



# **Computing Sciences: Programming**

## **CIS1048-N-KL1-2019**


**Diving Right In!**

Week 1: Lecture 1

**School of Computing, Engineering & Digital Technologies**

[tees.ac.uk/computing](http://tees.ac.uk/computing)

# Schedule



Week	Lectures	Subject Matter
1	Diving Right In!	Getting set up, writing our first simple programs and essential information regarding the course.
	Program Flow Control	Getting our programs to make decisions using conditional statements, and unlocking the power of programming with loops.
2	Functions and Advanced Lists	Writing structured and reusable code using functions, and processing lists of data using functional techniques.
	Data Persistence, Dictionaries and Tuples	Reading from and writing to files, and taking advantage of Python's non-list collection types: tuples and dictionaries!
3	Data Processing and Visualisation	Processing and visualising large amounts of data using the Pandas and Matplotlib libraries.
	Design, Testing and Version Control	Taking advantage of design and testing principles as well as source control using GitHub to manage and organise our software projects as they grow in complexity.
4	Object-Oriented Programming	Putting object-oriented principles to work in our code to create powerful and reusable application logic that captures real-world requirements.
	Network Programming	Writing programs that connect to and communicate over networks, including the internet.

Note: The schedule may be adjusted based on class progress.

# A (Very) Brief Introduction

I'm Saul, a software verification researcher and lecturer in programming and cybersecurity here at Teesside University. I currently work mainly with the programming languages Python and Java.

You can find me on GitHub (don't worry if you don't know what this is yet!), on Twitter and through my website:

GitHub: [@lambdacasserole](https://github.com/lambdacasserole)

Twitter: [@lambdacasserole](https://twitter.com/lambdacasserole)

Web: <https://sauljohnson.com>



# Learning Objectives

## Introducing the Module!

- Get familiar with the module aims and learning outcomes.
- Familiarize ourselves with how this module will be delivered.
- Take a look at how the module will be assessed.

## Getting Started Coding!

- Define programming: what is it?
- Introduce the Python programming language.
- Discover how to install, read about and work with Python.
- Begin simple Python programming using IDLE.

