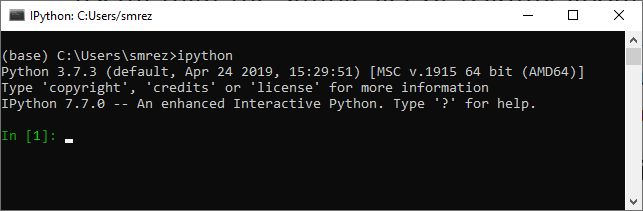
**Dear Team Members!**

I hope you are having a wonderful day!

The idea of this part is to make sure we cover most of the book’s commands. So, let’s do it!

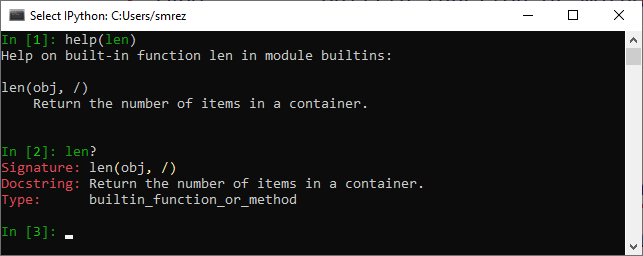
**Starting the program and getting help**

ipython



help(len)

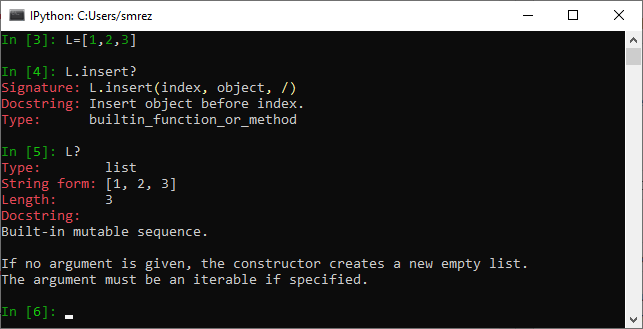
len?



L=[1,2,3]

L.insert?

L?



**Defining our first function:**

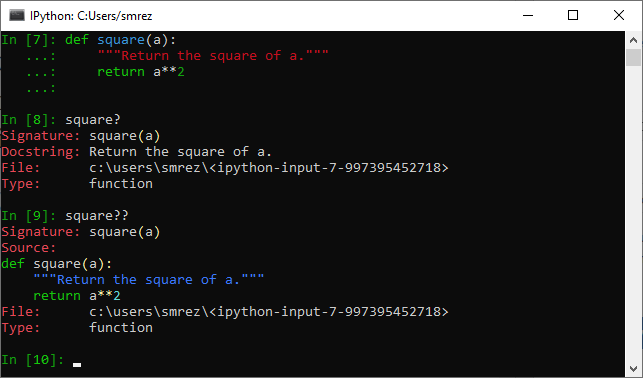
def square(a):

"""Return the square of a."""

return a\*\*2

Then getting help by typing the name of the function and ?.

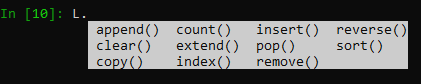
And also getting the code of the function by typing the name of the function and ??.



**Using the Tab key:**

Please type L. and then press the Tab key:

L.<TAB>



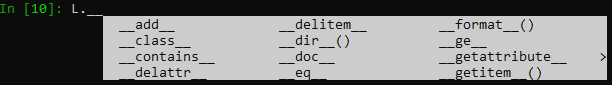
And now try the followings:

L.c<TAB>

L.cou<TAB>

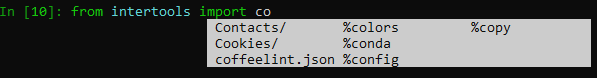
And the followings are the **private** and **special** methods:

L.\_<TAB>

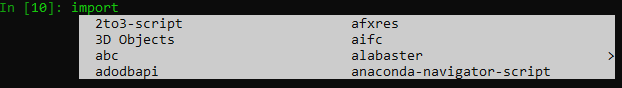


**Importing objects from packages:**

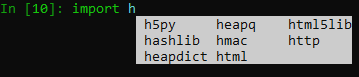
from intertools import co<TAB>



import <TAB>

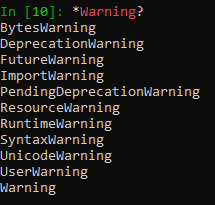


import h<TAB>



**Wildcard matching**

\*Warning?



str.\*find\*?



