LABX_blinkArduino

Created: 4 May 2018

By: Samuel Bechara, PhD

Table of Contents

Introduction

In this lab, we will setup the Arduino in MATLAB. Before we start, it is important to remember that Arduinos are great for giving tasks but we need to be careful of what those tasks are. We need to make sure we understand **how to tell MATLAB to talk to the Arduino**.

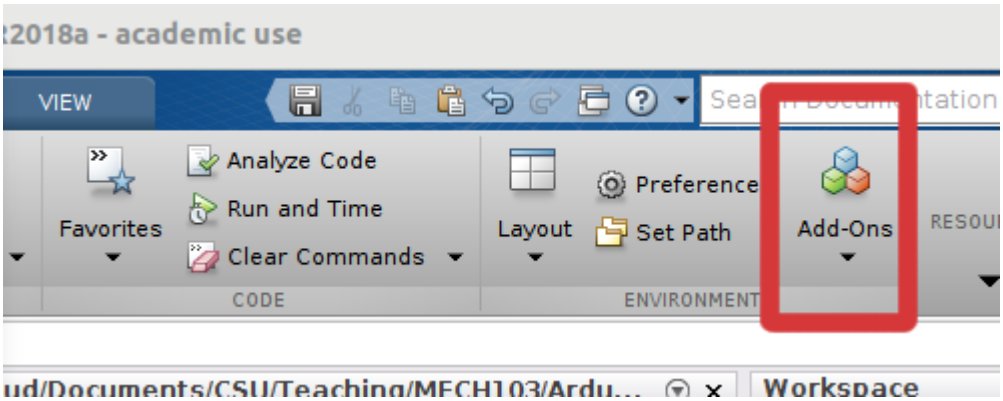
There will be 3 main steps:

1. **Download and install the support packages**. MATLAB is already a monster of a program. Mathworks says you can expect 3-4gb in a typical installation. To save space, the functionality to program an Arduino is not included in the base installation.
2. We need to **tell MATLAB we want to talk to the Arduino**. To do that, we create an Arduino object. Since Arduinos do what we tell them to do, I recommend giving them names that suggest what you want it to do. Think of Arduino, sitting at a restaurant, waiting for you. It's fancy, so you need to tall the Matier D' you would like to be sat with Arduino so you guys can talk. Business or Love? I guess you decide.
3. **Learn Arduino-Matlab-ese**. The weird thing about Arduinos is that they can speak a couple of different languages. The majority of the code you see online for Arduinos will be written in "Arduino programming language" but we will be talking MATLAB. Its all aces, we already speak MATLAB and there is no code online to copy*!

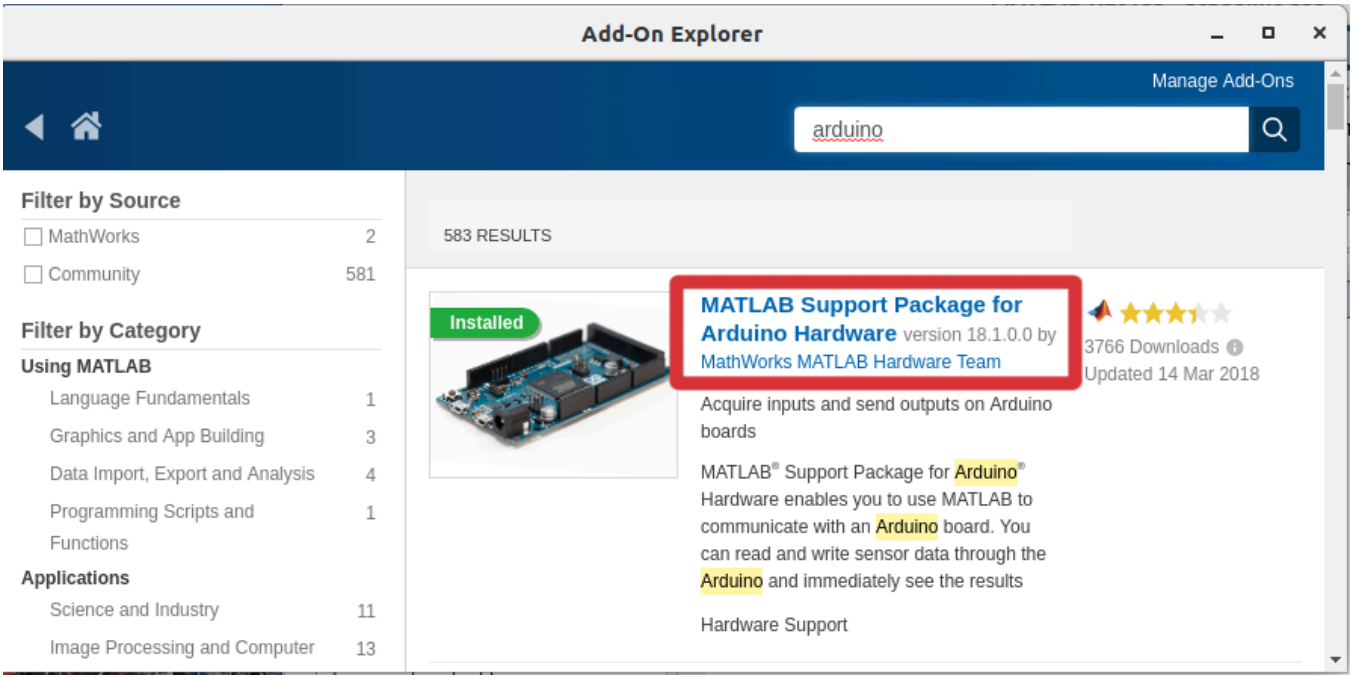
* *I acknowledge that my pros/cons list may look different than yours.*

