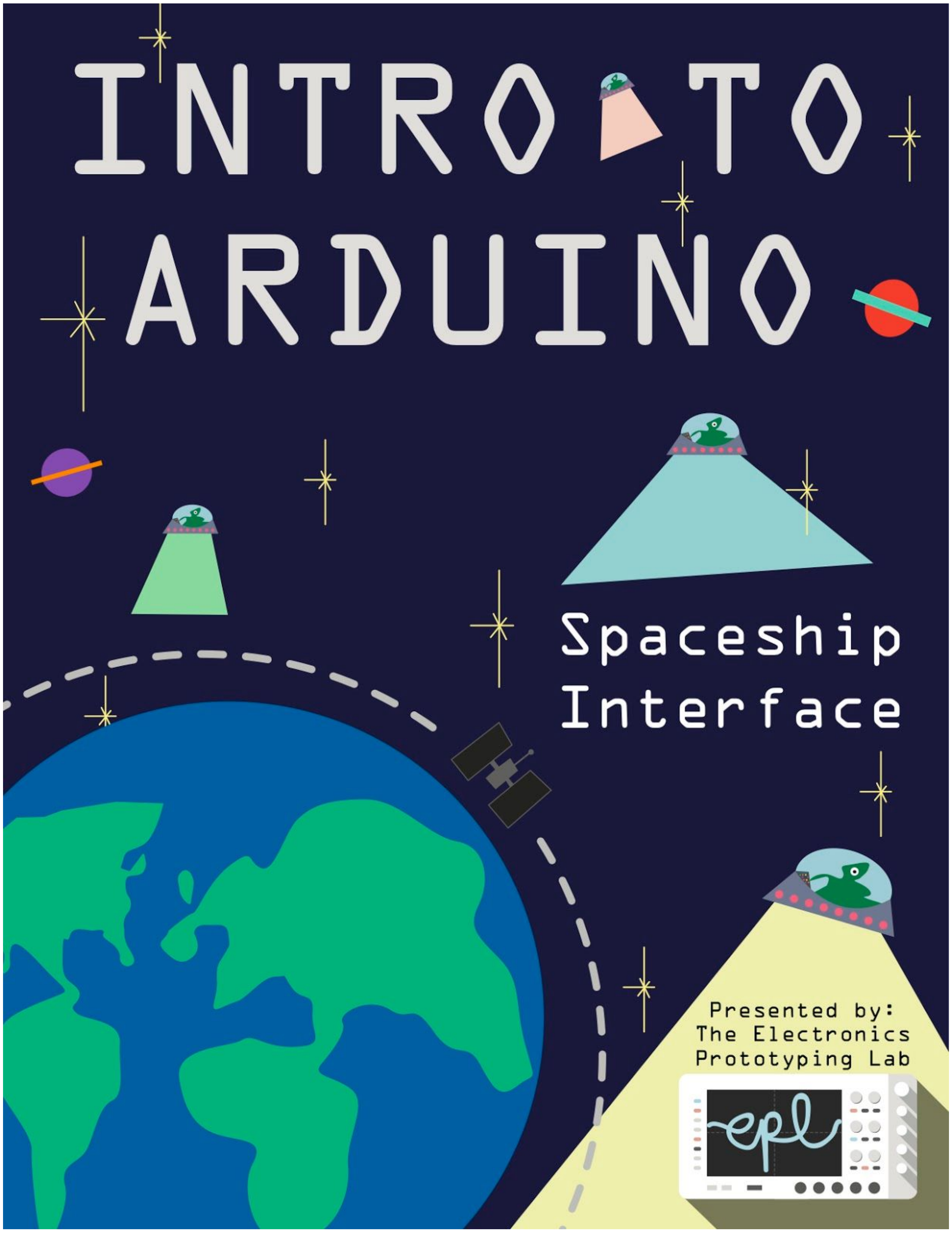
INTRO TO ARDUINO

Spaceship Interface

Presented by:
The Electronics Prototyping Lab



The Electronic Prototyping Lab's Intro to Arduino Workshop

Welcome to the Electronics Prototyping Lab's Introduction to Arduino Workshop! We're so excited to have you here and looking forward to getting going with this popular microcontroller. This workshop is 3 hours long which gives us a chance to cover some basics of the Arduino. Then we will use common electronic components and write our own program to create an interface system for a spaceship. No prior knowledge is needed for the class, and materials will be provided.

In addition to this workshop, the EPL also offers workshops on many other popular technology topics. Workshop Weekend, the event where these workshops are held, happens three times per year, during the Fall, Winter, and Spring terms. Our standalone workshops are all about 3 hours long and open to the public. Our goal for these workshops is to provide additional training that helps students to be successful in university STEM programs. That makes our courses a great fit for PSU students interested in technology, high school students preparing for college, and community members or hobbyists!

Many people take 3 or 4 workshops over the course of the weekend and you are encouraged to sign up for as many as you like. We do limit our class sizes in order to provide an optimal student to teacher ratio, so make sure you sign up soon! You can find out about more of our workshop at our website, or sign up for one on our eventbrite page.

Intro to Arduino - Spaceship Interface
Written and Compiled by Tyler Hull
Modified by David Lay
For EPL Intro to Arduino Workshop
Used for Educational Purposes only

With information or inspiration from:
http://www.arduino.cc
http://www.sparkfun.com
https://en.wikipedia.org

Arduino Overview

Arduino is a single-board microcontroller created in Italy to make using electronics in multidisciplinary projects more accessible. The hardware consists of a simple open source hardware board, also known as a development board, designed around an 8-bit Atmel AVR microcontroller, or a 32-bit Atmel ARM. This board allows the user to easily interact with and control the functions of the microcontroller chip. The software consists of a standard programming language compiler and a boot loader that executes on the microcontroller.

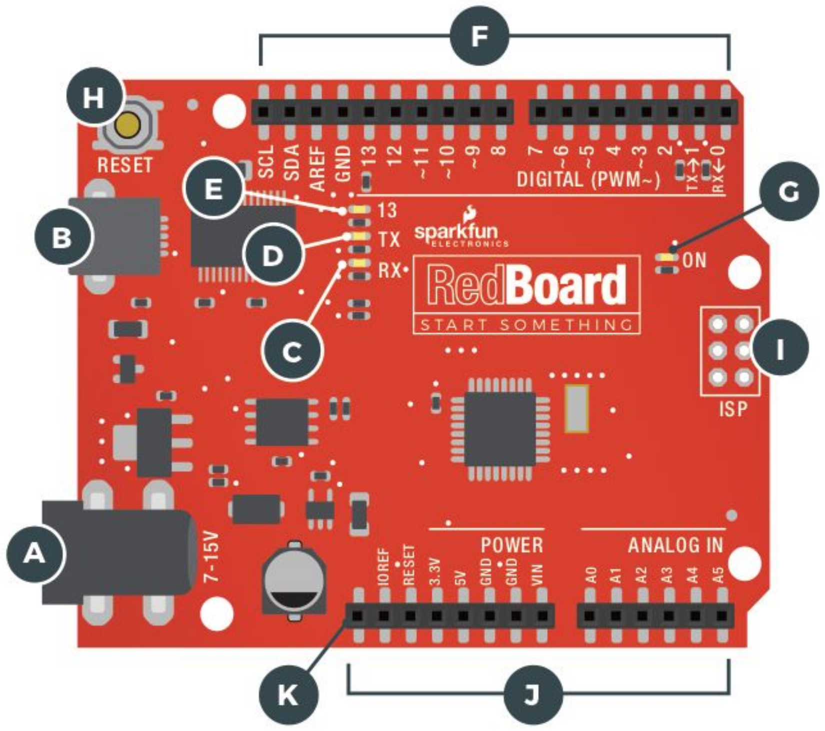
The Arduino project started at the Interaction Design Institute Ivrea (IDII) in Italy. At that time, the students used a microcontroller that cost $100, which was a considerable expense for many students. In 2003 Hernando Barragán created the development platform "Wiring" as a Master's thesis project. The project goal was to create simple, low cost tools for creating digital projects by non-engineers.

