

# PH 250

Intermediate Physics Laboratory for Physics and  
Physical Science Teaching Majors

R Todd Lines  
David Oliphant

January 3, 2026



# Contents

<b>Introduction</b>	<b>vii</b>
<b>I Computer Interfacing</b>	<b>1</b>
<b>1 Introduction to the Arduino and Computer Control</b>	<b>3</b>
1.1 First Computer Control: LED blink (Step-by-step)	4
1.1.1 Prototyping boards	4
1.1.2 LED light blink Sketch	9
1.1.3 Arduino LED light blink sketch (program)	11
1.2 Second Computer Control: Two LED blink	15
1.3 Third Computer Control: Two LED blink using math	16
1.4 Fourth Computer Control: Two LED blink in the Fibonacci sequence	20
<b>2 Introduction to Electrical Measuring Devices</b>	<b>25</b>
2.1 What we measure: Voltage (and Current)	25
2.2 Stand-Alone Experimental Hardware	27
2.2.1 Power Supply	28
2.2.2 Signal Generator	29
2.2.3 Voltmeter	30
2.2.4 Oscilloscope	31
2.3 Current	35
2.3.1 Measuring Current	36
2.4 Building a New Instrument	38
2.4.1 Start with the Physics:	38
2.4.2 Knowing the Physics, Design the new instrument	43
2.4.3 Testing the new instrument	45
2.5 Calculating uncertainty, a review	46
2.5.1 How do we find the slope?	53
2.6 Practice Measurements	57
2.6.1 Use a Multimeter	57
2.6.2 Use an Oscilloscope	57
2.6.3 Build a new instrument from an old instrument	58

<b>3</b>	<b>First DAQ Measurements: Voltage</b>	<b>59</b>
3.1	Building a voltmeter . . . . .	61
3.1.1	Wiring the simple voltmeter . . . . .	65
3.1.2	Seeing the data . . . . .	66
3.2	Extending our voltmeter with a voltage divider . . . . .	67
3.3	Practice Problems . . . . .	74
3.4	Lab Assignment . . . . .	75
<b>4</b>	<b>Getting data to the Computer</b>	<b>77</b>
4.1	How to get Python – Skip if you have python . . . . .	77
4.1.1	Getting Anaconda Python . . . . .	77
4.2	Getting the PySerial library for Anaconda – Don’t skip this . . .	80
4.3	Getting data from the Arduino . . . . .	82
4.4	Getting Pyserial if you have a Mac – skip if you don’t have a Mac	88
4.4.1	Anaconda Mac Users 4.1.2 . . . . .	88
4.4.2	Canopy Mac Users 4.2.1 – skip if you don’t have Canopy	88
4.4.3	Manually install pyserial through the terminal on a Mac. Codingin emacs/VI . . . . .	88
4.4.4	Mac Pathway and Port Notation – Skip if you don’t have a Mac . . . . .	89
4.4.5	Mac version of the python code . . . . .	89
4.5	Lab Assignment – Really don’t skip . . . . .	91
<b>II</b>	<b>Testing Models</b>	<b>93</b>
<b>5</b>	<b>Validation of Ohm’s Law</b>	<b>95</b>
5.1	Ohm’s Law Revisited . . . . .	95
5.1.1	Measuring current with our Arduino . . . . .	96
5.1.2	Actually making an Arduino measure current . . . . .	98
5.1.3	Finding Uncertainty in a calculated value . . . . .	103
5.1.4	Using statistics to calculate uncertainty . . . . .	108
5.1.5	Philosophical warning . . . . .	111
5.2	Proposals . . . . .	111
5.2.1	Statement of the experimental problem . . . . .	111
5.2.2	Procedures and anticipated difficulties . . . . .	112
5.2.3	Proposed analysis and expected results . . . . .	112
5.2.4	Preliminary List of equipment needed . . . . .	113
5.2.5	Designing the Experiment . . . . .	113
5.2.6	Using Uncertainty to refine experimental design. . . . .	114
5.3	Lab Assignment . . . . .	117
<b>6</b>	<b>Resistors and Capacitors</b>	<b>119</b>
6.1	The Model to Test . . . . .	119
6.2	The Instrument . . . . .	121
6.2.1	Python Advanced Curve Fitting. . . . .	127

6.3	Lab Assignment . . . . .	132
<b>7</b>	<b>DataLogging</b>	<b>135</b>
7.1	Arduino Shields . . . . .	136
7.2	Data Logger Shield . . . . .	137
7.3	Set up . . . . .	138
7.4	Adding in your sensor . . . . .	143
7.5	Digital Data . . . . .	145
7.6	Difficulties with Larger Data Sets and Sensor Libraries . . . . .	145
7.7	Lab Assignment . . . . .	146
<b>8</b>	<b>Inductance and Series RLC Circuits Part 1</b>	<b>147</b>
8.1	The Model: Self Inductance . . . . .	147
8.1.1	Inductance of a solenoid . . . . .	149
8.2	RLC Series circuits . . . . .	149
8.3	Lab Assignment . . . . .	153
<b>III</b>	<b>Student Designed Experiments</b>	<b>155</b>
<b>9</b>	<b>Student Designed Experiments</b>	<b>157</b>
9.1	Proposal . . . . .	157
9.2	Performing the experiment . . . . .	158
9.3	Written report . . . . .	158
9.4	Oral report . . . . .	158
9.5	Lab Notebook . . . . .	160
9.5.1	Designing the Experiment . . . . .	160
9.5.2	Performing the Experiment . . . . .	161
<b>IV</b>	<b>Testing Models II</b>	<b>163</b>
<b>10</b>	<b>Inductance and Series RLC circuits Part 2</b>	<b>165</b>
10.1	The Instrument . . . . .	165
10.2	Sampling Theory, a complication . . . . .	169
10.3	Lab Assignment . . . . .	172



# Preface

Experimental physics is the art of testing our physical models to see if they really work. Thousands of measuring devices have been designed and built to make the detailed measurements needed to perform this testing. We would like to interface these measuring devices to computers so data collection is all digital. Then the data can be analyzed, and displayed on our computers. To do this we need to understand computer interfacing.

