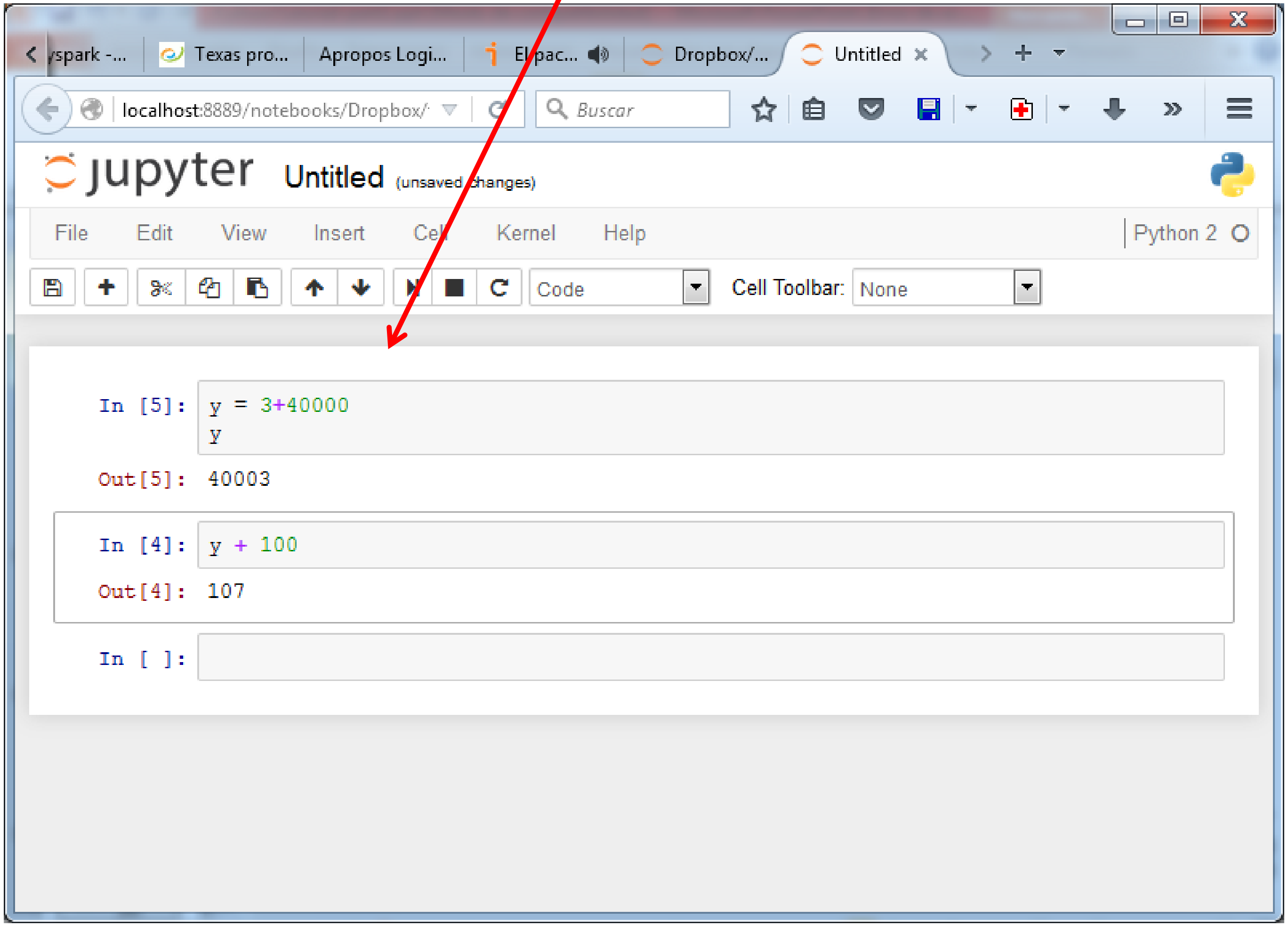


- Important:
  - "Enter" changes to a new line WITHIN the cell
  - In order to execute the commands in the cell, you have to type shift+enter
  - Once you type shift+enter, a new cell is created. You can type new commands



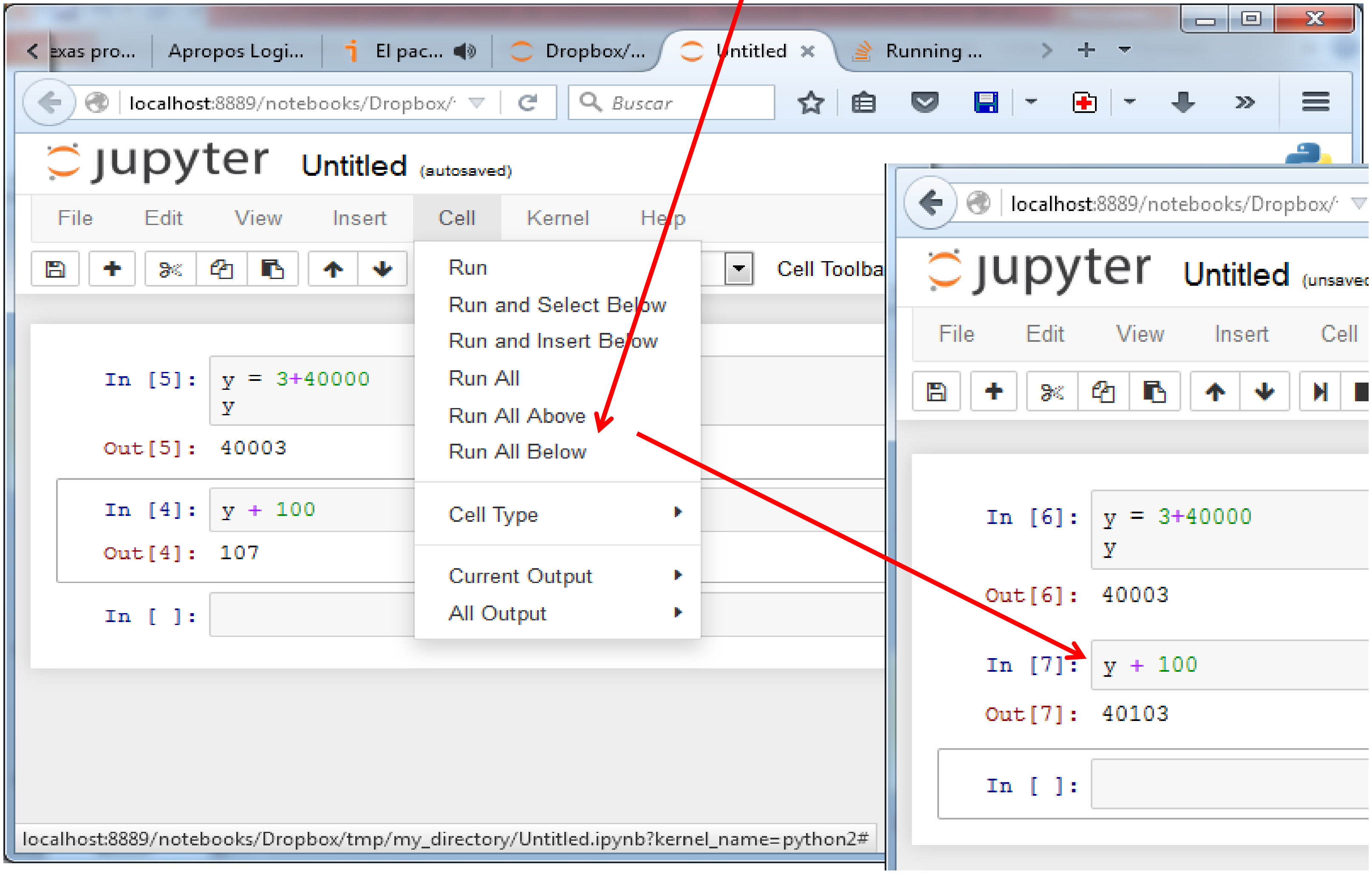
```
In [1]: y = 3+4
        y

Out[1]: 7

In [ ]:
```

- You can return to a previous cell and change it. You need to re-execute it with shift+enter (or ctrl+enter)



```
In [5]: y = 3+40000
        y

Out[5]: 40003

In [4]: y + 100

Out[4]: 107

In [ ]:
```

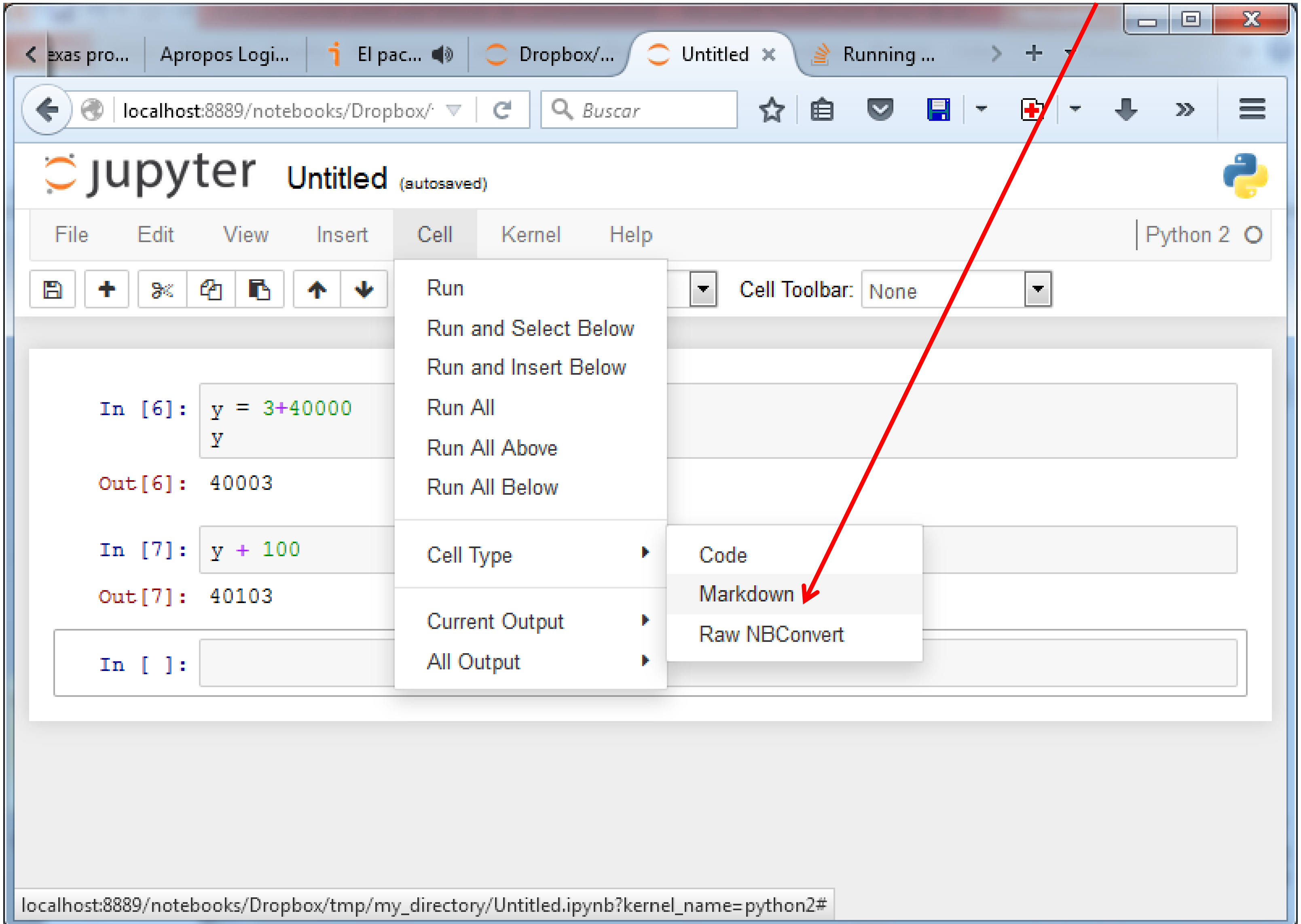
- If you want the changes to propagate to the following cells, you have to execute all of them again.



```
In [5]: y = 3+40000
        y
Out[5]: 40003

In [4]: y + 100
Out[4]: 107

In [ ]:
```

```
In [6]: y = 3+40000
        y
Out[6]: 40003

In [7]: y + 100
Out[7]: 40103

In [ ]:
```

