UNIT-2

IOT WITH ARDUINO:

Introduction to the Arduino, Creating an Arduino programming Environment, Using the Arduino IDE, Creating an Arduino program, Using Libraries, Working with Digital Interfaces, Interfacing with Analog devices, Adding Interrupts, Communicating with devices, Using sensors, Working with Motors, Using an LCD. (Text Book -2)

Introduction to the Arduino:

- What is an Arduino?
- Introducing the Arduino Family
- Exploring Arduino shields

Introduction to Arduino

The concept of an open source hardware platform for creating microcontroller applications has sparked an interest in both hobbyists and professionals who are looking for simple ways to control electronic projects.

What Is an Arduino?

The Arduino is an open source microcontroller platform that you use for sensing both digital and analog input signals and for sending digital and analog output signals to control devices.

Arduino Is a Microcontroller

A microcontroller is a computer chip with minimal processing power and that often is designed for automatically controlling some type of external device. Instead of having a large feature set and lots of components squeezed into the chip, microcontrollers are designed with simplicity in mind.

Because microcontrollers don't need large processors, they have plenty of room on the chip to incorporate other features that high-powered computer chips use external circuits for.

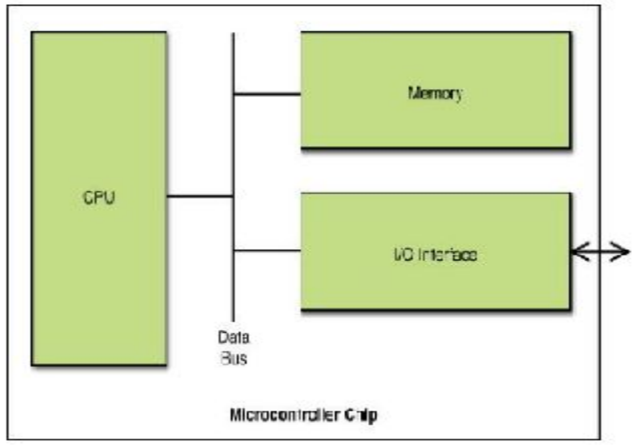


FIGURE 1.1 Block diagram of a microcontroller.

Using Open Source Hardware:

- In the world of open source software, a group of developers release the software program code to the general public so that anyone can make suggestions for changes and bug fixes. This often results in feature-rich software that has fewer bugs.
- The idea of open source hardware is much the same, except with using physical hardware rather than software. With open source hardware projects, the physical hardware that you use to control devices is open to the general public to freely use and modify as needed.
- The Arduino project developers have designed a full microcontroller system that uses a standard interface to interact with external devices.
- The design plans and architecture have been released to the public as open source, allowing anyone both to use the Arduino free of charge in their own projects and to even modify the Arduino without violating any patent or copyright laws.

The ATmega AVR Microcontroller Family

The Arduino developers selected the Atmel ATmega AVR family of microcontroller chips to power the Arduino.

The ATmega AVR microcontroller is an 8-bit computer that contains several features built in:

- Flash memory for storing program code statements
- Static random-access memory (SRAM) for storing program data
- Erasable electronic programmable read-only memory (EEPROM) for storing long-term data
- Digital input and output ports
- Analog-to-digital converter (ADC) for converting analog input signals to digital format

The ATmega AVR chip is a complete microcontroller system built in to a single chip. That allows the Arduino developers to create a small circuit board with minimal external components.

Processor	Flash	SRAM	EEPROM
ATmega48	4KB	512B	256B
ATmega88	8KB	1KB	512B
ATmega168	16KB	1KB	512B
ATmega328	32KB	2KB	1KB
ATmega32u4	32KB	2.5KB	1KB

TABLE 1.1 The ATmega Processor Family

The Arduino Layout

The Arduino developers created a standard interface so that other developers could easily incorporate the Arduino directly into their projects. (This is part of the open source hardware method.) The Arduino uses header sockets to make the input and output pins of the ATmega AVR chip available to external devices. Figure 1.2 shows the layout of the Arduino Uno.

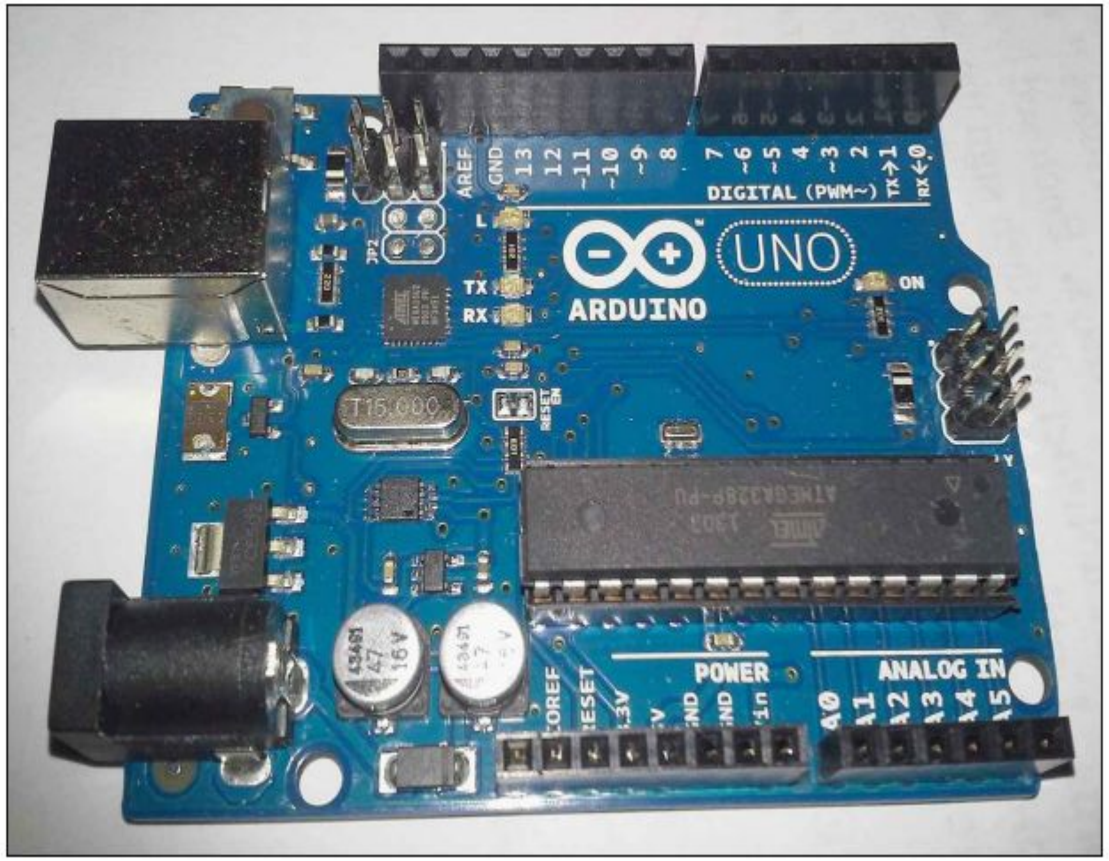


FIGURE 1.2 The Arduino Uno layout.

Examining the Arduino Uno:

It’s the most commonly used unit for interacting with projects that require a simple microcontroller.

The Uno R3 unit provides the following:

14 digital I/O interfaces

6 analog input interfaces

6 PWM interfaces

1 I2C controller interface

1 SPI controller interface

1 UART serial interface, connected to a USB interface

The digital I/O sockets have a double use. Not only are they used for digital input or output signals, but some of them are used for secondary purposes, depending on how you program the ATmega microcontroller:

- Digital sockets 0 and 1 are also used as the RX and TX lines for the serial UART.
- Digital sockets 3, 5, 6, 9, 10, and 11 are used for PWM outputs.
- The leftmost four sockets in the top header socket row are for the SPI controller interface

The Arduino Uno also has four built-in LEDs, shown in Figure, that help you see what's going on in the Arduino:

- A green LED (marked ON) that indicates when the Arduino is receiving power
- Two yellow LEDs (marked TX and RX) that indicate when the UART serial interface is receiving or sending data
- A yellow LED (marked L) connected to digital output socket 13 that you can control from your programs

Introducing the Arduino Family

The History of the Arduino

- The Arduino wasn’t designed by a large electronics corporation or even by a group of computer science majors. Instead, it was designed out of necessity by a group of students and instructors looking for a solution to animate their art projects.
- The Arduino project built upon the basic features of the Wiring project, but at a lower cost for students.
- Every part of the Arduino system was designed for simplicity for nontechnical people.
- The hardware interface is somewhat forgiving. For instance, if you hook up the wrong wires to the wrong ports, you won’t usually blow up your Arduino unit.
- If you do happen to manage to blow up the ATmega microcontroller chip, the Arduino was designed to easily replace the microcontroller chip without having to purchase an entire Arduino unit.
- Likewise, the software for the Arduino was designed with nonprogrammers in mind. In an interesting tie to the art world, programs that you create for the Arduino are called **sketches**, and the folders where you store your sketches are called **sketchbooks**.

Exploring the Arduino Models

Part of the open source hardware method is to provide lots of options for developers. This allows developers to find just the right Arduino to fit into their project.

Arduino Uno :

- It’s the most popular Arduino unit and provides the basic functionality of the ATmega AVR microcontroller within the standard Arduino footprint.
- It uses an ATmega328 microcontroller, which provides 14 digital I/O interfaces, 6 analog input interfaces, and 6 pulse-width modulation (PWM) interfaces for controlling motors.
- The Uno circuitry also contains a USB interface for communicating with workstations as a serial communications device, a separate power jack so that you can power the Arduino Uno without plugging it into a workstation, and a reset button to restart the program running on the Arduino.
- The size and layout of the Arduino Uno has been made the standard format for most Arduino units, so just about all Arduino shields fit into the header sockets of the Arduino Uno.

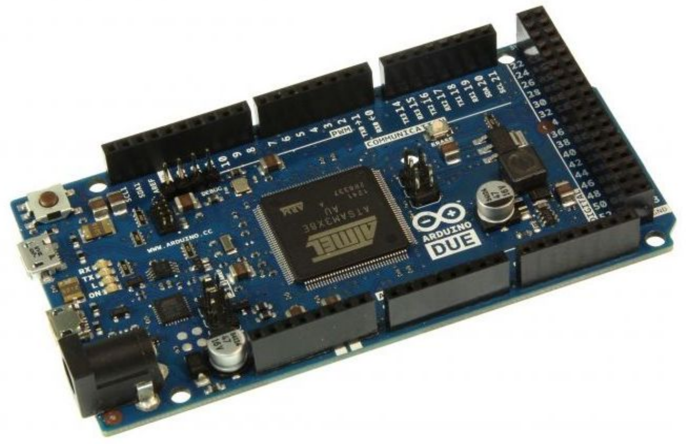
Arduino Due:

The Arduino Due unit uses a more powerful 32-bit ARM Cortex-M3 CPU instead of the standard ATmega328 microcontroller.

This provides significantly more processing power than the standard Arduino Uno.

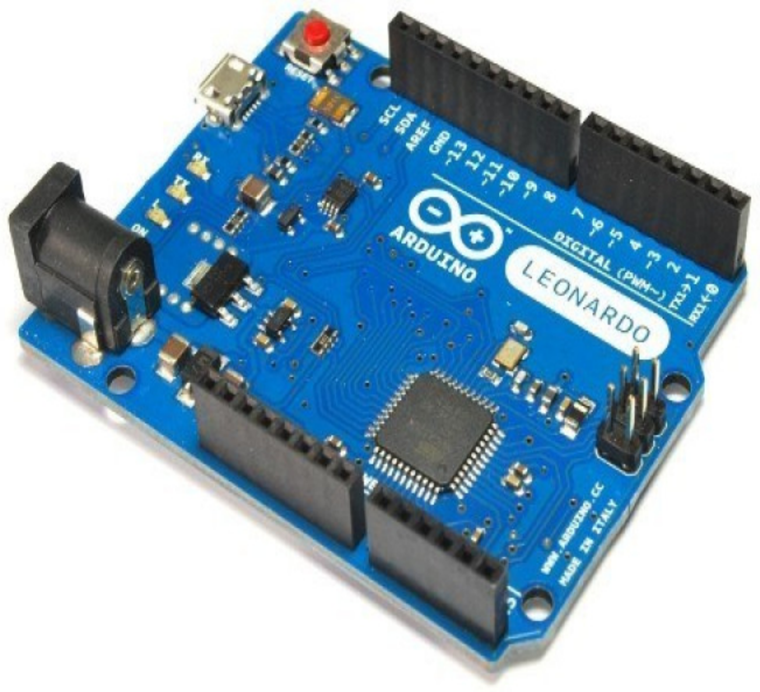
The Due provides 54 digital I/O interfaces, 12 PWM outputs, 12 analog inputs, and 4 serial Universal Asynchronous Receiver/Transmitter (UART) interfaces.

It also provides considerably more memory, with 96KB of SRAM for data storage, and 512KB of flash memory for program code storage.



Arduino Leonardo

- The Arduino Leonardo uses the ATmega32u4 microcontroller, which provides 20 digital I/O interfaces, 7 PWM outputs, and 12 analog inputs.
- The Arduino Leonardo has the same header layout as the Uno, with additional header sockets for the extra interfaces.
- One nice feature of the Arduino Leonardo is that it can emulate a keyboard and mouse when connected to a workstation using the USB port.
- You can write code to run on the Arduino Leonardo that sends standard keyboard or mouse signals to the workstation for processing.



Arduino Mega

- The Arduino Mega provides more interfaces than the standard Arduino Uno.
- It uses an ATmega2560 microcontroller, which provides 54 digital I/O interfaces, 15 PWM outputs, 16 analog inputs, and 4 UART serial ports.
- You can use all the standard Arduino shields with the Mega unit. The Arduino Mega provides a lot more interfaces, but it comes at a cost of a larger footprint.
- The Arduino Uno is a small unit that can easily fit into a project, but the Arduino Mega device is considerably large

Arduino Micro

- The Arduino Micro is a small-sized Arduino unit that provides basic microcontroller capabilities, but in a much smaller footprint.
- It uses the same ATmega32u4 microcontroller as the Lenardo, which provides 20 digital I/O interfaces, 7 PWM outputs, and 12 analog inputs. The selling point of the Arduino Micro is that it is only 4.8cm long and 1.77cm wide, small enough to fit into just about any electronics project.
- It uses a USB port to communicate as a serial device with workstations, or as a keyboard and mouse emulator like the Lenardo can.
- The downside to the Micro is that because of its smaller size, it doesn’t work with the standard Arduino shields, so you can’t expand its capabilities with other features.

