```
/*
  AnalogReadSerial
  Reads an analog input on pin 0, prints the result to the serial monitor.
*/

// the setup routine runs once when you press reset:
void setup() {
  Serial.begin(9600);
}

void loop() {
  int sensorValue = analogRead(A0);
  Serial.println(sensorValue);
  delay(1);        // delay in between reads for stability
}

/*
  Blink
  Turns on an LED on for one second, then off for one second, repeatedly.
 */

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

// the loop function runs over and over again forever
void loop() {
  digitalWrite(13, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);              // wait for a second
  digitalWrite(13, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);              // wait for a second
}
/*
  DigitalReadSerial
 Reads a digital input on pin 2, prints the result to the serial monitor
 */
int pushButton = 2;

void setup() {
  Serial.begin(9600);
  pinMode(pushButton, INPUT);
}
void loop() {
  int buttonState = digitalRead(pushButton);
  Serial.println(buttonState);
  delay(1);        // delay in between reads for stability
}
/*
 Fade
This example shows how to fade an LED on pin 9
 using the analogWrite() function.

 The analogWrite() function uses PWM,
 a "~" sign, like ~3, ~5, ~6, ~9, ~10 and ~11.
 */
int led = 9;           // the PWM pin the LED is attached to
int brightness = 0;    // how bright the LED is
int fadeAmount = 5;    // how many points to fade the LED by

void setup() {
  // declare pin 9 to be an output:
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  // set the brightness of pin 9:
  analogWrite(led, brightness);
```

```
  // change the brightness for next time through the loop:
  brightness = brightness + fadeAmount;
  // reverse the direction of the fading at the ends of the fade:
  if (brightness == 0 || brightness == 255) {
    fadeAmount = -fadeAmount ;
  }
  // wait for 30 milliseconds to see the dimming effect
  delay(30);
}
/*
  ReadAnalogVoltage
  Reads an analog input on pin 0, converts it to voltage, and prints the result to the
  serial monitor.
*/

void setup() {
  Serial.begin(9600);
}

// the loop routine runs over and over again forever:
void loop() {
  // read the input on analog pin 0:
  int sensorValue = analogRead(A0);
  // Convert the analog reading (which goes from 0 - 1023) to a voltage (0 - 5V):
  float voltage = sensorValue * (5.0 / 1023.0);
  // print out the value you read:
  Serial.println(voltage);
}
/* Blink without Delay

 Turns on and off a light emitting diode (LED) connected to a digital
 pin, without using the delay() function.  This means that other code
 can run at the same time without being interrupted by the LED code.
 */

const int ledPin =  13;      // the number of the LED pin
// Variables will change :
int ledState = LOW;             // ledState used to set the LED
// Generally, you should use "unsigned long" for variables that hold time
// The value will quickly become too large for an int to store
unsigned long previousMillis = 0;        // will store last time LED was updated
// constants won't change :
const long interval = 1000;           // interval at which to blink (milliseconds)

void setup() {
  // set the digital pin as output:
  pinMode(ledPin, OUTPUT);
}

void loop() {
  // here is where you'd put code that needs to be running all the time.
  // check to see if it's time to blink the LED; that is, if the
  // difference between the current time and last time you blinked
  // the LED is bigger than the interval at which you want to
  // blink the LED.
  unsigned long currentMillis = millis();
  if (currentMillis - previousMillis >= interval) {
    // save the last time you blinked the LED
    previousMillis = currentMillis;
    // if the LED is off turn it on and vice-versa:
    if (ledState == LOW) {
      ledState = HIGH;
    } else {
      ledState = LOW;
    }
    // set the LED with the ledState of the variable:
    digitalWrite(ledPin, ledState);
  }
```

```
}
/*
  State change detection (edge detection)

 */

// this constant won't change:
const int  buttonPin = 2;    // the pin that the pushbutton is attached to
const int ledPin = 13;       // the pin that the LED is attached to

// Variables will change:
int buttonPushCounter = 0;   // counter for the number of button presses
int buttonState = 0;         // current state of the button
int lastButtonState = 0;     // previous state of the button

void setup() {
  // initialize the button pin as a input:
  pinMode(buttonPin, INPUT);
  // initialize the LED as an output:
  pinMode(ledPin, OUTPUT);
  // initialize serial communication:
  Serial.begin(9600);
}


void loop() {
  // read the pushbutton input pin:
  buttonState = digitalRead(buttonPin);
  // compare the buttonState to its previous state
  if (buttonState != lastButtonState) {
    // if the state has changed, increment the counter
    if (buttonState == HIGH) {
      // if the current state is HIGH then the button
      // wend from off to on:
      buttonPushCounter++;
      Serial.println("on");
      Serial.print("number of button pushes:  ");
      Serial.println(buttonPushCounter);
    } else {
      // if the current state is LOW then the button
      // wend from on to off:
      Serial.println("off");
    }
    // Delay a little bit to avoid bouncing
    delay(50);
  }
  // save the current state as the last state,
  //for next time through the loop
  lastButtonState = buttonState;


  // turns on the LED every four button pushes by
  // checking the modulo of the button push counter.
