Lecture 01

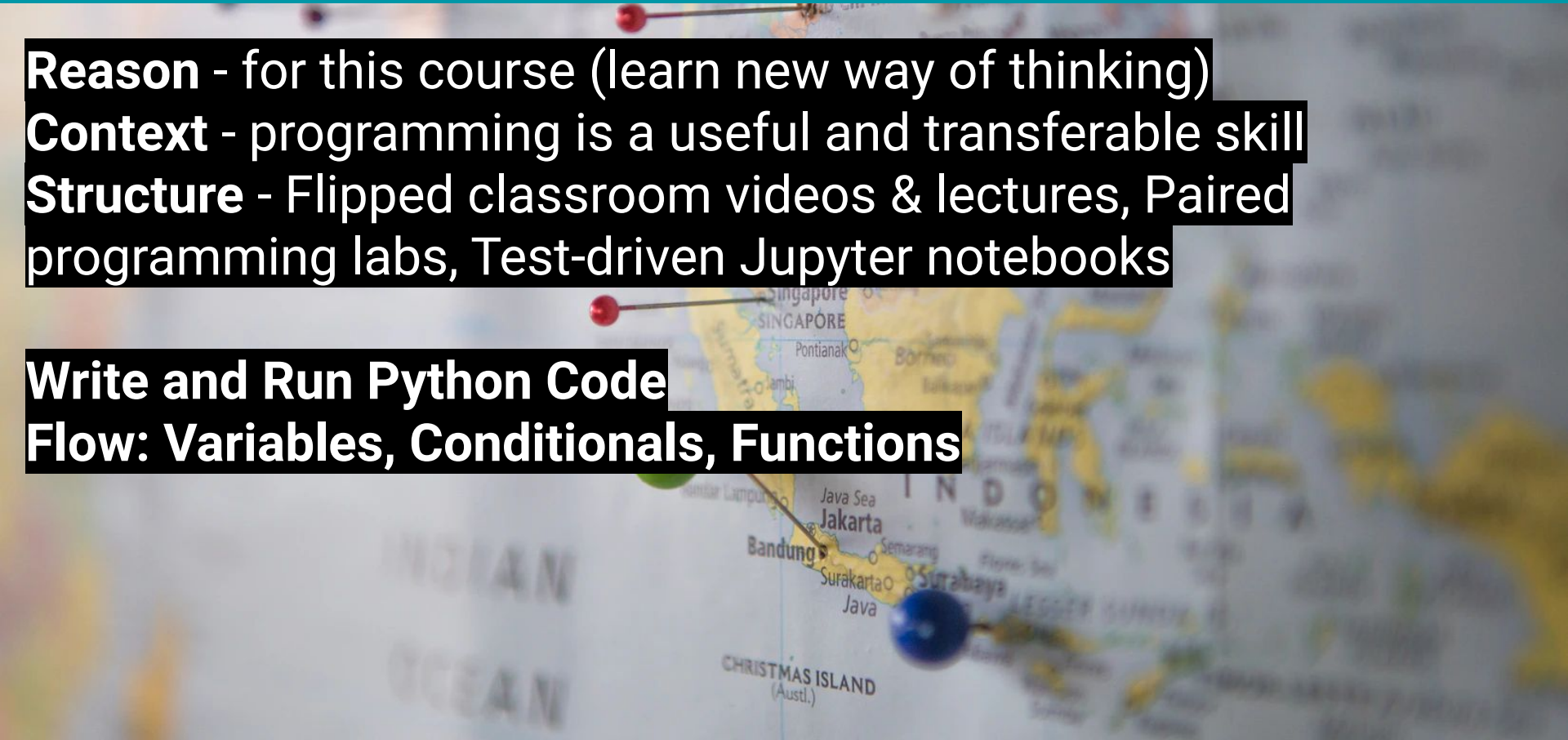
Intro + Flow (Variables, Conditionals, Functions)

Python Programming

Dr Pawel Orzechowski

UNIVERSITY OF EDINBURGH
Business School

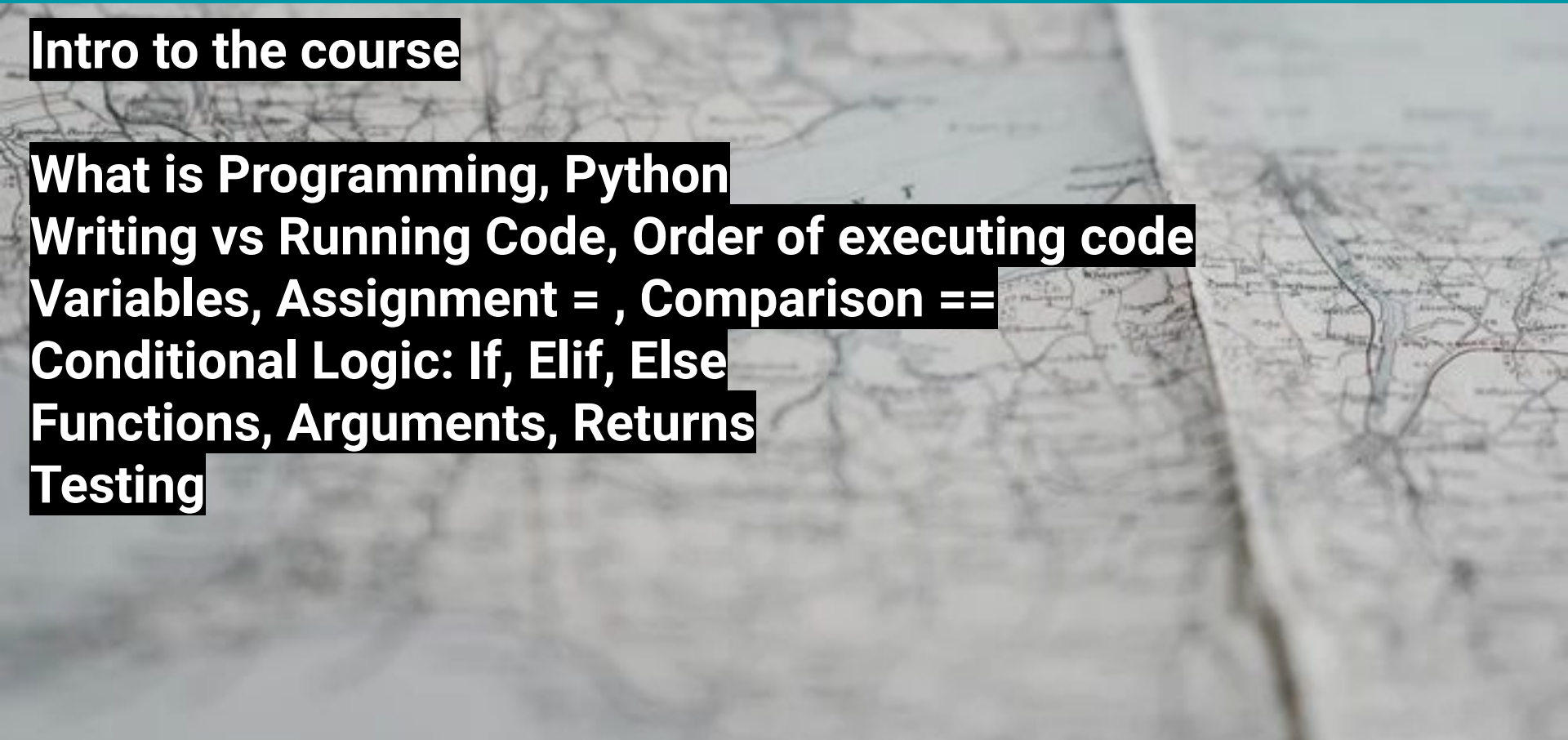
Learning goals and objectives



Reason - for this course (learn new way of thinking)
Context - programming is a useful and transferable skill
Structure - Flipped classroom videos & lectures, Paired programming labs, Test-driven Jupyter notebooks

