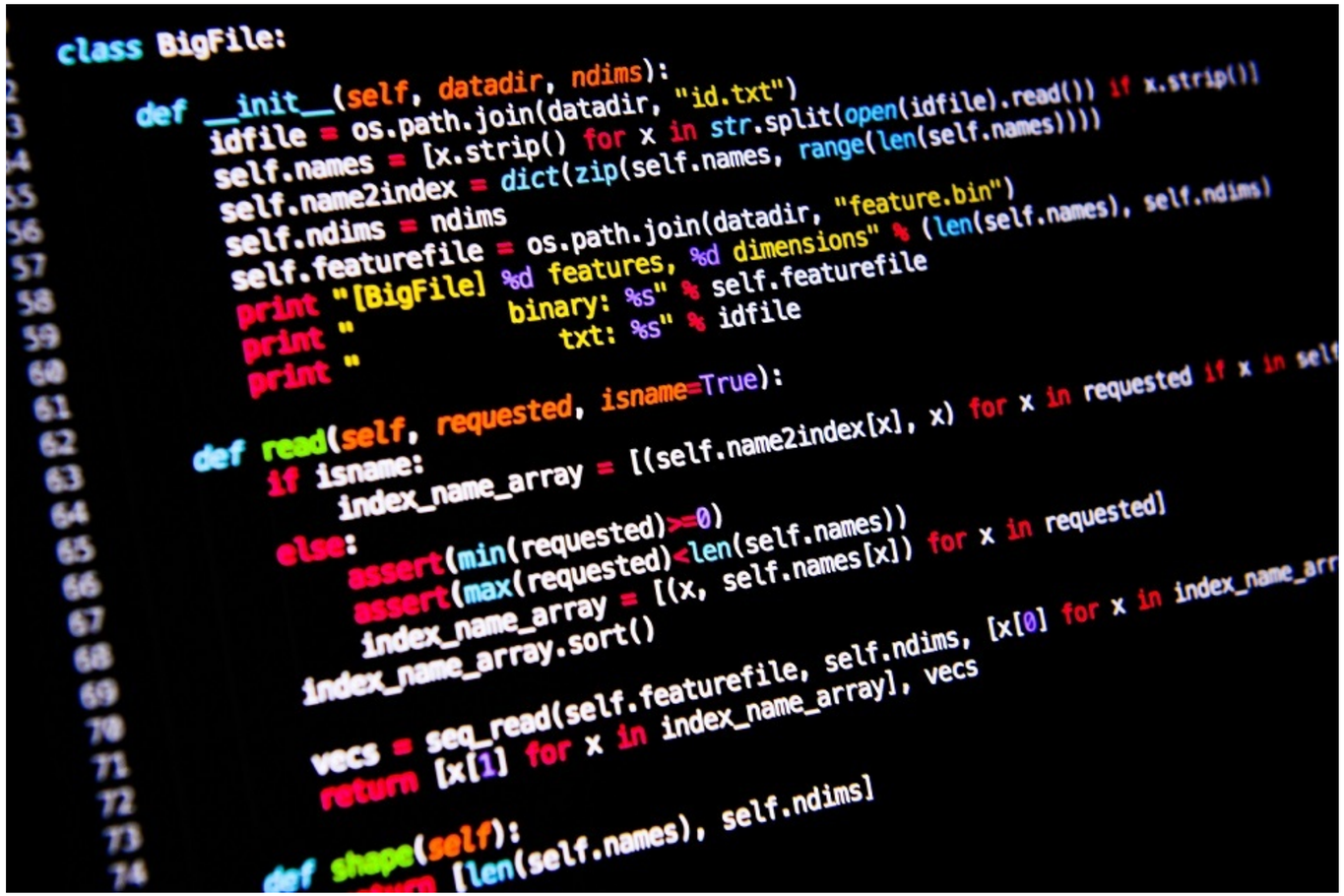
# 1. Introduction to Programming

DataKirk

# Overview

1. What is programming?
2. Programming with Python
3. Python language basics

# What is programming?



# What is programming?

- Programming is the art of giving instructions to a computer in order to solve a problem.
- Usually this is done by writing out your instructions as computer *code*. This code needs to be written in terms that a computer can understand.
- A specific set of instructions is called an *algorithm*. We use algorithms all the time, for example a cake recipe is a type of algorithm.

```
1. Preheat oven to 180C
2. Mix 4 eggs, 200g of sugar and 200g of flour in a bowl
3. Place mixture in a tin and put into oven
4. Bake for 25 minutes then remove
```