  // the modulo function gives you the remainder of
  // the division of two numbers:
  if (buttonPushCounter % 4 == 0) {
    digitalWrite(ledPin, HIGH);
  } else {
    digitalWrite(ledPin, LOW);
  }

}
/*
  Analog input, analog output, serial output

 Reads an analog input pin, maps the result to a range from 0 to 255
 and uses the result to set the pulsewidth modulation (PWM) of an output pin.
 Also prints the results to the serial monitor.
```

```
 */

const int analogInPin = A0;  // Analog input pin that the potentiometer is attached to
const int analogOutPin = 9; // Analog output pin that the LED is attached to

int sensorValue = 0;        // value read from the pot
int outputValue = 0;        // value output to the PWM (analog out)

void setup() {
  // initialize serial communications at 9600 bps:
  Serial.begin(9600);
}

void loop() {
  // read the analog in value:
  sensorValue = analogRead(analogInPin);
  // map it to the range of the analog out:
  outputValue = map(sensorValue, 0, 1023, 0, 255);
  // change the analog out value:
  analogWrite(analogOutPin, outputValue);
  // print the results to the serial monitor:
  Serial.print("sensor = ");
  Serial.print(sensorValue);
  Serial.print("\t output = ");
  Serial.println(outputValue);
  // wait 2 milliseconds before the next loop
  // for the analog-to-digital converter to settle
  // after the last reading:
  delay(2);
}
/*
  Smoothing
AVERAGE*/

const int numReadings = 10;

int readings[numReadings];      // the readings from the analog input
int readIndex = 0;              // the index of the current reading
int total = 0;                  // the running total
int average = 0;                // the average
int inputPin = A0;

void setup() {
  // initialize serial communication with computer:
  Serial.begin(9600);
  // initialize all the readings to 0:
  for (int thisReading = 0; thisReading < numReadings; thisReading++) {
    readings[thisReading] = 0;
  }
}

void loop() {
  // subtract the last reading:
  total = total - readings[readIndex];
  // read from the sensor:
  readings[readIndex] = analogRead(inputPin);
  // add the reading to the total:
  total = total + readings[readIndex];
  // advance to the next position in the array:
  readIndex = readIndex + 1;
  // if we're at the end of the array...
  if (readIndex >= numReadings) {
    // ...wrap around to the beginning:
    readIndex = 0;
  }
  // calculate the average:
  average = total / numReadings;
  // send it to the computer as ASCII digits
```

```
  Serial.println(average);
  delay(1);        // delay in between reads for stability
}
/*
  Switch statement
*/
void loop() {
  // read the sensor:
  int sensorReading = analogRead(A0);
  // map the sensor range to a range of four options:
  int range = map(sensorReading, sensorMin, sensorMax, 0, 3);
  // do something different depending on the
  // range value:
  switch (range) {
    case 0:    // your hand is on the sensor
      Serial.println("dark");
      break;
    case 1:    // your hand is close to the sensor
      Serial.println("dim");
      break;
    case 2:    // your hand is a few inches from the sensor
      Serial.println("medium");
      break;
    case 3:    // your hand is nowhere near the sensor
      Serial.println("bright");
      break;
  }
  delay(1);        // delay in between reads for stability
}
```

Appendix C Code

Code 1: the program for the whole system

```
#include <FreqCounter.h>
#include <LiquidCrystal.h>
#include <Bridge.h>
#include <HttpClient.h>
#include <Console.h>
#include <math.h>
#define HEAT 1
#define COOL 0
LiquidCrystal lcd(12, 11, 9, 4, 3, 2);
long double temperature = 0;
const int buttonPin = 8;     // the number of the pushbutton pin
int mode = HEAT;
int buttonState = 0; // variable for reading the pushbutton status
int lastButtonState = buttonState;
const long double p1 =  -1.243e-12;//  (-1.985e-12, -5.015e-13)
const long double p2 =   2.528e-08;//  (1.067e-08, 3.989e-08)
const long double p3 =  -0.0001957;//  (-0.0003026, -8.88e-05)
const long double p4 =      0.6987;//  (0.3545, 1.043)
const long double p5 =      -944.8 ;// (-1356, -533.4)
const long double q1 =   7.912e-12;//  (6.001e-12, 9.824e-12)
const long double q2 =  -1.456e-07 ;// (-1.826e-07, -1.087e-07)
const long double q3 =    0.001004 ;// (0.0007378, 0.001271)
const long double q4 =      -3.062 ;// (-3.911, -2.213)
const long double q5 =        3481 ;// (2471, 4490)
/* AD590: FOR CALIBRATION */
int AD590_PIN = A2;
int AD590RAW = 0;
int mappedValue = 0;

int analogIn = A0;
int analogVal = 0;

void LCD_SETUP(){
  lcd.begin(16, 2);
  // Print a message to the LCD.