Arduino Esplora

- The Arduino Esplora is an attempt at creating an open source game controller.
- It uses the ATmega 32u4 microcontroller just like the Lenardo and Micro do, but also contains hardware commonly found on commercial game controllers: Analog joystick , Three-axis accelerator, Light sensor, Temperature sensor, Microphone, Linear potentiometer, Connector for an LCD display, Buzzer, LED lights, Switches for up, down, left, and right.
- The Esplora also has a unique design to help it fit into a game controller case that could be handheld, providing easier access to the switches and analog joystick

Arduino Yun

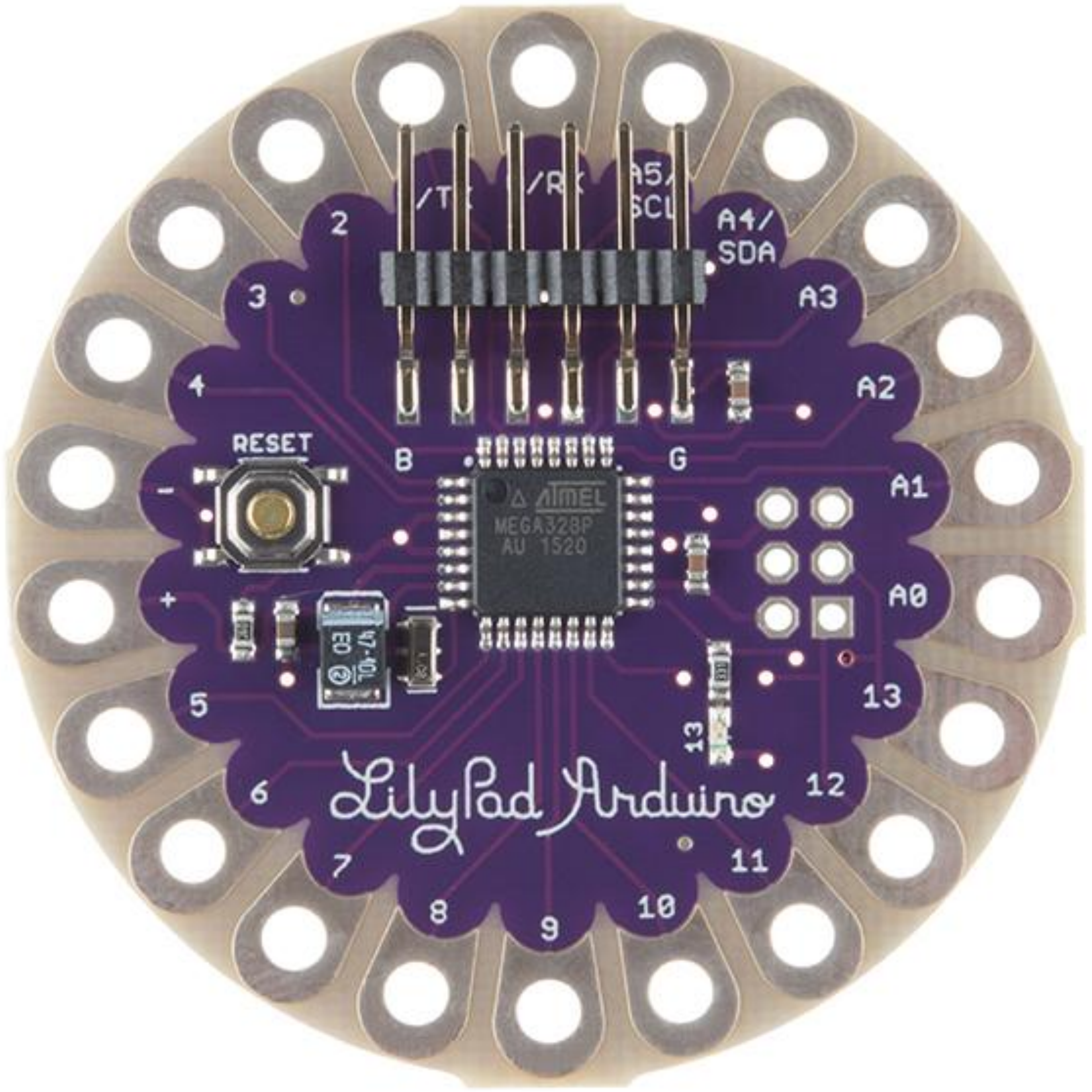
- The Arduino Yun is an interesting experiment in combining the hardware versatility of an ATMega32u4 microcontroller with an Atheros microprocessor running a Linux operating system environment.
- The two chips communicate with each other directly on the Yun circuit board, so that the ATmega microcontroller can pass data directly to the Atheros system for processing on the Linux system.
- The Linux system also includes built-in Ethernet and Wi-Fi ports for external communication, in addition to a standard USB port to run a remote serial console. This is the ultimate in embedding a full Linux system with a microcontroller.

Arduino Ethernet

- The Arduino Ethernet project combines the microcontroller of the Arduino Uno with a wired Ethernet interface in one unit.
- This enables you to interface with the Arduino remotely across a network to read data collected or to trigger events in the microcontroller.

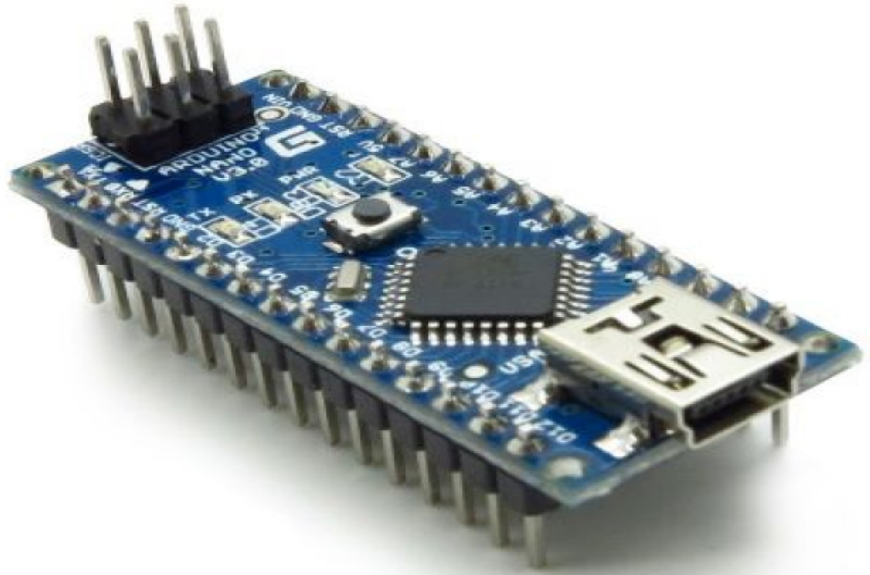
LilyPad Arduino

- The Arduino LilyPad is certainly an interesting device. It was designed for the textile arts world, embedding a microcontroller within textiles.
- Not only is it small, but it's also very thin and lightweight, perfect for sewing within just about any type of fabric.
- One interesting feature of the LilyPad Arduino is the built-in MCP73831 LiPo battery charging chip.
- This chip allows you to embed rechargeable batteries with the device; when the unit is plugged into a USB port of a workstation, the rechargeable batteries recharge automatically.



Arduino Nano

- The **Arduino Nano** is a small, complete, and breadboard-friendly board based on the ATmega328P released in 2008. It offers the same connectivity and specs of the Arduino UNO board in a smaller form factor.
- The Arduino Nano is equipped with 30 male I/O headers, in a DIP30like configuration, which can be programmed using the Arduino Software IDE, which is common to all Arduino boards and running both online and offline.
- The board can be powered through a type-B micro-USB cable or from a 9 V battery.



- Operating voltage: 5 volts
- Input voltage: 6 to 20 volts
- Digital I/O pins: 14 (6 optional PWM outputs)
- Analog input pins: 8
  - Flash memory: 32 KB, of which 0.5 KB is used by bootloader
  - SRAM: 2 KB
  - EEPROM: 1 KB
  - Clock speed: 16 MHz

Accessories You Might Need

A USB A-B cable: The Arduino USB port uses a B-type USB interface, so you'll need to find an A-B USB cable to connect the Arduino to your workstation. Most printers use a B-type USB interface, so these cables aren't hard to find.

An external power source: The Arduino Uno includes a 2.1mm center positive power jack so that you can plug in an external power source. This can either be an AC/DCV converter or a batter power pack that provides 5V power

A breadboard: As you experiment with interfacing various electronic components to your Arduino, a breadboard comes in handy to quickly build connections. Once you have your circuits designed, you can purchase prototype shields to make the interface connections more permanent

Resistors: Resistors are used for limiting the current through a circuit, usually to avoid burning out the microcontroller inputs or the LED outputs.

Variable resistors: You can also use variable resistors (called potentiometers) to adjust the voltage going into an analog input socket to control a circuit.

Switches: Switches allow you to provide a digital input for controlling your programs.

Wires: You'll need some wires for connecting the breadboard components together and interfacing them into the Arduino header sockets.

Sensors: You can interface a number of both digital and analog sensors into your Arduino. There are temperature sensors and light sensors, for instance, that produce varying analog outputs based on the temperature or amount of light.

Motors: Controlling moving parts requires some type of motor. The two main types of motors you'll most often require are servo motors, which rotate to a fixed position, and DC motors, which can spin at a variable rate based on the voltage provided.

Exploring Arduino Shields

The standard interface to the Arduino provides a common way for developers to build external circuits that interface to the Arduino and thus provide additional functionality. These units are called shields.

There are plenty of Arduino shields available for a myriad of different projects. This section discusses a few popular shields.

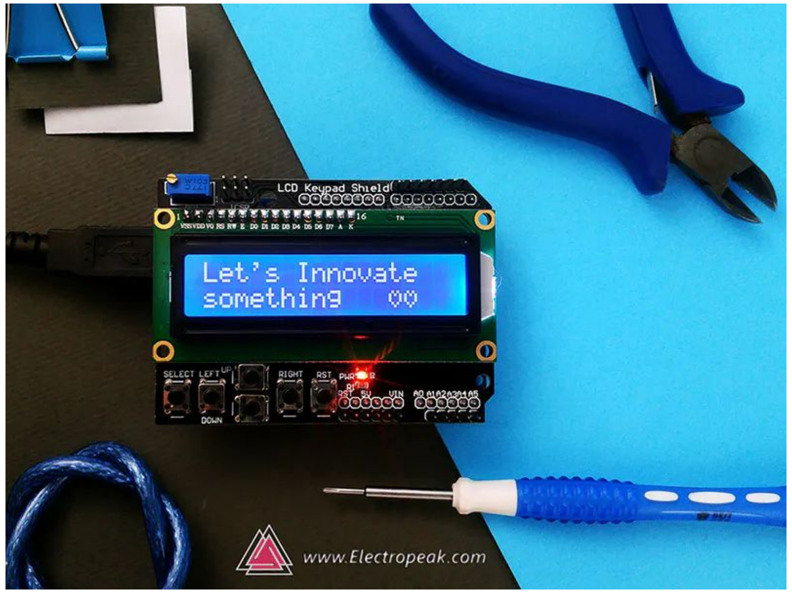
Connecting with the Ethernet Shield:

These days just about every project needs some type of network connectivity. Microcontroller projects can use a network to send data to a remote monitoring device, to allow remote connectivity to monitor data, or to just store data in a remote location. The Ethernet shield provides a common wired network interface to the Arduino, along with a software library for using the network features. Figure shows the Ethernet shield plugged into an Arduino Uno R3 unit

While the Ethernet shield plugs into the standard header sockets of the Arduino Uno, it also has the same standard header sockets built in, so you can piggy-back additional shields onto the Ethernet shield (so long as they don't use the same signals to process data)

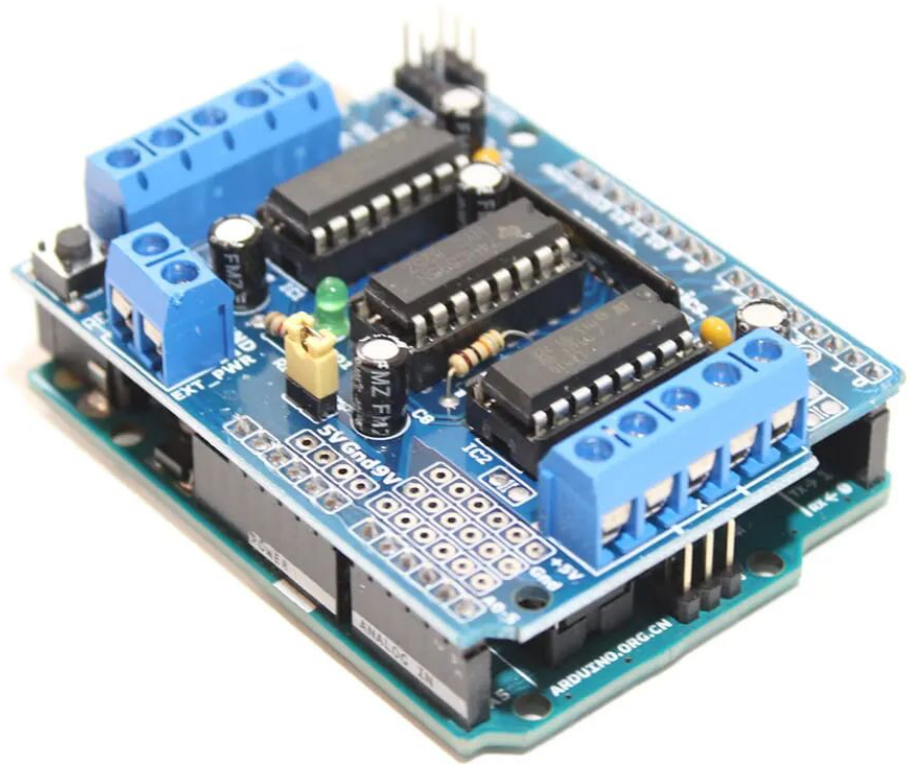
Displaying Data with the LCD Shield

Often, instead of connecting to the Arduino unit to retrieve information, it would be nice to just view data quickly. This is where the LCD shield comes into play. It uses a standard LCD display to display 2 rows of 25 characters that you can easily view as the Arduino unit is running. The LCD shield also contains five push buttons, enabling you to control the program with basic commands



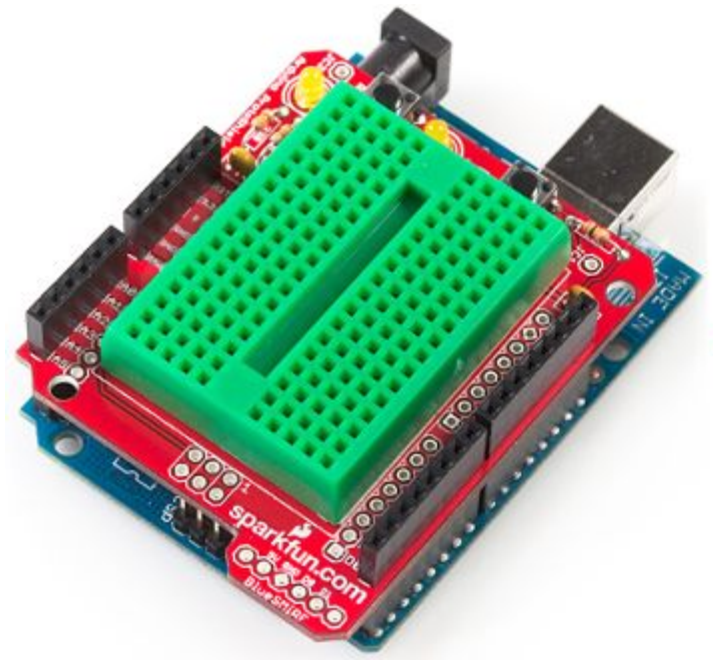
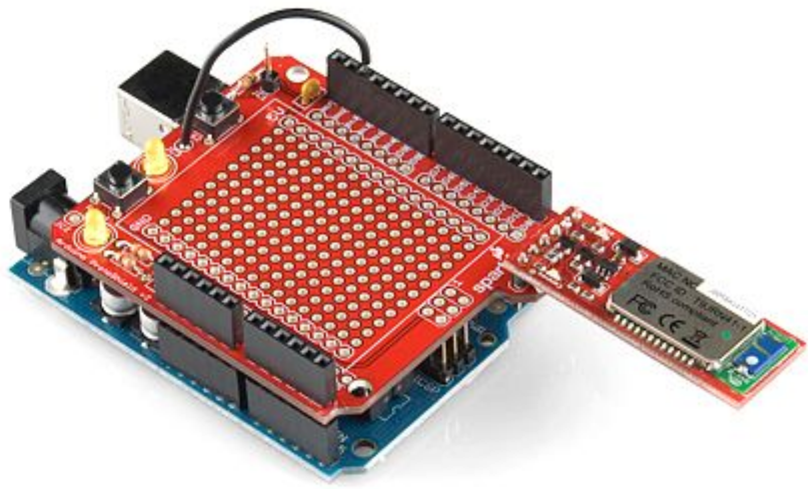
Running Motors with the Motor Shield

A popular use for the Arduino is to control motors. Motors come in handy when working with robotics and when making your project mobile using wheels. Most motors require the use of PWM inputs so that you can control the speed of the motor. The basic Arduino system contains six PWM outputs, enabling you to control up to six separate motors. The motor shield expands that capability.



