Big Data Analytics in Python 2020:
FAQs & Additional Resources

Yr2.5PAHPPSY: BSc Psychology, King's College London

jodie.lord@kcl.ac.uk

@JodieLord5

To cover

- Loops
- Merge types
- Lists and tuples
- Markdown vs code chunks in jupyter notebooks
- Extra resources

To cover

- Loops
- Merge types
- Lists and tuples
- Markdown vs code chunks in jupyter notebooks
- Extra resources

Loops

- The common point of confusion:

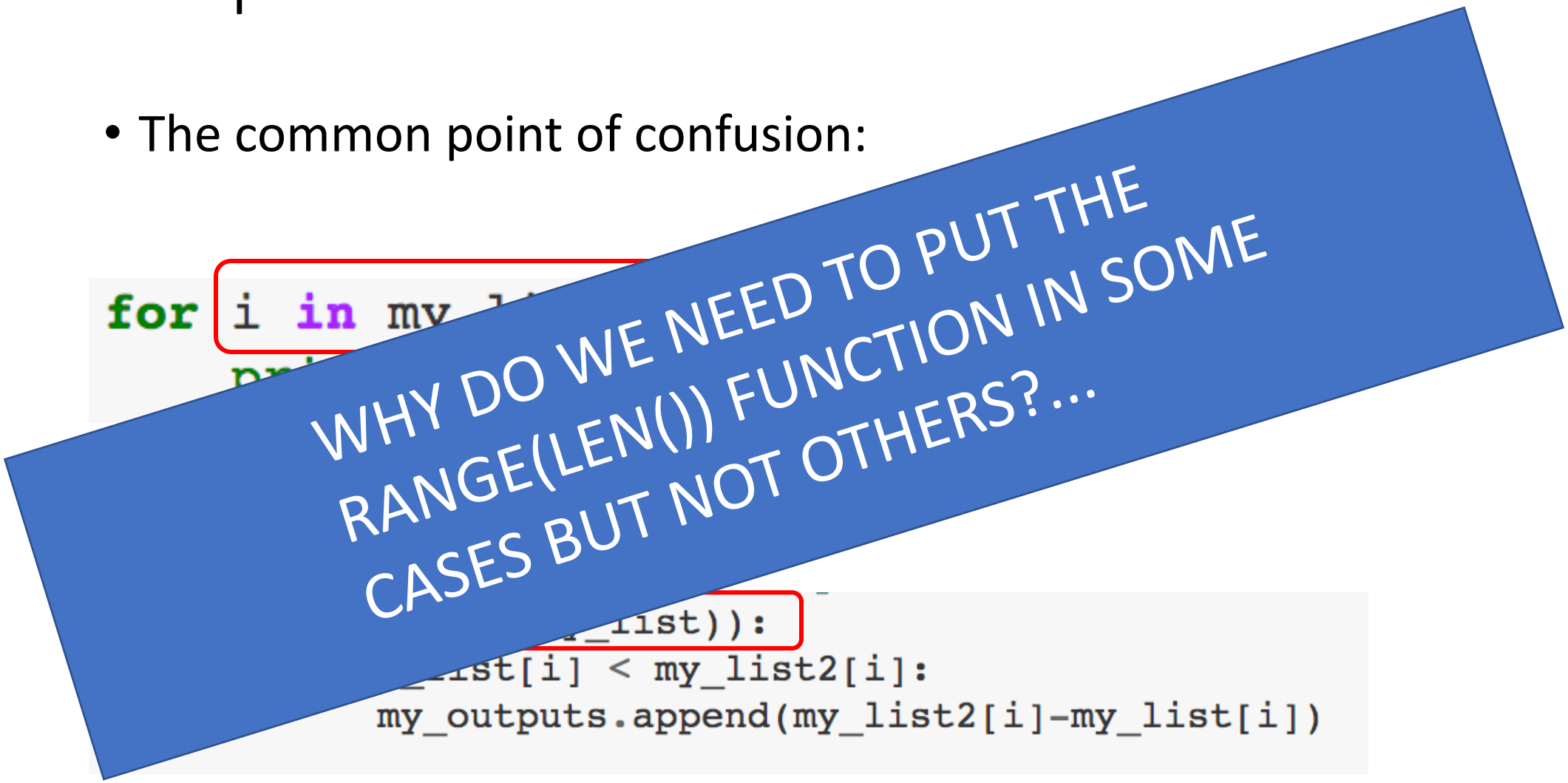
```
for i in my_list:
    print(i)
```

vs

```
for i in range(len(my_list)):
        if my_list[i] < my_list2[i]:
            my_outputs.append(my_list2[i]-my_list[i])
```

Loops

- The common point of confusion:



Loops

- Remember what your “iterator” is doing here…

  In this example, we want “i” to iterate through each **item** within our list

```
for i in my_list:
    print(i)
```

```
my_list=[2,4,6,8,10,12,14,16]
```

So using the iterator, we are telling python to:
print(2)
print(4)
print(6)
print(8)
print(10)
…… etc

“i” *becomes* each item in my_list and performs then function you have told it to perform on each of those items

Loops

(items in my_list):

```
my_list=[2,4,6,8,10,12,14,16]
```

(items in my_list2):

```
my_list2=[2,4,10,11,12,12.5,14,6]
```

- Now imagine applying the same logic to our second example.

