

INTRODUCTION TO PROGRAMMING WITH PYTHON



THE UNIVERSITY OF EDINBURGH
Centre for Data, Culture & Society



Welcome!

- **Introductions**
- **Can everyone access Noteable?**



www.cdcs.ed.ac.uk



THE UNIVERSITY OF EDINBURGH
Centre for Data, Culture & Society



Week 1: Writing and running code, Conditions & Logic



www.cdcs.ed.ac.uk

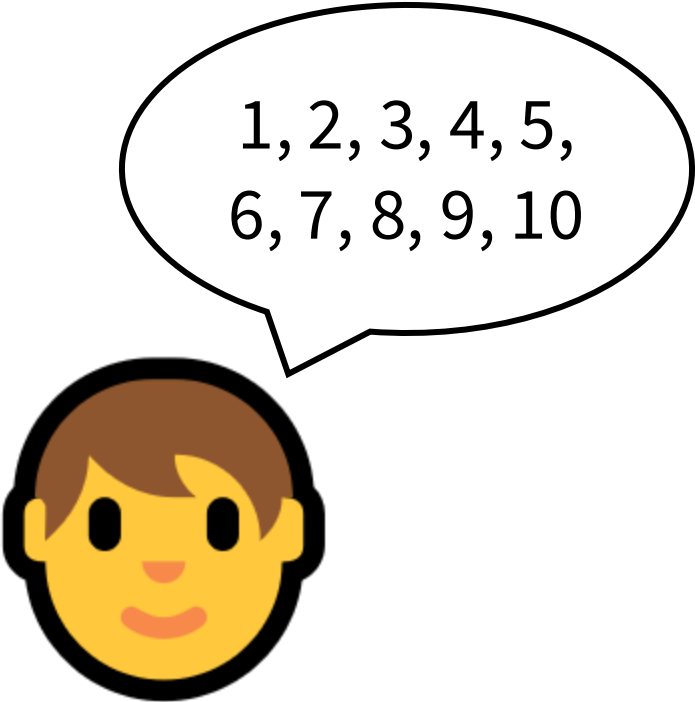
Programming = Telling the computer what to do

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



Syntax

Count from 1 to 10.



1, 2, 3, 4, 5,
6, 7, 8, 9, 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

2. There are **TEXT CELLS** like this one with explanations of concepts.
3. And **CODE CELLS** with Python code (see below). Code cells have a `In []` written to the left.
4. You can **RUN CODE CELLS** by clicking on them and pressing **Shift + Enter**. When you run a cell code in it is run (it "happens", computer will do what appear underneath the cell. and RUN code cells.

Jupyter Notebooks

✓ Remember to RUN ALL THE CELLS IN ORDER AS YOU GO THROUGH THIS NOTEBOOK (if you skip some, you might see some unexpected errors).

Learning objectives:

At the end of this badge you will know:

- How is code executed by Python.
- What are variables.
- That are variables types (string, int, float, bool), and how they impact on how code behaves e.g. $20 + 20$ is different than `"20"+"20"`.
- How to change variable types with Casting.
- How to ask user for some input.

→ SOON SPOILER ALERT:

You will also understand these lines of code:

```
In [ ]: # Remember – run all code cells from top to bottom as you go through notebook. (Shift + Enter)
# Lines starting with '#' are just comments, they output nothing when run.
```

```
In [ ]: # Running code and printing things.
# Prints appear immediately under the cell.

print(3 + 1)
print(3 + 2)
print(3 + 3)
```

```
In [ ]: # but Output/return of ONLY THE LAST LINE of code appears in the Out[ ]: section below your print.
# That's why printing is a nice way to know what's going on

2 + 1
2 + 2
2 + 3
```

