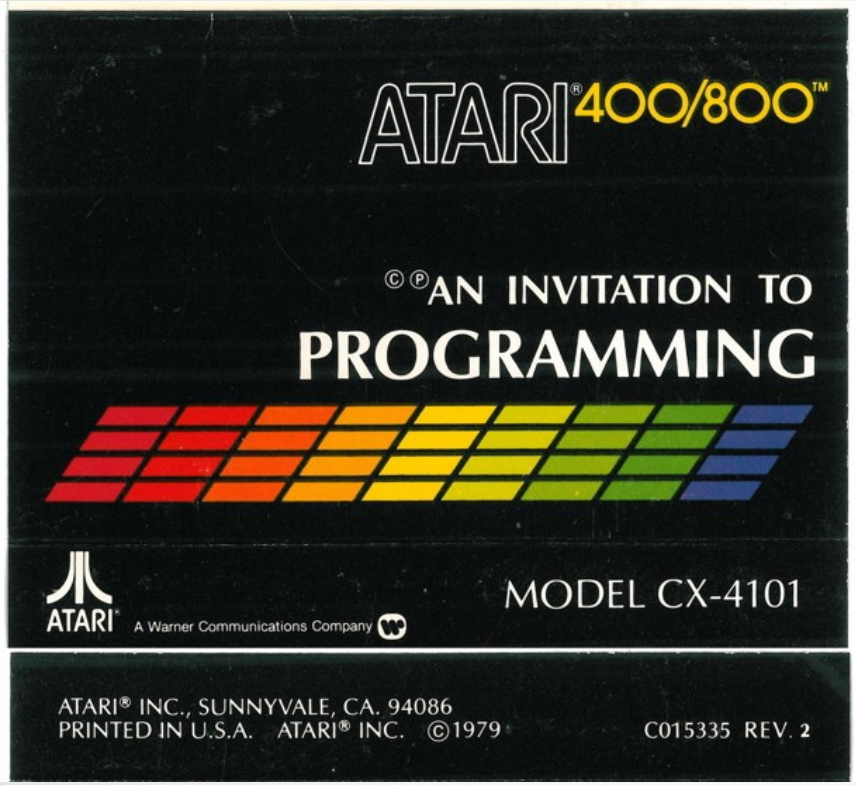
Module 1 : Introduction to OR, Data Science and Programming

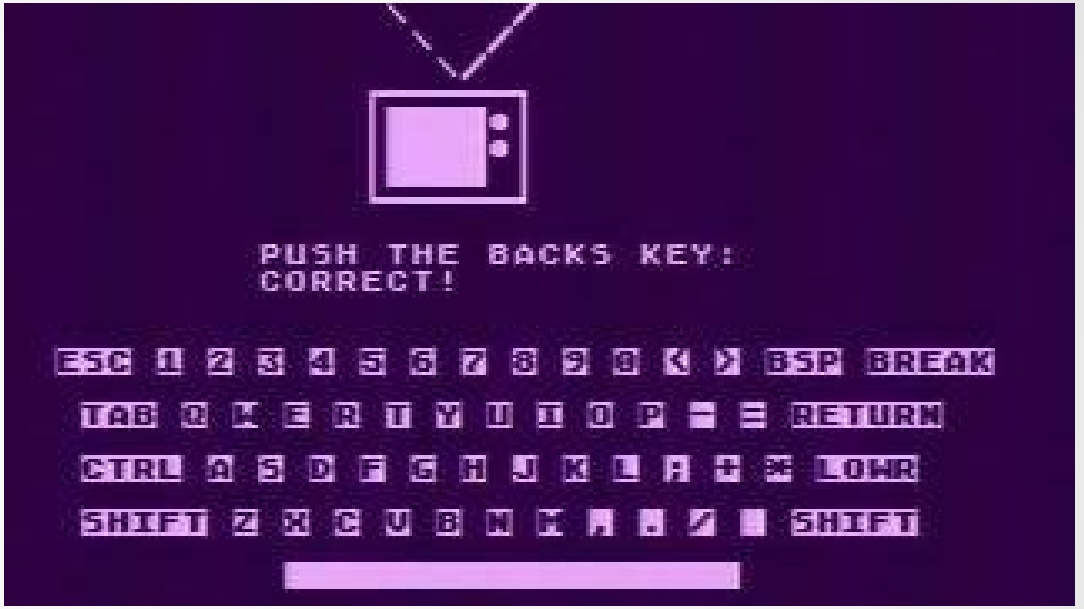
Session 1B : Principles of Programming

Dr Daniel Chalk

"Hello World"

With thanks to my first programming teacher when I was four years old...





What is Programming?

Computers carry out *instructions* given to them by the *user*

Instructions are a series of steps that the computer works its way through

Programming languages allow us to communicate with the computer to provide it with instructions

We can write *computer programs* that bundle up these instructions, to be executed when the program is *run*

Computer Architecture

One or more *input* devices (keyboards / mice etc) are used by the user to issue instructions.

