Student name:

Student ID:

SIT225: Data Capture Technologies

Activity 3.1: Arduino IoT Cloud and Dashboard with Arduino Nano 33 IoT devices

Arduino Cloud is your next exciting journey to build, control and monitor your connected projects. You can connect anything to Arduino Cloud including a wide range of compatible Arduino boards such as Arduino Nano 33 IoT or a third-party device that speaks Python. Arduino Cloud is an all-in-one IoT solution that empowers makers to create from anywhere, control their devices with stunning dashboards.

In this activity, you will connect your Arduino board to the cloud as a device, register a thing (in terms of a cloud variable) with your device, create a dashboard with a graphical widget which show value that is sent from the sketch running in your Arduino board.

Hardware Required

Arduino Nano 33 IoT Board

Wi-Fi hotspot (preferably your smartphone hotspot)

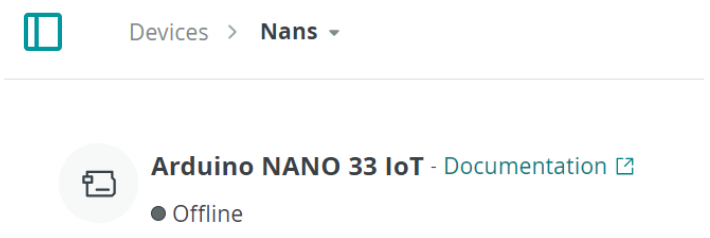
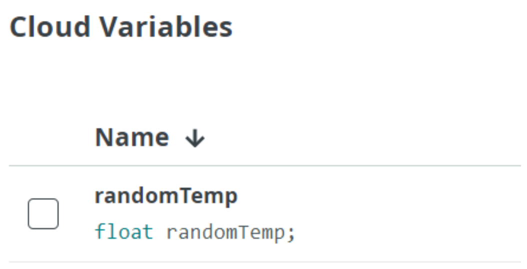
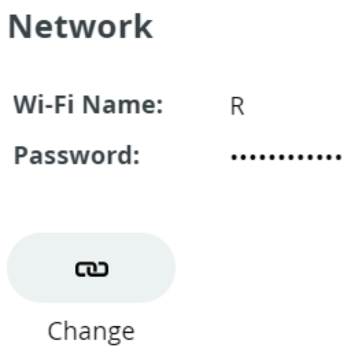
USB cable