Computer interfacing is a bit of engineering. We need to either design and build, or purchase instruments, and then get data from those instruments into a computer. But even though it is an engineering task, it is a necessary skill for experimental physicists. It is this skill that we wish to take on in this laboratory class. Of course we can't learn all that there is to know on computer interfacing in one semester. But we can make a good start.

We will use the open source Arduino microcontroller board as our computer interface and we will use Python and the Arduino C++ languages to write controlling software.

Students should leave this class with confidence that they can build a simple computer interface that will read in sensor data and save it on a computer.

Students in this class will also spend time investigating physical models from introductory electricity and magnetism theory.

Instrumentation is sometimes difficult, sometimes frustrating, but also a lot of fun. I hope you will find these lab experiences both informative and entertaining.

I would like to acknowledge Tyler Miller who has spent countless hours testing the codes and writing the Mac versions and helping to improve the text.





# Introduction

As a PH250 student, you are probably taking PH220 concurrently with PH250. This lab course is designed to teach electronics and computer skills while your PH220 professor teaches electromagnetic field theory. Once you have a little bit of electric field theory under your belt from PH220, then our experiments designed to test out models of electric charge and electromagnetic fields begin in earnest. While we are waiting we will spend some time learning about how to control experiments with a computer, and how to import data from an experiment to a computer.

You should read the material for each lab before the lab begins. There will sometimes be practice problems to do to make sure you will be effective in lab. By preparing before lab you will have the full 2 hours and 45 minutes to make sure you can finish the lab work. Some labs may go fast, but most take the entire lab period. I also suggest you practice your computer and electronics skills a little. Build some blinking lights for your apartment, or measure how loud your roommates are, or something. The Arduino can be the data collection and control part of thousands fun projects.

This class is sometimes frustrating. But it is also a lot of fun. You will be introduced to computer instrumentation and will be able to perform an experiment that you and your lab group design. The student designed experiments are only limited by your imagination and our ability to find equipment. If you have concerns during the semester, don't hesitate to find your instructor or TA and ask.



Part I

# Computer Interfacing



# Chapter 1

## Introduction to the Arduino and Computer Control

In PH250 we have a goal of getting our computers to talk to our physics experiments (see the course syllabus). This comes in two forms. One is to have the computer control the experiment. For example, we could have the computer turn on our apparatus, or turn it off. The second form is having the experimental device provide data to the computer that we can later analyze. In both cases, we need a piece of electronics that goes between the experimental measuring devices and the computer. These electronic pieces are often called Data Acquisition (DAQ) boards. There are many different forms of DAQ's. They can cost anywhere from a few dollars to many thousands of dollars.

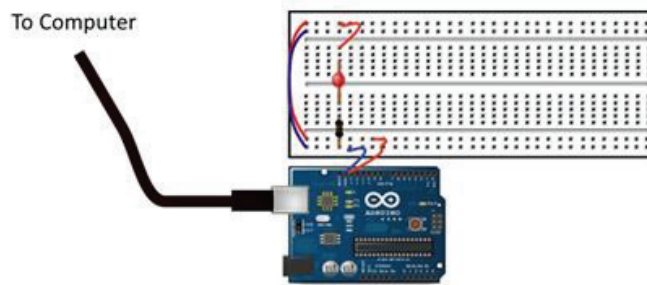
We will use a DAQ developed for learning. It is low cost because it's developers placed the plans for it in the open-source world so that no one would get royalties (including themselves) for the design. Thus board manufacturer's pay less, so we pay less. It is low cost, but not low quality. They named their DAQ "Arduino."

The Arduino is fragile, like all DAQ's. We will need to be careful with our Arduino's. They will be destroyed by putting too large a voltage or electrical current (things you will study in PH220 if you haven't already) into the input pins. They also can be destroyed by being dropped or squashed, so some caution is warranted.

Let's start our study of computer control by controlling something simple with our Arduino DAQ. Earlier we thought about turning an apparatus on or off. We can practice doing this with any device. While it might be exciting to try this with a nuclear reactor, it will be easier (and safer!) to start off simple. Let's turn on and off Light Emitting Diode (LED) lights.

## 1.1 First Computer Control: LED blink (Step-by-step)

Your Super Starter Kit has a set of light emitting diodes (LEDs), a set of resistors, some wires, and a prototyping board. This last item helps hold the other things in place so we can make electricity go through them. Here is what we will build for our light blinking experience:



You are probably familiar with words like “voltage” and “current” even if you have not yet studied these in PH220. You have been plugging in things for many years now. And so you know that “voltage” has something to do with electrical energy. That is enough for now. Our voltage source is just a source of energy to make our circuits work.

A resistor is kind of what it sounds like. It tries to stop or slow down the electricity in the circuit. The LED is just a circuit element that lights up when electricity goes through it. You see them on computers and cell phones as indicators that something has happened.

Let’s assemble our system one step at a time.

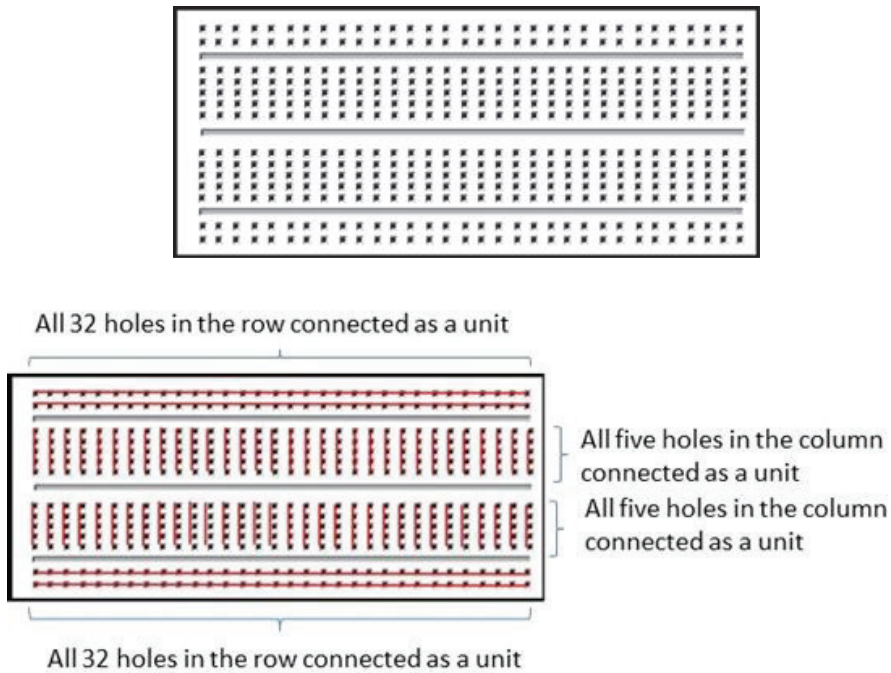
### 1.1.1 Prototyping boards

Let’s start with your prototyping board.

