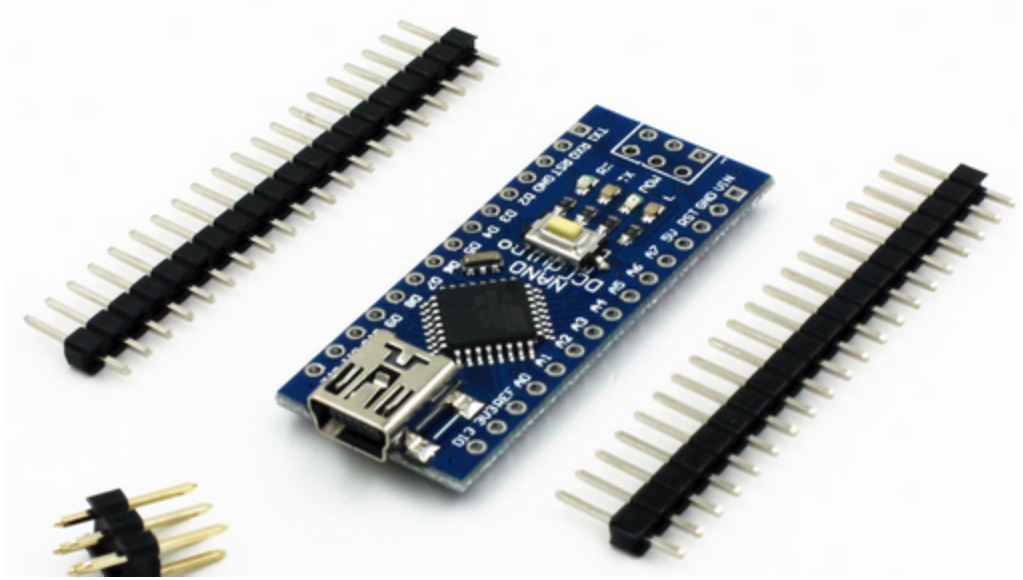
...RRRduino...

Random Things to Lite Up

(for absolute beginners, like me)



by Speed Muller

v0.01

If you don't care, don't like electronics and don't want to be bothered, write this down:

www.TxNamib.com

and

www.RRRduino.com

(if the latter fails, use blog.rrrduino.com)

and then go ahead, take that nap!

What is an Arduino?

Quoted: (http://www.circuitstoday.com/story-and-history-of-development-of-arduino)

The new prototype board, the Arduino, created by Massimo Banzi and other founders, is a **low cost microcontroller board** that allows even a **novice** to do great things in electronics. An Arduino can be connected to all kinds of **lights**, **motors**, **sensors** and **other devices**; an easy-to-learn programming language can be used to program how the new creation behaves. Using the Arduino, you can build an **interactive display** or a **mobile robot** or anything that you can imagine.

You can purchase an Arduino board for **just about US $30** or **build your own** board from scratch. Consequently, Arduino has become the most powerful open source hardware movement of its time.

Today, there are Arduino-based **LED cubes**, **Twitter displays**, **DNA analysis kits**, **breathalyzers** and so much more. There are Arduino parties and Arduino clubs. As a feather to its crown, Google has recently released an Arduino-based development kit for its Android Smartphone!

FOR US? **Flickering lights, fire trucks, ambulances, police cars, crossing gates (even bringing the gates down WITH the bell ringing), signals, semaphores, turnout control, airfield lights, animation with servos, steppers and DC motors. Sensors, counting and reporting axles, and randomly nagging about a hot wheel! Also LCC, BlueTooth, WiFi, CAN Bus, transmitting data across your layout.**

How did it come about?

Quoted (http://www.circuitstoday.com/story-and-history-of-development-of-arduino):
It was in the Interactive Design Institute that a **hardware thesis** was contributed for a wiring design by a Colombian student named **Hernando Barragan**. The title of the thesis was "Arduino–La rivoluzione dell'open hardware" ("Arduino – The Revolution of Open Hardware"). Yes, it sounded a little different from the usual thesis but none would have imagined that it would carve a niche in the field of electronics. A team of **five developers** worked on this thesis and when the new wiring platform was complete, they worked to make it much lighter, less expensive, and available to the open source community.

...the Story in more Detail…

As mentioned earlier, it all started in **Ivrea, Italy**. To begin with, let's have a look at how the name Arduino, which sounds quite strange for an electronic device, was chosen. This beautiful town of Ivrea, situated in Northern Italy, is quite famous for its underdog kings. In the year 1002 AD, **King Arduin** (you got it right!) ruled the country; two years later, he was dethroned by King Henry II of Germany. In the memoir of this King Arduin, there is this 'Bar Di Re Arduino', a **pub on the cobble stoned street** in the town. Well, this place is where a new era in electronics had its roots!

This bar was frequently visited by **Massimo Banzi**, one of the founders of Arduino, who taught at Ivrea. He was the one who gave the name Arduino to this low-cost microcontroller board in honor of the place!
Before getting into how the Arduino was developed and used, let's know who the core members of the Arduino developer team are: **Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino, and David Mellis**.

The First Prototype Board

Well, Banzi succeeded in creating the first prototype board in the year **2005**; it was a simple design and at that time, it wasn't called Arduino. *Of course, by now, you would know how he had coined the name later that year.*

Open Source Model – A Big Decision

Banzi and his collaborators strongly believed in **open-source software.** As the purpose was to develop a quick and easily accessible platform, they thought it would be better to open up the project to as many people as possible instead of keeping it closed. Another crucial factor that contributed to that big decision was that after operating for nearly five years, IDII had no more funds left and was in fact going to shut its doors. All the faculty members feared that their projects might not survive or would be embezzled. It was at this crucial point of time that Banzi decided to go ahead and make it open source!

How Banzi and team managed to create Arduino and make it available for public

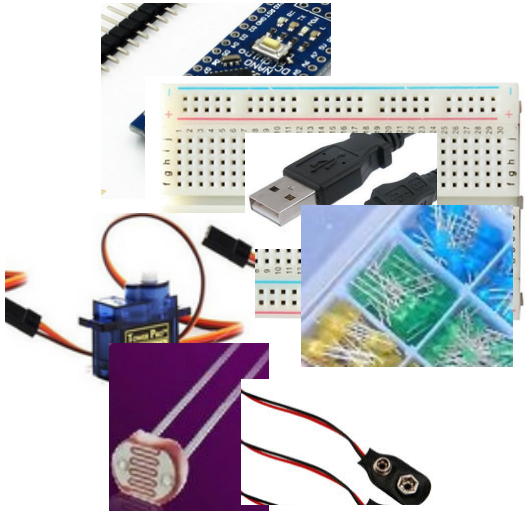
Pretty obviously, the open source model had always been used to fuel innovation for software and never **hardware**. If they had to make it work, they had to find a suitable licensing solution that could apply to the board. After a little investigation, Banzi and team looked at the whole thing from a different angle and decided to use a license from **Creative Commons**, a nonprofit group whose agreements were normally used for cultural works like writing and music. *According to Banzi, hardware is a piece of culture that must be shared with other people!*

Well, the next step was to make the board. The group decided to fix a specific, student-friendly price of **$30** as their **goal**. Banzi felt that the Arduino should be affordable for all students. However, they also wanted to make it really quirky, something that would stand out and look cool as well. While other boards were green, they wanted to make theirs **blue**. While a few manufacturers saved on input and output pins, they added a lot to their board. Quite weirdly, they added a little **map of Italy** on the back of the Arduino board!

Division 3 opted for a Make and Take Clinic!

We bring (or what your $$$ gets you):

1 x **Arduino Nano** (with a USB connector, read: "no separate programmer")
1 x USB to **Mini USB Cable**
1 x **9V** battery **cable**
1 x **9g Micro Servo** Motor (Grade Crossing, Semaphore?)
2 x 3mm **RED** LEDs (Crossing Gates?)
2 x 3mm **YELLOW** LEDs
2 x 3mm **GREEN** LEDs (Signals?)
2 x 3mm BLUE LEDs (Ambulance? Firetruck?)
2 x 3mm WHITE LEDs – yes, that was white, :)
10 x 1k 1% resistors (Why? Another clinic!)
1 x Push Button
1 x Light sensitive Photo Resistors
1 x 400 contact breadboard
and some wires to connect some or all of these together!



You bring:

a Laptop, or a friend with his/her laptop.
a 9V battery (and maybe a solder iron too)
an IDEA! Yes, you tell ahead of time what you want your Arduino to do going home, and we get the code 99% there.

– Usually a 2 PART CLINIC –

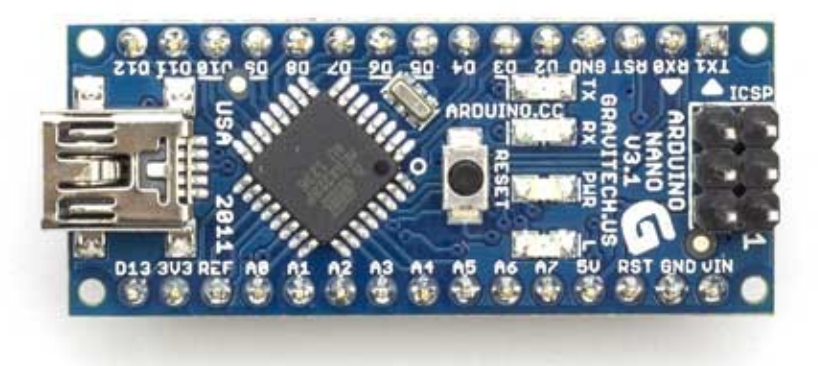
1) What+Solder+Laptop: The history, what it is, and what you can do with it.
Then we solder (a half clinic on soldering too) the pins and wires on
And install the software on your laptop or computer.
Your Arduino will have a blinking LED at the end of this
2) Coding 101: We get into the software and get your IDEA on your board!

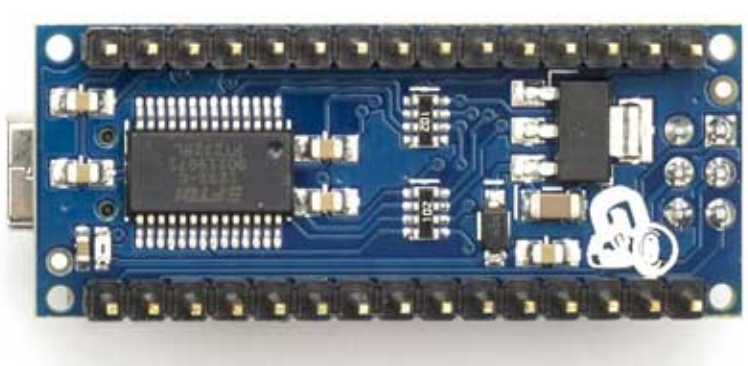
*(yes, we are going to use the term "click" a few times and Larry will follow along on **my** laptop!)*

Where to Start...

Buy one ... and plug it in!

[PC → USB Cable → Nano]





See a **solid** red light and a **blinking** red* light?

