THE UNIVERSITY *of* EDINBURGH
Centre for Data, Culture & Society

CDCS TRAINING PROGRAMME

AN INTRODUCTION TO PYTHON.

ARRANGEMENTS FOR THE COURSE

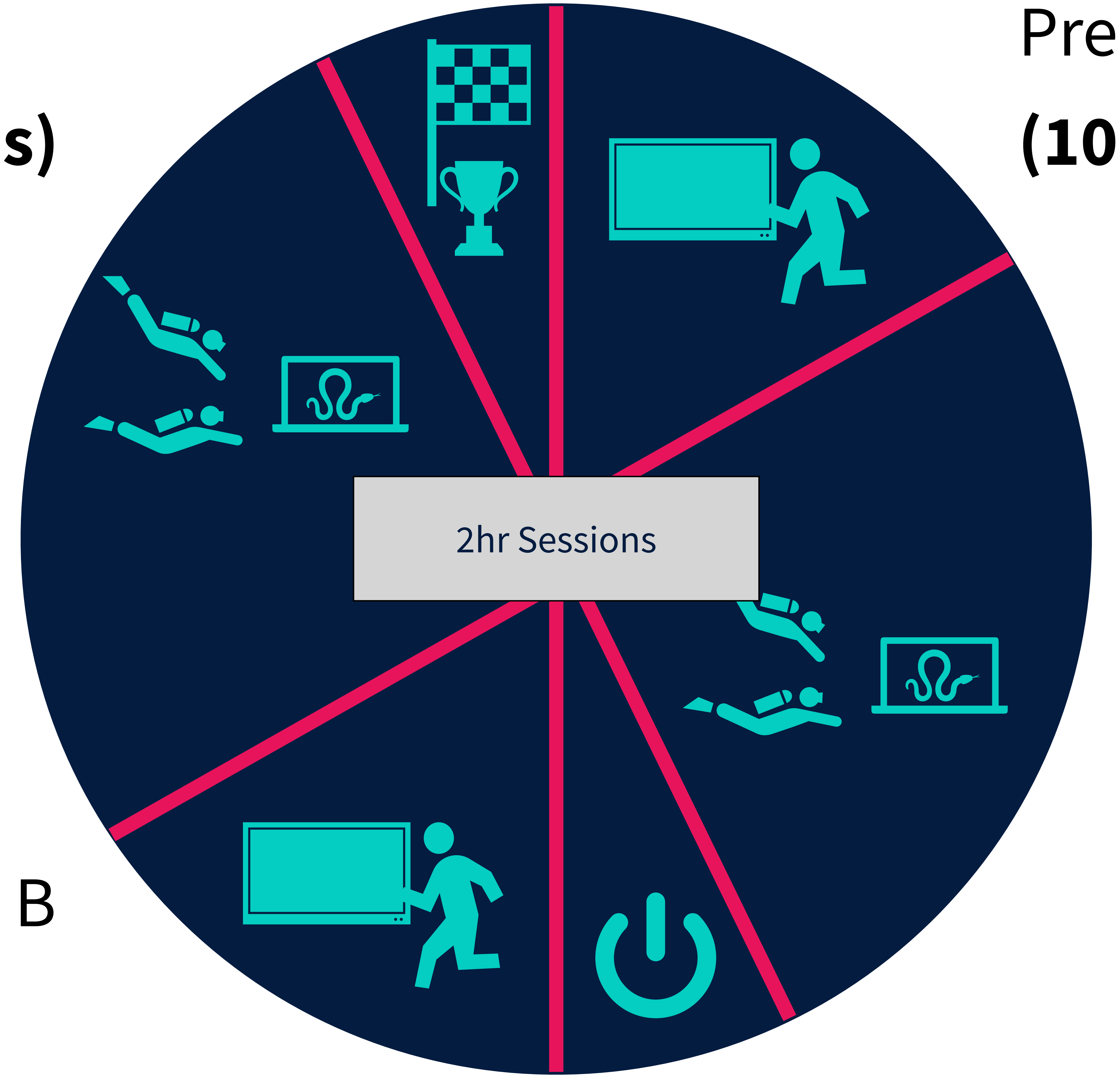
Time	Session 1 Friday 24th January	Session 2 Friday 31st January	Session 3 Friday 7th February
Topic A	Introduction to Noteable and Python	Functions	Collections
Topic B	Conditions and Logic		List Comprehensions

THE TEAM

Chris Oldnall

Martin Disley

SESSIONS THROUGHOUT THE COURSE



Presentation of Topic A

(10-15 minutes)

Paired Programming Practice A

(35-40 minutes)

Mini-Break

(10 minutes)

Presentation of Topic B

(10-15 minutes)

Paired Programming Practice B

(30-35 minutes)

Wrap-Up

(10 minutes)

PAIRED PROGRAMMING

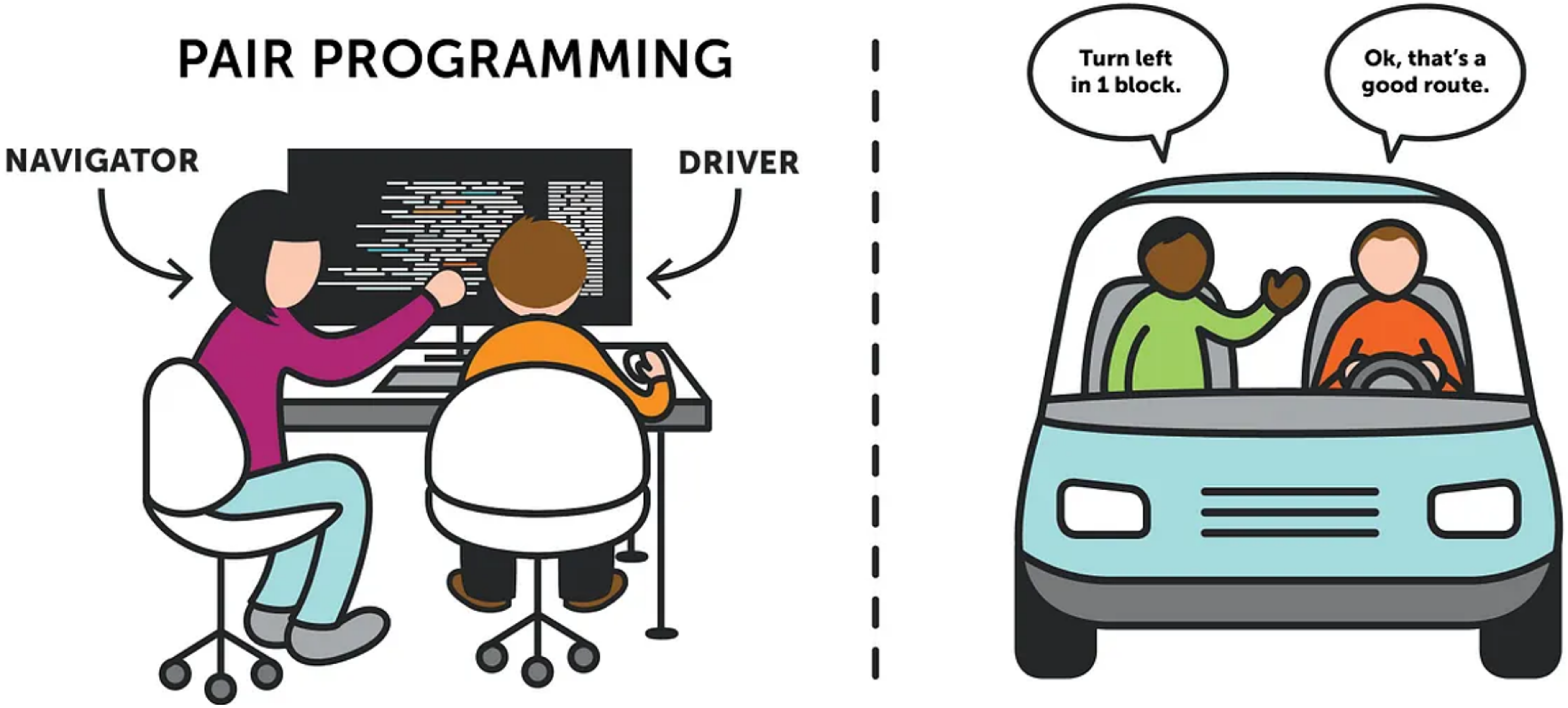


Image from https://medium.com/@tomspencer_uk/pair-programming-and-problem-solving-4531ef3bf171

OTHER THINGS YOU WILL SEE THROUGHOUT THE COURSE

DEMONSTRATIONS

Sometimes you might see the typewriter symbol. This means we are going to demonstrate something in Python/Noteable.

Bear with us if it takes a moment to switch windows.

```
variable_name = sensible
print(variable_name)

“sensible”
```

CODE CHUNK TEXT

In the slides we may see text which is ‘pink’ in colour and a different font. This is to indicate it is a chunk of text, written in Python. The colour/font don’t matter just noticing it is code is important!

INTRODUCTION TO NOTEABLE AND PYTHON

Programming = Telling the computer what to do

You need to speak a language the computer will understand, e.g. Python

HOW DO WE ‘SPEAK’ PYTHON?