Write and Run Python Code
Flow: Variables, Conditionals, Functions

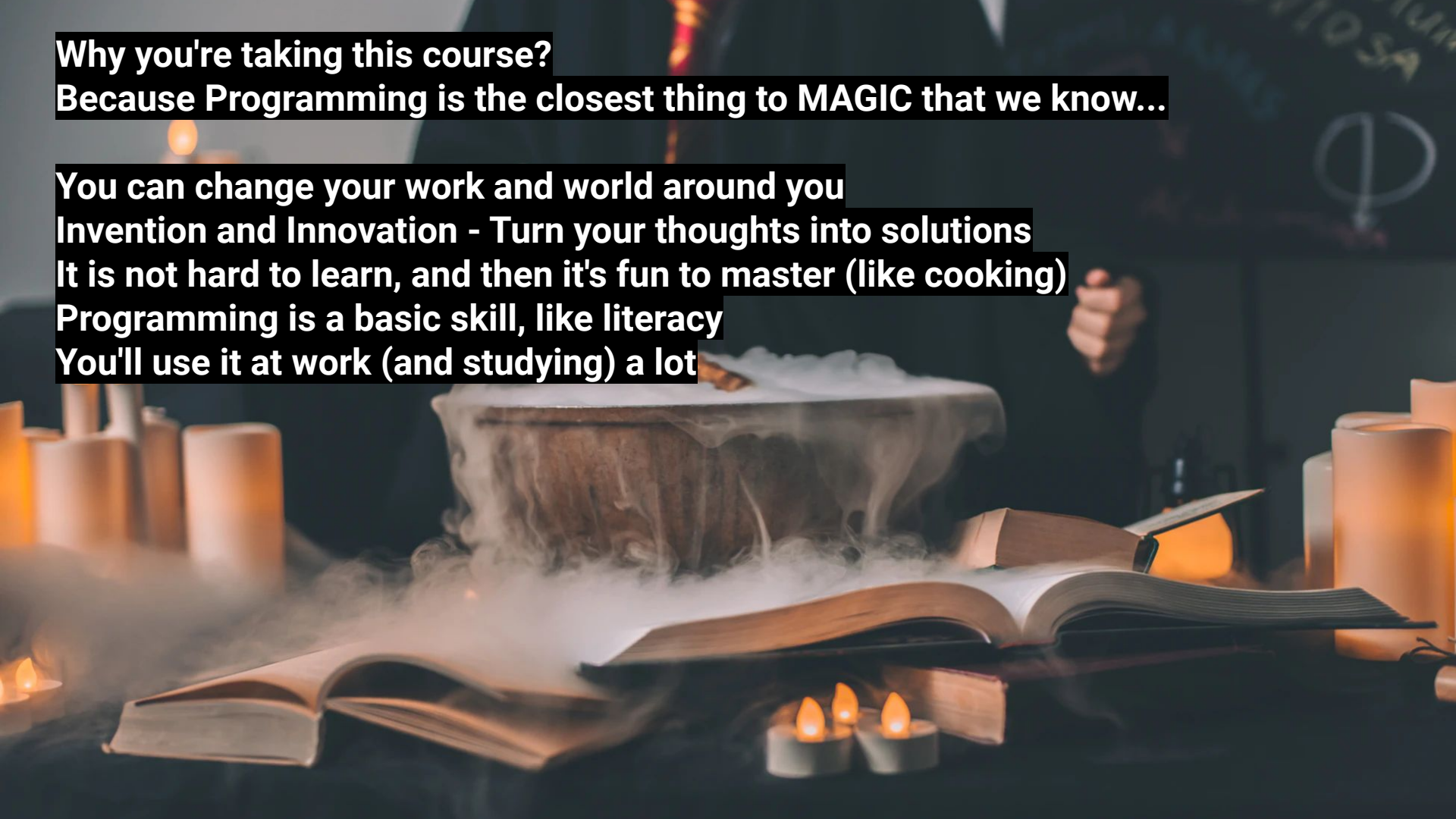
Outline of this lecture



Intro to the course

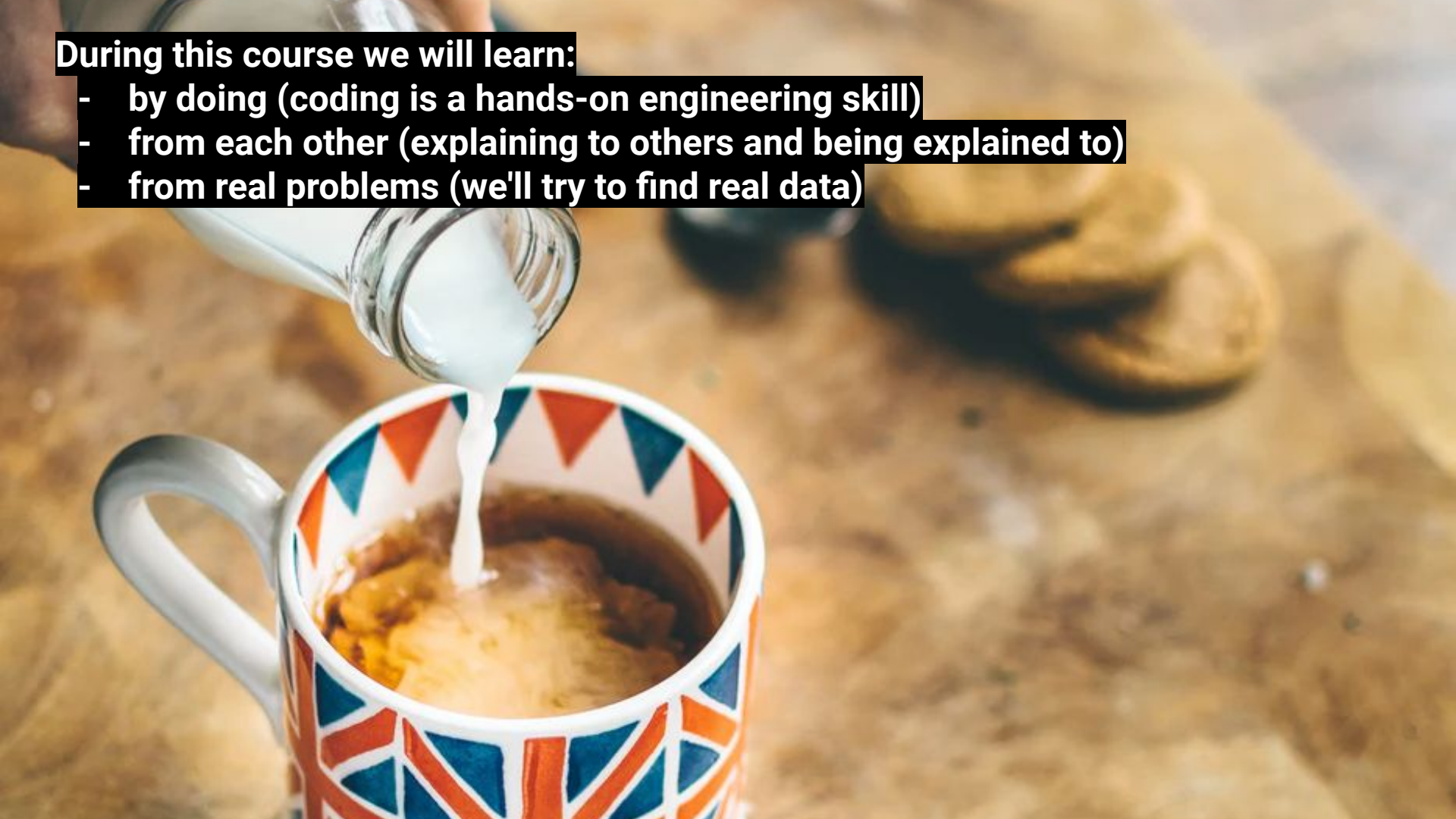
What is Programming, Python
Writing vs Running Code, Order of executing code
Variables, Assignment = , Comparison ==
Conditional Logic: If, Elif, Else
Functions, Arguments, Returns
Testing

Why and how are we going to do this?



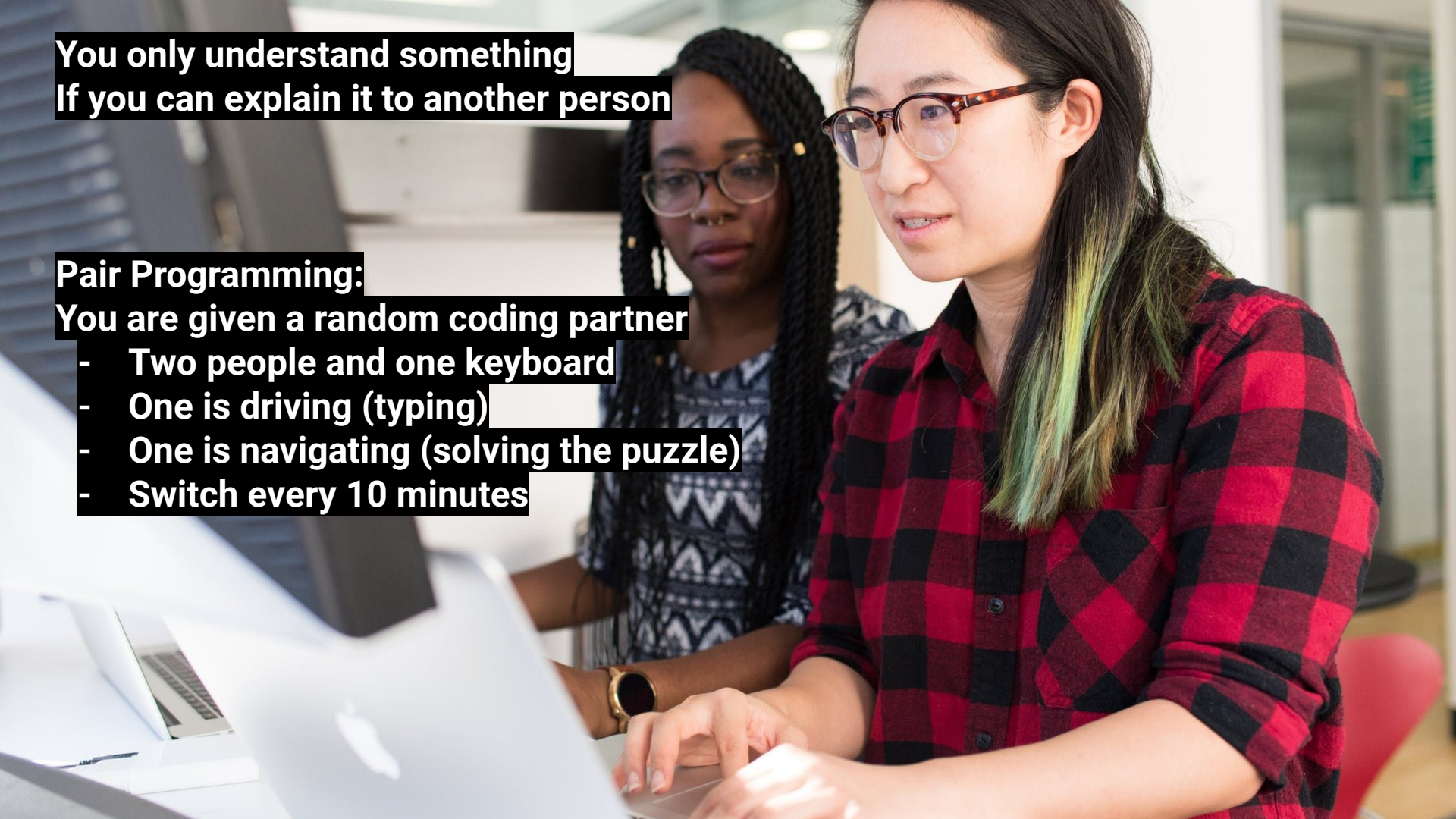
Why you're taking this course?
Because Programming is the closest thing to MAGIC that we know...