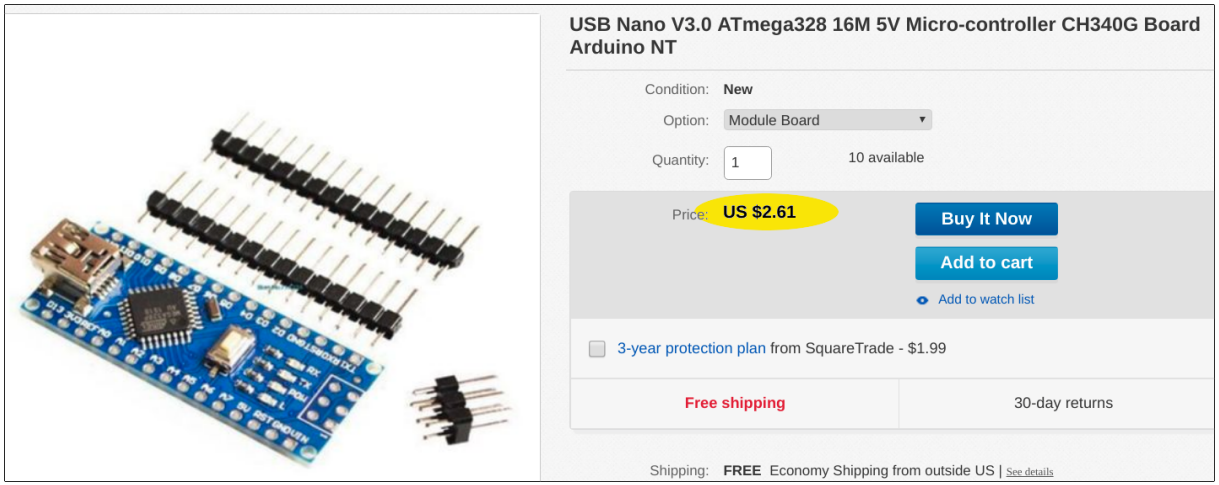
Where did we get it?

Search Ebay for “**Arduino Nano USB”** … I buy the Hong Kong ones, and it feels like they ship faster.

Version 3.0 is new, 5V is good too.

The CH340G USB/Serial chip requires a device driver, but Google can help you disable the device driver signing in Windows 8+, just one reboot required.

(Linux guys, sorry for that time waster, you were good without installing anything)

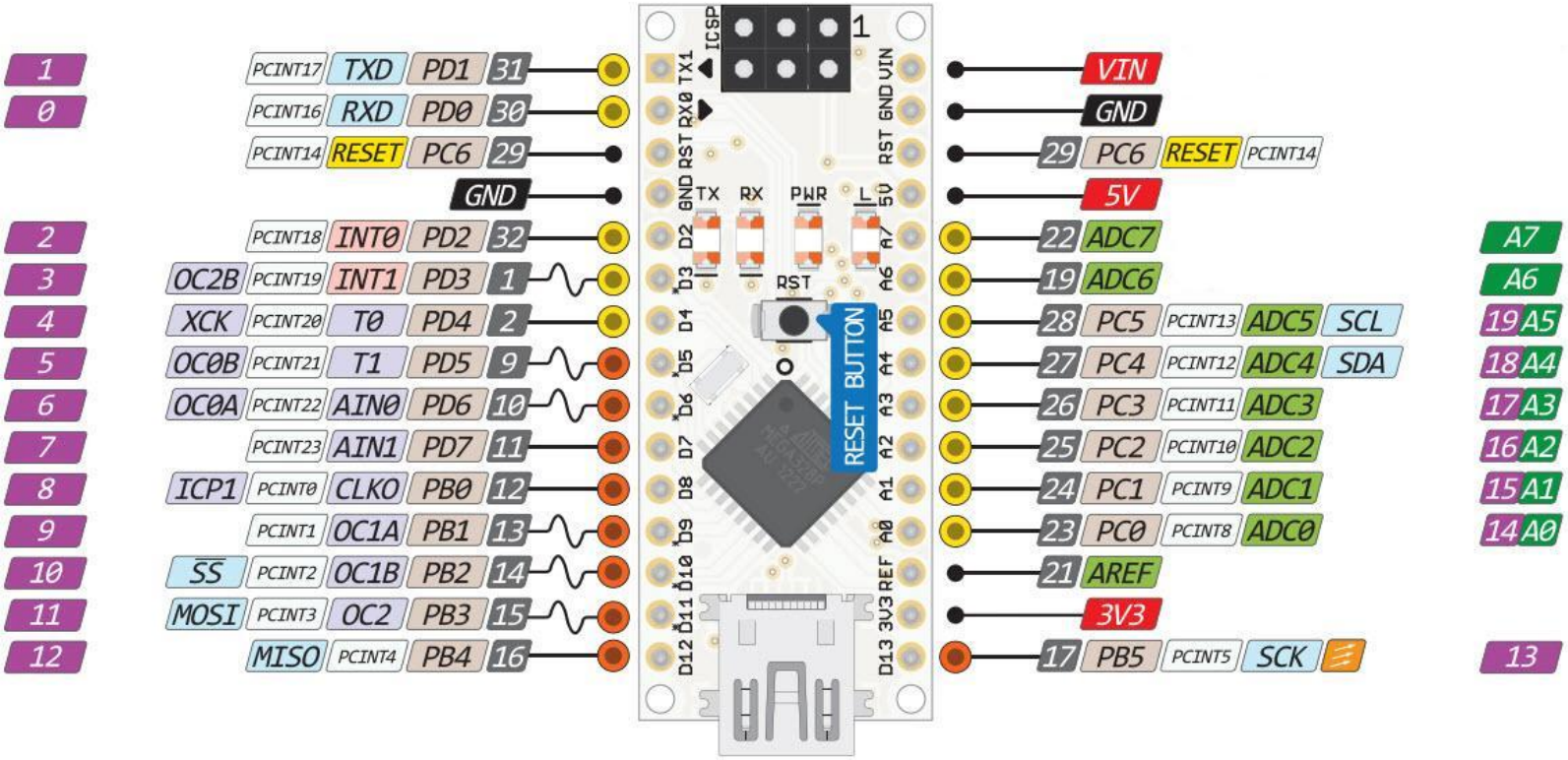


Of course, Adafruit, Sparkfun and Amazon would help you too.

Even MicroCenter has them in stock.

Here is what we just plugged in:

(Remember, Open Source, Open Hardware)

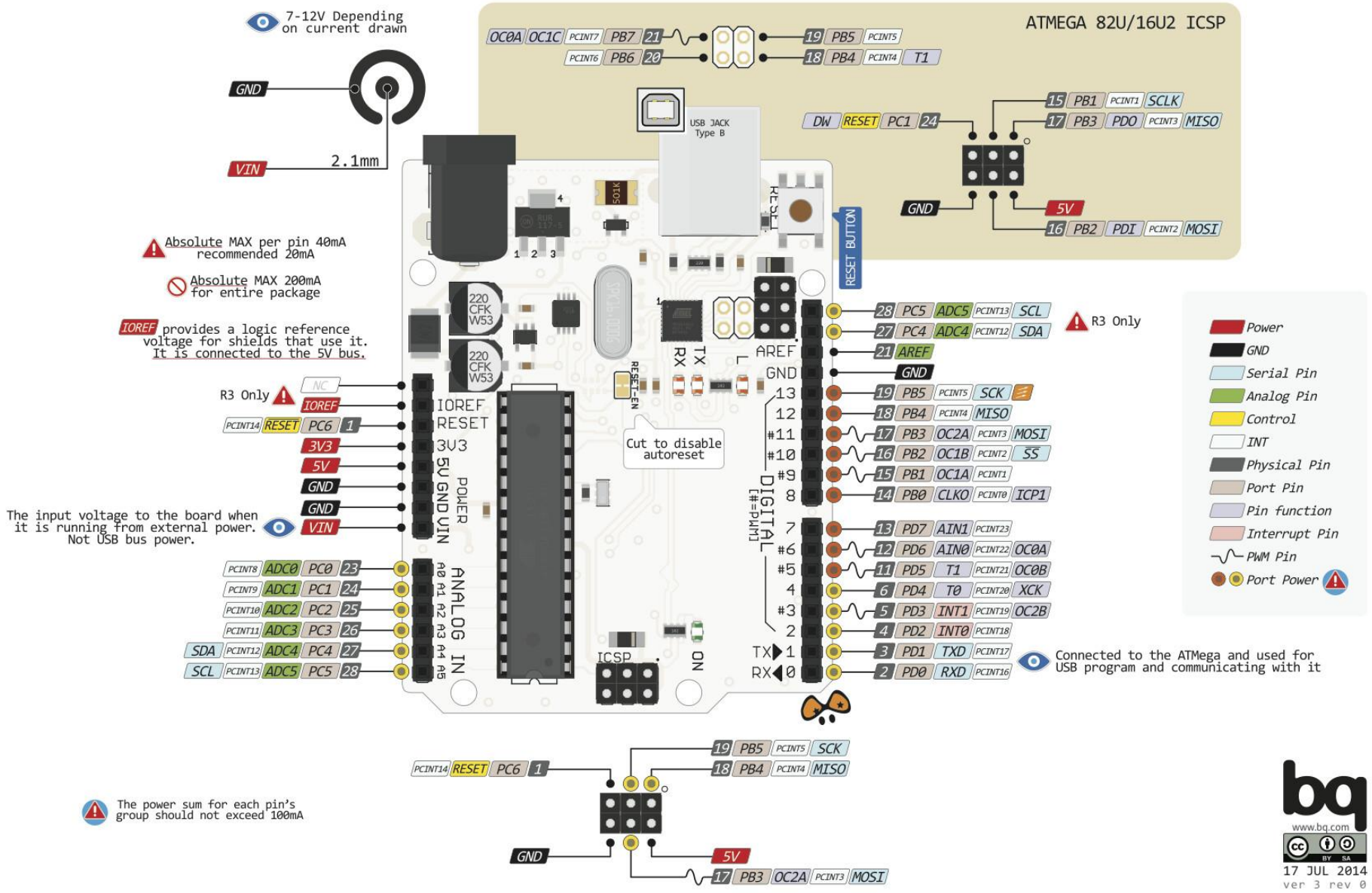


What you care about, is the purple and green numbers on the far left and right, since those are the numbers the software needs.

Green indicated pins needed for Analog **inputs**, but all the pins can do digital things.

