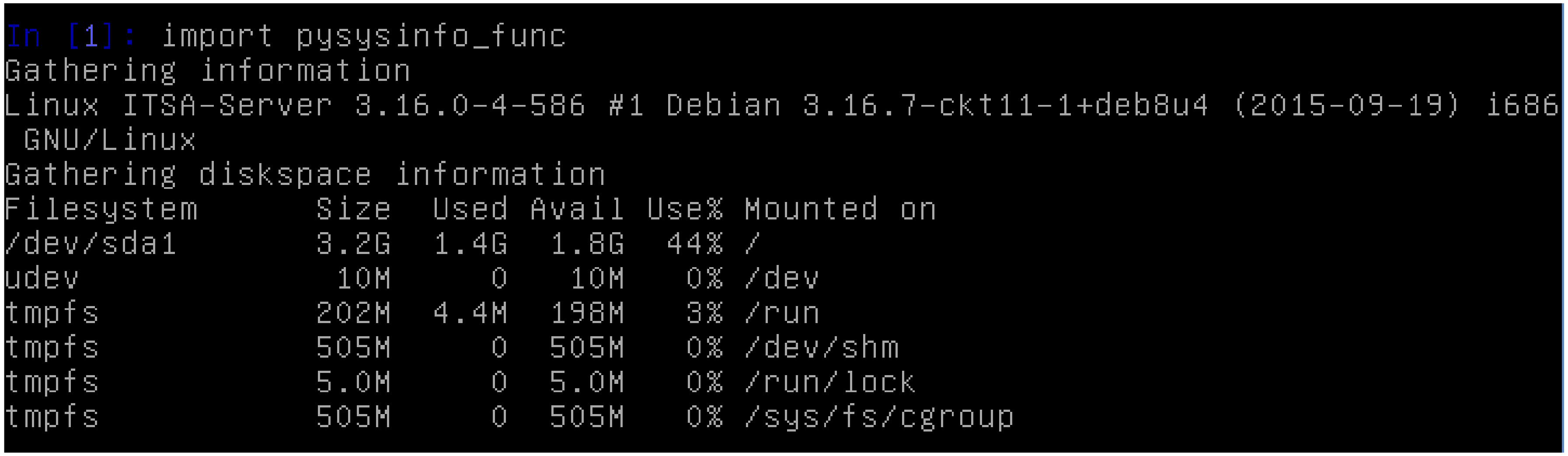
IT Scripting and Automation

Python
Part 2

Lecturer: Art Ó Coileáin

Reusing Code with import Statement

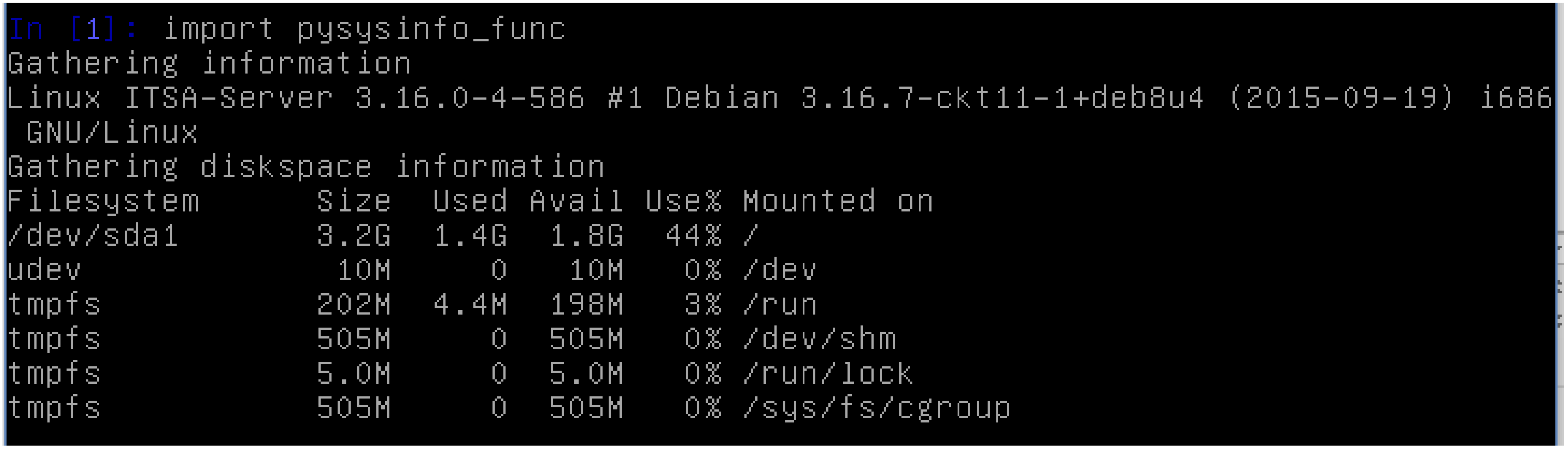
- One of the first scripts we created named: pysysinfo.py (no functions on it)
- In Python a file is a module and vice versa, we can import this script file into Ipython3. You do not need to specify the .py portion of the file you are importing.



```
In [1]: import pysysinfo_func
Gathering information
Linux ITSA-Server 3.16.0-4-586 #1 Debian 3.16.7-ckt11-1+deb8u4 (2015-09-19) i686
 GNU/Linux
Gathering diskspace information
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda1       3.2G  1.4G  1.8G  44% /
udev             10M     0   10M   0% /dev
tmpfs           202M  4.4M  198M   3% /run
tmpfs           505M     0  505M   0% /dev/shm
tmpfs           5.0M     0  5.0M   0% /run/lock
tmpfs           505M     0  505M   0% /sys/fs/cgroup
```

Reusing Code with import Statement

- There are a few problems with pysysinfo.py, i.e., if you plan to run Python code, it should always be executed from the command line as a part of a script or program. Using import is to help with this "reusing code" idea.
- What if you only wanted to print the output of the 'diskspace' portion of the script? You can't. You should use functions.
- We import the example script with functions: pysysinfo_func



```
In [1]: import pysysinfo_func
Gathering information
Linux ITSA-Server 3.16.0-4-586 #1 Debian 3.16.7-ckt11-1+deb8u4 (2015-09-19) i686
 GNU/Linux
Gathering diskspace information
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda1       3.2G  1.4G  1.8G  44% /
udev             10M     0   10M   0% /dev
tmpfs           202M  4.4M  198M   3% /run
tmpfs           505M     0  505M   0% /dev/shm
tmpfs           5.0M     0  5.0M   0% /run/lock
tmpfs           505M     0  505M   0% /sys/fs/cgroup
```

Reusing Code with import Statement

- We get the same output that we get from script that does not contain functions. The problem is that main function we created in pysysinfo_func.
- On one hand we want to be able to run our script on the command line to get the output, but on the other hand when we import it we don’t want all of the output all at once.
- Fortunately, the need to use a module as both a script that gets executed from the command line and as a reusable module is very common in Python.
- The solution is to change the way the main method gets called by replacing the last part of the script to look like this:

Reusing Code with import Statement

```
#Main function that call other functions
def main():
        uname_func()
        disk_func()
if __name__ == "__main__":
        main()
```

- Any code that you indent underneath this statement gets run only then it is executed from the command line.
- See pysysinfo_func_2.py

