Community Experience Distilled

Smart Internet of Things Projects

A project-based guide to enhance your capability to build smart IoT projects

Agus Kurniawan

[PACKT] PUBLISHING open source* community experience distilled

Table of Contents

Chapter 1: Make your IoT Project Smart 1

Introducing basic statistics and data science 1

Python for computational statistics and data science 4

Python libraries for computational statistics and data science 6

NumPy 7

pandas 7

SciPy 7

Scikit-learn 7

Shogun 7

SymPy 8

Statsmodels 8

Building a simple program for statistics 8

IoT devices and platforms 9

Arduino 10

Raspberry Pi 13

BeagleBone Black and Green 15

IoT boards-based ESP8266 MCU 16

IoT boards-based TI CC3200 MCU 18

Sensing and actuating on IoT devices 20

Sensing and actuating on Arduino devices 20

Sensing and actuating on Raspberry Pi devices 32

Setting up 32

Accessing Raspberry Pi GPIO 33

Sensing through sensor devices 37

Building a smart temperature controller for your room 41

Introducing PID controller 42

Implementing PID controller in Python 42

Controlling room temperature using PID controller 49

Summary 53

References 54

Index 55

1

Make your IoT Project Smart

We're going to begin by reviewing basic Statistics. Then, we learn how to sense and actuate from IoT devices such as Arduino and Raspberry Pi. Exploring various Python libraries related to Statistics and Data Science is introduced so we know their existance. These libraries are useful to build our project throughout this book.

By the end of this chapter, you'll know how to

- Introduce basic Statistics and Data Science.
- Review several IoT devices and platforms.
- Sense and actuate through external devices on IoT devices
- Build a smart IoT project.

So let's get started!

Introducing basic statistics and data science

Once upon a time you want to know your room temperature so you measure it every hour during the day using a particular tool. This data is needed because you want to decide to buy an AC (Air Conditioning) machine or not. After measurement is done, you obtain a list of temperature data. The result of measurement can be seen in Table 1.

Time	Temperature (Celsius)	Time	Temperature (Celsius)
01:00	18	13:00	28
02:00	17	14:00	29
03:00	18	15:00	28

04:00	19	16:00	27
05:00	20	17:00	25
06:00	20	18:00	24
07:00	21	19:00	24
08:00	22	20:00	23
09:00	22	21:00	22
10:00	24	22:00	20
11:00	25	23:00	19
12:00	26	24:00	19

Table 1. Temperature data in a room.

Table 1 shows a list of Temperature data in tabular form. You try to understand data meaning. From this situation, you need knowledge of Statistics. Some Statistics terms such as Mean, Median, Variance, and Standard Deviation.

Suppose we have a sample of n data which designated by x_1, x_2, x_3, …, x_n. We can calculate Mean, Median, Variance, and Standard Deviation using the following formula.

$$mean = \bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}$$

$$median = value\ on\ position \frac{n+1}{2}, if\ n\ is\ odd$$

$$median = value\ on\ position\ between\ n/2\ and\ (n/2)+1, if\ n\ is\ even$$

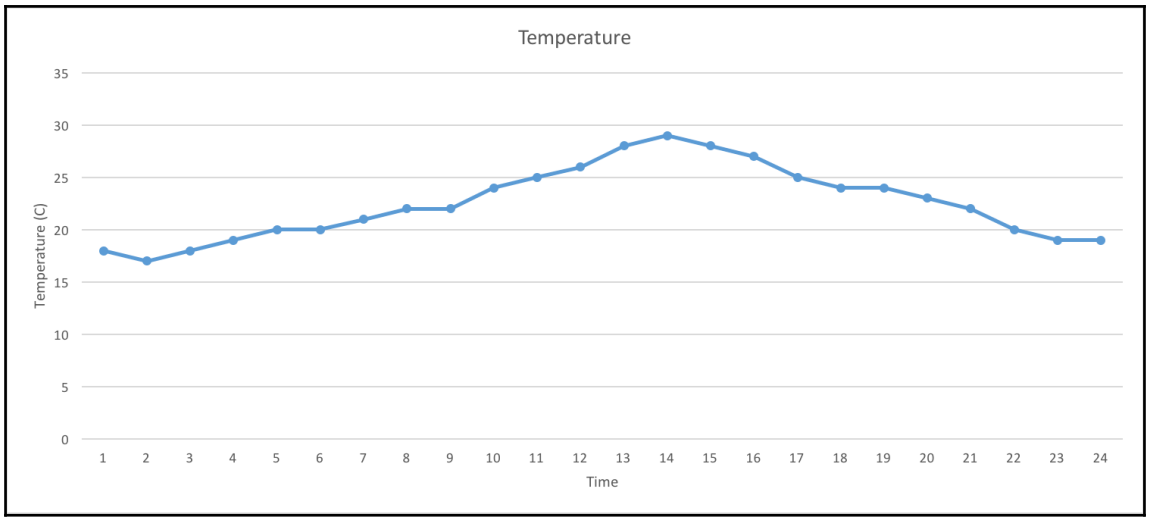
$$variance = s^2 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}$$

$$Standard\ deviation\ =\ s\ =\ \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}}$$

To compute median value, you should arrange the data in ascending order.

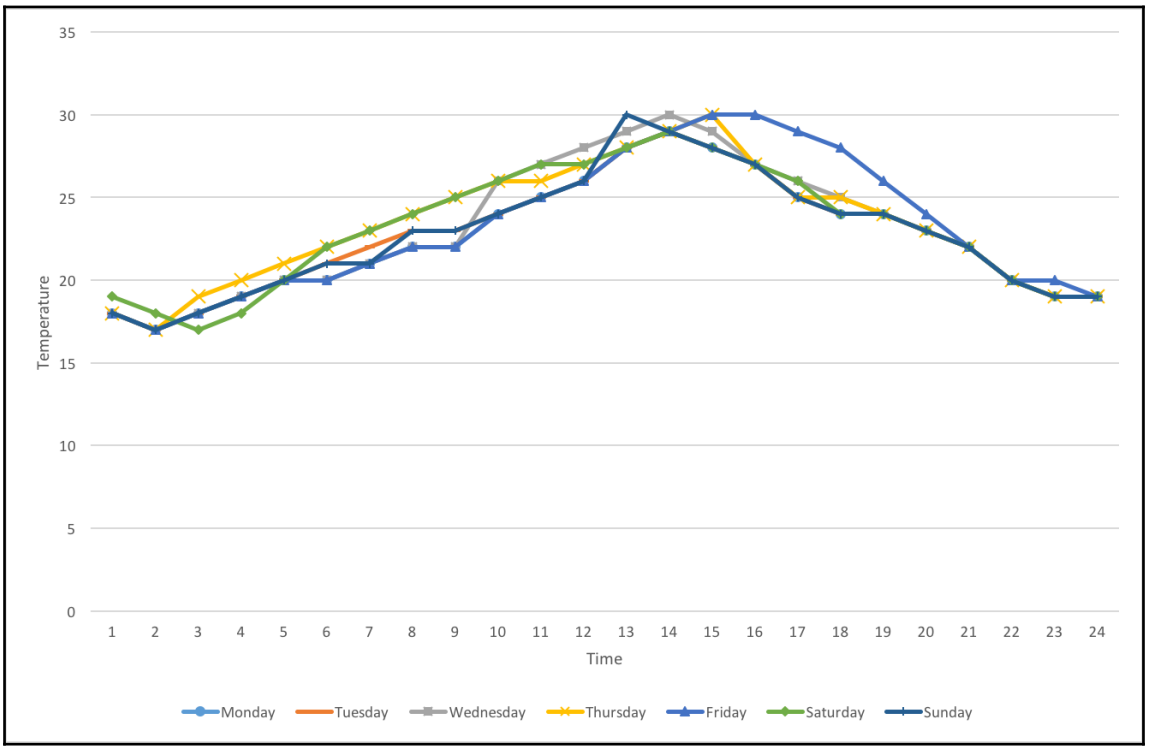
From Table 1, you can calculate Mean, Median, Variance and Standard Deviation using the above formulas. You can should obtain values 22.5, 22, 12.348, and 3.514 respectively.

To understanding a pattern of those data, you try to visualize them in Graphics form, for instance, using Microsoft Excel. The result can be seen in Figure below.



You can see that average value of your room temperature is 22.5 Celsius. The Temperature maximum and minimum values are 19 and 17 respectively. From this situation, you can think how type of the AC machine you want to buy.

Furthermore, you can extend your investigation by measuring room temperature for a week. After measured, you can plot them on graphics form, for instance, Microsoft Excel. A sample of temperature measurement is shown in Figure below.



The Graphics shows room temperature changes every day. If you measure it for all days in a year, you should see how temperature trends on your room. Knowledge about Data Science can improve you how to learn from data. Of course, some Statistics and Machine Learning computing are involved to get insight how data behaviors are.

This book helps you to get started how to apply Data Science and Machine Learning in real-cases with focusing on IoT fields.

Python for computational statistics and data science

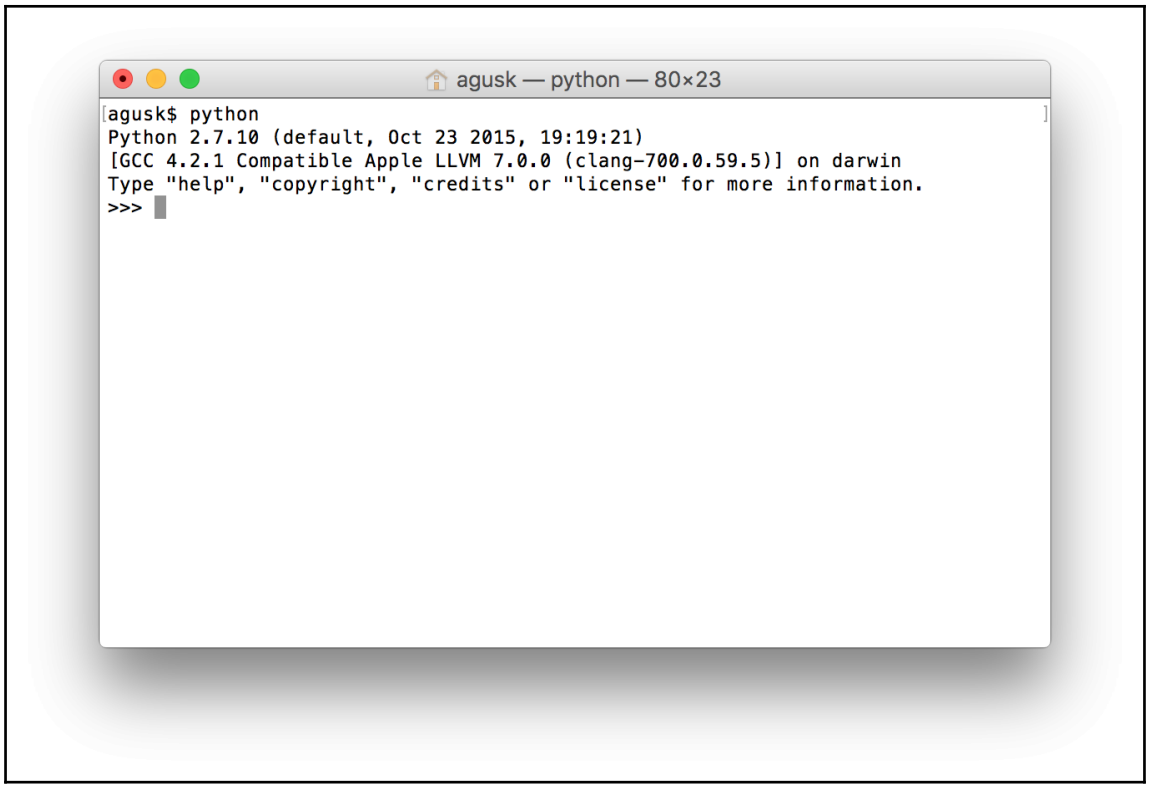
Python is one of programming language and widely used for general-purpose. Starting to program with Python is a good point. Python provides simple programming syntax and a lot of APIs which we can use to expand our program.

To use Python in your computer, you can download and install it on https://www.python.org/downloads/. After completed installation, we can run Python program via Terminal or Command Prompt on Windows platform by typing this command.

```
$ python
```

Note: remove $ sign. Just type `python` on Terminal. This is applicable to Python 2.x

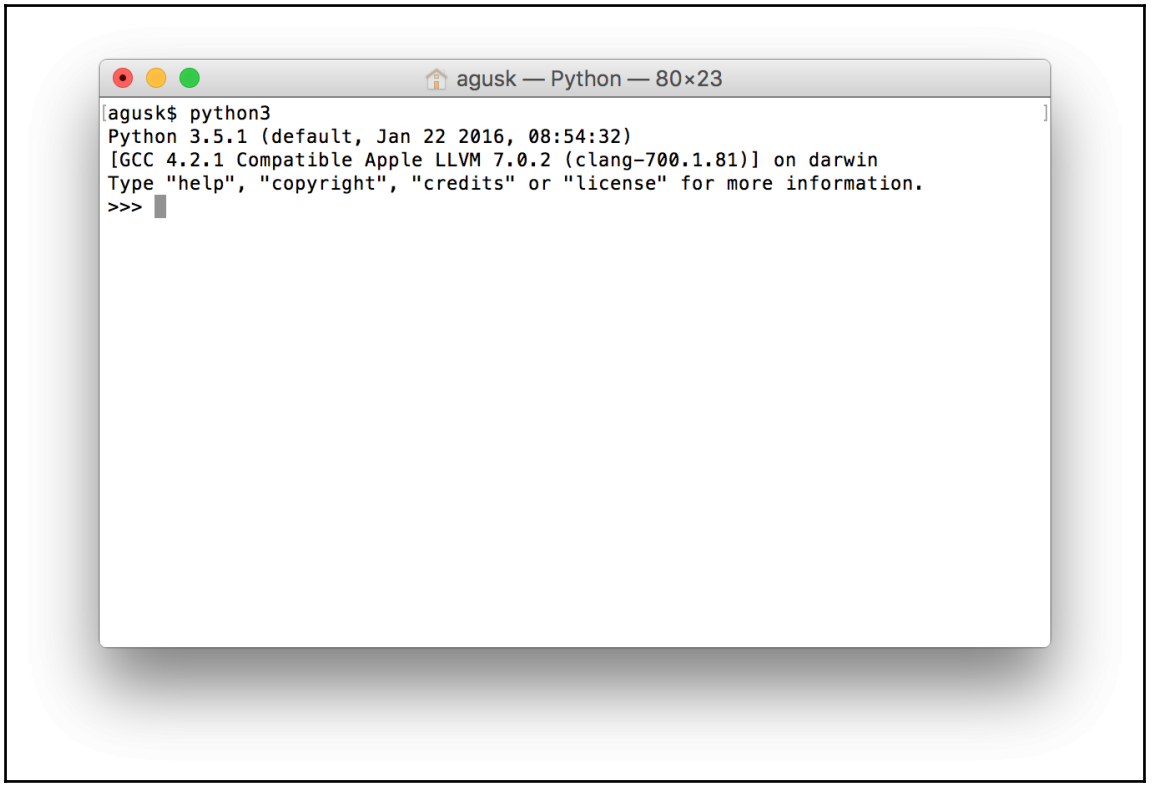
After executed, you should see Python command prompt, shown in Figure below.



