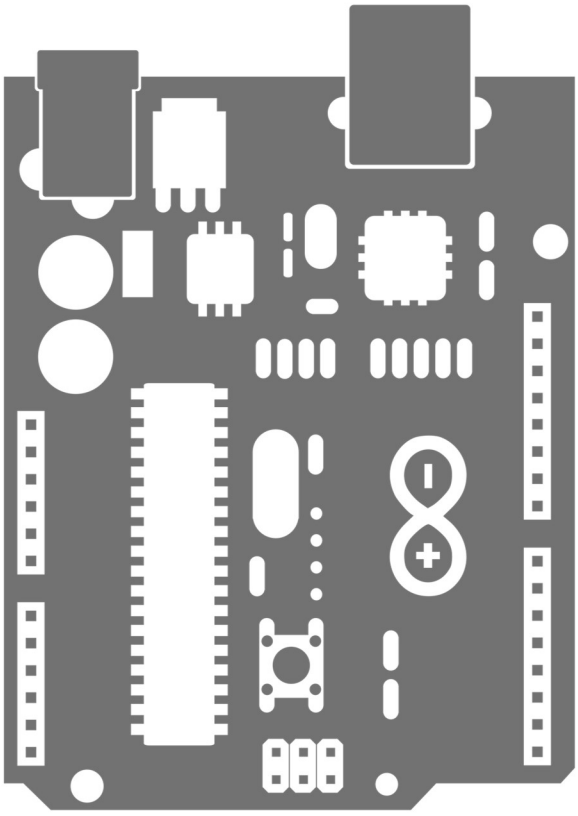
Beginner's Guide to Arduino



Tejas S Shah

Electrical & Electronics Engineer

Table of Contents

Chapter 1
Introduction

1.1 Hello, Arduino!

Have you ever looked at some gadget and wondered how it really worked? Maybe it was a remote control boat, the system that controls an elevator, a vending machine, or an electronic toy? Or have you wanted to create your own robot or electronic signals for a model railroad, or perhaps you'd like to capture and analyse weather data over time? Where and how do you start?

The Arduino board can help you find some of the answers to the mysteries of electronics in a hands-on way. The original creation of Massimo Banzi and David Cuartielles, the Arduino system offers an inexpensive way to build interactive projects, such as remote-controlled robots, GPS tracking systems, and electronic games.

Arduino project has grown exponentially since its introduction in 2005. It's now a thriving industry, supported by a community of people united with the common bond of creating something new. You'll find both individuals and groups, ranging from interest groups and clubs to local hackerspaces and educational institutions, all interested in toying with the Arduino.

The Arduino platform increases in popularity every day. If you're more of a social learner and enjoy class-oriented situations, search the Web for "Cult of Arduino" to see what people are making and to find Arduino-related groups. Members of Arduino groups introduce the world of Arduino from an artist's perspective. Many group members work to create a small Arduino-compatible board at the same time. These groups can be a lot of fun, introduce you to interesting people, and let you share your Arduino knowledge with others.

The Arduino environment has been designed to be easy to use for beginners who have no software or electronics experience. With Arduino, you can build objects that can respond to and/or control light, sound, touch, and movement. Arduino has been used to create an amazing variety of things, including musical instruments, robots, light sculptures, games, interactive furniture, and even interactive clothing.

Though it is easy to use, Arduino's underlying hardware works at the same level of sophistication that engineers employ to build embedded devices. People already working with microcontrollers are also attracted to Arduino because of its agile development capabilities and its facility for quick implementation of ideas.

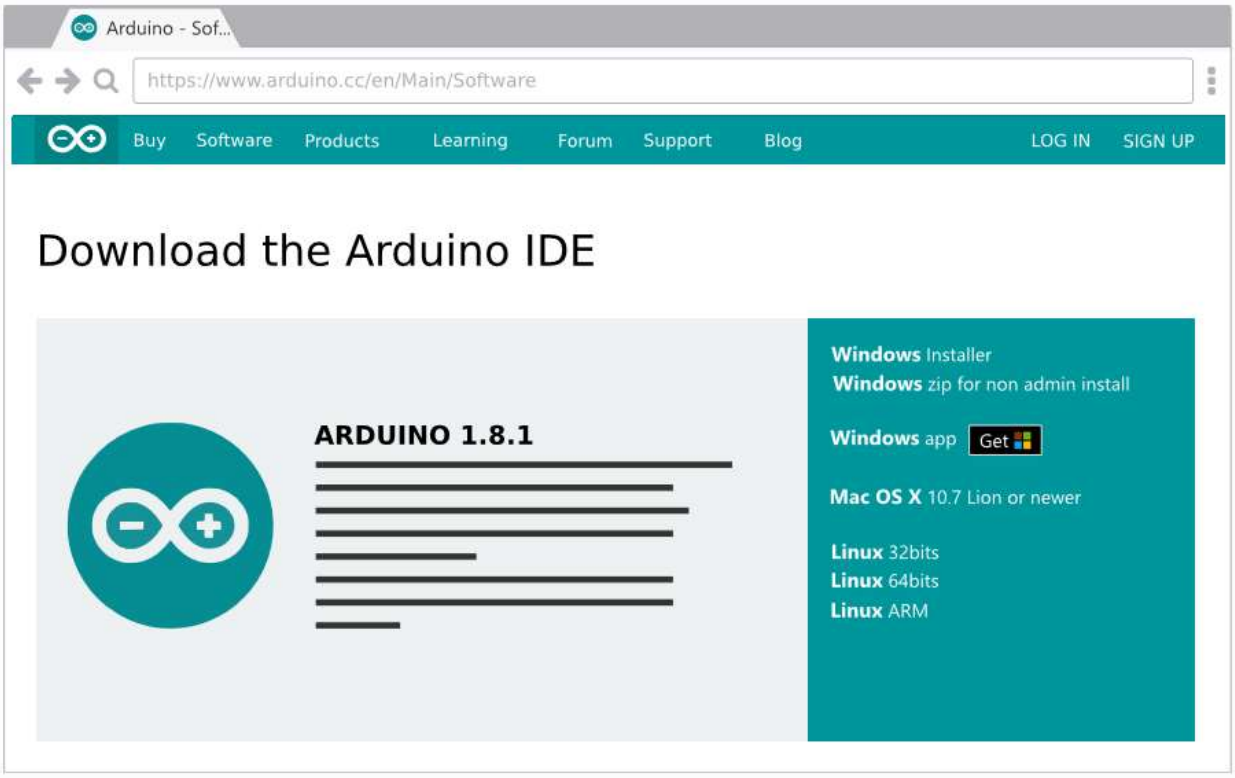
Arduino is best known for its hardware, but you also need software to program that hardware. Both the hardware and the software are called "Arduino." The combination enables you to create projects that sense and control the physical world. The software is free, open source, and cross-platform. The boards are inexpensive to buy, or you can build your own (the hardware designs are also open source). In addition, there is an active and supportive Arduino community that is accessible worldwide through the Arduino forums and the wiki (known as the Arduino Playground).

These boards are equipped with sets of digital and analog input/output (I/O) pins that may be interfaced to various expansion boards ('shields') or breadboards (For prototyping) and other circuits. The boards feature serial communications interfaces, including Universal Serial Bus (USB) on some models, which are also used for loading programs from personal computers. The microcontrollers can be programmed using C and C++ programming languages.

A quick scan through this book will show you that you can use the Arduino to do something as simple as blinking a small light, or even something more complicated and many different things in between.

1.2 Arduino – Software

Software programs, called sketches, are created on a computer using the Arduino integrated development environment (IDE). The IDE enables you to write and edit code and convert this code into instructions that Arduino hardware understands. The IDE also transfers those instructions to the Arduino board (a process called uploading).