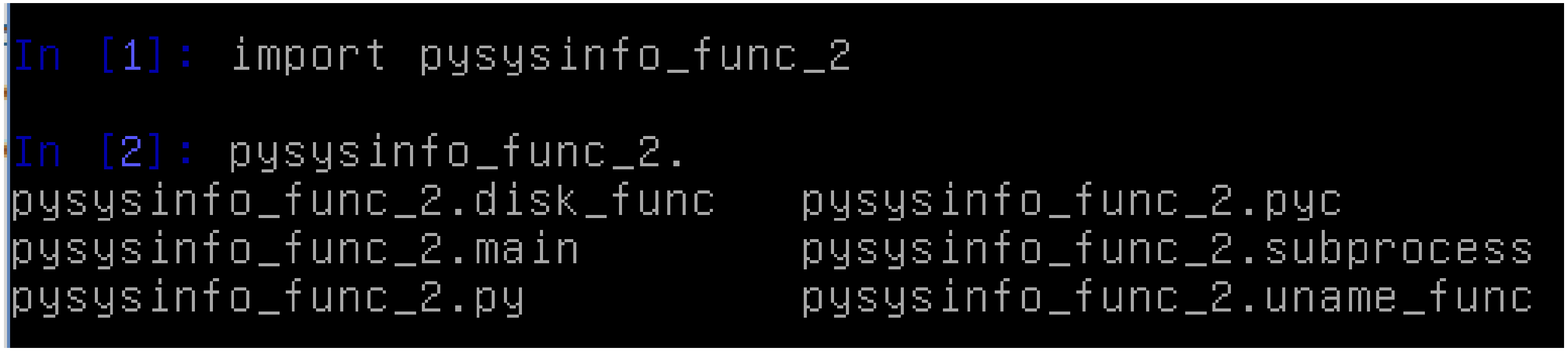
Reusing Code (pysysinfo_func_2.py)

```
#!/usr/bin/env python
#A System Information Gathering Script
import subprocess
#Command 1

def uname_func():
  uname = "uname“
  uname_arg = "-a"
  print ("Gathering system information
  with %s command:\n" % uname)

subprocess.call([uname, uname_arg])
```

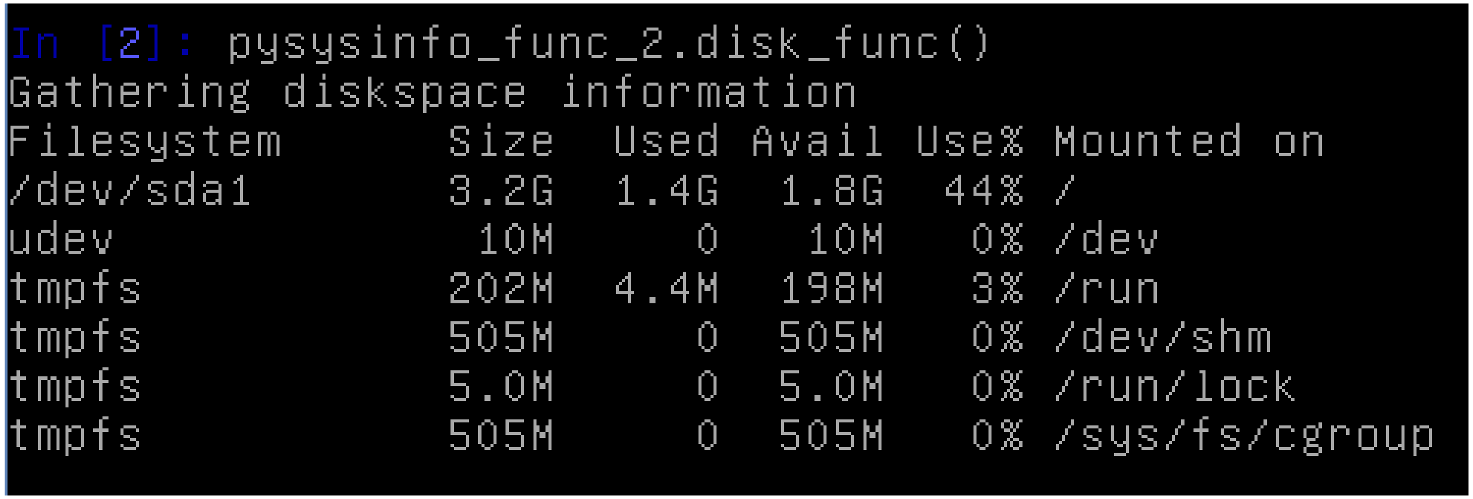
```
#Command 2
def disk_func():
  diskspace = "df"
  diskspace_arg = "-h"
  print ("Gathering diskspace
           information %s
           command:\n" % diskspace)
  subprocess.call([diskspace,  diskspace_arg])

#Main function that call other functions
def main():
  uname_func()
  disk_func()

if __name__ == "__main__":
  main()
```