# What is programming?

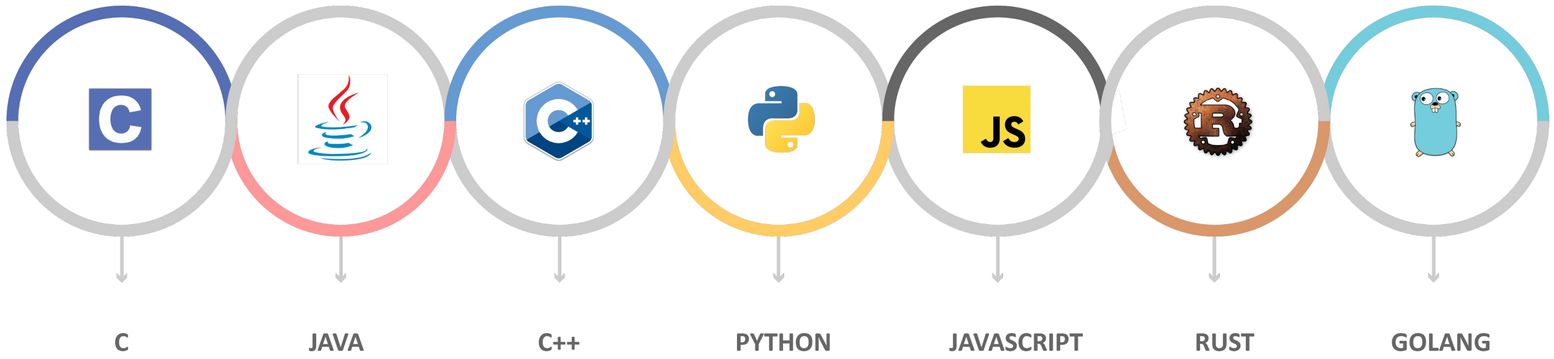
- However, when giving instructions to a computer you have to be *very* precise.
- If you ask a friend to make you a cup of coffee, chances are they will know exactly what you mean and have no problem completing the task.
- But for a computer, you would need to say something like

    1. Walk towards the coffee jug at 1m/s.
    2. When you reach the jug, extend your arm and grip the handle with 2N of force.
    3. Lift the jug ...

- A computer will do **exactly what you tell it to** - nothing more, nothing less.

# Programming languages

- When we give instructions to a computer as code, they need to be written in a specific programming *language*.
- Just like human languages, you can express more or less anything you want with programming languages, however there are rules about what is and isn't "correct".
- There are many different languages to choose from. Different languages are usually specialised for different tasks.