**Command History Shortcuts:**

Now, please first press **Ctrl + R** and then type in and press **up arrow key** or press the **Ctrl + R** again and see the results:



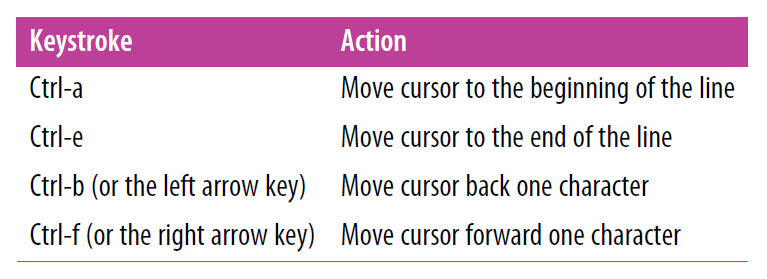




Navigation Shortcuts

Let’s try Navigation Shortcuts on the following command:

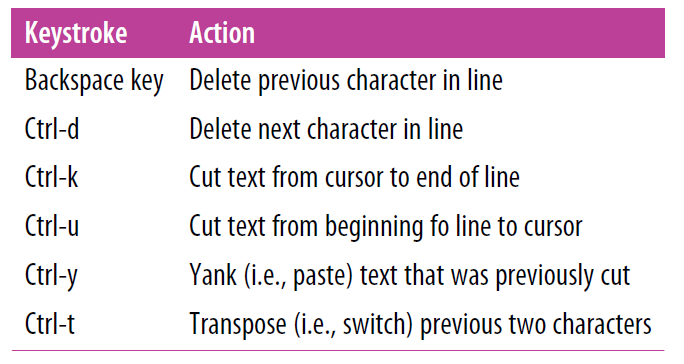
**from itertools import compress**



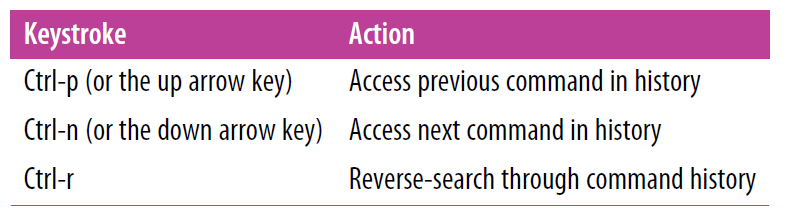
Text Entry Shortcuts

Let’s try Text Entry Shortcuts on the following command:

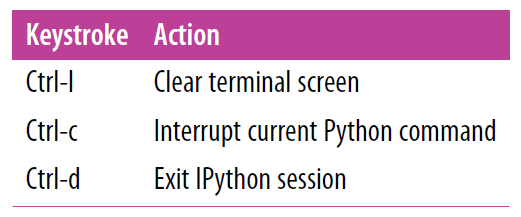
**from itertools import compress**



Command History Shortcuts



Miscellaneous Shortcuts



Try the following commands and see the result.

Ctrl + l

The following is to cancel running the current command, if you are in the middle of the process.

Ctrl + c

Ctrl + d

Sorry for the last command… Please type **ipython** to get back to the environment again.

**IPython Magic Commands:**

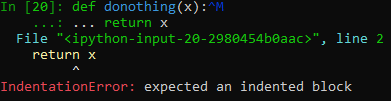
Please copy the following lines:

**In [4]: def donothing(x):**

**...: return x**

And now paste them into command line… the result could be something like the following:

**Sometimes works fine, sometimes does not!!!**

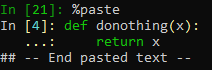


But now, please copy the following code again:

**In [4]: def donothing(x):**

**...: return x**

And then type %paste and press the enter. It takes care of everything and executes the code, considering the command prompt or extra unwanted characters.



And now test the function as follows:

donothing(100)



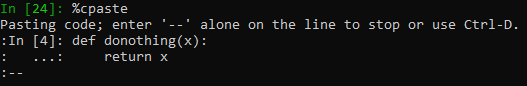
Yes! It works!

Let’s use %cpaste which is an interactive multi-line version of %paste. You need to type -- to finish the job.

Type:

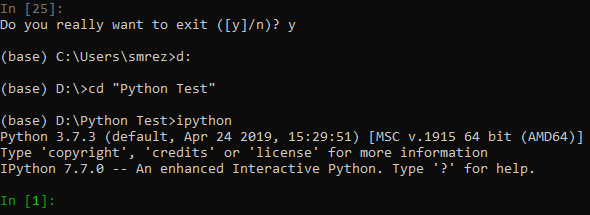
%cpaste

Then press Ctrl + V to paste and finally press -- to finish the paste process.



