1. Programming in Arduino

machine code, and it exists as a series of ones and zeros inside the memory of the microcontroller or computer.
machine code might be great for computers and microcontrollers, but it's very difficult for humans to understand.
And early computers in the 1940s assembler.
This is something that translated assembly language to machine code.
Users could write instructions for the computer known as a program with numbers and things that almost looked like words.

While it was still difficult to understand and the language was unique to each type of computer, it was better than writing a machine code.

But having to write, assembly and learn a new set of instructions for each new computer was quite burdensome.
So people began to create high level languages in the nineteen fifties, such as COBOL, Fortran and Lisp.
These languages did not talk in the native language of the computer, but instead relied on compilers to translate the high level language into something the computer can understand.

A compiler is simply a program meant to transform one language into another, often resulting in assembly or machine language that's meant to run on that computer.
we're using a high level language to create instructions for Arduino.

When you write in Arduino program, you're actually writing in a combination of C and C++.
The programming language C was created by Dennis Ritchie while he was working at Bell Labs.
Then in 1979, computer scientist Biyani Strew Strupp created C with classes and this became known as C++.
C and C++ are some of the most popular languages in the world and the freely available.

2. Syntax, Program Flow and Comments

Throughout this section will be going over the syntax for C and C++, when I say syntax, I mean the rules we need to follow in order to have a complete program that will compile and run syntax is similar to grammar in a spoken language.
English,
for example.
I need to have my words in a certain order subject, verb object in order to be understood by the listener.
However, many times I can break some of the rules and still be understood.
But when it comes to syntax, I can't break those rules.
Otherwise, my computer program won't compile. Or if it does run, it'll be full of bugs.
Let's examine the blank program we wrote.

You will first notice the two functions.

1. **Setup and loop functions** encapsulate pieces of code.
2. They combine many lines of code into one.
3. **Functions** are able to return a value after they finish running.
4. But in this case, **setup and loop** return nothing.
5. Since they have the **key word void** in front of the function name.
6. The multiple lines of code that a function encapsulates are contained between a set of curly braces.

The idea is that you could write setup later in your code in order to execute all the lines of code contained in the setup function.
You really shouldn't call setup or loop though, as they are automatically called by Arduino.

Next, you might notice that the **lines within functions are indented.**
C does not care about white space spaces and tabs before lines.
I can delete the spaces before the lines in my functions and the program will still compile and run.
However, it is often considered good style to indent lines between curly braces to make your code easier to read.
That way you'll know that a particular section of code belongs together in a function definition, loop or conditional statement.

Compiler also ignores whitespace when it comes to parentheses and commas used to specify parameters for a function.
Removing all the white space in the pin mode line compiles, as does adding lots of unnecessary white space.
However, adding an appropriate amount of white space between things like commas can make your code easier to read.

Clicking the verify button in our Compiles Your program, but does not upload it to your Arduino board.
This can be a useful quick check to see if your syntax is correct.

Another type of white space

1. **The blank line** is also ignored by the compiler.

You can add blank lines between lines of code to make it easier to read, and it will have no effect on how the program runs.

2. **Parentheses** are used for a variety of reasons and c you will see them used to denote a function.
   parameters inputs to the function are placed between the parentheses.

You will also find them used in mathematical operations to change the order of operations.

3. **double slash** mark is a special indicator to the compiler that we want to ignore everything on the line after the double slash.

It's called a comment and it is not compiled, but is often useful for leaving yourself and others notes in the program about what a line or section of code might be doing.
You can also easily prevent a line of code from being compiled by turning it into a comment by adding slashes before it.

The **asterisk c**ombination to create a **block comment** are doing so will automatically style multiple comment
lines of this manner.

But you just need to begin a comment block with a **slash asterisk** and end it with an **asterisk slash.** Anything in between will be treated as a comment.

4.The **semicolon** is a statement Terminator.
A statement is a command given to the computer arduino in this case that instructs it to take some kind of action.
And the compiler needs to know where the end of your statement is as it does not care about lines.

For example, I can put DeLay and Digital right on the same line and it will make no difference.
The compiler uses the semicolon in the same way.
if you remove the semicolon between these statements, the program would not compile as this line becomes a kind of run on sentence and the compiler has no
idea what to do with it.