If instead of this:

```
for i in range(len(my_list)):
        if my_list[i] < my_list2[i]:
            my_outputs.append(my_list2[i]-my_list[i])
```

We put this:

```
for i in (my_list):
        if my_list[i] < my_list2[i]:
            my_outputs.append(my_list2[i]-my_list[i])
```

We would be saying…

Loops

- We would be saying…

```
for i in (my_list):
        if my_list[i] < my_list2[i]:
            my_outputs.append(my_list2[i]-my_list[i])
```

(items in my_list):

```
my_list=[2,4,6,8,10,12,14,16]
```

(items in my_list2):

```
my_list2=[2,4,10,11,12,12.5,14,6]
```

First iteration:

if my [2] < my_list2[2]:
my_outputs.append(my_list2[2]-my_list[2])

or:

”if 6 is less than 10, append the my_outputs list with 10-6…

Because “2” is the first item in my_list. i becomes 2 in the first iteration, so our loop will look for the second index in my_list and in my_list 2…

Loops

- We would be saying…

```
for i in (my_list):
        if my_list[i] < my_list
            my_outputs.ap
```

First iteration:

if my

”if 6 the my_outputs list with 10-6…

(items in my_list):

```
my_list=[2,4,6,8,10,12,14,16]
```

(items in my_list2):

```
my_list2=[2,4,1                12.5,14,6]
```

THIS IS NOT WHAT WE WANT TO DO!!
We want it to iterate through from the first index!!

Loops

(items in my_list):

```
my_list=[2,4,6,8,10,12,14,16]
```

(items in my_list2):

```
my_list2=[2,4,10,11,12,12.5,14,6]
```

- We would be saying…

```
for i in (my_list):
        if my_list[i] < my_list2[i]:
            my_outputs.append(my_list2[i]-my_list[i])
```

Fifth iteration:

if my list[10] < my_list2[10]:
my_outputs.append(my_list2[10]-my_list[10])

INDEX 10
DOESN'T EXIST!

THIS DOESN'T MAKE SENSE!!
Python will throw an "out of range" error!!

Loops

(items in my_list):

```
my_list=[2,4,6,8,10,12,14,16]
```

(items in my_list2):

```
my_list2=[2,4,10,11,12,12.5,14,6]
```

- Because we are using the iterator to compare items across 2 lists, we want to tell python to look across a range of indices, starting from the first element (index 0) to the last.

```
my_list=[2,4,6,8,10,12,14,16]

print(len(my_list))
```

```
8
```

Our my_list object contains 8 items – confirmed with the len() function. We want to tell python we wish to iterate over 8 list items

BUT if we JUST tell it to iterate over the len(my_list), 'i' will *only* become 8 (because len=8 so i=8)…

Loops

(items in my_list):

```
my_list=[2,4,6,8,10,12,14,16]
```

(items in my_list2):

```
my_list2=[2,4,10,11,12,12.5,14,6]
```

- Instead, because we are using the iterator to compare items across 2 lists, what we want to do here to to refer to the **index** of each of these lists so they are comparable…
- We want to tell python to look across a range of indices, starting from the first element (index 0) to the last.

```
my_list=[2,4,6,8,10,12,14,16]

print(range(len(my_list)))
```

```
range(0, 8)
```

By adding the range() function, we are telling python that we want it iterate through the full range of the length of the list (from 0 to 8 in this case).

Loops

(items in my_list):

```
my_list=[2,4,6,8,10,12,14,16]
```

(items in my_list2):

```
my_list2=[2,4,10,11,12,12.5,14,6]
```

```
for i in range(len(my_list)):
        if my_list[i] < my_list2[i]:
            my_outputs.append(my_list2[i]-my_list[i])
```

So now we are saying:

First iteration:
[i becomes 0 because that's the first number in our range]:

If my_list[0] < my_list2[0]:
my_outputs.append(my_list2[0]-my_list[0])

Python is now looking at the 0 index element within both the first and second list and comparing them

Loops

(items in my_list):

```
my_list=[2,4,6,8,10,12,14,16]
```

(items in my_list2):

```
my_list2=[2,4,10,11,12,12.5,14,6]
```

```
for i in range(len(my_list)):
        if my_list[i] < my_list2[i]:
            my_outputs.append(my_list2[i]-my_list[i])
```

So now we are saying:

Third iteration:

If my_list[2] < my_list2[2]:

my_outputs.append(my_list2[2]-my_list[2])

OR:

If 6 is less than 10:

append the my_outputs list with(10-6)

Python is now looking at the 2nd index element within both the first and second list and comparing them

Loops

(items in my_list):

```
my_list=[2,4,6,8,10,12,14,16]
```

(items in my_list2):

```
st2=[2,4,10,11,12,12.5,14,6]
```

```
for i in range(len(my_list)):
        if my_list[i]
            my
```

THIS MAKES SENSE!!
This is what we want python to do here!!

If

(my_list2[2]-my_list[2])

OR

If 6 is less than 10:
append the my_outputs list with(10-6)

Python is now looking at the 2nd index element within both the first and second list and comparing them

Loops

This would also work in the same way:

```
for i in range(0,8):
        if my_list[i] < my_list2[i]:
            my_outputs.append(my_list2[i]-my_list[i])
```

But the previous way is less error prone as:

- Your list items might change (e.g. new items added, changing you range).
- In this example, you would have to manually update your number range every time (and may forget)
- By stating the range of the length of the object, this number will always update dynamically as the number of list items change

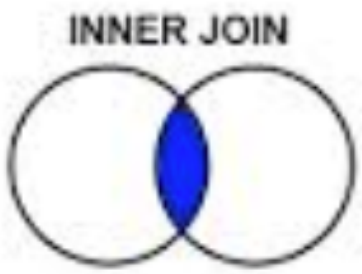
To cover

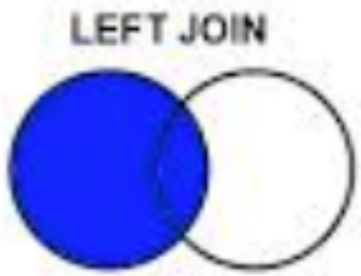
- Loops
- Merge types
- Lists and tuples
- Markdown vs code chunks in jupyter notebooks
- Extra resources

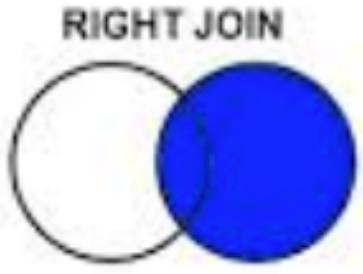
Merge Types

- The common point of confusion:

How do the different merge types we covered behave?:







Merge Types

- Imagine we have 2 dataframes: df1 and df2:

df1

ID	Height	Weight	Gender
1	160	75	Male
2	182	92	Male
3	152	58	Female
4	157	77	Female
5	175	86	Male
6	163	55	Female
7	159	67	Female

df2

ID	CogScore1	CogScore2	CogScore3
1	324	298	53
2	324	559	67
3	384	747	23
6	397	920	45
7	333	234	87
8	350	567	92

- Each has a common column: ID which we can use as a key to merge the two tables together…

Merge Types

- Some IDs within df1 are the same as the IDs in df2:

df1

ID	Height	Weight	Gender
1	160	75	Male
2	182	92	Male
3	152	58	Female
4	157	77	Female
5	175	86	Male
6	163	55	Female
7	159	67	Female

df2

ID	CogScore1	CogScore2	CogScore3
1	324	298	53
2	324	559	67
3	384	747	23
6	397	920	45
7	333	234	87
8	350	567	92

Merge Types

- Some of the IDs appear only in df1 but not in df2:

df1

ID	Height	Weight	Gender
1	160	75	Male
2	182	92	Male
3	152	58	Female
4	157	77	Female
5	175	86	Male
6	163	55	Female
7	159	67	Female

df2

ID	CogScore1	CogScore2	CogScore3
1	324	298	53
2	324	559	67
3	384	747	23
6	397	920	45
7	333	234	87
8	350	567	92

Merge Types

- Some of the IDs appear only in df2 but not in df1:

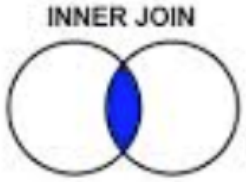
df1

ID	Height	Weight	Gender
1	160	75	Male
2	182	92	Male
3	152	58	Female
4	157	77	Female
5	175	86	Male
6	163	55	Female
7	159	67	Female

df2

ID	CogScore1	CogScore2	CogScore3
1	324	298	53
2	324	559	67
3	384	747	23
6	397	920	45
7	333	234	87
8	350	567	92

Merge Types:




```
df_inner=df1.merge(df2, on = ['ID'],how = 'inner')
```

- An inner join will merge together columns within df1 and columns within df2, ONLY for the row IDs which exist in both datasets.
- Any row IDs which exist ONLY in df1 will be removed in the merge.
- Any IDs which exit ONLY in df2 will be removed in the merge

df1

ID	Height	Weight	Gender
1	160	75	Male
2	182	92	Male
3	152	58	Female
4	157	77	Female
5	175	86	Male
6	163	55	Female
7	159	67	Female

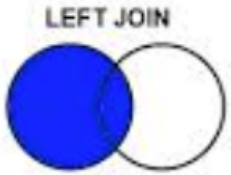
df2

ID	CogScore1	CogScore2	CogScore3
1	324	298	53
2	324	559	67
3	384	747	23
6	397	920	45
7	333	234	87
8	350	567	92

df_inner merged dataframe:

ID	Height	Weight	Gender	CogScore1	CogScore2	CogScore3
1	160	75	Male	324	298	53
2	182	92	Male	324	559	67
3	152	58	Female	384	747	23
6	163	55	Female	397	920	45
7	159	67	Female	333	234	87

Merge Types:



First df = "left" df

Second df = "right" df

```
df_left=df1.merge(df2, on = ['ID'],how = 'left')
```

- A left join will merge together columns within df1 and columns within df2, keeping ALL row IDs for the first dataframe specified within the merge argument (e.g. df1), regardless of whether that ID is in the second dataset.
- Any row IDs which exist ONLY in the second dataset will be removed in the merge.
- Any row IDs in the second dataset which also appear in the first dataset retained.

df1

ID	Height	Weight	Gender
1	160	75	Male
2	182	92	Male
3	152	58	Female
4	157	77	Female
5	175	86	Male
6	163	55	Female
7	159	67	Female

df2

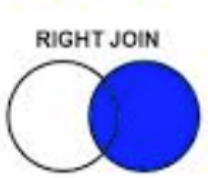
ID	CogScore1	CogScore2	CogScore3
1	324	298	53
2	324	559	67
3	384	747	23
6	397	920	45
7	333	234	87
8	350	567	92

Left merged df:

ID	Height	Weight	Gender	CogScore1	CogScore2	CogScore3
1	160	75	Male	324	298	53
2	182	92	Male	324	559	67
3	152	58	Female	384	747	23
4	157	77	Female	NaN	NaN	NaN
5	175	86	Male	NaN	NaN	NaN
6	163	55	Female	333	234	87
7	159	67	Female	350	567	92

Notice how IDs which were not available in df2 are still merged into the df, but as we don't have information for these IDs for df2 measures, entries for these columns are filled with missing values

Merge Types:



First df = "left" df

Second df = "right" df

```
df_right=df1.merge(df2, on = ['ID'],how = 'right')
```

- A right join will merge together columns within df1 and columns within df2, keeping ALL row IDs for the second dataframe specified within the merge argument (e.g. df2), regardless of whether that ID is in the first dataset.
- Any row IDs which exist ONLY in the first dataset will be removed in the merge.
- Any row IDs in the first dataset which also appear in the second dataset retained.

df1

ID	Height	Weight	Gender
1	160	75	Male
2	182	92	Male
3	152	58	Female
4	157	77	Female
5	175	86	Male
6	163	55	Female
7	159	67	Female

df2

ID	CogScore1	CogScore2	CogScore3
1	324	298	53
2	324	559	67
3	384	747	23
6	397	920	45
7	333	234	87
8	350	567	92

Left merged df:

ID	Height	Weight	Gender	CogScore1	CogScore2	CogScore3
1	160	75	Male	324	298	53
2	182	92	Male	324	559	67
3	152	58	Female	384	747	23
6	163	55	Female	333	234	87
7	159	67	Female	350	567	92
8	NaN	NaN	NaN	350	567	92

This time, IDs which were not available in df1 are still merged into the df, but as we don't have information for these IDs for df1 measures, entries for these columns are filled with missing values.