Developing New Projects with the Prototype Shield

If you plan on doing your own electronic circuit development, you'll need some type of environment to build your circuits as you experiment. The Prototype shield provides such an environment. The top of the Prototype shield provides access to all the Arduino header pins, so you can connect to any pin in the ATmega microcontroller. The middle of the Prototype shield provides standard-spaced soldering holes for connecting electronic parts, such as resistors, sensors, diodes, and LEDs. If you would like to experiment with your projects before soldering the components in place, you can also use a small solderless breadboard on top of the Prototype shield, which allows you to create temporary layouts of your electronic components for testing, before committing them to the soldering holes.

Creating an Arduino programming Environment

- Exploring Microcontroller Internals
- Moving beyond Machine code
- Creating Arduino Programs
- Installing the Arduino IDE

Exploring Microcontroller Internals

At the core of all Arduino units is an ATmega AVR series microcontroller. The programs that you write to control the Arduino must run on the ATmega microcontroller.

Peeking Inside the ATmega Microcontroller:

3 basic components reside inside of a microcontroller:

- The CPU
- Memory
- The input/output (I/O) interface

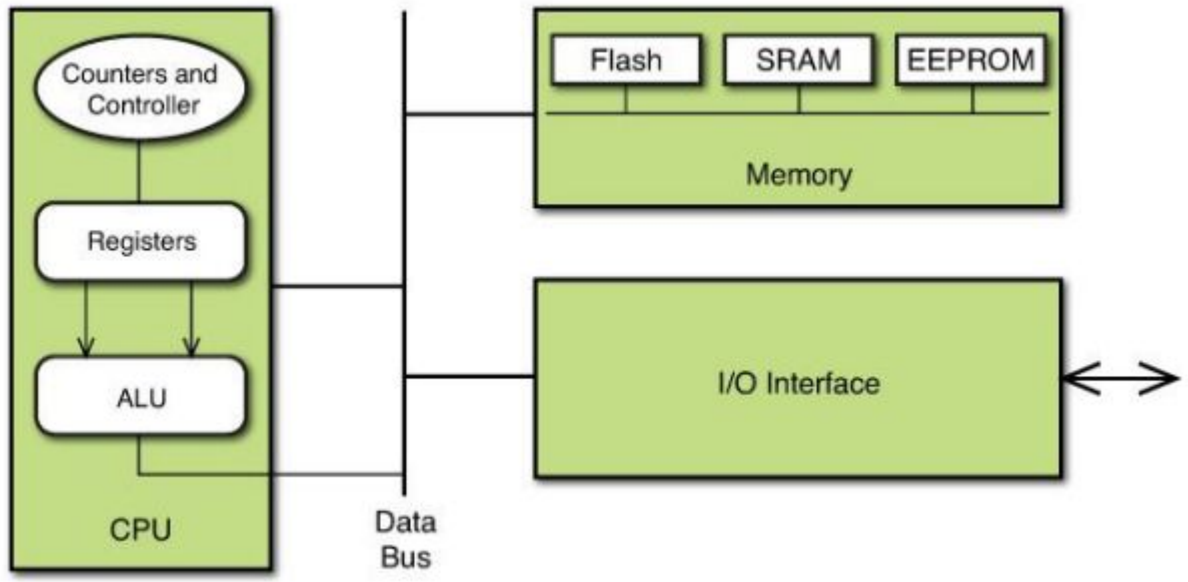


FIGURE 2.1 The ATmega microcontroller block layout.

The CPU:

- An Arithmetic Logic Unit (ALU), which performs the mathematical and logical operations
- Data registers, which store data for processing in the ALU
- A status register, which contains the status information of the most recently executed arithmetic instruction
- A program counter, which keeps track of the memory location of the next instruction to run
- A stack pointer, which keeps track of the location of temporary data stored in memory
- A controller, which handles loading program code and memory data into the registers, ALU, and pointers

All the CPU components interact within the CPU based on a clocking signal provided to the microcontroller. At each **clock pulse**, the CPU performs one operation. The speed of the clock pulse determines how fast the CPU processes instructions. For example, the Arduino Uno utilizes a 16MHz clock, which provides 16 million clock pulses per second. Most of the ATmega microcontroller CPU instructions require only one or two clock pulses to complete, making the Arduino capable of sampling inputs at **a very high rate**.

Memory :

The Atmel ATmega microcontroller family uses the Harvard architecture for computers(one memory section to store the program code, and another memory section to store the program data).

The ATmega microcontroller family uses a separate form of memory for each type:

- Flash memory to store program code - The flash memory retains data after you remove power from the microcontroller.
- Static random-access memory (SRAM) to store program data -SRAM memory does not retain data when the power is off, so any data your program stores in SRAM will be lost at power down.
- To compensate for that, the ATmega microcontroller also incorporates a separate electronically erasable programmable read-only memory (**EEPROM**) memory section besides the SRAM for storing data.
- The EEPROM memory can retain data after loss of power, until you manually clear it out. It's not as fast as the SRAM memory, so you should use it only when you need to save data for later use.

Input/Output interface:

- The I/O interface allows the microcontroller to interact with the outside world.
- This is what allows the ATmega controller to read analog and digital input signals and send digital output signals back to external devices.
- The ATmega controller contains a built-in analog-to-digital converter (ADC), which converts analog input signals into a digital value that you can read from your programs.
- It also contains circuitry to control the digital interface pins so that you can use the same pins for either digital input or digital output.
- Analog Output You might have noticed that the ATmega microcontroller doesn't have any analog output interfaces. The reason for that is you can simulate an analog output by using digital outputs and a technique called **pulse-width modulation** (PWM).
- PWM sends a digital signal of a varying duration and frequency, thus emulating an analog output signal. The ATmega328 microcontroller used in the Arduino Uno supports six PWM outputs.

Programming the Microcontroller

- The CPU can controller using software program code. The CPU reads the program code instructions, and then performs the specified actions one step at a time.
- Each type of CPU has its own specific instructions that it understands, called its **instruction set.** The instruction set is what tells the CPU what data to retrieve and how to process that data.
- Operations such as reading data, performing a mathematical operation, or outputting data are defined in the instruction set.
- Because the CPU can only read and write binary data, the instruction set consists of groups of binary data, usually consisting of 1- or 2-byte commands. These commands are **called machine code** (also commonly called operation codes, or opcodes for short).
- The ATmega AVR instruction set uses 16-bit (2-byte) instructions to control the CPU actions. For example, the instruction to add the data values stored in registers R1 and R2, and then place the result in register R1, looks like this: 0000 1100 0001 0010
- When the CPU processes this instruction, it knows to fetch the data values from the registers, use the ALU to add them, and then place the result back in register R1. Knowing just what machine codes do what functions, and require what data, can be an almost impossible task for programmers.

Moving Beyond Machine Code

- Machine code is used directly by the microcontroller to process data, but it’s not the easiest programming language for humans to read and understand.
- To solve that problem, developers have created some other ways to create program code.
- Coding with Assembly Language Assembly language is closely related to machine code; it assigns a text mnemonic code to represent each individual machine code instruction.
- The text mnemonic is usually a short string that symbolizes the function performed (such as ADD to add two values). That makes writing basic microcontroller programs a lot easier.
- For example, if you remember from the preceding section, the machine code to add the values stored in register R1 to the value stored in register R2 and place the result in register R1 looks like this:

```
0000 1100 0001 0010
```

- However, the assembly language code for that function looks like this: **add r1, r2**
- Each of the machine code instructions for the ATmega AVR family of microcontrollers has an associated assembly language mnemonic.

The ATmega AVR series of microcontrollers (which is what the Arduino uses) has 282 separate instructions that they recognize, so there are 282 separate assembly language mnemonics.

The instructions can be broken down into six categories:

- Load data from a memory location into a register.
- Perform mathematical operations on register values.
- Compare two register values.
- Copy a register value to memory.
- Branch to another program code location.
- Interact with a digital I/O interface.

There are separate instructions for each register, each mathematical operation, and each method for copying data to and from memory. It’s not hard to see why there are 282 separate instructions. While the Atmel AVR assembly language is an improvement over coding machine language, creating fancy programs for the Arduino using assembly language is still somewhat difficult and usually left to the more advanced programmers.

Making Life Easier with C

- Hard to remember the 282 separate instructions.
- Higher-level programming languages help separate out having to know the inner workings of the microcontroller from the programming code.
- Higher-level programming languages hide most of the internal parts of the CPU operation from the programmer, so that the programmer can concentrate on just coding.
- For example, with higher-level programming languages, you can just assign variable names for data, and the compiler converts those variable names into the proper memory locations or registers for use with the CPU.
- With a higher level programming language, to add two numbers and place the result back into the same location, you just write something like this: value1 = value1 + value2;
- Now that's even better than the machine code version. The key to using higher-level programming languages is that there must **be a compiler** that can convert the user-friendly program code into the machine language code that runs in the microcontroller.
- That's done with a combination of a compiler and a code library

- The compiler reads the higher-level programming language code and converts it into the machine language code.
- Because it must generate machine language code, the compile is specific to the underlying CPU that the program will run.
- **Different CPUs require different compilers.** Besides being able to convert the higher-level programming language code into machine code, most compilers also have a set of common functions that make it easier for the programmer to write code specific for the CPU. This is called a **code library**.
- Again, different CPUs have different libraries to access their features. Many different higher-level programming languages are available these days, but the Atmel developers chose the popular **C programming language** as the language to use for creating code for the AVR microcontroller family.
- They released a compiler for converting C code into the AVR microcontroller machine language code, in addition to a library of functions for interacting with the specific digital and analog ports on the microcontroller.

Creating Arduino Programs

Currently, two main C language libraries are available for the **Arduino environment**:

- The Atmel C library -- created for using ATmega microcontrollers in general
- The Arduino project library -- created specifically for nonprogrammers to use for the Arduino unit.

Exploring the Atmel C Library The Atmel C environment consists of two **basic packages:**

- A command-line compiler environment
- A graphical front end to the compiler environment

The AVR Libc project (http://www.nongnu.org/avr-libc/) has worked on creating a complete C language command-line compiler and library for the ATmega AVR family of microcontrollers.

It consists of three core packages:

The **avr-gcc compiler** for creating the microcontroller machine code from C language code

The **avr-libc package** for providing C language libraries for the AVR microcontroller

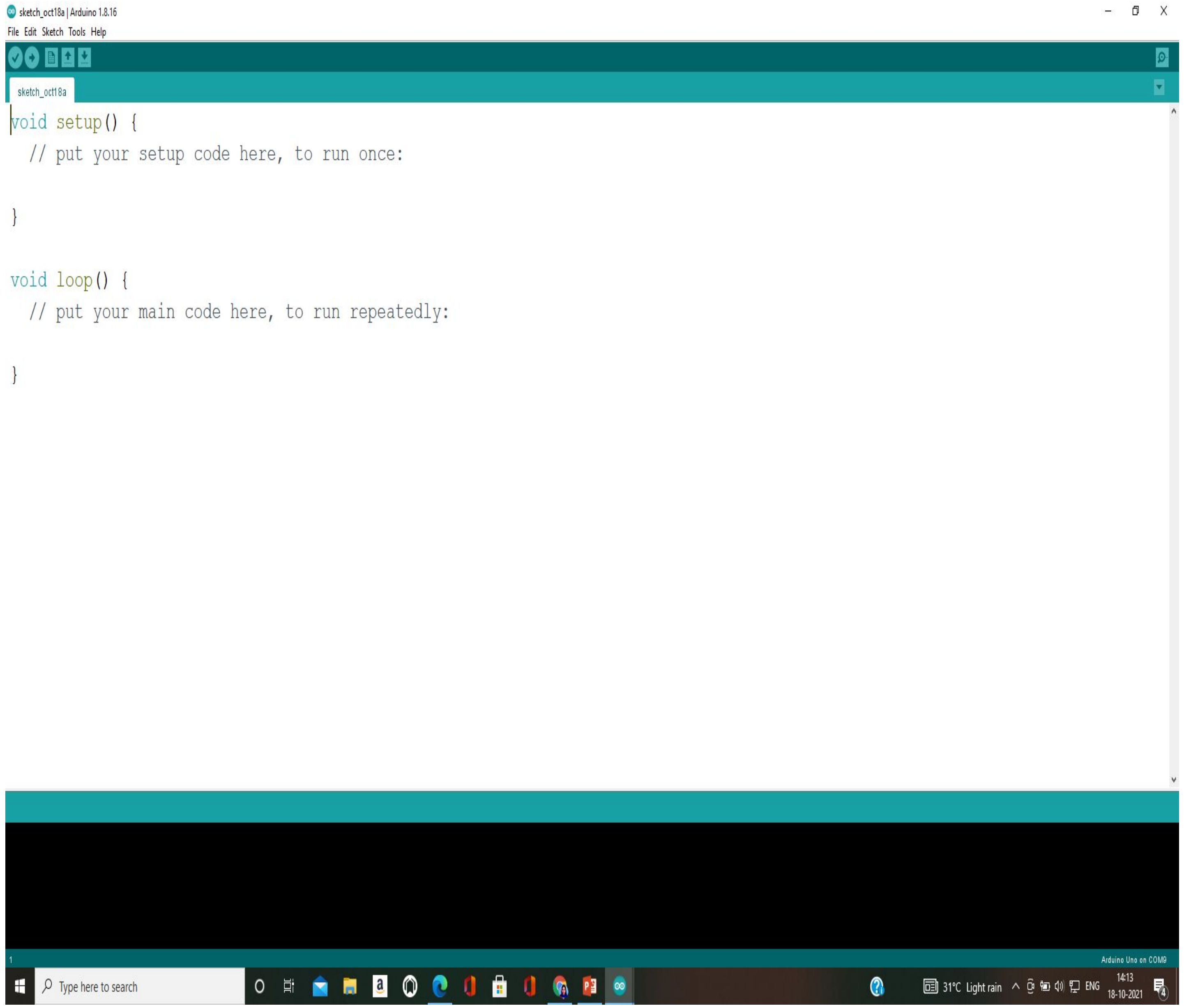
The **avr-binutils package** for providing extra utilities for working with the C language programs

- However, these days just about every application uses a graphical interface, including compilers.
- The Atmel developers have released a full integrated development environment (IDE) package that provides a graphical windows environment for editing and compiling C programs for the AVR microcontroller.
- The Atmel Studio package combines a graphical text editor for entering code with all the word processing features you’re familiar with (such as cutting and pasting text) with the avr-gcc compiler.
- It also incorporates a full C language debugger, making it easier to debug your program code.
- In addition, it outputs all the error messages generated by the microcontroller into the graphical window.