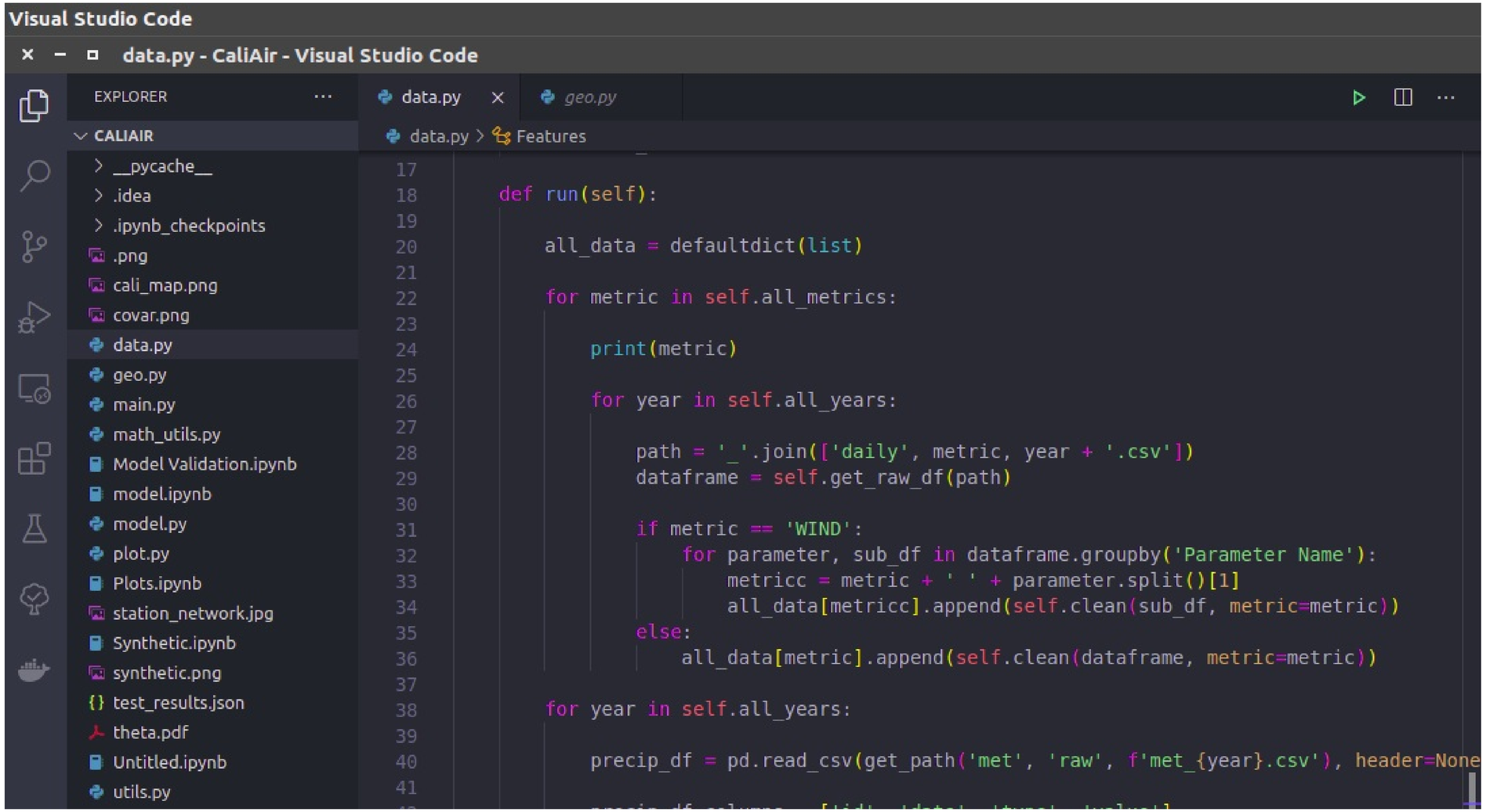
# Python


- In this course, you will be learning to program with the language *Python*. Python is a very popular language both with beginners and experts. Some of the benefits are:
    - Easier to pick up than other languages.
    - Very flexible - you can do anything with it.
    - One of the highest-paying languages in the job market.
    - "Batteries included" - lots of features to use out of the box.
- Python has many applications, from artificial intelligence, to hacking, to web development, to data analytics and more.

# How do I write code?

- Code is written in plain-text format, in a file.
- Code should **never** be written in something like Microsoft word. Word documents have special features like fonts, colours etc. None of this exists in a code file - it's just text.
- Code should be written and edited using a text editor. A very simple text editor that exists on all windows computers is called Notepad.
- However, notepad is a little too basic. Usually people choose to edit their code in an IDE (Interactive Development Environment). Examples of great IDEs include VS Code, Atom and Sublime.
- IDEs include features such as *syntax highlighting*, where the code gets coloured in ways that make it easier to read, and auto-complete.

# An example IDE: VS Code



# How do I run code?

- First of all, you need to have Python installed on your computer. We won't go through this now, but we will share with you step-by-step instructions on how to do this (it's not too difficult!)

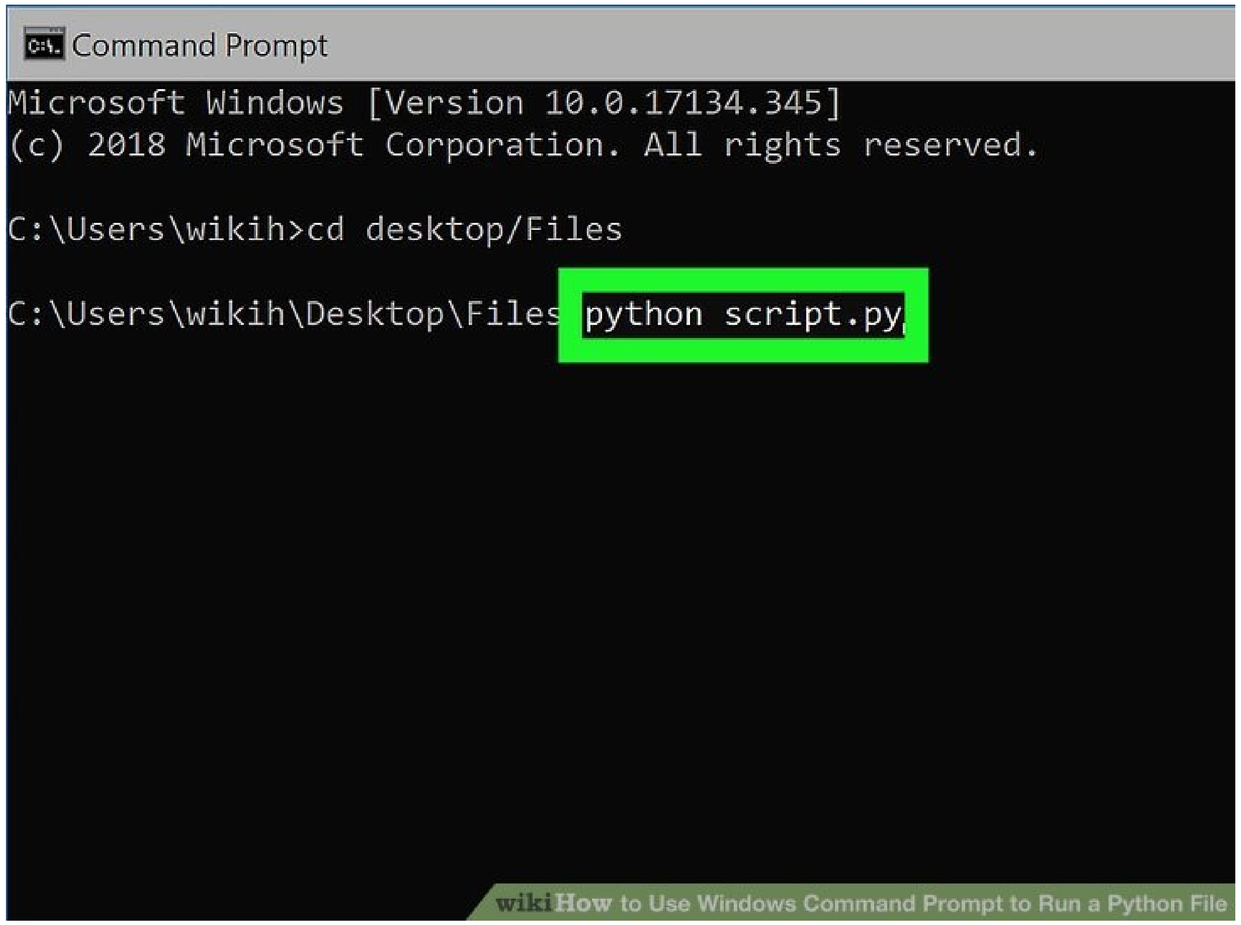
- You then have several options for running the code itself. The easiest way is to just use an IDE with a 'run' button which you can click.