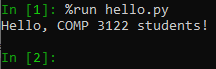
**Running External Code: %run**

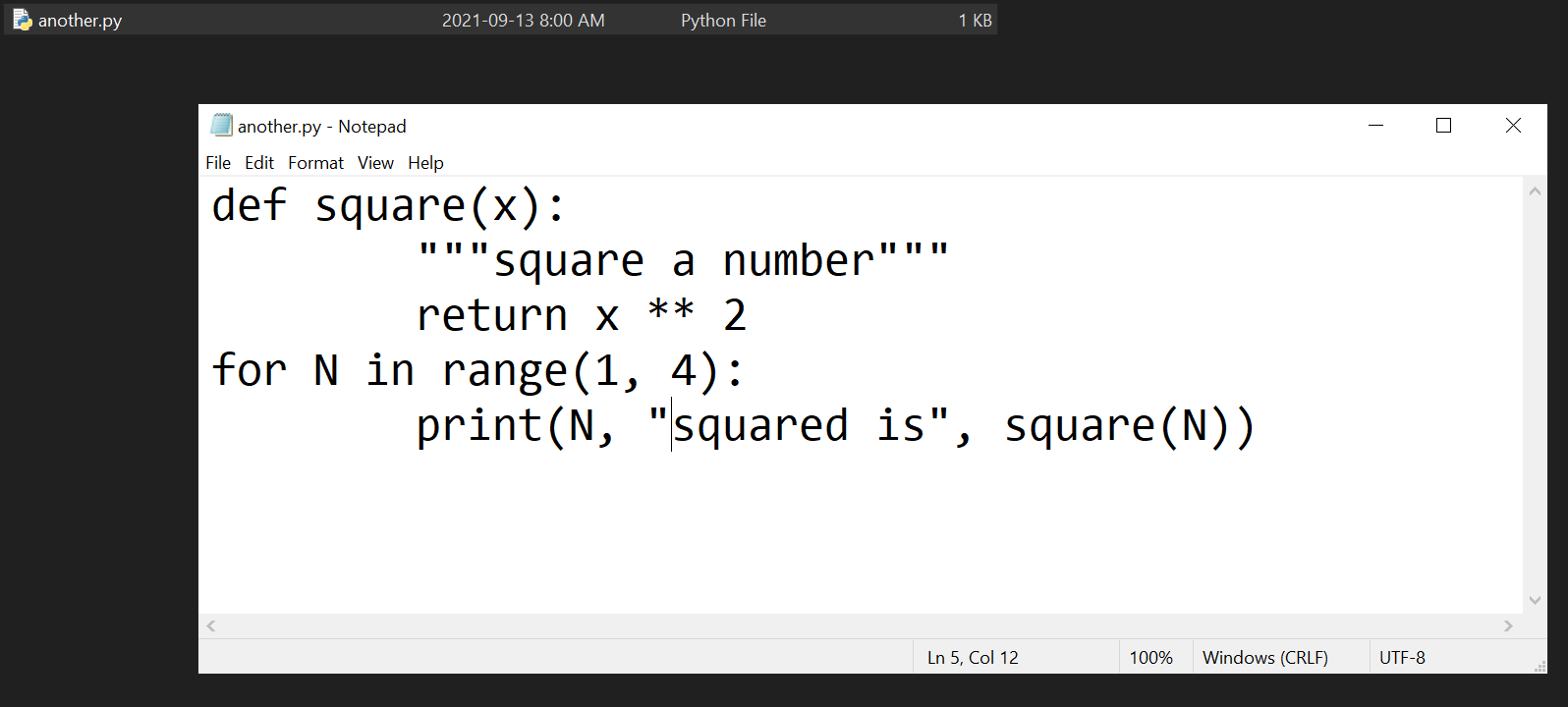
Now, please copy the hello.py file that you have in your week 1 contents into your current folder that you have run ipython in. If you are not sure about the current path, or if you want to change the current path, you can simply quit the ipyhton and then go to the folder that you want (the one that you have copied hello.py inside), then type ipython and then use the following command:



And now type the following:

%run hello.py





def square(x):

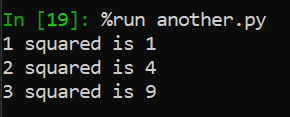
"""square a number"""

return x \*\* 2

for N in range(1, 4):

print(N, "squared is", square(N))

%run another.py



**Timing Code Execution: %timeit**

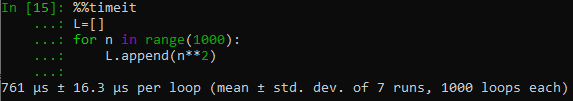
Please run the following command:

(Please be patient! It may take a few seconds….)

%timeit L = [n \*\* 2 for n in range(1000)]



And now let’s test %%timeit as follows:



Attention, please! Intentionally I have not copied the code, in order to let you face an error and fix it yourself 😊

YESSSSSS!!!! You found it!

**Help on Magic Functions: ?, %magic, and %lsmagic**

To read the documentation of the %timeit magic:

%timeit?

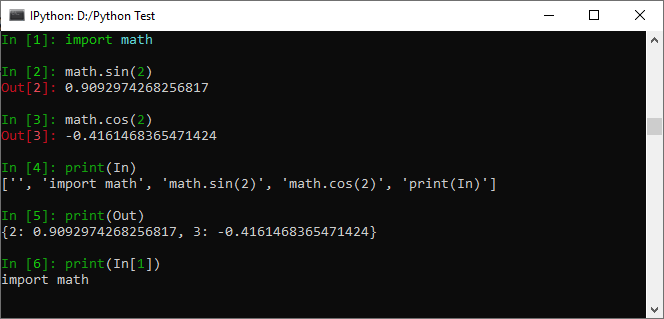
To access a general description of available magic functions:

%magic

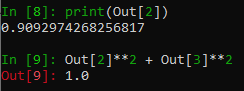
For a quick and simple list of all available magic functions:

%lsmagic

**IPython’s In and Out Objects:**



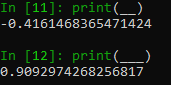
And now continue the following commands to work with output values:



To accessing previous output:

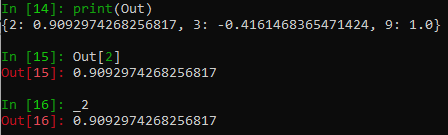


Also, to access the second last and the third last we have the following commands:



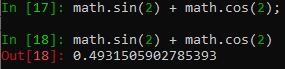
But, three is the max that we can reach using the \_.