The Wiring platform consisted of a printed circuit board (PCB) with an ATmega168 microcontroller, an IDE based on Processing and library functions to easily program the microcontroller. In 2003, Massimo Banzi, with David Mellis, another IDII student, and David Cuartielles, added support for the cheaper ATmega8 microcontroller to Wiring. But instead of continuing the work on Wiring, they forked the project and renamed it Arduino. The initial Arduino core team consisted of Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino, and David Mellis, but Barragán was not invited to participate.

Arduino is open-source hardware. The hardware reference designs are distributed under a Creative Commons license and are available on the Arduino website. Layout and production files for some versions of the hardware are also available. The source code for the IDE is released under the GNU General Public License, version 2.

Although the hardware and software designs are freely available, the developers have requested the name Arduino to be exclusive to the official product and not be used for derived works without permission. Many Arduino-compatible and Arduino-derived boards exist. Some are functionally equivalent to an Arduino and can be used interchangeably. Many enhance the basic Arduino, others are electrically equivalent but change the form factor. Some variants use different processors, with varying compatibility.

Anatomy of the SparkFun RedBoard



REDBOARD HARDWARE OVERVIEW		
A	POWER IN (BARREL JACK)	Can be used with either a 9V or 12V "wall-wart" or a battery pack.
B	POWER IN (USB PORT)	Provides power and communicates with your board when plugged into your computer via USB.
C	LED (RX: RECEIVING)	Shows when the FTDI chip is receiving data bits from the computer.
D	LED (TX: TRANSMITTING)	Shows when the FTDI chip is transmitting data bits to the computer.
E	ONBOARD LED PIN D13	This LED, connected to digital pin 13, can be controlled in your program and is great for troubleshooting.
F	PINS AREF, GROUND, DIGITAL, RX, TX, SDA, SCL	These pins can be used for inputs, outputs, power and ground.
G	POWER LED	Illuminated when the board is connected to a power source.
H	RESET BUTTON	A manual reset switch that will restart the RedBoard and your code.
I	ISP HEADER	This is the In-System Programming header. It is used to program the ATMega328 directly. It will not be used in this guide.
J	ANALOG IN, VOLTAGE IN, GROUND, 3.3 AND 5V OUT, RESET	The power bus has pins to power your circuits with various voltages. The analog inputs allow you to read analog signals.
K	RFU	This stands for Reserved for Future Use.

Hardware Specifications

- Microcontroller: ATmega328
- Operating Voltage: 5V
- Input Voltage (recommended):7-12V
- Input Voltage (limits): 6- 20V
- Digital I/O Pins: 14 (of which 6 provide PWM output)
- Analog Input Pins: 6
- DC Current per I/O Pin: 40 mA
- DC Current for 3.3V Pin: 50 mA
- Flash Memory: 32 KB (ATmega328)
- SRAM: 2 KB (ATmega328)
- EEPROM: 1 KB (ATmega328)
- Clock Speed: 16 MHz

Solderless Breadboards

A solderless breadboard is a tool used for prototyping circuits. Allowing connections to be made quickly without needing to spend the time soldering, breadboards are used with jumper wires to create circuits and connect different sensors and actuators to the Arduino board. Each side of the breadboard has a pair of vertical connections marked "+" and "-". These connect the entire length of the breadboard while the horizontal rows are only connected in set of 5. The center channel separates these rows both physically and electrically. This allows IC chips (integrated circuits) to bridge the gap and all of their connections to be isolated.

Software

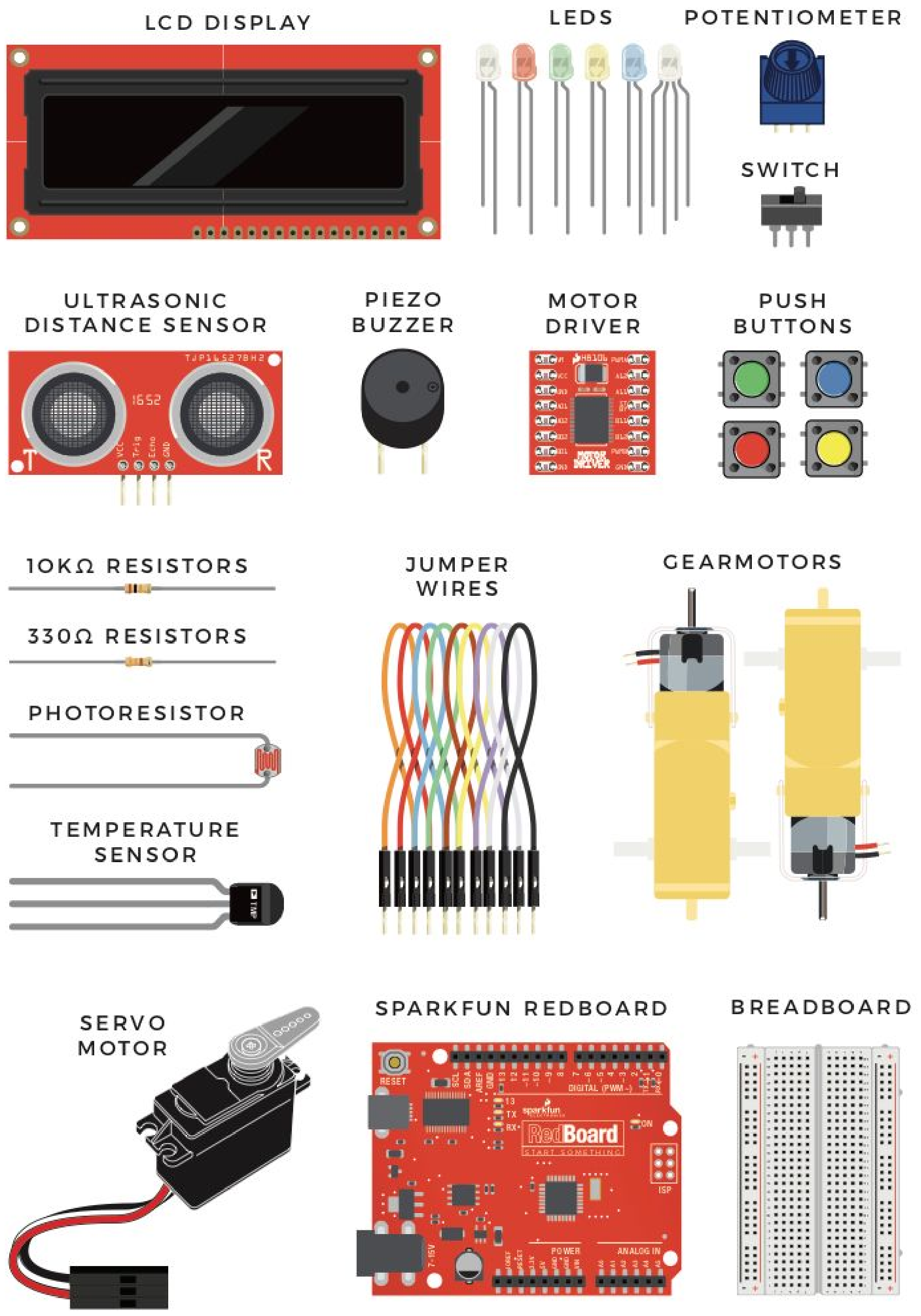
Arduino is its own coding language, sharing many similarities with C programming. You can find out all about the details of the Arduino coding language and how to use the commands on the Arduino website.