You can change your work and world around you
Invention and Innovation - Turn your thoughts into solutions
It is not hard to learn, and then it's fun to master (like cooking)
Programming is a basic skill, like literacy
You'll use it at work (and studying) a lot



During this course we will learn:

- by doing (coding is a hands-on engineering skill)
- from each other (explaining to others and being explained to)
- from real problems (we'll try to find real data)



You only understand something
If you can explain it to another person

Pair Programming:
You are given a random coding partner

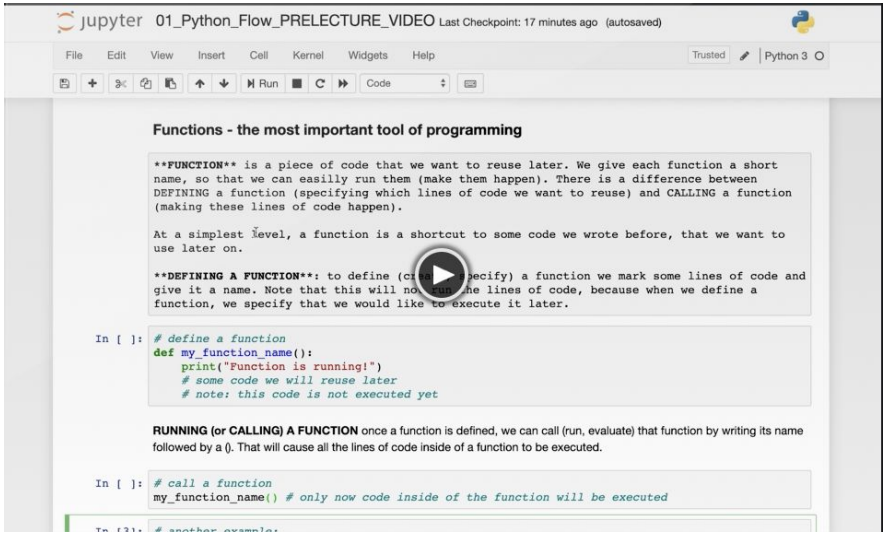
- Two people and one keyboard
- One is driving (typing)
- One is navigating (solving the puzzle)
- Switch every 10 minutes

Interactive code notebooks (Jupyter notebooks, available via Learn and Noteable)

- Notes and explanations of concepts
- Editable and runable code examples
- Available via a web browser from anywhere

Pre-Lecture Videos of me talking you through the notebook

- Watch any time before the lecture
- Pause, Rewind, Change Speed
- Watch at least once
- Accompanied by a notebook



Top Tip:
Shift+Enter runs code in the selected cell

Structure of this course:

Two-week schedule:

Week 1 only: Content: Pre-lecture Video and Lecture

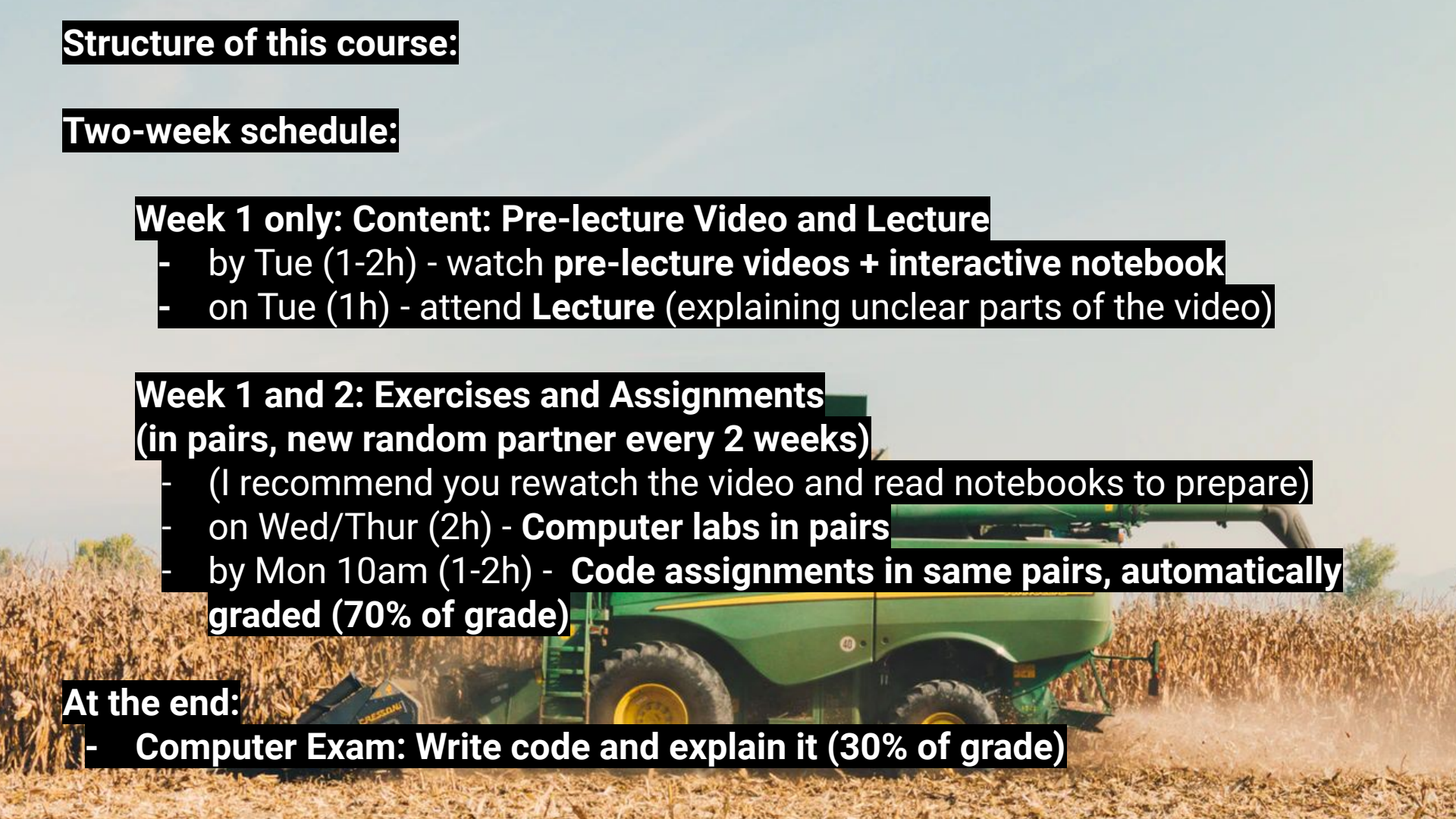
- by Tue (1-2h) - watch **pre-lecture videos + interactive notebook**
- on Tue (1h) - attend **Lecture** (explaining unclear parts of the video)

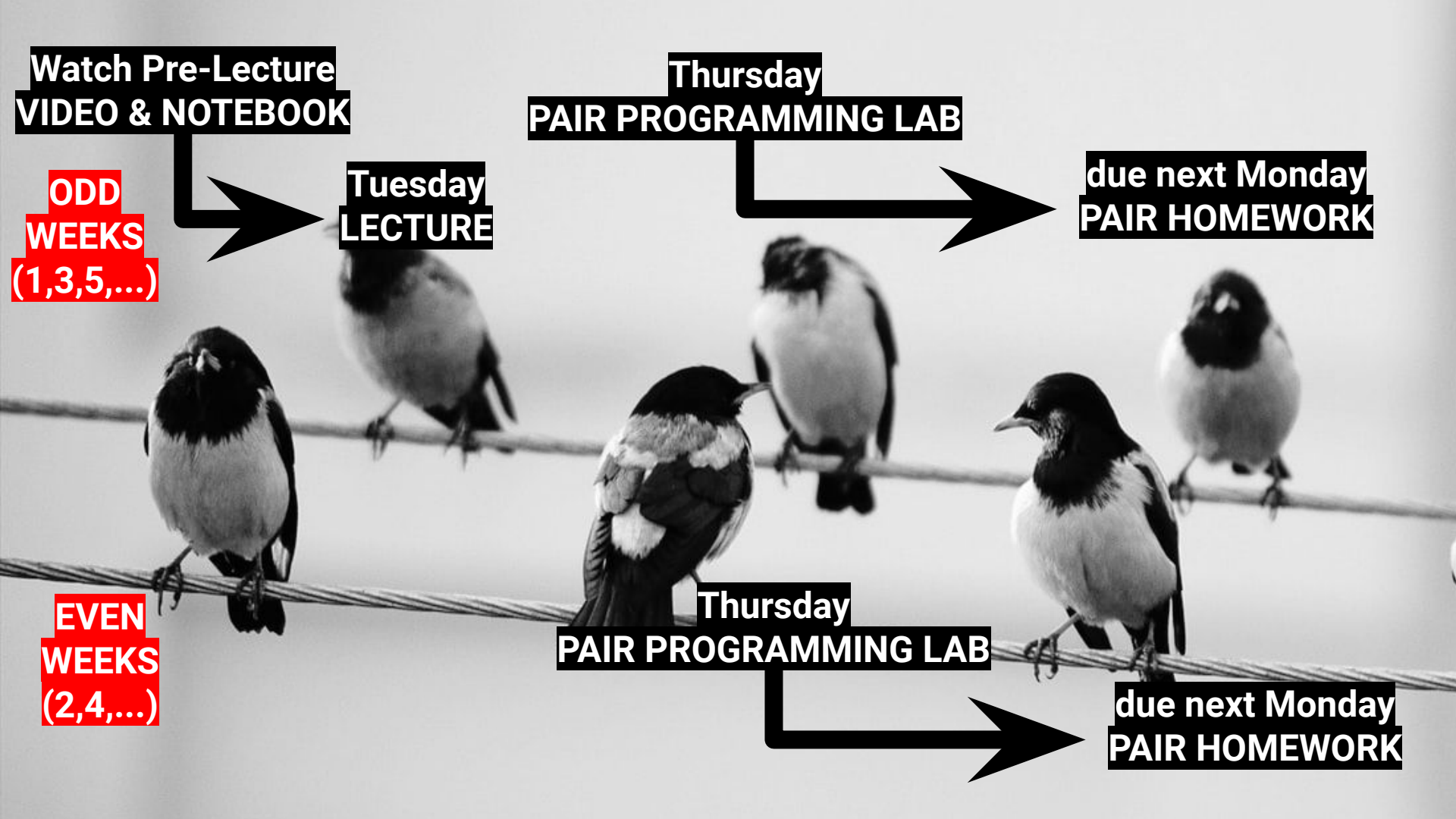
Week 1 and 2: Exercises and Assignments
(in pairs, new random partner every 2 weeks)

- (I recommend you rewatch the video and read notebooks to prepare)
- on Wed/Thur (2h) - **Computer labs in pairs**
- by Mon 10am (1-2h) - **Code assignments in same pairs, automatically graded (70% of grade)**

At the end:

- **Computer Exam: Write code and explain it (30% of grade)**



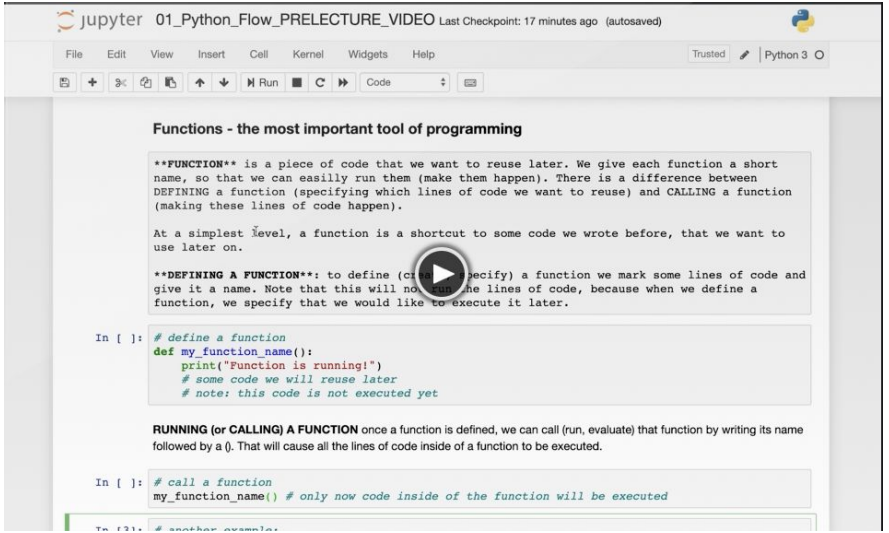

Watch Pre-Lecture
VIDEO & NOTEBOOK
ODD
WEEKS
(1,3,5,...)
Tuesday
LECTURE
Thursday
PAIR PROGRAMMING LAB
due next Monday
PAIR HOMEWORK
EVEN
WEEKS
(2,4,...)
Thursday
PAIR PROGRAMMING LAB
due next Monday
PAIR HOMEWORK

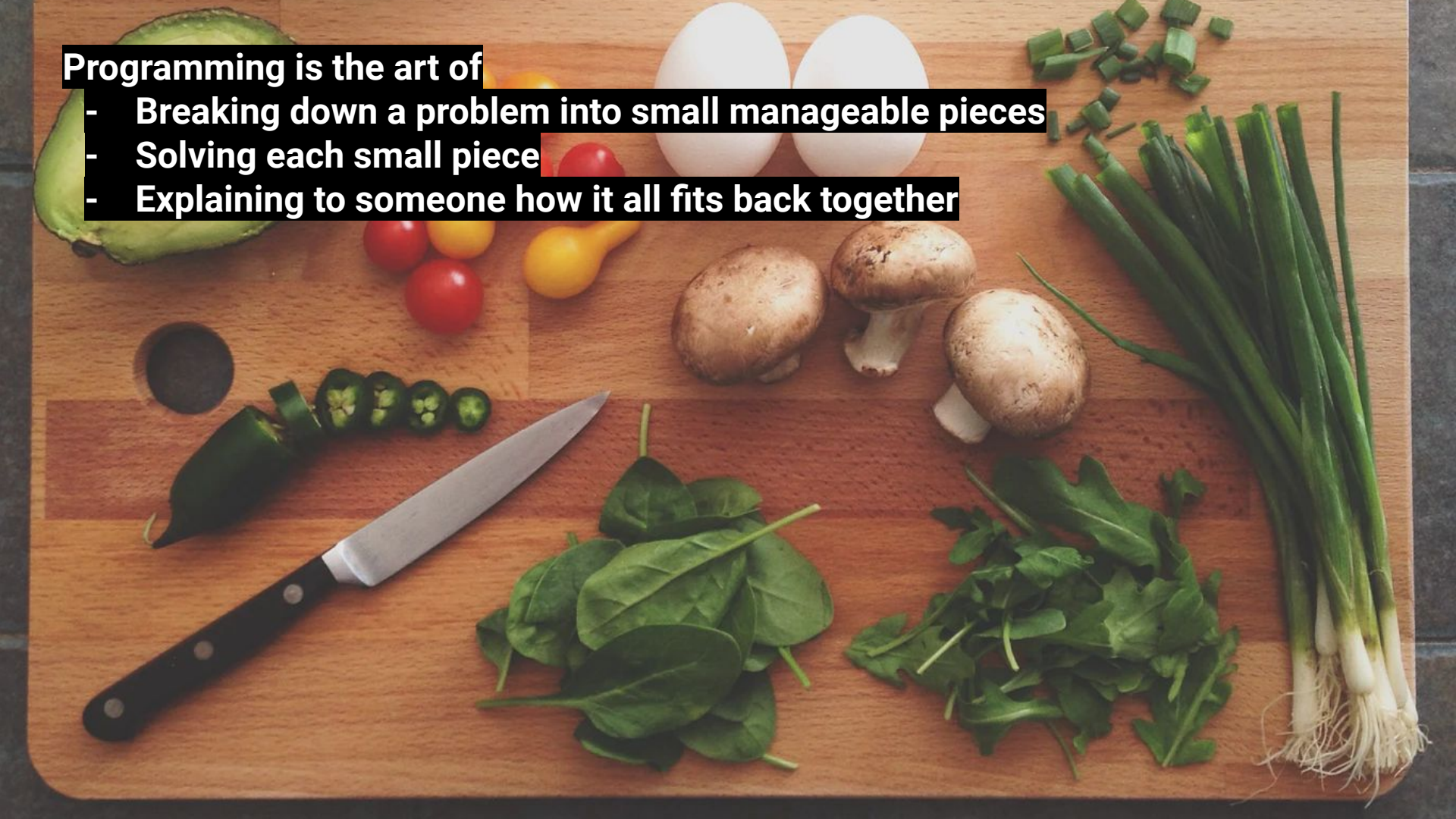
"Previously on ..." - pre-lecture video

Normally you will get the pre-lecture video a few days before the lecture.

In video 1:

- Using Jupyter Notebooks
- Variables
- Conditionals
- Functions
- Testing




Programming is the art of
- Breaking down a problem into small manageable pieces
- Solving each small piece
- Explaining to someone how it all fits back together

Why you should understand flow of code well

We leverage skills that computers are good at:

- **Following simple commands**
- **Remembering things**
- **Logical decisions**
- **Repeating actions**
- Combining many moving parts
- Communicating in agreed ways

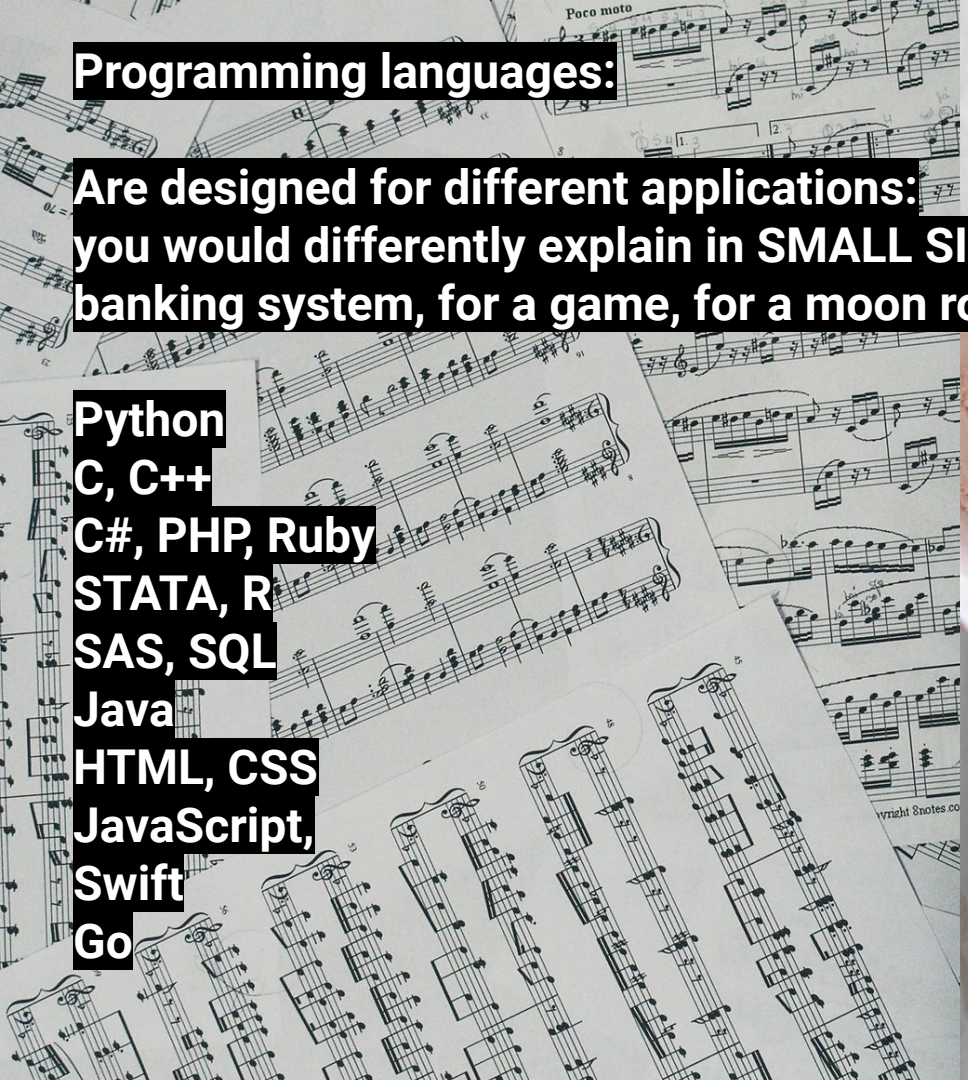
Why you should understand flow of code well

We leverage skills that computers are good at:

- **Following simple commands**
- **Remembering things**
- **Logical decisions**
- **Repeating actions**
- **Combining many moving parts**
- **Communicating in agreed ways**

It's an implicit arrangement we have with computers:

Computers gives us power
We promise to exactly specify what we want

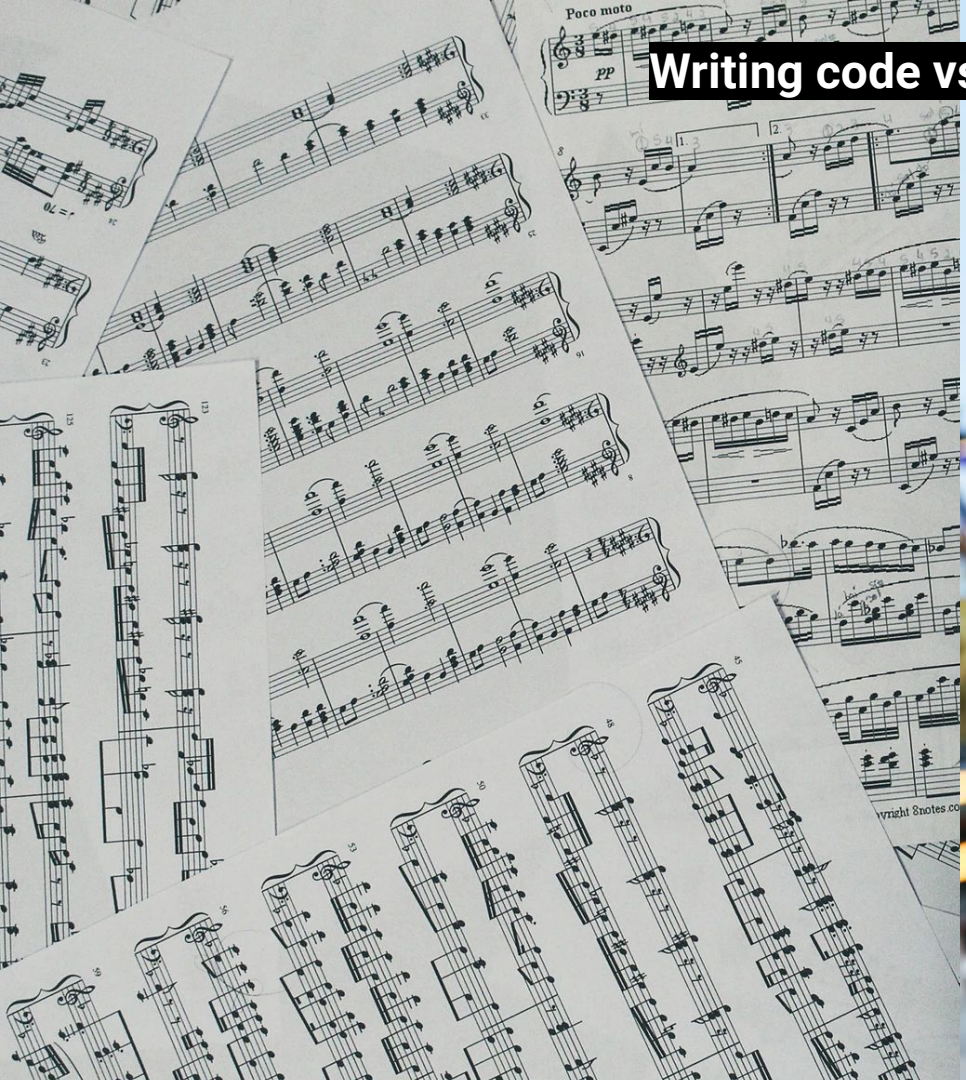


Programming languages:

Are designed for different applications:
you would differently explain in SMALL SIMPLE STEPS how to do something for a banking system, for a game, for a moon rover

Python
C, C++
C#, PHP, Ruby
STATA, R
SAS, SQL
Java
HTML, CSS
JavaScript,
Swift
Go

Writing code vs running code

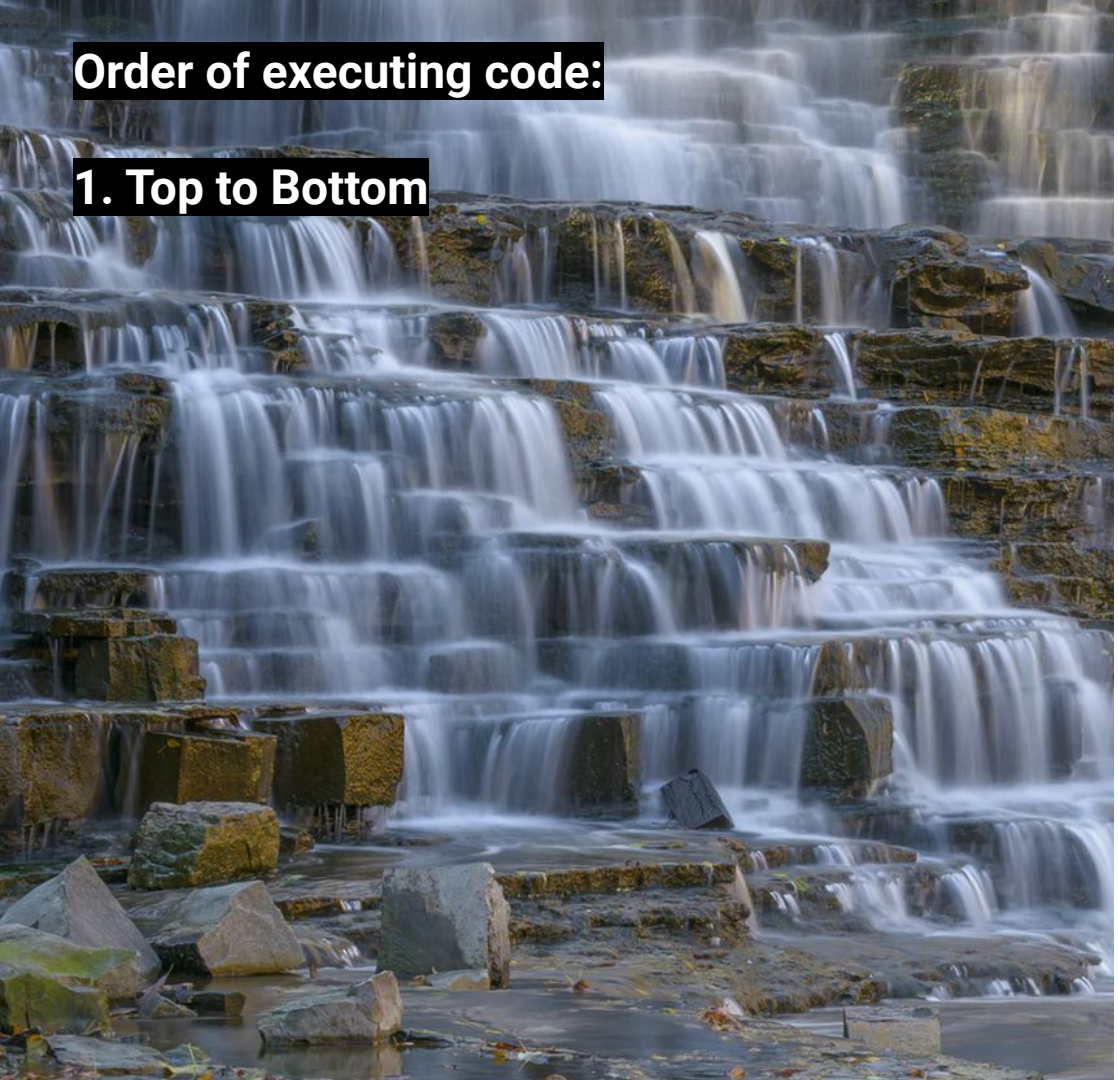


Writing code vs running code

Ada Lovelace (1815 – 1852)
The First Programmer

She created theoretical foundations for most concepts we will learn about during this course

***I also recommend the fantastic comic book about her life**



Order of executing code:

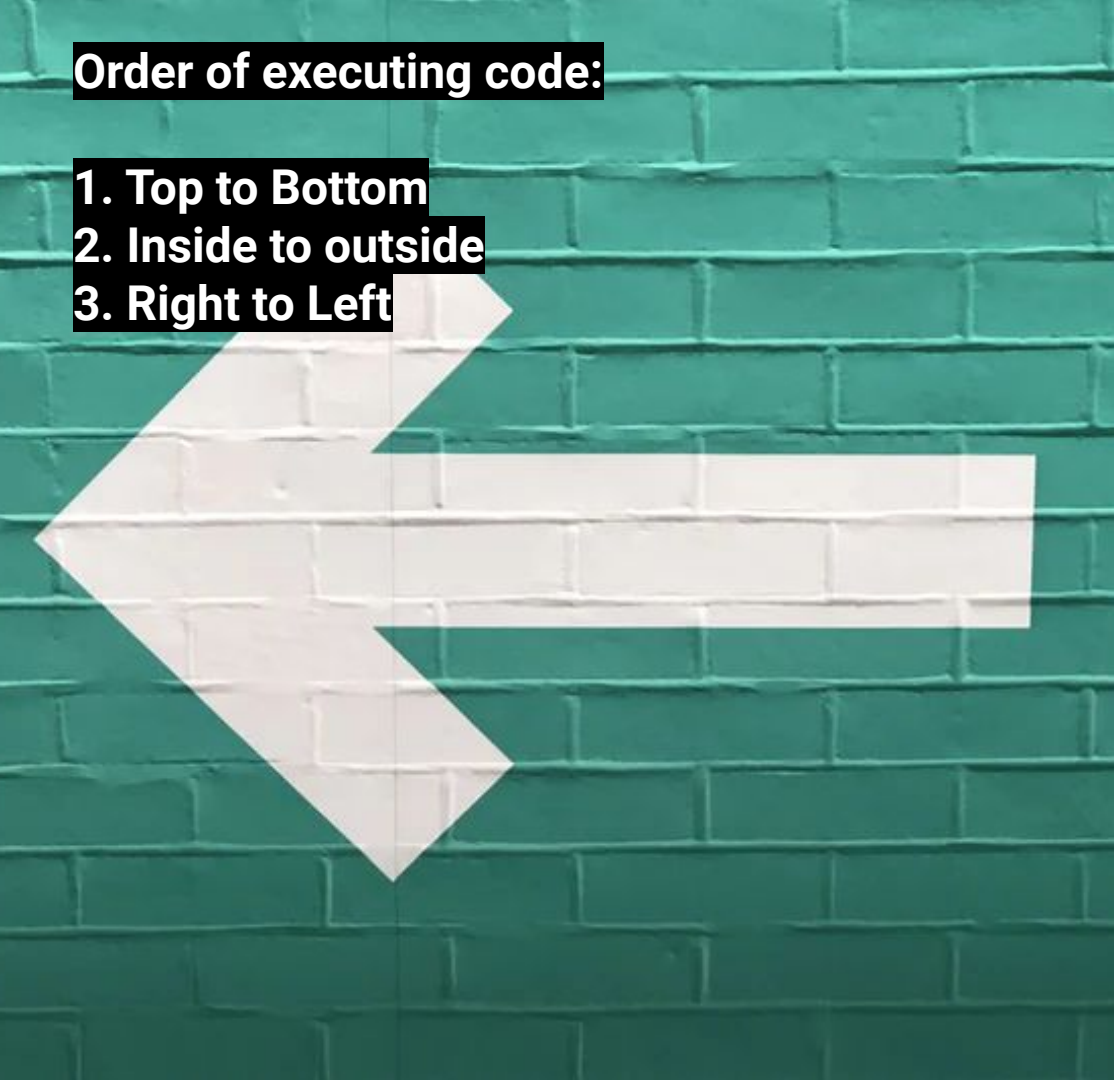
1. Top to Bottom

```
print("1")
print("2")
print("3")
print("4")
print("5")
print("6")
```

