Teaching notes: Python Part One

To remember

- check there is whiteboard / A-board
- Make a little slip with what will be covered
- check what was covered in bash: shell scripts? for loops?
- who are the helpers?
- who is teaching part 2?
- whiteboard pens / marker pens
- ipad for teaching notes
- laptop connector
- stickies

Intro & Outline

Hello, my name is Lucy. I'll be teaching the Python course this morning, along with the helpers x and y who can help to answer your questions. I've been programming for just just over a decade, though its really been over the last four years, whilst doing my PhD, that I've come to depend on Python for my day to day work - and I use Python because it's a great language if you want to just get stuff done - you can write useful programmes quite quickly with Python - hopefully you'll see that today.

But every programming language makes a trade off. So all languages balance how long it takes to write a programe, human time, against how long it takes that program to run, machine time. A programme written in C will run faster than Python - but it will take longer to write. Who was here for the unix shell lesson yesterday? A programme written with the unix shell will run faster - but writing a script for data analysis will take, in most cases, longer in unix than in python.

For the type of calculations and analysis I do, all my programmes run in seconds - so it doens't matter to me if Python takes longer - what matters to me is I can write the programmes quickly - which is one of the reasons I use Python.

Here are some other reasons to use Python:

- It is readable, and so its easier to learn than some other languages.
- It is free to use
- It is cross-platform - can work on Mac, Windows, linux
- It is widely used
- It is well documented

The last two are linked. Because there are a lot of people using it, there is a lot of information about Python online, lots of tutorials and lots of discussion forums with questions and answers. It's very rare that I'm stuck on something and can't find the answer online. Because it's a popular language, there are a lot of python programmes and scripts available online. These are collected together into Python packages. These are some of the most popular scientific python packages. So for example, you are doing astrophysics, you can download the astropy package and this will contain lots of functions - pieces of code - useful to your work.

So here is the outline for this morning

```
- writing and running python code
- variables
- data types
- functions, help and errors

 break

- lists
- for loops
- conditionals

 lunch
```

After lunch, x will take over to teach part 2 of the course. The overall aim of this course is to teach the basics of programming, with some examples using Python. What you learn here will be applicable to other languages

Writing and running Python code

There are few different approaches you can use to write and run python code. The plain text approach is where we use a text editor to write a programme. We save that programme with a .py extension to let the OS and us know that it is a python programme. We go to our terminal, and we run the programme. You probably did something similar in the unix lesson.

The second approach is to use a Jupyter notebook. We write and run the code in the notebook environment. Everything is done in a single place. We save as a .ipynb. This is what we will use to write and run our python code. Open your terminal - the one you used yesterday - and type

```
jupyter notebook
```

This should open a default web browser and open a server (don't close) - this is where all the work is happening. It is running locally on your machine: so although it opens a browser, it is not using the internet. This is the landing page, which you should all see on your computers.

If you have problems, please use the stickies

- Click on new notebook and select Python 3 - few different versions, this is the latest one and the one we will use.
- We can rename the notebook "Python_Workshop"
- We can enter code into a code cell: 2+3
- You can see it's a code cell because it says Python here.
- And we can run the cell
- The output is printed beneath
- You can add a comment using a hash "# some simple maths"
- And if we run the cell again, nothing changes - the notebook ignores anything after a hash.

There are lots of keyboard shortcuts. The one I always use is shift and enter to run the cell. Click on the cell, press shift and enter at the same time, and it will run.

- You can bring up the other shortcuts in the following way.
- Click onto the cell and press escape.
- The cell changes to blue. This is command mode - when you type nothing will appear in the cell.
- H to bring up shortcut options. You can add new cells, delete cells using shortcuts - I need to get better at using these myself.

We have entered code so far. But you can also enter markdown - format for writing plain text.
Switch to markdown.

- Now when you write something and run it, it will appear as plain text:
- You can add headings:
- And you can add bullet points
- You can add links
- And you can use markdown to insert images

```
# This is a heading
This is plain text

- item one
- item two

[This is a link to XKCD](http://xkcd.com)

!\[](http://bit.ly/python_cat)
```

You cant have markdown and python in same cell.

On whiteboard write basic markdown syntax

fact that you get comments, pictures, and code in one place is very useful.