- However, you will often see people use the 'command prompt' to run code. This is a special windows application that can help you interact with the computer.

- The command prompt is always 'at' a certain location in your file system. If there is a python file at this same location called `script.py` you can run it by typing

```
python script.py
```

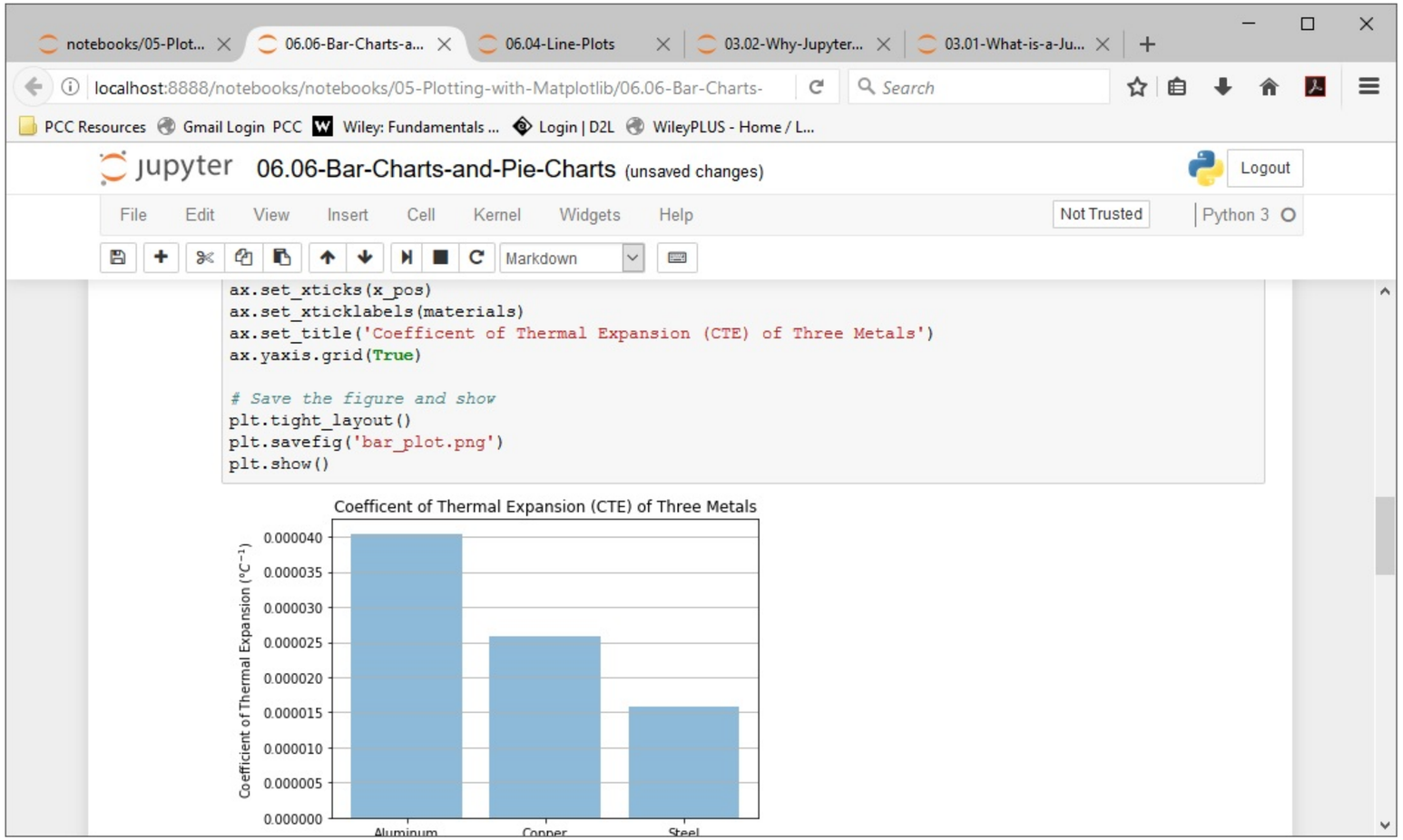
# How do I run code?



# Jupyter notebooks

- Another great place to read and write Python code is in Jupyter Notebooks. This is what we will be using in this course.
- Jupyter is an application, similar to an IDE, that runs in your browser.
- Jupyter is made up of 'cells'. Each cell can either be a *code* cell or a *markdown* cell.
- In code cells, you can write and run little snippets of code individually.
- In markdown cells you can write actual text with things like bullet points, images, and tables. This is useful for explaining what's going on in the code.
- In this course you will only need to write in the code cells.

