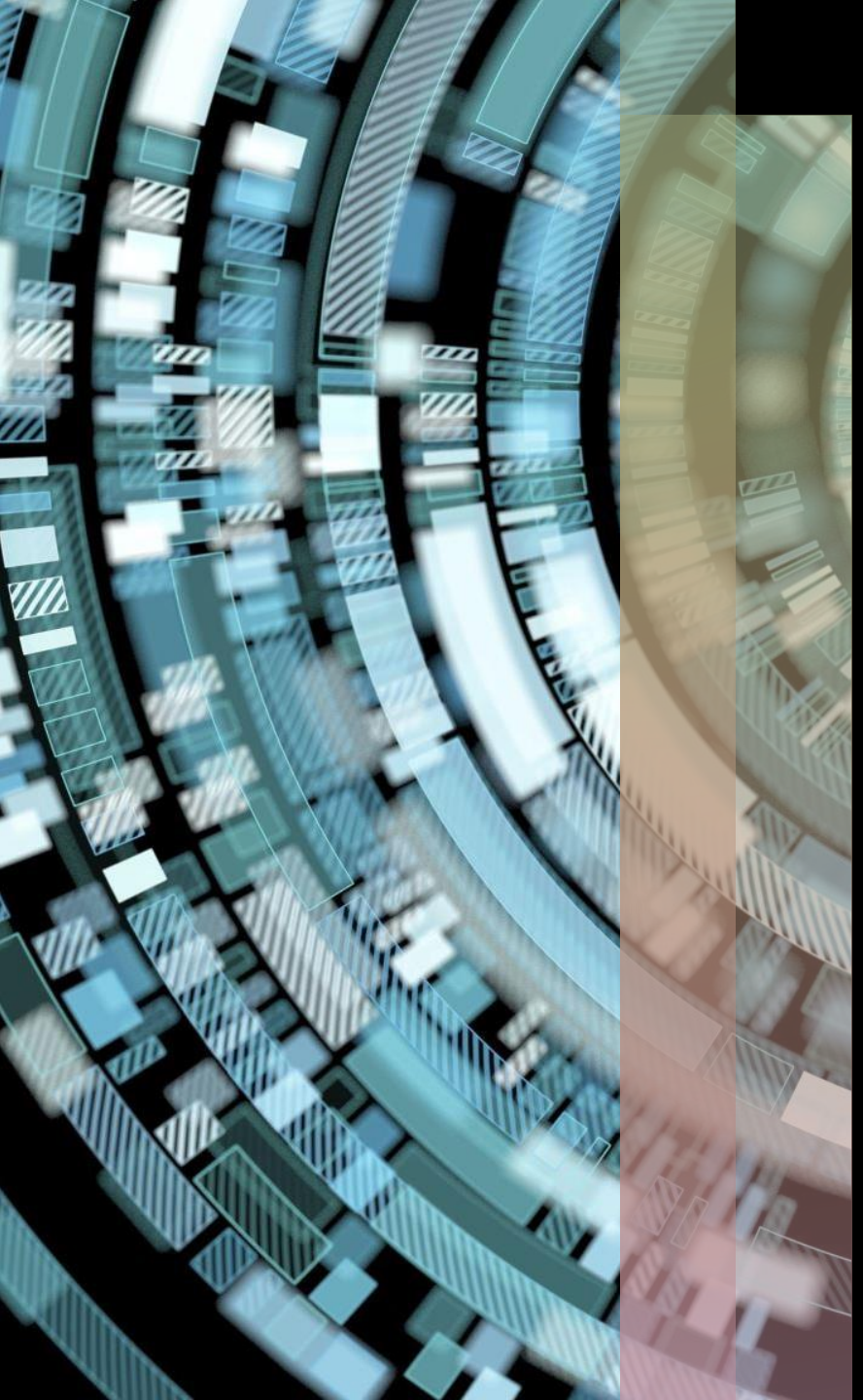
COM4013
Introduction to Software Development

- Week 1
- *Module Introduction and Some First Steps*
- *[So, what's this programming thing all about then...?]*

