Engineering

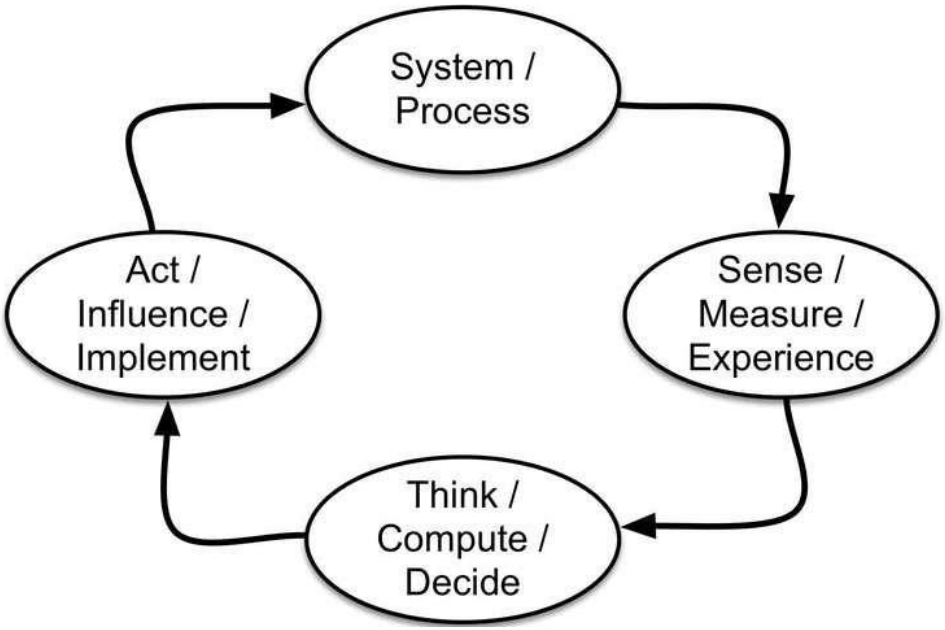
KCL Summer School 2019

Introduction to the Arduino Microcontroller

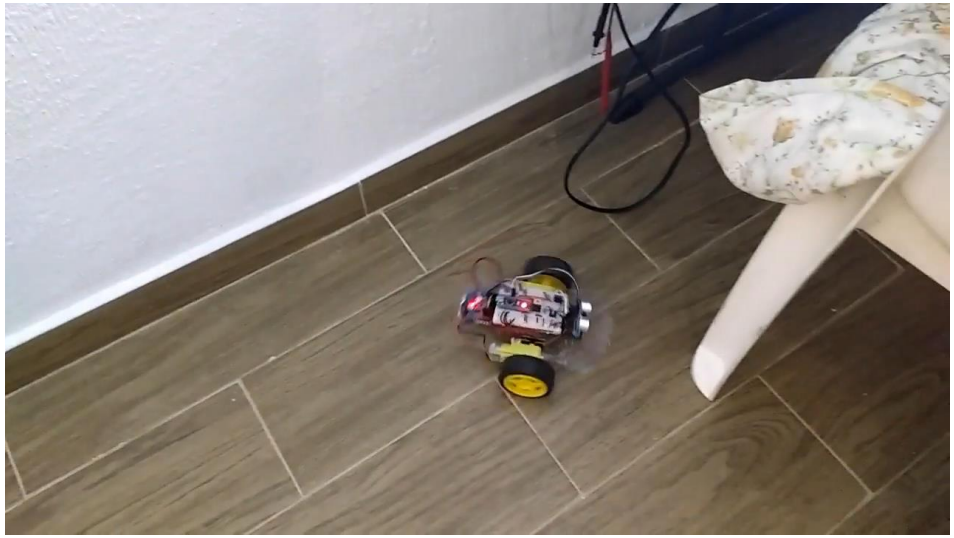
Summary

- Introduce the Arduino Microcontroller
- Present some simple examples of programs for the microcontroller
- Introduce some simple passive circuit elements for use with the microcontroller

sense, compute, decide, act – remember this?

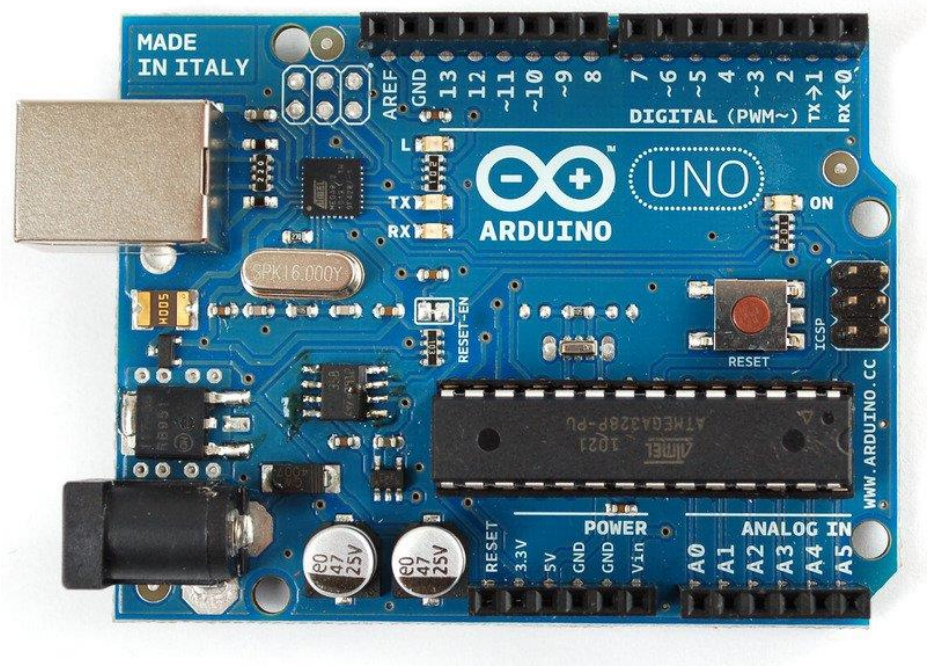


Here's a robot navigating its way around a room –

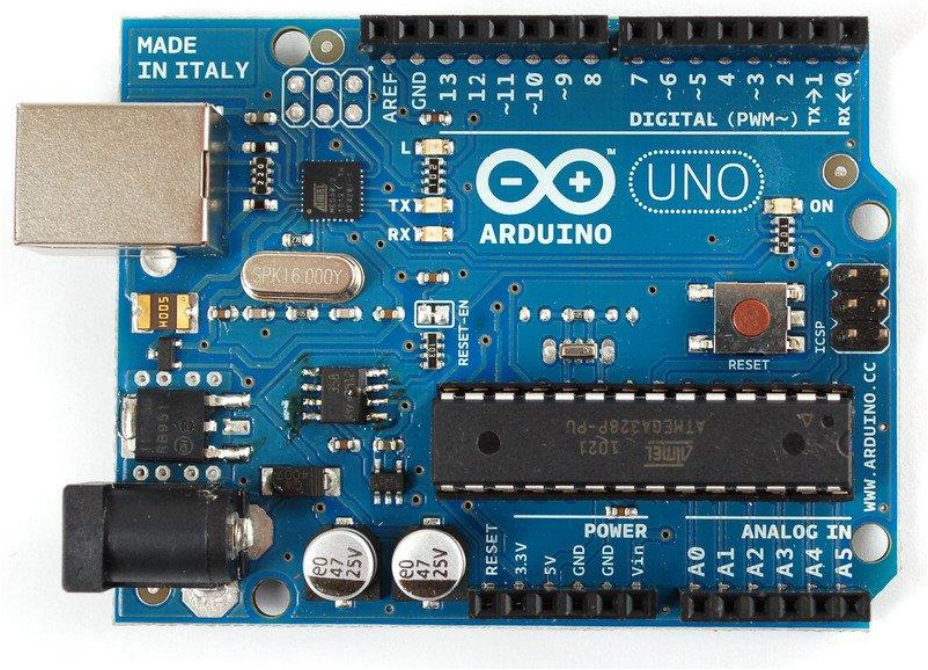


what **sense, compute, decide, act** steps is it following?

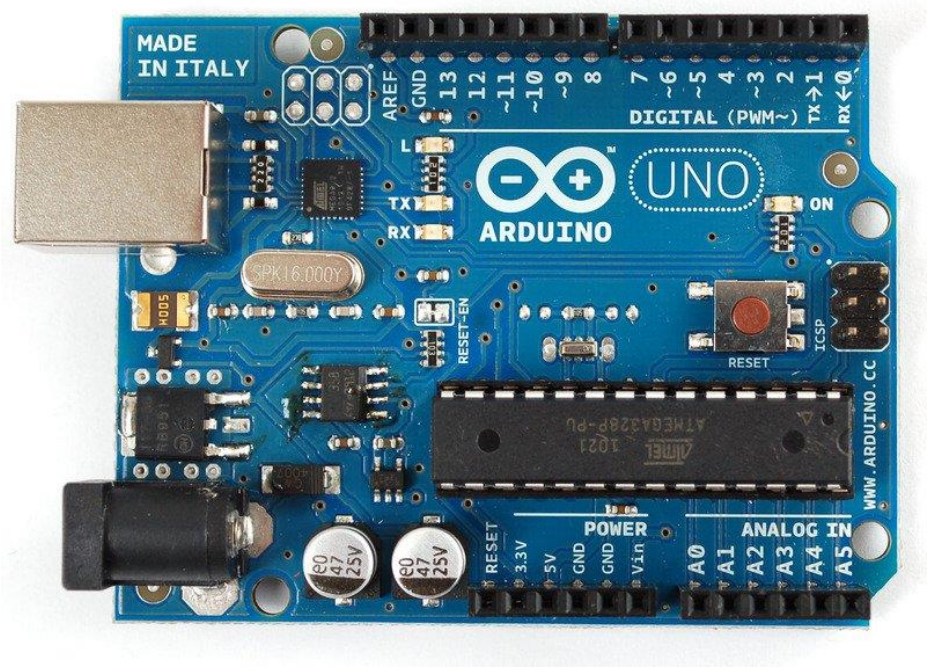
We have many aids to teaching in engineering; today I am going to introduce one of the most widely-used of the modern engineering teaching aids, the **Arduino microcontroller**