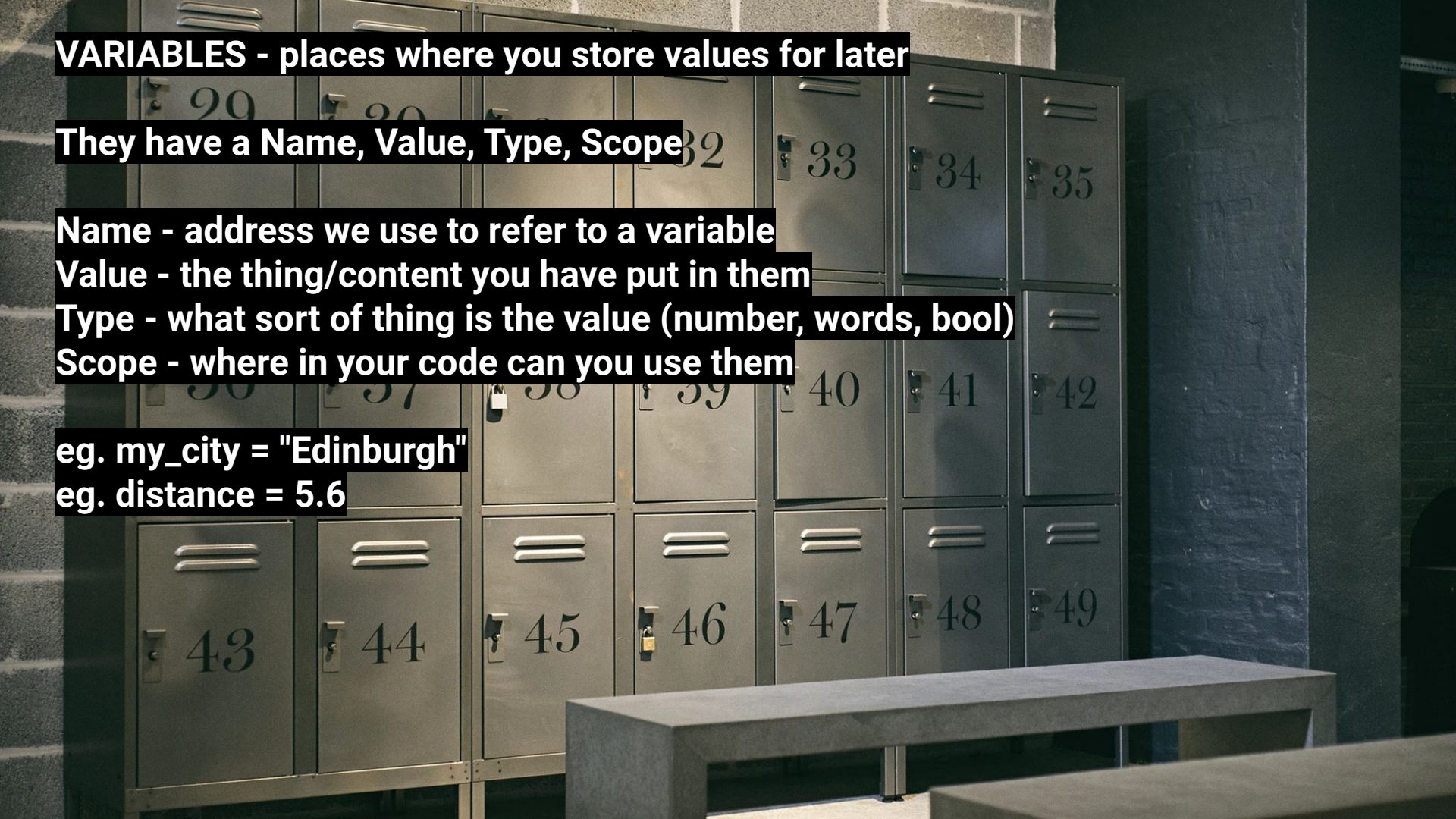
Order of executing code:

1. Top to Bottom
2. Inside to outside

```
print( 2 + 2 )
doThis( butThisFirst( 2 + 2 ) )
doThisLast( doThis() + doThat() )
```




```
sum  =  2 + 2
your_name  = input("Who R U?")
is_this_fiona = name == "Fiona"
```

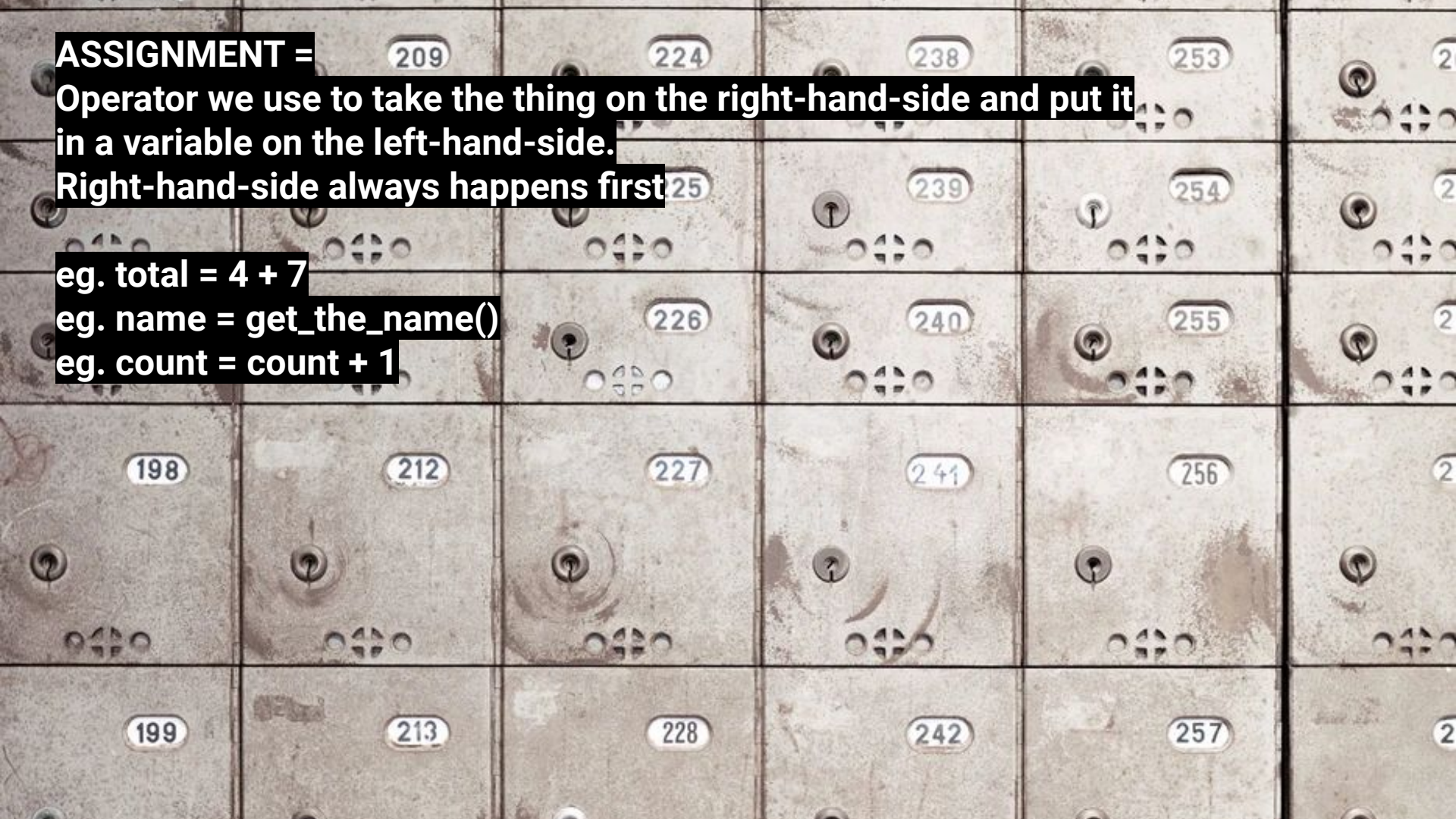


VARIABLES - places where you store values for later

They have a Name, Value, Type, Scope

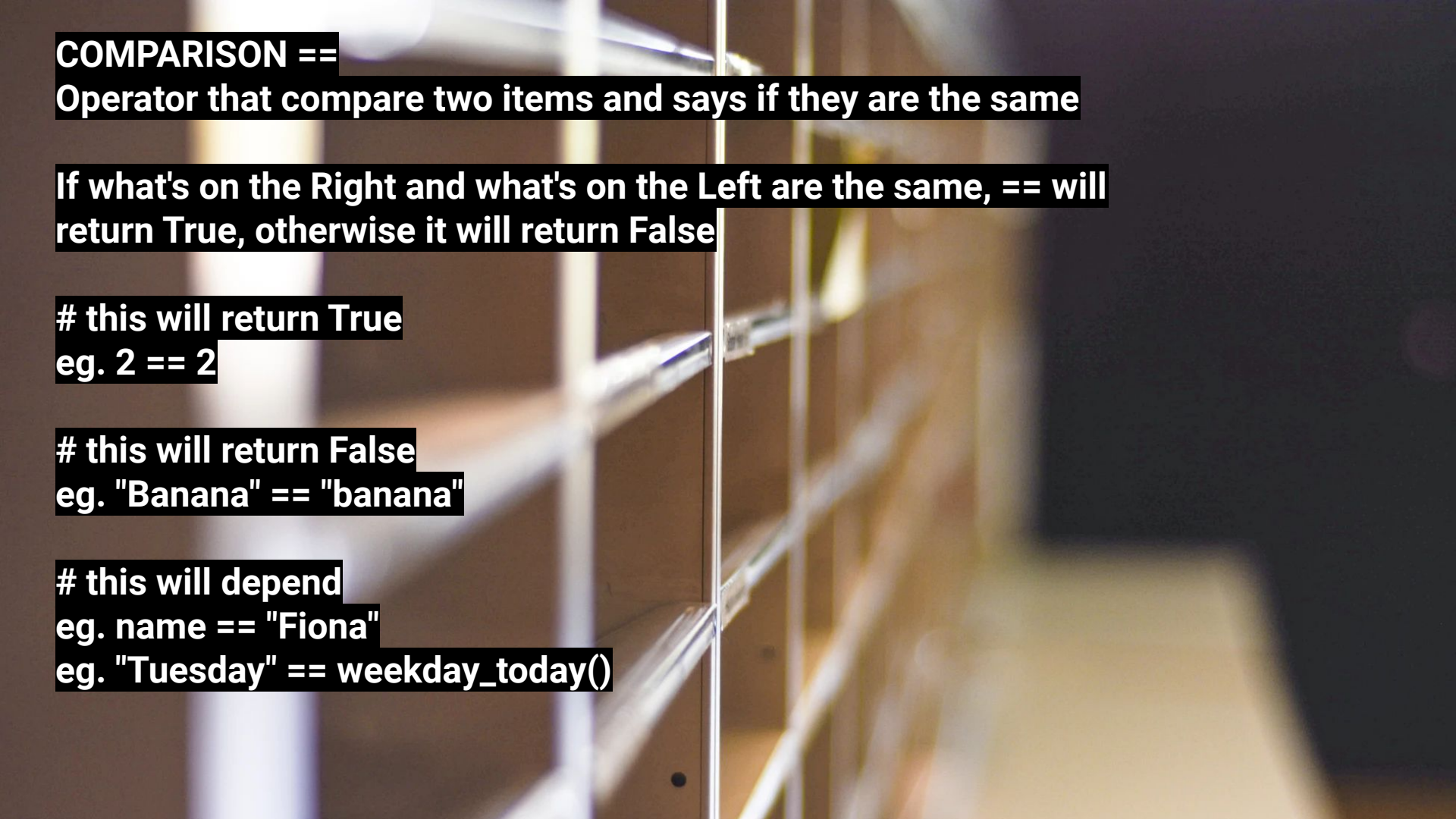
Name - address we use to refer to a variable
Value - the thing/content you have put in them
Type - what sort of thing is the value (number, words, bool)
Scope - where in your code can you use them

