Semicolon (;)

- Must be used to end a statement and separate elements in a for loop.
- Forgetting to end a line in a semicolon will result in a compiler error.

```
int x = 13;            // declares variable ‘x’ as the integer 13
```

Structure

- The basic structure of the Arduino programming language is fairly simple and runs in at least two parts.
- These two required parts; or functions, enclose blocks of statements.

```
void setup()
{
      statements;
}
void loop()
      statements;
}
```

- Where setup() is the preparation, loop() is the execution. Both functions are required for the program to work.

- The setup function should follow the declaration of any variables at the very beginning of the program. It is the first function to run in the program, it is run only once, and is used to set pinMode or initialize serial communication.

- The loop function follows next and includes the code to be executed continuously - reading inputs, triggering outputs, etc. This function is the core of all Arduino programs and does the bulk of the work.

void setup()

- The setup() function is called once when your program starts.
- Use it to initialize pin modes, or begin serial.
- It must be included in a program even if there are no statements to run.

```
void setup()
{
      pinmode(pin, OUTPUT)  // sets the ‘pin’ as output
}
```

void loop()

- After calling the setup() function, the loop() function does precisely what its name suggests, and loops consecutively, allowing the program to change, respond, and control the Arduino board.

```
void loop()
{
      digitalWrite(pin, HIGH);    // turns 'pin' on
      delay(1000);                // pauses for one second
      digitalWrite(pin, LOW);     // turns 'pin' off
      delay(1000);                // pauses for one second
}
```

delay(ms)

- Pauses the program for the amount of time (in milliseconds) specified as parameter. (There are 1000 milliseconds in a second.)

```
delay(1000)      //waits for a second
```

- The purpose of the code "delay()" is for the circuit to have a time to accept the command in the circuit.

pinMode(pin, mode):

- Used in void setup() to configure a specified pin to behave either as an INPUT or an OUTPUT.

```
pinMode (pin, OUTPUT);      // sets' pin ' to output
```

- Arduino digital pins default to inputs, so they don't need to be explicitly declared as inputs with pinMode().
- Pins configured as INPUT are said to be in high-impedance state.

digitalWrite(pin, value)

- Outputs either logic level HIGH or LOW at (turns on or off) a specified digital pin.
- The pin can be specified as either a variable or constant (0-13).

```
digitalWrite(pin, HIGH);    // sets 'pin' to high
```

The following example reads a pushbutton connected to a digital input and turns on a led connected to a digital output when the button has been pressed:

```
int led = 13;               // connect LED to pin 13
int pin =   7;               // connect pushbutton to pin 7
int value = 0;               // variable to store the read value

void setup()
{
      pinMode(led, OUTPUT); // sets pin 13 as output
      pinMode(led, INPUT);        // sets pin 7 as input
}
void loop()
{
      value = digitalRead(pin);   // sets ‘value’ equal to the input pin
      digitalWrite(led, value)    // sets ‘led’ to the button’s value
}
```

{} curly braces:

- Define the beginning and end of function blocks and statement block such as the void loop function and the for and if statements.Opening { must always be followed by a closing } (braces being balanced).
- Unbalanced braces can often lead to cryptic, impenetrable compiler errors.

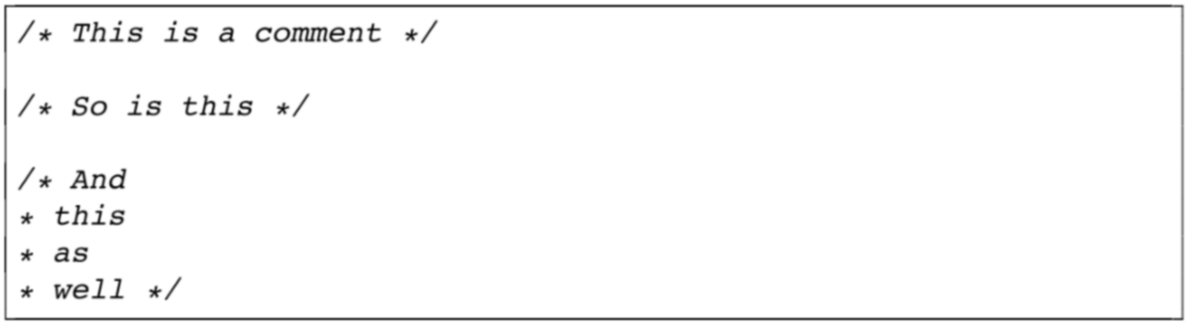
```
type function ()
{
      statements;
}
```

Comments are text that is not used by the Arduino at all, it's only there to help humans like us understand what is going on.

Two types of comments:

- *The block comment style.*

  It starts with a **/*** and continues until a ***/** is encountered. This can cross multiple lines.



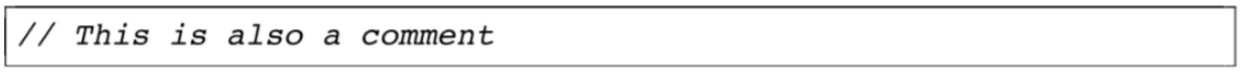
```
/* This is a comment */

/* So is this */

/* And
* this
* as
* well */
```

- *A single line comment.*

  It starts with a // and tells the computer to ignore the rest of the line.



```
// This is also a comment
```

Constants

- The Arduino language has a few predefined values, which are called constants.
- They are used to make the programs easier to read.
- Constants are classified in groups.
- We classify constants in groups:
    - Defining Logical Levels: true and false (Boolean Constants)

True/false

- These are Boolean constants that define logic levels, FALSE is easily defined as 0 (zero) while TRUE is often defined as 1, but can also be anything else except zero.
- So in Boolean sense, -1, 2, and -200 are all defined as TRUE.

```
if (b == TRUE);

{

      doSomething;

}
```

- Defining Pin Levels: HIGH and LOW

HIGH/LOW

- These constants define pin levels as HIGH or LOW and are used when reading or writing to digital pins.
- HIGH is defined as logic level 1, ON, or 5 volts while LOW is logic level 0, OFF, 0 volts.

```
digitalWrite (13, HIGH);
```

- Defining Digital Pins modes: INPUT, and OUTPUT

Input/Output:

- Constants used with the pinMode() function to define the mode of a digital pin as either INPUT or OUTPUT.

```
pinMode (13, OUTPUT)
```

REMEMBER!

- The programming language is case sensitive. In other words, **myVar** is different than **MyVar**
- Whitespace (spaces, tabs, blank lines) is all collapsed to the equivalent of a single space. It is for the human reader only.
- Blocks of code are encapsulated with curly braces '{' and '}'
- Every open parenthesis '(' must have a matching close parenthesis ')'
- There are no commas in numbers. So you must say 1000 and **NOT** 1,000.
- Each program statement needs to end with a semicolon ';'. In general, this means that each line of your program will have a semicolon. Exceptions are:
  - Semicolons (like everything) are ignored in comments
  - Semicolons are not used after the end curly brace. '}'