Introduction to

Wearable Electronics
Wearable electronics refers to electronic devices that a person wears on their person usually on a daily basis and usually integrated into clothing. Wearable devices come in many forms and levels of complexity, but at their most basic, they consist of the four main elements that make up most electronics. These are:

**The controller.** This is most often a microcontroller based development board (like the Arduino) which is attached to various peripherals, and is loaded with a program which determines how peripherals will interact.

**Peripherals.** These are things that either turn some action into INPUT for the controller, or something that takes OUTPUT from the controller and does something. Peripherals include things like: sensors, buttons, motors, lights, etc.

**Power.** All electronics needs a power source in order to run. For a lot of projects, you can use a fixed source of power, such as a computer USB port or a wall plug, but for wearable projects, power almost always HAS to come from batteries.

**The program.** This is written by you, the designer, to load onto the controller, so that the peripherals will know how to interact.

If you have any questions about this, or anything electronics, feel free to email me at: ben@nestermix.com
HARDWARE CONTROLLERS

The easiest way to get your project underway as a novice is to use an Arduino compatible development board. These boards are simply a microcontroller (the brains) on a board with all the other electronics it needs to operate. Being 'Arduino compatible' means that you are able to use the existing Arduino code libraries and examples to program your project, which makes programming infinitely easier.

There are a few fundamentals of microcontroller theory that need to be covered to help you understand what is going on between your controller and peripherals.

**Logic Level**
The 'logic' level of a board refers to the voltage at which the board considers something 'on'. This voltage is almost always the voltage that you run your circuit at. The most common logic voltages are 3.3V and 5V (your controller board should indicate its logic voltage). Take care when choosing peripherals because plugging something into a higher voltage than it can handle will destroy things. When something is at the logic level voltage, it is called HIGH, and when it is at 0V (or ground voltage) it is considered LOW.

**Signal Type**
There are two kinds of signal: DIGITAL and ANALOG. Digital signals are those that are only capable of HIGH or LOW voltages. Analog signals allow for voltages to be anywhere between HIGH or LOW. ANALOG signals transmit data by sending waves of voltages.

Reading digital values from a sensor gives patterns of HIGH and LOW representing the sensor values.

Reading from an analog sensor gives some voltage value between LOW and HIGH, that value being the sensor reading.

**Controller Choice**
There is a huge range of options out there for controlling your projects. They all vary in what they offer, so choice usually comes down to what your project needs. Each microcontroller has it's own set of features. These features determine how many, and what, things you can connect to it.

**Features**
The features of a microcontroller are broken out to ‘pins’ on the chip, and these pins are further broken out to ‘headers’ on the edge of the control board. These headers allow you to plug things into them to build up your project. Note that each 'pin' usually has several features that it can provide (but only one feature at a time usually).

The most Common Microcontroller Features include:

**Digital I/O (Input/Output)**
This is the most basic of features. You can make a pin an INPUT, in which case it will allow you (in your program) to detect if something switches the pin 'on' or 'off'. Inversely, if you make a pin an OUTPUT, you can tell your program to turn a pin 'on' or 'off', causing whatever you have attached to turn on or off, i.e. an attached LED will light up, or turn off as the pin outputs HIGH or LOW.
HARDWARE > Controller

PWM - Standing for 'Pulse Width Modulation', this feature toggles a digital pin HIGH and LOW at a fast pace producing what looks like an analog signal (effectively analog output).

ADC - 'Analog to Digital Converter'. A microcontroller uses ADC to look at an analog voltage and turn it into digital data that it can understand and act upon. (this is used to INPUT analog signals)

Communication.
Communication between microcontrollers, or them and compatible peripherals, can be achieved by 'talking' to each other with ones and zeros (HIGHs and LOWs). The different ways of structuring this speech are called 'communication protocols'. Many microcontrollers have built in decoders for several of these protocols, the most popular being USART/UART, SPI, I2C, and USB. Being standardised allows for devices to be compatible by simply following one of the above standards. Devices don't HAVE to follow a known protocol. If they do not, they will have to provide documentation on how to interface with it.

Some good Arduino compatible boards (for wearable electronics and otherwise)

**Teensy (3.3V Logic)**
The teensy is a really powerful board for the price. It cost about $20 per board and is very small (hence the name). It operates at a speed 5 times faster than most Arduino boards, and is completely Arduino compatible (after installing the teensy software to program it). It boasts touch pins (for capacitive touch sensing), as well as enough PWM and ADC pins for you to do almost anything. It also has a DAC (Digital to analog converter) which is capable of generating audio (but that is quite advanced).