Markdown cells

Code cells

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)
```



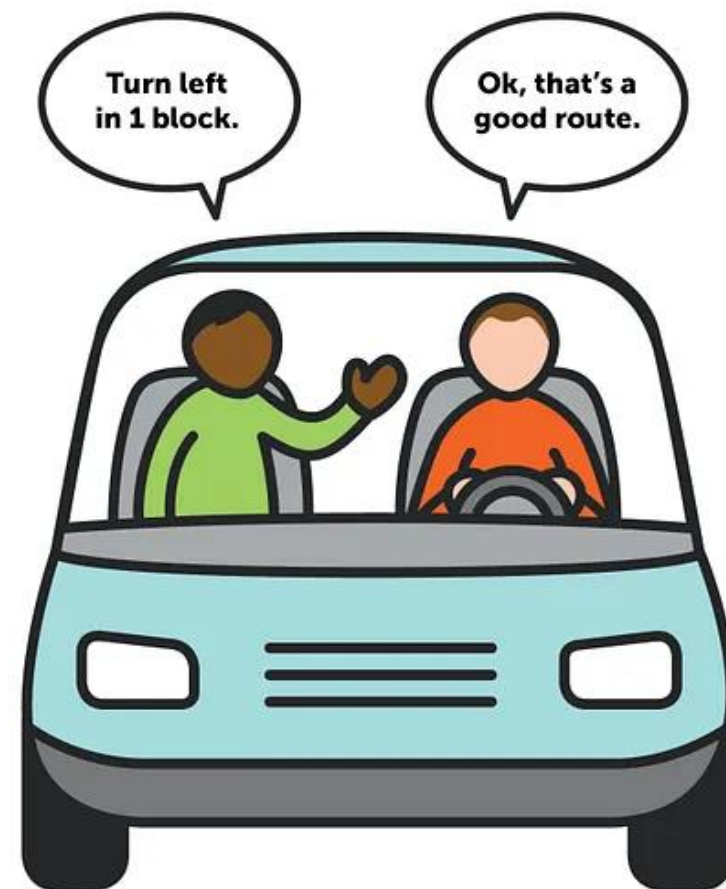
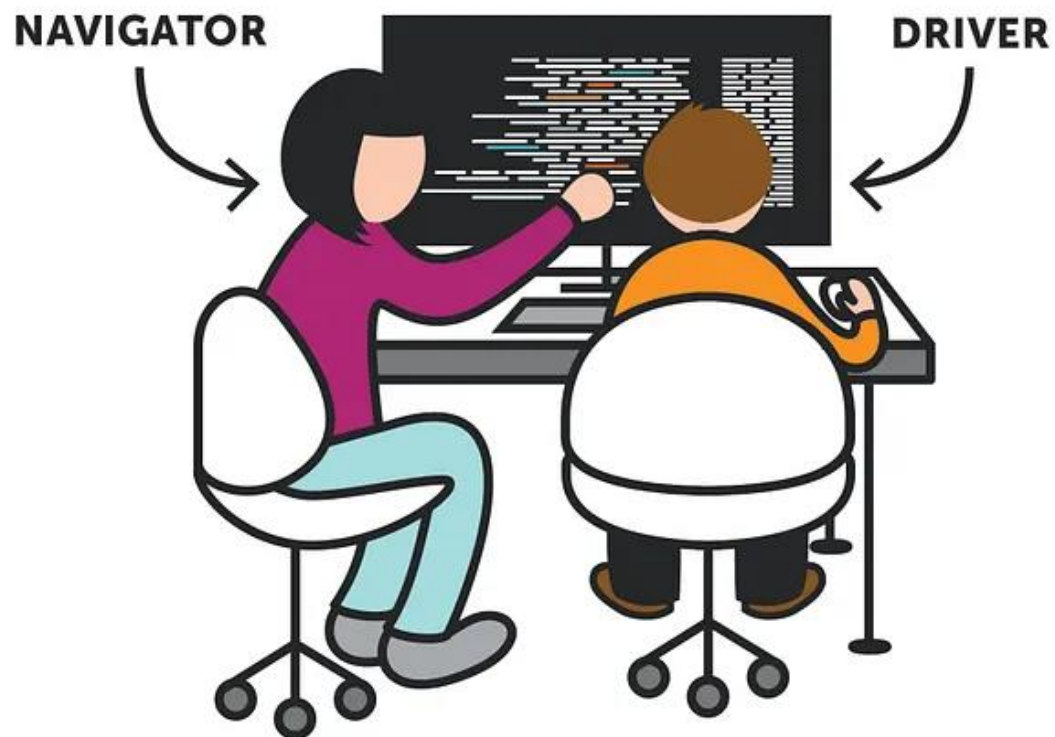
Variables are places to store values for later. There are different types of variables:

- **string** for text, e.g., “*banana*” 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_fruit = “apple”
```