Task (5 minutes): use your notebook to: a) link to the Imperial webpage; b) calculate 34226/359; c) make a bullet pointed shopping list with heading "shopping list"

Variables and Assignment

- Variables are names given to values
- In python, the = symbol assigns the value on the right to the name on the left
- Variables contain letters, digits, _
- Variables cannot start with a digit, and variables which start with a _ have a special meaning so at this point we won't use them.
- Python is case sensitive

Type after me

```
age = 73
first_name = 'Clarence'
```

- Age is a variable to which we assign the integer value 42.
- First name is a variable to which we assign the string value Clarence.
- Python has a built in function called print that prints its function arguments as text.

```
print(first_name, 'is', age, 'years old')
```

- This is the function name `print`
- These are the function arguments which will be printed

On whiteboard write function_name(arguments)

- First name and age are variables, so we don't need to put quotes around them. 'is' and 'years old' are strings and will be printed as they appear - we know this because they have quotes around them.
- Run the cell with shift and enter
- The output is printed beneath

```
print(last_name)
```

- We get an error message. the first of many! These can be useful, it tells us the variable last_name has not been defined yet.
- We can add

```
last_name = 'Barlow'
```

- And now there is no error message.
- I can use tab completion. Instead of writing `last_name` I can write `l` then press tab and it will auto-complete.
- Notice that I have assigned the variable in this cell. And used the variable in another cell. That's fine: the variables are saved for use in the whole of this notebook.
- However, you need to be careful because it is the order of code execution that matters - not the order that the code is written in. And this can cause problems - in fact, when I talk to programmers, I'd say this is there number one complaint about Jupyter Notebooks. For example

```
print(middle_name)
```

```
middle_name = Dorothy
```

- If I run the top then the bottom I get an error
- If I run the top again, the error has gone. The first time I asked to print the variable `middle_name` it had not been assigned a value, it had not been created, so I got an error. I then assigned a value, this was stored, so when I run the cell again it works. This can cause confusion because I've not changed the code, but the output has changed. We do need to be careful. If you get confused as to the value of different variables. I suggest to click `restart and run all` : it will run everything from the top, so you can see errors, and fix them.

- The final thing I will show you in this section is that variables can be used in calculations:

```
age=42
future_age=age+3
print('Age in three years',future_age)
```

- Python doesn't care what variable names you use but you should make them meaningful - this will make it easier for other people, or your future self, to work out how the programme works.
- I can change the variable to sausages and I'll get the same output. But sausages in this context is meaningless so its an example of a bad variable name.

Do the swapping values task

Data Types and Type Conversion

Each value in Python has a specific type.
`integer - (int) - positive or negative whole numbers - 256 -3 float - (float) - real number - 3.16436 -0.53 string - str - alphanumeric`

These are the ones we will cover today.

- Integers and floats are numeric types - they represent numbers - they can be positive or negative.
- String and List are sequence types - they are ordered sets of elements
- Boolean (true or false) and dictionaries are also very useful but we don't have time to cover today. I suggest to look them up when you have time to though.

```
print(type(52))
```

I am going to nest two functions together. I am asking the python interpreter to print the type of the value 52. Its an int.

```
fitness = 'not bad'
print(type(fitness))
```

- This tells me that the value assigned to the variable fitness is a string.
- Types are important because they determine which operations can be performed on a value.
- For example, I can subtract two integers

```
print(5-3)
```

- But I can't subtract two strings

```
print('hello'-'h')
```

Here are the operations you can do with strings.

- You can add two strings together
- You can multiply a string with an integer. The string will repeat 10 times.

```
full_name = "Lucy" + "Geoty" ,print(full_name)
separator= "-"*10, print(seperator)
```

The order of a string matters. `gold` does not equal `dlog` . Because strings are ordered, you can index a string.

Following moving between the board and projector

- `metal='gold'`
- each position in the string is given a number
- this number is called an index.
- indexing is from zero!
- If I want to return the first character of a string I write variable name then the index in square brackets.

```
print(metal[0])
```

- indexing from 0!!!!
- we can also slice the string. this means take a slice, a piece of it. for example, we can slice gold to get old. to do this we use `[start:stop]` - `start` is where we start the slice, `stop` is the index one after where we want to stop. Easiest is with an example:

```
print(metal[1:4])
```

- Stop is at 4, not 3

```
print(metal)
```

- The variable metal is unchanged - still gold. The slice makes a copy, it doens't replace the original.
- indexing, slicing - we can do this because we are treating the string like a list of letters, and this is something we can do to any list - we'll see more of it after the break.
- some functions only work with strings:

```
print(len(full_name))
print(len(52))
```

This gives length of the string, whilst the length of a number gives an error

Question: what do you think will happen when I run this command?

```
print(1+'2')
```

