# Intro to Programming

## Session 0 - Foundations of Python for Data Science

DIME & DECDG

DEC, The World Bank Group

## Session overview

## Scope of this session

The *DEC Foundations of Python* course does not require any background in programming.

If you have no programming background, then this session will prepare you for the rest of the course.

To successfully solve data work tasks using Python code, you need to learn two skills:

- Learn how to write Python code
- Learn how to approach a task like a coder

**Content of this session:**

1. Why programming? Work flow in data work programming
2. Why Python?
3. How does one write "*good code*"?
4. Some common concepts in programming

## What we assume about you:

Python can be used for countless of topics in computer science. However, the scope of this course is based on:

1. Participants want to use Python for data work. Numeric data, text data, geo-data etc.
2. This course does not cover how to use Python for software development, backends for websites etc.
3. Participants have some experience working with data. For example, extensive work in Excel or some work in Stata or R.

X
STATA
R
YOU
python

## Questions?

## Why programming?

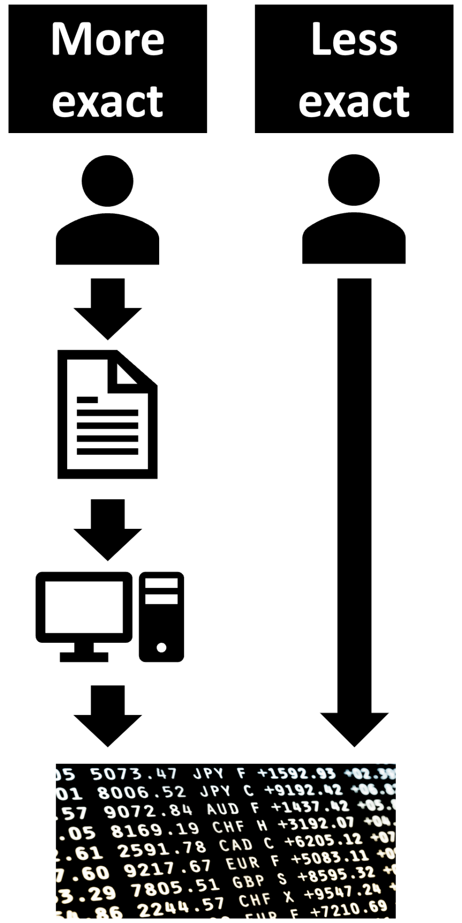
## Work flow in data work programming

## Why use programming for data work?

Data work should be very exact. Computers are more exact than humans.

Instead of manually working directly on the data, we write code that are instructions for computers on what to do with the data.

Since computers are very exact, our code becomes an exact record of what was done to the data and how outputs were generated.



## Work flow - Excel vs. programming

| | Excel | Programming |
|---|---|---|
| Manual edits in the files containing data? | Yes, raw data, computations, outputs etc., in the same file | No, manual edits in code files that are seperate |
| Write instructions for a computer to modify our data? | Yes, this is an option (Excel formulas) | Yes, and this is the only way (code) |
| Mix computer instructions and manual edits? | Yes, and it is common. | No, code should always start from unaltered raw data |
| An unambiguous way of sharing exactly what was done to the data? | Only formulas are shown, not manual edits | Yes, it is the code |
| Is it easy for a reviewer to reproduce all steps? | Its challenging or impossible to follow any manual edits as there is no record of them | Yes, anyone who has the code and have access to the input data can reproduce everything |

## Programming is like writing recipes

(*but more exact recipes*)

### Sponge cake:

- Set oven at 200C
- Melt 200g butter and mix with 200g sugar
- Mix in 4 eggs - one at the time
- Mix in 200g flour
- Bake in the middle of the oven for 30 min

### Data science visualization:

- Get GDP per capita data from for all countries
- Get investment data from all WB client countries
- Remove data that was not found in both dataset
- Sort ascendingly on GDP per capita
- Export a scatter plot and format the colors and the text

## All aspect of data work can be made exact with code

If code is a recipe, then the ingredients of data work is the project's original data.