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

In addition to the basic commands included in the language, the Arduino environment can be extended by using libraries, just like many other programming platforms. Libraries provide extra functionality for use in sketches that simplify working with hardware or manipulating data. A number of libraries come installed with the IDE, but you can also download or create your own! Find out more on the Arduino website.

https://www.arduino.cc/en/Reference/Libraries

Common Electronic Parts/Components



Solderless Breadboard Usage

POWER BUS

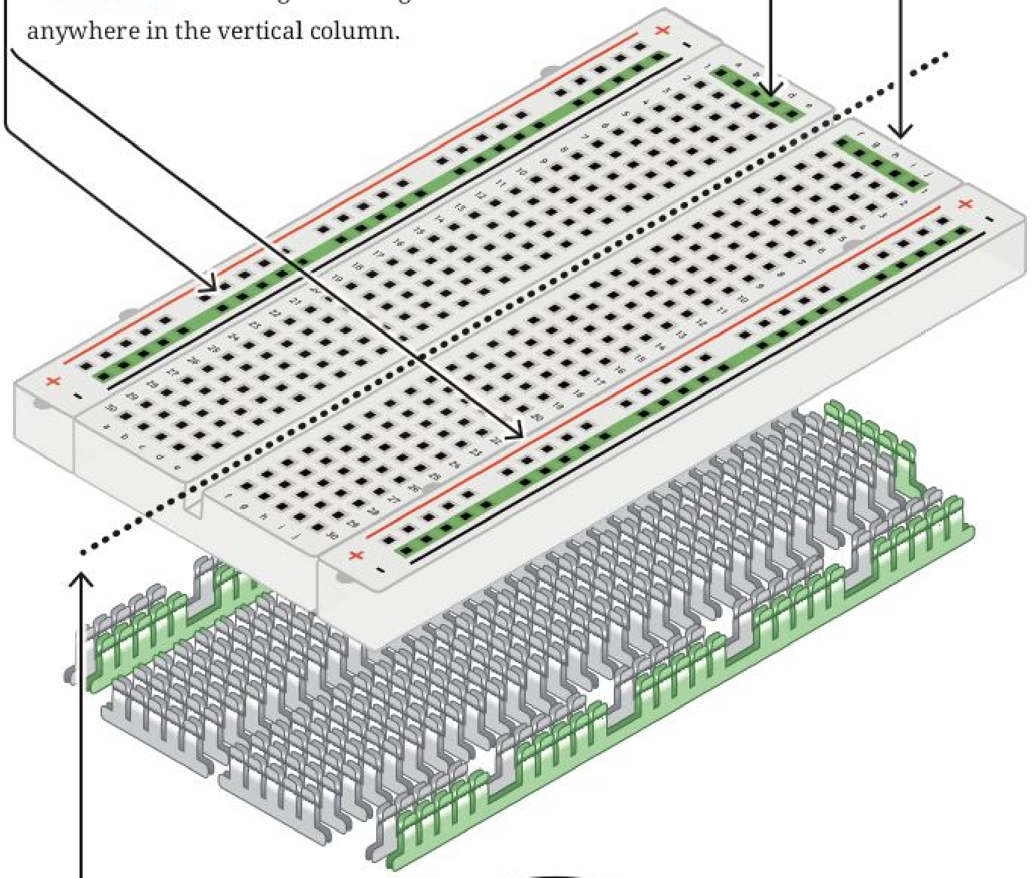
Each side of the breadboard has a pair of vertical connections marked – and +

+ POWER: Each + sign runs power anywhere in the vertical column.

- GROUND: Each – sign runs to ground anywhere in the vertical column.

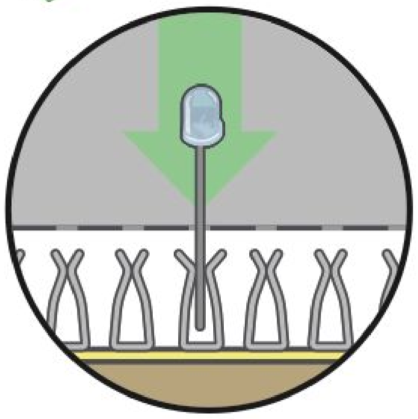
HORIZONTAL ROWS

Each series of 5 sockets marked a–e and f–j are connected. Components connected to a row will be connected to any other part inserted in the same row.



CENTERLINE

This line divides the breadboard in half, restricting electricity to one half or the other.



MAKING A CONNECTION

Most of the components in this kit are breadboard-friendly and can be easily installed and removed.

Arduino Language Functions, Variables, and Structure

FUNCTIONS

For controlling the Arduino board and performing computations.

Digital I/O

digitalRead()
digitalWrite()
pinMode()

Analog I/O

analogRead()
analogReference()
analogWrite()

Zero, Due & MKR Family

analogReadResolution()
analogWriteResolution()

Advanced I/O

noTone()
pulseIn()
pulseInLong()
shiftIn()
shiftOut()
tone()

Time

delay()
delayMicroseconds()
micros()
millis()

Math

abs()
constrain()
map()
max()
min()
pow()
sq()
sqrt()

Trigonometry

cos()
sin()
tan()

Characters

isAlpha()
isAlphaNumeric()
isAscii()
isControl()
isDigit()
isGraph()
isHexadecimalDigit()
isLowerCase()
isPrintable()
isPunct()
isSpace()
isUpperCase()
isWhitespace()

Random Numbers

random()
randomSeed()

Bits and Bytes

bit()
bitClear()
bitRead()
bitSet()
bitWrite()
highByte()
lowByte()

External Interrupts

attachInterrupt()
detachInterrupt()

Interrupts

interrupts()
noInterrupts()

Communication

serial
stream

USB

Keyboard
Mouse

VARIABLES

Arduino data types and constants.

Constants

Floating Point Constants

Integer Constants

HIGH | LOW

INPUT | OUTPUT | INPUT_PULLUP

LED_BUILTIN

true | false

Conversion

byte()

char()

float()

int()

long()

word()

Data Types

String()

array

bool

boolean

byte

char

double

float

int

long

short

string

unsigned char

unsigned int

unsigned long

void

word

Variable Scope & Qualifiers

const

scope

static

volatile

Utilities

PROGMEM

sizeof()

STRUCTURE

The elements of Arduino (C++) code.

Sketch

loop()

setup()

Control Structure

break

continue

do...while

else

for

goto

if...else

return

switch...case

while

Further Syntax

#define (define)

#include (include)

/* */ (block comment)

// (single line comment)

; (semicolon)

{} (curly braces)

Arithmetic Operators

% (modulo)

* (multiplication)

+ (addition)

- (subtraction)

/ (division)

= (assignment operator)

Comparison Operators

!= (not equal to)

< (less than)

<= (less than or equal to)

== (equal to)

> (greater than)

>= (greater than or equal to)

Boolean Operators

! (logical not)

&& (logical and)

|| (logical or)

Pointer Access Operators

& (reference operator)

* (dereference operator)

Bitwise Operators

& (bitwise and)

<< (bitshift left)

>> (bitshift right)

^ (bitwise xor)

| (bitwise or)

~ (bitwise not)

Compound Operators

&= (compound bitwise and)

*= (compound multiplication)

++ (increment)

+= (compound addition)

-- (decrement)

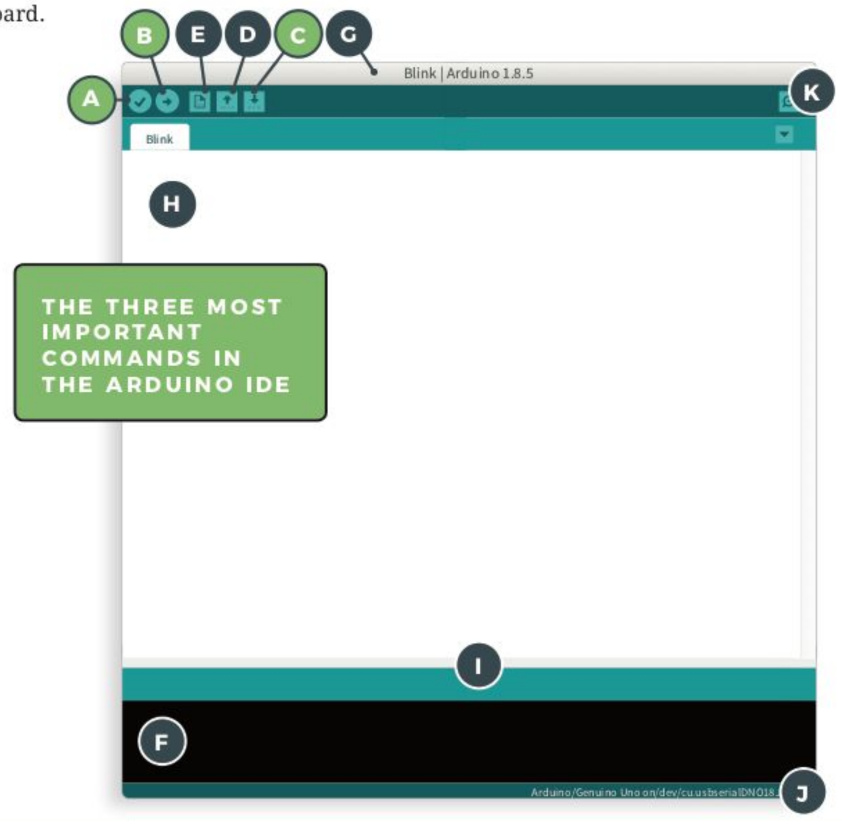
-= (compound subtraction)