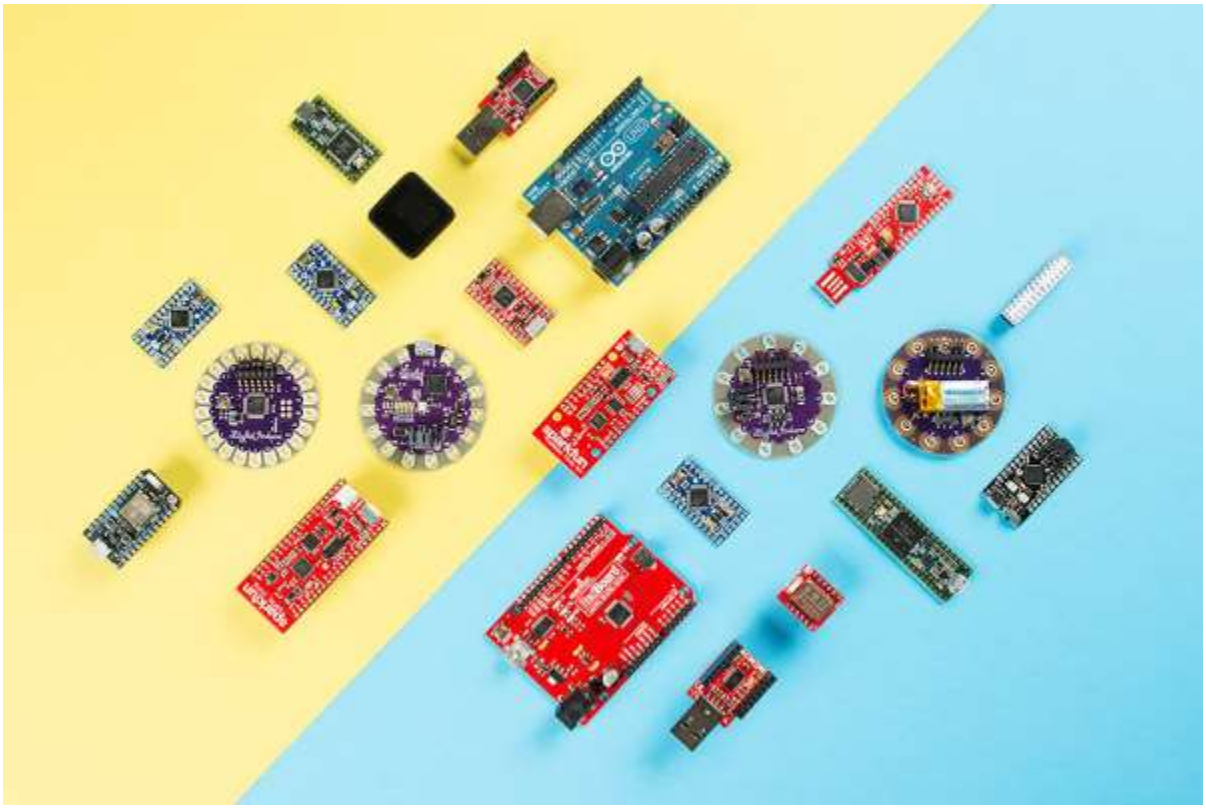


The Arduino IDE can be downloaded from the official Arduino website for free: https://www.arduino.cc/en/main/software. Moreover, you can also use Arduino's online editor to write your sketches: https://create.arduino.cc/editor.

In case you find any issue while installing the software, you can refer to the following website: https://www.arduino.cc/en/Guide/HomePage. Choose the relevant operating system and go through the installation guide.

1.3 Arduino – Hardware

The Arduino board is where the code you write is executed. The board can only control and respond to electricity, so specific components are attached to it to enable it to interact with the real world. These components can be sensors, which convert some aspect of the physical world to electricity so that the board can sense it, or actuators, which get electricity from the board and convert it into something that changes the world. Examples of sensors include switches, accelerometers, and ultrasound distance sensors. Actuators are things like lights and LEDs, speakers, motors, and displays.



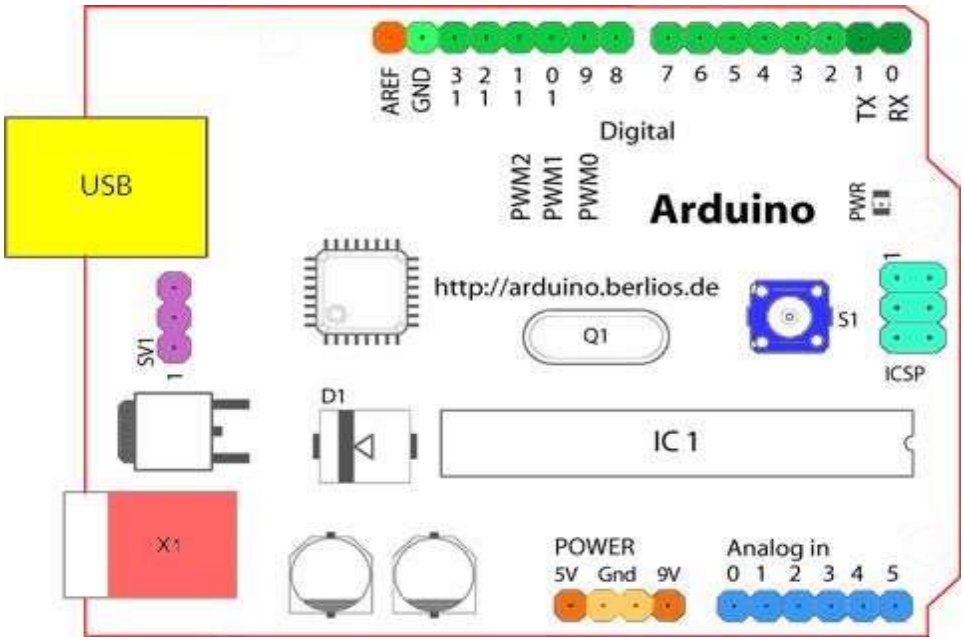
Chapter 2
Understanding Arduino Board and IDE

2.1 Arduino Board

It is an open-source electronics prototyping platform based on flexible, easy-to-use hardware and software. It's intended for artists, designers, hobbyists, and anyone interested in creating interactive objects or environments. In simple terms, the Arduino is a tiny computer system that can be programmed with your instructions to interact with various forms of input and output.

Although it might not look like much to the new observer, the Arduino system allows you to create devices that can interact with the world around you. By using an almost unlimited range of input and output devices, sensors, indicators, displays, motors, and more, you can program the exact interactions required to create a functional device. For example, artists have created installations with patterns of blinking lights that respond to the movements of passers-by, high school students have built autonomous robots that can detect an open flame and extinguish it, and geographers have designed systems that monitor temperature and humidity and transmit this data back to their offices via text message. In fact, you'll find an almost infinite number of examples with a quick search on the Internet.

You can get boards as small as a postage stamp, such as the Arduino Mini and Pro Mini; larger boards that have more connection options and more powerful processors, such as the Arduino Mega; and boards tailored for specific applications, such as the LilyPad for wearable applications, the Fio for wireless projects, and the Arduino Pro for embedded applications (standalone projects that are often battery-operated).



Starting clockwise from the top center:

1. Analog Reference pin (orange)
2. Digital Ground (light green)
3. Digital Pins 2-13 (green)
4. Digital Pins 0-1/Serial In/Out - TX/RX (dark green) - These pins cannot be used for digital i/o (digitalRead and digitalWrite) if you are also using serial communication.
5. Reset Button - S1 (dark blue)
6. In-circuit Serial Programmer (blue-green)
7. Analog In Pins 0-5 (light blue)
8. Power and Ground Pins (power: orange, grounds: light orange)
9. External Power Supply In (9-12VDC) - X1 (pink)
10. Toggles External Power and USB Power (place jumper on two pins closest to desired supply) - SV1 (purple)
11. USB (used for uploading sketches to the board and for serial communication between the board and the computer; can be used to power the board) (yellow)

2.1.1 Theory

Let's take a quick tour of the Uno. Starting at the left side of the board, you'll see two connectors. On the far left is the Universal Serial Bus (USB) connector. This connects the board to your computer for three reasons: to supply power to the board, to upload your instructions to the Arduino, and to send data to and receive it from a computer. On the right is the power connector. Through this connector, you can power the Arduino with a standard mains power adapter. At the lower middle is the heart of the board: the microcontroller

The microcontroller is the "brains" of the Arduino. It is a tiny computer that contains a processor to execute instructions, includes various types of memory to hold data and instructions from our sketches, and provides various avenues of sending and receiving data. Just below the microcontroller are two rows of small sockets, the first row offers power connections and the ability to use an external RESET button. The second row offers six analog inputs that are used to measure electrical signals that vary in voltage. Furthermore, pins A4 and A5 can also be used for sending data to and receiving it from other devices.

The Arduino board has four LEDs: one on the far right labelled ON, which indicates when the board has power, and three in another group. The LEDs labelled TX and RX light up when data is being transmitted or received between the Arduino and attached devices via the serial port and USB. The little black square part to the left of the LEDs is a tiny microcontroller that controls the USB interface that allows your Arduino to send data to and receive it from a computer, but you don't generally have to concern yourself with it.

As with a normal computer, sometimes things can go wrong with the Arduino, and when all else fails, you might need to reset the system and restart your Arduino. This simple RESET button on the board is used to restart the system to resolve these problems.

2.1.2 Digital Pins

In addition to the specific functions listed below, the digital pins on an Arduino board can be used for general purpose input and output via the pinMode(), digitalRead(), and digitalWrite() commands. Each pin has an internal pull-up resistor which can be turned on and off using digitalWrite() (with a value of HIGH or LOW, respectively) when the pin is configured as an input. The maximum current per pin is 40 mA.

2.1.3 Analog Pins

In addition to the specific functions listed below, the analog input pins support 10-bit analog-to-digital conversion (ADC) using the analogRead() function. Most of the analog inputs can also be used as digital pins: analog input 0 as digital pin 14 through analog input 5 as digital pin 19. Analog inputs 6 and 7 (present on the Mini and BT) cannot be used as digital pins.