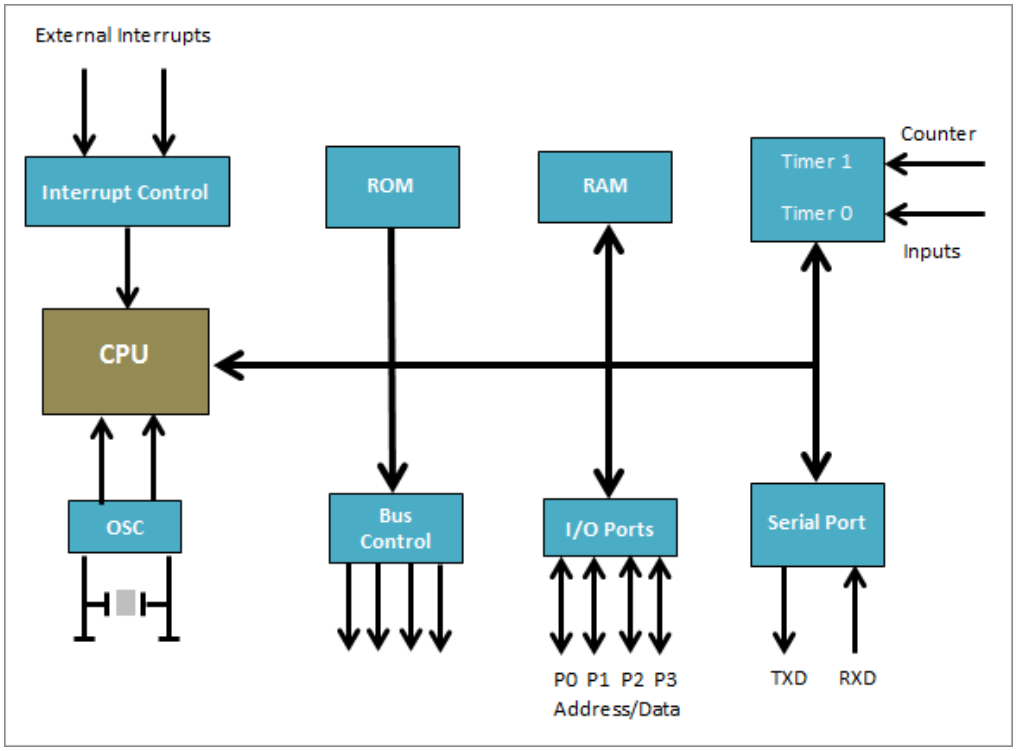
But the first question has to be: what is a “microcontroller”



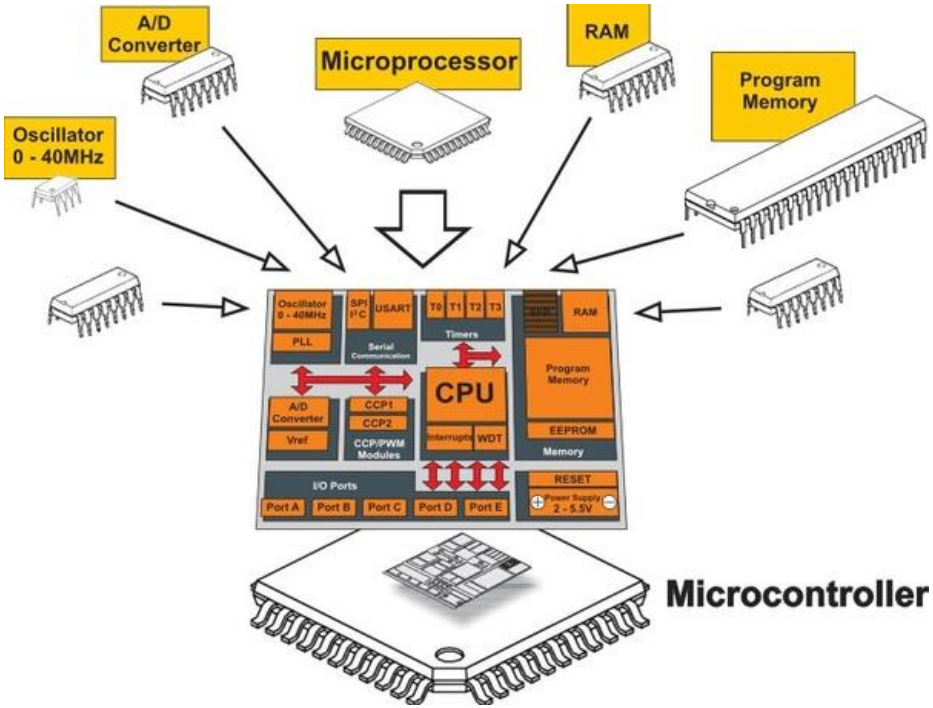
The short answer is it’s a complete computer on a single chip, but let’s expand on that



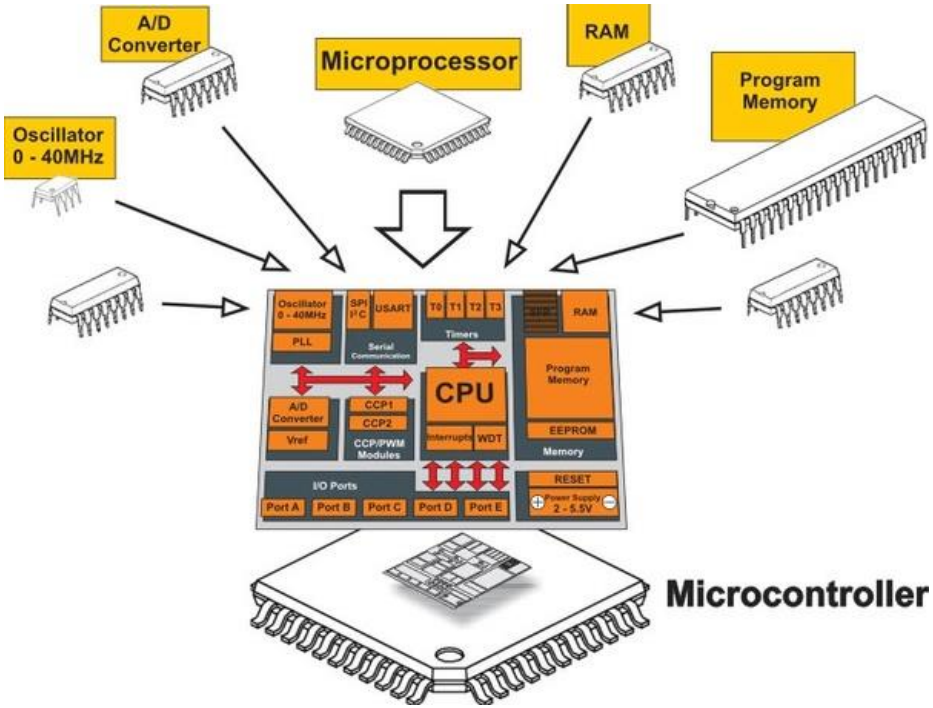
You will be familiar with the concept of a "**microprocessor**" (CPU), the brains of a computer, phone etc.



Although programs are run on the microprocessor, it has no internal memory and no facility to receive inputs or generate outputs, so these programs and functions must be stored on separate devices



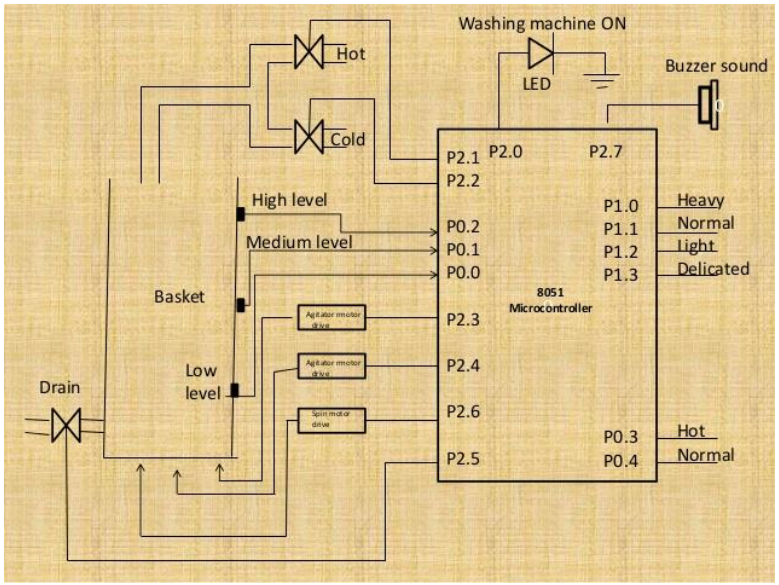
A **microcontroller** combines all these devices on a single chip, sacrificing flexibility and performance for simplicity. They are often referred to as an "**MCU**" not a CPU



So, while microcontrollers cannot replace microprocessors for many applications – they aren't "smart" enough – they can provide a simple and cheap alternative to computers for some dedicated applications

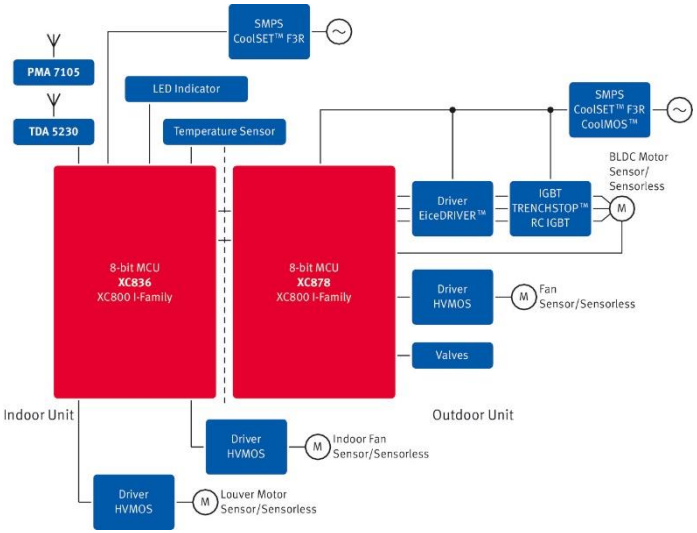
How many of you have a washing machine at home?

