Demo notebook for Jupyter

This short notebook provides some examples of you can do in Jupyter while reminding you some particularities of Python.

In Jupyter everything happens in "Cells", if you click three times (or press enter) on this text you will enter the content of the cell and observe that there is a green vertical bar on the left-hand-side indicating that the cell is "active" and that you can modify it.

Navigation and shortcuts

If you single click on this cell, you will have selected it in **command mode**. Double clicking on it enters you into **edit mode**. When in **edit mode**, switch to **command mode** by clicking away, hitting esc, or by running the cell (shift + enter). When in **command mode** you can switch to **edit mode** on the selected cell by hitting enter. You can tell what mode you're in by the colour surrounding the selected cell **edit mode** is green and **command mode mode** is blue

Try switching in and out of these modes on this cell.

Enter **command mode** and hit `h` to view the shortcut menu.

Markdown cells

Note that this cell here is a *markdown*-cell meaning that it's just formatted text with

- *emphasis*
- **bold**
- titles (see `# Demo notebook` at the top, you get lower level titles by adding `#` so `## Subtitle`)
- bullet points

You can also have

- code by surrounding something with backticks e.g.: `x = 5`
- display code by surrounding a block of code with triple backticks

```
print("Hello world!")
```

- maths (if you know LaTeX) by surrounding the expression with dollar signs like $\sqrt{x^2} = |x|$
- display maths by surrounding the expression with two dollar signs

$$\exp(i\pi) + 1 = 0$$

- And even HTML functionality such as colour

Code cells

By default, when created, cells are code cells (executable python code which you can run). The output is shown below the cell in much the same was as if you were in a python terminal. In fact, the notebook uses `IPython.display.display()` (https://ipython.readthedocs.io/en/stable/api/generated/IPython.display.html?

highlight=display#IPython.display.display) to choose how to format the output.

In order to run a cell, you need to press **SHIFT+ENTER** or **CTRL+ENTER** (the effect is slightly different, the first one executes then goes to the next cell where the second executes and stay in the current cell)

Run the cells below for example.

In [1]:

```
# some python here
my_dict = {'name': ['Alice', 'Bob', 'Ali'], 'gender': ['M', 'F']}
my_dict['name'][0]  # this won't be shown
my_dict['name'][1]
```

Out[1]:

```
'Bob'
```

In [2]:

```
my_dict['name'][0], my_dict['name'][1]
```

Out[2]:

```
('Alice', 'Bob')
```

In [3]:

```
print(my_dict['gender'][0])  # you can still call print and get output mid cell
print(my_dict['gender'][1])
my_dict['name'][0]
```

```
M
F
```

Out[3]:

```
'Alice'
```

Have a play around with the above cells to see when things are and are not displayed.

Switching from Markdown to Code

If you want a cell to be interpreted as markdown (formatted text) instead of python code, press **ESC then M** . To interpret it as code (default), press **ESC then Y**. The **ESC** key opens the "command" mode of the Jupyter notebook. If you press it in an active cell you will see that the vertical green bar on the left becomes blue. There are a number of useful commands when you are in that "blue mode"

- **A** add a cell above the current one
- **B** add a cell below the current one
- **M** markdown mode
- **Y** code mode
- **D-D** delete cell

Again, hit `esc + h` for the full list of shortcuts.

Try to add cells below and above with shortcuts, then delete them.

Scope in Jupyter

You can see a number next to every code cell. This shows the order in which they were executed. Generally, for readability and to avoid bugs, you'll want your notebooks to read top-to-bottom.

All that is going on is that there is a *kernel* running in the background that is being interacted with exactly like a normal python terminal session. For example, if you executed a code cell before and created a variable, it's still there for you to access when you execute another cell later.

In [4]:

```
# re-access the dictionary from earlier

my_dict['gender'][1]
```

Out[4]:

```
'F'
```

So can I do everything in Jupyter?

Pretty much yes. It's very useful for storing plots and notes in one place. This is very useful if you're executing code on a server which isn't local (viewing plots, or listening to audio can be a pain then). Here are some more examples for you to execute.