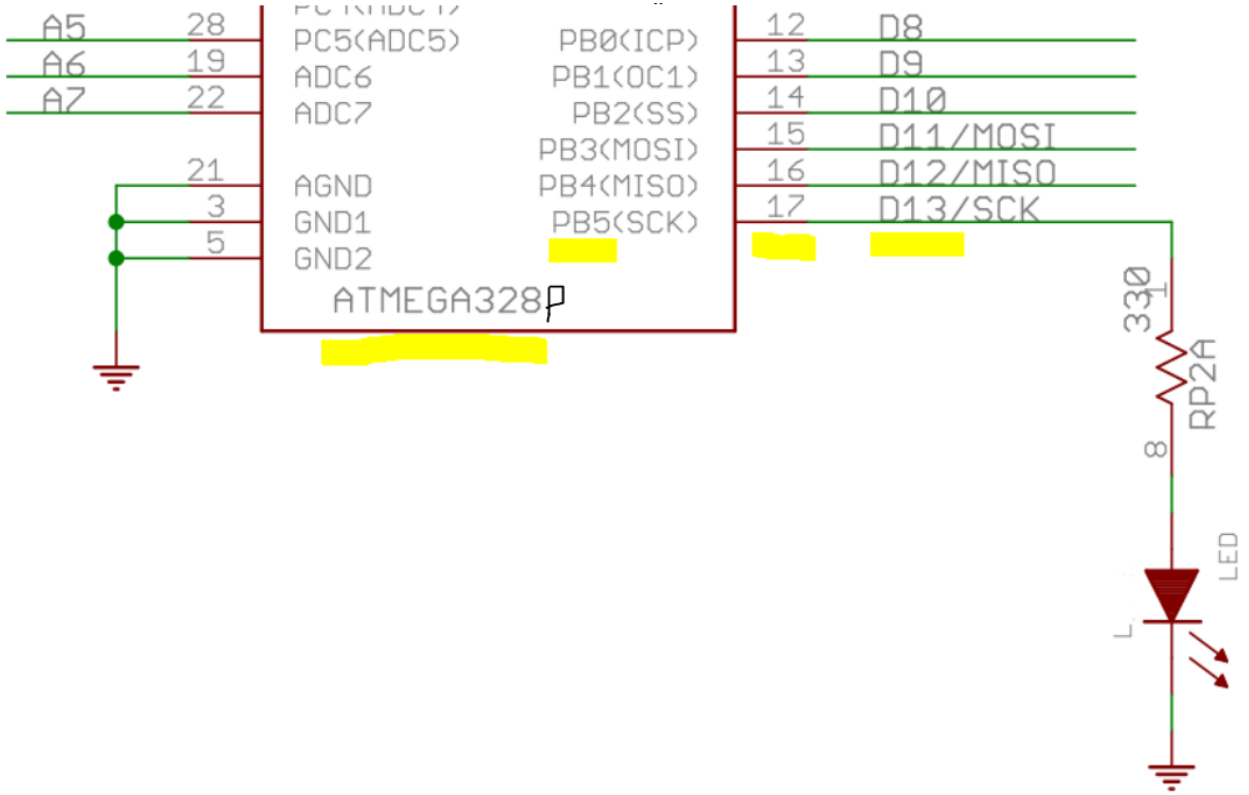
The other thing to note is the lines on D3, D5, D6, D9, D10 and D11 that have a wiggle, those can be **PWM** outputs...in simple terms, they could look like an Analog **output** with a resistor and capacitor as filter.

UNO PINOUT



www.bq.com
17 JUL 2014
ver 3 rev 0

Pins Ports and Numbers:



- The Atmel ATMEGA328P-PU chip has pin 17 (labeled Port.B bit 5 and Serial Clock) connected to the Arduino pin D13 … and hooked to an LED on the Arduino board. So if you toggle D13, the LED toggles on and off.

OK, Back up for 1 minute:

What is **Arduino** again?

- It certainly has Hardware

- yes we know (and you get to see the schematic and board layout, like in the real files containing it, I mean)

- it has shields and plug-ins with interfaces to almost everything else in the world, and you can add the missing ones!

- It has a Software development platform

- IDE, C/C++ language, library extensions, compiler, upload tool and serial monitor, oops Bootloader too.

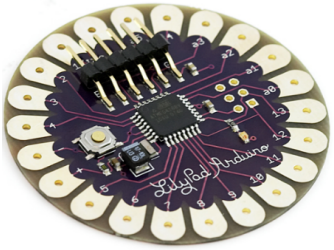
- It has a **following**, even today's kids know about it

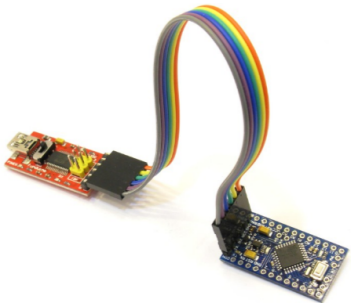
- It is world wide! It has websites, blogs, examples, even facebook and twitter, don't you?

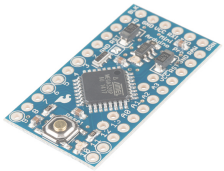
...It is a whole platform with tools...

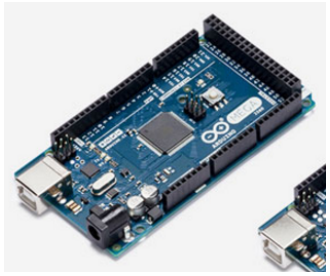
Hardware?

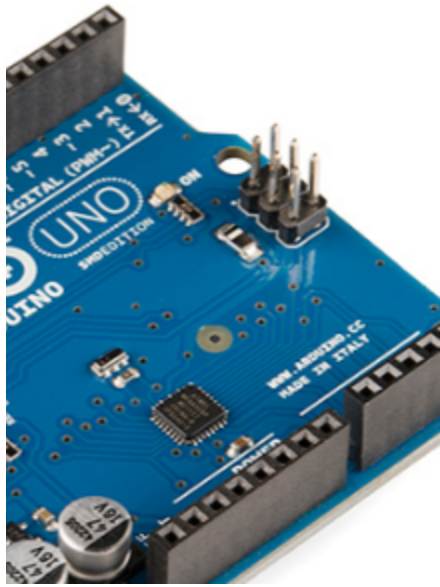
UNO was first: Atmel ATMEGA328P in a PU package, with an “NFL” size USB plug:







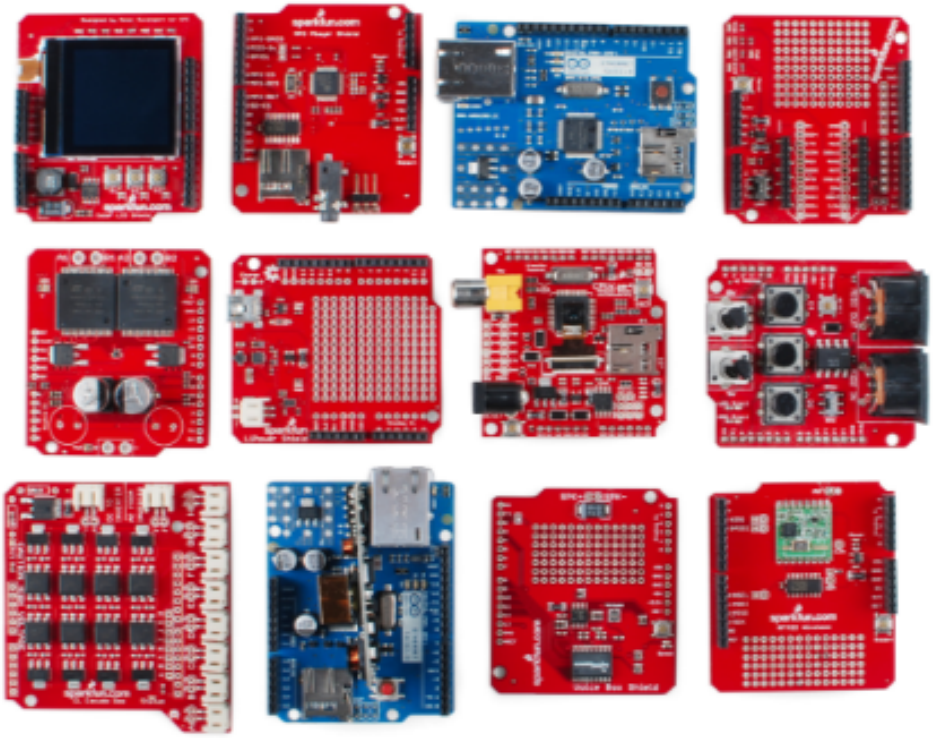




Then an SMD UNO version was made.

Wait, my Nano has that chip too!!!
...and a mini USB connector
...which the Micro does not have.

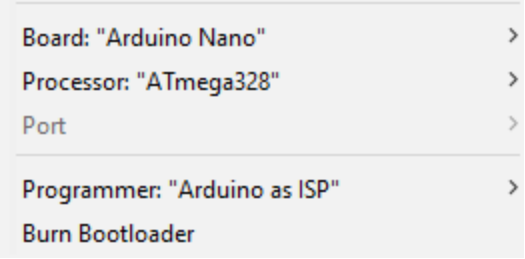
A short word on Shields



- That is how Arduinos connect to the world, if it is not already present
  - Simple “breadboard” Shield, so you could solder a wire to something
  - Motor Shield: DC, Stepper, Servo Motor
  - LCD Shield
  - Audio Amplifier
  - SD Card reader
  - Ethernet or WiFi Shield
  - CAN Bus
- Of course, different Shields needed for different Arduinos, Nano vs Uno
  - Plan ahead!
  - Or make your own, Fritzing and Eagle is free

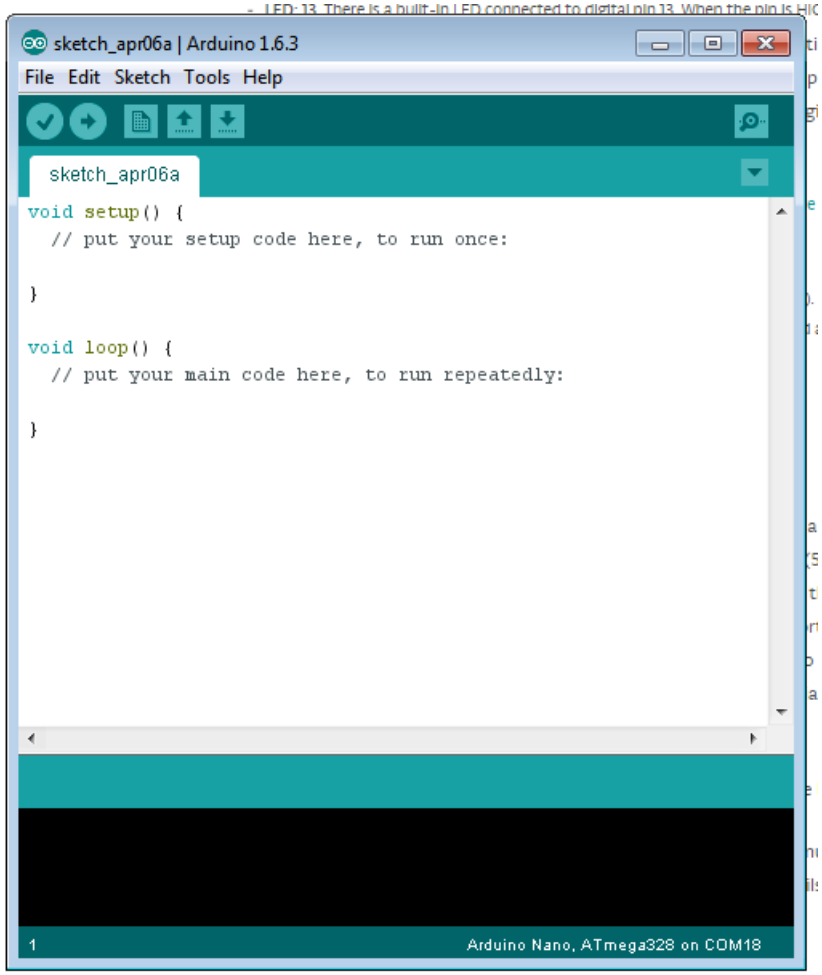
Programmers

- Bootloader
- FTDI most common, CH340G needs a device driver. Read about FTDI-gate.
- An Arduino can become a programmer to program another
- ATMEGA328P-xx vs no’P’-xx
  - Low power version vs lower cost version
  - Difference in Social Security #



And then you said Software!