In order to ‘speak’ Python we will need two things…

1. The syntax (equivalent to the grammar of a language),
2. an interpreter (an interface to take in our Python syntax).

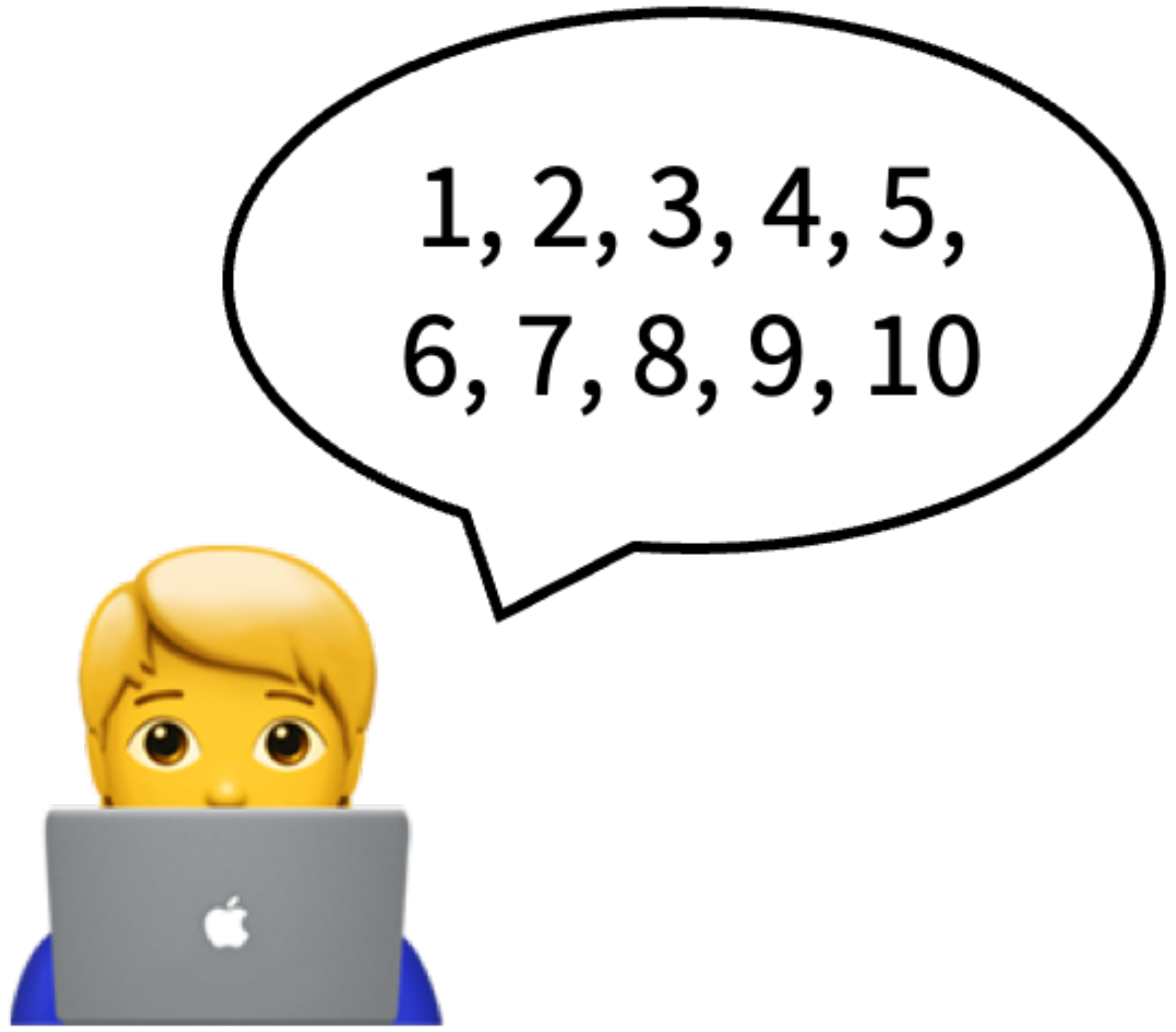
SYNTAX

Count from 1 to 10.

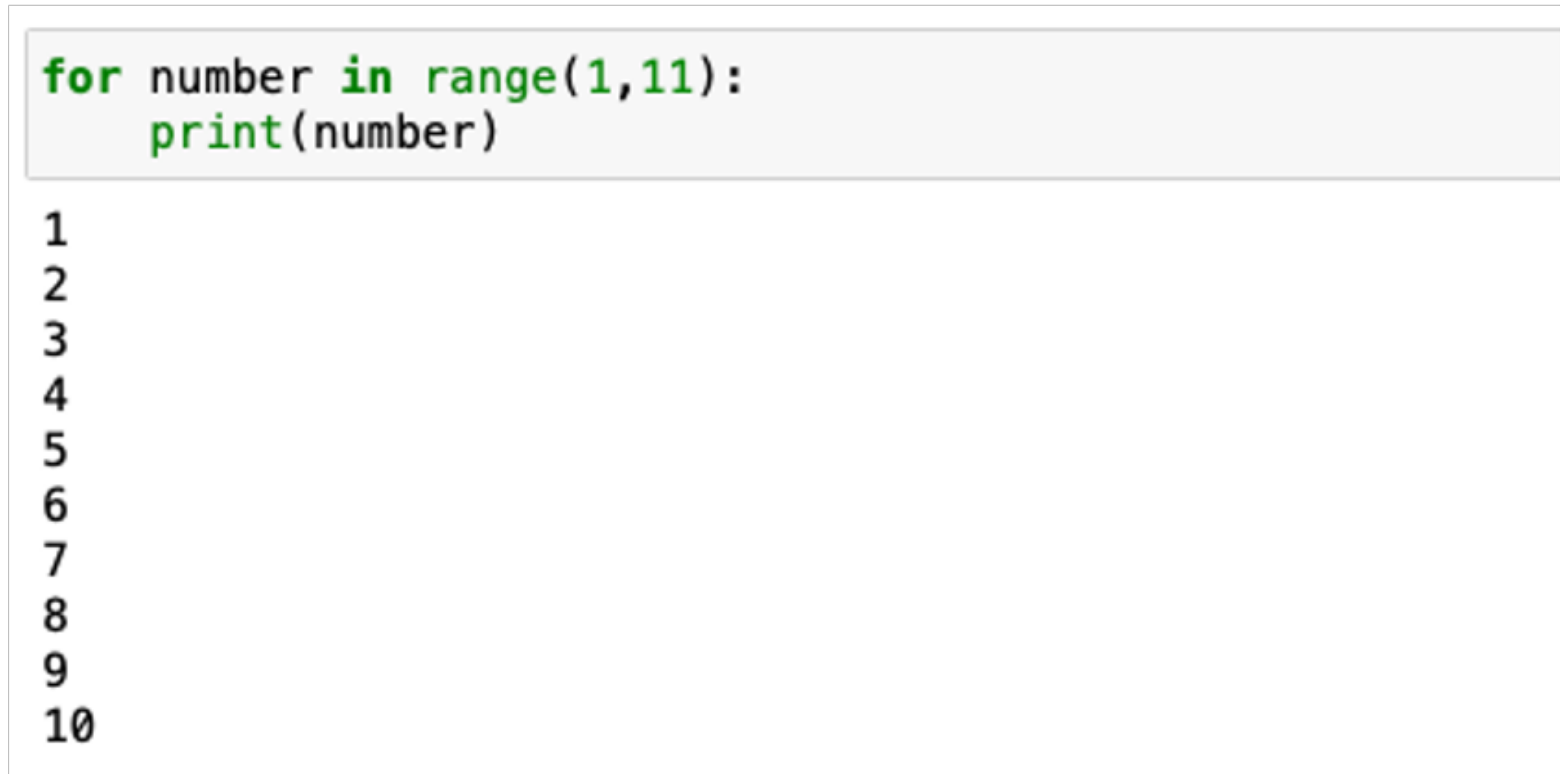
```
for number in range(1,11):
    print(number)
```

THE INTERPRETER

Count from 1 to 10.



```
for number in range(1,11):
    print(number)
```



```
for number in range(1,11):
    print(number)
```