However, for many definitions like our set up and loop function definitions, you don't need a semicolon after the curly braces.

We're not telling the processor to do something with these definitions.
We're just telling it what it should do when it encounters calls to set up or loop.

In general, however, it is recommended that you put each statement on its own line so that you have one semicolon per line in order to make the code easier to read.

For the most part, C programs are executed sequentially.
That means our processor in our Arduino is executing one statement at a time before moving on to the next.

5.Flowcharts offer a great way to diagram how a program is executed.
In the Arduino, a flowchart uses shapes and arrows to show the sequence of actions in a system in **Oval or ellipse** shows the start of a sequence, and the **squares** show an action or process that needs to be performed.

All our programs require a set up function and a loop function in our programs.

The setup function acts as an entry point.
This is where the processor enters our code and execution begins after setup is run.

Everything in **the loop function** is executed and the loop function is **run over and over again.**

In blinking LED program, we see that the program starts with the code and setup setting in 13 as an output.
In this case, we then move to the top of the loop function where the first statement is to set PIN 13 to high, make the PIN output be five volts, then we delay for one second, make pin 13 low, which is zero volts, and then delay for another one second.
After that execution returns to the top of the loop function with setting pin thirteen to high again.

3. Literals, Variables and Data Types

literal is simply a fixed value that does not change throughout the program.

For example, when we specify the PIN number for digital right in our Blinkx example, we're using the literal 13, which is an integer.

It's considered a literal because the number 13 won't change while running our program.

1.Other literals include the floating point type, which can be in decimal form or scientific notation where the number after the E is the exponent part.

2.literals can be a single character or a collection of characters known as a string.
3.Boolean phrases true and false as literals.

Datatypes
void, which means no value.

You will generally see this in function definitions to show that the function has no return type or no parameters as inputs.

int
The basic integer takes up **two bytes** in our memory and as such can be a number between negative -32768 to 32767

Floating point numbers are abbreviated as float and can take up four bytes of memory

char characters are stored as a one byte number.
That means only 256 possible characters are available for us in the char type.
Characters are stored as numbers in the ASCII format, the American Standard Code for Information Interchange.

Boolean isn't an original C data type, but it **will work in Arduino** and we can declare one as bool or boolean.

How data types apply to variables

Variable is a storage location paired with a symbolic name often called a reference or a label.

EX: It might help to think about a variable as a type of container.

Here I've got two cups.
The first cup is labeled A and is of type int.
The second cup is labeled B and is of type char.

I would not, for example, be able to store the number three in the char container.
However, I can store in the integer container and I can store the letter Y in the char container but not in the integer container.

Whenever I reference a variable, say A, I would be given my number three and if I reference B I would be given the character Y.
I can swap out the values in a variable, for example, replacing three with the number forty two.

A variable can only contain one value at a time.

how to make and use variables in code

We tell the compiler that we want to reserve a section of memory to store of value with the given data type.
In other words, we create our container.

Notice that we're doing this outside of the setup and loop functions.

This will allow both setup and loop to access the variable and is known as **global scope.**