If you installed Python 3, you usually run the program using this command.

```
$ python3
```

You should see Python 3 shell on your Terminal.



What's next?

There are a lot of Python resources to help you learn how to write program using Python. I recommend to learn Python programming with reading Python documents on https://www.python.org/doc/. You also can read Python books to accelerate your learning. This book does not cover topics about basic Python programming language.

Python libraries for computational statistics and data science

Python has big communities. They help their member to learn and share. Several community members have been open sources related to computational Statistics and Data Science which can be used for our works. We use these libraries for our implementation.

The following is several Python libraries for Statistics and Data Science.

NumPy

NumPy is fundamental package for efficient scientific computing in Python. This library has capabilities in handling N-dimensional arrray and integrating C/C++ and Fortran code. It also provides features for linear algebra, Fourier transform and random number.

Official website for NumPy can be reached on http://www.numpy.org.

pandas

pandas is a library for handling table-like structures which called DataFrame object. This is powerful and efficient numerical operations similar to NumPy's array object.

Further information about pandas, you can visit on this URL, http://pandas.pydata.org.

SciPy

SciPy is expansion of the NumPy library. It contains functions of linear algebra, interpolation, integration, clustering, and etc.

Official website: http://scipy.org/scipylib/index.html.

Scikit-learn

Scikit-learn is the most popular Machine Learning library for Python. It provides many functionalities such as preprocessing data, classification, regression, clustering, dimensionality reduction, and model selection.

Further information about Scikit-learn, you can visit on http://scikit-learn.org/stable/.

Shogun

Shogun is a Machine Learning library for Python which focuses on large-scale kernel methods such as Support Vector Machines (SVMs). This library comes with a range of different SVM implementation.

Official website: http://www.shogun-toolbox.org.

SymPy

SymPy is a Python library for symbolic mathematical computations. It has capabilities in calculus, algebra, geometry, discrete Mathematics, quantum Physics and more.

Official website: http://www.sympygamma.com.

Statsmodels

Statsmodels is a Python module which we can perform to process data, estimate statistical models and test data.

You can find out more about Statsmodels by visiting on the official website, http://statsmodels.sourceforge.net.

Building a simple program for statistics

On the first section, you already measure room temperature. Now we try to play a simple computational Statistics using Statsmodels. We use our measurement result data and then, build a linear regression for our data.

Firstly, we should install Statsmodels. This library needs required libraries such as NumPy, SciPy, pandas, and patsy. We can install them using pip. Type this command.

```
$ pip install numpy scipy pandas patsy statsmodels
```

If your computer doesn't install it yet, you can install pip with following the guideline on https://pip.pypa.io/en/stable/installing/.

For testing, we create a Python program. Write these scripts.

```
import numpy as np
import statsmodels.api as sm

# room temperature
Y = [18, 17, 18, 19, 20, 20, 21, 22, 22, 24, 25, 26, 28, 29, 28, 27, 25,
24, 24, 23, 22, 20, 19, 19]
X = range(1, 25)
X = sm.add_constant(X)

model = sm.OLS(Y, X)
results = model.fit()

# print
```

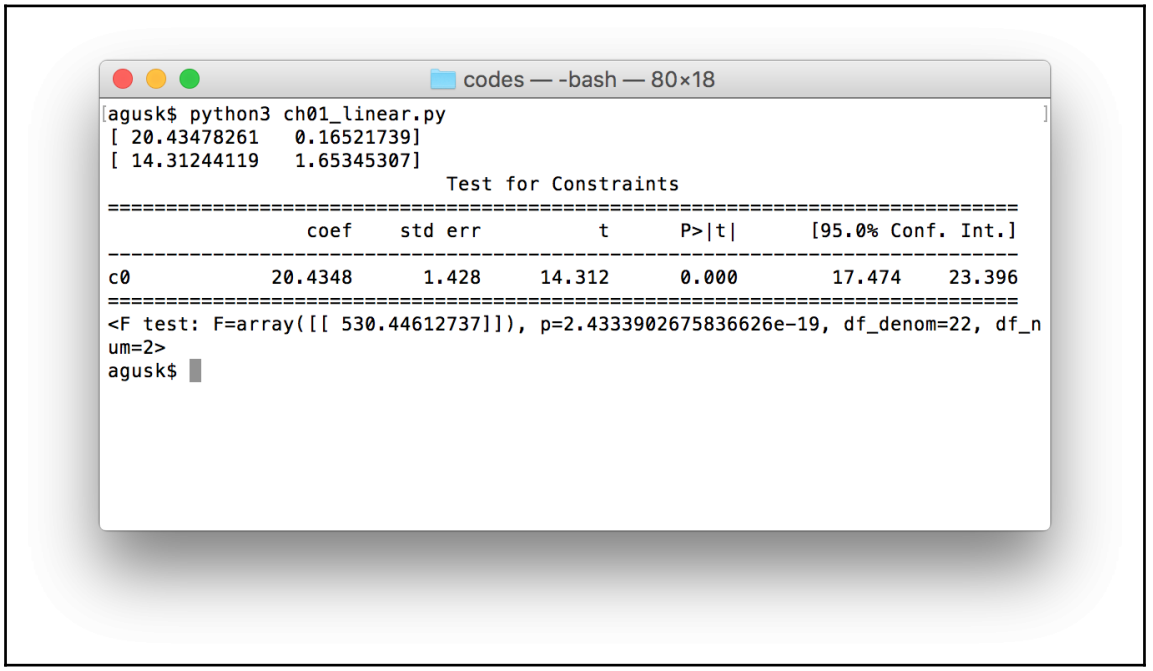
```
print(results.params)
print(results.tvalues)


print(results.t_test([1, 0]))
print(results.f_test(np.identity(2)))
```

We build a linear regression using `sm.OLS()`. We do estimation using `model.fit()`. The last we print the computation result. Save the program into a file, called ch01_linear.py.

Now you can run this program using this command.

```
$ python ch01_linear.py
```

You should see the program output, shown in Figure below.



IoT devices and platforms

Internet of Things (IoT) platform has capabilites to connect an Internet network and interact with other platforms. Generally speaking, talking about IoT in term of device platform is

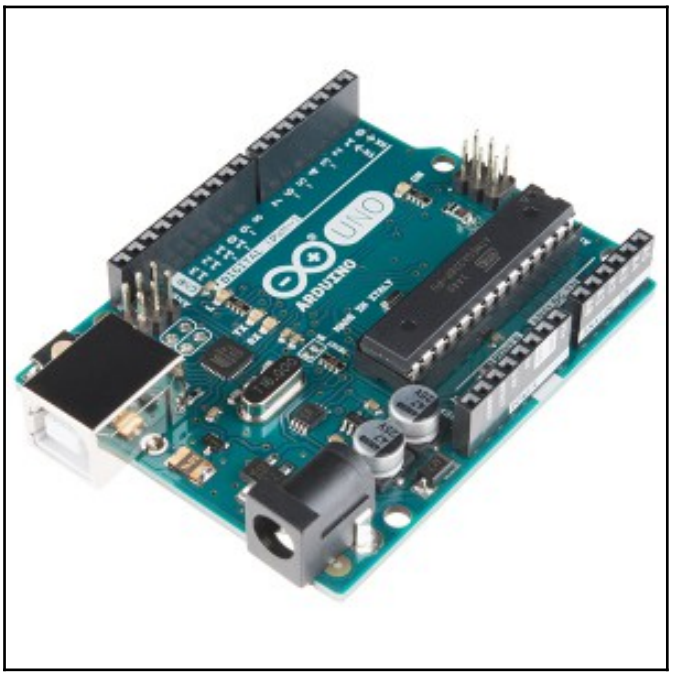
huge topics. In this section, we explore several IoT device platforms which are widely used in customer side.

Arduino

Arduino is a development board which is widely used. This board is well-known board in embedded community. Mostly Arduino boards are built using Atmel AVR but some boards use other MCUs regarding to who joints venture with Arduino. Currently, Arduino boards are built by Arduino.cc and Arduino.org. Please make sure you use board and software from the same company.

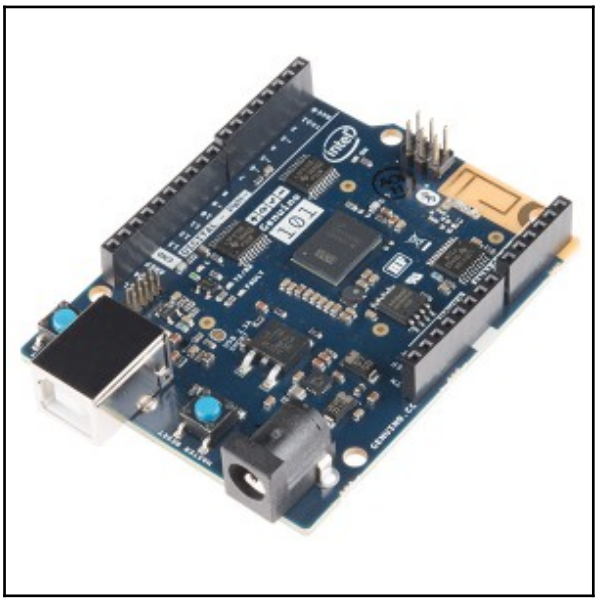
We're going to review several Arduino boards from Arduino.cc. We can read comparison of all Arduino boards from Arduino.cc by visiting this site, http://www.arduino.cc/en/Products/Compare. We will review some Arduino boards such as Arduino Uno, Arduino 101 and Arduino MKR1000.

Arduino Uno model is widely used in Arduino development. It's built on the top of MCU ATmega328P. The board provides several digital and analog I/O pins which we can attach our sensor and actuator devices. SPI and I2C Protocols also are provided by Arduino Uno. Further information about the board, I recommend you to read the board specification on http://www.arduino.cc/en/Main/ArduinoBoardUno. You can see Arduino board in Figure below.



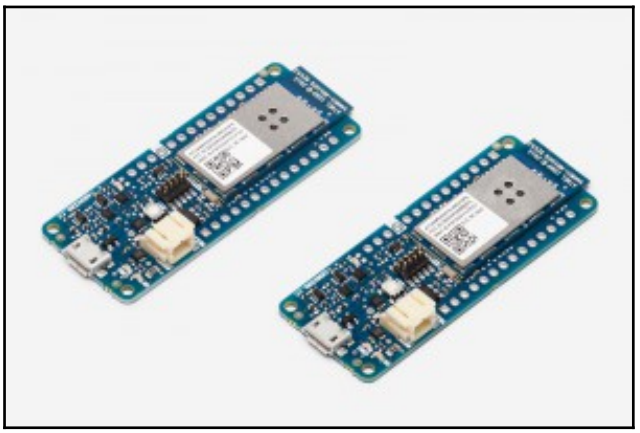
source: https://www.sparkfun.com/products/11021

Arduino 101 has the same model as Arduino Uno in terms of I/O pins. Arduino 101 runs Intel® Curie™, http://www.intel.com/content/www/us/en/wearables/wearable-soc.html, as core module. This board has built-in Bluetooth module. If you want to Arduino 101 works with WiFi network, you should add additional WiFi shield. I recommend to use Arduino WiFi Shield 101, http://www.arduino.cc/en/Main/ArduinoWiFiShield101.



https://www.sparkfun.com/products/13850

Arduino MKR1000 is a new board while this book is writing. This board uses the Atmel ATSAMW25 SoC which provides built-in WiFi module. I recommend to use this board for IoT solution for Arduino platform because the WiFi module, WINC1500, is supported for SSL and ECC508 CryptoAuthentication. Further information about this board, you can read it on http://www.arduino.cc/en/Main/ArduinoMKR1000.

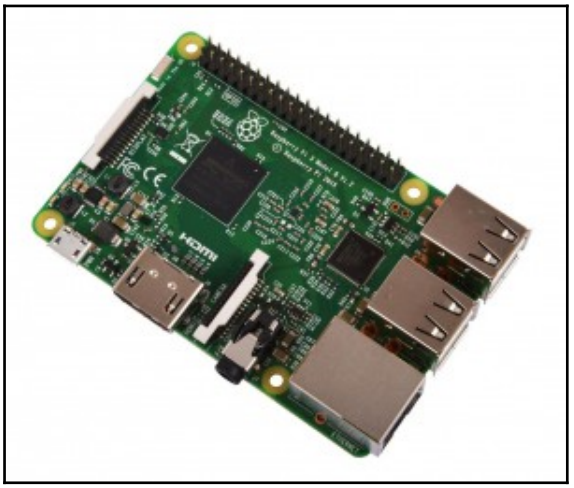


source: http://www.arduino.cc/en/Main/ArduinoMKR1000

Raspberry Pi

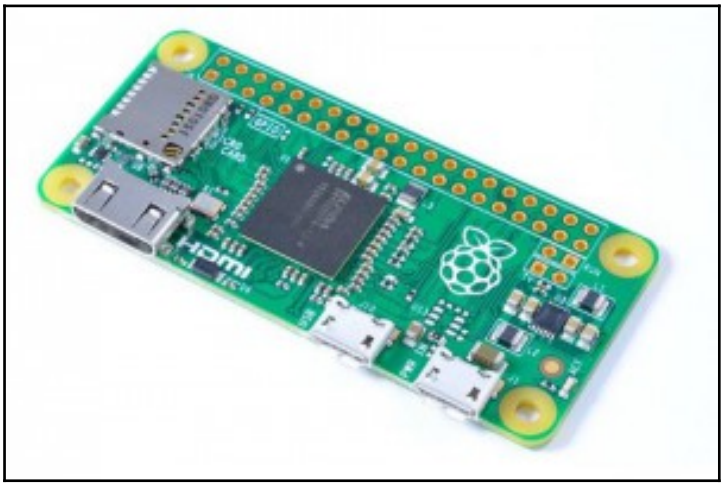
The Raspberry Pi is a low cost with credit-card sized computer which is created by Eben Upton. It's a mini computer for education purpose. To see all Raspberry Pi models, you can read it on https://www.raspberrypi.org/products/. You can see Raspberry Pi 3 Model B and Raspberry Pi Zero on the following explanation.

The Raspberry Pi 3 Model B is the third generation Raspberry Pi. This board consists of a Quad-Core 64bit CPU, WiFi & Bluetooth. It's very well to be recommended for your IoT solution.