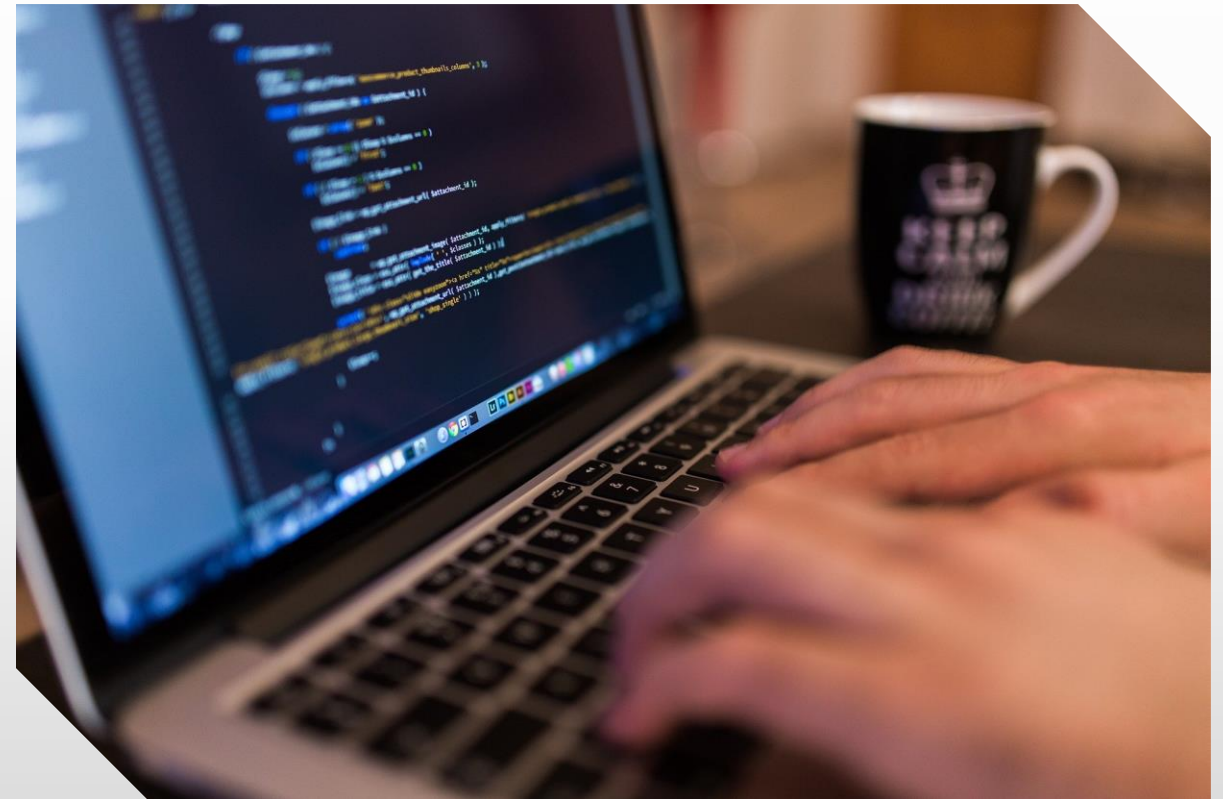
# Module Introduction

Lecture 1: Part 1, Section 1

# Module Outcomes

**During this module, we want to...**

- Provide you with the knowledge of programming language constructs typical of programming languages used in mainstream computing.
- Apply this knowledge to solve problems typical of those where programmers are required to implement solutions.



# During this Module

## During this module, we'll be looking at...

- Busting programming jargon
- Sequences and selection
- Conditional constructs
- Loop constructs
- Arrays/Lists
- Strings
- Functions
- Object-Oriented Programming
- Reading and writing files
- Testing our code
- Debugging tips and tools
- Common frameworks and libraries
- Data processing/visualization
- Exploring similarities and differences between Python and other programming languages.

# Skills You'll Acquire

- **Personal and Transferable Skills:**

- Demonstrate a developing ability to communicate in an academic context.
- Competently select and utilise the appropriate language constructs.

- **Research, Knowledge and Cognitive Skills:**

- Understand the fundamental concepts of imperative programming languages.
- Identify and design appropriate solutions to a variety of problems.
- Recognise what is and is not syntactically and structurally correct code.
- Implement and test programs that satisfy the requirements set.

- **Professional Skills:**

- Create solutions to problems using programming tools and techniques.

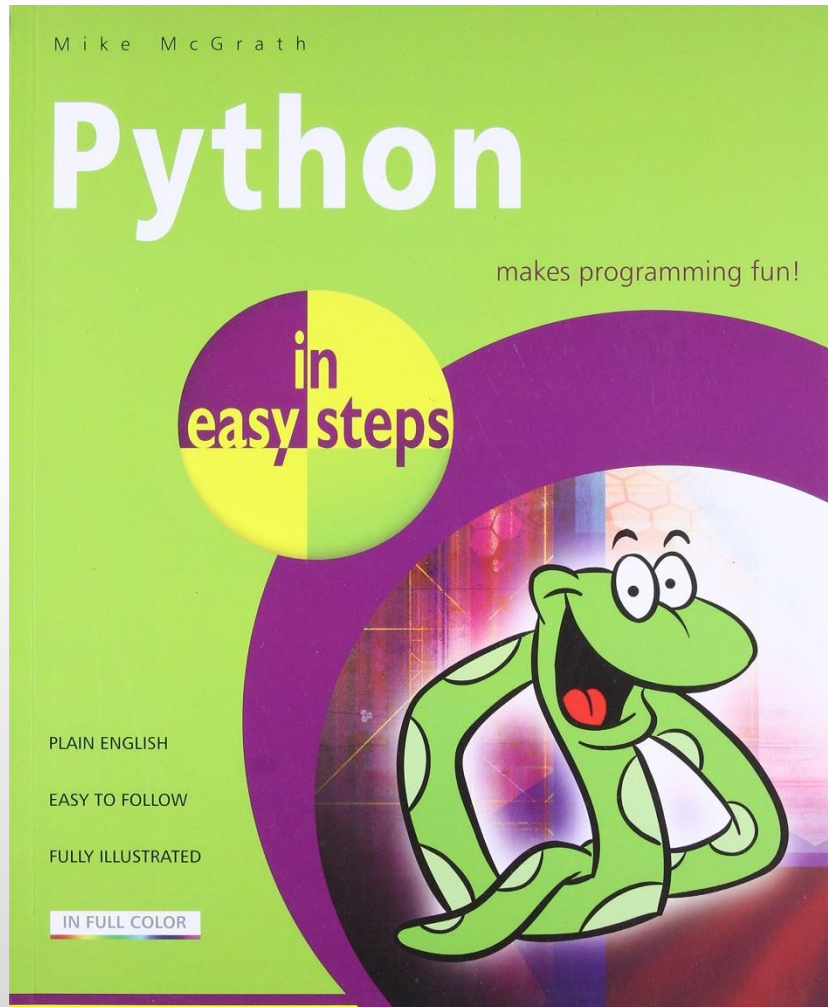


# Assessment

- This module is accredited, and as such **is assessed!**
- As long as you engage with the module and invest time and effort in it, you'll be absolutely fine.
- You'll be completing worksheets week-by-week. A total of 4 over the course of the module.
- At the end of the module, I'll collect these in, mark them and give you your grade, which will be either pass or fail.