Step 1) Download Support Package

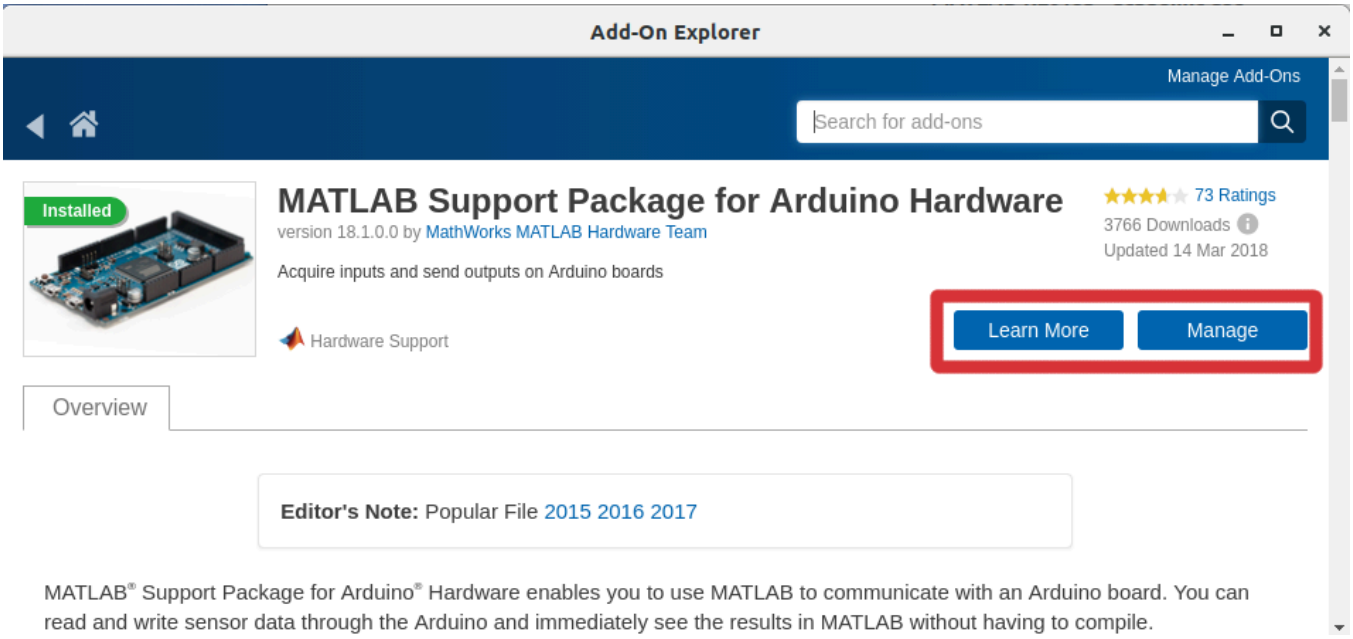
Step 1.1) In the "HOME" tab on the main page click the Add-Ons button.



Step 1.2) In the window that pops up, search for Arduino and install the "MATLAB Support Package for Arduino Hardware" by clicking the text highlighted below. Make sure you have your Arduino with a USB cable handy.



Step 1.3) Next where mine says, "Learn More" and "Manage" highlighted below, you should have an option to install. **Begin the installation procedure by clicking install.** When you are all finished, you should see these buttons. This Manage button is how you can restart the automatic setup wizard that begins after installation (hence needing the USB cable handy).



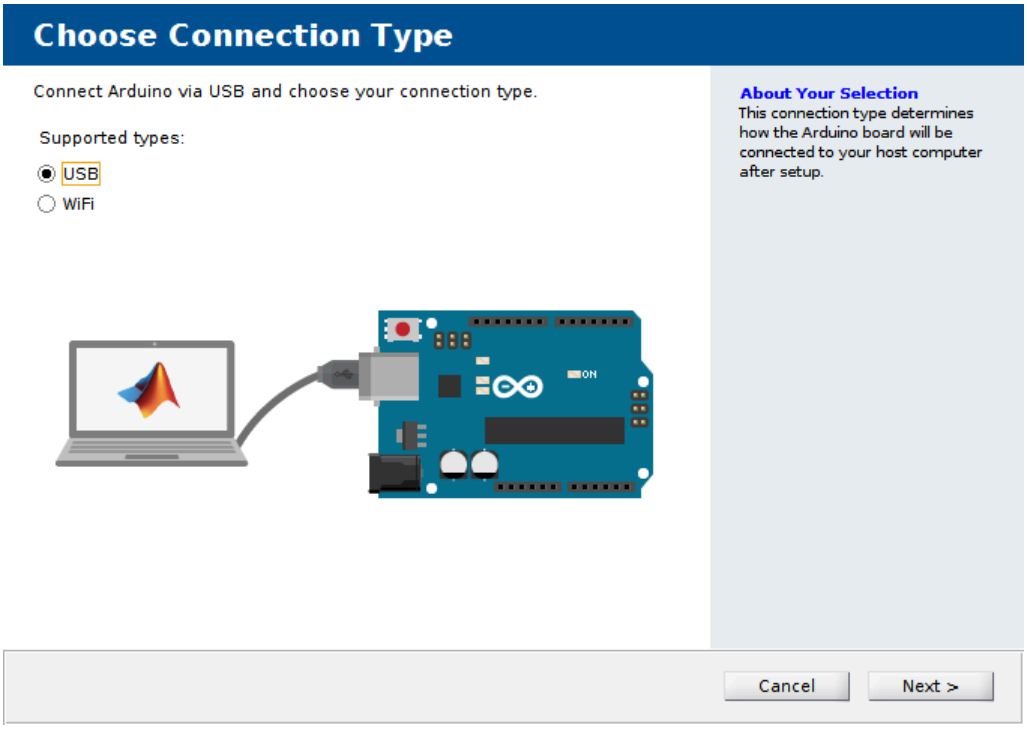
Step 2) Follow the automated setup wizard