- *Umar Arif – u.arif@leedstrinity.ac.uk*



A Little about Me

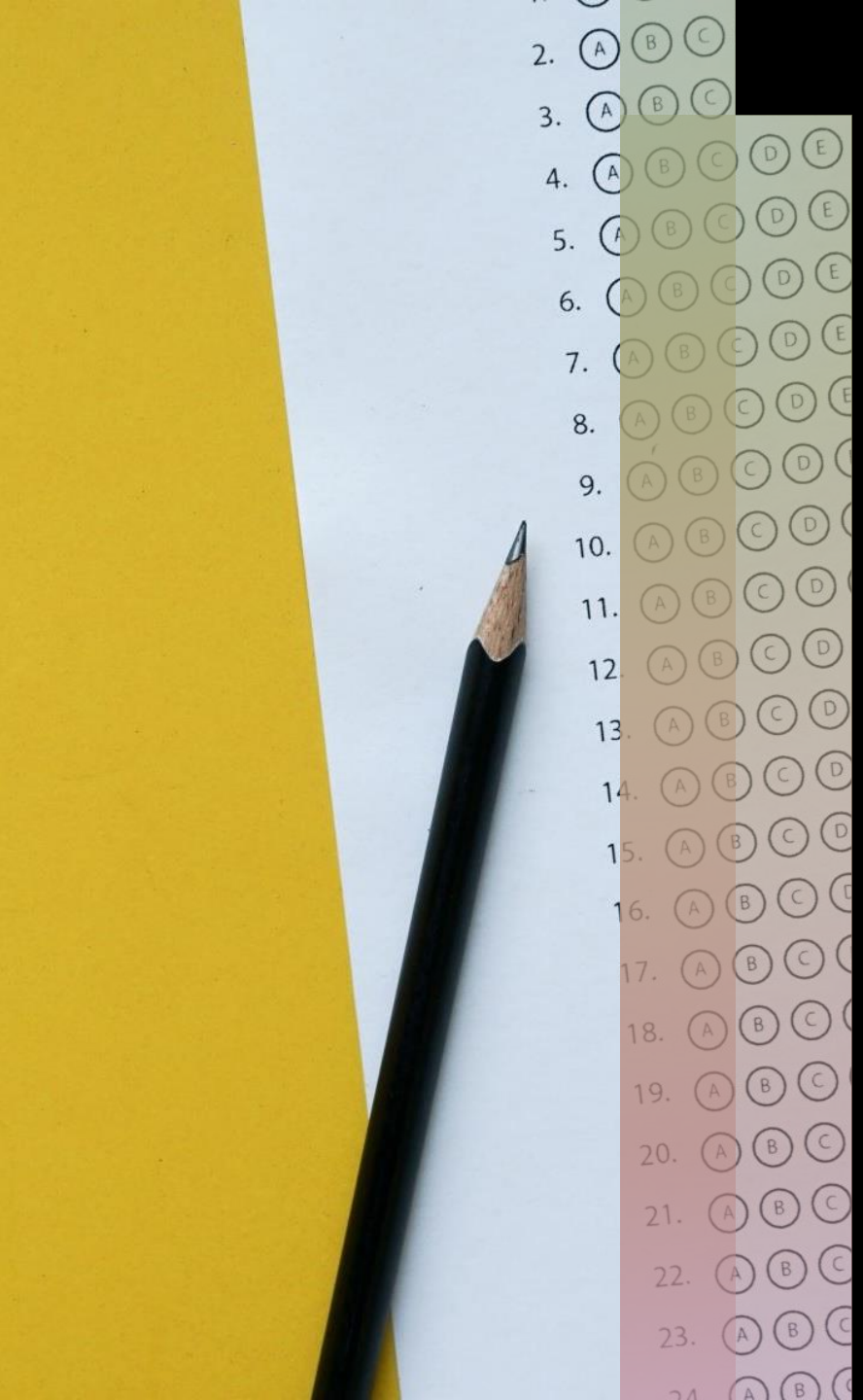
- I am an experienced Operational Researcher (Data Analyst) with an interest in Data Science and AI
- Experience
  - Modelling and forecasting for the Department for Work and Pensions. My work has contributed towards the government ability to set aside the correct amount of resources for both AME (Annual Means Expenditure) and DEL (Departmental Expenditure Limits) requirements
- Education
  - BSc Computer Science (HONS) (Foundation) (Placement) from UCLan
  - Graduated with First Class honors (with Best Overall Academic Performance and Best Project)

Module Overview

- Part 1 of the module (week 1-10) will be taught by me
  - Runs until early February