- **IDE** (integrated development environment), free download
  - Windows, Mac and Linux (last one sudo apt-get arduino, Raspberry Pi too)
  - 1.8.19 is newest during this Temple Clinic in May 2022
  - Also a version 2.0 RC (Release Candidate)
- **Fancy editor** colors and calling things in the background!
- **Select** your board, processor, programmer and serial port from the Tools menu
- Type **Code**, **verify** (compile) and then **upload**!
- Tip: the upload button does it all, wait, it does not type your code, ;)
- When you save a file, it will be under the Files->Skecthbook menu
- **Serial Port Monitor** (Ctrl+Shift+M) and now also a **Serial Plotter** (Ctrl+Shift+L)



Programming

icons for: VERIFY, UPLOAD, NEW, OPEN, SAVE, SERIAL MONITOR.

First example on the Uno

Uno plugged in?

Start the software 

Tools → Board → Arduino Uno // only once

Tools → Port → COMx (or /dev/ttyUSB0) // only once

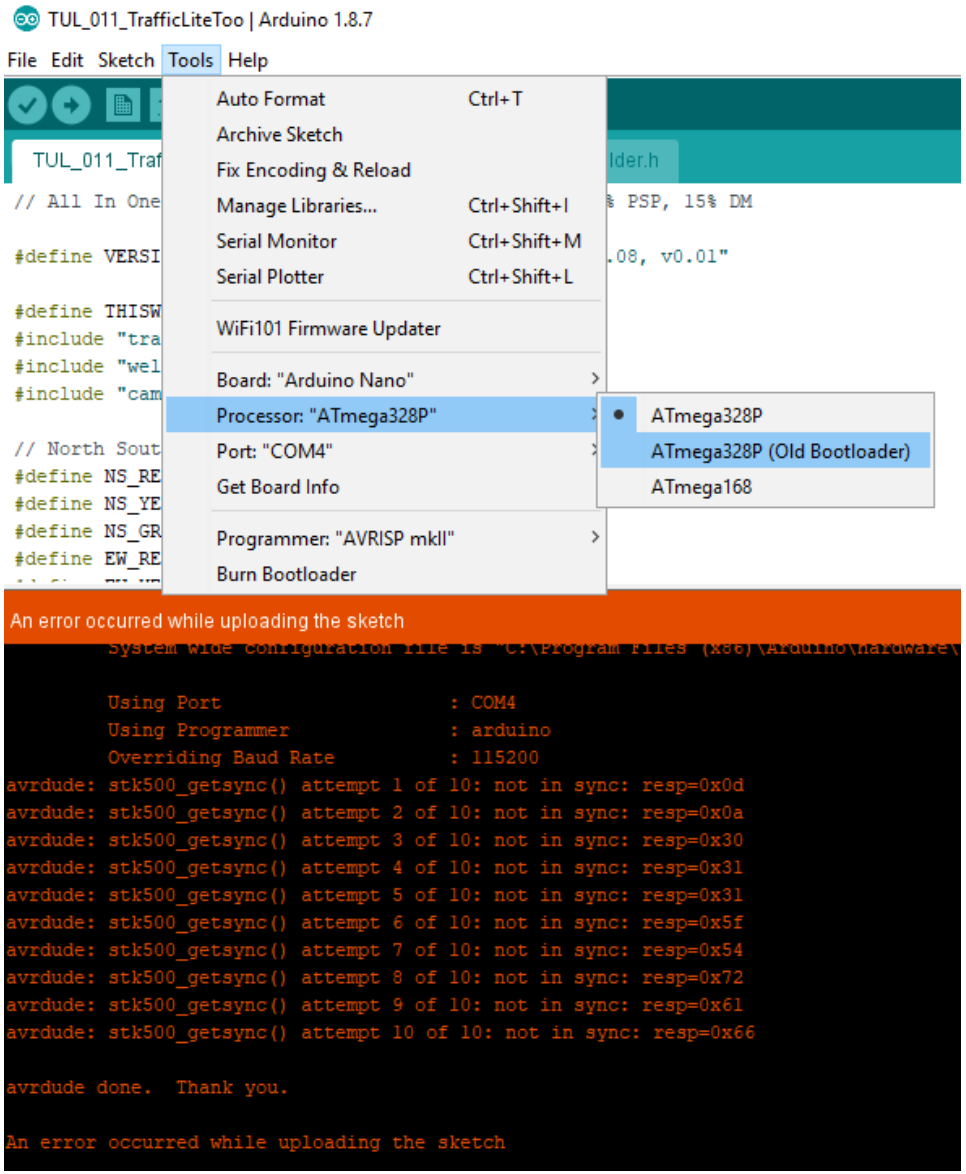
File → Examples → 01.Basics → **Blink**

```
void setup( ) {
  pinMode( 13, OUTPUT );
} // setup( )

void loop( ) {
  digitalWrite( 13, HIGH );
  delay( 750 );
  digitalWrite( 13, LOW );
  delay( 250 );
} // loop( )
```

Upload Error?

Try the correct **Board**, **Port** and Other Bootloader in **Processor**



Useful Arduino C/C++ commands for a beginner

pinMode(pin, IN/OUTPUT);
digitalWrite(pin, HIGH/LOW);
val_Digital_True_False = **digitalRead**(pin);
analogWrite(pwm_Pin, value);
val_Analog_0_To_1023 = **analogRead**(analog_Pin);
delay(millisecond);
val_Long = **millis**( );
if(x > 5) ;
while(j < 10) { j++; }
for(j = 0; j < 10; j++) { ; }
val = **map**(value, fromLo, fromHi, toLo, toHi);
random(min, max - 1);

Wiring? Processing? Just think it is C or C++ and move on...

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

Language Reference

Arduino programs can be divided in three main parts: *structure*, *values* (variables and constants), and *functions*.

Structure

- setup()
- loop()

Control Structures

- if
- if...else
- for
- switch case
- while
- do... while
- break
- continue
- return

Variables

Constants

- HIGH | LOW
- INPUT | OUTPUT | INPUT_PULLUP
- LED_BUILTIN
- true | false
- integer constants
- floating point constants

Data Types

- void
- boolean
- char
- unsigned char

Functions

Digital I/O

- pinMode()
- digitalWrite()
- digitalRead()

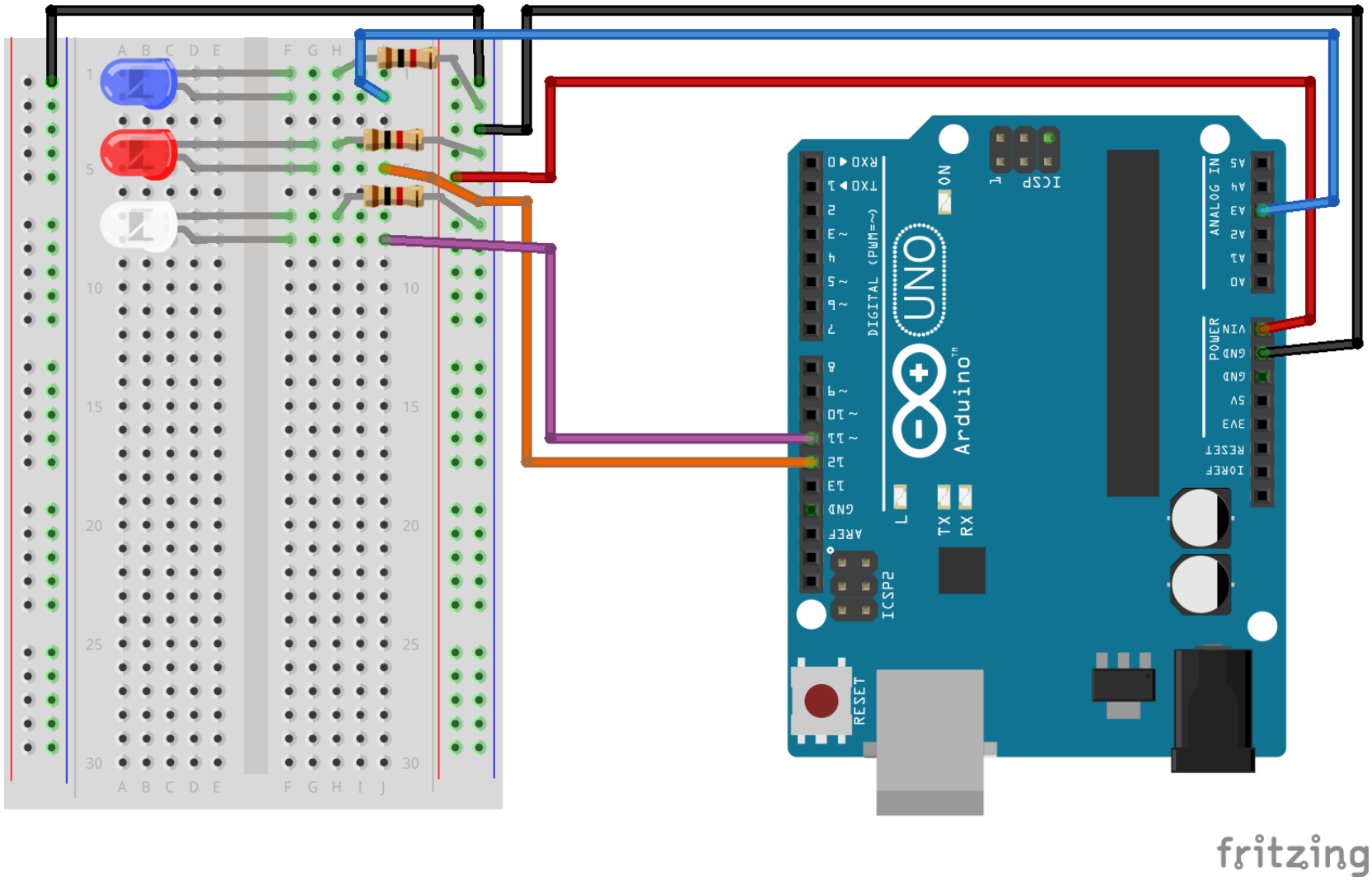
Analog I/O

- analogReference()
- analogRead()
- analogWrite() - *PWM*

Due & Zero only

- analogReadResolution()
- analogWriteResolution()

Let’s go!