/= (compound division)

|= (compound bitwise or)

Arduino IDE/Sketch

Open the Arduino IDE software on your computer. Poke around and get to know the interface. We aren't going to code right away; this step is to set your IDE to identify your RedBoard.



GRAPHICAL USER INTERFACE (GUI)		
A	VERIFY	Compiles and approves your code. It will catch errors in syntax (like missing semicolons or parentheses).
B	UPLOAD	Sends your code to the RedBoard. When you click it, you should see the lights on your board blink rapidly.
C	SAVE	Saves the currently active sketch.
D	OPEN	Opens an existing sketch.
E	NEW	Opens up a new code window tab.
F	DEBUG WINDOW	Displays any errors generated by your sketch.
G	SKETCH NAME	Displays the name of the sketch you are currently working on.
H	CODE AREA	Where you compose or edit the code for your sketch.
I	MESSAGE AREA	Indicates if the code is compiling, uploading or has errors.
J	CONNECTION AREA	Displays the board and serial port currently selected.
K	SERIAL MONITOR	Opens a window that displays any serial information your RedBoard is transmitting (useful for debugging).

Arduino IDE/Sketch

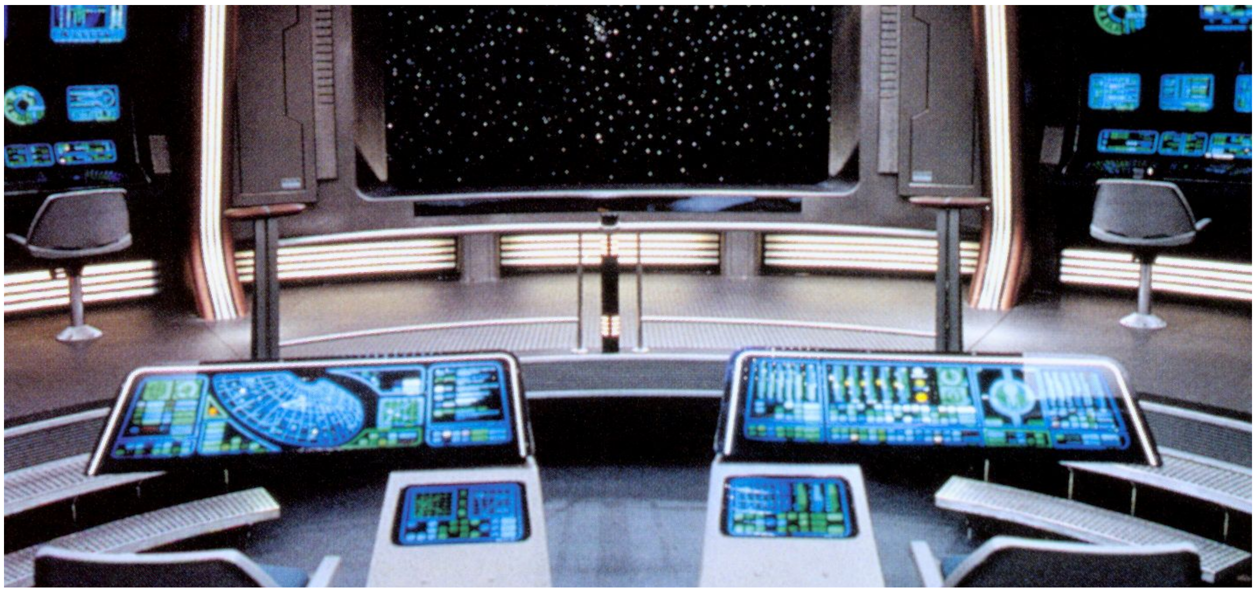
Let’s get started with the traditional “1st” Arduino program. This simple program makes a LED connected to pin 13 blink on and off. Since the Arduino’s onboard LED is also connected to pin 13, we won’t need to connect anything to the Arduino to run this sketch.

- **Type the code below into your IDE, or load the example from the example menu.**

```
void setup() {

 pinMode(13, OUTPUT);

}

// the loop routine runs over and over again forever:

void loop() {

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

}
```

- **Now try changing the delay to make the LED blink faster. What do we need to modify to make this happen?**

SPACESHIP INTERFACE



We're so glad you joined the team at EPL (Explorer Propulsion Laboratories)!!!

All of our previous ship designs required the space team to run around the bridge to each station since they were all controlled separately. Turns out that's not a very effective way to manage a starship….

Anyways… We needed your expertise to create an interface system that can control all of the spaceship's functions in one place. Let's get started.

Libraries, Variables & Pin Setup

We'll be using the standard flight controller board (Arduino) for this design. Let's start by defining some variables and setting up the pins.

- **Can you define 2 variables (called 'switchState' and 'switchStateTwo') to hold the state of the switches, then create a variable to hold the state of the instructions menu?**
- **Then, we need a variable for holding laser power data and a counter for button presses.**
- **Next, we need to #include the Engine (servo) library, then create an Engine (servo) object and a variable to hold its position.**
- **Next, we will need to use pins 3-7 as outputs. Pins 2, 8 and A0 will be used as inputs. Make sure to set those pins and attach the Engine (servo) object to pin 10.**
- **Finally, set up the flight controller to read from pin 2, pin 8, and A0. These values should be stored in the variables you created.**

```
#include <Servo.h>

int switchState = 0; //Sets initial value for switchState.
int switchStateTwo = 0; //Sets initial value for switchState.
int instructions = 1; // Instructions will print while this variable set to 1.
int laserPower = 0; // variable to store the laser power data.
int buttonCounter = 0; // Sets a counter for button 2 presses to zero.
int enginePosition = 0;    // variable to store the servo position.
Servo galacticEngine;  // create servo object to control our space engine.

void setup() {
  pinMode(3,OUTPUT);
  pinMode(4,OUTPUT);
  pinMode(5,OUTPUT);
  pinMode(6,OUTPUT);
  pinMode(7,OUTPUT);
  pinMode(2,INPUT);
  pinMode(8,INPUT);
  pinMode(A0,INPUT);
  galacticEngine.attach(10);  // attaches the servo on pin 10 to the servo
object
}


void loop(){
  switchState = digitalRead(2); // Assigning the variable switchState with the
values read from button 1
  switchStateTwo = digitalRead(8); // Assigning the variable switchState with
the values read from button 2
  laserPower = analogRead(A0);

  Serial.begin(9600); // This command allows us to print to the serial
terminal

} // This brace closes the program loop
```

Nice Work! Now the flight computer knows where to read our inputs, and where to send our outputs!

We should get started on the initialization sequence next.