You cant add a string and an int. You can do this - convert a string to an int or an int to a string.

```
print(1 + int('2'))
print(str(1)+'2')
```

We are able to mix float and ints

```
1 / 2.0
```

Discuss what you think this will print..

```
first = 1
second = 5\*first
first=2
print('first is', first, 'and second is', second)
```

key point: second does not update automatically when you change the value of first*

Functions, help and errors

- We're now going to cover a few miscellaneous things before breaking for coffee: functions, help and errors.
- We've come across several built-in functions already. Built in because they are built into the core Python language.
- *write on board* `len, int, str, float, print`
- all take an argument, all need parantheses so python knows a function is being called: `function_name(arguments)
- here are a couple more: the last one doesn't work as you can't compare a number and a float

```
print(max(3,6,7))
print(min(2,7,8))
print(max(1,'a'))
```

`round` rounds to zero decimal places, but there is an optional argument you can use

```
round(3.712)
round(3.712, 1)
```

- but you may ask, how would I know about this optional arguement?
- well you can use `help(round)` and this gives documentation
- In a Jupyter notebook you can also use `help`

```
help(round)
round?
```

- Lets move onto errors.
- There are two types: runtime errors and syntax errors.
- Runtime errors, also known as exceptions, happen whilst the python code is executing. For example:

```
age=65
print(aege)
```

This is an exception : specifically the NameError, aege is not defined - if I correct it, it will run.

```
name=lucy
print(name[4])
```

So here is a runtime error. IndexError - attempted to access a value that doesn't exist. Remember, indexing from 0!

```
print(age
```

Now this is a syntax error, I haven't closed the bracket: the program can't be parsed and won't even run.

Recap

Let's quickly recap what we have covered so far:

```
- writing and running python code: Jupyter Notebooks, markdown basics
- variables: variable names, variable assignment, print(), execution order
- data types: integer, float, string, string operations/indexing/slicing, type conversion: int(), float()
- functions, help and errors: min(),max(),round(),help(),runtime errors (exceptions), syntax errors,

 break

 - lists
 - for loops
 - conditionals

 lunch
```

After the break, we return to look at lists.

---

coffee break

Lists

```
Integer - positive or negative whole numbers (int)
float - real number like 3.16436 or -0.53 (float)
string - this is a type (str) text in single or double quotes
List - a list of one or more values [3,4,5] or ['frog',2,8], [].
```

- We've covered the first three.
- We've not looked at list:
- A list has square brackets and the elements of the list are separated by commas.
- It can store many values in a single structure.
- It can be an empty list.
- We discussed because a string is an ordered list of letters, you can index and slice. So we can do that for lists too, and we'll build on that now.

```
pressures = [0.3255,0.2462,0.2352]
print('pressures: ', pressures)
print('length: ', len(pressure))
print('last pressure in list: ', pressures[2])
print('last pressure in list: ', pressures[-1])
```

- You can print lists, get the length of lists, and access the final element it two different ways. `-1` indexes the final element of a list in python.
- If I want to modify an item in the list I can do

```
pressures[0] = 0.001
print ('pressure is now: ',pressures)
```

- I've replaced the first item of the list with 0.001
- I can try and do this for a string

```
name="lucy"
lucy[0] = b
```

- And I get an error. But I told you that a string is a list of characters: so why can I replace the first item of a list, but not replace the first item of a string?
- Because string characters are immutable. This means they can't be changed after creation. But lists are mutable, the items of a list can be modified.
- Each Python type has a set of methods which can be used. For example:

```
pressures.append(0.252)
print ('longer pressure list: ', pressures)
```

- Here I've used the list method `append` to append a value onto the end of the list.

```
age = 32
age.append(14)
```

What do you think will happen here?

An error - `append` is a method for a list. This is an integer, it doens't have the append method.

```
pressures.append([0.1355])
print ('longest pressure list: ', pressures)
```

- We now have a list within our lists: it is a 2-dimensional list.
- If I want to remove the last item I can use

```
del pressures[-1]
```

Which removes the last item.

