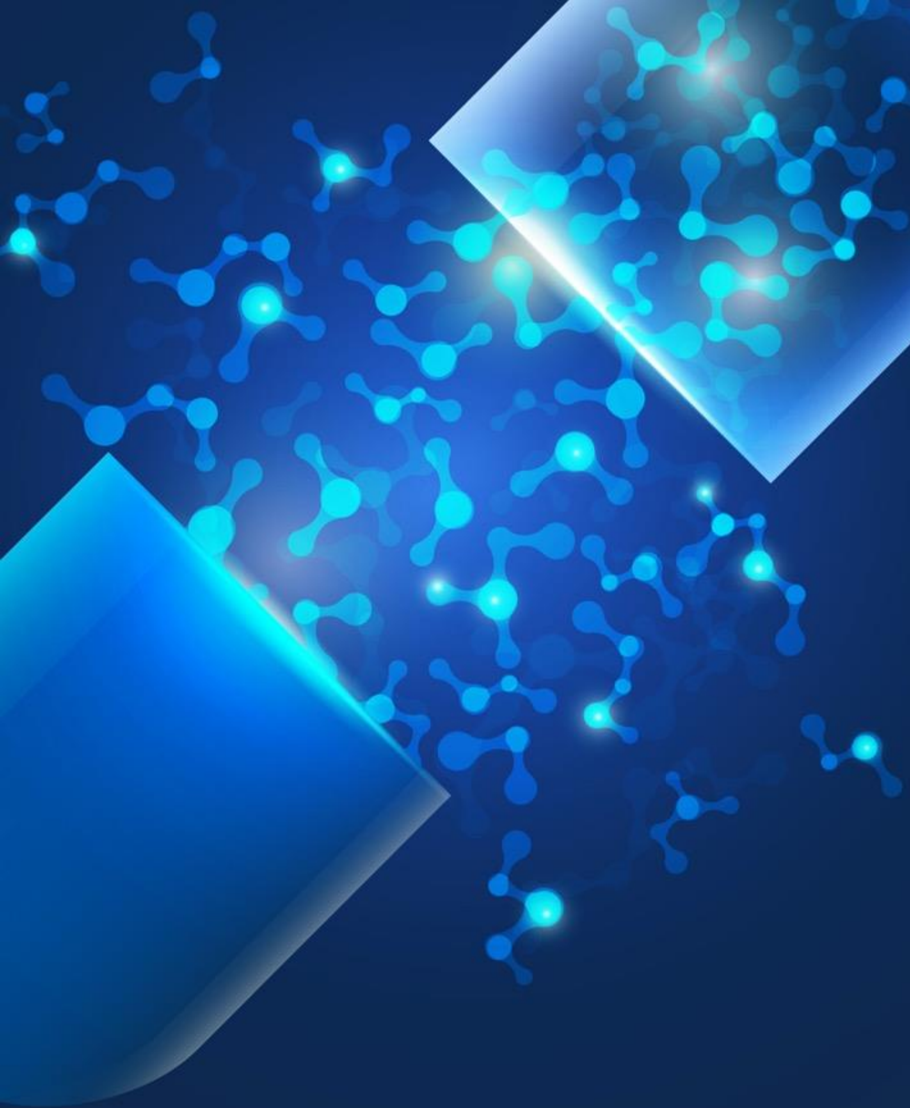


Introduction to Coding and the Python Programming Language

John Armstrong, Chantal Mustoe and Tony Vassileiou

CMAC
FUTURE MANUFACTURING
RESEARCH HUB

When do we Need to Code?

- Data processing – the extraction of important information from experimental data can be automated (if there is a decent amount of data).
- Data Analysis – fitting a model to establish relationships within your data
- Data Generation – running a simulation to produce synthetic data to compliment data collected in the lab

CMAC FUTURE MANUFACTURING RESEARCH HUB

Why do we Need to Code?

- It makes our life easier in the long run
- It makes other people's lives easier if they want to replicate a study or build upon work you've done
- It encourages collaboration: data may be different but similar between experiments
- Employability – "Coding is the new typing." – Tony (and probably some philosopher).

Aims for This Morning

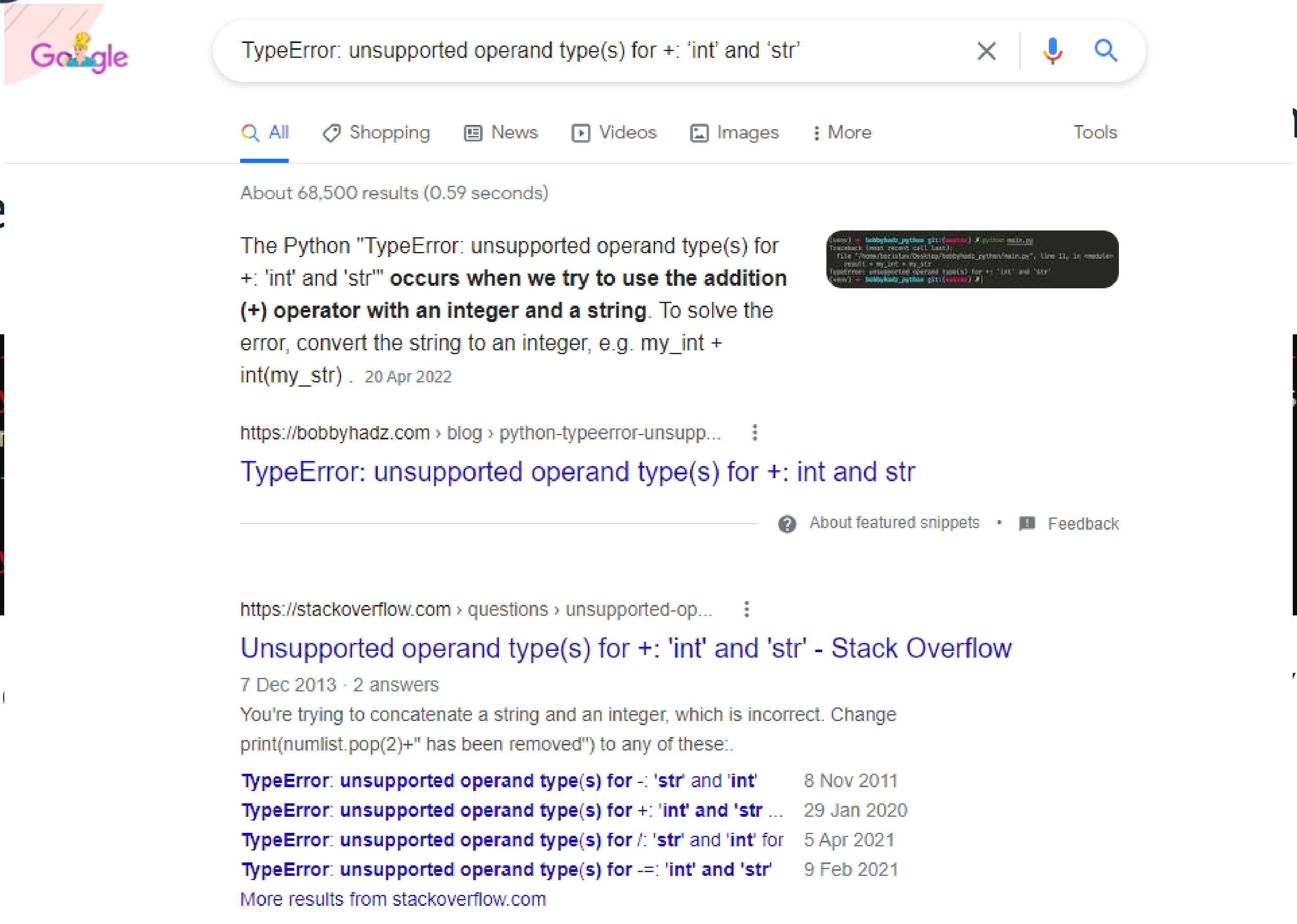
- Learn how to Google
- Introduction to Python
  - Common Scientific Packages
  - Basic datatypes and syntax
  - How to use Anaconda/Jupyter Notebook
- Extracting solubility data from files and doing some analysis