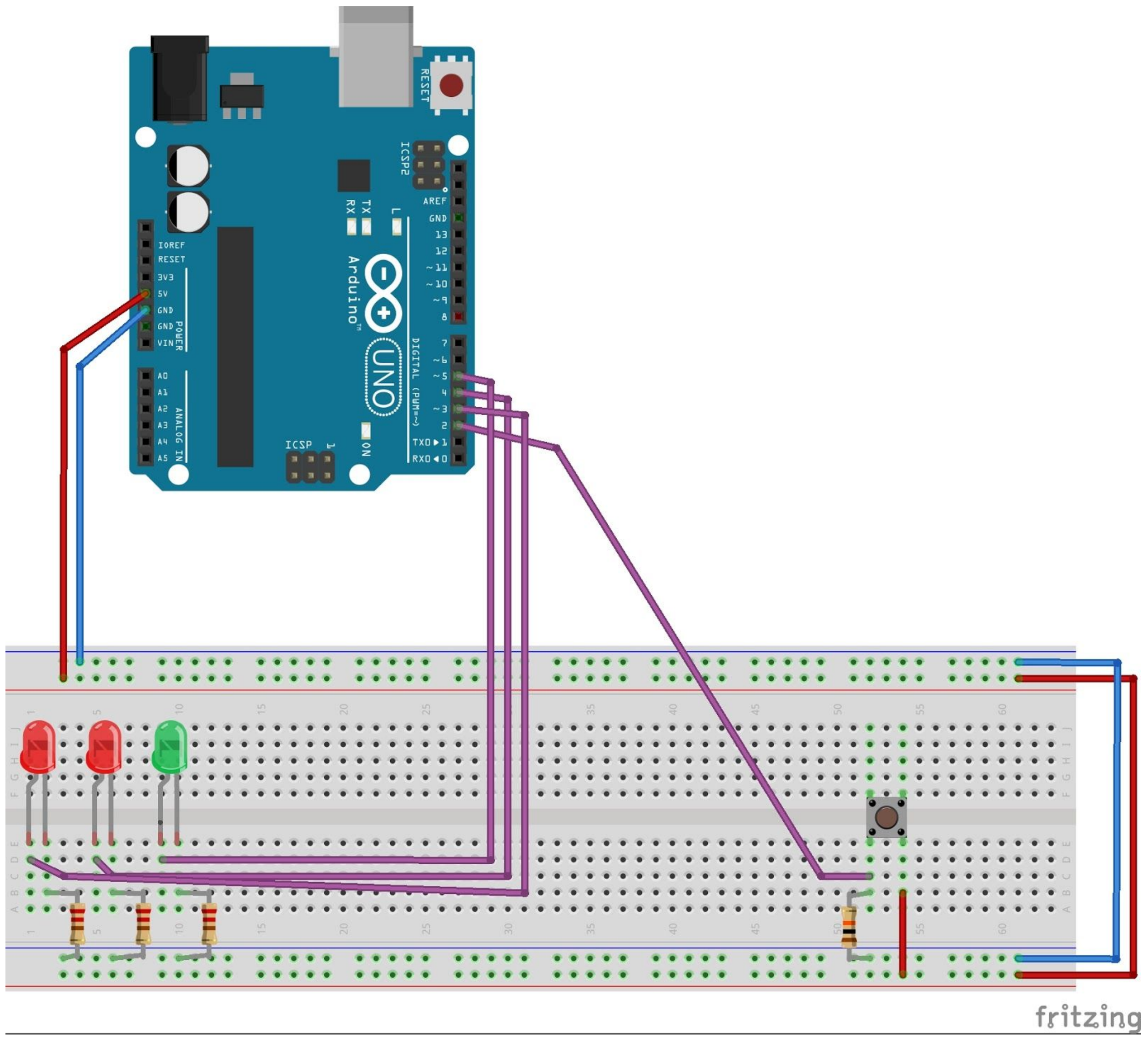
Printing to the Serial Window

On our spaceships, the systems need time to initialize before the interface can be used.

Unfortunately, our last interface didn't let the user know that it was starting up, so the space team just started mashing all the buttons...

This led to a lot of broken buttons and a lot angry space emails....

- **This time around, we want you to design a soothing startup message that displays on the user interface to let the space team know that the system is starting up.**
- **Then let the user know when the initialization is complete and they can begin mashing buttons.**



```
void loop(){
  switchState = digitalRead(2); // Assigning the variable switchState with the
values read from button 1
  switchStateTwo = digitalRead(8); // Assigning the variable switchState with
the values read from button 2
  laserPower = analogRead(A0);

  Serial.begin(9600); // This command allows us to print to the serial
terminal

//*******************************************************************
// This section of code uses a while statement to print startup instructions
to the serial monitor.
//*******************************************************************

while (instructions == 1) {
  Serial.print("Powering Up Interface.");
  delay(1000);
  Serial.print("..\n");
  Serial.print("Energyzing Photons.");
  delay(1000);
  Serial.print("..\n");
  Serial.print("Generating Space Haiku.");
  delay(1000);
  Serial.print("..\n\n\n");
  delay(1000);
  Serial.print("___________SPACE HAIKU___________\n");
  delay(500);
  Serial.print("******* Satellite Observe *******\n");
  delay(1000);
  Serial.print("**** Mist Cloud Unexpectedly ****\n");
  delay(1000);
  Serial.print("******** Reactor Flaming ********\n\n");
  delay(1000);
  Serial.print("You Are Now Connected To “Galactic Spaceship Interface”\n");
  delay(500);
  Serial.print("You may now mash the buttons of your spaceship.\n\n");
  delay(500);
  Serial.print("******* UNENCRYPTED SPACE INTERFACE DATA READOUT DISPLAY
MODULE ******\n");
  delay(500);
  instructions = 0; // Stops instructions from printing by making while
statement false.
} // This brace closes the while loop
} // This brace closes the program loop
```

Great job!!! Now the space team can take a moment to relax while the interface boots up.

The next problem we need to tackle is with the console lighting.

LEDs & Conditional Logic Statements

The lights on the control console of our last model just flashed randomly….

Based on the feedback from the customer service team, the space teams thought that the lights were indicating errors, enemy ships, explosions, etc…. And they kind of kept freaking out.

EPL Corporate has decided that they want to implement a console lighting system that uses “Conditional Logic”... Whatever that means…

- **Can you update the flight computer to run a console lighting test when one of the buttons is pushed?**
- **It would be great if you could also display a message to the interface window to let the user know a test is happening.**
- **Add all of this code in the loop() below the other code you have written.**

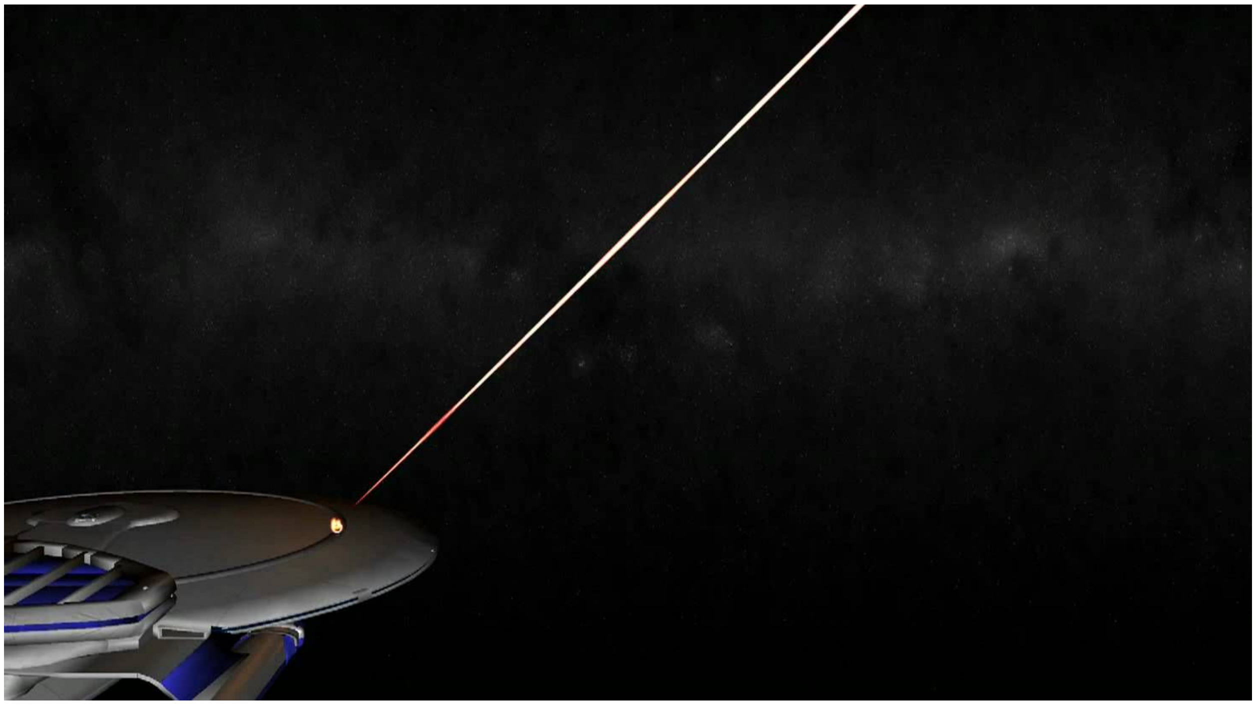
```
//**************************************************************************
// This section of code uses an if-else statement to read button 1. When the
button is pressed, it turns on the lights.
//**************************************************************************

if (switchState == LOW) { // the button is not pressed
  digitalWrite(3, HIGH); // green LED
  digitalWrite(4, LOW); // red LED
  digitalWrite(5, LOW); // red LED
}

else { // the button is pressed
  Serial.print("Performing Console Lighting Test\n");
  digitalWrite(3, LOW);
  digitalWrite(4, LOW);
  digitalWrite(5, HIGH);
  delay(250); // wait for a quarter second then toggle the LEDs
  digitalWrite(4, HIGH);
  digitalWrite(5, LOW);
  delay(250); // wait for a quarter second

} // This brace the if-else statement
} // This brace closes the program loop
```

