Universiti Malaysia Terengganu
UMT

Faculty of Ocean Engineering Technologies and Informatics

Internet of Things

Lab Module

Ahmad Shukri Mohd Noor
Farizah Binti Yunus
Fadzli Syed Abdullah

Version 1.0

Synopsis

This course introduces concepts and main components of the Internet of Things (IoT). The student will be exposed to the concept of IoT thru the network technology and protocol as well as the wireless environment. Students also will be exposed to data analytics in an IoT environment. Exposure to the selected IoT application development will be carried out in the lab to increase the student learning experiences. This course is essential for introducing students to the fundamentals of the IoT and its relationship to everyday life.

Ahmad Shukri Mohd Noor
Farizah Binti Yunus
Fadzli Syed Abdullah
Version 1.0
2023

Table of Contents

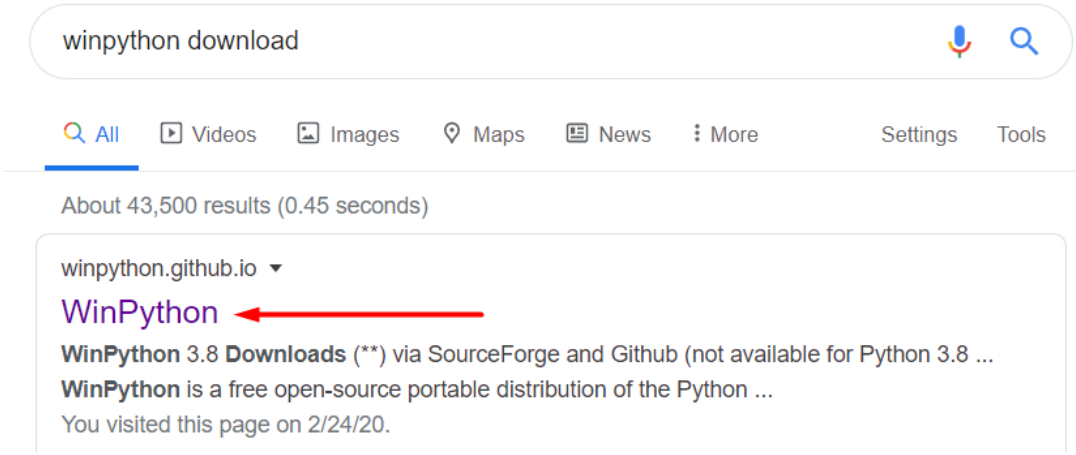
Topic 1: Programming Languages for IoT

Module 1: Getting Started with Python [1hr]

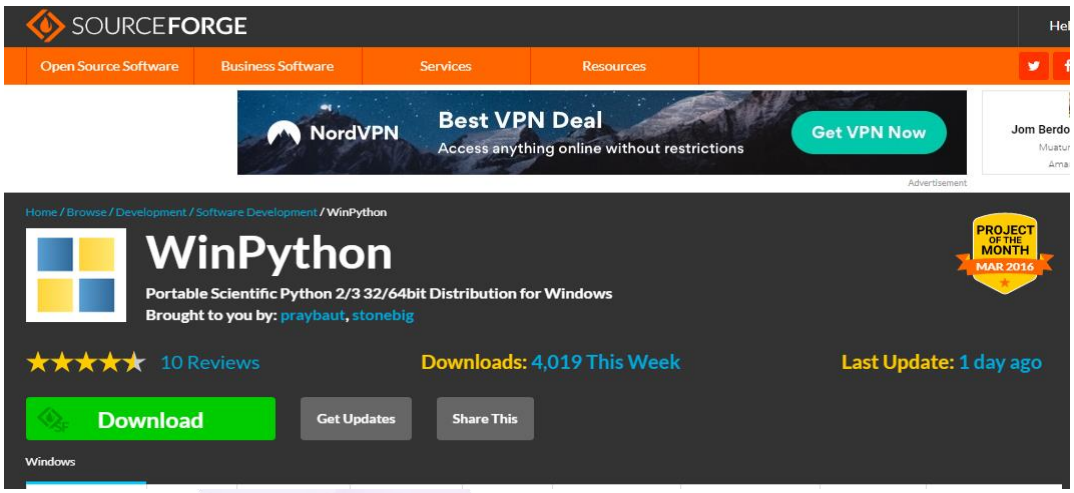
Objective: In this lab we are going to install software used for python programming. WinPython is a free open-source portable distribution of the Python programming language for Windows 8/10 and scientific and educational usage. Project Jupyter exists to develop open-source software, open-standards, and services for interactive computing across dozens of programming languages.