  lcd.print("REAL_TIME  USER");
}

void LCD_DISPLAY(int temp_real_time, double USER_input, int mode){
    // set the cursor to column 0, line 1
  // (note: line 1 is the second row, since counting begins with 0):
  lcd.setCursor(0, 1);
  String USER_input_string = String(USER_input);
  String temp_real_time_string = String(temp_real_time);
  String mode_string = String(mode);
  String total = temp_real_time_string + "    " + USER_input_string +" "+ mode_string;
  lcd.print(total);
}
void setup() {
  // Bridge takes about two seconds to start up
  pinMode(6,OUTPUT);
  pinMode(7,OUTPUT);
  LCD_SETUP();
  Bridge.begin();
  Console.begin();
    // Wait for Console port to connect
 while (!Console);
  Console.println("CONSOLE ON");//Data flow: Arduino --> Yun Shield --> Arduino IDE
}

void loop() {
  buttonState = digitalRead(buttonPin);
  // compare the buttonState to its previous state
  if (buttonState != lastButtonState) {
    // if the state has changed, increment the counter
    if (buttonState == HIGH) {
      if (mode == HIGH)
          mode = LOW;
      else
          mode = HIGH;
```

```
    }
  }
  /* FOR CALIBRATION
   * AD590RAW = analogRead(AD590_PIN);
   * float voltageAD = (AD590RAW*1.0)/1024 * 4.0;
   * float AD590_temperature = (voltageAD) * 100.0 - 273.15;
   * Console.print(AD590_temperature);
   */
  lastButtonState = buttonState;
  /* COUNT THE FREQUENCY */
  FreqCounter::f_comp = 8;
  FreqCounter::start(1000);
  while(FreqCounter::f_ready == 0);
  unsigned long frequency = FreqCounter::f_freq;
  /* READ THE USER INPUT */
  analogVal = analogRead(analogIn);
  int set = map(analogVal,0,1024,0,60);
  set = constrain(set, 5, 45);
   LCD_DISPLAY((int)temperature,set,mode);
  /* INITIALIZE HTTPCILENT */
  HttpClient client;
  // Make a HTTP request:To send analog input values of A0 and A1
  //interfacing with dweet
  //access by https://dweet.io/follow/YUN_ANALOG_IN_DWEETING_2
  client.get("http://www.dweet.io/dweet/for/YUN_ANALOG_IN_DWEETING_2?UserInput="+String(set)
  +"&Frequency="+String(frequency)+"&temperature="+String((double)temperature)+"&mode="+String(mode));
  /* BANG BANG CONTROL */
  if(mode == HEAT){
    temperature =  p1 * pow(frequency,4) + p2 * pow(frequency,3) + p3 * pow(frequency,2) + p4 *frequency + p5;
    if(abs(temperature - set) >= 1){
      digitalWrite(6,LOW);
      digitalWrite(7,HIGH);
    }
    else {
      digitalWrite(6,LOW);
      digitalWrite(7,LOW);
    }

  } else if( mode == COOL){
    temperature = q1 * pow(frequency,4) + q2 * pow(frequency,3) + q3 * pow(frequency,2) + q4 *frequency + q5;
    if(abs(temperature - set) >= 1){
      digitalWrite(6,HIGH);
      digitalWrite(7,LOW);
    }
    else {
      digitalWrite(6,LOW);
      digitalWrite(7,LOW);
    }

  }

}
```

Code 2: A segment of code using PID Control

```
unsigned long now = millis();
double timeChange = (double)(now - lastTime);
if(mode == HEAT){
  temperature =  p1 * pow(frequency,4) + p2 * pow(frequency,3) + p3 * pow(frequency,2) + p4 *frequency + p5;
  double error = set - temperature;
  errSum+=(error * timeChange);
  double dErr = (error - lastErr)/timeChange;
  Output = kp * error + ki*errSum + kd *dErr;
  lastErr = error;
  lastTime = now;
  if(Output > 255){
    Output = 255;
  }
  if(Output < 0){
    Output = 0;
  }
  analogWrite(6,Output);
  digitalWrite(7,LOW);
} else if( mode == COOL){
  temperature = q1 * pow(frequency,4) + q2 * pow(frequency,3) + q3 * pow(frequency,2) + q4 *frequency + q5;
  double error = temperature - set;
  errSum+=(error * timeChange);
  double dErr = (error - lastErr)/timeChange;
  Output = kp * error + ki*errSum + kd *dErr;
  lastErr = error;
  lastTime = now;
  if(Output > 255){
    Output = 255;
  }
  if(Output < 0){
    Output = 0;
  }
  analogWrite(7,Output);
  digitalWrite(6,LOW);
}
```