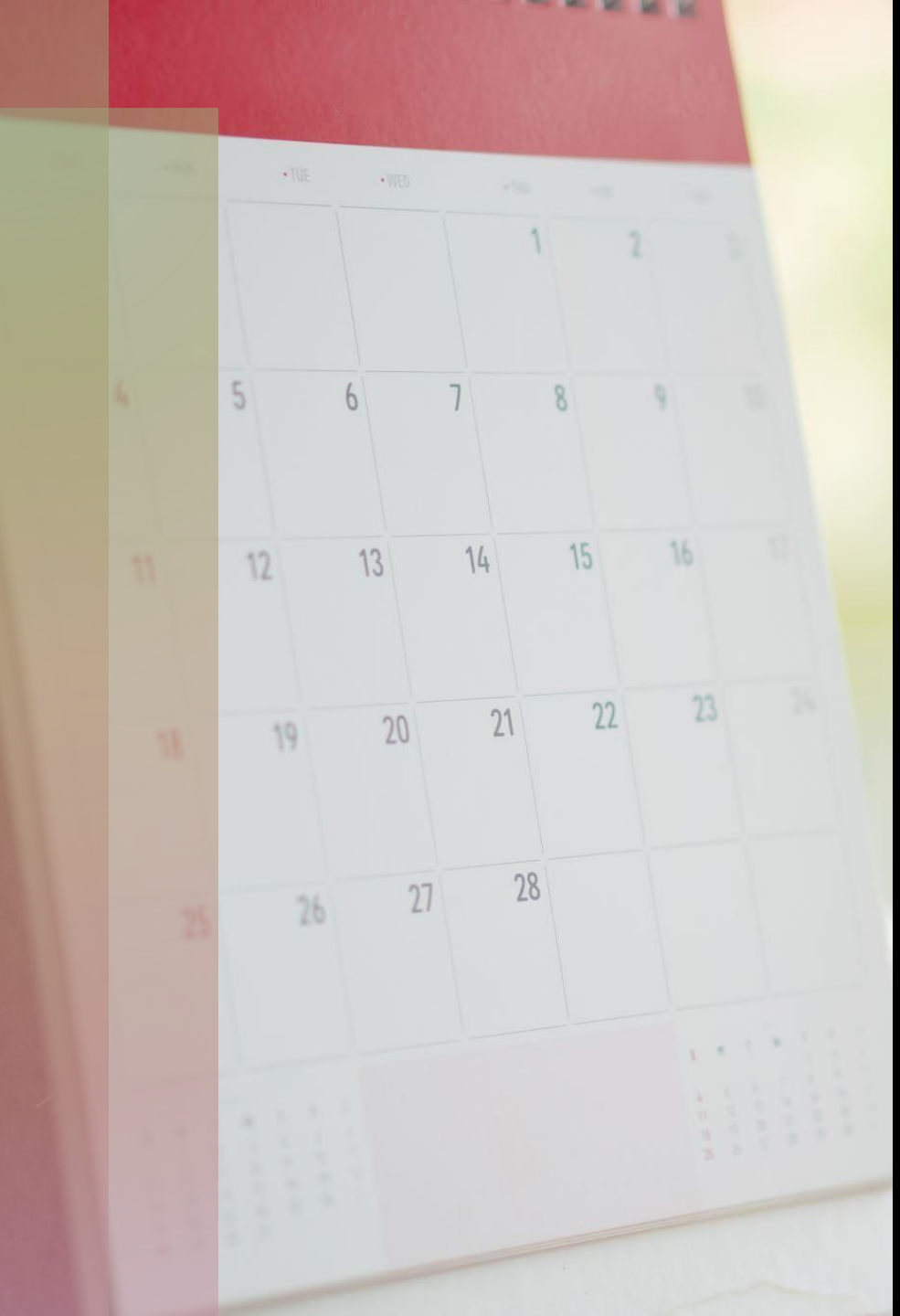
- Ten three-hour workshops with me. Message me if you require additional support.

- Focuses on the core programming concepts needed by all computing students
  - Can’t call yourself a computing student without ***some*** knowledge in this area
  - Assumes you have no programming experience
    - But sets some challenges for those who do



Module Overview

- Assessment (i.e., how do I get marked?)
- You are required to submit five assignments as part of your portfolio, which will be distributed during our lab sessions.
- This portfolio is worth 20% of the modules weighting
- Please ensure that these assignments are submitted by the beginning of February
  - You must Pass this aspect of the module. And to pass you must attempt at least 75% of the work set
  - If you fail, you might be offered a resit at the end of the second semester (along with any other resits)
    - The marks you can gain from a resit are capped.
- The module tutor will present the lectures
- Other tutors may be around to help with queries



Module Overview

- By now, you're likely familiar with LTU's Pre-Live-Post model. This entails work before, during, and after our sessions for a comprehensive learning experience
- Module resources are all held on Moodle, they will all be available in time (I may make minor changes though)
- Solutions to some of the lab will appear the week after the lab.
- Programming is a **very** practical subject
  - You will be given the concepts in lectures, but you will only learn by trying them for yourself
- Making your best effort in labs and working in your own time is vital to pass this module
  - Use the sessions as best as you can to keep up
  - Reminder: attendance at all sessions is compulsory

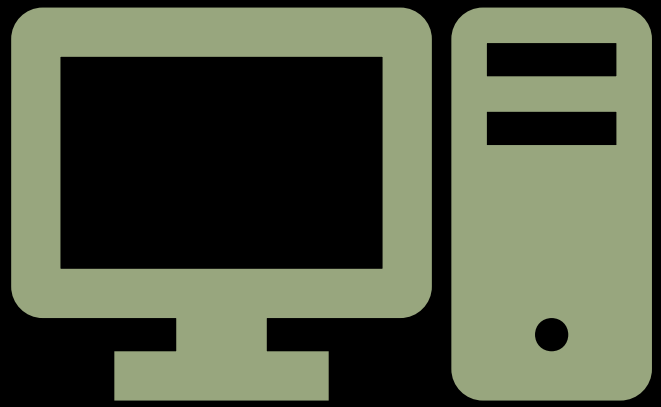
What is programming?

Exercise: Take the next 3 minutes to reflect on the concept of programming.