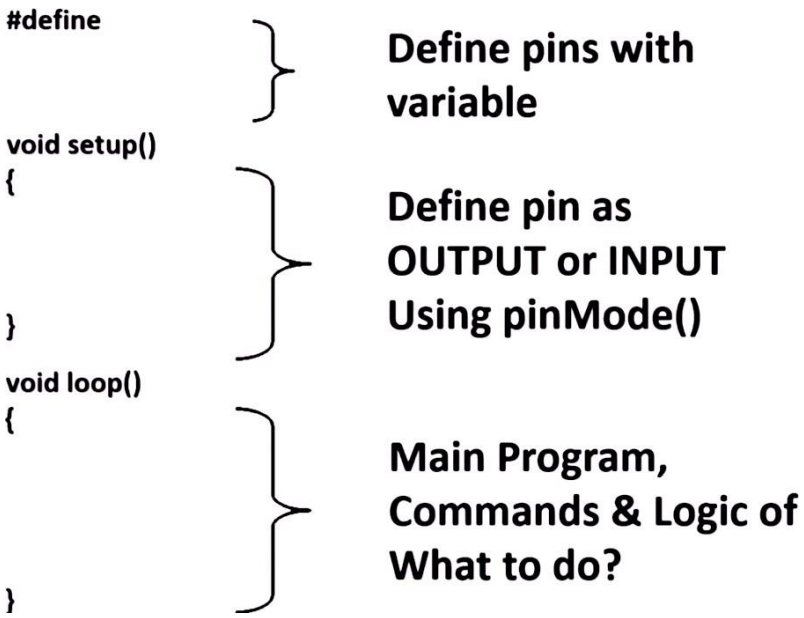


Figure 1: Arduino Blank Sketch

variable names may only contain letters, numbers and underscores.
Also, you are not allowed to use **reserved keywords.**

We then need to initialize the variable, which is where we assign a value to it.

We do that by typing equals some number semicolon, which ends our statement.
Ex in Blink Led
creates a variable named LED and sets it to an initial value of thirteen.
Then whenever we want to access this variable, just use the name led.
We can replace all other instances of 13 in our program with the variable led.

The advantage is that by using a variable, we can change which pin we want to use by changing only one line of code.
Additionally, we can change the value of the variable later in the code at the end of the loop function.

4. Arithmetic Operators

Computers are fantastic at solving basic math problems and microcontrollers are no exception

For example, to calculate a temperature given by a sensor based on some analog voltage.
To do this will first need to understand how to use the arithmetic operators.
There are **six basic operators for performing simple math.**

1. assignment operator "="

This is used to set a variables value and is quite different from the equals sign you might find in regular mathematics.

2. addition, which is used to add two numbers
3. subtraction, subtracting one value from another.
4. Multiplication can be used to multiply two numbers together
5. division is used to determine the number of times one number is contained within another number.
6. Modulo is used to calculate the remainder after the division of one number by another.

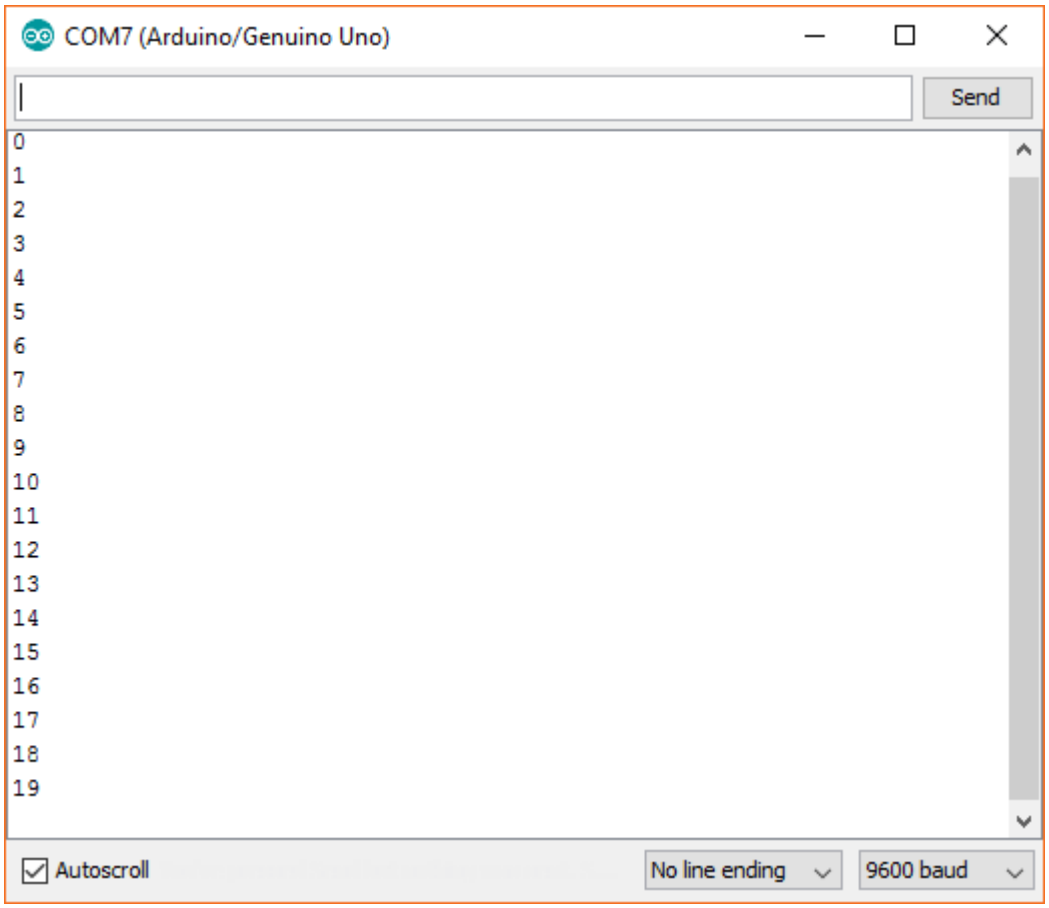
Let's show some examples to see how these work.