2.1.4 Power Pins

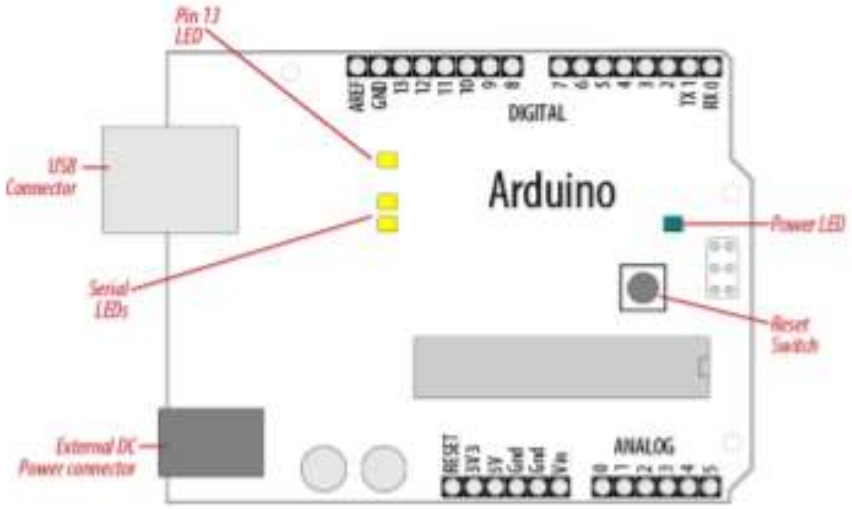
1. VIN (sometimes labelled "9V"): The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin. Note that different boards accept different input voltages ranges, please see the documentation for your board. Also note that the LilyPad has no VIN pin and accepts only a regulated input.
2. 5V: The regulated power supply used to power the microcontroller and other components on the board. This can come either from VIN via an on-board regulator, or be supplied by USB or another regulated 5V supply.
3. 3V3: (Diecimila-only) A 3.3 volt supply generated by the on-board FTDI chip.
4. GND: Ground pins.

2.1.5 Communication Peripherals and Miscellaneous

1. Serial: 0 (RX) and 1 (TX). Used to receive (RX) and transmit (TX) TTL serial data. On the Arduino Diecimila, these pins are connected to the corresponding pins of the FTDI USB-to-TTL Serial chip. On the Arduino BT, they are connected to the corresponding pins of the WT11 Bluetooth module. On the Arduino Mini and LilyPad Arduino, they are intended for use with an external TTL serial module (e.g. the Mini-USB Adapter).
2. SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). These pins support SPI communication, which, although provided by the underlying hardware, is not currently included in the Arduino language.
3. I2C: 4 (SDA) and 5 (SCL). Support I2C (TWI) communication using the Wire library (documentation on the Wiring website).
4. External Interrupts: 2 and 3. These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt() function for details.
5. PWM: 3, 5, 6, 9, 10, and 11. Provide 8-bit PWM output with the analogWrite() function. On boards with an ATmega8, PWM output is available only on pins 9, 10, and 11.
6. BT Reset: 7. (Arduino BT-only) Connected to the reset line of the bluetooth module.
7. LED: 13. On the Diecimila and LilyPad, there is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.

2.2 Setting Up the Arduino Board

Plug the board into a USB port on your computer and check that the green LED power indicator on the board illuminates. Standard Arduino boards (Uno, Duemilanove, and Mega) have a green LED power indicator located near the reset switch. An orange LED near the center of the board (labeled “Pin 13 LED”) should flash on and off when the board is powered up (boards come from the factory preloaded with software to flash the LED as a simple check that the board is working).

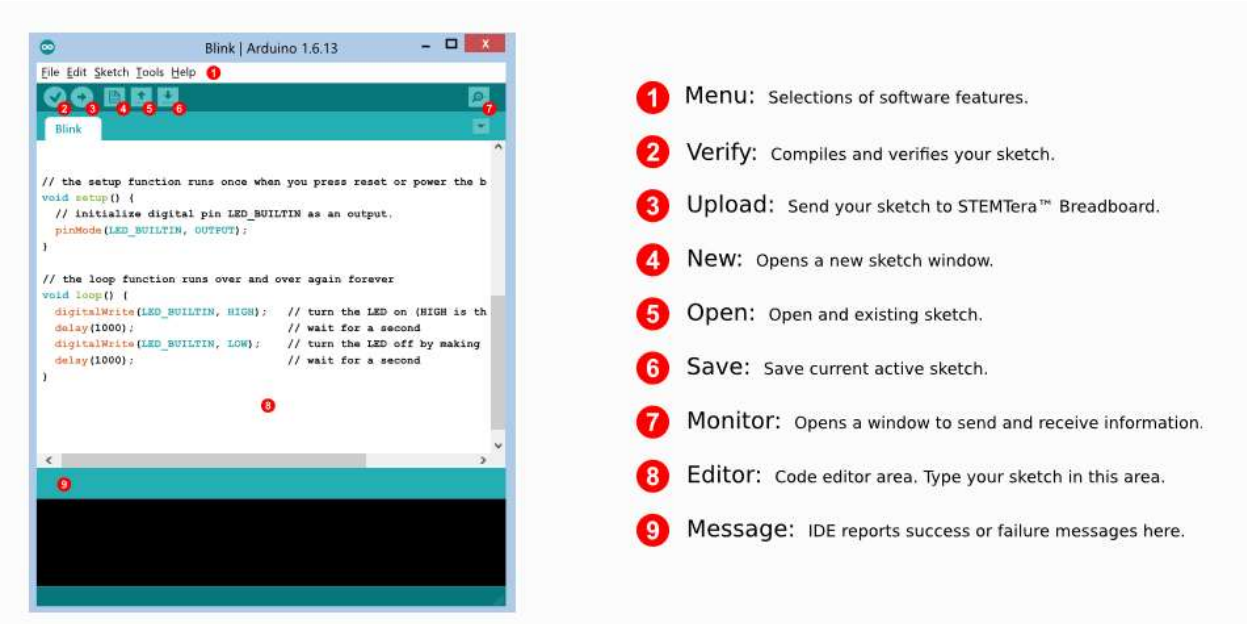


Online guides for getting started with Arduino are available at http://arduino.cc/en/Guide/Windows for Windows, http://arduino.cc/en/Guide/MacOSX for Mac OS X, and http://www.arduino.cc/playground/Learning/Linux for Linux.

A troubleshooting guide can be found at http://arduino.cc/en/Guide/Troubleshooting.

2.3 Understanding the IDE

Use the Arduino IDE to create, open, and modify sketches that define what the board will do. You can use buttons along the top of the IDE to perform these actions, or you can use the menus or keyboard shortcuts. The Sketch Editor area is where you view and edit code for a sketch. It supports common text editing keys such as Ctrl-F (⌘+F on a Mac) for find, Ctrl-Z (⌘+Z on a Mac) for undo, Ctrl-C (⌘+C on a Mac) to copy highlighted text, and Ctrl-V (⌘+V on a Mac) to paste highlighted text.



2.3.1 The Command Area

The command area includes the title bar, menu items, and icons. The title bar displays the sketch's filename (sketch_mar22a), as well as the version of the IDE. Below this is a series of menu items (File, Edit, Sketch, Tools, and Help) and icons.

2.3.2 Menu Items

As with any word processor or text editor, you can click one of the menu items to display its various options.

[1] File. Contains options to save, load, and print sketches; a thorough set of example sketches to open; as well as the Preferences submenu

[2] Edit. Contains the usual copy, paste, and search functions common to any word processor

[3] Sketch. Contains the function to verify your sketch before uploading to a board, and some sketch folder and import options

[4] Tools. Contains a variety of functions as well as the commands to select the Arduino board type and USB port

[5] Help. Contains links to various topics of interest and the version of the IDE

2.3.3 The Icons

Below the menu toolbar are six icons. Mouse over each icon to display its name. The icons, from left to right, are as follows:

1. Verify. Click this to check that the Arduino sketch is valid and doesn't contain any programming mistakes.
2. Upload. Click this to verify and then upload your sketch to the Arduino board.
3. New. Click this to open a new blank sketch in a new window.
4. Open. Click this to open a saved sketch.
5. Save. Click this to save the open sketch. If the sketch doesn't have a name, you will be prompted to create one.
6. Serial Monitor. Click this to open a new window for use in sending and receiving data between your Arduino and the IDE.

2.3.4 The Text Area

The text area is where you'll create your sketches. The name of the current sketch is displayed in the tab at the upper left of the text area. (The default name is the current date.) You'll enter the contents of your sketch here as you would in any text editor.