- You did remember a pen and paper, didn't you?
- You should be taking notes in the lecture – I say more than I write

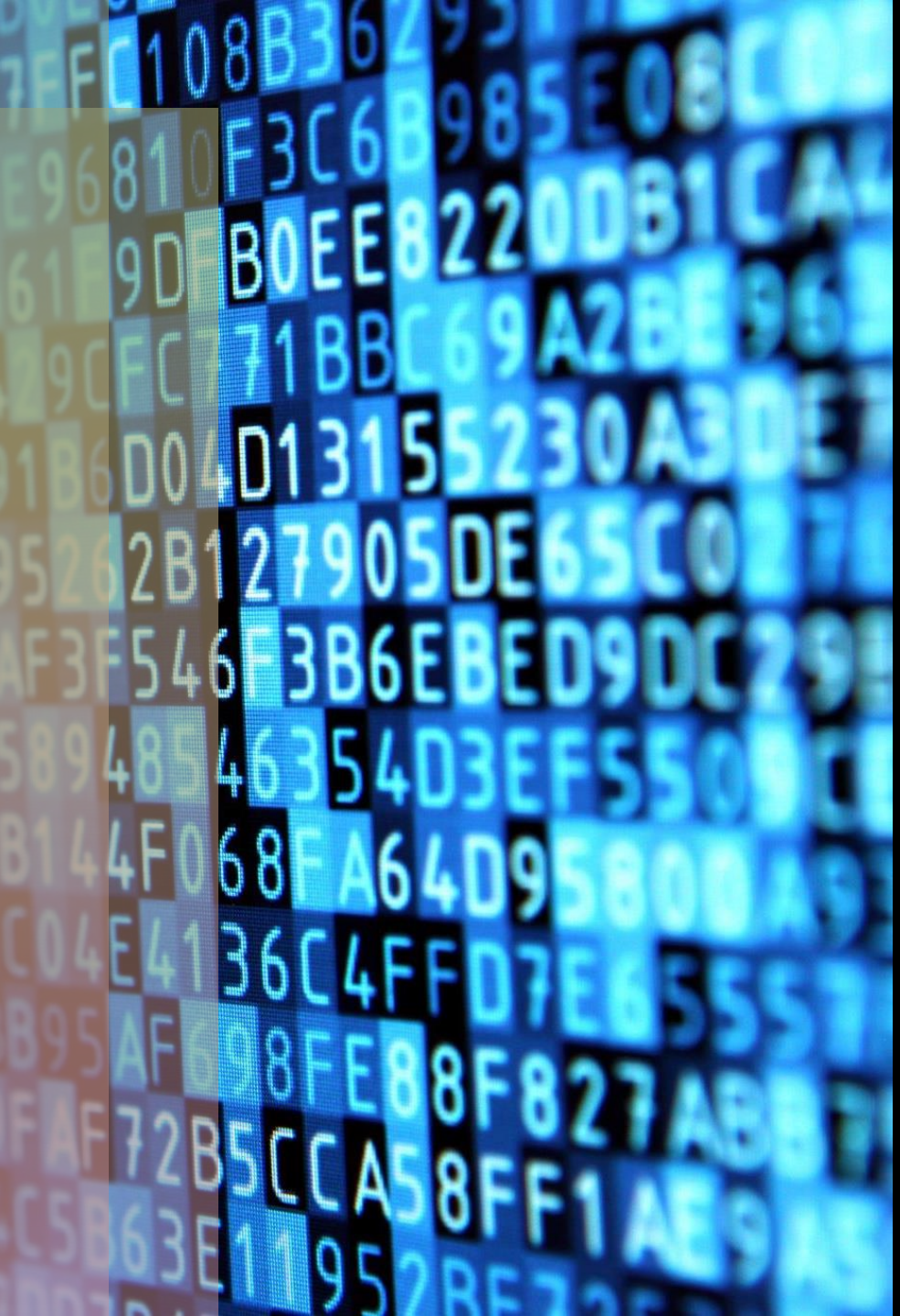
What is programming?

- A program is a sequence of instructions given to the computer
  - All a computer can do is follow sequences of instructions
  - Programming is writing these instructions
- Everything you do on a computer involves running a program:
  - Switching it on, using the operating system, every app you use...
- Every aspect of computing involves programming



Programming Languages

- Programs can be written in different languages
- *Machine language* or *machine code* is the language that the computer uses internally
  - Difficult to write so humans rarely program in this language
- Instead, we use *higher level* languages that are more *human-readable*
  - Use a special program to translate the high-level language into machine language
  - This translation step can be done in different ways:
    - In advance before you need the program - called ***compiling***
    - On demand, only when you run the program – called ***interpreting***
    - Or a combination of the above – ***part-compiled***



Programming Languages

- A few common programming languages:
  - C++, C#, Java, Javascript, Visual Basic, PHP, Python, Actionscript...
  - And there are hundreds more...
  - You will meet some these languages in your various courses

- But how am I going to learn so many languages?
- Surely this is very difficult...?