Installing the Arduino IDE

https://www.arduino.cc/en/software

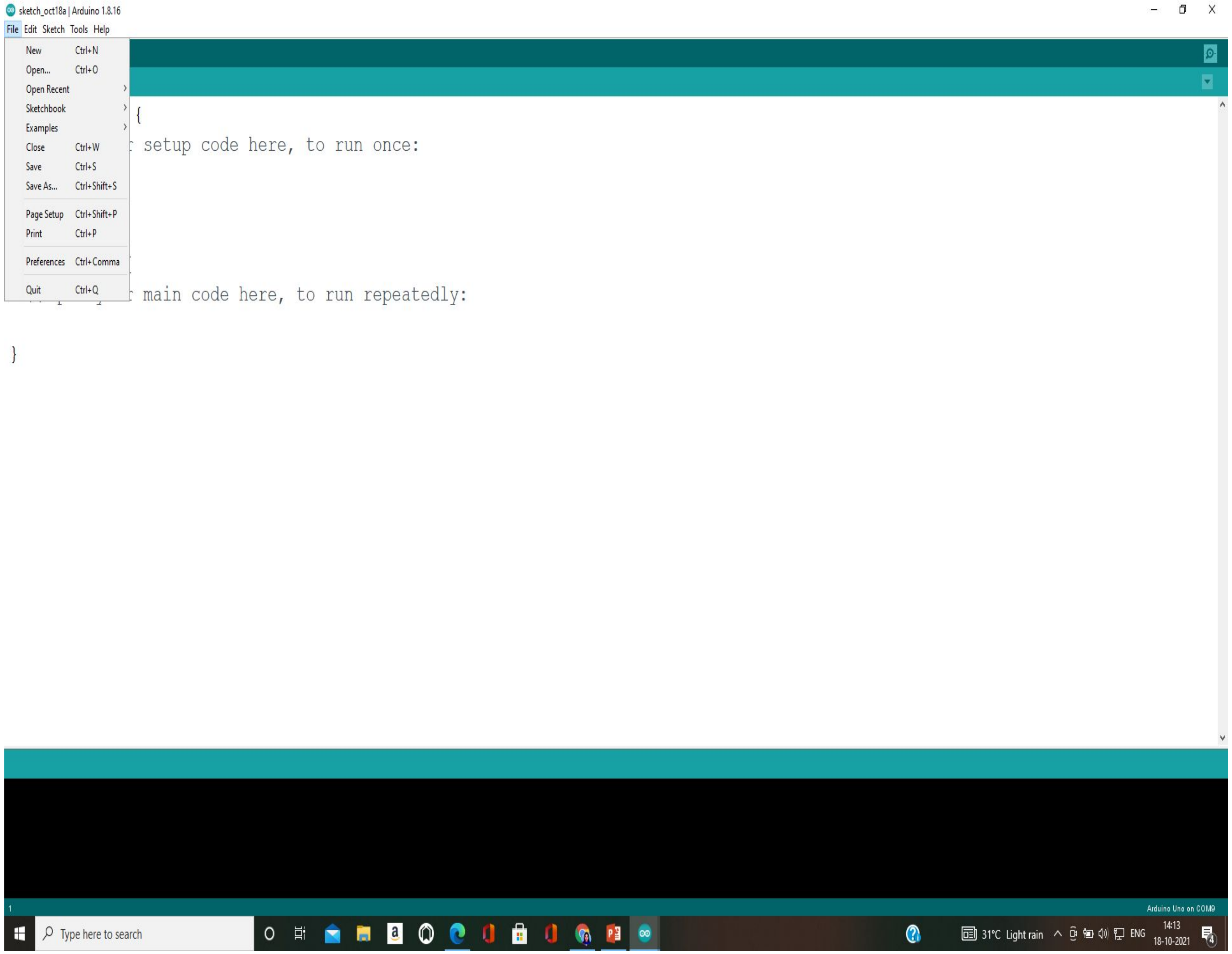
Using the Arduino IDE



```
void setup() {
  // put your setup code here, to run once:

}

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

}
```

The Arduino IDE interface contains five main sections:

- The menu bar
- The taskbar
- The editor window
- The message area
- The console window



```
{
  setup code here, to run once:

  main code here, to run repeatedly:

}
```

Walking Through the Menus The menu bar provides access to all the features of the Arduino IDE. From here,

create a new program,

retrieve an existing program,

automatically format your code,

compile your program, and

upload your program to the Arduino unit.

The File Menu :

The File menu contains all the file-handling options required to load and save sketches, along with a couple of miscellaneous options that don’t fit in any other menu bar category.

New :

It creates a new Arduino sketch tab in the main window. When you select the option to create a new sketch, the Arduino IDE automatically assigns it a name, starting with sketch_, followed by the month and date (such as jan01), followed by a letter making the sketch name unique. When you save the sketch, you can change this sketch name to something that makes more sense.

Open:

The Open option opens an existing sketch using the File Open dialog box of the operating system. The Open dialog box will open in the default sketchbook folder for your Arduino IDE. The File Open dialog box allows you to navigate to another folder area if you’ve saved sketches on a removable media device, such as a USB thumb drive.

Sketchbook:

The Sketchbook option provides a quick way to open your existing sketches. It lists all the sketches that you've saved in the default sketchbook area in the Arduino IDE. You can select the sketch to load from this listing.

Examples:

The Examples option provides links to lots of different types of example sketches provided by the Arduino developers. The examples are divided into different categories, so you can quickly find an example sketch for whatever type of project you're working on. When you select an example sketch, the sketch code automatically loads into the editor window. You can then verify and upload the example sketch directly, or you can modify the sketch code before compiling and uploading.

Close:

The Close option closes out the current sketch editor window and safely closes the sketch file. Closing the sketch will also exit the Arduino IDE interface.

Save:

The Save option allows you to save a previously saved sketch in your sketchbook area using the same filename. Be careful when using this option, as the Arduino IDE will just overwrite the existing saved sketch and it won't keep any past versions automatically for you.

Save As:

The Save As option allows you to save a sketch using a new filename. The Arduino IDE saves each sketch under a separate folder under the Arduino IDE sketchbook folder area. It saves the sketch file using the .ino file extension.

Upload:

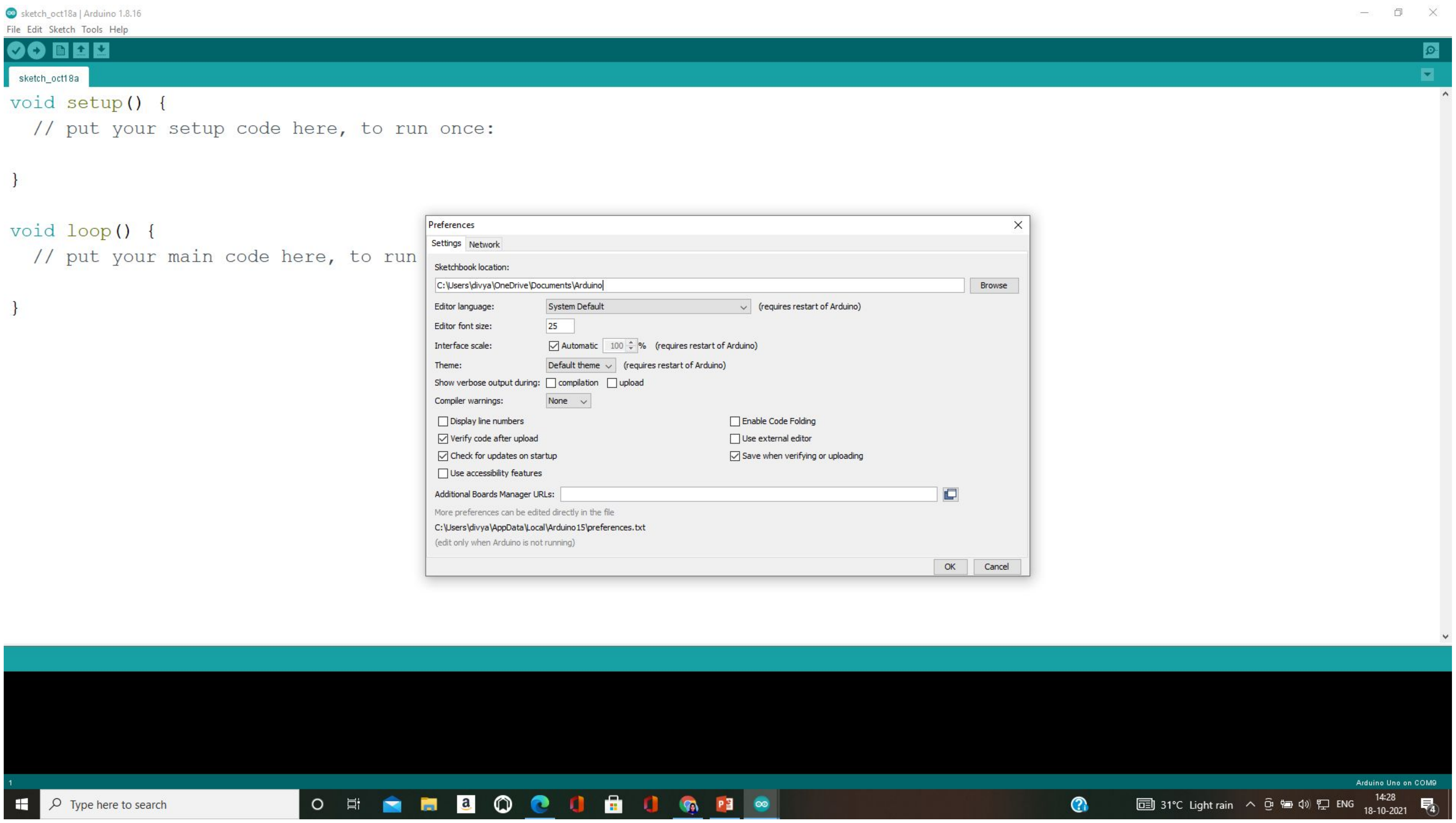
The Upload option uploads the compiled sketch code to the Arduino unit using the USB serial interface. Make sure that the Arduino unit is connected to the USB port before you try to upload the sketch. Upload Using Programmer The Upload Using Programmer option is for more advanced users. The normal Upload option uploads your sketch to a special location in the flash program memory area on the ATmega microcontroller. The Upload Using Programmer option writes the program to the start of the flash memory area on the ATmega microcontroller. This requires additional coding in your sketch so that the Arduino can boot from your program.

Page Setup:

The Page Setup option allows you to define formatting options for printing your sketch on a printer connected to your workstation.

Print:

The Print option allows you to send the sketch text to a printer connected to your workstation



```
void setup() {
  // put your setup code here, to run once:

}

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

```
}
```

Preferences

Sketchbook location:
C:\Users\divya\OneDrive\Documents\Arduino

Editor language: System Default (requires restart of Arduino)

Editor font size: 25

Interface scale: ☑ Automatic 100 % (requires restart of Arduino)

Theme: Default theme (requires restart of Arduino)

Show verbose output during: ☐ compilation ☐ upload

Compiler warnings: None

☐ Display line numbers
☑ Verify code after upload
☑ Check for updates on startup
☐ Use accessibility features
☐ Enable Code Folding
☐ Use external editor
☑ Save when verifying or uploading

Additional Boards Manager URLs:

More preferences can be edited directly in the file
C:\Users\divya\AppData\Local\Arduino15\preferences.txt
(edit only when Arduino is not running)

Preferences :

Sketchbook Location: Sets the folder used to store sketches saved from the Arduino IDE.

Editor Language: Sets the default language recognized in the text editor.

Editor Font Size: Sets the font size used in the text editor.

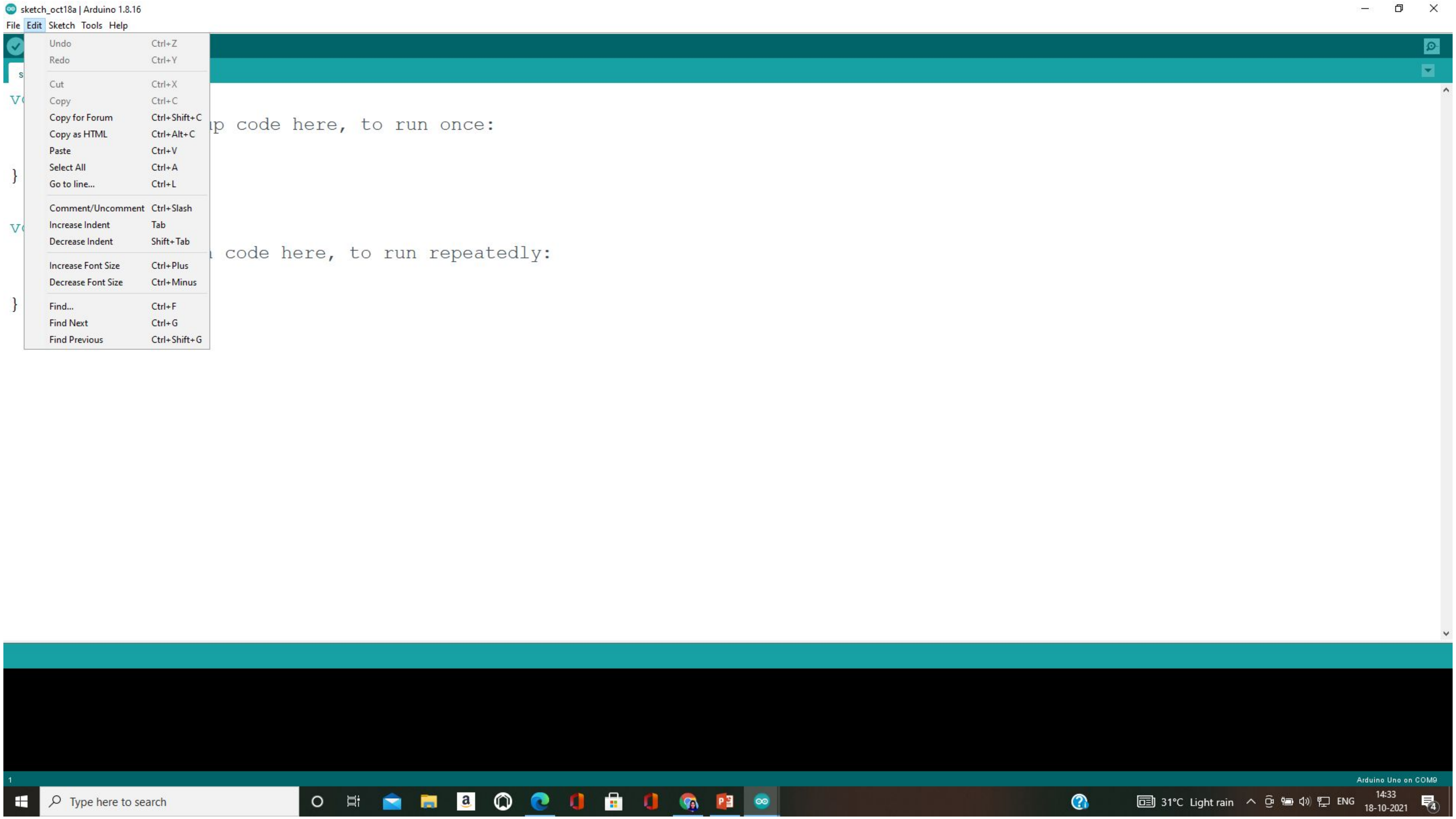
Show Verbose Output: Displays more output during either compiling or uploading. This proves handy if you run into any problems during either of these processes.

Verify Code After Upload: Verifies the machine code uploaded into the Arduino against the code stored on the workstation.

Use External Editor: Allows you to use a separate editor program instead of the built-in editor in the Arduino IDE. Check for Updates on Startup: Connects to the main Arduino.cc website to check for newer versions of the Arduino IDE package each time you start the Arduino IDE.

Update Sketch Files to New Extension on Save: If you have sketches created using older versions of the Arduino IDE, enable this setting to automatically save them in the new .ino format.

Automatically Associate .ino Files with Arduino: Allows you to double-click .ino sketch files from the Windows Explorer program to open them using the Arduino IDE. You can change these settings at any time, and they will take effect immediately in the current Arduino IDE window. Quit Closes the existing sketch and exits the Arduino IDE window.



sketch_oct18a | Arduino 1.8.16

File Edit Sketch Tools Help

Undo	Ctrl+Z
Redo	Ctrl+Y
Cut	Ctrl+X
Copy	Ctrl+C
Copy for Forum	Ctrl+Shift+C
Copy as HTML	Ctrl+Alt+C
Paste	Ctrl+V
Select All	Ctrl+A
Go to line...	Ctrl+L
Comment/Uncomment	Ctrl+Slash
Increase Indent	Tab
Decrease Indent	Shift+Tab
Increase Font Size	Ctrl+Plus
Decrease Font Size	Ctrl+Minus
Find...	Ctrl+F
Find Next	Ctrl+G
Find Previous	Ctrl+Shift+G