A *Central Processing Unit (CPU)* interprets the instructions and executes them

Memory is used to store information that may be required later, either in the short-term (RAM - *Random Access Memory)* or after the computer has been shut down (*Storage – e.g. hard drive)*

One or more *output* devices (monitors / speakers etc) are used to communicate the outputs of instructions to the user

Low vs High Level Languages

Low Level Programming Languages

Assembly Language is unique to the architecture of the computer we're programming. It requires us to give low-level instructions that deal directly with processor-level instructions.

High Level Programming Languages

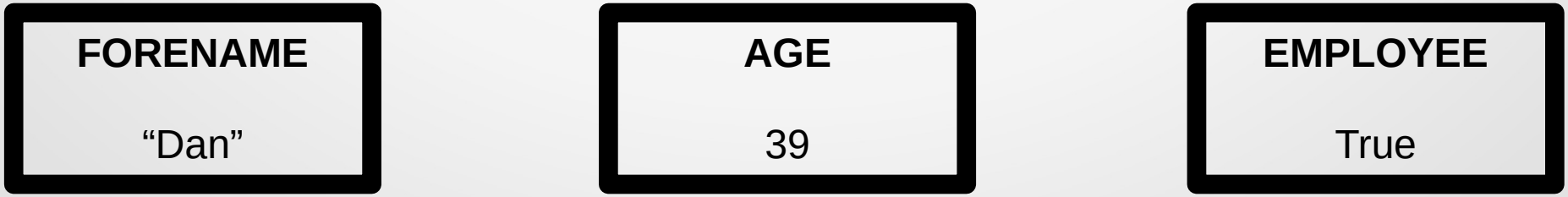
These languages *strongly abstract* from the details of the computer. They use natural language and are much less complicated to use. Examples : BASIC, C, C++, Java, Python, and many more...

Variables

Often, when programming, we need to store pieces of information for later use.

We store this information in the computer's *memory*.

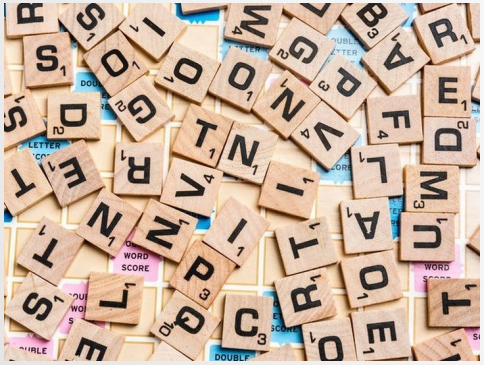
We organise this information into "boxes" called *variables.* Each variable has a *name* and a *value*.



Variable Types

Information stored in variables come in all shapes and flavours, known as *types :*

Numbers
Integers (whole numbers)
Floating Point Numbers (with decimal places)



Text
Characters (individual letters, numbers and symbols)
Strings (sequences of letters, numbers and symbols)

Boolean
Data type that indicates if something is *True* or *False*

Collections
Lists (ordered sequence of things)
Sets (unordered sequence of unique things)
Dictionaries (unordered collection of pairs of “index” and “value”)
Tuples (like a list, but can never change)

Example

Let’s say we want to build a simple calculator that just adds up two numbers. We would need to tell the computer to :

1. Ask the user for the first number
2. Store the first number in memory as a variable
3. Ask the user for the second number
4. Store the second number in memory as a variable
5. Perform an *addition* operation on the two variable values
6. Store the result of the addition in memory as a variable
7. Display the result variable to the user.

Conditional Logic

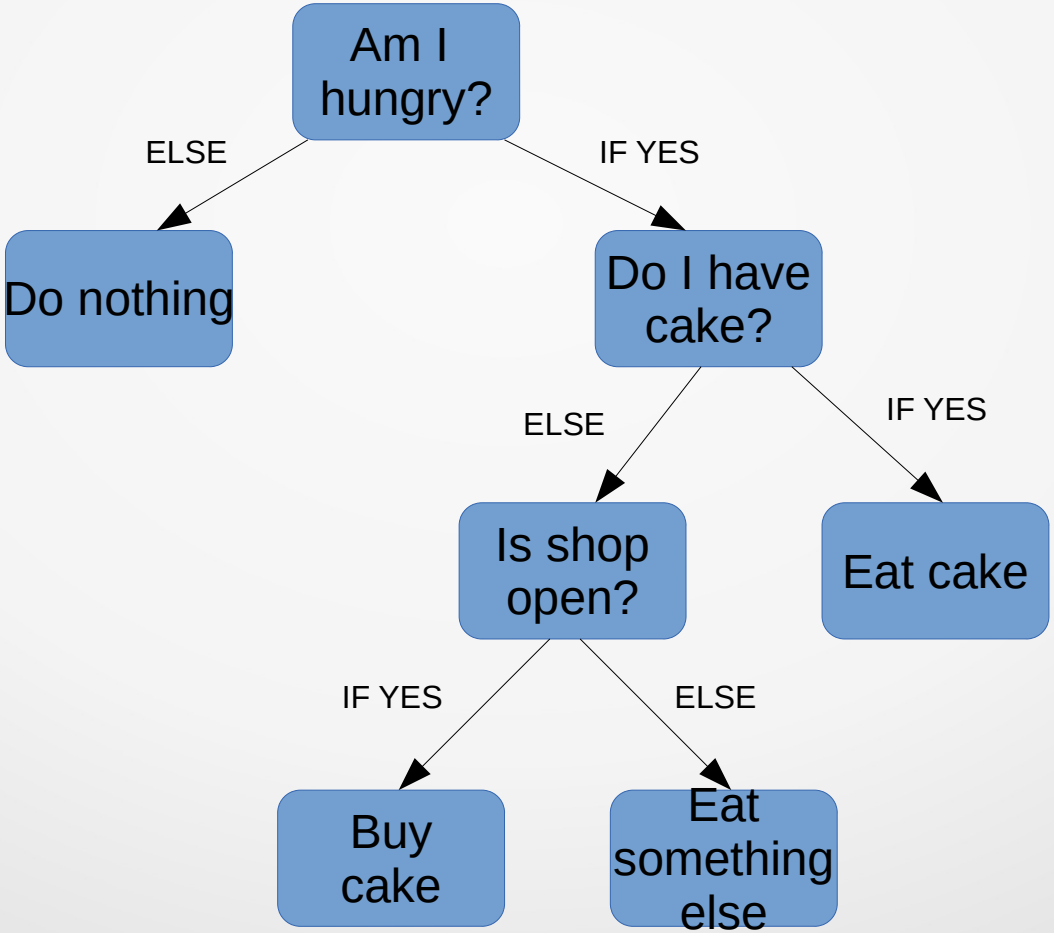
It is rare that our programs are completely linear. Often, we need to change what happens depending on whether or not something is true.