# Jupyter notebooks



# Before we begin...

1. Learning to code is like learning a new spoken language. You won't be fluent over night! But, given time, **anyone** can become excellent.

2. Programming is 5% talent and 95% how much time you put in.

3. Programming is a creative process! It's about solving problems the way *you* want to solve them. Once you have the basic tools, it's up to you to build your own solutions.

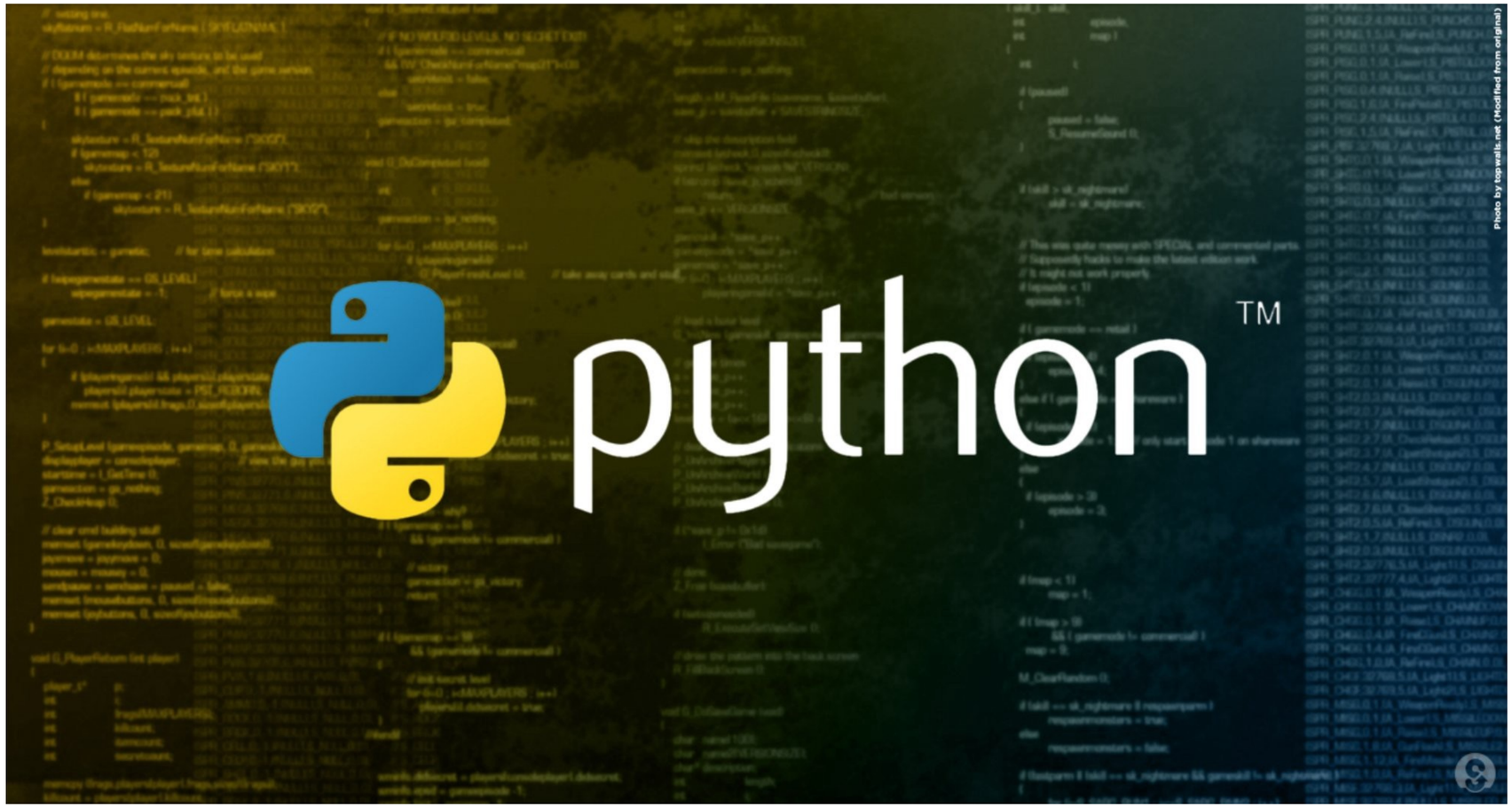
4. Learning to program, like any new skill, is tough at the start. Try to push through that! It *really* is worth it in the end.

5. The key to staying motivated is to work on problems that you find interesting.

6. **Google is your friend**. All programmers, no matter how experienced, search for solutions to their problems online. Check especially stackoverflow.com - it's a fantastic resource.

# Ready?

# Programming with Python



# The print function

- One of the most fundamental and useful tools in all of Python is the `print()` function.

- The print function is used to display something on the screen (it has nothing to do with ink printers!)

```
print('Hello world!')
```

```
Hello world!
```

- The print function takes in some input and then displays it.

- The thing that you want to display **must be placed inside a pair of brackets**.