While organic ingredients in food can be slightly different each time, **code inputs can and should be made as exact as the code**.

Derivatives of original data (new data products and analysis outputs) should be created from code and original data only. They will be as exact as the code and inputs are.

How to make original data very exact depends on the type of source of data:

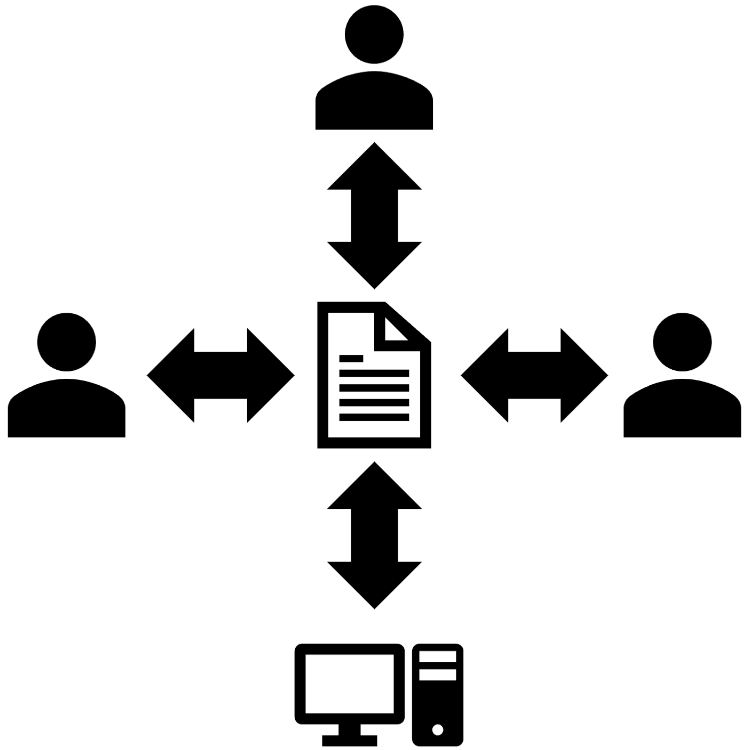
- **Primary data**: Proper back-up of raw data
- **Secondary shared data**: Cite source *and* proper back-up of data exactly as it was received
- **Secondary published data**: Cite source

## Code is how you communicate about data work

Chefs use recipes to communicate about food, architects use blue prints to communicate about future buildings etc.

Data scientists use code to communicate about data work.

A cook book of poorly written recipes will lead to bad food. Poorly written code will lead to bad communication about data work.



Code is not just a means to an end - **code is an end in itself**

Code is a comprehensive and exact method
to communicate about a project's data work

**The code is an output** just as much as
the project's final report, data visualizations, etc.

## Questions?

## Why Python?

## Why Python?

Why are there so many programming languages?
How does Python compare to other languages?