I'm so glad we hired you. I never would have thought to have the lighting test toggle all the console lights.

Maybe now our robot customer service staff will stop complaining about being overworked…

Crazy idea… What if we think of some way to test the laser weapons to see if they are working properly? That might help the space teams to stop picking fights when their lasers are down…

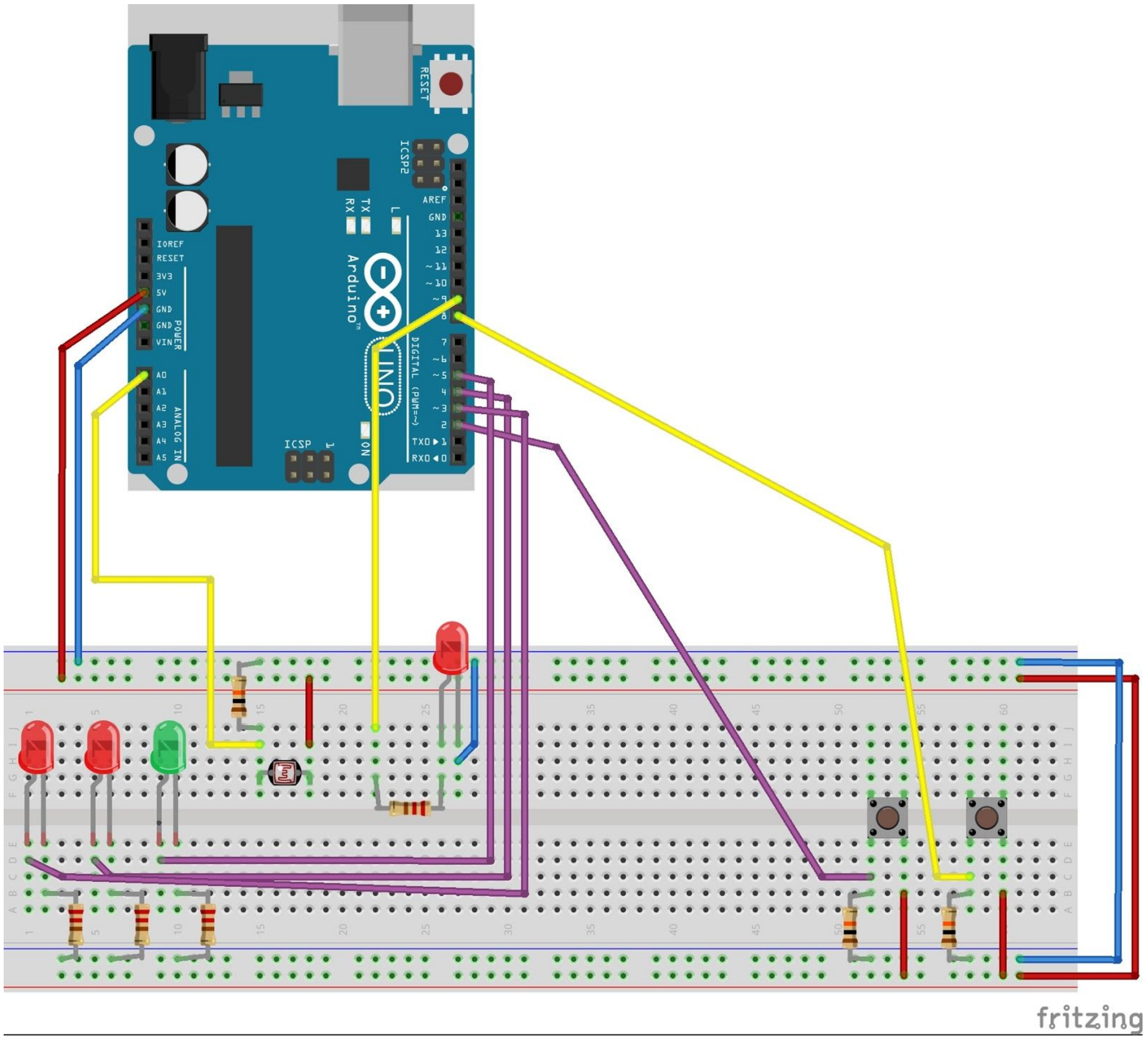
Reading Sensors & Pulse Width Modulation



Okay, so we need some sort of sensor to determine if the laser is working or not…. Have you ever used a sensor before?

If we can get this working I'm confident you'll be getting a hefty bonus. Any idea what you'll buy with all those shiny new Imperial Credits?

- **Read inputs (button presses) from a second button on the console.**
- **Print to the interface window to let the space team know a test is running.**
- **Use an LED to indicate the lasers are firing.**
- **Can you use PWM (pulse width modulation) to control the brightness of the laser?**
- **Make sure the value for the laser beam power prints to the interface window as well.**
- **Add all of this code in the loop() below the other code you have written.**



```
//*****************************************************************************
// This section of code uses an if-else statement to read button 2. When the
button is pressed, it resets to 0. (later it increments a counter, see below)
// The counter is used to cycle through various functions of the spaceship
interface. When done, it returns to the start.
//*****************************************************************************

if (switchStateTwo == LOW) { // the button is not pressed
  digitalWrite(3, HIGH); // green LED
  digitalWrite(4, LOW); // red LED
  digitalWrite(5, LOW); // red LED
}

else { // the button is pressed. We then increment the counter
  buttonCounter ++;
}

//*****************************************************************************
// This section of code uses a if-else statement to output a spaceship
interface funtion based on the value of the counter.
// (later) When the counter reaches three, it is reset to zero in order to
return to the beginning of the cycle.
//*****************************************************************************

// If the counter is equal to 1, fire the laser.
if (buttonCounter == 1) {
  Serial.print("LASER BURST FIRED : Laser Power = ");

  digitalWrite(3, LOW);
  digitalWrite(4, LOW);
  digitalWrite(5, HIGH);
  analogWrite(9, 255); //Using PWM we can change the power from 0 to 255
  Serial.print(laserPower); // prints laser power to display
  Serial.print("\n");
  delay(1000);
  analogWrite(9, LOW);
  buttonCounter = 0; // Reset the counter to 0. In the next step we change
this to "buttonCounter++;" This will add 1 to the counter.

} // This brace closes the if statement
} // This brace closes the program loop
```

I can't believe it works!!! We're both going to be swimming in imperial credits soon.