[Step#01] Install WinPython

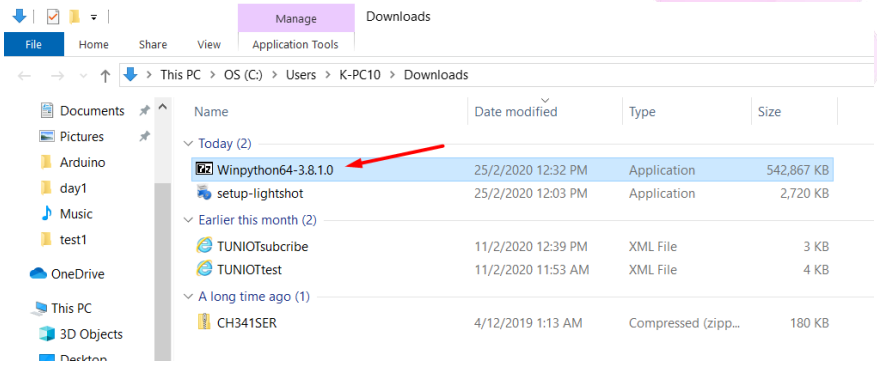
1. Search “WinPython download” in your browser
   https://sourceforge.net/projects/winpython/



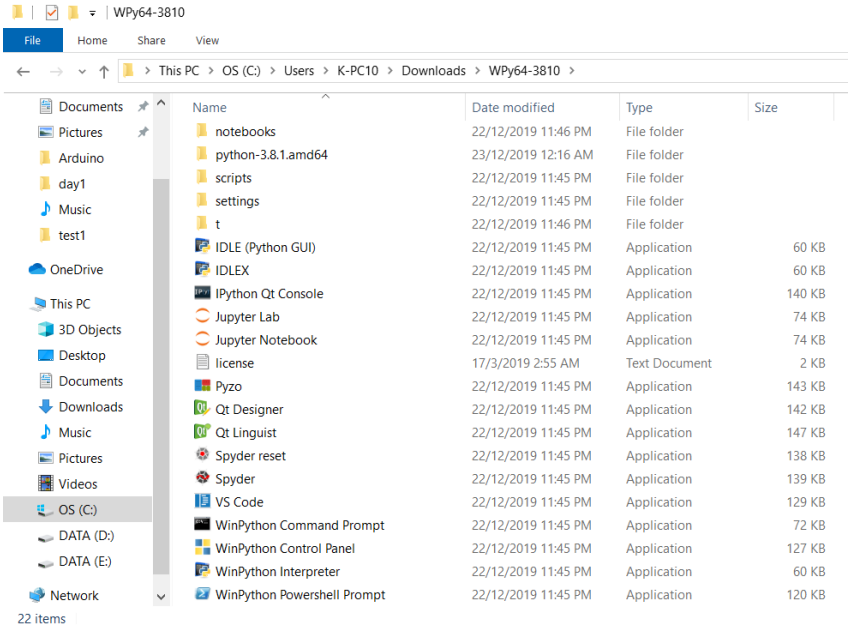
2. Click at downloads, and it will starts shortly



