

Teaching notes: Python Part One

To remember

- think about internet
- Change Keyboard! or use headphone/microphone
- check with Cai - questions, monitoring, note taking..have an instructor to instructor chat.
- ASK FOR FEEDBACK at end
- check Binder running ok

Setup

- Teams for the call
- etherpad for notes, questions, links: <https://etherpad.wikimedia.org/p/cdt-renu>
- directpoll for polling: <https://www.directpoll.com/c?XDVhEt2FQnm1qDVqy6t8Hp1pNW2T1N#opt0> is the admin page (have it ready)
- microsoft forms for post-workshop quiz:
- jitsi for break out
- online notebook for people have problems running locally:
https://mybinder.org/v2/gh/lucydot/empty_binder.git/HEAD
- so student will have: Me on Teams, their notebook webpage on as standard. Then the etherpad for notes, and the quiz webpage for the quiz.

Intro & Outline

- Paste link to: etherpad, quiz, website and slides.
- Hi! I'm Lucy
- In this workshop we are going to...

Motivation

Why not excel? It is easy to make mistakes using excel. JP Morgan - error on excel spreadsheet led to the loss of 4.6billion pounds. The problem with excel is that you cannot test an excel document. And you can overwrite numbers by mistake. And people make mistakes.

The second is that by taking a programmatic approach to data analysis and plotting you are ensuring the

reproducibility of your science. If you write code to analyse data, and crucially if you share that code, you are ensuring that your analysis can be easily reproduced by other scientists. They can verify your work. As an example, take a paper I published with a colleague a few years ago. So we did the research, published our method and results in a paper, and alongside this we gathered our raw data and the scripts we used to analyse that data into an online repository, which we also published. The vast majority of standalone journal papers do not include enough details for you to reproduce the work. So supplementing each one with analysis scripts is a relatively straight forward way to increase the reproducibility of your work.

The third reason is repeatability. This is about efficiency. Once you have a script working for one set of data, you can take the data from your next experimental run and process it with the same script at no additional time cost. You can share it with colleagues.

Now I've been talking about why a general programmatic approach is important. But why Python in particular?

First of all, Python gives you wings. You can do a lot with a little bit of code. That is because, unlike a language like C or C++, you can easily import code that other people have written and use that in your programmes. For instance, if I want to do a weighted least squares fit of a 5th order polynomial to a set of data, I can do this in a single line of code.

Second, when it comes to plotting, Python is capable of producing beautiful publication ready plots. For my papers, I always use Python - it looks professional, and there is a flexible plotting library called matplotlib. And because I write scripts for each plot, if I want to re-plot a new set of data I can do this in a single command.

So I'll start by emphasising that every programming language makes a trade off. So all languages balance how long it takes to write a programme, human time, against how long it takes that program to run, machine time. For example, a programme written in the programming language C will run faster than Python. For the type of calculations and analysis I do, all my programmes run in seconds - so it doesn'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 - scientific programming, data science, machine learning, cybersecurity. Indeed.com and stack overflow - top for job listings and for average salary on offer. - 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.

Outline

So here is the outline for this morning

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

break (30 mins)

- lists
- for loops
- conditionals

finish @ 1pm

Half an hour break in the middle. 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.

*Can I re-explain or clarify anything for you? *

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.

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 / git bash 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 raise your hand

Setup

- Arrange your screen so you can see my screen and your Jupyter Notebook at the same time
- At some points during the training you will also need to access:

- etherpad: add your name to the etherpad (at the top and in the doc)

If you've got a question or comment raise your hand. Cai will be monitoring the chat. Cai, if there are several people and we need to pause then let me know. Use the emojis to give me feedback! especially thumbs up Stay on mute unless your speaking. Let me know if I've forgotten to unmute etc.

- set students up with jitsi (even numbers create a jitsi, and give this link to the odd number before them)

Writing and running Python code (ctd)

- 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)


```

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 predicting 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 text - "hello" "20 pence."
List - a list of one or more values [3,4,5] or ['frog',2,8]
+ boolean, dict, tuple, complex, None, set
```

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 doesn'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 doesn'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 iterate 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 disastrous 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 intended

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

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, methods, append, del
- for loops: dummy variable, loop syntax, index from 0
- conditionals: if, elif, else, ordering

lunch

Return tomorrow for Python part 2, where we will be bringing all this together

Outline

- functions:
- variable scope:
- libraries:
- cleaning data with pandas:

break

- analysing data with numpy:
- plotting data with matplotlib:
- running code as a Python script:
- programming good practice:

Functions

- We can only keep so much in our mind at one time
- Most of us will have had the experience of breaking a complicated problem down into smaller parts - divide and conquer approach.
- Same thing exists in programming: we write functions which are units of code.
- We've seen functions already. For example the function `max` which returns the maximum of a list. `max` is an in-built python function - it comes installed with python - but we can also write our own functions.
- Dividing things into units also means we avoid repetition - we can re-use and re-combine our units in different ways instead of typing the same thing many times.

hand out an example, with and without functions

- Begin the definition of a new function with `def`
- Followed by the name of the function (same rules as variable names)
- parameters in parentheses
- these could be empty
- there is a colon and indented block of code

```
def print_greeting():  
    print('Hello!')
```

- I've defined the function but I haven't run it.
- Need to call the function to execute the code it contains

```
print_greeting()
```

- Functions are more useful when they can operate on different data.

- Specify parameters when defining a function. These parameters are automatically assigned a value when the function is called.

```
def print_greeting(name,appearance):  
    print('Hello ',name,' you look ',appearance)  
  
print_greeting("Lucy","excellent")
```

- When I call print_greeting, the parameter `name` is assigned the value `Lucy`.
- the order matters

```
def print_greeting(name,appearance):  
    print('Hello ',name,' you look ',appearance)  
  
print_greeting("excellent","Lucy")
```

- to avoid this you can name the parameters when we call the function, and then you can specify in any order

```
def print_greeting(name,appearance):  
    print('Hello ',name,' you look ',appearance)  
  
print_greeting(appearance="excellent",name="excellent")
```

- Functions can return a result to their caller using return
- This is a good example of a function you shouldn't write because numpy has this average function already defined in it's library. But for pedagogy's sake...

```
def average(values):  
    return sum(values) / len(values)  
  
print ('average of list:',average([5,2,6]))
```

- We don't need to put return at the end. We could put a return at the beginning to handle special cases

```
def average(values):  
  
    if len(values) == 0:  
        return None  
  
    return sum(values) / len(values)  
  
print ('average of empty list:', average([]))
```

- In fact, every function returns something. If you don't explicitly include a return statement, your function

will return `None` . `print_greeting()` will return `None` .

```
result = print_greeting("Lucy","excited")
print(result)
```

- could mention here the difference between an `argument` and a `parameter`

Task: Find the First

Fill in the blanks to create a function that takes a list of numbers as an argument and returns the first negative value in the list.

```
def first_negative(values):
    for v in ____:
        if ____:
            return ____
```

- What does your function do if the list has only positive numbers?

Variable Scope

- The scope of a variable is the part of a program that can 'see' that variable.
- `pressure` is a global variable: defined outside the function, can be seen everywhere
- `t` and `temperature` are local variables in `adjust`: defined in the function, not visible in the main program

```
pressure = 103.9

def adjust(t):
    temperature = t * 1.43 / pressure
```

```
pressure = 103.9

def adjust(t):
    temperature = t * 1.43 / pressure

adjust(0.9)
print('temperature after call:', temperature)
```

Question: how can I adapt the code so that I can access the adjusted temperature?

Python Libraries

- I mentioned at the start of the course that Python is a popular course, so that there are a lot of Python code already written and 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.

- Related Functions are collected together in a file called a module.
- A module can also contain data values (numerical constants for example).
- Modules correspond to a single file.
- Related Modules are collected together in a package
- Related packages and modules are collected together in a library
- Package and Library are often used interchangeably; in Python a library is not strictly defined.
- python standard library comes with python itself
- additional libraries are available via the python package index (show python package index)
- we use import to add a library or module into a program's memory
- lets import the math module which is in the Python standard library

```
import math

print(math.pi)
print(math.cos(0))
```

- We can then access the constant `pi`, we need to put the library name before it.
- A good analogy for this is family names. My name is `Lucy Whalley` - I, Lucy, belong to the family Whalley. So we could use the dot notation `Whalley.Lucy` to refer to me using this structure.
- We can access help

```
help(math)
```

- We can create an alias for a library module to reduce typing

```
import math as m

print(m.cos(0))
```

- To reduce typing further we could import specific items from a library module

```
from math import pi

print(pi)
print(math.cos(0))
```

- Note we don't have use of the `math.cos` because we haven't imported all of the math module, only the `pi` constant from the math module.

Question: You want to select a random character from a string. ``` bases = 'ATCHAGHRASG'

1. which standard library module could help you?
2. which function could you select from that module?
3. try to write a program that uses that function ``

Recap

- Ok now we have all the basic building blocks in place

```
- functions: function syntax, return statement, parameters and arguments
- variable scope: local and global variables
- libraries: modules, packages, libraries, import statements, aliases
- cleaning data with pandas:
```

```
break
```

```
- analysing data with numpy:
- plotting data with matplotlib:
- running code as a Python script:
- programming good practice:
```

Reading and cleaning data with Pandas

- We are now going to work with the UV-Vis data until the end of the session.
- It should be downloaded/moved to your desktop
- Start a new notebook with the name "UV-Vis_example"
- Thumbs up when you are done
- First we need to read it in. There are a couple of libraries which could do this for us. We will use the `pandas` library which is a popular library for data analysis.

```
import pandas
```

- Now that we have imported it we can call the `read_csv` function in the library

```
pandas.read_csv("./desktop/UVVis-01.csv")
```

- Remember that it is important to put `pandas.` before `read_csv`
- The `read_csv` function has one argument: the name of the file we want to read.
- This data is from a UV-Vis experiment. The rows are the data for each wavelength, and the columns are the individual samples. **sketch on board**
- This is in a pandas dataframe. A pandas dataframe is 2D data structure with columns of potentially different types. We are going to analyse the data, but first we need to clear up this dataframe. There are 3 things I want to do:
 - delete repeated wavelength column

- set column headings (pandas has assumed the first row of the data is a column heading)
- transpose the data: swap rows and columns
- we can do the first two things with the following command

```
pandas.read_csv("./Desktop/UVVis-01.csv",usecols=[1,3,5,7,9,11,13,15,17,19],header=None)
```

- We've passed two additional arguments to pandas: The usecols keyword which specifies the columns to read in and the header keyword which, when we set to False, tells the read_csv function that there is no heading data in the file and that the headers should be set to an integer range.
- Ugh, the usecols list explicitly typed out is ugly. Is there a nicer way of writing this list?

```
columns_tokeep = [0]+list(range(1,21,2))
```

- I've done two things here. Included a step in the range function, and converted from iterable to a list. Then added them together.
- Our call to pandas.read_csv read our file but didn't save the data in memory. To do that, we need to assign the array to a variable. Just as we can assign a single value to a variable, we can also assign an array of values to a variable using the same syntax. Let's re-run `pandas.read_csv` and save the returned data:

```
data = pandas.read_csv("./Desktop/UVVis-01.csv",usecols=columns_tokeep,header=None)
```

- This statement doesn't produce any output because we've assigned the output to the variable data. If we want to check that the data have been loaded, we can print the variable's value:

```
print(data)
```

- We can have a look and inspect this data type

```
type(data)
```

- The output tells us that data currently refers to an pandas DataFrame.
- what is a pandas dataframe? It is a 2D data structure. It can hold heterogeneous data (in each column the datatype is the same, but across columns it can be different). It can handle missing values - NaNs nicely. The axes are labelled. The rows are given index labels. And the columns are given column names.
- We can see that it's in this frame module. We can have a look at the docstring for dataframe, or for the frame module

```
pandas.core.frame.DataFrame?
```

- Now that the data are in memory, we can manipulate them.
- We can now use the transpose method to swap the rows and columns of data, and assign this to the variable data.

```
data = data.transpose()  
print(data)
```

- Now it makes sense to set this first row as our header. To do this we need to know how to access the first row. We do this with the `loc` (location) command. This selects the row with the label 0.

```
new_header = data.loc[0]  
new_header
```

- We then take the dataframe but without the header row. We do this by slicing. We can slice a dataframe. This slices the rows. The empty after colon means slice to the end.

```
data = data.loc[1:]
```

- So this is our dataframe without the first row. Now we need to add our new header to this

```
data.columns = new_header
```

- Now our data set is nice and clean and ready to analyse.
- The final thing left to do is print our cleaned dataset to a file for analysing later.
- To do this we can use the `to_csv` DataFrame method. We set the header to True, but the index parameters to False.

```
data.to_csv('./Desktop/UVVis-01-cleaned.csv', header=True, index=False)
```

- So we've now saved a file containing a cleaned version of our UV-Vis data. If we'd finished work for the day we could save it and look at it later. Note that this is much better than deleting columns or rows by hand in an excel or similar. You have a copy of the raw data and a copy of the notebook used to manipulate it, so you can go back and see exactly what you've done. You have a record of your steps.

Analysing data with Pandas

- Let's read in the data we saved. Remember columns are wavelengths, rows are individual samples.

```
data = pandas.read_csv("./Desktop/UVVis-01-cleaned.csv")
```

- So we have seen how to slice the rows. We do that with

```
data.loc[3:8]
```

- This selects the third to 8th row **inclusive**. We can also slice rows and columns. We do this with

```
data.loc[3:8, '1500.0': '1492.0']
```

- The rows we slice first, then a comma, then the columns we want to slice. The column headers are strings, that is why they have to be in quotes.
- This could get annoying so we can use the map method to map them all to floats.

```
data.columns.map(float)
data.columns = data.columns.map(float)
```

In fact, they are all integers so map them to integers

```
data.columns.map(int)
data.columns = data.columns.map(int)
```

- Now we can slice with integers

```
data.loc[3:8, 1500:1492]
```

- The key thing here is that we index or slice the rows first (before the comma), then the columns (after the comma).
- Task: Crack the code!
- We can also perform common mathematical operations across all values in a column or row. Perhaps you find out the machine wasn't calibrated correctly and you need to add 0.007 to all values. This is easy

```
data = data + 0.007
data += 0.007
```

- To add 0.0004 to the values of a given row we do

```
data.loc[7] = data.loc[7]+0.0004
data.loc[7] += 0.0004
```

- To add 0.0004 to the values of a given column we do

```
data.loc[:,1495] = data.loc[:,1495]+0.0004  
data.loc[:,1495] += 0.0004
```

- Pandas can also do more complex operations like calculating the mean across the rows

```
print(data.mean(axis=0))
```

-data is a dataframe type. Dataframe types have the `mean` method. Specifying `axis=0`, the zeroth axis is rows so its saying calculate the mean over the rows. I could do the mean over the columns

```
print(data.mean(axis=1))
```

- To see all the dataframe methods we type `data.` then press tab
- For this data it is more likely that we'd want to calculate something like the maximum absorption per sample. One way to do this is to create a new temporary array of the data we want

```
sample_0 = data.loc[0] # this selects the first row (first sample)  
sample_0.max() # maximum of the first sample
```

- If we wanted to calculate the maximum absorption per sample for every sample, not just for the first sample, we can use

```
print(data.max(axis=1))
```

- `axis=1` tells pandas to find the maximum of the axis 1, which is the columns.

*Question: what command would be use to calculate the minimum absorption at each wavelength?

```
print(data.min(axis=0))
```

Visualizing data with matplotlib

- There is no official python plotting library, but matplotlib is by far the most widely used. It's great! You can make a range of beautiful plots (show examples and link to online gallery)
- Let's import the library

```
%matplotlib inline # this allows images to appear in the notebook when I use the show() command  
  
import matplotlib.pyplot
```

- It's now straightforward to plot the average absorption data across all wavelengths


```
ave_absorption = data.mean(axis=0)
ave_absorption # this contains wavelengths and average absorption
```

```
ave_plot = matplotlib.pyplot.plot(ave_absorption)
matplotlib.pyplot.show()
```

- We can add labels to the axes

```
ave_plot = matplotlib.pyplot.plot(ave_absorption)
matplotlib.pyplot.xlabel("wavelength")
matplotlib.pyplot.ylabel("ave. absorption")
matplotlib.pyplot.show()
```

- We are interested in analysing a sub-set of the data, from wavelengths 300 to 1000.

```
ave_plot = matplotlib.pyplot.plot(ave_absorption.loc[1000:300])
matplotlib.pyplot.xlabel("wavelength")
matplotlib.pyplot.ylabel("ave. absorption")
matplotlib.pyplot.show()
```

- Note the order - remember these are not indices, but column headings
- It's easy to copy this code and adapt to calculate two other statistics.

```
max_absorption = data.max(axis=0)
min_absorption = data.min(axis=0)

ave_plot = matplotlib.pyplot.plot(max_absorption)
matplotlib.pyplot.xlabel("wavelength")
matplotlib.pyplot.ylabel("max. absorption")
matplotlib.pyplot.show()
```

```
ave_plot = matplotlib.pyplot.plot(min_absorption)
matplotlib.pyplot.xlabel("wavelength")
matplotlib.pyplot.ylabel("min. absorption")
matplotlib.pyplot.show()
```

- If I want to save this plot, I use the `savefig` method

```
ave_plot = matplotlib.pyplot.plot(min_absorption)
matplotlib.pyplot.xlabel("wavelength")
matplotlib.pyplot.ylabel("min. absorption")
matplotlib.pyplot.savefig("./min_absorption_plot.png")
```

Running your Code as a Python Script

- If you wanted to share this script with other researchers you may want to export this code cell as a plain text file.
- It can then be run from a terminal rather than opening up Jupyter Notebooks.
- This might be useful if you are trying to automate your workflow
- To convert the code in your Jupyter Notebook cell into a plain text .py file we can Use the Jupyter Magic command `%%writefile myfile.py` in your code block. first of all we need to include all the code in a single cell. and then we need to add the writefile command

```
%%writefile absorption_plot.py

import pandas
import matplotlib.pyplot as plt

data = pandas.read_csv("./desktop/UVVis-01-cleaned.csv")
data.columns = data.columns.map(float).map(int)

min_absorption = data.min(axis=0)
min_plot = matplotlib.pyplot.plot(min_absorption)
plt.xlabel("wavelength")
plt.ylabel("min. absorption")
plt.savefig("./min_absorption_plot.png")
```

- This will create a file with the name `absorption_plots.py`
- You can open it up to see it
- We can run it with

```
python3 absorption_plots.py
```

- No news is good news
- And the `.png` file has been created
- The data files that are read in by the script are currently hard coded - they cannot be changed without changing the code itself.
- Instead, we can import the `sys` library which allows us to provide command line arguments specifying the filenames for our data.
- The first command line argument is accessed with the variable `sys.argv[1]` , the second command line argument is accessed with the `sys.argv[2]` argument and so on.

- We can adapt the `absorption_plots.py` script as follows:

```
%%writefile absorption_plot.py

import sys
import pandas
import matplotlib.pyplot as plt

data = pandas.read_csv(sys.argv[1])
data.columns = data.columns.map(float).map(int)

min_absorption = data.min(axis=0)
plt.plot(min_absorption)
plt.xlabel("wavelength")
plt.ylabel("min. absorption")
plt.savefig("./min_absorption_plot.png")
```

- In this example `sys.argv[1]` should correspond to the cleaned data file

```
python3 absorption_plots.py ./Desktop/UVVis-01-cleaned.csv
```

- We've done the very basics, but you can do so much with matplotlib (see gallery).

Programming Good Practice

- We are now nearing the end, in fact - you can put your keyboards down as there is no more typing to do.
- I want to end with three comments on "programming good practice"
- There could be a days worth of material just talking programming good practice and I only have a couple of minutes
- If you want to know more, I've included some links on this slide
- It's never too early to start doing this - in fact, I'd recommend doing this right from the off
- Follow standard Python style as defined in PEP8 (Python Enhancement Proposal):
 - focus on readability (code is read much more than it is written)--> consistency
 - maximum line length of 79 characters
 - whitespace: `spam(ham[1], {eggs: 2})` vs `spam(ham[1], { eggs: 2})`
 - use clear, meaningful variable names
- Use assertions to check for errors

```
def calc_bulk_density(mass, volume):  
    assert volume > 0  
    return mass / volume
```

- If the assertion is False, the Python interpreter raises an AssertionError runtime exception.
- Document your code with docstrings

```
def calc_bulk_density(mass, volume):  
    "Return dry bulk density = powder mass / powder volume."  
    assert volume > 0  
    return mass / volume
```

Recap

- 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
- lists: sequence type, immutable vs mutable, list method append, del
- for loops: dummy variable, loop syntax, index from 0
- conditionals: if, elif, else, ordering
- functions: function syntax, return statement, parameters and arguments
- variable scope: local and global variables
- libraries: modules, packages, libraries, import statements, aliases
- cleaning data with pandas: `pandas.read_csv`, DataFrames, `pandas.to_csv`
- analysing data with numpy: `numpy.loadtxt`, N-dimensional arrays, attributes
- plotting data with matplotlib: %matplotlib inline, `plot()`, `xlabel()`, `ylabel()`, `show()`, `savefig()`
- running code as a Python script: %writefile filename.py, `python3 filename.py`, `sys` library for command line arguments
- programming good practice: Python style, `assert` statements, docstrings

Closing Comments

- Document 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 it. Document why you have written something a certain way.
- Version control your code: look out for software carpentry sessions at your university. I use git to version control all of my code. That way, If I write something 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.

- Aim for reproducibility: use Jupyter Notebooks as SI to papers. 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.
- Keep going! 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, it does get easier, also the people around you probably aren't finding it as easy as you imagine.

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.

Feedback time!