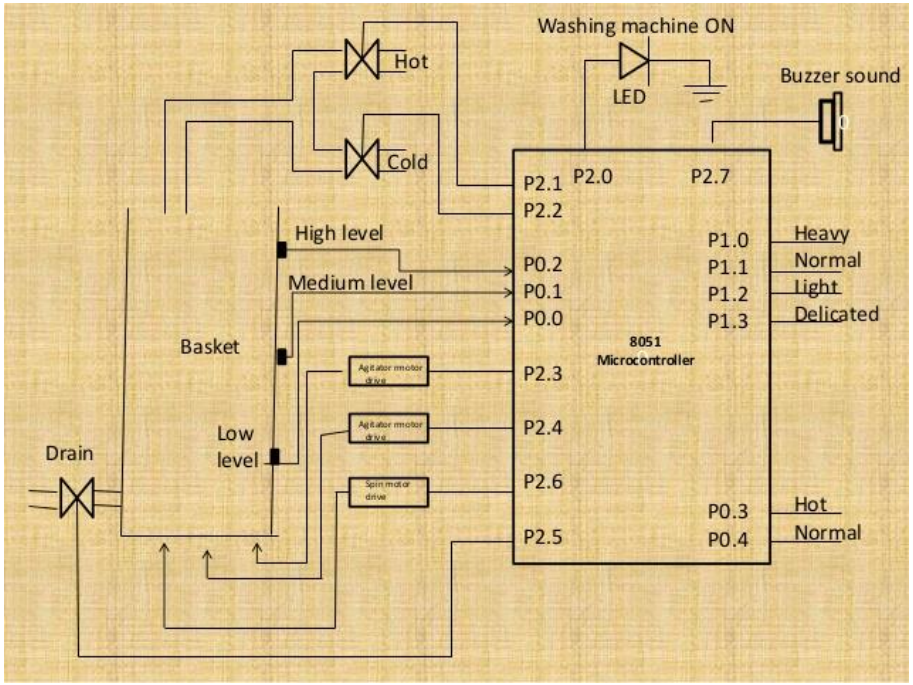
How about air conditioning units?



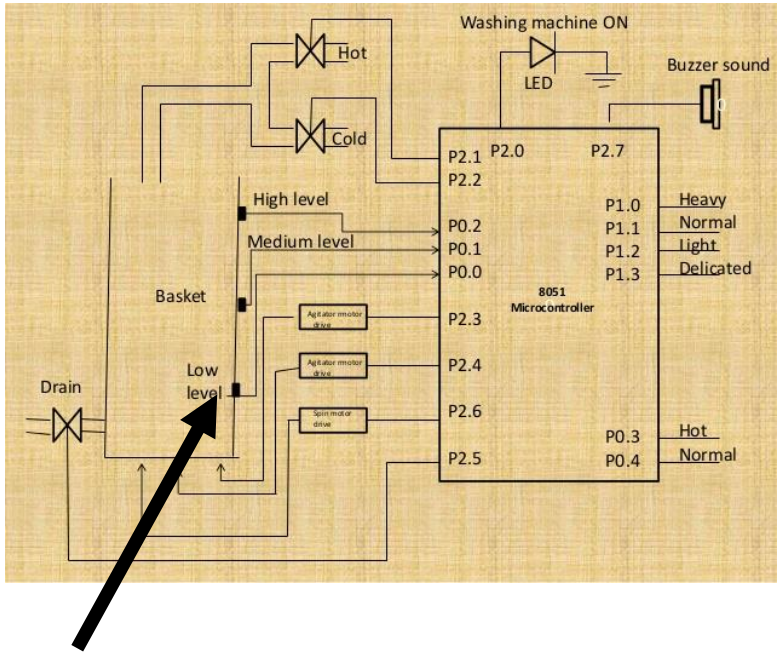
Your washing machine [probably] contains a microcontroller

As will your air conditioning unit



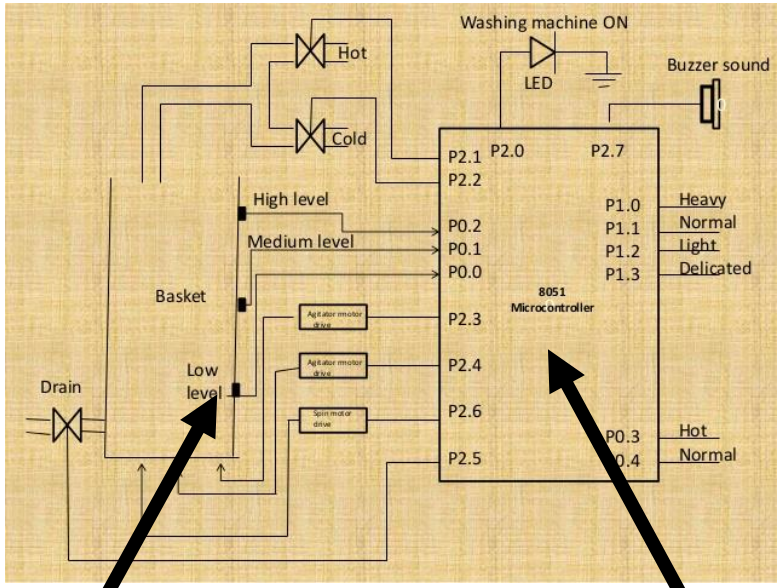


With a microcontroller, you have a device that can take a **reading/measurement** from a sensor as an input, perform a **computation** using that reading, and, based on the result, make a **decision** on what **action** to take



Reading (water level)

With a microcontroller, you have a device that can take a **reading/measurement** from a sensor as an input, perform a **computation** using that reading, and, based on the result, make a **decision** on what **action** to take

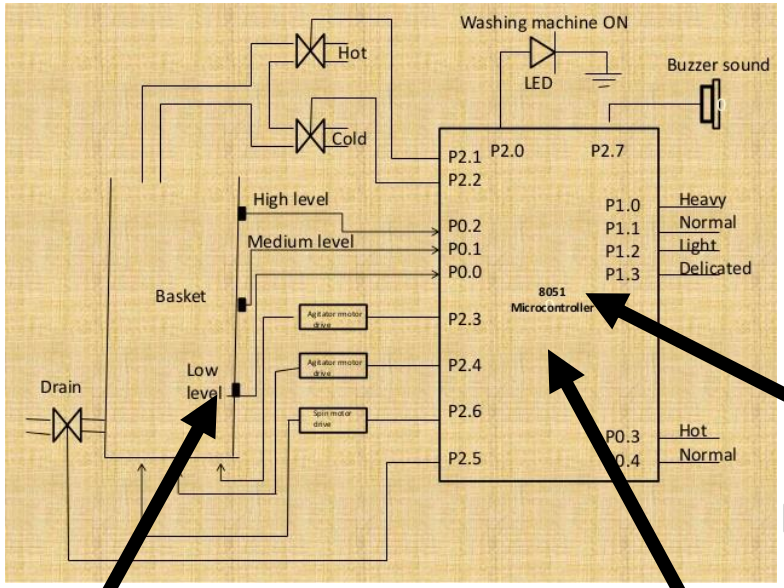


Reading (water level)

Computation (is water level = x?)

With a microcontroller, you have a device that can take a **reading/measurement** from a sensor as an input, perform a **computation** using that reading, and, based on the result, make a **decision** on what **action** to take

With a microcontroller, you have a device that can take a **reading/measurement** from a sensor as an input, perform a **computation** using that reading, and, based on the result, make a **decision** on what **action** to take

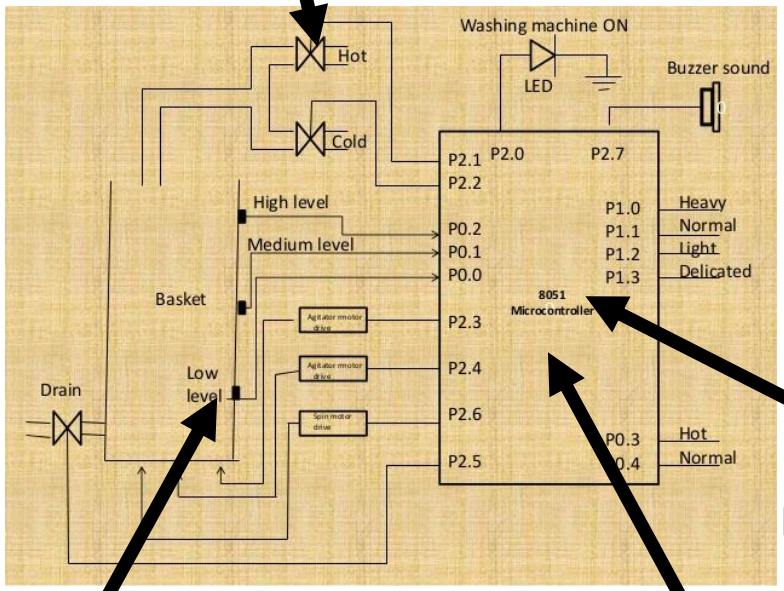


Decision (close water valve(s) y/n?)

