Jupyter Cheat Sheet

The following notebook is intended for use after reading the step listed below. The following tasks are designed to test your knowledge on what we have learned during the current step, make sure to read each section carefully and then edit the code cells to fit the requirements. You are encouraged to ask other students for peer feedback from the tutorials, this provides valuable insight and can show you different methods to complete the same task. Try to complete the tasks by yourself, but if you would like some help you can either: - Look back at the content for this step on Future Learn. - Ask another student how they completed the task. - Use Google to your advantage!

Introduction to Jupyter Notebook

This Notebook will give you a brief overview of the different markdown and code features that Jupyter notebook has to offer.

What is Jupyter Notebook?

Jupyter Notebook is a web application which allows you to create and share documents that contain live code, graphs, visualisations and narrative text which makes it a perfect medium for presenting analysis descriptions for data analysis.

The Kernel

The kernel is the program that runs the users code cells. You may have noticed it when you opened Jupyter Notebook, it is the command window that opens before your browser does. This initialises the local server on your machine which your browser connects to via the Jupyter Dashboard.

The Dashboard

The dashboard of Jupyter notebook gives you a visual user interface in which you can open notebooks and view your work. It can also be used to open a terminal into which you can send commands to your Kernel. Markdown cells I am a Markdown cell! I contain narrative text which helps explain code! You can double click me to edit my contents and press Ctrl+Enter to save my changes and apply formatting! Try double clicking this cell, change some text and apply it using Ctrl+Enter!

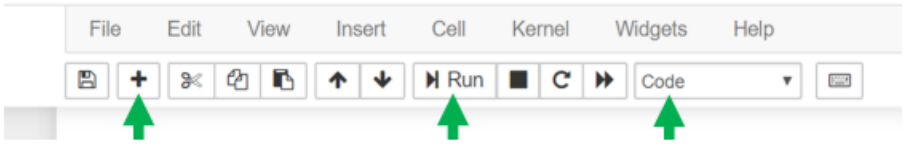
Code cells

The code in the cell below will run and output below it, when you select the cell and then press “Run” on the toolbar above, or if you press Ctrl+Enter.

```
[ ]: #I am a Python Code cell, I can run Python scripts!
     20+12
```

Making your own cell

You can make your own cell by clicking the “+” button, by clicking “Insert” on the toolbar above or pressing b on your keyboard. You can then change the cell type using the dropdown box on the right side of the toolbar. Or you can use y to change it to a code cell, or m to change it to a Markdown cell.



Try making your own cell below this one and running some basic Python code like 10+10. 2

Having more than 1 output

In some cases, we would like to see more than 1 value in the output for a cell. The cell below uses the default output of Jupyter Notebook

```
[ ]: 28+32
     5+10
```

See how only 1 output was shown, when we had two statements? To get around this we can use the print() function

```
[ ]: print(28+32)
     print(5+10)
```

Now both statements have been printed!

Keyboard Shortcuts

There are a bunch of handy keyboard shortcuts that allow you to speed up your workflow in Jupyter Notebook, here are a few examples: - Ctrl + Enter: Runs the

currently selected cell. - m: Converts the currently selected cell to Markdown. - y: Converts the currently selected cell to Code. - d + d: Deletes the currently selected cell. (Double tap the d key) - o: Toggles the Output of a code cell between Hidden and Shown. - b: Creates a new Cell. - 0 + 0: Resets the kernel (Resets all code variables and definitions)

- Shift + Tab: In code, will display information about a currently highlighted function. The following image shows what it looks like when we use the shortcut on the Image.open() function:
- Shift + M: Merges the currently selected cells.
- Shift + Left Click or Shift + Up/Down Arrow: Select/highlights multiple cells.

Markdown Formatting

Headers

To create a header, you can use a # symbol followed by a space and your header name. You can include up to six # symbols to make your headers smaller.

The largest header
The second largest header
The smallest header

Text Styling

You can style text in markdown cells through the use of special symbols. The table below shows some of the different combinations of symbols you can use to create these effects such as making 3 text Bold or Italic.

Style	Syntax	Example	Output
Bold	** ** or __ __	**Playtime**	**Playtime**
Italics	* * or _ _	_Ball games_	*Ball games*
Strikethrough	~~ ~~	~~Piano lessons~~	~~Piano lessons~~
Bold and italics	** ** and _ _	_**Dancing**_	***Dancing***

Text Quoting

You can quote text using the > symbol followed by a space, or you can have multiline quotes. For example:

> Two things are infinite: the universe and human stupidity;
>
> and I'm not sure about the universe.

Would produce:

> *Two things are infinite: the universe and human stupidity;*
>
> *and I'm not sure about the universe.*

Lists

You can make lists in markdown cells, they can be either ordered, unordered or task lists. You can have sublists by indenting your lists.

Un-Ordered List

An Un-ordered list is a list that uses bullet points, they are produced by either using * or - followed by a space. For example:

```
* Apples
* Bananas
    * Oranges
- Pears
```

Would produce:

- Apples
- Bananas
    - Oranges
- Pears

Ordered List

An Ordered list is a list that uses incrementing numbers for each member of the list and letters for each member of a sublist, they are produced by using 1. followed by a space. For example:

```
1. Apples
```

1. Bananas
    1. Oranges
    1. Pineapples
1. Pears

Would produce:

1. Apples
2. Bananas
    a. Oranges
    b. Pineapples
3. Pears

Task Lists

You can make task lists by using a - symbol followed by a [ ] to make an empty check box or a [x] to make a ticked checkbox. For example, a shopping list might look like this:

- [ ] Milk
- [ ] Biscuits
- [x] Chocolate
- [ ] Digestive
- [ ] Eggs
- [x] Bread
- [x] Sugar

Would produce:

☐ Milk
☐ Biscuits
☒ Chocolate
☐ Digestive
☐ Eggs
☒ Bread
☒ Sugar

Code Quoting

In Markdown cells you can quote code and even give it syntax highlighting, this helps narrate code without using a code cell. You can either use inline code quoting or multi-line code quoting.