Reusing Code with import Statement

- Now we have three functions that can be used in other programs or use to interact with the ipython3 shell.
- We can see that there is a pysysinfo_func_2.disk_func()



```
In [1]: import pysysinfo_func_2

In [2]: pysysinfo_func_2.
pysysinfo_func_2.disk_func    pysysinfo_func_2.pyc
pysysinfo_func_2.main         pysysinfo_func_2.subprocess
pysysinfo_func_2.py           pysysinfo_func_2.uname_func
```



```
In [2]: pysysinfo_func_2.disk_func()
Gathering diskspace information
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda1       3.2G  1.4G  1.8G  44% /
udev             10M     0   10M   0% /dev
tmpfs           202M  4.4M  198M   3% /run
tmpfs           505M     0  505M   0% /dev/shm
tmpfs           5.0M     0  5.0M   0% /run/lock
tmpfs           505M     0  505M   0% /sys/fs/cgroup
```

Reusing Code: new_pysysinfo.py

```
#Very short script that reuses pysysinfo_func_2 code
from pysysinfo_func_2 import disk_func
import subprocess
def tmp_space():
        tmp_usage = "du"
        tmp_arg = "-h"
        path = "/tmp"
        (print "Space used in /tmp directory”)
        subprocess.call([tmp_usage, tmp_arg, path])
def main():
        disk_func()
        tmp_space()
if __name__ == "__main__":
main()
```

Reusing Code: new_pysysinfo.py (2)

- In this example, not only do we reuse the code we wrote in pysysinfo_func_2, but we use a special Python syntax that allows us to import the exact function needed.
- It is possible to make a completely different program just by importing the functions from our previous programs.
- If you want to run the program as script, place that special

Syntax:

- **if __name__==“__main__”**

More about iPython

- **iPython3** is a bundle of interactive Python features.
  - As a shell, it is far superior to the standard Python shell.
  - It also provides the ability to create highly customised console-based command environments.
- **iPython3** allows for easy inclusion of an interactive Python application; and it can even be used as system shell, with some level of success.
- The biggest difference you'll see between the **iPython3** and standard Python shells is that **iPython3** gives you a numbered prompt.

Help with Magic Functions

What is a magic function?