Chapter 3
Syntax and Libraries

3.1 Syntax

The coding language that Arduino uses is very much like C++, which is a common language in the world of computing. The code you learn to write for Arduino will be very similar to the code you write in any other computer language – all the basic concepts remain the same – it is just a matter of learning a new dialect should you pursue other programming languages.

An Arduino program run in two parts:

1. void setup()

2. void loop()

Where, setup() is preparation, and loop() is execution. They are explained with example ahead:

setup( )

The setup( ) function is called when your program starts. Use it to initialize your variables, pin modes, start using libraries, etc. The setup function will only run once, after each powerup or reset of the Arduino board.

Example:

```
int buttonPin = 3;
void setup()
{
  Serial.begin(9600);
  pinMode(buttonPin, INPUT);
}
void loop()
{
  // ...
}
```

loop( )

After creating a setup( ) function, which initializes and sets the initial values, the loop( ) function does precisely what its name suggests, and loops consecutively, allowing your program to change and respond. Use it to actively control the Arduino board.

Example

```
int buttonPin = 3;
// setup initializes serial and the button pin
void setup( )
{
  Serial.begin(9600);
  pinMode(buttonPin, INPUT);
}
// loop checks the button pin each time
// and will send serial if it is pressed
void loop( )
{
  if (digitalRead(buttonPin) == HIGH)
  serialWrite('H');
  else   serialWrite('L');
  delay(1000);
}
```

3.1.1 Functions:

Table 1: Digital I/O Functions

Syntax	Functions
digitalWrite(pin, value)	Sets a pin configured as OUTPUT to either a HIGH or a LOW state at the specified pin.
digitalRead(pin)	Reads the value from a specified pin, it will be either HIGH or LOW.
pinMode(pin, mode)	Configures the specified pin to behave either as an input or an output. See the reference page below.

Table 2: Analog I/O Functions

Syntax	Functions
analogWrite(pin, value)	Writes an analog value (PWM wave*) to a pin. Can be used to light a LED at varying brightness or drive a motor at various speeds.
analogRead(pin)	Reads the value from the specified analog pin. The 10-bit Analog-to-Digital Converter (ADC) will map input voltages between 0 and 5 volts into integer values between 0 and 1023.

*Please refer to: https://www.arduino.cc/en/Tutorial/PWM to know more about PWM.

Table 3: Time Functions

Syntax	Functions
delay(ms)	Pauses your program for the amount of time (in milliseconds) specified as parameter.
delayMicroseconds(us)	Pauses the program for the amount of time (in microseconds) specified as parameter.*
millis()	Returns the number of milliseconds since the Arduino board began running the current program.

*For delays longer than a few thousand microseconds, you should use delay() instead.

3.1.2 Serial Monitor Commands:

Syntax	Function
Serial.begin( )	To begin the Serial Monitor
Serial.read( )	To read data from Serial Monitor
Serial.available( )	To check if Serial Monitor
Serial.print( )	To Print data in Serial Monitor
Serial.println( )	To Print data in the next line in Serial Monitor
Serial.write( )	To write data on another device connected to Arduino from Serial Monitor

3.1.3 Math Functions

1. min(x, y)
2. max(x, y)
3. abs(x)
4. constrain(x, a, b)
5. map(value, fromLow, fromHigh, toLow, toHigh)
6. pow(base, exponent)
7. sqrt(x)
8. sin(rad)
9. cos(rad)
10. tan(rad)

3.2 Variables

Variables are expressions that you can use in programs to store values, such as a sensor reading from an analog pin.

3.2.1 Constants

Constants are particular values with specific meanings.

1. HIGH | LOW
2. INPUT | OUTPUT
3. true | false
4. Integer Constants

3.2.2 Data Types

Variables can have various types, which are described below.

1. boolean
2. long
3. char
4. float
5. byte
6. double
7. int
8. string
9. unsigned int
10. array

3.3 Further Syntax

Syntax	Function
; (semicolon)	Used to end a statement
{} (curly braces)	Curly braces (also referred to as just "braces" or as "curly brackets") are a major part of the C programming language. They are used in several different constructs, outlined below, and this can sometimes be confusing for beginners.
// (single line comment) or /* */ (multi-line comment)	Comments are lines in the program that are used to inform yourself or others about the way the program works. They are ignored by the compiler, and not exported to the processor, so they don't take up any space on the Atmega chip.
#define	#define is a useful C component that allows you to give a name to a constant value before the program is compiled. The compiler will replace references to these constants with the defined value at compile time.
#include	#include is used to include outside libraries in your sketch. This gives the programmer access to a large group of standard C libraries (groups of pre-made functions), and also libraries written especially for Arduino.

3.4 Libraries

Libraries add functionality to the Arduino environment. They extend the commands available to provide capabilities not available in the core Arduino language. Libraries provide a way to add features that can be accessed from any of your sketches once you have installed the library.

Libraries are also a good way for people to share code that may be useful to others. Many third-party libraries provide specialized capabilities; these can be downloaded from the Arduino Playground and other sites. Libraries are often written to simplify the use of a particular piece of hardware. Many of the devices covered in earlier chapters use libraries to make it easier to connect to the devices.

To see the list of available libraries from the IDE menu, click Sketch→Import Library. A list will drop down showing all the available libraries. The first dozen or so are the libraries distributed with Arduino. A horizontal line separates that list from the libraries that you download and install yourself.

Clicking on a library will add that library to the current sketch, by adding the following line to the top of the sketch: **#include <nameOfTheLibrarySelected.h>**. This results in the functions within the library becoming available to use in your sketch. The Arduino libraries are documented in the reference at http://arduino.cc/en/Reference/Libraries and each library includes example sketches demonstrating their use.

Here is a comprehensive list of libraries available on Arduino IDE by default:

1. EEPROM: Used to store and read information in memory that is preserved when power is removed.
2. Ethernet: Used to communicate with the Arduino Ethernet shield.
3. Firmata: A protocol used to simplify serial communication and control of the board.
4. LiquidCrystal: For controlling compatible LCD displays

5. Matrix: Helps manage a matrix of LEDs
6. SD: Supports reading and writing files to an SD card using external hardware
7. Servo: Used to control servo motors
8. SoftwareSerial: Enables additional serial ports
9. SPI: Used for Ethernet and SPI hardware
10. Sprite: Enables the use of sprites with an LED matrix
11. Stepper: For working with stepper motors
12. Wire: Works with I2C devices attached to the Arduino

Chapter 4
First Sketch

4.1 Creating Your First Sketch in the IDE

An Arduino sketch is a set of instructions that you create to accomplish a task; in other words, a sketch is a program. In this section you'll create and upload a simple sketch that will cause the Arduino's LED to blink repeatedly, by turning it on and then off for 1 second intervals.

To begin, connect your Arduino to the computer with the USB cable. Then open the IDE, choose Tools -> Serial Port, and make sure the USB port is selected. This ensures that the Arduino board is properly connected.

4.1.1 Comments

First, enter a comment as a reminder of what your sketch will be used for. A comment is a note of any length in a sketch, written for the user's benefit. Comments in sketches are useful for adding notes to yourself or others, for entering instructions, or for noting miscellaneous details. When programming your Arduino (creating sketches), it's a good idea to add comments regarding your intentions; these comments can prove useful later when you're revisiting a sketch. To add a comment on a single line, enter two forward slashes and then the comment, like this: // Blink LED sketch by Tejas Shah, created 09/09/19

The two forward slashes tell the IDE to ignore the text that follows when verifying a sketch. (As mentioned earlier, when you verify a sketch, you're asking the IDE to check that everything is written properly with no errors.)

To enter a comment that spans two or more lines, enter the characters /* on a line before the comment, and then end the comment with the characters */ on the following line, like this:

```
/*
        Arduino Blink LED Sketch by Tejas Shah, created 09/09/19
*/
```

As with the two forward slashes that precede a single line comment, the /* and */ tell the IDE to ignore the text that they bracket.

Enter a comment describing your Arduino sketch using one of these methods, and then save your sketch by choosing File -> Save As. Enter a short name for your sketch (such as blinky), and then click OK. The default filename extension for Arduino sketches is ".ino", and the IDE should add this automatically. The name for your sketch should be, in this case, "blinky.ino", and you should be able to see it in your Sketchbook.

4.1.2 The Setup Function

The next stage in creating any sketch is to add the "void setup()" function. This function contains a set of instructions for the Arduino to execute once only, each time it is reset or turned on. To create the setup function, add the following lines to your sketch, after the comments:

```
void setup()
{

}
```

4.1.3 Controlling the Hardware

Our program will blink the user LED on the Arduino. The user LED is connected to the Arduino's digital pin 13. A digital pin can either detect an electrical signal or generate one on command. In this project, we'll generate an electrical signal that will light the LED.