**Free/Open-source**: Python is and will always be 100% free to use

**Easy syntax**: Python code reads more similarly to regular English

**General purpose**: Python can be used for data science, servers, software etc.

**Widely used**: Many users means many user developed tools

**#1 in data science**: The language with most data science tools

## Questions?

## How does one write good code?

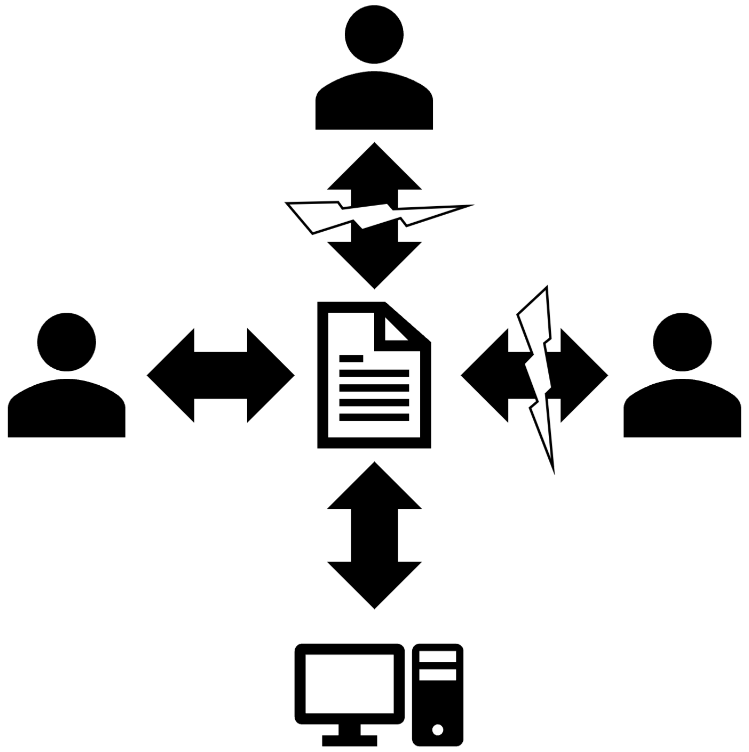
## Learn how to communicate about data work well

**Correct code:** Code that generates correct results

**Good code:** Code that can be understood and used by others

Code should be **both** correct code and good code. Only then is code useful as an output of your data work.

Start by writing *correct code*, then keep going to write *good code*.



## Errors and bugs

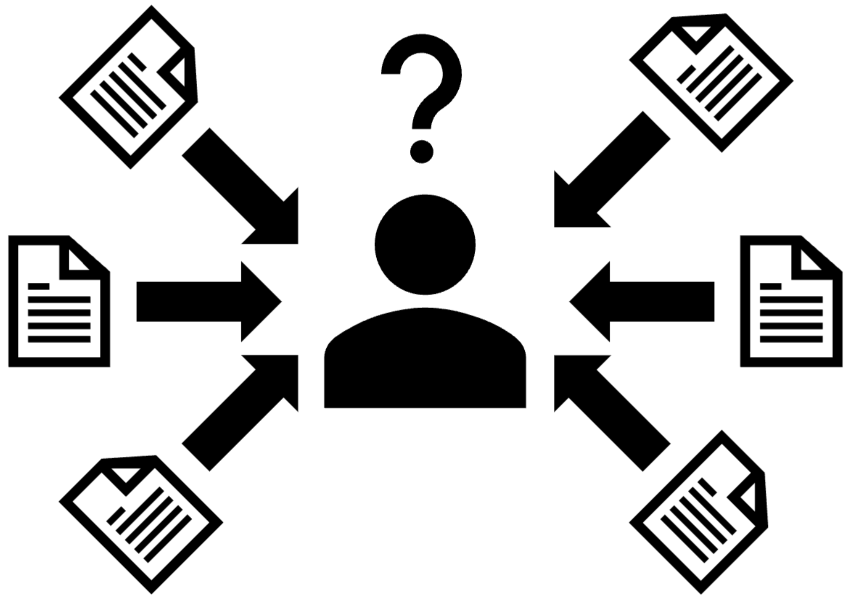
- **Writing correct code usually takes many errors** - errors should be your friends!
- Don't be afraid errors as code should not edit the original data!
- Great coders gets errors - but great coders know how to read error messages
- Learn how to interpret error messages - Google them
- You will have thousands of errors and bugs - embrace errors, run your code and find out where the bug is

## Learn good code from expert coders

**Be careful** - great code examples can be deceptive

Example code is often only meant to explain how to do something complex *correctly*, and not how to write *good code* in a real-life setting.

An advance example might not fit your project, regardless how *correct* it solved a well-defined school book example. Use critical thinking.



## When your code works you are only half-done

- ancient proverb

## A path towards good code

Ask yourself "*Would someone with your level of Python knowledge be able to follow the code?*"

- Simplify your code
- Document your code
- Peer-review your code

Settling for "*just*" correct code does not save you or your team any time in the long run.