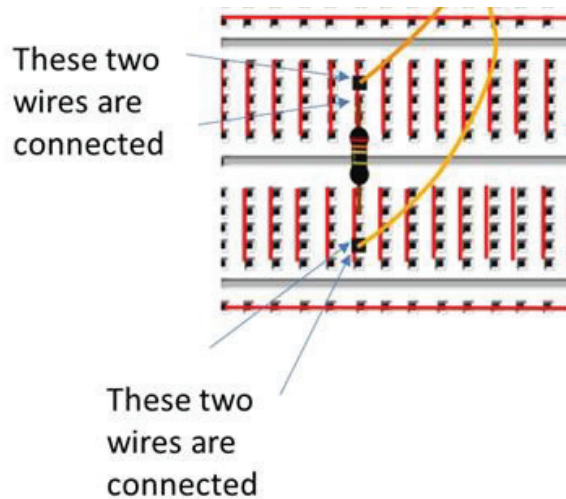
You might notice that the LED and the resistor in the diagram seems to be stuck in some strange board full of holes. This is our prototyping board. You should have one in your Arduino kit. Your instructor will have several that you can borrow if you need another one.

We will connect a lot of electrical wires together in this class. We have devices to make this job easier. Our prototyping board is one of these. They are officially called prototyping boards, but you might hear them called proto-boards, or even “breadboards.” They are designed to allow you to put an electronic element on the board and then connect other things to that element. The boards look like this.

Notice that in the center of the board there are sets of five holes. Under the board surface, these holes are connected together as shown by the red lines in the next figure.



Any wire that goes in one of the connected holes is therefore connected to any other wire that goes in the set of five connected holes. So you can connect up to four other wires to the first wire. In the next figure, you can see an example of a resistor that is placed on the proto-board and is connected to two yellow wires.



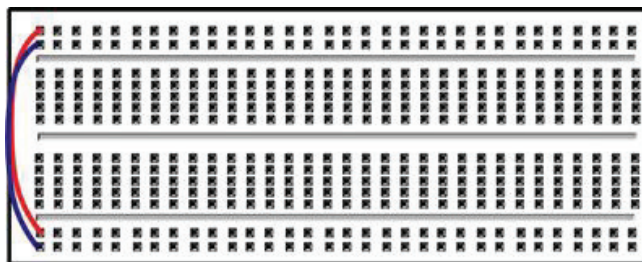
Notice that each end of the resistor is connected to a wire by placing the end of the resistor in one of a set of five connected holes and by placing a wire in another of the set of five connected holes. This would be a silly circuit to build. There is no connection to a source of electrical energy. But this gives the idea of how the prototyping board works.

There are also two long rows of holes on the side of the proto-board. These are usually all connected along the whole row.



Not all proto-boards are alike when it comes to these long rows. But most are connected along the whole row. These long rows are often used as a convenient way to make input power available all along the whole board. Of course, once you know how the board is wired underneath, you can use the connections any way that is consistent with that design. We could skip using proto-boards and just connect everything with wires and alligator clips. But proto-boards often make holding everything in place much easier.

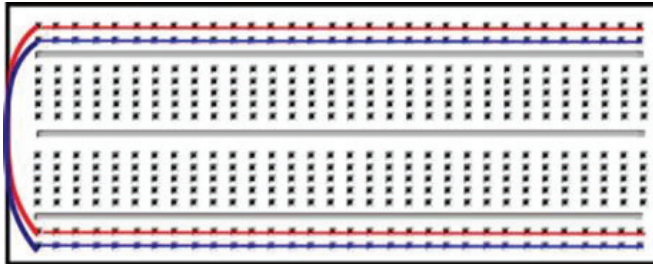
In today's lab experience, let's start by using the long rows as a place for electrical energy to be used. We have long rows on the top and bottom of the proto-board. For our first experience, we will use just 0V and +5V. Let's make these voltages available on both the top and the bottom of the board by wiring the two sets of rows together.



Looking at our wiring diagram for the inside of the proto-board we can see that how the top row and the second from the bottom row have been connected with a wire (it is red if you are reading this in color) and the second from the top and the bottom row are also wired together (the wire is blue if you are seeing this in color). And we know that these entire rows are wired underneath.

Some proto-boards even have red and blue lines to indicate that these rows can be used for electrical power. Some of ours do.



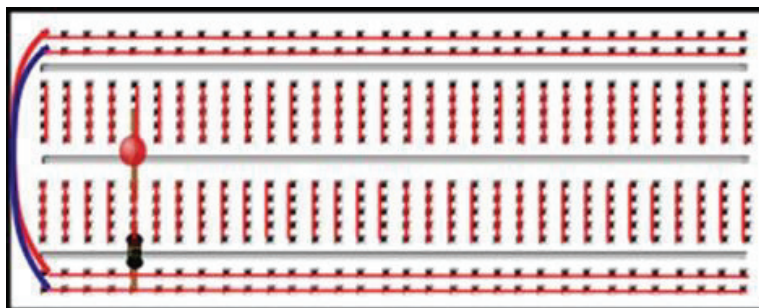


Next let's add our LED light and our resistor.



Notice that the LED spans the gap between the top and the bottom of the board.