Enter the following into your sketch between the braces ({ and }):

```
pinMode(13, OUTPUT); // set digital pin 13 to output
```

The number 13 in the listing represents the digital pin you're addressing. You're setting this pin to OUTPUT, which means it will generate (output) an electrical signal. If you wanted it to detect an incoming electrical signal, then you would use INPUT instead. Notice that the function "pinMode()" ends with a semicolon (;). Every function in your Arduino sketches will end with a semicolon. Save your sketch again to make sure that you don't lose any of your work.

4.1.4 The Loop Function

Remember that our goal is to make the LED blink repeatedly. To do this, we'll create a loop function to tell the Arduino to execute an instruction over and over until the power is shut off or someone presses the RESET button.

Enter the code shown in boldface after the void setup() section in the following listing to create an empty loop function. Be sure to end this new section with another brace (}), and then save your sketch again.

```
/*
Arduino Blink LED Sketch by Tejas Shah, created 09/09/19
*/

void setup()
{
  pinMode(13, OUTPUT); // set digital pin 13 to output
} void loop()
{
  // place your main loop code here:
}
```

Next, enter the actual functions into void loop() for the Arduino to execute. Enter the following between the loop function's braces, and then click Verify to make sure that you've entered everything correctly:

```
 digitalWrite(13, HIGH); // turn on digital pin 13 delay(1000); // pause for one second
digitalWrite(13, LOW); // turn off digital pin 13 delay(1000); // pause for one second
```

The "digitalWrite()" function controls the voltage that is output from a digital pin: in this case, pin 13 to the LED. By setting the second parameter of this function to HIGH, a "high" digital voltage is output; then current will flow from the pin and the LED will turn on. (If you were to set this parameter to LOW, then the current flowing through the LED would stop). With the LED turned on, the light pauses for 1 second with delay(1000). The delay() function causes the sketch to do nothing for a period of time—in this case, 1,000 milliseconds, or 1 second.

Next, we turn off the voltage to the LED with digitalWrite(13, LOW);. Finally, we pause again for 1 second while the LED is off, with delay(1000);.The completed sketch should look like this:

```
/*
 Arduino Blink LED Sketch  by Tejas Shah, created 09/09/19
 */

void setup()
{
  pinMode(13, OUTPUT); // set digital pin 13 to output
}

void loop()
{  digitalWrite(13, HIGH); // turn on digital pin 13   delay(1000); // pause for one second
digitalWrite(13, LOW); // turn off digital pin 13   delay(1000); // pause for one second
}
```

4.2 Verifying Your Sketch

When you verify your sketch, you ensure that it has been written correctly in a way that the Arduino can understand. To verify your complete sketch, click Verify in the IDE and wait a moment. Once the sketch has been verified, a note should appear in the message window. The “Done compiling” message tells you that the sketch is okay to upload to your Arduino. It also shows how much memory it will use (1,076 bytes in this case) of the total available on the Arduino (32,256 bytes).

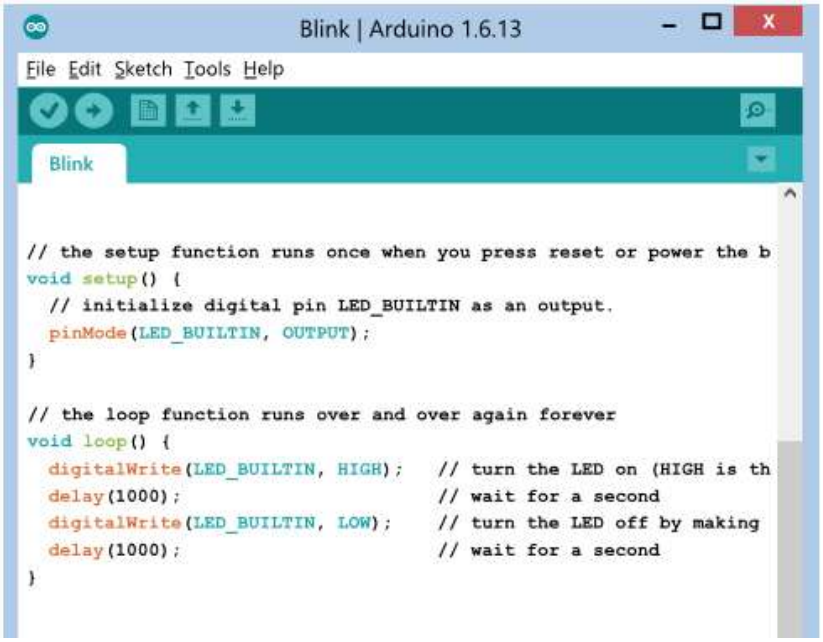
But what if your sketch isn’t okay? Say, for example, you forgot to add a semicolon at the end of the second delay(1000) function. If something is broken in your sketch, then when you click Verify, the message window should display a verification error message.

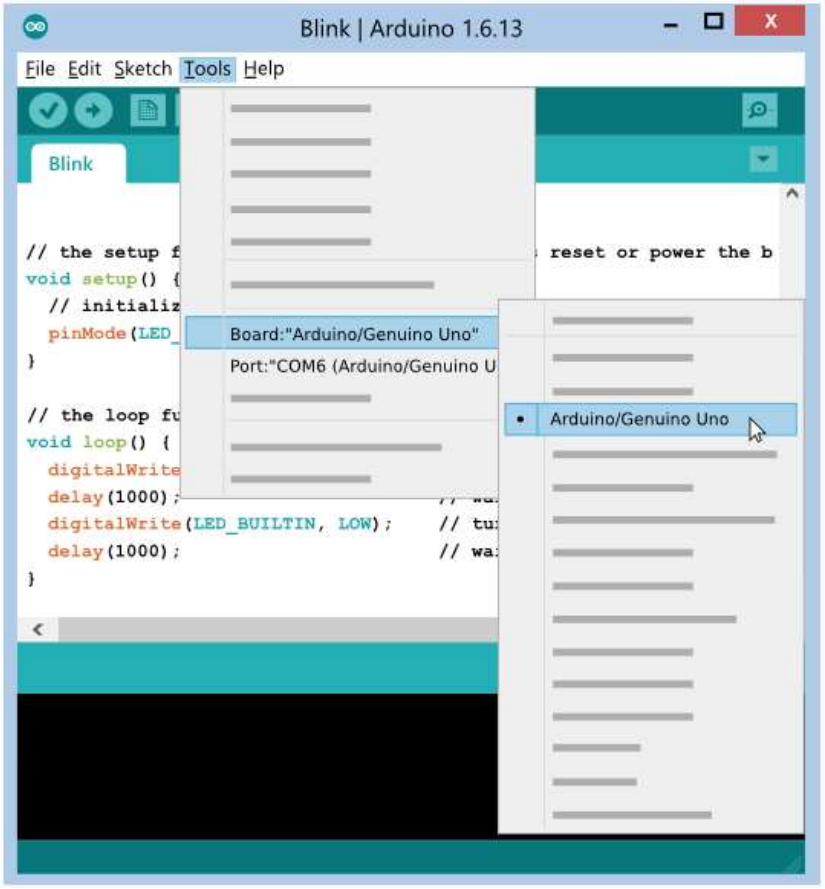
The message tells you that the error occurs in the void loop function, lists the line number of the sketch where the IDE thinks the error is located (blinky:16, or line 16 of your blinky sketch), and displays the error itself (the missing semicolon, error: expected ';' before '}' token). Furthermore, the IDE should also highlight in yellow the location of the error or a spot just after it. This helps you easily locate and rectify the mistake.

4.2.1 Uploading and Running Your Sketch

Once you’re satisfied that your sketch has been entered correctly, save it, ensure that your Arduino board is connected, and click Upload in the IDE. The IDE may verify your sketch again and then upload it to your Arduino board. During this process, the TX/RX LEDs on your board should blink, indicating that information is traveling between the Arduino and your computer.

Now for the moment of truth: Your Arduino should start running the sketch. If you’ve done everything correctly, then the LED should blink on and off once every second!