## Documentation - comments

All programming languages has a feature called *comments*.

Comments are text in the same file as the code that the computer skips when running the code.

Comments' only (but important) purpose is to provide context for a human reading the code.

Comments best compliment the code if they explains why the code is written a certain way.

| OK comment | Good comment |
|---|---|
| *What* code does | *What* code does **and** *why* |
| `Load data` | `Loading region definitions shared by NGO used in sampling` |
| `Deleting variable "id" and "name"` | `Removing identifying variables to create a non-confidential data set` |

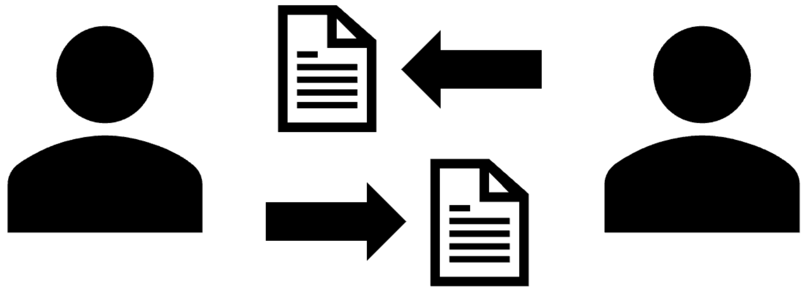
## Peer-review and learn about good code

The feedback you get will improve your code - would you submit a report without a proof read?

Peer-reviewing each other's code is the best way to learn how to write better code.

A peer is likely to be at about the same skill level as you and write code on a topic relevant to you.

Ask yourself "*What would make this code easier to follow?*"



## Re-use code

We can easily re-use code within and across projects
as code is separate from inputs and outputs.

Three levels of re-using code:

- **Copy-paste:** Copy code is ok. Use critical thinking and modify to fit your context
- **Functions:** A block of code that can be referenced across a project
- **Libraries:** Code packages developed by others to solve difficult tasks but made easy to use in any project

---

**Do not reinvent the wheel:**

- Aim to have each section of code be built on someone else' efforts
- Someone else's efforts can be the efforts of past you
- Compare with Excel - how do you build on the efforts of other users?