Merge Types:

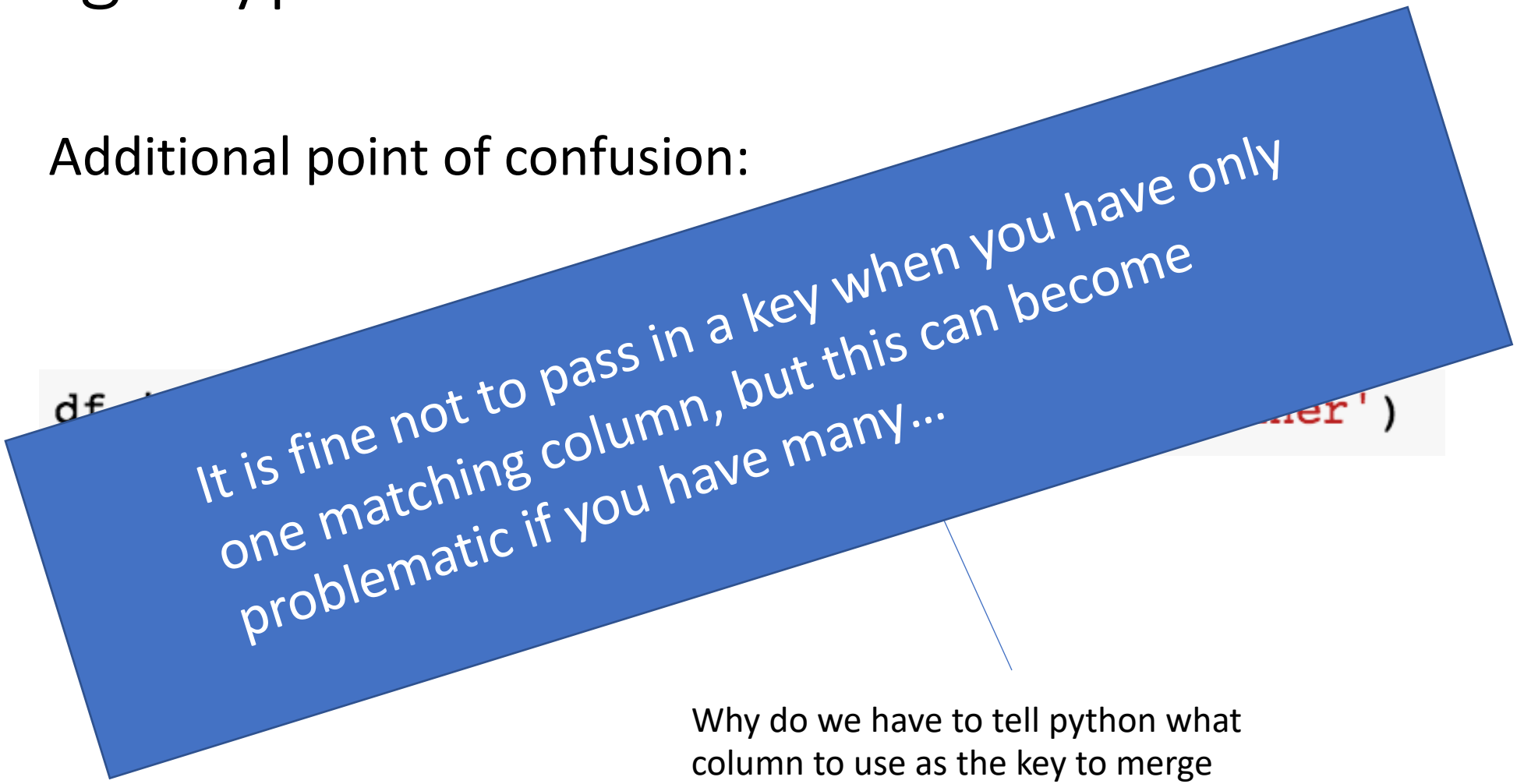
- Additional point of confusion:

```
df_inner=df1.merge(df2, on = ['ID'],how = 'inner')
```

Why do we have to tell python what column to use as the key to merge on? It still works when I don't tell it specifically what to use?...

Merge Types:

- Additional point of confusion:



Why do we have to tell python what column to use as the key to merge on? It still works when I don't tell it specifically what to use?...

Merge Types:

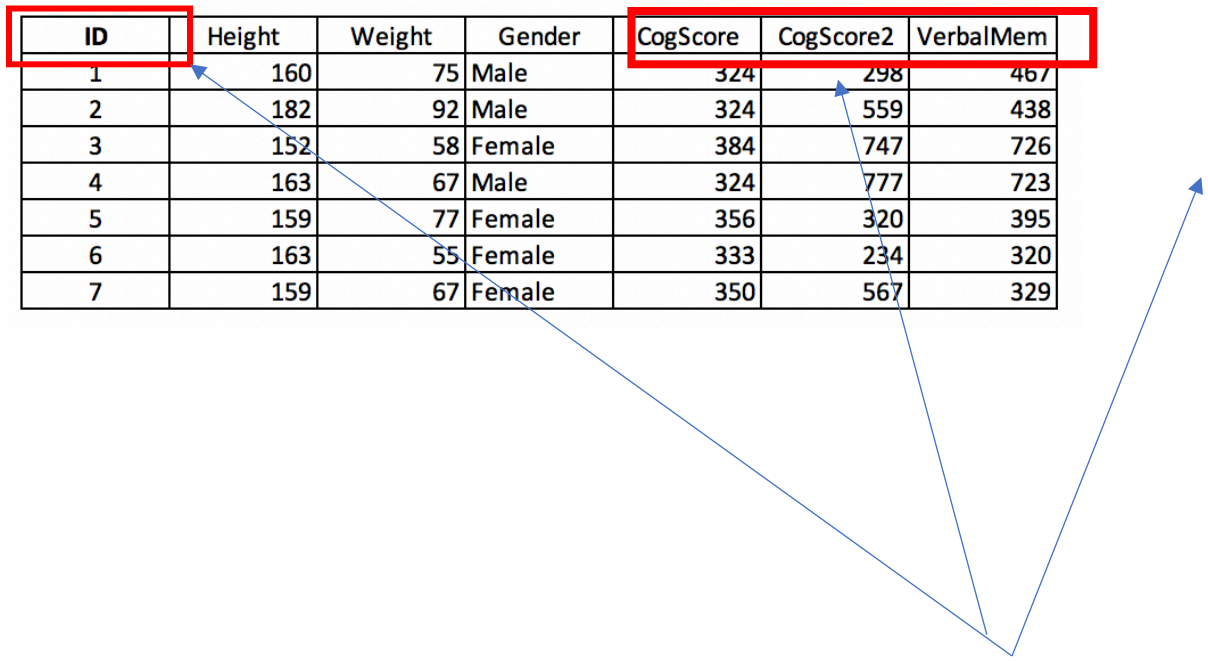
- Consider the following: we have 2 dfs we wish to merge:

df1

ID	Height	Weight	Gender	CogScore	CogScore2	VerbalMem
1	160	75	Male	324	298	467
2	182	92	Male	324	559	438
3	152	58	Female	384	747	726
4	163	67	Male	324	777	723
5	159	77	Female	356	320	395
6	163	55	Female	333	234	320
7	159	67	Female	350	567	329

df2

ID	CogScore	CogScore2	CogScore3	VerbalMem
1	234	546	53	467
2	456	345	67	438
3	345	355	23	910
8	345	345	92	564



Now we have multiple columns with the same name in each dataset?!
If you don't tell python which one to use – it will try and use the ALL!

Merge Types:

- Consider the following: we have 2 dfs we wish to merge:

df1

ID	Height	Weight	Gender	CogScore	CogScore2	VerbalMem
1	160	75	Male	324	298	467
2	182	92	Male	324	559	438
3	152	58	Female	384	747	726
4	163	67	Male	324	777	723
5	159	77	Female	356	320	395
6	163	55	Female	333	234	320
7	159	67	Female	350	567	329

df2

ID	CogScore	CogScore2	CogScore3	VerbalMem
1	234	546	53	467
2	456	345	67	438
3	345	355	23	910
8	345	345	92	564

This can become extremely cumbersome and messy.

ESPECIALLY if values across different “keys” match in some places but not others… Like here!

Merge Types:

- Consider the following: we have 2 dfs

df1

ID	Height	Weight	Gender	
1	160	75	Mal	
2	182			
3				
4				
5				
6				
7				

			balMem
		53	467
	345	67	438
	355	23	910
345	345	92	564

It's good practice to get into the habit of **explicitly** specifying which key you want to merge on (even if you think there is only one obvious one in the datasets).

…mely cumbersome and messy.

…ECIALLY if values across different "keys" match in some places but not others… Like here!

To cover

- Loops
- Merge types
- Lists and tuples
- Markdown vs code chunks in Jupyter notebooks
- Extra resources

List and Tuples

- The common point of confusion:

```
my_list=[67,"hello",99.7,len("12345"),8762]
print(my_list)
```

```
[67, 'hello', 99.7, 5, 8762]
```

```
my_tuple=(67,"hello",99.7,len("12345"),8762)
print(my_tuple)
```

```
(67, 'hello', 99.7, 5, 8762)
```

What is the difference between a list and tuple?! THEY LOOK THE SAME. What is the point of having both?

List and Tuples

- Indeed – list and tuples are extremely similar. Both can
  - Store a sequence of “things”
  - These “things” can range from anything from integers, functions, strings, lists (within lists), etc etc.
  - Be sliced.
  - Have specific elements accessed by their index.

- The key distinction is that:
  - Lists are **MUTABLE –** they can be changed, things can be added, things can be removed etc.
  - Tuples are **IMMUTABLE –** they cannot be changed once initiated.

List and Tuples

- Indeed – list and tuples are extremely similar. B
  - Store a sequence of “things”
  - This “things” can range f ions, strings, lists (
  - Be
- The
  - L **– they can be changed, things can be added, th be removed etc.
  - Tuples are **IMMUTABLE –** they cannot be changed once initiated.

WHAT’S THE POINT?!

List and Tuples

- There are some instances where you have a sequence of instances, but these may frequently change. E.g. patient weights which may fluctuate over time and you may want to update this.
- You may have a sequence of instances which may expand over time (e.g. time series data which is added to as more data is collected, with new instances added to represent new time points)

In this case, you want **a list** – something which allows you to store a sequence which can dynamically change when required.

List and Tuples

- There are some instances where you have a sequence of things which you want to act as “keys” to find other information (e.g. keys within a data dictionary)
- You may want to define “fixed” things to use in things like functions which remain static across time so the same function arg can be defined each time.

In this case, you want **a tuple** – something which actively prevents you from editing/updating the content because this will mess up anything which links to it.

“Immutable objects are a way to avoid side-effects that can make it difficult to reason about the meaning of your piece of code”..

List and Tuples

- There are some inst
  s
  to
  di
- You
  thir
  so t

BOTTOM LINE: If you have a sequence of “things” you think are likely to frequently change / need updating – use a list. If your thing’s are likely to remain static or are linking other items – use a tuple

atic across time

arg can be defined each time.