The MATLAB Arduino setup wizard will install software onto the arduino so that the Arduino can talk to MATLAB. It is pretty straightforward and easy to use. If you are using Linux, you need to make sure that your permissions are setup correctly. It is critical to note, you can re-enter the arduino setup wizard at any time by typing:

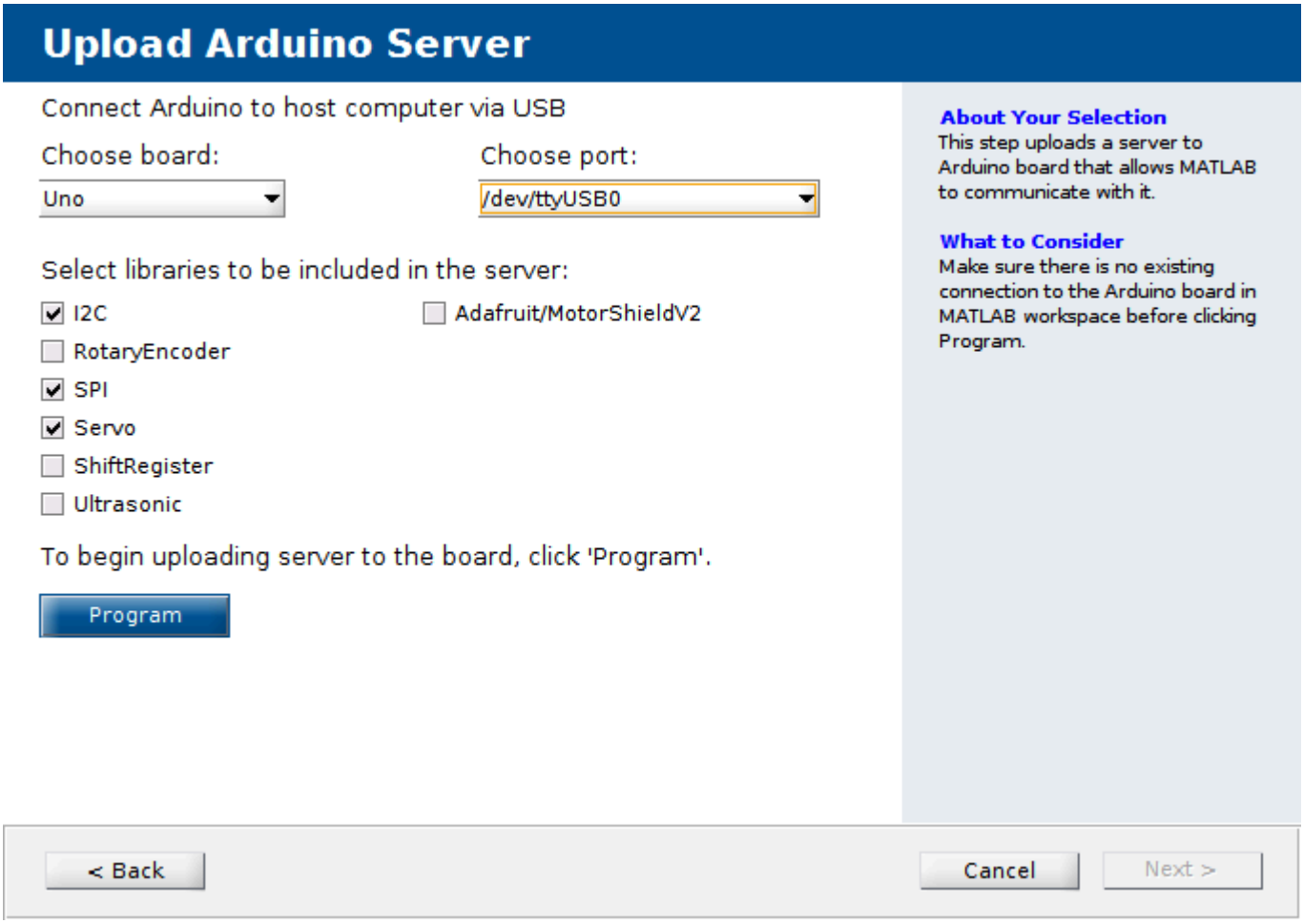
```
arduinosetup % command will launch the arduino setup wizard
```

into the command window. You MAY need to do this at a later time so don't forget this!

