JupyterGuide

November 1, 2019

Do not attempt to submit this file on Moodle

1 Jupyter Quickstart

This tutorial very briefly introduces the Jupyter Notebook environment. The most important things to know:

- code cells are grey and you can select them by clicking on them
- pressing `SHIFT-ENTER` runs the selected cell and moves to the next one.
- pressing `CTRL-ENTER` runs the selected cell and stays there.
- pressing `TAB` will autocomplete
- pressing `SHIFT-TAB` will bring up documentation for any function
- `ESC-H` will bring up the full keyboard shortcut list
- `CTRL-S` will save the document

Actions for you to do are highlighted in green.

1.0.1 Cells

The notebook is made up of **cells**. A cell is a snippet of code or text. You can break up code into cells as you wish. Each cell with code in it can be run on its own.

- Click on a cell to highlight it (you will see a highlight box appear around it)
- Press SHIFT-ENTER to run the code (hold SHIFT, then press ENTER).

A.1 Try this on the cell below:

```
[ ]: print("Hello, world")
```

After selecting the cell above and pressing SHIFT-ENTER. you should see:

```
In [1]: print("Hello, world")
Hello, world
```

The line underneath is the **output** of the code you ran. All the code in the cell is run when you press SHIFT-ENTER.

1.0.2 Editing

You can edit a code cell just by clicking on it, and editing the text. Press SHIFT-ENTER to run it again.

A.2 Edit the code in the cell above to print out a different message, then run it again.

1.0.3 Text cells

The instructions you are reading are also in cells, but these are **text cells**. If you *double*-click on these text cells you can edit them as well. Press SHIFT-ENTER to "format" the cell.

The text inside these boxes is Markdown.

Jupyter will also recognise LaTeX formulae if you surround them with dollar symbols `$ x = x + 1 $` $ x = x + 1 $ And you can make display equations using double dollar signs: `$$ \sum_{i=0}^{N} i^{\alpha_i} + \beta_i $$` becomes

$$\sum_{i=0}^{N} i^{\alpha_i} + \beta_i$$

A.3 Double-click on this cell. It will turn gray and the font will change. Edit this text to say something different. Then press SHIFT-ENTER to format it.

1.0.4 Navigation and editing

- Editing *inside* cells works like a normal text form on a web page.

A.4 Try typing `print("Hello Jupyter")` in the box below, then pressing SHIFT-ENTER to run it.

[ ]:
```

```

1.0.5 Copy and paste

- You can cut, copy and paste entire cells using the toolbar or menu, or use `CTRL-X` (cut), `CTRL-C` (copy) and `CTRL-V` (paste).

A.5) Try copying the cell below that says "copy me!" and paste it twice below.

1.1 Copy me!

1.1.1 Adding new cells

- You can insert new cells (e.g. to write code in) by using `Insert/Cell Above` or `Insert/Cell Below` from the menu. Or use the + button on the toolbar.

A.6 Insert a cell below, and enter `print("hello from a new cell")`

1.1.2 Adding notes

Sometimes you want to add a text cell, to write notes, for example. To do this, add a cell as before, *then* go to the `Cell` menu and select `Cell Type/Markdown` (or press `Ctrl-M`). To change the cell type back to code, go to `Cell/Cell Type/Code` (or press `Ctrl-Y`).

** A.7 Insert a text cell below. Put the text `# Title` in the cell. Press SHIFT-ENTER to "format" the text. Notice that when you press SHIFT-ENTER the text will become a title line. This is caused by the leading # symbol. Jupyter uses markdown to format text**

1.1.3 Saving

- Remember to **save** regularly! Press `CTRL-S` to save, or use the menu option `File/Save and Checkpoint`

1.1.4 Making copies of notebooks

- You can make a copy of your work using `File/Make a Copy` (e.g. if you want to change something but have a "good" version to go back to).

  ** A.9 Make a copy of this notebook. You can change the copy's name using the menu option `File/Rename...` Then save. **

1.1.5 Undo

- You can undo typing inside cells (`Edit/Undo` or `CTRL-Z`) but be aware that undo works independently inside each cell (i.e. each cell maintains its own undo history).

1.1.6 Stopping and restarting

You can make Python run for a very long time, for example if you create an infinite loop. No other cells can run while we are waiting.

If this happens, you will see an asterisk `In [*]` beside the cell that is running. The circle next to `Python 3` at the very top right of the screen will be filled while the process is busy. To stop it, press the stop button on the toolbar at the top.

A.10 Run the cell below. It will run forever. Look for the asterisk next to the cell, check the circle at the very top right, and then stop the cell. You'll see a `KeyboardInterrupt` exception message. This is correct.

[88]:
```
# infinite loop
while 1:
    pass
```

```
---------------------------------------------------------------------------

KeyboardInterrupt                         Traceback (most recent call last)

<ipython-input-88-84f17e00042e> in <module>()
```

```
      1 # infinite loop
      2 while 1:
----> 3     pass


    KeyboardInterrupt:
```

1.1.7 Variables

When you run a cell, it changes the Python state. For example if you assign a value to a variable, the next cell you run will know about that value, regardless of where it is in the notebook.

** A.11 There are two cells below. Run them both. **

[89]:
```
# greeting 1
greeting = "Hello, world"
```

[92]:
```
# print greeting
# Python remembers the value of greeting from the last run cell.
print(greeting)
```

```
Is this thing on?
```

12) Now run the cell below (`#greeting 2`), then run the `print(greeting)` cell again. Notice that the output changes. Try running the first greeting cell again.

Python runs each cell when you tell it, and remembers the values you set. The order the cells appear in the notebook isn't important – it is the order in which you **run** the cells.