This is known as *Conditional Logic*.

Conditional Logic allows us to tell the computer “if this is true, then do this”

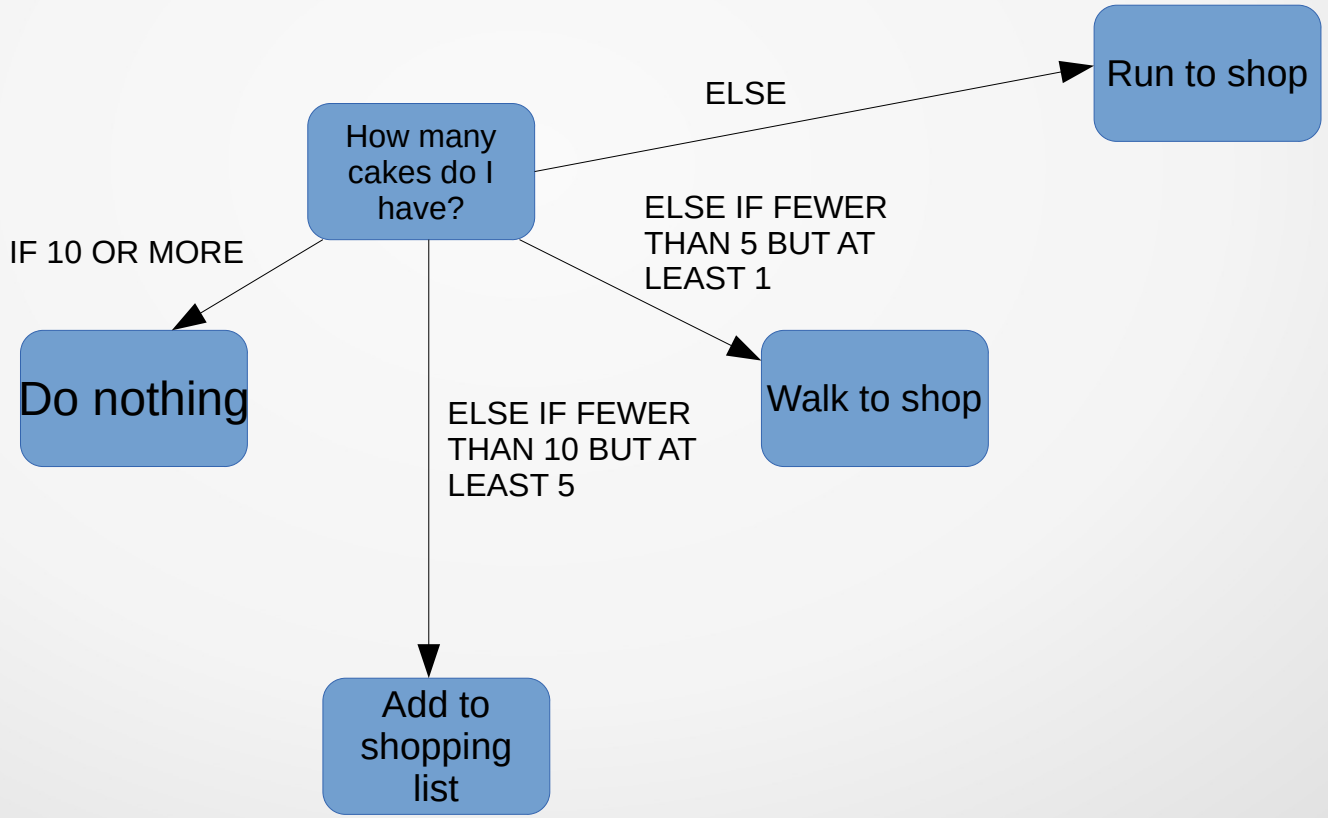
Conditional Logic

A good way to think about Conditional Logic is as a *Decision Tree* :