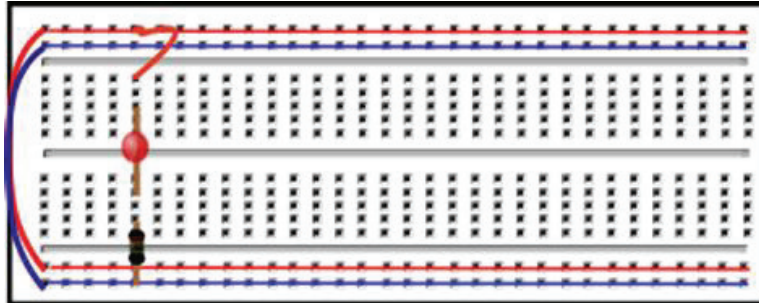
In the next figure I have included our red connection lines so we can keep in mind which parts of the proto-board are connected underneath.



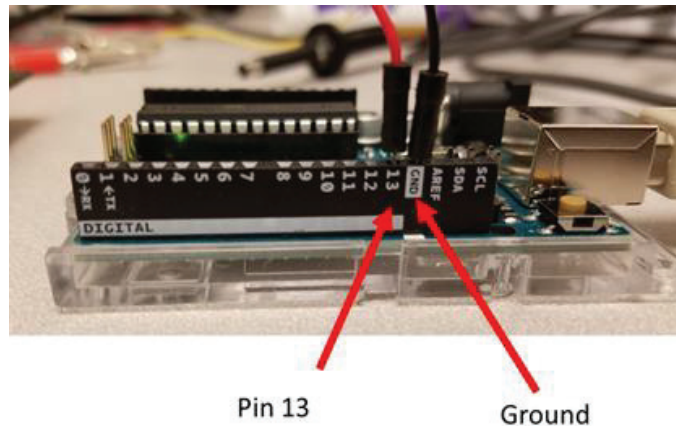
We placed one LED light wire in one column. On the other side of the board we place the other wire. The wires that come out of an electronic device are often called “leads” and I will use this term. So one LED lead goes in a group of five holes on one side of the board and the other lead goes into a group of five hole on the other side of the board. If you look closely, you will see that one side of your LED is flat. The lead on the flat side should be toward the bottom (toward our 0V connection that we will make). LED lights only work one direction. So when using LED's, if they don't work, turn them around.

## 8CHAPTER 1. INTRODUCTION TO THE ARDUINO AND COMPUTER CONTROL

Notice that the way we have done this one LED lead and one resistor lead are connected in one group of five holes. The resistor is connected to the very bottom row. The top of the LED is not connected to anything yet. It is in a set of five holes that are connected together, but we need another wire to connect the top of the LED to the +5V. Let's choose the top of our power rows to be +5V and add the wire to connect the LED.

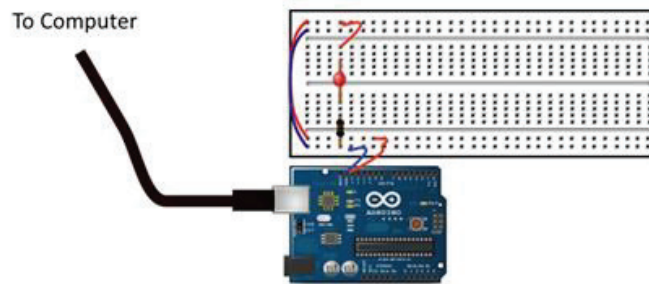


Now we can wire our light to our Arduino. This will take two wires. One should be wired to pin13. The pin numbers are given on the side of the wiring areas.



This pin 13 is a digital output. That means it will either have 5V or it will have 0V. The 5V will be our “on” value that could turn on an instrument. In our case it will turn on our LED light. The 0V is the “off” value. When pin 13 has a value of 0V our light will go off. We know that voltage is related to electrical energy. We will study voltage more next week. But for now we need to know that voltage is a comparison. So we need to know “5V compared to what?” We compare to the voltage of the ground. This is literal. Most buildings have a large copper rod pounded into the ground. All of the plugs in the building have one of their three wires connected to this rod. Thus they are all “grounded.” This is our reference. Our computer is connected to a plug so it is grounded.

Our Arduino is connected to the computer so it is grounded. And one of our pins is a connection to the ground. It is marked “GND.” This will be our 0V. So we need to connect pin 13 and the GND pin to our +5V and our 0V rows on our proto-board. The next figure shows one way to do this.



We now have our “hardware” built for blinking a light. But we need to give instructions to our Arduino that tells it what to do with pin 13 and pin GND so our light will blink. Our Arduino has a small computer on board, and we need a computer program to be uploaded to the Arduino for this small computer to run. We will write and upload that program next.

### 1.1.2 LED light blink Sketch

You wrote computer programs in Python in PH150. Our Arduino programs are similar. They have loops and mathematical statements. There are some differences as well. Our Arduino programs have special code “objects” for controlling our Arduino. Most Arduino programs are very short. We just need instructions to tell the Arduino what to turn on, turn off, or what data to collect. For today’s lab, we just need to address the digital output pins.

Let me introduce the commands we will use. Each digital output has a number. The code defines a variable of type integer (int) and sets it equal to the pin number. That pin can be HIGH or LOW with HIGH = +5V and LOW = 0V. Each digital pin needs to be set up for output or input. This is done with a pinMode statement:

```
pinMode(ledPin, OUTPUT)
```

The variable ledPin is the pin number (for us 13). And the key word “OUTPUT” sets up our pin 13 to turn things on or off.