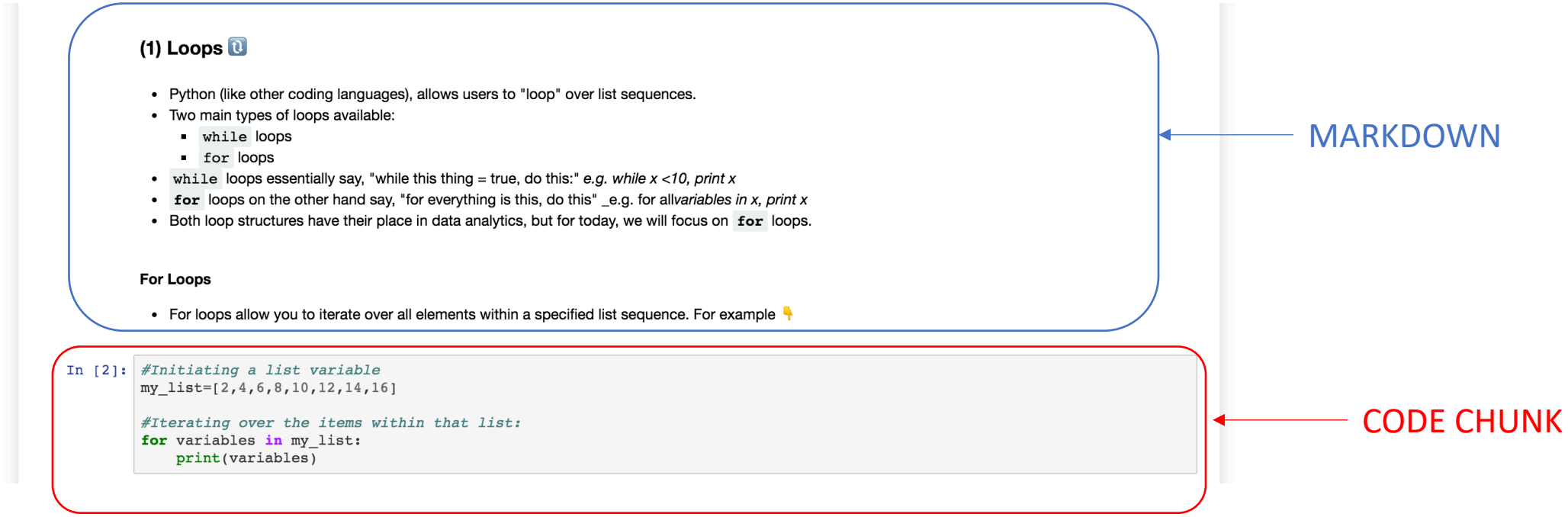
“Immutable objects are a way to avoid side-effects that can make it difficult to reason about the meaning of your piece of code”..

To cover

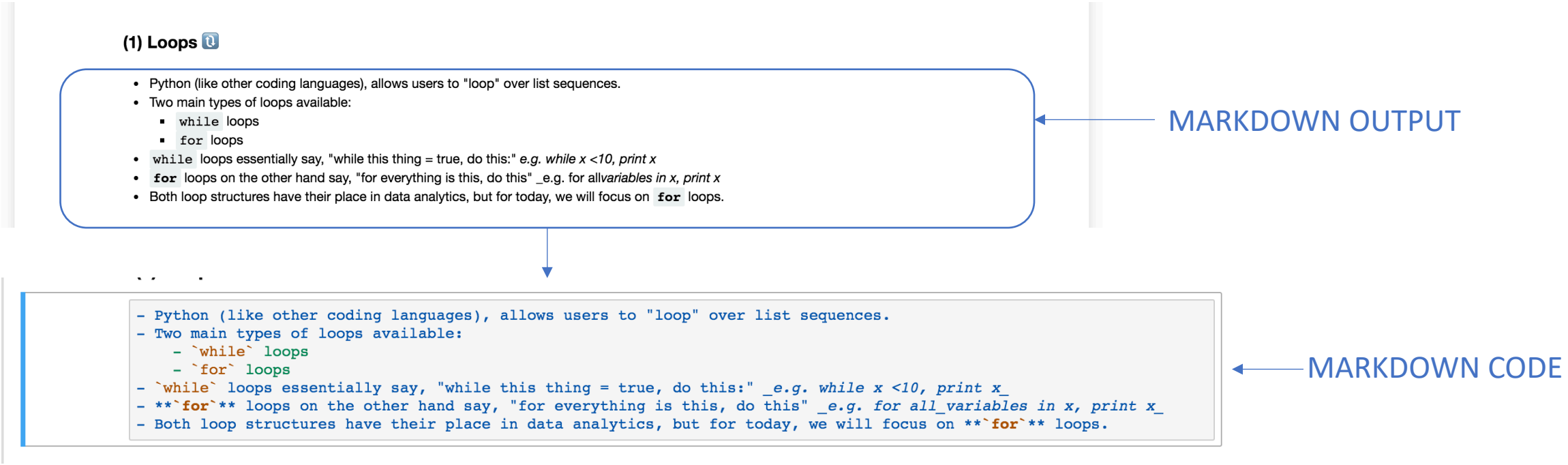
- Loops
- Merge types
- Lists and tuples
- Markdown vs code chunks in Jupyter Notebooks
- Extra resources

Markdown vs code chunks in Jupyter Notebooks

- You may have noticed that dispersed amongst your notebooks, in between the code chunks where you produce your python code, there are bits of formatted text providing instructions:

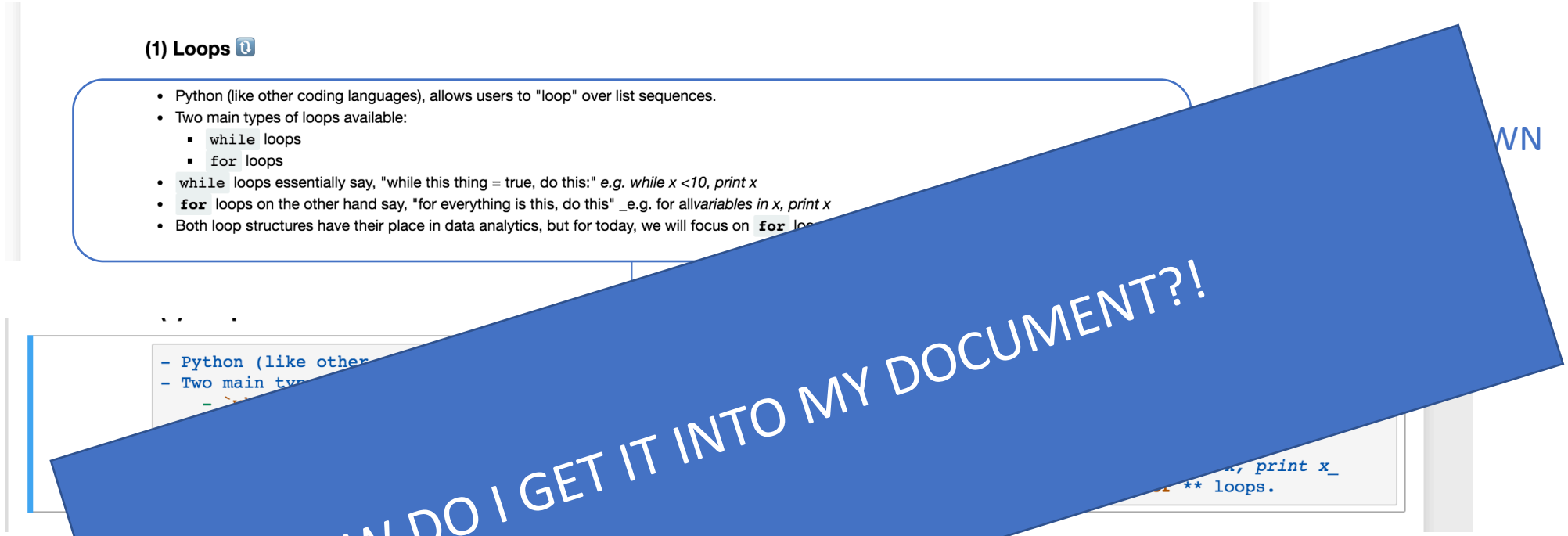


Markdown vs code chunks in Jupyter Notebooks



- If you double click into the cell with the formatted text, you'll see it's written in a special language known as "markdown"…
- This is similar to, but a simplified version of html language.
- It allows you to input non-python chunks into your notebook document, with special formatting such as **bolding,** *italics,* colours, images, •bullet points etc.
- This allows you to produce a nicely formatted document which can be disseminated with both code and context.

Markdown vs code chunks in Jupyter Notebooks



BUT HOW DO I GET IT INTO MY DOCUMENT?!

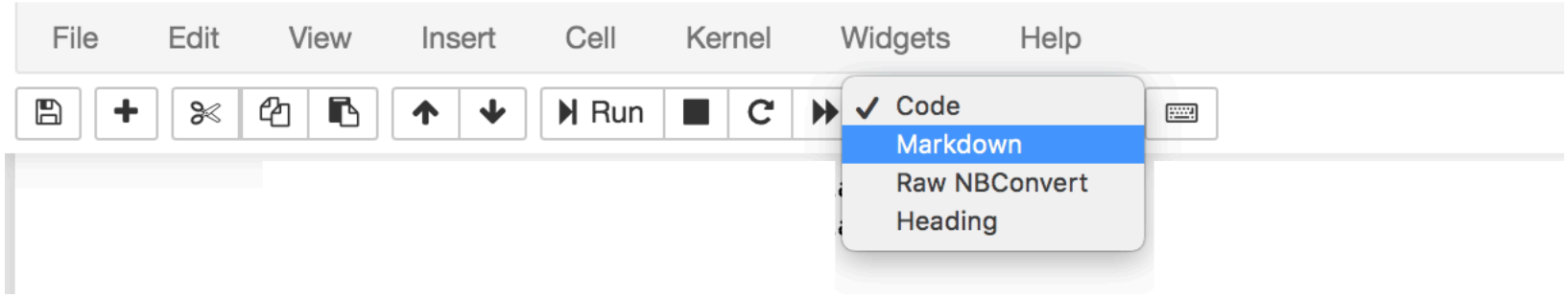
- If yo …, you'll see it's written in a special language known as "markdown"…
- This is …ersion of html language.
- It allows … input non-python chunks into your notebook document, with special formatting such as **bolding,** *italics,* colours, images, •bullet points etc.
- This allows you to produce a nicely formatted document which can be disseminated with both code and context.

Markdown

1. Create a new code chunk within your notebook

Make sure you're sitting within that specific codechunk, but in "edit" (rather than "command" mode – as indicated by a blue (rather than green) border.

2. Toggle the dropdown menu at the top of your workbook from "code" to "markdown



3. Use the cell to write some markdown. Once ready, run the cell in the same way you would run a python code chunk!

```
- Python (like other coding languages), allows users to "loop" over list sequences.
- Two main types of loops available:
    - `while` loops
    - `for` loops
```

Markdown

1. Create a new code chunk within your notebook

Make sure you're sitting within that specific codechunk, but in "edit" (rath *border.*

2. Toggle the dropdown menu at th

BUT I DON'T KNOW HOW TO WRITE IN MARKDOWN?!

3. Use the c **arkdown. Once ready, run the cell in the same way you would run a python code chunk!**

```
- Python (like other coding languages), allows users to "loop" over list sequences.
- Two main types of loops available:
    - `while` loops
    - `for` loops
```

Markdown

Some useful resources to get you started:

- Github:
  - https://guides.github.com/features/mastering-markdown/
  - https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet
  - https://github.com/adam-p/markdown-here
- Youtube:
  - https://www.youtube.com/watch?v=HUBNt18RFbo
  - https://www.youtube.com/watch?v=jBCB23pQeIA

Markdown

Some useful resources to get you started:

- Github:
  - https://guides.github.com/feat
  - https://gith heet
- Y
  - be.com/watch?v=HUBNt18RFbo
  - https://www.youtube.com/watch?v=jBCB23pQeIA

ALTERNATIVELY: Double click into my markdown cell chunks and see how I have produced my output... 🙌

Markdown

- You are **NOT** expected to produce markdown as part of your assessment
- Using comments (#) within your code chunks to explain what you have done is perfectly acceptable.
- Markdown is just a good skill to have in your toolbox, and if used in your assessment, will make the output more visually appealing to the reader.
- Try it out, have fun with it. If it's too overwhelming, forget it for now – your code is what's important.