# Recommended Reading



Though the material provided for you in this module contains everything you need, I'm often asked by students to recommend reading.

My personal favorite Python book is *Python in Easy Steps* by Mike McGrath. It's available to buy from Amazon etc. for under £10.

Work through it to keep your skills sharp. Keep it on your desk as a reference. Refer to it as and when you need it!

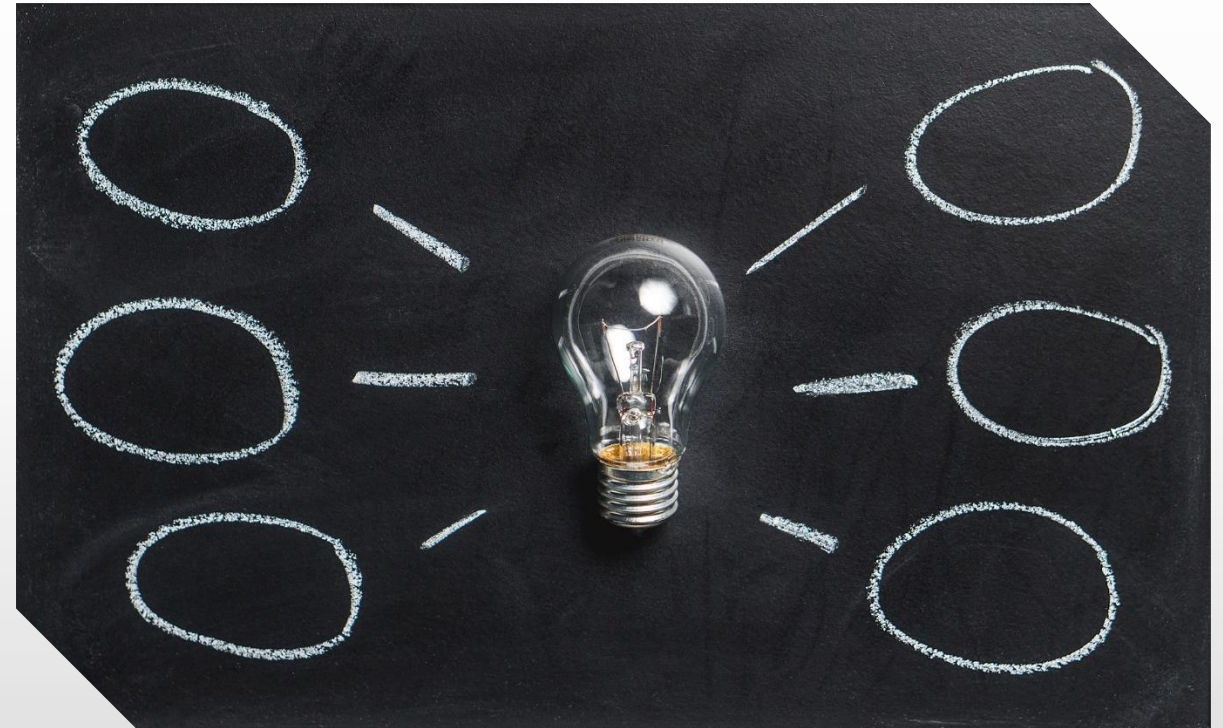
# Beginning Programming

Week 1: Lecture 1, Section 2

# Why do you want to be a programmer?

You'll all be programmers by the end of this 4-week course. But why do you want to become a programmer?

- Supercharging your productivity?
- As a rewarding creative outlet?
- To upskill in order to enhance your career prospects?