To set the pin to HIGH use a digitalWrite command

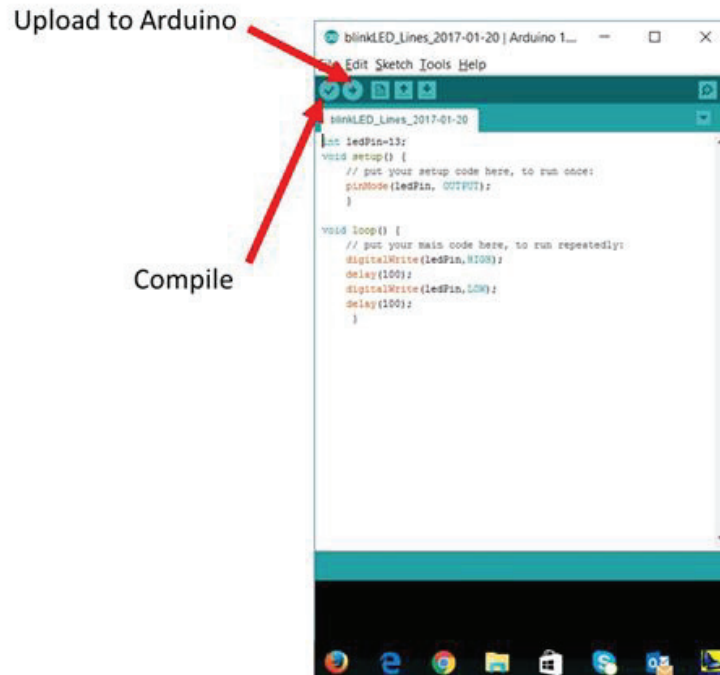
```
digitalWrite(ledPin,HIGH);
```

and finally, to let the light be on for a while, use a delay command where the delay time is given in milliseconds (ms)

```
delay(100);
```

These would not be normal Python commands. They are part of a specific code library for use in making things with Arduino's.

We have a special development environment for programing our Arduino as well. It looks like this.



We will need to install this development environment. To do so we go to <https://www.arduino.cc/en/Guide/HomePage> and choose to instal the Arduino IDE for your computer. IDE stands for “integrated development environment.” This is the environment that you see in the last figure. It has two unusual buttons on its toolbar. One is the “compile” button. When you used Python in PH150, you could run your code by pushing on a green arrow button in most development environments. If you used Jupyter scripts, they also had a function to run your code. But our Arduino is not big enough to be able to run code this way. We need to convert our human-understandable code that looks mostly like Python into a digital format that the Arduino can understand. The compile button does this. As the code is converted, the Arduino software looks for errors in the code. If there are errors, it will tell you at this point. This is different than Python which only told you about errors as the code ran. But remember we don’t want to run our program on our computer. We want it to run on our Arduino. So this check is good before we send the translated code to the Arduino.

We also need a way to send our code to our Arduino and that is what the upload to Arduino button does. Once the code is compiled without errors,

connect the USB cable to the Arduino and push the upload button.

There are also some syntax differences between the Arduino computer language and Python. If you have taken CS124 or know the “C” language, you will recognize some of these changes.

- Everything in a function needs to be in curly braces { }
- Indenting is a good idea, but not required
- comments can be started with two slashes //
- every line needs a semicolon at the end;

### 1.1.3 Arduino LED light blink sketch (program)

We are ready to write code to blink our LED light. Here is an example code. We write this code right in the Arduino development environment main window. You can download it directly here

```

////////////////////////////////////
//Arduino Sketch to blink one LED light
//  Written by Brother Lines
//  (place your name here in your code)
//  Feb 6, 2017
//
//  Define our Arduino Variables
//      We will call pin 13 "ledPin"
////////////////////////////////////
int ledPin=13;

////////////////////////////////////
// Arduino setup function comes next
//  Every Arduino sketch needs a setup function
//  We will set up our ledPin (pin 13) as an output pin
void setup() {
  // put your setup code here, to run once to set up:
  pinMode(ledPin, OUTPUT);
}

////////////////////////////////////
// Arduino loop function
//  Every Arduino sketch has a loop function
//  This is where we put what we want the Arduino to do
//  The Arduino will do whatever is in the loop function
//  until the Arduino is unplugged.

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

```

```

// turn on the LED
digitalWrite(ledPin,HIGH);

// leave in on for 100ms
delay(100);

// turn off the LED
digitalWrite(ledPin,LOW);

// leave in off for 100ms
delay(100);
}
////////////////////////////////////
////////////////////////////////////

```

Notice that we used a lot of comments. There are only ten lines that are actually required to make the sketch run.

```

int ledPin=13;
void setup() {
    pinMode(ledPin, OUTPUT);
}
void loop() {
    digitalWrite(ledPin,HIGH);
    delay(100);
    digitalWrite(ledPin,LOW);
    delay(100);
}

```

The comments are not required for the sketch to work, but they help us remember what we did later. In our class, **comments are required for full credit**. So don't leave out the comments. In fact, you can add any comments that might help you remember why the code works.

Also notice that Arduino programs are called “sketches.” Most of the commands are special Arduino commands. And luckily, they make sense when we read them. Try it out. Type in this sketch including the comments and compile and upload it to your Arduino. If all goes well, the LED light will begin to blink. If all did go well, go on to the next section. If it did not, call over an instructor.

You will want to make a record of what you have done so you can look it up later when you need to repeat what we just did in a future project (like making a blinking light on your instrument so you know it is working!). Scientists record what they do in lab notebooks. Traditionally these are bound blank books that you write in with a pen. But in industry today companies use electronic lab notebooks. You are probably very familiar with Google Docs or other on-line document sharing tools. We will use such a tool that is widely used in both industry and academia. That is a Jupyter notebook. Note that Jupyter



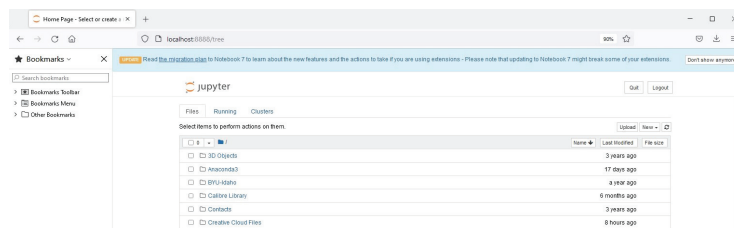
## 1.1. FIRST COMPUTER CONTROL: LED BLINK (STEP-BY-STEP) 13

notebooks on their own are not secure enough to protect intellectual property. They would need to be safeguarded withing a professional lab notebook system. But our homework for this class isn't considered a national security risk or even an intellectual property risk. So Jupyter notebooks are just fine for us. You may have seen them in other classes that use python. If not, don't worry, we can help you become comfortable with them.