Challenge: Count with a Variable

We covered a lot of material in the last few lectures, including program flow, syntax, variables, and arithmetic. Now is a good time to take a break from the videos and have you apply some of the concepts you learned. Open the Arduino program to begin.

Write an Arduino program that starts with an integer at 0, prints it to the Serial Monitor, and increments the integer. It should continue printing and incrementing the integer forever.

Hint: you will want to add some kind of delay so that you can actually read the output. If you open the Serial Monitor, you should see the following:



Solution: Count with a Variable

```
1.  int counter = 0;
2.
3.  void setup() {
4.    Serial.begin(9600);
5.  }
6.
7.  void loop() {
8.    Serial.println(counter);
9.    counter = counter + 1;
10.   delay(500);
11. }
```

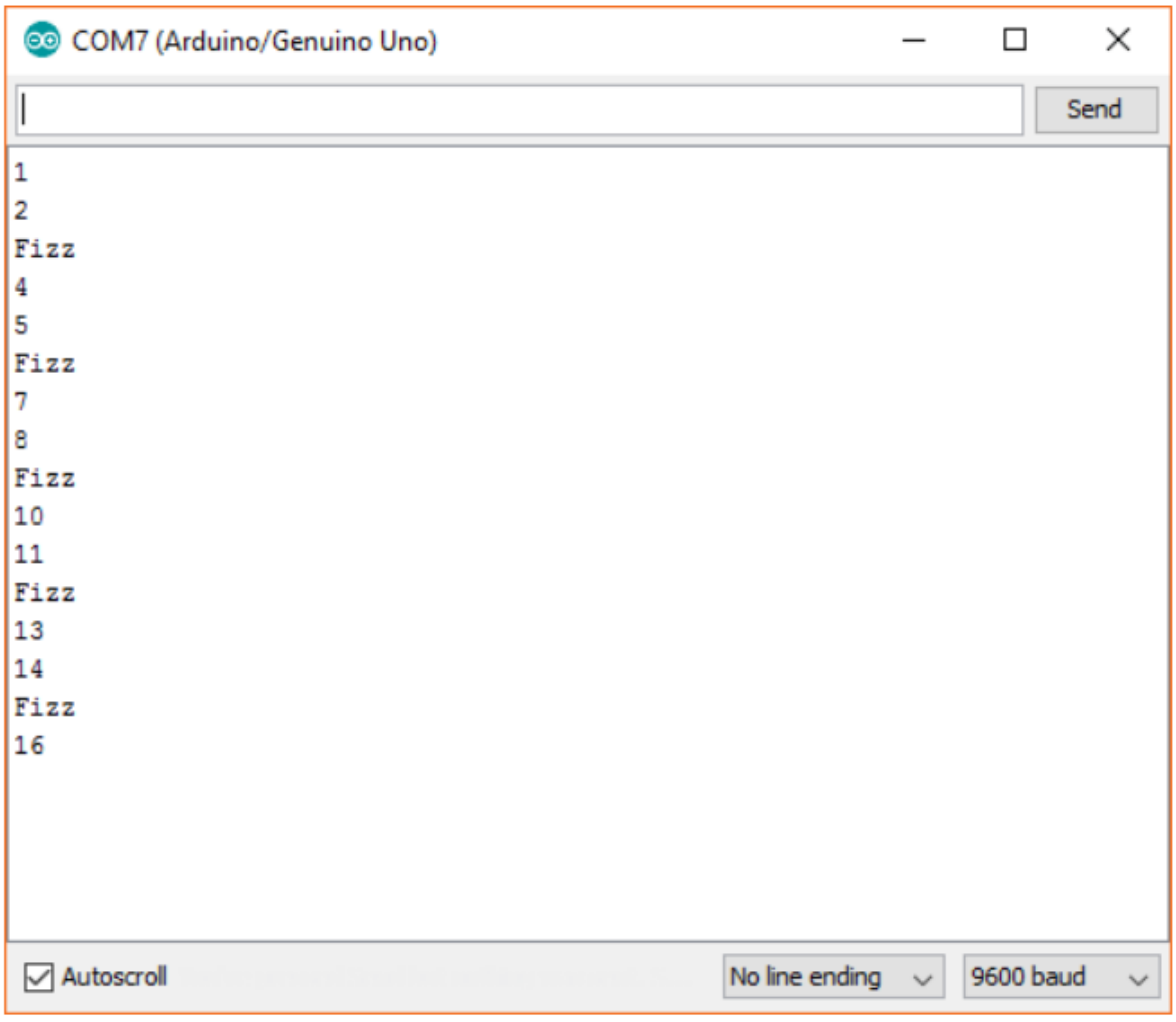
Simulator: https://tinkercad.com/things/2AFVYMPsmhy