PAIR PROGRAMMING



Pair Programming

- find a partner
- switch driver & navigator roles regularly, e.g., after every task in the notebook
- you can work with the same partner throughout the course, or switch between sessions
- ask us for help! (yes, even for small things)



Let's get programming – Notebook 1a

<https://noteable.edina.ac.uk/login>

- log in with your student/staff account
- select Python 3 and start the server
- click “+GitRepo” (top right) add the GitHub link below, and click “clone”.

https://github.com/DCS-training/IntroToPython_2023



Comparing Variables

<code>2 == 2</code>	<i>True</i>	<code>2 < 2</code>	<i>False</i>
<code>2 == 3</code>	<i>False</i>	<code>2 < 3</code>	<i>True</i>
<code>"apple" == "apple"</code>	<i>True</i>	<code>3 > 2</code>	<i>True</i>
<code>"apple" == "orange"</code>	<i>False</i>	<code>2 <= 2</code>	<i>True</i>
<code>"apple" == "Apple"</code>	<i>False</i>	<code>3 != 2</code>	<i>True</i>
<code>"2023" == 2023</code>	<i>False</i>	<code>3 != 3</code>	<i>False</i>



Conditionals

- Controlling what part of your code gets executed based on conditions
- 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





Let's get programming:

Notebook 1b – Conditions & Logic





THE UNIVERSITY OF EDINBURGH
Centre for Data, Culture & Society



Week 2:

Using and writing Functions



www.cdcs.ed.ac.uk

Welcome Back!

Whilst you are coming in, open up your web browser and load up Noteable:

<https://noteable.edina.ac.uk/login>

→ log in with your student/staff account

→ select Python 3 and start the server

→ from last week you should have all the files, if you do not then...

click “+GitRepo” (top right) add the GitHub link below, and click “clone”.

https://github.com/DCS-training/IntroToPython_2023



Last week...

- Write and run,
- logic and conditionals.

...this week...

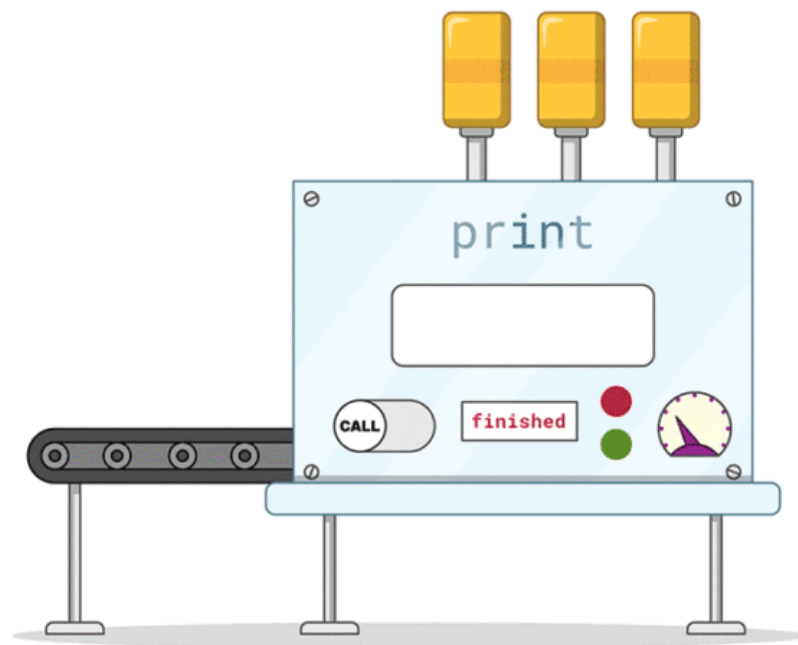
- Functions,
- and more functions!