I think I'm actually getting the hang of this. Could we try to sweeten the deal before we run this up the space pole?

Making Noise & Using For Loops

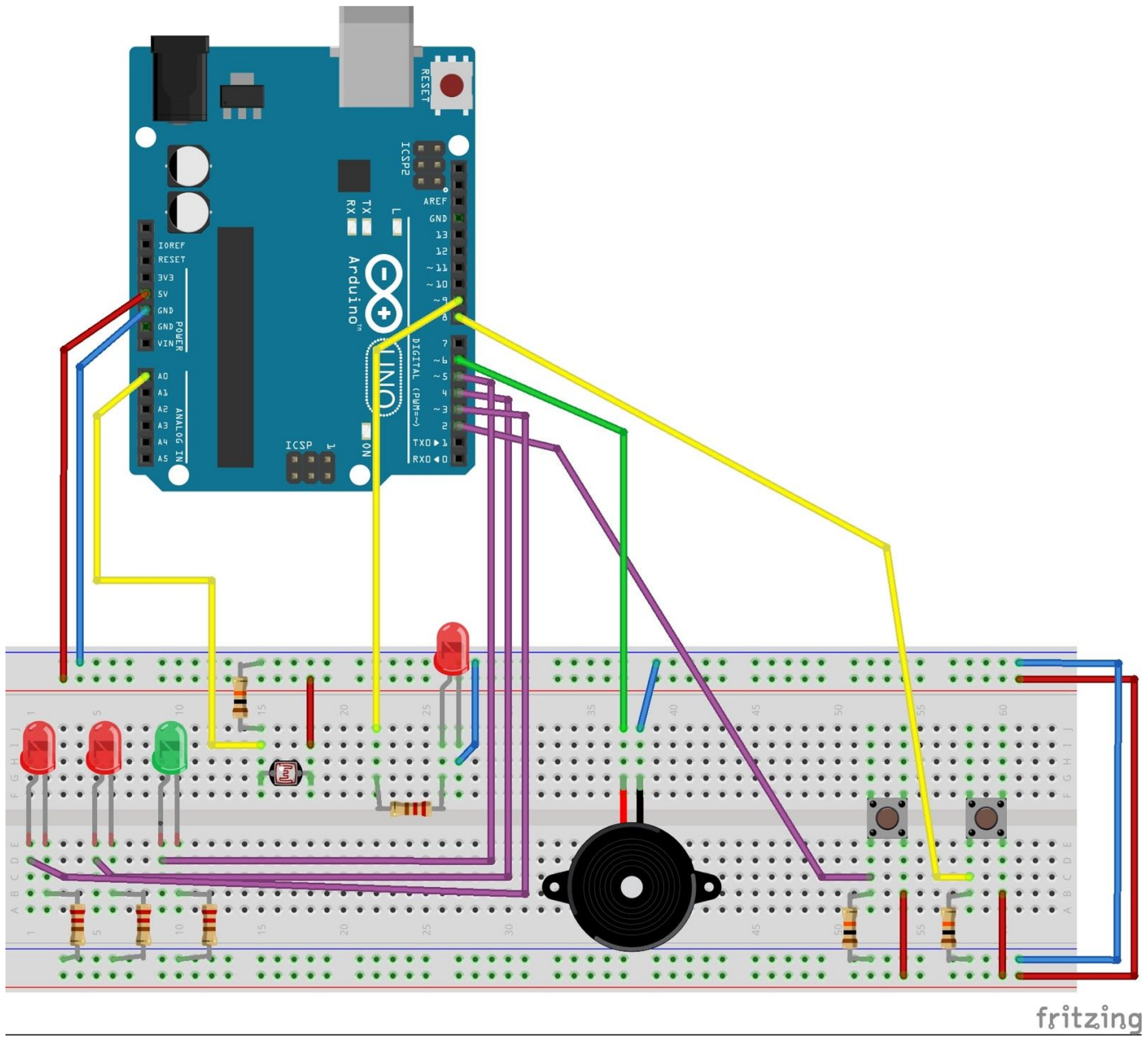
Okay, i’m just thinking out loud here but… What if we use the same button we used for the laser test to test something else on the ship….

What’s that…? Test the Red Alert Alarm…?

Yeah…No, of course we have one of those… We definitely put one of those on the last version...

(Your boss quietly walks over to another engineer and reprimands them for not putting an alarm on the last ship. Then he tells them to immediately get one installed on the new prototype)

- **Change the last command in your program to buttonCounter++;**
- **Then add code to accomplish the following tasks with your program.**
- **When the systems test button is pressed again, print an alert to the interface window.**
- **Use a “for” loop to flash the console lights while the alarm sounds.**
- **Try making the “for” loop run for more iterations to increase or decrease the time the alarm sounds.**
- **Then, turn off the alarm and reset the counter to zero.**


RESET
ICSP2
AREF
GND
13
12
~11
~10
~9
8
7
~6
~5
4
~3
2
TX▶1
RX◀0
DIGITAL (PWM=~)
TX
RX
L
Arduino
UNO
ON
ICSP
IOREF
RESET
3V3
5V
GND
GND
VIN
POWER
A0
A1
A2
A3
A4
A5
ANALOG IN
fritzing

NOTE: Make sure to change the buttonCounter = 0; command above to buttonCounter++;

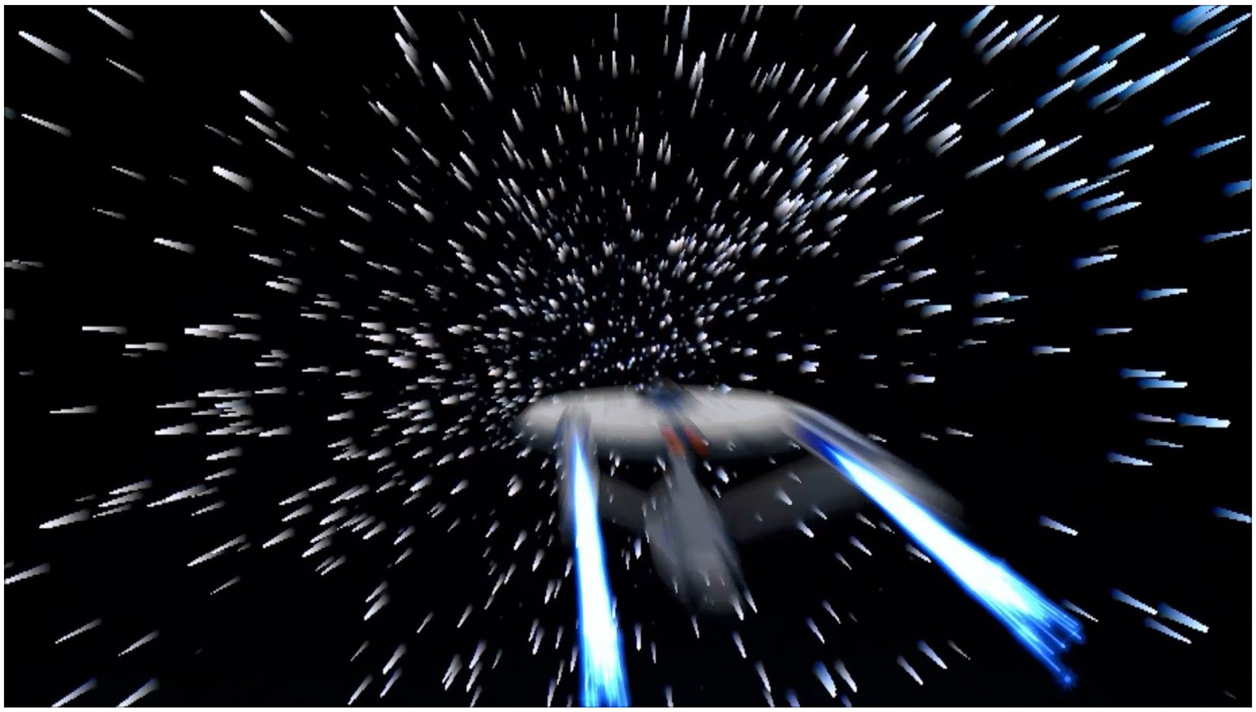
```
// If the button is pressed again, it will equal 3 and sound the alarm.
 else if (buttonCounter == 3) {
  digitalWrite(3, LOW);
  digitalWrite(4, HIGH);
  digitalWrite(5, LOW);
  digitalWrite(6, HIGH);
  Serial.print("Sounding Annoying RED ALERT Alarm\n");

    // This for loop will run 5 times. If you want the alarm and light to run
longer you can increase the number from 15, or decrease it to shorten the
alarm.

    for (int i = 0; i < 5; i++) {
        delay(100);
        digitalWrite(4, HIGH);
        delay(100);
        digitalWrite(4, LOW);
        delay(100);
        digitalWrite(4, HIGH);
        i++;
    }

  digitalWrite(6, LOW);
  buttonCounter = 0; // Reset the counter to 0. In the next step we change
this to "buttonCounter++;" This will add 1 to the counter.

} // This brace closes the if-else statement
} // This brace closes the program loop
```

Yep… *Cough* Cough*... That Red Alert Alarm works just like our last one did….