- Open **001_BlinkSlow_with_Serial.ino**

```
// Comments:  \__________/----------\__________/----------\__

#define LEDPIN      13

// Comments
void setup( ) {
  Serial.begin( 115200 );
  Serial.println( );
  Serial.println( VERSION_STR );

  pinMode( LEDPIN, OUTPUT );
  digitalWrite( LEDPIN, HIGH );
} // setup( )

// Comments
void loop( ) {
  delay( 250 );
  digitalWrite( LEDPIN, HIGH );

  delay( 250 );
  digitalWrite( LEDPIN, LOW );
} // loop( )
```

Serial Port Monitor, Ctrl+Shift+m

Press Ctrl-Shift-m (this opens the Serial Monitor from the IDE)

- And this is how you find out 3 years later what is on the ATMEGA328 chip!

Tip: If your boss does not allow you to install TeraTerm, RealTerm, putty or Serial Port Monitor, install the Arduino IDE! Who can tell which Windows version lost Hyper Terminal?

IDE #1, Sketch

- A sketch is the .ino file in a folder with the same name.

- No spaces in filename

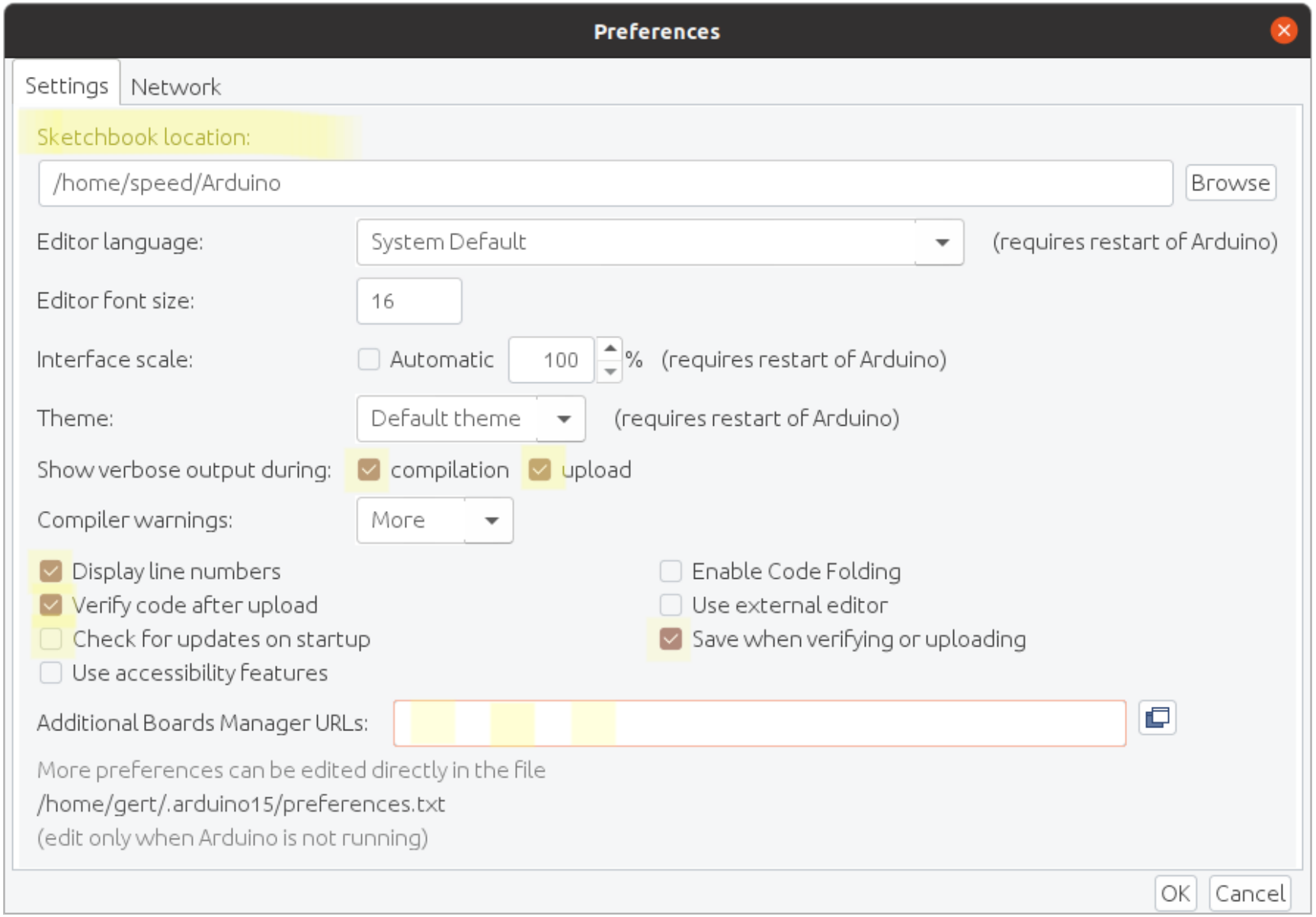
- No leading numbers or funny characters

- main.cpp calls
  setup( ) once, and
  loop( ) from a repeating for(;;) { … }

- It also checks for incoming serial data, to switch to the bootloader (already programmed in) to upload the next program when you click Upload

IDE #2, Setup

File -> Preferences (Ctrl+Comma), OK when done



Next…

- Open **002_BlinkFaster.ino**

```
/*
 * Blink by setting the ONTIME and the OFFTIME separately
 */

// \__________________/----  ,  \__________________/----  ,  \__
// x                  x      ,  x                  x      ,  x
```

- Now **see 003_BeaconBlink.ino**

```
/*
 * A beacon can be implemented with an ON time and an OFF time
 */

// \__________________/----  ,  \__________________/----  ,  \____
// x                  x      ,  x                  x      ,  x
```

- **004_TwoBeaconBlink.ino**

More than 1 beacon on the Arduino now!

Pay attention to the red ‘x’s shown here:

```
/*
 * Making two beacons work! Need external LEDs on pins 11 and 12
 */

// \_____________________/----  ,  \_____________________/----  ,  \____
// --\_____________________/--  ,  --\_____________________/--  ,  --\__
// x x                   x x    ,  x x                   x x    ,  x x

#define LEDPIN1          12
#define LEDPIN2          11
#define TIME1            50
#define TIME2          1850
#define TIME3            50
#define TIME4            50
```

- **005_TwoBeaconToggle.ino**

Instead of writing HIGH or LOW, we can read and then write the opposite, i.e. toggle( )

```
void loop( ) {
  digitalWrite( LEDPIN1, LOW );
  delay( TIME1 );
  digitalWrite( LEDPIN2, LOW );
  delay( TIME2 );
  digitalWrite( LEDPIN1, HIGH );
  delay( TIME3 );
  digitalWrite( LEDPIN2, HIGH );
  delay( TIME4 );
} // loop( )
```

```
void loop( ) {
  toggle( LEDPIN1 );
  delay( TIME1 );
  toggle( LEDPIN2 );
  delay( TIME2 );
  toggle( LEDPIN1 );
  delay( TIME3 );
  toggle( LEDPIN2 );
  delay( TIME4 );
} // loop( )
```

- **007_ManyBeaconsBlink.ino**

4 Beacons now, also shifted. Notice that we need 8 values to delay with:

```
/*
 * Making many beacons work! Need external LEDs on pins 11, 12 and A3
 */

// \___________________/----  ,  \___________________/----  ,  \______
// --\___________________/--  ,  --\___________________/--  ,  --\____
// ----\___________________/  ,  ----\___________________/  ,  ----\__
// __________/----\_________  ,  __________/----\_________  ,  ________
// x x x    x    x   x x x  ,  x x x    x    x   x x x  ,  x x x


#define TIME1           50
#define TIME2           50
#define TIME3          200
#define TIME4           50
#define TIME5         1550
#define TIME6           50
#define TIME7           50
#define TIME8           12
```

- **008_TooMany.ino**

Don’t worry, it will compile and upload but no LEDs will turn on.

14 values to compute first!

```
/*
 * Making many LEDs work! Need a computer to do the math!
 * All sequences need to be the same length to repeat.
 *
 * And how do you add a random light like Welder in here???
 */

// \__________________/----  ,  \__________________/----  ,  \______
// --\__________________/--  ,  --\__________________/--  ,  --\____
// ----\__________________/  ,  ----\__________________/  ,  ----\__
// _________/----\________  ,  _________/----\________  ,  ________
// \_/-\_/-\_/-\_/-\_/-\/-  ,  \_/-\_/-\_/-\_/-\_/-\/-  ,  \_/-\_/
// x x x x xxx x x x x xxx  ,  x x x x xxx x x x x xxx  ,  x x x x
```

Timing in software…

It works, kind of: **Blocking** -> delay( )

Just wait here, do nothing else.

Better: **Polling** -> check every so often, millis( )

Is it time yet?

Best: **Interrupts** -> Let ME interrupt YOU when it is ready. Egg timer, microwave, kettle whistling

But, the ATMEGA328P only has **3 timers** to generate timer interrupts!

So, **polling** with **millis( )** it is!

https://www.arxterra.com/10-atmega328p-interrupts/#Interrupt_Basics

Polling with millis( )

- A value that increments every millisecond from power up (and clears on a reset)
- Will overflow in about 49.7 days (but not a problem here)
- Write down the time now (call it “previous”), and check every so often if a certain amount of time has passed since “previous”
- If something needs to repeat, update the “previous” and keep checking if another period has expired.
- **009_Millis.ino**, GO!

OOP: Object Oriented Programming

What is object-oriented programming?

Object-oriented programming (OOP) is a computer programming model that organizes software design around data, or objects, rather than functions and logic. An object can be defined as a data field that has unique attributes and behavior.

https://www.techtarget.com/searchapparchitecture/definition/object-oriented-programming-OOP

- Don’t fear, you were already using OOP with Serial.begin(11500) and Serial.println( )
- It allows us to have a Beacon keep track of its own variables, like pin number and the “previous”
- It allows us to call methods in Beacon that does not clutter all our code in the Sketch
- It gives us an easy way to create another Beacon with less duplicating code

Introducing: “Heartbeat”!

- If nothing works, how do you know your code is running?
- Let’s put a heartbeat in the system and then we know!
- I like it on pin 13, always! Why? Because the builtin LED is almost always there!
- **010_OOP.ino** it is!
- Sketch folders can also contain other files
- Notice the New Tab (Ctrl+Shift+N) down arrow at the top right
- It is common practice to #include .h file in C/C++, so we will do the same and stick our new OOP Class in there.
- Note, a **Class** tells what the code can do, but an **Object** is created to do the job with a **Constructor**.

Now for Beacon done by OOP!