```
v
  p code here, to run once:

}

v
   code here, to run repeatedly:

}
```

Arduino Uno on COM9

The Edit Menu:

The Edit menu contains options for working with the program code text in the editor window.

Undo: This feature enables you to take back changes that you make to the sketch code. This returns the code to the original version.

Redo: This feature enables you to revert back to the changes you made (undoes an Undo command).

Cut: This feature enables you to cut selected text from the sketch code to the system Clipboard. The selected text disappears from the editor window.

Copy: This feature allows you to copy selected text from the sketch code to the system Clipboard. With the Copy feature, the selected text remains in place in the sketch code.

Copy for Forum: This interesting option is for use if you interact in the Arduino community forums. If you post a problem in the forums, members will often ask to see your code. When you use this feature, the Arduino IDE copies your sketch code to the Clipboard and retains the color formatting that the editor uses to highlight Arduino functions. When you paste the code into your forum message, it retains that formatting, making it easier for other forum members to follow your code. Copy as HTML The Copy as HTML option also keeps the color and formatting features of the editor, but embeds them as HTML code in your program code. This option comes in handy if you want to post your sketch code onto a web page to share with others.

Paste :

The Paste option copies any text currently in your system Clipboard into the editor window.

Select All:

The Select All option highlights all the text in the editor window to use with the Copy or Cut options.

Comment/Uncomment :

You use this handy feature when troubleshooting your sketch code. You can highlight a block of code and then select this option to automatically mark the code as commented text. The compiler ignores any commented text in the sketch code. When you finish troubleshooting, you can highlight the commented code block and select this option to remove the comment tags and make the code active again.

Increase Indent :

This option helps format your code to make it more readable. It increases the space used to indent code blocks, such as if-then statements, while loops, and for loops. This helps the readability of your code.

Decrease Indent :

This option enables you to reduce the amount of space placed in front of indented code blocks. Sometimes if code lines run too long, it helps to reduce the indentation spaces so that the code more easily fits within the editor window.

Find :

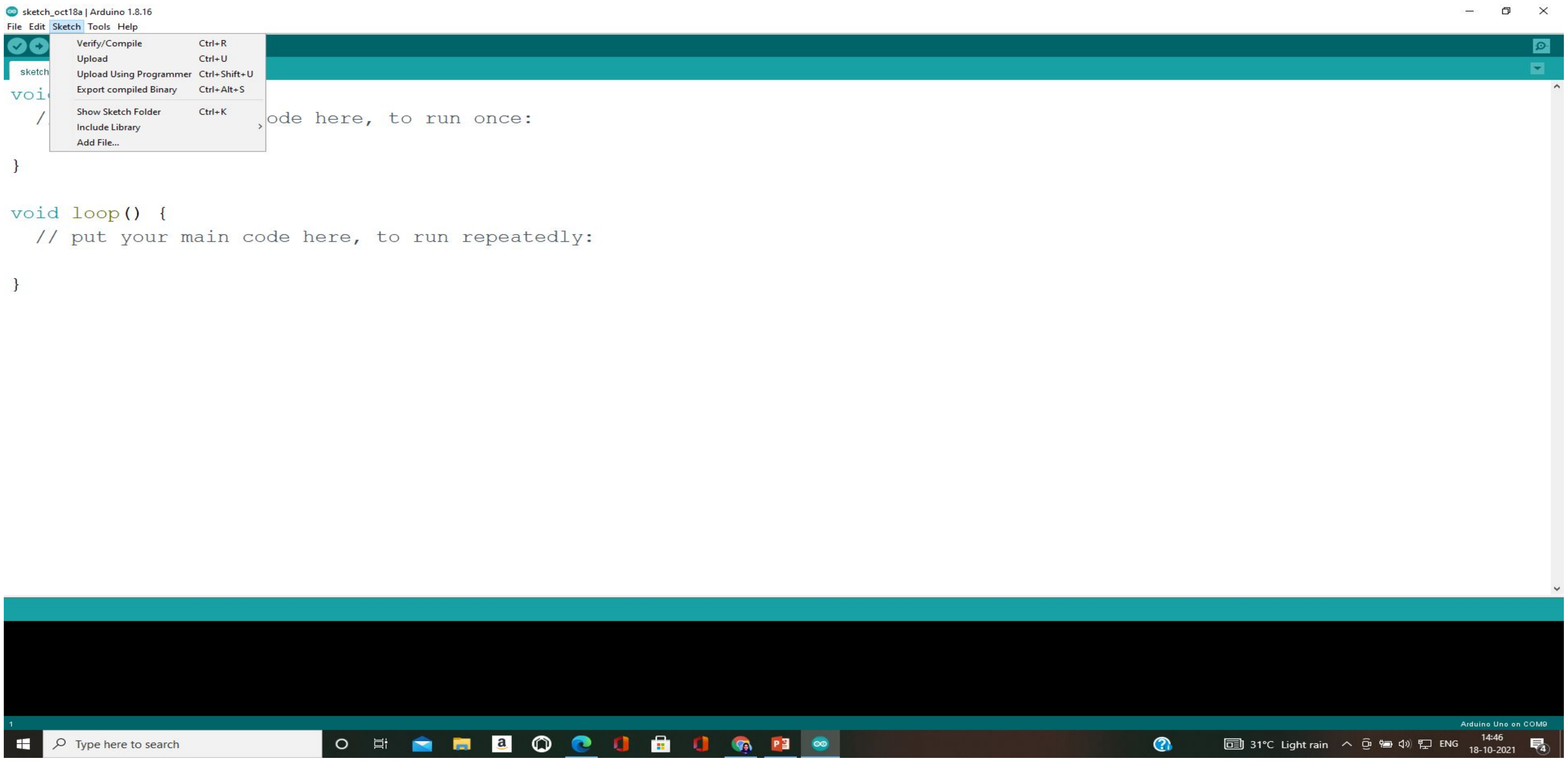
The Find option provides **simple search and replace features** in the editor window. If you just want to find a specific text string in the code, enter it into the Find text box. You can select or deselect the Ignore Case check box as required. The Wrap Around check box helps you find longer text strings that may have wrapped around to the next line in the editor window. If you need to quickly **replace text**, such as if you're changing a variable name in your sketch code, you can enter the replacement text in the Replace With text box. You can then replace a single instance of the found text, or click the Replace All button to replace all occurrences of the text with the replacement text.

Find Next: The Find Next option finds the next occurrence of the previously submitted search text.

Find Previous: The Find Previous option finds the previous occurrence of the previously submitted search text.

The Sketch Menu :

The Sketch menu provides options for working with the actual sketch code.



sketch_oct18a | Arduino 1.8.16
File Edit Sketch Tools Help
Verify/Compile Ctrl+R
Upload Ctrl+U
Upload Using Programmer Ctrl+Shift+U
Export compiled Binary Ctrl+Alt+S
Show Sketch Folder Ctrl+K
Include Library
Add File...
sketch
Arduino Uno on COM9

```
voi
  /                    ode here, to run once:

}

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

}
```

Verify/Compile:

The Verify/Compile option checks the Arduino program for any syntax errors, and then processes the code through the avr-gcc compiler to generate the ATmega microcontroller machine code. The result of the Verify/Compile process displays in the console window of the Arduino IDE. If any errors in the program code generate compiler errors, the errors will appear in the console window.

Show Sketch Folder:

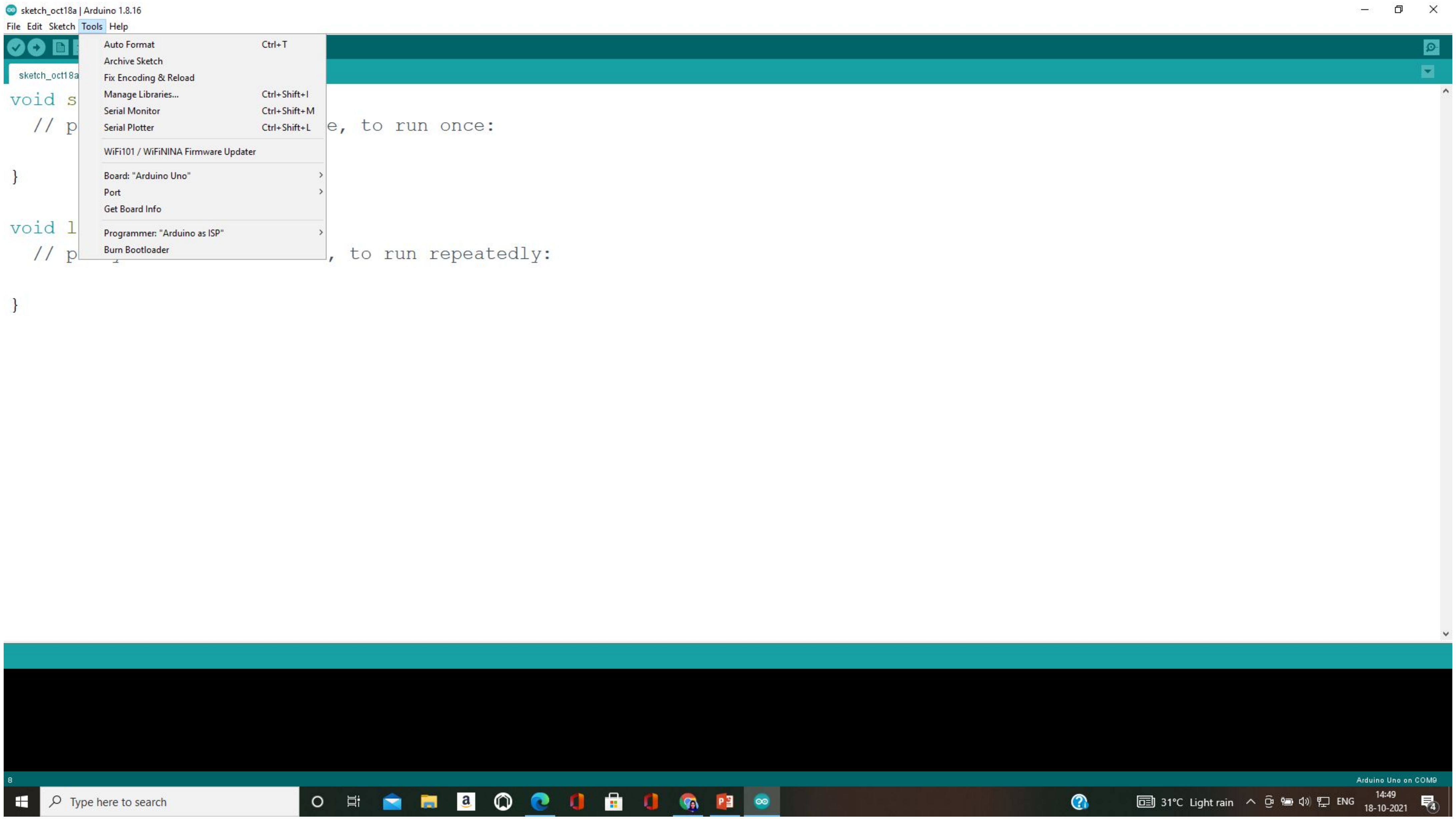
The Show Sketch Folder option shows all the files stored in the folder associated with the sketch. This comes in handy if you have multiple library files stored in the sketch folder.

Add File :

The Add File menu option enables you to add a library file to the sketch folder.

Import Library : The Import Library menu option automatically adds a C language #include directive related to the specific library that you select. You can then use functions from that library in your program code.

The Tools Menu The Tools menu provides some miscellaneous options for your sketch environment,



sketch_oct18a | Arduino 1.8.16

File Edit Sketch Tools Help

Auto Format	Ctrl+T
Archive Sketch	
Fix Encoding & Reload	
Manage Libraries...	Ctrl+Shift+I
Serial Monitor	Ctrl+Shift+M
Serial Plotter	Ctrl+Shift+L
WiFi101 / WiFiNINA Firmware Updater	
Board: "Arduino Uno"	›
Port	›
Get Board Info	
Programmer: "Arduino as ISP"	›
Burn Bootloader	

```
void s
  // p                         e, to run once:

}

void l
  // p                        , to run repeatedly:

}
```

8

Arduino Uno on COM9

Auto Format:

The Auto Format option helps tidy up your program code by indenting lines of code contained within a common code block, such as if-then statements, while loops, and for loops. Using the Auto Format feature helps make your program code more readable, both for you and for anyone else reading your code.

Archive Sketch :

This feature saves the sketch file as an archive file for the operating system platform used. For Windows, this appears as a compressed folder using the zip format. For Linux, this appears as a .tar.gz file. Fix Encoding and Reload This option is one of the more confusing options available in the Arduino IDE. If you load a sketch file that contains non-ASCII characters, you'll get odd looking characters appear in the editor window where the non-ASCII characters are located. When you select this option, the Arduino IDE reloads the file and saves it using UTF-8 encoding.

Serial Monitor:

The Serial Monitor option enables you to interact with the serial interface on the Arduino unit. It produces a separate window dialog box, which displays any output text from the Arduino and allows you to enter text to send to the serial interface on the Arduino. You must write the code in your programs to send and receive the data using the Arduino USB serial port for this to work.

Board :

The Board option is extremely important. You must set the Board option to the Arduino unit that you're using so that the Arduino IDE compiler can create the proper machine code from your sketch code.

Serial Port:

The Serial Port option selects the workstation serial port to use to upload the compiled sketch to the Arduino unit. You must select the serial port that the Arduino is plugged into.

Programmer:

The Programmer option is for advanced users to select what version of code to upload to the ATmega microcontroller. If you use the Arduino bootloader format (which is recommended), you don't need to set the Programmer option.

Burn Bootloader:

This option enables you to upload the Arduino bootloader program onto a new ATmega microcontroller. If you purchase an Arduino unit, this code is already loaded onto the microcontroller installed in your Arduino, so you don't need to use this feature. However, if you have to replace the ATmega microcontroller chip in your Arduino, or if you purchase a new ATmega microcontroller to use in a separate breadboard project, you must burn the bootloader program into the new microcontroller unit before you can use the Arduino IDE program format for your sketches.

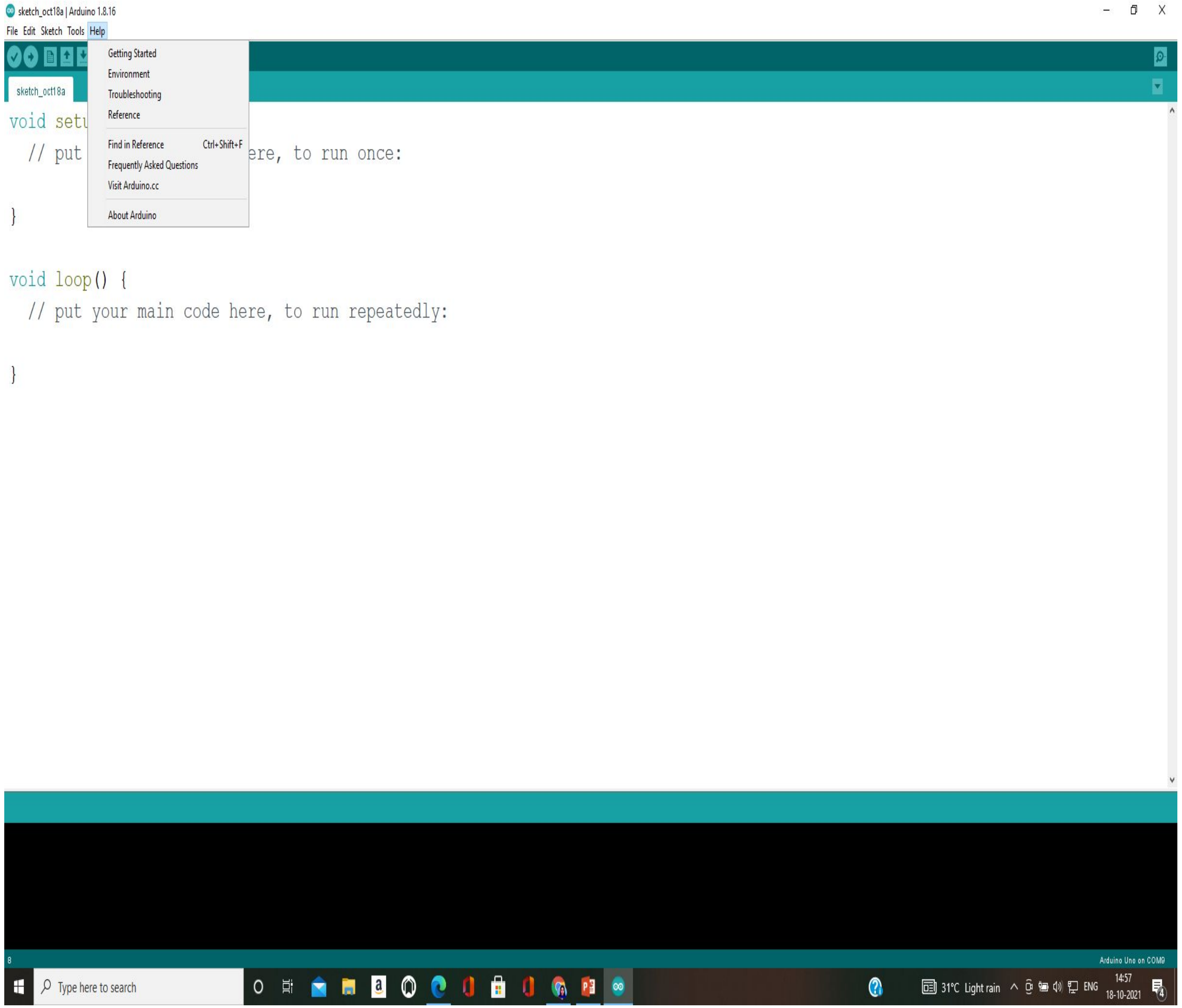
The Help Menu

The Help menu consists of six sections that relate to the Arduino IDE documentation:

- Getting Started
- Environment
- Troubleshooting
- Reference
- Find in Reference
- Frequently Asked Questions

Besides these six topics, two links provide information about the Arduino IDE:

Visit Arduino.cc About Arduino