## Questions?

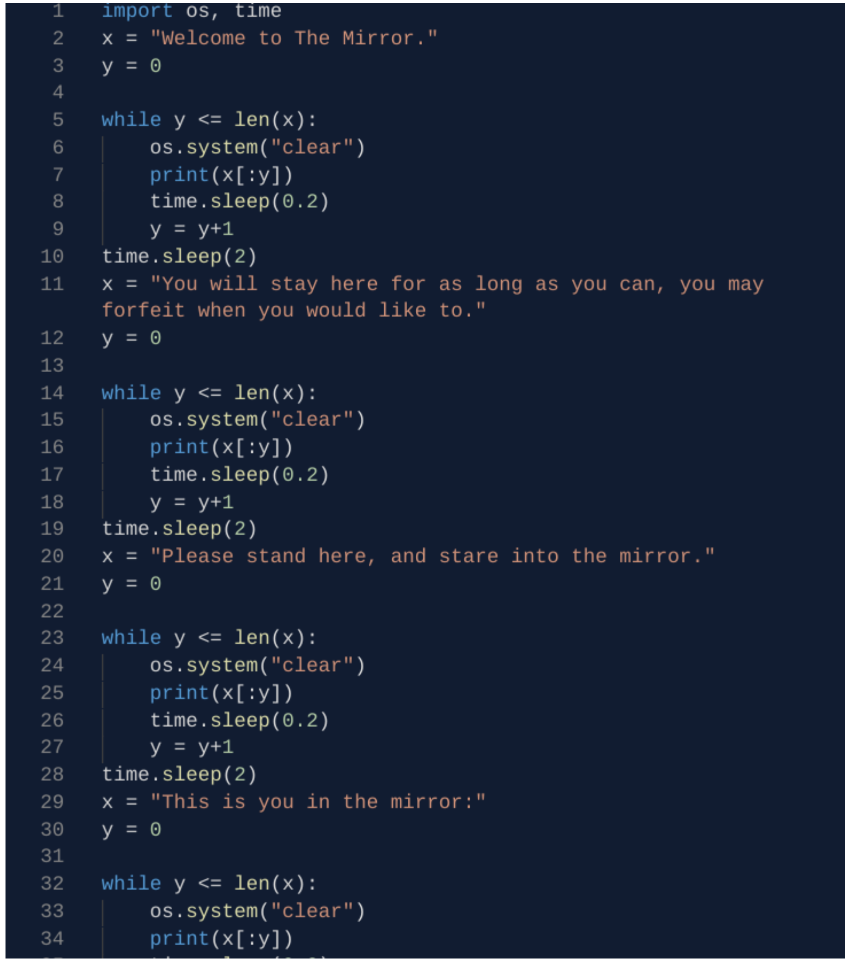
## Some common concepts in programming

## What is a script?

**A script is file with code** that is meant to run in a certain order, usually from top to bottom (*not a formal definition*).

**A project often consists of several scripts.** If they are not fully independent, then the order to run them should be clear.

Compare again to recipes that you follow top to bottom, and how you can have sub-recipes for common small tasks.



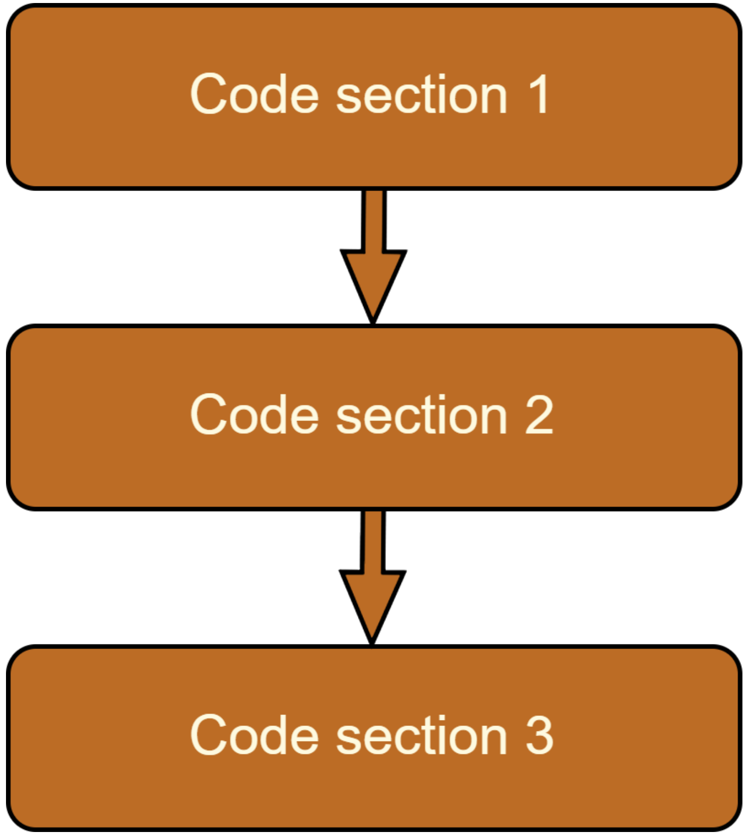
```python
import os, time
x = "Welcome to The Mirror."
y = 0

while y <= len(x):
    os.system("clear")
    print(x[:y])
    time.sleep(0.2)
    y = y+1
time.sleep(2)
x = "You will stay here for as long as you can, you may forfeit when you would like to."
y = 0

while y <= len(x):
    os.system("clear")
    print(x[:y])
    time.sleep(0.2)
    y = y+1
time.sleep(2)
x = "Please stand here, and stare into the mirror."
y = 0

while y <= len(x):
    os.system("clear")
    print(x[:y])
    time.sleep(0.2)
    y = y+1
time.sleep(2)
x = "This is you in the mirror:"
y = 0

while y <= len(x):
    os.system("clear")
    print(x[:y])
```

## Control flow

For a script to run from top to bottom and still be *good code*, we need to introduce two ways of controlling the flow of the code.