- 1. What is a function?**
- 2. Why do we use functions?**
- 3. How do I create a function in Python?**



1. What is a function?



1. What is a function?

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



2. Why do we use functions?

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



2. Why do we use functions?



3. How do I make 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)*

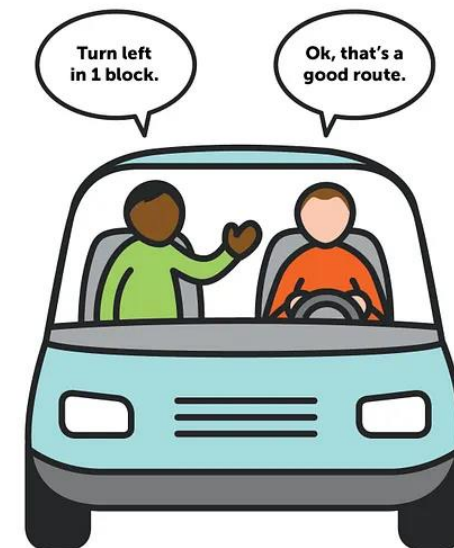
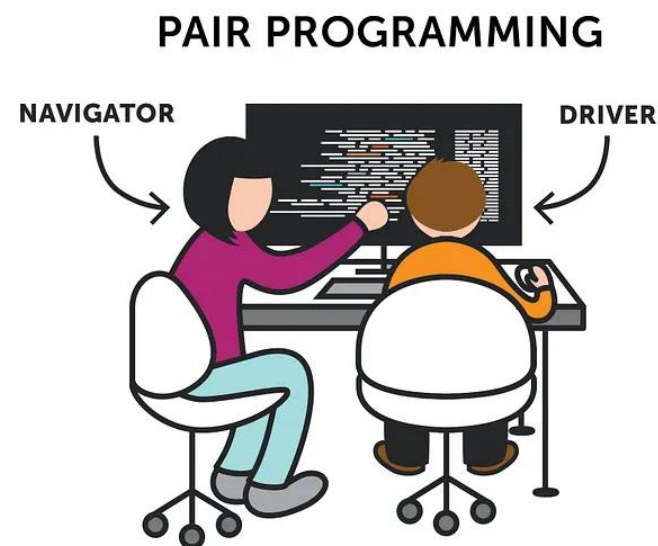


```
1 def bake_a_cake(cake_type, cake_size, cake_flavor, cake_filling, cake_frosting):
2     """This function bakes a cake of the specified type, size, flavor, filling, and frosting.
3
4     Args:
5         cake_type: The type of cake to bake, e.g. "chocolate", "vanilla", "red velvet".
6         cake_size: The size of the cake to bake, e.g. "small", "medium", "large".
7         cake_flavor: The flavor of the cake to bake, e.g. "chocolate", "vanilla", "strawberry".
8         cake_filling: The filling for the cake, e.g. "chocolate ganache", "vanilla buttercream", "strawberry jam".
9         cake_frosting: The frosting for the cake, e.g. "chocolate ganache", "vanilla buttercream".
10
11     Returns:
12         A cake of the specified type, size, flavor, filling, and frosting.
13     """
14
15     print(f"Baking a {cake_type} {cake_size} {cake_flavor} cake...")
16
17     # Prepare the cake batter
18     # ...
19
20     # Pour the batter into a cake pan
21     # ...
22
23     # Bake the cake
24     # ...
25
26     # Let the cake cool
27     # ...
28
29     # Fill the cake if specified
30     if cake_filling:
31         # Fill the cake
32         # ...
33
34     # Frost the cake if specified
35     if cake_frosting:
36         # Frost the cake
37         # ...
38
39     print("Cake is ready!")
40     return f"{cake_type} {cake_size} {cake_flavor} cake with {cake_filling} and {cake_frosting}"
```

3. How do I make a function in Python?

Pair Programming – Reminder

- work with your partner from last week, or find someone new
- switch driver & navigator roles regularly, e.g., after every task in the notebook
- ask us for help!



