01_01_Ipython

August 11, 2019

1 01. IPython

1.1 01.01 What is IPython?

According to the IPython Website: https://ipython.org IPython provides a rich architecture for interactive computing with:

- A powerful interactive shell. *(I confirm)*
- A kernel for Jupyter. *(I confirm)*
- Support for interactive data visualization and use of GUI toolkits. *(I confirm)*
- Flexible, embeddable interpreters to load into your own projects. *(I have no experience)*
- Easy to use, high performance tools for parallel computing. *(I have no experience)*

According to REF1 : IPython (short for Interactive Python) was started in 2001 by Fernando Perez as an enhanced Python interpreter, and has since grown into a project aiming to provide, in Perez's words, "Tools for the entire lifecycle of research computing." **If Python is the engine of our data science task, you might think of IPython as the interactive control panel.**

Run IPython:

1.2 01.02 Useful IPython tips&tricks

1.2.1 01.02.01 Quick access with ? and ??

This is a small thing but it is amazing!

To see documentation

```
print?
```

To see source code (better check on 3rd party packages, not Built-in Functions)

```
import matplotlib.pyplot as plt
plt.subplots??
```

1.2.2 01.02.02 IPython Magic Commands

Run the named file inside IPython as a program:

[1]:
```
%run meet_us.py
```

```
Hello J|o|h|n!
Hello M|i|k|e!
Hello E|m|i|l|y!
```

Measure time with `%timeit`

[2]:
```
%timeit full_sum=sum([k for k in range(10_000)])
```

```
350 ţs ś 2.34 ţs per loop (mean ś std. dev. of 7 runs, 1000 loops each)
```

Double percent sign eg. `%%timeit` means: Apply for the whole cell instead of a single line

[3]:
```
%%timeit
k = 0
full_sum = 0
while full_sum < 10e5:
    full_sum += k
    k += 1
```

```
166 ţs ś 19.5 ţs per loop (mean ś std. dev. of 7 runs, 10000 loops each)
```

Arguments can be passed to Magic Commands via:

[4]:
```
%%timeit -n 5 -r 2
k = 0
full_sum = 0
while full_sum < 10e5:
    full_sum += k
    k += 1
```

```
190 ţs ś 1.41 ţs per loop (mean ś std. dev. of 2 runs, 5 loops each)
```

1.2.3 01.02.03 Input and output history

`In` and `Out` variables are set automatically and stores all intput ad output values related with cells

[5]:
```
print(In)
```

```
['', "get_ipython().run_line_magic('run', 'meet_us.py')",
"get_ipython().run_line_magic('timeit', 'full_sum=sum([k for k in
range(10_000)])')", "get_ipython().run_cell_magic('timeit', '', 'k =
0\\nfull_sum = 0\\nwhile full_sum < 10e5:\\n    full_sum += k\\n    k += 1')",
"get_ipython().run_cell_magic('timeit', '-n 5 -r 2', 'k = 0\\nfull_sum =
0\\nwhile full_sum < 10e5:\\n    full_sum += k\\n    k += 1')", 'print(In)']
```

[6]:
```
x = 100
```

[7]:
```
x += 1
x
```

[7]:
```
101
```

[8]:
```
x += 1
x
```
[8]:
```
102
```
[9]:
```
x += 1
x
```
[9]:
```
103
```
[10]:
```
x += 1
x
```
[10]:
```
104
```
[11]:
```
print(Out)
```
```
{7: 101, 8: 102, 9: 103, 10: 104}
```
[12]:
```
print(Out[9])
```
```
103
```
Pure `python` allows us to explore returning history with with one underscore `_`. In `IPython` we have upt to 3 underscores avaliable

[20]:
```
print(_)
```
```
206
```
[21]:
```
print(__)
```
```
104
```
[22]:
```
print(___)
```
```
104
```
To surpass output add `;` at the end of code line

[27]:
```
sum([100, x, x**2])
```
[27]:
```
11020
```
[28]:
```
sum([100, x, x**2]);
```

1.2.4 01.02.04 Shell commands

To run shell command start the line with exclamation mark

[30]:
```
!echo $PATH
```
```
/home/lcs123/venvs/dstip_venv/bin:/home/lcs123/anaconda3/bin:/usr/local/sbin:/us
r/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
```
You can easy combine `IPython` and `shell` functionality

[33]:
```
directory_content = !ls
print(type(directory_content))
print(directory_content)
```

```
<class 'IPython.utils.text.SList'>
['01_01_Ipython.ipynb', 'img', 'meet_us.py']
```

[34]:
```
current_working_dir = !pwd
print(type(current_working_dir))
print(current_working_dir)
```

```
<class 'IPython.utils.text.SList'>
['/home/lcs123/DS/dstip/01_Ipython']
```

[35]:
```
text_to_print = "Welcome stranger!"
```

[36]:
```
!echo $text_to_print
```

```
Welcome stranger!
```

[38]:
```
!echo {text_to_print}
```

```
Welcome stranger!
```

[39]:
```
!echo text_to_print
```

```
text_to_print
```

Other `IPython` funcionalities like: - profiling - debugging - and many more you can find in REF1

1.2.5 01.02.04 When to use `IPython`?

[8]:
```
from dstip_utils.utils import yes_no_table
yes_no_dict = {'yes': ["Interactive computing", "Exploring, prototyping,␣
 ↪learning"],
               'no':['Software Development']}
yes_no_table(yes_no_dict)
```

```
<IPython.core.display.HTML object>
```