# So What's a Program?

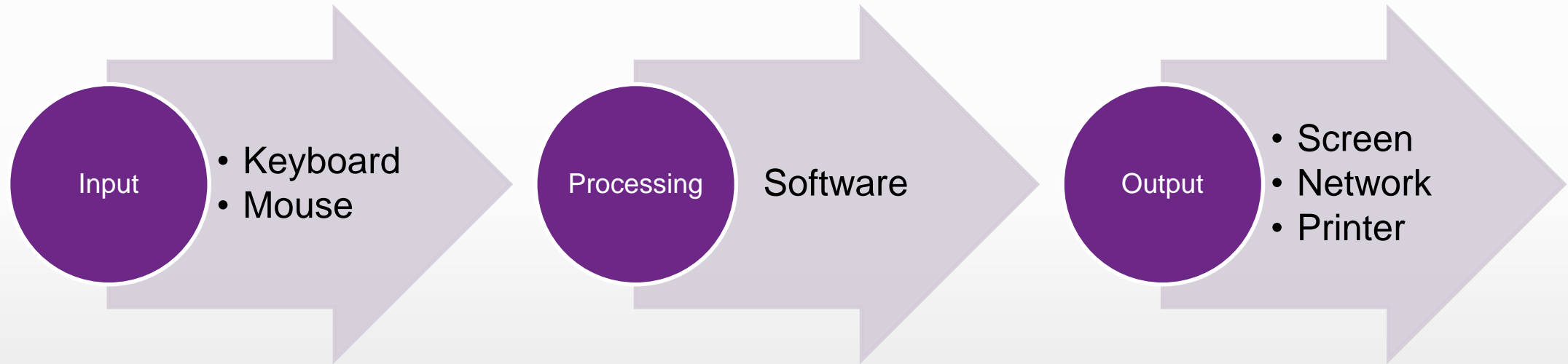
- A program is a set of *statements* in a *programming language*. These can be:
  - *Operators and operands*
  - *Expressions*
  - *Assignments*
  - *Keywords*
- Syntax (grammar) and semantics (logic)
- We must be very precise when we write programs because...

```
x = 10
y = 20
z = x + y
print(x)
```

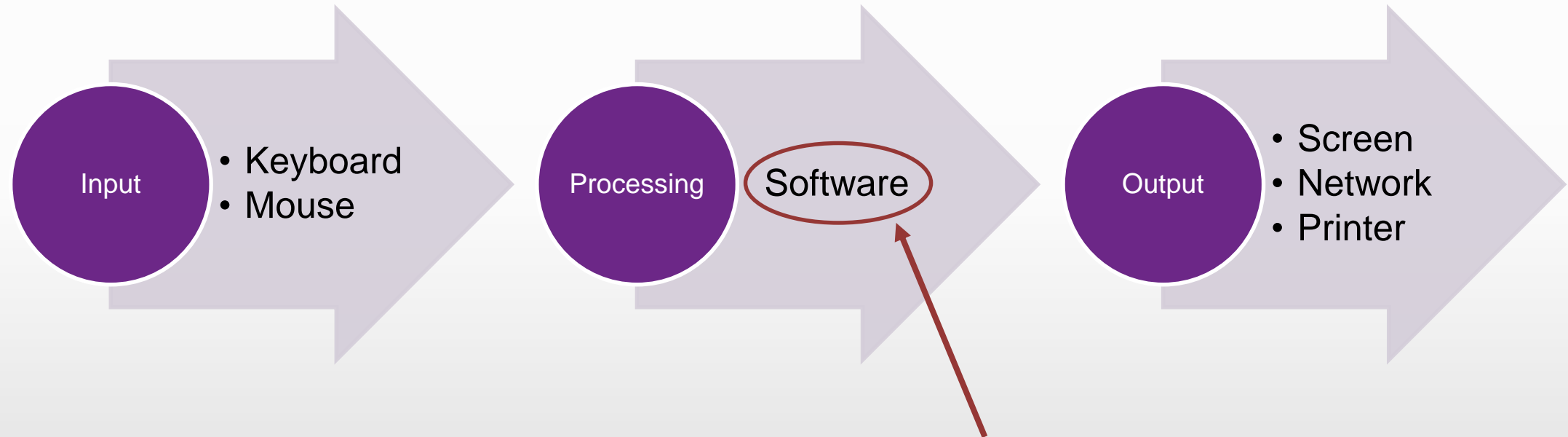
# So What's a Program? (Cont.)

Computers don't think! They **don't truly understand** the program you write, **only how to run it!** They have no notion of context; they will do as you tell them - **nothing more and nothing less.**

# A Generic Program



# A Generic Program (cont.)



The bit we're most interested in!