- Don’t compare this with learning human languages
- Programming languages are mostly very similar to each other

Programming Languages

- **Exercise:** What programming language is this code written in?

```
for (int i=0; i<100; i++)
```

- Note: the word ***code*** here is shorthand for *source code*, which simply means some of the text of the program

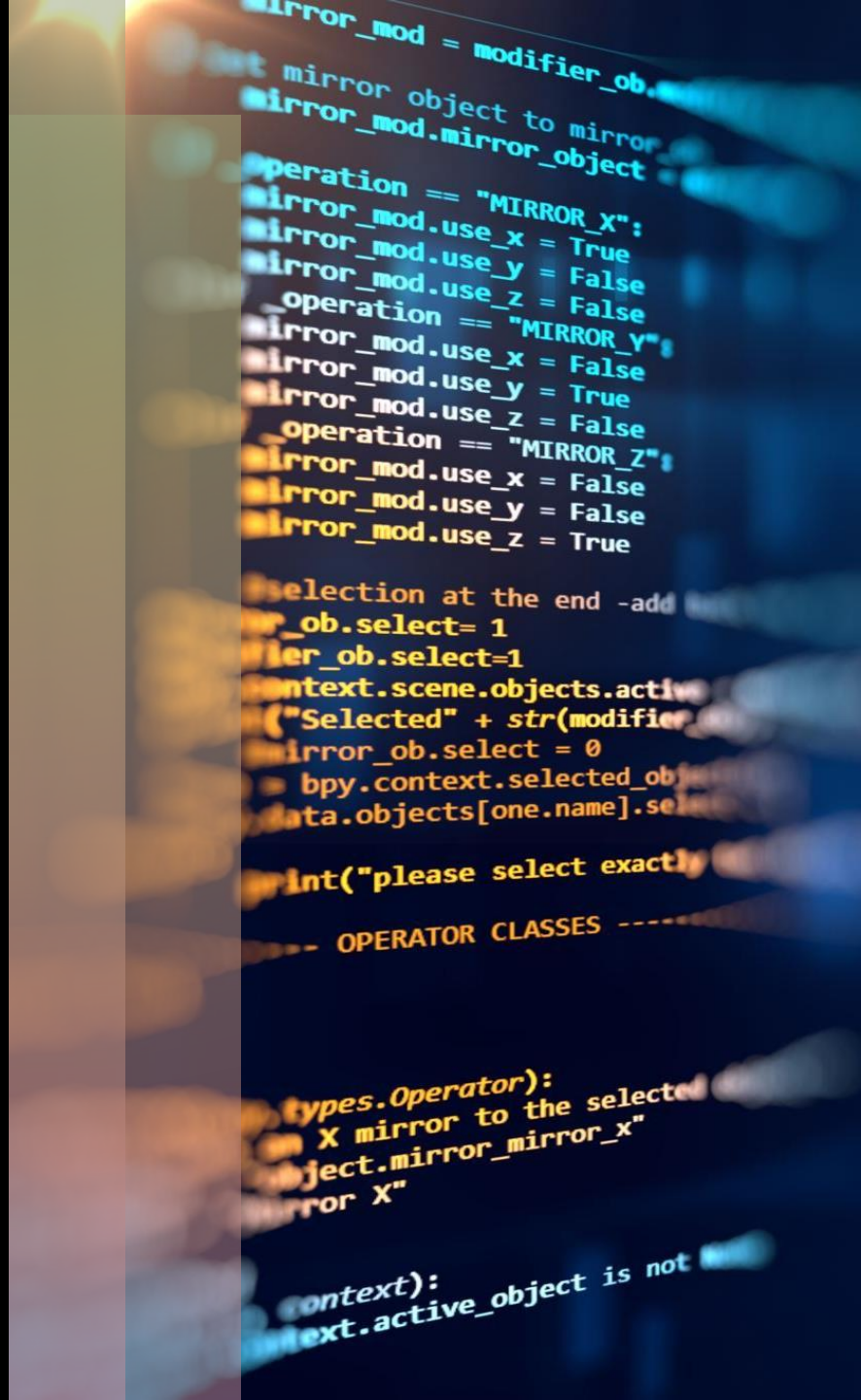
Programming Languages

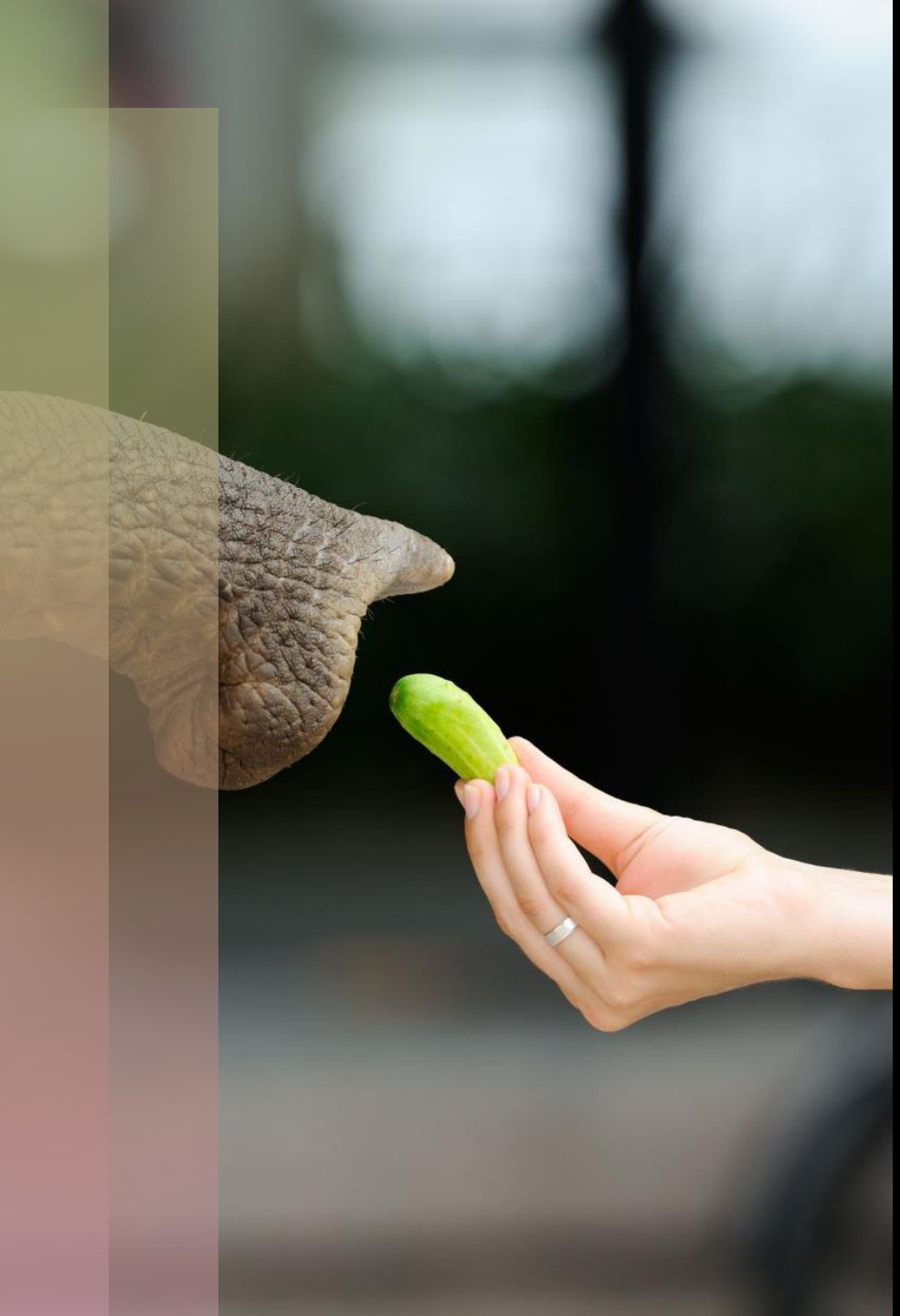
- That example code means the same in:
- C++, C#, Java, Javascript, Actionscript, Swift etc....
- Almost correct for PHP (write $i instead of i) and others...

- Of course, there are some language differences
- But the core concepts remain the same

- That example was a ***for loop***
- Most languages have this feature, often written in a similar way

- So, it is most important to learn core concepts of programming
- Then be ready for differences in detail between languages





Which Language then?

- Teaching Language: Python will be the main language of instruction in this module. Python is renowned for its simplicity and readability, making it an ideal choice for learners.
- Tool of Choice: Anaconda Suite: For the remainder of the year, we will use the Anaconda suite, a powerful platform for data science and scientific computing in Python. It provides an array of tools and libraries to enhance your learning experience.

Computers are Stupid

- A program is a sequence of instructions, so all we need to do to program is to break down our problem into a sequence of steps