- And with millis( )
- Run **011_OOPBeacon.ino**

```
Beacon    myBeacon = Beacon( BEACONPIN, BEACONONTIME, BEACONOFFTIME );
Heartbeat myHeart  = Heartbeat( HEARTBEATPIN, HEARTBEATTIME );

void setup( ) {
  …
  myBeacon.begin( );
  myHeart.begin( );
} // setup( )

void loop( ) {
  myBeacon.update( );
  myHeart.update( );
} // loop( )
```

- How about two beacons again? Go with **012_TwoBeaconsAgain.ino**

Much easier to stick **Pointers** in arrays…

- Not going to bore you with the details, just note the *, the **new** and the **->**
- Just note that **pinMode( )** is called with **begin**( ) in **setup**( ) and not in the **Constructor** which is run when the code starts up.
- **013_TwoBeaconPointers.ino**

```
Beacon *myBeacon = new Beacon( BEACON1PIN, BEACONONTIME, BEACONOFFTIME );

myBeacon->begin( );

myBeacon->update( );
```

- Still the same code in the Class, try two Beacons with different configuration values: **014_TwoDifferentBeacons.ino**
- How **easy** would this have been using delay( )?

Much easier to stick Pointers in **arrays…**

- And now you can also do 10 Beacons with the config values in arrays

```
Beacon    *myBeacons[ BEACONS ];
```

- Just make sure to update each one!

```
void loop( ) {
  for( uint8_t i = 0; i< BEACONS; ++i ) {
    myBeacons[i]->update( );
  } // for

  myHeart->update( );
} // loop( )
```

- 015_TenBeacons.ino

Ready for Random? Welder it is!

- This might seem like a lot to take in, but we determine with 4 calls to random, how many short burst are we going to do with what on and off period, also random
- And then a random long completely OFF period to replace the rod, clean the unit being welded, etc.
- We use a small State Machine to determine if we are in the ‘F’ long off state, the short ‘f’ state or the short ‘o’ state. See the “switch( ) case:” code.
- Most people do this with if then else, but nesting many ifs become hard to debug.
- And even if it is all Greek to you, just know that you only have to create it with **new**, call **begin( )** and then **update( )**! See **016_Welder.ino**

Variables (YOUR choice, but stay consistent):

- The C and C++ languages define what are illegal names for variables.
- There are many different types of variables and they all take up various amounts of memory.
- A char is 8 bits (0-255), and int is usually 16 (depending on what you compile it for) (-32,768 to 32,767) and a bool is simply true or false (but HIGH or LOW and 1 or 0 is also used)
- To have some consistency across platforms, I do not like to use **char** or **int** or **unsigned int**, I prefer the predefined **uint8_t** and **int16_t** and **unint16_t** types.
- To help remember what the variable type is, you will notice that I use a character or two in front: **u8**Value, **u16**Value or **i16**Value and **b**AreWeReady.
- This also helps with the CamelCase, a way to use English to name a variable with more descriptive words. The b, 8 or 16 is followed by a capital letter.

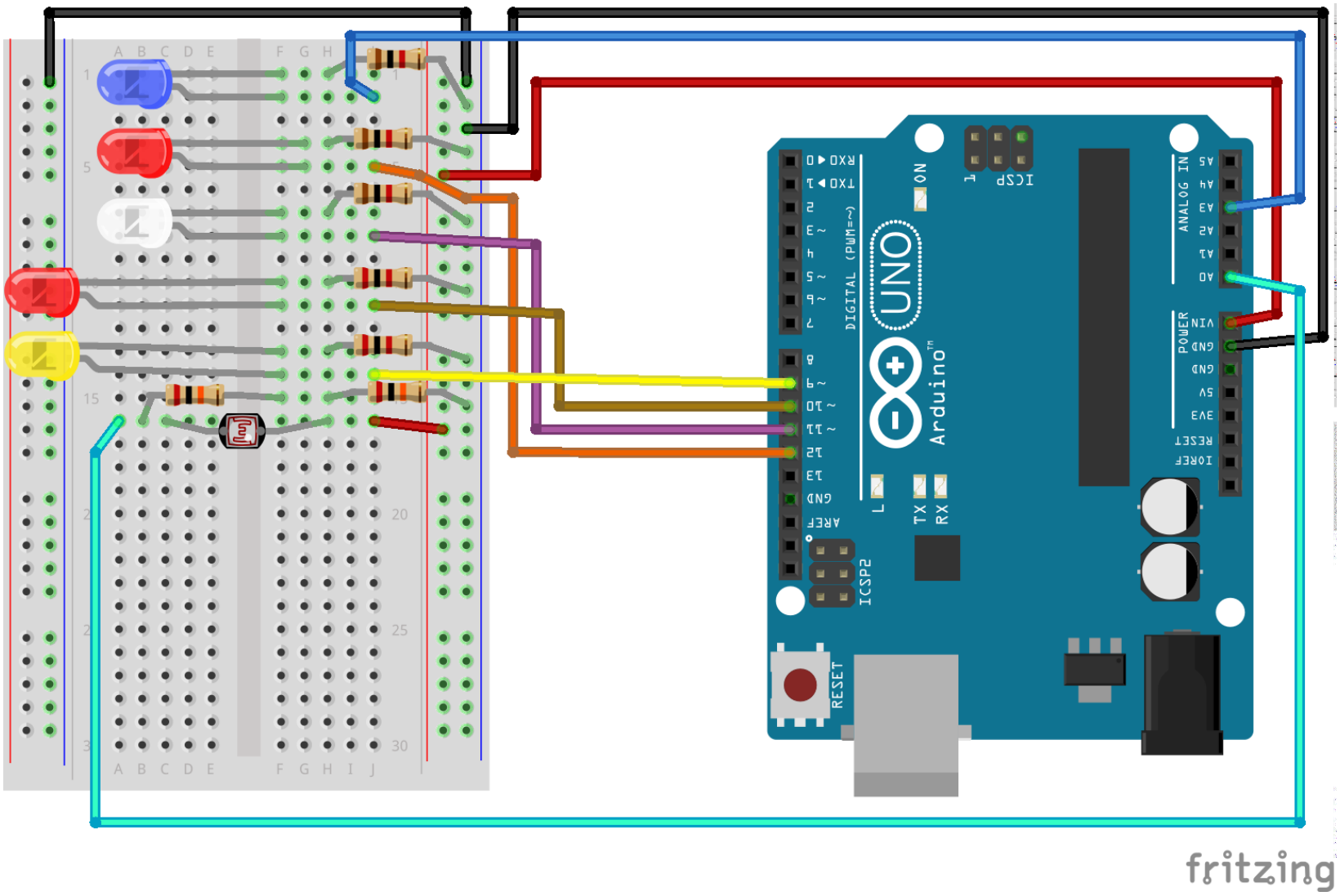
Welder with 2 Beacons then!

- Run **017_WelderTwoBeacons.ino**

```
void setup( ) {
  myBeacon1->begin( );
  myBeacon2->begin( );
  myWelder->begin( );
  myHeart->begin( );
} // setup( )

void loop( ) {
  myBeacon1->update( );
  myBeacon2->update( );
  myWelder->update( );
  myHeart->update( );
} // loop( )
```

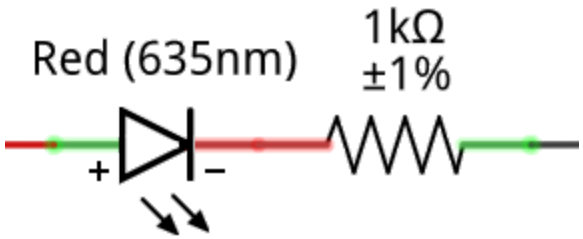
Ready for crazy Random? Let's go camping…



- Use **2 kOhm** resistors for these LEDs to save enough 1 kOhms for TrafficLites

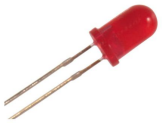
LEDs and their Resistors

An LED works like a one-way street. Just like cars accomplishing something by following everyone else in the same direction, an LED only allows current to flow in the direction of the triangle's arrow (from anode to cathode). It makes light once the forward voltage (Vf) across the diode is exceeded and electrical current starts flowing.



If you take a quick look at the datasheet for this Radio Shack 5 mm red LED (https://www.radioshack.com/products/5mm-red-led), you'll see that the Vf is somewhere between 2.2 and 2.8 Volts:

Forward voltage: 2.2 V (typical)/ 2.8 (max) @ 20 mA



So to make sure it will turn on, you need to do two things, provide more than 2.2 Volts **and put a resistor big enough in series** to avoid reaching the maximum rated current:

Forward Current (If): 25 mA (**Max**)

Assume we have a 5 V source to turn the LED on with and we believe the 2.2 Vf voltage, we need to use 5 - 2.2 V or 2.8 V across the current limiting resistor. From V = I x R, we can tell that if the current through the LED is the same as the current through the resistor, and we choose to run the LED with medium brightness at 3 mA:

R = V / I or R = 2.8 / 0.003 = **933 Ohms**. < - - looks like 1k Ohm to me!

Crazy Random and using PWM!

- The Arduino pins with wiggles, can output a PWM signal.
- Connecting a PWM signal to an LED will deliver less current on average, as the PWM duty cycle goes down. This is perceived as dimming.
- So we can randomly switch between different levels of dimming and create the same effect as a fire. Run **018_Campfire.ino**
- We set the min and max dim levels, and pick a level at which the light's dim level would change if the new random number exceeds it. So time in the flickering is also random.

```
void update( ) {
   unsigned long ulNow = millis( );
   if( ulNow - ulPrevious >= ulTimeout ) {
     ulPrevious = ulNow;
     if( random( 0, TRIGGERMAX ) > u8TriggerVal ) {
       analogWrite( u8Pin, random( u8RandomMin, u8RandomMax ) );
     } // if triggered
   } // if time
 } // Update( )
```

Crazy Random and using 2 PWMs!