The Art of Googling

- Most of your time while coding will be spent Googling how to do certain things – pretty much anything you will want to code will have been done before, don’t spend too much time trying to reinvent the wheel
- That doesn’t mean there isn’t a lot of garbage code out there which is either not robust, not actually suited to your problem or both.
- Knowing the right thing to search for is a very good skill to have.
- **DON’T BE SCARED OF PACKAGE DOCUMENTATION!**

The Art of Googling: Error Messages

- Don't be … hen coding
- When de … e error message
  - ➢ e.g.
  - ➢ Googl … 'str'"



TypeError: unsupported operand type(s) for +: 'int' and 'str'

All · Shopping · News · Videos · Images · More · Tools

About 68,500 results (0.59 seconds)

The Python "TypeError: unsupported operand type(s) for +: 'int' and 'str'" **occurs when we try to use the addition (+) operator with an integer and a string**. To solve the error, convert the string to an integer, e.g. my_int + int(my_str) . 20 Apr 2022

https://bobbyhadz.com › blog › python-typeerror-unsupp...

TypeError: unsupported operand type(s) for +: int and str

About featured snippets • Feedback

https://stackoverflow.com › questions › unsupported-op...

Unsupported operand type(s) for +: 'int' and 'str' - Stack Overflow

7 Dec 2013 · 2 answers

You're trying to concatenate a string and an integer, which is incorrect. Change print(numlist.pop(2)+" has been removed") to any of these:.

TypeError: unsupported operand type(s) for -: 'str' and 'int'	8 Nov 2011
TypeError: unsupported operand type(s) for +: 'int' and 'str ...	29 Jan 2020
TypeError: unsupported operand type(s) for /: 'str' and 'int' for	5 Apr 2021
TypeError: unsupported operand type(s) for -=: 'int' and 'str'	9 Feb 2021

More results from stackoverflow.com

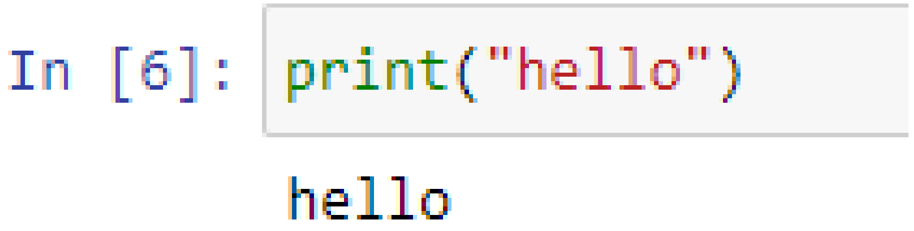
Python!

Python

- **Basic editing: commenting**
  - ➢ Commenting code is the practice of adding a comprehensible explanation to a potentially otherwise incomprehensible line of code: this helps others use the code and also helps YOU when you go back to a code 2 years later
  - ➢ In Python commenting is achieved using #
  - ➢ Any text after a # will be assumed to be a comment

Python

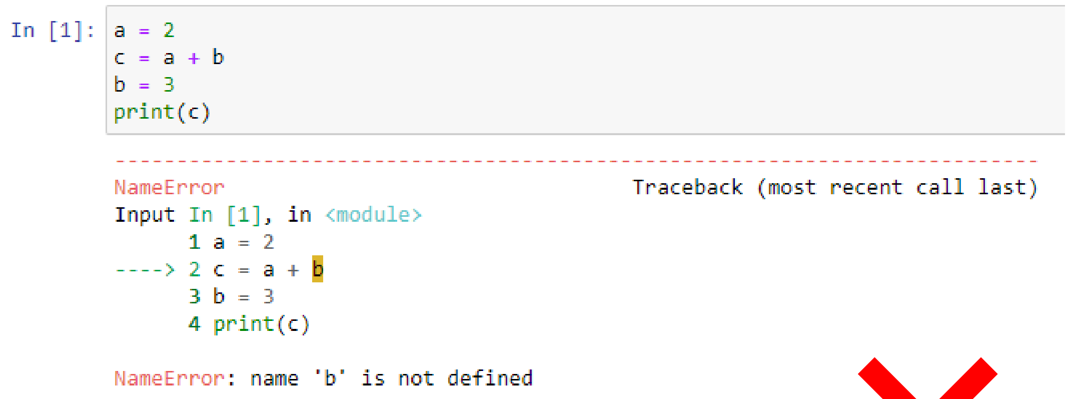
- **Basic editing: printing**
  - ➢ To display text, numbers or variable contents using the print() function in Python where the thing to be displayed goes between the brackets
  - ➢ e.g.



```
In [6]: print("hello")

        hello
```

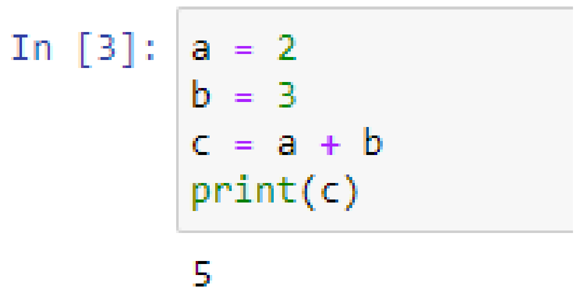
Python

- **Interpretive coding – each line is run in sequence within a code**



```
In [1]: a = 2
        c = a + b
        b = 3
        print(c)

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [1], in <module>
      1 a = 2
----> 2 c = a + b
      3 b = 3
      4 print(c)

NameError: name 'b' is not defined
```