Inline Code Quoting

Inline code quoting is useful for emphasising variable names or quoting functions, it is done by surrounding words with ‘ symbols. For example: The variable `fMeanAbsoluteError` is useful for... Would produce: The variable `fMeanAbsoluteError` is useful for…

Multi-Line Code Quoting

To quote multiple lines of code, you can either surround your code in three “‘ symbols or just use a code cell.

For example:

“’def SumOfList(lList, fThresh):
    return sum([fNumber for fNumber in lList if fNumber > fThresh])”’

Which displays as:

```
def SumOfList(lList, fThresh):
    return sum([fNumber for fNumber in lList if fNumber > fThresh])
```

Or you can add syntax highlighting by naming the programming language at the start of the quote like so:

“’python
def SumOfList(lList, fThresh):
return sum([fNumber for fNumber in lList if fNumber > fThresh])”’

Which displays as:

```python
def SumOfList(lList, fThresh):

    return sum([fNumber for fNumber in lList if fNumber > fThresh])
```

Quoting Equations

You can quote equations by surrounding it with *< /kbd > foraninlineequationorwith < kbd >$* to put the equation on its own line. For example:

What is the value of y when $x = 4$?
$$y= (x/3)^2 + 14$$

What is the value of y when $x = 4$?

$$y = (x/3)2 + 14$$

Links

You can apply a hyper-link to text using `[Text to add Link to](Link)`

For example:

```
Here is a [Link to Google](http://www.google.com)
```

Would produce:

Here is a Link to Google

Images

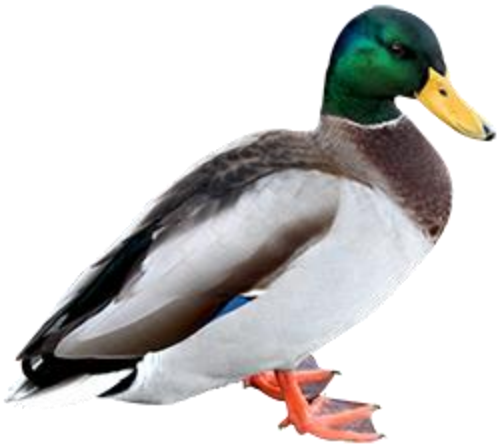
Adding images to Markdown

Images can be used in Markdown cells which can be used for a range of reasons like displaying graphs, emphasising a point, showing a logo etc. In the above section, one was used to show you the Jupyter Notebook toolbar.

The syntax for doing this is `![ImageTooltip](ImageLocation)`

- ImageTooltip is the text that displays when you hover over the image.
- Image location will be the location of the image in relevance to where your Jupyter Notebook file is stored. e.g. img/cat.png
    - We do not need quotation marks around the file path for markdown.

Running `![PictureOfDuck](img/duck.png)` will present you with:



Create a markdown cell below this one and load the example image located at img/cat.png

Coding

You can use code cells to code in Python, here are some examples to get you started.

Defining Variables

You can use code cells to define variables, and then use markdown cells to explain what they represent. The variables can then be used in cells below it!

```
[ ]: iNumber = 5
     fRational = 0.125
     bReady = False
     sName = "Dave"
```

After you have run that cell, you can then use those variables in proceeding cells

```
[ ]: fTestAddition = fRational + iNumber
     print(fTestAddition)
     print(bReady)
     print(sName)
```

Defining a Function

Similarly to variables, you can also use code cells to define functions, and then use markdown cells to explain what they do. The functions can then be used in cells below it!

This function returns the sum of all the values in a given list that are greater than the given threshold.

```
[ ]: def SumOfList(lList, fThresh):
         return sum([fNumber for fNumber in lList if fNumber > fThresh])
```

The function can then be used in proceeding cells.

```
[ ]: lNumbers = [20, 1, 204, 32, 2]
     SumOfList(lNumbers, 2)
```

Using If and else statements

You can use If and else statements in code cells, but they must be in the same cell, otherwise you will recieve the following error: So they must be in the same cell like so

```
[ ]: if(True):
         bReady = False
     else:
         bReady = True
```

Using Loops

Using loops is the same

```
[ ]: iIncrement = 0
     while(iIncrement < 20):
         print(iIncrement),
         iIncrement += 1;

     for num in range(iIncrement):
         print(iIncrement),
         iIncrement -= 1
```

Importing Modules

You can also import modules and libraries, which then can be accessed by proceeding cells.

```
[ ]: import math as MathLibrary
     print(MathLibrary.pi)
```

An example

This is a function that will return how many negatives are in the given list.

[5]:
```
def NegativeCount(a_lList):
    iNegativeCount = 0
    for number in a_lList:
        if number < 0:
            iNegativeCount += 1
    return iNegativeCount
```

This is a more concise, but less readable version of the function above that counts negatives in a given list.

[6]:
```
def NegativeCountShorter(a_lList):
    return sum([number < 0 for number in _NumberList])
```

We can then test these functions and print their output.

[7]:
```
_NumberList = [-4, -2, 3, 324, 2, -1343, -535]
NegativeCount(_NumberList)
```

[7]: 4

[8]:
```
NegativeCountShorter(_NumberList)
```

[8]: 4