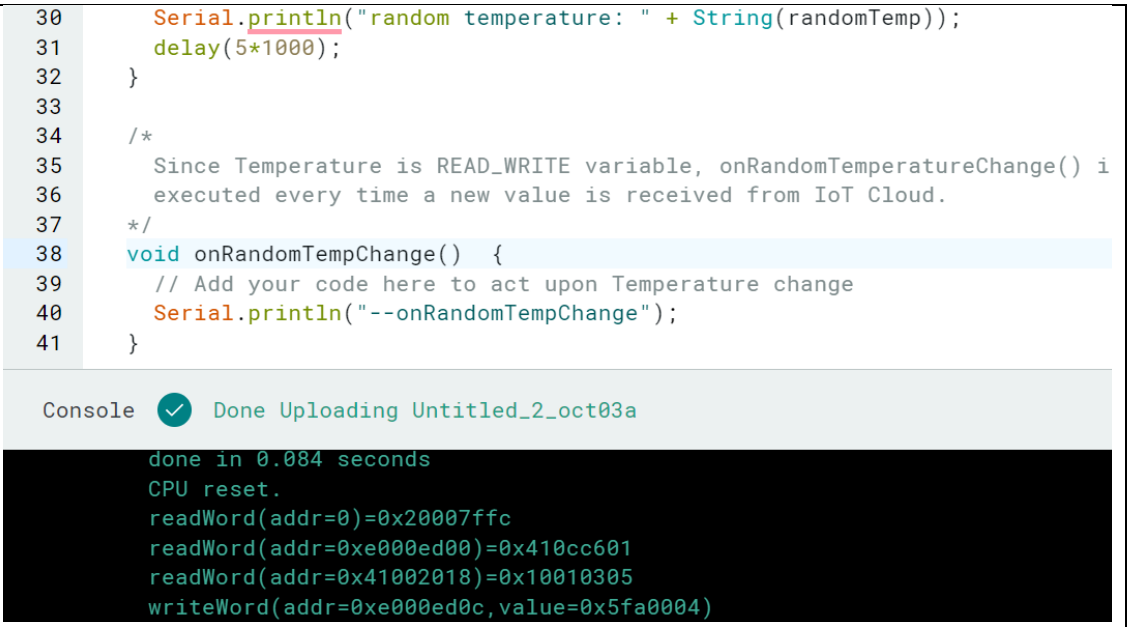
Software Required

Arduino programming environment (Arduino IDE)
Arduino IoT Cloud (https://app.arduino.cc)

Steps

Step	Action
1	**Account creation:** Create an account in Arduino IoT Cloud (https://app.arduino.cc). Once done, login to your account.

	You can follow Arduino tutorial (https://support.arduino.cc/hc/en-us/articles/360016495559-Add-and-connect-a-device-to-Arduino-Cloud) for detail. This tutorial is referred to in the steps below.
2	**Add the device**:  Devices > Nans ▾ Arduino NANO 33 IoT - Documentation ● Offline
3	**Create and configure a Thing**:  Cloud Variables Name ↓ randomTemp `float randomTemp;`
4	**Configure your Wi-Fi**:  Network Wi-Fi Name: R Password: •••••••••••• Change
5	**Sketch tab**:



```
    Serial.println("random temperature: " + String(randomTemp));
    delay(5*1000);
  }

  /*
    Since Temperature is READ_WRITE variable, onRandomTemperatureChange() i
    executed every time a new value is received from IoT Cloud.
  */
  void onRandomTempChange()  {
    // Add your code here to act upon Temperature change
    Serial.println("--onRandomTempChange");
  }
```

Console Done Uploading Untitled_2_oct03a

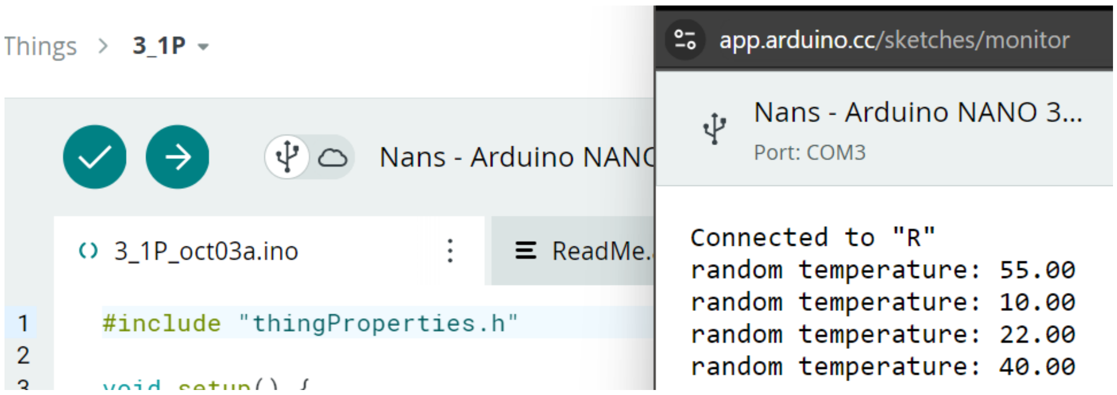
```
done in 0.084 seconds
CPU reset.
readWord(addr=0)=0x20007ffc
readWord(addr=0xe000ed00)=0x410cc601
readWord(addr=0x41002018)=0x10010305
writeWord(addr=0xe000ed0c,value=0x5fa0004)
```

6

Deploy code to board:
You can upload code to Arduino board from Arduino IDE. You can observe the random temperature readings in the serial monitor.

Question: Screenshot the output of the serial monitor with random temperature values along with the initial Wi-Fi connection codes. Comment on the output lines.

Answer:

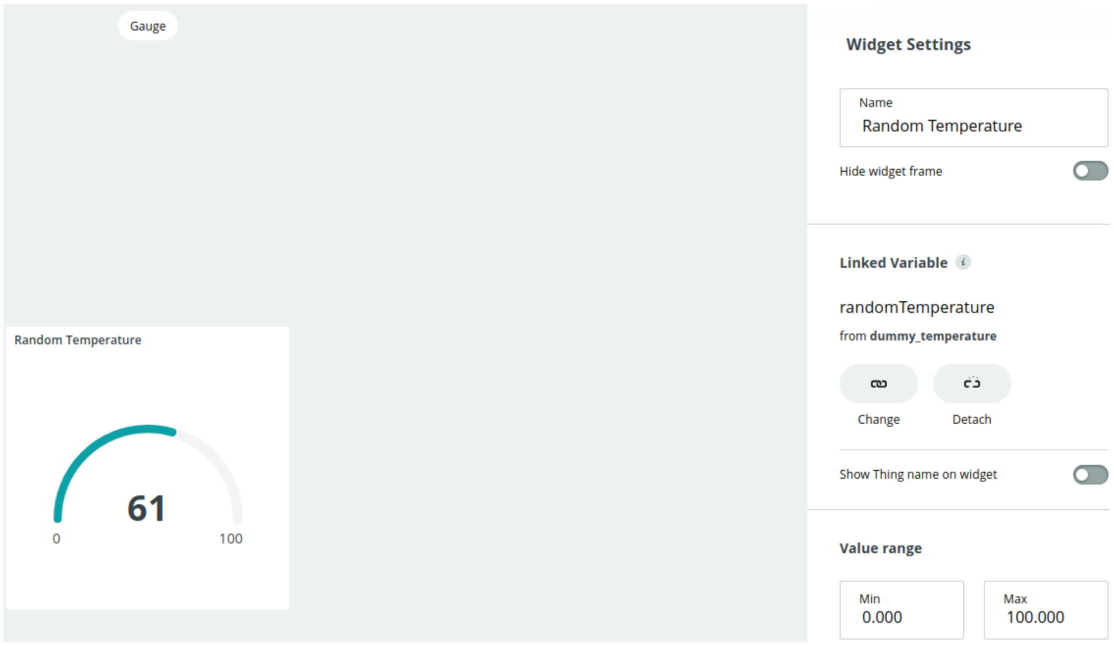
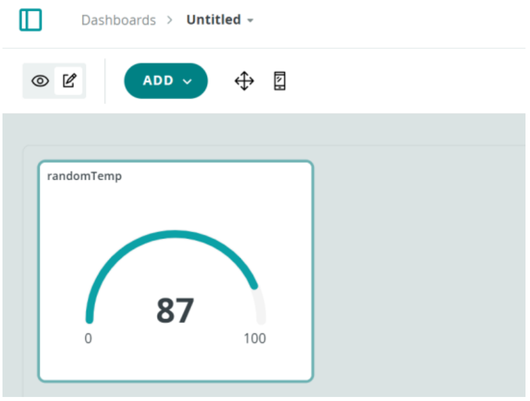


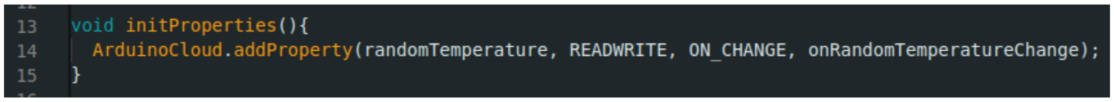
```
#include "thingProperties.h"

void setup() {
```

```
Connected to "R"
random temperature: 55.00
random temperature: 10.00
random temperature: 22.00
random temperature: 40.00
```

7

Create a dashboard:
A dashboard in Arduino Cloud can be created to visualse the sensor readings sent by the Things connected to your Arduino Nano 33 IoT board. A list of Dashboard widgets is described in this tutorial (https://docs.arduino.cc/arduino-cloud/cloud-interface/dashboard-widgets).

	You can create a new dashboard from the Dashboards left menu items (https://app.arduino.cc/dashboards) where there are other menu items such as Devices and Things you have seen earlier. Creating a dashboard is simply choosing a dashboard widget, such as a Guage and link it to the Thing cloud variable you have created.  Widget Settings Name: Random Temperature Hide widget frame Linked Variable randomTemperature from **dummy_temperature** Show Thing name on widget Value range Min: 0.000 Max: 100.000 Once you click the Done button, you should look the Gauge widget is updating based on the value shown in the Arduino IDE serial monitor. Question: Screenshot the output of the serial monitor with random temperature values and the dashboard output so the Gauge value can be found in the serial monitor output. Answer: 

8	Question: If you recall, there is a function *initProperties()* in *thingProperties.h* file where there is a single line -
	
	
	`void initProperties(){`
	`  ArduinoCloud.addProperty(randomTemperature, READWRITE, ON_CHANGE, onRandomTemperatureChange);`
	`}`
	A function *onRandomTemperatureChange()* exists (in sketch_iot_cloud.ino file) to respond to ON_CHANGE event. Can you explore what is the use of this function and when the ON_CHANGE event will trigger?
	
	Answer:
	The **onRandomTemperatureChange()** function is triggered whenever the value of the cloud variable randomTemperature is changed from the **Arduino IoT Cloud**.
	It handles the **ON_CHANGE** event, which occurs when the cloud pushes a new value to the variable.
	This function allows to respond to changes made in the cloud and perform actions accordingly (e.g., print messages, trigger alarms, control hardware, etc.).

Activity 3.2: Arduino IoT Cloud with custom Python devices

You can connect anything to Arduino Cloud including a wide range of compatible Arduino boards such as Arduino Nano 33 IoT or a third-party device that **speaks Python**.

In this activity, you will connect a custom Python board to Arduino IoT Cloud and synchronise to another cloud variable, the one you created earlier, *randomTemperature*. This will enable you to receive data from Arduino board in your Python script.

Hardware Required

Arduino Nano 33 IoT Board

Wi-Fi hotspot (preferably your smartphone hotspot)

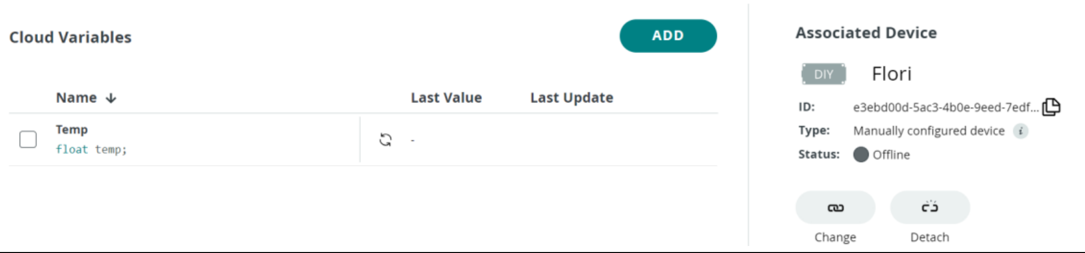
USB cable

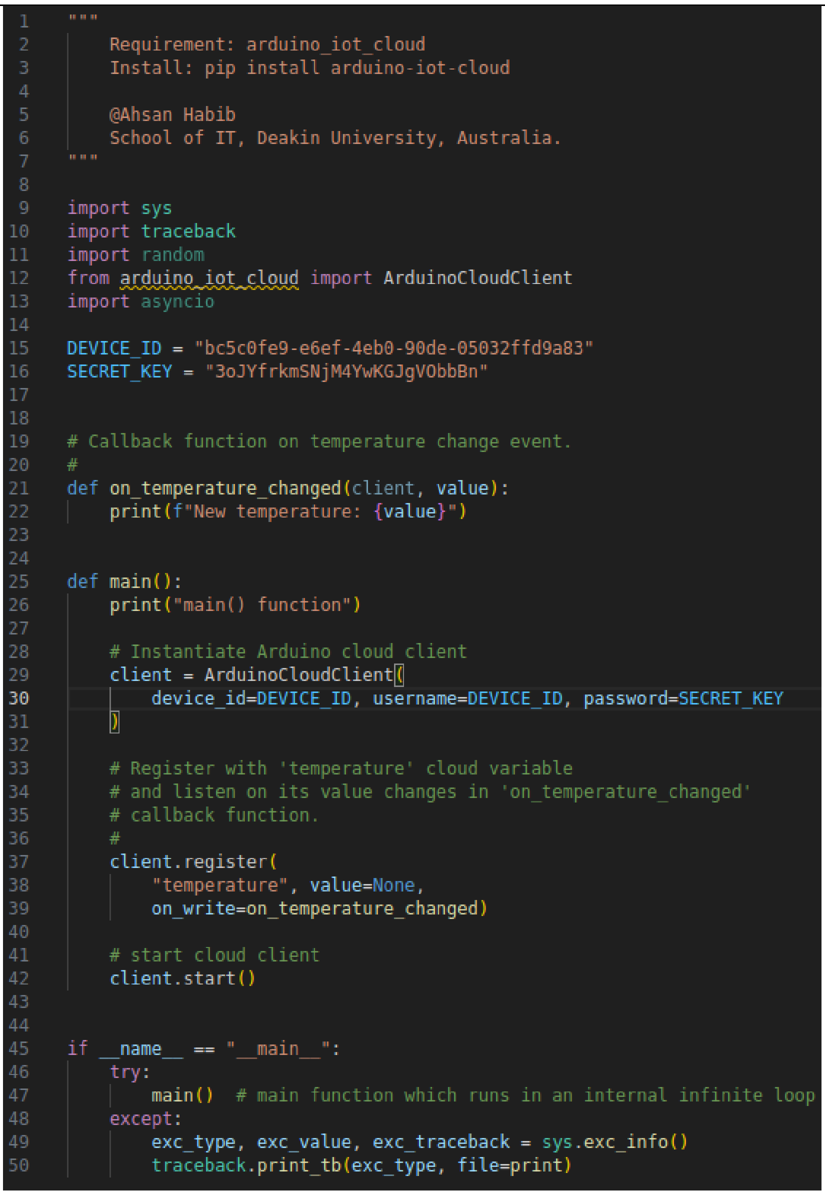
Software Required

Arduino programming environment (Arduino IDE)
Arduino IoT Cloud (https://app.arduino.cc)
Python 3

Steps

Step	Action
1	**Thing & Device Configuration**: Following the tutorial (https://docs.arduino.cc/arduino-cloud/guides/python), a. Create a new Thing, by clicking on the "Create Thing" button. b. Click on the "Select Device" in the "Associated Devices" section of your Thing. c. Click on "Set Up New Device" and select the bottom category ("Manual Device"). Click continue in the next window and choose a name for your device. d. Finally, you will see a new Device ID and a Secret Key generate. You can download them as a PDF. Make sure to save it as you cannot access your Secret Key again.
2	**Create Variables**:

	Follow the same tutorial mentioned above and do the following steps:  Cloud Variables ADD Name Last Value Last Update Temp `float temp;` - Associated Device DIY Flori ID: e3ebd00d-5ac3-4b0e-9eed-7edf... Type: Manually configured device Status: Offline Change Detach
3	**Create Python script**: Now you need to create a Python script to register to Arduino Cloud, register for a cloud variable called temperature and write a callback function for on_write event handling. You can write the code as below or download the code from here (https://github.com/deakin-deep-dreamer/sit225/blob/main/week_3/arduino_variable_sync.py).



```python
"""
    Requirement: arduino_iot_cloud
    Install: pip install arduino-iot-cloud

    @Ahsan Habib
    School of IT, Deakin University, Australia.
"""

import sys
import traceback
import random
from arduino_iot_cloud import ArduinoCloudClient
import asyncio

DEVICE_ID = "bc5c0fe9-e6ef-4eb0-90de-05032ffd9a83"
SECRET_KEY = "3oJYfrkmSNjM4YwKGJgVObbBn"


# Callback function on temperature change event.
#
def on_temperature_changed(client, value):
    print(f"New temperature: {value}")


def main():
    print("main() function")

    # Instantiate Arduino cloud client
    client = ArduinoCloudClient(
        device_id=DEVICE_ID, username=DEVICE_ID, password=SECRET_KEY
    )

    # Register with 'temperature' cloud variable
    # and listen on its value changes in 'on_temperature_changed'
    # callback function.
    #
    client.register(
        "temperature", value=None,
        on_write=on_temperature_changed)

    # start cloud client
    client.start()


if __name__ == "__main__":
    try:
        main()  # main function which runs in an internal infinite loop
    except:
        exc_type, exc_value, exc_traceback = sys.exc_info()
        traceback.print_tb(exc_type, file=print)
```

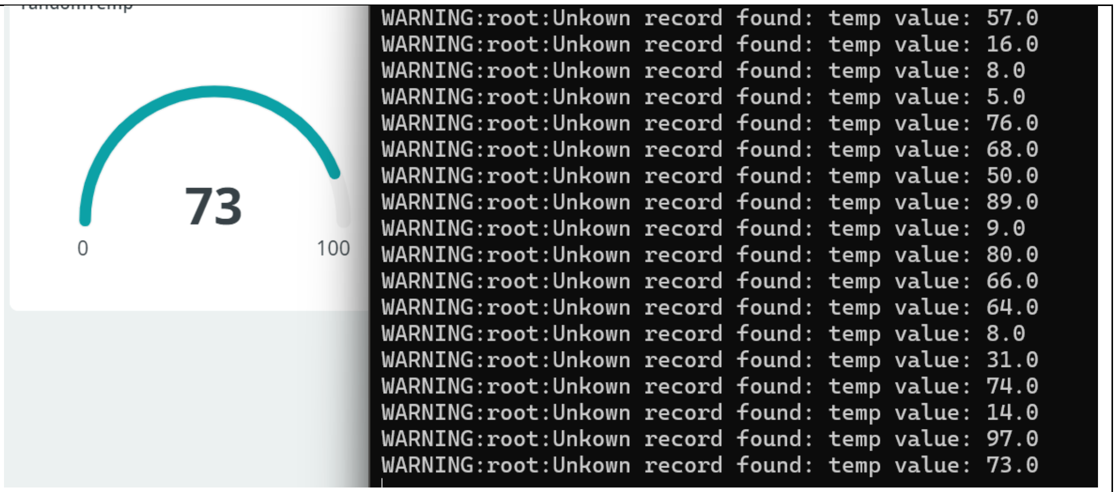
You should replace DEVICE_ID and SECRET_KEY as you were given in step 1-d above.

Question: Study the code and describe in your word how the statements match to the purpose mentioned in this step-3 above?

Answer:
DEVICE_ID and SECRET_KEY are used to authenticate the Python script with the Arduino IoT Cloud. These identifiers are generated when you manually set up a new device in the Arduino IoT Cloud.