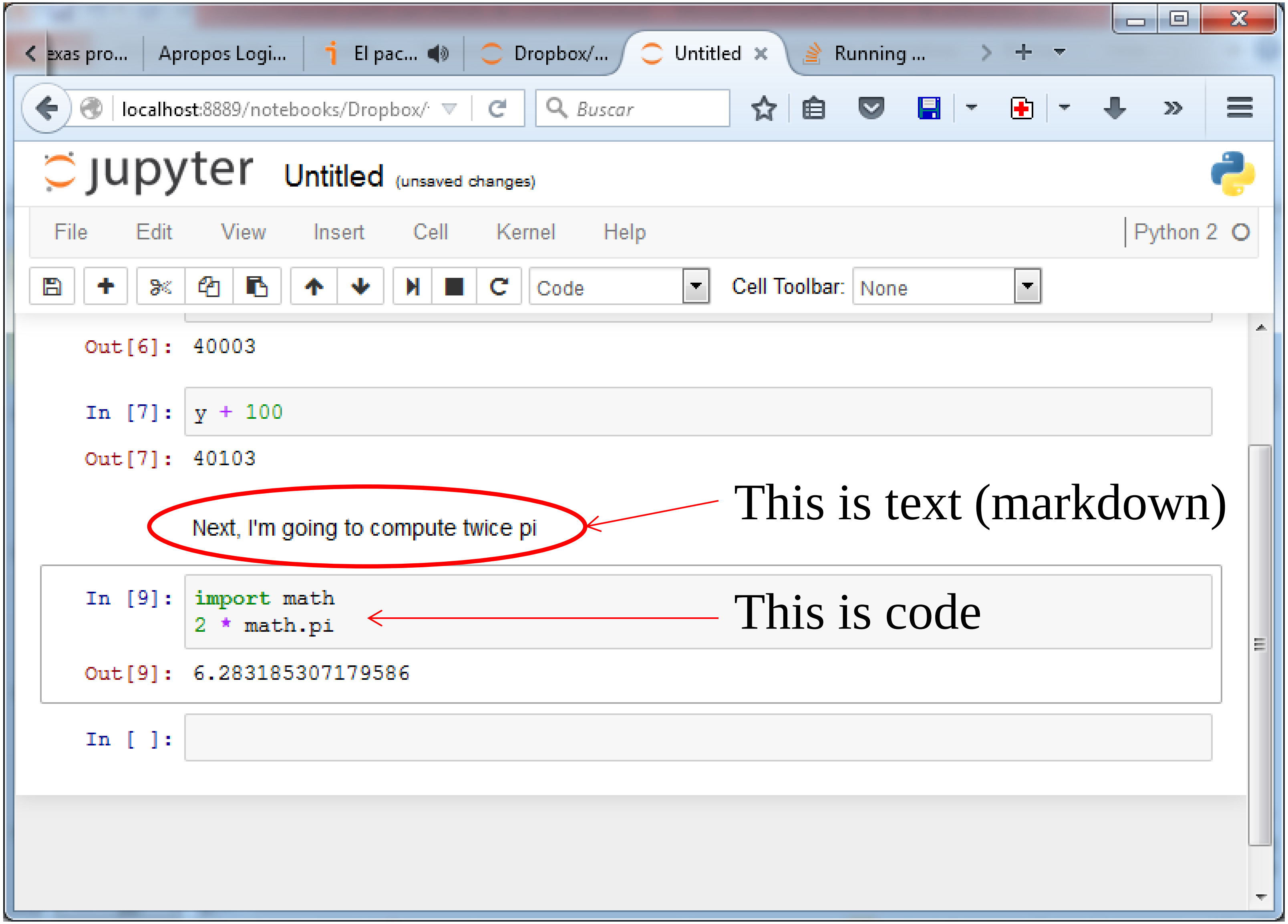
- In a Python notebook, you can mix text, python commands and results, by changing the cell type



```
In [6]: y = 3+40000
        y
Out[6]: 40003

In [7]: y + 100
Out[7]: 40103

In [ ]:
```

localhost:8889/notebooks/Dropbox/tmp/my_directory/Untitled.ipynb?kernel_name=python2#

- Text mixed with code



```
Out[6]: 40003

In [7]: y + 100

Out[7]: 40103
```

Next, I'm going to compute twice pi

```
In [9]: import math
        2 * math.pi

Out[9]: 6.283185307179586

In [ ]:
```

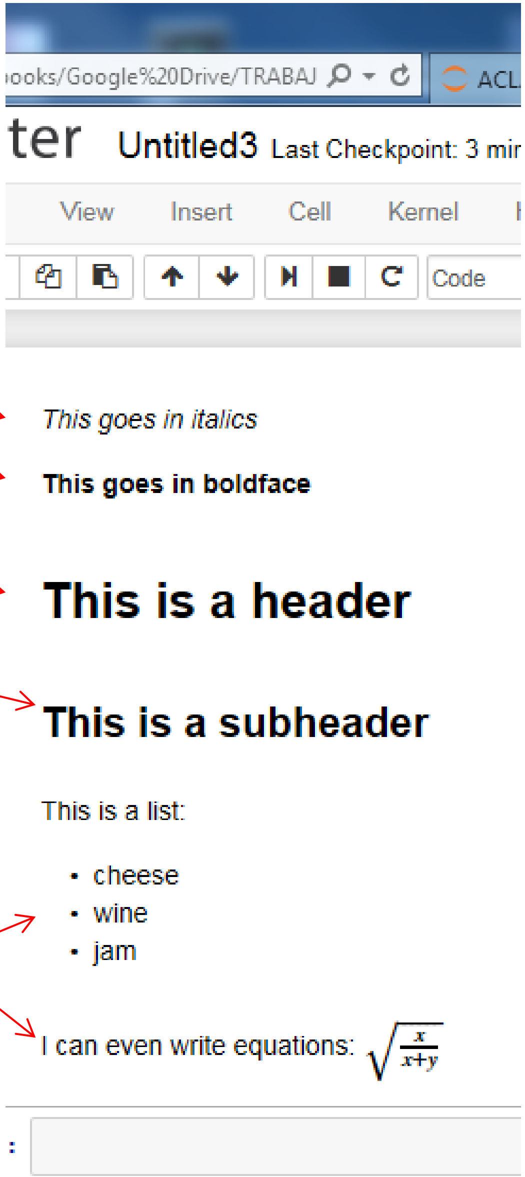
This is text (markdown)

This is code

Markdown

- Markdown is a language to format text:
  - *this goes in italics*
  - **this goes in boldface**
  - #This is a header
  - ##This is a subheader
  - I can even write equations (in LaTeX):
    - $\sqrt{\frac{x}{x+y}}$

This is a list:
- Cheese
- Wine
- Jam



This goes in italics

This goes in boldface

# This is a header

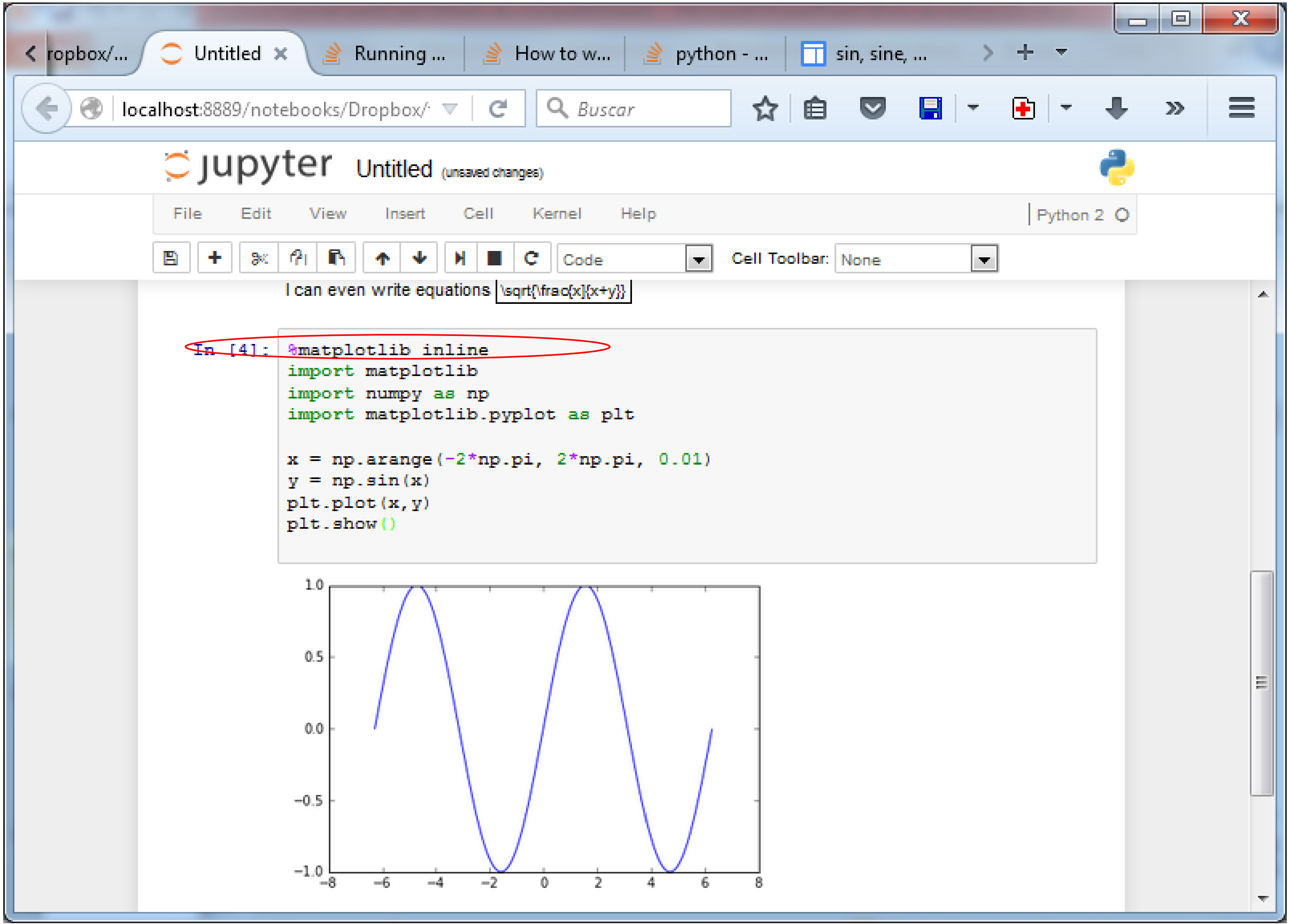
## This is a subheader

This is a list:

- cheese
- wine
- jam

I can even write equations: $\sqrt{\frac{x}{x+y}}$

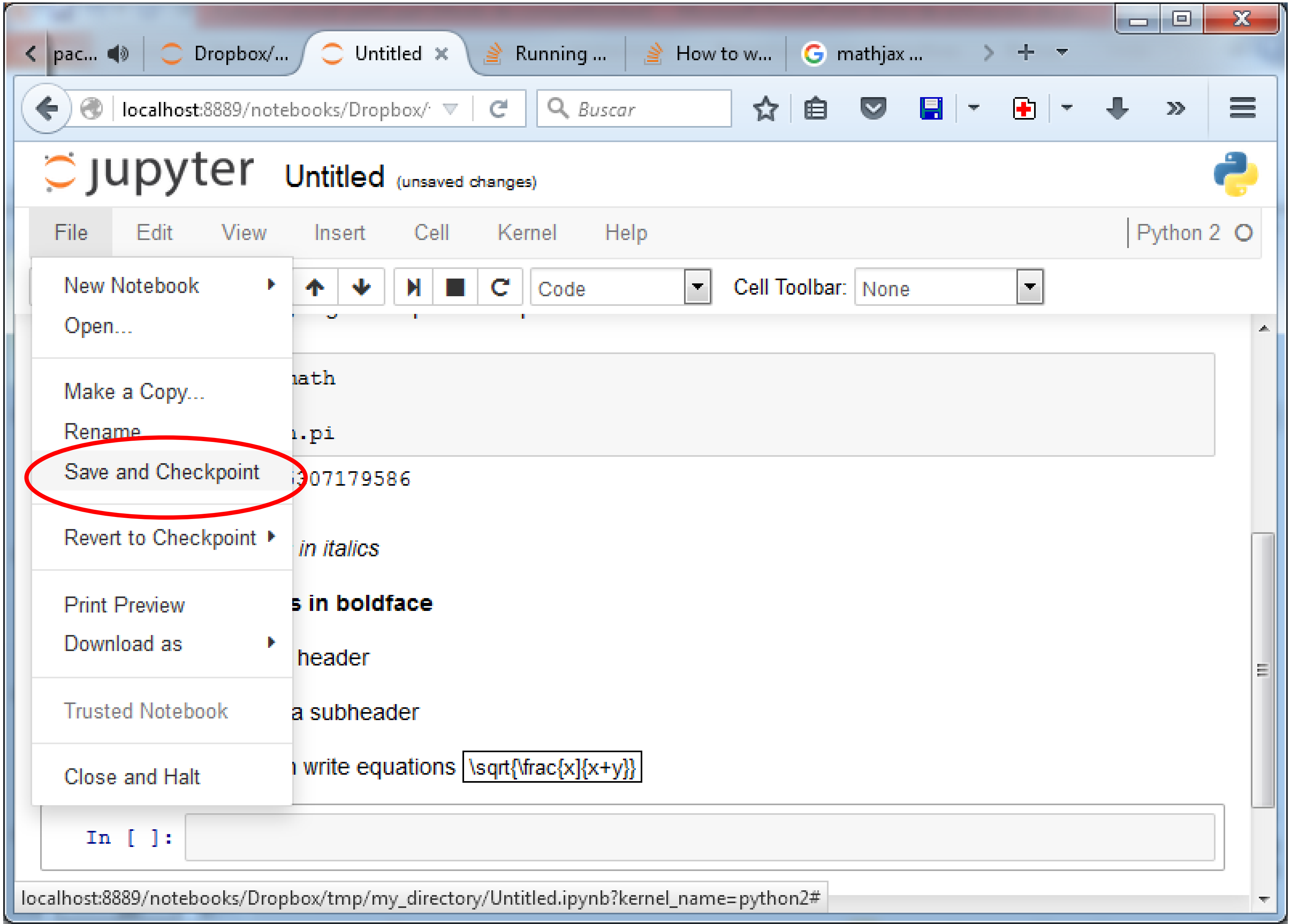
You can even embed plots



I can even write equations \sqrt{\frac{x}{x+y}}

```
%matplotlib inline
import matplotlib
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(-2*np.pi, 2*np.pi, 0.01)
y = np.sin(x)
plt.plot(x,y)
plt.show()
```

Saving the notebook



```
ath
.pi
07179586
```

in italics

s in boldface

header

a subheader

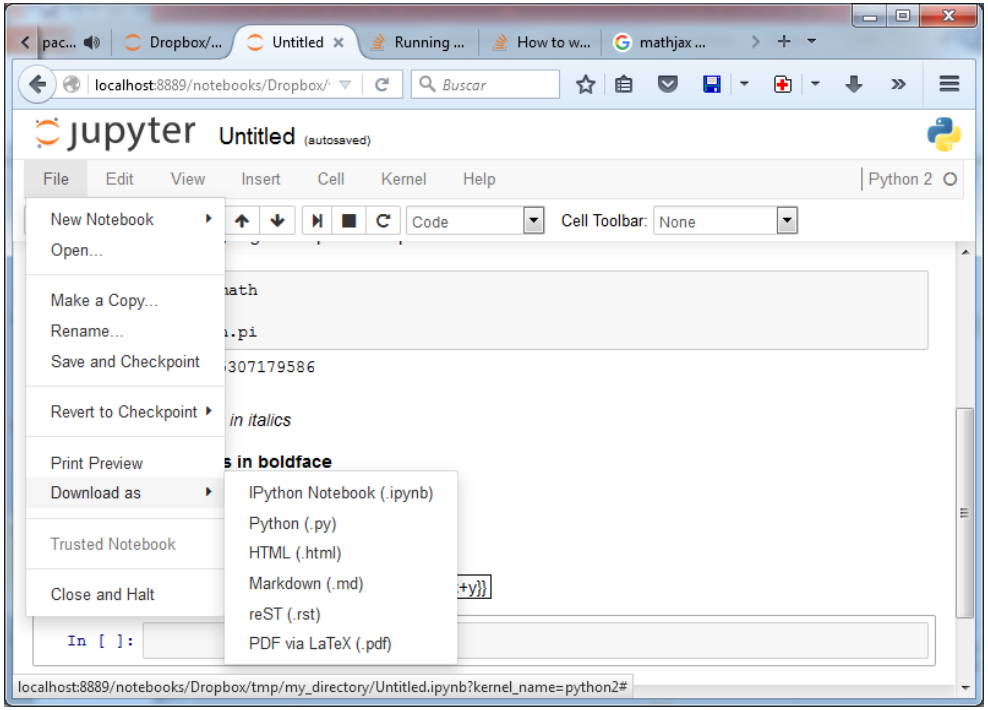
write equations \sqrt{\frac{x}{x+y}}

```
In [ ]:
```

localhost:8889/notebooks/Dropbox/tmp/my_directory/Untitled.ipynb?kernel_name=python2#

Download the notebook

- In several formats: (filename can be changed in File/Rename)
  - Python notebook: it can be loaded again as a notebook
  - Python script: this is a text file containing the sequence of Python commands. Text is also stored as comments (#)
  - html: it can be loaded later in a browser
  - pdf (it might not work because it requires LaTeX)

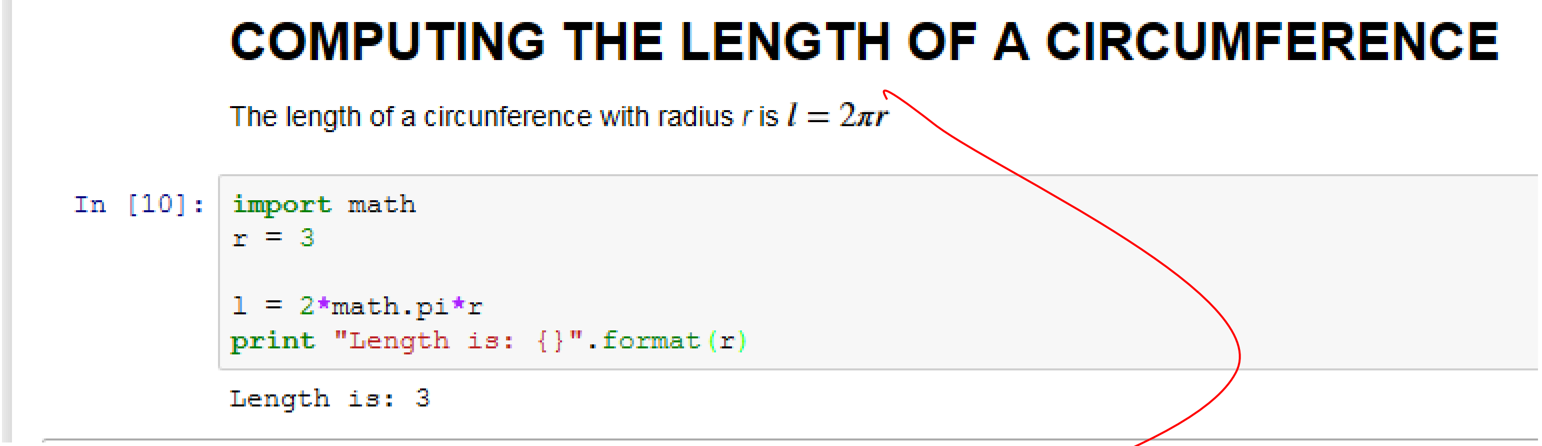


Etc.

- In order to finish the notebook:
  - File / close and halt
- Jupyter notebooks have more options but you can explore them yourselves

Exercise

- Try to get something similar to:

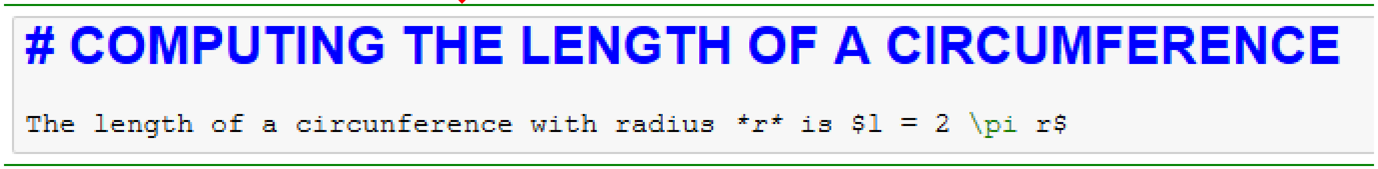


COMPUTING THE LENGTH OF A CIRCUMFERENCE

The length of a circunference with radius *r* is $l = 2\pi r$

```
In [10]: import math
         r = 3

         l = 2*math.pi*r
         print "Length is: {}".format(r)

         Length is: 3
```

HINT



```
# COMPUTING THE LENGTH OF A CIRCUMFERENCE

The length of a circunference with radius *r* is $l = 2 \pi r$
```

Topics

1. If … then … else
2. Loops:
   - While condition …
   - For …
3. Functions
4. High-level functions (map, filter, reduce)

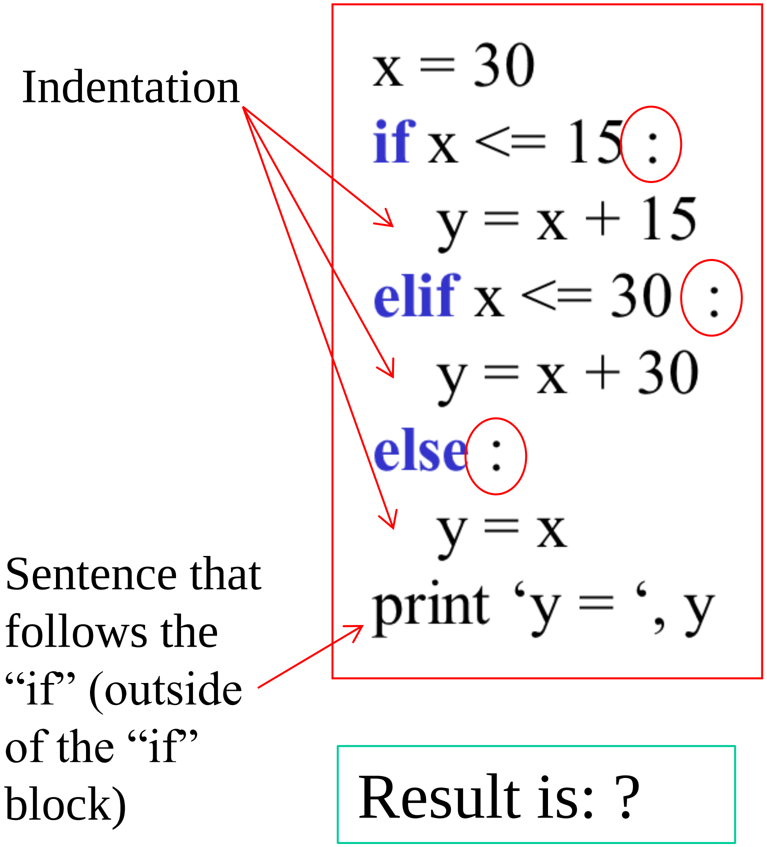
If Statements

```
if condition :
    sentence1
    sentence2
    …
next sentence
```

```
if condition :
    sentence1
    sentence2
    …
else :
   sentencea
   sentenceb
   …
next sentence
```

```
if condition :
    sentence1
    sentence2
    …
elif condition3 :
   sentencea
   sentenceb
   …
else :
   sentencex
   sentencey
   …
next sentence
```

Example:

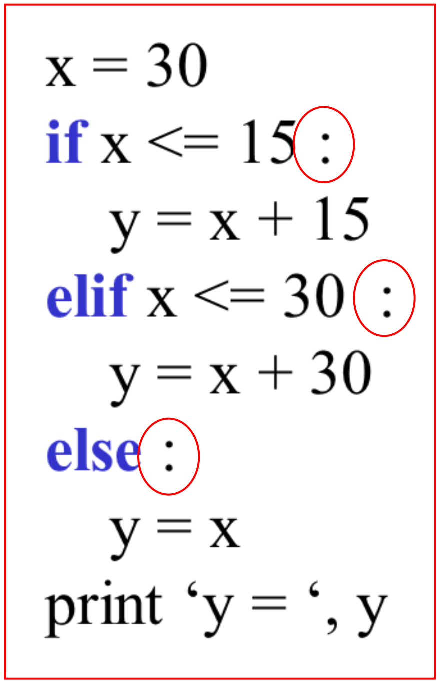


```
x = 30
if x <= 15 :
    y = x + 15
elif x <= 30 :
    y = x + 30
else :
    y = x
print 'y = ', y
```

Result is: ?