eg. my_city = "Edinburgh"
eg. distance = 5.6



ASSIGNMENT =
Operator we use to take the thing on the right-hand-side and put it in a variable on the left-hand-side.
Right-hand-side always happens first

eg. total = 4 + 7
eg. name = get_the_name()
eg. count = count + 1



COMPARISON ==
Operator that compare two items and says if they are the same

If what's on the Right and what's on the Left are the same, == will return True, otherwise it will return False

this will return True
eg. 2 == 2

this will return False
eg. "Banana" == "banana"

this will depend
eg. name == "Fiona"
eg. "Tuesday" == weekday_today()

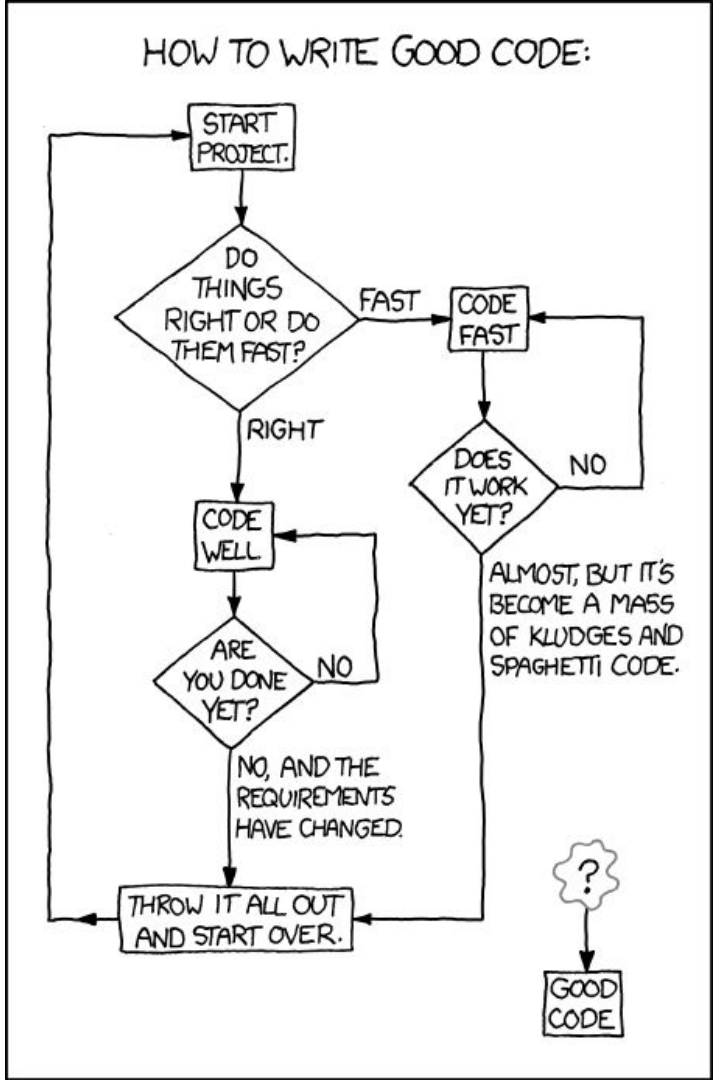
CONDITIONAL LOGIC
deciding which lines of code to run based on something being True or False

```
print("1. Start here")

if something:
    print("1a. Do this!")
else:
    print("1b. or do that!")

print("2. and then come back")
```

```
today  = "Monday"
Time = 9

if today == "Friday" and time > 17:
        print("Yay! Weekend is coming!")
elif today == "Saturday":
        print("Weekend is here!")
elif today == "Sunday":
        print("Weekend is almost over!")
else:
        print("Ask me again tomorrow")
```

Conditional logic - Flow chart

Rectangles: Actions
Diamonds: Choices



Ethical dimensions
of making decisions
by following simple rules

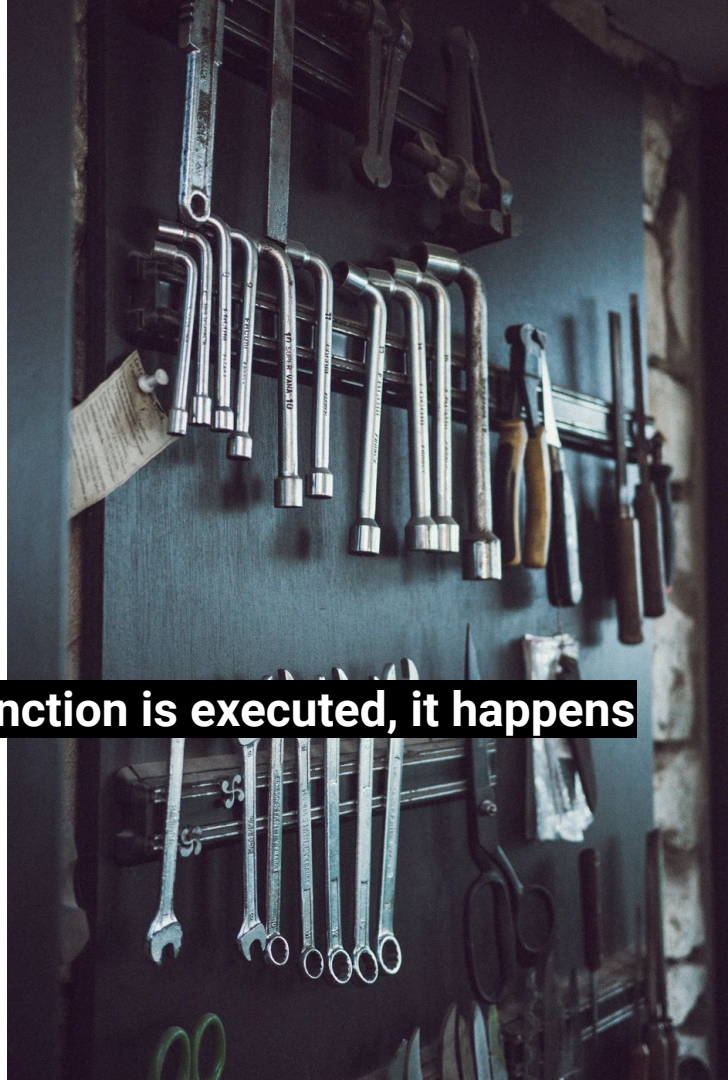
i.e. Simple rule-based
dialog systems
can be annoying
but dependable

What are they good for?
What are they terrible for?

Functions:

- Skills you can teach your computer to perform
- Some of code you can re-use later

```
# DEFINE THE FUNCTION - nothing will happen yet

def say_hello_3_times():
    print("Hello")
    print("Hello")
    print("Hello")

# RUN THE FUNCTION - only now code inside the function is executed, it happens

say_hello_3_times()
say_hello_3_times()
```



Functions can take parameters
eg. fetch("stick")
eg. fetch("ball")

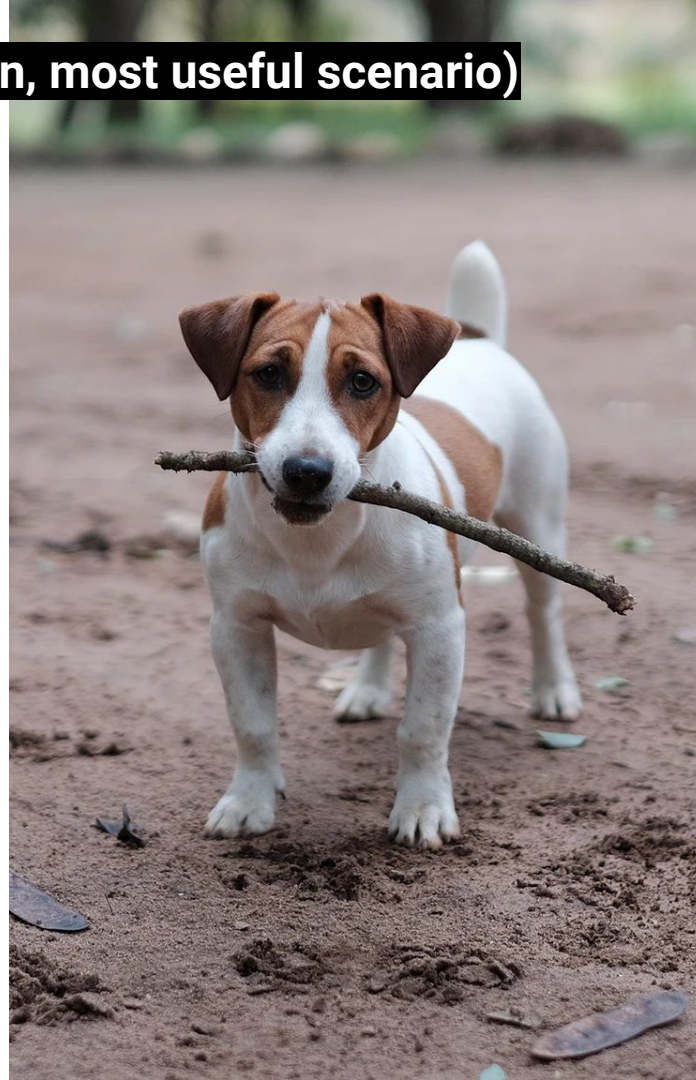
```
def fetch(thing_to_fetch):
    print("I am fetching", thing_to_fetch)
```

eg. make_coffee("large", "no milk", "1 sugar")

```
def make_coffee(size, milk, sugars):
    print("Making one", size, "coffee with", milk, sugar)
```