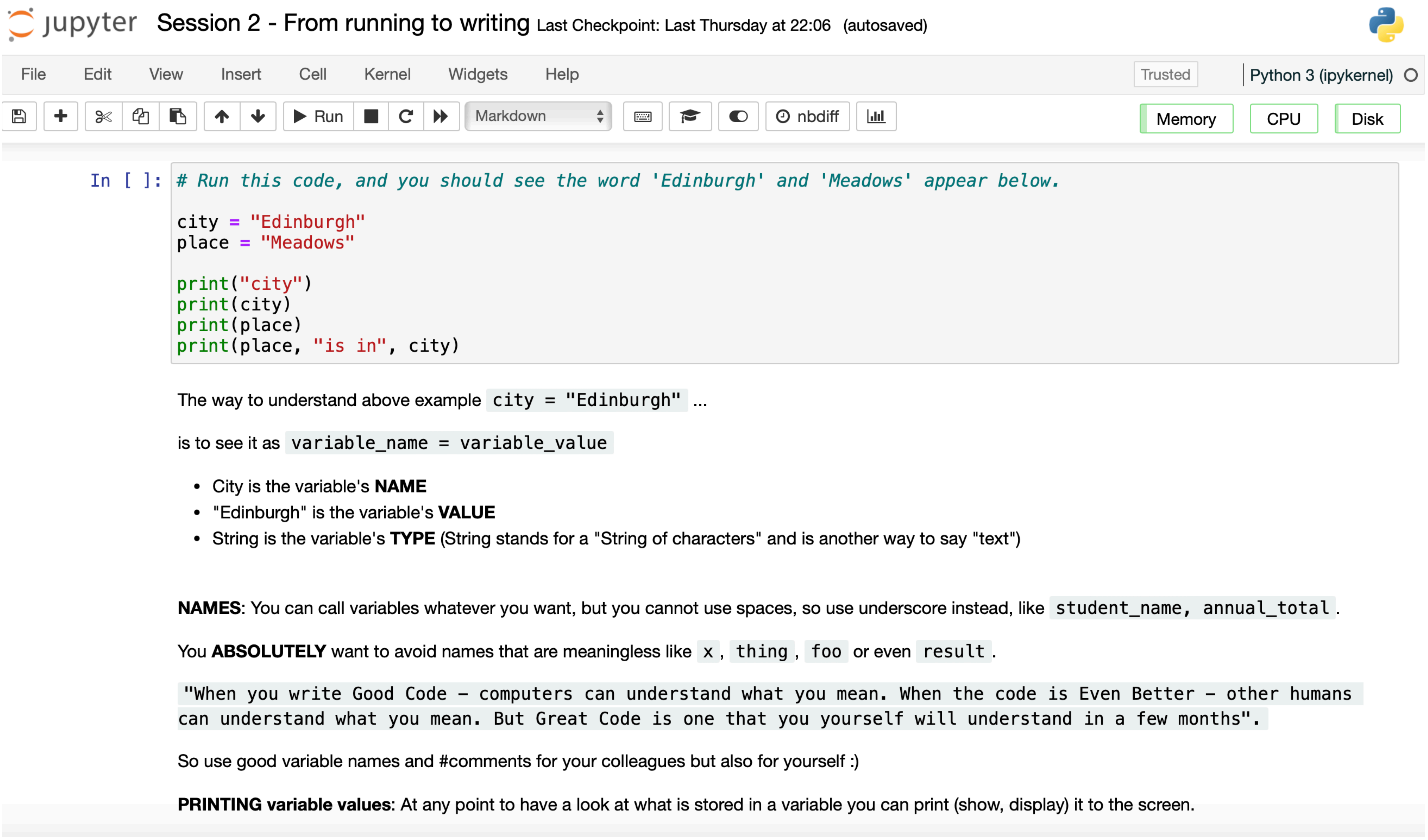
```
1
2
3
4
5
6
7
8
9
10
```

JUPYTER NOTEBOOKS

Our choice of interpreter throughout the week will be Jupyter Notebooks. These have many benefits:

1. Easy to use and share with others,
2. can be run on the Noteable service,
3. very hard to break anything (properly).

JUPYTER NOTEBOOKS



```python
# Run this code, and you should see the word 'Edinburgh' and 'Meadows' appear below.

city = "Edinburgh"
place = "Meadows"

print("city")
print(city)
print(place)
print(place, "is in", city)
```

The way to understand above example `city = "Edinburgh"` ...

is to see it as `variable_name = variable_value`

- City is the variable's **NAME**
- "Edinburgh" is the variable's **VALUE**
- String is the variable's **TYPE** (String stands for a "String of characters" and is another way to say "text")

NAMES: You can call variables whatever you want, but you cannot use spaces, so use underscore instead, like `student_name, annual_total`.

You **ABSOLUTELY** want to avoid names that are meaningless like `x`, `thing`, `foo` or even `result`.

`"When you write Good Code - computers can understand what you mean. When the code is Even Better - other humans can understand what you mean. But Great Code is one that you yourself will understand in a few months".`

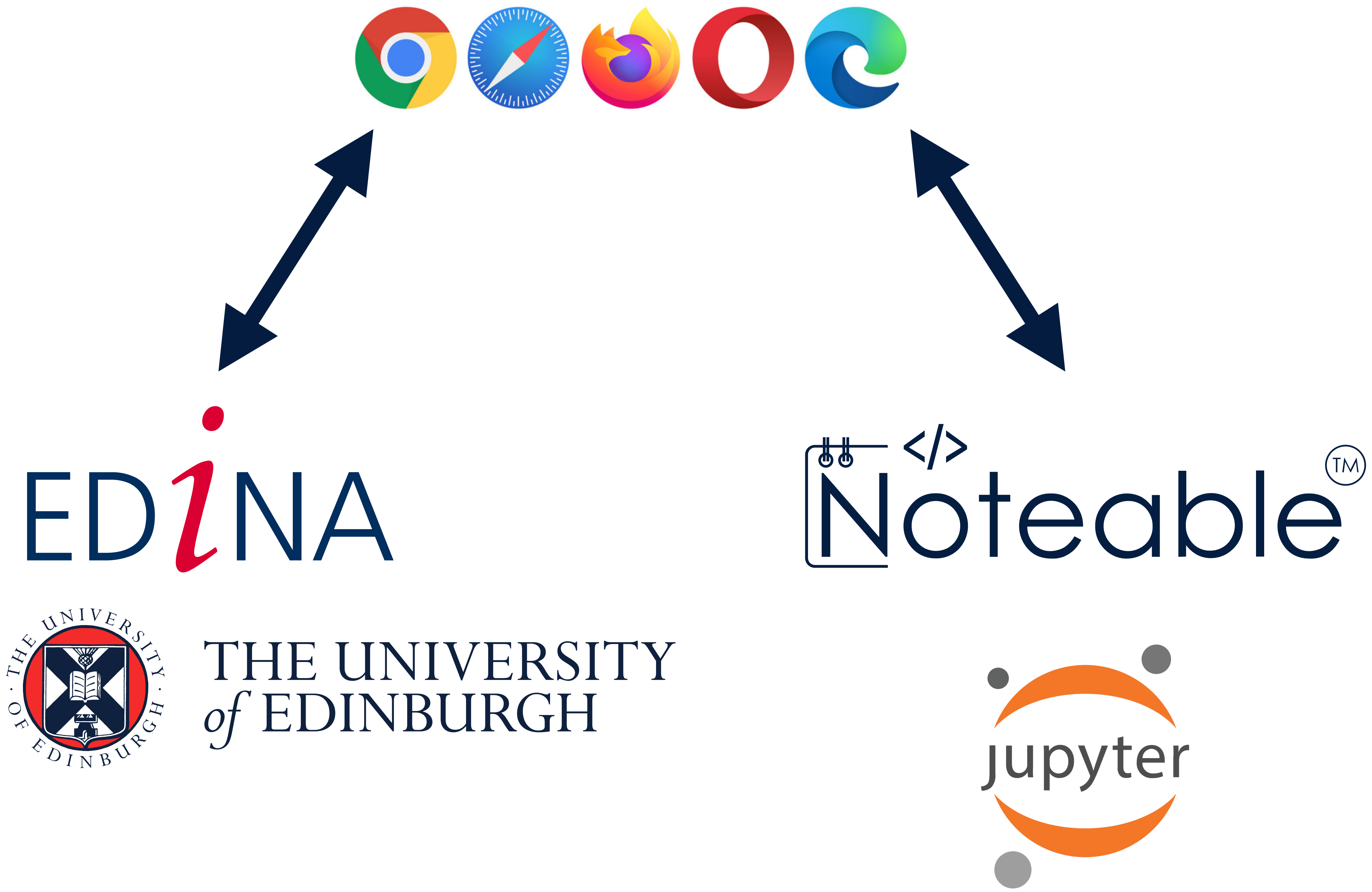
So use good variable names and #comments for your colleagues but also for yourself :)

PRINTING variable values: At any point to have a look at what is stored in a variable you can print (show, display) it to the screen.

} Code (Python)

} Text (Markdown)

HOW DOES NOTEABLE COME INTO ALL OF THIS?



Noteable is a university of Edinburgh service which provides online web-based access to a Jupyter Notebook ‘server’. As it is not local this makes it great and easy for research and teaching!