Source: https://thepihut.com/collections/raspberry-pi/products/raspberry-pi-3-model-b

Raspberry Pi Zero is a small computer with half the size of model A+. It runs with single core CPU and no network module but it provides micro HDMI to be connected to a monitor. Due to no network module, you should be extended module, for instance, Ethernet USB or WiFi USB to connect Raspberry Pi Zero into a network.



Source: https://thepihut.com/collections/raspberry-pi-zero/products/raspberry-pi-zero

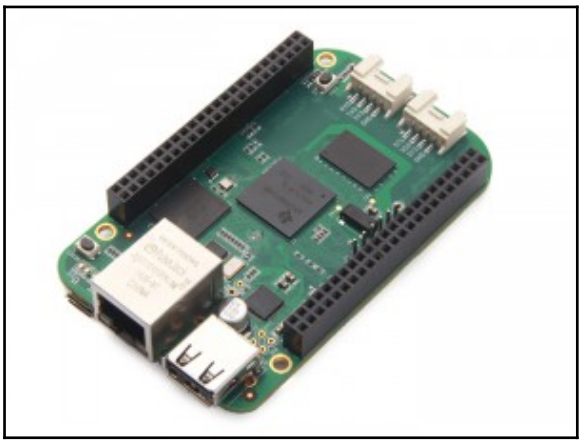
BeagleBone Black and Green

BeagleBone Black (BBB) Rev.C is a development kit based on an AM335x processor which integrates an ARM Cortex™ -A8 core operating at up to 1GHz. BBB is more powerful than Raspberry Pi. BBB board also provides internal 4GB 8-bit eMMC on-board flash storage.

BBB supports several OS such Debian, Android, and Ubuntu. Find out more about BBB, you can read it on https://beagleboard.org/black.

BeagleBone Black boad. Source: http://www.exp-tech.de/beaglebone-black-rev-c-element14

SeeedStudio BeagleBone Green (BBG) is a joint effort by BeagleBoard.org and Seeed Studio. BBG has the same features like BBB unless HDMI port is replaced by Grove connectors so BBG price is lower than BBB. You can review and buy this board on http://www.seeedstudio.com/depot/SeeedStudio-BeagleBone-Green-p-2504.html.



BeagleBone Green board. Source:
http://www.seeedstudio.com/depot/SeeedStudio-BeagleBone-Green-p-2504.html

IoT boards-based ESP8266 MCU

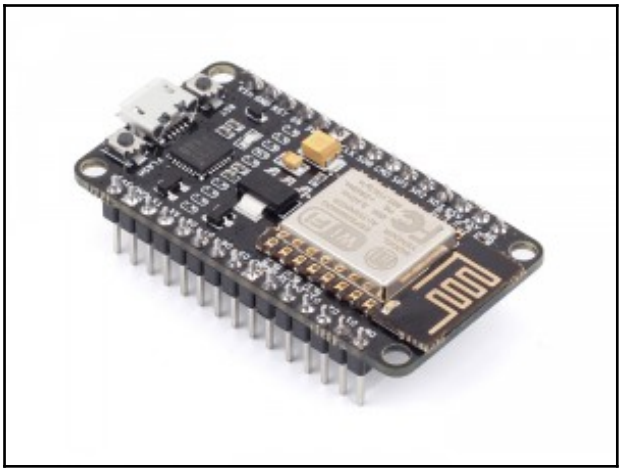
ESP8266 is a low cost WiFi MCU with supported full TCP/IP. It's built by Espressif, China manufacturer. Further information about this chip, you can review it on http://espressif.com/en/products/hardware/esp8266ex/overview.

There are many boards based on ESP8266 chip. The following is a list of board platform which is built on top of ESP8266 MCU

- NodeMCU. This board use NodeMCU firmware with Lua as programming language. Official website: http://www.nodemcu.com/.
- SparkFun ESP8266 Thing. It's developed by SparkFun. You should Serial hardware for instance, FTDI, to write program into this board but this product is ready for LiPo charger. You can read it on https://www.sparkfun.com/products/13231.
- SparkFun ESP8266 Thing-Dev. This board already included FTDI-to USB tool but no LiPo charger. It's developed by SparkFun and product information can be read on https://www.sparkfun.com/products/13711.
- SparkFun Blynk board – ESP8266. This board is included temperature and humidity sensor devices. You can read it on https://www.sparkfun.com/products/13794
- Adafruit HUZZAH with ESP8266 WiFi. It's developed by Adafruit. Product

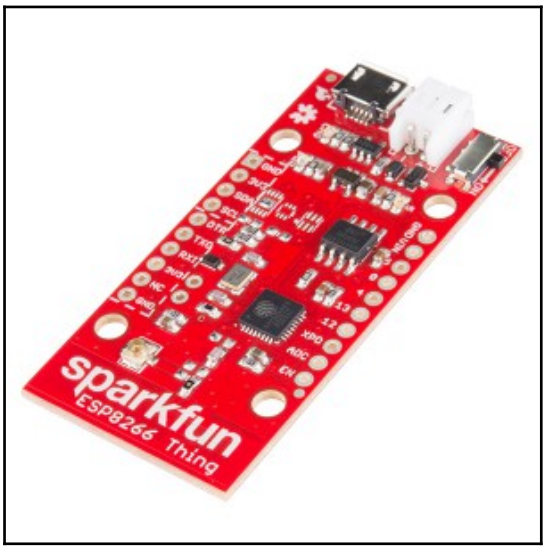
information: https://www.adafruit.com/products/2821.

If you're interested with ESP8266 chip, I recommend to join ESP8266 forum on http://www.esp8266.com. The following is a list of product forms for NodeMCU v2 and SparkFun ESP8266 Thing.



NodeMCU v2 board. Source:
http://www.seeedstudio.com/depot/NodeMCU-v2-Lua-based-ESP8266-development-kit-p-2415.html

Although NodeMCU v2 and SparkFun ESP8266 Thing boards have the same chip but their chip model is different. NodeMCU v2 uses ESP8266 module. Otherwise, SparkFun ESP8266 Thing board uses ESP8266EX chip. In addition, SparkFun ESP8266 Thing board provides LiPo connector which you can attach an external battery.



SparkFun ESP8266 Thing board. Source: https://www.sparkfun.com/products/13231

IoT boards-based TI CC3200 MCU

TI CC3200 is WiFi MCU based on ARM® Cortex®-M4 from Texas Instruments. This board is a completed solution for IoT. This chip is supported for Station, Access Point, and Wi-Fi Direct modes. In security view, TI CC3200 supports WPA2 personal and enterprise security and WPS 2.0. You can review this module on this site, http://www.ti.com/product/cc3200.

For IoT development, Texas Instruments provides The SimpleLink Wi-Fi CC3200 LaunchPad evaluation kit. It's a completed kit for development and debugging.

The SimpleLink Wi-Fi CC3200 LaunchPad board. Source:
https://www.conrad.de/de/entwicklungsboard-texas-instruments-cc3200-launchxl-1273804.html

TI CC3200 also is used by Readbear, http://redbear.cc, to develop RedBearLab CC3200 and RedBearLab WiFi Micro boards. These boards have the same functionalities like SimpleLink Wi-Fi CC3200 LaunchPad board but it excludes CC3200 debugger tool. These board prices also are lower than SimpleLink Wi-Fi CC3200 LaunchPad board's price.

RedBearLab CC3200 board. Source: http://www.exp-tech.de/redbearlab-cc3200

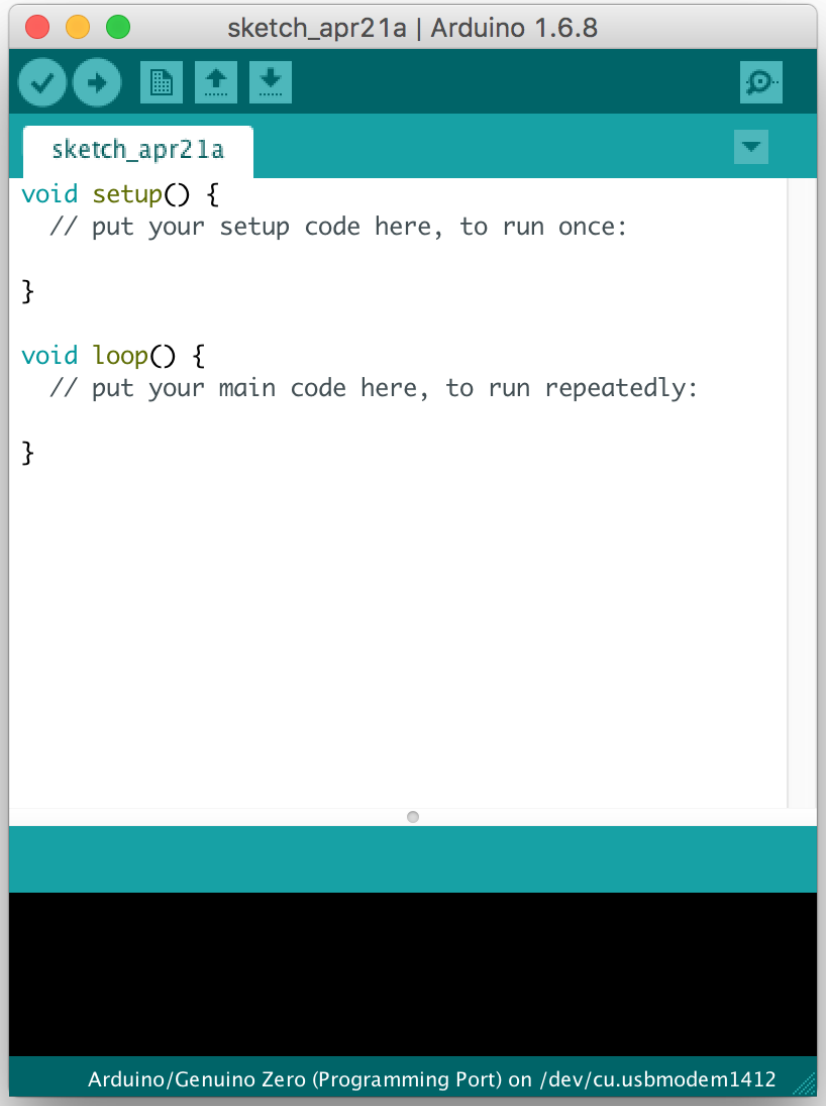
Sensing and actuating on IoT devices

In this section, we learn how to sense and actuate from IoT Devices. This part is important because we can gather data through sensor devices or interact with environment through actuator devices. For testing, I use Arduino from Arduino.cc and Raspberry Pi boards.

Sensing and actuating on Arduino devices

Most Arduino development using Sketch languange. This is a simple programming languange to build Arduino application. If you have experience in C/C++, you'll familiar with program syntax.

To stat your development, you can download Arduino software on https://www.arduino.cc/en/Main/Software. After that, you read Arduino API on this site, http://www.arduino.cc/en/Reference/HomePage.


sketch_apr21a | Arduino 1.6.8
sketch_apr21a
void setup() {
// put your setup code here, to run once:
}
void loop() {
// put your main code here, to run repeatedly:
}
Arduino/Genuino Zero (Programming Port) on /dev/cu.usbmodem1412

Related to Digital and Analog I/O, you should be familiar with the following Sketch API:

- `digitalRead()` is to read data from a digital pin on Arduino
- `digitalWrite()` is to write data to a digital pin on Arduino
- `analogRead()` is to read analog data from an analog pun on Arduino
- `analogWrite()` is to write data analog to an analog pin on Arduino

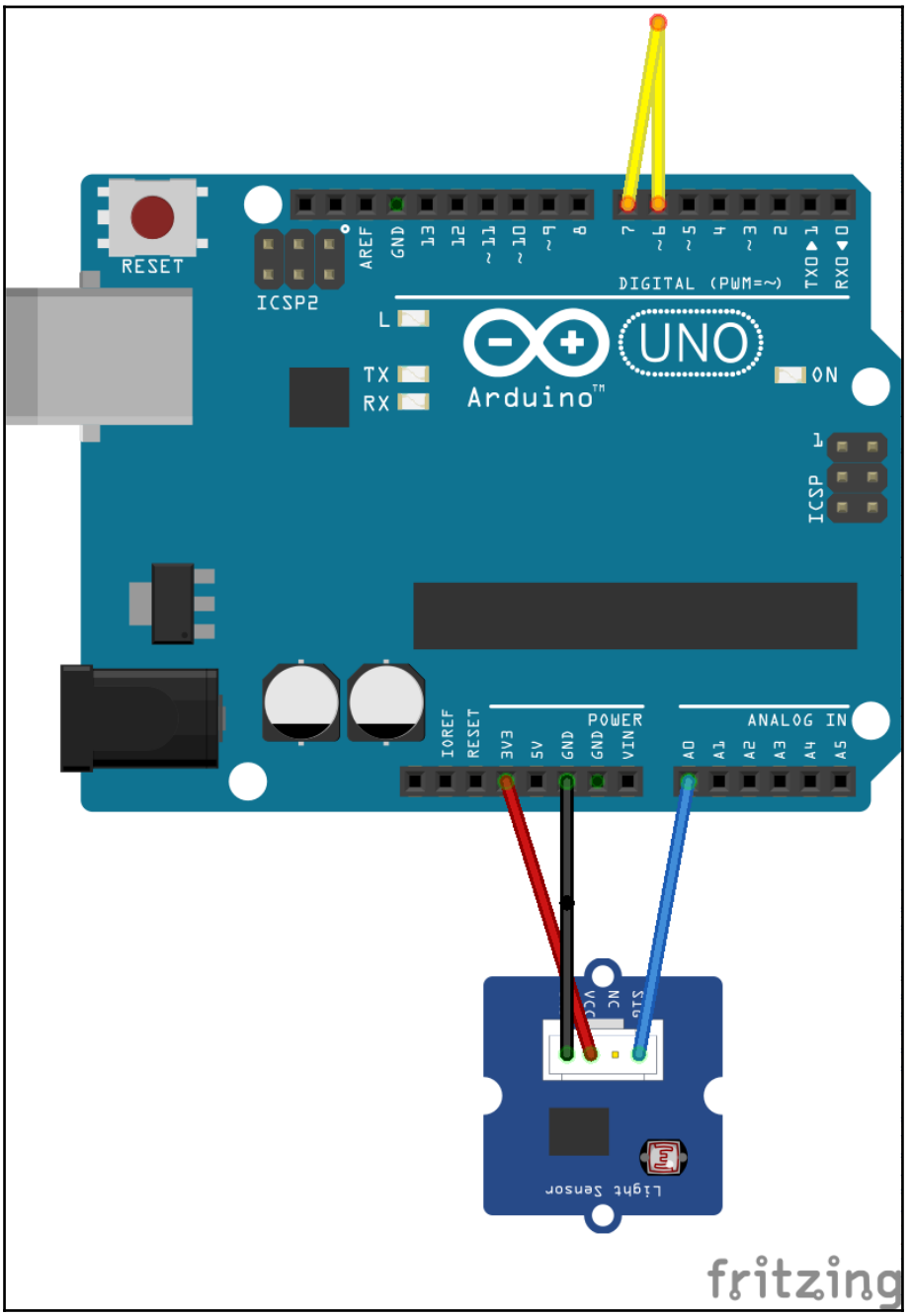
In this section, I'll show you how to work with Analog and Digital I/O on Arduino. I use a light sensor. LDR or Photoresistor sensor is low price for a light sensor device even Seeedstudio has already designed in module form, Grove – Light Sensor(P). You can review it on http://www.seeedstudio.com/depot/Grove-Light-SensorP-p-1253.html. This module has four pins but you only connect VCC and GND to VCC and GND of Arduino. Then, SIG pin is connected to an analog pin of Arduino board.