- It also looks better when there is a different color light in your fire flashing with a slightly different random pattern. Run **019_CampfireByTwo.ino**

```
#define CAMPFIRE1PIN         9
#define CAMPFIRE2PIN        10

// How often to evaluate
#define CAMPFIRE1TIMEOUT    10
#define CAMPFIRE2TIMEOUT    15

// How often to trigger (0-100)
#define CAMPFIRE1TRIGGER    80
#define CAMPFIRE2TRIGGER    60

// Dimmest levels
#define CAMPFIRE1MIN        20
#define CAMPFIRE2MIN        10

// Brightest levels
#define CAMPFIRE1MAX       100
#define CAMPFIRE2MAX        50
```

- #define is used by the preprocessor to replace every CAMPFIRE1MAX with 100

And now for something that needs coordination:

- The north-south traffic light has to stay coordinated with the east-west light
- Another State Machine to keep track in which phase a light is:

```
 u8State--; // 4 goes to 3 only once, so no need to switch case 4:
switch ( u8State ) {
  case 0:
    u8State = 3;
    // fall through
  case 3:
    u16CountDown = GREEN_TIME;
    break;
  case 2:
    u16CountDown = YELLOW_TIME;
    break;
  case 1:
    u16CountDown = RED_TIME;
    break;
  case 4:
    // fall through
  default:
    break;
}; // switch
```

- And setting the output accordingly:

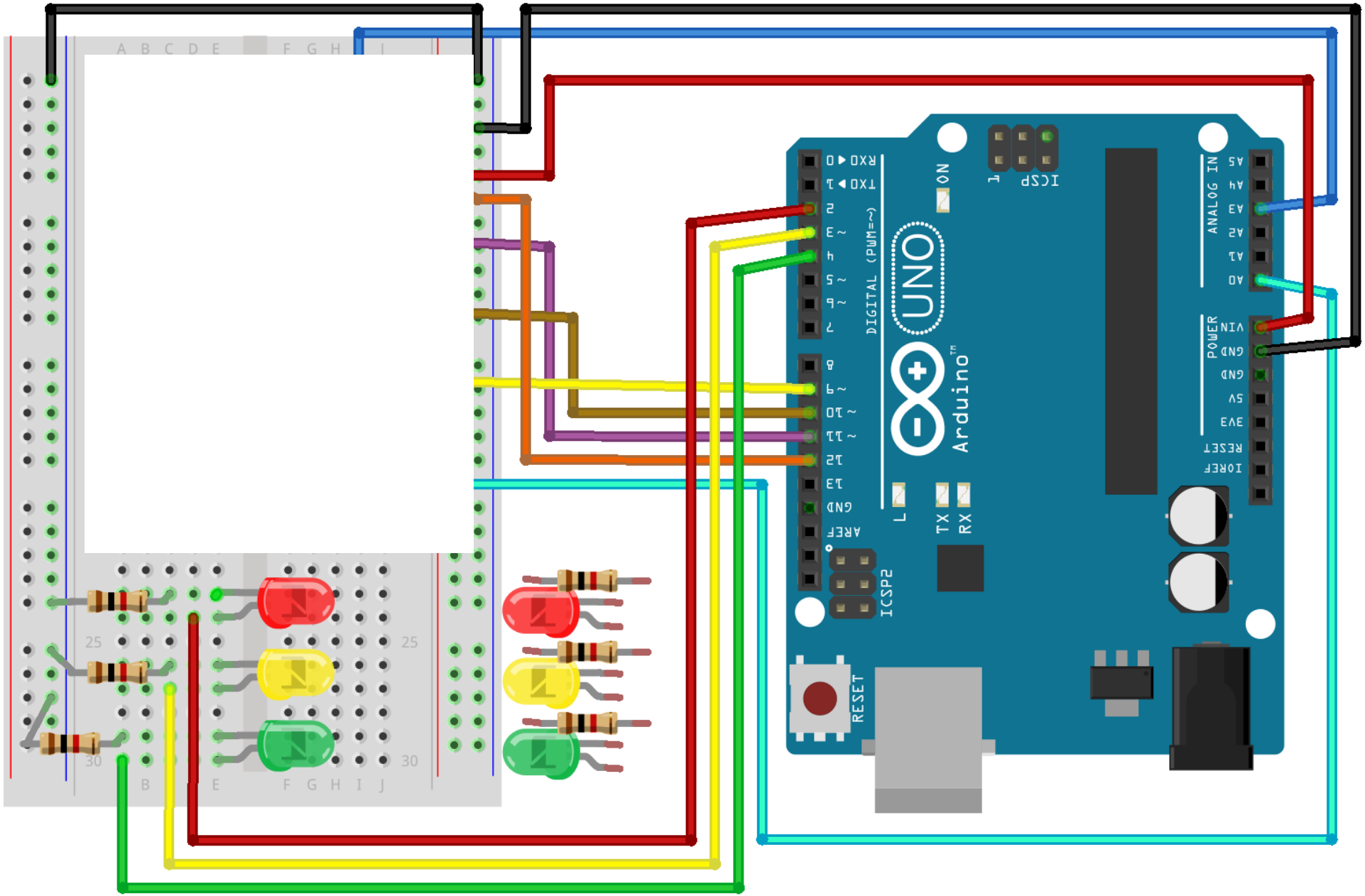
```
void output( ) {
   digitalWrite( u8RedPin, !bWhenOn ); // turn all off
   digitalWrite( u8YelPin, !bWhenOn ); // turn all off
   digitalWrite( u8GrnPin, !bWhenOn ); // turn all off
   switch ( u8State ) {
     case 0: break;   // stay off
     case 4:          // red, while the other light is red too
       // fall through (no break, so continue into the next "case"
     case 1:          // red
       digitalWrite( u8RedPin, bWhenOn ); break;
     case 2:          // yellow
       digitalWrite( u8YelPin, bWhenOn ); break;
     case 3:          // green
       digitalWrite( u8GrnPin, bWhenOn ); break;
   }; // switch
 } // void output( )
```

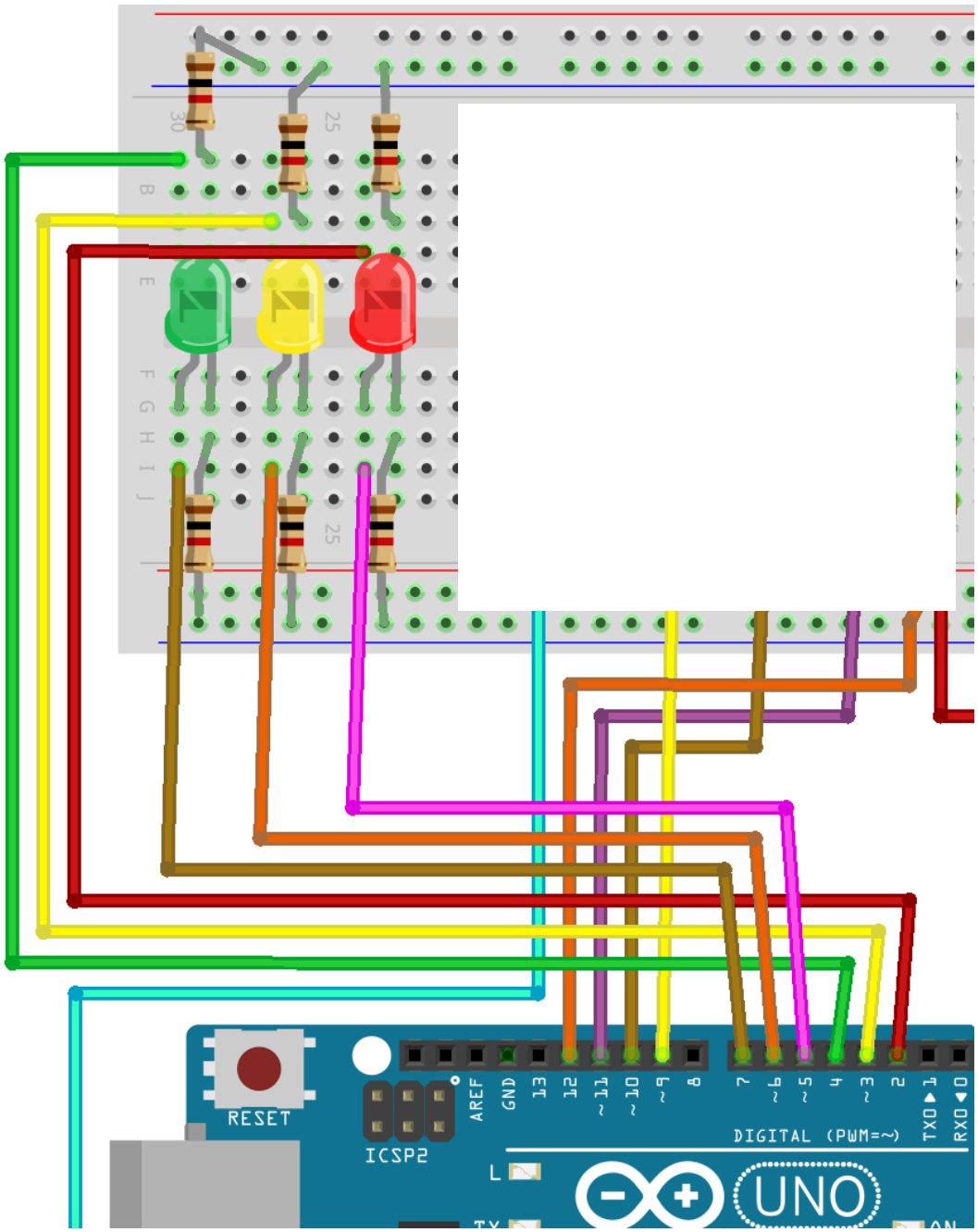
- Note the use of **bWhenOn**, sometimes you want to use a Common Anode multi-color LED and then you need to sink current by making the pin low to turn it on. So if you set **bWhenOn** to **LOW**, "active" will be **LOW**.

- Silicon can usually also sink more current than source current.

Traffic Light:

- Only hiding the OTHER lights, don't remove them!
- Install the LEDs as shown in the next images, if you run out of 1 kOhm resistors, use 2k Ohms resistors for the Campfire LEDs
- North-South shown in first image
- Then add East-West in second image
- Now run **020_TrafficLite.ino**
- The first time we determine the trafficLiteNS signal is red, we run the delayStart( ) method of the trafficLiteEW to delay its red, so there is an overlap where both reds are on.


A B C D E F G H I J
25
30
UNO
Arduino™
DIGITAL (PWM~)
ANALOG IN
POWER
ICSP
ICSP2
RESET
L
TX
RX
ON
fritzing


RESET
ICSP2
AREF
GND
13
12
11
10
9
8
7
6
5
4
3
2
TX0 1
RX0 0
DIGITAL (PWM=~)
L
UNO

Stick it ALL TOGETHER now:

- Now run **021_AllTogether.ino**
- Then turn the sensor on (use 2 x 20k resistors if you bought the kit per instructions): **5 Vdc -> Sensor -> A0 -> 40 kOhm -> GND**
- Run **022_AllTogetherWithSensor.ino**
- Check Serial Port Monitor for values shown when sensor is block (a) and when not (b)
- Use a value between (a) and (b) for **LIGHTTRIGGERLEVEL**, line 48, recompile and upload
- Now the sensor is changing the **Heartbeat frequency** and enables or disabled the **Welder**.

Library instead of .h file in Sketch folder?

- File->Preferences will show where your Sketchbook Location is.
- If it does not exist, create a “libraries” folder in that location.
- Move the WelderInLib folder into the libraries folder and restart the IDE.
- Now you can `#include “WelderInFolder.h”` from ANY sketch.
- The “InLib” was only added to the .h and folder name for demonstration
- Run **023_UsingLibrary.ino**

FadeLight

- Something for you to play with.
- You will need to use a PWM (wiggle mark) pin.
- See 024_FadeLight.ino
- In State 2 and 3, we increase or decrease the brightness of the LED to fade it in or out.
- Enjoy!