✗



```
In [3]: a = 2
        b = 3
        c = a + b
        print(c)

5
```

✓

Python

- **Basic Python Datatypes: numbers (integers/floats)**
  - Numbers in Python to be used for maths can be represented as an integer or a float (both types can be combined with the resultant being a float by default).
  - Integers are whole numbers, floats contain decimal places
  - Examples of an integer: 1, -2, 34567, 10000
  - Example of a float: 0.34, 4.5e5, -3.14159265

Python supports exponential notation: $4.5\mathrm{e}5 = 4.5 \times 10^5$

Python

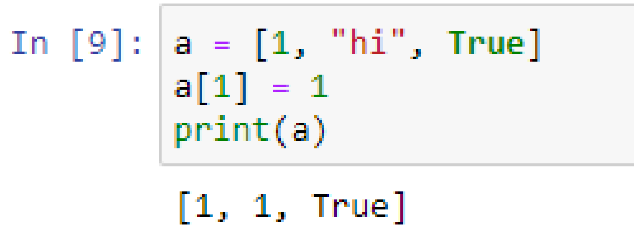
- **Basic Python Datatypes: strings**
  - ➢ Strings are useful for storing information like words or sentences or filenames
  - ➢ They are indicated by ‘’ or “” e.g. both ‘hi’ and “hi” are valid strings
  - ➢ Strings and numbers are not compatible (see earlier failing example) as “2” and 2 are not the same thing in Python
  - ➢ Strings can be added together e.g. “2” + “2” = “22” or “hello” + “world” = “hello world”

Python

- **Basic Python Datatypes: lists**
  - A list is a collection of data and is indicated by square brackets []
  - This can be a mix of data or data of all one type depending on which you would like to use the list for
  - e.g. [1, “hi”, True] and [1, 2, 3] are both valid lists
  - Adding two lists together simply creates a bigger list with all the elements in it e.g. [1, “hi”, True] + [1, 2, 3] = [1, “hi”, True, 1, 2, 3]
  - Multiplying a list by a number creates a big list where the initial list is repeated the number of times it was multiplied by e.g. [1, 2]*2 = [1, 2, 1, 2]

Python

- **Basic Python Datatypes: lists**
  - ➢ Single entries of a list can be accessed via **indexing** the list e.g. if we create a list and assign it to the variable name “a”, a = [1, “hi”, True] and we want to examine the first entry of a this would be accessed by doing a[0] in the Python code
  - ➢ Entries in a list can be changed by indexing e.g.



```
In [9]: a = [1, "hi", True]
        a[1] = 1
        print(a)

        [1, 1, True]
```

V. Important: Indexing in Python starts from 0, i.e. the first entry in a list is entry 0, the second is entry 1 etc.

Python

- **Basic Python Datatypes: lists**
  - ➢ Empty lists can be created by not including anything between the square brackets e.g. empty_list = [].
  - ➢ The most useful list feature is the option to append data to the list, this allows the list to be expanded as more data is generated, this is done through the ".append()" method
  - ➢ **Using an empty list and ".append()" in conjunction can be useful when generating data and needing an object to store it in.**

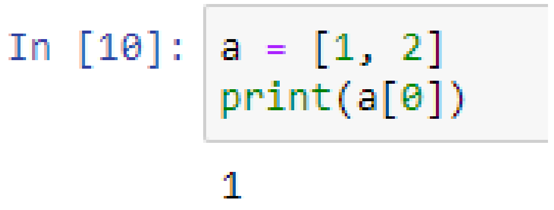
Python

- **Basic Python Datatypes: tuples**
  - ➢ tuples are similar to lists but are defined using circular brackets: () e.g. (1,2)
  - ➢ the main difference between tuples and lists is that tuples are *immutable*: this means that the elements cannot be changed

Python

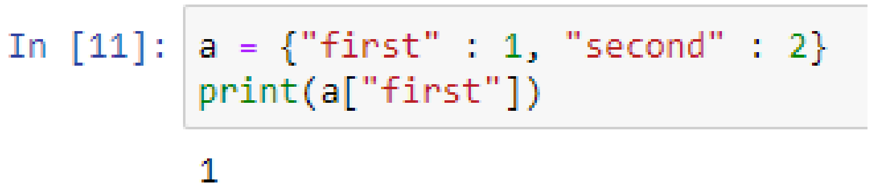
- **Basic Python Datatypes: dictionary**
  - ➢ Dictionaries are defined using curly brackets: {}
  - ➢ Dictionaries are similar to lists in that they are collections of data and objects but the main difference is that they contain *keys* to the data: while you have to index a list to access a certain element, to access an element in a dictionary you need to use its key e.g.

List



```
In [10]: a = [1, 2]
         print(a[0])

1
```

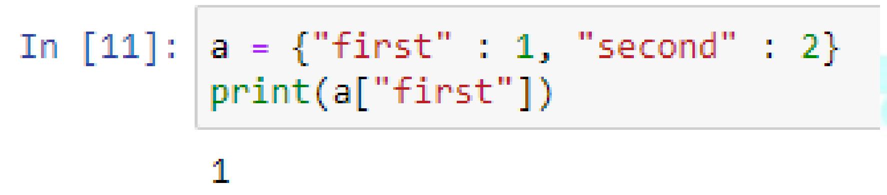
Dictionary



```
In [11]: a = {"first" : 1, "second" : 2}
         print(a["first"])

1
```

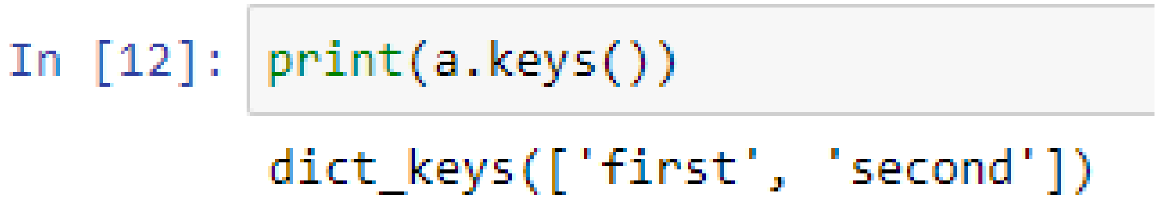
Python

Dictionary



```
In [11]: a = {"first" : 1, "second" : 2}
         print(a["first"])

1
```