```
void setu
  // put                ere, to run once:

}

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

}
```

Exploring the Toolbar:

The toolbar provides icons for some of the more popular menu bar functions:

- Verify (the checkmark icon): Performs the Verify/Compile option.
- Upload (the right arrow icon): Uploads the compiled sketch code to the Arduino unit.
- New (the page icon): Opens a new sketch tab in the Arduino IDE window.
- Open (the up arrow icon): Displays the File Open dialog box that allows you to select an existing sketch to load into the editor.
- Save (the down arrow icon): Saves the sketch code into the current sketch filename.
- Serial monitor (the magnifying glass icon): Opens the serial monitor window to interact with the serial port on the Arduino unit.

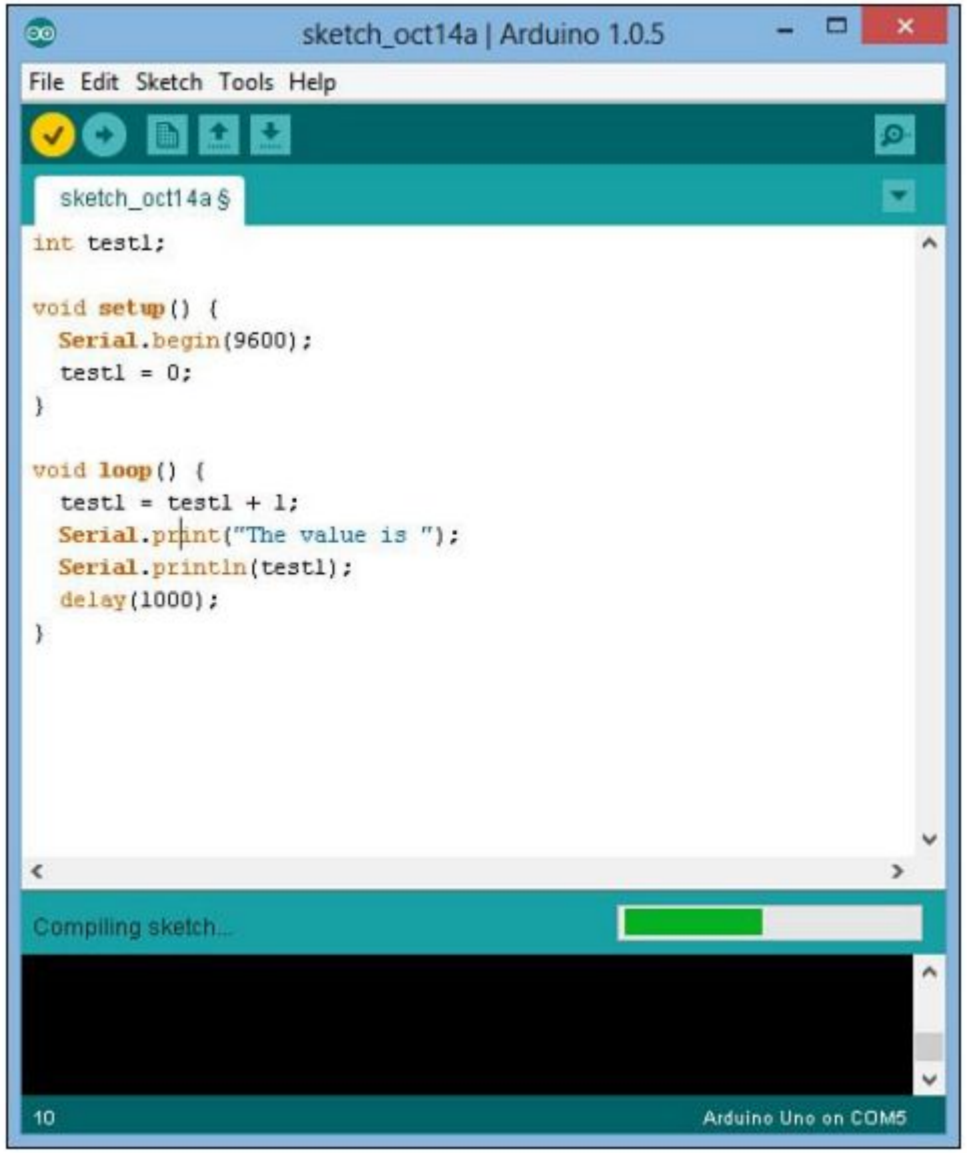
Exploring the Message Area and Console Window

The last two sections of the Arduino IDE are provided to give you some idea of what’s going on when you compile or upload your sketch. The message area is a one-line section of the Arduino IDE interface that displays quick messages about what’s happening. For example, as you compile the sketch, it shows a progress bar indicating the progress of the compile process, as shown in Figure

Exploring the Message Area and Console Window

The message area is a one-line section of the Arduino IDE interface that displays quick messages about what’s happening.

For example, as you compile the sketch, it shows a progress bar indicating the progress of the compile process, as shown in Figure



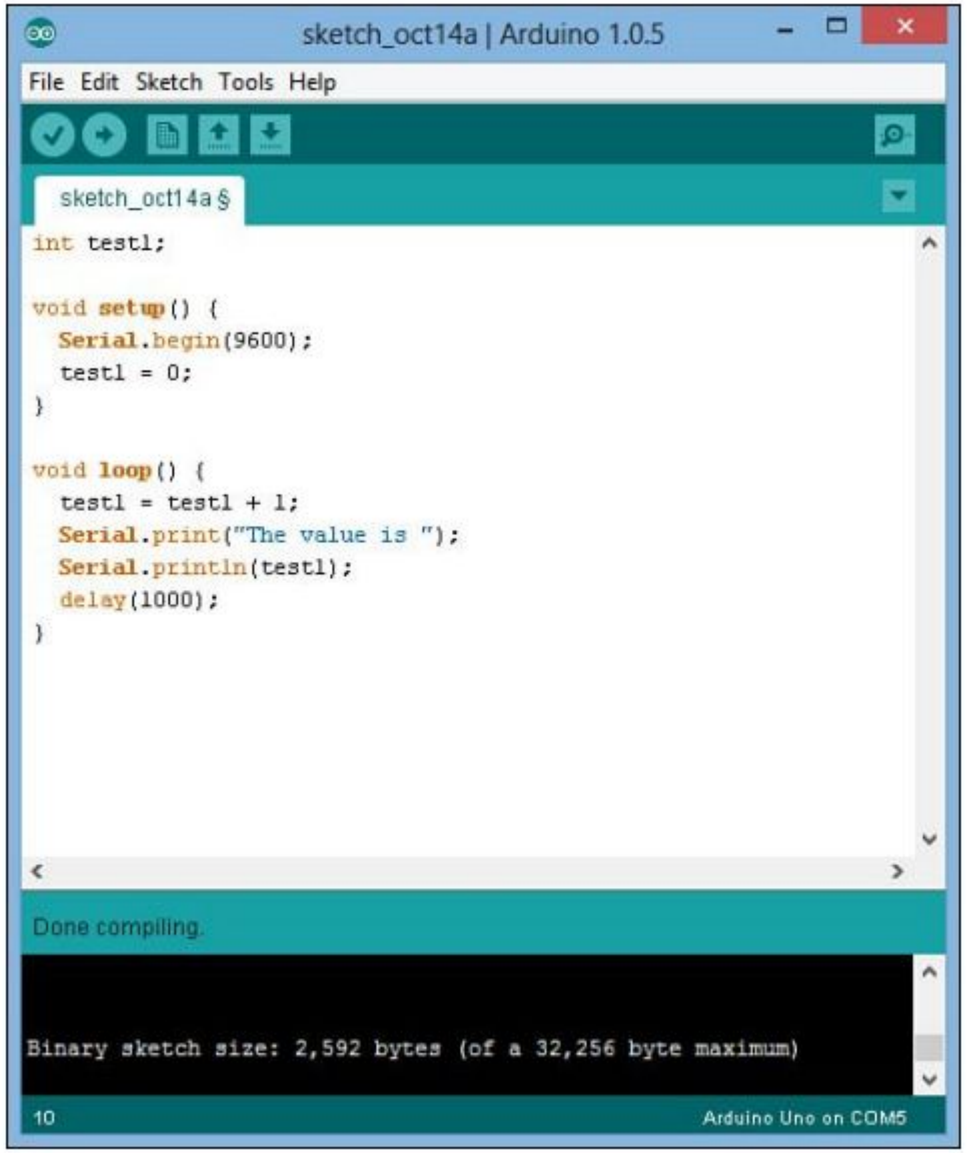
```
int test1;

void setup() {
  Serial.begin(9600);
  test1 = 0;
}

void loop() {
  test1 = test1 + 1;
  Serial.print("The value is ");
  Serial.println(test1);
  delay(1000);
}
```

Compiling sketch...

FIGURE 3.2 The Arduino IDE message area indicating the compile progress.

The console window often displays informational messages from the last command you entered. For example, after a successful compile, you'll see a message showing how large the program is, as shown in Figure 3.3

If there are any errors in the compiled sketch code, both the message area and the console window will display messages pointing to the line or lines that contain the errors.



```
int test1;

void setup() {
  Serial.begin(9600);
  test1 = 0;
}

void loop() {
  test1 = test1 + 1;
  Serial.print("The value is ");
  Serial.println(test1);
  delay(1000);
}
```

Done compiling.

```
Binary sketch size: 2,592 bytes (of a 32,256 byte maximum)
```

FIGURE 3.3 The Arduino IDE message window displaying the compiler result.

Using the Serial Monitor:

The serial monitor is a special feature in the Arduino IDE that can come in handy when troubleshooting code running on your Arduino, or just for some simple communication with the program running on the Arduino. The serial monitor acts like a serial terminal, enabling you to send data to the Arduino serial port and receive data back from the Arduino serial port. The serial monitor displays the data it receives on the selected serial port in a popup window. You can activate the serial monitor feature in the Arduino IDE in three ways:

- Choose Tools > Serial Monitor from the menu bar.
- Press the Ctrl+Shift+M key combination.
- Click the serial monitor icon on the toolbar.

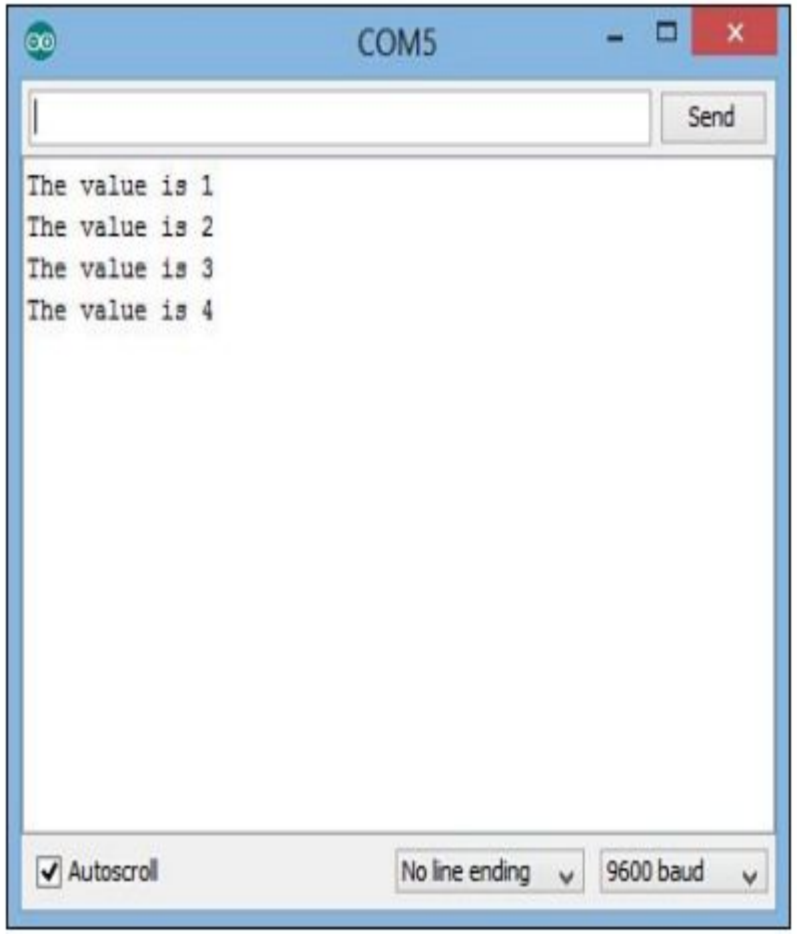
If you've selected the correct serial port for your Arduino unit, a pop-up window will appear and connect to the serial port to communicate with the Arduino unit serial port using the USB connection. Figure 3.6 shows the serial monitor window

Three setting options are available at the bottom of the serial monitor window:

Autoscroll: Checking this option always displays the last line in the scroll window. Removing the check freezes the window so that you can manually scroll through the output.

Newline: This option controls what type of end-of-line marking the serial monitor sends after the text that you enter. You can choose to send no end-of-line marking, a standard UNIX-style newline character, a carriage control character, or a Windows-style newline and carriage control characters. Which you select depends on how you write your Arduino sketch to look for data.

Baud Rate: The communications speed that the serial monitor connects to the Arduino serial port. By default, the serial monitor will detect the baud rate based on the Serial.begin() function used in the sketch code