3. Extract the downloaded file

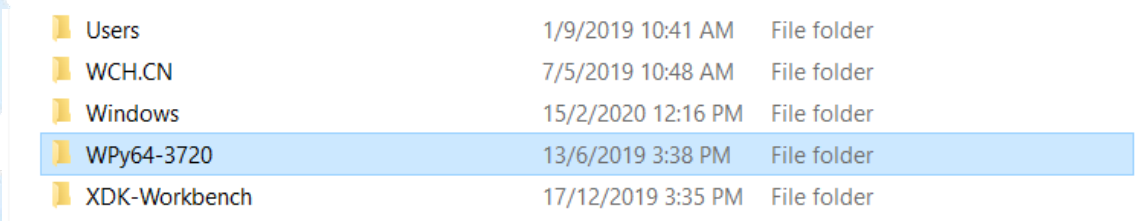


4. Application inside the folder

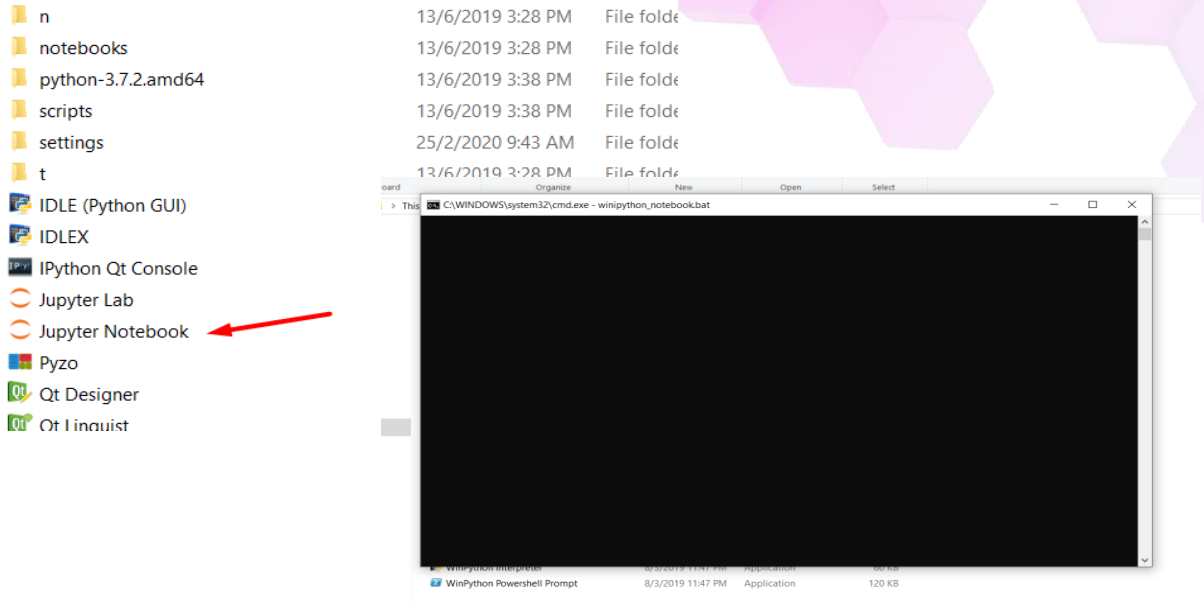


[Step#02] Use Jupyter Notebook

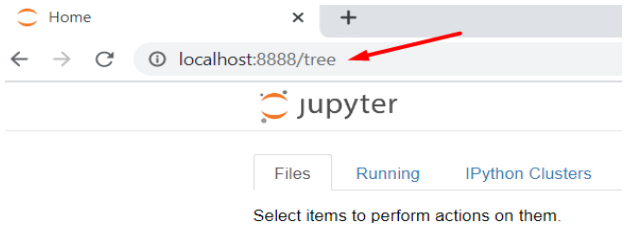
1. Open WinPython folder



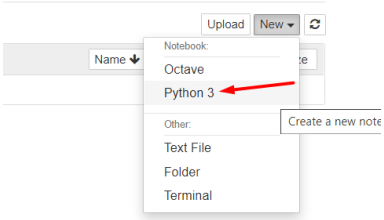
2. Start Jupyter notebook



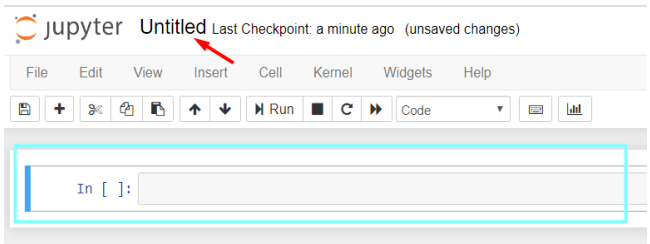
3. Open web browser and open URL localhost:8888/tree



4. Create new python 3



5. Change the title and start code



References:

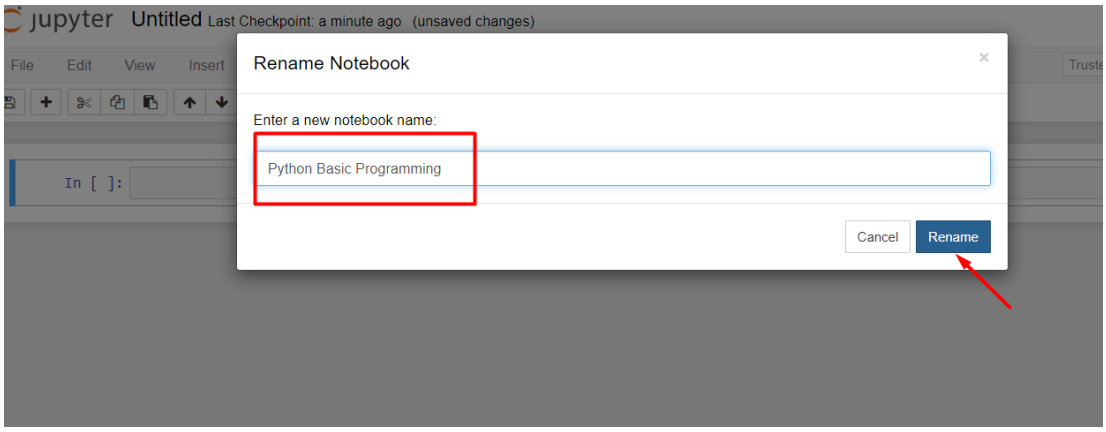
1. https://github.com/winpython
2. https://jupyter.org/

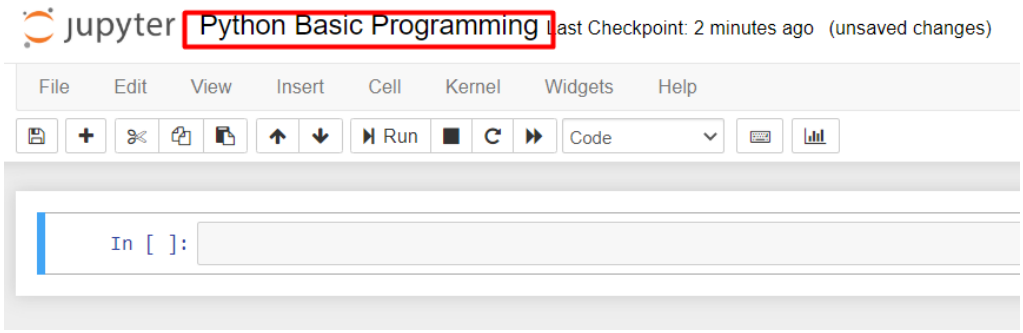
Module 2: Python Basic Programming [2hrs]

Objective: In this lab we are going to code using Jupyter Notebook. Throughout this lab, we will cover python syntax, element, comment, variable, data types and basic operators.