Reading (water level)

Computation (is water level = x?)

Action (close water valve(s))



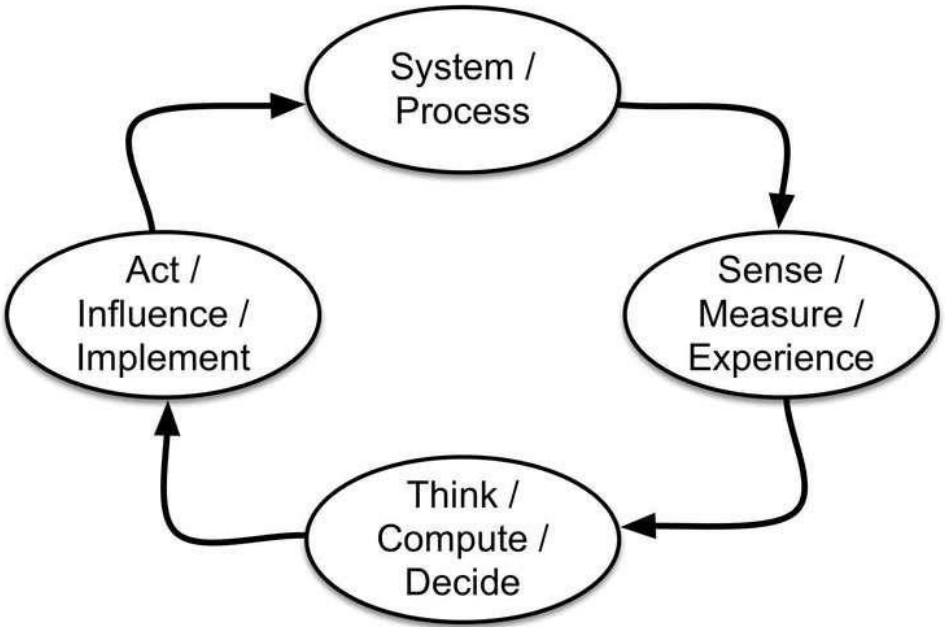
With a microcontroller, you have a device that can take a **reading/measurement** from a sensor as an input, perform a **computation** using that reading, and, based on the result, make a **decision** on what **action** to take

Decision (close water valve(s) y/n?)

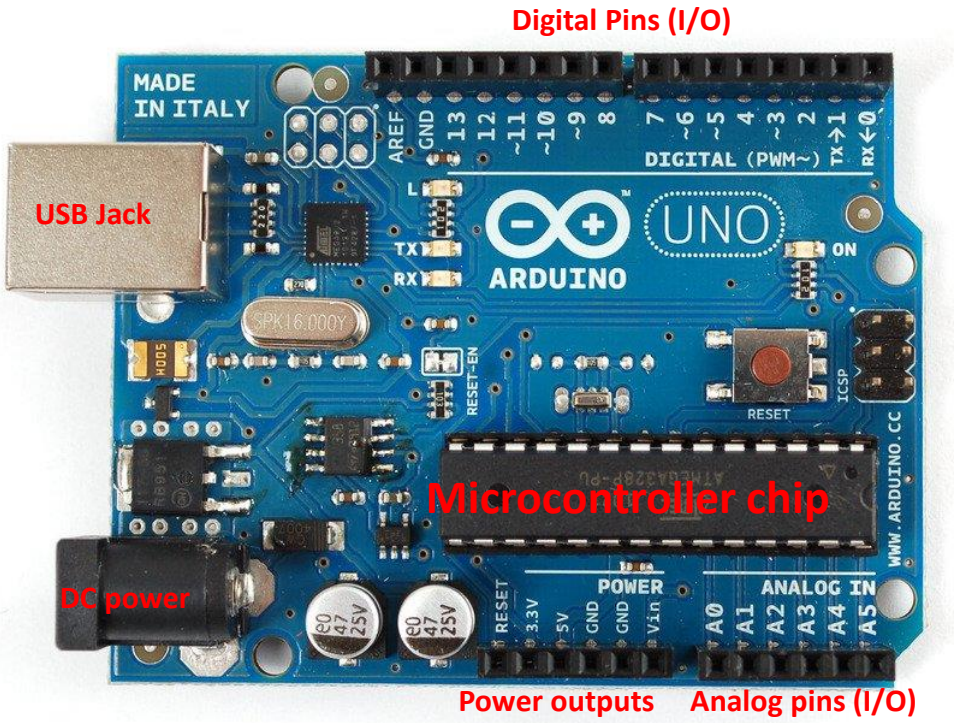
Reading (water level)

Computation (is water level = x?)

So there we have our **sense, compute, decide, act** process. This is why microcontrollers are really useful aids in engineering teaching

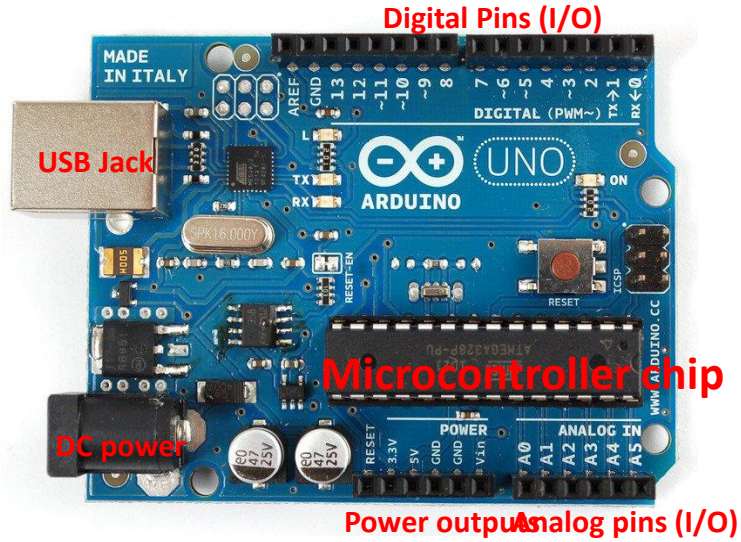


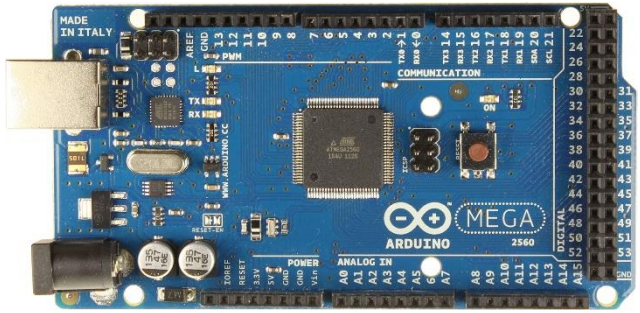
The Arduino microcontroller is a board mounting a microcontroller chip providing access to the input and output pins of the chip and allowing for easy connection to a computer for programming and power

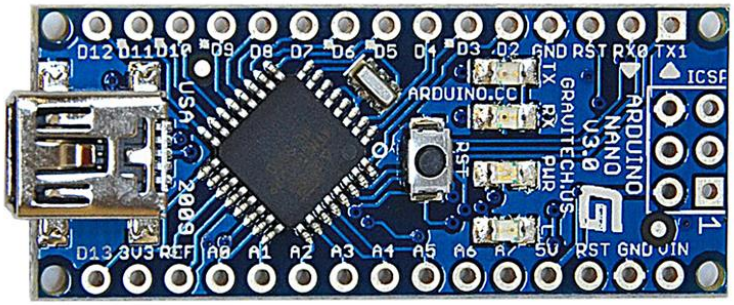


There are several boards (we're using the UNO). All are open source (anyone can make and program them) and supported by:

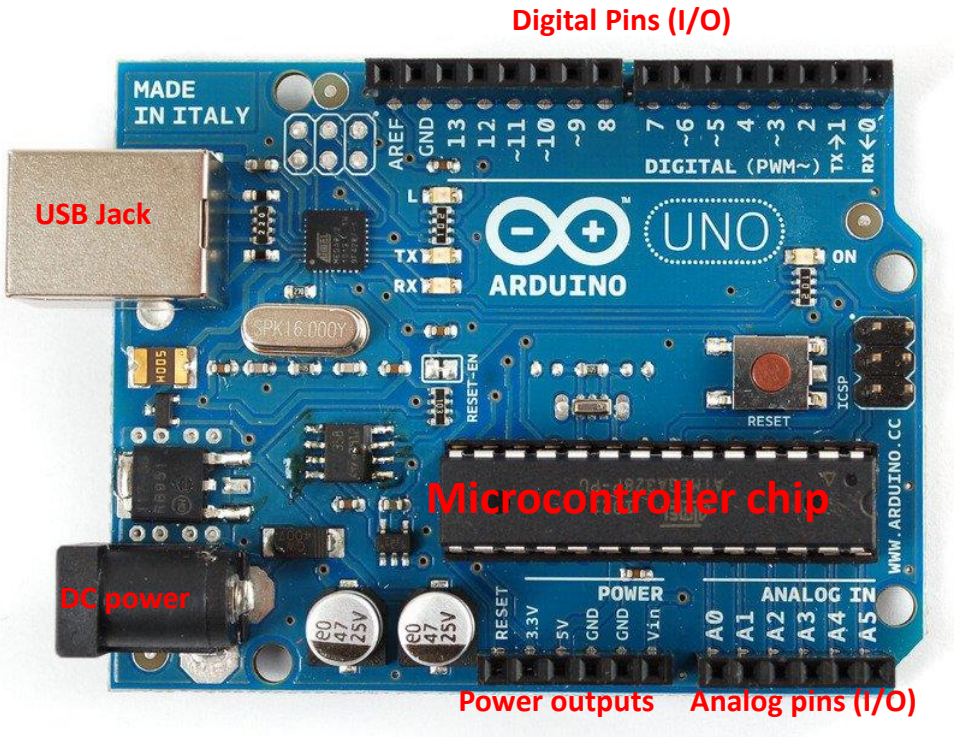
www.arduino.cc

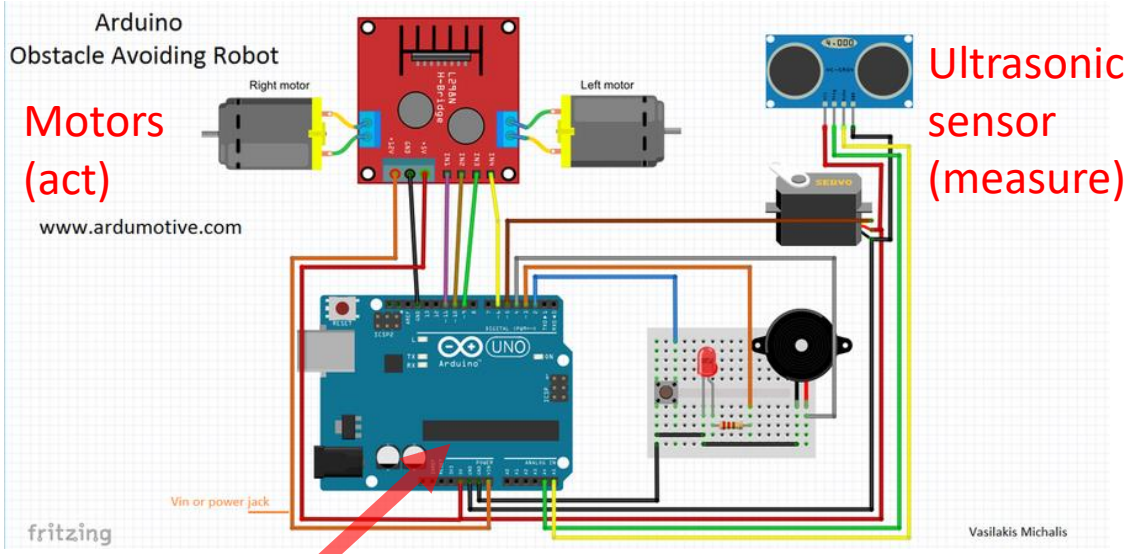




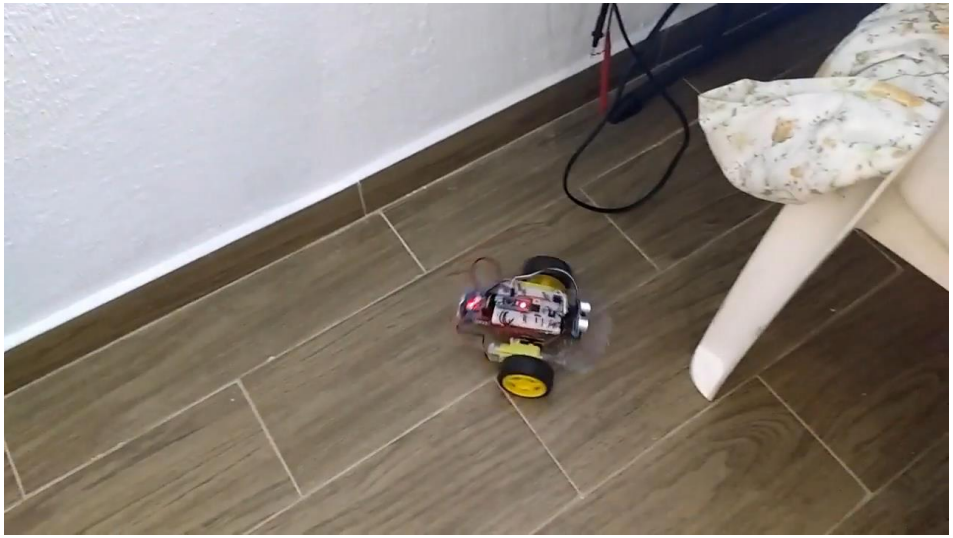


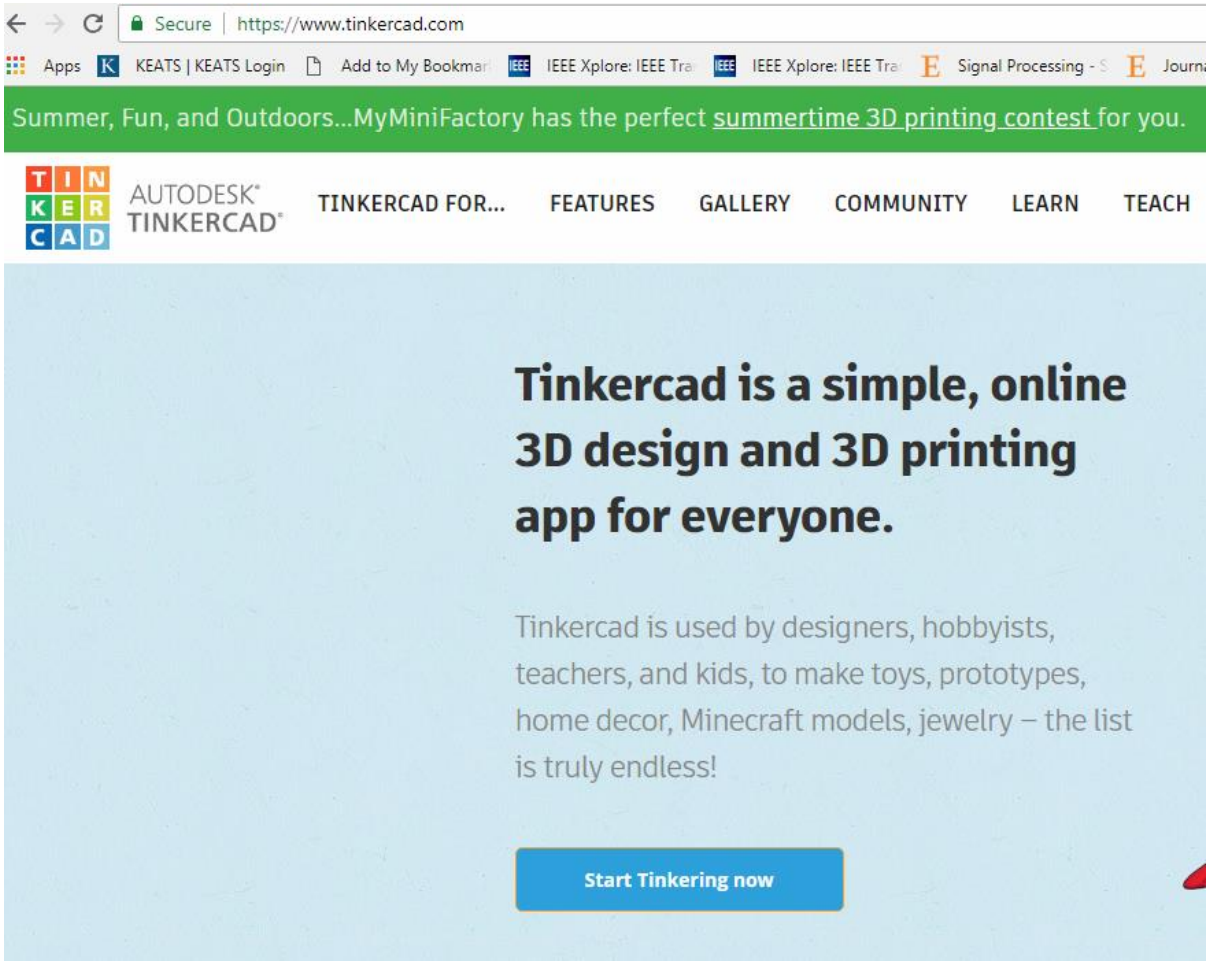
This platform, with both input and output pins, and a chip that can not only run programs, but also store them too, allows us to carry out many operations of interest to engineers, particularly when used in conjunction with external electronic circuits and devices





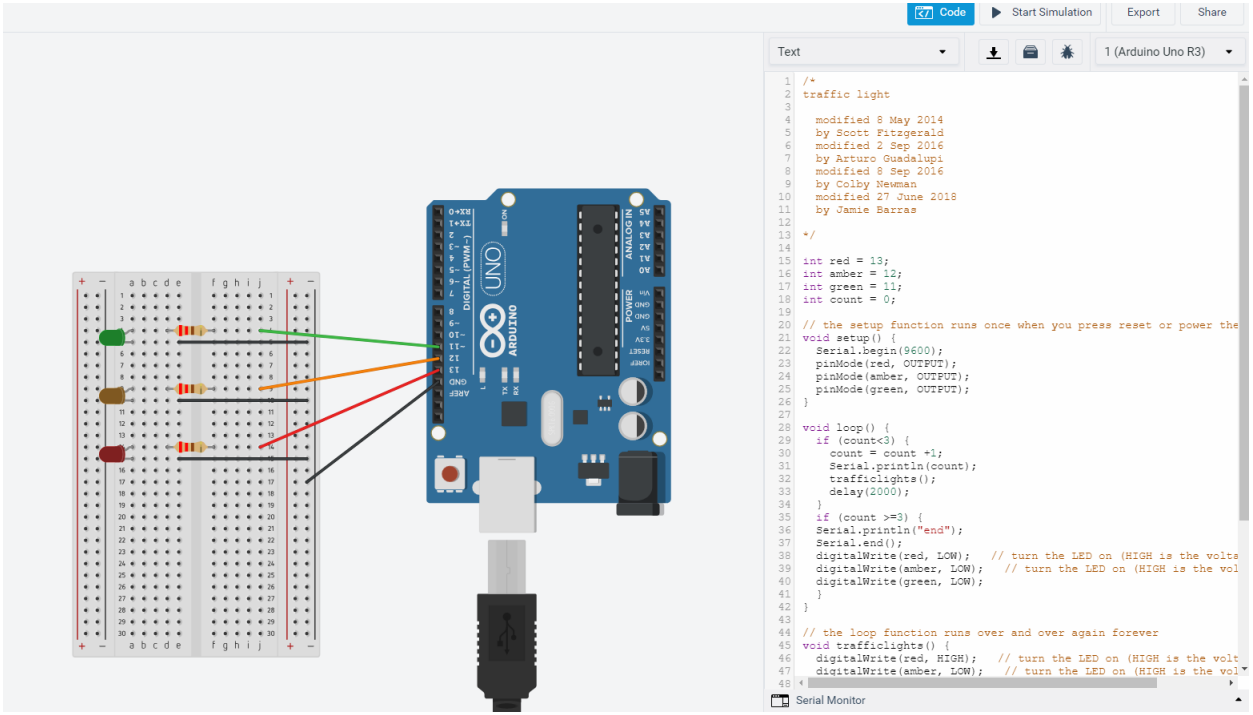
Control program loaded on microcontroller (compute, decide)