  - Often compared to writing a recipe
- The computer will do *exactly* what we tell it to do...
- ...if we are not careful, that might not be what we *want* it to do
- Computers have no knowledge of the world
- Computers don’t *understand* what you're trying to do
  - So, our step-by-step instructions must be small and precise
  - You can’t miss out a step because you assume it is obvious...

Program to Make a Cup of Tea

- **Exercise:** Write down the instructions required to make a cup of tea...

Cup of Tea: Solution

1. Boil Water: - Turn on the kettle. - Wait until the water reaches boiling point (100°C).
2. Prepare the Teapot: - Place a tea bag or tea leaves into the teapot.
3. Pour Hot Water: - Once the water has boiled, carefully pour it into the teapot over the tea bag/leaves. - Leave enough space in the teapot for milk (if desired).
4. Steep Tea: - Cover the teapot with a lid. - Let the tea steep for a few minutes (3-5 minutes is typical but adjust to taste).
5. Prepare a Cup: - While the tea is steeping, get a clean cup or mug.
6. Add Milk and Sugar (Optional): - If you like your tea with milk and/or sugar, add them to your cup.
7. Pour Tea: - After steeping, pour the brewed tea from the teapot into your cup.
8. Enjoy: - Carefully lift the cup and enjoy your freshly brewed tea!
9. Clean Up: - Turn off the kettle. - Dispose of the used tea bag/leaves. - Wash the teapot and cup.
10. End: - Your cup of tea is ready to be enjoyed!