To cover

- Loops
- Merge types
- Lists and tuples
- Markdown vs code chunks in Jupyter Notebooks
- Additional resources

Additional Resources:

Some extra resources you may find useful:

- Expanding your charting repertoire:
  - https://mode.com/blog/python-data-visualization-libraries/
  - https://python-graph-gallery.com/
  - https://plot.ly/python/

- Expanding your machine learning repertoire:
  - https://scikit-learn.org/stable/
  - https://machinelearningmastery.com/machine-learning-in-python-step-by-step/
  - https://towardsdatascience.com/beginners-guide-to-machine-learning-with-python-b9ff35bc9c51

Additional Resources:

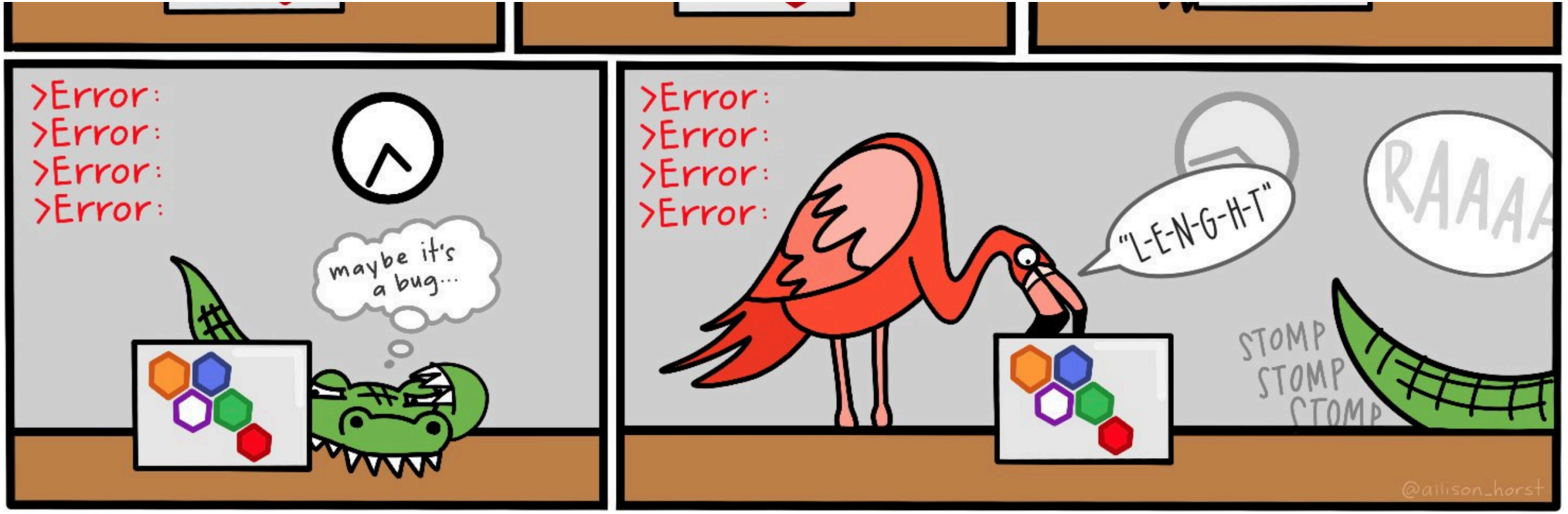
Some extra resources you may find useful:

- General online coding tutorials:
    - COURSERA (some free, some require payment – large repertoire of learning material).
    - DataCamp (My favourite. Very comprehensive, but most courses require paid subscription).
    - Kaggle (also do things like hackathons, where you have the opportunity to win money for your code!).

FINALLY:

I can almost guarantee that 99%* of the time, errors will be due to subtle things missed or misplaced in the code (it happens to us all (**OFTEN**)).

!! CHECK THE OBVIOUS BEFORE GIVING UP ON YOUR CODE !!



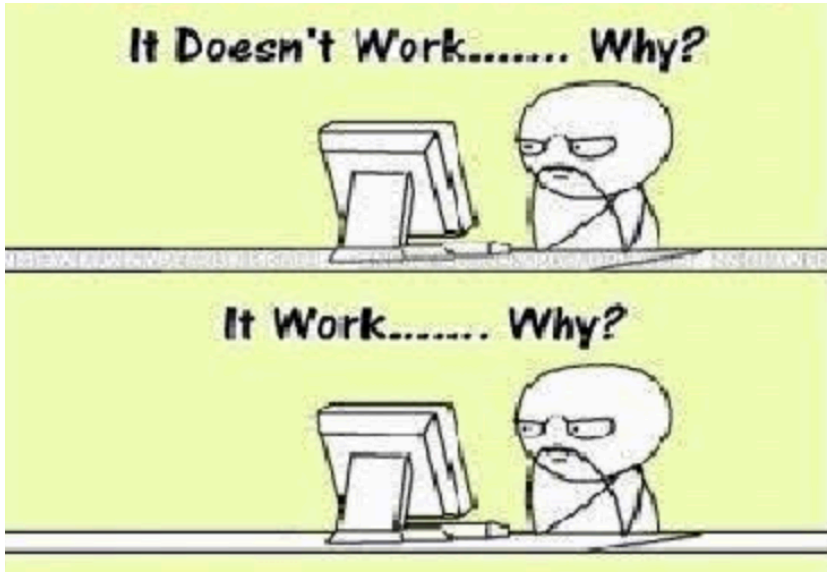
*based on no stats whatsoever – just personal experience!!

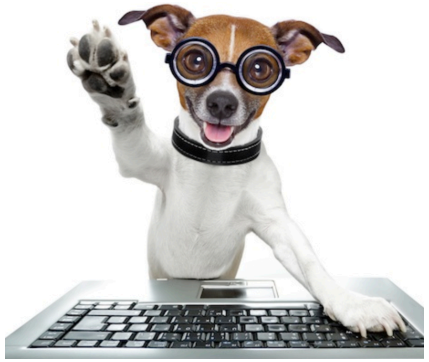
QUESTIONS:

- **Forum** (preference)

  Or:

- Email (slower): **jodie.lord@kcl.ac.uk**





HAPPY CODING!!!