This online circuit simulator includes options to simulate Arduino builds AND run sketches

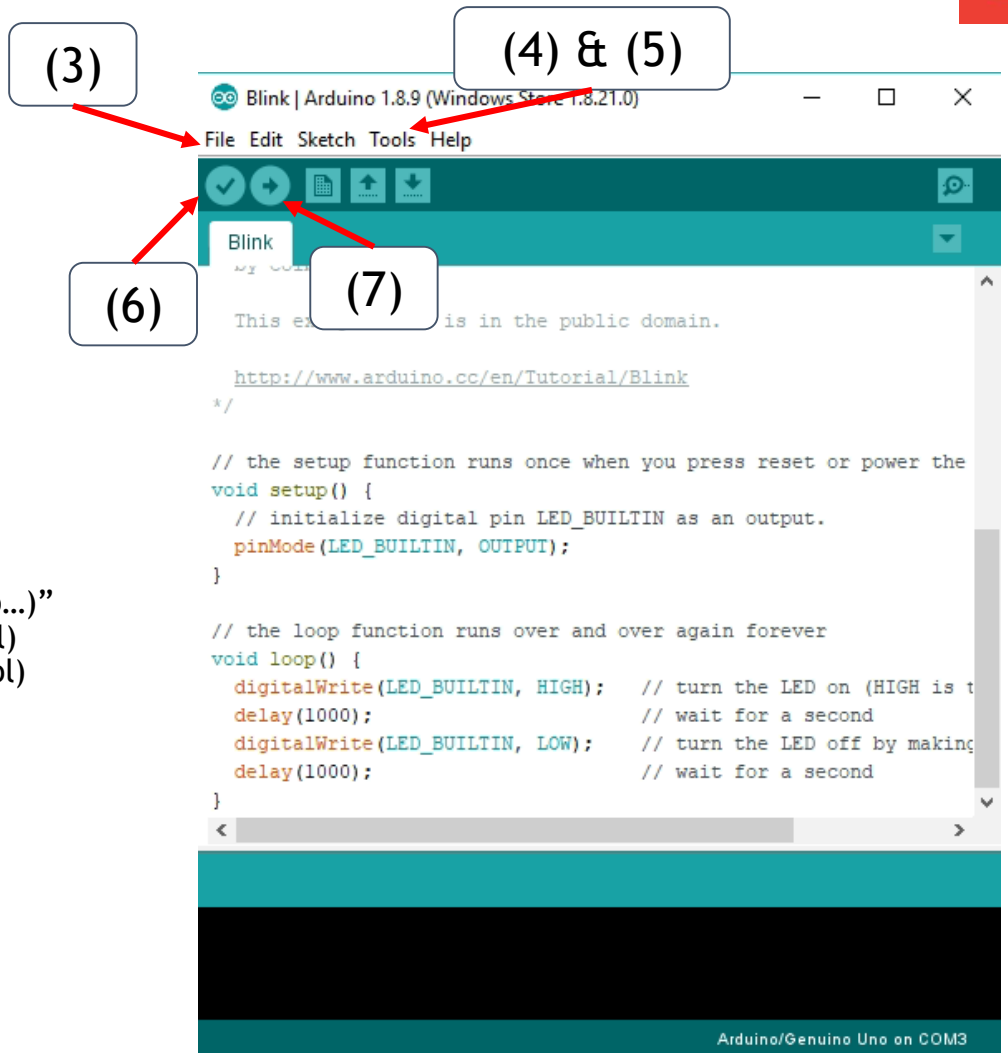
Simply create a free account



Here you can see a simulated build and sketch for a traffic light sequence

Uploading & Running Sketches

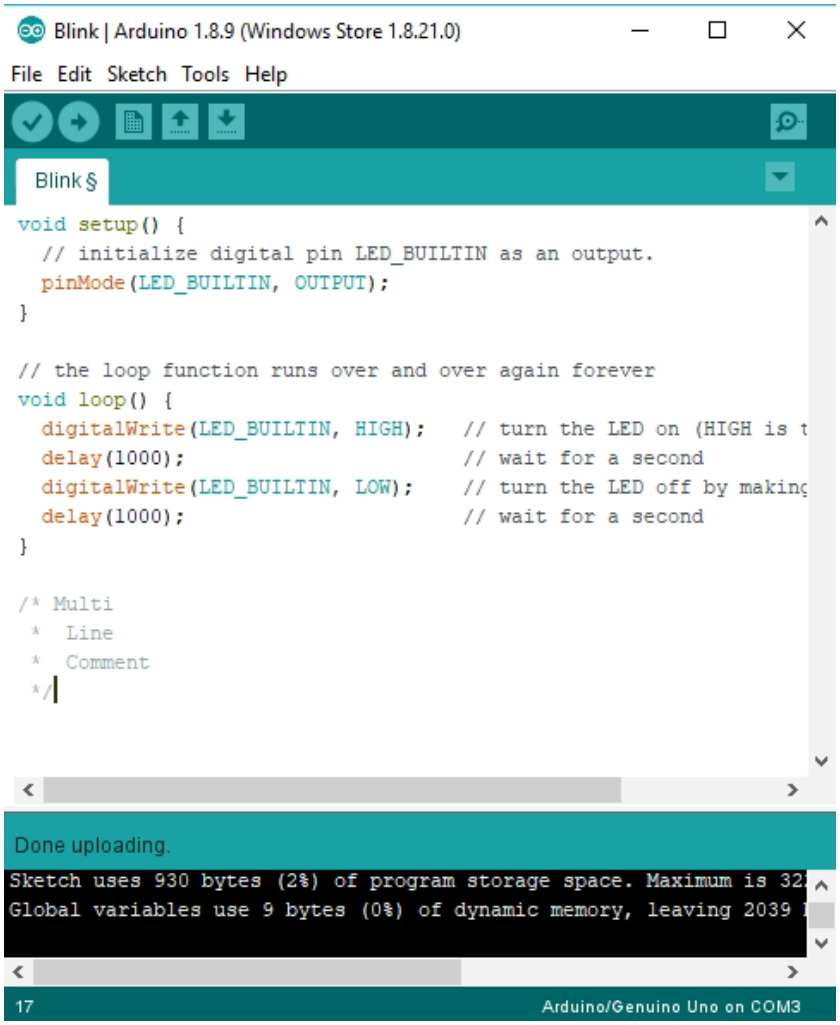
1. Connect your Arduino to the PC
2. Open the Arduino IDE
3. File > Examples > 01.Basics > Blink
4. Tools > Board > Arduino/Genuino Uno
5. Tools > Port > *Select your Arduino*
   a. Something like “COM3 (Arduino…)”
6. Click the Compile button (Tick Symbol)
7. Click the Upload button (Arrow Symbol)



Try this now!

Basic Sketch Anatomy

1. void setup(){...}
    a. Runs **once**
2. void loop(){...}
    a. Called after setup
    b. Runs until Arduino powered off/reset.
3. Comments
    a. “//” at start of line
    b. Or **“/* Many lines of comments */”** for multi-line comments
4. End-of line semi-color **“;”**
5. Curly Braces around functions **“{...}”**



Code Basics - Variables

Variable	Range / Contents	Storage Size	Code
int	-32,768 -> 32767	16-bit (2-byte)	Int var_name = val;
long	-2,147,483,647 -> 2,147,483,648	32-bit (4-byte)	long var_name = val;
bool	true or false	8-bit (1-byte)	bool var_name = val;
byte	0 -> 255	8-bit (1-byte)	byte var_name = val;
char	Used for single characters, e.g. 'A'	8-bit (1-byte)	char var_name = 'A'; char var_name = 65
String	Used for multiple characters, and *conversion*	Variable	String var_name = String(val); Int x = 10; String str_x = String(x);
float	-3.4E+38 -> 3.4E+38 (roughly) 6-7 decimal digits of precision	32-bits (4-bytes)	float var_name = val; float x = 1.19;

There are more variable types, and examples of their use here:
Variables reference page - https://www.arduino.cc/reference/en/#variables
ASCII reference chart - https://www.arduino.cc/en/Reference/ASCIIchart

There is **32,256 Bytes** available on your Arduino (Uno)