**Flora/lilypad (3.3V Logic)**
These are a series of controller boards and peripherals designed specifically for wearable applications. They use conductive thread for wiring, so you don't have to worry about soldering wires. These are effectively standard Arduino boards reshaped to be wearable. As such, they are fully Arduino compatible. The biggest drawback with these is their price. The main board is around $20 and peripherals are $8 to $40.

**Digispark/Trinket (both 3.3V and 5V available)**
These boards are based on a more cost effective microcontroller, and have a limited number of features, but with a cost of $2-$7 (look on eBay for ‘Digispark’), they are by far the cheapest option for Arduino compatible boards. Most projects don't require too many pins, so often you can use these boards instead of their bigger Arduino cousins.
Programming.
The trinket is available from Adafruit and installation and setup instructions can be found on their website here:

https://learn.adafruit.com/introducing-trinket/setting-up-with-arduino-ide

Programming with the trinket is very similar to programming any other Arduino. The main difference is that you must press the reset button on the board to put it into programming mode before you upload from the Arduino software.

The following reference card shows the pins of the Trinket and also the common arduino commands to use them.
Peripherals, which include all ‘sensors and actuators’, are things that you connect to your board that allow you to receive input from outside the device (temperature, press of a button, light level, etc) or to output to things outside the device (turn on light, make a noise, move a motor).

Sensors will either connect to the ADC of a microcontroller, sending sensor data as an analog signal, or connect and send data through some digital communication protocol. An obvious exception to this is a pushbutton which will toggle a pin HIGH or LOW when pressed.

Actuators and other output peripherals more often need some more sophisticated control than simply toggling a pin, and more often use a more complicated communication protocol or control circuitry.

One of the simplest output peripherals is an LED. It is possible to switch an LED on and off with a digital output. Using an analog output, you are able to fade/dim that same LED.

When wiring almost any peripheral, you will have to make sure that you give it power (Voltage and Ground) and make sure that you connect it to pins that are capable of reading/writing an appropriate signal.
Colorful Lights  (RGB LED Strips)

These specific strips of colorful lights are both easy to use and AWESOME to look at. The strips use an LED containing Red, Green, and Blue colors, which have an integrated controller. That means that instead of controlling each LED separately, you can put whole strings of them together and control all of them with one controller pin. Put simply, they have almost no extra requirements to run. Simply give it 5V, ground, and any pin, load the Arduino library, and there you are! You can even cut the strips up and so long as you connect the ‘data in’ pin to the controller, you can use them still!

Software
The Arduino library used to control these is the ‘Neopixel’ library and you can find examples and the library on the Adafruit website.

Purchasing
These strips can be purchased from Adafruit, who sell them as ‘neopixels’. OR you can search eBay for ‘WS2812’ for strips or single LEDs (make sure you buy them with boards already soldered).

Sound Sensor  (Microphone with Amplifier)

The sound sensor allows you to detect noises. Most microcontrollers aren’t powerful enough to interpret sound, but this sensor will allow you to detect when a loud noise occurs. The microphone by itself outputs an analog signal representing what it hears. This signal is very quiet, so most often it is attached to an amplifier to make it louder (amplifiers can be applied to any ‘quiet signal’).

Software
The output from this sensor is an analog signal. Therefore you can use “analogRead()” in the Arduino software to read sensor data.

Purchasing
The pictured sound sensor is available from Sparkfun: https://www.sparkfun.com/products/9964

This sensor can also be purchased from eBay for quite cheap. (Search ‘electret microphone’ and make sure it is attached to a circuit board, ie. has an amplifier circuit).
**Light Sensor** (Light Dependent Resistor)

This light sensor is basically a resistor that changes resistance based on the intensity of the light it sees. Using a common circuit called a ‘voltage divider’ we can use this Light Dependent Resistor (LDR), and a normal resistor to read the analog voltage changes through an ADC pin. The LDR will have a specific resistance in full dark and a much lower resistance in intense light.

**Software**

The output from this sensor is an analog signal. Therefore you can use “analogRead()” in the Arduino software to read sensor data. The analog reading will not be 0-1023 as with most analog sensors. Due to the voltage divider, the minimum value will be:

\[ V = \frac{R2}{R1 + R2} \]

where:
- \( V \) = voltage into ADC.
- \( R2 \) = Resistance of LDR (changes with light)
- \( R1 \) = Resistance of normal resistor

**Purchasing**

These are very common and can be purchased from ‘JayCar’, ‘Element14’, ‘Sparkfun’, ‘eBay’; almost any store with electronic components, really.

---

**Touch Sensor** (Capacitive Touch Button)

Touch sensing is very simple in theory: As some big fleshy thing approaches (and touches) a lump of metal, the voltage will change slightly. Thus, with some fancy software and an analog input, we can detect when a finger gets really close to a metal plate.