TOP TO BOTTOM

Code is written in lines. Each line does something, and they are executed from top to bottom.

```
count = 43
print(count)
```

____

```
count = 44
print(count)
```

____

```
count = 44
print(counter)
```

____

```
count = 44
print(counter)
```

____

```
count = count + 1
print(count)
```

____

TOP TO BOTTOM

Code is written in lines. Each line does something, and they are executed from top to bottom.

```
count = 43
print(count)

43

count = 44
print(count)

44
```

```
count = 44
print(counter)

ERROR
```

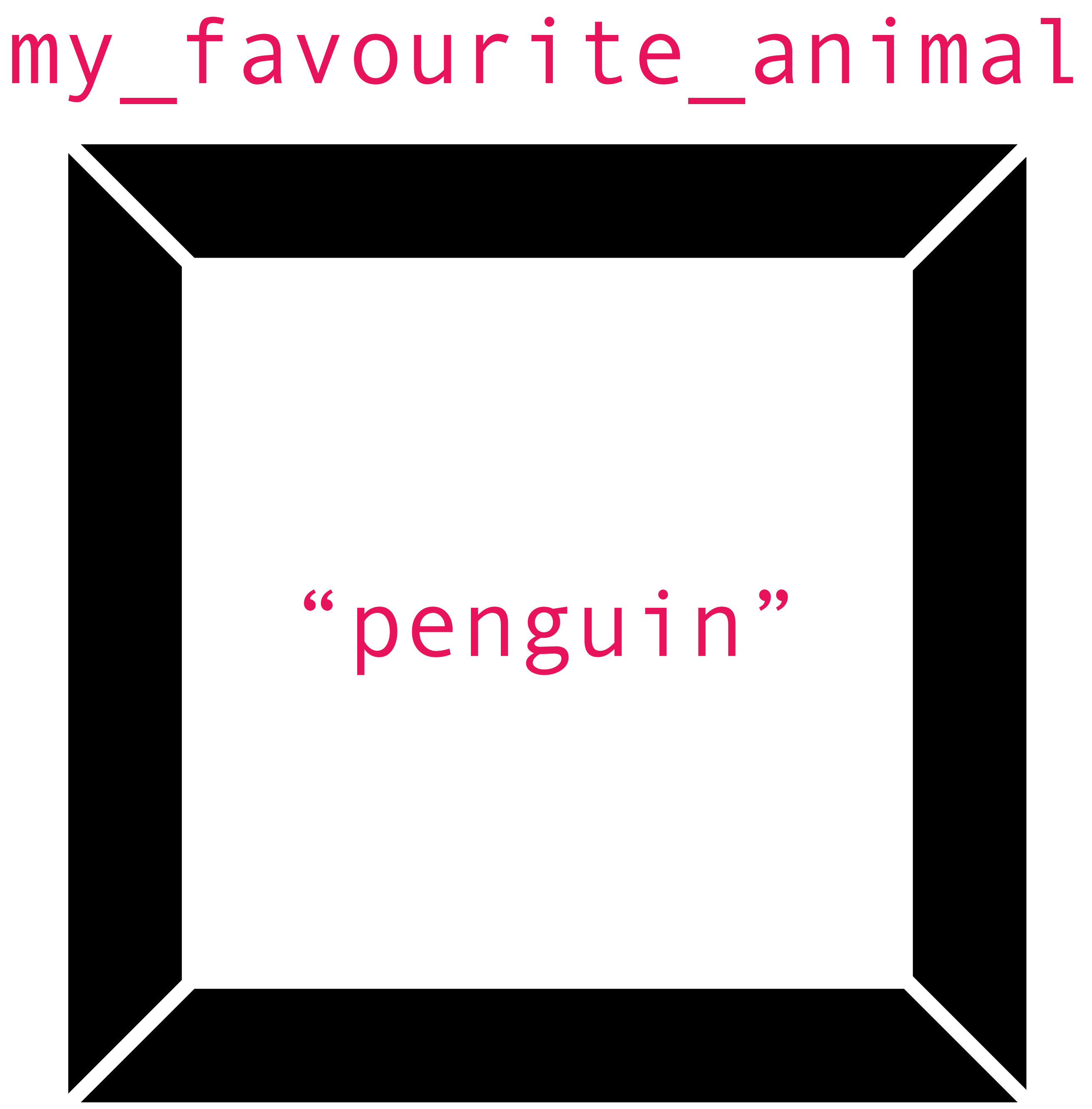
```
count = 44
print(count)

44

count = count + 1
print(count)


45
```

VARIABLES

Variables are places to store values for later.
There are different types of variables:

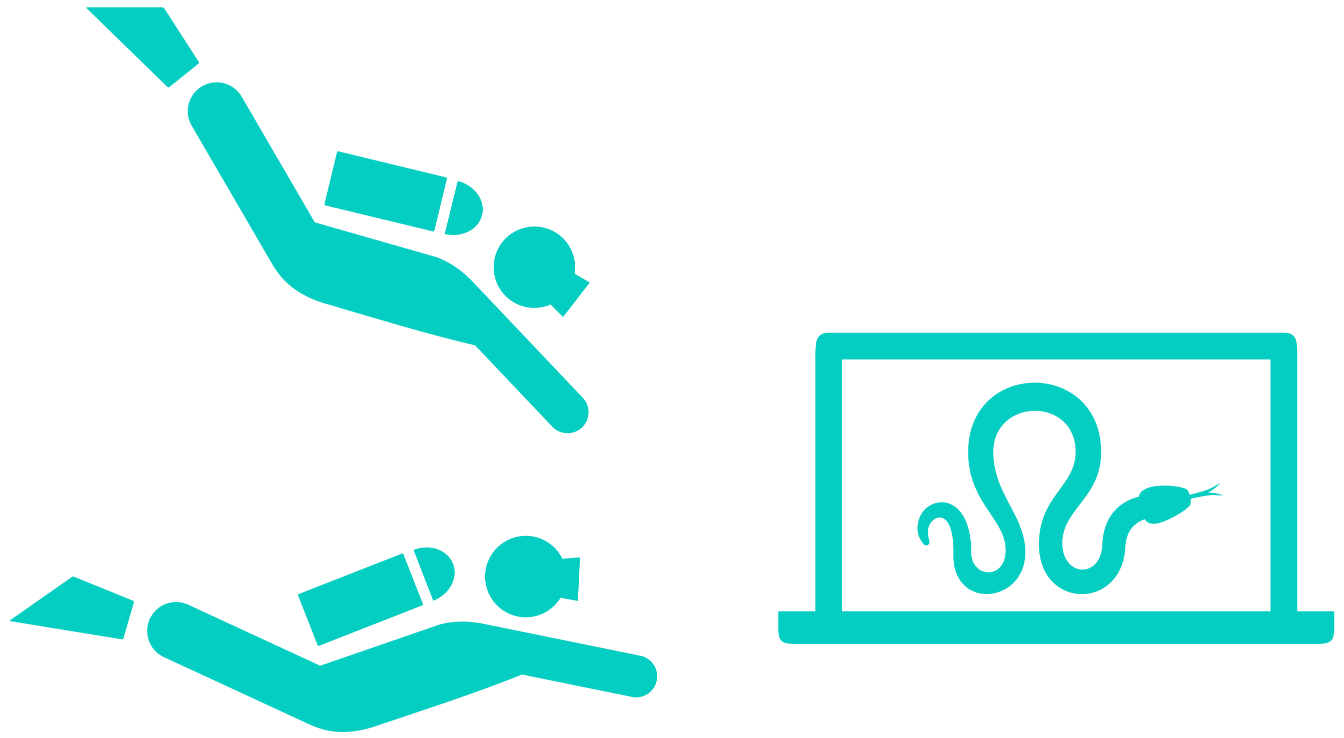
- **string** for text, e.g. *“penguin” or “I like Python”*
- **int** (integer) for whole numbers, *e.g. 1, 5, 2014*
- **float** for decimal numbers, *e.g. 2.25, 6.1246, 16.2*
- **bool** (Boolean) for logic values: *True, False*

```
my_favourite_animal = “penguin”
```



LETS GET PROGRAMMING

Session 1a: From Running to Writing



CONDITIONS AND LOGIC

COMPARING VARIABLES

```
2 == 2
2 == 3

"penguin" == "penguin"
"penguin" == “whale"
"penguin" == "Penguin"

"2024" == 2024
```

COMPARING VARIABLES

```
2 == 2                  True
2 == 3                  False

"penguin" == "penguin"  True
"penguin" == “whale"    False
"penguin" == "Penguin"  False

"2024" == 2024          False
```