Please type the following commands and guess what their functionalities are:



**Suppressing Output**

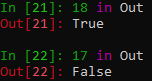
How could we suppress the output? BTW, what does it mean? And do you know any other language that has the same concept and the same solution for suppression?



Please explain the following result and tell me why we do not have any output for number 17?

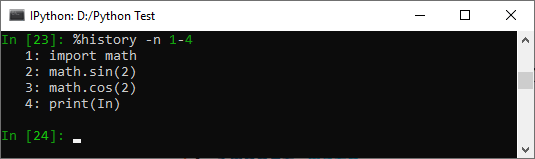


The following commands can help you for your explanation:



Our book explains as follows: “Note that the result is computed silently, and the output is neither displayed on the screen or stored in the Out dictionary.”

**Related Magic Commands:**



Also, try

%history -n

And see the result.

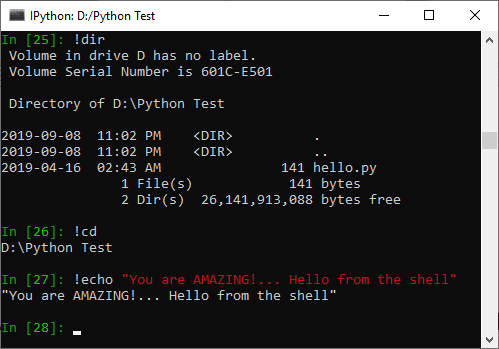
Please google the following Linux/OS X shell commands:

* echo
* pwd
* ls
* cd
* mkdir
* mv

And now please search for the following DOS shell commands:

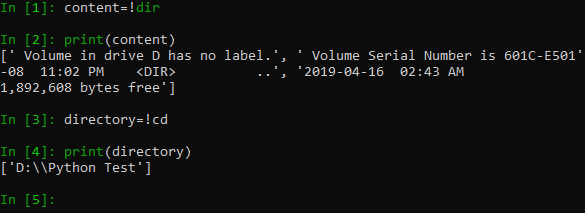
* echo
* dir
* cd
* md

Please try the following commands:



**Passing Values to and from the Shell:**

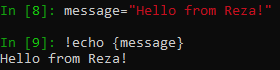
First, passing values to the shell…



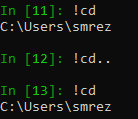
Note that these results are not returned as lists, but as a special shell return type defined in IPython:

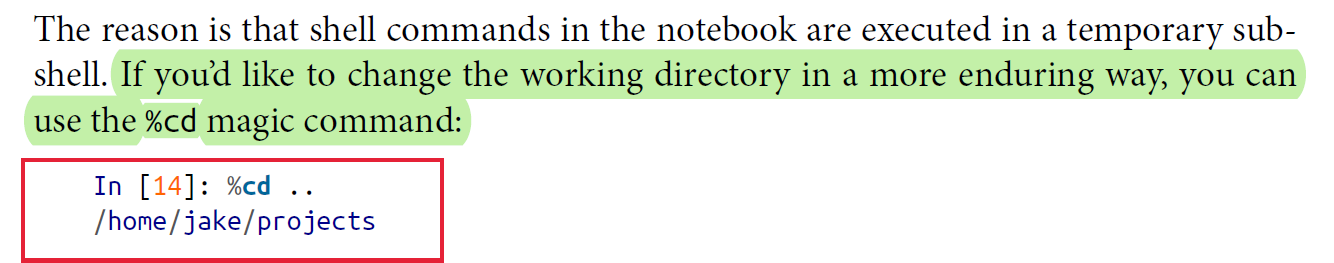


Passing Python variables into the shell….

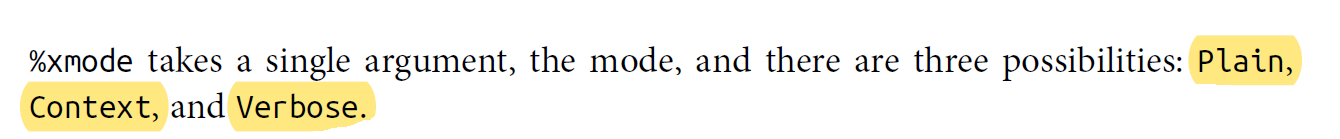


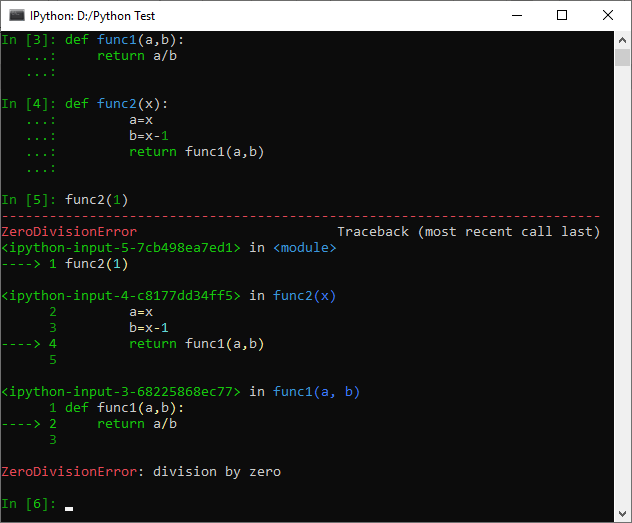
Please explain the following lines:



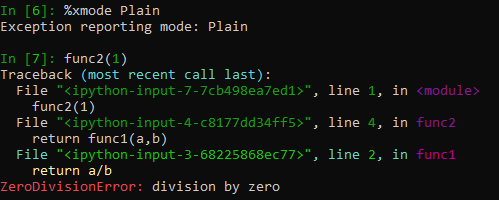


**Controlling Exceptions: %xmode:**

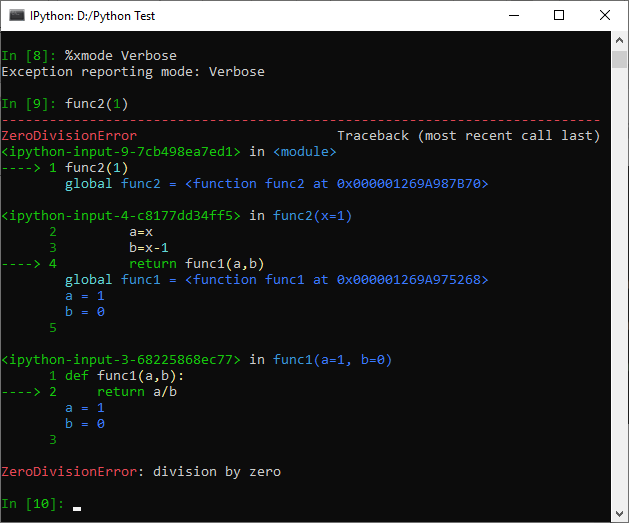
**Firstly, please type %xmode, and paste your output in the chatbox.**



A brief version of error explanation by using the plain mode as follows:

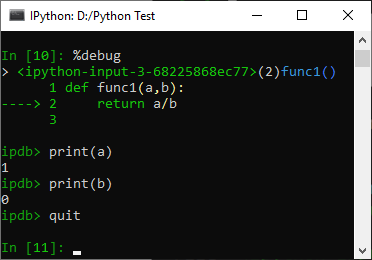


And now changing the mode to verbose as follows:



**Debugging: When Reading Tracebacks Is Not Enough**

We can type %debug and trace the parameters by commands, such as print(a) and finally quit the debugging:

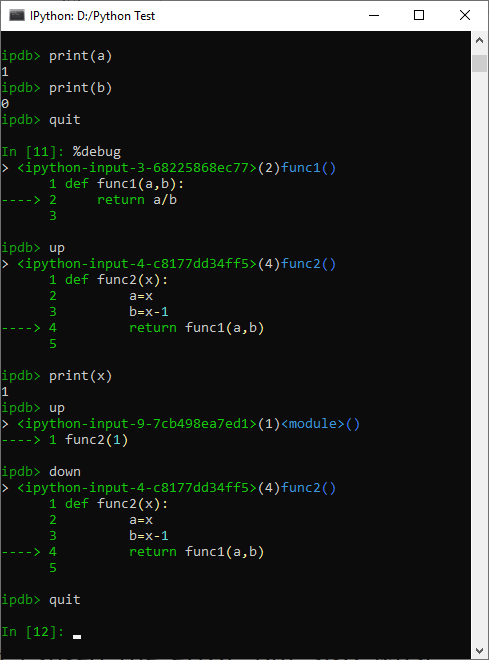


Before quit, try the following command:

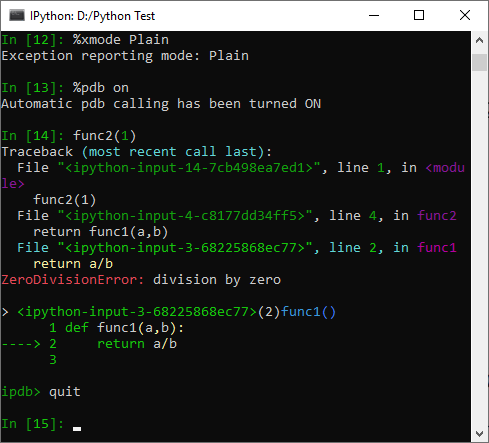
print(func2(1))