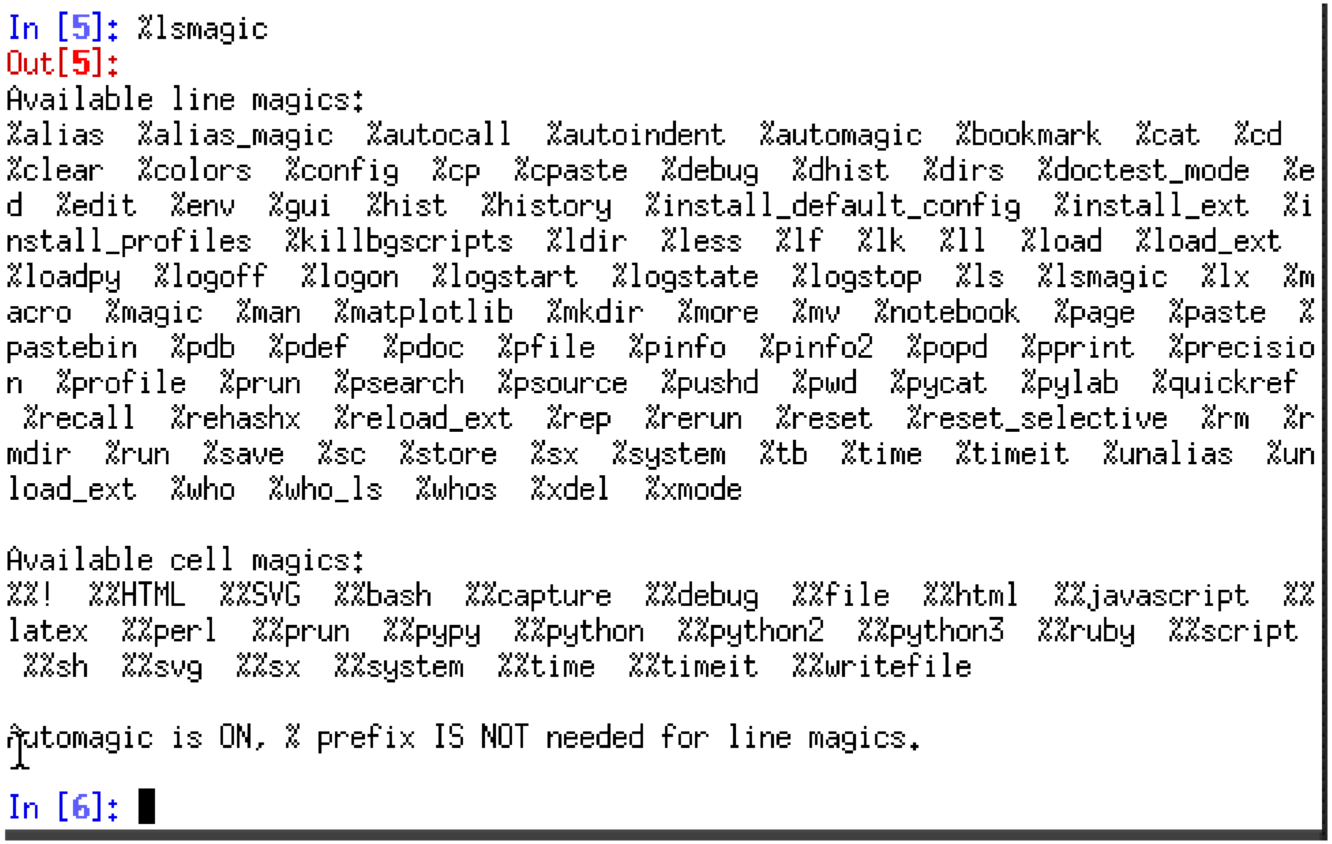
- "IPython will treat any line whose first character is a % as a special call to a 'magic' function. These allow you to control the behaviour of Ipython itself, plus a lot of system-type features. They are all prefixed with a % character, but parameters are given without parentheses or quotes."

Example:

typing **%cd mydir** changes your working directory to 'mydir', if it exists.