**Software**

Though the circuit is very simple, the software is quite complicated. But, in true Arduino fashion, there is a library for that! The ‘TinyTouchLib’ is a library written specifically for boards using the Attiny microcontroller (like the trinket and digispark) which uses TWO ADC PINS (one connected to the touch plate, one which must be left free). Note: this library doesn’t work when running the Trinket at 16MHz.

**Purchasing**

This sensor is almost free! all you need is one 1kOhm resistor and something metal! Join them as shown in the image on the right. I like to use copper tape to make the electrodes as it is easy to work with and sticky. Copper tape can be purchased from ‘http://littlebirdelectronics.com.au/’ or eBay.
Powering your project can be one of the more difficult parts of it. You want it to be able to run forever, but you also don’t want it to weigh a ton. What is the best type of battery to use? How big does it need to be?

Well, for wearable electronics, the best option is to use lithium batteries. They are rechargable and have the best power to weight ratio. They are also 3.7V when fully charged, so they are perfect for 3.3V boards (only through a power regulator board, though - the Trinket has this if you connect the battery to the BAT+ pin). To charge lithium batteries, you MUST use a specific lithium battery charger. You will damage the battery if you don’t, and it can explode.

Another good power source for small, low power electronics is the coin cell battery. They are small and cheap, though they are not rechargable.
This project aims to help you build up a wicked cool LED armband. It has six LEDs that you can program to be any color, and do anything you like. It also has a capacitive button and light sensor. Example code is included, but you can change it to do whatever you want it to do! You can use this project as a starting point and add other sensors to it to do different things. Or come up with your own wearable project! The sky is the limit!

If you have something you want to add/do, but are unsure how to do it, feel free to email me at: ben@nestermix.com

PARTS:
1x Adafruit Trinket (5V version)
1x Battery Pack (5.5-16V)
1x Copper Tape (approx 15x15mm)
1x 1kOhm Resistor
1x Light Sensor
6x Neopixel (WS2812) LEDs in a strip
1x piece of fabric for base. (approx 40mm wide and long enough to go around wrist, minimum 15cm)
1x piece of fabric to cover base piece.
1x needle and thread.
1x Insulating tape
2x Clasps (mechanical or velcro)
Step One: Layout the design

Place all of the pieces of the armband in approximately the places where you would like them to sit (image on the right shows an example of this). Keep in mind where the LEDs will sit when the band is on (the desired position is the top of the wrist). Ideally, the battery should sit under the wrist so that gravity is less likely to pull the armband sideways. Make sure to keep a flap of the cloth long enough to fold up over the battery.

Step Two: Build the button

Decide where you want the touch button to go (example places it above the center of the LED strip as shown in photo on previous page) and place one copper tape strip horizontal at that point. Then take the 1kOhm Resistor (Brown, Black, Red rings) and coil one end of it an place on the tape. Use the remaining strips of copper tape to stick the coil down. Then take one of the included wires and twist it to the free end of the resistor. the other end of the wire goes to pin two of the Trinket.

Step Three: Wire up the circuit

Before wiring the battery, make sure that the switch is in the off position (should come in the off position - ask if you are unsure). To wire up the circuit, insert the wires from the peripherals through the holes of the controller as shown in the diagram on the previous page. Once a wire is inserted, fold it back on itself and then wrap tape around the folded wire to hold it in place and insulate it from touching its neighbours. Note that the 5V and Ground holes will have several wires through them.

Step Four: Initial Test

At this point, it should be all electrically connected. Perform an initial test by flipping the switch on the battery. If nothing explodes, awesome! If two LEDs (red and green) light up, even better! If it doesn't light up, make sure that the battery is connected correctly and no voltage wires (generally red) and ground wires (generally black) are touching. If it is all wired correctly, but still no lights, check that it isn't the power source by trying another battery.

Step Five: Stitch it all down

Once it has been wired up (double check!) it is time to stitch it all down!
1. Stitch down the LEDs: The strips have small holes between the LEDs which you can use. just make sure that you don't destroy the connection.
2. Stitch down the Trinket using the large mounting holes.
3. Attach one side of the clasp on the underside of the cloth under the clasp (further out will yeild better results - see image on the right).
4. Stitch the flap over the battery. This is so that replacing them is possible.
Step Six: System Test

Now that it should be all electrically done and secure, it is time to test it out and make sure all the peripherals are working. When testing, the device will draw power from the USB port, and you should not need to use the battery at all. It is best not to turn on the battery while it is plugged into the USB.

Using the Adafruit's customised Arduino IDE, load the below code onto the device to test the various systems.

Step Seven: Next...

Once you have verified that it all works, use the materials provided to cover the electronics and generally pretty-up your armband. This will include putting the other side of the clasp on and fitting it comfortably to your wrist.