Chances are you already have the Jupyter system because it comes standard as part of the Anaconda python system. If you installed Anaconda, you can just go to your list of programs and start Jupyter. If you don't find it, call over your instructor or TA. Here is how that looks in Windows (For those with Macs, it will look more Macish).

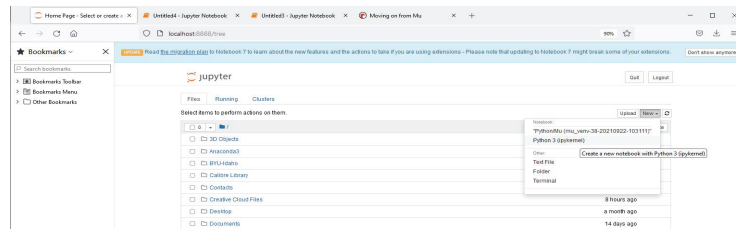


This is a little weird, but Jupyter will start in your web browser!

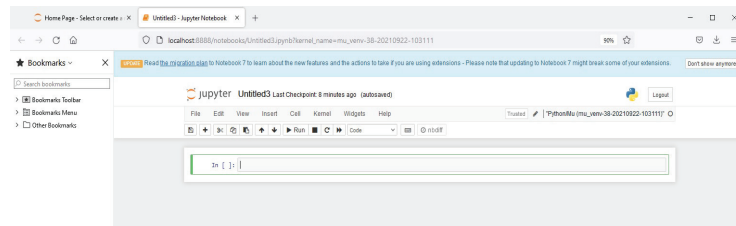


Jupyter notebooks use a browser as the program to display things and to let us input things into the notebook. You open a new notebook by going to the “New” button on the top right and selecting the Python3 option.

You get a new tab in your browser with an input window. The input window



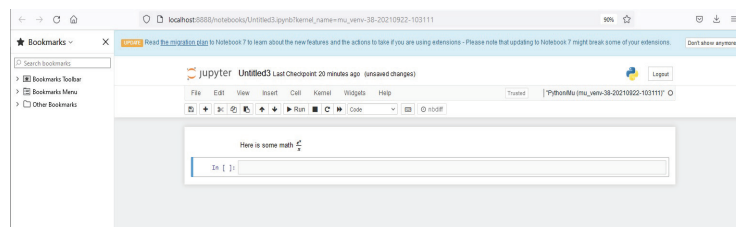
is called a “cell” and you can choose what kind of cell it is by going up to the Cell drop down menu and choosing “Cell Type.” Notice that we get the choices “code” and “markdown” among others. Choose the “Markdown” option and you can input text, like a description of what we have been doing so far in this lab!



The “Code” type cells are a good place to put your Arduino code. And you can drop photos into the Markdown type notebook cells as well. You can even type math into the Markdown cells by placing the math symbols between dollar signs. That will take some practice, but the code

Here is some math  $\frac{e^x}{x}$

will become this!



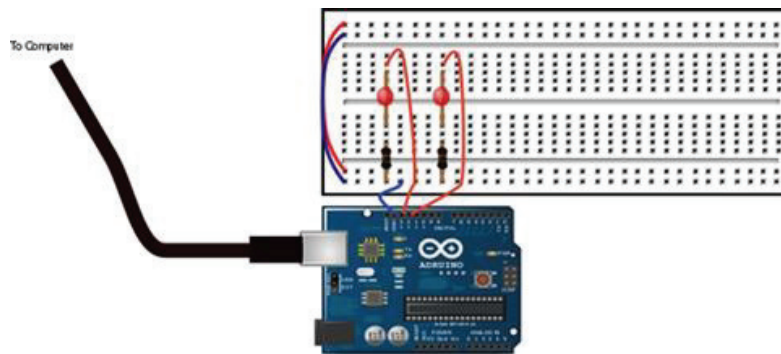
It looks like real math! This will take some getting used to. So start with what we have done so far today. Make sure you have saved sketch. Then write a description of what we have done so far in a Markdown cell and copy your code into a Code cell. Maybe take a photo of your hardware set up and include it in another Markdown cell. We will build on this sketch, so spend some time documenting what you did in your lab notebook so you can refer to it later.

An example lab notebook can be downloaded [here](#) for reference.



Now let's see if we can apply what we have learned. Let's change our hardware so that it has two LED lights. Most of the hardware setup will be the same. But We will wire them to two Arduino pins, say 12 and 13 (along with GND of course) and have them blink, but blink independently.

This is why we can't wire both LED lights to the top row and wire the top row to pin 13 as we did last time. That would make both lights blink at once, but would not let one be off and the other on. Instead, let's wire the top lead of one LED directly to pin 13 of our Arduino, and let's wire the top lead of the other LED to pin 12. The two resistors can share a connection to the GND pin, so let's keep using the 0V bottom row of the proto-board. Your hardware should look something like this.



```

////////////////////////////////////
//Arduino Sketch to blink two LED light
//    Written by Brother Lines
//    Feb 6, 2017
//
//    Define our Arduino Variables
//        We will call pin 13 "ledPin1"
//        We will call pin 12 "ledPin2"
////////////////////////////////////
int ledPin1=13;
int ledPin2=12;

////////////////////////////////////
void setup() {

```

```

// put your setup code here, to run once:
// set both pins to OUTPUT
pinMode(ledPin1, OUTPUT);
pinMode(ledPin2, OUTPUT);
}

/////////////////////////////////////////////////////////////////
void loop() {
  // put your main code here, to run repeatedly:
  // turn on one LED and turn off the other
  digitalWrite(ledPin1, HIGH);
  digitalWrite(ledPin2, LOW);
  // wait
  delay(100);
  // now switch so the first LED is off
  // and the second is on
  digitalWrite(ledPin1, LOW);
  digitalWrite(ledPin2, HIGH);
  // wait
  delay(100);
}
/////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////

```

Again save your sketch and maybe take a photo of your hardware set up. Place both in your lab notebook along with notes on how you got it to work. Make sure others at your table are able to get their setup to work.

