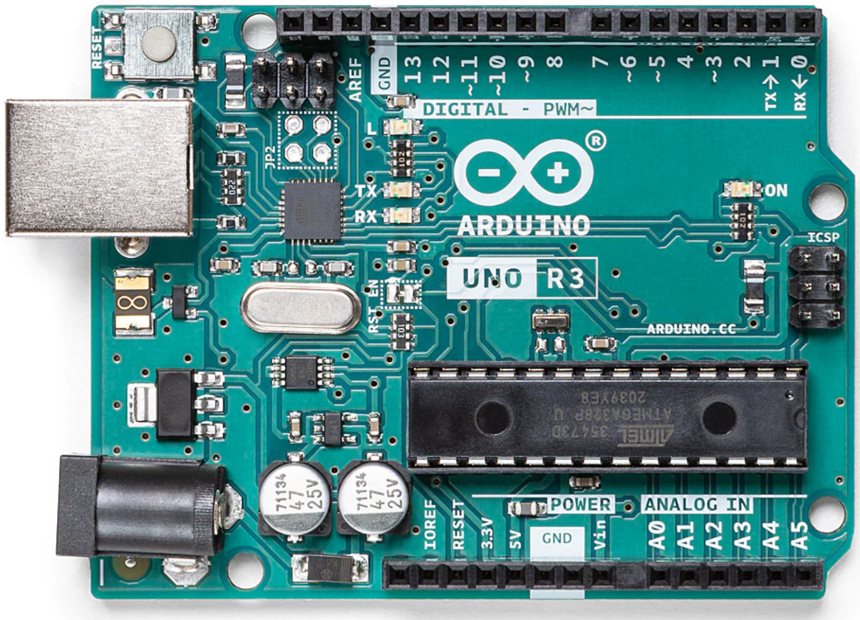
Arduino … Revisited

An overview of what I hope to cover

- A review of the physical characteristics of the Arduino Uno and it's variants
- A review of basic programming considerations including
    - How to use the Arduino IDE
    - How to install library modules required for each project (sketch)
    - Arduino programming syntax and quirks
    - Several examples of simple Arduino sketches.
- How to connect the computer to the Arduino and identify it correctly in the App.
- Looking once more at a few basic sketches like "Blink".
- Using the Arduino Uno to implement the following relevant functions
    - An I2C Liquid Crystal Display that uses only 4 wires!
    - A Real Time Clock
    - Frequency Counter
    - Using a keypad
    - A temperature and humidity sensor
    - An Infrared Remote control
    - An Ultrasound Distance measurement device
    - A Servo to controller to turn knobs for you.
    - An SWR meter
        - An analog one
        - A digital one
    - An RF frequency generator (like the VFO we've already looked at)
    - A 7 segment display
    - A motion sensor
    - A Stepper motor controller (also to turn knobs and aim antennas)
    - An Arduino Keyer
    - An Arduino controlled Switch/Relay to control as much power as you need.

The Arduino Uno

- It comes in two “flavors”
  - One, with a full sized ATMega328P processor plugged into a socket on the board.
  - The second, with a SMD version of the ATMega328 soldered in place on the board.
  - You’ll probably prefer the former but either will work.
  - One more thing: The Arduino Uno is really only a ATMega328P microcontroller with its pins relabeled. It also include a voltage regulator and another microcontroller to handle the USB interface between the ATMega328P and your computer.



- A survey of the board itself
  - First thing you’ll note are two rows of header sockets along two edges of the board. This is where we’ll be plugging wires when we connect this board to the breadboard we’ll be using for various projects. Note also, that they have labels printed on both the board itself and on the header sockets (on the outside).
  - On the left are two means of connection to the board. The upper one is for a USB connection to your computer. That’s how it’s programmed. The second is just to power the board after it has been programmed. A DC power source is connected

    here for that purpose.   It's only necessary if the USB socket is NOT connected to the board.
  - There is, of course, the ATMega328 chip itself. That's where the programs will be stored.
  - There are other chips, resistors and capacitors along with a 16 mHz crystal.
  - There is a reset button next to the USB connection. Press this and the program within the microcontroller (the ATMega328) restarts.
  - 4 LEDs. They are tiny! There is one near the label "13" on the upper part of the board. Thats connected to that pin. Our "Blink" example will be flashing that LED.
  - Also two header pins, one at the top, and one at the right end, that we won't be using.
- Connecting the board to your computer.
  - First, download and install the Arduino IDE app on your computer. Get the software from: https://www.arduino.cc/en/software Use the latest version, 2.2.1 as of this writing, because it has a few handy features not found on the older, 1,8.19 version.
  - Now, using the USB cable that come with the board, attach it to the Arduino (or the Uno R3 knockoff) and the other end to a USB port on your computer. No drivers should be required.
  - Start the app. Then do the following two things:
    - Under "Tools"/"Board", select "Arduino Uno"
    - Under "Tools"/"Port", select the port that has (Arduino Uno) after it.
    - You're ready to go!

MyBlink.