COMPARING VARIABLES

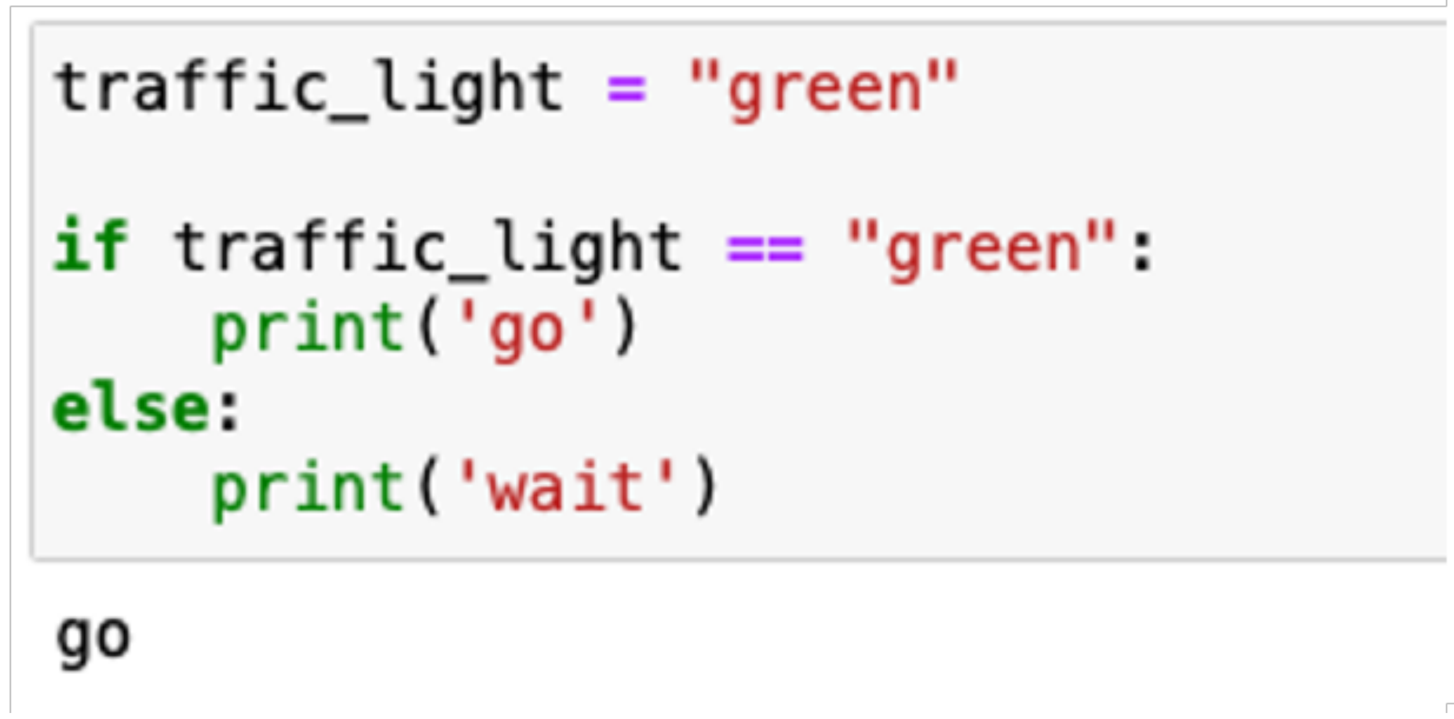
```
2  <  2
2  <  3
3  >  2
2  <=  2

3  !=  2
3  !=  3
```

COMPARING VARIABLES

```
2 < 2     False
2 < 3     True
3 > 2     True
2 <= 2    True

3 != 2    True
3 != 3    False
```

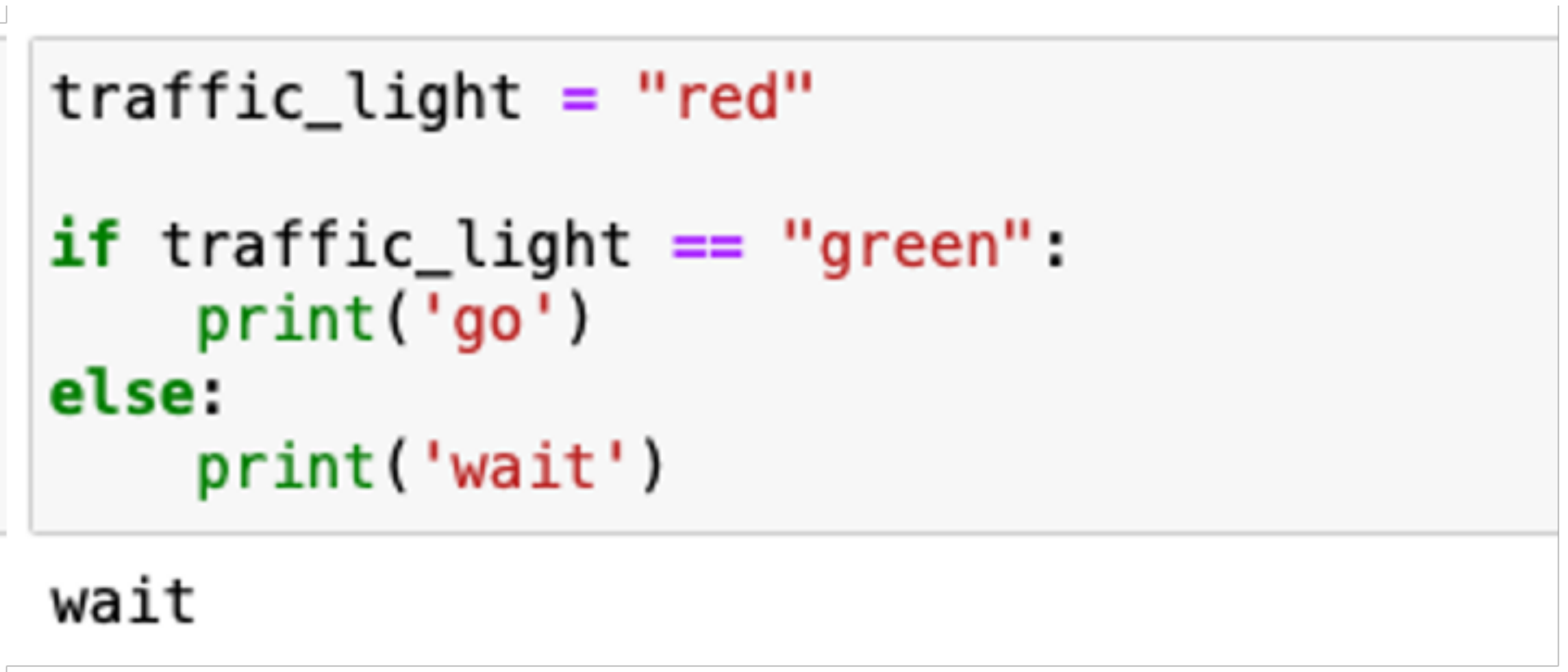
CONDITIONALS

Controlling what part of your code gets executed based on some conditions can be very useful in capturing real life conditionals. E.g. “If the traffic light is green, go, otherwise, wait.”



```
traffic_light = "green"

if traffic_light == "green":
    print('go')
else:
    print('wait')
```

```
go
```

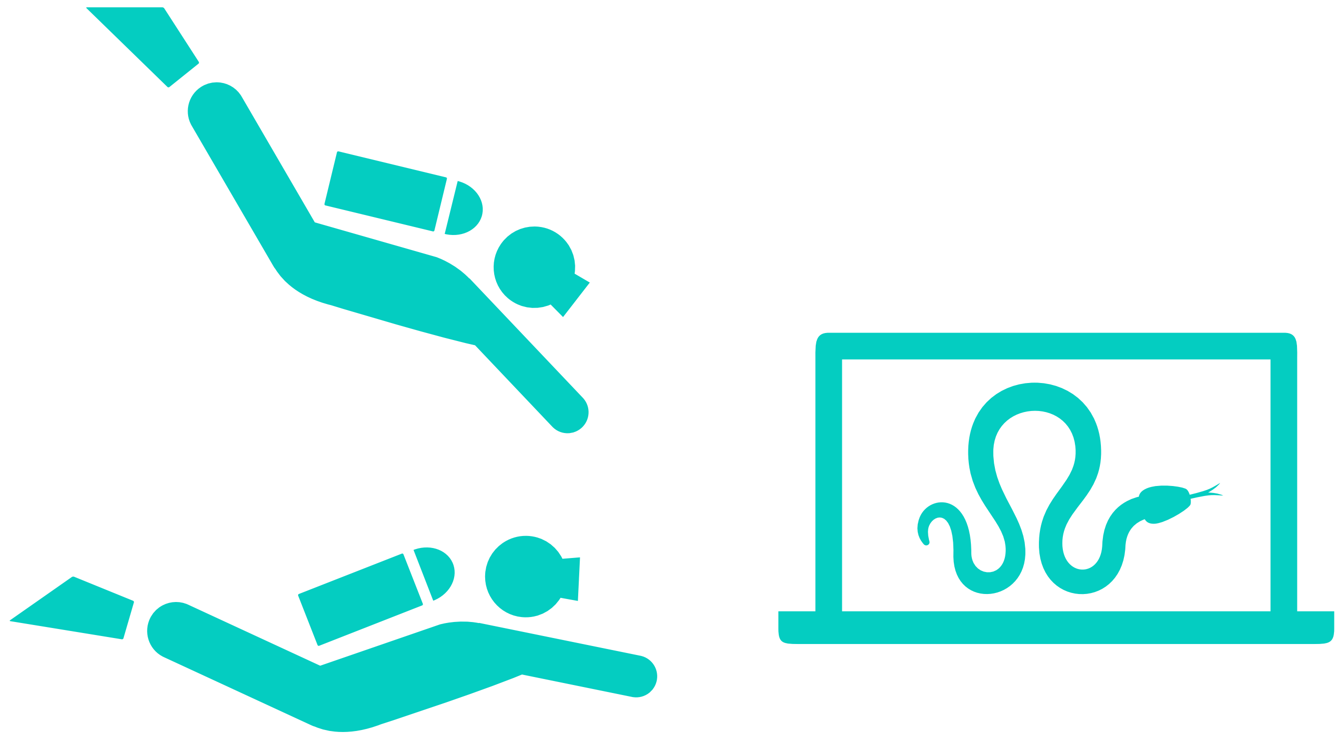


```
traffic_light = "red"

if traffic_light == "green":
    print('go')
else:
    print('wait')
```

```
wait
```

LETS GET PROGRAMMING

Session 1b: If (This AND That)



