introduction_jupyter_notebooks

February 15, 2022

1 Introduction to Jupyter Notebooks

Jupyter Notebooks are an enviornment allowing you to code and to take notes in one same document. That means that in this notebook you can: - write cells like this one in markdown (try and press `Enter` when this cell is selected, it will show you the code for it... press `Ctrl-Enter` or `Shift-Enter` and it's back to how it was!); - write code cells like the one just below (press `Ctrl-Enter` or `Shift-Enter` to run it as well):

```
In [1]: print("Hello Jupyter!")
```

```
Hello Jupyter!
```

1.1 Shortcuts

The Jupyter environment is full of shortcuts and great things that make your life easier.

A quick list: - press `Enter` to start typing into a cell, `Escape` to navigate your notebook (you can then press `j` or `k`, or the arrow keys, to go up and down; - when navigating (not typing in a cell), you can press `a` to create a new cell above, `b` for below; - if a cell is selected (but you are not typing in it), you can press `m` to turn it into a `markdown` cell, and `y` to make it a code cell; - if you press `Ctrl-Enter`, you will run the cell and stay on it, whereas if you press `Shift+Enter`, you go one cell below (and if it is the last one, a new one will be created); - if a cell is selected, you can press `x` to cut it, `c` to copy it, and then `v` to paste after the current cell, and `V` to paste before it; - **Many** other shortcuts, which you can also customise, check in the menu above "Help" > "Keyboard Shortcuts".

1.2 Markdown

Markdown is an easy way to write text in a web context without going for the full html syntax.

You can check out this cheatsheet.

Basic things: - Separate paragraphs with an empty line, or two spaces at the end of the line; - italic with `*asterisks*` or `_underscores_`, bold with `**two**` or `__two__`, (the two can be combined); - lists, numbered or unnumbered, like so:

```
unnumbered list with a line starting with -
numbered list with 1., 2., etc.
```

- code snippets using this the grave accent ‘, multiline with three of them “‘;
- titles (h1-6) using # starting the line, from just one to six;
-

1.3 three stars *** or dashes --- give you a horizontal line like tihs one (it needs to be at the start of the line):

- links are generated like so: `[text](link.com);`
- images similarly: `![image text](image.jpg);`

For more details, check out the cheatsheet!

---

2 Simple coding

As seen in the videos, any code cell allows you to input code and get results immediately.

```
In [2]: x = 10
        y = x + 2
        y
```

```
Out[2]: 12
```

Notice that if you run a cell with a variable at the end of it, it will act like a print out of that variable.

```
In [3]: print(y) # same as above, and our y still exist!
```

```
12
```

You can do all sorts of nice things now in Python!

```
In [9]: mylist = [1,2,3,4]
        mylist_squared = [element**2 for element in mylist] # nice and Pythonic list comprehen
        print(mylist_squared)
        print(max(mylist_squared)) # the maximum element
```

```
[1, 4, 9, 16]
16
```

When in doubt, ask for help! It can be quite dry at first but if you get used to how it’s written, it’s a goldmine of information!

```
In [8]: help(max)
```

```
Help on built-in function max in module builtins:

max(...)
    max(iterable, *[, default=obj, key=func]) -> value
    max(arg1, arg2, *args, *[, key=func]) -> value

    With a single iterable argument, return its biggest item. The
    default keyword-only argument specifies an object to return if
    the provided iterable is empty.
    With two or more arguments, return the largest argument.
```

Don't ever forget to press the `TAB` button as well, which allows you to access both the auto-complete (for variables, classes and functions names) and methods (in-built functionality from libraries).