Step 2.1) **Follow the automated setup wizard.** You should use USB to connect.

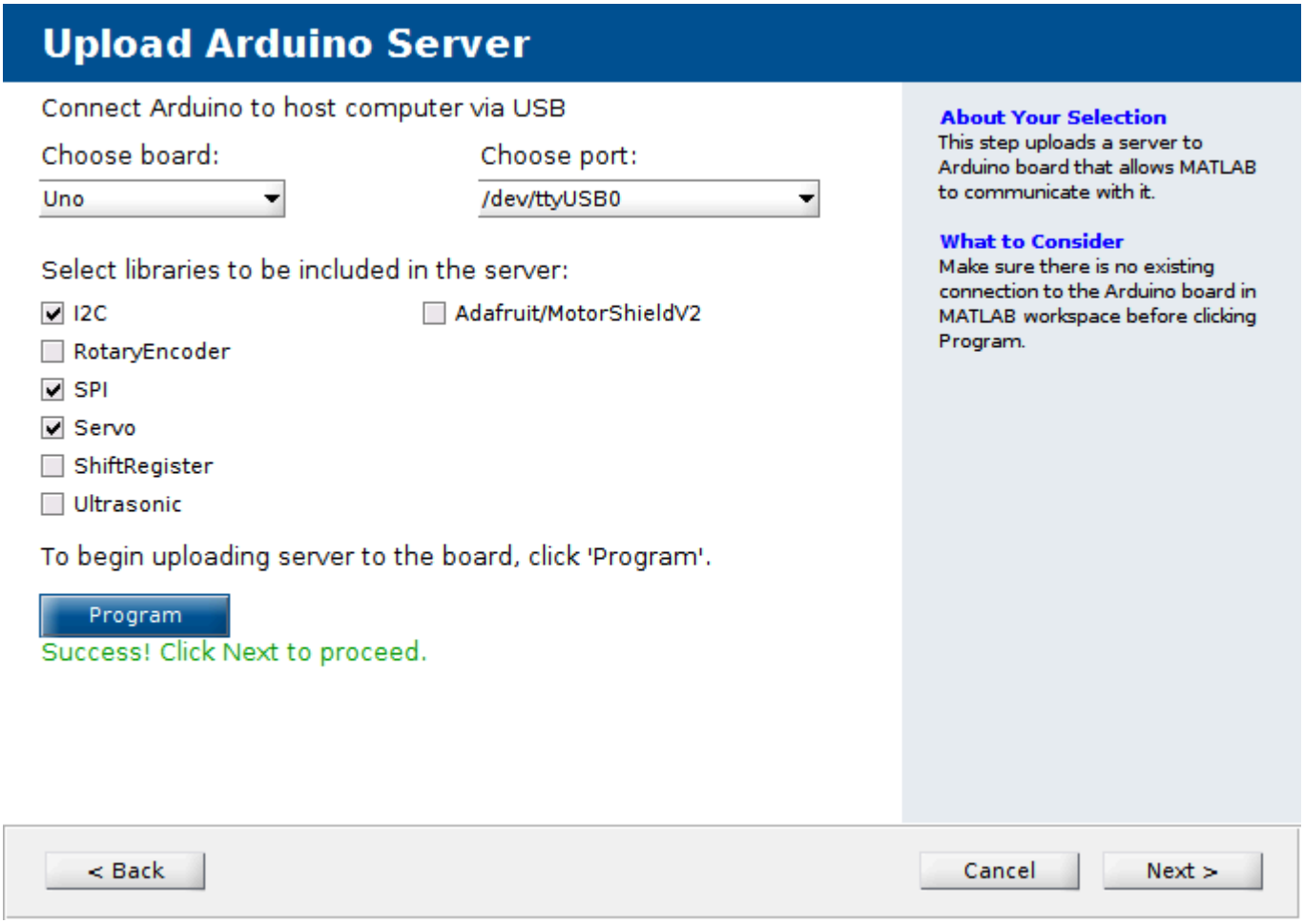


Step 2.2) Use dropdown boxes to specify the Arduino and port. If you have a lot of things plugged into USB drives in your computer you may have several ports. You will know you selected the correct port when you click the "Program" button and you see no errors and lights start flashing on your arduino.