[Step#01] First code with Python

1. Start Jupyter Notebook.
2. Create a new python 3 file.
3. Change the title to “Python Basic Programming” and start code.





4. Write our first code to familiarise with the python syntax and jupyter notebook. Press “Enter” to see the result.

```
In [1]: print("Hello world")
        Hello world
```

5. The code below shows the importance of indentation in python.

```
In [2]: if 5>3:
        print("wrong indentation")
```

```
  File "<ipython-input-2-ed14205c5890>", line 2
    print("wrong indentation")
        ^
IndentationError: expected an indented block
```

```
In [3]: if 5>3:
            print("right indentation")
```

```
right indentation
```

6. The code below using # for comment

```
In [9]: # this is comment
        print("comment test")
```

```
comment test
```

```
In [8]: # this is comment
        # python do not have multiline comment
        print("comment test")
```

```
comment test
```

[Step#02] Variables in Python

1. The code below creates a variable in python.

```
In [11]: a = 5
         b = 6.0098
         _car = 'BMW' # can use '' or " "

         print(_car)
         print(b)
         print(a)
```

```
BMW
6.0098
5
```

In [13]:
```
#value assigned to multiple variables

car1,car2,car3 = "BMW",'Mercedes',"Volkswagen"
print(car1)
print(car2)
print(car3)
```

```
BMW
Mercedes
Volkswagen
```

In [16]:
```
#same value assigned to multiple variables

car1=car2=car3 = "Viva"
print(car1)
print(car2)
print(car3)
```

```
Viva
Viva
Viva
```

In [18]:
```
nama = 'Dan'
print('My name is ' + nama)
```

```
My name is Dan
```

In [19]:
```
nama = 'Dan'
ayat = 'My name is '
print( ayat + nama)
```

```
My name is Dan
```

In [20]:
```
number1 = 20
number2 = 2020
print(number1 + number2)
```

```
2040
```

2. Variables created outside of a function are called global variables. We may use the Global Keyword to create a global variable within a function.

In [22]:
```
location = 'Sungai Petani'

def function1():
    print(location)

function1()
```

```
Sungai Petani
```

In [23]:
```
#global variable
location = 'Sungai Petani'

def function1():
    #local variable
    location = 'Jitra'
    print(location)

function1()
print(location)
```

```
Jitra
Sungai Petani
```

In [25]:
```
#global variable
location = 'Sungai Petani'

def function1():
    #global variable created
    #inside function
    global location

    location = 'Jitra'
    print(location)

function1()
print(location)
```

```
Jitra
Jitra
```

3. The code below is to assign variables to a particular data type.

```
In [39]: e = ["wij","dan","mohamad"]
         print(e)
         print(type(e))
```

```
['wij', 'dan', 'mohamad']
<class 'list'>
```

```
In [40]: f = ("wij","dan","mohamad")
         print(f)
         print(type(f))
```

```
('wij', 'dan', 'mohamad')
<class 'tuple'>
```

```
In [41]: e = ["wij","dan","mohamad"]
         print(e)

         e[2] = "ariff"
         print(e)
```

```
['wij', 'dan', 'mohamad']
['wij', 'dan', 'ariff']
```

```
In [42]: f = ("wij","dan","mohamad")
         print(f)
         f[2] = ariff
         print(f)
```

```
('wij', 'dan', 'mohamad')

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-42-07d2fee7e8f1> in <module>
      1 f = ("wij","dan","mohamad")
      2 print(f)
----> 3 f[2] = ariff
      4 print(f)

NameError: name 'ariff' is not defined
```

```
In [45]: e = ["wij","dan","mohamad"]
         f = ("wij","dan","mohamad")

         print(e.__sizeof__())
         print(f.__sizeof__())
```

```
64
48
```

```
In [51]: h = {'name': 'wijdan', 'age': 20}

         print("his name is",h['name'])

         print("his age is",h['age'])
```

```
his name is wijdan
his age is 20
```

```
In [53]: i = {'dan', 'dan', 'dan', 'wij', 'mohamad'}
         print(i)
         print(i[1])
```

```
{'dan', 'wij', 'mohamad'}
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-53-1bf0bce0a741> in <module>
      1 i = {'dan', 'dan', 'dan', 'wij', 'mohamad'}
      2 print(i)
----> 3 print(i[1])

TypeError: 'set' object does not support indexing
```

[Step#03] Input and Output in Python

1. The code below is to use Input and Output in python.

```
In [ ]: print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)
```