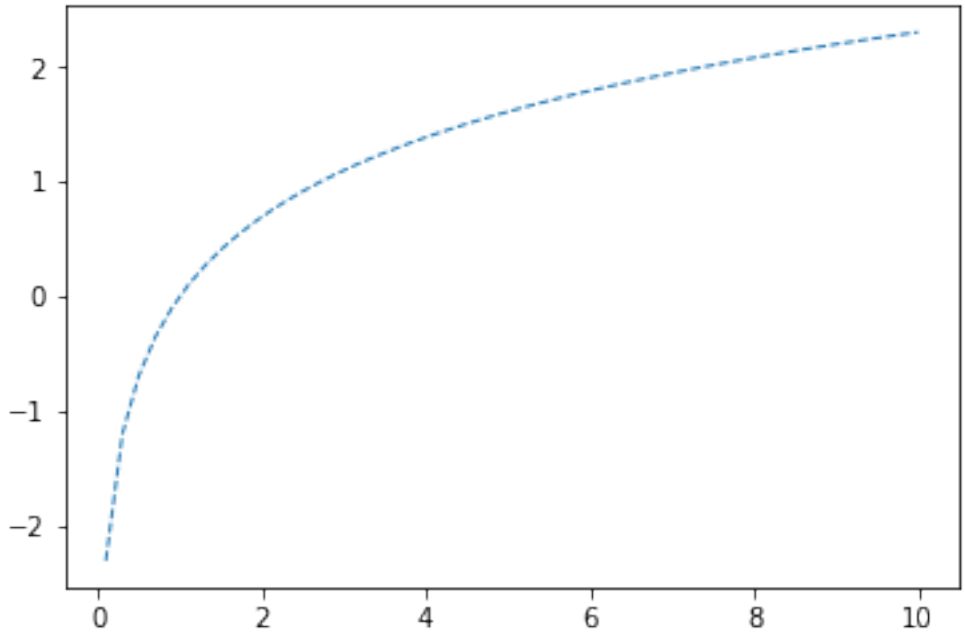
In the cell below, put your cursor at the end of each of those, press `TAB` and look at the list of options. (These will only work if you have run the cells defining the variables `mylist` and `mylist_squared` above!)

Beware! These cells are now incomplete, and will throw an error if you run them like that!

```
In [12]: my # put your cursor after y and hit `TAB`: you will be able to choose between the tw

In [ ]: mylist. # hitting `TAB` after the . will show all list methods that Python has (many!)
```

2.1 Now a Data Science example!

```
In [19]: %matplotlib inline

         import matplotlib # remember in Python you have to import the relevant libraries

         import numpy as np

         import matplotlib.pyplot as plt

         x = np.linspace(0.1,10) # return evenly spaced numbers over a specified interval, fro

         line = plt.plot(x, np.log(x),'--',linewidth = 1) # plot these evenly spaced numbers a
                                                          # use '--' as line (try with just on
                                                          # and control the width
```




```
In [10]: help(np.linspace)

Help on function linspace in module numpy.core.function_base:

linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None)
    Return evenly spaced numbers over a specified interval.

    Returns `num` evenly spaced samples, calculated over the
    interval [`start`, `stop`].

    The endpoint of the interval can optionally be excluded.

    Parameters
    ----------
    start : scalar
        The starting value of the sequence.
    stop : scalar
        The end value of the sequence, unless `endpoint` is set to False.
        In that case, the sequence consists of all but the last of ``num + 1``
        evenly spaced samples, so that `stop` is excluded.  Note that the step
        size changes when `endpoint` is False.
    num : int, optional
        Number of samples to generate. Default is 50. Must be non-negative.
    endpoint : bool, optional
        If True, `stop` is the last sample. Otherwise, it is not included.
        Default is True.
```

```
retstep : bool, optional
    If True, return (`samples`, `step`), where `step` is the spacing
    between samples.
dtype : dtype, optional
    The type of the output array.  If `dtype` is not given, infer the data
    type from the other input arguments.

    .. versionadded:: 1.9.0

Returns
-------
samples : ndarray
    There are `num` equally spaced samples in the closed interval
    ``[start, stop]`` or the half-open interval ``[start, stop)``
    (depending on whether `endpoint` is True or False).
step : float, optional
    Only returned if `retstep` is True

    Size of spacing between samples.


See Also
--------
arange : Similar to `linspace`, but uses a step size (instead of the
         number of samples).
logspace : Samples uniformly distributed in log space.

Examples
--------
>>> np.linspace(2.0, 3.0, num=5)
array([ 2.  ,  2.25,  2.5 ,  2.75,  3.  ])
>>> np.linspace(2.0, 3.0, num=5, endpoint=False)
array([ 2. ,  2.2,  2.4,  2.6,  2.8])
>>> np.linspace(2.0, 3.0, num=5, retstep=True)
(array([ 2.  ,  2.25,  2.5 ,  2.75,  3.  ]), 0.25)

Graphical illustration:

>>> import matplotlib.pyplot as plt
>>> N = 8
>>> y = np.zeros(N)
>>> x1 = np.linspace(0, 10, N, endpoint=True)
>>> x2 = np.linspace(0, 10, N, endpoint=False)
>>> plt.plot(x1, y, 'o')
[<matplotlib.lines.Line2D object at 0x...>]
>>> plt.plot(x2, y + 0.5, 'o')
[<matplotlib.lines.Line2D object at 0x...>]
>>> plt.ylim([-0.5, 1])
```

```
(-0.5, 1)
>>> plt.show()
```