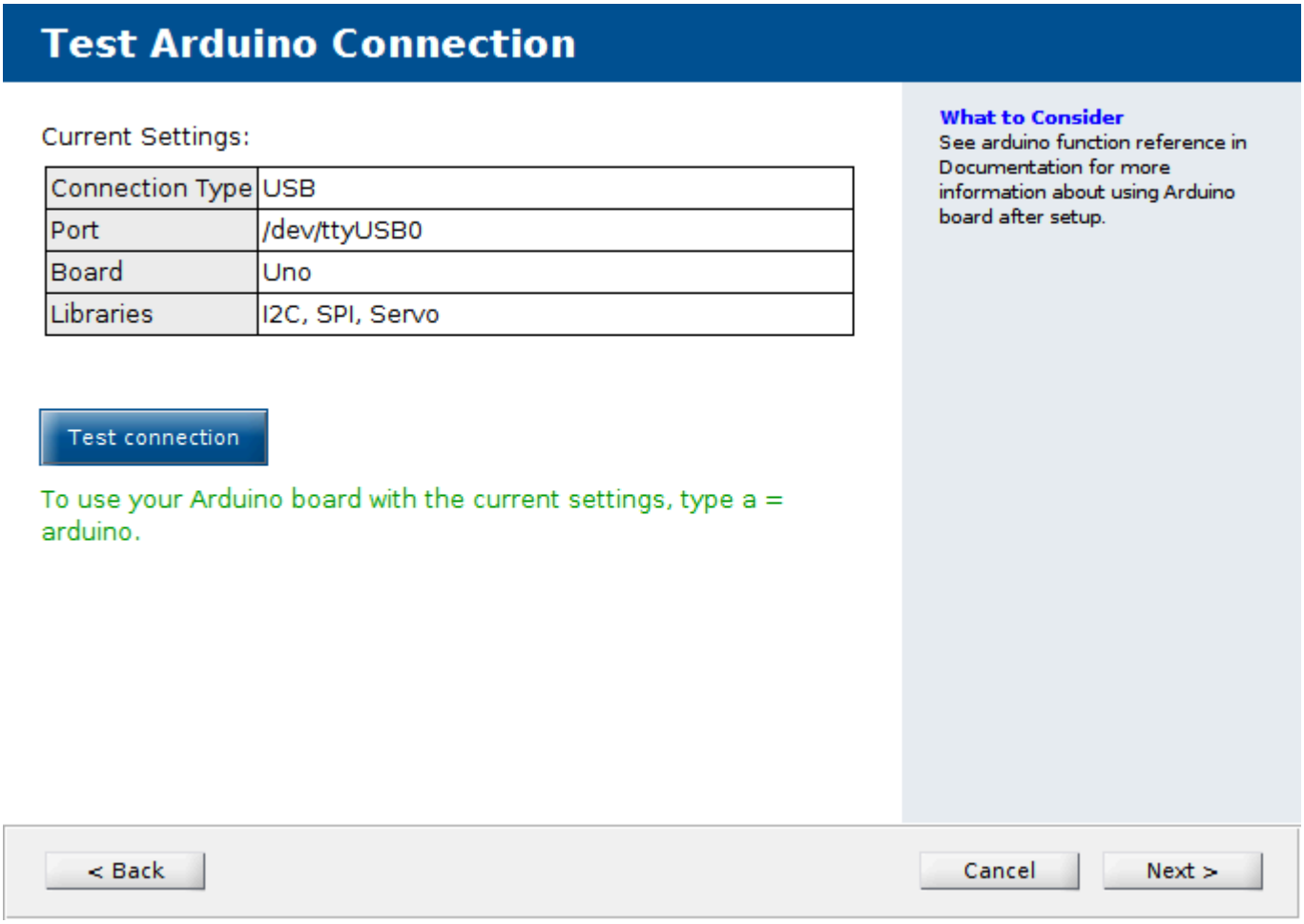


If you encounter problems, it is likely permission errors. **Try and figure it out on your own before seeking help from a classmate or professor!** This works out your brain and helps you to get to know your computer and online help forums.

After you are done, you should see the following screen. Congratulations! Your computer speaks Arduino-MATLAB-ese!



[Optional] Finally, if you would like to test and make sure that everything worked ok, MATLAB presents you with a "Test Arduino Connection" wizard. If you click "Test connection" you should get something like this out.



Step 2) Let MATLAB know we want to talk to the Arduino

```
% Make sure to remove the arduino object from workspace every time you run your code
clear

% Setup Arduino - let the Matire D' know you would like to be seated

% Give the Arduino a name that helps user understand what it does
blinky_Arduino = arduino();
```

IF you get an error that says something like "Cannot detect Arduino hardware" do not panic! All you have to do is specify the port and the board type in the blinky_Arduino line. It should look something like this:

```
binky_Arduino = arduino('/dev/ttyUSB0','uno'); % blinky_Arduino=('port goes here','arduino type goe
```

Note, your port number will be different if you are on windows, macOS, or linux. Use the same port from the `arduinowizard` indicated above in step 2.

```
% Digital13 is the pin for the built-in LED, next lab we will use external LEDs.
ledPin = 'D13';
```

Note: the `ledPin = 'D13'` *line of code is TECHNICALLY unnecessary but helps for readability and so we don't have to keep writing* `D13` *over and over.*

Step 3) Learn Arduino-MATLAB-ese

Ok, we have a way to talk. But we need to make sure we are speaking the same language.

In my mind, it helps to think of Arduinos as a really, really, dumb extremely good looking person.

Ask Arduino (pictured above) to grab you a glass of water.

Example 1 - Won't Work with super dumb person) "Good day, Arduino. This is Dr.B. Yes, my family is flourishing. How is Pat? Send my regaurds. I have found myself in quite the predicament. I am parched and without water. From one old chemist to another, would you be a gem and fetch me a glass of dihydrogen monoxide?"

Example 2 - MIGHT work with super dumb person, but no gaurantees) "Arduino, do as I say. From your present position, stand, walk to the kitchen. Stop after 4.2 seconds. Open the cabinet directly in front of you. Get a glass..."

Luckily for us, Arduinos aren't complicated like people. They really can only do a few things. One of the things they are best at is writing voltages to pins.