# Programs are Just Equations!

- Don't get worried, this will become clear!
- All a program does is take input from the user, change (or, more technically, *transform*) the data and give it as output.
- On the right, there's a mathematical equation specifically a *function definition* (not a Python one, we'll get to that later!).
- See how it works? Depending on what we give as  $x$ , we get a different answer!
- Here,  $x$  is the input,  $f$  is the program and the right-hand side is the output.

$$f(x) = x \times 2$$

$$f(2) = 4$$

$$f(10) = ?$$

# Here's the Python Equivalent!

$$f(x) = x \times 2$$

```
x = input()  
ans = x * 2  
print(ans)
```

# What is Python?

Week 1: Lecture 1, Section 3

# Python: Origins

- Python is a programming language that was invented by Dutch programmer Guido van Rossum in 1990. It's named after the Monty Python comedy group!
- It's free, powerful, expressive, readable, approachable and cross-platform (it'll run on Mac, Linux and Windows).
- It has a massive following, and mountains of support is available online and in person!
- We'll be learning Python 3 on this module, which is a **different language** to Python 2.



# Python: Origins (cont.)

- Python is also an *interpreted language*. This means that additional software is required to run Python programs, which are written in plain text. That's right, **all you need to write Python code is Notepad!**
- To run Python programs however, you'll need the Python interpreter, available here:  
<https://www.python.org/downloads/>
- We'll be taking a look at getting this set up on Windows. **It is critical that you download and install this software!**



# Why Python?

- It has a relatively simple language syntax that is easier to learn than some other languages.
- Is widely used in many areas of computing:
  - Data analysis and data science
  - Networks, cybersecurity, digital forensics
  - Task automation
  - Scientific and research computation
  - Games and 3D animation
  - Web development



# Java vs. Python (cont.)

## Java: Traditional Object-Oriented Language

```
public class HelloWorld
{
    public static void main(String[] args)
    {
        System.out.println("Hello World");
    }
}
```

## Python: Designed for Expressiveness

```
# Python
print("Hello World")
```

# Java vs. Python (cont.)

## Java: Traditional Object-Oriented Language

```
// Java  
int temp = var1  
var1 = var2  
var2 = temp
```

## Python: Designed for Expressiveness

```
# Python  
var2, var1 = var1, var2
```



# Some drawbacks



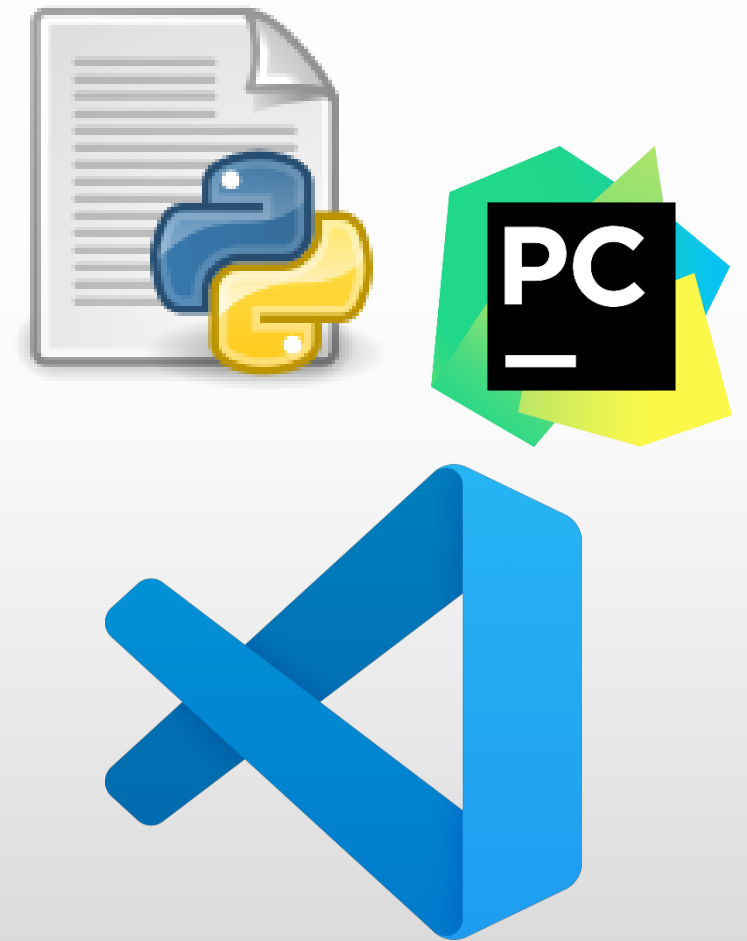
- Python can sometimes be a bit *slower* than so-called “compiled languages” (such as C++) that are converted to raw machine code before they are run.
- Due to its interpreted nature, Python does not check variable types at compile time (more on this later).

