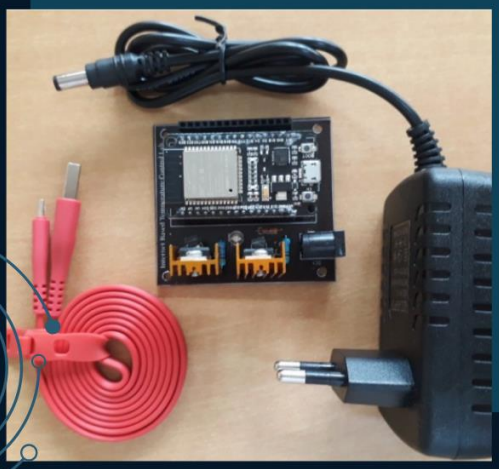
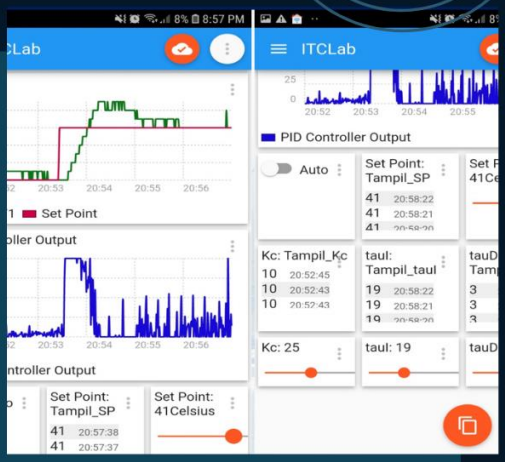
ITCLAB

INTERNET-BASED TEMPERATUE CONTROL LAB



iTCLab Kits for:
- Internet of Things (IoT)
- System Dynamics
- Control System



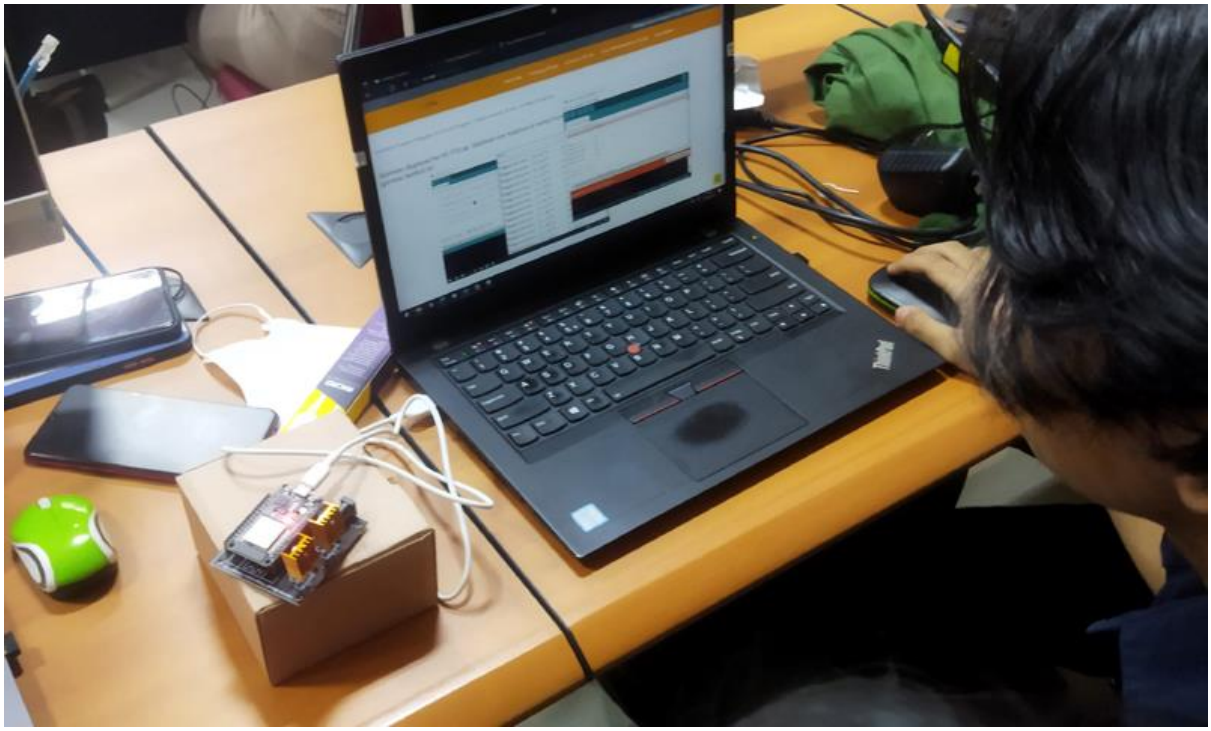
iTCLab Kits for:
- Machine Learning
- Arduino and Python programming
- System Modeling

https://io-t.net/itclab
https://github.com/bsrahmat
https://shopee.co.id/product/78709625/11589970517/

https://io-t.net/itclab
https://github.com/bsrahmat
https://shopee.co.id/product/78709625/11589970517

Python Testing

Assoc. Prof. Dr. Basuki Rahmat, S.Si, MT, ITS-AI, et al
Intelligent Control, Robotics and Automation Systems Research Group
Universitas Pembangunan Nasional "Veteran" Jawa Timur, Indonesia
i-ot.net, io-t.net
https://github.com/bsrahmat



Python Testing

Python _Testing is a simple program to test the iTCLab Kit using Python. Before the Python program is run, of course, the appropriate Arduino program must have been embedded in the iTCLab Kit.

About these iTCLab kits :

iTCLab - Internet-Based Temperatue Control Lab. Temperature control kit for feedback control applications with ESP32 microcontroller, LEDs, two heaters, and two temperature sensors. The heating power output is adjusted to maintain the desired temperature setpoint. Thermal energy from the heater is transferred by conduction, convection, and radiation to the temperature sensor. Heat is also transferred from the device to the environment.

More about iTCLab:

- Inspired by BYU (Brigham Young University) TCLab Product. A private university in Provo, Utah, United States.
- Miniature Control System in Our Pocket.
- Practical IoT Learning Package.
- Introduction to IoT Systems.
- IoT Programming.
- Practice of IoT-Based Control Systems.
- Can be used to learn System Dynamics and Control Systems.
- Can be used to learn Arduino and Python Programming.
- Can be used to learn Machine Learning Programming.
- And others.

The fundamental difference between iTCLab and BYU's TCLab product is the replacement of the Arduino Uno microcontroller with the ESP32. By using the ESP32, iTCLab has the ability to connect to the Internet of Things (IoT).

iTCLab Upper Temperature Limit Description:

The upper temperature limit of the iTCLab Kit is 60 degrees Celsius. Therefore, when experimenting with this Kit, this Upper Temperature Limit must not be exceeded. Violation of this provision could cause damage (burning) to the components.

Although the upper limit is 60 degrees Celsius, it is still sufficient for experimenting with this Kit. And it is sufficient to see the performance of a control method. For example, control using Proportional Integral and Derivative (PID). Or to see the effect of tuning the PID parameters using the Machine Learning method. An illustration of the capabilities of this iTCLab Kit can be seen from the illustration of the performance of the BYU TCLab, as seen in the following simulation.

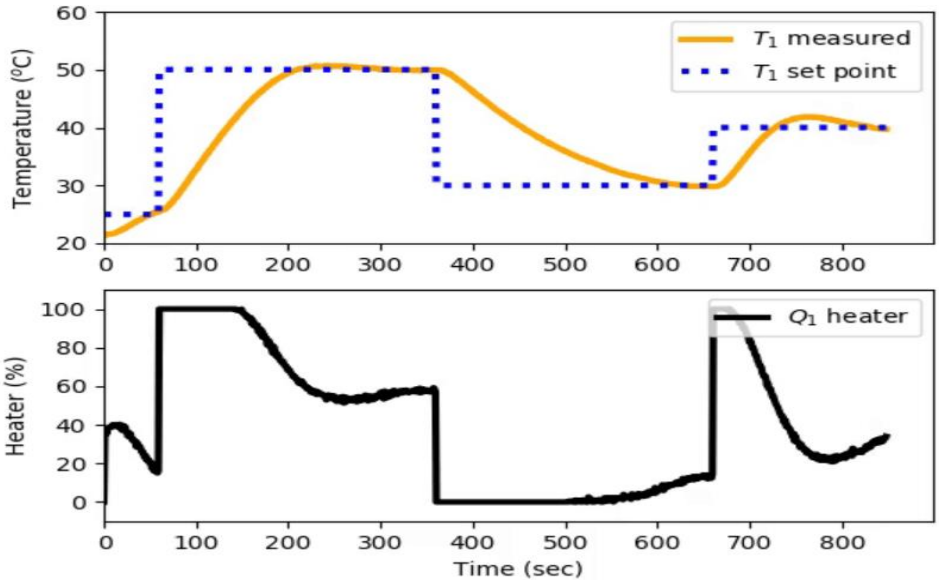


Figure 1. BYU TCLab Kit performance overview

Coding Upper Temperature Limit

It is necessary to set a limit so that the iTCLab Kit always operates in a safe area. It must not exceed the upper limit of 60 degrees Celsius. The following is an example of an Arduino program script that must be added every time you experiment with this Kit. In the Loop, it is added that if it reaches the specified upper limit (it can be lowered slightly, for example 55 degrees Celsius), then the heater must be turned off.

```
void loop() {
  // put your main code here, to run repeatedly:
  cektemp();
  if (cel > upper_temperature_limit){
    Q1off();
    ledon();
  }
  else {
    Q1on();
    ledoff();
  }
  if (cel1 > upper_temperature_limit){
    Q2off();
    ledon();
  }
  else {
    Q2on();
    ledoff();
  }
  delay (100);
}
```

Python Testing

Python _Testing is a simple program to test the iTCLab Kit using Python. Before the Python program is run, of course, the appropriate Arduino program must have been embedded in the iTCLab Kit.

Required Equipment:

- iTCLab Kit
- arduino_python.ino Program
- python_testing.ipynb Program
- itclab.py Program
- Python_Testing.pdf Tutorial

Steps:

1. Upload the arduino_python.ino program to the iTCLab Kit
2. Put the itclab.py file in the same folder as the python_testing.ipynb program
3. Run the python_testing.ipynb program via Jupyter Notebook

Here is the arduino_python.ino coding script:

Program : arduino_python.ino

```
/*
  iTCLab Internet-Based Temperature Control Lab Firmware
  Jeffrey Kantor, Initial Version
  John Hedengren, Modified
  Oct 2017
  Basuki Rahmat, Modified
  April 2022

  This firmware is loaded into the Internet-Based Temperature Control
Laboratory ESP32 to
  provide a high level interface to the Internet-Based Temperature Control
Lab. The firmware
  scans the serial port looking for case-insensitive commands:

  Q1        set Heater 1, range 0 to 100% subject to limit (0-255 int)
  Q2        set Heater 2, range 0 to 100% subject to limit (0-255 int)
  T1        get Temperature T1, returns deg C as string
  T2        get Temperature T2, returns dec C as string
  VER       get firmware version string
  X         stop, enter sleep mode

  Limits on the heater can be configured with the constants below.
*/

#include <Arduino.h>

// constants
const String vers = "1.04";     // version of this firmware
const int baud = 115200;        // serial baud rate
const char sp = ' ';            // command separator
const char nl = '\n';           // command terminator

// pin numbers corresponding to signals on the iTCLab Shield
const int pinT1   = 34;         // T1
const int pinT2   = 35;         // T2
const int pinQ1   = 32;         // Q1
const int pinQ2   = 33;         // Q2
const int pinLED  = 26;         // LED
```

```
//Q1 32 - T1 34
//Q2 33 - T2 35

// setting PWM properties
const int freq = 5000; //5000
const int ledChannel = 0;
const int Q1Channel = 1;
const int Q2Channel = 2;
const int resolutionLedChannel = 8; //Resolution 8, 10, 12, 15
const int resolutionQ1Channel = 8; //Resolution 8, 10, 12, 15
const int resolutionQ2Channel = 8; //Resolution 8, 10, 12, 15

const double upper_temperature_limit = 59;

// global variables
char Buffer[64];                // buffer for parsing serial input
String cmd;                     // command
double pv = 0;                  // pin value
float level;                    // LED level (0-100%)
double Q1 = 0;                  // value written to Q1 pin
double Q2 = 0;                  // value written to Q2 pin
int iwrite = 0;                 // integer value for writing
float dwrite = 0;               // float value for writing
int n = 10;                     // number of samples for each temperature
measurement

void parseSerial(void) {
  int ByteCount = Serial.readBytesUntil(nl,Buffer,sizeof(Buffer));
  String read_ = String(Buffer);
  memset(Buffer,0,sizeof(Buffer));

  // separate command from associated data
  int idx = read_.indexOf(sp);
  cmd = read_.substring(0,idx);
  cmd.trim();
  cmd.toUpperCase();

  // extract data. toInt() returns 0 on error
  String data = read_.substring(idx+1);
  data.trim();
  pv = data.toFloat();
}

// Q1_max = 100%
// Q2_max = 100%

void dispatchCommand(void) {
```

```
  if (cmd == "Q1") {
    Q1 = max(0.0, min(25.0, pv));
    iwrite = int(Q1 * 2.0); // 10.? max
    iwrite = max(0, min(255, iwrite));
    ledcWrite(Q1Channel,iwrite);
    Serial.println(Q1);
  }
  else if (cmd == "Q2") {
    Q2 = max(0.0, min(25.0, pv));
    iwrite = int(Q2 * 2.0); // 10.? max
    iwrite = max(0, min(255, iwrite));
    ledcWrite(Q2Channel,iwrite);
    Serial.println(Q2);
  }
  else if (cmd == "T1") {
    float mV = 0.0;
    float degC = 0.0;
    for (int i = 0; i < n; i++) {
        mV = (float) analogRead(pinT1) * 0.322265625;
        degC = degC + mV/10.0;
    }
    degC = degC / float(n);

    Serial.println(degC);
  }
  else if (cmd == "T2") {
    float mV = 0.0;
    float degC = 0.0;
    for (int i = 0; i < n; i++) {
         mV = (float) analogRead(pinT2) * 0.322265625;
         degC = degC + mV/10.0;
   }
    degC = degC / float(n);
    Serial.println(degC);
  }
  else if ((cmd == "V") or (cmd == "VER")) {
    Serial.println("TCLab Firmware Version " + vers);
  }
  else if (cmd == "LED") {
    level = max(0.0, min(100.0, pv));
    iwrite = int(level * 0.5);
    iwrite = max(0, min(50, iwrite));
    ledcWrite(ledChannel, iwrite);
    Serial.println(level);
  }
  else if (cmd == "X") {
    ledcWrite(Q1Channel,0);
    ledcWrite(Q2Channel,0);
```

```
    Serial.println("Stop");
  }
}

// check temperature and shut-off heaters if above high limit
void checkTemp(void) {
    float mV = (float) analogRead(pinT1) * 0.322265625;
    //float degC = (mV - 500.0)/10.0;
    float degC = mV/10.0;
    if (degC >= upper_temperature_limit) {
      Q1 = 0.0;
      Q2 = 0.0;
      ledcWrite(Q1Channel,0);
      ledcWrite(Q2Channel,0);
      //Serial.println("High Temp 1 (> upper_temperature_limit): ");
      Serial.println(degC);
    }
    mV = (float) analogRead(pinT2) * 0.322265625;
    //degC = (mV - 500.0)/10.0;
    degC = mV/10.0;
    if (degC >= upper_temperature_limit) {
      Q1 = 0.0;
      Q2 = 0.0;
      ledcWrite(Q1Channel,0);
      ledcWrite(Q2Channel,0);
      //Serial.println("High Temp 2 (> upper_temperature_limit): ");
      Serial.println(degC);
    }
}

// arduino startup
void setup() {
  //analogReference(EXTERNAL);
  Serial.begin(baud);
  while (!Serial) {
    ; // wait for serial port to connect.
  }

  // configure pinQ1 PWM functionalitites
  ledcSetup(Q1Channel, freq, resolutionQ1Channel);

  // attach the channel to the pinQ1 to be controlled
  ledcAttachPin(pinQ1, Q1Channel);

  // configure pinQ2 PWM functionalitites
  ledcSetup(Q2Channel, freq, resolutionQ2Channel);

  // attach the channel to the pinQ2 to be controlled
```

```
  ledcAttachPin(pinQ2, Q2Channel);

  // configure pinLED PWM functionalitites
  ledcSetup(ledChannel, freq, resolutionLedChannel);

  // attach the channel to the pinLED to be controlled
  ledcAttachPin(pinLED, ledChannel);

  ledcWrite(Q1Channel,0);
  ledcWrite(Q2Channel,0);
}

// arduino main event loop
void loop() {
  parseSerial();
  dispatchCommand();
  checkTemp();
}
```

Here is the itclab.py coding script:

```
import sys
import time
import numpy as np
try:
   import serial
except:
   import pip
   pip.main(['install','pyserial'])
   import serial
from serial.tools import list_ports

class iTCLab(object):

   def __init__(self, port=None, baud=115200):
      port = self.findPort()
      print('Opening connection')
      self.sp = serial.Serial(port=port, baudrate=baud, timeout=2)
      self.sp.flushInput()
      self.sp.flushOutput()
      time.sleep(3)
      print('iTCLab connected via Arduino on port ' + port)

   def findPort(self):
      found = False
      for port in list(list_ports.comports()):
         # Arduino Uno
         if port[2].startswith('USB VID:PID=16D0:0613'):
            port = port[0]
            found = True
```

```
        # Arduino HDuino
        if port[2].startswith('USB VID:PID=1A86:7523'):
            port = port[0]
            found = True
        # Arduino Leonardo
        if port[2].startswith('USB VID:PID=2341:8036'):
            port = port[0]
            found = True
        # Arduino ESP32
        if port[2].startswith('USB VID:PID=10C4:EA60'):
            port = port[0]
            found = True
        # Arduino ESP32 - Tipe yg berbeda
        if port[2].startswith('USB VID:PID=1A86:55D4'):
            port = port[0]
            found = True
    if (not found):
        print('Arduino COM port not found')
        print('Please ensure that the USB cable is connected')
        print('--- Printing Serial Ports ---')
        for port in list(serial.tools.list_ports.comports()):
            print(port[0] + ' ' + port[1] + ' ' + port[2])
        print('For Windows:')
        print('  Open device manager, select "Ports (COM & LPT)"')
        print('  Look for COM port of Arduino such as COM4')
        print('For MacOS:')
        print('  Open terminal and type: ls /dev/*.')
        print('  Search for /dev/tty.usbmodem* or /dev/tty.usbserial*. The port number is *.')
        print('For Linux')
        print('  Open terminal and type: ls /dev/tty*')
        print('  Search for /dev/ttyUSB* or /dev/ttyACM*. The port number is *.')
        print('')
        port = input('Input port: ')
        # or hard-code it here
        #port = 'COM3' # for Windows
        #port = '/dev/tty.wchusbserial1410' # for MacOS
    return port

def stop(self):
    return self.read('X')

def version(self):
    return self.read('VER')

@property
def T1(self):
    self._T1 = float(self.read('T1'))
    return self._T1

@property
def T2(self):
    self._T2 = float(self.read('T2'))
    return self._T2

def LED(self,pwm):
    pwm = max(0.0,min(100.0,pwm))/2.0
    self.write('LED',pwm)
```

```
    return pwm

def Q1(self,pwm):
    pwm = max(0.0,min(100.0,pwm))
    self.write('Q1',pwm)
    return pwm

def Q2(self,pwm):
    pwm = max(0.0,min(100.0,pwm))
    self.write('Q2',pwm)
    return pwm

# save txt file with data and set point
# t = time
# u1,u2 = heaters
# y1,y2 = tempeatures
# sp1,sp2 = setpoints
def save_txt(self,t,u1,u2,y1,y2,sp1,sp2):
    data = np.vstack((t,u1,u2,y1,y2,sp1,sp2))  # vertical stack
    data = data.T             # transpose data
    top = 'Time (sec), Heater 1 (%), Heater 2 (%), ' \
      + 'Temperature 1 (degC), Temperature 2 (degC), ' \
      + 'Set Point 1 (degC), Set Point 2 (degC)'
    np.savetxt('data.txt',data,delimiter=',',header=top,comments='')

def read(self,cmd):
    cmd_str = self.build_cmd_str(cmd,'')
    try:
        self.sp.write(cmd_str.encode())
        self.sp.flush()
    except Exception:
        return None
    return self.sp.readline().decode('UTF-8').replace("\r\n", "")

def write(self,cmd,pwm):
    cmd_str = self.build_cmd_str(cmd,(pwm,))
    try:
        self.sp.write(cmd_str.encode())
        self.sp.flush()
    except:
        return None
    return self.sp.readline().decode('UTF-8').replace("\r\n", "")

def build_cmd_str(self,cmd, args=None):
    """
    Build a command string that can be sent to the arduino.

    Input:
        cmd (str): the command to send to the arduino, must not
            contain a % character
        args (iterable): the arguments to send to the command
    """
    if args:
        args = ' '.join(map(str, args))
    else:
        args = ''
    return "{cmd} {args}\n".format(cmd=cmd, args=args)
```

```
def close(self):
    try:
        self.sp.close()
        print('Arduino disconnected successfully')
    except:
        print('Problems disconnecting from Arduino.')
        print('Please unplug and reconnect Arduino.')
    return True
```

Here is the python_testing.ipynb coding script:

```
import itclab
import time
# Connect to Arduino
a = itclab.iTCLab()
print('LED On')
a.LED(100)
# Pause for 1 second
time.sleep(1.0)
print('LED Off')
a.LED(0)
a.close()
```

```
import itclab
import time

# Connect to Arduino
a = itclab.iTCLab()

# Get Version
print(a.version)

# Turn LED on
print('LED On')
a.LED(100)

# Taper LED off
for i in range(100,-1,-10):
    print('LED Power ' + str(i))
    time.sleep(0.5)
    a.LED(i)

a.close()
```

Please run it in the Jupyter notebook environment, with the iTCLab Kit connected to a PC or Laptop (with the arduino_python.ino program already embedded in it). Then the results should be as shown in Figure 2 and Figure 3.

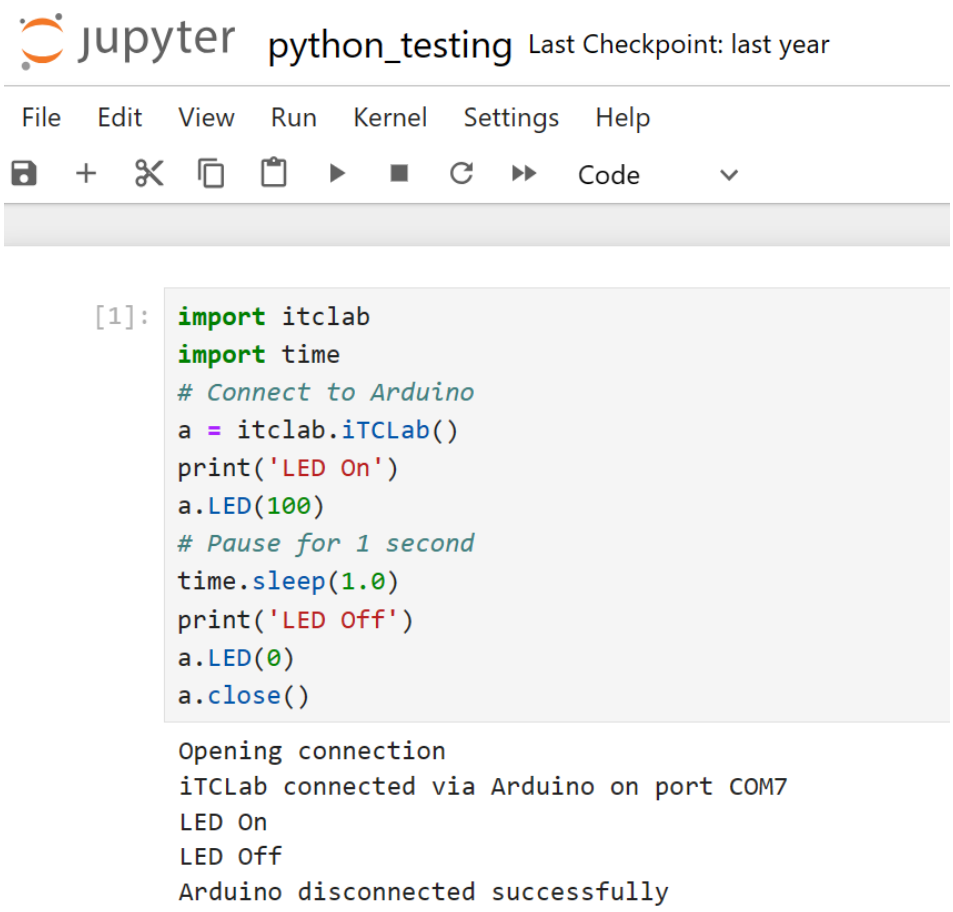


Figure 2. Test Results of the iTCLab Kit with Python

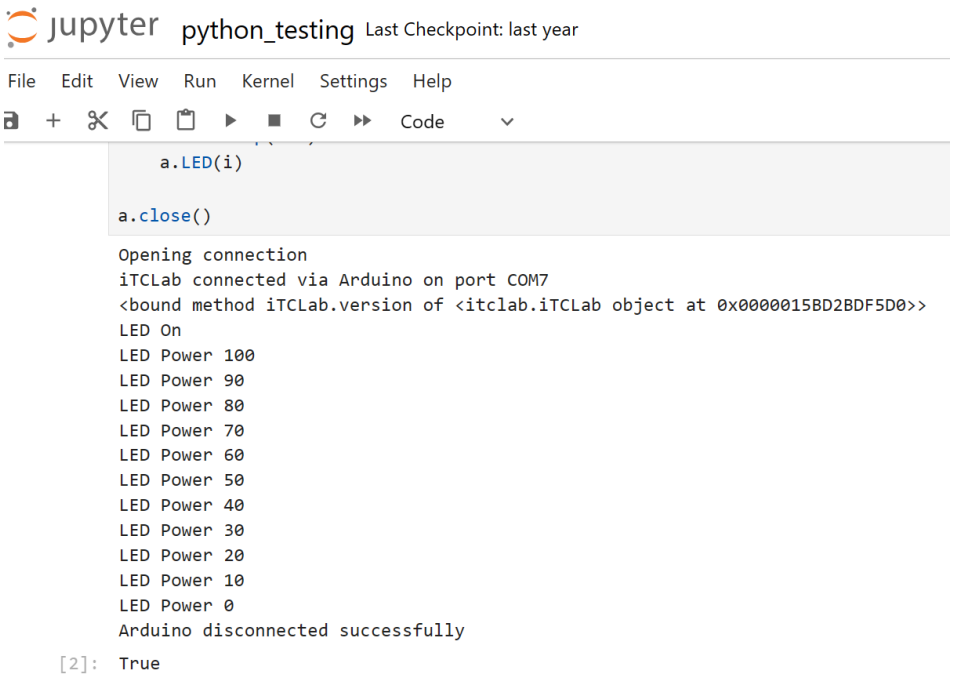


Figure 3. Other Test Results of the iTCLab Kit with Python