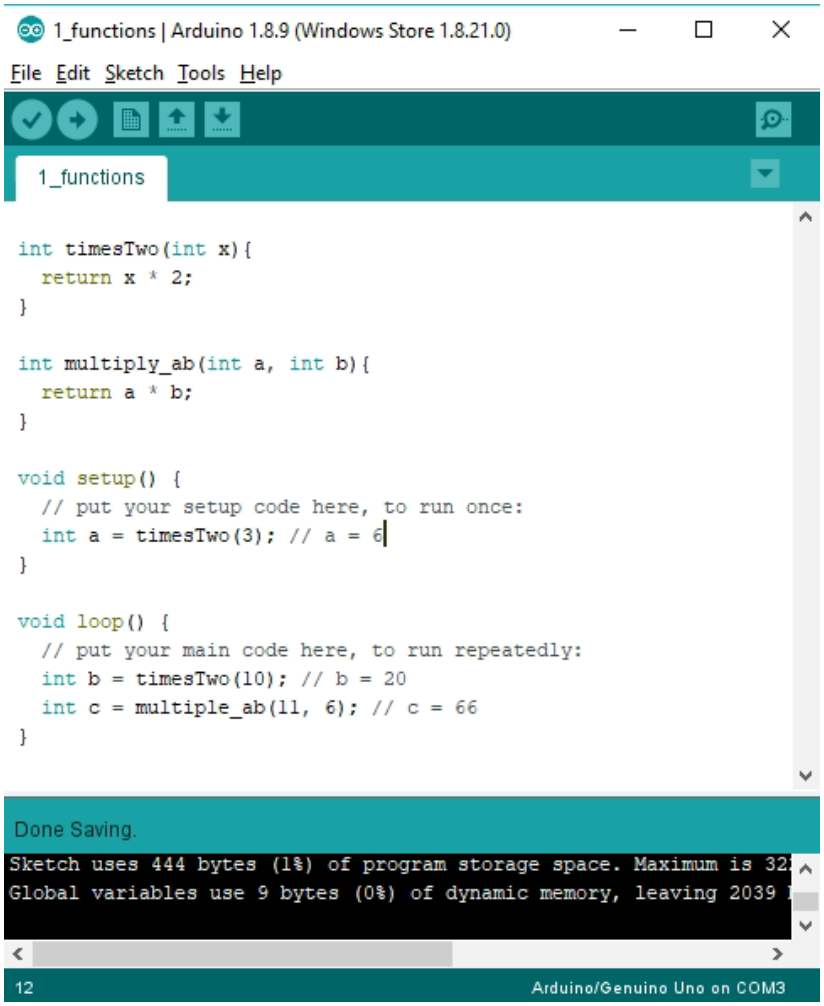
Code Basics - Operators

Operator	Action	Example
==	Equality comparison	int a = 1; int b = 1; bool c = a == b; // c is true
!=	Inequality comparison	int a = 1; int b = 1; bool c = a != b; // c is false
<=	Less-than or equal comparison	int a = 2; int b = 3; bool c = a <= b; // c is true
>=	Greater-than or equal comparison	int a = 2; int b = 3; bool c = a >= b; // c is false
!	Logical NOT	bool c = true; c = !c; // c is false
&&	Logical AND	bool a = true; bool b = false; bool c = a && b; // c is false
||	Logical OR	bool a = true; bool b = false; bool c = a || b; // c is true
%	Remainder	int a = 5 % 3; // a is 2
++	Increment	int a = 3; a++; // a is 4
--	Decrement	int a = 3; a--; // a is 2
+=	Compound addition	int a = 3; a+4; // a is 7

For further details, see “Structure” section in https://www.arduino.cc/reference/en/

Code Basics - Functions

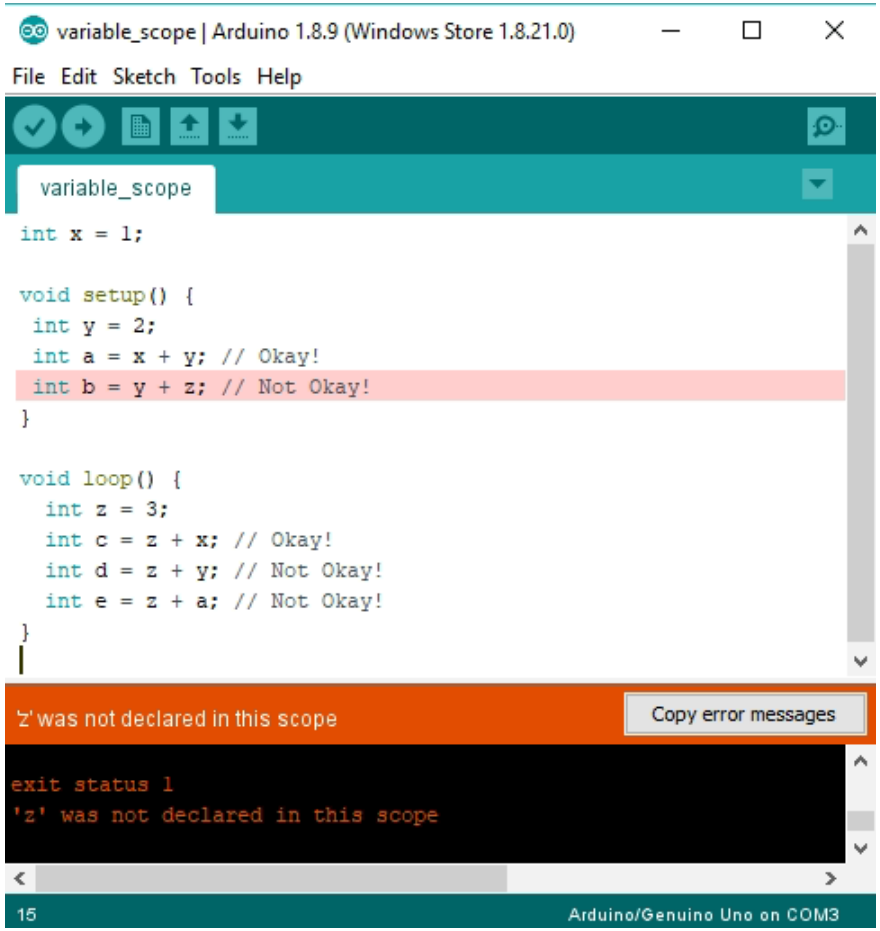
- void setup() and void loop() are our basic functions in Arduino.
- void indicates that these functions do not output any values.
- The empty brackets **()** indicates these functions do not take any *inputs*.
- We can define new functions to help *modularise* our code.
- The output of a function can be set to any of the variable types.
- E.g.
  - int timesTwo(int x){
    - return x * 2;
  - }
- We can have multiple inputs as well.



Code Basics - Variable Scope

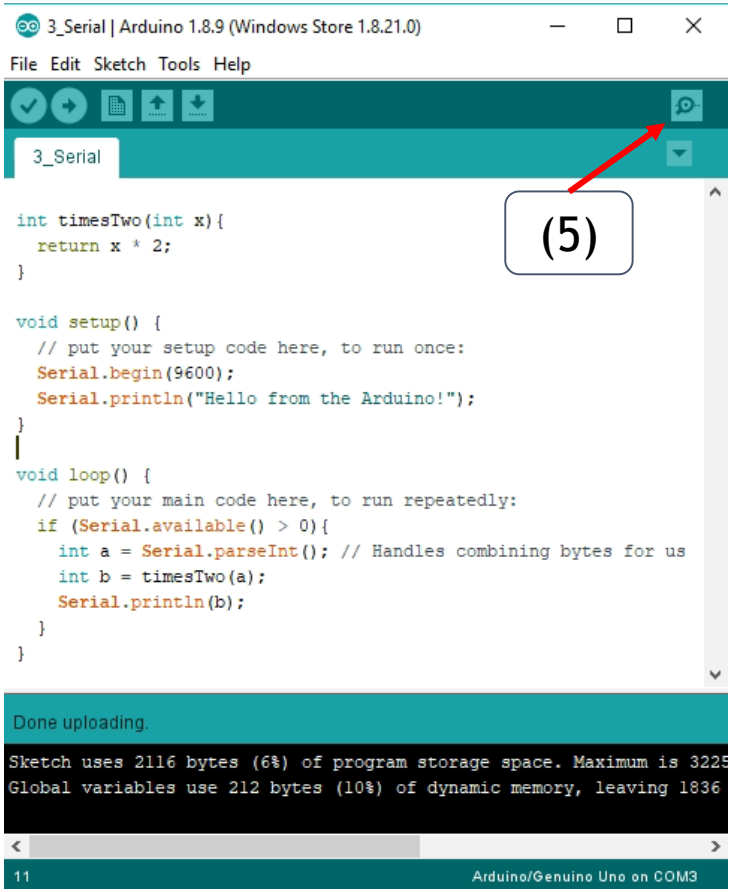
- Variables declared outside of functions are in *global* scope.
  - Can be used by any function.

- Variables declared inside a function are in *local* scope.
  - Can only be used by the function they are in.