If Statements

<u>Example:</u>



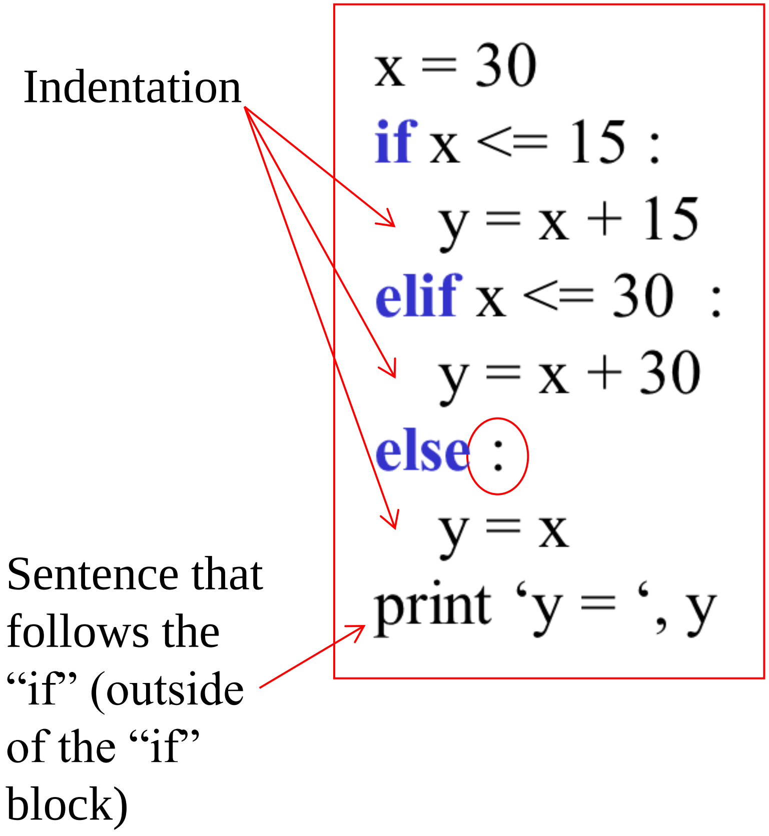
```
x = 30
if x <= 15 :
    y = x + 15
elif x <= 30 :
    y = x + 30
else :
    y = x
print ‘y = ‘, y
```

Result is: **y = 60**

Note on indentation

- Python uses indentation instead of braces (or curly brackets) to determine the scope of expressions
- All lines must be indented the same amount to be part of the scope (or indented more if part of an inner scope)
- This forces the programmer to use proper indentation since the indenting is part of the program!
- Indentation made of four spaces is recommended

Example:



```
x = 30
if x <= 15 :
    y = x + 15
elif x <= 30  :
    y = x + 30
else :
    y = x
print ‘y = ‘, y
```

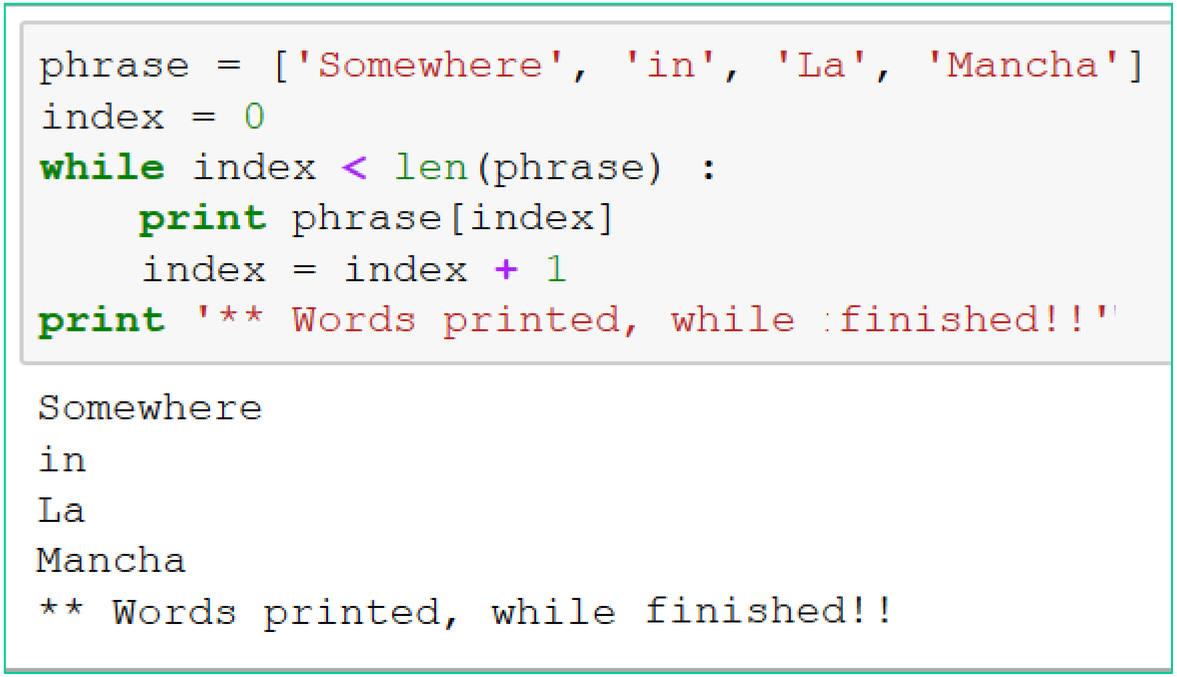
Indentation

Sentence that follows the “if” (outside of the “if” block)

While Loops

While *condition* is true, execute sentences in the *while block* (*sentence1*, *sentence2*, …)

while condition :
    sentence1
    sentence2
    …
Next sentence
(outside while block)



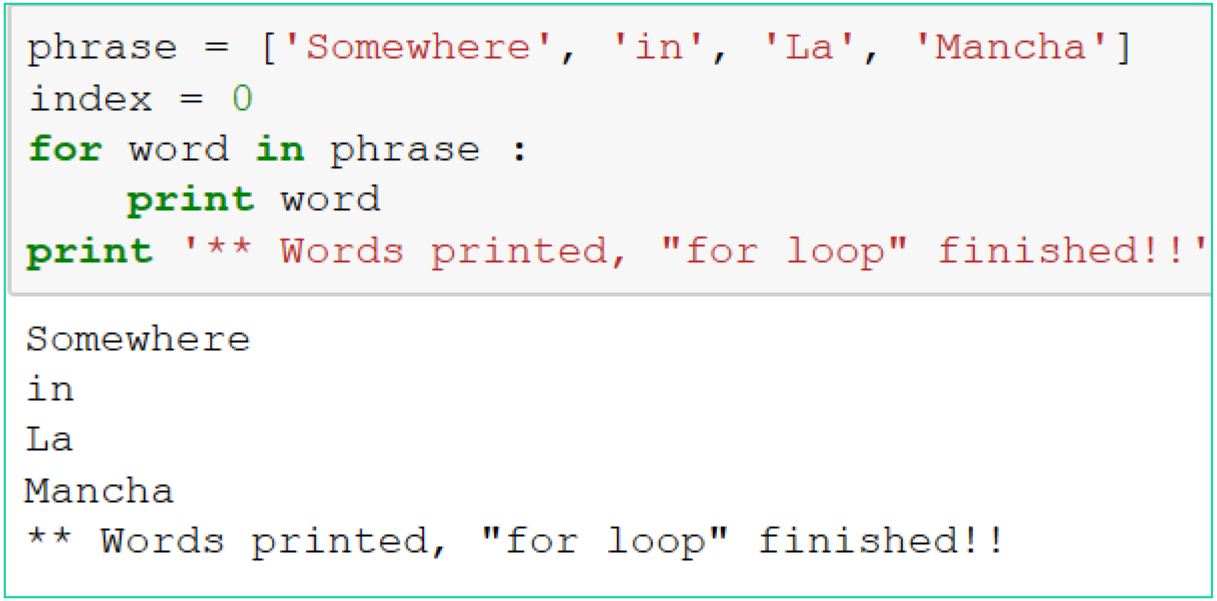
```
phrase = ['Somewhere', 'in', 'La', 'Mancha']
index = 0
while index < len(phrase) :
    print phrase[index]
    index = index + 1
print '** Words printed, while :finished!!'
```

```
Somewhere
in
La
Mancha
** Words printed, while finished!!
```

For Loops

variable takes succesive values in the *sequence*

```
for variable in sequence :
    sentence1
    sentence2
    …
Next sentence (outside for block)
```



```
phrase = ['Somewhere', 'in', 'La', 'Mancha']
index = 0
for word in phrase :
    print word
print '** Words printed, "for loop" finished!!'
```