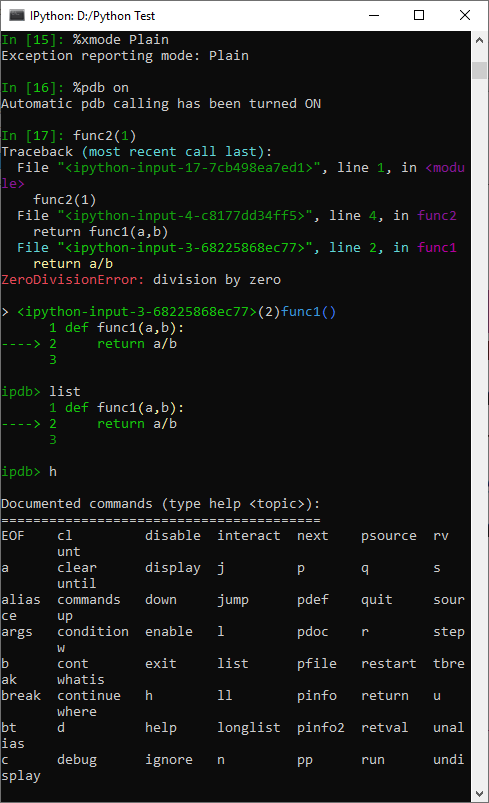
And now the following commands:

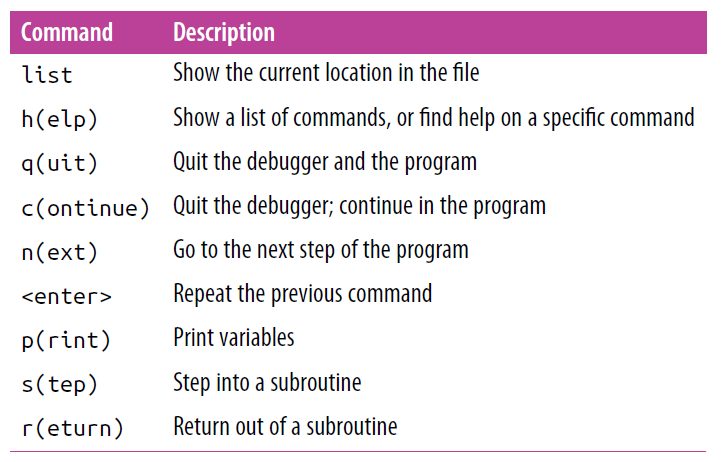
We can even step up and down through the stack and explore the values of variables there.



If you’d like the debugger to launch automatically whenever an exception is raised, you can use the %pdb magic function to turn on this automatic behavior:



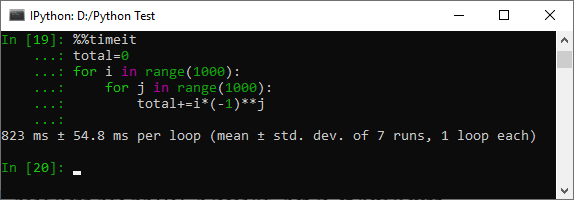
Partial list of debugging commands:



**Timing Code Snippets: %timeit and %time:**

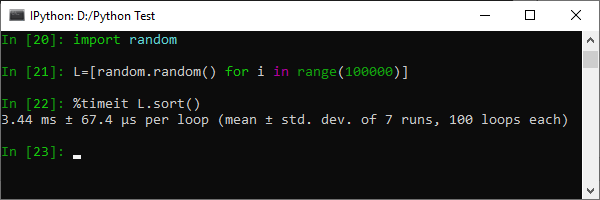


And now….

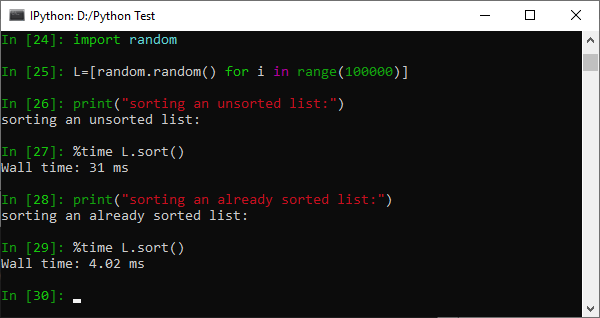


Sometimes repeating an operation is not the best option.

So…..

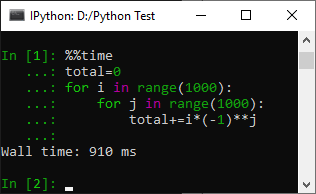


But if we use %time the result would be as follows:

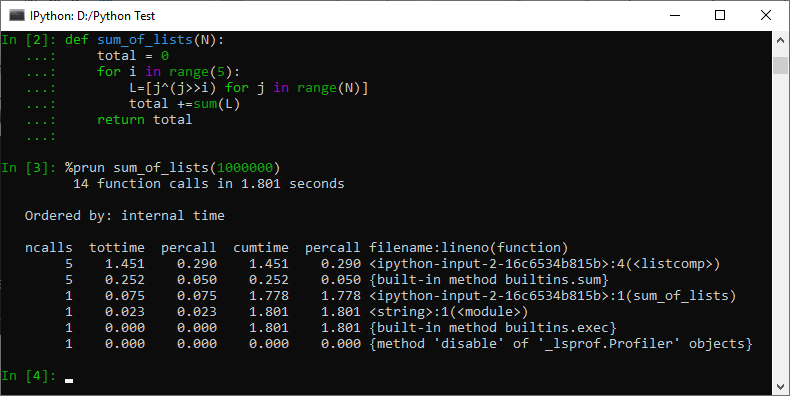


Please share your opinions with your team members and explain the differences.