# The print function

In this case, our command

```
print('Hello world!')
```

can be translated as

> Display to the screen the piece of text 'Hello world!'

- The quotation marks here indicate that what we want python to display is a piece of text - *more on this later.*

# Python and numbers

- We will return to text later. However, one thing python is great for is numbers.

- Python can do many things. But one of the simplest, is just to use it like a regular calculator.

```
print(2 + 2)
```

```
4
```

- Python will accept all the regular mathematical expressions that you are used to in Excel.

# Python and numbers

- Here is a list of some of the most common mathematical expressions you are able to use in Python.

| Operation | Maths symbol | Python symbol | Example |
| --- | --- | --- | --- |
| Addition | $+$ | `+` | `2 + 2` |
| Subtraction | $-$ | `-` | `5 - 2` |
| Multiplication | $\times$ | `*` | `2 * 2` |
| Division | $\div$ | `/` | `6 / 2` |
| Powers | $x^y$ | `**` | `5 ** 2` |

# Maths examples

```
print(0.99 * 100)
print(8 / 2)
print(5 ** 2)
```

```
99
4
25
```

- BE CAREFUL! The order of operations follows BODMAS (Brackets, Orders, Division, Multiplication, Addition, Subtraction).

```
print(10 + 10 / 2)
print((10 + 10) / 2)
```

```
15
10
```

# Let's try and break Python...

```
print(1 / 0)
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-1-bc757c3fda29> in <module>
----> 1 1 / 0

ZeroDivisionError: division by zero
```

- In Python, if you do something "illegal" you will get an error message. This means Python doesn't know how to handle your request.
- Error messages can come from lots of different sources. Mathematical impossibilities like this or, more commonly, syntax errors like forgetting brackets!

```
print 10
```

```
  File "<ipython-input-4-36fa854e030d>", line 1
    print 10
          ^
SyntaxError: Missing parentheses in call to 'print'. Did you mean print(10)?
```

# What to do when you get an error

- Errors are a fact of life when it comes to programming. In fact, the majority of a programmer's time can be taken up by fixing errors or "debugging".

- When you do get an error, don't panic! It's totally normal even though all the red text makes you feel like you've done something terrible!

- What you should do:

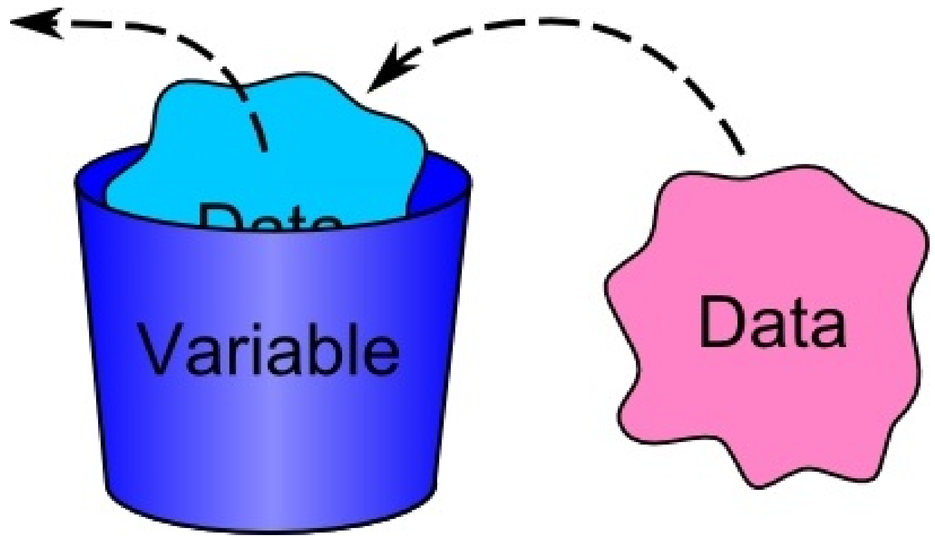
1. First try reading the error message. Sometimes it will tell you exactly what you did wrong.

2. Other times it may be more difficult to work out what you did from the error message. In this case it's a great idea to copy and paste some of the message into Google.

# Variables

- Variables are fundamental to all programming languages.

- You can think of a variable as a box, with a name, that stores some data. This data can be read and modified.



# Variables

- Every variable you create has its own unique name. To create a new variable simply type the name you want, and assign it a value using the equals `=` symbol.

```
speed = 28
```

- To check the value of this variable, use the `print()` function.

```
print(speed)
```

```
28
```

# Variables

- Once we have variables defined, we can use them in some further calculation.
- Here we imagine a plane flying at 567 mph for 15 hours. How far does it travel in miles?

```
speed = 567
time = 15
print(speed * time)
```

```
8505
```

- In fact, this value could also be assigned to a new variable

```
distance = speed * time
print(distance)
```

```
8505
```

# Naming variables

- You can give variables more-or-less any name you like. However, there are a few rules to be aware of:

1. Variable names can only contain letters, digits, and underscores (`_`) and cannot start with a digit.
2. Variables cannot contain spaces in their name.
3. Variables cannot contain any special characters such as `", $, %, &` etc.
4. Variables are case sensitive, so `my_variable` is not the same as `My_Variable`.