FUNCTIONS

PART 1

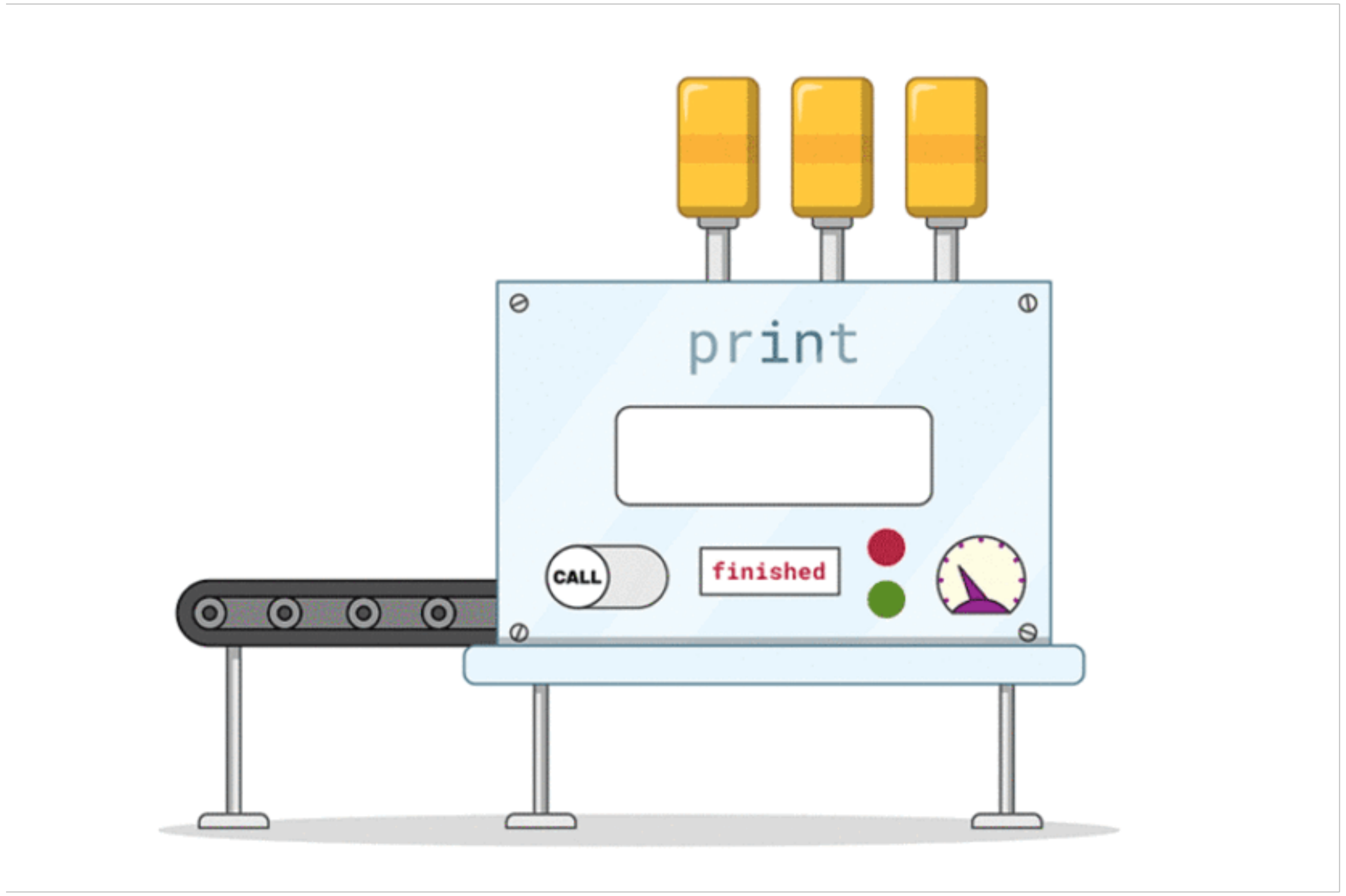
QUESTIONS TO ASK…

1.What is a function?

2. Why do we use functions?

3. How do I make a function in Python?

WHAT IS A FUNCTION?



WHAT IS A FUNCTION?

A way to generalise a process that will need to be done over and over again.

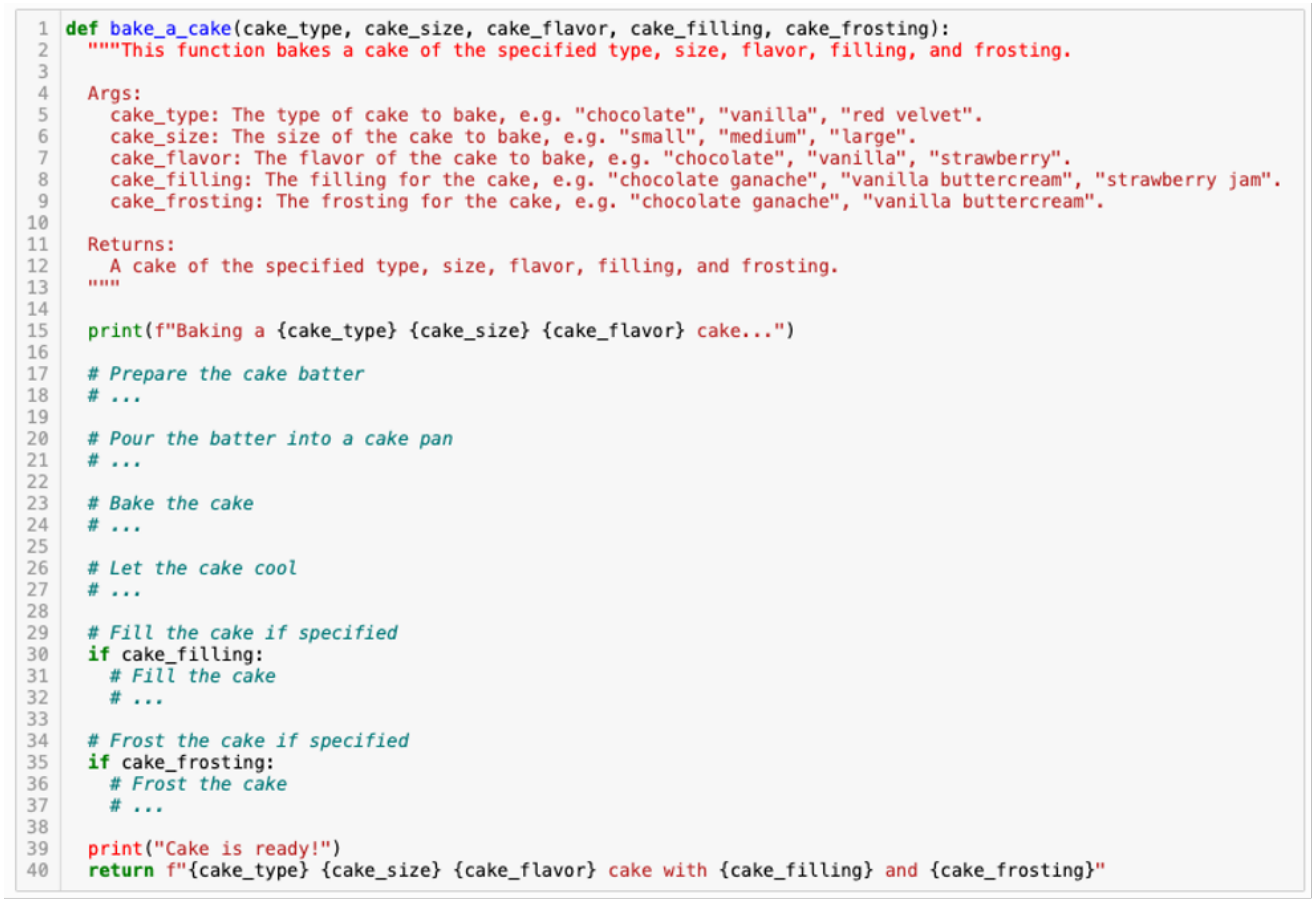
WHY DO WE USE FUNCTIONS?

- Reduce lines of code,
- Enhance computing performance,
- Make life easier!

WHY DO WE USE FUNCTIONS?

WHAT IS THE RECIPE FOR A FUNCTION IN PYTHON?