- OK, we're ready for your first programming challenge. Instead of loading a preexisting Sketch, you're going to create your own.
- Follow the these steps:
  - Start, by selecting "New Sketch". This sketch won't have a name yet, that's for later …
  - What you'll see is the following:

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

}

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

}
```

- You'll note that there are only two sections in this Sketch. One called "void setup()" and the other called "void loop()"
- Position your cursor in "setup()" section between the braces. Just under the two slash marks //. Use the Tab key to put the cursor directly under the 1st slash mark. (This only make the Sketch look better but won't change how it functions.) Now type the following: (Careful: Type EXACTLY what you see below!)

```
  pinMode(LED_BUILTIN, OUTPUT);
```

- This simply tells the program to set Pin 13 where the builtin LED is, to be an Output pin. The words (LED_BUILTIN) are an alias for the number 13. In fact you can put the number 13 there if you like).
- Now, position your cursor under the two hashmarks in the "loop()" section and type the following:

```
  digitalWrite(LED_BUILTIN, HIGH);
  delay(1000);
  digitalWrite(LED_BUILTIN, LOW);
  delay(1000);
```

- The words "digitalWrite" instruct the Sketch to set the Pin specified, in this case pin 13 (LED_BUILTIN) to the HIGH state. I.E. To + 5 volts.
- "delay" with the number in parenthesis tells the Sketch to wait that number of Milliseconds. 1000 ms = 1 second.
- Next we use digitalWrite again to set Pin 13 to the LOW state. I.E. To 0 volts.
- Then we instruct the sketch to wait another second.
- Since this is in the "loop()" section of the Sketch, it will repeat indefinitely.

- Now compile the sketch and upload it to the Arduino via the -> arrow at the top left of the window. Watch the tiny LED next to Pin 13 blink!!!

- Last thing to do before we move on is to Save this to a name that make sense.
  - At the top of the Computer's Screen, next to where it says "Arduino IDE", click on "File" and then select "Save As" and hit <Return>.
  - OK, give it a name like "MyBlink" and make sure it's saved in the "Arduino" folder by selecting that folder in the dropdown called "where".
- Now let's change something and do this again.
  - In the loop() section, change the second delay number from 1000 to 100.
  - Upload your new program.
  - Actually, you can change either number and watch what happens to the timing of the LED blink.
- One last comment before we move on. When you change any "script" that is either built in or one you write yourself, any changes you make are always saved to that file when you Upload it to the Arduino Uno. What that means is the next time you open "MyBlink", it will be as you've changed it, not as it originally was.

Arduino C

- Now a digression. Let me talk a bit about the C programming language and the specific additions and modifications to that language the Arduino IDE implements.
- For starters, Arduino C adds some functions to standard C. These are functions like:
  - digitalRead
  - delay
  - pinMode
  - Etc. …
- These additions and most standard C functions can be found at the arduino.cc website under "Documentation". Here's the link: https://www.arduino.cc/reference/en/
- Now, back to standard C.
  - Capitalization is IMPORTANT. That means that, in standard C, all lines of code must be typed exactly as shown in examples etc. I.E. "digitalWrite" is not the same thing as "digitalwrite" or "DIGITALWRITE".
  - All program lines must end with a semicolon ;
  - When several lines of code are part of the same execution sequence - like when they follow an "if" conditional statement, they are enclosed in braces {}. Note: You've seen these braces already in the setup() section and the loop() section.
- Standard C has only one required section. It is called "main()".
- Arduino C has two required sections: "setup()" and "loop()". It does not have a section called "main()"

- In Standard C (as well as Arduino C), all functions, including the “setup()” and “loop()” and “main()” function have three parts to their “declaration”
  - The first part is whether or not that function returns anything. If it does not (i.e. it just does something like make a LED blink) then that section is called “void”. It just means it returns *nothing.*
  - The second part is the function *name*. That what you type when you want to use this function in your Sketch or program. By convention, this always starts with a lower case letter.
  - The third part of the function is what is enclosed in the parenthesis (). That part is what information is sent to the function to do something with. If you’re sending nothing, that also is either empty or says *void*.
  - Here’s an example:

    ```
    val =  digitalRead(inPin);
    ```

  - In this example, function “digitalRead()” is sent the value of the pin to read, “inPin” (this needs to have been previously defined in the program), and assigns value it reads from that pin to the variable “val” (also previously defined in the program).
- In Standard C, and also in Arduino C, functions and variables must be defined before they are referenced and used.

MyBlink2

- Here is a revision of the “MyBlink” sketch using variables and definitions:

```
// define section
#define OUT_PIN 13

// variable definition section
int delayTime = 1000;