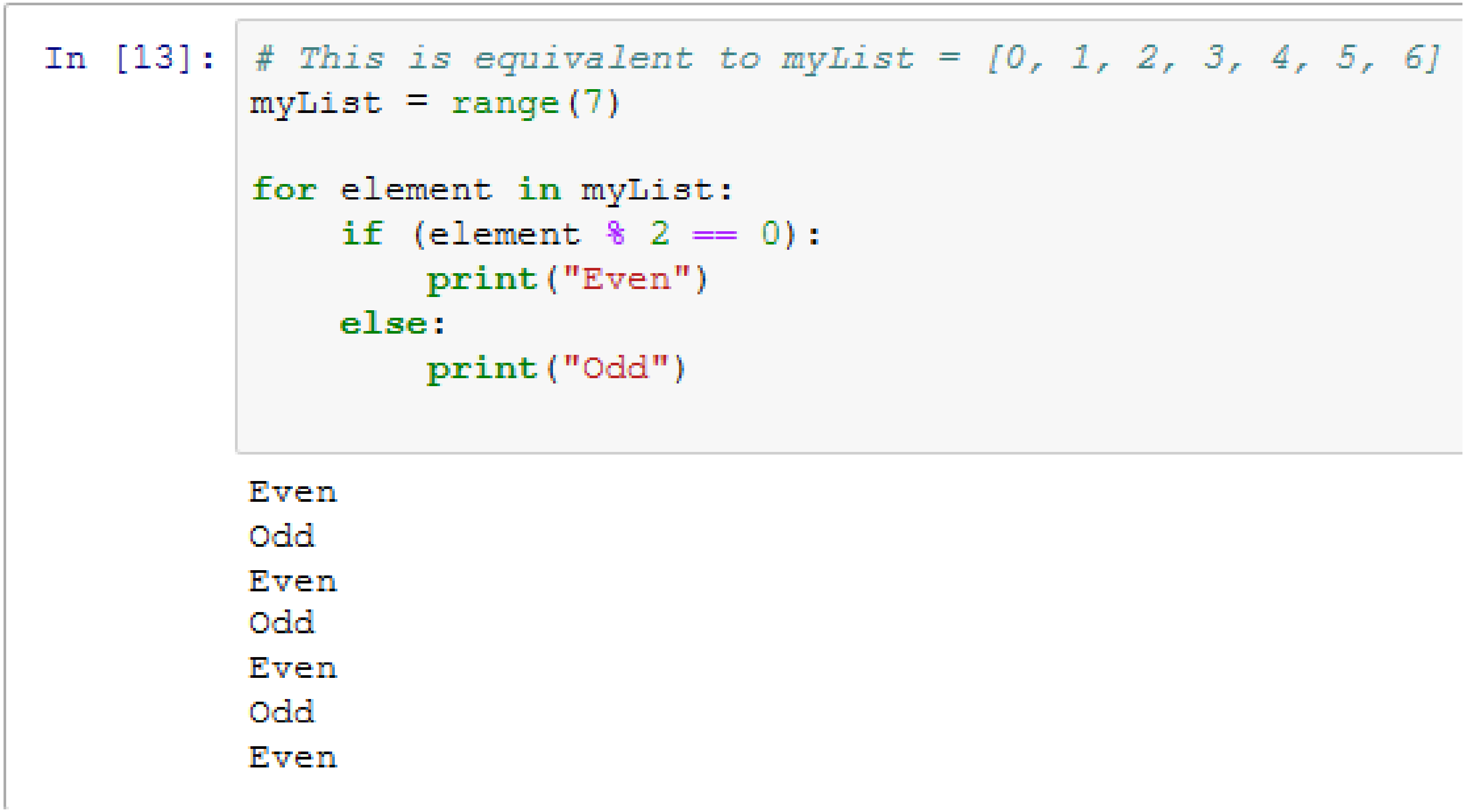
```
Somewhere
in
La
Mancha
** Words printed, "for loop" finished!!
```

Exercise

- Create a list of numbers [0, 1, 3, 4, 5, 6]
- Iterate over this list by using a for loop
  - For each element in the list, print “even” if the number is even and “odd” if the number is odd
- Reminder: a number *x* is even if the remainder of the division by 2 is zero. That is: (x % 2 == 0)
- Once you are done, try with another list:

  [1, 7, 3, 2, 0]

Solution



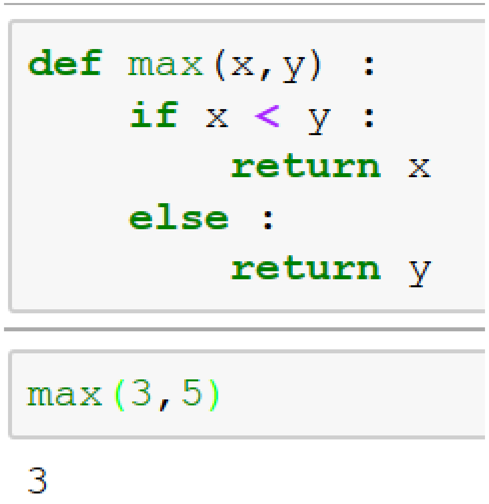
```
# This is equivalent to myList = [0, 1, 2, 3, 4, 5, 6]
myList = range(7)

for element in myList:
    if (element % 2 == 0):
        print("Even")
    else:
        print("Odd")
```

```
Even
Odd
Even
Odd
Even
Odd
Even
```

Function Definition

“return x” returns the value and ends the function exectution

```
def functionName(argument1, argument2, …) :
    sentence1
    sentence2
    …
```



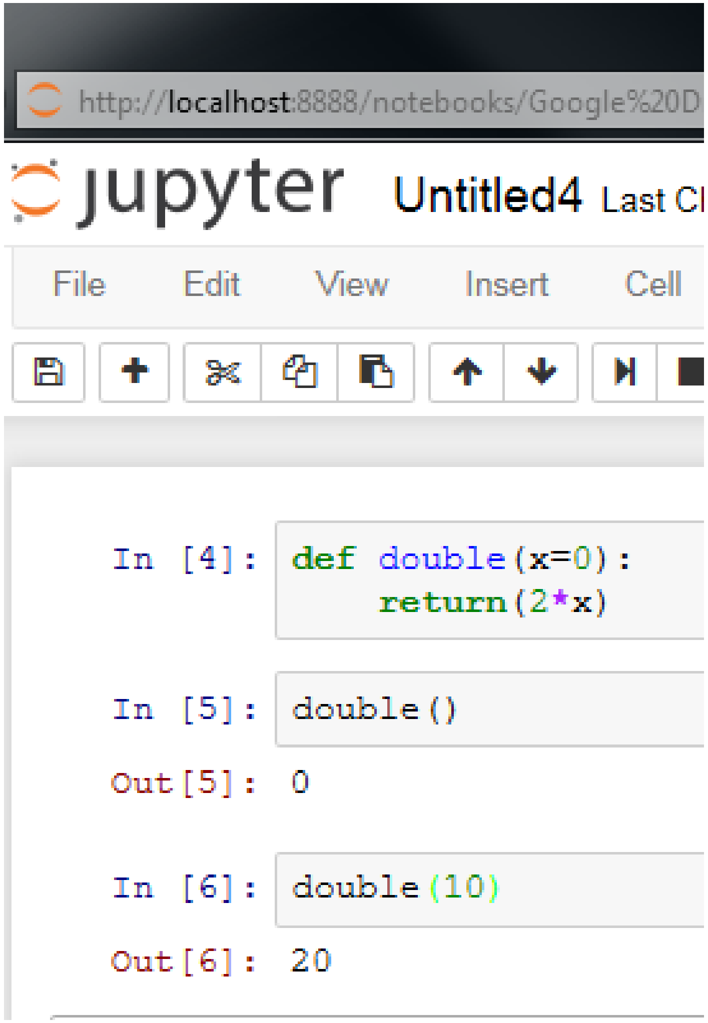
```
def max(x,y) :
    if x < y :
        return x
    else :
        return y
```

```
max(3,5)
```

```
3
```

Parameters: Defaults

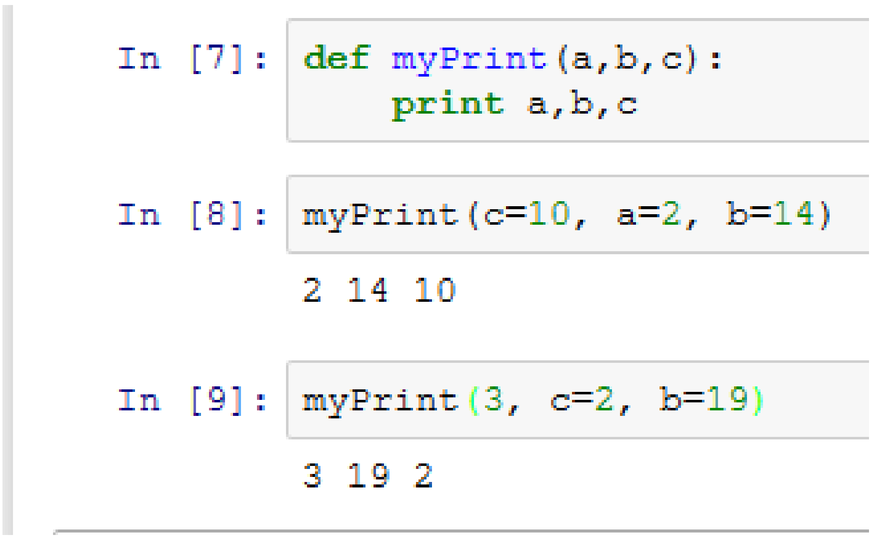
- Parameters can be assigned default values
- They are overridden if a parameter is given for them



```
In [4]: def double(x=0):
            return(2*x)

In [5]: double()

Out[5]: 0

In [6]: double(10)

Out[6]: 20
```

Parameters: Named

- Call by name
- Any positional arguments must come before named ones in a call

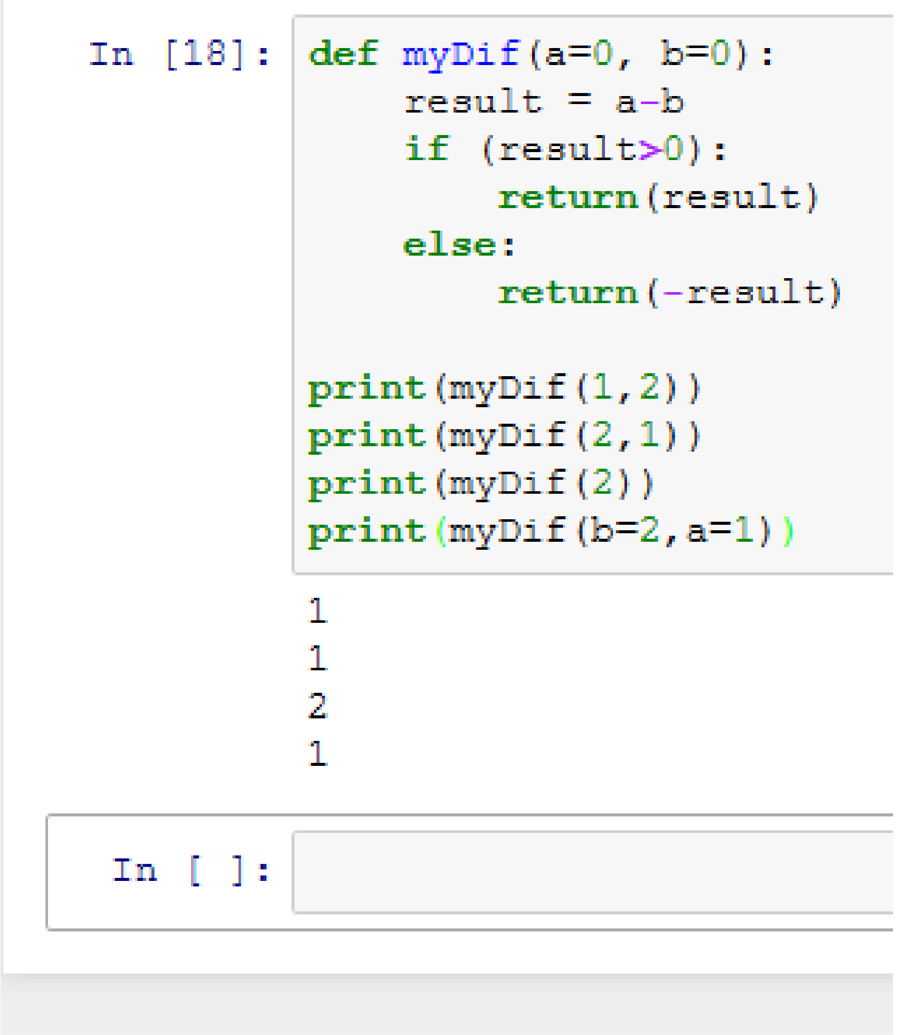


```
In [7]: def myPrint(a,b,c):
            print a,b,c

In [8]: myPrint(c=10, a=2, b=14)

        2 14 10

In [9]: myPrint(3, c=2, b=19)

        3 19 2
```

Exercise

- Define a function *myDif* that returns:
  - If (a-b)>0 then (a-b)
  - Otherwise b-a
- Both *a* and *b* should have default values of 0
- You need to use *if*
- Try the following function calls and see what happens:
  - myDif(1,2)
  - myDif(2,1)
  - myDif(2)
  - myDif(b=2,a=1)

Solution



```
def myDif(a=0, b=0):
    result = a-b
    if (result>0):
        return(result)
    else:
        return(-result)

print(myDif(1,2))
print(myDif(2,1))
print(myDif(2))
print(myDif(b=2,a=1))
```