- value to be printed
- separator used between values
- printed after all values are printed. default is new line
- sys.stdout is screen as default

```
In [1]: print(1,2,3,4)
        print(1,2,3,4, sep='#', end='.')

        1 2 3 4
        1#2#3#4.
```

```
In [2]: x = 10
        y = 2020

        print("I am {} years old in {}".format(x,y))

        I am 10 years old in 2020
```

```
In [3]: print("i love {0} and {1}".format("roti canai","teh tarik"))
        print("i love {1} and {0}".format("roti canai","teh tarik"))

        i love roti canai and teh tarik
        i love teh tarik and roti canai
```

```
In [5]: z = input('Enter a number :')
        z

        Enter a number :200

Out[5]: '200'
```

```
In [20]: name,age = input('enter your name:'),int(input('enter your age:'))

         enter your name:wijdan
         enter your age:20
```

2. To allow flexibility we might want to take the input from the user. In Python, we have the input() function to allow this. It is save in string data type. Use a cast to take numeric data.

```
In [5]: z = input('Enter a number :')
        z

        Enter a number :200

Out[5]: '200'
```

```
In [20]: name,age = input('enter your name:'),int(input('enter your age:'))

         enter your name:wijdan
         enter your age:20
```

[Step#04] Operators in Python

3. Arithmetic

```
In [8]: x = 2
        y = 4
        print("x + y = ", x+y)
        print("x - y = ", x-y)
        print("x * y = ", x*y)
        print("x / y = ", x/y)
        print("x // y = ", x//y)
        print("x ** y = ", x**y)
```

```
x + y =  6
x - y =  -2
x * y =  8
x / y =  0.5
x // y =  0
x ** y =  16
```

4. Comparison

```
In [10]: x = 2
         y = 4
         print("x > y = ", x>y)
         print("x < y = ", x<y)
         print("x == y = ", x==y)
         print("x != y = ", x!=y)
         print("x >= y = ", x>=y)
         print("x <= y = ", x<=y)
```

```
x > y =  False
x < y =  True
x == y =  False
x != y =  True
x >= y =  False
x <= y =  True
```

5. Logical

```
In [11]: x = True
         y = False
         print("x and y = ", x>y)
         print("x or y = ", x<y)
         print("x not y = ", x==y)
```

```
x and y =  True
x or y =  False
x not y =  False
```

6. Bitwise

In [13]:
```
x = 8
y = 4

print(x&y) #and
print(x|y) #or
print(~x) #not
print(x^y) #exclusive or
print(x>>2) #bitwise right shift
print(x<<2) #bitwise left shift
```

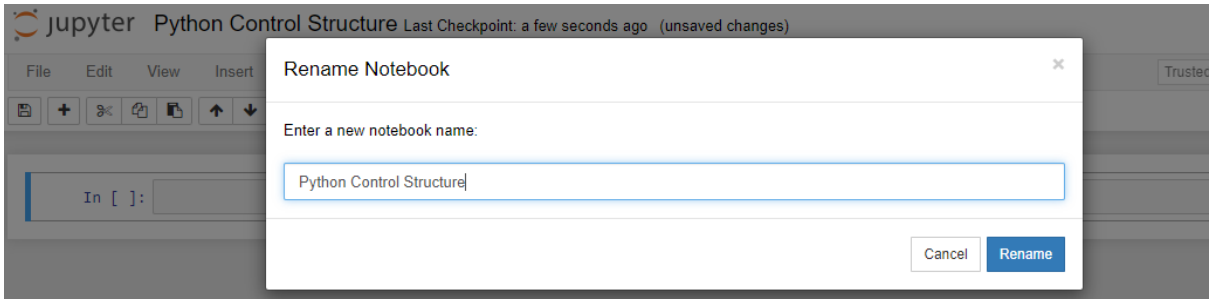
```
0
12
-9
12
2
32
```

Module 3: Python Control Structure [3hrs]

Objective: In this lab we are going to code using Jupyter Notebook. Throughout this lab, we will cover python control structure.

[Step#01] Create a new file

1. Start Jupyter Notebook.
2. Create a new python 3 file.
3. Change the title to “Python Control Structure” and start code.



[Step#02] Conditions

1. The code below is for decision-making when we only want code to be executed if a certain requirement is met. The program evaluates the condition and will execute statements if the condition result is True.

```
In [21]: value = int(input('enter a number:'))

if value > 0:
    print('positive number')
elif value == 0:
    print('zero')
else:
    print('negative number')
```

```
enter a number:20
positive number
```

[Step#02] Iterations

1. The for loop in Python is used to iterate over a sequence (list, tuple, string) or other iterable objects. Here, val is the variable that takes the value of the item inside the sequence on each iteration. Loop continues until we reach the last item in the sequence.

```
In [22]: car = ['BMW','Merc','Proton']
         for x in car:
             print(x)
```

```
BMW
Merc
Proton
```

```
In [24]: for x in 'Mercedes':
             print(x)
```

```
M
e
r
c
e
d
e
s
```

2. The while loop in Python is used to iterate over a block of code as long as the test expression (condition) is True. We generally use this loop when we don't know beforehand, the number of times to iterate.

```
In [39]: a = 1
         b = 10

         while a < b:
             print('a lower than b')
             a = a+1
```

```
a lower than b
a lower than b
a lower than b
a lower than b
a lower than b
a lower than b
a lower than b
a lower than b
a lower than b
```