Help with Magic Functions

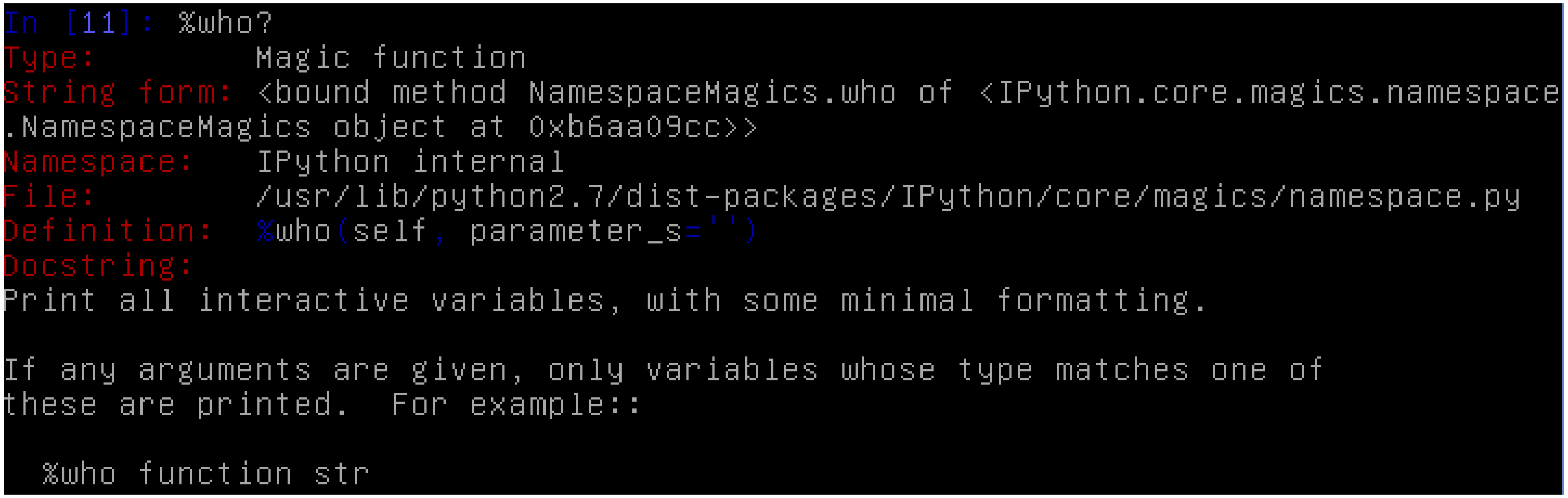
- **%lsmagic** gives a listing of all the “magic” functions.



```
In [5]: %lsmagic
Out[5]:
Available line magics:
%alias  %alias_magic  %autocall  %autoindent  %automagic  %bookmark  %cat  %cd
%clear  %colors  %config  %cp  %cpaste  %debug  %dhist  %dirs  %doctest_mode  %e
d  %edit  %env  %gui  %hist  %history  %install_default_config  %install_ext  %i
nstall_profiles  %killbgscripts  %ldir  %less  %lf  %lk  %ll  %load  %load_ext
%loadpy  %logoff  %logon  %logstart  %logstate  %logstop  %ls  %lsmagic  %lx  %m
acro  %magic  %man  %matplotlib  %mkdir  %more  %mv  %notebook  %page  %paste  %
pastebin  %pdb  %pdef  %pdoc  %pfile  %pinfo  %pinfo2  %popd  %pprint  %precisio
n  %profile  %prun  %psearch  %psource  %pushd  %pwd  %pycat  %pylab  %quickref
 %recall  %rehashx  %reload_ext  %rep  %rerun  %reset  %reset_selective  %rm  %r
mdir  %run  %save  %sc  %store  %sx  %system  %tb  %time  %timeit  %unalias  %un
load_ext  %who  %who_ls  %whos  %xdel  %xmode

Available cell magics:
%%!  %%HTML  %%SVG  %%bash  %%capture  %%debug  %%file  %%html  %%javascript  %%
latex  %%perl  %%prun  %%pypy  %%python  %%python2  %%python3  %%ruby  %%script
 %%sh  %%svg  %%sx  %%system  %%time  %%timeit  %%writefile

Automagic is ON, % prefix IS NOT needed for line magics.

In [6]:
```

Help with Magic Functions

- There are more than 100 magic functions.
- In order to get help, you should type the name of the magic function followed by a question mark (**?**), it will give almost the same information that **%magic** will give.
- Example: **In [1]: %who ?**



```
In [11]: %who?
Type:          Magic function
String form: <bound method NamespaceMagics.who of <IPython.core.magics.namespace
.NamespaceMagics object at 0xb6aa09cc>>
Namespace:     IPython internal
File:          /usr/lib/python2.7/dist-packages/IPython/core/magics/namespace.py
Definition:    %who(self, parameter_s='')
Docstring:
Print all interactive variables, with some minimal formatting.

If any arguments are given, only variables whose type matches one of
these are printed.  For example::

  %who function str
```