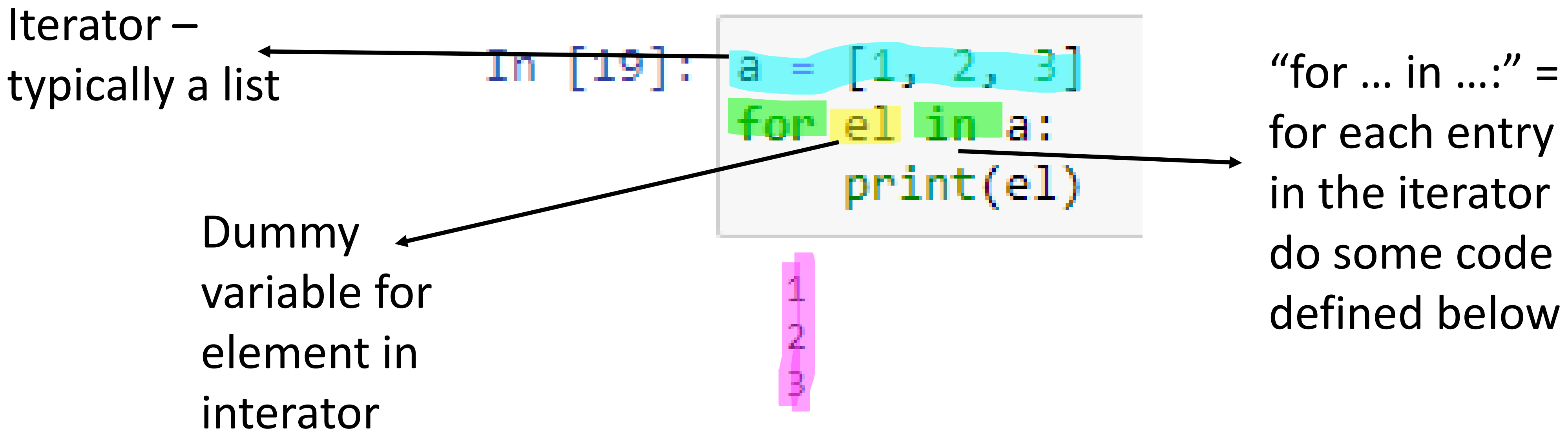
- **Basic Python Datatypes: dictionary**
  - In the example, “first” and “second” are the keys of the dictionary and 1 and 2 are the values – the keys can be whatever you want but are used to get the value you want
  - The keys do need to be strings however
  - The “.keys()” method can be used to return a list of the keys in the dictionary which can be useful when dealing with big dictionaries



```
In [12]: print(a.keys())

dict_keys(['first', 'second'])
```

Python

- **Python Logic: for loops**
  - A loop in programming is useful to repeat an action a set number of times without having to code each individual instance of the code
  - In Python this is achieved using a "for" loop which has the following structure

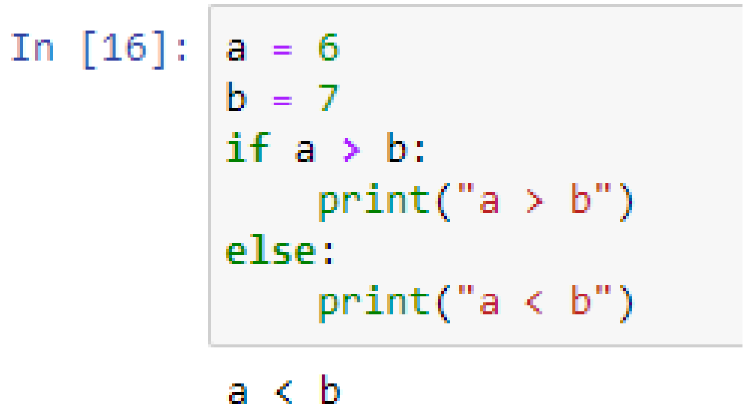


```
In [19]: a = [1, 2, 3]
         for el in a:
             print(el)

1
2
3
```

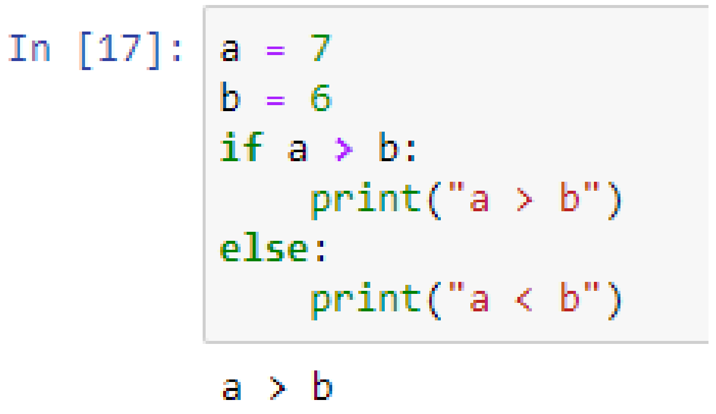
Iterator – typically a list
Dummy variable for element in interator
"for … in …:" = for each entry in the iterator do some code defined below

Python

- **Python Logic: if statements**
  - ➢ An if statement in Python is very useful when you need a certain piece of code to run under particular circumstances
  - ➢ For example, two numbers can be compared with one outcome based on one being larger than the other and another for the converse

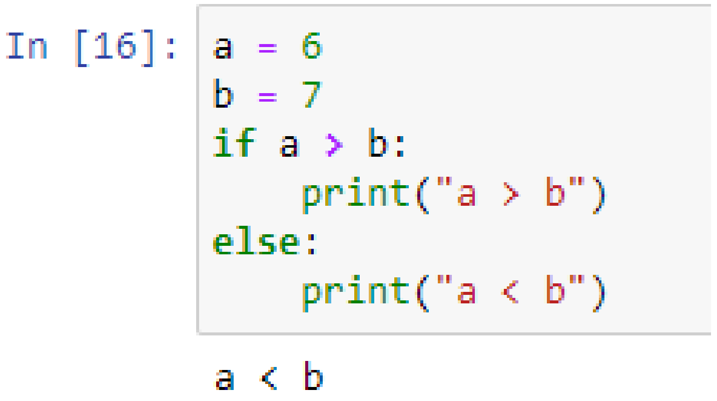


```
In [16]: a = 6
         b = 7
         if a > b:
             print("a > b")
         else:
             print("a < b")

         a < b
```

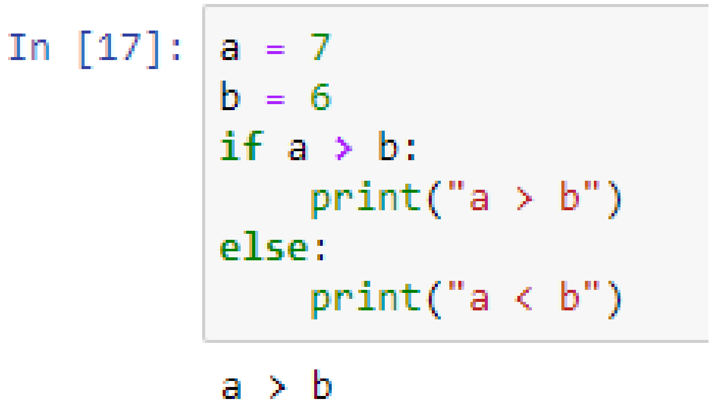


```
In [17]: a = 7
         b = 6
         if a > b:
             print("a > b")
         else:
             print("a < b")

         a > b
```