Grove – Light Sensor(P). Source: http://www.seeedstudio.com/depot/Grove-Light-SensorP-p-1253.html

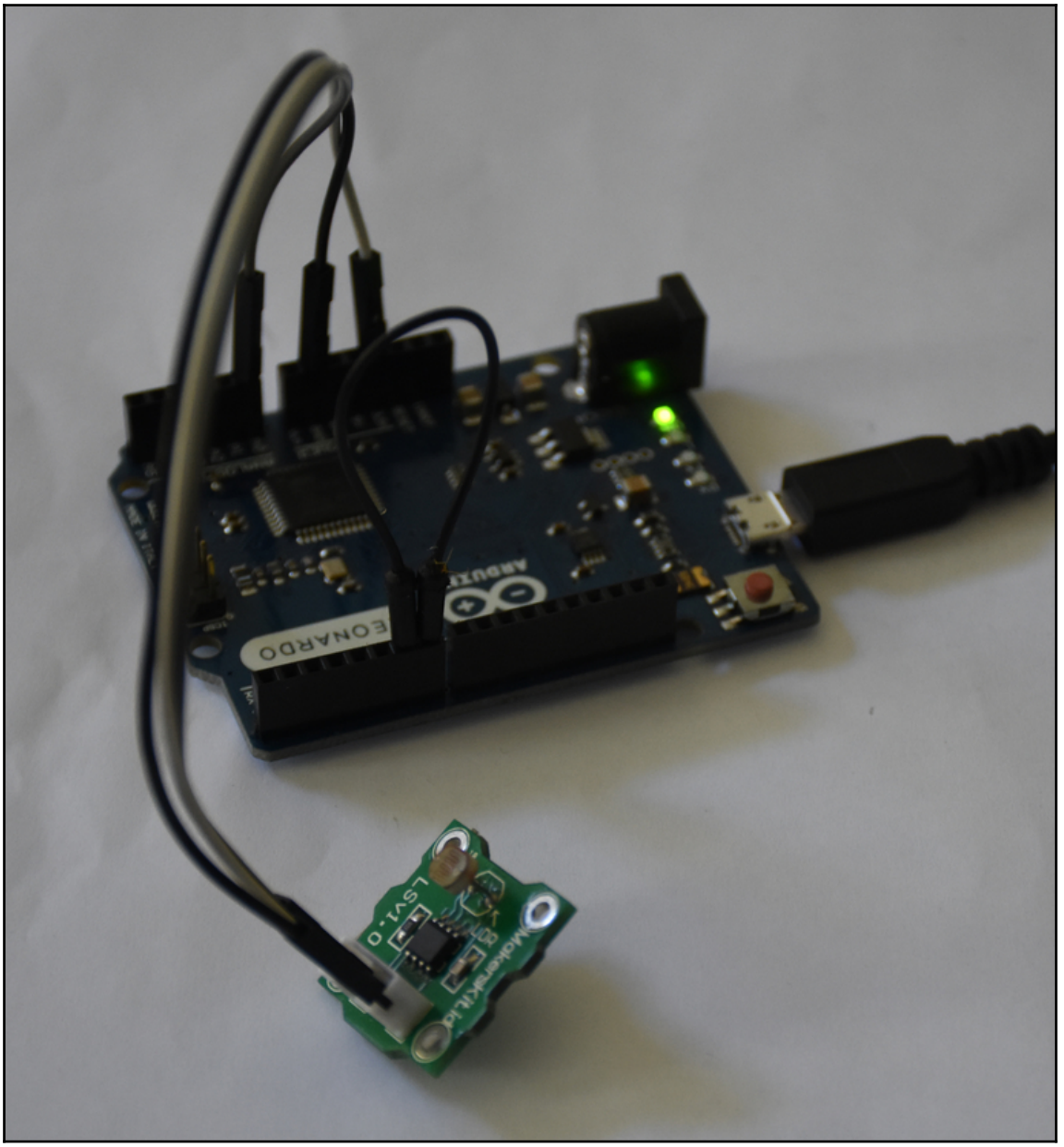
So you need the following resources for testing:

- Jumper cables
- Light sensor module, http://www.seeedstudio.com/depot/Grove-Light-SensorP-p-1253.html

Now you can connect digital 6 to digital 7. Then, SIG pin from LDR module is connected to A0. VCC and GND of LDR module are connected 3V3 and GND. You can see the hardware wiring in Figure below.


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

My wiring implementation is shown in Figure below.



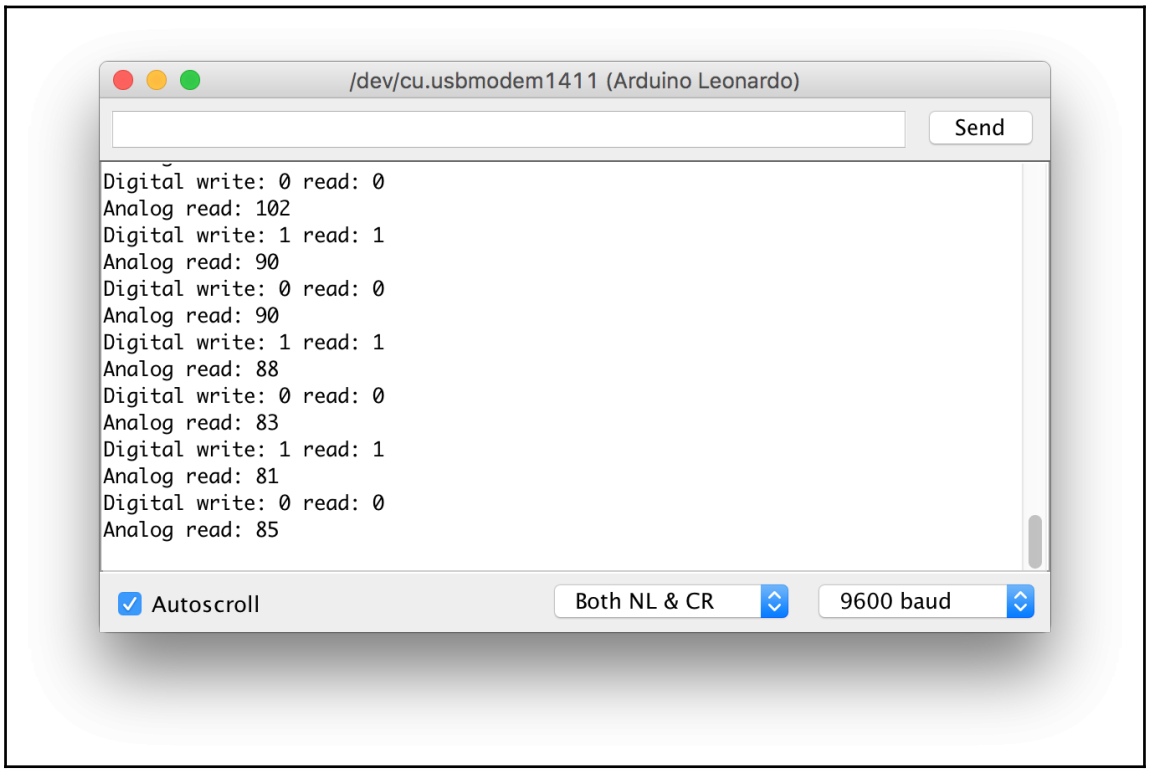
The next step is to write an Arduino program. You can open Arduino software. Then, write

this program.

```
int dig_output = 7;
int dig_input = 6;
int analog_input = A0;

int digital_val = LOW;

void setup() {
  Serial.begin(9600);
  pinMode(dig_output, OUTPUT);
  pinMode(dig_input, INPUT);
}

void loop() {

  digitalWrite(dig_output,digital_val);
  int read_digital = digitalRead(dig_input);
  Serial.print("Digital write: ");
  Serial.print(digital_val);
  Serial.print(" read: ");
  Serial.println(read_digital);
  int ldr = analogRead(analog_input);
  Serial.print("Analog read: ");
  Serial.println(ldr);

  if(digital_val==LOW)
    digital_val = HIGH;
  else
    digital_val = LOW;

  delay(1000);
}
```

Save this program as ArduinoIO. You can deploy this program into Arduino board through Arduino software.

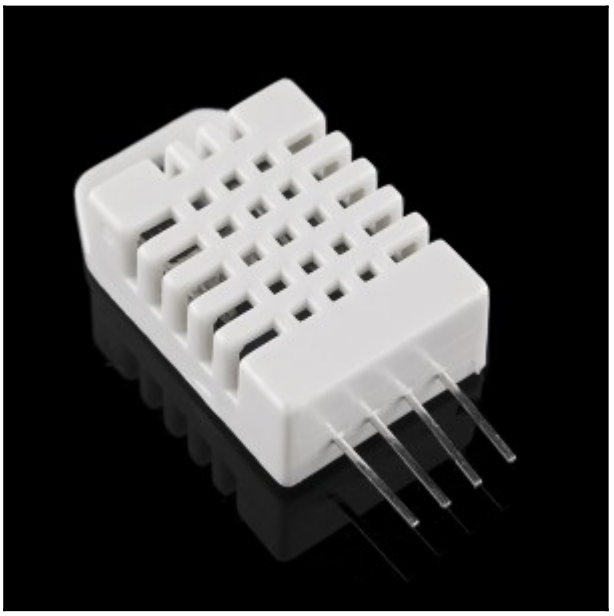
If finished, you can open Serial Monitor tool from Arduino Software



How to work?

This program sends digital data (HIGH and LOW values) from digital pin 7 to digital pin 6. You can read a light value from LDR module via `analogRead()` on A0 pin.

For the second sensing testing scenario, we try to read temperature and humidity data from a sensor device. I use DHT22. The RHT03 (also known by DHT-22) is a low cost humidity and temperature sensor with a single wire digital interface. You can obtain this module on Sparkfun, https://www.sparkfun.com/products/10167 and Adafruit, https://www.adafruit.com/products/393. You could also find this module on your local online or local electronics store.



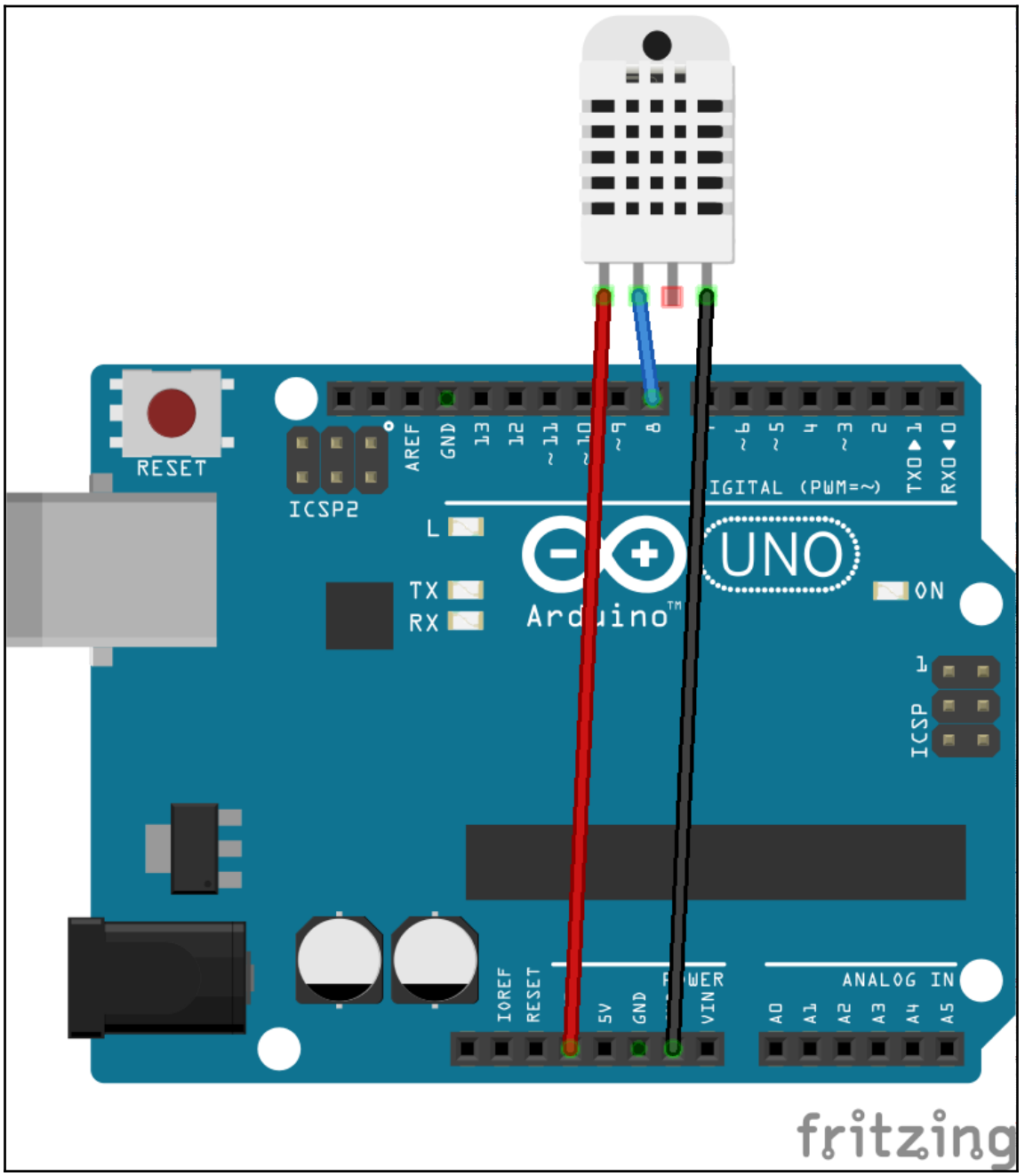
DHT22 module. Source: https://www.adafruit.com/products/385

Further information about DHT-22 module, you can read DHT-22 datasheet on http://cdn.sparkfun.com/datasheets/Sensors/Weather/RHT03.pdf.