# Naming variables

- Below are some examples of valid variable names, and some invalid variable names.

| Valid name | Invalid name | Reason invalid |
| --- | --- | --- |
| `a2` | `2a` | Starts with a number |
| `my_variable` | `my variable` | Contains a space |
| `a_long_variable` | `a long variable` | Contains multiple spaces |
| `Two_Pounds` | `£2` | Contains a special character |
| `a_plus_b` | `a+b` | Contains a special character |

# Variables

- Once you have created a variable, it can be overwritten or reassigned.

```
my_number = 5
print(my_number)
my_number = 6
print(my_number)
```

```
5
6
```

- Be careful! Once you reassign a variable, the old value is gone forever.
- One slightly subtle feature of variables is that you can reassign a variable using it's current value

```
my_number = my_number + 1
print(my_number)
```

```
7
```

# Text data (strings)

- In addition to numbers, Python also handles text. In programming, pieces of text are referred to as "strings".
- Any piece of text needs to be enclosed in either single `'` or double `"` quotes.

```
print('a piece of text')
```

```
a piece of text
```

- Like numbers, text can be assigned to a variable.

```
my_text = "another piece of text :)"
print(my_text)
```

```
another piece of text :)
```

# Text data (strings)

- Long pieces of text that flow over multiple lines need to be enclosed in either triple single quotes `'''` or triple double quotes `"""`.

```
long_text = '''Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. '''

print(long_text)
```

```
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
```

# Working with strings

- Strings can be added together ("concatenated")

```
first_name = 'Daveed'
second_name = 'Diggs'
full_name = first_name + ' ' + second_name
print(full_name)
```

```
Daveed Diggs
```

- Strings can also be multiplied by a number

```
print('=' * 10)
```

```
==========
```

# Strings

- However, strings cannot be subtracted. Attempting to do so will result in an error!

```
print('a' - 'b')
```

```
TypeError                       Traceback (most recent call last)
----> 1 print('a' - 'b')
TypeError: unsupported operand type(s) for -: 'str' and 'str'
```

- You also can't multiply a string by a string

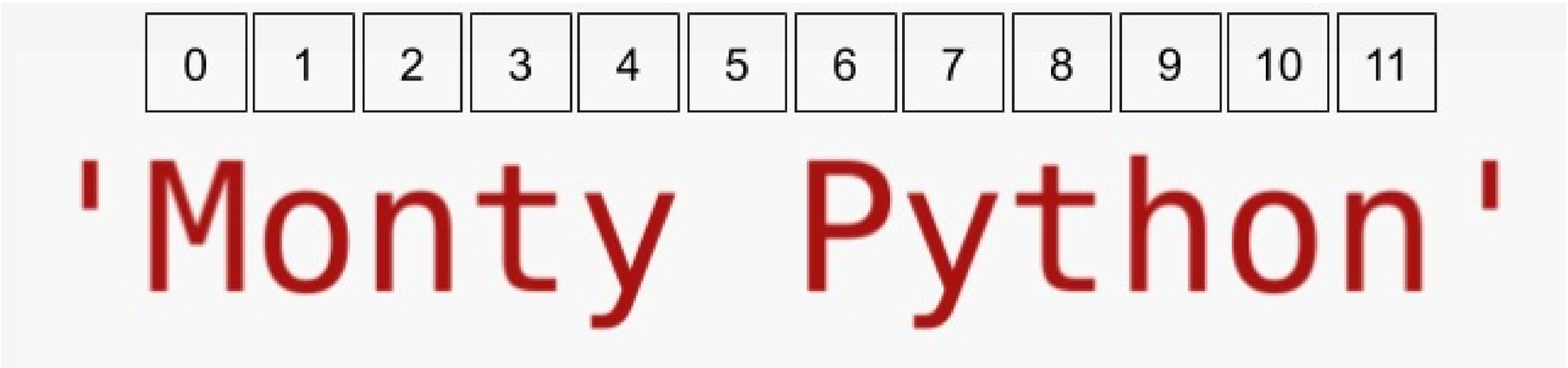
```
print('hello' * 'h')
```

```
TypeError                       Traceback (most recent call last)
----> 1 print('hello' - 'h')
TypeError: cant multiply sequence by non-int of type 'str'
```

# String-indexing

- Really, a string is made composed of an ordered sequence of individual characters. (That's where it gets the name - its a *string* of characters).