Blink | Arduino 1.6.13
File Edit Sketch Tools Help
Blink
// the setup function runs once when you press reset or power the b
void setup() {
// initialize digital pin LED_BUILTIN as an output.
pinMode(LED_BUILTIN, OUTPUT);
}
// the loop function runs over and over again forever
void loop() {
digitalWrite(LED_BUILTIN, HIGH); // turn the LED on (HIGH is th
delay(1000); // wait for a second
digitalWrite(LED_BUILTIN, LOW); // turn the LED off by making
delay(1000); // wait for a second
}


Blink | Arduino 1.6.13
File Edit Sketch Tools Help
Blink
Board:"Arduino/Genuino Uno"
Port:"COM6 (Arduino/Genuino U
Arduino/Genuino Uno

4.2.2 Modifying Your Sketch

After running your sketch, you may want to change how it operates, by, for example, adjusting the on or off delay time for the LED. Because the IDE is a lot like a word processor, you can open your saved sketch, adjust the values, and then save your sketch again and upload it to the Arduino. For example, to increase the rate of blinking, change both delay functions to make the LEDs blink for one-quarter of a second by adjusting the delay to 250 like this:

```
delay(250); // pause for one-quarter of one second
```

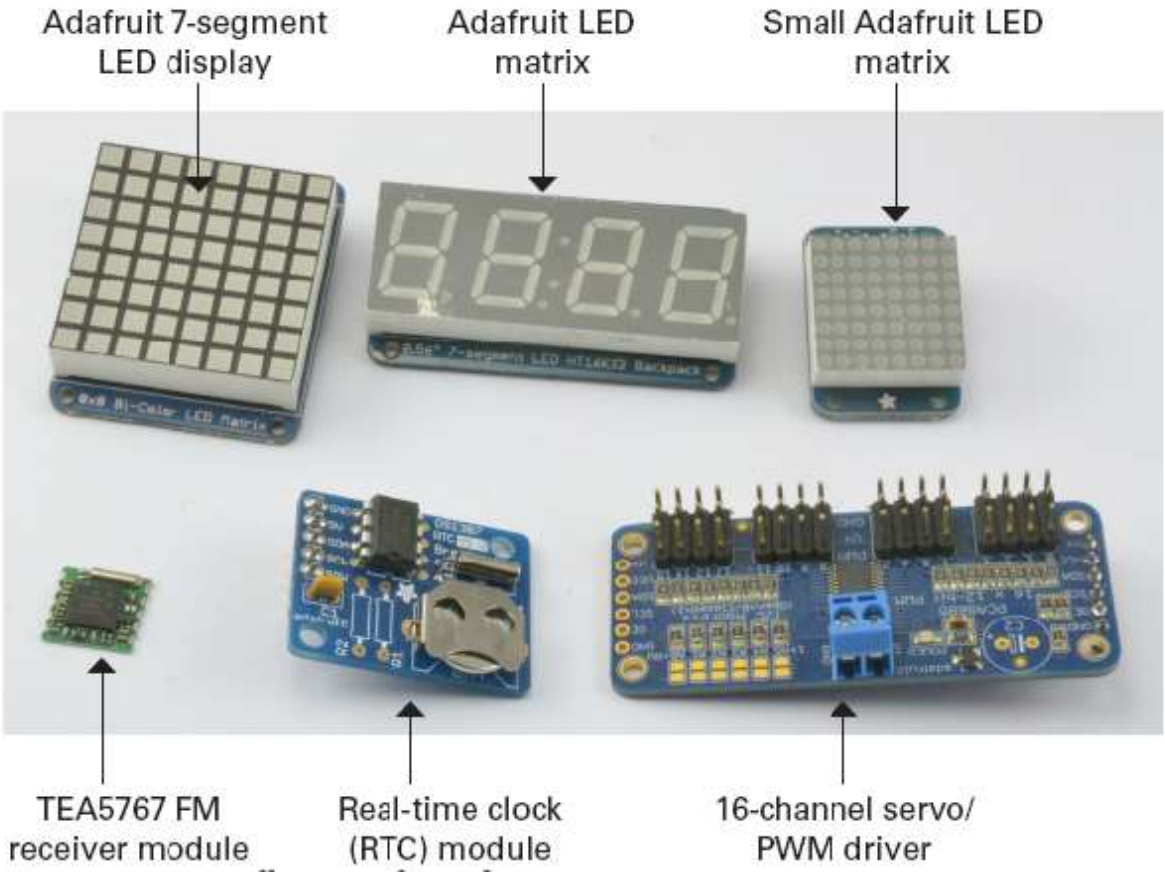
Then upload the sketch again. The LED should now blink faster, for one-quarter of a second each time.

Chapter 5
Communication Protocols

Embedded electronics is all about interlinking circuits (processors or other integrated circuits) to create a symbiotic system. In order for those individual circuits to swap their information, they must share a common communication protocol. Hundreds of communication protocols have been defined to achieve this data exchange, and, in general, each can be separated into one of two categories: parallel or serial.

5.1 I2C Protocol

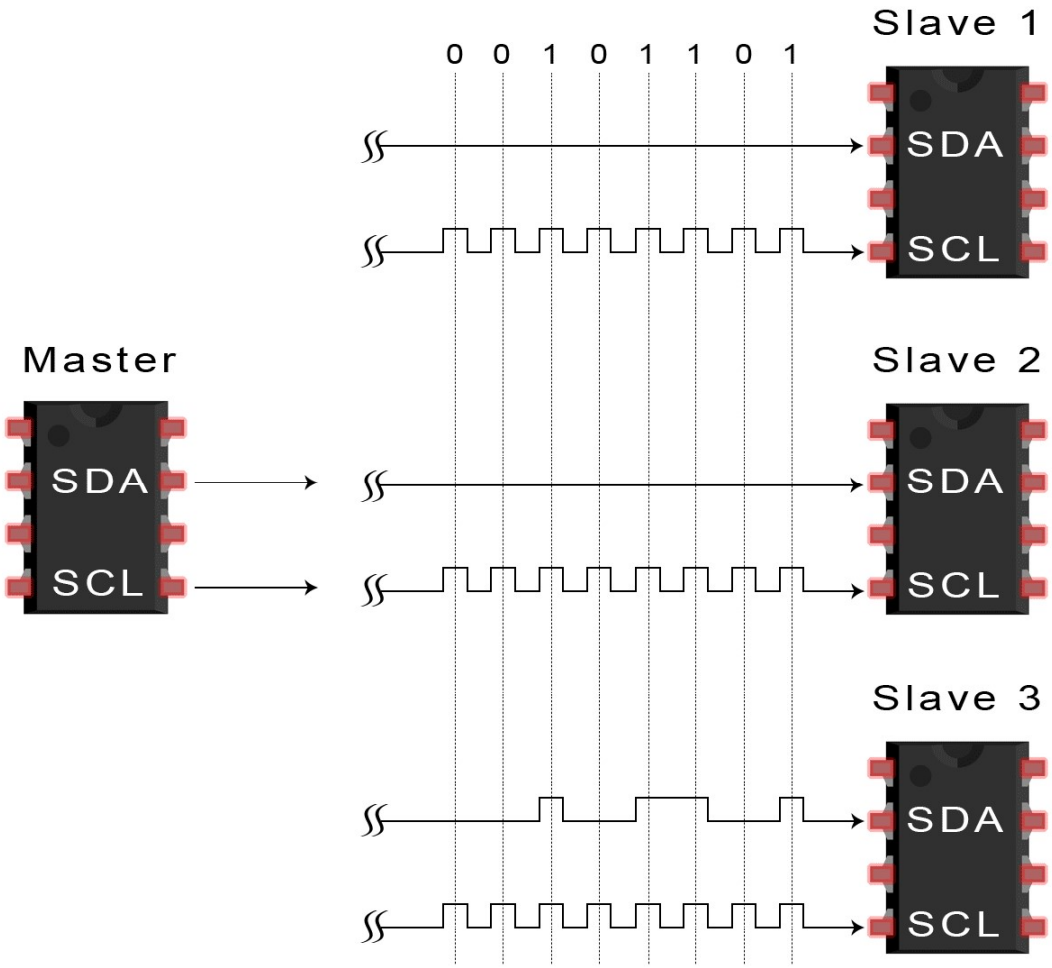
The I2C (pronounced “I squared C”) interface bus is a standard for connecting microcontrollers and peripherals together. I2C is sometimes referred to as Two Wire Interface (TWI). All the Arduino boards have at least one I2C interface to which you can attach a wide range of peripherals.



The I2C standard is defined as a “bus” standard because its use is not limited to connecting one component directly to another. Say you have a display connected to a microcontroller; using the same two bus pins, you can connect a whole set of “slave” devices to a “master” device. The Arduino acts as the “master,” and each of the “slaves” has a unique address that identifies the device on the bus. You can also use I2C to connect two Arduinos together so they can exchange data. In this case, one of the Arduinos will be configured to act as a “master” and one as a “slave.”

5.1.1 Timing Diagram

I2C uses two wires to transmit and receive data (hence, the alternative name of Two Wire Interface). These two lines are called the Serial Clock Line (SCL) and the Serial Data Line (SDA). The master supplies the SCL clock, and when there is data to be transmitted, the sender (master or slave) takes the SDA line out of tri-state (digital input mode) and sends data as logic highs or lows in time with the clock signal.



5.1.2 The Wire Library

You could, of course, generate these pulses yourself by bit banging—that is, turning digital outputs on and off in your code. To make life easier for us, however, the Arduino software includes a library called Wire that handles all the timing complexity, so we can just send and receive bytes of data.