Callback Function:

	The function on_temperature_changed(client, value) is a **callback** function that handles any changes to the **temperature** variable in the Arduino IoT Cloud. Whenever the cloud variable temperature is updated, this function is triggered, and the new value is printed to the console. ArduinoCloudClient(): The main function creates an instance of the ArduinoCloudClient, connecting the Python device to the Arduino IoT Cloud using the provided DEVICE_ID and SECRET_KEY. client.register(): This function registers the **temperature** variable with the Arduino IoT Cloud. The on_write=on_temperature_changed argument links the callback function to handle any updates to the temperature variable. client.start(): This function starts the cloud client and initiates continuous communication with the cloud, allowing the script to listen for updates to the temperature variable in an infinite loop.
4	**Run Python script**: Run the python script from command line below - `$ python arduino_variable_sync.py` At this point, the command line output should show connecting to Arduino cloud and print new temperature values periodically. Question: Screenshot the Arduino Cloud Dashboard gauge showing the Arduino side randomTemperature value and screenshot the Python command-line output showing similar values. Note that there might be some lag due to network connectivity. Explain your answer accordingly. Answer:

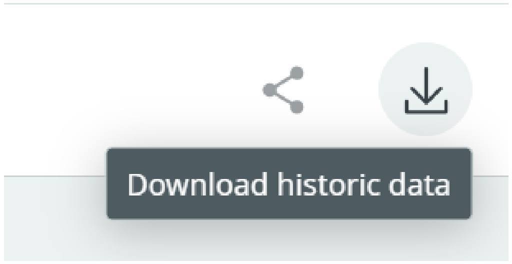


```
WARNING:root:Unkown record found: temp value: 57.0
WARNING:root:Unkown record found: temp value: 16.0
WARNING:root:Unkown record found: temp value: 8.0
WARNING:root:Unkown record found: temp value: 5.0
WARNING:root:Unkown record found: temp value: 76.0
WARNING:root:Unkown record found: temp value: 68.0
WARNING:root:Unkown record found: temp value: 50.0
WARNING:root:Unkown record found: temp value: 89.0
WARNING:root:Unkown record found: temp value: 9.0
WARNING:root:Unkown record found: temp value: 80.0
WARNING:root:Unkown record found: temp value: 66.0
WARNING:root:Unkown record found: temp value: 64.0
WARNING:root:Unkown record found: temp value: 8.0
WARNING:root:Unkown record found: temp value: 31.0
WARNING:root:Unkown record found: temp value: 74.0
WARNING:root:Unkown record found: temp value: 14.0
WARNING:root:Unkown record found: temp value: 97.0
WARNING:root:Unkown record found: temp value: 73.0
```

There is a small lag between the values shown on the dashboard and the Python output due to network delays or cloud processing times. However, both should show the same values after synchronization.
The **on_temperature_changed** function in the Python script ensures that every update in the cloud is captured and printed so I will see the correct values sometimes with a delay.

5 Question: Research how you can download data from Arduino IoT Cloud, say the *randomTemperature* variable if you continue the setup running for 10 minutes or so?

Answer: The historical data can be downloaded directly from the dashboard



6 Question: Discuss how you can save the data while you are receiving in Python script in a file? You can discuss in a group and come up with a solution.
Hint: An algorithm of appending data can be as below. Write Python code for the algorithm. You can put the code in the call-back function *on_temperature_changed()*.

Algorithm: Append timestamp and data value to a file:

a. Open a file in append mode
b. Create a CSV string <timestamp>, <value> <NEWLINE>

c. Call write function and pass CSV string, otherwise, call write_line function and pass the CSV string removing the ending <NEWLINE> (otherwise, a blank new line will be written to file).
d. Call flash to push data to be written to file immediately.
e. Close the file.

You can modify the algorithm to make it efficient, such as instead of opening/closing the file every time, move them to the beginning and end of the execution and keeping the CSV string formation and writing and flashing to file inside the callback function.

Answer:
The data can be saved while receiving it in the Python script by appending the timestamp and the new value to a file in **CSV format**.
To make the process more efficient:

- **Open the file once** at the start and close it after the script finishes.
- **Use buffering** to avoid frequent writes. However, I would have to make sure to flush the buffer periodically to ensure data is saved.