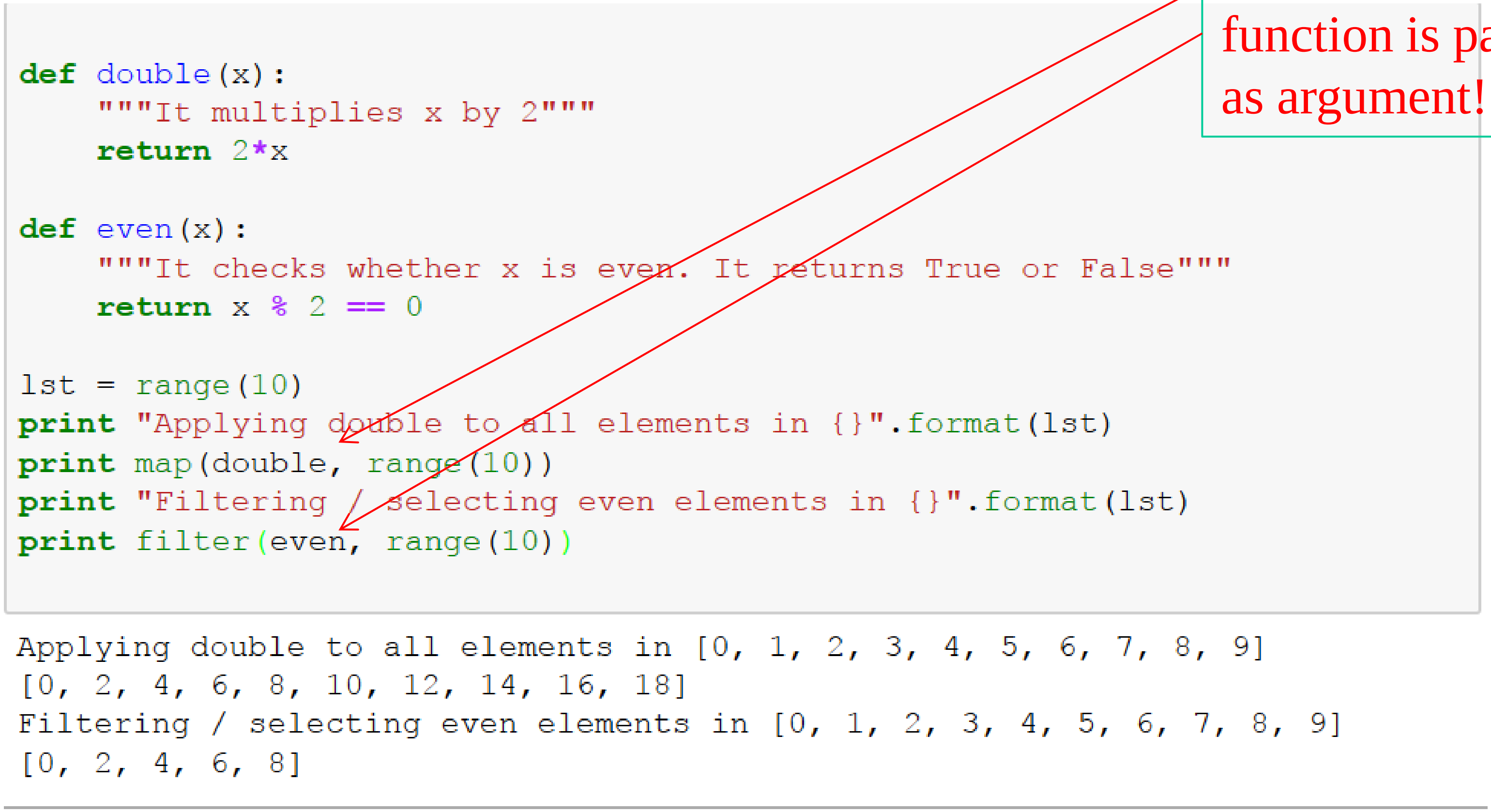
```
1
1
2
1
```

Higher-Order Functions

map(func,seq) – for all i, applies func(seq[i]) and returns the corresponding sequence of the calculated results.

filter(boolfunc,seq) – returns a sequence containing all those items in seq for which boolfunc is True.

Notice that a function is passed as argument!!



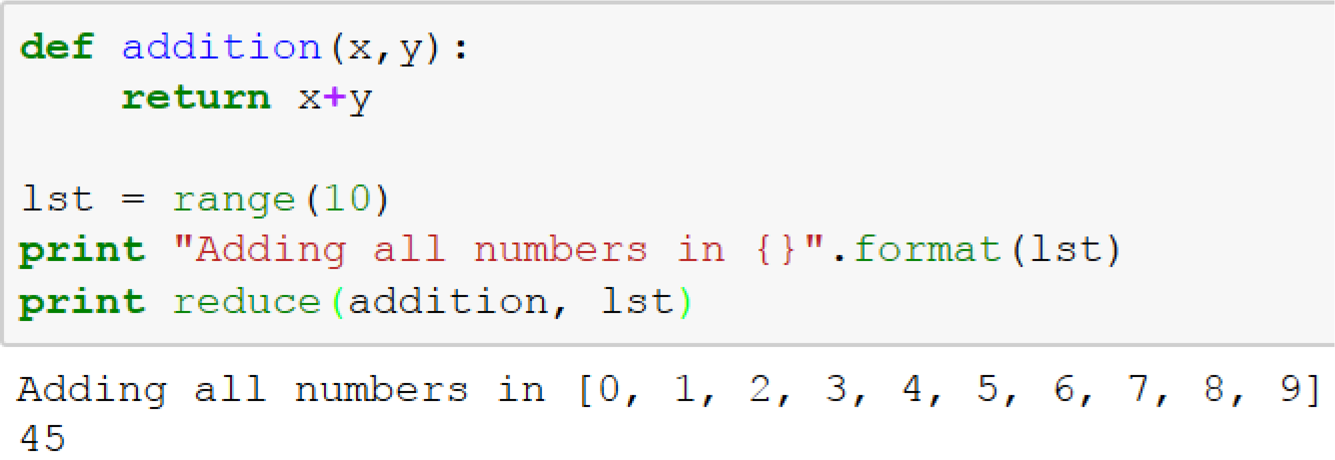
```
def double(x):
    """It multiplies x by 2"""
    return 2*x

def even(x):
    """It checks whether x is even. It returns True or False"""
    return x % 2 == 0

lst = range(10)
print "Applying double to all elements in {}".format(lst)
print map(double, range(10))
print "Filtering / selecting even elements in {}".format(lst)
print filter(even, range(10))
```

```
Applying double to all elements in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
Filtering / selecting even elements in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[0, 2, 4, 6, 8]
```

Higher-Order Functions

reduce(func,seq) – applies func to the items of seq, from left to right, two-at-time, to reduce the seq to a single value.

Example: reduce(addition, [1,2,3,4]) = 1+2+3+4 = 10



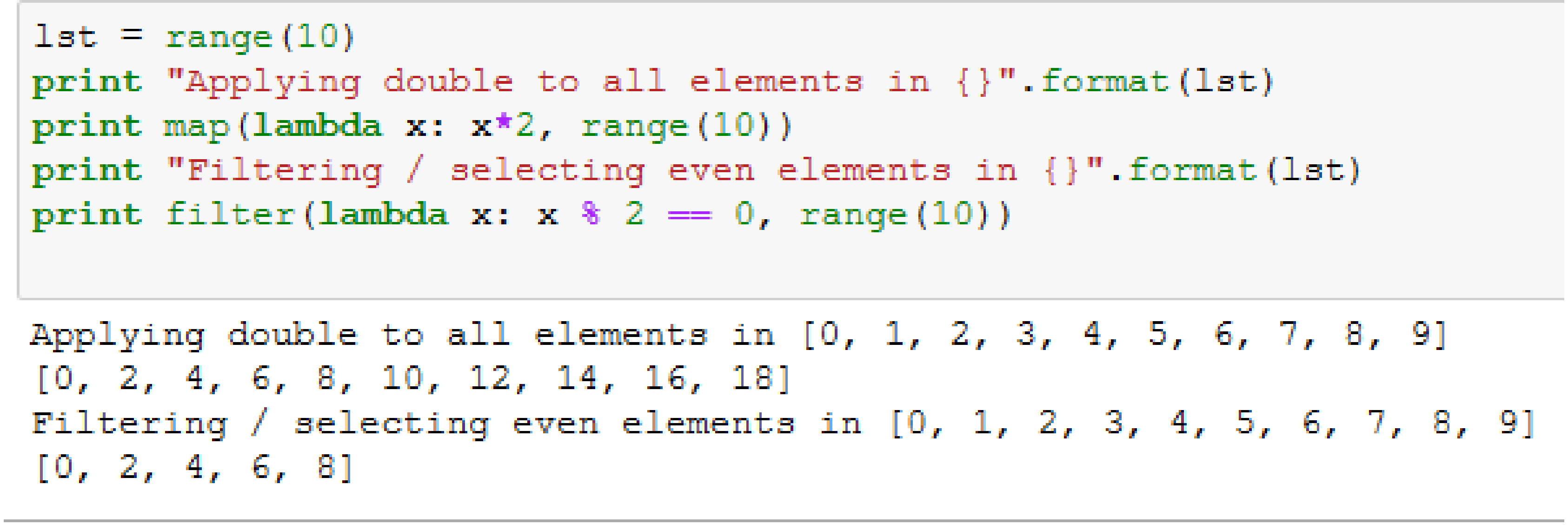
```
def addition(x,y):
    return x+y

lst = range(10)
print "Adding all numbers in {}".format(lst)
print reduce(addition, lst)
```

```
Adding all numbers in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
45
```

Higher-Order Functions with lambda functions

map(func,seq) – for all i, applies func(seq[i]) and returns the corresponding sequence of the calculated results.

filter(boolfunc,seq) – returns a sequence containing all those items in seq for which boolfunc is True.

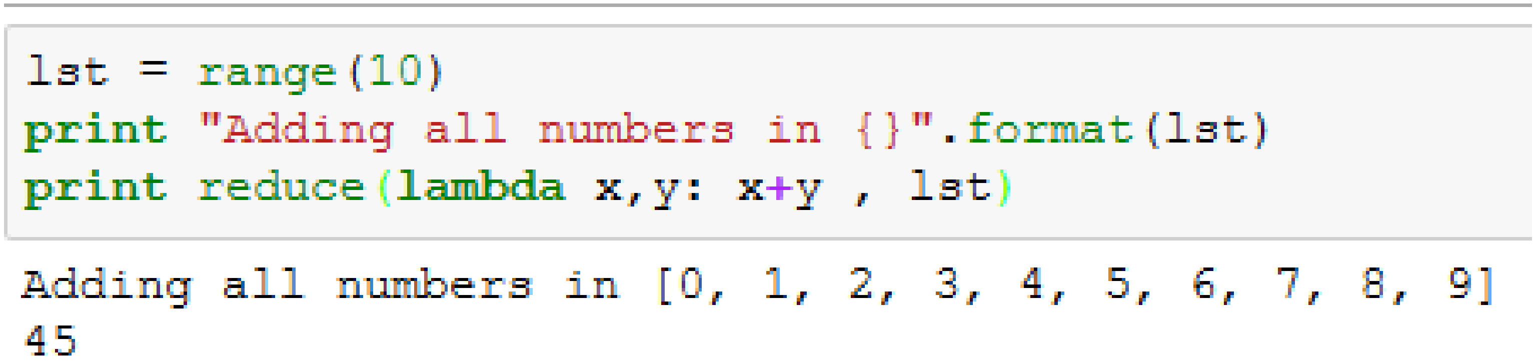


```
lst = range(10)
print "Applying double to all elements in {}".format(lst)
print map(lambda x: x*2, range(10))
print "Filtering / selecting even elements in {}".format(lst)
print filter(lambda x: x % 2 == 0, range(10))
```

```
Applying double to all elements in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
Filtering / selecting even elements in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[0, 2, 4, 6, 8]
```

Higher-Order Functions with lambda functions

reduce(func,seq) – applies func to the items of seq, from left to right, two-at-time, to reduce the seq to a single value.