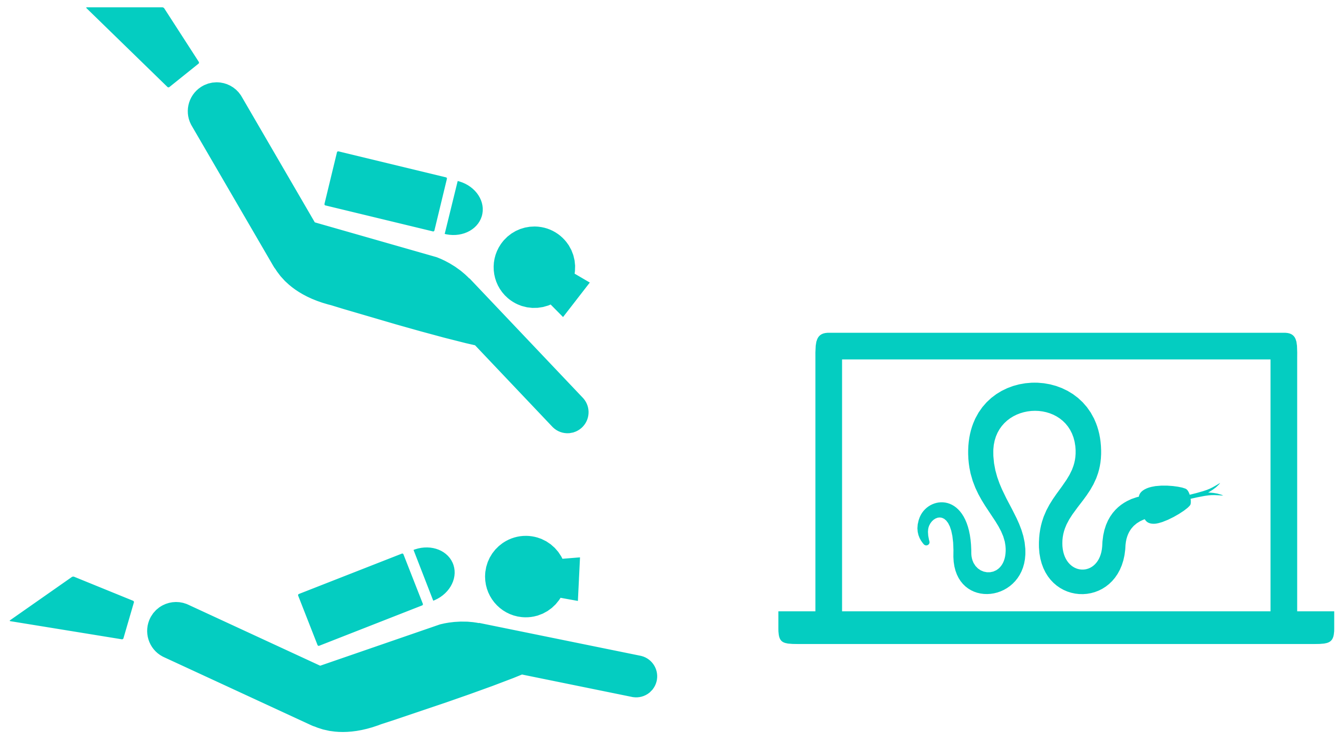
1. 'def'
2. Name
3. What goes into it (arguments)
4. What it does (the steps)
5. What it gives back (return value)



```python
def bake_a_cake(cake_type, cake_size, cake_flavor, cake_filling, cake_frosting):
  """This function bakes a cake of the specified type, size, flavor, filling, and frosting.

  Args:
    cake_type: The type of cake to bake, e.g. "chocolate", "vanilla", "red velvet".
    cake_size: The size of the cake to bake, e.g. "small", "medium", "large".
    cake_flavor: The flavor of the cake to bake, e.g. "chocolate", "vanilla", "strawberry".
    cake_filling: The filling for the cake, e.g. "chocolate ganache", "vanilla buttercream", "strawberry jam".
    cake_frosting: The frosting for the cake, e.g. "chocolate ganache", "vanilla buttercream".

  Returns:
    A cake of the specified type, size, flavor, filling, and frosting.
  """

  print(f"Baking a {cake_type} {cake_size} {cake_flavor} cake...")

  # Prepare the cake batter
  # ...

  # Pour the batter into a cake pan
  # ...

  # Bake the cake
  # ...

  # Let the cake cool
  # ...

  # Fill the cake if specified
  if cake_filling:
    # Fill the cake
    # ...

  # Frost the cake if specified
  if cake_frosting:
    # Frost the cake
    # ...

  print("Cake is ready!")
  return f"{cake_type} {cake_size} {cake_flavor} cake with {cake_filling} and {cake_frosting}"
```

LETS GET PROGRAMMING

Session 2a: Write the Recipe



FUNCTIONS PART 2

LOCAL AND GLOBAL

Local:
Only those close to him *(within the same function)* who know about what he can do.

Global:
Can be accessed by anyone, anywhere – everyone knows what he can do!

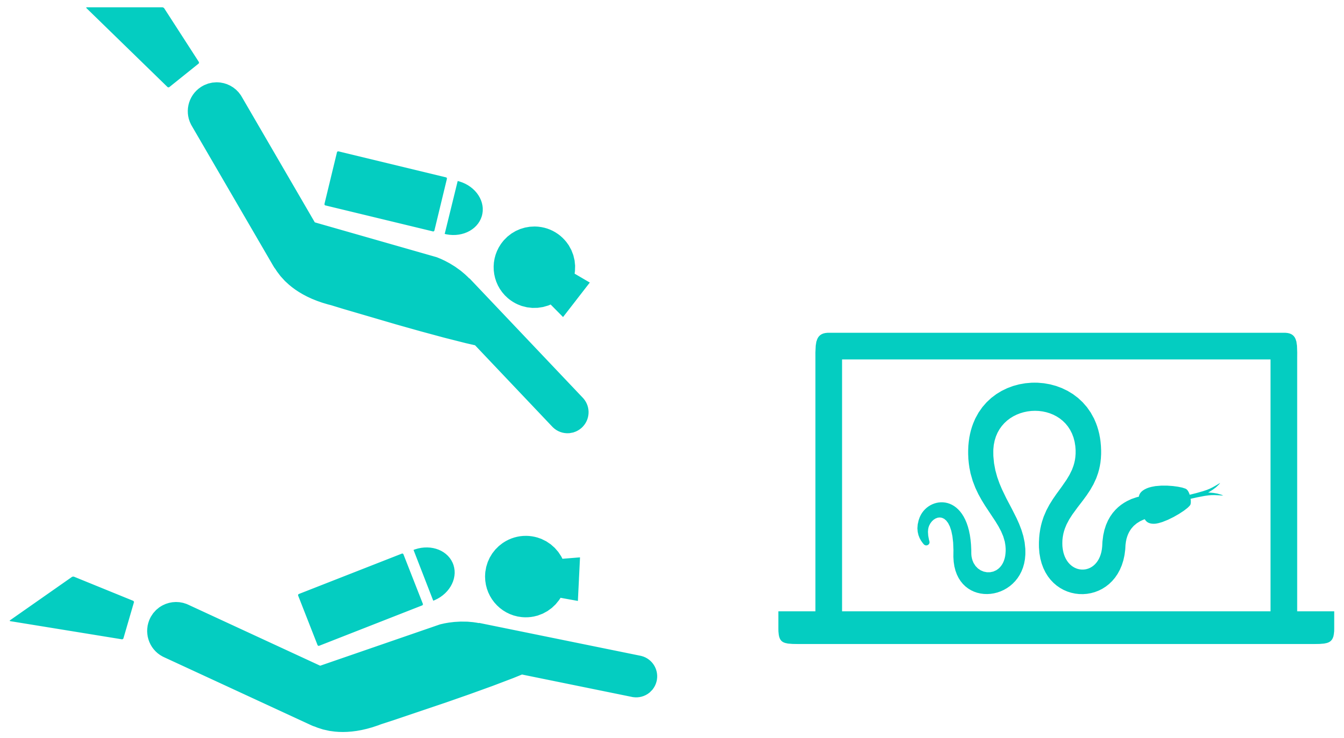
THE SCOPE RULES

Rule 1: Anything inside a function is mysterious to the outside...
You are not able to peek inside of a function elsewhere in code.
Only things returned will become available to the 'global' environment.

Rule 2: Functions can look outside, but shouldn't...
Things can get complicated when a function looks outside.
We tackle this by carefully specifying arguments with relevant names.

LETS GET PROGRAMMING

Session 2b: SCOPE



COLLECTIONS

LISTS

```
planet0 = "Mercury"
planet1 = "Venus"
planet2 = "Earth"
planet3 = "Mars"
planet4 = "Jupyter"
planet5 = "Saturn"
planet6 = "Uranus”
```

LISTS

```
planet0 = "Mercury"
planet1 = "Venus"
planet2 = "Earth"
planet3 = "Mars"
planet4 = "Jupyter"
planet5 = "Saturn"
planet6 = "Uranus”
```

```
planets = ["Mercury", "Venus", "Earth",
"Mars", "Jupyter", "Saturn", "Uranus"]
```

WHY USE LISTS?

```
planets = ["Mercury", "Venus", "Earth", "Mars",
"Jupyter", "Saturn", "Uranus"]
```

- ✅ Count how many items are in the list
- ✅ Check if a specific item is in the list
- ✅ Find the location of a specific item
- ✅ Add and remove items
- ✅ Sort (alphabetically or otherwise)

Pluto (not a planet)

ALTERNATIVES TO LISTS

Tuple

planets = ("Mercury", "Venus", "Earth", "Mars", "Jupyter", "Saturn", "Uranus”)

- Uses () instead of []
- Same as a list, except it cannot be changed after creating it

ALTERNATIVES TO LISTS

Set

```
planets = {"Mercury", "Venus", "Earth", "Mars",
"Jupyter", "Saturn", "Uranus”}
```

- Uses {} instead of [] (list) or () (tuple)
- Same as a list, except:
  - Every item is unique (items cannot be listed twice)
  - Order does not matter and will change (no indexing)