```
The value is 1
The value is 2
The value is 3
The value is 4
```

FIGURE 3.6 The serial monitor interface window.

Creating an Arduino Program

- Building an Arduino sketch
- Compiling and running a sketch
- Interfacing your Arduino to electronic circuits

Building an Arduino Sketch

Setup
Loop

void setup() {
code lines
}
void loop() {
code lines
}

Including Libraries

#include <*library*>

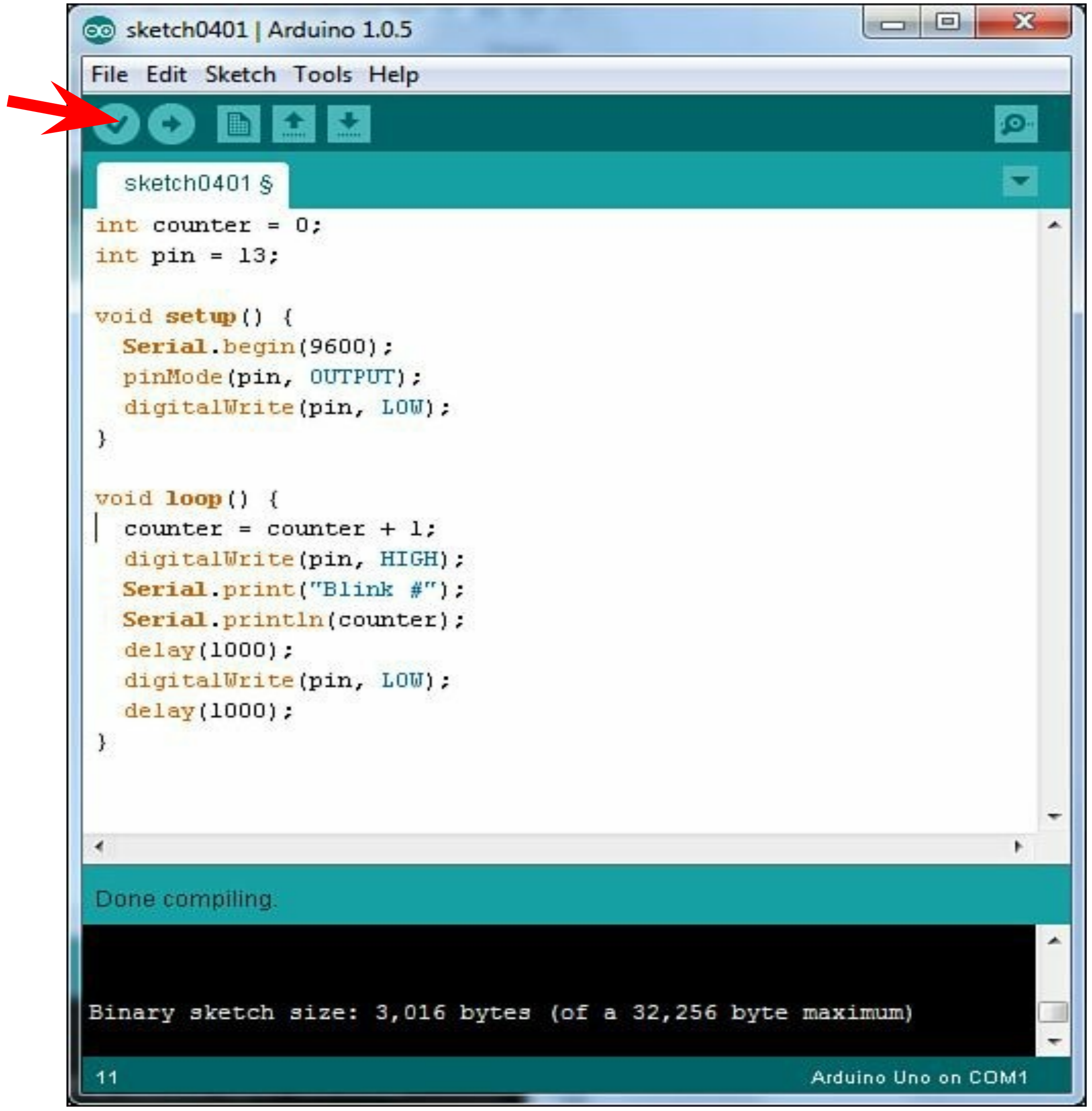
Examples:

#include <Dhcp.h>
#include <Dns.h>
#include <Ethernet.h>

Example Program

```
int counter = 0; int pin = 13;
void setup()
 { Serial.begin(9600);
 pinMode(pin, OUTPUT);
digitalWrite(pin, LOW);
}
void loop() {
counter = counter + 1;
digitalWrite(pin, HIGH);
Serial.print("Blink #");
Serial.println(counter);
delay(1000);
digitalWrite(pin, LOW);
delay(1000);
}
```

Compiling the Sketch



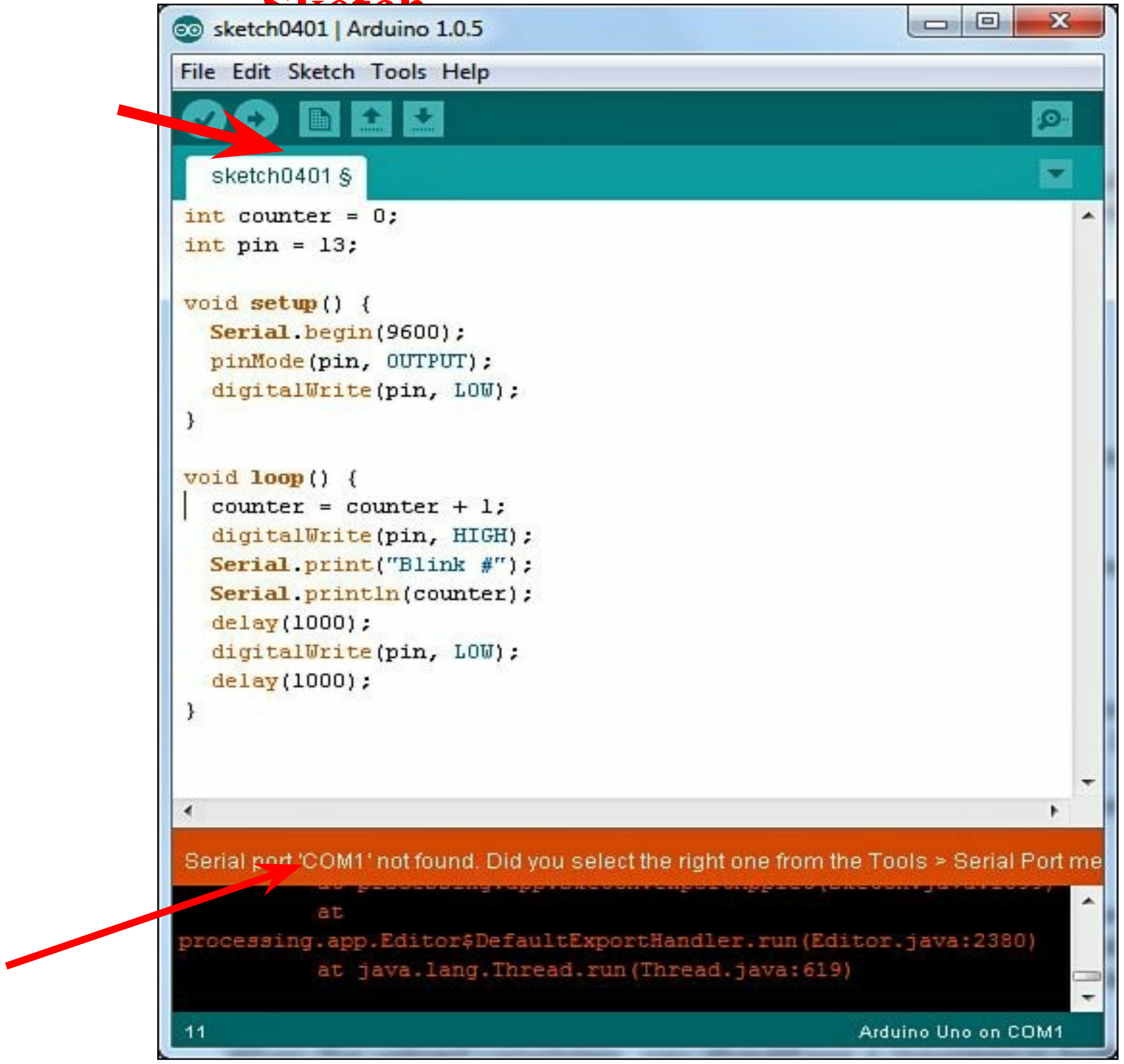
```
int counter = 0;
int pin = 13;

void setup() {
  Serial.begin(9600);
  pinMode(pin, OUTPUT);
  digitalWrite(pin, LOW);
}

void loop() {
  counter = counter + 1;
  digitalWrite(pin, HIGH);
  Serial.print("Blink #");
  Serial.println(counter);
  delay(1000);
  digitalWrite(pin, LOW);
  delay(1000);
}
```

Done compiling.

```
Binary sketch size: 3,016 bytes (of a 32,256 byte maximum)
```

11 Arduino Uno on COM1

Uploading Your Sketch


sketch0401 | Arduino 1.0.5

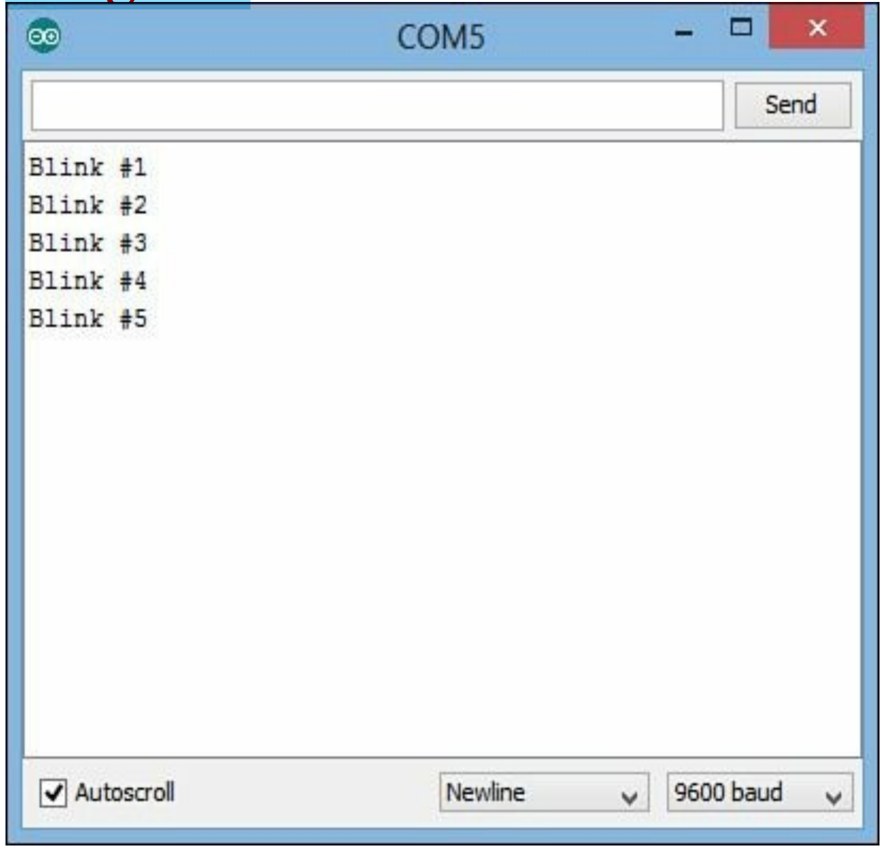
```
int counter = 0;
int pin = 13;

void setup() {
  Serial.begin(9600);
  pinMode(pin, OUTPUT);
  digitalWrite(pin, LOW);
}

void loop() {
  counter = counter + 1;
  digitalWrite(pin, HIGH);
  Serial.print("Blink #");
  Serial.println(counter);
  delay(1000);
  digitalWrite(pin, LOW);
  delay(1000);
}
```

Serial port 'COM1' not found. Did you select the right one from the Tools > Serial Port me