No recipes would say "*do step A, then do it again, and then do it a third time*". Instead it would say "*repeat step A 3 times*". Repeats in code are called **loops**.

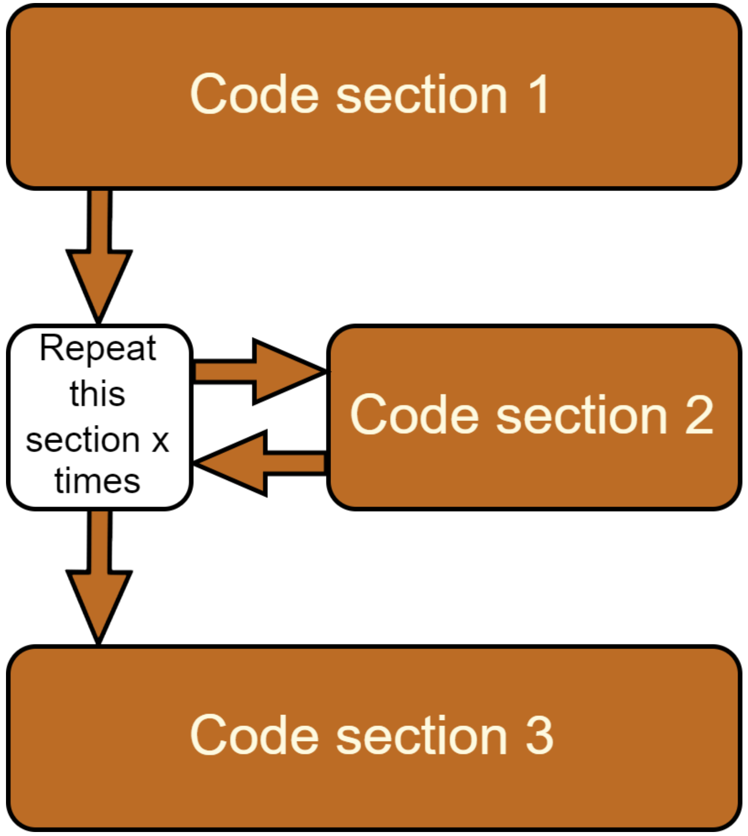
In recipes, you can read "*if X applies to you, then skip to step Y*". In code, such skips are created using **if-conditions**.



## Repeating parts of the code

When using **loops** we need a way to decide how many iterations in a loop.

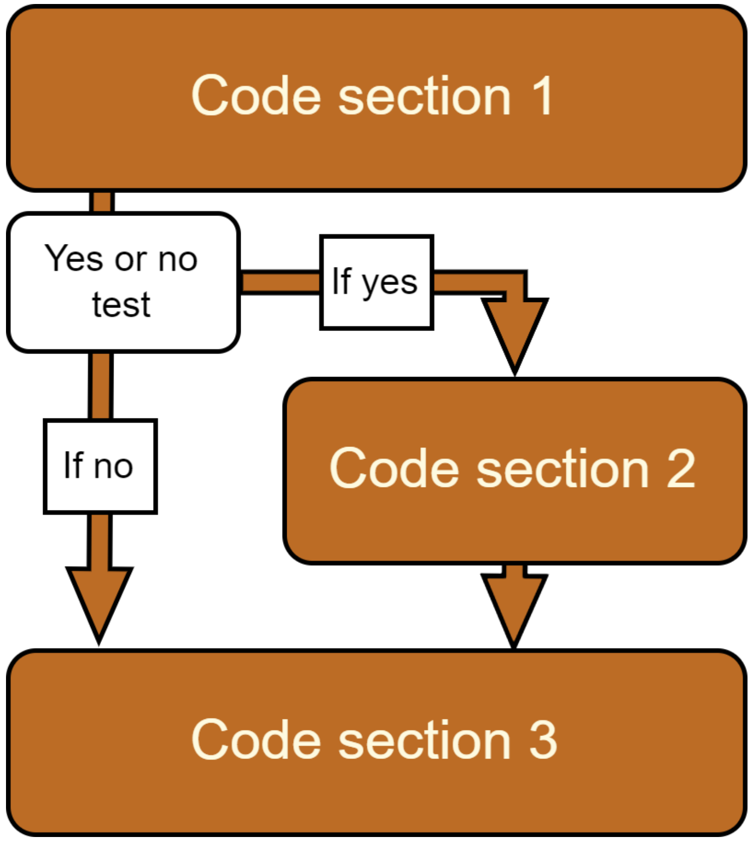
- **Hard coded**. Repeat an exact number of times. For example, 10 times
- **While-loop**. Repeat until a condition is met. For example, a match is found
- **Dynamically**. Use a characteristic in the data to decide the number of repeats. For example, repeat once for each country in the data set.