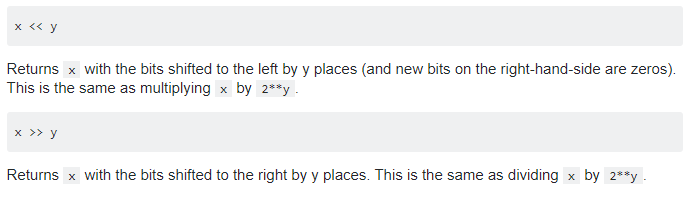
And in order to time a block of code we similarly can use double % sign as follows:

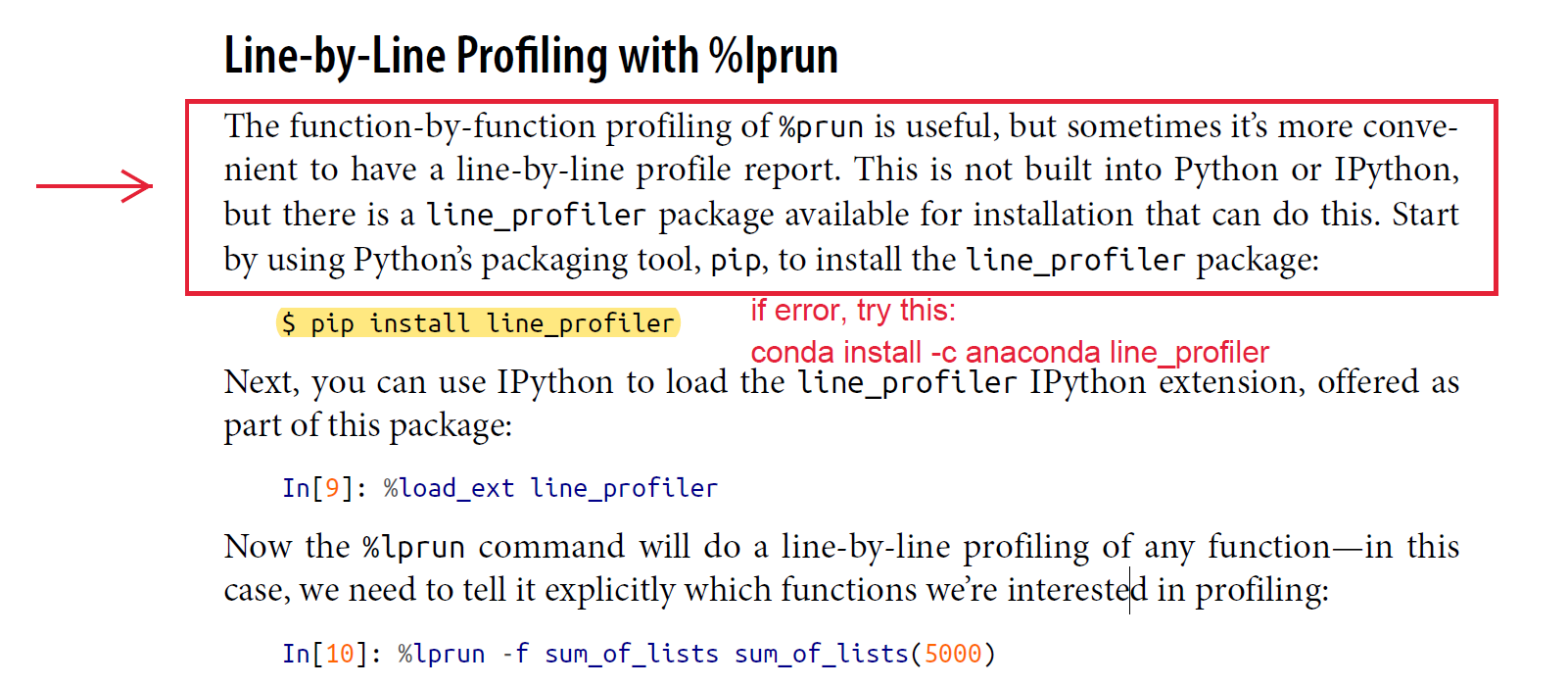


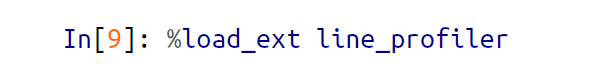
**Profiling Full Scripts: %prun:**

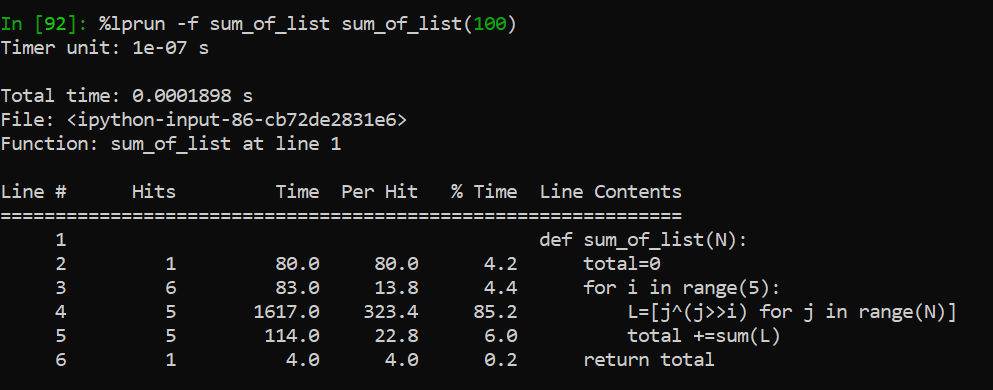


**Point:**







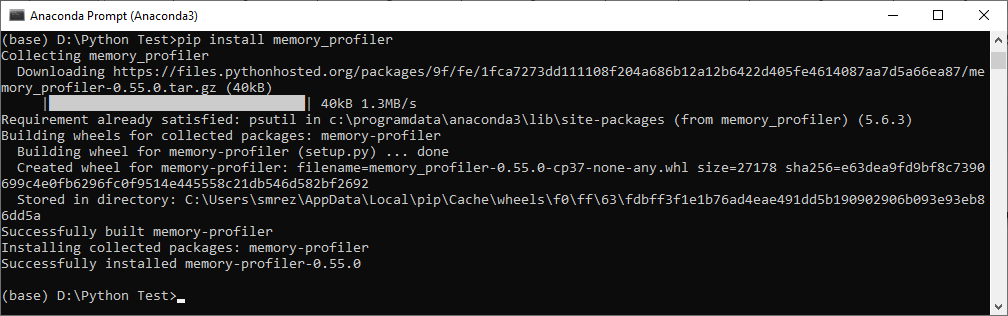


**Profiling Memory Use: %memit:**

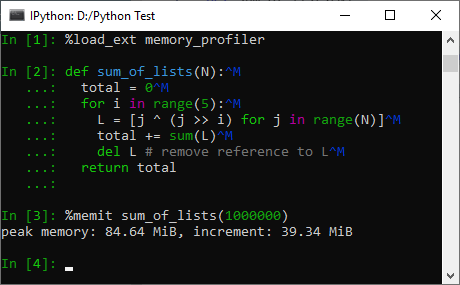
In the (DOS) shell command prompt we should type:

pip install memory\_profiler

**Point:** %memit is similar to %timeit, but is show the used memory instead of the used time 😊



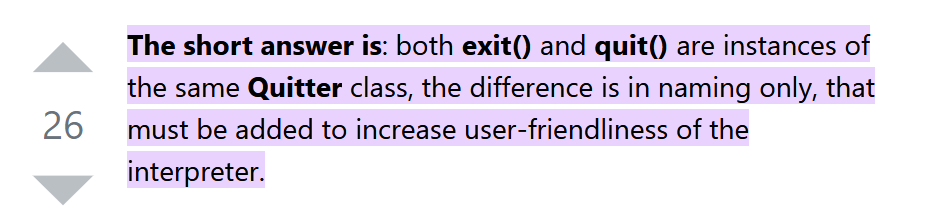
Then we return to python by typing ipython and then loading the memory\_profiler:



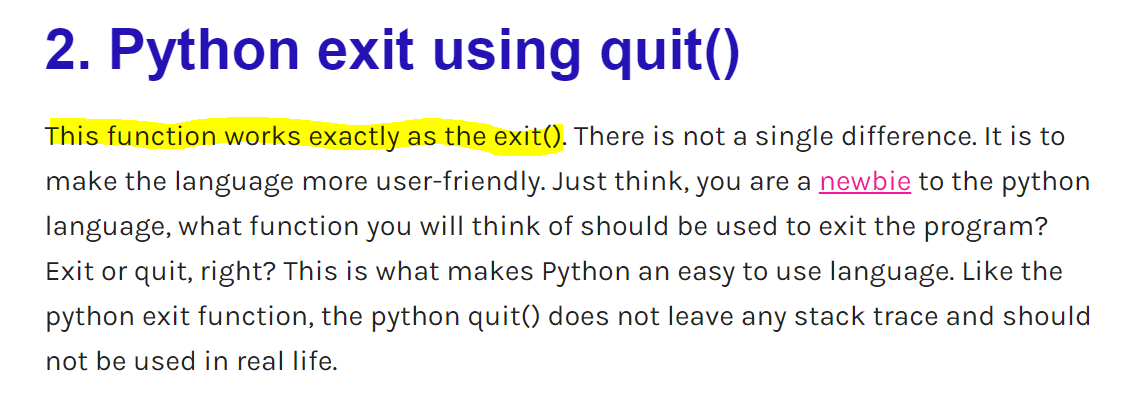
**One question which was asked in the class:**

**The difference between quit and exit:**

[**https://stackoverflow.com/questions/19288707/difference-between-quit-and-exit-in-python**](https://stackoverflow.com/questions/19288707/difference-between-quit-and-exit-in-python)



[**https://www.pythonpool.com/python-exit/**](https://www.pythonpool.com/python-exit/)



**What I have learned from Vanja:**

**If you use Ctrl + o you can have multiple commands, and it acts as all of the commands are in one Jupyter Notebook cell. Then you can press Enter, and run them all.**

**a=2+5**

**b=a\*10**

**print(b)**