Getting Started - Lets Blink an LED.

```
% Blink LED by writing 1 (aka "on") to a pin, waiting, then turning it off.

% To turn it on, we need to write 1, to the correct location on blicky_Arduino
writeDigitalPin(blinky_Arduino, ledPin, 0);
pause(1);
writeDigitalPin(blinky_Arduino, ledPin, 1);
```

Alright. That is pretty much all you need for today. Understand how to turn off/on the LED and how to pause for given periods of time.

The cool thing is, you can just use MATLAB code to do programming stuff. For example, if I wanted to "blink" the LED 4 times I would:

```
% Blink built-in LED 10 times
writeDigitalPin(blinky_Arduino,ledPin,0);
pause(0.25)
writeDigitalPin(blinky_Arduino,ledPin,1);
pause(0.25)
writeDigitalPin(blinky_Arduino,ledPin,0);
pause(0.25)
writeDigitalPin(blinky_Arduino,ledPin,1);
pause(0.25)
writeDigitalPin(blinky_Arduino,ledPin,0);
pause(0.25)
writeDigitalPin(blinky_Arduino,ledPin,1);
pause(0.25)
writeDigitalPin(blinky_Arduino,ledPin,0);
pause(0.25)
writeDigitalPin(blinky_Arduino,ledPin,1);

disp('Done!')
```

Note: This is a terribly inefficient way to blink the led 4 times, there is a much better way that we will learn soon! If you have programmed before you probably know where this is going...

BEFORE YOU MOVE ON

Remember: *before you move on* questions MUST be completed before LAs will help you.

- Write your own pattern on a piece of paper. Try and program it.
- Change the delay timing. Does it change what you expect it to change?
- Make sure that you can get Arduinos to work using MATLAB scripts. Idea, find a loop in a previous program and implement the blinking into a loop structure. Hint: make delays long enough to see!

Sending messages the old fashioned way

You and your friend are racing your yachts around Tahiti (totally relatable, right?) when the ~~kraken~~ attacks your ship and destroys your engine, your radio, and your fancy satellite phone. Conveniently enough you have a signal lamp hooked up to Matlab.

Your signal lamp looks like this but hooked up to a computer.

All you need to do is provide Matlab arrays containing the message you want to send. Fortunately your friend wrote a Matlab function to generate the arrays when you give it a message. Unfortunately your friend is terrible at either Matlab Code or Morse Code and you need to fix the message before you send it to the lamp.

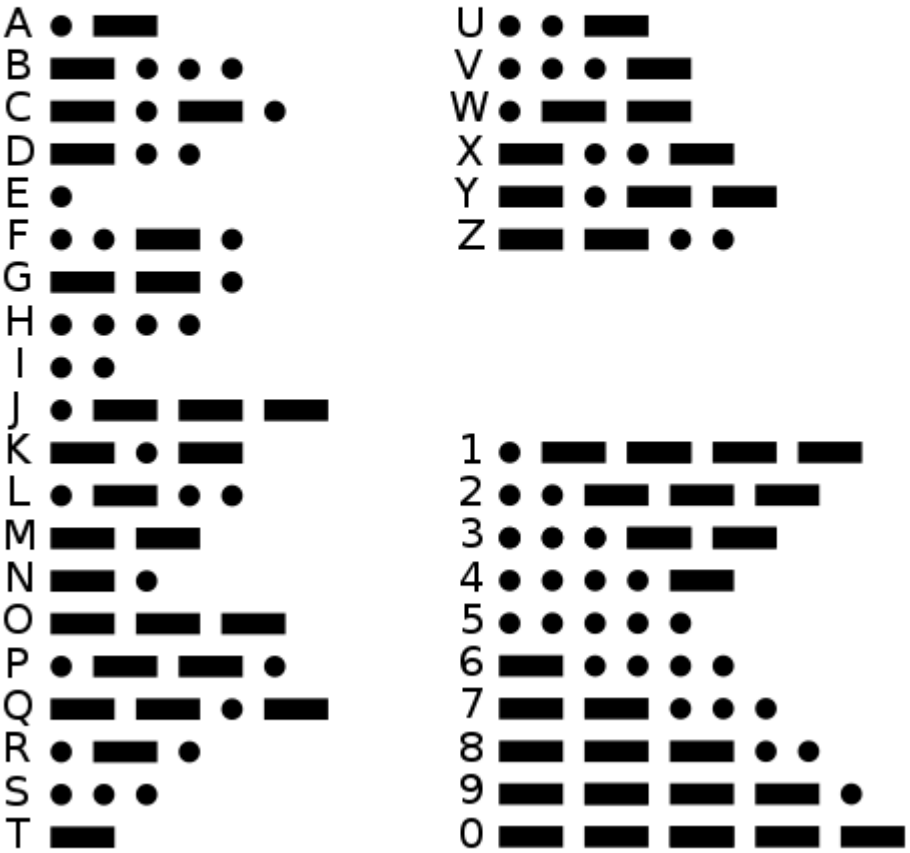
You gave the function the message "HELP" and it gave you the following arrays:

```
H = [1 0 0 1 1 0 1]
E = [0]
L = [1 0 1 1 0 1 0 1 ]
P = [1 0 0 0 0 1 1 1 0]
```

Dots are given by a single 1, and dashes are given by two sequential 1s. See the below chart for a Matlab-Lamp morse code specification.