Let's do some programming:

Session 2 –

Functions = superpowers.

Let's try imagining the ORDER in which code is executed, in cycles or steps.

```
In [ ]: # Think about the order in which a computer needs to find answers to some questions.  
        # In order to then seek answers to other questions  
  
        # E.g. you need to know what int("20") is before you can add it to 20.  
        # And only once you add them you can print the result.  
  
        print( int("20") + 20)
```

```
In [ ]: # Same here. As a human, which parts would you have to solve first?  
        # try to solve it before you run the code. What's the result you expect?  
  
        print( (20 + 50) / 7 == (1+4) * (9-7))
```





What is scope all about?





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!



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.





Let's keep programming:

Session 2 –

Functions = superpowers.





THE UNIVERSITY of EDINBURGH
Centre for Data, Culture & Society



Week 3:

Lists and collections



www.cdcs.ed.ac.uk

Recap: Variables

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

- **string** for text, e.g., “*banana*” 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_fruit = “apple”
```



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

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)



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}
```

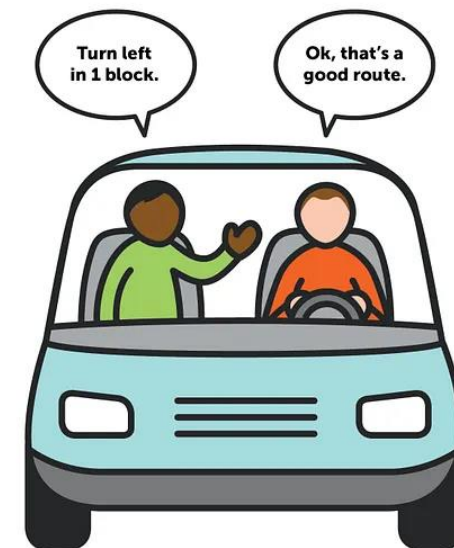
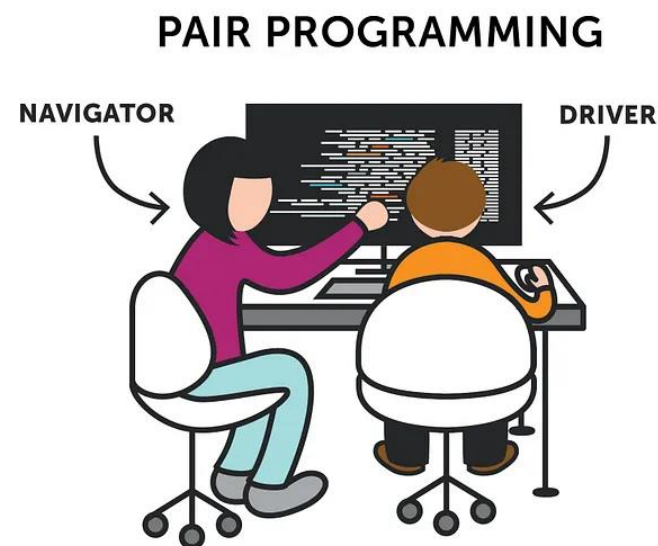
- Combining lists and dictionaries is useful for real-world data: We often have multiple pieces of information about lots of different things and want to work with all of it at the same time!

```
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},  
    ...  
]
```



Pair Programming – Reminder

- work with your partner from last week, or find someone new
- switch driver & navigator roles regularly, e.g., after every task in the notebook
- ask us for help!





Let's get programming:

Session 3a – Collections





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>  
]
```



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.





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.





Let's keep programming:

Session 3b –

List Comprehensions



Finishing up...

- **How to import data**
- **Additional resources**
 - **Feedback for us**



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!)



Thank you from us!

Python

- Introduction to Data Analysis in Python (26/11-03/12/2024, with Aislinn Keogh)
- Silent Disco: Facial Recognition in Historical Photographs with Artificial Intelligence in Python (23/09/2024, 14:00–16:00, with Martin Disley)

Other

- Build Your Personal or Project Website with GitHub Pages (4-11/11/2024, with Sarah Schöttler & Aislinn Keogh)

