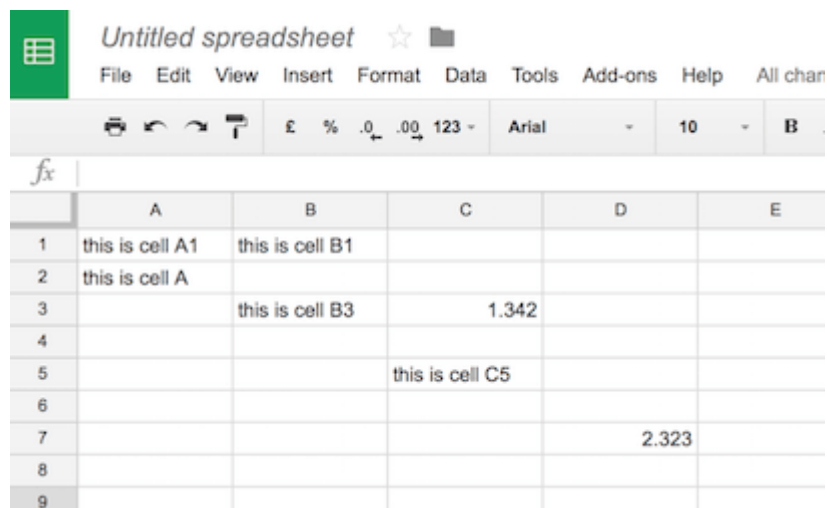


Arrays

Introduction to Arrays and their syntax in Python.

Arrays are just tables of values, and they come in really handy. Like tables, you refer to particular cells according to the row and column number. If you think of spreadsheets, you'll know that cells are referred to in this way, *B1* or *C5* for example, and the values in those cells can be used in calculations, *C3 + D7* for example.



The screenshot shows a Google Sheets interface with a menu bar (File, Edit, View, Insert, Format, Data, Tools, Add-ons, Help, All charts) and a toolbar with various icons. The spreadsheet grid has columns A through E and rows 1 through 9. The data is as follows:

	A	B	C	D	E
1	this is cell A1	this is cell B1			
2	this is cell A2				
3		this is cell B3	1.342		
4					
5			this is cell C5		
6					
7				2.323	
8					
9					

When we get to coding our neural network, we will use arrays to represent the matrices of input signals, weights and output signals too. And not just those, we'll use them to represent the signals inside the neural networks as they feed forward, and also the errors as they propagate backwards. So let's get familiar with them. Run the following code.

```
1 import numpy
```



What does this do? The import command tells Python to bring additional powers from elsewhere, to add new tools to its belt. Sometimes these additional tools are part of Python but aren't made immediately ready to use, to keep Python lean and mean, and only carry extra stuff if you intend to use it. Often these additional tools not core parts of Python, but are created by

others as useful extras, contributed for everyone to use. Here we've imported an additional set of tools packaged into a module named *numpy*. Numpy is really popular, and contains some useful stuff, like arrays and an ability to do calculations with them. Have a look at the following code:

```
1 import numpy
2
3 a = numpy.zeros( [3,2] )
4 print(a)
```

This uses the imported numpy module to create an array of shape 3 by 2, with all the cells set to the value zero and assigns the whole thing to a variable named *a*. We then print *a*. If you run the code you will get an array full of zeros which looks like a table with 3 rows and 2 columns. Now let's modify the contents of this array and change some of those zeros to other values. The following code shows how you can refer to specific cells to overwrite them with new values. It's just like referring to spreadsheet cells or street map grid references.

```
a[0,0] = 1
a[0,1] = 2
a[1,0] = 9
a[2,1] = 12
print(a)
```

The first line updates the cell at row zero and column zero with the value 1, overwriting whatever was there before. The other lines are similar updates, with a final printout with `print(a)`. The following shows us what the array looks like after these changes. Now that we know how to set the value of cells in an array, how do we look them up without printing out the entire array? We've been doing it already. We simply use the expressions like `a[1,2]` or `a[2,1]` to refer to the content of these cells which we can print or assign to other variables. The code shows us doing just this.

```
print(a[0,1])
v = a[1,0]
```

```
a[1,0]  
print(v)
```



You can see from the output that the first print instruction produced the value 2.0 which is what's inside the cell at $[0, 1]$. Next, the value inside `a[1,0]` is assigned to the variable `v` and this variable is printed. We get the expected 9.0 printed out. The column and row numbering start from 0 and not 1. The top left is at $[0, 0]$ not $[1, 1]$. This also means that the bottom right is at $[2, 1]$ not $[3, 2]$. This catches me out sometimes because I keep forgetting that many things in the computer world begin with 0 and not 1. If we tried to refer to `a[3,2]` we would get an error message telling us we were trying to locate a cell which didn't exist. We would get the same if we mixed up our columns and rows. Let's try accessing `a[0,2]` which doesn't exist just to see what error message is reported.

```
print(a[0,2])
```



Arrays, or matrices, will be used for neural networks because we can simplify the instructions to do the many many calculations for feeding signals forward and errors backward through a network. We saw this in the first part of this course.