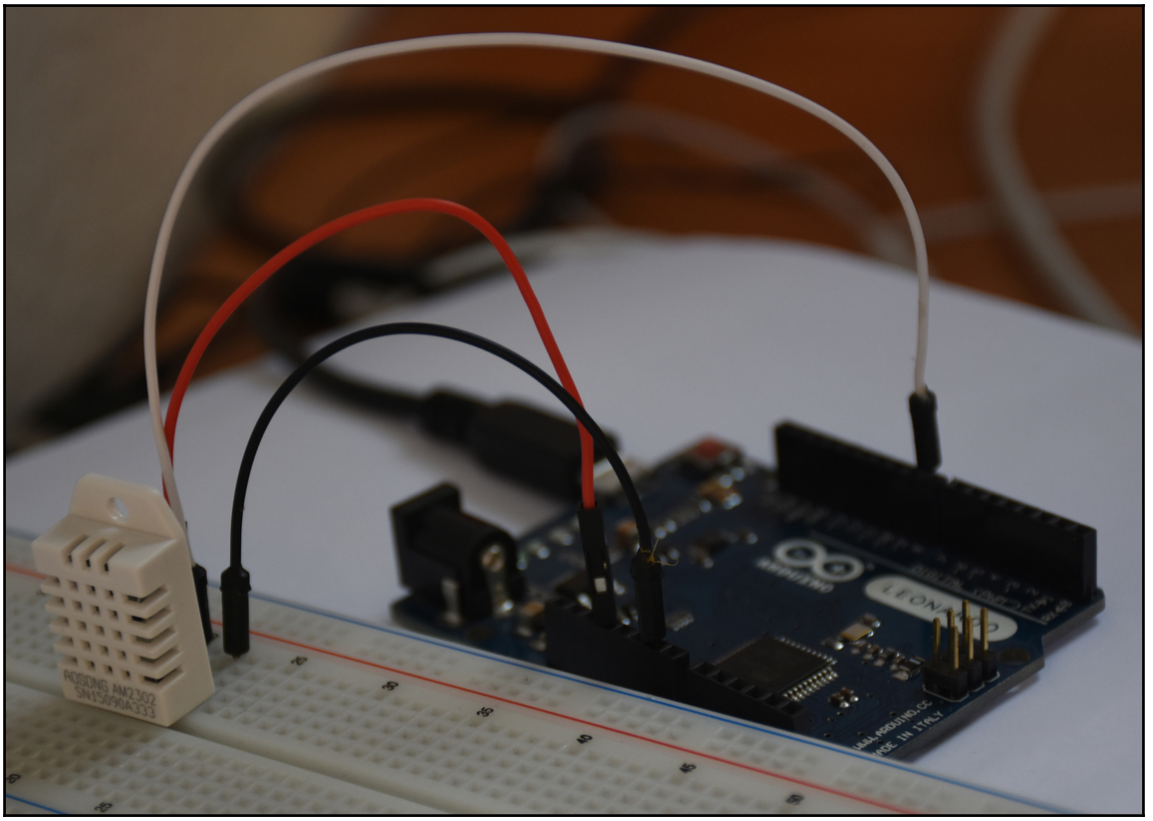
Now we connect DHT-22 module to Arduino. The following is the wiring:

- VDD (pin 1) is connected to V3V pin on Arduino
- SIG (pin 2) is connected to digital pin 8 on Arduino
- GND (pin 4) is connected to GND on Arduino

You see this wiring in Figure below.

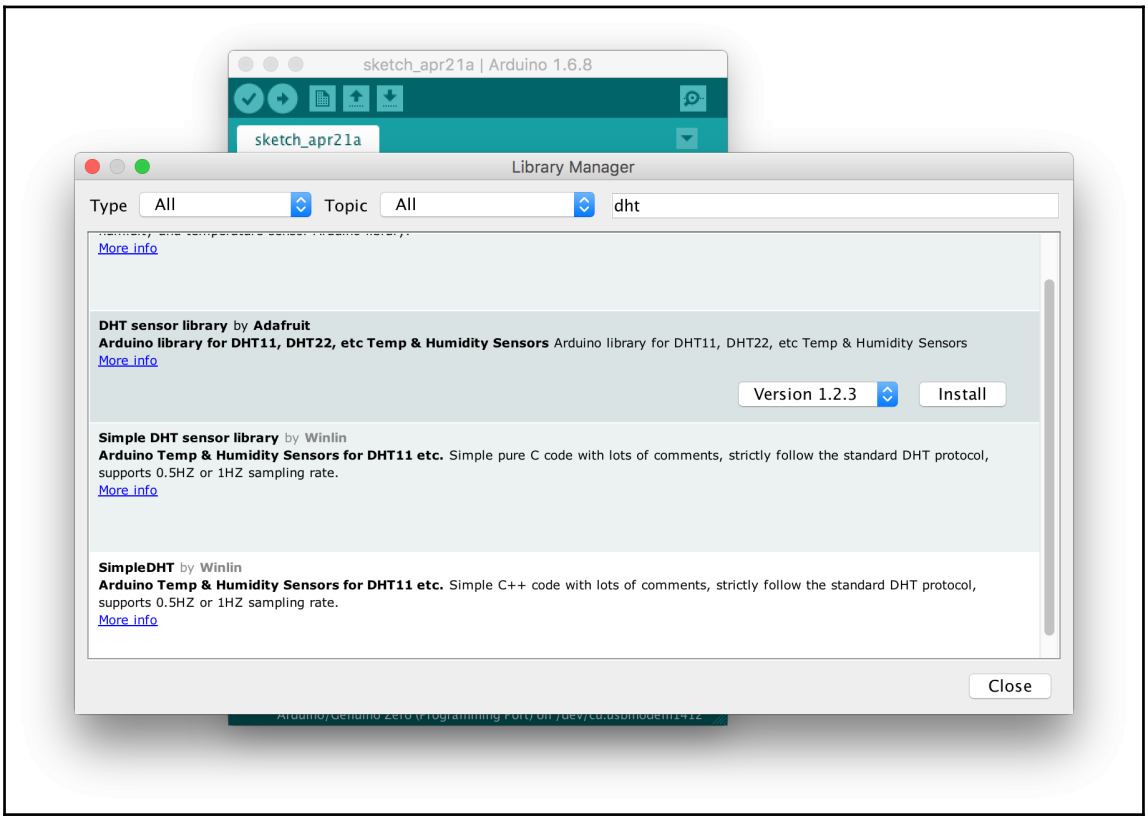

RESET
ICSP2
AREF
GND
13
12
~11
~10
~9
8
7
~6
~5
4
~3
2
TX→1
RX←0
DIGITAL (PWM=~)
L
TX
RX
Arduino™
UNO
ON
ICSP
POWER
IOREF
RESET
3V3
5V
GND
GND
VIN
ANALOG IN
A0
A1
A2
A3
A4
A5
fritzing

You can see my wiring implementation in Figure below.



To access DHT-22 on Arduino, we can use DHT Sensor library from Adafruit, https://github.com/adafruit/DHT-sensor-library. We can install this library from Arduino software. Click menu **Sketch** | **Include Library** | **Manage Libraries** so you will get a dialog.

Search dht on **LibraryManager**. You should see DHT sensor library by Adafruit. Install this library.

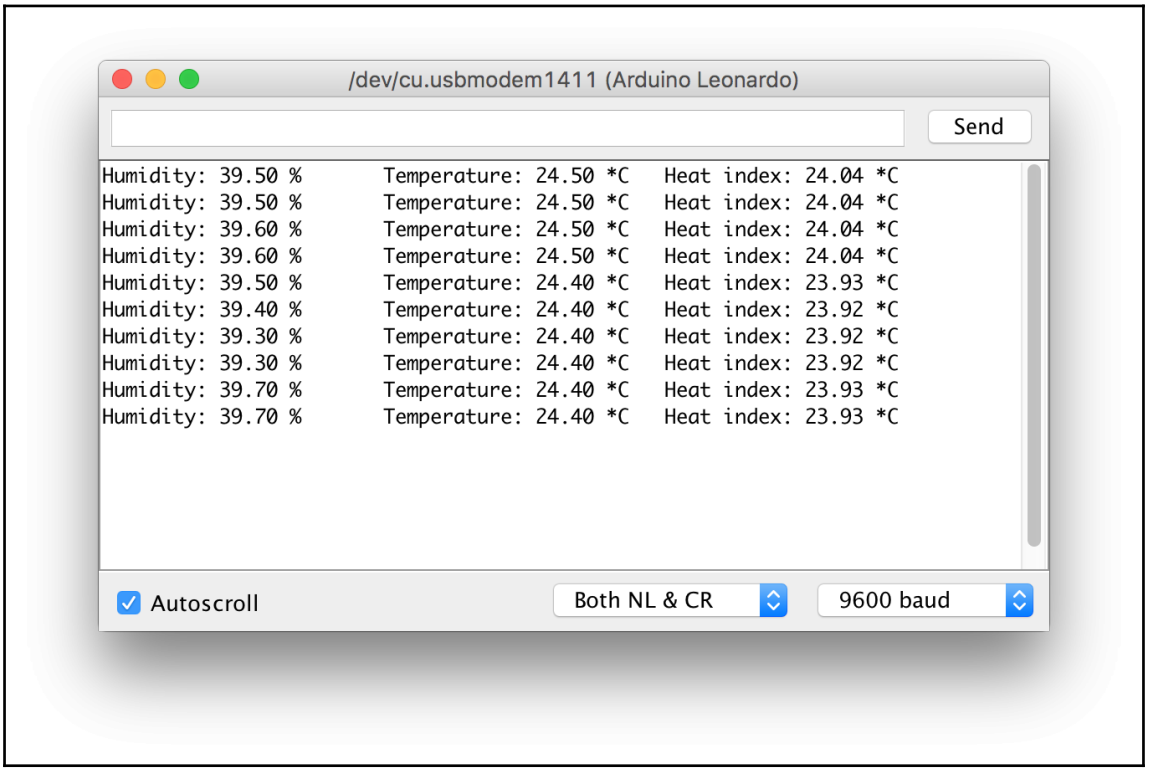


If finished installation, you can start to write program on Arduino software. The following is program sample to read temperature and humidity from DHT-22 module.

```
#include "DHT.h"

// define DHT22
#define DHTTYPE DHT22
// define pin on DHT22
#define DHTPIN 8

DHT dht(DHTPIN, DHTTYPE);

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

```
void loop() {
  delay(2000);

  // Reading temperature or humidity takes about 250 milliseconds!
  // Sensor readings may also be up to 2 seconds 'old' (its a very slow
sensor)
  float h = dht.readHumidity();
  // Read temperature as Celsius (the default)
  float t = dht.readTemperature();

  // Check if any reads failed and exit early (to try again).
  if (isnan(h) || isnan(t)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }

  // Compute heat index in Celsius (isFahreheit = false)
  float hic = dht.computeHeatIndex(t, h, false);

  Serial.print("Humidity: ");
  Serial.print(h);
  Serial.print(" %\t");
  Serial.print("Temperature: ");
  Serial.print(t);
  Serial.print(" *C\t");
  Serial.print("Heat index: ");
  Serial.print(hic);
  Serial.println(" *C ");
}
```

Save this program as ArduinoDHT. Now you can compile and upload the program into Arduino board. After that, open Serial Monitor to see temperature and humidity data.



How to work?

In `setup()` function, we initialize DHT module by calling dht.begin(). To read temperature and humidity, you can use `dht.readTemperature()` and `dht.readHumidity()`. You also can get a heat index using `dht.computeHeatIndex()` function.

Sensing and actuating on Raspberry Pi devices

Raspberry Pi board is one of board tester for experimental in this book. In this section, we use Raspberry Pi to sense and actuate with external devices. I use Raspberry Pi 3 board for testing.

Setting up

Before you use Raspberry Pi board, you should setup OS on the board. OS software can be

deployed on micro SD card. It's recommended micro SD card with 8 GB sized. There are a lot of OS software you can use on Raspberry Pi board. You can check it on https://www.raspberrypi.org/downloads/.

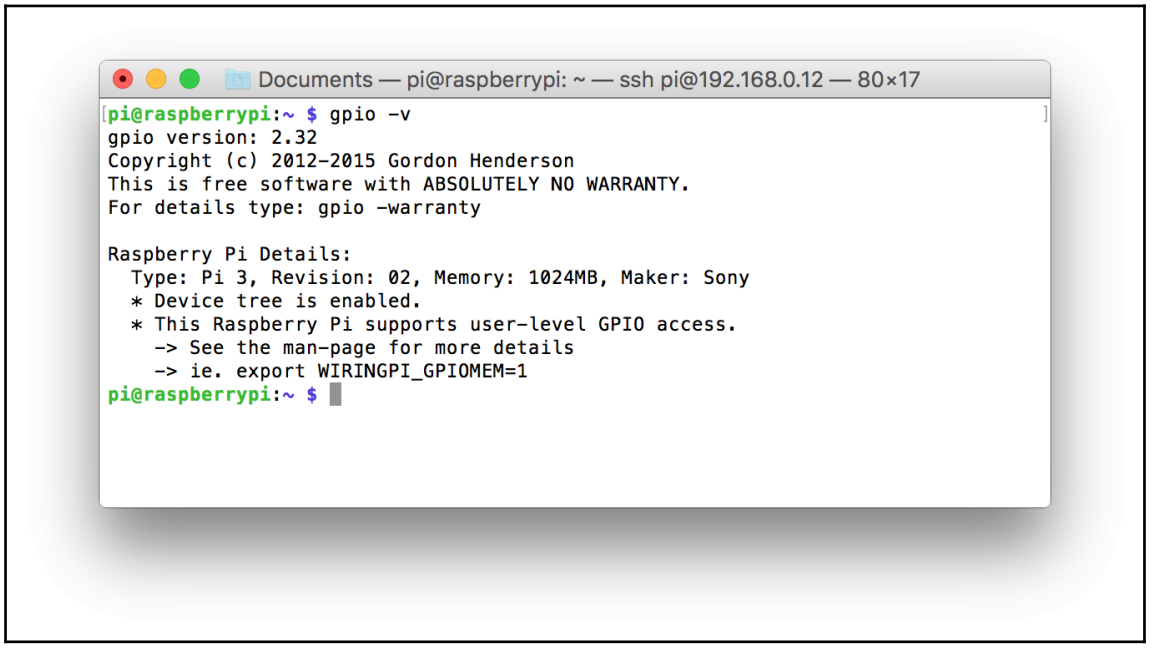
For testing, I use Raspbian, https://www.raspberrypi.org/downloads/raspbian/, for OS on Raspberry Pi board. Follow installation guideline on this site, https://www.raspberrypi.org/documentation/installation/installing-images/README.md.

Accessing Raspberry Pi GPIO

If you use the latest OS of Raspbian (Jessie or later), wiringPi module, http://wiringpi.com, has already installed for you. You can verify wiringPi version using this command on Raspberry Pi Terminal.