Matlab-Lamp Morse Code

1. The length of a dot is one unit.
2. A dash is two units.
3. The space between parts of the same letter is one unit.
4. The space between letters is two units.
5. Each word gets its own array

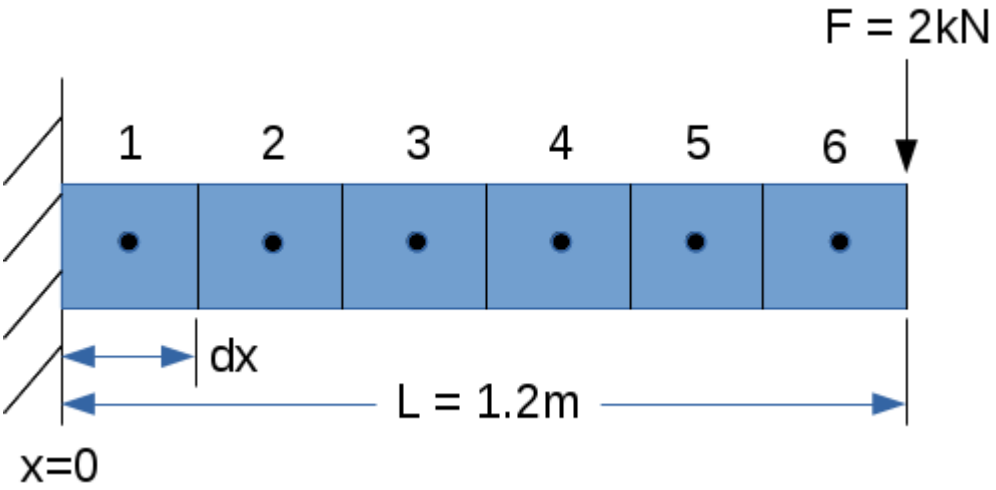


Looking at this spec, you can tell that the L is correct but the other three letters are incorrect. Using array indexing, fix the arrays so that each letter is correct. Then concatenate the arrays together so that they form the whole word (remember to separate your letters with two zeros!). You may want to refer to the Matlab documentation on array concatenation: https://www.mathworks.com/help/matlab/math/creating-and-concatenating-matrices.html (note the syntax is the same for matrices as for arrays).

Write a MATLAB **script** and submit it online to Matlab Grader that meets the following requirements:

- A seperate variable for each of the letters **H**, **E**, **L**, and **P**. These variables should contain the correct sequence of 1s and 0s for the letter's morse code.
- Modify the arrays using a sequence of array indices so that the letter variables conform to the Morse Code specifcation in the diagram. For example, H(3) = ???
- A variable called **message** that contains the concatenation of the individual letters. Don't forget to add in the letter separators specified by the diagram above!

Cantilever Analysis



Every mechanical engineer's favorite class is CIVE260: Statics. Thats why its in the Civil Engineering college! During that class, your professor asks you to perform a finite element analysis (FEA) of the above pictured cantilevered beam. You are asked to perform the analysis by hand, but why would you do that when you have Matlab?

As you know, the first step in performing an FEA is to create a mesh of the object you are analysing. The picture above shows the mesh you are going to use: 6 cells that we are going to pretend are prefectly square and evenly spaced (I just couldn't get them exact in my drawing software). The cantilever beam is 1.2m long and is fixed on the left side at x=0. Each cell is 'dx' wide. Use Matlab's linspace function to create an array containing the cell centers (labelled 1-6 in the figure).

That is a very coarse mesh! Lets get a much finer solution by increasing the number of cells by two orders of magnitude to 600 cells. With the linspace function you need to know the first and last cell centers and the number of cells. Your textbook mentions another method to generate an array in Matlab that only requires the first cell's location, the dx, and the length of the cantilever. Generate an array using that method.

Write a MATLAB **script** and submit it online to Matlab Grader that meets the following requirements:

- A variable called **sixCellLocations** that is an array containing x-locations of the centers of the 6 cells in the diagram above. Use the linspace function to generate the array.
- A variable called **dx600** that contains the Δx(or dx), the width of each cell when we have 600 evenly spaced cells across the L=1.2m geometry.
- A variable called **sixHundredCellLocations** that is an array containing the x-locations of the centers of the 600 cells. Instead of using linspace, use the other array generation method mentioned by your textbook.

Steam Data Overload

In your thermodynamics class, you will get to spend many fun hours looking up data in the steam table. The steam table looks something like the image below:

Pressure Conversions:
1 bar = 0.1 MPa
= 10^2 kPa

Properties of Saturated Water (Liquid–Vapor): Temperature Table

Temp. °C	Press. bar	Specific Volume m^3/kg Sat. Liquid $v_f \times 10^3$	Specific Volume m^3/kg Sat. Vapor v_g	Internal Energy kJ/kg Sat. Liquid u_f	Internal Energy kJ/kg Sat. Vapor u_g	Enthalpy kJ/kg Sat. Liquid h_f	Enthalpy kJ/kg Evap. h_{fg}	Enthalpy kJ/kg Sat. Vapor h_g	Entropy kJ/kg · K Sat. Liquid s_f	Entropy kJ/kg · K Sat. Vapor s_g	Temp. °C
10	0.01228	1.0004	106.379	42.00	2389.2	42.01	2477.7	2519.8	0.1510	8.9008	10
11	0.01312	1.0004	99.857	46.20	2390.5	46.20	2475.4	2521.6	0.1658	8.8765	11
12	0.01402	1.0005	93.784	50.41	2391.9	50.41	2473.0	2523.4	0.1806	8.8524	12
13	0.01497	1.0007	88.124	54.60	2393.3	54.60	2470.7	2525.3	0.1953	8.8285	13
14	0.01598	1.0008	82.848	58.79	2394.7	58.80	2468.3	2527.1	0.2099	8.8048	14
15	0.01705	1.0009	77.926	62.99	2396.1	62.99	2465.9	2528.9	0.2245	8.7814	15
16	0.01818	1.0011	73.333	67.18	2397.4	67.19	2463.6	2530.8	0.2390	8.7582	16
17	0.01938	1.0012	69.044	71.38	2398.8	71.38	2461.2	2532.6	0.2535	8.7351	17
18	0.02064	1.0014	65.038	75.57	2400.2	75.58	2458.8	2534.4	0.2679	8.7123	18
19	0.02198	1.0016	61.293	79.76	2401.6	79.77	2456.5	2536.2	0.2823	8.6897	19
20	0.02339	1.0018	57.791	83.95	2402.9	83.96	2454.1	2538.1	0.2966	8.6672	20
21	0.02487	1.0020	54.514	88.14	2404.3	88.14	2451.8	2539.9	0.3109	8.6450	21
22	0.02645	1.0022	51.447	92.32	2405.7	92.33	2449.4	2541.7	0.3251	8.6229	22
23	0.02810	1.0024	48.574	96.51	2407.0	96.52	2447.0	2543.5	0.3393	8.6011	23
24	0.02985	1.0027	45.883	100.70	2408.4	100.70	2444.7	2545.4	0.3534	8.5794	24
25	0.03169	1.0029	43.360	104.88	2409.8	104.89	2442.3	2547.2	0.3674	8.5580	25
26	0.03363	1.0032	40.994	109.06	2411.1	109.07	2439.9	2549.0	0.3814	8.5367	26
27	0.03567	1.0035	38.774	113.25	2412.5	113.25	2437.6	2550.8	0.3954	8.5156	27
28	0.03782	1.0037	36.690	117.42	2413.9	117.43	2435.2	2552.6	0.4093	8.4946	28
29	0.04008	1.0040	34.733	121.60	2415.2	121.61	2432.8	2554.5	0.4231	8.4739	29
30	0.04246	1.0043	32.894	125.78	2416.6	125.79	2430.5	2556.3	0.4369	8.4533	30
31	0.04496	1.0046	31.165	129.96	2418.0	129.97	2428.1	2558.1	0.4507	8.4329	31
32	0.04759	1.0050	29.540	134.14	2419.3	134.15	2425.7	2559.9	0.4644	8.4127	32
33	0.05034	1.0053	28.011	138.32	2420.7	138.33	2423.4	2561.7	0.4781	8.3927	33
34	0.05324	1.0056	26.571	142.50	2422.0	142.50	2421.0	2563.5	0.4917	8.3728	34

Properties of the steam, such as the specific volume, enthalpy, and so on, are given as a function of temperature. The provided steam table file, getSteamTable.m is a Matlab function that will provide you with the steam tables. Save getSteamTable.m into your current Matlab folder (or in the same folder as a script you are writing), then call the function like this in your command window (or in your script):

```
[head,data] = getSteamTable()
```

Two new variables were added to your workspace: head, and data. Print out the values of those variables, what do they contain? Thats a lot of data! We need to sort it out. Lets grab some chunks of it that we want to work with, create matrices with the following information:

1. Create a matrix called **hotProperties** that contains data rows 83 and greater, and all columns.
2. Create a matrix called **warmLiquid** that contains data rows 13-24 and columns with pressure, temperature, specific_volume_liquid, internal_energy_liquid, enthalpy_liquid, and entropy_liquid (in that order).
3. Create a matrix that contains the first 5 data rows and only the columns for pressure and temperature. Then transpose that matrix so that your first row contains pressure data, and the second row contains temperature data. The final transposed matrix should be called **transposedProperties**. How many rows and columns should this transposed table have?

Your matrices should only contain data, don't include the column head. Your addressing should be in the format:

```
myTableName = data(<PUT YOUR ADDRESSING IN HERE>);
```

```
Invalid use of operator.
```

If you try to create these matrices Als by hand, it will take you all week. If you use advanced addressing with the ":" operator it will take you much less time.

NOTE: I have no idea if the data returned by that function is correct. I found the data on an untrustworthy website. I wouldn't use it for doing your thermo homework.

Write a MATLAB **script** and submit it online to Matlab Grader that meets the following requirements:

- A variable called **head** that is an array containing the header info from the getSteamTable() function.
- A variable called **data** that is a matrix containing the data from the getSteamTable() function.
- A variable called **hotProperties** that is a matrix containing the data from the steam table as specified above.
- A variable called **warmLiquid** that is a matrix containing the data from the steam table as specified above.
- A variable called **transposedProperties** that is a matrix containing the data from the steam table as specified above.