5. Conditional statements

- Conditional statements let you control the flow of your program.
- They are also useful for comparing values of variables.
- Make sure that the code you want executing as part of the IF statement is located between the open and close curly braces.
- can use the statement after the if statement closing curly brace.
- if you are having an if statement, or exhibit weird behavior, check your equal signs.
- We can compare two or more different expressions at once in a single if statement to combine the comparisons, we need to use the boolean operators first.

Write an Arduino program that prints integers that count up from 1, except that for every multiple of 3 (3, 6, 9, 12, etc.), the word "Fizz" is printed instead.

Hint: you will want to add some kind of delay so that you can actually read the output. If you open the Serial Monitor, you should see the following:



```
int counter = 1;

void setup() {
  Serial.begin(9600);
}

void loop() {

  // Use an if statement and modulo to determine if
  // our number is a multiple of 3
  if ( (counter % 3) == 0 ) {
    Serial.println("Fizz");
  } else {
    Serial.println(counter);
  }

  // Increment our counter and wait
  counter = counter + 1;
  delay(500);
}
```

Simulator: https://tinkercad.com/things/1qho5tgaRMD

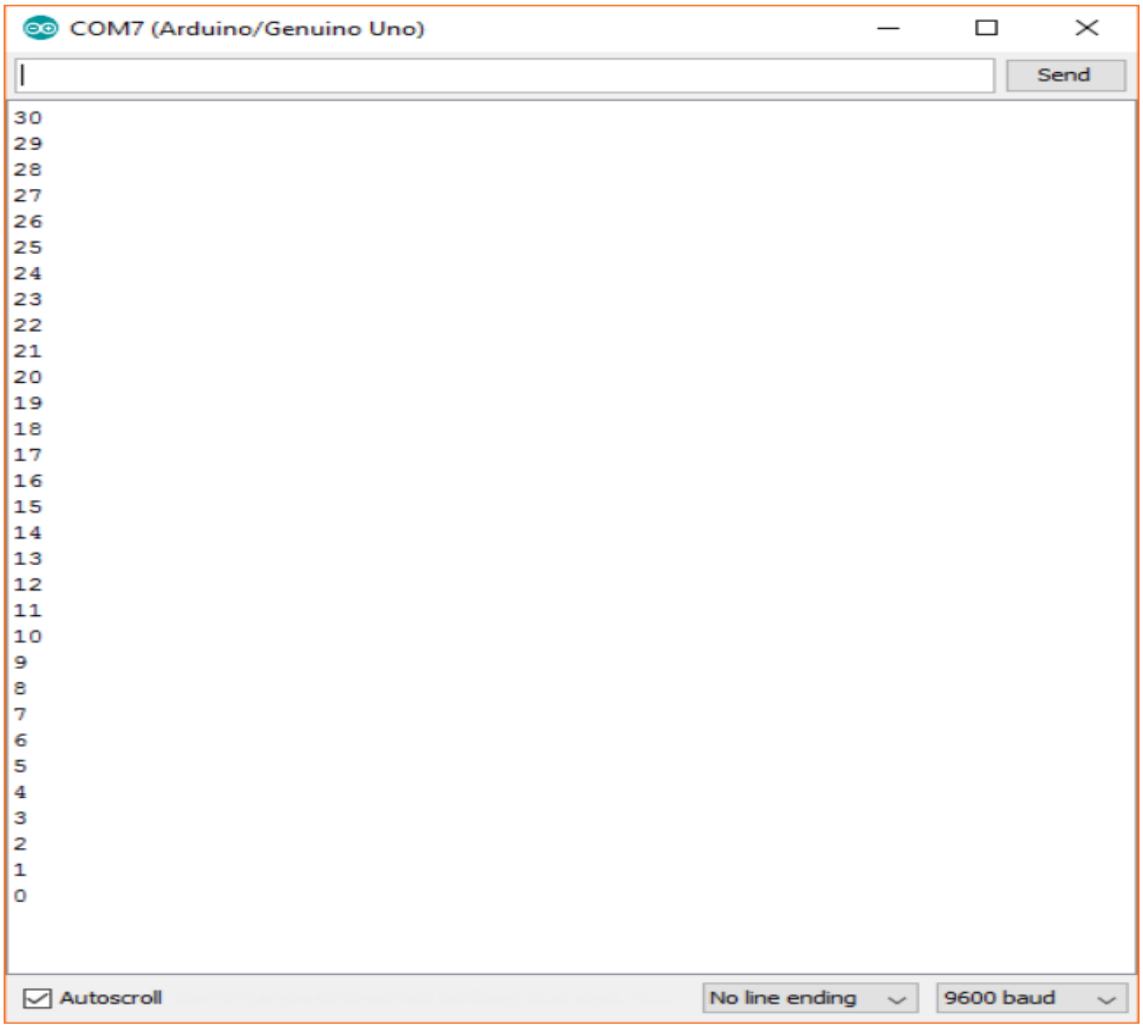
6. Loops

While we have the loop function in Arduino to run a section of code over and over again, let's take a look at some types of control structures known as Loops, which are not to be confused with the loop function.

- Let's say we want to perform a task three times in the setup function like print hello world to the serial monitor.
- We could write serial println(“ Hello World”) three times.
- We first need a variable that will store the number of times we've executed the code inside the loop at the top of the program.
- Much like our IF statements, the code inside the parentheses must evaluate to true or false in order to change the flow of the program.

Write an Arduino program that counts down from 30, printing each integer along the way. Once the program reaches 0, it should print 0 and then stop counting.