### 1.3 Third Computer Control: Two LED blink using math

Let's leave our hardware alone in the two LED setup from the last section. And let's make the LED's blink the same way. But this time, let's calculate when they should be on or off. Why would we do this? Because sometimes in computer control of experiments we need to turn something on or off based on a calculation. You may have your computer watching to make sure the experiment doesn't get too hot or cold. The Arduino can bring in temperature information, but you would have to write the code to tell it to turn off the heater when your experiments gets to hot and to turn it on when it gets too cold. This could be done with a mathematical comparison. We will use such a comparison in the next sketch.

Suppose we want to know if a number is even or odd. Even numbers are evenly divisible by 2. We could divide a number by 2 and see if the remainder is zero. Our Arduino language has a good set of mathematical functions. The remainder function is a "%" sign. For example

### 1.3. THIRD COMPUTER CONTROL: TWO LED BLINK USING MATH17

$$\begin{aligned}3\%2 &= 1 \\6\%2 &= 0\end{aligned}$$

Let's have one light turn on if a number is even, then switch to the other light if the number is odd.

In our code we will introduce a variable,  $i$ , that we will increment (add one to) every time the loop runs. So the first time the Arduino loop runs it will be zero (even) and the next time 1 (odd) and the next time 2 (even) and the next time 3 (odd) and so on. If you studied Python you would call such a variable an “integer” and might even know to call it a “loop counter.”

In our Arduino sketch we will test to see if  $i$  is even in an if-statement. If-statements go like this

```
if (test condition ) {  
    do something;  
}  
else {  
    do something else;  
}
```

Notice that the parts of the if-statement need curly braces. Our condition to test is

```
i % 2 == 0
```

Note that there are two equals signs. That makes it a test for equality rather than an assignment. So we will have an if-statement like this

```
if (i % 2 == 0 ) {  
    digitalWrite(ledPin1,HIGH);  
    digitalWrite(ledPin2,LOW);  
    delay(1000);  
}  
else {  
    digitalWrite(ledPin1,LOW);  
    digitalWrite(ledPin2,HIGH);  
    delay(1000);  
}
```

One last addition, our Arduino language has a shortcut for the statement

```
i=i+1
```

It is simply

```
i++
```

We will use this to make  $i$  increase by one each time the loop runs. The whole code might look like this.

```

////////////////////////////////////////
// code to blink two LED's using a mathematical expression
// to determine when they should light. Note that the
// Arduino code is closer to C++ than python.
////////////////////////////////////////
int ledPin1=13;
int ledPin2=12;
int i=0; //loop counter

////////////////////////////////////////
void setup() {
  // put your setup code here, to run once:
  pinMode(ledPin1, OUTPUT);
  pinMode(ledPin2, OUTPUT);
}

////////////////////////////////////////
void loop() {
  // put your main code here, to run repeatedly:
  // blink the LED's with the number of blinks being
  // the Fibonacci sequence.
  if (i % 2 == 0 ) {
    digitalWrite(ledPin1,HIGH);
    digitalWrite(ledPin2,LOW);
    delay(1000);
  }
  else {
    digitalWrite(ledPin1,LOW);
    digitalWrite(ledPin2,HIGH);
    delay(1000);
  }
  i++;    // increment the loop counter by adding 1 to i
}

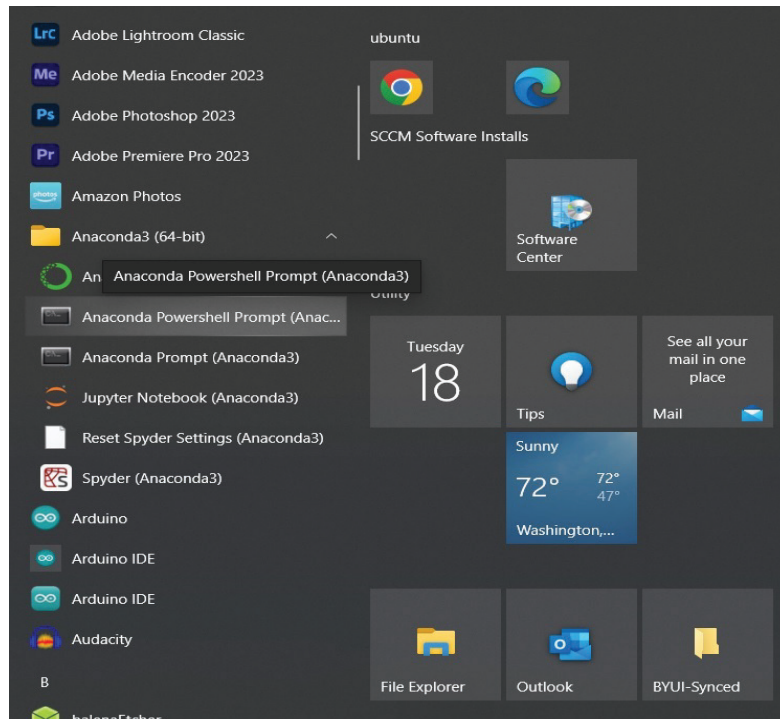
////////////////////////////////////////
////////////////////////////////////////

```

Again save your sketch. You should probably say in your lab notebook that you used the previous hardware setup. You might want to describe in your lab notebook how the mathematical algorithm works.

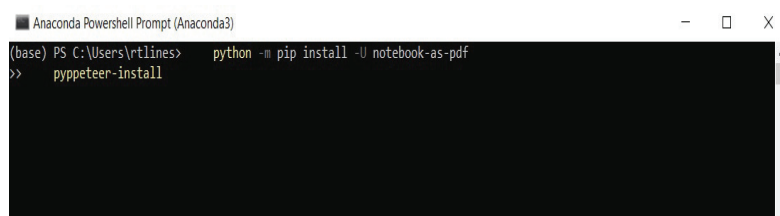
But at this point, you probably want to be able to submit your lab notebook for grading. This seems simple, but we need to modify your Anaconda system to make it work well. To do this, start an Anaconda Powershell Prompt.