# Getting set up to write code!

Lecture 1: Part 1, Section 4

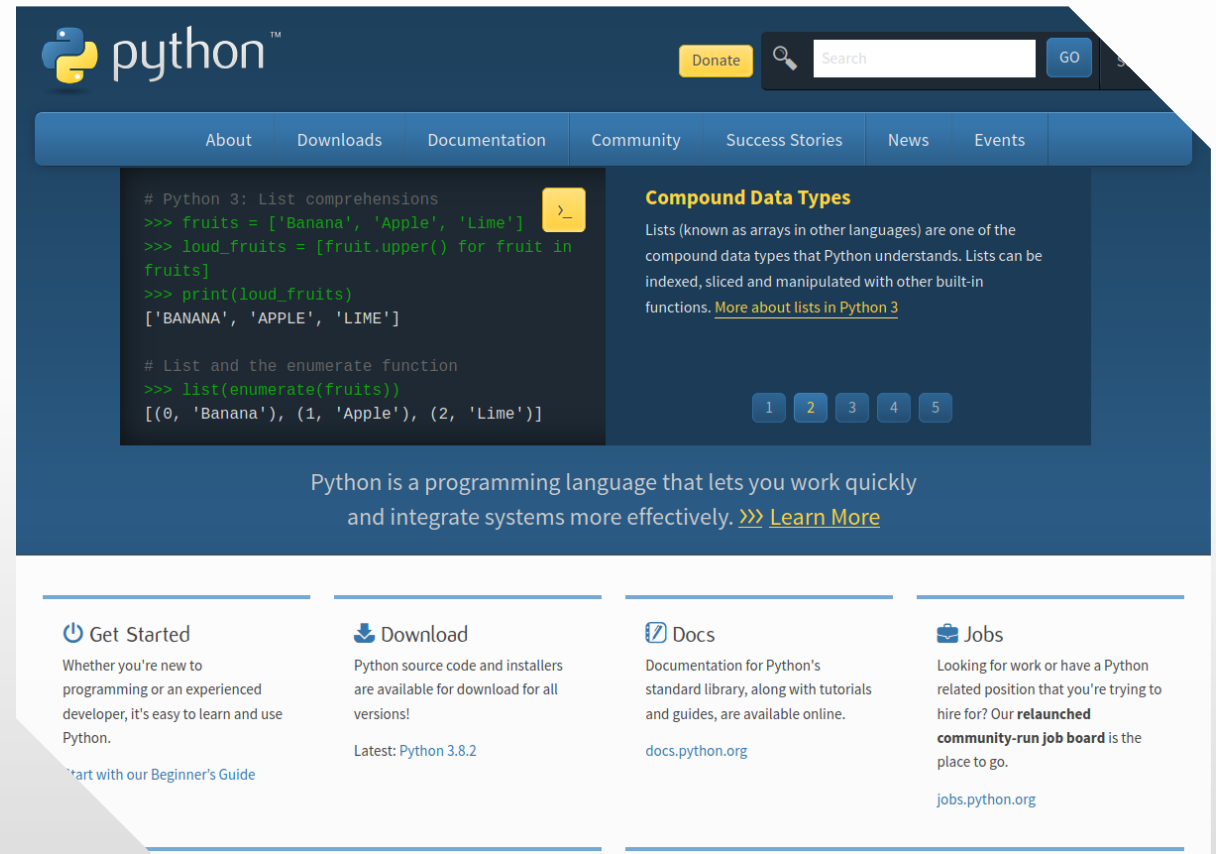
# Getting set up with an IDE

- An IDE, or *integrated development environment*, is a text editor that will allow you to write and run programs from one piece of software. Some Python IDEs include:
  - **Python IDLE** (Integrated Development and Learning Environment): **We'll be using this!**
  - **PyCharm**: More complex, but potentially more powerful. Free for students!
  - **Visual Studio Code**: Provides Python support through a plugin (extension).



# Installing and Launching IDLE

- IDLE is without question the simplest Python IDE to get started with.
- It's free and installed alongside Python by default. Bear this in mind when you install it from home.
- Let's get it installed right now...