Conditional Logic

Sometimes we have multiple initial checks to make on the original condition :



Loops

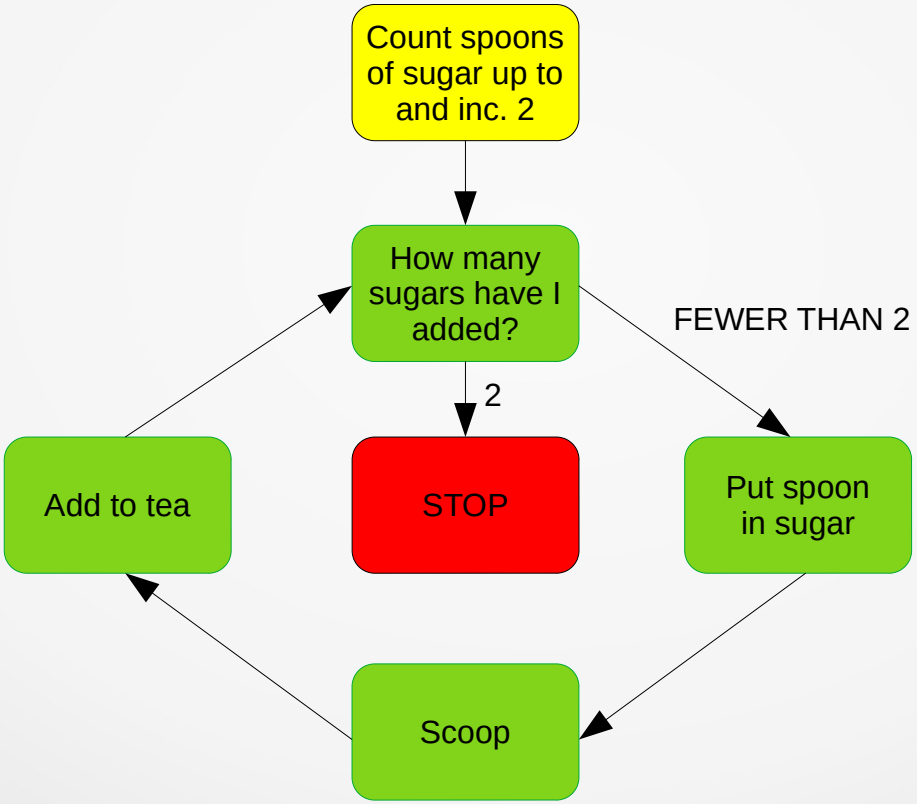
Often, we need the computer to perform the same operation a number of times. It might be that :

- we need to do the same thing for a given number of iterations
- we need to continue to do something whilst a certain condition is true

We can use two different types of loop to achieve the above – *for* loops and *while* loops

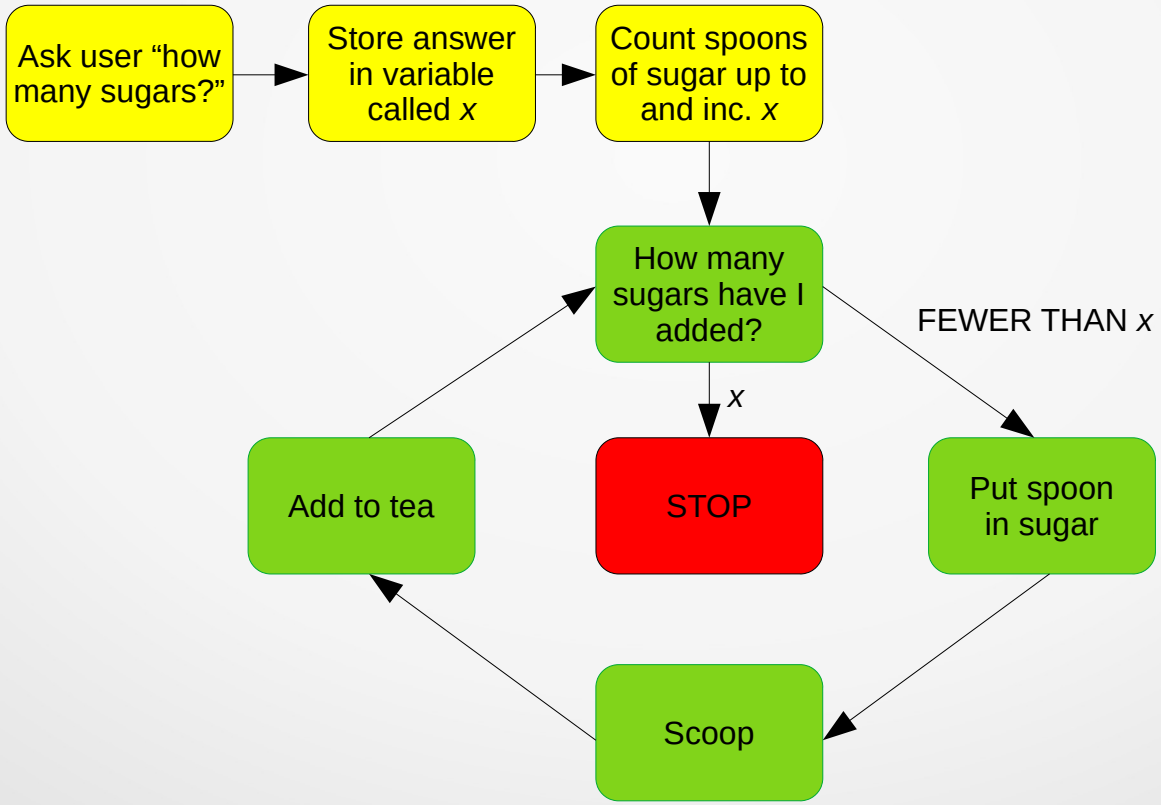
For Loops

Let’s say we want the computer to add two sugars to our cup of tea :



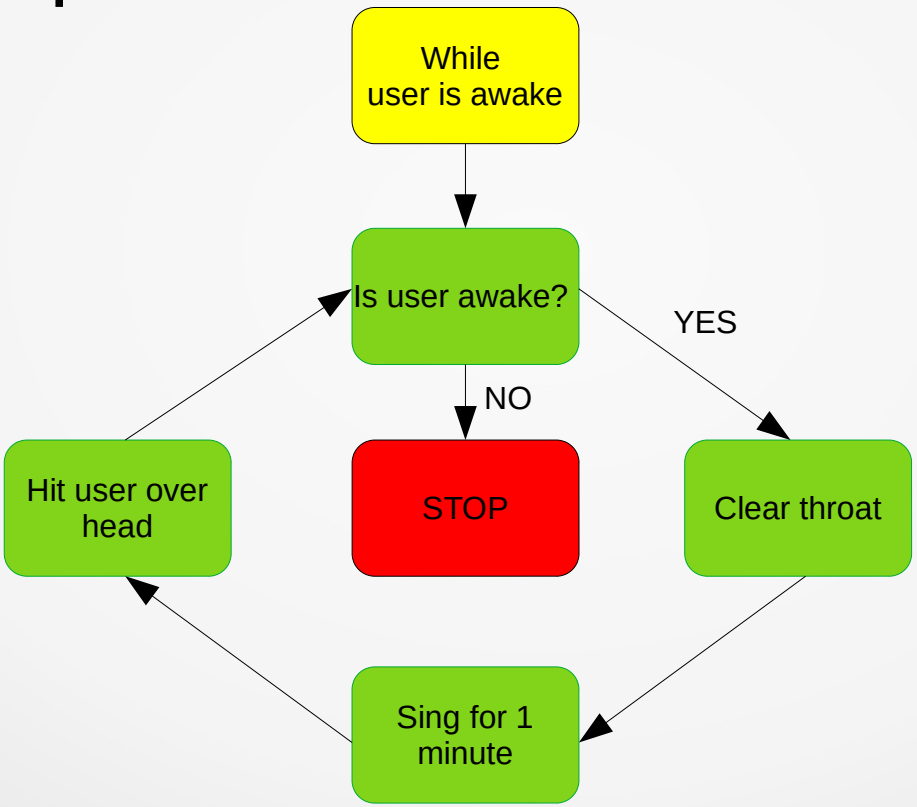
For Loops

Of course, we could easily adapt this to ask the user how many spoonfuls of sugar they want :



While Loops

Let’s say we want the computer to sing to us until we fall asleep :



Exercise 1

Chalk Showers and Baths Inc (a division of Chalk Technologies – “building a brighter future”) has approached you to ask you to develop the software for a new automated shower system – the **C**halk **A**utomated **Sh**ower with **G**enius **R**esponses **A**nd **B**anter (CASH-GRAB) System.

In the new system, the user steps into the shower, and the shower asks them for their name. It then greets them with their name, saying “good morning”, “good afternoon” etc depending on the time of day. It then waits for the user to say “begin”, at which point the shower checks whether the user has a previous perfect temperature stored. If they do, it turns on at that temperature, and will remain at that temperature until the user says “too warm” or “too cold”.

If the user does not have a perfect temperature stored, the shower turns on at 30 Degrees C.
The shower continues to raise the water temperature by 2 degrees C every 5 seconds. If the user says “too warm”, the shower drops the temperature by 1 degree C every 5 seconds. If the user says “too cold”, the shower again starts raising the temperature by 2 degrees C every five seconds. If the user says “perfect”, then the shower holds the current temperature, and records it in a variable called *perfect_temp*, alongside their name in a variable called *user_name*, and stores both of them in a list called *user_preferences*. If the user had a previous perfect temperature, their new temperature is overwritten. Also, their calls of “too warm”, “too cold” and “perfect” will work as described above.

The shower continues to run until the user says “end shower”, at which point the shower says “Goodbye” followed by their name. In your groups, you have 45 minutes (+10 min break) to draw up an outline of the system. You should :
- Draw a diagram outlining how the system will work, including any conditional logic, loops and variable storage.
- Write a list of the variables that will be used in the system, along with the type of each variable, and a short (single sentence) description of what the variable is storing.

When we return, I’ll ask a few teams to share what they’ve done.

Functions

Often, when we write things we want a computer to do, we want to do them more than once, maybe at different points in our computer program. Or even in a different program altogether.

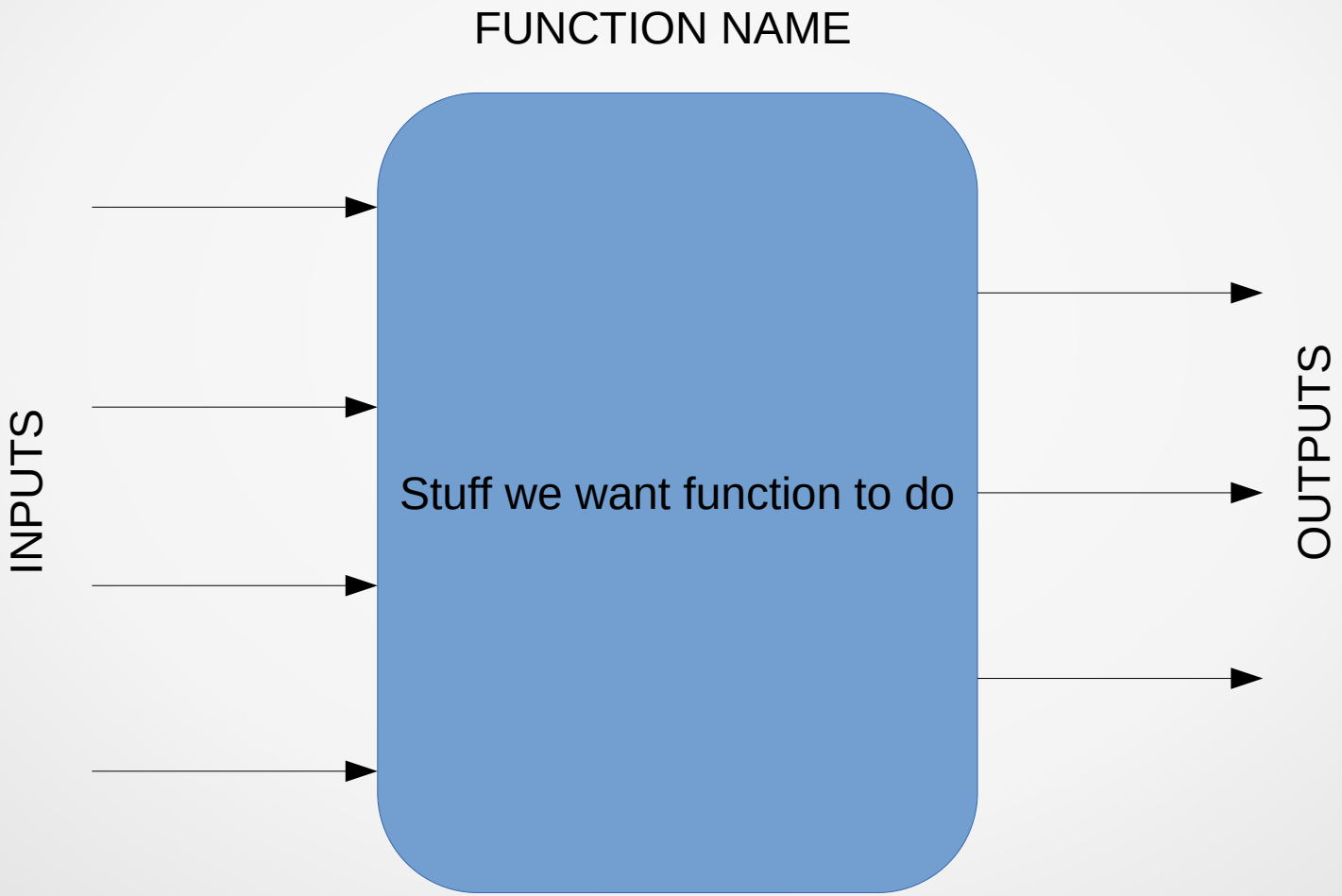
To prevent us having to write out the same sets of instructions time and again, we can wrap a set of instructions up in a *function*.

A function can take *inputs*, do something with them, and then return an *output*. The things it does are just sets of instructions.