For Loops

A for loop is used to execute the same command multiple times.

```

for number in [2,3,5]:
    print(number)

same as

print(2)
print(3)
print(5)
```

- It's a bit of a silly example but you can see how this could be useful if you wanted, for example, to do the same piece of analysis on several sets of data.
- You've seen for loops in the bash course yesterday, so the concept is the same:
  - all for loops have a sequence [2,3,5]
  - a loop variable
  - a body

- We loop through the sequence. first time, number=2, second time number=3,..
- The syntax is a little different from bash
  - first line ends in colon
  - body indented : indentation is important in python

```
for number in [2,3,5]:
print(number)

for number in [2,3,5]
    print (number)

for number in [2,3,5]:
    print(number)
```

- You can change this loop variable to anything - its a dummy variable. For example:

```
for cat in [2,3,5]:
    print (cat)
```

- This gives the same output. Of course cat is a silly variable name in this instance, should use something meaningful, like number.
- We can add multiple statements to the loop body

```
for number in [2,3,4]:
    squared = number\*2
    cubed = number\*3
    print(number, squared, cubed)
```

- But what about if I wanted to loop over a large number of numbers? Use `range`

```
for number in range(0,5):
    print(number)
```

- `range` is not a list, it is an iterable - you can interate over it. It is more efficient to loop over range than a list of integers.

```
for number in range(5):
    print(number)
```

-By default, if one argument is given to the `range` function then the first value is set to zero - it will start at 0.

Question: I want to sum the first 10 integers. What is wrong with this code? How can I fix it (there is more than one way)

```
total = 0
for number in range(10):
    total = total + number
print(total)
```

- It will add 0 to 9, not 1 to 10.
- There are two ways to fix this. `range(11)` or `range(1,11)` or `number+1`.

Conditionals

- This is the final section!
- `if` statements are used to control whether a block of code is executed
- The structure is similar to a `for` loop. First line opens with if and ends in colon.
- The body which can contain one or more statements is indented
- Mass is 4.2. If it's more than 3 it will print "4.2 is large"
- If it's between 2 and 3 inclusive it will print "4.2 just right"

```
mass = 4.2

if mass > 3:
    print(mass, ' is large')

if mass < 2:
    print(mass, ' is small')

if 2 <= mass <= 3:  (check this allowed!)
    print(mass, ' is just right')
```

- We can combine conditionals and a `for` loop.
- For each mass it will test if the condition is met. If so, it will print.
- We can add an else statement: this executes a block of code when an if condition is not true. `else` must come after `if`
- We can add an elif to specify additional tests. `elif` must come after an `if` and before any `else` statement.

```
masses = [2.4,2.3,7.4,1.4,5.6]
for m in masses:
    if m>3.0:
        print(m, 'is large')
    elif m < 2.0:
        print(m, 'is small')
    else:
        print(m, ' is just right')
```

- The order matters. Once a condition is met python will not test any of the following conditions.
- For example, if you were teaching an undergraduate class a mistake like this could be disasttrous for you students

```
grade = 95

if grade >= 70:
    print ("grade is C")
elif grade >= 80:
    print("grae is B")
elif grade >= 90:
    print("grade is A")
```

Question: what is wrong with the code as it is written? Use your Jupyter notebook to investigate. Re-write it so that it works as intented

Correct answer:

```
grade=95

if grade >= 90:
    print("grade is A")
elif grade >= 80:
    print("grae is B")
elif grade >= 70:
    print ("grade is C")
```

Recap and closing comments

This is what we have covered since coffee:

```
- writing and running python code: Jupyter Notebooks, markdown basics
- variables: variable names, variable assignment, print(), execution order
- data types: integer, float, string, string operations/indexing/slicing, type conversion: int(), float()
- functions, help and errors: min(),max(),round(),help(),runtime errors (exceptions), syntax errors

break

 - lists: sequence type, immutable vs mutable, list method append, del
 - for loops: dummy variable, loop syntax, index from 0
 - conditionals: if, elif, else, ordering

lunch
```

What I've covered this morning follows the one online. So if you want to recap, go to this link, I've followed it almost teaching. Return at 2pm for Python part 2, which x is teaching.

Before we break I'd like to make five closing comments:

```
- First: comment your code. It may be boring, you may not feel like you have the time - but trust me, your future self will thank you for i
t. Document why you have written something a certain way.

- Second: use version control. Yesterday you learnt how to use git. I use git to version control all of my code. That way, If I write somet
hing and it breaks the code, I can easily go back to an earlier version of the code - again, some time investment now will save you time in
 the future.

- Third: care about reproducibility. One of the cornerstones of science, but it is still very hard to reproduce scientific results. You'll
see this afternoon how you can use Jupyter notebooks to import and analyse data. This can then be published alongside an academic paper so
that people can reproduce your results. I've done this, and I know people have used this.

- Fourth: stay confident. Even though Python is "easy" it can still feel pretty hard sometimes. And sometimes it can feel like you are the
only one who finds it difficult - it can get demoralising. All I can say is stick with it, and realise that most of the people around you a
re probably finding it just as difficult.
```