# Let's write our first program!

Week 1: Lecture 1, Section 5 | *See video lecture*

# What just happened?

Week 1: Lecture 1, Section 6



# Types: What are They?

- Type is a concept in programming that relates to the kind of data the computer has stored.
- For example, adding 1 and 2 together we get 3, but what about trying to add 1 to the word "bananas".
- We can't do it, of course! This is known as a *type error*—trying to make incompatible data types do something they can't.



# Types: What are They? (cont.)

- Types in Python include:
  - Integers (whole numbers)
  - Floating point numbers (decimals)
  - Boolean values (true/false, yes/no, 1/0)
  - None (represents nothing or *null*)
- The above types are so-called *scalar* types. However:
  - Strings, sequences of letters like "hello" are *non-scalar*. What does this mean?





# What's a "Variable"?

- A variable is like a box that can be used to store a value of some *type*.
- We give variables identifiers, called *names* in order to work with them easily.
- In Python, these names can only contain letters, numbers and underscores (\_), and cannot start with a number.
- Which of the variables on the right are valid (allowed) *variable declarations*?

```
x = 10
x123 = 8
x123! = False
123x = "Hello!"
X_123 = 23.53
```

# What's a "Variable"? (Cont.)

- Some words have special meaning in Python which means we can't use them as variables.
- Some of them on the right. A complete list can be found here:  
<https://www.programiz.com/python-programming/keyword-list>
- Some so-called *reserved words* in Python include:
  - `import`
  - `class`
  - `def`
  - `True`
  - `False`
  - `if`
  - `continue`

# The `print()` Function

- The `print()` function will write some text to the screen.
- The `print()` function can take multiple arguments separated by a comma.
- The comma introduces a space in the output separating the printed content of adjacent arguments.
- The *named argument* `end=""` can be used in the print function to stop a *linefeed* prior to next use of a `print()` statement (we'll use this in a later session).

```
print()  
print("Result is: ", result)  
print(x, "*", y, "=", (x * y))
```

# The `input()` Function

- The `input()` function will get a value from the user. **It always returns a string!**
- The `input()` function should normally be given with a prompt to observe good user interface practice (single argument only)
- The prompt can be a literal (as indicated) or a string variable.
- The `input()` function would normally include a variable name assignment to record whatever was entered in response.

```
input()  
input("Please enter something: ")  
name = input("And your name is?")
```

# Casting

- Casting is the process of converting a value of one type to another.
- For example, if we read two numbers from the user as strings, we cannot add them together until we turn them into integers! To cast to values of a different type, we use functions including:
  - `float(x)` - converts x to a float
  - `int(x)` - converts x to an integer
  - `str(x)` - converts x to a string

```
x = input("Enter x: ")
y = input("Enter y: ")
print(x * y) # This won't work!
print(int(x) * int(y)) # This will!
# Notice the use of casting here!
```

# Operators

- Operators are functions that work on *operands*.
- Examples, with operands  $x$  as 8 and  $y$  as 3 are shown on the right.
- These are called *binary* operators. *Binary* just means '2', and because these operators take 2 operands, they get their name.
- This has nothing to do with the concept of binary numbers (i.e. 1s and 0s)!

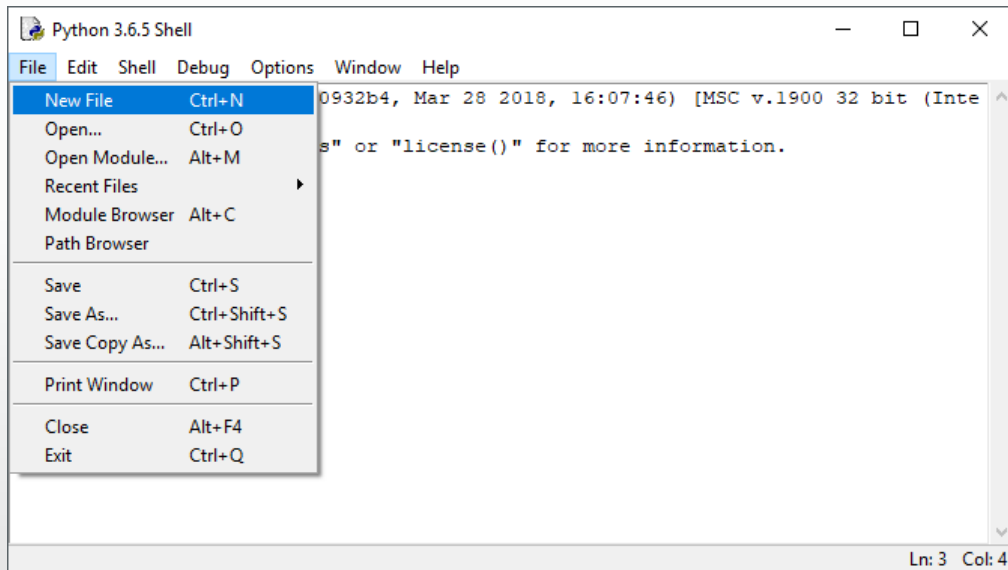
Operation	Python	Example
Addition, subtraction	+ or -	$x + y = 11$
Multiplication	*	$x * y = 24$
Division	/	$x / y = 2.67$
Exponentiation (Power)	**	$x ** y = 256$
Integer division	//	$x // y = 2$