In [5]:

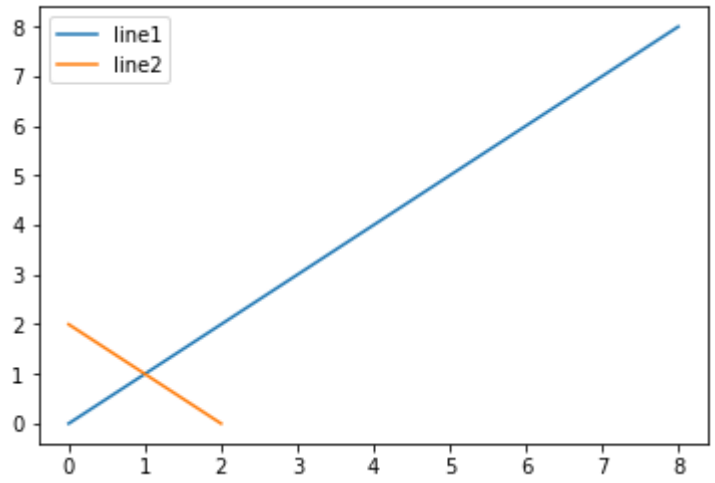
```
import matplotlib.pyplot as plt
import numpy as np
```

In [6]:

```
plt.plot(np.arange(9), label='line1')
plt.plot(np.arange(3)[::-1], label='line2')
plt.legend()
```

Out[6]:

```
<matplotlib.legend.Legend at 0x23fc4904c50>
```



In [7]:

```
import IPython.display
```

In [8]:

```
X = np.arange(16).reshape(4,4)
IPython.display.display([1,2,3])
print(X)
IPython.display.display(X)
IPython.display.display("you can call the display method manually")
```

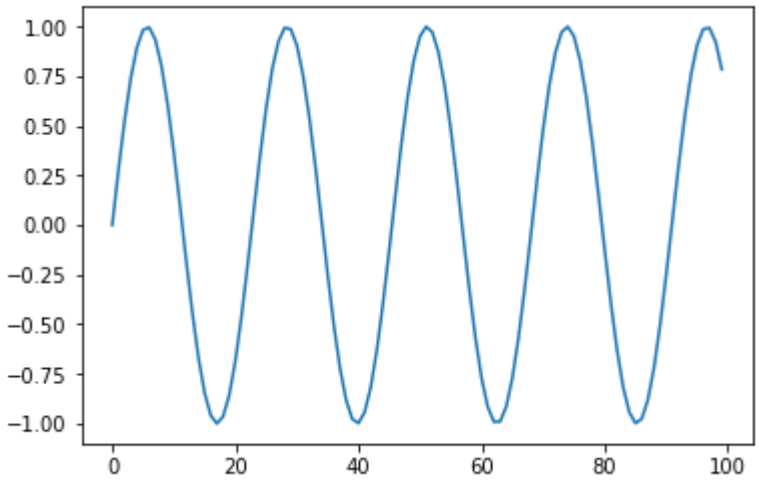
```
[1, 2, 3]

[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]

array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])

'you can call the display method manually'
```

In [9]:

```
# you can even display and listen to audio nicely
sample_rate = 10000
frequency = 440  # concert A
wave = np.sin(np.linspace(0, frequency, sample_rate)*2*np.pi)
plt.plot(wave[:100])
IPython.display.Audio(wave, rate=sample_rate)
```

Out[9]:

0:00 / 0:01



Getting help in Jupyter

Two extremely useful things to note in Jupyter:

- **TAB completion**, enter the name of a variable, function or method and press **TAB** for it to be autocompleted. Get used to it as it can make you a lot faster and avoid annoying typos.
- **help** and **?** to access documentation

```
help(print)
```

will display the documentation of the `print` function

```
?print
```

will do the same

In [10]:

```
cambridge_spark = "fun"
cambridge_spark2 = "more fun"
```

In [11]:

```
import pandas as pd

df = pd.DataFrame(data=[])
```

In [12]:

```
?df.append
```

In [13]:

```
# Get some help about the "len" function, do you remember what it does?
```

Error messages in Jupyter

A good thing to keep in mind is that when you see a big hairy error mesasge in Jupyter, you should *scroll to the bottom*, the **last line** is usually the one that tells you what went wrong!

In [14]:

```
blist = [1, 2, 3]
blist[2]
```

Out[14]:

```
3
```

Here the last line indicates clearly that we got an index wrong. The message is rather short so it's pretty easy but it's not uncommon to get a rather long error message and in that case it's useful to remember to check the bottom line first!

When things go wrong...

Sometimes you may have made a mistake or something went wrong and Jupyter gets blocked. You will usually note that instead of having a number next to a code cell you have a star: `[*]` meaning "it's computing". Now if it's a big computation, that's not an issue, but if you expected it to be rather quick then it may mean that the Jupyter session has crashed. If you think that's the case,

- click on "Kernel" in the menu
- click on "Restart"
- you will need to re-execute all cells up to the point you were