- **What is Abstraction?**
  - Abstraction is the process of simplifying complex systems.
  - It involves breaking down a problem into smaller, manageable parts.
  - Unnecessary details are hidden to focus on essential elements.
- **Why Abstraction Matters**
  - Abstraction enhances code readability and maintainability.
  - It enables scalability and the creation of reusable components.
  - Abstraction is a key practice in software design and development.
- **Finding the Right Level of Abstraction**
  - Striking a balance between too much and too little detail.
  - Ensuring important aspects are not overlooked.
  - Abstraction is a skill that improves with practice.

Cup of Tea: Abstraction

- **Example: Making Tea**
  - Let's connect abstraction to a real-world example: making tea.
  - In our previous tea-making pseudocode, we abstracted the tea-making process.
  - We focused on essential steps while omitting specific details like the tea brand or cup size.
- **Abstraction in Tea-Making**
  - Consider making tea as a complex system.
  - Abstraction simplifies it into manageable steps:
    - Boil water
    - Prepare teapot
    - Pour hot water
    - Steep tea
    - Prepare a cup
    - Add milk/sugar (optional)
    - Pour tea
    - Enjoy
    - Clean up

Abstract Away...

- **Exercise**: Discuss Your Understanding
  - How does abstraction in the tea-making process relate to programming?
  - How does one go about finding the correct level of abstraction?

Python in 2 minutes

Variables

```
x = 5
x = x + 2

z = 5

def func():
    global z
    z = z + 1
```

- Python: declared by assignment
- Bank account metaphor:
  - add or remove
  - quantities
- Global keyword

If Statements

```
if x !=5:
    x = 5
```

- Indentation and colon are used to delimit blocks

```
x == 5
x != 5
x > 5
x <= 5
```

comparison operator

For Loops

```
for i in range(10):
    print(i)

for a in ["apple", "banana", "lemon"]:
    print(a)
```

Functions

$$f(x) = x^2 + 4x + 1$$

```
def f(x):
    return x**2 + 4 * x + 1
```

- Mathematical function:
- Python function:
- Functions may be written by you, or provided by a library someone else wrote. Python has a built-in library of functions.

Setting Up

- Now, let's take a few minutes to set up your PCs and log into your Azure labs. While we hope that Anaconda Studios is readily available, if it isn't, please proceed to download it to ensure you have the necessary tools at your disposal for our upcoming sessions.

Guided Tour

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace HelloApp
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello");
            Console.ReadLine();
        }
    }
}
```

The equivalent of this C# code in python is:

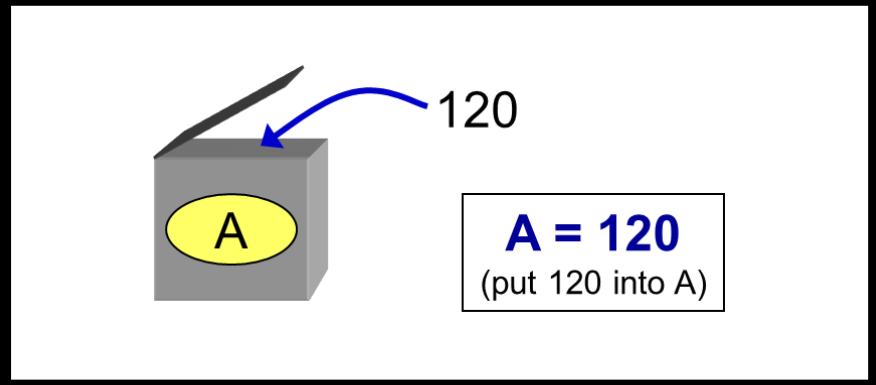
```
print("Hello")
```

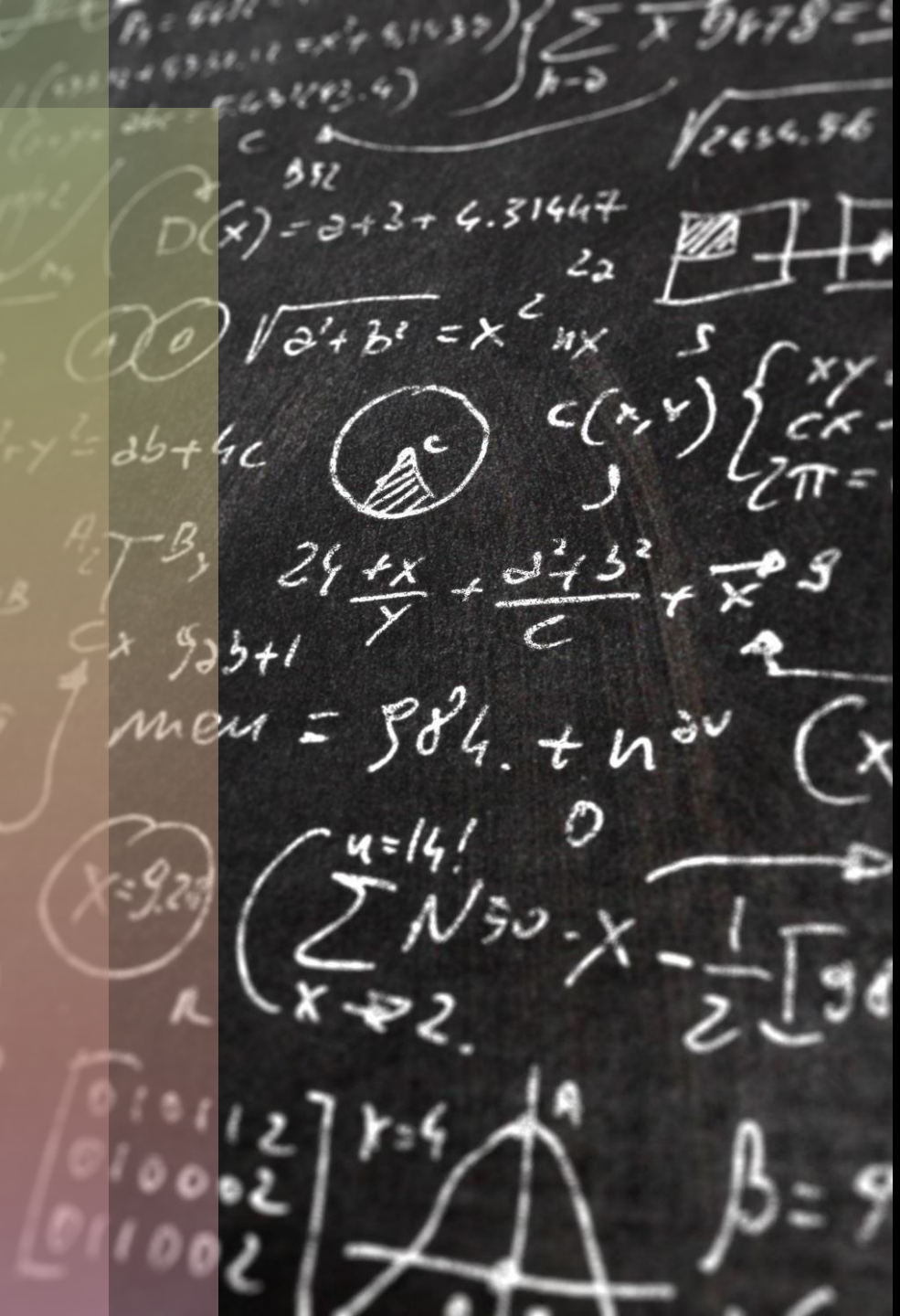
```python
# Importing this module for no reason
import random