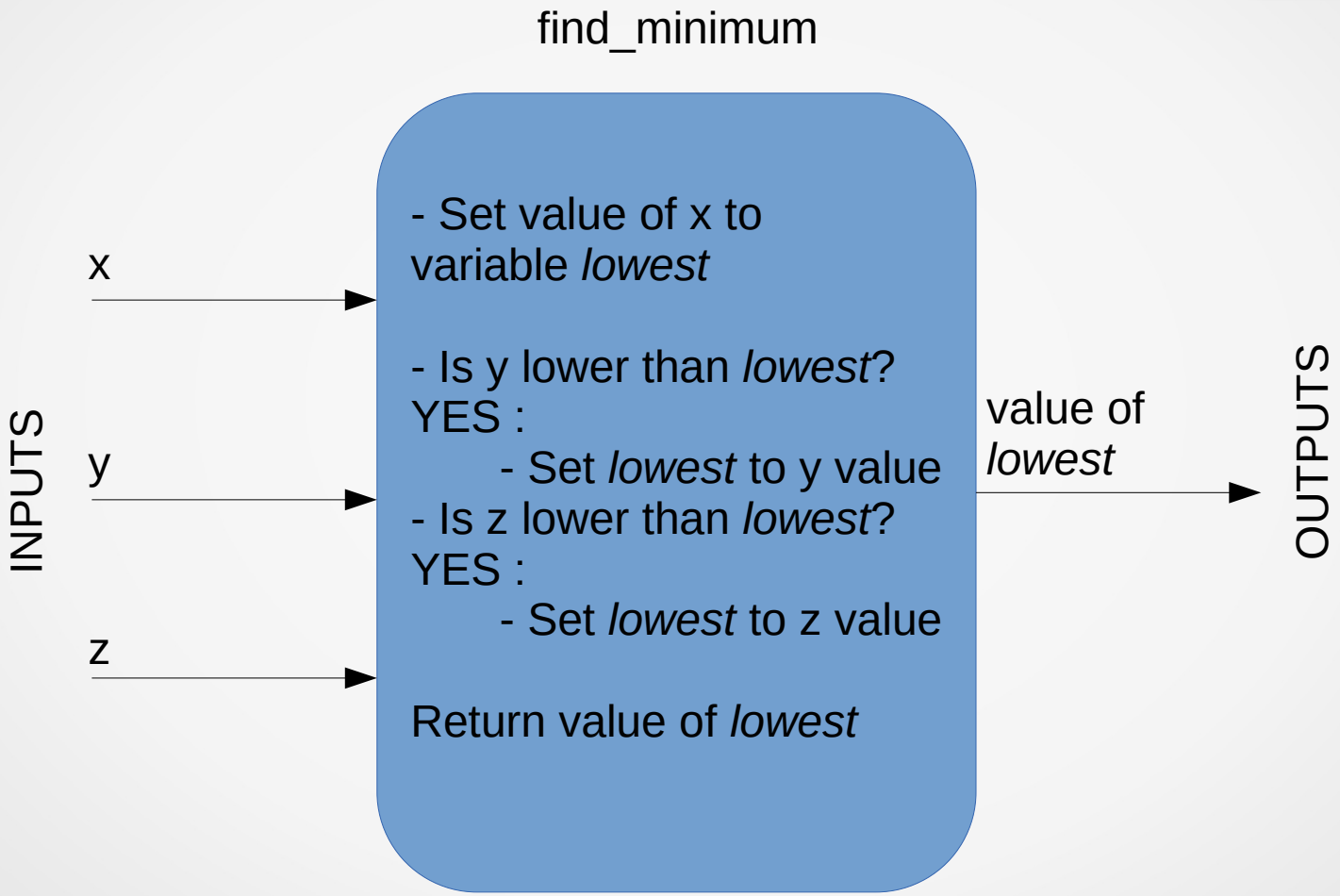
A function is defined by giving it a *name*, the *list of inputs* it should expect (if any), the things we want it to do, and the *outputs* it should *return* (if any).

The function doesn’t do anything until it is called, however.

Functions



Functions



Example : find_minimum(3,7,2) would return the value 2

Object Oriented Programming

So far, we've talked about programming using the *procedural programming* methodology. This is where we write a program purely as a list of instructions to be carried out.

But an important concept in modern programming is *Object Oriented Programming (OOP).*

In OOP, everything is centred around *objects* that have their own attributes and methods (functions). Objects are created as *instances* of *classes*, which are essentially generalised blueprints for how certain types of objects should work.

Let's look at an example.

