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

  /*
This code uses the Pulse Sensor Amped by Joel Murphy
and Yury Gitman
    www.pulsesensor.com
    >>> Pulse Sensor purple wire goes to Analog Pin 0
<<<
Pulse Sensor sample aquisition and processing happens
in the background via Timer 2 interrupt. 2mS sample
rate.
PWM on pins 3 and 11 will not work when using this
code, because we are using Timer 2!
The following variables are automatically updated:
Signal :    int that holds the analog signal data
straight from the sensor. updated every 2mS.
IBI  :      int that holds the time interval between
beats. 2mS resolution.
BPM  :      int that holds the heart rate value,
derived every beat, from averaging previous 10 IBI
values.
QS  :       boolean that is made true whenever Pulse is
found and BPM is updated. User must reset.
Pulse :     boolean that is true when a heartbeat is
sensed then false in time with pin13 LED going out.

NOTE: This code works with Arduino UNO or Arduino PRO
or Arduino Pro Mini 5V or any Arduino running with an
ATmega328 and 16MHz clock. This will disable PWM output
on pin 3 and 11. Also, it will disable the tone()
command.
```

```
All the work to find the heartbeat and determine the
heartrate happens in the code below.
For using the Pulse Sensor code, see the link below for
a code walkthrough:
http://pulsesensor.myshopify.com/pages/pulse-sensor-
amped-arduino-v1dot1

Code Version 02 by Joel Murphy & Yury Gitman  Fall 2012
*/

volatile int rate[10];                    // used to
hold last ten IBI values
volatile unsigned long sampleCounter = 0;          //
used to determine pulse timing
volatile unsigned long lastBeatTime = 0;           //
used to find the inter beat interval
volatile int P =512;                      // used to
find peak in pulse wave
volatile int T = 512;                     // used to
find trough in pulse wave
volatile int thresh = 512;                // used to
find instant moment of heart beat
volatile int amp = 100;                   // used to
hold amplitude of pulse waveform
volatile boolean firstBeat = true;        // used to
seed rate array so we startup with reasonable BPM
volatile boolean secondBeat = true;       // used to
seed rate array so we startup with reasonable BPM

void interruptSetup(){
  // Initializes Timer2 to throw an interrupt every
2mS.
```

```
  TCCR2A = 0x02;     // DISABLE PWM ON DIGITAL PINS 3
AND 11, AND GO INTO CTC MODE
  TCCR2B = 0x06;     // DON'T FORCE COMPARE, 256
PRESCALER
  OCR2A = 0X7C;      // SET THE TOP OF THE COUNT TO 124
FOR 500Hz SAMPLE RATE
  TIMSK2 = 0x02;     // ENABLE INTERRUPT ON MATCH
BETWEEN TIMER2 AND OCR2A
  sei();             // MAKE SURE GLOBAL INTERRUPTS ARE
ENABLED
}

// THIS IS THE TIMER 2 INTERRUPT SERVICE ROUTINE.
// Timer 2 makes sure that we take a reading every 2
miliseconds
ISR(TIMER2_COMPA_vect){                              //
triggered when Timer2 counts to 124
    cli();                                           //
disable interrupts while we do this
    Signal = analogRead(pulsePin);                   // read
the Pulse Sensor
    sampleCounter += 2;                              // keep
track of the time in mS with this variable
    int N = sampleCounter - lastBeatTime;            //
monitor the time since the last beat to avoid noise

//  find the peak and trough of the pulse wave
    if(Signal < thresh && N > (IBI/5)*3){            //
avoid dichrotic noise by waiting 3/5 of last IBI
        if (Signal < T){                             // T is
the trough
            T = Signal;                              // keep
track of lowest point in pulse wave
```

```
        }
      }

    if(Signal > thresh && Signal > P){          //
thresh condition helps avoid noise
       P = Signal;                             // P is
the peak
      }                                        // keep
track of highest point in pulse wave

  //  NOW IT'S TIME TO LOOK FOR THE HEART BEAT
  // signal surges up in value every time there is a
pulse
if (N > 250){                                  //
avoid high frequency noise
  if ( (Signal > thresh) && (Pulse == false) && (N >
(IBI/5)*3) ){
    Pulse = true;                              // set
the Pulse flag when we think there is a pulse
    digitalWrite(blinkPin,HIGH);               // turn
on pin 13 LED
    IBI = sampleCounter - lastBeatTime;        //
measure time between beats in mS
    lastBeatTime = sampleCounter;              // keep
track of time for next pulse

        if(firstBeat){                         // if
it's the first time we found a beat, if firstBeat ==
TRUE
            firstBeat = false;                 //
clear firstBeat flag
            return;                            // IBI
value is unreliable so discard it
```

```
            }
         if(secondBeat){                          // if
this is the second beat, if secondBeat == TRUE
            secondBeat = false;                   //
clear secondBeat flag
               for(int i=0; i<=9; i++){           // seed
the running total to get a realisitic BPM at startup
                  rate[i] = IBI;