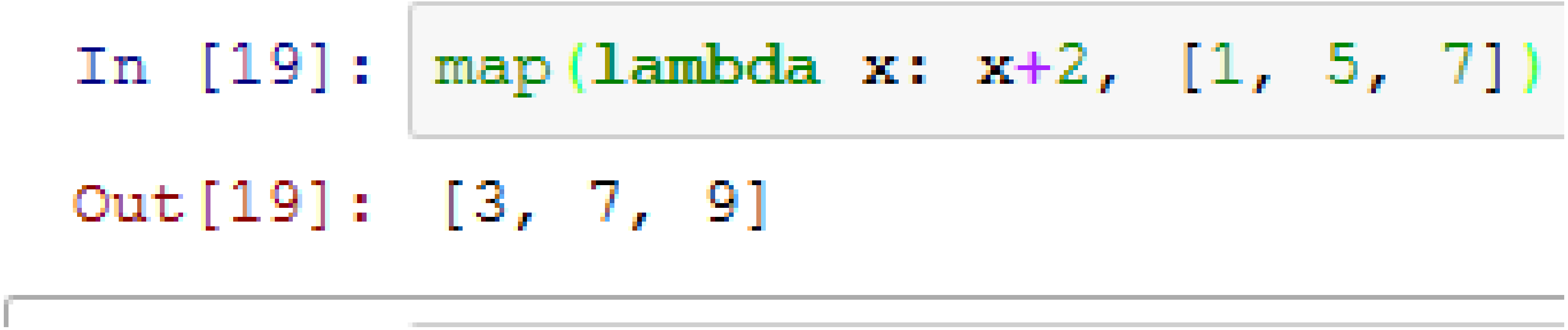
```
lst = range(10)
print "Adding all numbers in {}".format(lst)
print reduce(lambda x,y: x+y , lst)
```

```
Adding all numbers in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
45
```

Exercise

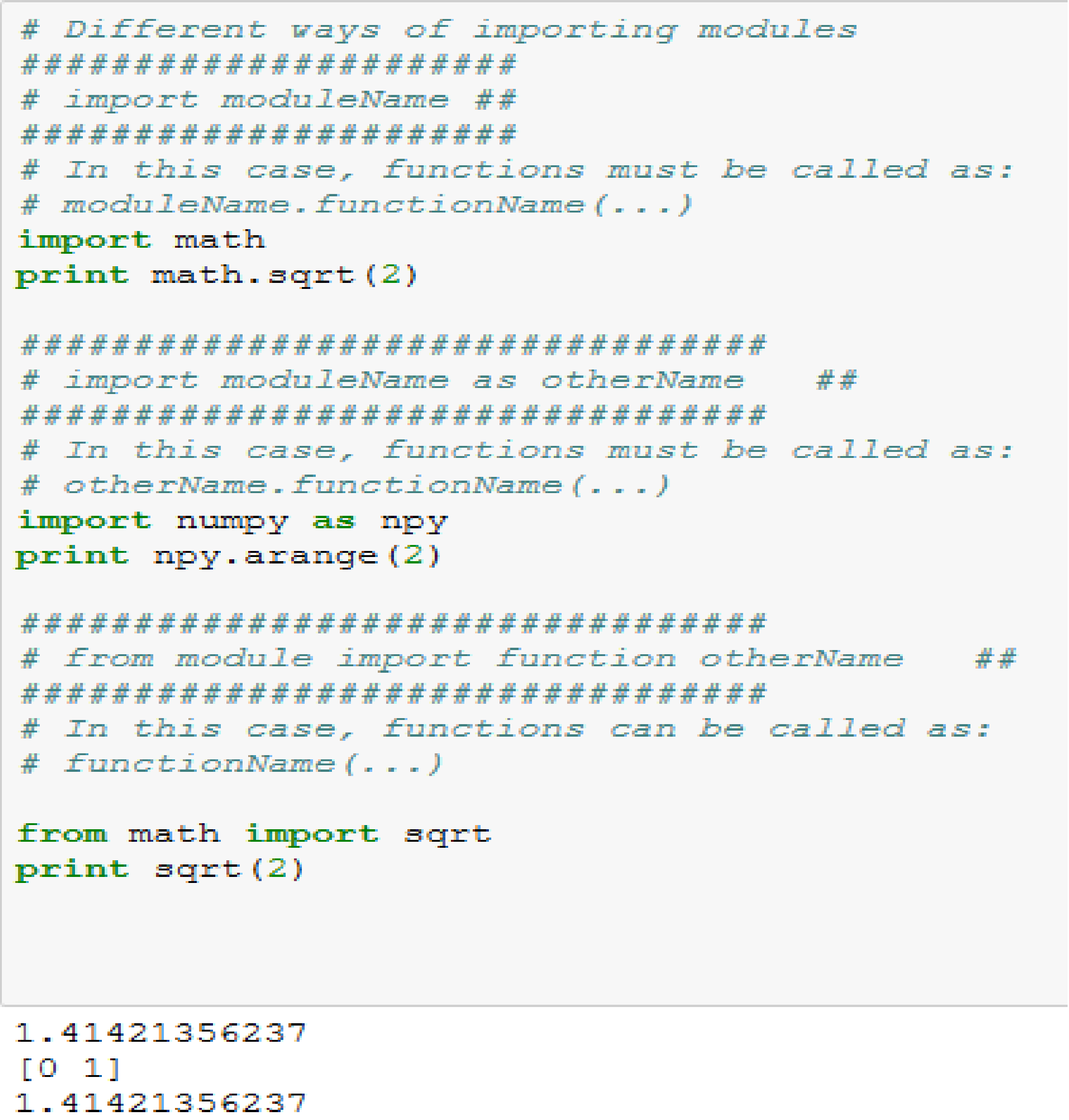
- Use a higher-order function (*map*) with lambda-function that adds 2 to every number in a list
- Apply it to this list: [1, 5, 7]

Solution



```
In [19]:  map(lambda x: x+2, [1, 5, 7])

Out[19]:  [3, 7, 9]
```

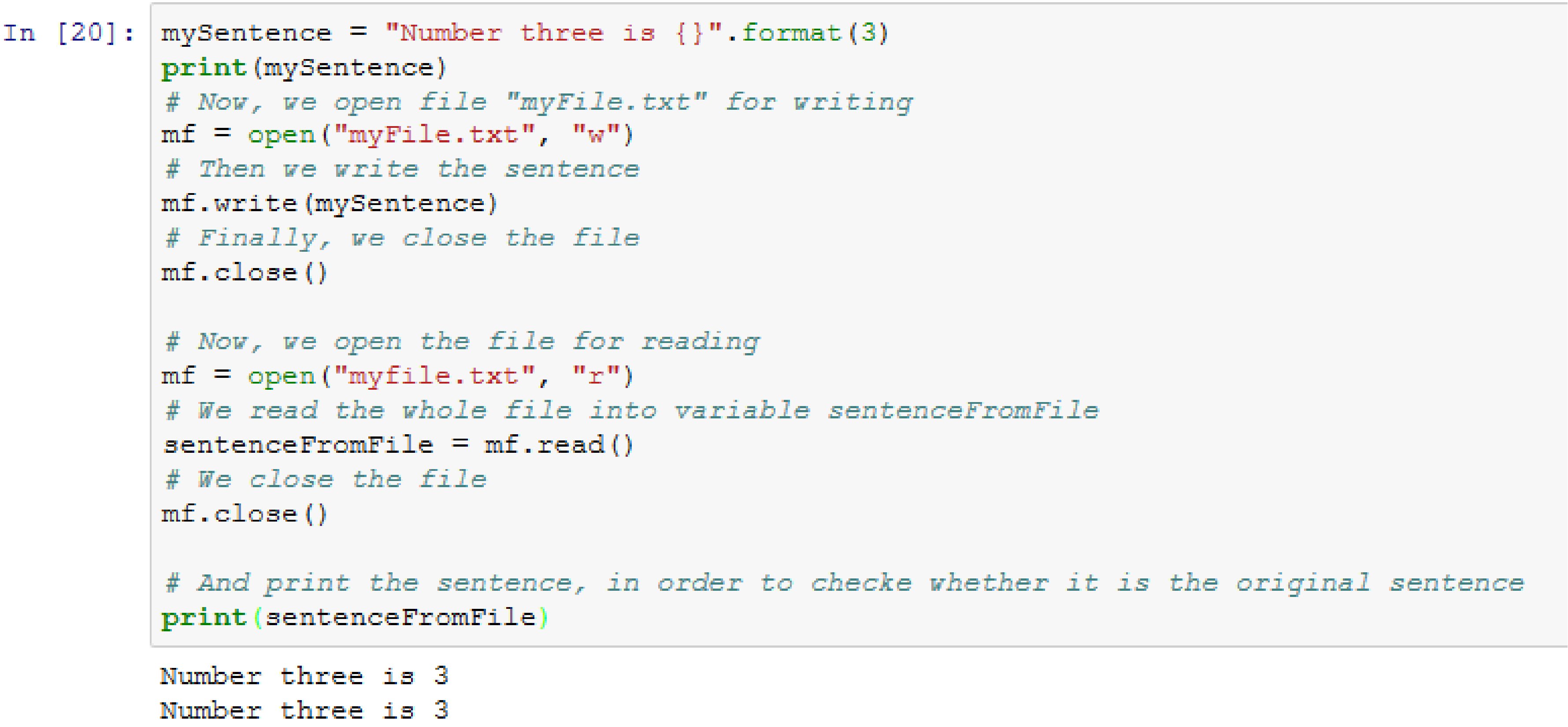
Modules: Imports



```
# Different ways of importing modules
######################
# import moduleName ##
######################
# In this case, functions must be called as:
# moduleName.functionName(...)
import math
print math.sqrt(2)

#################################
# import moduleName as otherName   ##
#################################
# In this case, functions must be called as:
# otherName.functionName(...)
import numpy as npy
print npy.arange(2)

#################################
# from module import function otherName   ##
#################################
# In this case, functions can be called as:
# functionName(...)

from math import sqrt
print sqrt(2)
```

```
1.41421356237
[0 1]
1.41421356237
```

Writing and reading files



In [20]:

```
mySentence = "Number three is {}".format(3)
print(mySentence)
# Now, we open file "myFile.txt" for writing
mf = open("myFile.txt", "w")
# Then we write the sentence
mf.write(mySentence)
# Finally, we close the file
mf.close()

# Now, we open the file for reading
mf = open("myfile.txt", "r")
# We read the whole file into variable sentenceFromFile
sentenceFromFile = mf.read()
# We close the file
mf.close()

# And print the sentence, in order to checke whether it is the original sentence
print(sentenceFromFile)
```

```
Number three is 3
Number three is 3
```

Files: Input

inflobj = open(‘data’, ‘r’)	Open the file ‘data’ for input.
S = inflobj.read()	Read whole file into one String
S = inflobj.read(N)	Reads N bytes (N >= 1)
L = inflobj.readlines()	Returns a list of line strings

Files: Output

outflobj = open(‘data’, ‘w’)	Open the file ‘data’ for writing
outflobj.write(S)	Writes the string S to file
outflobj.writelines(L)	Writes each of the strings in list L to file
outflobj.close()	Closes the file

EXTRA MATERIAL: LOOPS AND LIST COMPREHENSIONS

Loop Control Statements

break	Jumps out of the closest enclosing loop (or while)
continue	Jumps to the top of the closest enclosing loop (or while)
pass	Does nothing, empty statement placeholder