# Define the main function
def main():
	# Code blocks defined by indentation not curly braces

	# Display the text on the screen
	print("Hello")

	# Ask for a user input
	user_input = input("What is your name and occupation")

	# Display the user input
	print("You entered: ", user_input )

# Run the main program here
if __name__ == "__main__":
	main() # Call main function here to execute code
```

A Python Program Using Variables

```
A = 120
B = 68
result = A * B
print(result)
```

- This program does a calculation and displays the result
- It uses ***variables*** to do this
- A variable is like a box that contains a value of interest
- Each box has a unique name





Looking at the rest of the code again:

- A = 120
- B = 68
- result = A * B
- pint(result)

The steps are:

- Declare an integer called A and initialise its value to 120
- Declare an integer called B and initialise its value to 68
- Declare an integer called result and set its value to A * B
  - * means multiply, so this sets variable result to 120 x 68 = 8160
- Then display the contents of the variable result
  - Don’t use quotes around result, or it will display the text “result” rather than the value 8160 held in the variable result

Coding Practices

- **Dynamic Typing in Python**
  - In Python, variables are dynamically typed, meaning you can reassign them to different types during program execution.
  - Unlike some languages, you don't need to explicitly declare a type when defining a variable
- **Flexibility and Ambiguity**
  - This flexibility allows you to work with various data types easily
  - However, it can also lead to ambiguity if variable types change unexpectedly
- **Documentation is Key**
  - To maintain code clarity and prevent errors:
    - Explicitly explain the intended usage of each variable in your code
    - Document variable types and their expected values
    - Use meaningful variable names to indicate their purpose

```
count = 5       # Initially an integer
count = “five” # Now is a string
```

  - In this example, documenting the change in variable type is crucial for understanding the code's behaviour
- **Best Practices**
  - Embrace dynamic typing but use it responsibly
  - Consistently document variables to avoid confusion

Survival

- Check for errors in console
- Use the reference:
  - Reference (processing.org)
- Look at examples (File > Examples)
- Python:
  - Python Tutorial (w3schools.com)
  - Python Cheat Sheet (2023) - GeeksforGeeks

What is Software Development?

- Requirements and planning
- Data modelling
- Algorithm design
- Implementation
- Testing
- Documentation
- Deployment
- Maintenance

Compose a 500-word report comparing and contrasting the Waterfall and Spiral Model Software Development Life Cycles (SDLCs) while keeping their respective advantages and disadvantages in mind. Ensure that you use include at least two academic references and adhere to either the Harvard or APA7 citation style