Nothing new to see here…

Anyways…

Controlling a Servo Motor



Well...I have good news and bad news…

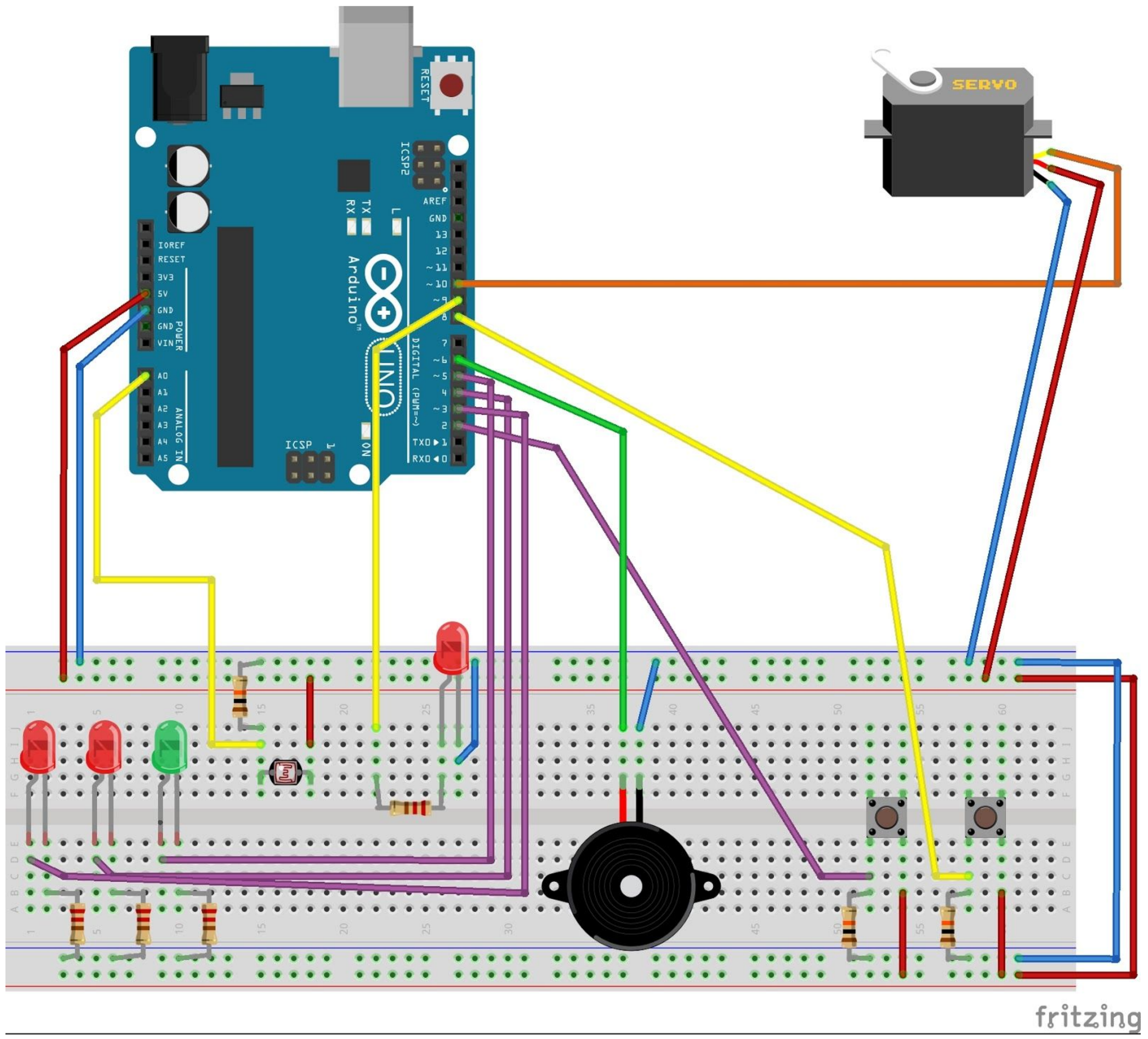
The good news is… I've been approved for that bonus and a promotion!!!

Oh… yeah… They gave you a promotion and bonus too!!! ...Honestly, I couldn't have done it without you.

The bad news is… They want us to squeeze in one more test on the systems test button….

It turns out that the space teams keep trying to escape battles but their warp engines are offline. Can you figure out a way to warm up the engines and run a test on them?

- **Change the last command in your program to buttonCounter++;**
- **Then add code to accomplish the following tasks with your program.**
- **When the systems test button is pressed again, print a message to tell the space team that the engines are warming up.**
- **Use that fancy "for" loop you learned to cycle the engines from 0 to 180 degrees.**
- **It would be great if you could blink the console lights while this happens.**
- **Then test fire the engines by returning them to 0 degrees.**
- **Make sure to reset your counter to zero.**


SERVO
RESET
ICSP2
AREF
GND
13
12
~11
~10
~9
8
7
~6
~5
4
~3
2
TX▶1
RX◀0
DIGITAL (PWM=~)
IOREF
RESET
3V3
5V
GND
GND
VIN
POWER
A0
A1
A2
A3
A4
A5
ANALOG IN
Arduino
UNO
ICSP
ON
TX
RX
L
fritzing

NOTE: Make sure to change the buttonCounter = 0; command above to buttonCounter++;

```
// If the counter is equal to 5, the button has been pressed 3 times. Engage
the engines.

else if (buttonCounter == 5) {
  Serial.print("Warming Up Warp Engines - ");

  for (enginePosition = 0; enginePosition <= 180; enginePosition += 5) { //
goes from 0 degrees to 180 degrees in steps of 5 degrees
    galacticEngine.write(enginePosition); // tell servo to go to position
stored in variable

    // Blink Lights during Engine Test
    digitalWrite(3, LOW);
    digitalWrite(4, LOW);
    digitalWrite(5, HIGH);
    delay(35);
    digitalWrite(3, LOW);
    digitalWrite(4, HIGH);
    digitalWrite(5, HIGH);
    delay(35);
    digitalWrite(3, HIGH);
    digitalWrite(4, HIGH);
    digitalWrite(5, HIGH);
    delay(35);
    digitalWrite(3, LOW);
    digitalWrite(4, LOW);
    digitalWrite(5, LOW);

  }
  // Return the servo to the start position.

  Serial.print("TEST FIRE WARP ENGINES\n");
   galacticEngine.write(0);  // tell servo to go back to degree 0.
   delay(15);  // waits 15ms for the servo to reach the position
   buttonCounter = 0;  // Reset buttonCounter to 0
  }


} // This brace closes the loop
```

Nicely done!!! We've deposited your bonus Imperial credits to your ImperialCoin wallet.

In addition, I'm proud to promote you to Intermediate Flight Controller (Arduino) Engineer!

We're all having astronaut ice cream and dehydrated cake in the break room to celebrate! Better hurry up before the customer service robots eat it all.