To use the Wire library, you first need to include it using the following command:

```
#include <Wire.h>
```

In most situations, an Arduino is the “master” in any I2C bus. To initialize an Arduino as the master, use the begin command in your setup function, as shown here:

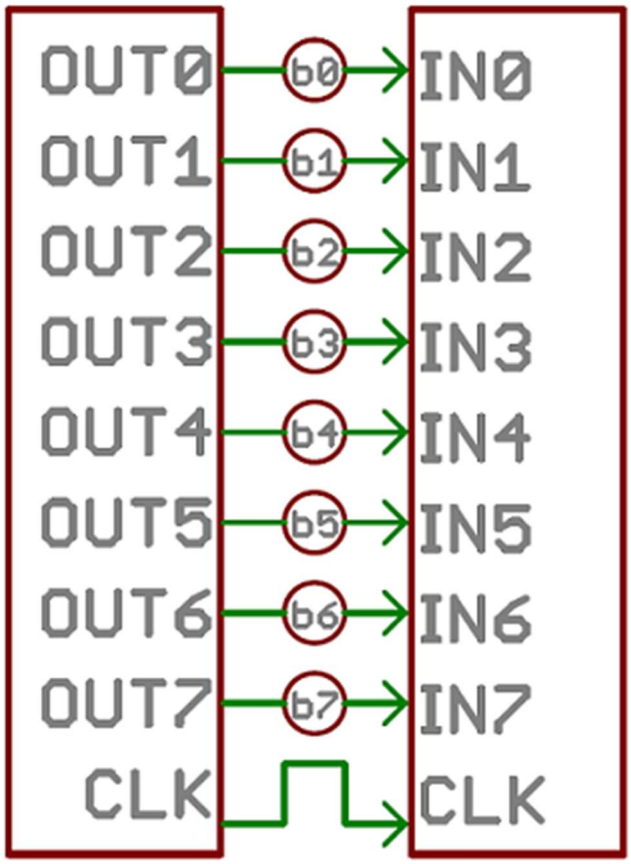
```
void main( )
{
        Wire.begin( );
}
```

Note that because the Arduino is the master in this arrangement, you don’t need to specify an address. If the Arduino were being initialized as a slave, then you would need to specify an address, 0 to 127, as its parameter to uniquely identify it on the I2C bus.

5.2 Serial Communication

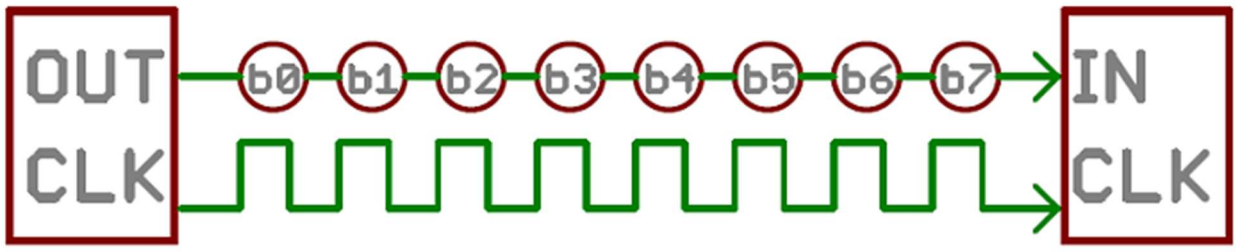
5.2.1 Parallel vs. Serial

Parallel interfaces transfer multiple bits at the same time. They usually require **buses** of data - transmitting across eight, sixteen, or more wires. Data is transferred in huge, crashing waves of 1's and 0's.



An 8-bit data bus, controlled by a clock, transmitting a byte every clock pulse. 9 wires are used.

Serial interfaces stream their data, one single bit at a time. These interfaces can operate on as little as one wire, usually never more than four.



Example of a serial interface, transmitting one bit every clock pulse. Just 2 wires required!

Think of the two interfaces as a stream of cars: a parallel interface would be the 8+ lane mega-highway, while a serial interface is more like a two-lane rural country road. Over a set amount of time, the mega-highway potentially gets more people to their destinations, but that rural two-laner serves its purpose and costs a fraction of the funds to build.

Parallel communication certainly has its benefits. It's fast, straightforward, and relatively easy to implement. But it requires many more input/output (I/O) lines. If you've ever had to move a project from a basic Arduino Uno to a Mega, you know that the I/O lines on a microprocessor can be precious and few. So, we often opt for serial communication, sacrificing potential speed for pin real estate.

5.2.2 Asynchronous Serial

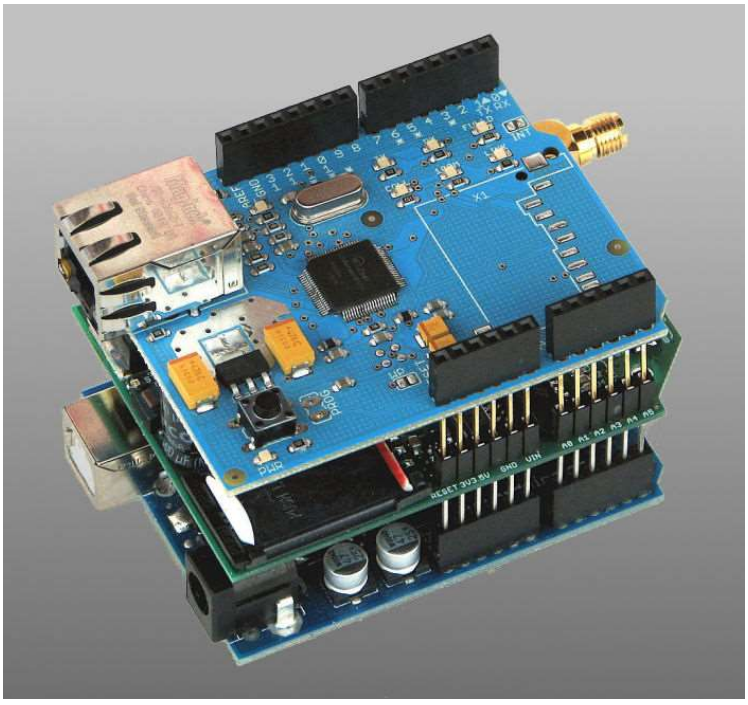
Over the years, dozens of serial protocols have been crafted to meet particular needs of embedded systems. USB (universal serial bus), and Ethernet, are a couple of the more well-known computing serial interfaces. Other very common serial interfaces include SPI, I2C, and the serial standard we're here to talk about today. Each of these serial interfaces can be sorted into one of two groups: synchronous or asynchronous.

A synchronous serial interface always pairs its data line(s) with a clock signal, so all devices on a synchronous serial bus share a common clock. This makes for a more straightforward, often faster serial transfer, but it also requires at least one extra wire between communicating devices. Examples of synchronous interfaces include SPI, and I2C.

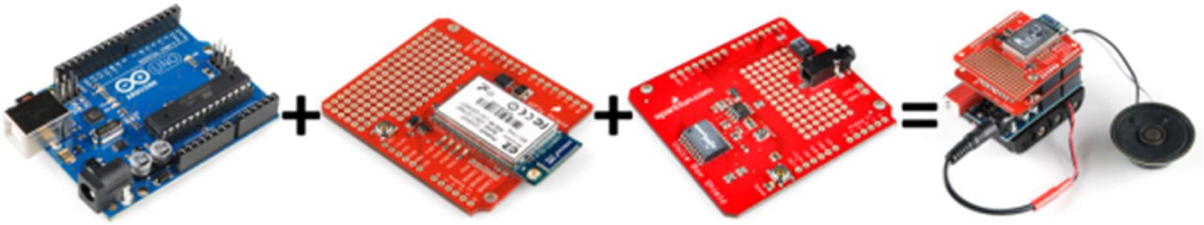
Asynchronous means that data is transferred without support from an external clock signal. This transmission method is perfect for minimizing the required wires and I/O pins, but it does mean we need to put some extra effort into reliably transferring and receiving data. The serial protocol we'll be discussing in this tutorial is the most common form of asynchronous transfers. It is so common, in fact, that when most folks say “serial” they’re talking about this protocol (something you’ll probably notice throughout this tutorial).

Chapter 6
Arduino Shields

Shields are modular circuit boards that piggyback onto your Arduino to instill it with extra functionality. Want to connect your Arduino to the Internet and post to Twitter? There's a shield for that. Want to make your Arduino an autonomous rover? There are shields for that. There are dozens (hundreds?) of shields out there, all of which make your Arduino more than just a development board with a blinky LED.



Many Arduino shields are stackable. You can connect many shields together to create a "Big Mac" of Arduino modules. You could, for example, combine an Arduino Uno with a Voice Box Shield, and a WiFly Shield to create a WiFi Talking Stephen Hawking(TM).



Read more about shields on: https://learn.sparkfun.com/tutorials/arduino-shields

Example Project
Thermometer

Creating a Quick-Read Thermometer That Blinks the Temperature

If the temperature is below 20 degrees Celsius, the LED will blink twice and then pause; if the temperature falls between 20 and 26 degrees, the LED will blink four times and then pause; and if the temperature is above 26 degrees, the LED will blink six times.