                  }
            }

    // keep a running total of the last 10 IBI values
    word runningTotal = 0;                     // clear
the runningTotal variable

    for(int i=0; i<=8; i++){                  // shift
data in the rate array
          rate[i] = rate[i+1];                // and drop
the oldest IBI value
          runningTotal += rate[i];            // add up
the 9 oldest IBI values
        }

    rate[9] = IBI;                            // add the
latest IBI to the rate array
    runningTotal += rate[9];                  // add the
latest IBI to runningTotal
    runningTotal /= 10;                       // average
the last 10 IBI values
    BPM = 60000/runningTotal;                 // how many
beats can fit into a minute? that's BPM!
    QS = true;                                // set
```

```
Quantified Self flag
    // QS FLAG IS NOT CLEARED INSIDE THIS ISR
    }
}

  if (Signal < thresh && Pulse == true){     // when
the values are going down, the beat is over
      digitalWrite(blinkPin,LOW);            // turn
off pin 13 LED
      Pulse = false;                         // reset
the Pulse flag so we can do it again
      amp = P - T;                           // get
amplitude of the pulse wave
      thresh = amp/2 + T;                    // set
thresh at 50% of the amplitude
      P = thresh;                            // reset
these for next time
      T = thresh;
     }

  if (N > 2500){                             // if 2.5
seconds go by without a beat
      thresh = 512;                          // set
thresh default
      P = 512;                               // set P
default
      T = 512;                               // set T
default
      lastBeatTime = sampleCounter;          // bring
the lastBeatTime up to date
      firstBeat = true;                      // set
these to avoid noise
      secondBeat = true;                     // when we
```

```
get the heartbeat back
     }

  sei();                                          // enable
interrupts when youre done!
}// end isr

}

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

  /* Pulse Sensor Amped with NeoPixels

Pulse an arbitrary number of Adafruit NeoPixels based
on a heartbeat sensor

The pulsesensor.com code needs to be in module
interrupt.ino in the sketch directory
   http://pulsesensor.com/pages/pulse-sensor-amped-
arduino-v1dot1
Code also uses the Adafruit NeoPixel library code
discussed at
   https://learn.adafruit.com/adafruit-neopixel-
uberguide

Version 1.0 by Mike Barela for Adafruit Industries,
Fall 2015
*/
#include <Adafruit_NeoPixel.h>    // Library
containing

// Behavior setting variables
```

```
int pulsePin = 0;                 // Pulse Sensor
purple wire connected to analog pin 0
int blinkPin = 13;                // Digital pin to
blink led at each beat
int fadePin  = 5;                 // pin to do fancy
neopixel effects at each beat
int fadeRate = 0;                 // used to fade LED
on with PWM on fadePin

// these variables are volatile because they are used
during the interrupt service routine
volatile int BPM;                   // used to hold the
pulse rate
volatile int Signal;                // holds the
incoming raw data
volatile int IBI = 600;             // holds the time
between beats, the Inter-Beat Interval
volatile boolean Pulse = false;     // true when pulse
wave is high, false when it's low
volatile boolean QS = false;        // becomes true
when Arduoino finds a beat.

// Set up use of NeoPixels
const int NUMPIXELS = 24;           // Put the number
of NeoPixels you are using here
const int BRIGHTNESS = 60;          // Set brightness
of NeoPixels here
Adafruit_NeoPixel strip = Adafruit_NeoPixel(NUMPIXELS,
fadePin, NEO_GRB + NEO_KHZ800);

void setup(){
  pinMode(blinkPin,OUTPUT);         // pin that will
blink to your heartbeat!
```

```
//  Serial.begin(115200);           // Serial output
data for debugging or external use
  strip.begin();
  strip.setBrightness(BRIGHTNESS);
  for (int x=0; x < NUMPIXELS; x++) {  // Initialize
all pixels to 'off'
     strip.setPixelColor(x, strip.Color(0, 0, 0));
  }
  strip.show();                       // Ensure the
pixels are off
  delay(1000);                        // Wait a second
  interruptSetup();                   // sets up to read
Pulse Sensor signal every 2mS
}

void loop(){
//  sendDataSerial('S', Signal);      // send
Processing the raw Pulse Sensor data
  if (QS == true){                    // Quantified
Self flag is true when arduino finds a heartbeat
     fadeRate = 255;                  // Set 'fadeRate'
Variable to 255 to fade LED with pulse
//     sendDataSerial('B',BPM);       // send heart
rate with a 'B' prefix
//     sendDataSerial('Q',IBI);       // send time
between beats with a 'Q' prefix
     QS = false;                      // reset the
Quantified Self flag for next time
  }
  ledFadeToBeat();                    // Routine that
fades color intensity to the beat
  delay(20);                          //  take a break
}
```

```
void ledFadeToBeat() {
    fadeRate -= 15;                         // Set LED
fade value
    fadeRate = constrain(fadeRate,0,255);   // Keep LED
fade value from going into negative numbers
    setStrip(fadeRate);                     // Write
the value to the NeoPixels
//    sendDataSerial('R',fadeRate);
}

void sendDataSerial(char symbol, int data ) {
//    Serial.print(symbol);                // symbol
prefix tells Processing what type of data is coming
//    Serial.println(data);                // the data
to send culminating in a carriage return
}

void setStrip(int r) {     // Set the strip to one
color intensity (red)
   int g = 0;              // Green is set to zero (for
non-red colors, change this)
   int b = 0;              // Blue is set to zero (for
non-red colors, change this)
   for (int x=0; x < NUMPIXELS; x++) {
      strip.setPixelColor(x, strip.Color(r, g, b));
   }
   strip.show();
}

}
```