[Step#03] Functions

1. In Python, function is a collection of associated statements that perform a specific task. Functions help break into smaller and more flexible parts of our program. As our system grows bigger and bigger, it's more structured and manageable by functions. It also prevents repetition, and makes code reusable.

```
In [43]: def my_function():
             """This function to
             print hello"""

             print('Hello')

         my_function()
```

```
Hello
```

```
In [42]: def my_function():
             """This function to make
             addition between a and b"""
             a = int(input('a:'))
             b = int(input('b:'))
             print(a+b)

         my_function()
```

```
a:20
b:30
50
```

Exercise

a. Create a function to determine fever
b. When the function is called
    i. Ask to enter body temperature
    ii. Answer whether or not you have a fever
        ➔ 38 and above - fever
        ➔ Below than 38 - healthy

The result should be as below :

```
Enter your body temperature: 38
Enter your body temperature:38
You have a fever. Go to the clinic.
Enter your body temperature: 37
Enter your body temperature:37
You are healthy.
```

References:

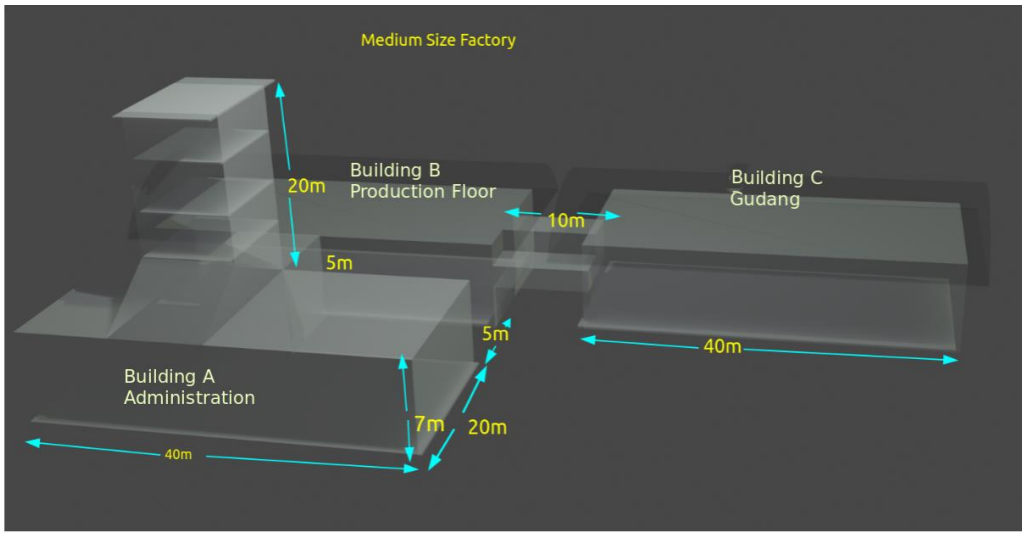
1. https://github.com/winpython
2. https://jupyter.org/
3. https://www.w3schools.com/python/default.asp

Topic 2: Architecture and IoT Network Protocol

Module 1: IoT IT Infrastructure [2hrs]

Objective: In this lab will guide you through understanding functions of some important IT Infrastructure and services to support IoT Application.

[Step#01] Structured cabling project for a medium size factory.



Estimated new building dimensions that require structured cabling to be designed and installed.