The Loop Else Clause

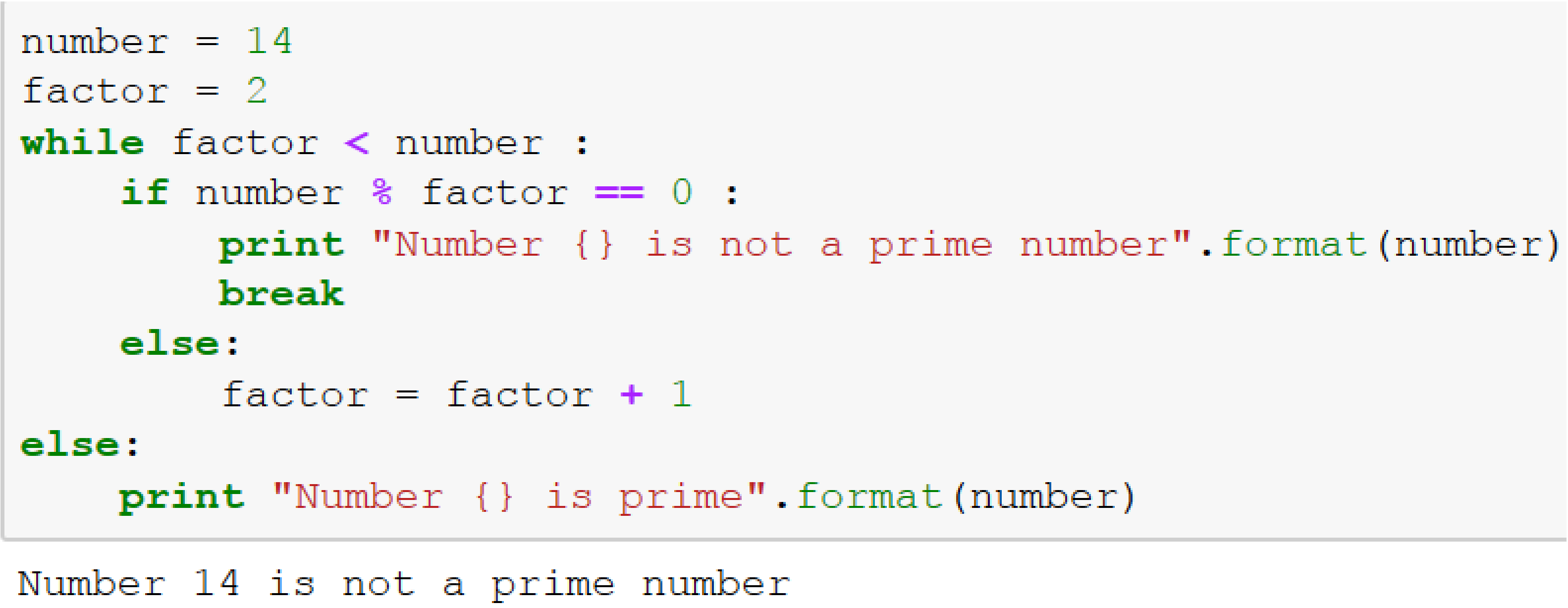
- The optional **else** clause runs only if the loop exits normally (not by break)

```
while condition :
    sentence1
    sentence2
    …
else:
    sentencea
    sentenceb
Next sentence
(outside while block)
```

```
for variable in sequence :
    sentence1
    sentence2
    …
else:
    sentencea
    sentenceb
Next sentence (outside
for block)
```

The Loop Else Clause

- The optional **else** clause runs only if the loop exits normally (not by break)

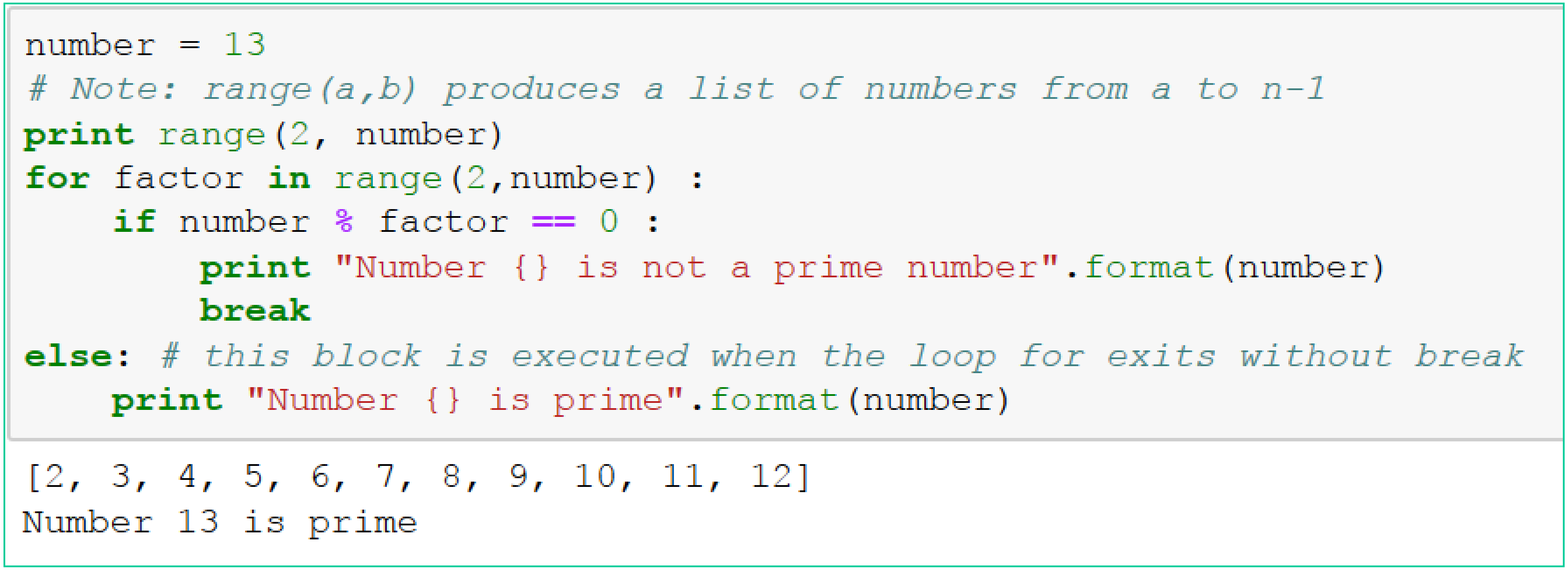


```
number = 14
factor = 2
while factor < number :
    if number % factor == 0 :
        print "Number {} is not a prime number".format(number)
        break
    else:
        factor = factor + 1
else:
    print "Number {} is prime".format(number)
```

```
Number 14 is not a prime number
```

The Loop Else Clause

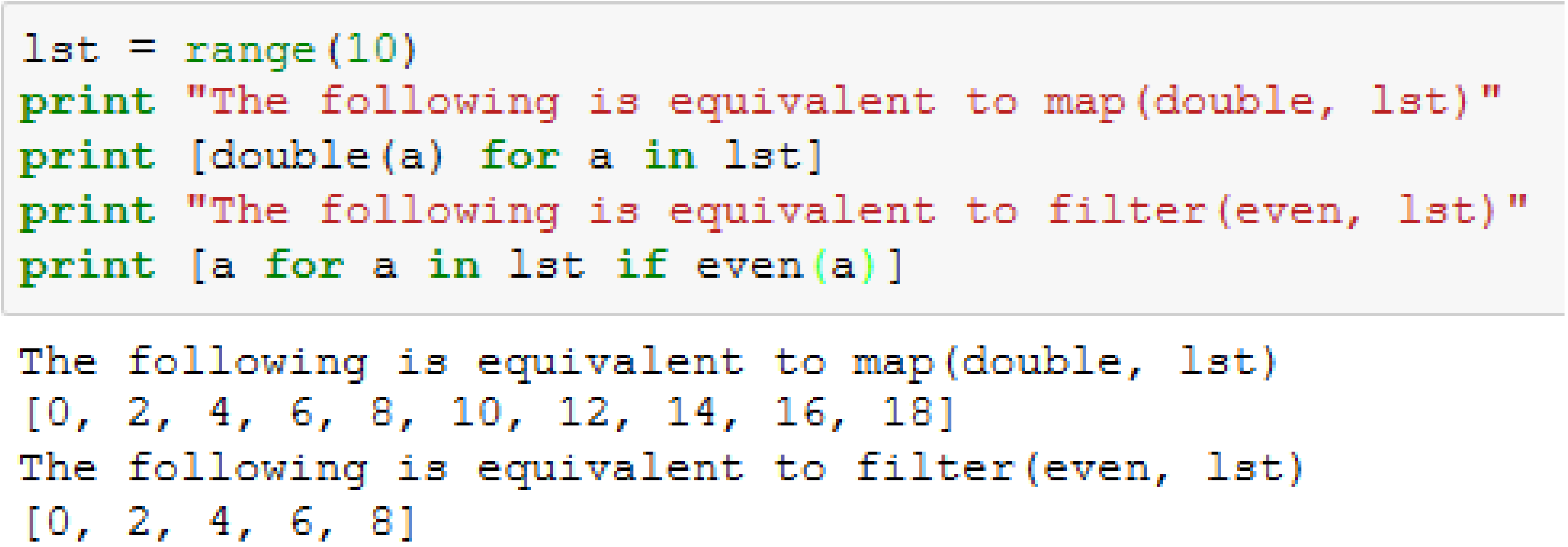
- The optional **else** clause runs only if the loop exits normally (not by break)



```
number = 13
# Note: range(a,b) produces a list of numbers from a to n-1
print range(2, number)
for factor in range(2,number) :
    if number % factor == 0 :
        print "Number {} is not a prime number".format(number)
        break
else: # this block is executed when the loop for exits without break
    print "Number {} is prime".format(number)
```

```
[2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
Number 13 is prime
```

Higher-Order Functions with list comprehensions

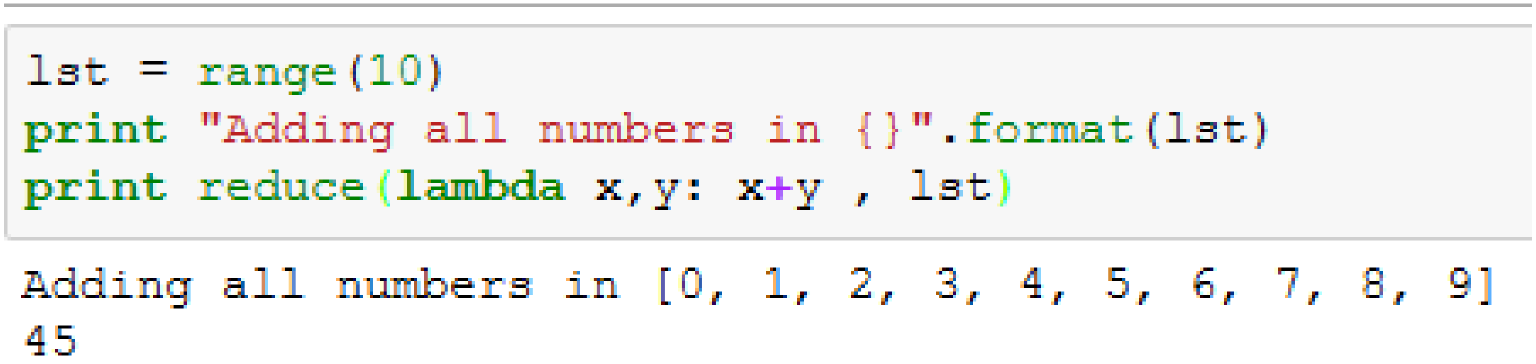


```
lst = range(10)
print "The following is equivalent to map(double, lst)"
print [double(a) for a in lst]
print "The following is equivalent to filter(even, lst)"
print [a for a in lst if even(a)]
```

```
The following is equivalent to map(double, lst)
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
The following is equivalent to filter(even, lst)
[0, 2, 4, 6, 8]
```

Higher-Order Functions with list comprehensions

reduce(func,seq) – applies func to the items of seq, from left to right, two-at-time, to reduce the seq to a single value.



```python
lst = range(10)
print "Adding all numbers in {}".format(lst)
print reduce(lambda x,y: x+y , lst)
```

```
Adding all numbers in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
45
```

Functions are first class objects

- Can be assigned to a variable

```
x = max
```

- Can be passed as a parameter
- Can be returned from a function
- Functions are treated like any other variable in Python, the **def** statement simply assigns a function to a variable

Anonymous Functions

- A lambda expression returns a function object
- The body can only be a simple expression, not complex statements

```
>>> f = lambda x,y : x + y
>>> f(2,3)
5
```