## Sometimes skipping parts of the code

When repeating or re-using code, some sections does not always apply. We use **if-conditions** to instruct the computer when to skip some code.

- Only execute certain blocks on certain countries in a loop
- Code intended to be re-used across projects can fit more cases when **if-conditions** are used to make the code adapt itself to what is needed in each context.

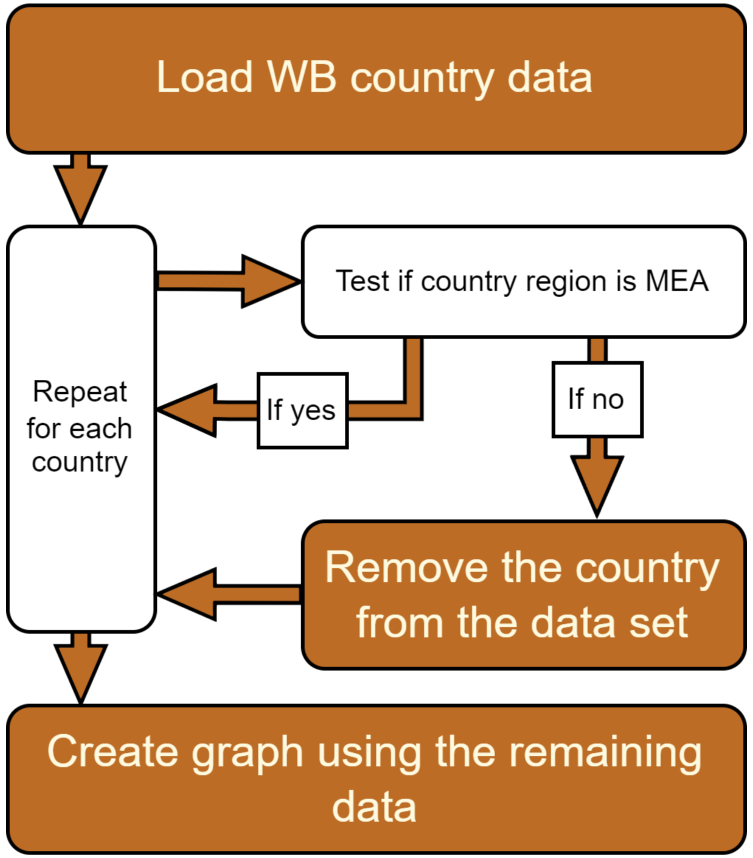


## Code flow - combine it all

Often loops and if-conditions are combined or they can be nested.

Session 2 of this course will show how this is done in Python.

Having neat, human readable and intuitive code flow using loops and if-conditions, is important to make code *good code*.