# Conform to the Style Guide!

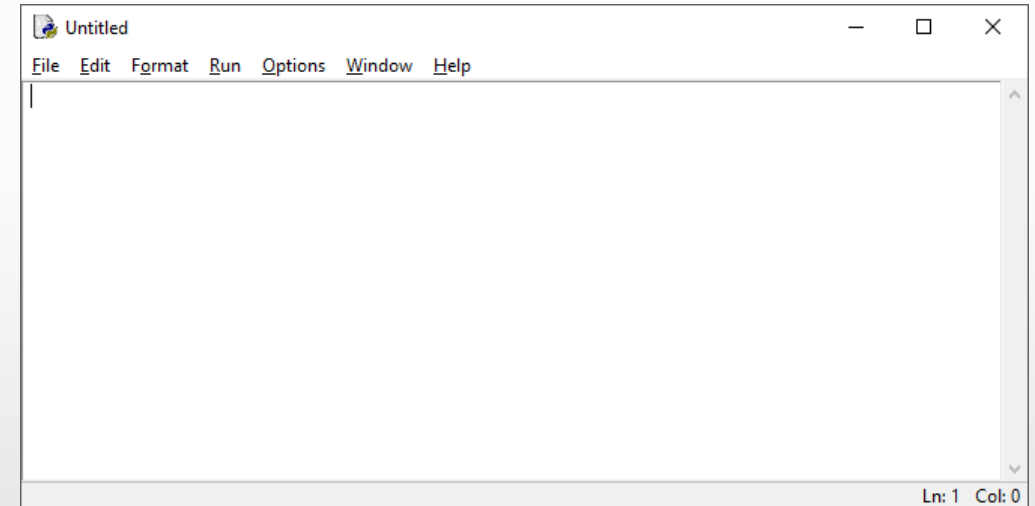
- Code must be written in a way that makes it **easy to read!**
- To help us with this, programmers often employ *style guides* to make sure that our code conforms to certain standards.
- Python's most popular style guide is called PEP 8 and you can look at it here: <https://www.python.org/dev/peps/pep-0008/>
- When we write code on this course, we want to conform to this as closely as we can.
- At the end of the session, once we've got started programming, we'll be taking a look at some well-formatted code alongside some code that has some serious readability problems.

# Creating Python Modules from IDLE

**Create a new file in IDLE...**



**Write it and run with F5!**





# Accessing support!

Week 1: Lecture 1, Section 5

# Getting Support

## I'm on Microsoft Teams!

- I'll be available throughout the module, every week via Microsoft Teams for the duration of our 3 hour sessions together from **18:00 until 21:00** on **Mondays and Wednesdays** from the **6<sup>th</sup> July to 29<sup>th</sup> July 2020**.
- Contact me there between these times to access support fast. You'll be able to share your screen with me etc. so we can get to the bottom of any difficulties you experience.



# Getting Support

## I'm contactable via e-mail!

- When I'm not available on Teams, you get in touch via e-mail at: [saul.johnson@tees.ac.uk](mailto:saul.johnson@tees.ac.uk)
- I'll aim to get back to you as soon as I possibly can.
- You can also send me your code via e-mail to get feedback on it! I'm usually able to get this feedback to you within 3 working days.



# But I don't have Office/Teams/Email!



- The University provides Microsoft Office (alongside Teams), and email services to all current students. This includes you!
- There are some additional videos on Blackboard (the University's e-learning platform) which walk you through getting all set up. These are titled:
  - Accessing University E-mail
  - Getting Started with Microsoft Teams for Remote Learning
  - Navigating Blackboard: Locating Lesson Material and Submitting Assessments
- These will all be up there already! Go check them out!

# So, let's wrap up!

Some final words before we wrap up this first lecture...