Functions can return something (it's the most common, most useful scenario)

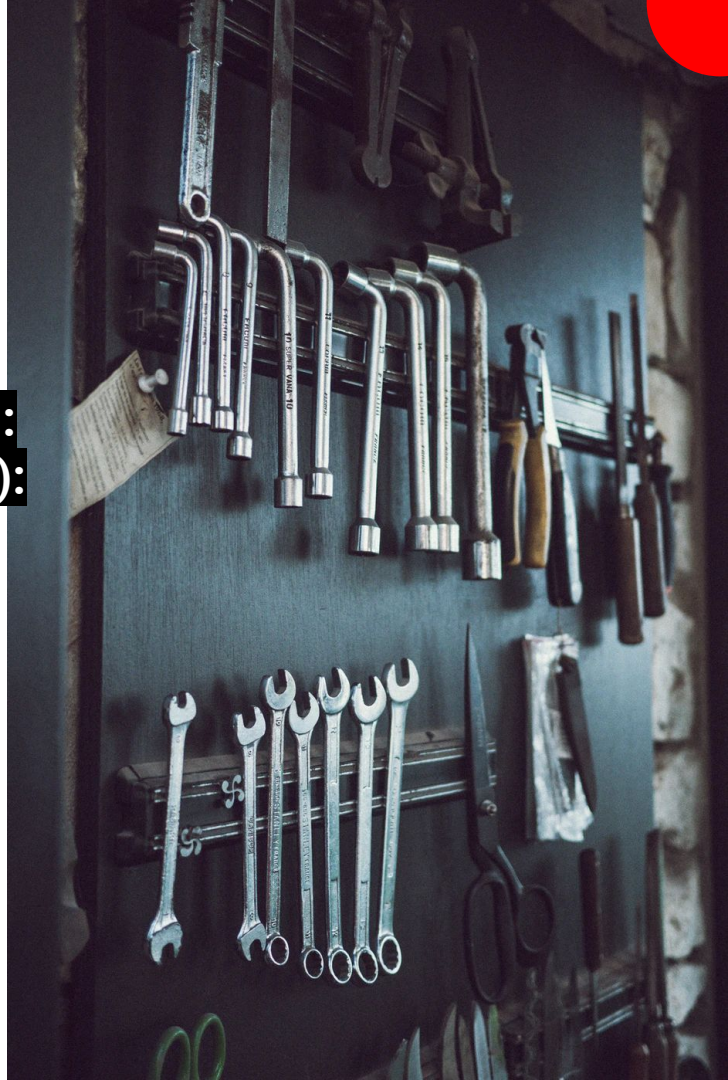
```
def add(num_1, num_2):
    return num_1 + num_2

total = add(4 , 5)

def larger_number(num_1, num_2):
    if num_1 > num2:
        return num_1
    else:
        return num_2

needed_size = larger_number(10, 30)
```



A few strategies for returning things:

```
# you can specify each return on the branch

def is_first_number_larger(num_1, num_2):
    if num_1 > num2:
        return True
    else:
        return False

# you can sometimes return result of a calculation

def is_first_number_larger(num_1, num_2):
    return num_1 > num2
```

```
def add(num_1, num_2):
    return num_1 + num_2

def multiply(num_1, num_2):
    return num_1 * num_2

def is_added_larger_than_multiplied(num_1, num_2):
    if add(num_1, num_2) > multiply(num_1, num_2):
        return True
    else:
        return False

is_added_larger_than_multiplied(1,2)
is_added_larger_than_multiplied(3,4)
is_added_larger_than_multiplied(-2,-5)
is_added_larger_than_multiplied(4,0)
```



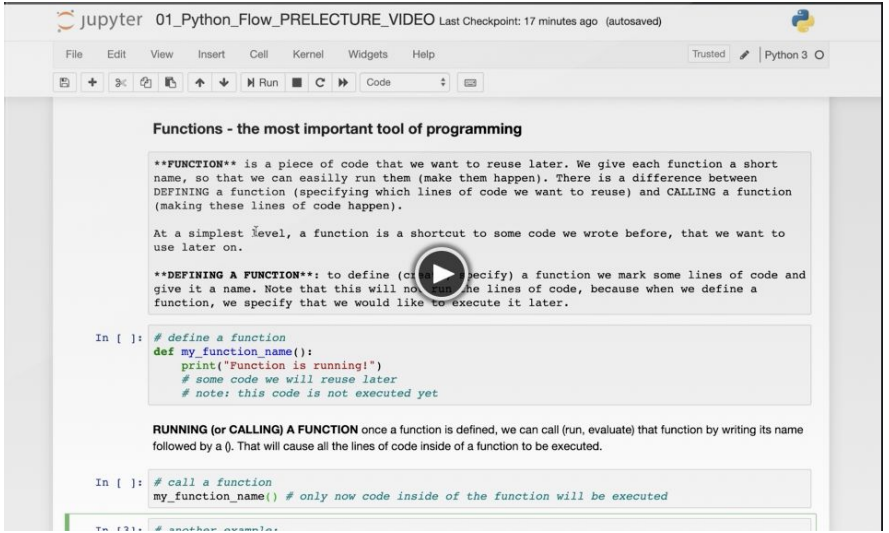
```
def add(num_1, num_2):
    return num_1 + num_2

def multiply(num_1, num_2):
    return num_1 * num_2

def is_added_larger_than_multiplied(num_1, num_2):
    if add(num_1, num_2) > multiply(num_1, num_2):
        return True
    else:
        return False

assert is_added_larger_than_multiplied(1,2) == False
assert is_added_larger_than_multiplied(3,4) == True
assert is_added_larger_than_multiplied(-2,-5) == False
assert is_added_larger_than_multiplied(4,0) == True
```

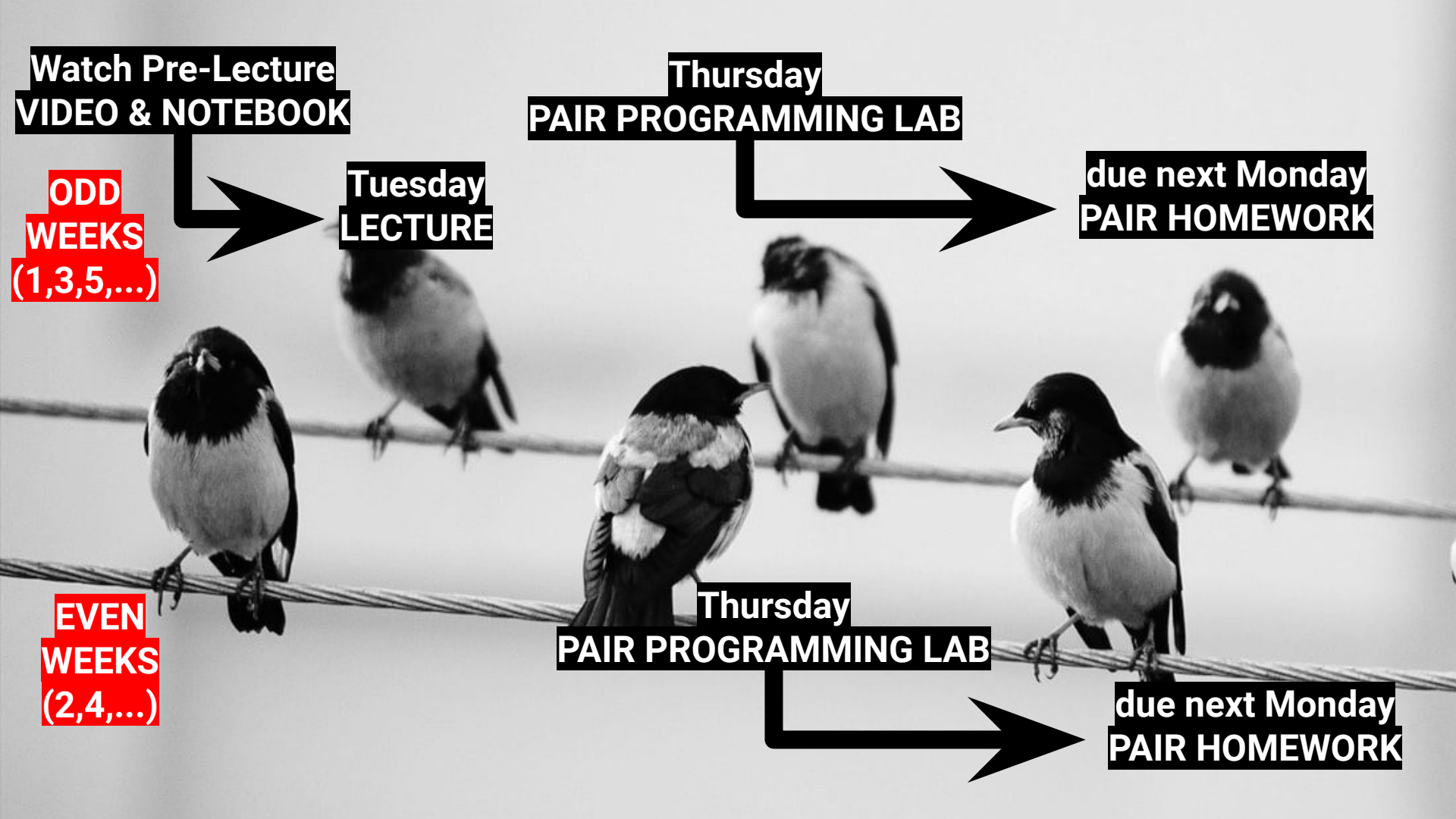
"Previously on ..." - pre-lecture video

Normally you will get the pre-lecture video a few days before the lecture.

In video 1:

- Using Jupyter Notebooks
- Variables
- Conditionals
- Functions
- Testing




Watch Pre-Lecture
VIDEO & NOTEBOOK
ODD
WEEKS
(1,3,5,...)
Tuesday
LECTURE
Thursday
PAIR PROGRAMMING LAB
due next Monday
PAIR HOMEWORK
EVEN
WEEKS
(2,4,...)
Thursday
PAIR PROGRAMMING LAB
due next Monday
PAIR HOMEWORK