ALTERNATIVES TO LISTS

Dictionaries

- Used to store multiple pieces of information about one thing
- Uses key-value pairs: each piece of data (value) has a label (key)

mercury = {“name”: “Mercury”, “day_length”: 59, “hottest_temp”: 430}

ALTERNATIVES TO LISTS

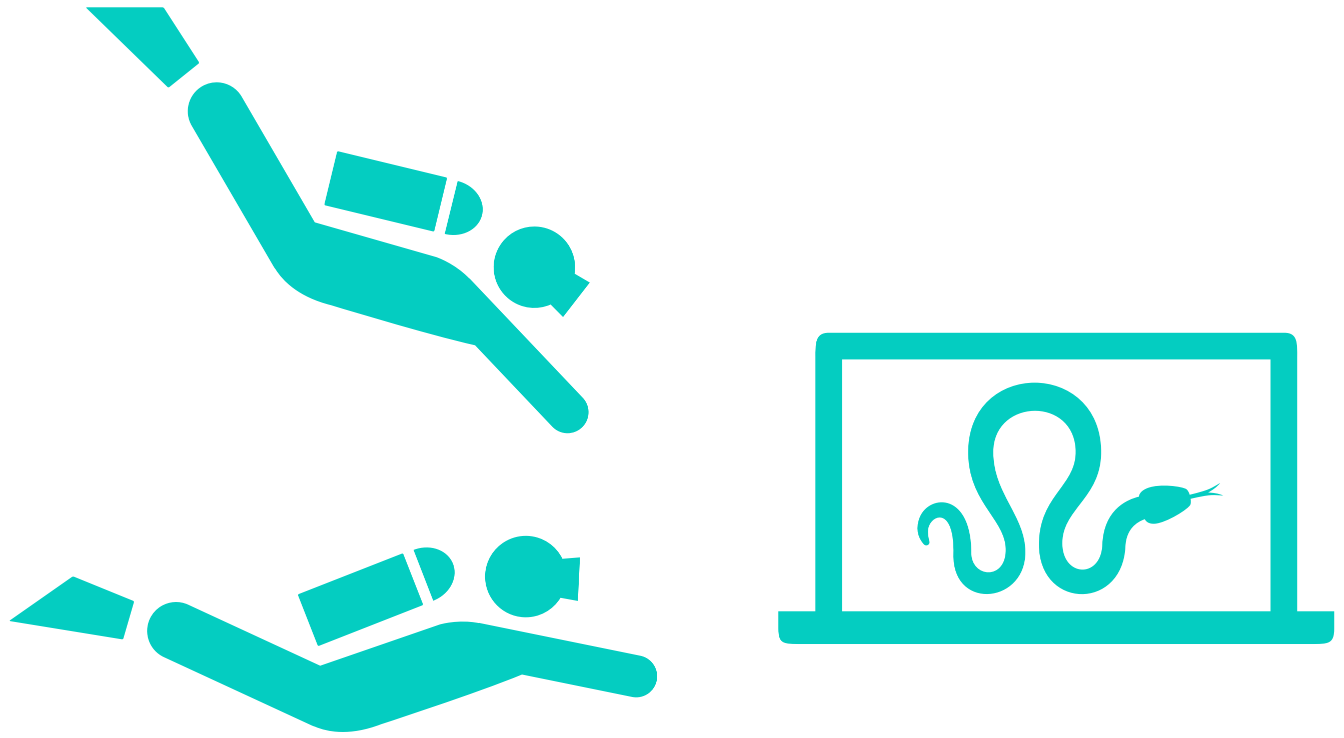
Dictionaries and Lists Together

- Combining lists and dictionaries is useful for real-world data: We often have multiple pieces of information about lots of different things and want to work with all of it at the same time!

```
planets = [
  {“name”: “Mercury”, “day_length”: 59, “hottest_temp”: 430},
  {“name”: “Venus”, “day_length”: 243.025, “hottest_temp”: 462},
  {“name”: “Earth”, “day_length”: 1, “hottest_temp”: 56.7},
  …
  ]
```

LETS GET PROGRAMMING

Session 3a: [({collections})]



LIST COMPREHENSIONS

WHAT ARE LIST COMPREHENSIONS

How do I get someone to pick the shirts from the wardrobe?

1. Say that it is the shirts you want,
2. For each item of clothing, check if it is a shirt,
3. If it’s a shirt, then take it out the wardrobe.

```
shirts = [
     item_of_clothing
     for item in wardrobe
     if item == shirt
     ]
```

```
initial_list = [thing1, thing2, …]
variable_name = [
     <thing to get new list of>
     for item in initial_list
     if item ==/>/< <some condition>
     ]
```

Some maths functions that may come in handy…

`max( ) / min()`

Get the largest/smallest element in a group. For letters it will mean 'highest/lowest in the alphabet'.

`len( )`

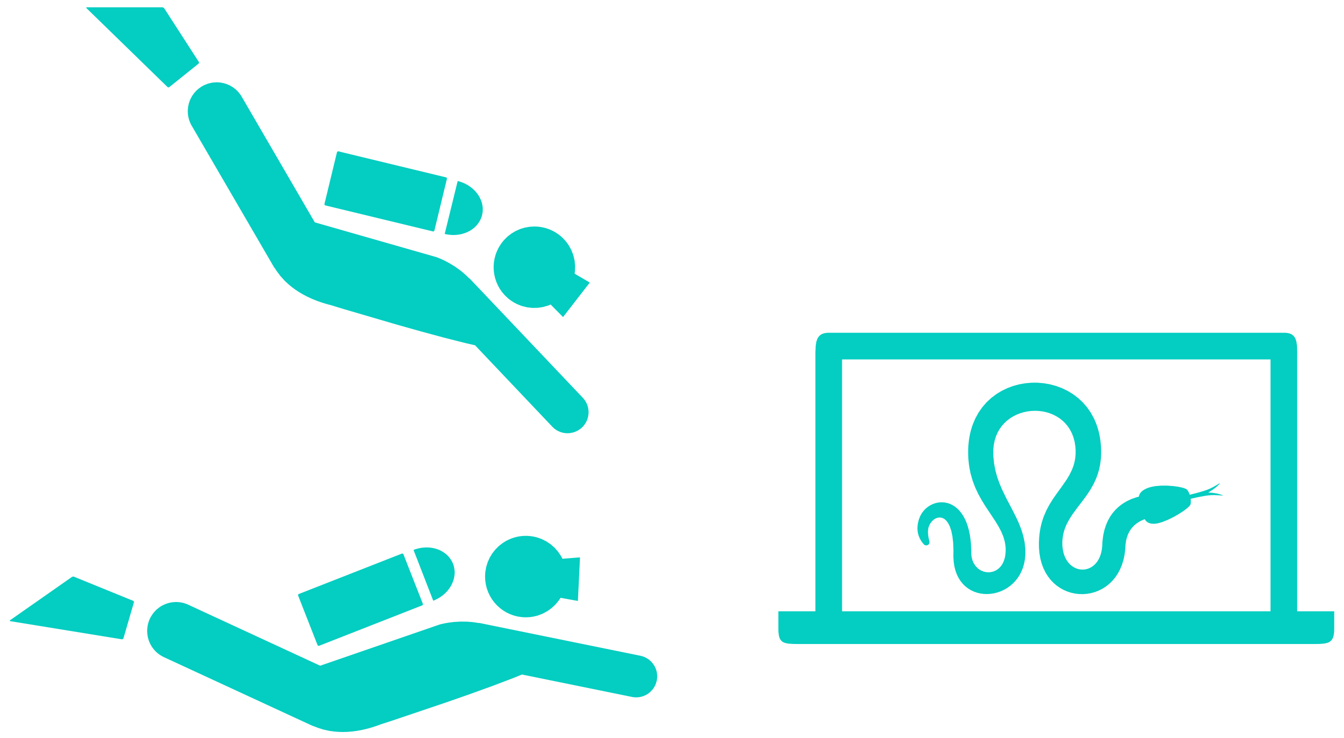
Size of the collection, can be used on lists, dicts, but also on strings.

`sum( )`

Combine all elements. Just used for numbers.

LETS GET PROGRAMMING

Session 3b: Lists of Lists



FINISHING UP

How to import data

To import data there is a range of methods, the easiest is using the package 'pandas'

Eg.

```
import pandas as pd
data_frame = pd.read_csv("<Your File Pathway>")
```

Additional Resources

- This course used (slightly modified) notebooks from Code Storytelling (http://www.codestorytelling.com). We covered badges 1,2,4,5,7, and 8. Consider working through some more of the notebooks and watching some of the videos.
- Think Python is an introductory Python book with many exercises, free to read online: https://greenteapress.com/wp/think-python-2e/

Feedback for us...

- We hope you've enjoyed the course as much as we did.
- It is really useful for us to hear your feedback

https://forms.office.com/r/YYNrqvuNr8
Should be really quick and only take 5 mins (maximum!)

Python

- Introduction to Text Analysis with Python (12th and 19th February, with Xan Cochran)

Other

- An Introduction to Machine Learning (14th and 21st April, with Chris Oldnall)
- Advanced Uses of LLMs (28th April, 5th and 12th May with Martin Disley