- Characters in a string are numbered **starting from zero.**



# String-indexing

- Individual characters can be accessed from within a string using square brackets. This is called *indexing*.

```
my_string = 'Monty Python'
print(my_string[6])
```

```
P
```

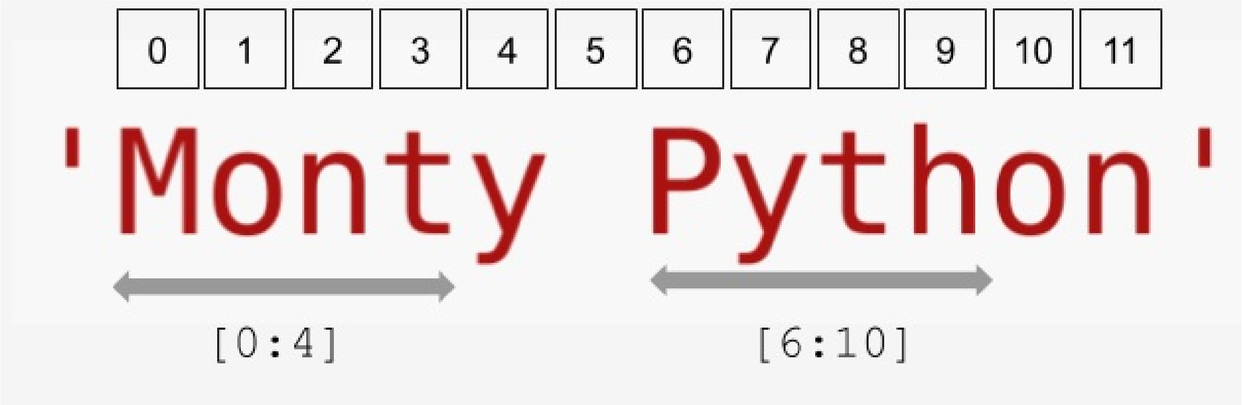
- Always remember that that characters start from zero.

```
print(my_string[0])
```

```
M
```

# String-indexing

- Sub strings (a chain of characters within a string) can be accessed using square brackets as `[start:stop]`. Here, `start` is inclusive, `stop` is non-inclusive.




```
print(my_string[6:10])
```

```
Pyth
```

# Lists

- A list is data type that holds an ordered sequence of other pieces of data.
- Lists can hold anything: numbers, strings, even other lists.
- A list is created by enclosing a set of items between square brackets, separated by commas.
- Each item in the list can be of a different type.

```
my_list = [42, 'dog', -0.5, 'cat', 'fish']
print(my_list)
```

```
[42, 'dog', -0.5, 'cat', 'fish']
```

# Lists

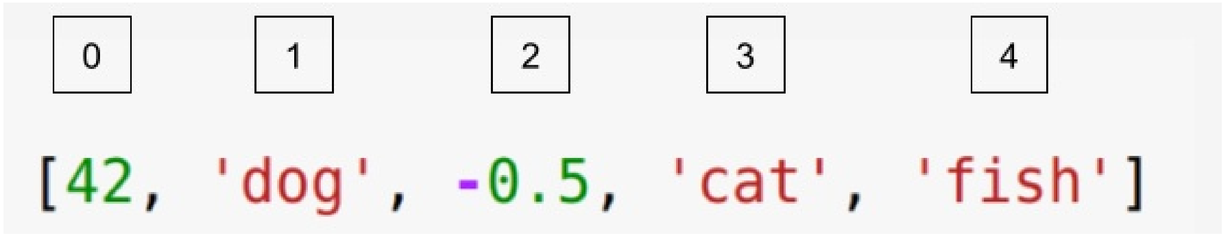
- You can find out the length of a list by using the `len()` special function.
- `len()` is a *function*, just like `print()`. That means it takes an input in brackets.
- If you pass a list into the `len()` function it will "return" the length of that list. That means `len(my_list)` can be assigned to a variable.

```
my_list = [42, 'dog', -0.5, 'cat', 'fish']

my_list_length = len(my_list)

print(my_list_length)
```

```
5
```

# Lists

- Just as with strings, individual items or subsections can be accessed by indexing.




```
print(my_list[0])
```

```
42
```

```
print(my_list[1:3])
```

```
['dog', -0.5]
```

# Lists

- As well as retrieving, you can also override individual items in a list.

```
my_list = [42, 'dog', -0.5, 'cat', 'fish']

my_list[0] = -11111

print(my_list)
```

```
[-11111, 'dog', -0.5, 'cat', 'fish']
```

# Lists

- Two lists can be added together

```
list1 = ['beans', 'cheese', 'jam']
list2 = ['butter', 'nutella', 'marmite']
list3 = list1 + list2
print(list3)
```

```
['beans', 'cheese', 'jam', 'butter', 'nutella', 'marmite']
```

- In this way, we can add a new item onto the end of a list

```
list3 = list3 + ['peanut butter']
print(list3)
```

```
['beans', 'cheese', 'jam', 'butter', 'nutella', 'marmite', 'peanut butter']
```

# Lists

- In fact, this process of adding a single item onto the end of a list is so common that there's a special way to do it. It's called the `.append()` *method*.

```
list3 = list3 + ['peanut butter']
```

Is exactly the same as saying

```
list3.append('peanut butter')
```

- A *method* is similar to a *function*: it has brackets in which something can be placed. The difference is a method is attached to anther object, in this case a list.

- So the `.append()` method is a special function that is attached to all lists.

# Lists

- One common way to create a list is to start with an empty list.

```
empty_list = []
```

- ... and then add new items one by one

```
empty_list.append('shoe')
empty_list.append('glove')
empty_list.append('sock')
```

- Now the list will have new items in it

```
print(empty_list)
```

```
['shoe', 'glove', 'sock']
```

# Thank You