Load WB country data
Repeat for each country
Test if country region is MEA
If yes
If no
Remove the country from the data set
Create graph using the remaining data

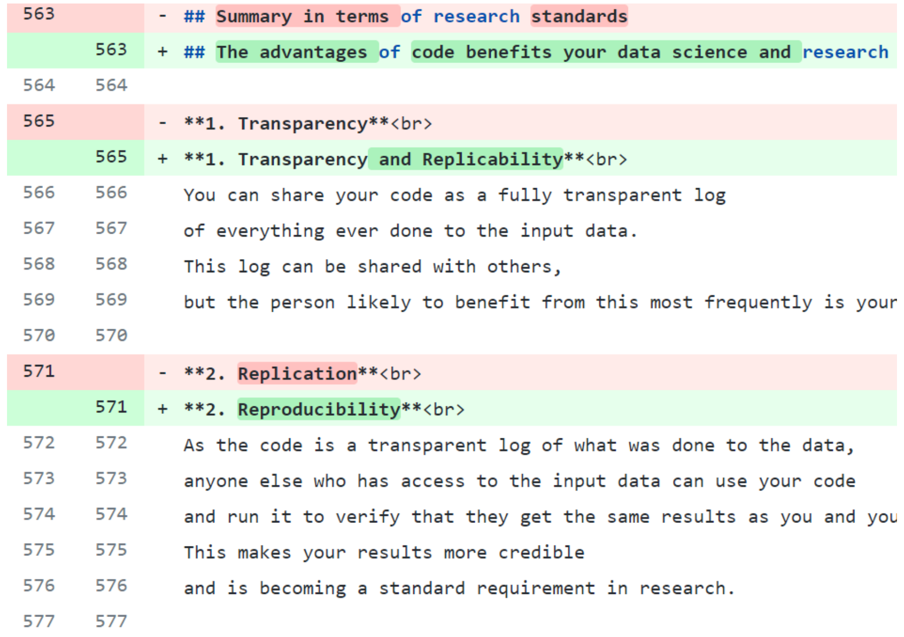
## Version control

*"Version control"* refers to tools used to:

- Keep track of multiple people making different contributions to the files of a project
- Give access to previous versions of the files of a project

Version control exists outside programming. For example, `ctrl+z`, restore files in OneDrive or Dropbox, recovered files in Windows, etc.

In coding version control is much more comprehensive, and can track every single edit anyone in the team do.



```
- ## Summary in terms of research standards
+ ## The advantages of code benefits your data science and research

- **1. Transparency**<br>
+ **1. Transparency and Replicability**<br>
  You can share your code as a fully transparent log
  of everything ever done to the input data.
  This log can be shared with others,
  but the person likely to benefit from this most frequently is your

- **2. Replication**<br>
+ **2. Reproducibility**<br>
  As the code is a transparent log of what was done to the data,
  anyone else who has access to the input data can use your code
  and run it to verify that they get the same results as you and you
  This makes your results more credible
  and is becoming a standard requirement in research.

```

## Version control of code = Version control of all data work

Code is very exact instructions on how to create outputs from original data.

If the original data is handled with exactness (backed-up or properly cited), and if we version control our code, then all outputs are also version controlled.

Version controlling all work is perhaps the most valuable feature of programming.

The most common standard for version control in code is called *Git* and most Git projects are hosted on *GitHub*.

## The advantages of code benefits your data science and research work

**1. Transparency and Replicability**
You can share your code as a fully transparent log of everything ever done to the input data. This log can be shared with others, but the person likely to benefit from this most frequently is yourself.

**2. Reproducibility**
As the code is a transparent log of what was done to the data, anyone else who has access to the input data can use your code and run it to verify that they get the same results as you and your team. This makes your results more credible and is becoming a standard requirement in research.

**3. Re-usability and Automation**
We can copy/paste a section of code from one project to another. Meaning that we can repeat all work on one project to another project with just some small adjustments. Over time we build a toolbox of code sections for tasks we frequently need, and we will be much more productive.

See DIME Analytic's book for more details.

Thank you!

Questions?