Table: IT Services Requirements by Building Locations

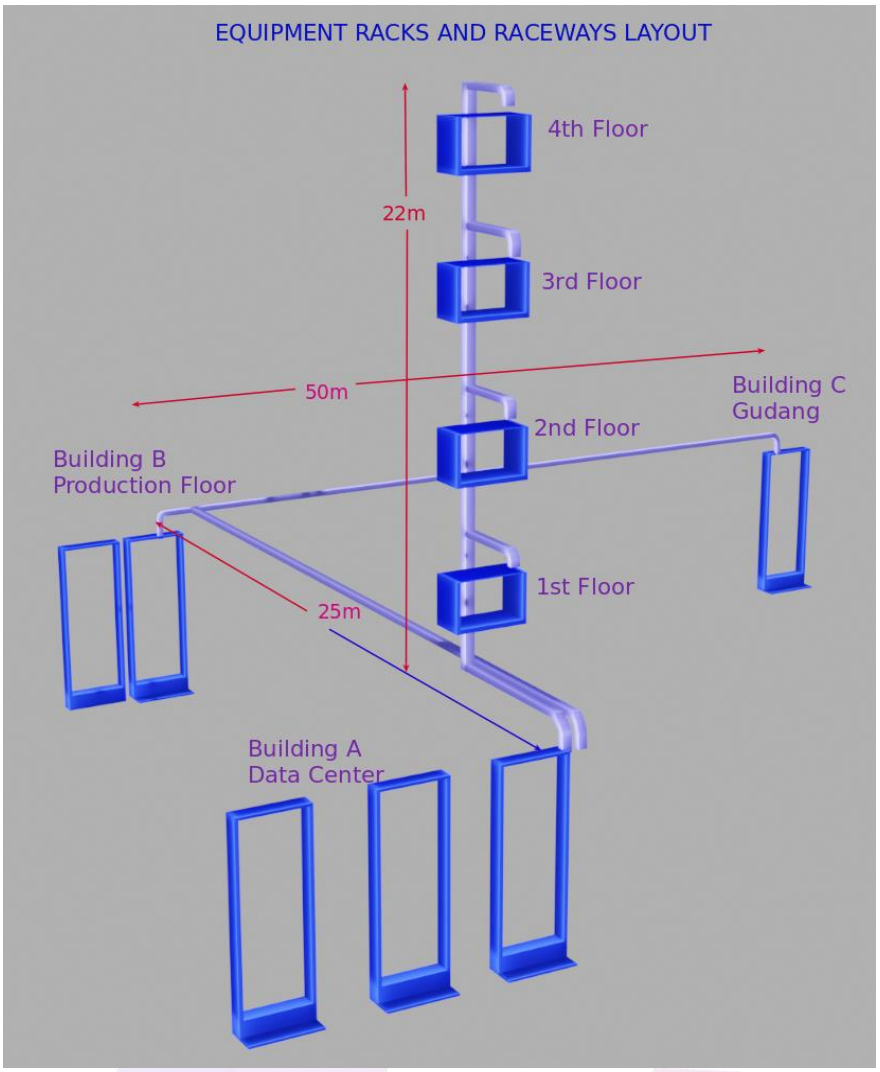
No	Location	Requirements	Qty
1	Administration - Gnd Floor	Work Area Access Point	150 6
2	Administration - 1st Floor	Work Area Access Point	40 2
3	Administration - 2nd Floor	Work Area Access Point	20 2
4	Administration - 3rd Floor	Work Area Access Point	20 2
5	Administration - 4th Floor	Work Area Access Point	20 2
6	Administration - Data Center	Rackmount servers Core Switches Access Switches	4 2 5

		Internet Routers UPS CCTV Controller	1 4 1
7	Production Floor	Data outlet Access Point	300 15
8	Gudang	Data outlet Access Point	40 4

Overall Wireless Requirements:

1. Bring Your Own Devices (BYOD) 800
2. Notebooks 200
3. Wireless IoT Devices 500

Below is the intended location for equipment racks and the data raceways for the entire building.



With the above information please calculate the bill of material of the components needed to complete the structure cabling.

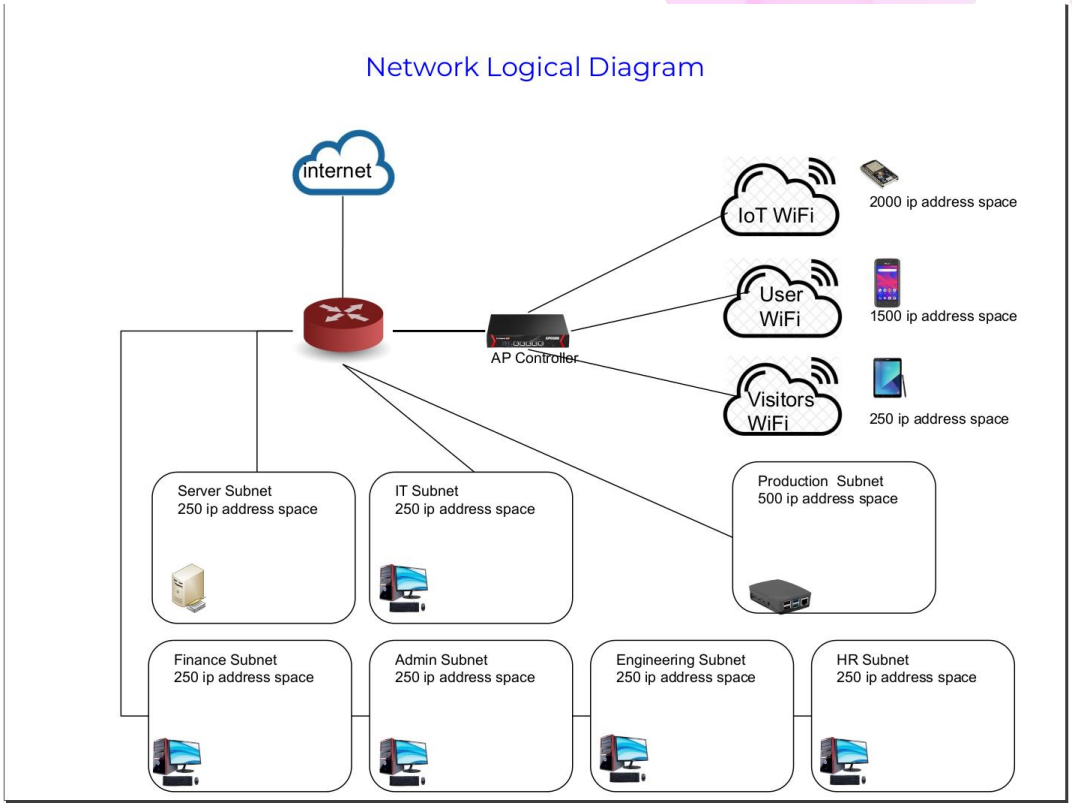


Figure: Medium Size LAN Logical Diagram for A Factory

[Step#02] Calculate the IPv4 addresses to be used in DHCP scopes for all the subnets shown.