### 1.3. THIRD COMPUTER CONTROL: TWO LED BLINK USING MATH19

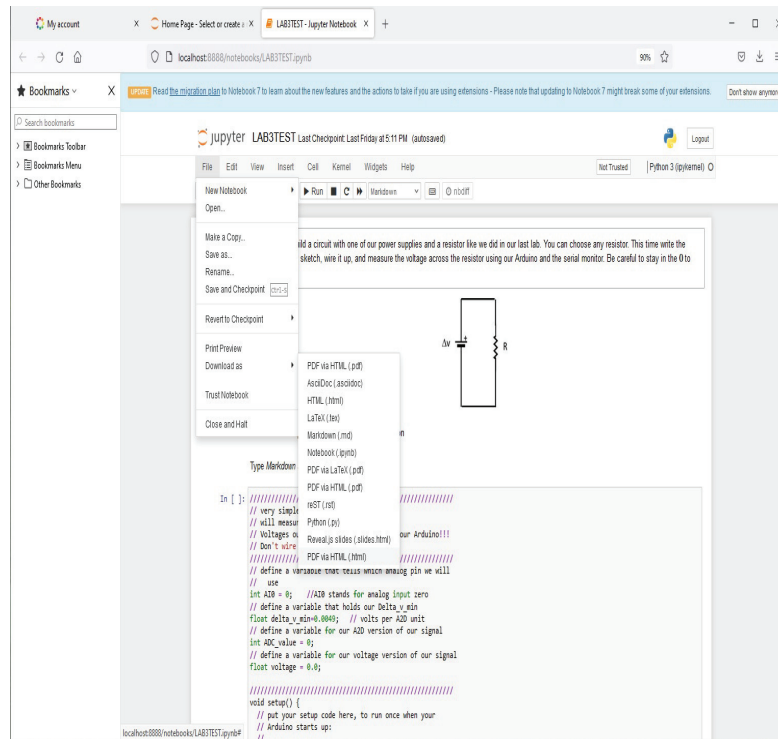


This opens a big black box. In the black box type the following lines.

```
python -m pip install -U notebook-as-pdf  
pypeteer-install
```



A lot of words and symbols will go by if you are successful. After they are done you can try to save your Jupyter notebook as a pdf file for submission. From the File drop down menu choose “Download as” and then “PDF via HTML.” You should find that your browser downloaded a PDF version of your notebook into your download directory. Try this before you go on to the next part of the lab.



Suppose instead of LED lights we had large radio transmitters. And suppose we were part of the Search for Extra-Terrestrial Intelligence (SETI). We wish to send a message to any intelligent life that they would understand. Intelligent life probably would be able to do mathematics and would understand how mathematics occurs in nature. One sequence of numbers that occurs over and over again in nature was discovered by Fibonacci. Let's blink our LED lights (representing those powerful radio transmitters) in the Fibonacci sequence.

We need to know now to calculate the Fibonacci sequence. One method is to know that the sequence goes like this

$$0, 1, 1, 2, 3, 5, 8 \dots$$

and that we can find the next number in the sequence by choosing  $f_1 = 0$  first, then  $f_2 = 1$  then using the formula

$$f(x-1) + f(x-2)$$

Let's see that this works. For the first of the sequence, we just write the 0. For the second we just write the 1. Then for the third

#### 1.4. FOURTH COMPUTER CONTROL: TWO LED BLINK IN THE FIBONACCI SEQUENCE21

$$\begin{aligned}f_3 &= f_2 + f_1 \\&= 1 + 0 \\&= 1\end{aligned}$$

So far so good. Let's try the next in the sequence

$$\begin{aligned}f_4 &= f_3 + f_2 \\&= 1 + 1 \\&= 2\end{aligned}$$

Again it worked. For the next one

$$\begin{aligned}f_5 &= f_4 + f_3 \\&= 2 + 1 \\&= 3\end{aligned}$$

and though we won't prove it, it works for every member of the sequence. See if you can figure out how to write this code. An example is given below, but see if you can figure out what the code should be.

This example is a much more complex version of a mathematical based computer control.

```
//////////////////////////////////////////
// code to blink two LED's using a mathematical expression
// to determine when they should light. Note that the
// Arduino code is closer to C++ than python.
//////////////////////////////////////////
int ledPin1=13;
int ledPin2=12;
int i=0; //loop counter

//////////////////////////////////////////
int fib_count=0; // number of blinks based on Fibonacci
int i_max=10; // maximum Fibonacci number before
// starting over

//////////////////////////////////////////
void setup() {
  // put your setup code here, to run once:
  pinMode(ledPin1, OUTPUT);
  pinMode(ledPin2, OUTPUT);
}
```

```
// calculates the Fibonacci sequence using recursion
int fib(int x) {
    if (x==0)
        return 0;
    if (x==1)
        return 1;
    return fib(x-1) + fib(x-2);
}

//////////////////////////////////////
void loop() {
    // put your main code here, to run repeatedly:
    // blink the LED's with the number of blinks being
    // the Fibonacci sequence.
    fib_count=fib(i);
    if (i % 2 == 0 ) {
        // turn off one light
        digitalWrite(ledPin2,LOW);
        // now blink the second light fib_count times
        for (int n=0; n<fib_count; n++) {
            digitalWrite(ledPin1,HIGH);
            delay(100);
            digitalWrite(ledPin1,LOW);
            delay(100);
        }
    }
    else {
        // turn off the other light
        digitalWrite(ledPin1,LOW);
        // now blink the first light fib_count times
        for (int n=0; n<fib_count ; n++) {
            digitalWrite(ledPin2,HIGH);
            delay(100);
            digitalWrite(ledPin2,LOW);
            delay(100);
        }
    }
    // increment i
    i++;
    // limit our blinks to the first i_max Fibonacci numbers
    if (i>i_max) i=0;
}

//////////////////////////////////////
```



#### 1.4. *FOURTH COMPUTER CONTROL: TWO LED BLINK IN THE FIBONACCI SEQUENCE*23

---

Again save your sketch. You should probably say in your lab notebook that you used the previous hardware setup. You really should describe in your lab notebook how the mathematical algorithm works.

For next week, you should read the lab before coming to class. So your assignment is to read Lab 2.