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

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

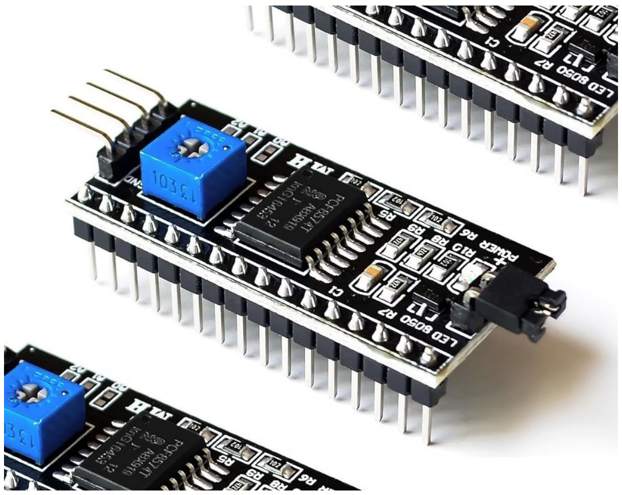
```
  digitalWrite(OUT_PIN, HIGH);  // turn the LED on (HIGH is
the voltage level)
  delay(delayTime);                      // wait for a
second
  digitalWrite(OUT_PIN, LOW);   // turn the LED off by
making the voltage LOW
  delay(delayTime);                      // wait for a
second

}
```

- In the above example, the line that starts with `#define` simply tells the Sketch that the wherever the word OUT_PIN is found, substitute the number 13.
- In the next line of code, the line that begins with `int` defines the variable `delayTime` as an integer (int) and assigns the value of 1000 to it.
- More about variables, definitions, and functions later. This is enough for now.

A better way to connect an LCD Display

- Remember back when we were trying to make a Arduino based VFO? Yeah, that project. Well, I did make a demo for you but I don't believe anyone actually made the VFO we talked about. One of the problems was dealing with all the wires we needed to connect the Arduino to the LCD. It required 8 wires and also required a variable resistor that was connected to the liquid crystal contrast pin. What a mess!
- Well, I always knew it was possible to make this simpler but, sadly, I didn't adequately understand just how simple this new display might be. It utilizes an I2C "translator" that fits onto the back of the display and does all the work/connections for you.



- Here's the link: https://www.amazon.com/dp/B0BVFTXN7Y?psc=1&ref=ppx_yo2ov_dt_b_product_details Only $2.33 each!
- The four(4) pins on the left connect to your Arduino / power supply.
- The pins on the bottom attach to the LCD display.
- Amazon also has these I2C interface connector preassembled with the LCD at $2.78 each! https://www.amazon.com/dp/B07T8ZG5D1?psc=1&ref=ppx_yo2ov_dt_b_product_details
- (I'll be bringing the latter to class for sale :-)
- To make this work, all you have to do is attach GND pin to the Arduino GND (there are 3 of them).
- Attach the VCC pin to the 5V pin on the Arduino (only one of these)
- Attach the SDA pin to the Arduino SDA
- Attach the SCL pin to the Arduino SCL
- You're done!!! … well, almost.

- Now for the code.
  • First, you have to install the "library" header file, "LiquidCrystal_I2C". Go to the library tab on the left side of the Arduino App and type in the above.
  • Scroll down and find the file "Liquid Crystal I2C" bv Frank de Bradander. Install that one.
  • Once you've installed that, here is the code you type onto your new Sketch:

```
// includes
#include <LiquidCrystal_I2C.h>  // don't forget the .h part

// class initializations.
LiquidCrystal_I2C lcd(0x27,16,2); //(I2C address, columns, rows)

// display function follows ...
void displayMessage(char *msg1, char *msg2) {
  // this function just displays two messages.
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print(msg1);
  lcd.setCursor(0,1);
  lcd.print(msg2);
}
```

```
// setup function - runs once.
void setup() {
  lcd.init();
  lcd.backlight();
  lcd.clear(); // this is technically unnecessary
}

void loop() {
  displayMessage("Hello World", "Dixon K3WNS");
  while(1);
}
```

- This program has 5 parts:
  - First, the #include part is where you tell the Sketch to fetch the header file you just installed: <LiquidCrystal_I2C> (note here: Anything that starts in the 1st column with a hash mark #, does NOT end in a semicolon ;
  - The second is the “class initialization” part. Here you name a variable, “lcd”, to be of the type “LiquidCrystal_I2C”. You also specify three things: The I2C address, the number of Columns, and the number of Rows. (This DOES end in a semicolon ;)
  - Next, I’ve created a new function called “displayMessage”. It doesn’t return anything so it is of the type “void”, and it requires two character strings “char *msg1” and “char *msg2”. Note the asterisk * before the msg1 and msg2. That’s important and it tells the function where these two character strings are located in memory.
    - The word “char”, above, identifies the variable *msg1 & *msg2 as strings.
    - There is also a comma , between the two strings.
    - Then there are a set of braces { … } that enclose the code of the function.
    - The code is straightforward in that it tells the Script to print msg1 on line 1 (row 0) of the LCD display and print msg2 on line 2 (row1) of the display.
  - Next part is the setup() function. This does three things:
    - Initialize the lcd.
    - Turn on the backlight (I have often forgotten to do this and been left in the dark)
    - Clear the display
  - Finally, there is the loop() part. It has only two lines:
    - Display two message, in double quotes, using the function I just created
    - Go into an infinite loop … i.e. do nothing else.
- We’ll be using this new display for many, if not all, of the succeeding projects.