Important notes:

- Copying code from anywhere else, watch out for the quotes: “ and ” is not the same as "

- Keep the description and version in your VERSION_STR up to date, you need to be able to find the source, since the HEX file from the Arduino will not show you the C/C++ code again

- Use dates and English words as much as you can. Easier to find and search later

- More interesting reading
https://roboticsbackend.com/the-arduino-language-in-10-points/

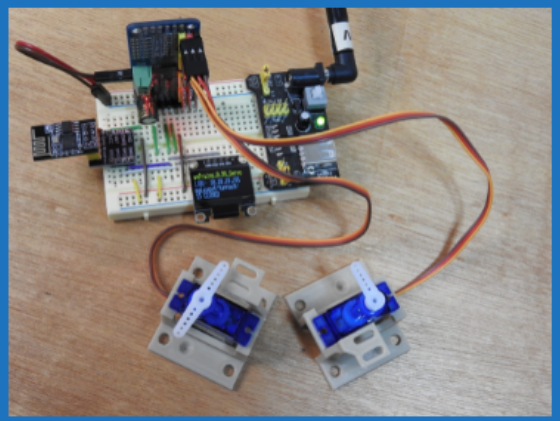
Special Plug for mqTrains!

Created with the Arduino IDE, but the ESP8266 is Espressif's WiFi connected microcontroller. Huh, only $1.50???

mqTrains MQTT in the model train world downloads

mqTrains – Home

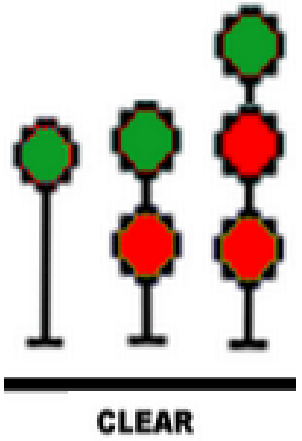
If you want a layout with less wires, why not use WiFi?



Any questions?

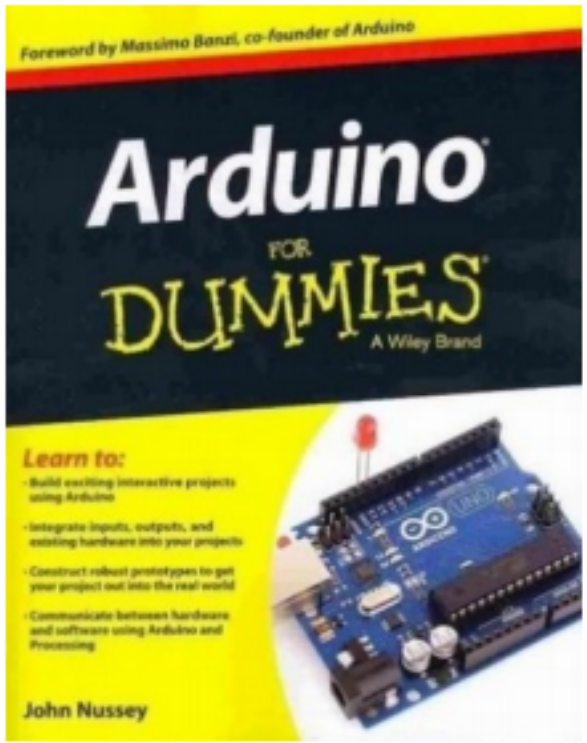
The only **dumb question** is the one **you did not ask**!

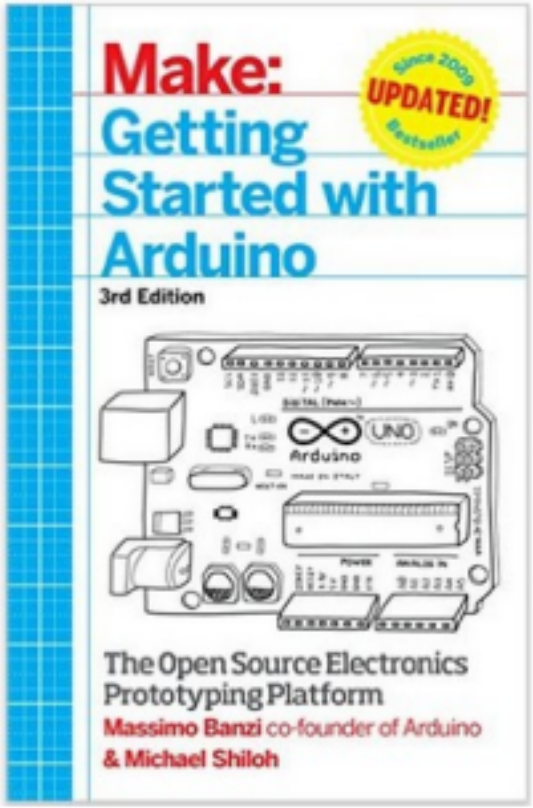
The End

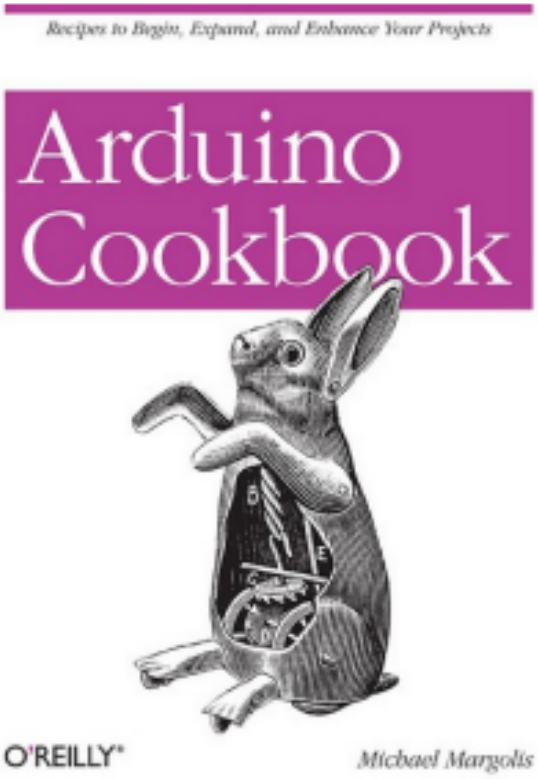


Thank you to Joel, David, Riley, *Bob, Donna, John, Stephanie, Alissa, Bianka, my Boss,*
and all the others that did not bother us while we were having fun doing this…

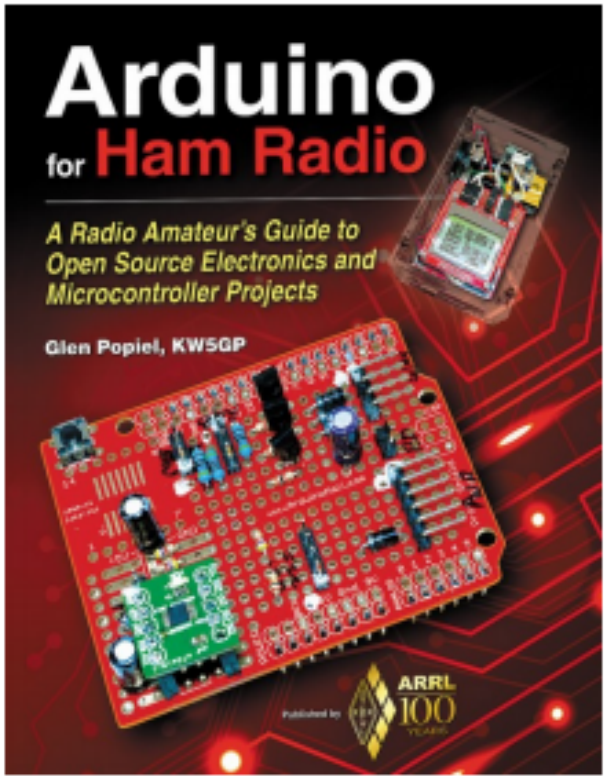
Get educated:






Recipes to Begin, Expand, and Enhance Your Projects
Arduino Cookbook
O'REILLY
Michael Margolis


John Baichtal, Matthew Beckler, & Adam Wolf
Make: LEGO and Arduino Projects
Learn by Discovery
Projects for Extending MINDSTORMS NXT with Open-Source Electronics
O'REILLY
Make:


Arduino for Ham Radio
A Radio Amateur's Guide to Open Source Electronics and Microcontroller Projects
Glen Popiel, KW5GP
ARRL
100

RrrDuino for real model railroads

So, what can an **Arduino** do for my railroad? (See the whole list at TxNamib.com)

Since we can turn things on and off:

Lighting:

Turn LEDs on and off at specified or random times
Welding, cutting, torching
Cycle lights in a building, house or church
Street lights
Traffic lights
Grade crossing lights
Signals for the railroad
Flickering lamps, like an Arc Welder and Bathroom light on another pin.
Candles, campfires, fire in a building, or just a stove...
Police car lights, Fire Trucks, Ambulances...

Servos and steppers:

A grade crossing gates
A wigwag
An animated scene, windmill, gate, logs, manufacturing plant
Change a turnout position
Moving an uncoupling magnet in and out of position
A belt driven saw mill
A helicopter, or just a windsock
An overhead crane, like the kind in an intermodal yard picking containers off cars
A forklift lifting

A water tank lowering its spout
A car or engine washing station
Building doors opening or sliding
Controlling your turntable or transfer table
Moving blocks of ice at an ice house
The guy popping out of a tower!

Sound:
A grade crossing with ding, ding, ding...
After the waterspout is down...water!
A coal loading facility
clickity clack wheel sounds, anywhere
"All Aboard?"
"64 wheels counted"

Data:
How 'bout an LCD or LEDs to show train departure time, or expected time of arrival

Since we can sense things:
Train occupancy
WigWag
Grade crossing
Street lights at dusk
Counting Axles: "86 Axles, zero defects!"

Speed:

Train speed, how fast were you going?

Touch:

Yes, **turn it on or off now**

Temperature:

Time to turn the fan on

Boiler is hot, let's go!

Humidity, water, pressure, you name it, if it has an optical, digital or analog output, we can measure it!

Acceleration:

If we speed up, the tires might scream

When we brake, we might want screeching tires

Tilt:

Oops, here comes that water

Or the crane is down, active that electromagnet to pick the metal up

...and for the somewhat advanced

DCC:

An accessory decoder, so your DCC throttle (or JMRI for that matter) could tell accessory #1042 to move a turnout, or turn something on, all over the DCC bus. (**And another site**)

...and for the even little more advance

LCC or OpenLCB:

Moving a few Servos

Since we can talk to things (Infrared, BLE (Bluetooth), WiFi, Morse code):

We could send data, this train left
Next train should be arriving at...
Block occupancy, tell the next or opposing signal of such
Controlling the lights and speed of your **Faller self driving car?**

We might talk to our home automation system to turn the A/C on, or heat
We could email a friend
We could update a website, or post on Facebook

And All Advanced…

Can't stand the difference between Verify and Upload and the fact that Upload verifies every time you need to program 25 boards?

Meet **arduino-builder**

C:\...\Arduino\TxHoMast_Test_Builder>**arduino-builder** -hardware="C:\Program Files (x86)\Arduino\hardware"
-tools="C:\Program Files (x86)\Arduino\tools-builder"
-tools="C:\Program Files (x86)\Arduino\hardware\tools"
-build-path="C:\Users\gert\Documents\Arduino\TxHoMast_Test_Builder\build"
-fqbn=a**rduino:avr:nano TxHoMast_Test.ino**

Now you also know where the .hex file is for upload next time: In the build-path folder!