Object Oriented Programming

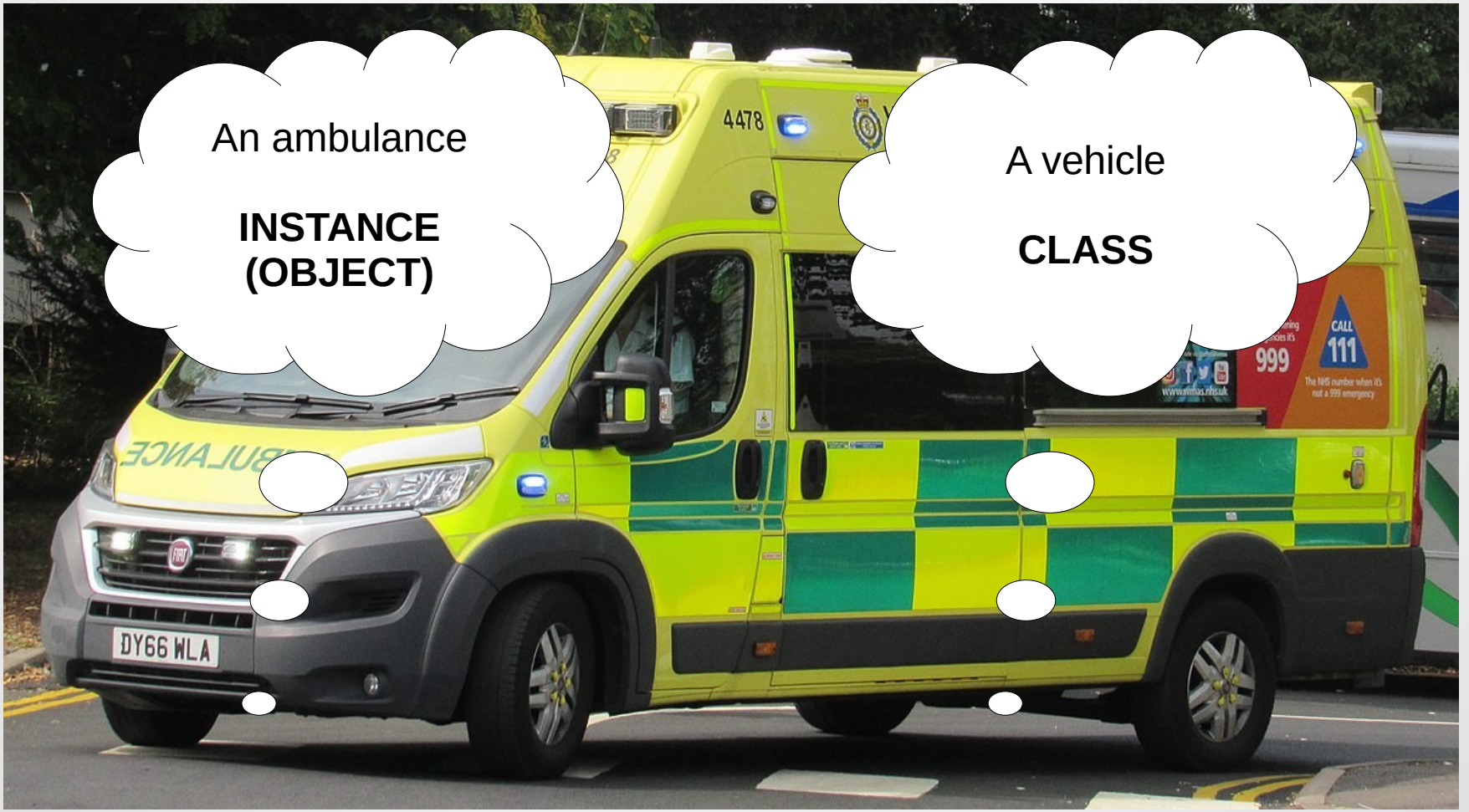
What is this?

Object Oriented Programming

What is this?

Object Oriented Programming

What is this?



OOP Example 1

CLASS : Vehicle

Attributes
common_name : string
number_of_wheels : integer
capacity : integer

Methods
drive(speed)

The Class contains the generalised description / blueprint

Objects are *instances* of a Class that *inherit* attributes and methods from the parent Class

Inheritance

OBJECT
common_name = “Ambulance”
number_of_wheels = 4
capacity = 3

drive(speed)
fuel_refill(level)
activate_siren()
deactivate_siren()
load_patient()
unload_patient()

OBJECT
common_name = “Bicycle”
number_of_wheels = 2
capacity = 1

drive(speed)
repair_puncture()

OBJECT
common_name = “Car”
number_of_wheels = 4
capacity = 5

drive(speed)
fuel_refill(level)

OOP Example 2

CLASS : Patient

Attributes
name : string
patient_id : integer
age : integer

Methods
attend_ed()
receive_treatment()

The Class contains the generalised description / blueprint

Objects are *instances* of a Class that *inherit* attributes and methods from the parent Class

Inheritance

OBJECT
name = “Bob Jones”
patient_id = 23195392
age = 64

attend_ed()
receive_treatment()

OBJECT
name = “Mary Smith”
patient_id = 64582990
age = 51

attend_ed()
receive_treatment()

OBJECT
name = “Janet Small”
patient_id = 45189927
Age = 37

attend_ed()
receive_treatment()

Constructors

A *constructor* defines what happens when an object is *instantiated* from a class (an instance object is created from a class).

The constructor essentially “constructs” the object, and specifies the (initial) values for the attributes and methods of the object.

The constructor is a method within the Class.

Exercise 2 – OOPs, Dan Did it Again

You have been asked to ensure that your code for the CASH-GRAB Shower system is Objected Oriented.

In your groups, you now have 40 minutes to draw up some diagrams showing the Classes you think you would include, along with the attributes (including their type) and methods for each. You should include a short single sentence description of what each attribute and method does, prefixed by # (all will become clear in next week's session...)

Here's an example of the format to use for each Class :

CLASS : Vehicle

Attributes
common_name : string
number_of_wheels : integer
capacity : integer

Methods
drive(speed)

common_name stores a string representing the name of the vehicle type (such as "car", "motorcycle" etc)

number_of_wheels stores an integer that represents the number of wheels on vehicle

capacity stores an integer that represents the passenger carrying capacity of the vehicle

the drive() method tells the vehicle to begin driving at the speed given as an input to the method