[93]:
```
# greeting 2
greeting = "Is this thing on?"
```

1.2 Resetting the interpreter state

Occasionally you might want to reset the Python state back to what it was before any code ran – to forget any variables you have stored.

12) Press the restart button on the toolbar, or go to the menu option `Kernel/Restart`. The contents of the notebook will not change. Now try running the `print(greeting)` cell above. You will get an error, because Python has forgotten the value. Run the `greeting 1` cell and then run the `print(greeting)` cell.

1.3 Autocompleting

If you start typing part of a Python name, then press `TAB`, a list of autocompletions will appear. For example, type `g` in the code cell below and then press `TAB`. You should see `greeting` appear in the list. Use the arrow keys to move down to `greeting` and then press `Enter` to complete the name (or just click on it).

If you type `gree` and press TAB it will complete without giving you a list, because there is only one completion beginning `gree` currently known to Python.

[ ]:
```

```

1.3.1 Built-in help

Trying pressing `Escape-H` (the `Escape` key, then `h`). This will bring up a complete list of keyboard shortcuts for th Jupyter notebook. You won't need most of them, but it can be a handy reference.

Python help You can also get help on Python commands and functions directly in the notebook. If you click anywhere inside the parentheses in the line below, and then press `SHIFT-TAB`, a little popup will appear with a brief description.

If you click the small ^ symbol at the top right of this popup, then a more complete description will be displayed at the bottom of the page. You can close this help window with the `X` at the top right.

[5]:
```
sum([1,2,3,4])
```

[5]:
```
10
```

1.4 Inline graphs

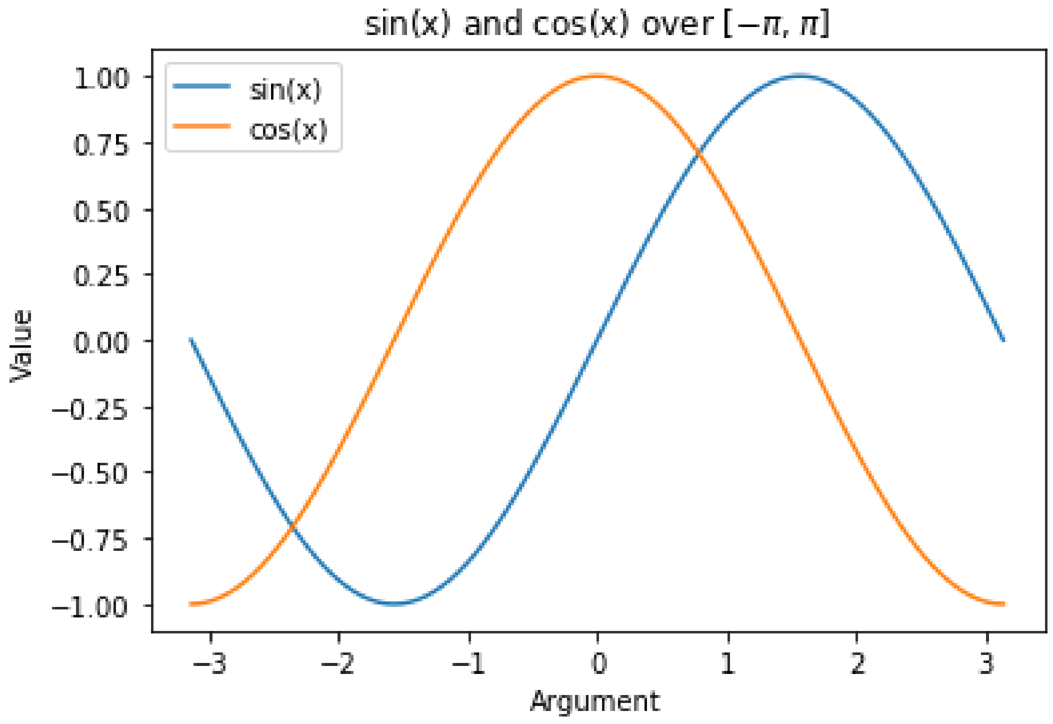
Jupyter supports inline graph visualisation (actually it supports images, sounds, videos, UI components, 3D visualisations and more; but we'll stick to graphs for now). Let's plot a graph of a sine and cosine wave. We'll look at some of these commands in more detail later in the exercise, but for now, just run them and see what happens.

We'll use **numpy** to manipulate vectors of values, and **matplotlib** to plot the graph. First we must import them. In future notebooks this will already be done at the start, but we do it explicitly here for clarity.

[6]:
```
import numpy as np # np is the conventional short name for numpy
import matplotlib.pyplot as plt # and plt is the conventional name for matplotlib
# this enables Jupyter integration
%matplotlib inline
```

[7]:
```
### Generate 500 numbers spaced evenly from -pi to pi
x = np.linspace(-np.pi, np.pi, 500)

## Create a new figure and put axes on it (a "subplot")
fig = plt.figure()
ax = fig.add_subplot(1,1,1)

## Now plot the function value against the x value
ax.plot(x, np.sin(x), label="sin(x)")
ax.plot(x, np.cos(x), label="cos(x)")

## Set the title and axis labels
ax.set_title("sin(x) and cos(x) over $[-\pi, \pi]$")
ax.set_xlabel("Argument")
ax.set_ylabel("Value")

## Use the labels given in the ax.plot() commands to create a legend
ax.legend()
```

[7]: `<matplotlib.legend.Legend at 0x2875ad149b0>`


sin(x) and cos(x) over [−π, π]
sin(x)
cos(x)
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Value
−3
−2
−1
0
1
2
3
Argument