```
$ gpio -v
```

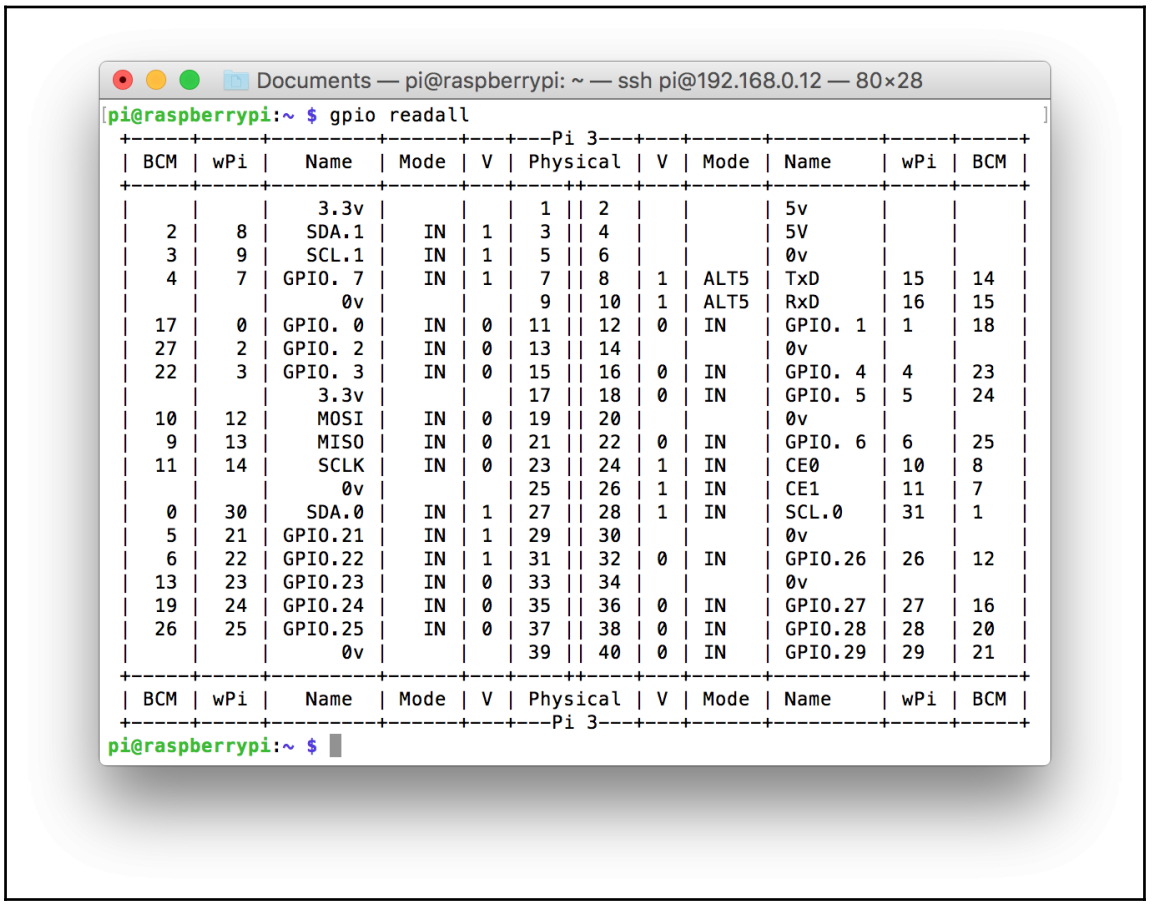
You should see wiringPi module version. A sample of program output can be seen in Figure below.



Furthermore, you can verify Raspberry GPIO layout using the following command.

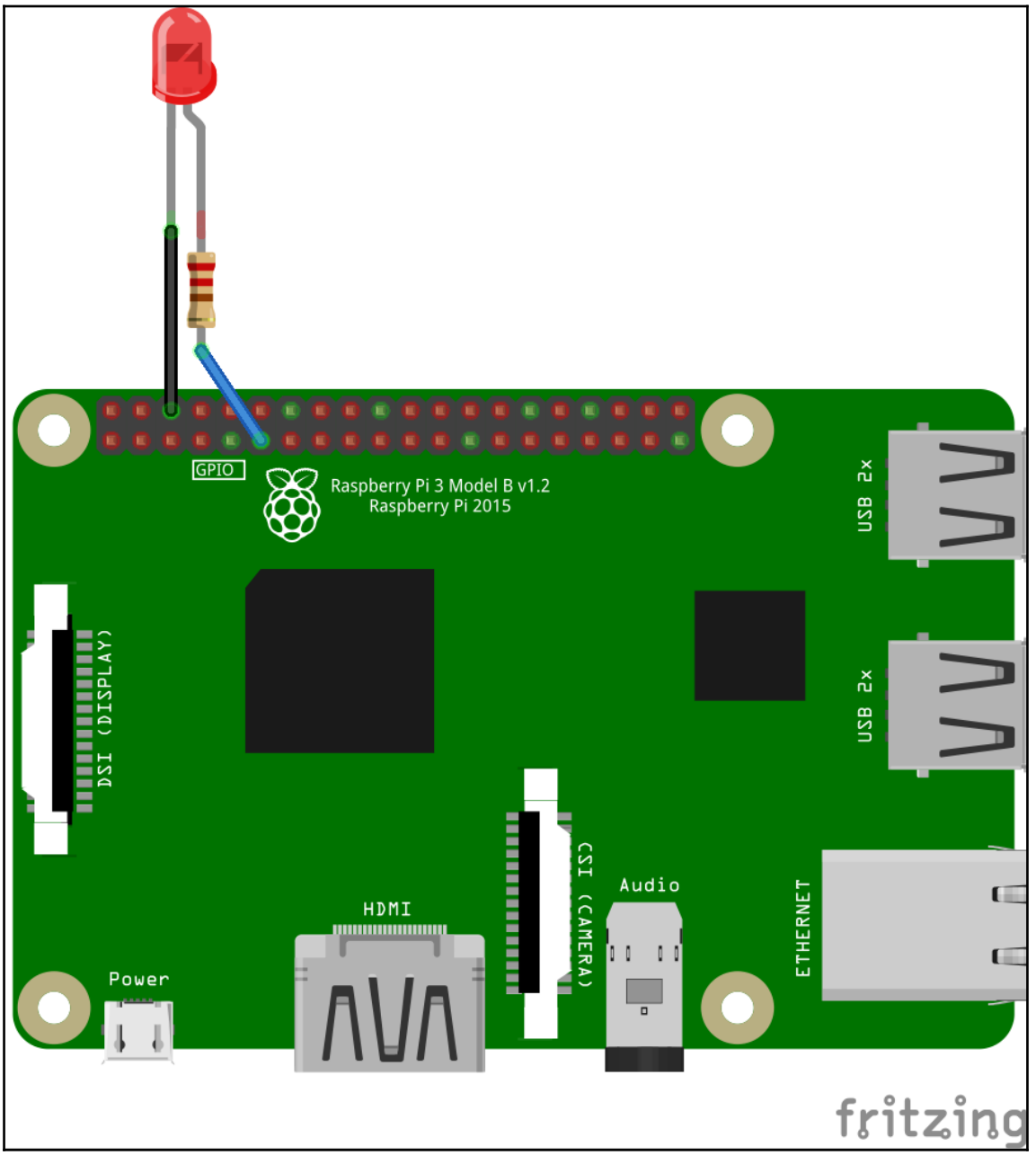
```
$ gpio – readall
```

This command will display Raspberry Pi layout. It can detect your Raspberry Pi model. A sample of program output for my board, Raspberry Pi 3, can be seen in Figure below.



For Raspberry Pi GPIO development, the latest Raspbian also already installed RPi.GPIO library, https://pypi.python.org/pypi/RPi.GPIO, for Python so we can use it directly now.

To test Raspberry Pi GPIO, we put a LED on GPIO11 (BCM 17). You can see the wiring on the following Figure.



Now you can write Python program with your own editor. Write the following program.

```
import RPi.GPIO as GPIO
import time


led_pin = 17
GPIO.setmode(GPIO.BCM)
GPIO.setup(led_pin, GPIO.OUT)


try:
    while 1:
        print("turn on led")
        GPIO.output(led_pin, GPIO.HIGH)
        time.sleep(2)
        print("turn off led")
        GPIO.output(led_pin, GPIO.LOW)
        time.sleep(2)

except KeyboardInterrupt:
    GPIO.output(led_pin, GPIO.LOW)
    GPIO.cleanup()


print("done")
```

Explanation code:

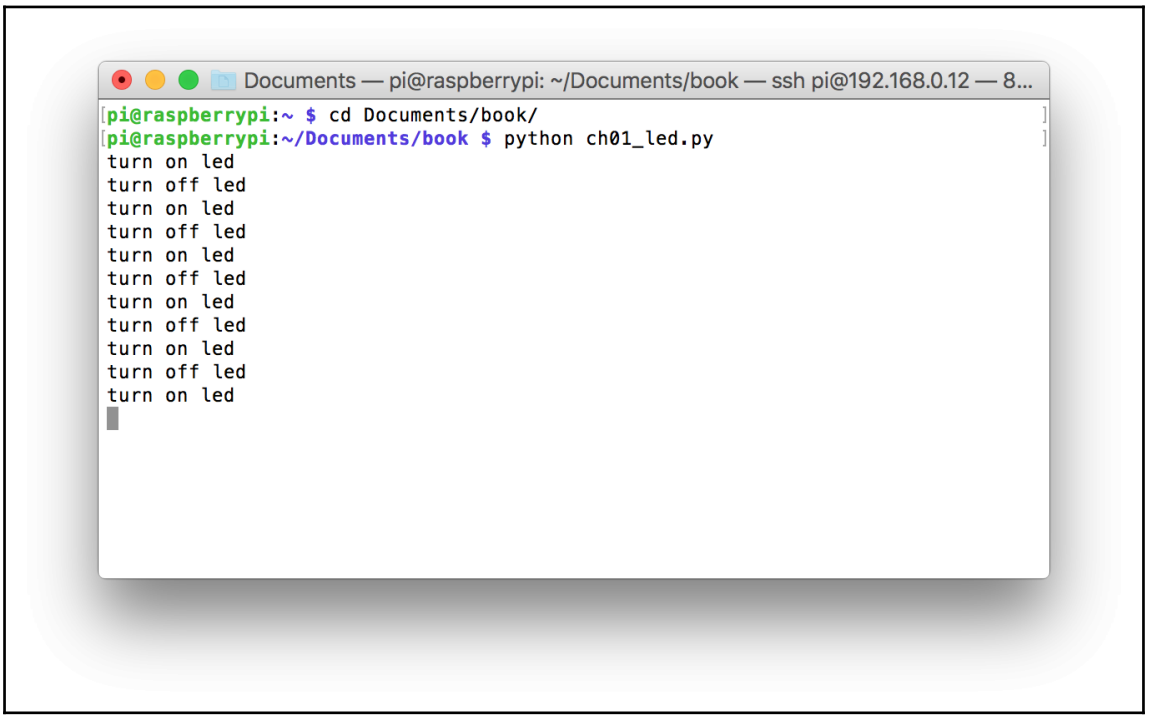
- We set GPIO type using `GPIO.setmode(GPIO.BCM)`. I use `GPIO.BCM` mode. In GPIO BCM, you should see GPIO values on BCM column from GPIO layout
- Define GPIO which will be used by calling `GPIO.setu()` as output mode
- To set digital output, we can call `GPIO.output()`. GPIO.HIGH is used to send 1 to digital output. Otherwise, GPIO.LOW is used for sending 0 to digital output

Save this program into a file, called ch01_led.py.

Now you can run the program by typing this command on Raspberry Pi Terminal.

```
$ sudo python ch01_led.py
```

You should see blinking LED and also get response from program. A sample of program output can be seen in Figure below.



Sensing through sensor devices

In this section, we explore how to sense from Raspberry Pi. We use DHT-22 to retrieve temperature and humidity on its environment.

To access DHT-22 using Python, we use Adafruit Python DHT Sensor library. You can review this module on https://github.com/adafruit/Adafruit_Python_DHT.

You need required libraries to build Adafruit Python DHT Sensor library. Type these commands on Raspberry Pi Terminal.

```
$ sudo apt-get update
$ sudo apt-get install build-essential python-dev
```

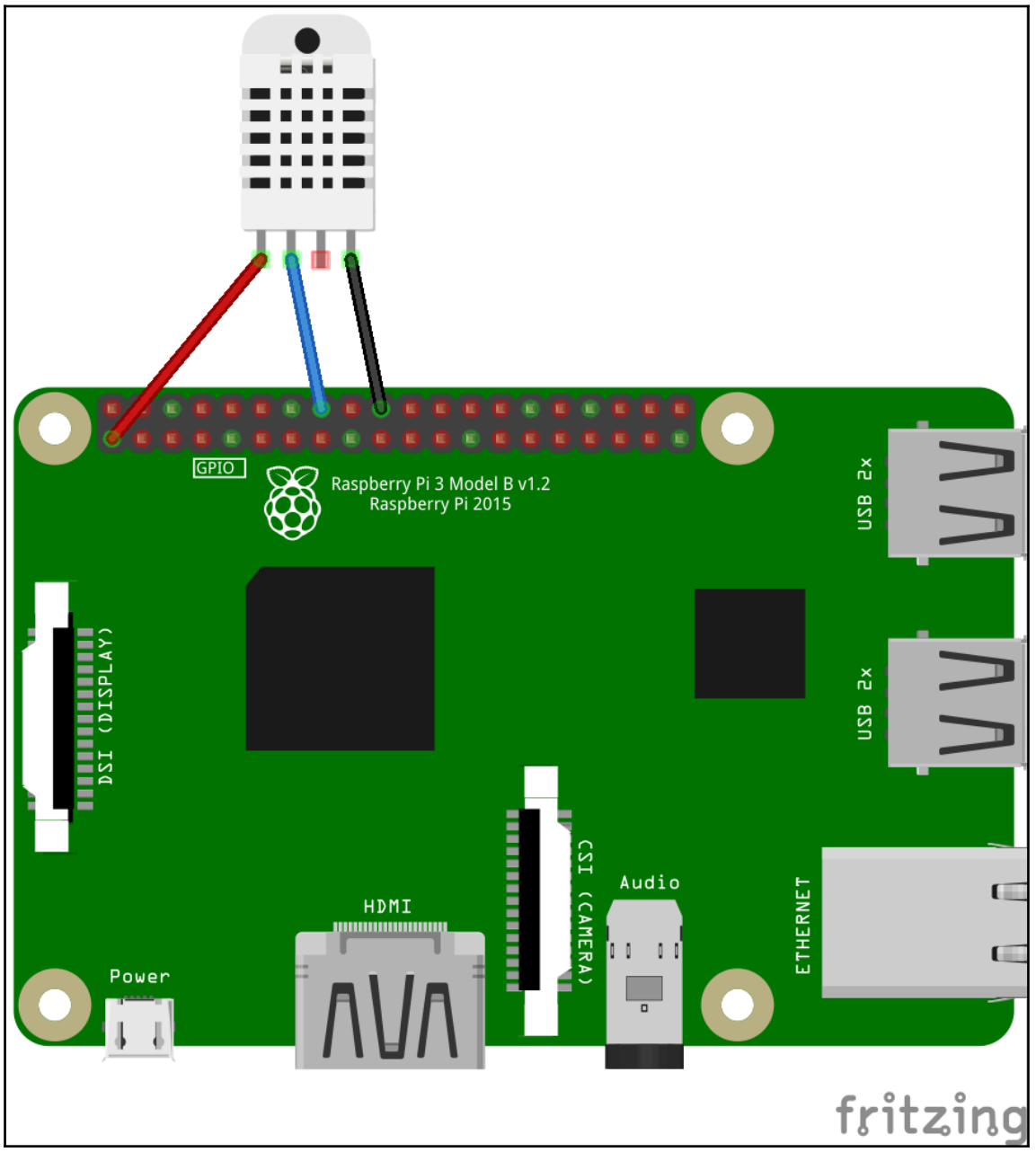
Now you can download and install Adafruit Python DHT Sensor library.