Hint: What kind of loop is useful for performing an action a specific number of times? If you open the Serial Monitor, you should see the following:



7. Functions

Syntax

returnType functionName(type parameter 1, type parameter2,….){
function body
optionally return value
}

- function names follow similar rules as naming variables
- use letters, numbers and underscores no spaces.
- define the parameters inside the parentheses after the function.
- between the curly braces is where you write the body of the function
- code runs whenever the function is called
- As your programs get longer, writing functions can help you keep everything neat and orderly
- One should write a function to avoid writing the same few lines of code again and again

Starting with the code below, implement the power() function that calculates the equation: x raised to the power of y. You may assume that the inputs (x and y) and return value are 0 or positive integers (do not worry about handling negative numbers).

```
1. // Use this in your power() function
2. int my_power;
3.
4. // Needed by the test
5. int answer;
6.
7. void setup() {
8.
9.   Serial.begin(9600);
10.
11.  // Should be 9
12.  answer = power(3, 2);
13.  Serial.println(answer);
14.
15.  // Should be 5
16.  answer = power(5, 1);
17.  Serial.println(answer);
18.
19.  // Should be 1
20.  answer = power(9, 0);
21.  Serial.println(answer);
22.
23.  // Should be 16384
24.  answer = power(2, 14);
25.  Serial.println(answer);
26.
27.  // Should be 0
28.  answer = power(0, 4);
29.  Serial.println(answer);
```

```
}

void loop() {
  // Do nothing
}

int power(int x, int y) {

  // Special case: if y is 0, return 1
  if ( y == 0 ) {
    return 1;
  }

  // Store x argument in the my_power variable
  my_power = x;

  // Count down using y. Multiply my_power by x for each iteration
  while ( y > 1 ) {
    my_power *= x;
    y--;
  }

  // Return the accumulated value in my_power
  return my_power;
}
```

Simulator: https://tinkercad.com/things/6a6fC8yK1Er

Basic Functions

//Syntax

//pinMode(pin, mode)

//Reconfigures a digital pin to behave either as an input or an output.