```
        at
processing.app.Editor$DefaultExportHandler.run(Editor.java:2380)
        at java.lang.Thread.run(Thread.java:619)
```

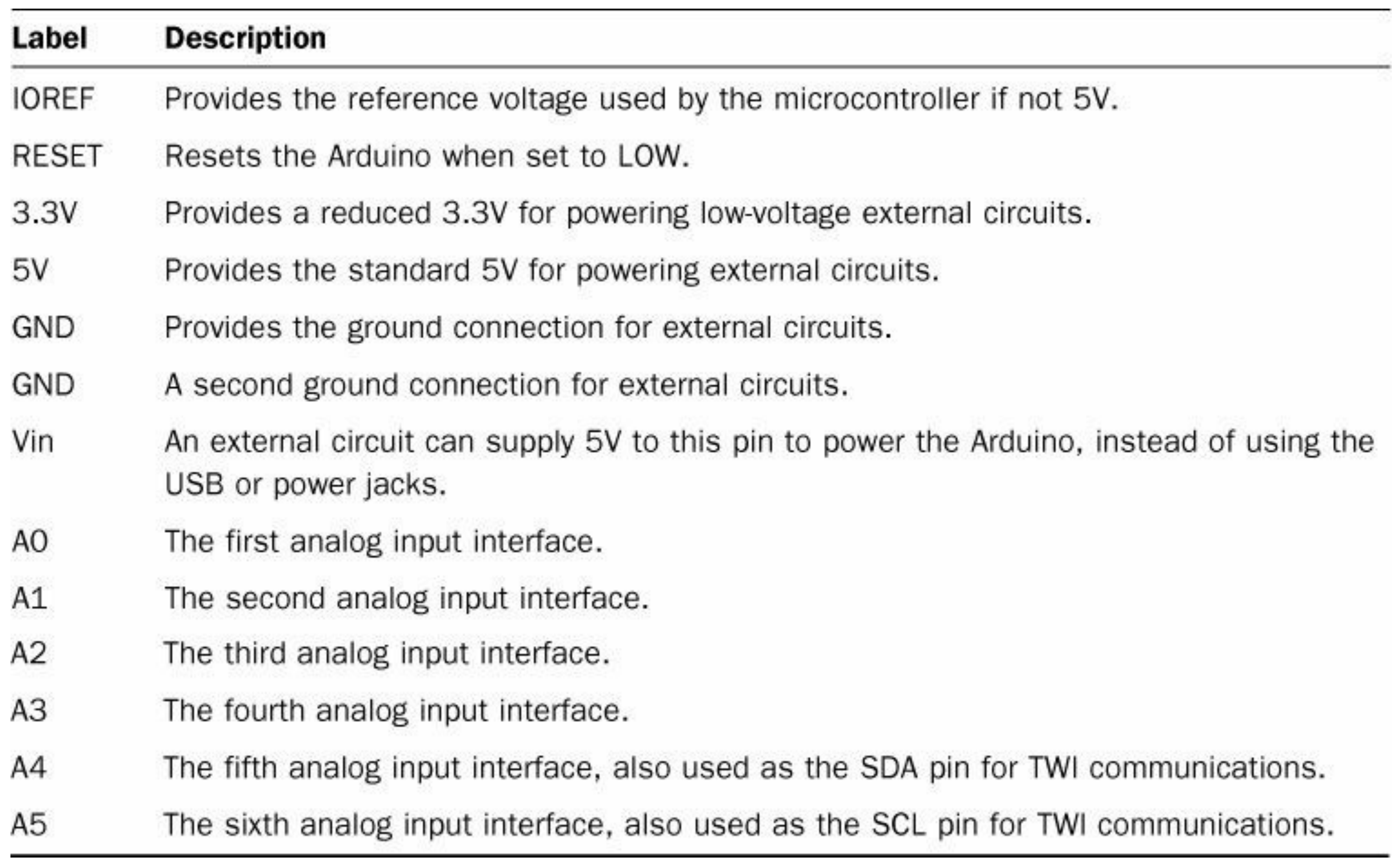
11 Arduino Uno on COM1

Running Your Program



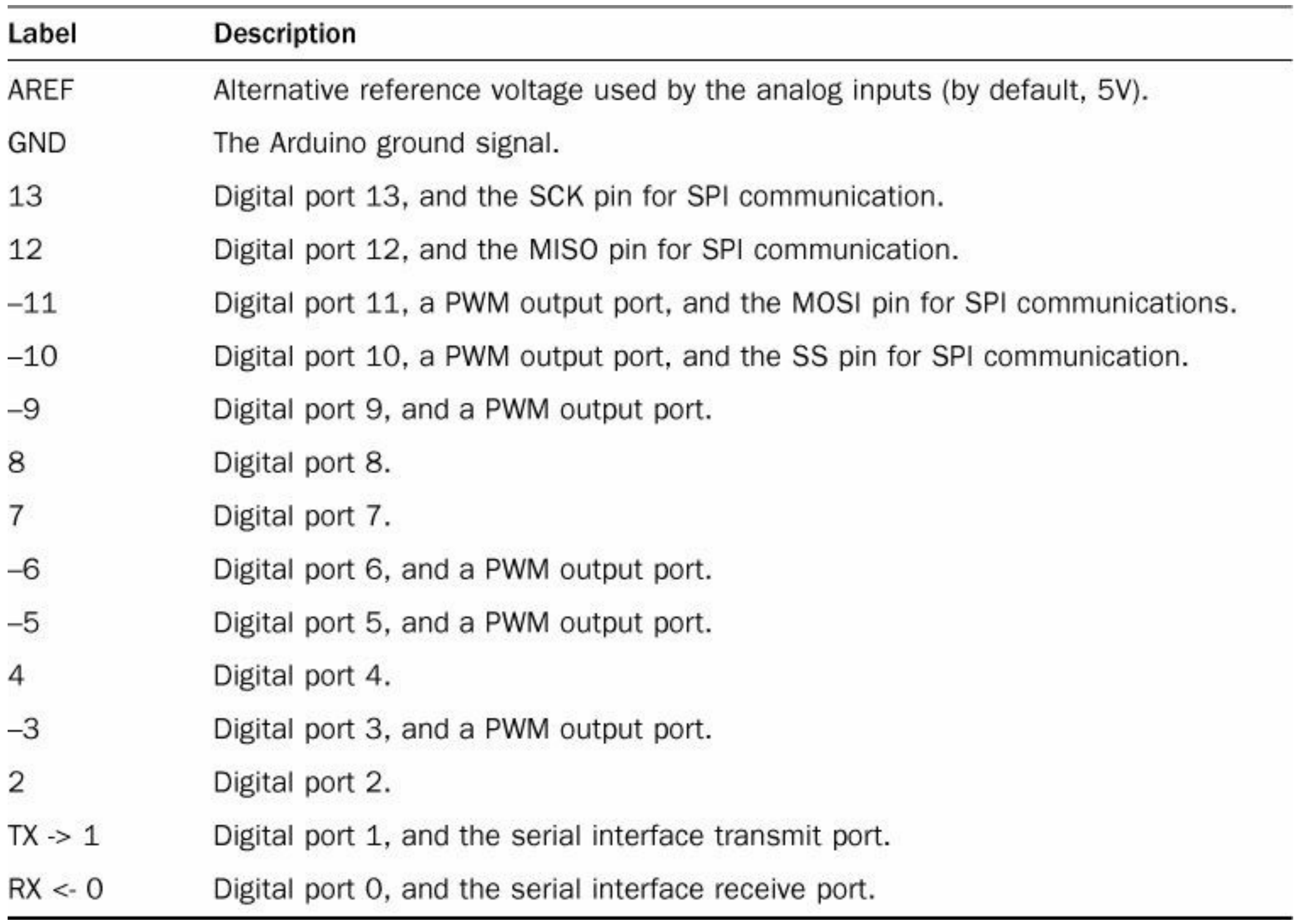
```
Blink #1
Blink #2
Blink #3
Blink #4
Blink #5
```

Interfacing with Electronic Circuits



Label	Description
IOREF	Provides the reference voltage used by the microcontroller if not 5V.
RESET	Resets the Arduino when set to LOW.
3.3V	Provides a reduced 3.3V for powering low-voltage external circuits.
5V	Provides the standard 5V for powering external circuits.
GND	Provides the ground connection for external circuits.
GND	A second ground connection for external circuits.
Vin	An external circuit can supply 5V to this pin to power the Arduino, instead of using the USB or power jacks.
A0	The first analog input interface.
A1	The second analog input interface.
A2	The third analog input interface.
A3	The fourth analog input interface.
A4	The fifth analog input interface, also used as the SDA pin for TWI communications.
A5	The sixth analog input interface, also used as the SCL pin for TWI communications.

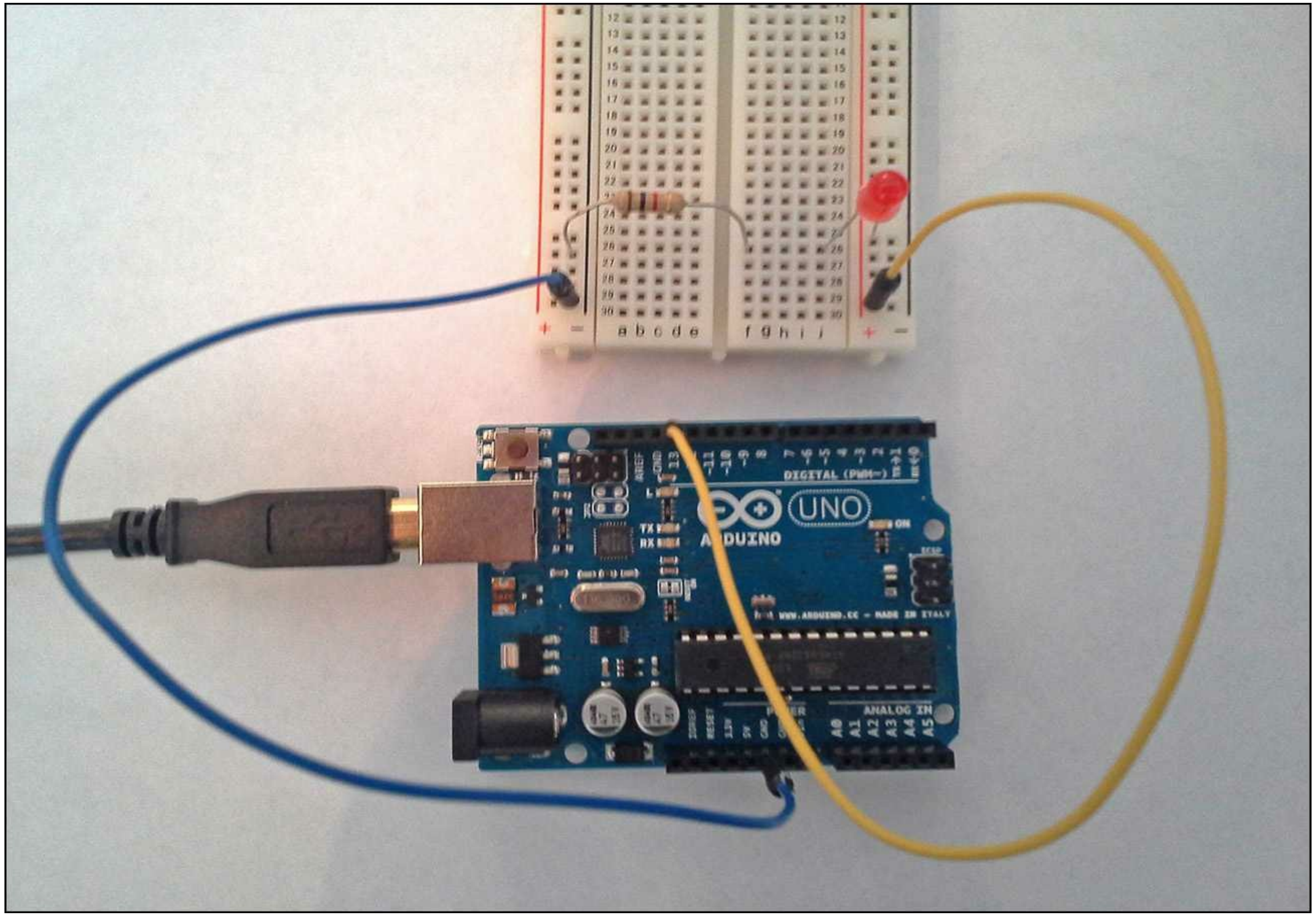
The Arduino Uno Lower Header Socket Ports



Label	Description
AREF	Alternative reference voltage used by the analog inputs (by default, 5V).
GND	The Arduino ground signal.
13	Digital port 13, and the SCK pin for SPI communication.
12	Digital port 12, and the MISO pin for SPI communication.
–11	Digital port 11, a PWM output port, and the MOSI pin for SPI communications.
–10	Digital port 10, a PWM output port, and the SS pin for SPI communication.
–9	Digital port 9, and a PWM output port.
8	Digital port 8.
7	Digital port 7.
–6	Digital port 6, and a PWM output port.
–5	Digital port 5, and a PWM output port.
4	Digital port 4.
–3	Digital port 3, and a PWM output port.
2	Digital port 2.
TX -> 1	Digital port 1, and the serial interface transmit port.
RX <- 0	Digital port 0, and the serial interface receive port.

The Arduino Uno Upper Header Socket Ports

Building Your own project



Using Libraries

- What an Arduino library is
- How to use standard Arduino libraries
- How to use contributed libraries
- Creating your own Arduino libraries

What an Arduino library is

- Libraries allow you to bundle related functions into a single file that the Arduino IDE can compile into your sketches.
- Instead of having to rewrite the functions in your code, you just reference the library file from your code, and all the library functions become available.

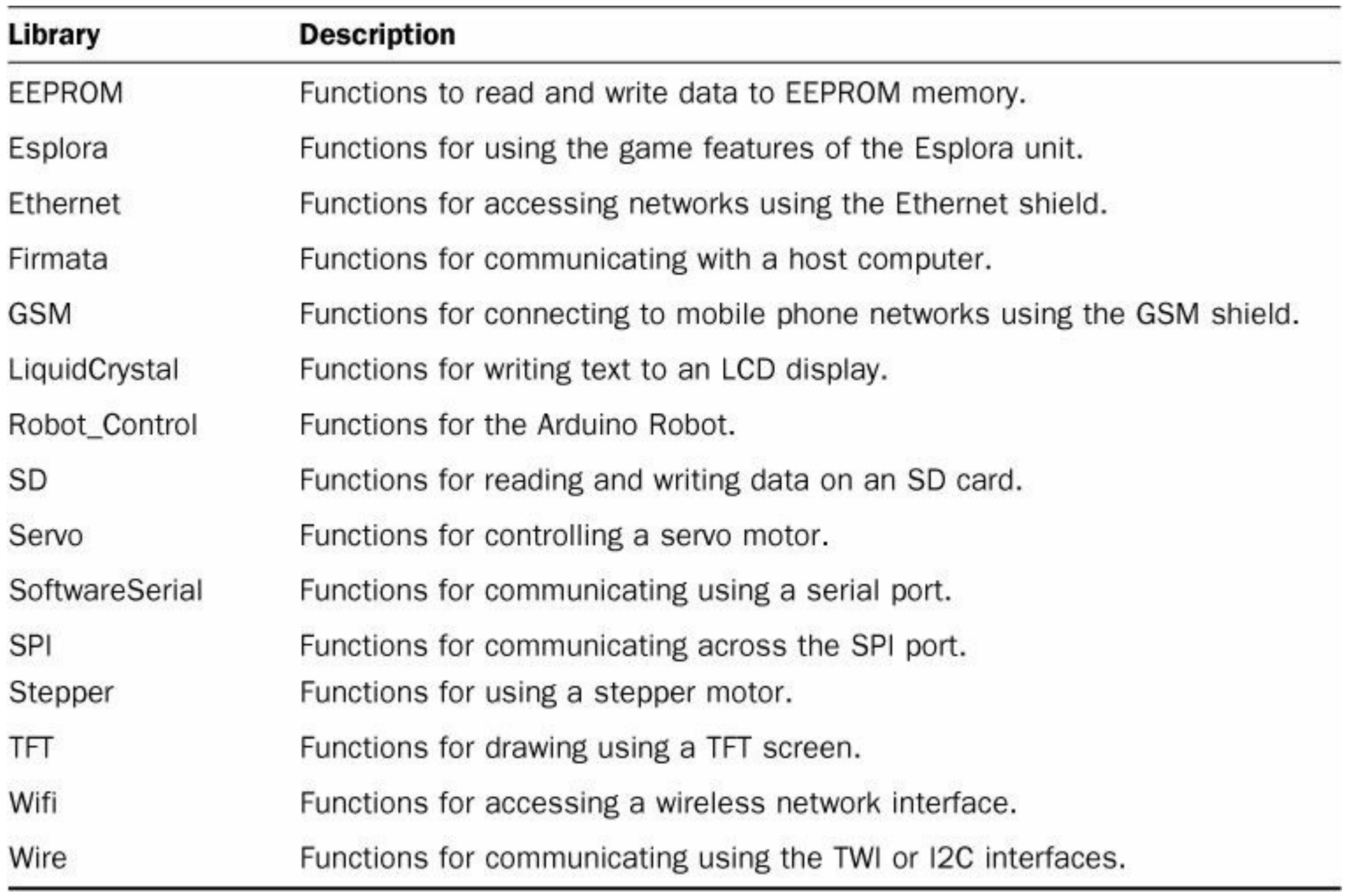
Parts of a Library

- A header file:The header file defines templates for any functions contained in the library. It doesn’t contain the full code for the functions, just the template that defines the parameters required for the function, and the data type returned by the function.

```
        int addem(int, int);
```

- A code file:The code file in the library contains the actual code required to build the function

How to use standard Arduino libraries



Library	Description
EEPROM	Functions to read and write data to EEPROM memory.
Esplora	Functions for using the game features of the Esplora unit.
Ethernet	Functions for accessing networks using the Ethernet shield.
Firmata	Functions for communicating with a host computer.
GSM	Functions for connecting to mobile phone networks using the GSM shield.
LiquidCrystal	Functions for writing text to an LCD display.
Robot_Control	Functions for the Arduino Robot.
SD	Functions for reading and writing data on an SD card.
Servo	Functions for controlling a servo motor.
SoftwareSerial	Functions for communicating using a serial port.
SPI	Functions for communicating across the SPI port.
Stepper	Functions for using a stepper motor.
TFT	Functions for drawing using a TFT screen.
Wifi	Functions for accessing a wireless network interface.
Wire	Functions for communicating using the TWI or I2C interfaces.

Defining the Library in Your Sketch

#include <*library*<*header*>

Referencing the Library Functions

Library.function()

Example: EEPROM.read(0);

Compiling the Library Functions

Using Contributed Libraries

Besides the standard libraries that the Arduino development group provides, Arduino users have created lots of other libraries. These are called *contributed libraries*.

Creating Your Own Libraries