We'll make our sketch more modular by breaking it up into distinct functions that will make the sketch easier to follow, and the functions will be reusable. Our thermometer will perform two main tasks: measure and categorize the temperature, and blink the LED a certain number of times (determined by the temperature).

Hardware

- The required hardware is minimal:
- One TMP36 temperature sensor
- One breadboard
- Various connecting wires
- Arduino and USB cable

The Sketch

We'll need to create two functions for the sketch. The first one will read the value from the TMP36, convert it to Celsius, and then return a value of 2, 4, or 6, corresponding to the number of times the LED should blink. We'll alter the sketch from Project 8 for this purpose. Our void loop will call the functions in order and then pause for 2 seconds before restarting.

Note: Remember to save your modified project sketches with new filenames so that you don't accidentally delete your existing work!

```
#define LED 13

int blinks = 0;

void setup()
{
  pinMode(LED, OUTPUT);
}

int checkTemp()
{
  float voltage  = 0;
  float celsius  = 0;
  float hotTemp  = 26;
  float coldTemp = 20;
  float sensor   = 0;
  int result;
  // read the temperature sensor and convert the result to degrees Celsius

  sensor = analogRead(0);
  voltage = (sensor * 5000) / 1024; // convert raw sensor value to millivolts
  voltage = voltage - 500;          // remove voltage offset
  celsius = voltage / 10;           // convert millivolts to Celsius

  // act on temperature range
  if (celsius < coldTemp)
  {
    result = 2;
  }
else if (celsius >= coldTemp && celsius <= hotTemp)
{
    result = 4;
  }
  else
  {
    result = 6;    // (celsius > hotTemp)
  }
  return result;
}

void blinkLED(int cycles, int del)
{
  for ( int z = 0 ; z < cycles ; z++ )
  {
    digitalWrite(LED, HIGH);
    delay(del);
    digitalWrite(LED, LOW);
    delay(del);
  }
```

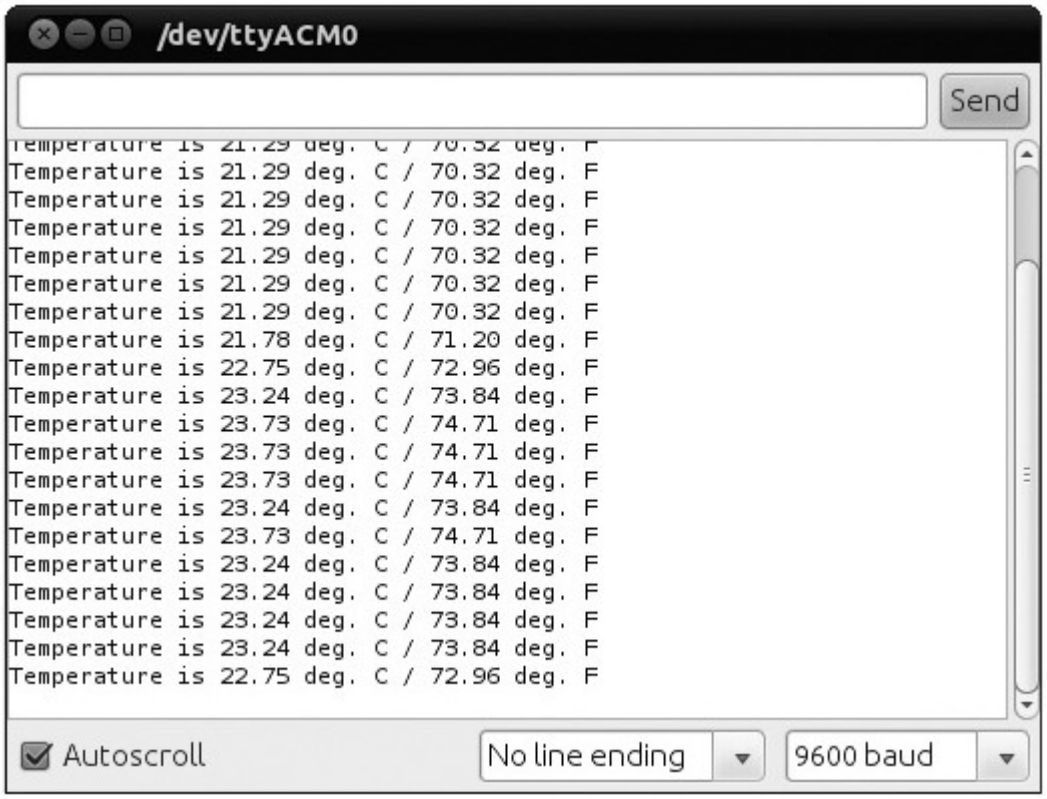
```
}

void loop()
{
  blinks = checkTemp();
  blinkLED(blinks, 500);
  delay(2000);
}
```

Because we use custom functions, all we have to do in void_loop() is call them and set the delay. The function checkTemp() returns a value to the integer variable blinks, and then blinkLED() will blink the LED blinks times with a delay of 500 milliseconds. The sketch then pauses for 2 seconds before repeating. Upload the sketch and watch the LED to see this thermometer in action. (Be sure to keep this circuit assembled, since we'll use it in the following examples.)

```
// Displaying the Temperature in the Serial Monitor
float celsius    = 0;
float fahrenheit = 0;

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

1 void findTemps()
  {
    float voltage = 0;
    float sensor  = 0;

    // read the temperature sensor and convert the result to degrees C and F
sensor  = analogRead(0);
    voltage = (sensor * 5000) / 1024;  // convert the raw sensor value to millivolts
    voltage = voltage - 500;          // remove the voltage offset
    celsius = voltage / 10;           // convert millivolts to Celsius
    fahrenheit = (1.8 * celsius) + 32; // convert Celsius to Fahrenheit
  }

void displayTemps()
  {
     Serial.print("Temperature is ");
     Serial.print(celsius, 2);
     Serial.print(" deg. C / ");
     Serial.print(fahrenheit, 2);
     Serial.println(" deg. F");
  // use .println here so the next reading starts on a new line
```

```
}

void loop()
{
  findTemps();
  displayTemps();
  delay(1000);
}
```

A lot is happening in this sketch, but we've created two functions, findTemps() and displayTemps(), to simplify things. These functions are called in void loop(), which is quite simple. Thus you see one reason to create your own functions: to make your sketches easier to understand and the code more modular and possibly reusable. After uploading the sketch, wait a few seconds, and then display the Serial Monitor. The temperature in your area should be displayed.



Recommended Reading

Books

a) Arduino Workshop: A hands-on introduction with 65 projects by John Boxall

b) Circuit building Do-It-Yourself For Dummies by Ward Silver

c) Arduino Cookbook by Michael Margolis

d) Programming Arduino – Next Steps by Simon Monk

Websites

a) http://www.Arduino.cc – The official site for the Arduino project has some of the best examples of Arduino Projects, tutorials, user forums, videos and much more.

b) http://www.twitter.com/Arduino – The official Twitter account for Arduino is a great way to learn about the most recent developments of Arduino, project examples, and to start following fellow Arduino enthusiasts.

c) https://en.wikipedia.org/wiki/Arduino – Wikipedia gives a great overview, history and current usage for the Arduino platform. A great place to begin your journey with the Arduino that will give a great base from which you can continue learning.

d) http://www.makershed.com/collections/arduino – Makershed is the official store of MAKE: Magazine which posts schematics for Arduino projects, and many Arduino components are available from Makershed that are unavailable anywhere else.

e) www.Sparkfun.com – Sparkfun is a company that sells Arduino Kits and components for enthusiasts. Products also available through Amazon.com

f) http://www.Adafruit.com – Similar to Makershed and Sparkfun, Adafruit provides Arduino Kits, components and other Arduino Printed Circuit Board types

such as the Arduino Nano (A smaller version of the Arduino Uno, which is the Arduino board most beginners get started with)

g) https://www.reddit.com/r/arduino/ – Reddit is a great resource to learn more about Arduino news and projects, as well as ask questions about Arduino. Once you begin to learn more and more you can start helping newbies to Arduino by answering their questions, which will help you to learn the Arduino even faster and helps build the community.

h) http://www.instructables.com/howto/Arduino+Projects/ – instructables is a fantastic website for makers of everything from furniture to art projects. Users submit instructions and projects which can then be built and improved by any member of the community. Instructables has a great section covering Arduino projects. Definitely a website to keep in

i) https://github.com/arduino/Arduino – Arduino is entirely open source, which means all instructions (plans for the printed circuit board and software) are available online to be downloaded and can be modified for your particular project. Github is a powerful tool in its own right and learning both Arduino and Github will help you in your prototyping career.

j) http://spectrum.ieee.org/geek-life/hands-on/the-making-of-arduino – Learn more in depth the detail and history of Arduino, including where the name "Arduino" came from.

Note: