

Chapter 1. Getting Started with Physical Computing

This is a textbook for those who wish to learn the programming basics of **physical computing** using the **Arduino programming software** with either the Arduino Due or Teensy 3.2 **development boards**. The language used in this text is Arduino C, which is based on the popular and robust C language. *I assume the readers of this book are **complete novices** with no knowledge or experience programming computers, building electrical circuits, or working with robots.* I do assume that my readers are familiar enough with a computer to browse the Internet, create folders, and install and run software applications.

1.1. What is Physical Computing and Why Should You Care?

Physical computing is the process by which humans write computer code that tells machines what to measure and how they should interact with their environment. This may sound futuristic, but in fact it is commonplace. Toilets in public restrooms now flush themselves. Thermostats are easily programmed to control the heat and air conditioning systems of buildings and houses. Video games respond to the motion of a handheld console or the human body itself. MP3 players play music at the touch of a button. Automobiles warn you when you are about to back into something. The list goes on and on.

Would you believe that it is possible for **you** to do many of these things (and more) – even if you are a complete novice – simply by reading this book and adding a dash of your imagination! I’ve been teaching students how to program computers for over 25 years, but nothing has captured my student’s imagination like physical computing! For the past 14 years I’ve taught a physical computing and robotics course to roughly 500 students, most of whom started with no programming or electronics experience. Within a few months, these beginning students are able to create some fairly remarkable projects, simply by following the lessons in this book and letting their imaginations run wild. Their projects include robots that can extinguish fires, play soccer, draw images, play music, water gardens, drive small cars, and record scientific data in space and under the ocean. (See this YouTube playlist (<https://www.youtube.com/playlist?list=PLCAD98CAD2337E4EC>) for videos of some of my student’s projects.) Again **YOU** can do these things, too!

In the world of physical computing, problems are resolved with both hardware (electronics) and software (computer code) solutions. When I was a kid growing up, physical computing was reserved for those doing research and development for huge tech companies. People knew how to program computers and people knew how to build electronic circuits, but it wasn’t easy (or even possible) to bridge those two worlds. Even into my college and grad school years, there were computer programmers and there were electrical engineers, and they rarely worked together. This made me sad, because I enjoyed both disciplines. Back then it was very difficult to get a computer to *do something* in the physical world, such as turn on a light, spin a motor, or determine if a person were in the room. Then along came the **embedded controller** and everything changed.

Embedded controllers are small, one chip **microcontrollers** that control or measure some aspect of their environment. The microcontroller, like the one shown at the right, is a remarkable device, integrating a Central Processing Unit (**CPU**), Random Access Memory (**RAM**), Electrically Erasable Programmable Read-Only Memory (**EEPROM**, pronounced “e-e-prom”), input and output (**I/O**) pins, and **timers** into one small, self-contained unit. They are so ubiquitous in today’s high-tech world that most people are unaware even of their existence, let alone how they work and how one might use them. You might say that embedded controllers are hidden in plain sight.

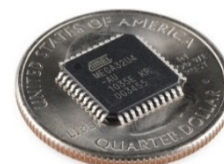


Photo credit: SparkFun
www.sparkfun.com

A *microcontroller* can perform the same calculating and decision-making tasks as the *microprocessor* inside your personal computer (albeit at slower speeds). But the microcontroller can easily do things your computer cannot do, such as turn on a light, spin a motor, or determine if a person is in the room. In this book you will learn how to do these (and other) things.

Physical computing has given rise to the popular [maker movement](#). To many, this is a *sociological revolution* in which regular people are able to develop the tools to build, design, and create machines that can enhance people's appreciation of reality. It is an exciting time to be alive because this is the first time in human history that we have this kind of control! Even though the maker movement is on the rise and the number of embedded controllers in use today is huge, the number of people who know how to program microcontrollers is relatively small. **Why shouldn't YOU be one of them?**

1.2. The Development Environments Covered in this Book

To make the task of programming the microcontroller and integrating it with the world at large, so-called **development boards** (or **development environments**) were created. These boards make it easy to connect wires, sensors, and actuators to the microcontroller. There are a large number of development boards on the market. You might have heard of a couple of them, especially the Arduino Uno or Raspberry Pi. While these are both fine boards, the ones covered in this text are the **Teensy 3.2** and **Arduino Due**, which are shown in Figure xxx.

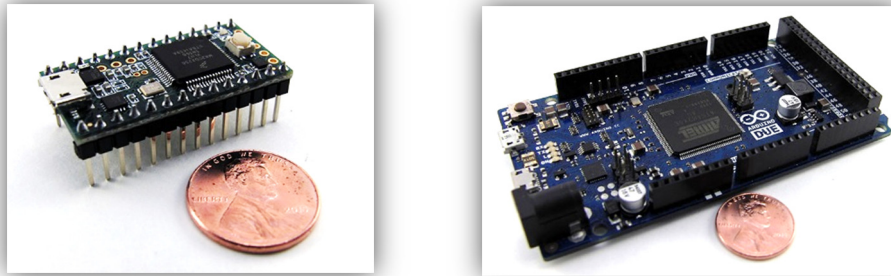


Figure xxx. The Teensy 3.2 (left) and Arduino Due (right) development boards.

So why have I chosen these among the plethora of others? It boiled down to keeping the cost low, the physical size small, and the processors powerful. For my classes, I needed devices that could control the motors of my students' small robots, sense their surroundings using both **analog** and **digital** sensors, while making rapid, complex calculations with a speedy processor.

It is likely that you have already decided which development environment, the Teensy 3.2 or the Arduino Due, is right for you. I, personally, find the Teensy 3.2 is the best choice for me and my students. I find the Teensy product line from [PJRC.com](#) offers the best combination of power, interchangeability, and connectivity at the lowest price. This makes the Teensy an affordable, robust, and easy-to-use classroom platform.

If you have not decided which environment is right for you, a detailed look at the Teensy 3.2 and Arduino Due development boards can be found in Section 7 at the end of this chapter. However, below is a quick break down of each development system.

The Teensy 3.2 Development Board and Patton Robotics PRT3 Motherboard

If you decide to go with the Teensy 3.2 development board, you can purchase everything you need from [Patton Robotics, LLC](#). The Teensy itself can be ordered with male header pins already soldered in place, as shown in Figure xxx or you can save a few dollars by buying the kit and soldering the pins on yourself, as shown in Figure xxx. You can find these under the **Shop >> Motherboards and Controllers** menu at [pattonrobotics.com](#).

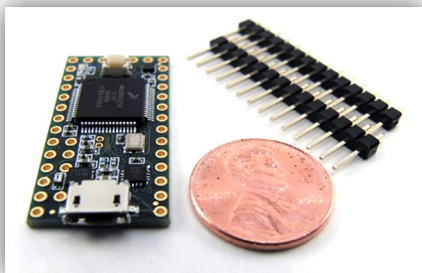


Figure xxx. The Teensy 3.2 kit. The male header pins must be soldered onto the board by the customer.

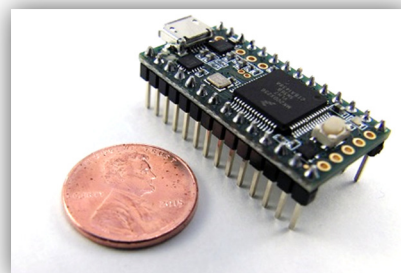


Figure xxx. The Teensy 3.2 with header pins pre-soldered into place.

Make sure you buy the Teensy 3.2, which has 28-pins and a powerful ARM processor. If you have an older Teensy, such as the Teensy 3.0 or 3.1, nearly all of this book will work fine with your board. However, you should avoid the older Teensy 2.0, which uses a less powerful and older AVR processor. The 2.0 development board will work for many applications discussed in this book but not all of them.

The Teensy 3.2 is all you need to get started. In fact, it is all you need to get through the programming basics of this first volume of this book. However, a time will come when you will want to jump into Volume Two of my text and connect a battery pack, LEDs, buzzers, sensors, motors, etc. to your controller. When that time comes, I recommend using the **Patton Robotics Teensy Motherboard** from Patton Robotics made specifically for the Teensy 3.x family. Known as the **PRT3**, it can be purchased as a kit (see Figure xxx) or fully assembled without and with the Teensy microcontroller (see Figures xxx and xxx, respectively). While you *can* fabricate your own motherboard using a solderless breadboard, you can't beat the small footprint, ease of use, and rugged dependability of the handy PRT3 motherboard!

If you already have the PRT3 Motherboard, take care to insert the Teensy with the proper alignment, as shown in Figure xxx. Note the orientation arrow: the USB connector on top of the microcontroller should be positioned between the two switches as shown in the image. Insert the microcontroller into the motherboard socket by pressing firmly along the microcontroller's edge with your thumbs. Also take care that the pins are not bent when inserting them into the socket.

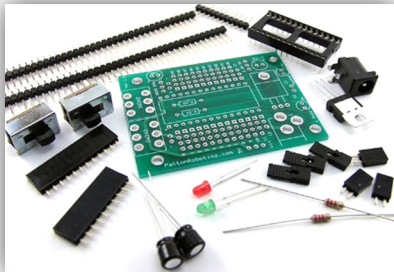


Figure xxx. The Patton Robotics Teensy Motherboard Kit. You can save a few dollars by assembling the parts yourself. It's fun and educational.

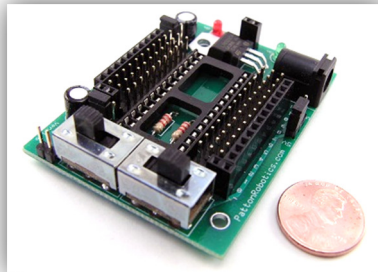


Figure xxx. The fully assembled Patton Robotics Teensy Motherboard or PRT3. Here, the Teensy development board has not yet been inserted.

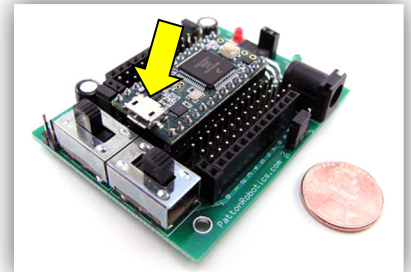


Figure xxx. The Teensy 3.2 is inserted into a Patton Robotics PRT3 Motherboard. Make sure the Teensy controller is inserted into the motherboard as shown. Take care that the pins are not bent when inserting.

The Arduino Due Development Board

If you elect to use the Arduino development board, make sure you buy the **Arduino Due**, shown in Figure xxx. This board is equipped with the same powerful **ARM processor** as the Teensy 3.2. Be careful **not** to buy the more common Arduino Uno, shown in Figure xxx, which uses the older and less powerful AVR processor. The Uno board will work for many applications discussed in this book but not all of them.

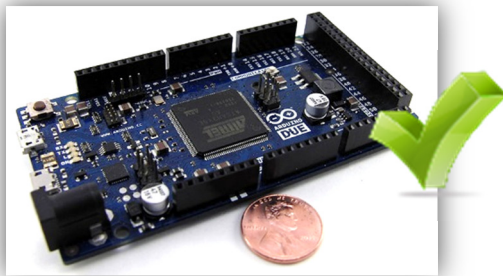


Figure xxx. The Arduino Due development board. This board used the same powerful ARM processor as the Teensy 3.2.

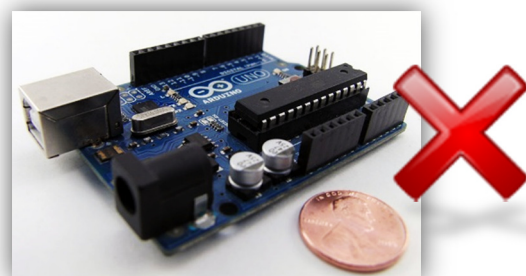


Figure xxx. The Arduino Uno development board. Refrain from using this more common, but less powerful, development board.

You can purchase the Arduino Due from a number of vendors, including Maker SHED (www.makershed.com), Spark Fun (www.sparkfun.com), Jameco (www.jameco.com), and the Arduino Store (arduino.cc/en/Main/Buy).

While I believe the Teensy 3.2/PRT3 combination is the best option for classroom instruction, this volume will also show you how to use the Arduino Due board to act as the brain and central nervous system of your physical computing devices.

1.3. Required Hardware for this Book

Before you can get started programming your microcontroller, you must gather the following few essential pieces of hardware.

Your Computer

To program either the Teensy 3.2 or the Arduino Due, you'll need a computer with a USB port. If your computer runs Windows, Mac OS, or Linux you can run the software used throughout this book.

A USB programming cable.

You will need a micro-B USB cable to serve as the programming cable between your computer and your development board. The development board vendors sell these as well, or you can use just about any micro-USB cable you might have around the house.



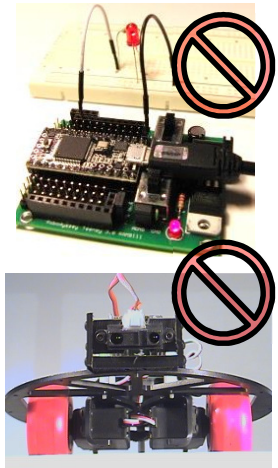
A Development Board

This book is written around both the **Teensy 3.2** and **Arduino Due** development environments. You will need one of these development boards before proceeding. While I believe the Teensy 3.2/PRT3 combination is the best option for classroom instruction, this volume will also show you how to use the Arduino Due board to act as the brain and central nervous system of your physical computing devices. To learn more about these powerful devices read Section 1.2 above and Section 1.7 below. If you use the Teensy 3.2, I highly recommend using it in conjunction with the **PRT3 Motherboard** from Patton Robotics.

Robots and Peripheral Devices

You don't need anything other than the items listed above to start reading this book and begin writing code for your development board. However, in Volume Two of my text, I explain how the attrition of an inexpensive **breadboard** and a few **electronic components** can turn your little development board into a work of art. In Volume Two, I will show you how to use your microcontroller with a variety of physical computing devices such as LEDs, buzzers, robots, sensors, motors, and other actuators.¹

Then, of course, there are robots! It goes without saying that robots are very appealing and are a fantastic way to introduce programming, electronics, and engineering topics to students of all ages. Over my many years of teaching, I have learned that students with access to kinesthetic tools like robots are more likely to enjoy their programming class and will stretch themselves well beyond their comfort zone. In my classroom every student has their own **OneBot robot** from **Patton Robotics**, shown at the right. This durable, wheeled robot fully integrates with the Teensy 3.2/PRT3 Motherboard combo as well as any of the Arduino development boards, and it is the centerpiece of the robot motion chapters in Volume Two of this text.



1.4. Install the Software Required for this Book

The Programming Language

Arduino is the name of the computer language used to program your development board. This book is dedicated to teaching you Arduino, which is based on the well-established and versatile **C language**.

The Software or Integrated Development Environment (IDE)

The fancy name for any software used by programmers to write code is called the **Integrated Development Environment**, or **IDE**. The programmer uses the IDE to write computer programs, which are called **sketches** by the Arduino community.² Once the computer code is written, the IDE is then used to translate the sketch into a language that is understood by your microcontroller and uploads the translated code to your development board.

Whether you use the Teensy 3.2 or the Arduino Due, you will need to download the Arduino IDE. The Arduino IDE, which works equally well on computers running **Windows**, **Mac OS X**, and **Linux**, makes programming the Arduino and Teensy boards quite easy for just about anyone.

¹ See Appendix xxx for a list of all the equipment I use in each volume of my text.

² They are named "sketches" to appeal to the artist in us all.

Download the Arduino IDE

Download the **latest, non-beta version** of the Arduino IDE.³ (**Teensy 3.2 users** should read the note in the gray box below before downloading the file.) You can find the file at Arduino's download page:

<http://www.arduino.cc/en/Main/Software>

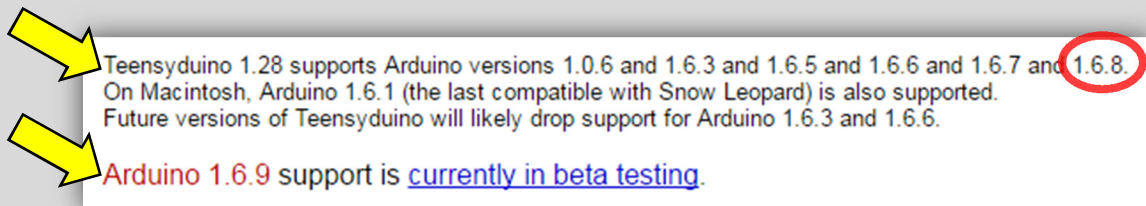


Make sure to download the version that is right for your computer system. If you are a Window's user and you are not your computer's administrator, make sure you download the **ZIP file for non-administrator installation**.

A note to Teensy 3.2 users:

Before downloading the Arduino IDE, check out this page: http://www.pjrc.com/teensy/td_download.html, and note which version of the Arduino IDE is supported by a piece of software called **Teensyduino**. In the next step you will download Teensyduino, but for now, simply make sure that you only download a version of the Arduino IDE that is supported by the latest version of Teensyduino.

For example, in early June 2016, Teensyduino supported Arduino version 1.6.8, but it did **not** support 1.6.9, as shown by this screenshot:



Therefore, Teensy 3.2 users in June 2106 should have downloaded Arduino 1.6.8, and avoided the most recent version of the Arduino IDE! You can find older versions of the IDE under the "**Previous Releases**" section of the Arduino Downloads page.

Install the Arduino IDE

After the file is downloaded, go ahead and **install** the Arduino IDE software.

If you are using the **Arduino Due** board, this is all you need to do for now and can skip to Section 1.5, below.

*Users of the **Teensy 3.2** development board should make note of the location of the Arduino program files, for you will soon need to install another piece of software called **Teensyduino** in that same location.*

³ At the time of this publishing, the most current version was 1.6.9. The download is free, but consider making a small donation.

Download and Install Teensyduino for the Teensy 3.2

If you are using a **Teensy 3.2** development board, you must download and install one more piece of software called **Teensyduino**. (Again, if you are an Arduino Due user, skip to Section 1.5.)



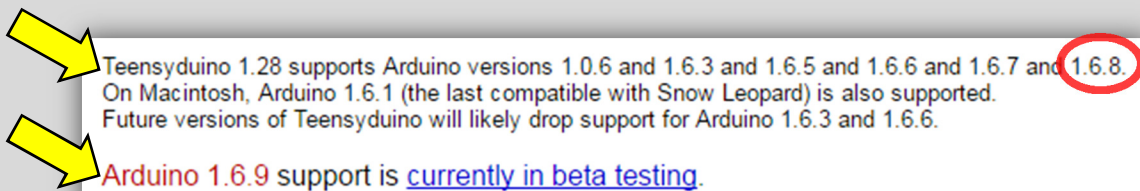
Teensyduino is a free add-on that copies all the necessary Teensy files into Arduino, which makes it possible for Teensy users to write and upload their code using the Arduino IDE.⁴ You can find this free download here:

http://www.pjrc.com/teensy/td_download.html



Once you are on the PJRC download page, Teensy users should select the appropriate file for your operating system and **download** it to your computer.

Once again, Teensy users should make sure to only download a version of the Arduino IDE that is supported by the latest version of Teensyduino! For example, in early June 2016, the Teensyduino website at www.pjrc.com/teensy/td_download.html showed that Teensyduino supported Arduino version 1.6.8, but did **not** support version 1.6.9. Therefore, Teensy 3.2 users in June 2016 should have downloaded Arduino 1.6.8, and avoided the 1.6.9 version of the Arduino IDE, which was in beta testing at that time as shown:



You can find older versions of the IDE under the “**Previous Releases**” section of the Arduino Downloads page.

Once the Teensyduino application file has been downloaded, follow the installation instructions below based on which computer system you are using:

- If you are on a **Mac**, simply run Teensyduino to install the software. When the window below pops up, click on it to run the installer. See page 7 for details about this installation.



⁴ Learn more about Teensyduino at www.pjrc.com/teensy/teensyduino.html.

- If you are on a **Windows-based PC**, locate the “teensyduino.exe” file, which is probably in your **Downloads** folder. You may need to run Teensyduino as an **administrator**, by right-clicking on the “teensyduino.exe” and select “Run as administrator”, as shown in Figure xxx below. Keep reading for details about this installation.

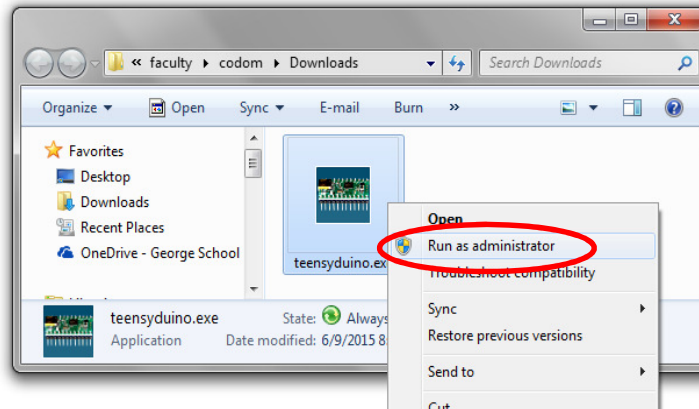
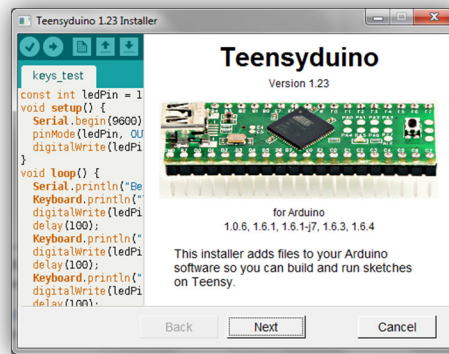
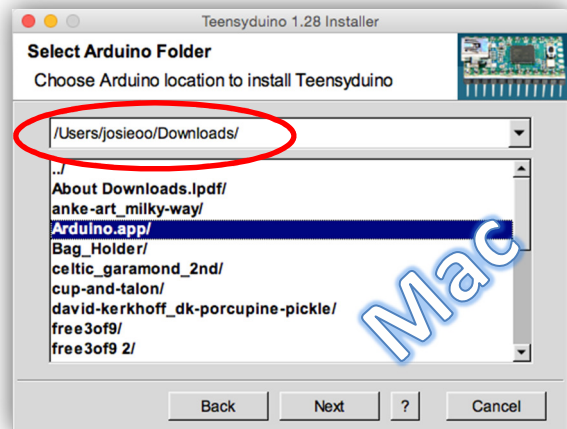
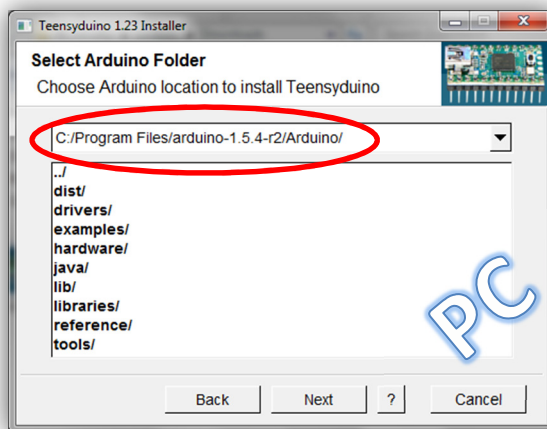


Figure xxx. On a Windows machine, run the Teensyduino installer by right-clicking on the file in the Downloads folder, and selecting “Run as administrator”.

Both Mac and PC users should see the following window once the Teensyduino installation software is opened:



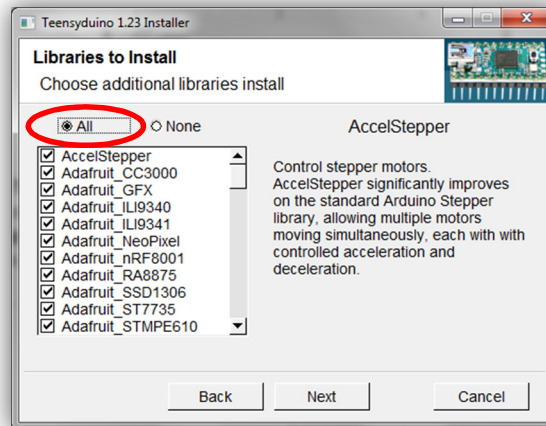
Clicking the **Next button** opens a new window which checks to see if the USB Serial driver has been installed. It will take a moment or two for the computer to determine if the proper USB drivers are installed. USB drivers *must* be installed, so click on the **Next button**, which takes you to this window (PC on the left, Mac on the right):



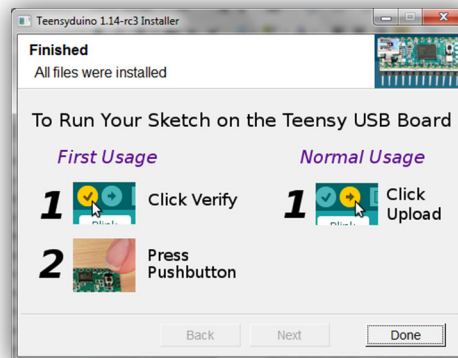
You must install Teensyduino into the **same folder** as your Arduino program file! The **Next button** will remain grayed-out until you locate the correct folder. It is likely that the default path will be the one you need. If not, browse to the folder in which Arduino was installed. **Mac users**, try looking in your Applications or Download folders. When this is done, press the **Next button**.



The window that opens will ask that you select which additional libraries you want imported into the Arduino program. I recommend that you chose to import **all** of them, as shown below. (Installing all of the libraries will make it super-easy to incorporate a *huge* array of third party sensors and actuators into your programs.) Continue the installation process by pressing the **Next** button.



In the window that appears, press the **Install** button to load the Teensyduino software and selected libraries. It will take a few minutes to install everything, but when the process is complete you will see the window below. I will explain this somewhat cryptic message to you later. For now, press the **Done** button and get ready to test your setup!



1.5. Setup Your Gear for the First Time

Now that all the equipment has been gathered and the software has been installed, it is time to set up the Arduino software so it properly recognizes your development board. It doesn't take long to do this, but you should carefully read and follow each of the steps below.

Electrically Insulate Your Device

Warning! The underside of your microcontroller and development board is covered with little metal pins or contacts. Before connecting the micro-USB cable to your embedded controller or development board, *be sure that your device is insulated from any metal* that could cause an **electrical short**. Placing the board down on anything metal can cause an electrical short that is likely to fry your board. Beware of metal tables and metallic computer cases! Even something as small and as innocuous as a paperclip or staple could cause a short and destroy your board, so take this warning to heart!



If you are using the Arduino Due or if your Teensy is inserted into the PRT3 Motherboard, you may want to get in the habit of always placing it on some non-metallic surface such as wood, paper, plastic, or cardboard, as shown in Figure xxx. My students use the Teensy 3.2 microcontroller with the PRT3 motherboard, but they do not build their motherboards right away, so at the start of the year I have them plug their Teensy boards to a **solderless breadboard**



or simply keep them in pink static-proof plastic bags (see Figure xxx) to eliminate the possibility of an electrical short.⁵ If you decide to use a breadboard here, place the micro-USB socket near one end of the breadboard, as shown in Figure xxx, so the programming cable can be inserted without putting pressure on the socket.

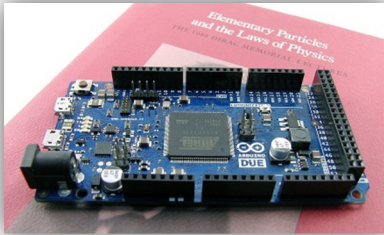


Figure xxx. Protect your development board by working with it on an electrically insulating surface such as wood, paper, or plastic. Keep the pins underneath away from metal!

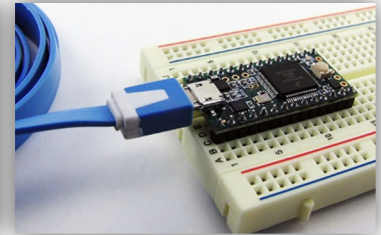
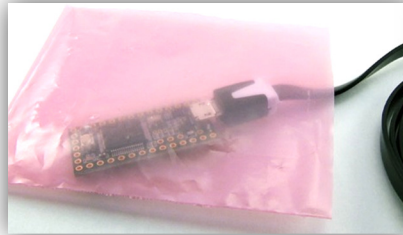


Figure xxx. To eliminate the possibility of an electrical short, you can simply place the Teensy in an anti-static plastic bag (left). Or, if the header pins have been soldered in place, it can be inserted into a solderless breadboard (left). Note the Teensy is placed near the end of the breadboard so the programming cable can be inserted easily.

Connect Your Development Board to Your Computer

Plug the large end of the USB programming cable into one of your computer's USB ports, as shown in Figure xxx. It is best if you use the same port each time you sit down to program your development board.



Figure xxx. Plug the USB programming cable into the computer's USB port. Try to use the same port each time you program our development board

Ensure your development board is not resting on anything metallic and then insert the tiny end of the programming cable into the micro-USB slot on your development board. For **Teensy 3.2 users** there is only one micro-USB port from which to choose, as shown in Figure xxx below. Users of the **Arduino Due** will note there are two micro-USB sockets on their board, as shown in Figure xxx. Turning the board over you will notice the two ports are labeled "Native USB" and "Programming", as shown in Figure xxx. Plug your programming cable into the "**Programming**" port, as shown.

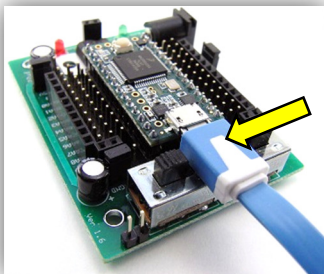


Figure xxx. Teensy users should plug the small end of the programming cable into the micro-USB socket on their development board.

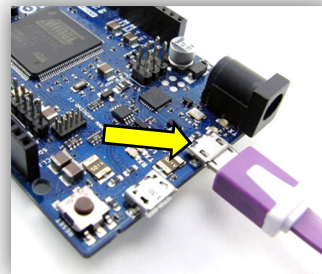


Figure xxx. Arduino Due users should plug the small end of the programming cable into the micro-USB socket labeled "Programming".

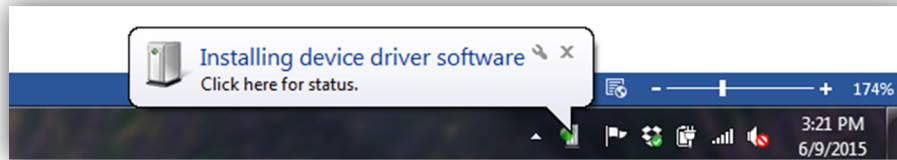


Be careful to get the orientation correct. The cable is supposed to fit snugly into the socket, but **do not force it!** Also, take care that you do not pry the tiny micro-USB connector off of the board; the cable should be inserted horizontally!

If your development board is brand new and you have powered it up for the first time, you may see its onboard LED blinking on and off. This is your board's way of saying hello. (Feel free to say hello as well.)

⁵ See Chapter xxx for more on the solderless breadboard.

If this is the first time you've plugged your board into this particular USB port, allow your computer a few moments to recognize this new piece of hardware and give it time to install the necessary drivers! This may take a few minutes, so be patient while the drivers are installed. Windows users will see a message in the system tray (similar to the one below) stating that the drivers are being installed.



This is an important step. If you rush it, the software will not make a proper handshake with the hardware and you will have to start over. Do yourself a favor: plug in the development board to your computer and then go grab a snack. By the time you return, the board should be ready to program! Windows users will see another notification pop up in the system tray that the device has been properly installed and that it is ready to be used.

If you ever plug your development board into another USB port, you will need to give the computer time to load these drivers once again! (Hence my recommendation to always use the same USB port each time you program the device.)

Tell the Arduino IDE which Development Board You Are Using

Open the Arduino software.⁶ In the IDE's **programming window** that opens, you should see the default sketch shown in Figure xxx below. The sketch is practically blank, with only the bare-bones shell of a program displayed in the code window.

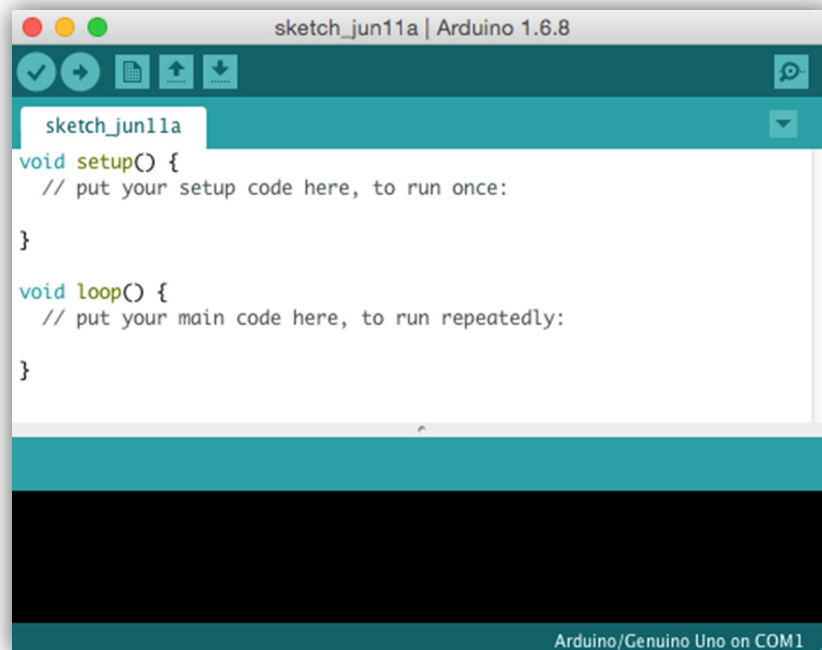


Figure xxx. The default blank program in the IDE's programming window.

⁶ The program name is **arduino.exe** for PC users, and **arduino.app** for Mac users.

Before Arduino can be used to upload and run your sketches, you must tell the IDE which development board you are using. You will only need to do this **once** if you use the *same* development board each time. Do so by clicking on the **Tools** menu, then change the **Board:** to your particular device:

1. **Teensy 3.2 users** should see the options shown in Figure xxx below. Select the **Teensy 3.2** board as shown. If these options are not available to you, walk through the steps for installing Teensyduino once again. The instructions for doing so can be found on page 6.

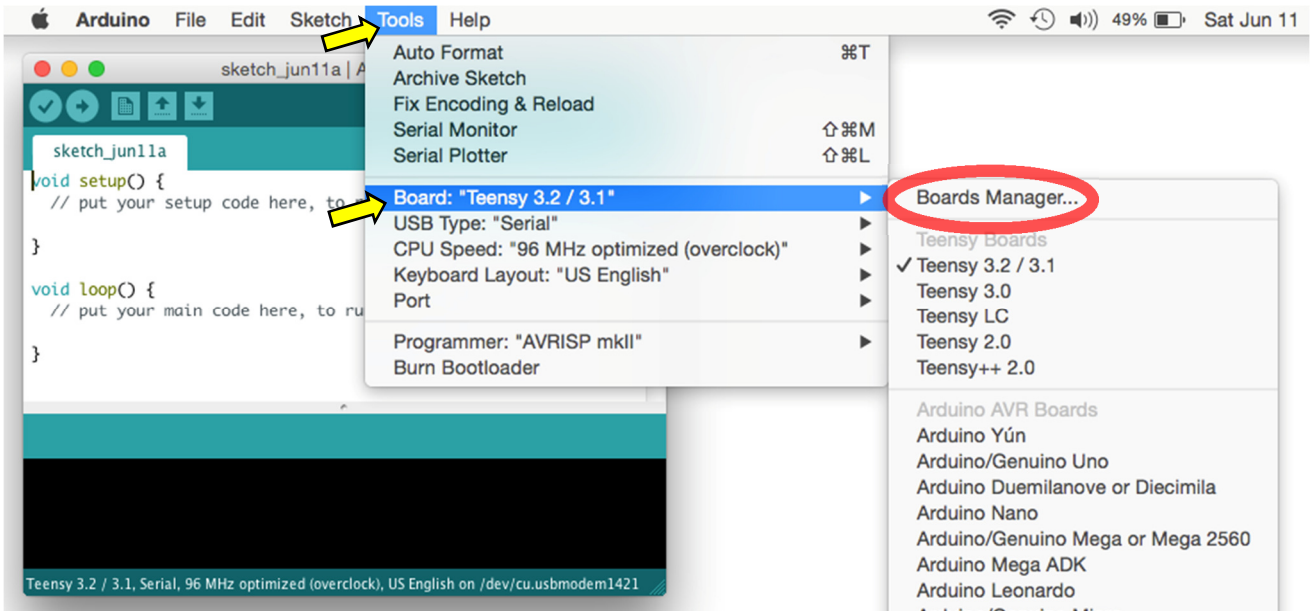


Figure xxx. Follow these steps to tell the Arduino IDE that you are using the Teensy 3.2 development board.

2. **Arduino Due users** need to install some more files specifically for the Due board. Click on **Tools >> Board: >> Boards Manager**, as shown below in Figure xxx.

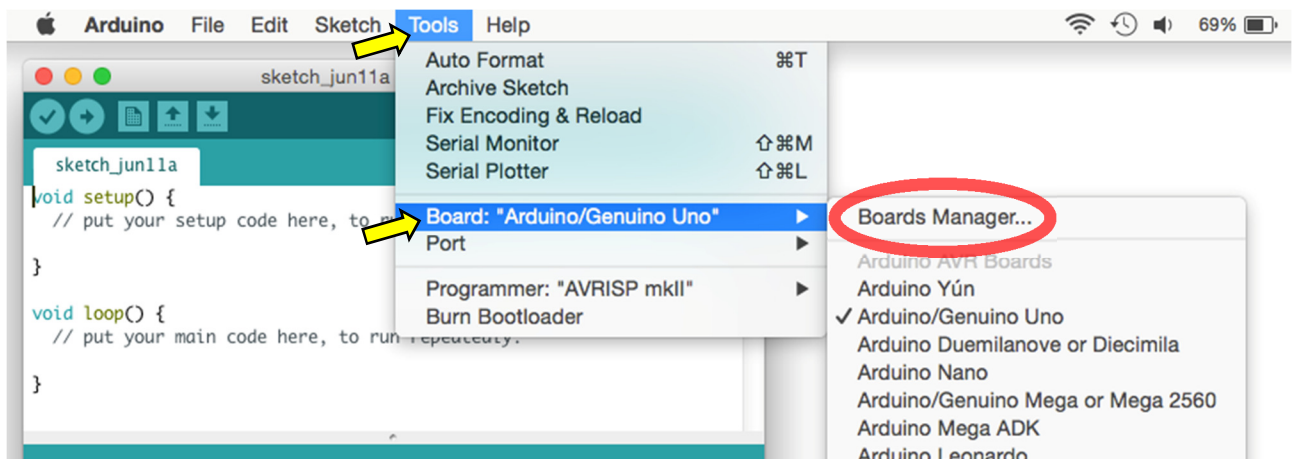


Figure xxx. Follow these steps to add further installation files for the Arduino Due development board.

In the **Boards Manager** window that opens, locate the package that includes files for the Arduino Due board. As shown in Figure xxx, the required package is titled “**Arduino SAM Boards**”.

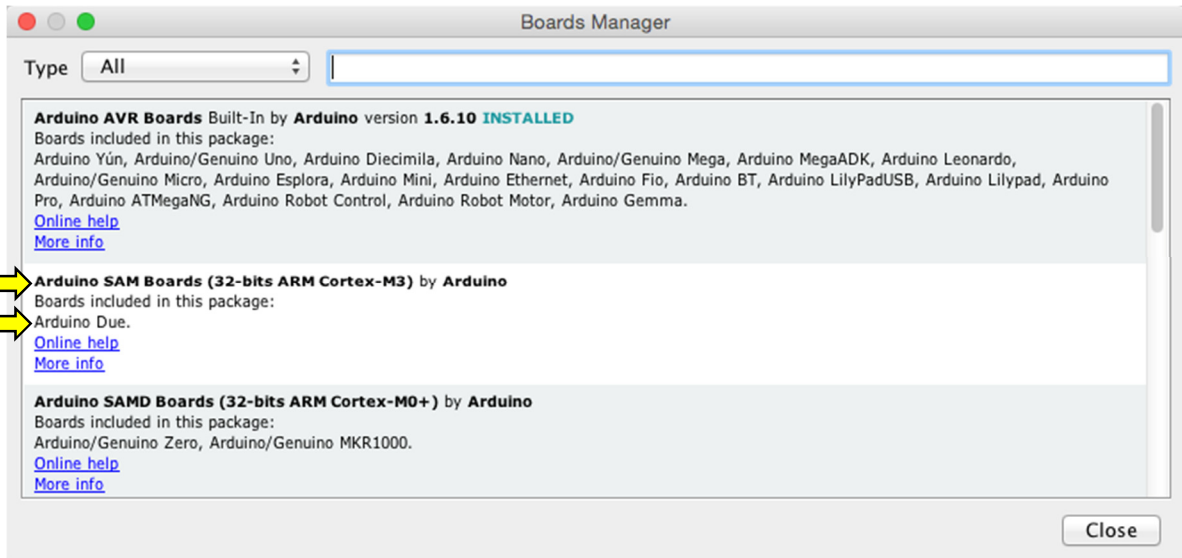


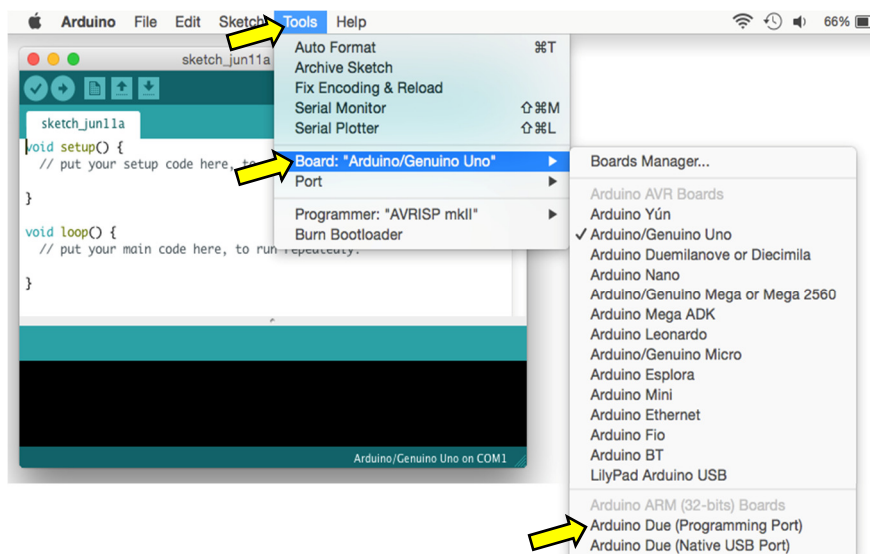
Figure xxx. Find the package of installation files specifically for the Arduino Due development board.

Clicking in the space will make visible an **Install button**, which is shown in Figure xxx. Select the appropriate version of the Arduino software, then click on the **Install button** to install the extra files needed for the Due board.



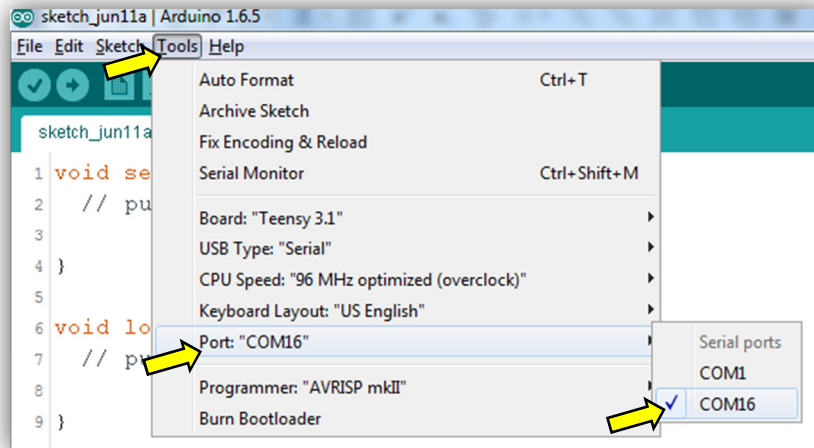
Figure xxx. Select and install the Arduino Due installation files.

When the files have finished installing, close the **Boards Manager** window and return to the Arduino IDE. Now tell the IDE which board you are using by clicking on **Tools >> Board: >> Arduino Due (Programming Port)** at the bottom of the list, as shown:



Tell the Arduino IDE which Communications Port You Are Using

Next, you must tell the Arduino IDE which **communications port** on your computer is used to talk with your development board. To do this, click on **Tools >> Port** from the IDE's menu bar, and then select the correct port, as shown below:

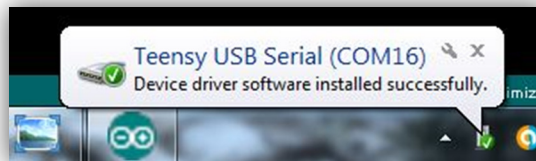


This is easier said than done, for the correct port depends on which board you are using (Teensy or Due), which type of computer you are using (PC or Mac), and which physical USB port your programming cable is plugged into. In other words, there will be a great deal of variability here. The good news is if you use the same development board and plug it into the same USB port each time, you only have to do this **once!**

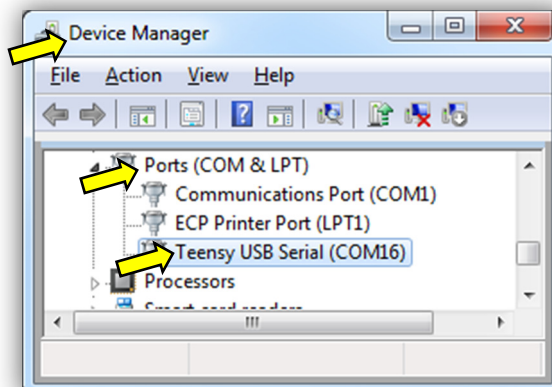
Here are some general guidelines for selecting the correct port:

- For **PC users**, selecting the correct port is usually a straightforward process. Often the only port listed is the correct one. The ports will be labeled with the "COM" prefix, such as COM3 or COM4. If COM1 is listed, it is probably **not** the correct port. If there are multiple ports to choose from, you can always employ a trial-and-error method to find the correct port.

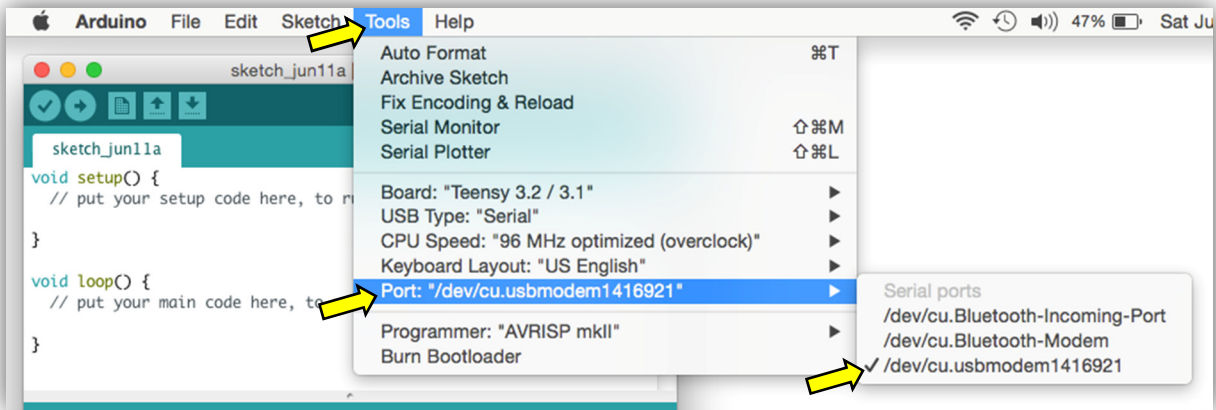
Trial-and-error is not necessary for PC users. For example, if you paid attention to your computer monitor when you first plugged in your development board, a message popped up telling you which was the active port. For instance, in the image below, we see that my Teensy board is connected to COM port 16.



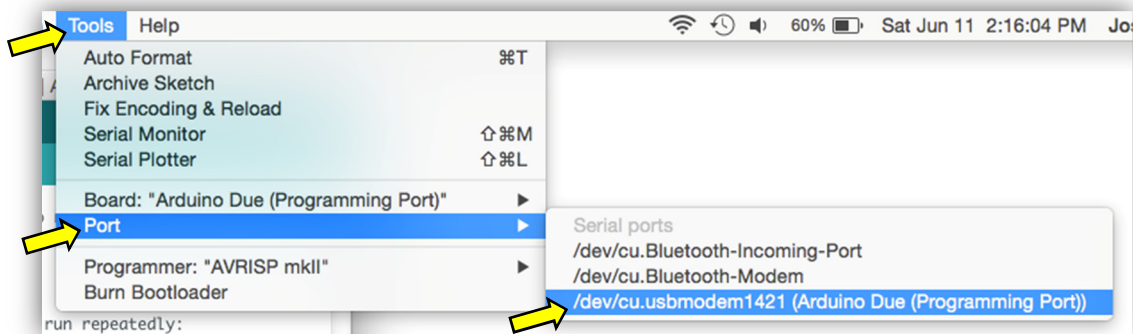
If you did not catch the message and are an administrator of your PC, you can examine the **Ports (COM & LPT)** section of your computer's **Device Manager**, as shown below:



- For **Mac users**, this task seems to be more troublesome. On a Mac, the label is usually associated with a number. For example, it may be listed as something similar to, “/dev/cu.usbmodem1416921”, as shown below:



For **Arduino Due users** on a **Mac**, the port label usually includes the words “Arduino Due”, such as, “/dev/cu.usbmodem1412 (Arduino Due (Programming Port))”, as shown below:



There is no question that **Mac users** have a more difficult time keeping the lines of communication open between the computer and the development board. Often the Port menu under the Tools menu is grayed-out. Restarting the Arduino software, unplugging the programming cable, and/or inserting the programming cable into another USB port are all things to try if you lose your port. If these things don't work, try rebooting your computer.

You Know You Are Ready to Proceed When...

You will know that the Arduino software is properly installed and set up when you see the name of your development board and the proper port number displayed on the bottom right of the Arduino IDE. For example, in the image below it is clear that the board used is the Teensy 3.1 and it is connected to COM3. This board is ready to be used!

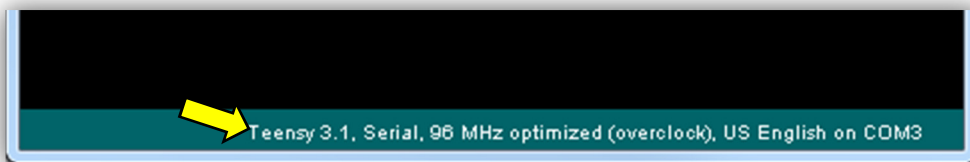


Figure xxx. You know the software is setup properly and ready to go when the name of your development board and the communications port is displayed at the bottom right of the Arduino IDE!

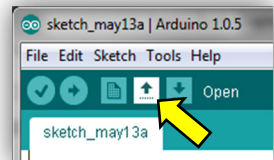
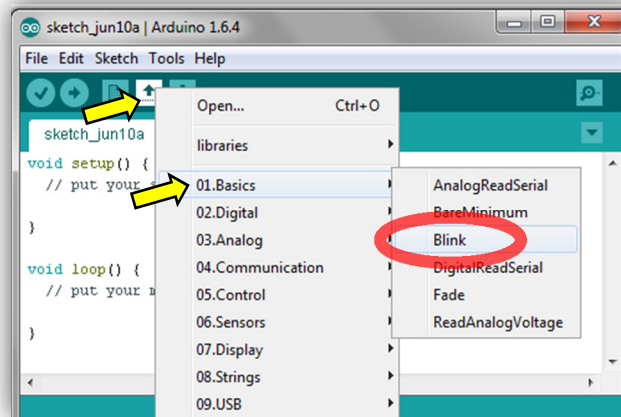
1.6. Test Your Gear with the Blinking LED Example Program

Now that all the equipment has been gathered and the software has been installed, you are ready to program your development board using the Arduino IDE. Now we are getting to the fun part! However, before you begin writing code from scratch, you should first *test your connection* by running a simple example program called “**Blink**”, which is pre-installed in the IDE. This sample code will make the LED on your development board blink once per second.

Open the Blink Sketch

In the world of Arduino, computer programs are called *sketches*. There are two ways to load the “**Blink**” sketch into your code window:

- One way is to select **File >> Examples >> 01.Basics >> Blink** from the menu bar.
- Another way is to use the **Open File button** (shown at the right). Click on the button and select **01.Basics >> Blink** from the drop-down menu, as shown in the figure below:



Stretch your IDE’s programming window so you can view all the coded commands for the **Blink** sketch. There are slight differences in the **Blink** code between the Arduino Due version and the Teensy 3.2 version. I’ve reproduced both versions below, starting with the Arduino Due version:

The Blink Code – Arduino Due Version

```

/*
  Blink
  Turns on an LED on for one second, then off for one second, repeatedly.

  Most Arduinos have an on-board LED you can control.  On the Uno and
  Leonardo, it is attached to digital pin 13.  If you're unsure what
  pin the on-board LED is connected to on your Arduino model, check
  the documentation at http://www.arduino.cc

  This example code is in the public domain.
  */

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin 13 as an output.
  pinMode(13, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(13, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);              // wait for a second
  digitalWrite(13, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);              // wait for a second
}

```

The Blink Code – Teensy 3.2 Version

```

/*
  Blink
  Turns on an LED on for one second, then off for one second, repeatedly.

  This example code is in the public domain.
*/

// Pin 13 has an LED connected on most Arduino boards.
// Pin 11 has the LED on Teensy 2.0
// Pin 6 has the LED on Teensy++ 2.0
// Pin 13 has the LED on Teensy 3.0
// give it a name:
int led = 13;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(led, HIGH); // turn the LED on (HIGH is the voltage level)
  delay(1000);             // wait for a second
  digitalWrite(led, LOW);  // turn the LED off by making the voltage LOW
  delay(1000);             // wait for a second
}

```

Examine the Code

The commands may look intimidating at first, but in no time you'll understand what each line means. In fact, we can take a cursory look at the code right now:

- First, the colors of the code tells us a lot.
 - Any code in **gray** is known as a *comment statement*, or simply as a **comment**. Comments are ignored by the compiler and are used to make notations for the humans reading or writing the sketches.
 - Any code in **red** or **green** is an Arduino **command** that is built into the IDE.
 - Any code in **blue** is an Arduino **keyword** that has special meaning.
 - Code in **black** is either a **variable** created by the user, a **number** (we say a *numeric literal* in the programming world), or a **symbol**, such as a comma, semicolon, parentheses, or curly brace.
- In the **Teensy version** of the sketch, the first *real* line of the code – below all those gray comment statements at the top – reads

```
int led = 13;
```

This statement establishes a user-defined **variable** named “**led**” (as in L.E.D., LED, or light-emitting diode) and was assigned the value 13. This is because the 13th pin on the Teensy 3.2 and Arduino Due boards are connected to their onboard LED lights. Programming with variables, rather than numeric literals, makes it easy decipher code, as you will soon learn.

The **Arduino version** of the code does not define the `led` variable, but rather uses the numeric literal, 13, to represent the LED's pin. However, the code runs exactly the same as the Teensy version.

- Each sketch in Arduino *must* have two **functions** that are named **setup** and **loop**. (A function is simply a small chunk of code than performs some task.) Allow me to briefly explain what is happening in each of these required functions:
 - Within the **setup** function, the line of code that reads:

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

or

```
pinMode(led, OUTPUT);
```

Arduino Due code

Teensy 3.2 code

is used to set the **mode** of the led pin (*i.e.*, pin 13) to **output**, so the microcontroller can *output* voltages, which can turn on and off the onboard LED. (Conversely, if the mode were set to **INPUT**, that pin could be used to read the *input* voltages sent to the microcontroller by a button or sensor.)

- The **loop** function is where all the action of this sketch is located. By default, this function will *loop* over and over and over – forever and ever. If you take a look at the code within this function, you can probably get a sense of what it is trying to do. The first line reads:

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

or

```
digitalWrite(led, HIGH);
```

Arduino Due code

Teensy 3.2 code

which simply sets the LED pin (*i.e.*, pin 13) to a **high** voltage. This is computer-speak for sending the *maximum* voltage to the LED pin, causing the onboard LED to illuminate in this example. In the case of the Teensy 3.2 and Arduino Due boards, the maximum voltage that can be *output* to any pin is 3.3 volts (or 3.3V).

The next line of code reads:

```
delay(1000);
```

This line delays, or *pauses*, the execution of a sketch for 1000 milliseconds, which is equivalent to one second. Therefore, the first two lines of the **loop** function turn **on** the onboard LED for one second.

The next two lines turn **off** the LED:

```
digitalWrite(13, LOW);
delay(1000);
```

or

```
digitalWrite(led, LOW);
delay(1000);
```

Arduino Due code

Teensy 3.2 code

Setting the LED pin to **LOW** simply means setting the voltage of that pin to the *minimum* value of zero volts (or 0V). Because the LED pin is no longer “receiving” a voltage, the light from the LED is extinguished. Convince yourself that the second delay of 1000 milliseconds is necessary. Without it, can you predict what would happen? Remember that the **loop** function *loops* forever!

Alter the Code so the LED Blinks Faster

You’ll learn much more about each of the commands in the **Blink** sketch in the coming chapters. For now, have some fun and mess around with the blinking rate by changing the delay values. Alter the length of the pauses so the LED will flash on for a tenth of a second and off for a tenth of a second. Can you figure out how to do this on your own? It is helpful to note that a tenth of a second is equivalent to 100 milliseconds. Go ahead and change the code – you’re not going to hurt anything! If you get stuck, my solution can be found below.

One way to make the onboard LED blink more rapidly is with the altered the code inside the `loop` function as follows:

```
void loop() {
  digitalWrite(led, HIGH);
  delay(100);
  digitalWrite(led, LOW);
  delay(100);
}
```

By decreasing the length of the delays that the LED is on and off, the light will blink more rapidly. If you haven't done so already, alter your code and examine your development board. You may be surprised to see that your onboard LED is **not** blinking rapidly! The reason for this is profound: you have changed the *code*, but you have **not uploaded** the altered code on your microcontroller!

Verify the Code – A Good First Check

To get the code from the Arduino IDE window to your microcontroller sitting on the desk, you must first **compile** the code into *machine language*. Arduino calls this *verifying the code*, because the code needs to be checked for any errors.

Compiling – or verifying – the code is easy to do. Simply click on the **Verify button** at the top of the IDE, as shown in Figure xxx, or press **CTRL+R** on your keyboard.

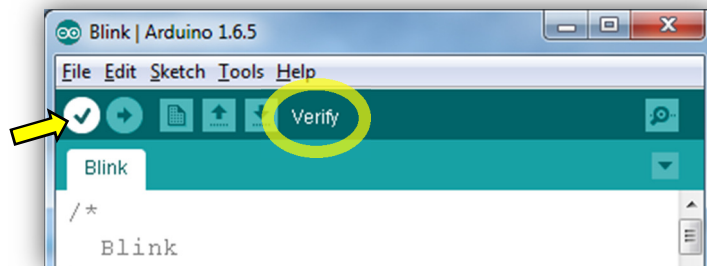


Figure xxx. Press the **Verify button** (or press **CTRL+R** on the keyboard) to *compile* the code on the Arduino IDE into *machine language*.

Compile your code now to verify that your sketch is error free, either by pressing the **Verify button** or pressing **CTRL+R** on your keyboard. If you correctly followed all of the above steps, you should see a "Compiling sketch..." message and a green or blue **progress bar** below the code window, as shown in Figure xxx below. The first time you compile any *new* sketch to your microcontroller it will require a number of seconds to verify the code. Anytime you alter your code, you should verify that your changes or additions are error free. **Verify your code often; you'll be happy that you do!**

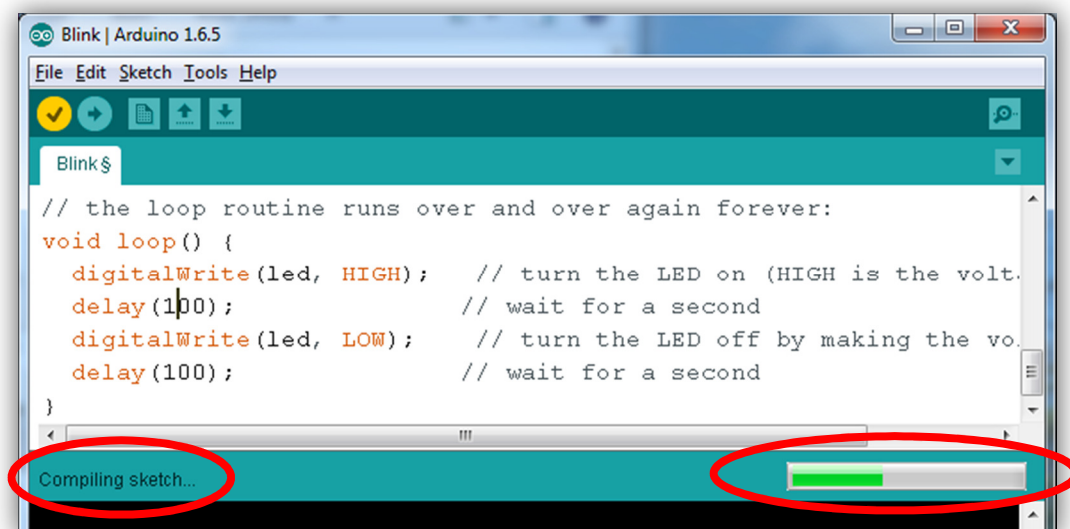


Figure xxx. The progress bar indicates the progress of the compile (verify) operation.

Debugging Any Errors

The Arduino software will not compile any code that contains errors, so if the sketch does contain errors the programmer must **debug the code**. A fault in one's code is highlighted by the IDE in a number of ways. For example, in the **Blink** sketch above, let's say that you mistakenly entered the line:

```
delay(i00);
```

That is, you typed **i00**, instead of **100**. Of course, this will confuse the compiler and it will tell you so. Compiling this code will yield three error messages: The offending line of code will be highlighted in red, and the **error message**, “*'i00' was not declared in this scope*” will appear in both the red **Message Area** and in the black **Text Console**, as shown in Figure xxx.

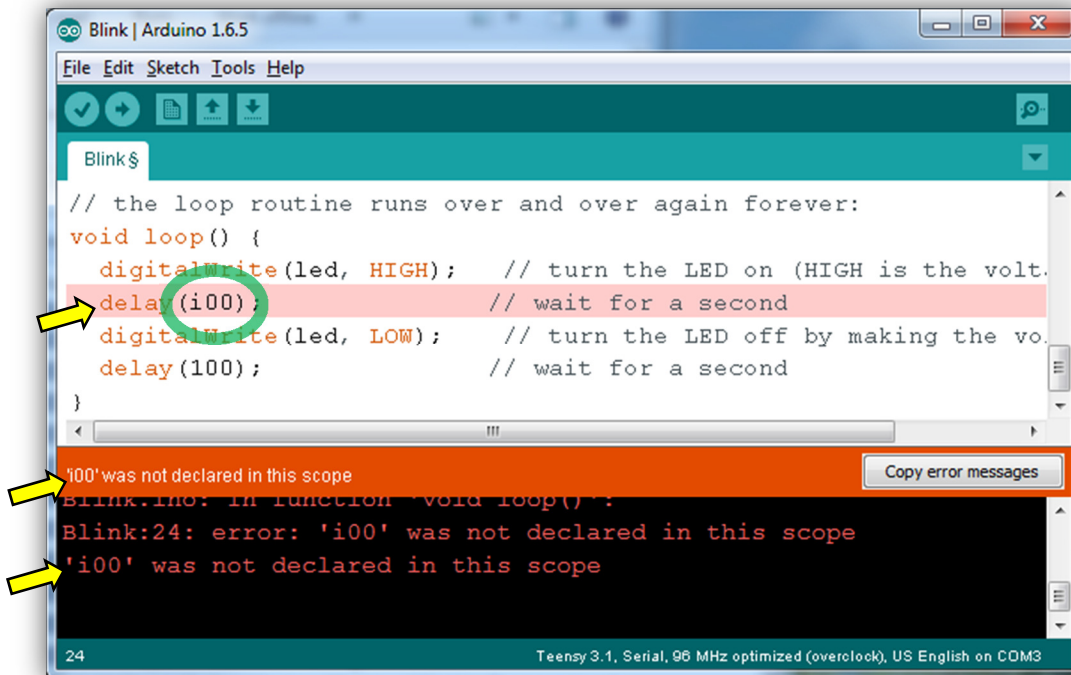


Figure xxx. Any errors in your code will be highlighted after the code is compiled.

To make these error messages disappear, simply fix your mistake (that is, change **i00** to **100**) and recompile your code. It's that easy.

In the coming chapters, don't be dismayed when your code contains errors! All programmers – novices and experts alike – typically spend as much time writing code as they do debugging code. It's a part of programming – get used to it!



The neat thing about this compile/verify step is **you do not need to a microcontroller to test the code**. That is, you can write and test (compile) your code without having to connect a development board to your computer! This is especially handy when you become a more experienced and proficient programmer. It may not be too helpful for the beginner, because without the development board, you are unable to see any *output* from the code. In other words, while you can check to see if there are any errors in the code, without a development board the sketch won't *do* anything. However, when a development board is plugged in, you will know the code will compile without errors.

Upload the Code to your Microcontroller

Warning! It is important that you do **not** unplug your development board during the next few steps. Read this section carefully, especially the part that explains when it is okay to unplug your microcontroller from your computer.



Now that you have verified and compiled your sketch, it is time to **upload** your code to the development board. That is, you are about to move the coded instructions from your Arduino IDE window to the microcontroller sitting on your desk. To make this happen, simply click on the **Upload button** at the top of the IDE, which is shown in Figure xxx. Or you can press the shortcut **CTRL+U** keys on your keyboard. The Arduino IDE will do all the heavy lifting!

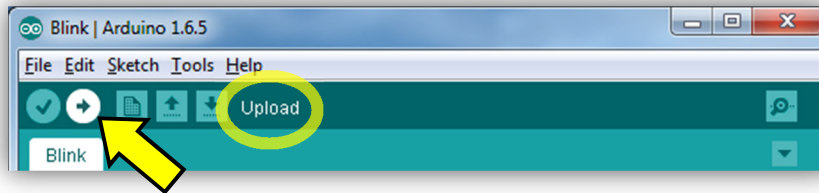


Figure xxx. To compile your code into machine language and upload it to your development board, press the **Upload button** as indicated.

As the code is being uploaded to your development board, it will be compiled yet again. You will see the same **progress bar** that you saw when you verified the code in the previous step. (See Figure xxx.) Uploading code to the development board may require a number of seconds to compile the code and send it over to your board. **Be patient here!** While your sketch is in the process of uploading to the microcontroller – that is, when the progress bar is showing – **never, ever unplug the device** from your computer. You must also **never press the Upload button** while the progress bar is showing! Interrupting the two-way communications during an upload can actually destroy your board and you will have to buy a new one. *Seriously, be patient!*

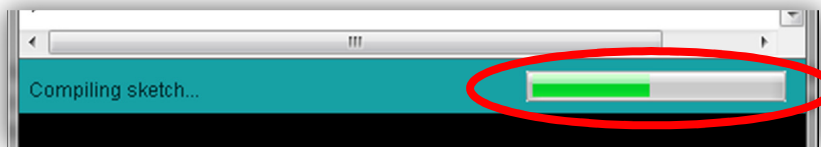


Figure xxx. It is **critical** that you **never unplug** your microcontroller or **press** the upload button again while the progress bar is showing! Doing so could destroy your microcontroller!

Examine Your Handiwork

Examine your development board now. If you see the onboard LED blinking rapidly every 0.2 seconds, you will know that the altered code for the new **Blink** sketch was properly uploaded to your development board. Cool, huh! (See Figure xxx for the location of the LEDs on your particular board.) If the LED is not blinking rapidly, check your code and ensure your delays are each 100ms long, and try uploading your code once again – but only if it is safe to do so, as explained above!

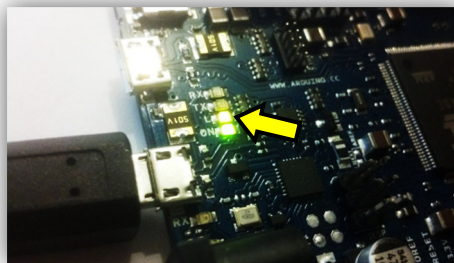
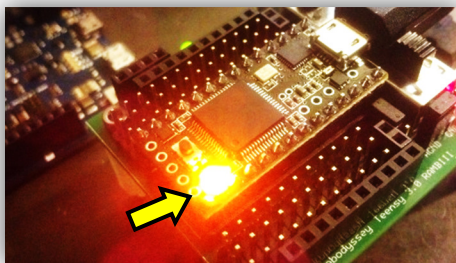


Figure xxx. If you see the flashing amber LED, you can relax, for you are set up and ready to go!

Feel free to play around and further alter the LED delay times. Can you make the LED blink even more rapidly? Can you turn it on for a short period of time and off for a longer period? Have some fun – you can't hurt anything!

Amazing Stuff!

Take a moment to think about what just happened here. When you pressed the **Upload button** many things were set in motion, but the bottom line is a computer program was used to turn on and off an LED – a *physical, real-world* device! Computer code – some abstract arrangement of words on a computer screen – was used to control an actual object. This is amazing!

Once the compiled sketch is loaded into the microcontroller’s memory, your *computer* is no longer needed. The code resides in your microcontroller’s memory, and it will run whenever it is powered up. Your computer is no longer part of the equation. Of course, if your board is probably getting power from your computer’s USB cable, the computer is still an important component! If you have a battery pack for your development board or a USB wall charger, test this by unplugging the USB cable from your computer and plug it into the wall charger or connect the battery pack. You should see the LED blinking even though it is not connected to the computer!

Upload Considerations

Always, always, always be patient until the code has finished uploading! After the sketch has been properly uploaded to the development board, the progress bar will disappear and the programming window will display a “Done uploading” message and information about the memory allocation on the microcontroller, as shown in Figure xxx below. If the volume to your computer’s speakers is turned up, you will also hear some electronic sounds, which indicate the code has been uploaded.⁷

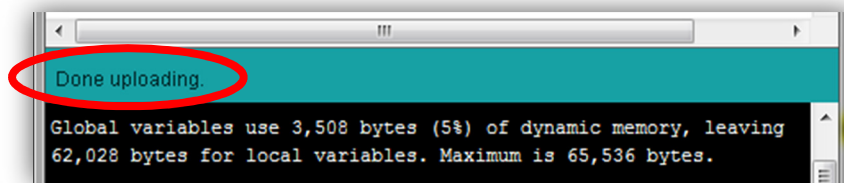


Figure xxx. Once the sketch has been properly uploaded, the progress bar will disappear and the programming window will display a “Done uploading” message. If the volume to your computer’s speakers is turned up, you will also hear some electronic sounds indicating the code has been uploaded.

Teensy 3.2 users should make one final check before proceeding. When the **Upload button** is pressed and the code is compiled, Teensy users will also see the Teensy Loader window pop up, shown in Figure xxx. The Teensy Loader is a program that takes code written for an Arduino board and translates it into code for the Teensy. You don’t need to *do* anything with this window, but you should wait for it to tell you that the code has been translated and has finished uploading to the Teensy microcontroller. You can tell when this has happened because the Teensyduino window goes from looking like the one in Figure xxx to the one in Figure xxx. Only when you see the window in Figure xxx do you know that the code is safely uploaded and only then may you proceed.

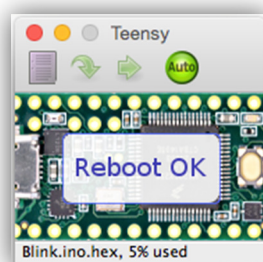


Figure xxx. Teensy users will see the Teensy Loader window pop up after uploading their code. This warns the programmer know to not unplug their board and not to re-upload the code at this time.

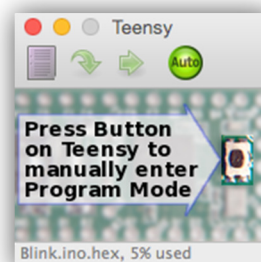


Figure xxx. When the code has been successfully downloaded to the Teensy, the Teensy Loader window will look like this. It is now safe to unplug your board or re-upload the code.

⁷ To help ensure my students do not destroy their development boards, I encourage them to unmute their computer speakers and listen for the tell-tale computer sound that is generated when the sketch has been being successfully loaded onto the board.

When I program my Teensy, I make sure that my code window is resized so that I always can see the Teensy Loader window. Even though you don't need to **do** anything with this window, keep it visible so you can easily **monitor** the status of your upload operations. Don't bother closing the Teensy Loader window because it will just re-open any time you upload new code.

The window also serves as a reminder that you may need to press the physical reset button on top of your Teensy for the code to upload, as shown in Figure xxx. You should only have to do this once *for each new program*. Any time you upload code after that, the Teensy Loader will run and your code should automatically upload to your board.⁸

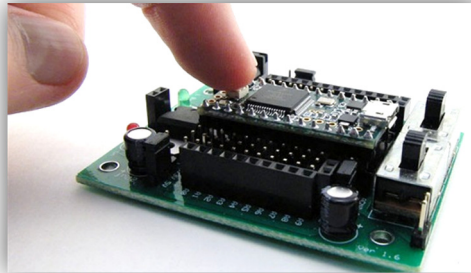


Figure xxx. In order to upload a new sketch to the Teensy development board, the user may need to press the reset button on the Teensy itself.

Putting Away Your Toys – What to Do When You Are Finished for the Day

When you are finished playing around with the onboard LED, *and you are positive that no code is being uploaded to your development board*, you can safely unplug your programming cable from the computer without worrying about damaging your microcontroller. Even though a program is running on your development board and the LED is blinking, it is safe to unplug the board. After the sketch has been uploaded, the only thing the computer is doing is providing power to the development board.



It is important to reiterate here that if code is in the process of being uploaded, you must not unplug the development board from the computer!



If you decide to unplug the programming cable from the development board, take care that you pull the cable out horizontally, never at an angle! Because the tiny micro-USB socket is not meant to withstand a lot of force, it is easy to pry the socket off of the board! Be kind to your board, and it will be kind to you!



Usually, when you are finished programming for the day, you would save your sketch just as you would save a Word document. However, because the **Blink** sketch was a *built-in example program* do not save any changes made to the sketch. Simply close the Arduino IDE without saving the sketch.

1.7. Details about Microcontrollers for *Serious* Programmers

In this *optional* section, I explain in more detail what is actually happening when the **Upload button** is pressed. I will also talk in more detail about the differences of a variety of microcontrollers and development boards, and why the ARM processors are the focus of this book. Finally, I will discuss the advantages and disadvantages to using the Teensy 3.2 and Arduino systems. If you would like to skip this section and jump straight to the Challenge problems at the end of this chapter, feel free to do so. However, if you have a desire to become a more knowledgeable and serious microcontroller programmer, you may want to take a few minutes (now or later) to read this section.

A Look behind the Scenes

The focus and intent of the book are to teach you about physical computing using the **Arduino programming language**. The Arduino language is based on the common and powerful C language and embedded controllers can be programmed using the Arduino IDE or *integrated development environment*. Regardless of what development board you have elected to use, all

⁸ If there is ever an interruption of the serial communications, you will be instructed to press the reset button.

Teensy and Arduino boards can be programmed with the Arduino IDE. You will use this programming software to create all the sketches in this book.

While it is not *necessary* that you understand the details of how sketches go from the IDE to the development board, it is *useful* to understand the general process by which your code moves from the computer to the microcontroller. It's actually not that complicated; here is the general idea:

1. A sketch is created by writing easy-to-read code within the Arduino IDE. The sketches are saved to a drive on your computer with the **.ino** file extension.
2. When the sketch is complete, pressing the **Upload button** takes your program and **compiles** (translates) it into binary (hex) machine language. Machine language is the only language that machines understand, but it cannot be read by ordinary humans. The binary version of your code is sent (uploaded) from your computer to your development board via the USB programming cable.
3. The microcontroller on the development board then uses a **bootloader** to store the new binary file in the memory space allocated for program storage on the chip.
4. At this point, your sketch resides on the microcontroller and can run independently of your computer. In other words, you can remove the development board from the computer and your program will run, provided the board is powered with batteries or some other voltage source.
5. Now, any time the boards are powered up, the code will run even when it is disconnected from your personal computer! Your microcontroller has the potential to become a very powerful tool, capable of operating on your dining room table, at the bottom of the ocean, or even in the far reaches of space! Are you ready to begin?

What is under the Hood of your Development Board?

Anyone *serious* about microcontroller programming needs to know a bit more of what's under the hood before proceeding. At the center of every microcontroller is the CPU or *processor*. There are a number of processors on the market, but the PIC, AVR, and ARM microcontrollers are at the top of the heap. Programming these tiny chips *from scratch* takes a lot of electronics know-how and therefore isn't suited for an introductory course.

To make it easier to use and program these processors, some companies have designed their own circuit boards to make it easier to communicate with the tiny chips. As you have learned, these boards are known as **development environments** or **development boards**. Some common development boards include the BX-24, Basic Stamp, Teensy 2.0, Teensy 3.2, Arduino Uno, and Arduino Due. Figure xxx shows some of these boards and which processors they use.

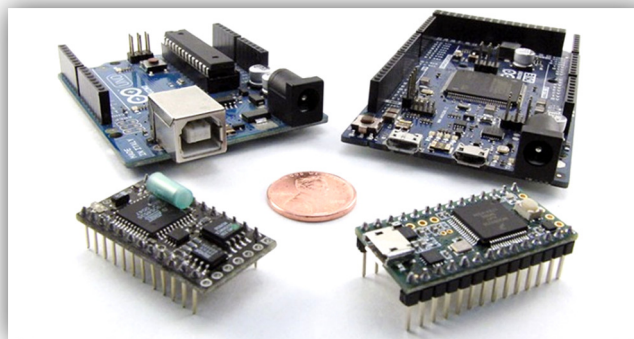


Figure xxx. Counterclockwise from the bottom left: NetMedia's BX-24, Arduino Uno, Arduino Due, and PJRC's Teensy 3.2. The BX-24 and Uno use the AVR processor. The Due and Teensy 3.2 use the more powerful ARM processor.

For a long while now, the AVR processor has been king. The Arduino Uno and Teensy 2.0 development boards both use AVR processors and have done a remarkable job galvanizing the **maker movement** and popularizing the field of home robotics and embedded controllers. But the AVR's days are numbered as the more advanced and powerful **ARM processors** grow in popularity. **This book is written around the Teensy 3.2 and Arduino Due development boards, which use the new and more powerful ARM processors.** While a great deal of what is covered in this text will apply directly to AVR-driven development boards such as the Teensy 2.0 and Arduino Uno, all of my code examples, diagrams, and images are with the Teensy 3.2 and Arduino Due in mind, with my personal preference for the Teensy 3.2.



For those of you with an AVR-based chip, most of what is covered in this text will apply to your board. I will do my best to point out when the lessons don't apply to you. When you see the red ARM button shown at the left, you'll know the material covered in that section may not work for your board. Besides having faster and more powerful processors, the biggest difference between ARM and AVR is their voltages. The older AVR chips operate on 5.0V, while the new, more efficient ARM chips require only 3.3V. It is important to realize that some of the older 5-volt sensors and actuators may *damage* your 3.3-volt ARM board. This will be discussed in more detail in subsequent chapters.



Some of these development boards, like the Arduino Due shown in Figure xxx, include all the electronics required to handle communications between the onboard microcontroller and your computer, as well as a variety of I/O ports. Others, like the Teensy 3.2 shown in Figure xxx work best with a third party **motherboard**, such as the **PRT3** from Patton Robotics to handle the communications between your computer and the Teensy and allow it to interact with sensors, lights, and motors. Figure xxx shows the Teensy 3.2 development board with PRT3 Motherboard.

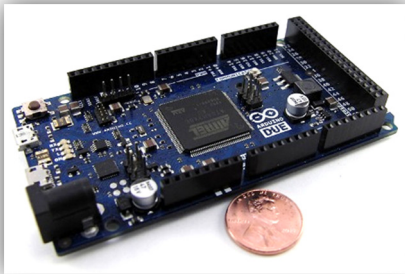


Figure xxx. The Arduino Due development board.

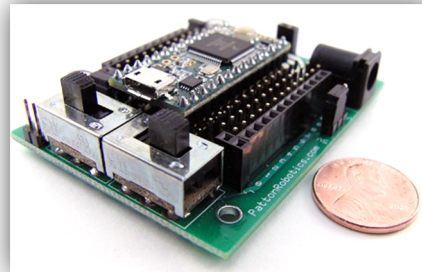


Figure xxx. The Teensy 3.2 development board mated with the PRT3 Motherboard from Patton Robotics. The PRT3 makes it easy to connect components and battery packs to the microcontroller.

The Pros and Cons Teensy 3.2 and Arduino Due Development Boards

This book was written for the Arduino Due and Teensy 3.2 development boards, shown in Figures xxx and xxx, respectively. While you can do everything in this textbook with either controller, for some things the Teensy is better and for others the Due is better. Below I try to give a fair and candid assessment of both boards.

Pros and Cons of the Arduino Due

- The Arduino Due is a development board that includes all the electronics required to handle communications between the onboard microcontroller and your computer, as well as a variety of I/O (input/output) ports. This board has the *advantage* of an all-inclusive package. The microcontroller, development board, and motherboard come bundled together; there is no need to buy an additional motherboard. Unfortunately, this means the Due is over twice the cost of the Teensy 3.2. Another *disadvantage* of the all-in-one-system is that if the microcontroller is destroyed (which can happen with beginners), you have to throw the entire thing away rather than simply replacing the controller.
- Another *advantage* of the Due is that right out of the box, you can easily connect *individual* wires to the board with no need of additional hardware. While it is possible use a system of *individual* connectors and jumper wires to attach components to the base Due development board, a *disadvantage* to using these wires is that they are easily pulled out and do not make for a very robust system.

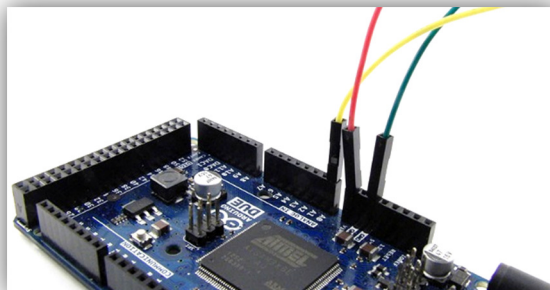


Figure xxx. Single connector wires are easily unplugged and create a mess of all but the most rudimentary of projects.

- One of the biggest *advantages* of the Due (and all Arduino systems, for that matter) is that it requires only the Arduino IDE software to run properly. (Teensy users need an additional piece of free software, called the Teensyduino, which, as you know, allows the Arduino IDE to communicate with Teensy development boards.)
- Another *advantage* of the Arduino systems is how easy it is to connect one or more add-on boards, or **shields** as Arduino calls them, to the base Due board, as shown in Figure xxx. Shields perform one or more functions and can be purchased from Arduino or from third-party vendors. For example, you can control a motor, read signals from a GPS satellite, and output data to an LCD screen by plugging in a motor shield, GPS shield, and LCD shield. However, this “advantage” is more of a *disadvantage* because the financial cost of adding shields to your project can be high. While the number of available shields is impressively large, in order to do anything but the most rudimentary of tasks requires one or more shields, and the cost and physical size of the project can grow rapidly. For example, the addition of the motor, GPS, and LCD shields will cost you an additional \$85-\$100, which is too much to spend for many hobbyists and educators, simply to attach a few components.

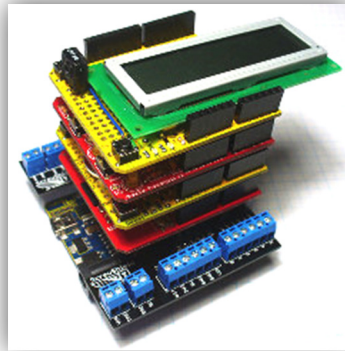


Photo credit: xxx John Boxall and www.freetronics.com

Figure xxx. Shields seem like a good idea until you consider how much they cost, how much space they take up, and how much board resources they consume.

- Another *disadvantage* of the Due is that there is no off/on power switch to the board. To turn off the board, or to turn off power to connected servomotors, for example, you are forced to unplug the battery pack or programming cable. This gets annoying after a while.
- Finally, the Due has the *advantage* of coming pre-assembled with nice and tiny **surface mounted components**. You pull it out of the box and you are ready to go! The *disadvantage* of this is those do-it-yourselfers do not have the opportunity to put the board together from a kit. As an educator, I find this especially upsetting because the soldering and building of the motherboards is a task students generally enjoy and take great pride in doing.

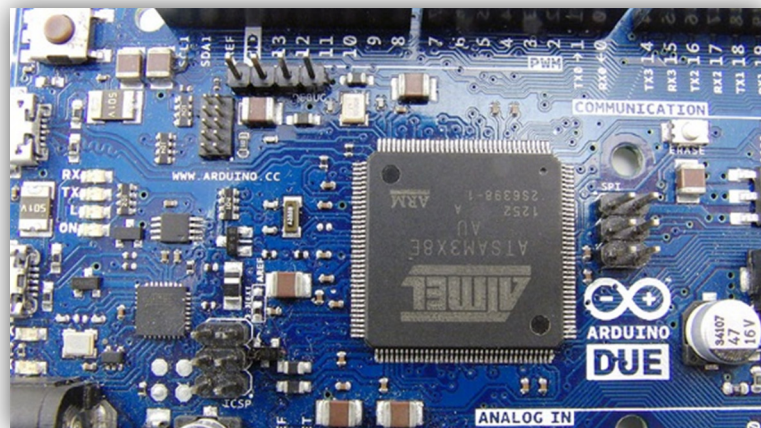


Figure xxx. Professionally made motherboards with surface mounted components are nice, but they prevent the DIY-er from building the motherboard from scratch.

Pros and Cons of the Teensy 3.2/PRT3 Combo

- The Teensy 3.2 is a powerful plug-and-play development board with the same powerful ARM processor as the Arduino Due. One main *advantage* of this board is the fact that it can be used as is or it can be inserted into a motherboard. The *disadvantage* of not being part of an all-inclusive controller is outweighed, in my opinion, with the flexibility of being able to use the Teensy as a stand-alone controller (you can build the Teensy directly into your projects) or in conjunction with a third-part motherboard, such as the **PRT3** from Patton Robotics. This has the added *advantage* of being able to cheaply and easily replace the microcontroller in the event it is damaged or destroyed, which, as I explained above, is not possible with the Arduino Due.
- The *biggest advantage* that the Teensy 3.2 has over any of the Arduino boards is being able to use the Patton Robotics **PRT3 Motherboard**. This versatile and inexpensive motherboard was designed specifically for the Teensy 3.2 and is much more user friendly than any of the Arduino boards. The connectors of the PRT3 allow for **3-wire connectivity**, meaning common servos and sensors can easily be plugged directly into the motherboard without needing to purchase expensive and unwieldy shields that are required by Arduino systems. This is a **huge advantage** that Teensy has over Arduino and cannot be overstated. I can find no *disadvantage* to using the PRT3 Motherboard over the Due board, other than the fact of having to purchase another piece of hardware. The cost of purchasing the Teensy/PRT3 combo is the same or even less expensive as buying the Due.

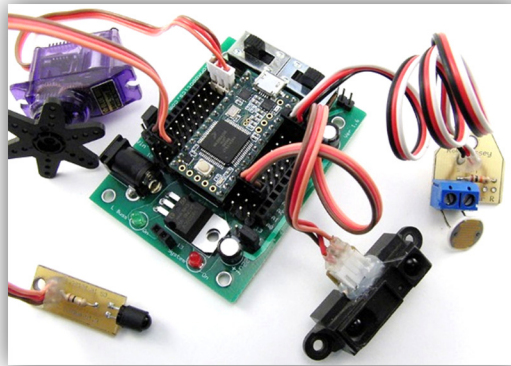


Figure xxx. The PRT3 Motherboard from Patton Robotics makes it super-easy to connect a multitude of sensors and actuators with 3-wire cables, while providing just as many single-wire female ports as the Arduino board.

- Another *advantage* of the PRT3 Motherboard is that it comes with two power switches. When disconnected from the computer and running on battery power, one of the power switches can be used to control power to one half of the board, allowing you for example, to turn on and off power to servomotors. The other switch can be used to turn on or off power to the microcontroller – the ultimate kill/reset switch. When the board is connected to the computer via the USB programming cable, the power switch that controls half of the board (for example, servomotors) can be utilized. This is handy when you want the robot brain and sensors to continue to function, but don't want the robot's wheels to spin.

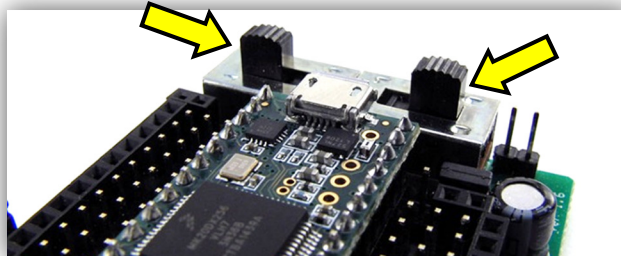


Figure xxx. The PRT3 Motherboard allows the user to turn on and off power to both side of the motherboard.

- One *disadvantage* of using the Teensy 3.2 (and all Teensy boards, for that matter) is that it requires an additional piece of free software, called the Teensyduino, to be installed. The Teensyduino allows the Arduino IDE to communicate with Teensy development boards. This is only an issue at the beginning when software is being installed for the first time. After that, the software runs in the background, practically unnoticed. There is no *advantage* to using a system that requires the Teensyduino.

- Finally, the Teensy/PRT3 Motherboard combo has the *advantage* of either product coming pre-assembled or as a do-it-yourself kit. As an educator, I find that having students solder their own boards is an educational and empowering lesson. Students acquire a valuable skill, take pride in their work, and enjoy the experience.

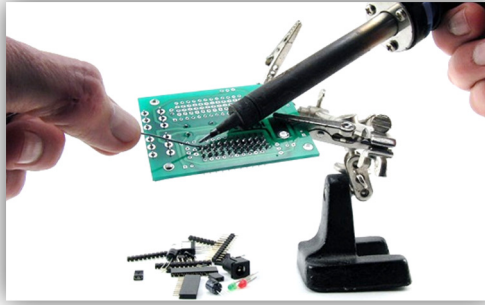


Figure xxx. Students find it enjoyable and rewarding to build their motherboards from scratch.

My Conclusions

Despite the popularity of the Arduino development boards, I much prefer the combination of the Teensy 3.2⁹ with the versatile Patton Robotics PRT3 Motherboard, shown in Figure xxx.¹⁰ In the classroom you can't beat the Teensy/PRT3 combination for ease of use and durability.

Brian Patton, the President and CEO of Patton Robotics and creator of the PRT3 Motherboard, has over 20 years of experience designing, building, and manufacturing robots for the sole purpose of educating children and adults in the exciting world of robotics and physical computing. The products his company sells and manufactures are durable enough for rowdy, untrained middle school students, yet sophisticated enough for polished students at Ivy League universities. I am so excited about the educational systems of Patton Robotics that I have written two books around their educational robot platforms.

Fortunately for users of either Arduino or Teensy systems, Patton Robotics has designed a well-made and robust robot platform for hobbyists, educators, students, and professionals, alike. Known as the **OneBot**, this chassis with over 50 tapped and 12 clearance holes gives the user a solid platform to connect motherboards (both the PRT3 and Arduino mount on the chassis), secure motors (servo and steppers), mount sensors, add additional decks, and secure battery packs. This versatile robot platform is showcased in Volume Two of this text.

While I believe the Teensy 3.2/PRT3 Motherboard combination is the best option for classroom instruction, this book will also show how to use the Arduino Due board as the brain and central nervous system of your physical computing devices.

1.8. Congratulations and a Challenge

Congratulations! Your equipment is all setup, and you are now ready to dive into the wonderful world of physical computing and robotics with your embedded microcontroller! The best way to reinforce what you just learned is to put it into practice. At the end of every chapter you'll find **Challenge Problems** for that purpose. Students using this book in a classroom setting may find that their teacher has already assigned some of these problems for them to do. For the hobbyist I also recommend trying to work some of the problems. Even if you *think* you understand the material, you won't *know* you do until you practice.

If you are a teacher and would like to follow the syllabus I have created for my *Computer Programming & Robotics* students at **George School**, see *Appendix xxx – George School Robotics Classwork Timeline*. Here you can see all the Challenge Problems I have assigned to the students who self-select themselves as either *Intermediate* or *Intensive* level. Additionally, the syllabus points my students to a variety of additional reading assignments, descriptions or required and optional projects, and interesting readings and Internet links. Feel free to use my syllabus for your own classroom instruction, and contact me if you have ideas and suggestions for ways I can improve it!

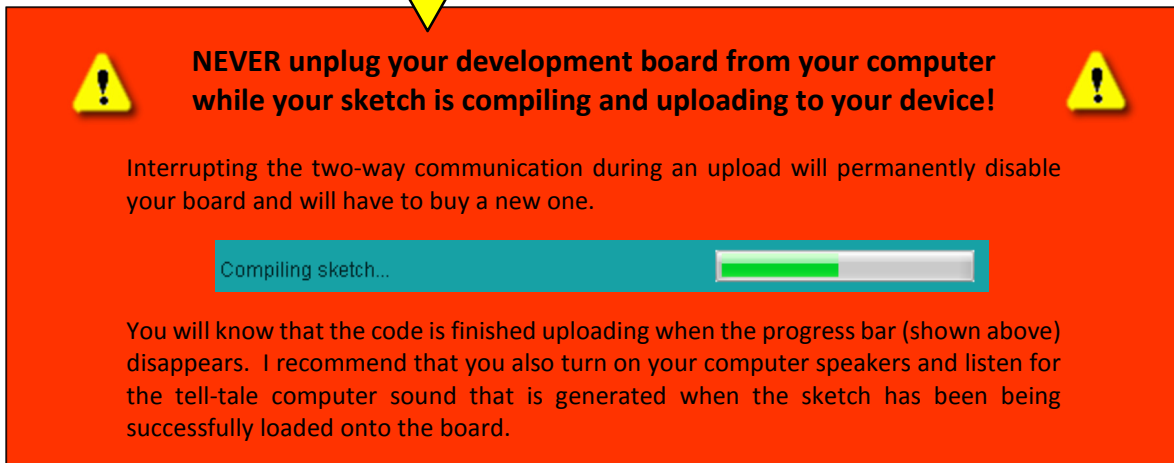
⁹ Teensy 3.2 development board is the creation of Paul J Stoffregen of PJRC.com, LLC, and uses the powerful 32-bit MK20DX256VLH7 microcontroller as its brain. For more information, see www.pjrc.com/

¹⁰ The PRT3 Motherboard is the brainchild of Brian Patton of Patton Robotics, LLC of New Hope, Pennsylvania. Visit <http://pattonrobotics.com> for more information.

Challenge Problems

Your instructor may want you to turn in written work for these **Challenge Problems**, or they may prefer to check your solutions in person. The first nine problems are probably best turned in as written work.

- 1-1. What does the term “IDE” stand for and, more importantly, what is it used for? What is the name of the IDE that you will be using?
- 1-2. What is the name of the computer *language* that will you use to program your microcontroller? What is the famous language on which this is based?
- 1-3. What are three advantages that ARM microcontrollers (processors) have over the AVR microcontrollers? Which processor is embedded on your development board?
- 1-4. What is the name of the development board are you using? If you are using a motherboard, what is its name and who makes it?
- 1-5. List three advantages and three disadvantages of your development board over the competitor.
- 1-6. True or False. You can only use an Arduino development board when programming with the Arduino IDE.
- 1-7. True or False. The Arduino IDE runs equally well on a PC, Mac, or Linux machine.
- 1-8. What role does the personal computer play in the world of physical computing?
- 1-9. When is it permissible to disconnect your development board from your computer? More importantly, when should you **never** unplug your development board from your computer? This question is so important, I’ve given you the answer in the warning message box below!



NEVER unplug your development board from your computer while your sketch is compiling and uploading to your device!

Interrupting the two-way communication during an upload will permanently disable your board and will have to buy a new one.



You will know that the code is finished uploading when the progress bar (shown above) disappears. I recommend that you also turn on your computer speakers and listen for the tell-tale computer sound that is generated when the sketch has been being successfully loaded onto the board.

- 1-10. In this chapter you learned how to load the **Blink** example sketch and change the blinking frequency of the onboard LED. Alter the code once again so the LED is on for 50ms and off for 700ms.
- 1-11. In this chapter you learned how to load the **Blink** example sketch and change the blinking frequency of the onboard LED. Alter the code once again so the LED is on for 1.5 seconds and off for 0.5 seconds.
- 1-12. In this chapter you learned how to load the **Blink** example sketch and change the blinking frequency of the onboard LED. Watch this YouTube video <https://youtu.be/t1E7XuT3HVE> of my blinking LED and alter the code of the **Blink** example sketch to reproduce my blinking frequency on your own development board. I’ll give you a hint: the on and off delays are both multiples of 10ms.

1-13. The following Challenge Problem is meant to reinforce what you just learned and introduce you to what is to come in the next chapter.

- A. To begin, whether you are using an Arduino or a Teensy development board, select the proper serial port by clicking **Tools >> Serial Port** from the menu bar, and choose the appropriate serial communications (COM) port, as shown in the figures below.
- On a Windows machine, the serial (COM) port is usually auto-detected for you, but check to make sure. Do **not** select COM1, which is reserved for other functions. It will **not** work with your microcontroller. If the only serial port showing is COM1, something is wrong. See “Troubleshooting the Serial Connection” below.

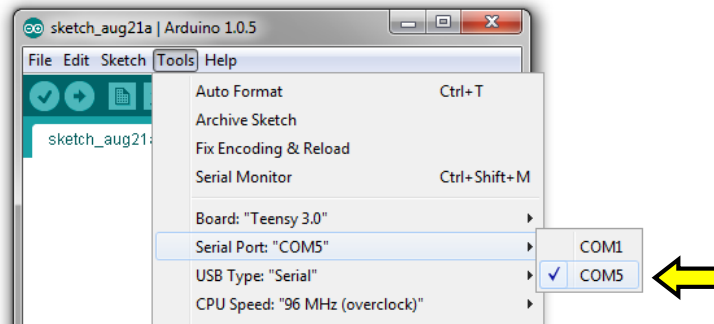


Figure xxx. Setting the Serial Port on a Windows machine.

- On a Mac, it may take some trial and error to locate the right port. Look for something like **dev/cu.usbserial...** or **/dev/cu...WirelessiAP** in the name as shown in Figures xxx and xxx. If you are unable to find the correct serial port, see “Troubleshooting the Serial Connection” below.

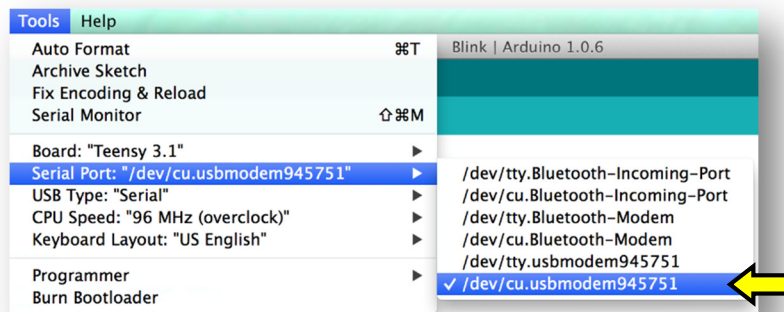


Figure xxx. A typical port setting the Serial Port on a Mac machine.
Photo credit: Tianyi “Martin” Ma.

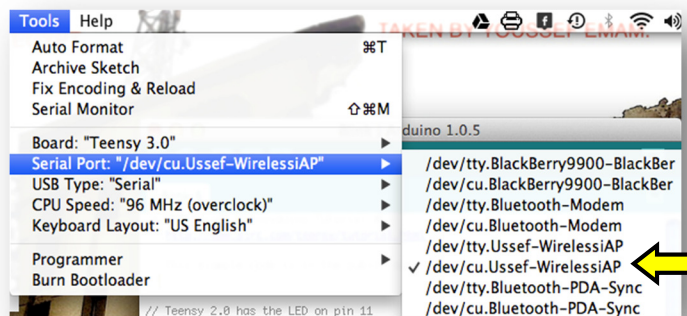


Figure xxx. Another typical Serial Port setting on a Mac.
Photo credit: Youssef Emam.

- B. Next, open the example sketch named **ASCIITable** from the example library file found by following this path: **File > Examples >> 04.Communication >> ASCIITable**.

- C. Now **upload** the **ASCIITable** code to your board in the usual way. You will find that once the program is running it will seem as if nothing is happening. To see the sketch's output, you must first press the **Serial Monitor button** shown in Figure xxx below.

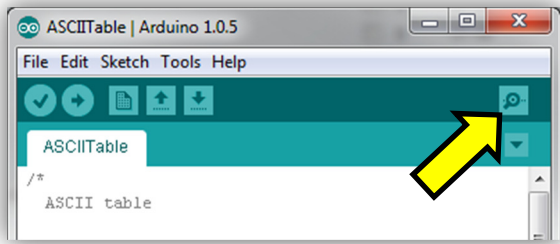


Figure xxx. To view the program's output, press the **Serial Monitor button** after uploading the sketch.

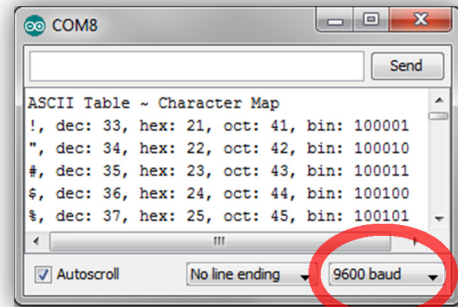


Figure xxx. The Serial Monitor Window showing proper output from the ASCIITable sketch. Arduino users may need to change the baud rate to 9600 as shown.

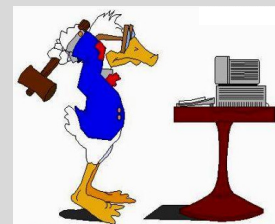
- D. The Serial Monitor window should pop up. If you have a **Teensy board**, your Serial Monitor should be filled with ASCII characters and the corresponding decimal, hexadecimal, octal, and binary codes, shown in Figure xxx.¹¹ If you have an **Arduino board**, you may see a bunch of random characters scroll across the screen. To fix that, simply change the **baud rate** (located at the lower right corner of the Serial Monitor window) to 9600 as shown in Figure xxx.

Don't worry about what the symbols and numbers mean right now; it will make more sense soon enough. This question was meant to test your ability to open and upload a new sketch to your development board. Once you see ASCII Table in the Serial Monitor, you are finished with the problem.

Troubleshooting the Serial Connection.

If the serial port for your device is not present, don't fret, for this happens *a lot!* Try the following:

1. Close the Arduino software and try again.
2. Unplug the USB programming the cable and plugging it in to the same port again.
3. Close the Arduino software. Unplug the USB programming cable and plug it into another USB port. Make sure you give the computer time to recognize the board! Restart Arduino.



¹¹ ASCII stands for American Standard Code for Information Interchange, and is pronounced, "ask-ee".