Python



```
In [16]: a = 6
         b = 7
         if a > b:
             print("a > b")
         else:
             print("a < b")

         a < b
```



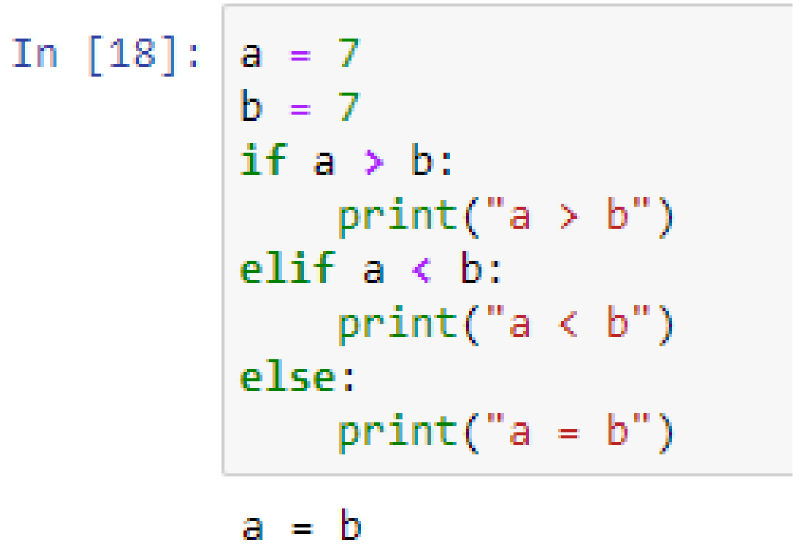
```
In [17]: a = 7
         b = 6
         if a > b:
             print("a > b")
         else:
             print("a < b")

         a > b
```

- **Python Logic: if statements**
  - ➢ Basic if statements have the form “if … else …”
  - ➢ This means that IF a condition is met then execute the first block of code and execute the second block of code otherwise

CMAC FUTURE MANUFACTURING RESEARCH HUB

Python

- **Python Logic: if statements**
  - ➢ This issue with the previous if statement is that the code does not take into account the two numbers being equal
  - ➢ When there are >2 possibilities for an if statement we can use "elif" (contraction of "else" and "if") to add the additional cases

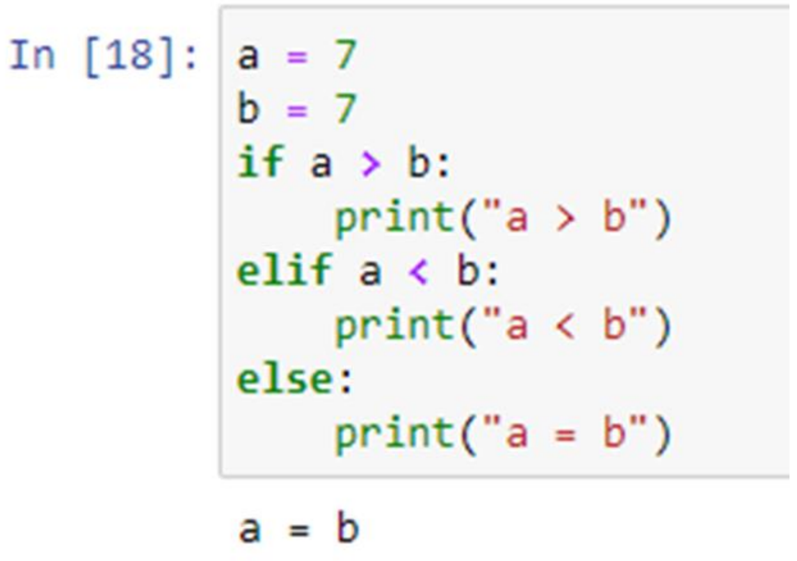


```
In [18]: a = 7
         b = 7
         if a > b:
             print("a > b")
         elif a < b:
             print("a < b")
         else:
             print("a = b")

         a = b
```

- ➢ As many "elif" can be used as are required
- ➢ "else" is not required but useful for general behaviour outside of any conditions you care about

Python

- **Python Formatting: blank space**
  - Python code is formatted using blank space indentation
  - When using an if statement or a for loop the that you want to be executed as a result of this condition must be indented by 4 spaces with respect to the condition

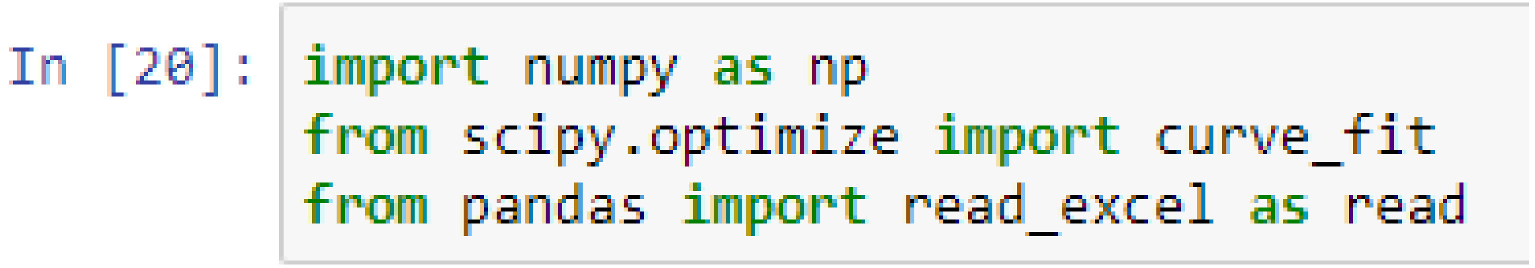


```
In [18]: a = 7
         b = 7
         if a > b:
             print("a > b")
         elif a < b:
             print("a < b")
         else:
             print("a = b")

         a = b
```

- The ":" is very important at the end of if statements and for loops
- Note the blank space after each defined case: this tells the computer only to execute the code that is indented when a condition is met

Python

- Python on its own is *fine* but the real power of the language comes from the packages that people have built for it – you will *always* start your code by loading the relevant packages
- This is done using the “import”, “as” and “from” commands



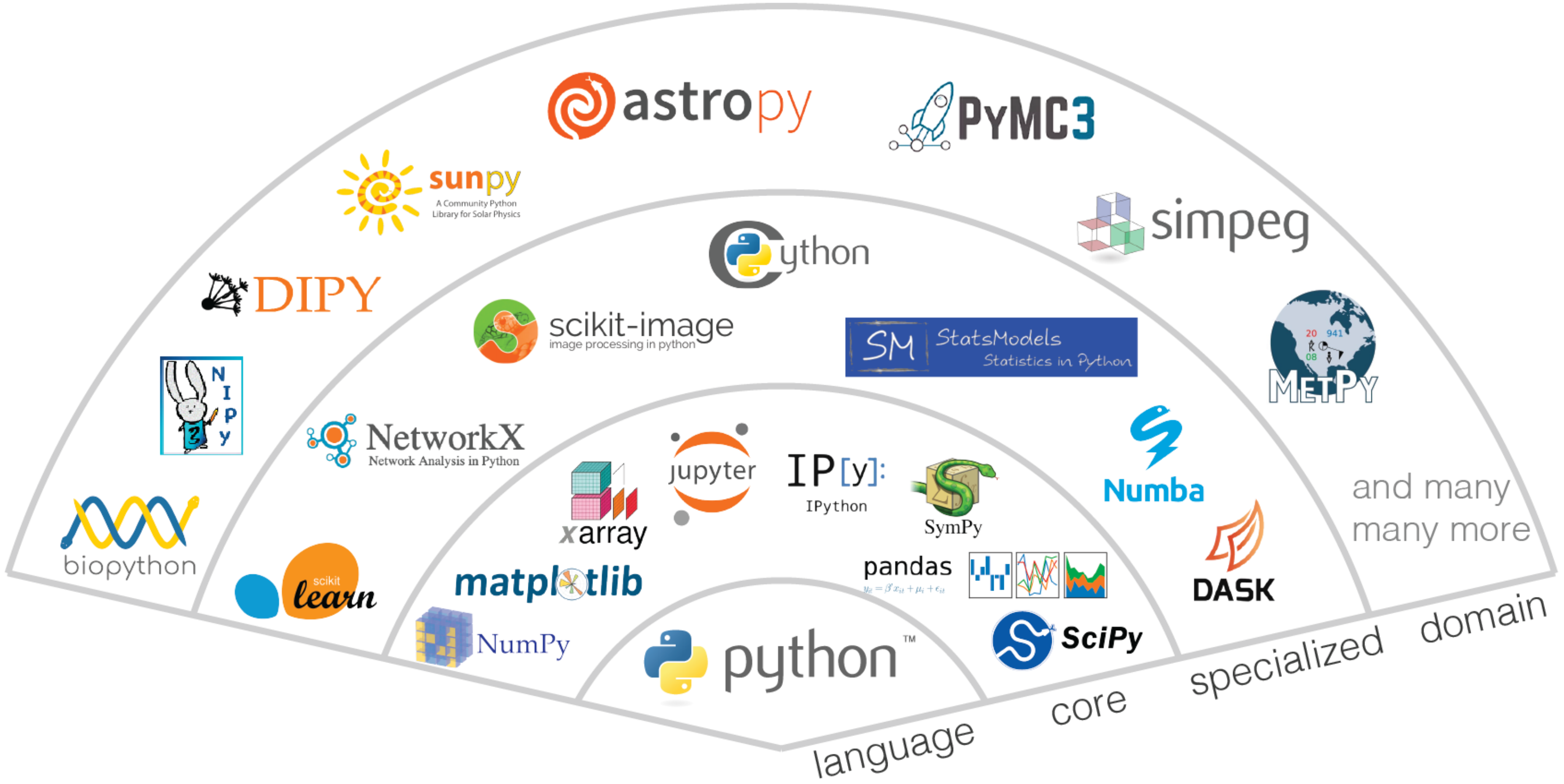
```
In [20]: import numpy as np
         from scipy.optimize import curve_fit
         from pandas import read_excel as read
```

- “as” creates an alias for a library or a function so the whole doesn’t need to be typed out every time
- Importing single functions such as “curve_fit” here is useful when you don’t want to load in the entirety of a package

CMAC FUTURE MANUFACTURING RESEARCH HUB

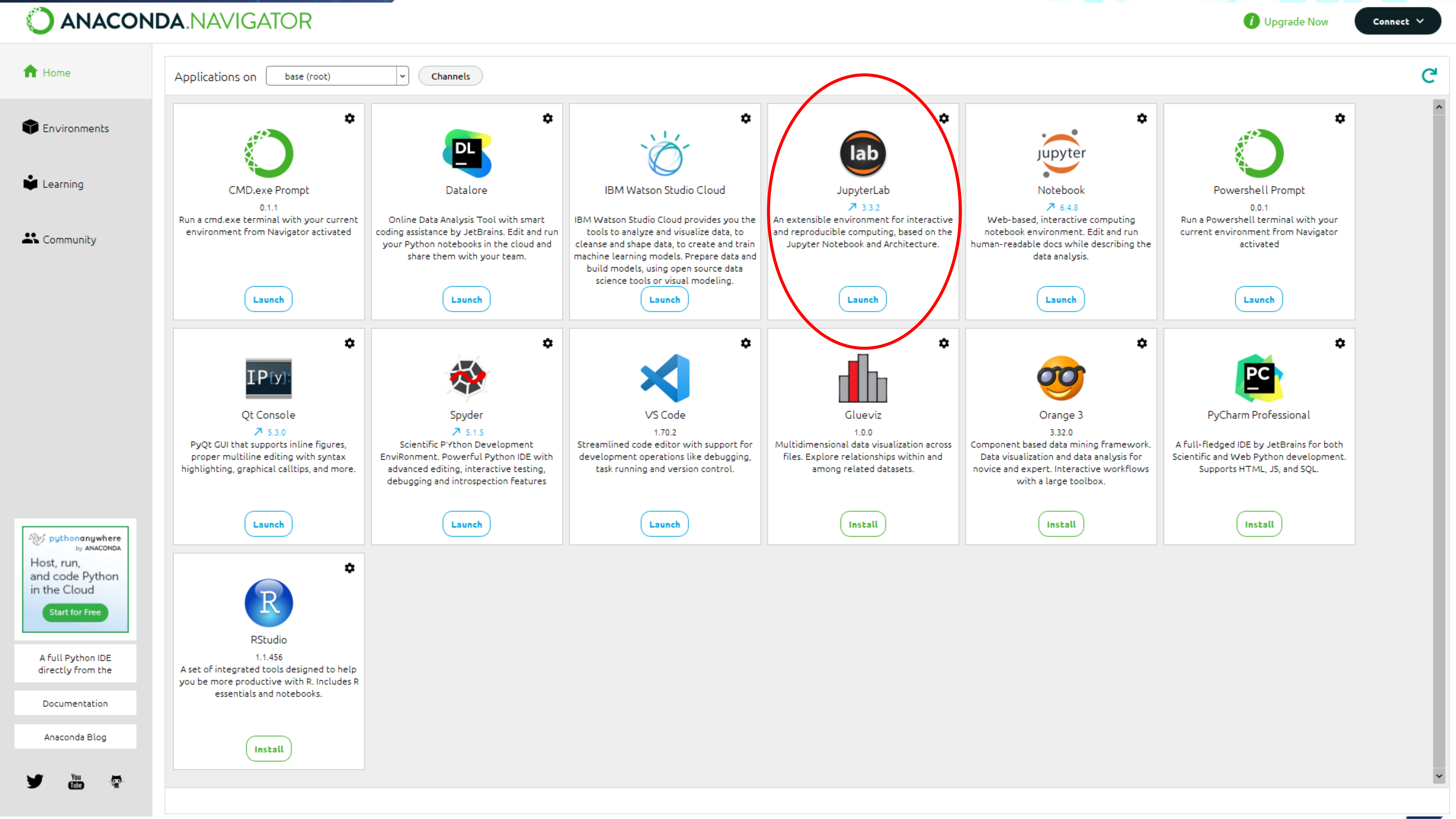
Python

- Where are these packages found? **The Python Scientific Ecosystem**



Let’s Get Coding!

There will be coffee soon I promise.



CMD.exe Prompt
0.1.1
Run a cmd.exe terminal with your current environment from Navigator activated
Launch

Datalore
Online Data Analysis Tool with smart coding assistance by JetBrains. Edit and run your Python notebooks in the cloud and share them with your team.
Launch

IBM Watson Studio Cloud
IBM Watson Studio Cloud provides you the tools to analyze and visualize data, to cleanse and shape data, to create and train machine learning models. Prepare data and build models, using open source data science tools or visual modeling.
Launch

JupyterLab
↗ 3.3.2
An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.
Launch

Notebook
↗ 6.4.8
Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.
Launch

Powershell Prompt
0.0.1
Run a Powershell terminal with your current environment from Navigator activated
Launch

Qt Console
↗ 5.3.0
PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.
Launch

Spyder
↗ 5.1.5
Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features
Launch

VS Code
1.70.2
Streamlined code editor with support for development operations like debugging, task running and version control.
Launch

Glueviz
1.0.0
Multidimensional data visualization across files. Explore relationships within and among related datasets.
Install

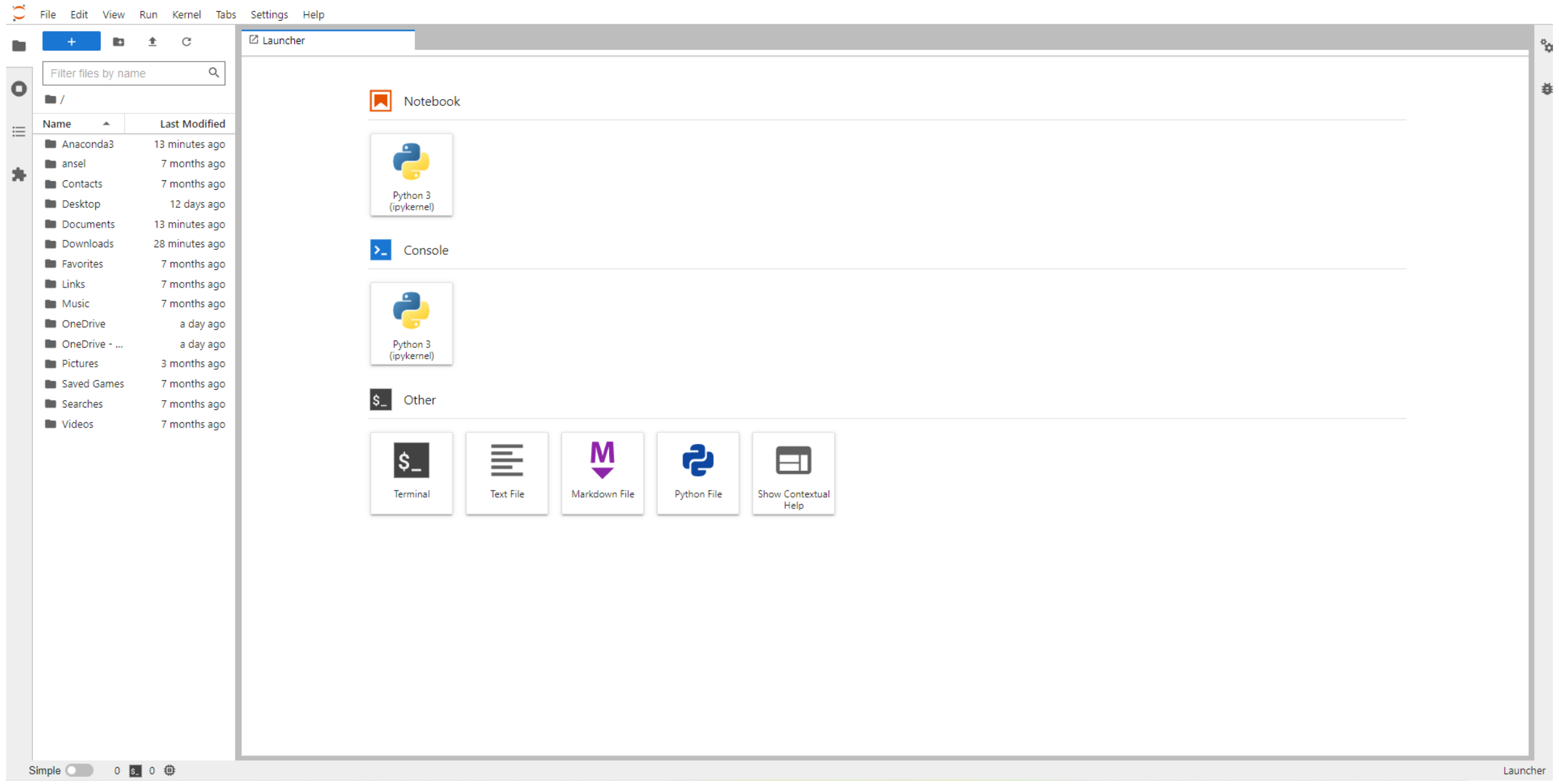
Orange 3
3.32.0
Component based data mining Framework. Data visualization and data analysis for novice and expert. Interactive workflows with a large toolbox.
Install

PyCharm Professional
A full-fledged IDE by JetBrains for both Scientific and Web Python development. Supports HTML, JS, and SQL.
Install

RStudio
1.1.456
A set of integrated tools designed to help you be more productive with R. Includes R essentials and notebooks.
Install

pythonanywhere by ANACONDA
Host, run, and code Python in the Cloud
Start for Free
A Full Python IDE directly from the
Documentation
Anaconda Blog

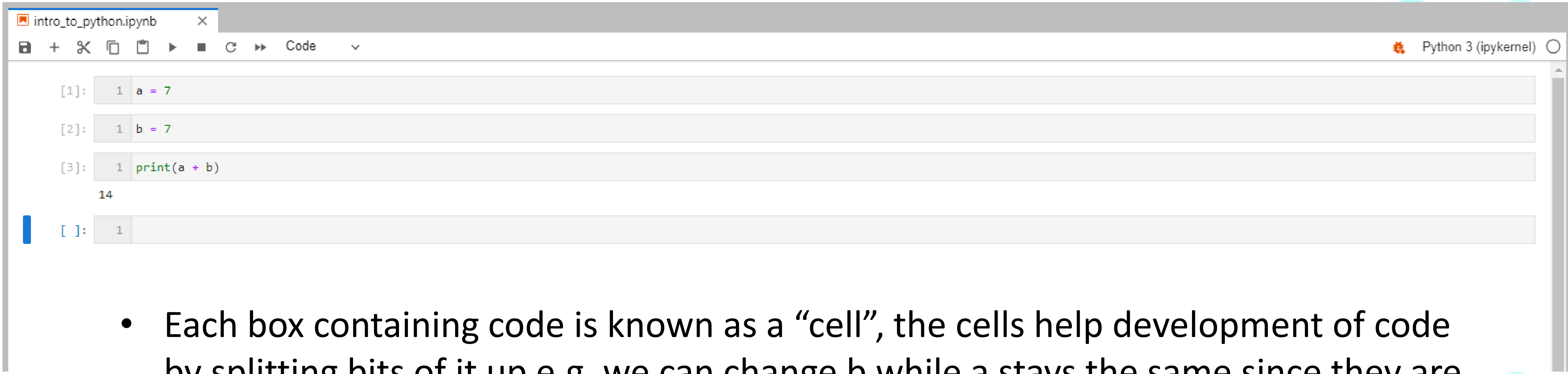
- Jupyter Lab will be opened in your browser after being launched and will look something like this



Name	Last Modified
Anaconda3	13 minutes ago
ansel	7 months ago
Contacts	7 months ago
Desktop	12 days ago
Documents	13 minutes ago
Downloads	28 minutes ago
Favorites	7 months ago
Links	7 months ago
Music	7 months ago
OneDrive	a day ago
OneDrive - ...	a day ago
Pictures	3 months ago
Saved Games	7 months ago
Searches	7 months ago
Videos	7 months ago

- Use the directories on the left hand side to navigate to the course materials (wherever you may have saved them)
- Inside the folder should be “.ipynb” files which are “interactive python notebooks” which is used for interactive development in Jupyter
    - ➢ I personally find this easier than running everything in the command line and I will typically convert my “.ipynb” files to “.py” files after I am happy with code development

- Example interactive python notebook!



```
[1]: a = 7
[2]: b = 7
[3]: print(a + b)
14
[ ]:
```

- Each box containing code is known as a “cell”, the cells help development of code by splitting bits of it up e.g. we can change b while a stays the same since they are in different cells
- Cells are run, created, deleted etc using the toolbar at the top
- Clicking the wee spider opens the debugger which can show you which variables are defined

Now Over to You! (yay?)

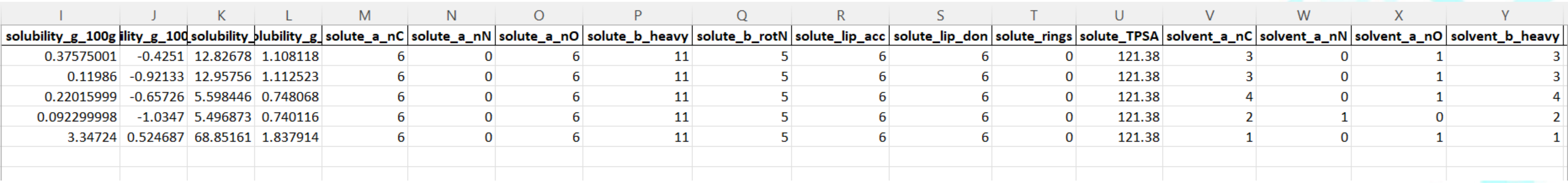
- Open the “intro_to_python.ipynb” file in the course directory
- It will contain a list of exercises to work through starting from learning some more of the important in-built Python stuff that I think is useful to know about e.g. range and enumerate and how to incorporate them in for loops and if statements

Dataset Introduction

What Data Will we be Using?

- Solubility data
- Original data structure is split by solutes: each solute has a file containing molecular descriptors of the solute and solvent along with its solubility in said solvent
- e.g. can be thought of like an Excel sheet where each row in a file differs only by the solvent
- Each solvent and solute has 9 molecular descriptors: # of C, # of N, # of O, # of bonds between heavy atoms, # of rotatable bonds, # of Lipinski H acceptors, # of Lipinski H donors, # of rings and total polar surface area

What Data Will we be Using?



I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
solubility_g_100g	ility_g_100	solubility_	plubility_g_	solute_a_nC	solute_a_nN	solute_a_nO	solute_b_heavy	solute_b_rotN	solute_lip_acc	solute_lip_don	solute_rings	solute_TPSA	solvent_a_nC	solvent_a_nN	solvent_a_nO	solvent_b_heavy
0.37575001	-0.4251	12.82678	1.108118	6	0	6	11	5	6	6	0	121.38	3	0	1	3
0.11986	-0.92133	12.95756	1.112523	6	0	6	11	5	6	6	0	121.38	3	0	1	3
0.22015999	-0.65726	5.598446	0.748068	6	0	6	11	5	6	6	0	121.38	4	0	1	4
0.092299998	-1.0347	5.496873	0.740116	6	0	6	11	5	6	6	0	121.38	2	1	0	2
3.34724	0.524687	68.85161	1.837914	6	0	6	11	5	6	6	0	121.38	1	0	1	1

- Solvent descriptors have “solvent” at the start and solute descriptors have “solute” at the start
- There are some junk columns towards the beginning that we will be getting rid of in the coding

CMAC
FUTURE MANUFACTURING RESEARCH HUB

Now Over to You! (yay?)

- The next set of exercises will take place in the “data_analysis.ipynb” notebook
- This will use the introduced dataset and show you how to load data using the Pandas Python package (woo alliteration) given that most data is in a CSV or Excel format
- Basic data manipulation and visualisation using Pandas and Matplotlib
- Curate the dataset and save out the data