Communicating with Arduino

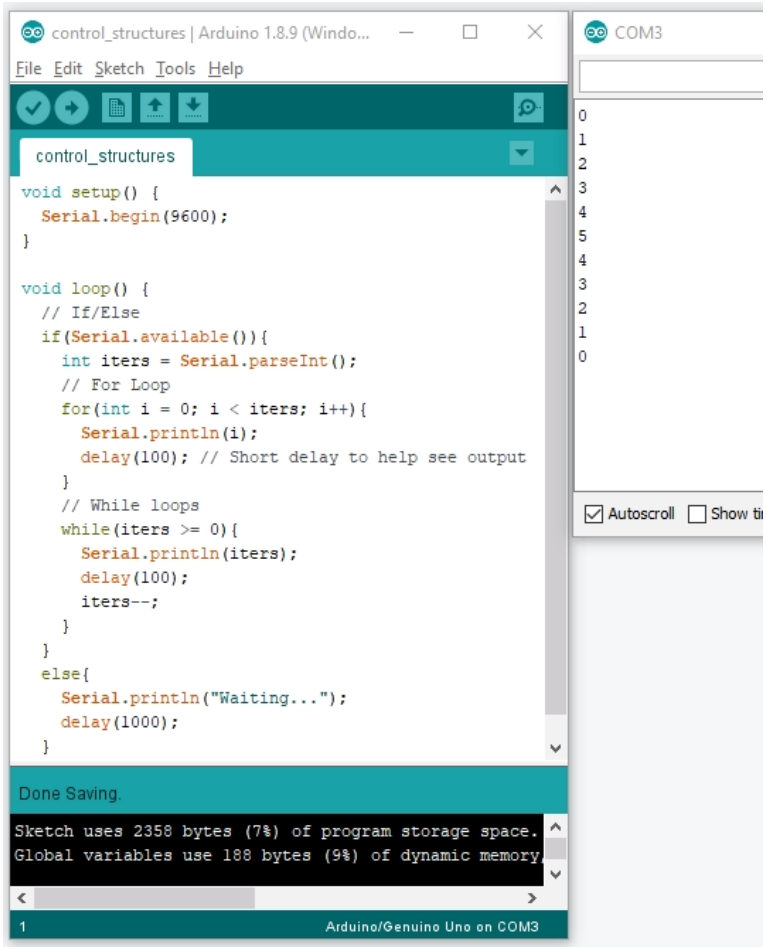
1. Remember, the Arduino is a separate computing device, independent of your PC.
2. We can use a Serial connection through the USB connection to communicate with the Arduino.
3. You can then send and receive strings and characters between your PC and the Arduino.
4. Useful for *debugging* your code if things are not working the way you expect, but there are no apparent errors.
5. Use the Serial Monitor to send/receive values to/from the Arduino.
6. The 9600 in Serial.begin(9600) is the *baud rate*. This is the rate at which symbols are sent over the Serial channel.
    - There are fixed speeds the baud can be set to, 9600 is the default.
    - The Arduino baud must match the PC baud.



More information on Serial functions can be found here:
Serial page - https://www.arduino.cc/reference/en/language/functions/communication/serial/

Code Basics - Control Structures

- If/Else
  - Controls whether sections of code execute based on a conditional check.
- For
  - Executes a section of code for a *defined* number of steps.
  - Keeps track of step count with an incrementing counter.
- While
  - Executes a section of code *until* an input conditional check has been satisfied.
  - Use cautiously on Arduinos. Remember, void loop(){...} handles the iterations of your main code.



For each control structure, the affected code is enclosed in curly brackets {...}

Warmup Exercises

1a) In a new sketch, send the message “Hello World!” from your Arduino to the PC.

1b) Modify this sketch to receive a name from the PC and respond with “Hello <name>!”.

2a) In a new sketch, write a function which calculates the circumference of a circle. Use the terminal to inspect your function output.

2b) Modify this sketch to give 5 decimal points.

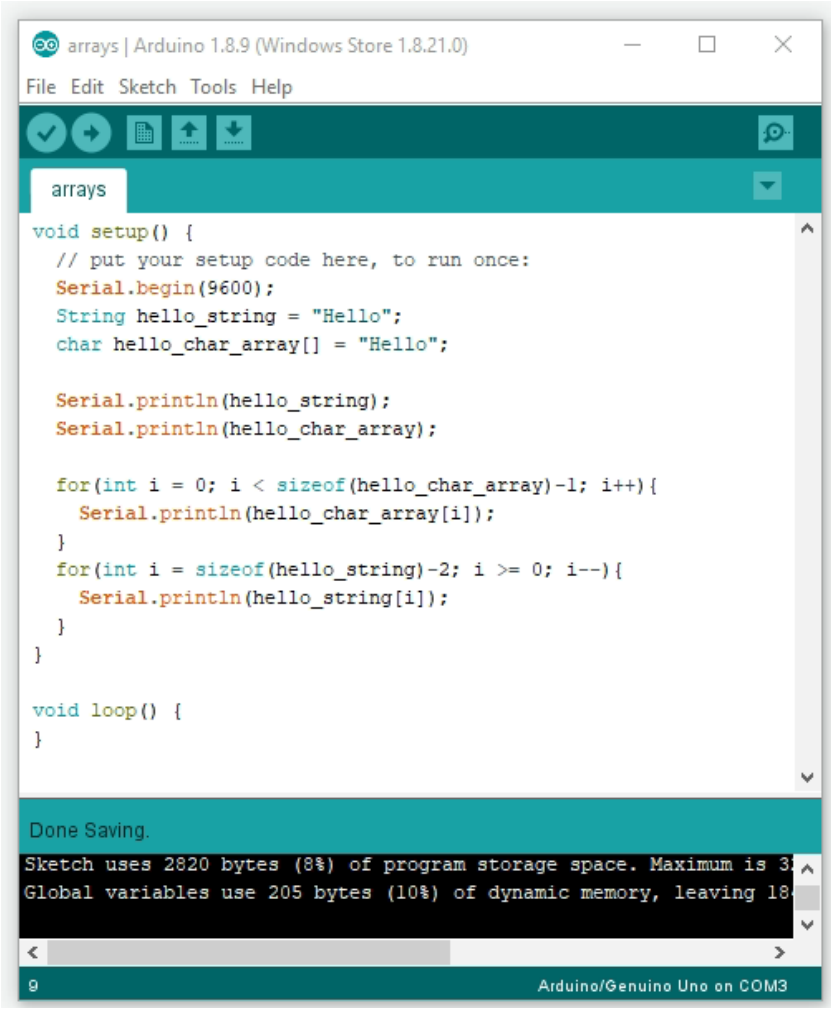
2c) Modify this sketch to take an input radius from the terminal and output the resulting circumference.

Summary

- Introduced the Arduino Microcontroller
- Presented some simple examples of program for the microcontroller
- Introduced some simple passive circuit elements for use with the microcontroller

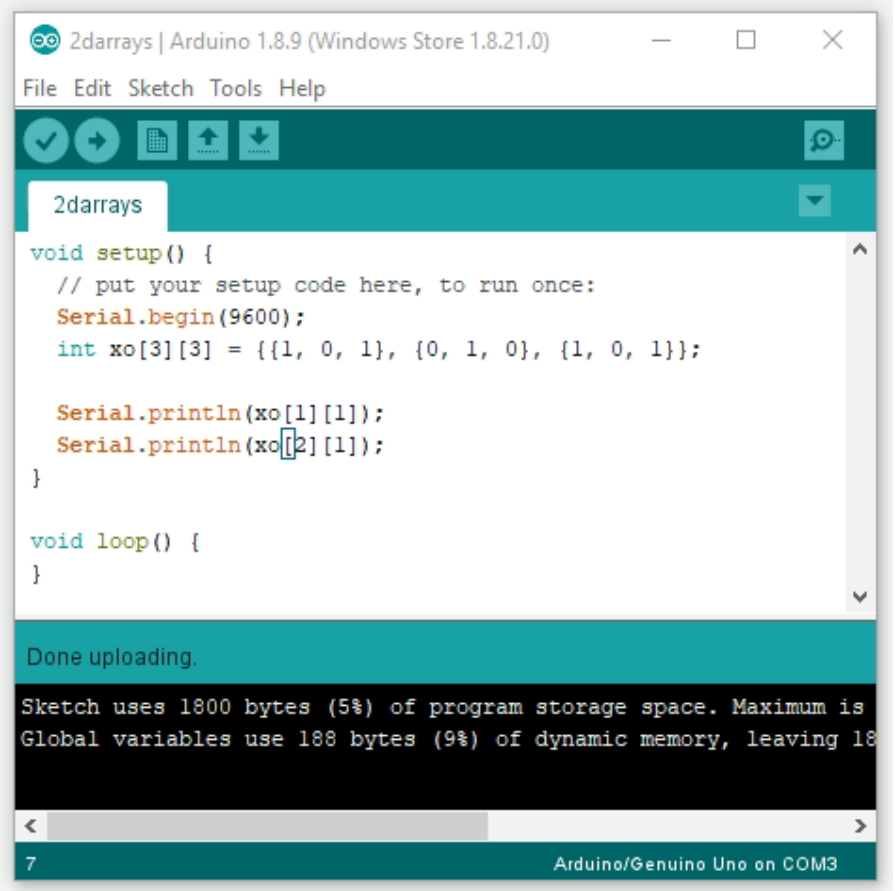
Code Basics - Arrays

- You can have arrays of different variables.
- Defined in same way as previous variables, but with [] after the variable name.
- You can predefine the array size (for efficiency), e.g. char arr[10] or leave it blank (indicating it varies in size).
- You access array elements using square brackets and an index that begins at **zero**.
  - char myName[] = “Aran”;
  - myName[3] = ?



Code Basics - 2D Arrays

- You can have arrays with more than one dimension.
- Similar to the 1D case, you access array elements using square brackets, but provide two indices indicating the [*row, column*] address of the array entry.



Exercise: Control Structures & Arrays

1a) In a new sketch, define a char array with your name and upload it to your Arduino. Note how much storage space the program takes. Change the name variable to type String and reupload the sketch. Note how much storage space the program now takes. Which takes more? Why?

1b) Modify the sketch to print your name to the terminal backwards on one line.

1c) Modify this sketch to only print the vowels in the provided name.

Part 2 is a longer exercise, see how far you can get before lunch (and maybe try finish it at home!)

2a) In a new sketch, write code which will print an 3x3 naughts & crosses grid to the terminal using ‘x’ for player 1, ‘o’ for player 2 and ‘ ’ for empty positions.

2b) Modify this sketch to receive a command input of [player, row, column], e.g. for player 1 placing an x in the top left position we would send “100”, the sketch should then output the updated grid. Ensure the provided move is allowed.

2c) Modify this sketch to check if a player has won, or if there has been a

Accessing I/O in Code

1. **Digital I/O**
   a. Set pinMode in setup()
   b. digitalWrite(pin, val);
   c. Here val is 0 or 1
   d. Can also set to false or
   e. digitalRead(pin,val);
2. **Analog Input**
   a. Set pinMode in setup()
   b. analogRead(pin);
3. **PWM Output**
   a. Set pinMode in setup()
   b. analogWrite(pin, val);
   c. Here val = 0-255;