//Example:

pinMode(7,INPUT); // turns pin 7 into an input

//Syntax

//digitalRead(pin)

Cal = digitalRead(3);

//Syntax

//digitalWrite(pin, value)

//Turns a digital pin either on or off. Pins must be explicitly made into an

//output using pinMode before digitalWrite will have any effect.

```
//Example:
digitalWrite(8,HIGH); // turns on digital pin 8


//Syntax
//analogRead(pin)
//Returns
//int (0 to 1023)
val = analogRead(analogPin);


//Syntax
//analogWrite(pin, value)
//value: the duty cycle: between 0 (always off) and 255 (always on).
val = analogRead(analogPin);   // read the input pin
analogWrite(ledPin, val / 4);  // analogRead values go from 0 to 1023, analogWrite values from 0 to 255


//Syntax
//delay(ms)
delay(1000);


//Syntax
//sizeof(variable)
sizeof(variable);
```

8. Arrays and Strings

Let's say that you have 50 values of one type that you want to keep together in your program, for example,
you might have 50 LEDs and you want to remember which ones are on and off.
You could declare 50 separate variables, but that would clutter up your program and make it kind of hard to read.
Instead of that, we could use an array.

All the elements in an array must be the same type.
But you add brackets at the end of the variable name with a number in between the brackets to show how many elements you want in the array.

Index in array: first element has an index of zero, The second element has the index of one and so on.

It's a bad idea to read from an array without first initializing the elements to some values, because they could have random numbers

In C, arrays are stored in memory along with other variables and potentially other pieces of the program.
When we index something out of bounds for the array we created, we get whatever's in that piece of memory.
This could be anything as we generally don't know how the compiler organizes the memory.
This is extremely dangerous and is known as having an index out of bounds of the array, trying to read an element that's out of bounds can give you random results.

Trying to write an element that's out of bounds can be even more dangerous.
You may overwrite another variable or worse, crash the program by overwriting a piece of program memory.
To avoid this, always carefully check how you are indexing your array.
It can help to define a constant before you array with the arrays length and use that instead of writing the number of elements every time.

To do this, we need to use the key word const before end, which tells the compiler that this variable will never change throughout the program.
Const int len equals 10 semicolon above my array declaration.

If you ever need to change the length of the array in the program, you only need to change one number instead of many numbers.

Now let's talk about a special kind of array.
The string, a string is an array of characters which can include numbers, letters, spaces and any special characters from the ASCII table.

Challenge: Compute the Average

Open a new Arduino sketch and copy the following lines to the top (creating 2 global variables):

```
1. const int len = 10;
2. int a[len] = {0, 2, -4, 12, -52, 42, -96, 7, -23, 99};
```

In setup(), compute the average of the a[] array and print it to the Serial Monitor.

In this case, the average of a[] is -1.3. Without modifying your code, you should be able to change the global variables to

```
1. const int len = 8;
2. int a[len] = {125, 1920, 503, 299, 67, 13578, 7632, 22043};
```

and get an average of 5770.8750.

Hint: You can use `Serial.println(float_variable, 4);` to print 4 decimal places.

Solution: Compute the Average

```
const int len = 10;
int a[len] = {0, 2, -4, 12, -52, 42, -96, 7, -23, 99};

void setup() {

  // Declare a local variable to store our average
  float avg = 0;

  // Use a for loop to step through each element, adding it to avg
  for ( int i = 0; i < len; i++ ) {
    avg += a[i];
  }

  // Divide by the length of the array to determine the average
  avg /= len;

  // Initialize the serial port and print the answer
  Serial.begin(9600);
  Serial.println(avg, 4);
}

void loop() {
  // put your main code here, to run repeatedly:
}
```

Simulator: https://tinkercad.com/things/iNTKSebDvIw