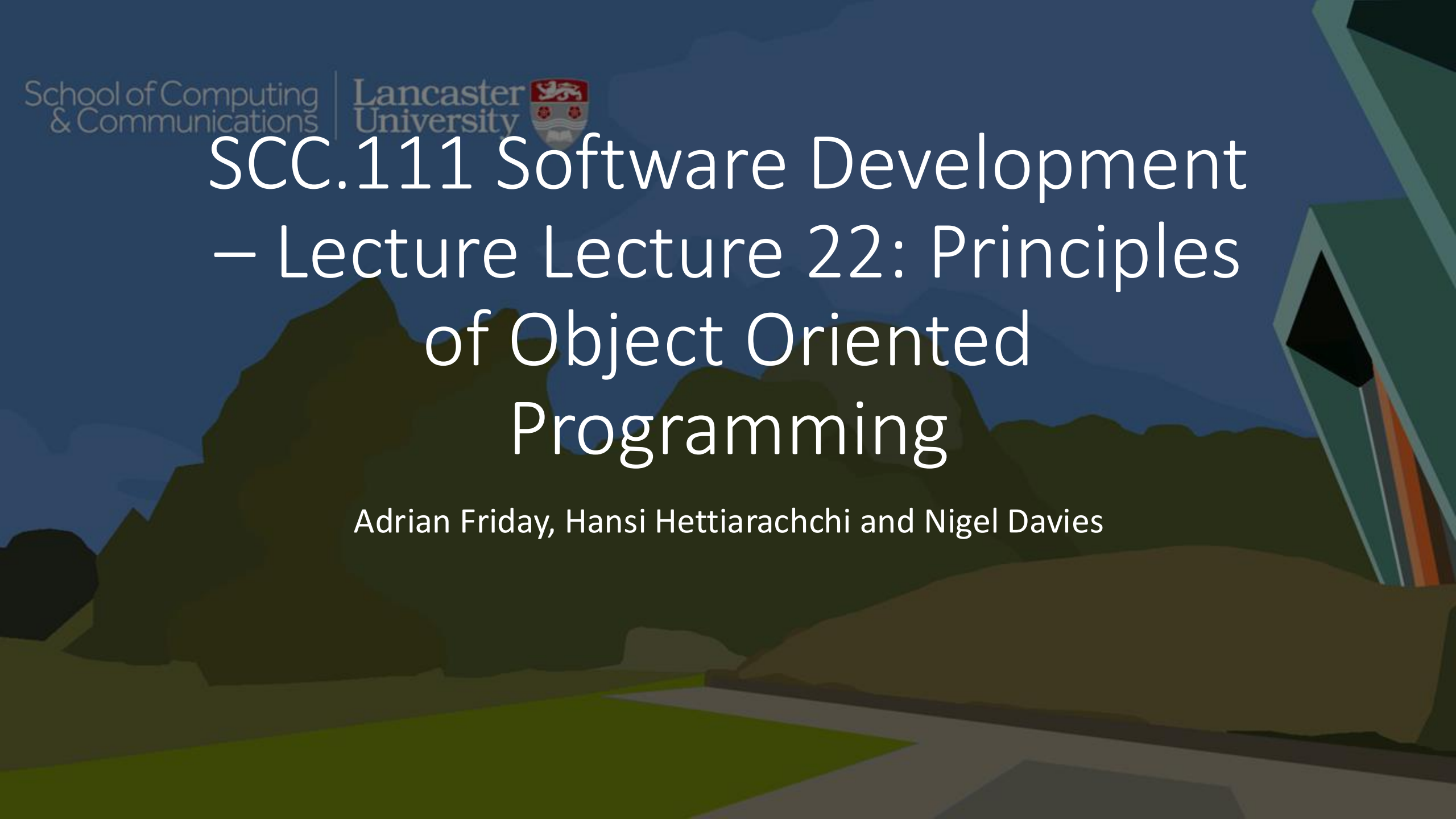



# SCC.111 Software Development – Lecture Lecture 22: Principles of Object Oriented Programming

Adrian Friday, Hansi Hettiarachchi and Nigel Davies

# Agenda

**Introduce a new programming language paradigm – Object Oriented Programming**

- Motivation: How to write scalable code.
- The importance of a well-defined API (Application Programmable Interface)
- The anatomy of an object

- Introductory C++ syntax
- A worked example

# Applicability

- **Let's briefly discuss the applicability of this lecture topic…**

- In our first lecture, we identified the most popular programming languages (currently):

  - C / C++
  - Java / C#
  - Python
  - JavaScript

- **Do you know which of these languages are object oriented?**

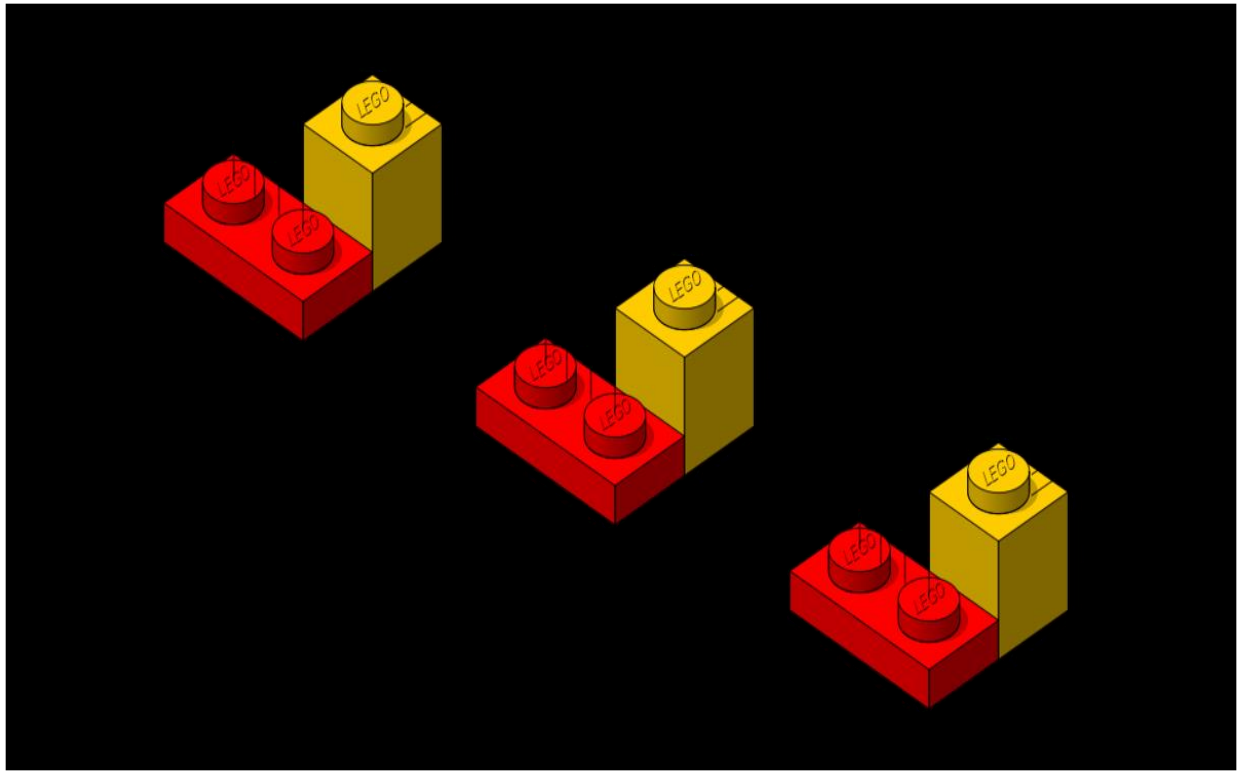
- **C++ is a backward compatible, evolution of C. The biggest difference between C and C++ is the introduction of Object Oriented principles into the language.**

# How to write scalable code

- **Rule #1: Write modular code**

# Modularity

Lancaster University

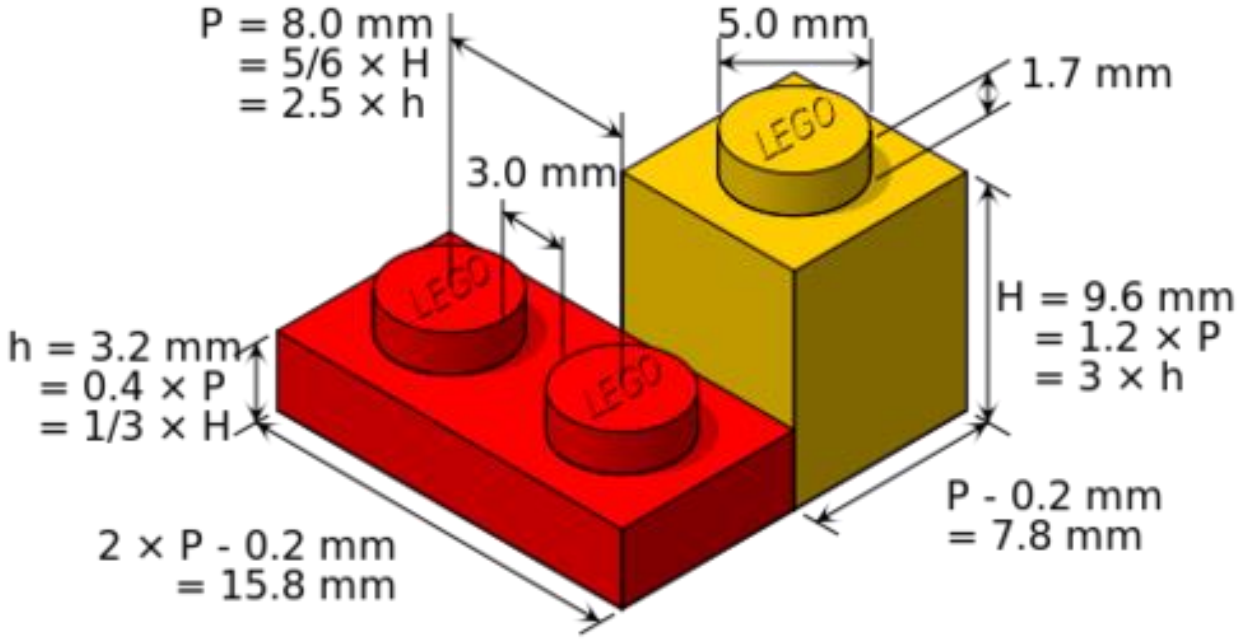


# Modularity: Lego examples

Lancaster University

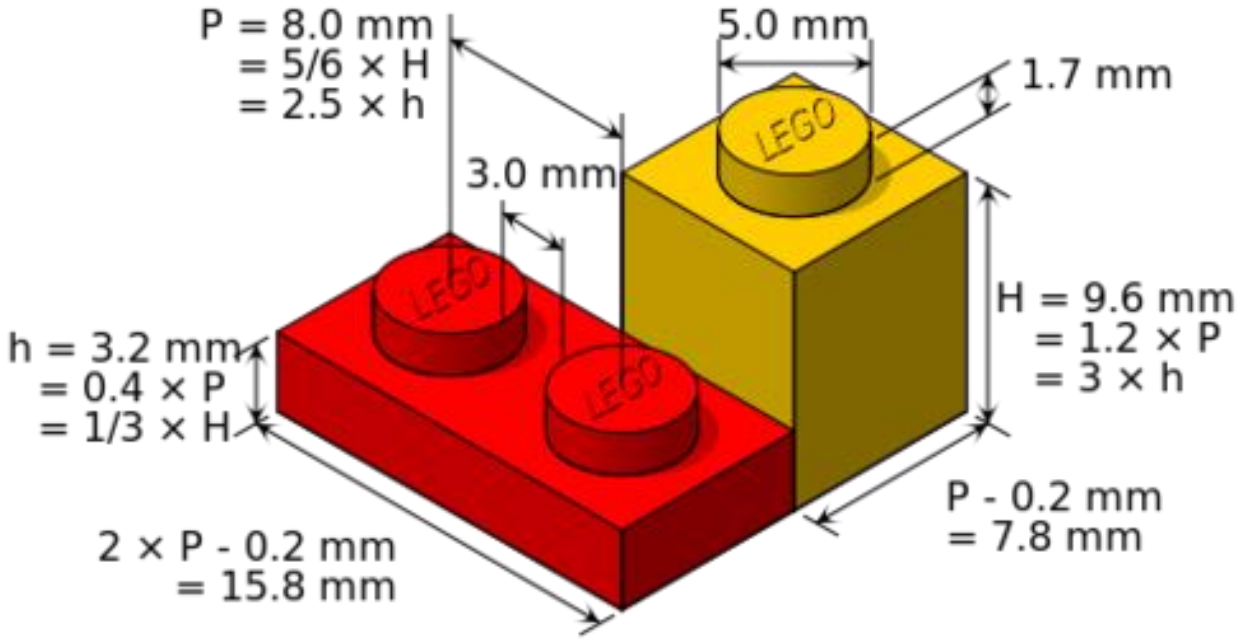
# Modularity: Standards

- Modularity relies on **standards**. The interface between a module and the world must be well defined.



# Modularity: Software

- **In software, those standards are called an Application Programmable Interface (API).**



# Modularity: Procedural Programming

Lancaster University

- Programming languages can be classified based on the core paradigm on which they are based
- C is an example of a **procedural** programming language:
  - There is a well-defined start point to a program (the main function)
  - Code is broken down into manageable chunks (functions)
  - These functions are called by one another
  - Programs hold information in the form of variables
  - Variables are passed as parameters to those functions, or can be global

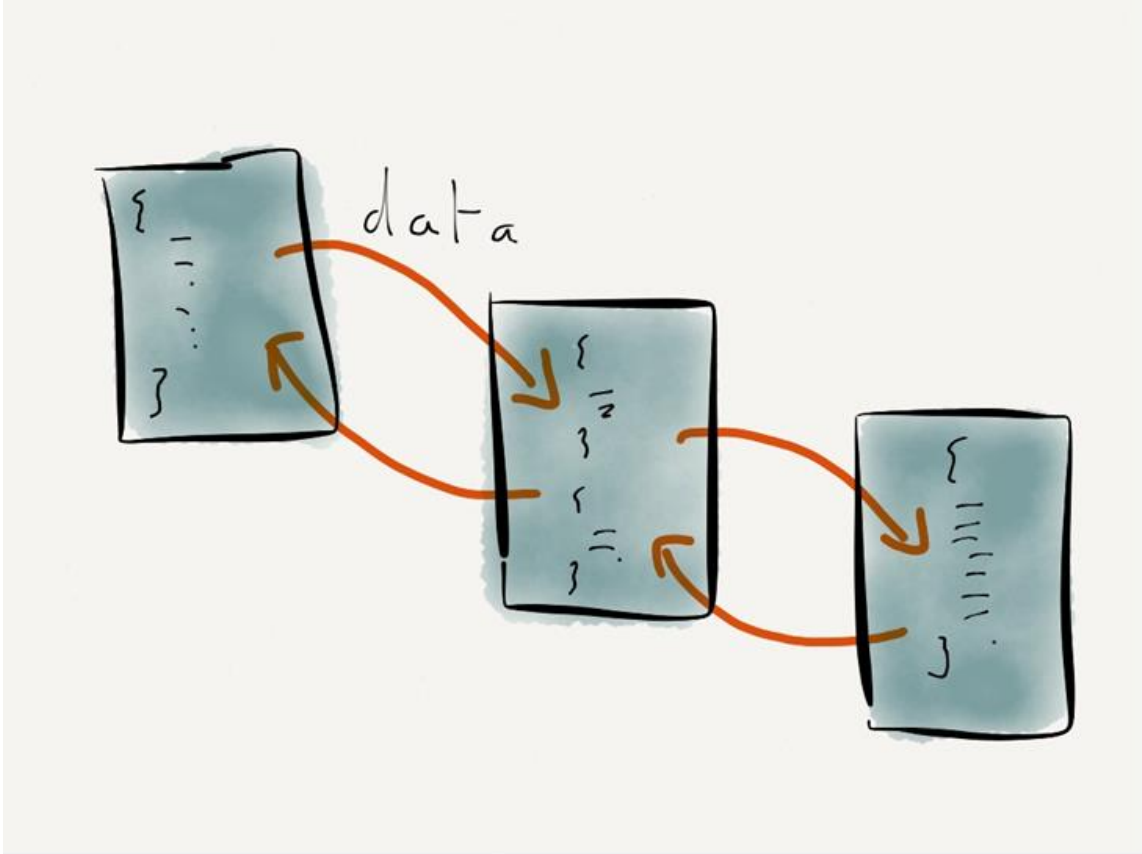
- Procedural programming languages are structured around the **code**

# Procedural Programming: code and data

Lancaster University

- Procedural programming languages give little thought to the location of the data!
  - Computer programs are made up of code and data
  - Code modularity occurs in functions and libraries (collection of functions)
  - Data modularity occurs only in data structures (or not at all!)

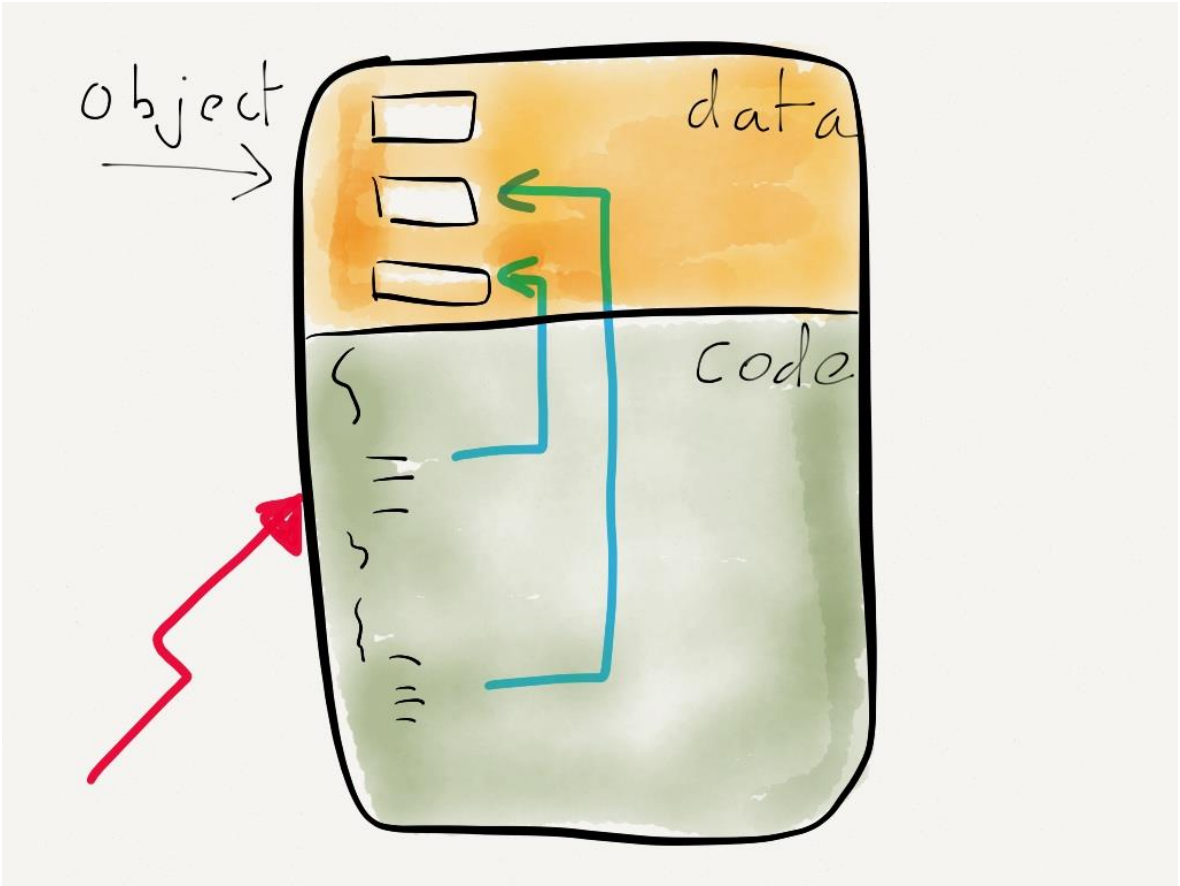
- **Procedural languages treat code and data as separate concepts**



# Object Oriented Programming

Lancaster University

- Object Oriented (OO) programming languages **combine code and data**
- Objects are the "lego bricks" of the software world
- They group similar data and functions (something known as tight cohesion)
- **They isolate parts of your program from the rest… therefore reducing the complexity of your software**



# Object Oriented Programming 2

Lancaster University

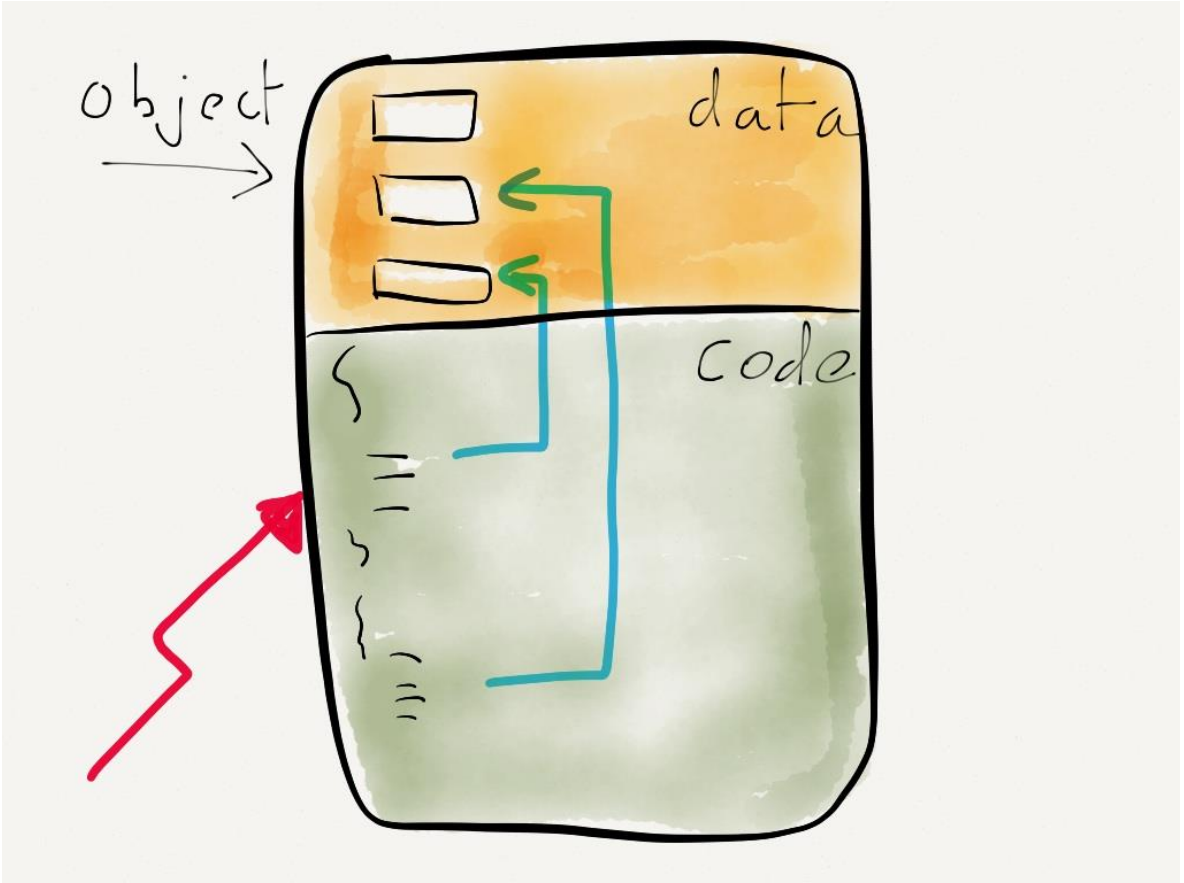
- Objects provide **encapsulation**
  - Data and code cannot be separated
  - Data is defined through **attributes**
  - Behaviour is defined through **methods**

- Objects protect their inner workings through an API
  - Interactions with objects occur through its methods
  - Programmer only exposes what they choose to
  - This provides very tight controls over how it can be used
  - **This allows programmers to provide sealed boxes of code in order to promote simplicity.**

# Design exercise

Lancaster University

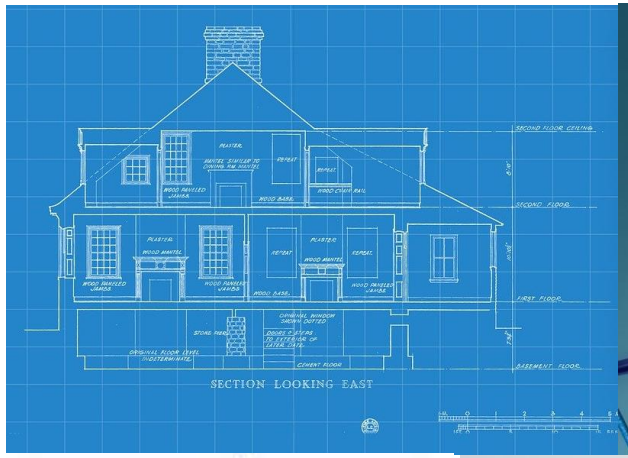
- Think of a real-world object
  - Define its attributes
  - Define its methods
- Write them down on a diagram that looks like this:



# Classes

- In OO languages, objects are defined by **classes**

- A class is a specification of an object someone might build
  - Like a blueprint
  - Like DNA
  - Like a cookie cutter
  - Like a jelly mould




# Classes

Lancaster University

From the options below, which is most likely a **class** in an object-oriented program?

A. John Doe
B. Bank Account
C. Current Balance
D. Java

Join at menti.com | use code 6148 9618

# Classes: writing classes

Lancaster University

- By convention, C++ classes are named in **capitalised camel case**
  - E.g. HelloWorld, UniversityLecturer, FluffyBunny, etc.

- The **class** keyword defines that everything in the following code block is part of the given class… much like a **struct**…

```
class Car
{

};
```

# Classes: attributes

Lancaster University

- An object's attributes are implemented through creating **variables**
  - These variables are declared **inside** a class, but **outside** any methods

- Attributes provide a **new level of scope**
  - more controlled than a global variables.
  - less specific than a local variable declared in a function.

```
class Car
{
    int milesDriven = 0;
    char *colour;
};
```

# Classes: methods

Lancaster University

- An object's behaviour is implemented through **methods**
- Methods are simply functions that are **declared within a class.**
- By convention, methods are also named in **camel case**

```
class Car
{
    int milesDriven = 0;
    char *colour;

public:
    void drive(int miles);
    void respray(char *c);
    void show();
};
```

# Classes: methods

Lancaster University

- C++ Method implementation is *normally* defined **outside** the class.
- The class name provides a new **scope** for your methods as well as your variables…
- **We use new notation to identify this new scope – the double colon notation…**

```
void Car::drive(int miles)
{
    milesDriven = milesDriven + miles;
}

void Car::respray(char *c)
{
    colour = (char *)c;
}

void Car::show()
{
    printf("I'm a %s car, and I've driven %d
miles.\n", colour, milesDriven);
}
```

# Creating objects from classes

Lancaster University

- Remember classes are **types**
  - You use them by creating variables of that type.
  - **By yourself, or by other programmers you give your class file to!**

- A realised class is called an **object instance** and is treated just like any other variable…

```
Car amysCooper;
```

# Using methods and attributes

Lancaster University

- We use the "." operator, just like we do on structs!
- The **left hand** side should be the variable you created.
- The **right hand** side is the name of the method / variable

```
Car amysCooper;

amysCooper.respray((char *)"White");
amysCooper.drive(16);
amysCooper.show();
```

# Conclusions

Lancaster University

- Object Orientated programming is all about modularity
  - Data + Methods = Object
- Classes provide a specification of objects
- Classes are custom types, defined using the class keyword.
- Object instances are real, concrete "variables" of these new types
  - Use **.** to access methods and attributed within an object instance
- **NOT SO SCARY IS IT?**