```
$ git clone https://github.com/adafruit/Adafruit_Python_DHT
$ cd Adafruit_Python_DHT/
$ sudo python setup.py install
```

If finished, we can start to build our wiring. Connect DHT-22 module into the following connection:

- DHT-22 pin 1 (VDD) is connected to 3.3V pin from Raspberry Pi
- DHT-22 pin 2 (SIG) is connected to GPIO23 (see BCM column) pin from Raspberry Pi
- DHT-22 pin 4 (GND) is connected to GND pin from Raspberry Pi

A complete of wiring is shown in Figure below.



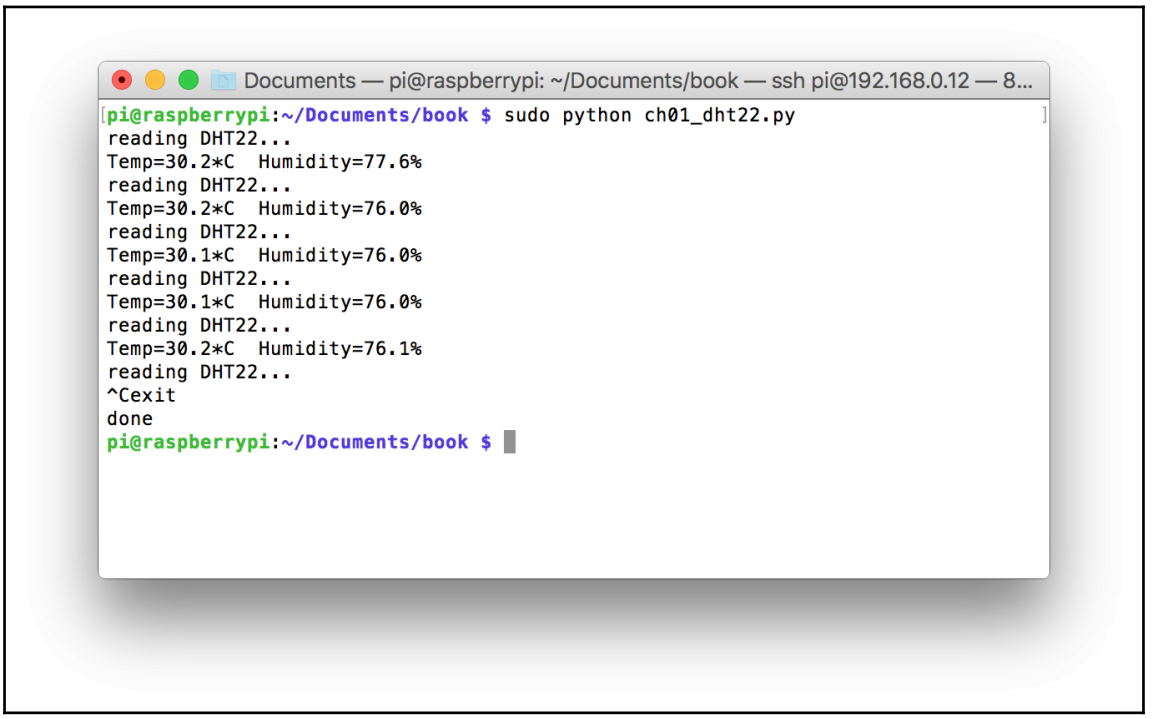
The next step is to write Python program. You can write the following codes.

```
import Adafruit_DHT
import time

sensor = Adafruit_DHT.DHT22

# DHT22 pin on Raspberry Pi
pin = 23


try:
    while 1:
        print("reading DHT22...")
        humidity, temperature = Adafruit_DHT.read_retry(sensor, pin)

        if humidity is not None and temperature is not None:
            print('Temp={0:0.1f}*C  Humidity={1:0.1f}%'.format(temperature,
humidity))

        time.sleep(2)

except KeyboardInterrupt:
    print("exit")


print("done")
```

Save this program into a file, called ch01_dht22.py. Then, you can run this file on Raspberry Pi Terminal. Type this command.

```
$ sudo python ch01_dht22.py
```

A sample of program output can be seen in Figure below.



How to work?

Firstly, we set our DHT module type by calling `Adafruit_DHT.DHT22` object. Set which DHT-22 pin is attached to Raspberry Pi board. In this case, I use GPIO23 (BCM).

To obtain temperature and humidity sensor data, we can call `Adafruit_DHT.read_retry(sensor, pin)`. Make sure the returning values are not NULL so we validate them using conditional-if.

Building a smart temperature controller for your room

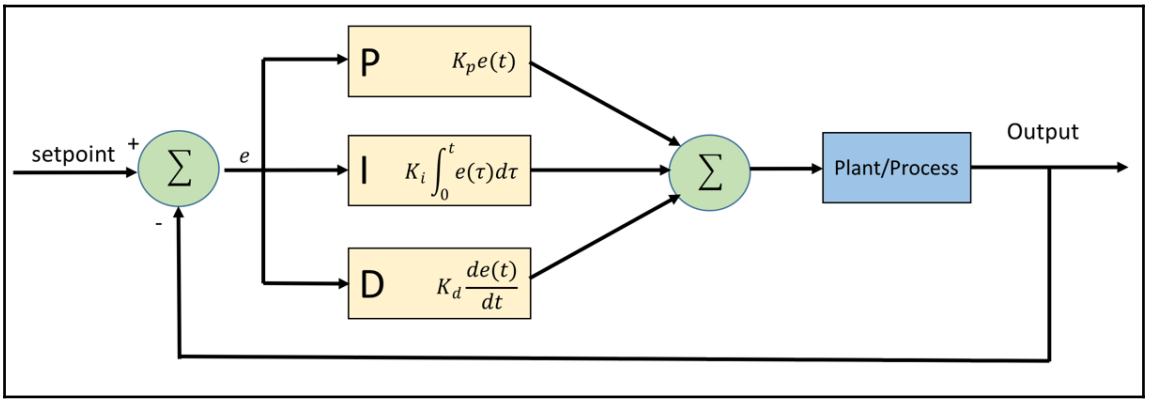
To control your room temperature, we can build a smart temperature controller. In this case, we use a PID (Proportional-Integral-Derivative) controller. When you set a certain temperature, a PID controller will change temperature by turning either cooler or heater. A PID controller program is developed using Python which runs on Raspberry Pi board.

Assume cooler and heater machines are connected via a relay. We can activate cooler and heater machine by sending HIGH signal on a relay.

Let's build!

Introducing PID controller

Proportional-Integral-Derivative (PID) control is the most common control algorithm widely used in industry and has been universally accepted in industrial control. The basic idea behind a PID controller is to read a sensor, then compute the desired actuator output by calculating proportional, integral, and derivative responses and summing those three components to compute the output. A design of general PID controller can be depicted in Figure below.



Furthermore, a PID controller formula can be defined as follows.

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt}$$

K_p, K_i, K_d represent the coefficients for the proportional, integral, and derivative. These parameters are non-negative values. The variable e represents the tracking error, the difference between the desired input value i and the actual output y. This error signal e will be sent to the PID controller.

Implementing PID controller in Python

You can translate PID controller formula into Python code easily. For implementation, I use PID class from https://github.com/ivmech/ivPID. The following is a content of `PID.py` file:

```
import time

class PID:
    """PID Controller
    """

    def __init__(self, P=0.2, I=0.0, D=0.0):

        self.Kp = P
        self.Ki = I
        self.Kd = D

        self.sample_time = 0.00
        self.current_time = time.time()
        self.last_time = self.current_time

        self.clear()

    def clear(self):
        """Clears PID computations and coefficients"""
        self.SetPoint = 0.0

        self.PTerm = 0.0
        self.ITerm = 0.0
        self.DTerm = 0.0
        self.last_error = 0.0

        # Windup Guard
        self.int_error = 0.0
        self.windup_guard = 20.0

        self.output = 0.0

    def update(self, feedback_value):
        """Calculates PID value for given reference feedback

        .. math::
            u(t) = K_p e(t) + K_i \int_{0}^{t} e(t)dt + K_d {de}/{dt}

        .. figure:: images/pid_1.png
           :align:   center

           Test PID with Kp=1.2, Ki=1, Kd=0.001 (test_pid.py)
```

```
        """
        error = self.SetPoint - feedback_value

        self.current_time = time.time()
        delta_time = self.current_time - self.last_time
        delta_error = error - self.last_error

        if (delta_time >= self.sample_time):
            self.PTerm = self.Kp * error
            self.ITerm += error * delta_time

            if (self.ITerm < -self.windup_guard):
                self.ITerm = -self.windup_guard
            elif (self.ITerm > self.windup_guard):
                self.ITerm = self.windup_guard

            self.DTerm = 0.0
            if delta_time > 0:
                self.DTerm = delta_error / delta_time

            # Remember last time and last error for next calculation
            self.last_time = self.current_time
            self.last_error = error

            self.output = self.PTerm + (self.Ki * self.ITerm) + (self.Kd *
self.DTerm)

    def setKp(self, proportional_gain):
        """Determines how aggressively the PID reacts to the current error
with setting Proportional Gain"""
        self.Kp = proportional_gain

    def setKi(self, integral_gain):
        """Determines how aggressively the PID reacts to the current error
with setting Integral Gain"""
        self.Ki = integral_gain

    def setKd(self, derivative_gain):
        """Determines how aggressively the PID reacts to the current error
with setting Derivative Gain"""
        self.Kd = derivative_gain

    def setWindup(self, windup):
        """Integral windup, also known as integrator windup or reset
windup,
        refers to the situation in a PID feedback controller where
        a large change in setpoint occurs (say a positive change)
        and the integral terms accumulates a significant error
```

```
            during the rise (windup), thus overshooting and continuing
            to increase as this accumulated error is unwound
            (offset by errors in the other direction).
            The specific problem is the excess overshooting.
            """
            self.windup_guard = windup

        def setSampleTime(self, sample_time):
            """PID that should be updated at a regular interval.
            Based on a pre-determined sampe time, the PID decides if it should
    compute or return immediately.
            """
            self.sample_time = sample_time
```

For testing, we create a simple program for simulation. We need required libraries such as numpy, scipy, pandas, patsy, and matplotlib libraries. Firstly, you should install python-dev for Python development. Type these commands on Raspberry Pi Terminal.

```
$ sudo apt-get update
$ sudo apt-get install python-dev
```

If done, you can install numpy, scipy, pandas and patsy libraries. Open Raspberry Pi Terminal and then, type these commands.

```
$ sudo apt-get install python-scipy
$ pip install numpy scipy pandas patsy
```

The last step is to install matplotlib library from source code. Type these commands on Raspberry Pi Terminal.

```
$ git clone https://github.com/matplotlib/matplotlib
$ cd matplotlib
$ python setup.py build
$ sudo python setup.py install
```

After required libraries are installed, we can test our PID.py for testing. Type this program.

```
import matplotlib
matplotlib.use('Agg')

import PID
import time
import matplotlib.pyplot as plt
import numpy as np
from scipy.interpolate import spline


P = 1.4
```

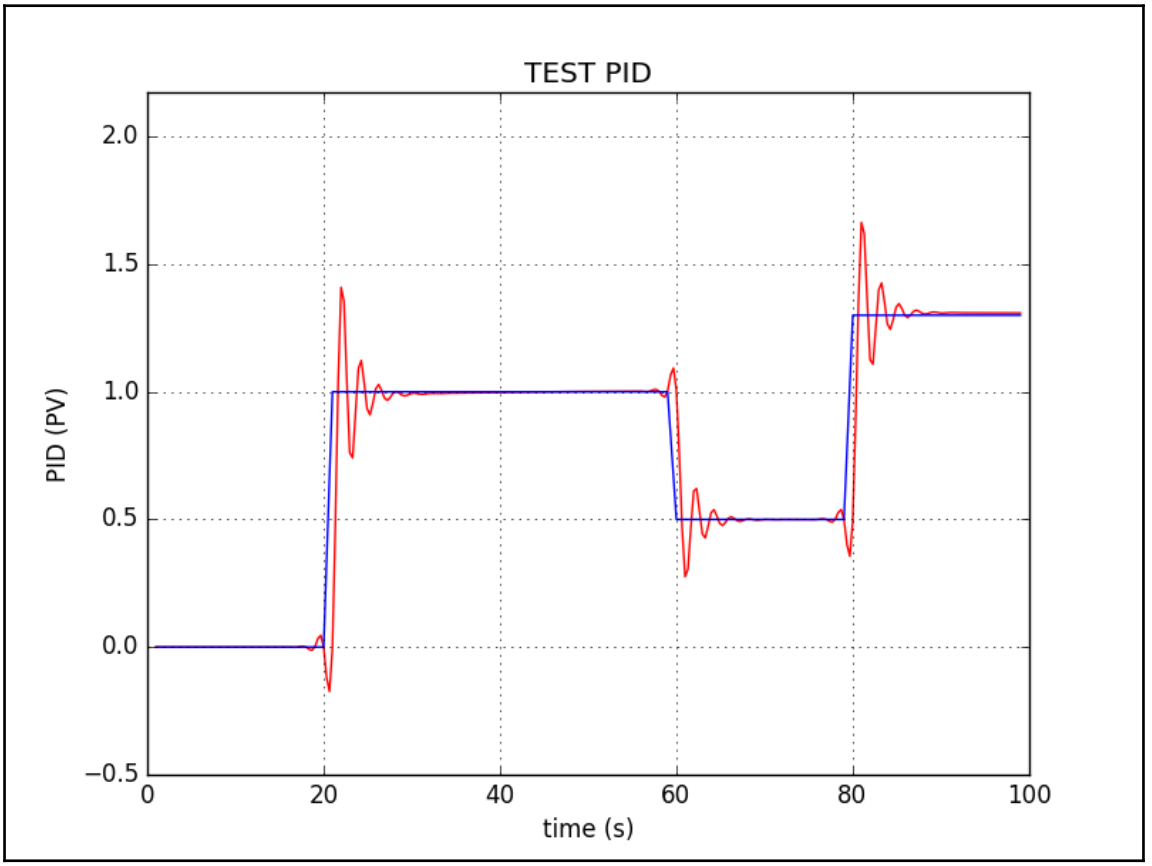
```
I = 1
D = 0.001
pid = PID.PID(P, I, D)

pid.SetPoint = 0.0
pid.setSampleTime(0.01)

total_sampling = 100
feedback = 0

feedback_list = []
time_list = []
setpoint_list = []

print("simulating....")
for i in range(1, total_sampling):
    pid.update(feedback)
    output = pid.output
    if pid.SetPoint > 0:
        feedback += (output - (1 / i))

    if 20 < i < 60:
        pid.SetPoint = 1

    if 60 <= i < 80:
        pid.SetPoint = 0.5

    if i >= 80:
        pid.SetPoint = 1.3

    time.sleep(0.02)

    feedback_list.append(feedback)
    setpoint_list.append(pid.SetPoint)
    time_list.append(i)

time_sm = np.array(time_list)
time_smooth = np.linspace(time_sm.min(), time_sm.max(), 300)
feedback_smooth = spline(time_list, feedback_list, time_smooth)

fig1 = plt.gcf()
fig1.subplots_adjust(bottom=0.15)

plt.plot(time_smooth, feedback_smooth, color='red')
plt.plot(time_list, setpoint_list, color='blue')
plt.xlim((0, total_sampling))
plt.ylim((min(feedback_list) - 0.5, max(feedback_list) + 0.5))
plt.xlabel('time (s)')
```