Step Eight: Programming!!!

You have built the device. Now, you have to program it to get the functionality we want.

Programming goals:

- Make the led strip 'bounce' an led of a single color back and forward.
- Make the touch button change the bouncing LED's color.
- Make the bouncing LED turn off when the light sensor is covered.

(Code for this can be found over the page. But you should try it yourself first.)

//Test LED Strip

/*
To run the test on the LED strip, from the Arduino software:

Go to 'File > Examples > Adafruit_Neopixel' and click 'strandtest'

change the line: '#define PIN 6' to '#define PIN 0'
(set the led strip pin to pin 0)

then, change the line:
'Adafruit_NeoPixel strip = Adafruit_NeoPixel(60, PIN, NEO_GRB + NEO_KHZ800);' to
'Adafruit_NeoPixel strip = Adafruit_NeoPixel(6, PIN, NEO_GRB + NEO_KHZ800);'
(change the strip to account for 6 LEDs, rather than 60)
*/

//Test Capacitive Button

//Touching the button should light up the little red onboard led

//include touch library
#include <TinyTouchLib.h>

void setup() {
  pinMode(1, OUTPUT); // red LED
  tinytouch_init(); // initialise touch
}

void loop() {

  //read touch sensor.
  uint8_t touchstate = tinytouch_sense();

  //if press detected, turn on LED, otherwise, turn off.
  if (touchstate==tt_push) {
    digitalWrite(1, HIGH);
  } else{
    digitalWrite(1, LOW);
  }
}

//Test Light Sensor

//Covering the Light sensor should turn on the onboard red LED.

void setup() {
  pinMode(1, OUTPUT);
}

void loop() {
  analogWrite(1, analogRead(3)/4);
}
/LED Armband Code

#include <Adafruit_NeoPixel.h>
#include <TinyTouchLib.h>

#define stripPIN 0 //define the Led Strip pin
#define lightSensorPin 3 //define the light sensor pin

//variable used to keep track of the moving led.
long time = 0;
int leddelay = 50;
int atled = 0;
int leddir = 1;

//initialise the LED strip
Adafruit_NeoPixel strip = Adafruit_NeoPixel(6, stripPIN, NEO_GRB + NEO_KHZ800);

//counter that tracks what color we are on.
uint8_t color = 0;

void setup() {
  strip.begin(); // begin strip function
  strip.setBrightness(10); // dim the LEDs a bit because they are blinding.
  strip.show(); // Initialize all pixels to ‘off’

  //set pin one which has the onboard red LED to output.
  pinMode(1, OUTPUT);

  //initialise the touch library
  tinytouch_init();
}

void loop() {

  //fade the onboard red LED proportiately to the light sensor reading
  // (analog write wants 0-255, analog read gives 0-1023, so divide by 4.)
  analogWrite(1, analogRead(lightSensorPin)/4);

  /*when using the ‘delay(value)’ function, it halts the program at that point. doing that would cause the
   touch button and everything else to stop working. so, instead we look at the time the device has been
   turned on, add some delay, then call that ‘time’. then, at the beginning of each loop, we check to see if that
   time has been reached. if it has, we move the led one position. we then check to see if we have reached the
   end of the strip. if we have, we flip the direction.*/
  if(millis() > time){
    atled += leddir;
    if(atled >=5 || atled <=0){
      leddir *= -1;
    }
    time = millis() + leddelay;
  }
//use the led function defined below to set the led we are currently at to be blue.
led(atled, 0,0,255);

//read the touch button state (see if it is being pushed)
uint8_t touchstate = tinytouch_sense();

//if we detect a button push, increment the color.
if (touchstate==tt_push) {
  color ++;
}

make the led a certain color based on the color variable.
if(color == 0){
  led(atled, 0,0,255);
} else if(color == 1){
  led(atled, 0,255,255);
} else if(color == 2){
  led(atled, 0,255,0);
} else if(color == 3){
  led(atled, 255,255,0);
} else if(color == 4){
  led(atled, 255,0,0);
} else if(color == 5){
  led(atled, 255,0,255);
} else if(color == 6){
  led(atled, 255,255,255);
} else{
  color = 0;
}
strip.show(); // this is needed to make the LED colors show.

//function to set the led pattern. turns all LEDS off, then turns the input one on.
void led(int n, int r, int g, int b){

  strip.setPixelColor(0, 0,0,0);
  strip.setPixelColor(1, 0,0,0);
  strip.setPixelColor(2, 0,0,0);
  strip.setPixelColor(3, 0,0,0);
  strip.setPixelColor(4, 0,0,0);
  strip.setPixelColor(5, 0,0,0);

  strip.setPixelColor(n, r,g,b);
}