```
plt.ylabel('PID (PV)')
plt.title('TEST PID')


plt.grid(True)
print("saving...")
fig1.savefig('result.png', dpi=100)
```

Save this program into a file, called test_pid.py. Then, run this program.

```
$ python test_pid.py
```

This program will generate `result.png` as result of PID process. A sample of output form, result.png, is shown in Figure below. You can see the Blue line is desired values and the Red line is an output of PID.



How to work?

Firstly, we define our PID parameters.

```
P = 1.4
I = 1
D = 0.001
pid = PID.PID(P, I, D)

pid.SetPoint = 0.0
pid.setSampleTime(0.01)

total_sampling = 100
feedback = 0

feedback_list = []
time_list = []
setpoint_list = []
```

After that, we compute PID value during sampling time. In this case, we set desired output value as follows:

- desired output 1 for sampling from 20 to 60
- desired output 0.5 for sampling from 60 to 80
- desired output 1.3 for sampling more than 80

```
for i in range(1, total_sampling):
    pid.update(feedback)
    output = pid.output
    if pid.SetPoint > 0:
        feedback += (output - (1 / i))

    if 20 < i < 60:
        pid.SetPoint = 1

    if 60 <= i < 80:
        pid.SetPoint = 0.5

    if i >= 80:
        pid.SetPoint = 1.3

    time.sleep(0.02)

    feedback_list.append(feedback)
    setpoint_list.append(pid.SetPoint)
    time_list.append(i)
```

The last step is to generate a report and saved into a file, called result.png.

```
time_sm = np.array(time_list)
time_smooth = np.linspace(time_sm.min(), time_sm.max(), 300)
feedback_smooth = spline(time_list, feedback_list, time_smooth)

fig1 = plt.gcf()
fig1.subplots_adjust(bottom=0.15)

plt.plot(time_smooth, feedback_smooth, color='red')
plt.plot(time_list, setpoint_list, color='blue')
plt.xlim((0, total_sampling))
plt.ylim((min(feedback_list) - 0.5, max(feedback_list) + 0.5))
plt.xlabel('time (s)')
plt.ylabel('PID (PV)')
plt.title('TEST PID')


plt.grid(True)
print("saving...")
fig1.savefig('result.png', dpi=100)
```

Controlling room temperature using PID controller

Now we can change our PID controller simulation using the real application. We use DHT-22 to check a room temperature. The output of measurement is used as feedback input for the PID controller.

If the PID output positive value, then we turn on heater. Otherwise, we activate cooler machine. It may not good approach but this good point to show how PID controller work.

We attach DHT-22 on GPIO23 (BCM). Let's write this program.

```
import matplotlib
matplotlib.use('Agg')

import PID
import Adafruit_DHT
import time
import matplotlib.pyplot as plt
import numpy as np
from scipy.interpolate import spline

sensor = Adafruit_DHT.DHT22
```

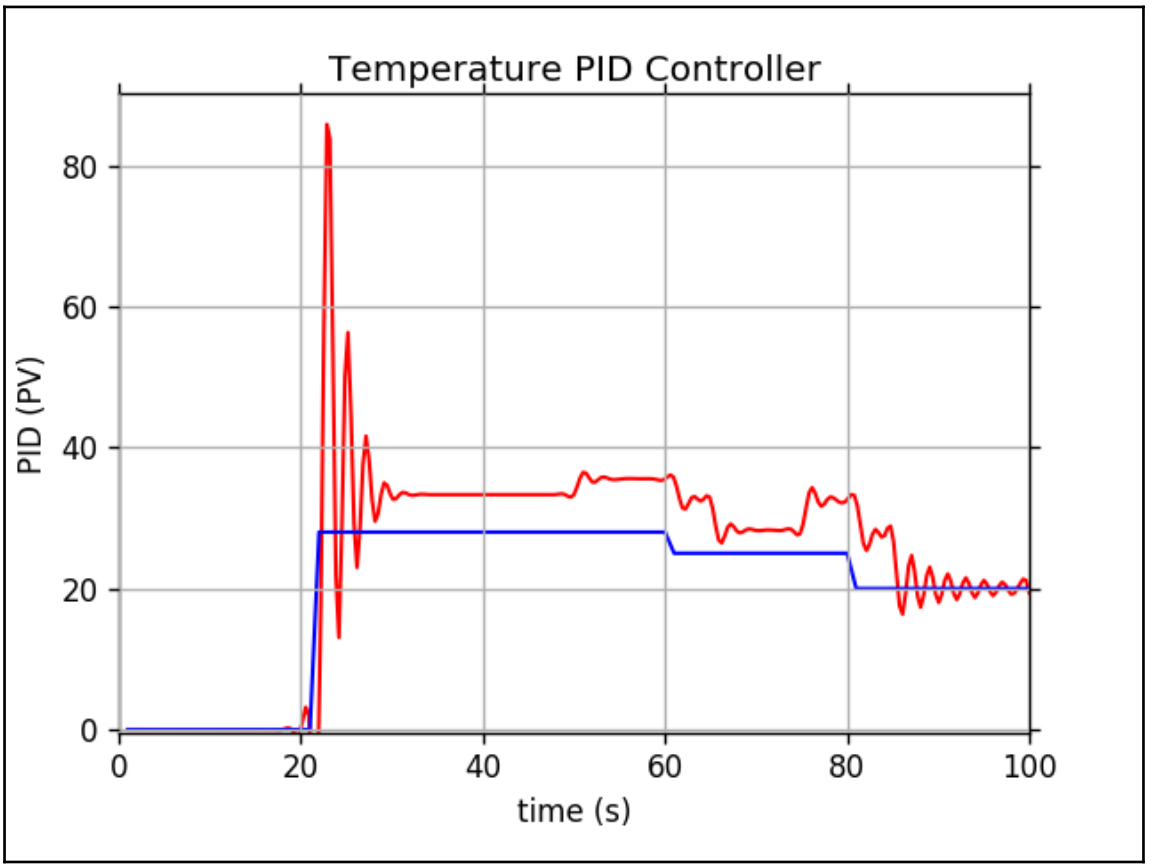
```
# DHT22 pin on Raspberry Pi
pin = 23

P = 1.4
I = 1
D = 0.001
pid = PID.PID(P, I, D)

pid.SetPoint = 0.0
pid.setSampleTime(0.25)  # a second

total_sampling = 100
sampling_i = 0
measurement = 0
feedback = 0

feedback_list = []
time_list = []
setpoint_list = []

print('PID controller is running..')
try:
    while 1:
        pid.update(feedback)
        output = pid.output

        humidity, temperature = Adafruit_DHT.read_retry(sensor, pin)
        if humidity is not None and temperature is not None:
            if pid.SetPoint > 0:
                feedback += temperature + output

            print('i={0} desired.temp={1:0.1f}*C temp={2:0.1f}*C
pid.out={3:0.1f} feedback={4:0.1f}'
                  .format(sampling_i, pid.SetPoint, temperature, output,
feedback))
            if output > 0:
                print('turn on heater')
            elif output < 0:
                print('turn on cooler')

        if 20 < sampling_i < 60:
            pid.SetPoint = 28  # celsius

        if 60 <= sampling_i < 80:
            pid.SetPoint = 25  # celsius

        if sampling_i >= 80:
            pid.SetPoint = 20  # celsius
```

```
            time.sleep(0.5)
            sampling_i += 1

            feedback_list.append(feedback)
            setpoint_list.append(pid.SetPoint)
            time_list.append(sampling_i)

            if sampling_i >= total_sampling:
                break

    except KeyboardInterrupt:
        print("exit")


    print("pid controller done.")
    print("generating a report...")
    time_sm = np.array(time_list)
    time_smooth = np.linspace(time_sm.min(), time_sm.max(), 300)
    feedback_smooth = spline(time_list, feedback_list, time_smooth)

    fig1 = plt.gcf()
    fig1.subplots_adjust(bottom=0.15, left=0.1)

    plt.plot(time_smooth, feedback_smooth, color='red')
    plt.plot(time_list, setpoint_list, color='blue')
    plt.xlim((0, total_sampling))
    plt.ylim((min(feedback_list) - 0.5, max(feedback_list) + 0.5))
    plt.xlabel('time (s)')
    plt.ylabel('PID (PV)')
    plt.title('Temperature PID Controller')


    plt.grid(True)
    fig1.savefig('pid_temperature.png', dpi=100)
    print("finish")
```

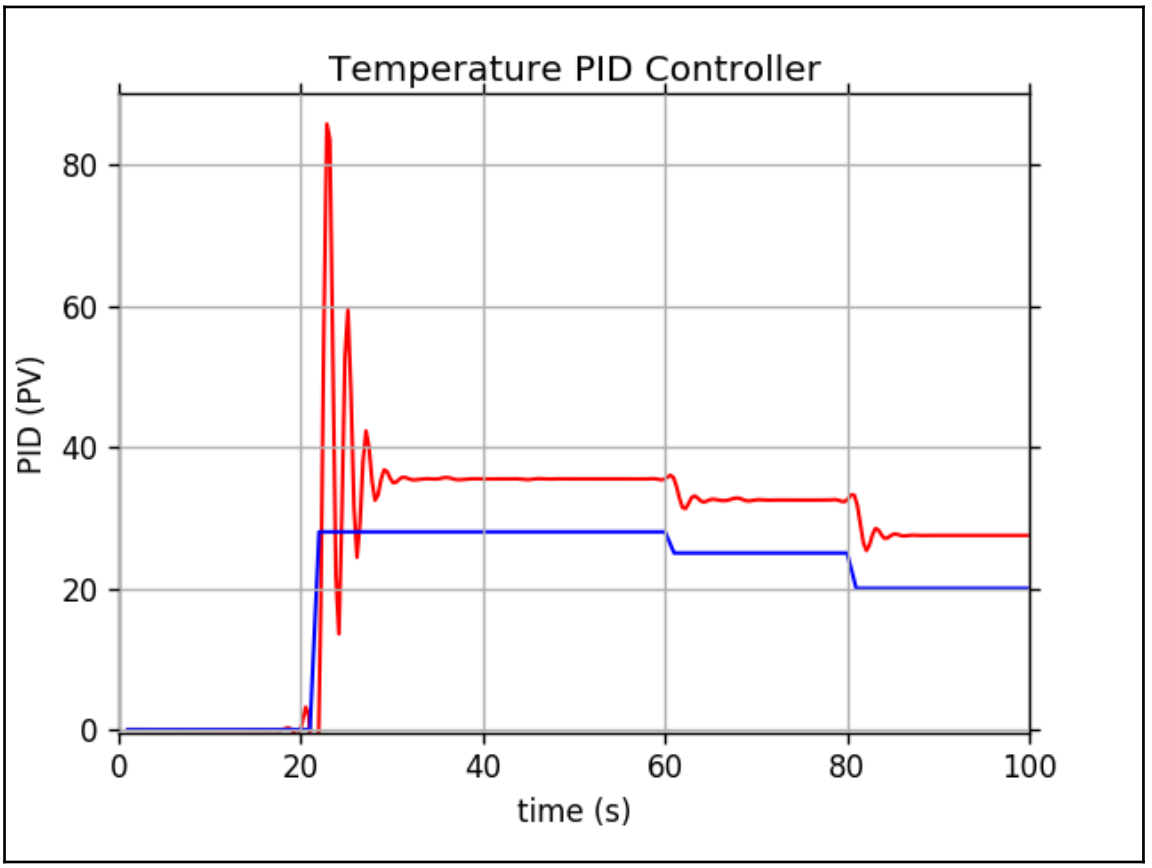
Save this program into a file, called ch01_pid.py. Now you can this program

```
$ sudo python ch01_pid.py
```

After executed the program, you should obtain a file, called pid_temperature.png. A sample output of this file can be seen in Figure below.



If I don't take action either turning on cooler or turning on heater, I obtain a result, shown in Figure below.



How to work?

Generally speaking, this program combines our two topics: reading current temperature through DHT-22 and implementing PID controller. After measured temperature, we send this value to PID controller program. The output of PID will take a certain action. In this case, it will turn on cooler and heater machines.

Summary

We have reviewed some basic Statistics and explored various Pyhon libraries related to Statistics and Data Science. We also learned several IoT device platforms and implemented how to sense and actuate.

At the last topic, we deployed a PID controller as study sample how to integrate a controller system on IoT project. The next chapter we will learn how to build a decision system for IoT project.

References

The following is a list of recommend books you can learn more about the topics in this chapter.

1. Richard D. De Veaux, Paul F. Velleman, and David E. Bock. Stats Data and Models, 4th Edition, 2015. Pearson Publishing
2. Sheldon M. Ross. Introductory Statistics. 3rd Edition. Academic Press. 2010