1. Network Number
2. Starting IP address
3. Ending IP address
4. Subnet Mask
5. Gateway IP address

Note:

- Please use Class B for IoT subnet
- Please use Class A for all office and Production subnets
- Please use Class C for Visitors subnet

[Step#03] Deploying NodeRED IoT Gateway using Docker Container.

1. Create a directory to place docker-compose.yml and setting.js files.
2. Use the following docker-compose.yml.

```
version: '3.1'
services:
   nodered:
      image: nodered/node-red-docker
      container_name: noderedsecure
      volumes:
         - "./settings.js:/usr/src/node-red/node_modules/node-red/settings.js"
      ports:
         - "1880:1880"
         - "1883:1883"
```

3. Create "settings.js" file in the directory and enter the following content and save the file. This setting file will make your node-red application secured by enabling admin password: node admin password is "adminpwd"

```
* Copyright JS Foundation and other contributors, http://js.foundation
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
* http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
**/

// The `https` setting requires the `fs` module. Uncomment the following
// to make it available:
//var fs = require("fs");

module.exports = {
   // the tcp port that the Node-RED web server is listening on
   uiPort: process.env.PORT || 1880,

   // By default, the Node-RED UI accepts connections on all IPv4 interfaces.
```

```
    // To listen on all IPv6 addresses, set uiHost to "::",
    // The following property can be used to listen on a specific interface. For
    // example, the following would only allow connections from the local machine.
    //uiHost: "127.0.0.1",

    // Retry time in milliseconds for MQTT connections
    mqttReconnectTime: 15000,

    // Retry time in milliseconds for Serial port connections
    serialReconnectTime: 15000,

    // Retry time in milliseconds for TCP socket connections
    //socketReconnectTime: 10000,

    // Timeout in milliseconds for TCP server socket connections
    //  defaults to no timeout
    //socketTimeout: 120000,

    // Maximum number of messages to wait in queue while attempting to connect to TCP
socket
    //  defaults to 1000
    //tcpMsgQueueSize: 2000,

    // Timeout in milliseconds for HTTP request connections
    //  defaults to 120 seconds
    //httpRequestTimeout: 120000,

    // The maximum length, in characters, of any message sent to the debug sidebar tab
    debugMaxLength: 1000,

    // The maximum number of messages nodes will buffer internally as part of their
    // operation. This applies across a range of nodes that operate on message sequences.
    //  defaults to no limit. A value of 0 also means no limit is applied.
    //nodeMessageBufferMaxLength: 0,

    // To disable the option for using local files for storing keys and certificates in the TLS
configuration
    //  node, set this to true
    //tlsConfigDisableLocalFiles: true,

    // Colourise the console output of the debug node
    //debugUseColors: true,

    // The file containing the flows. If not set, it defaults to flows_<hostname>.json
    //flowFile: 'flows.json',

    // To enabled pretty-printing of the flow within the flow file, set the following
    //  property to true:
    //flowFilePretty: true,
```

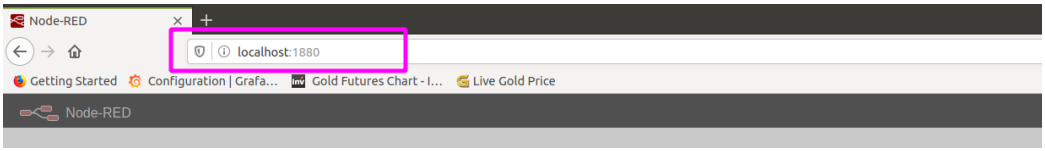
```
    // Securing Node-RED
    // -----------------
    // To password protect the Node-RED editor and admin API, the following
    // property can be used. See http://nodered.org/docs/security.html for details.
    adminAuth: {
        type: "credentials",
        users: [{
            username: "admin",
            password:
"$2a$08$sd0ZuGsa1G6TyC.VTE7SCet5TMSISz0ZW0l/b4AhBudWThNhNS6VK",
            permissions: "*"
        }]
    },

    // Configure the logging output
    logging: {
        // Only console logging is currently supported
        console: {
            // Level of logging to be recorded. Options are:
            // fatal - only those errors which make the application unusable should be
recorded
            // error - record errors which are deemed fatal for a particular request + fatal
errors
            // warn - record problems which are non fatal + errors + fatal errors
            // info - record information about the general running of the application + warn +
error + fatal errors
            // debug - record information which is more verbose than info + info + warn +
error + fatal errors
            // trace - record very detailed logging + debug + info + warn + error + fatal errors
            // off - turn off all logging (doesn't affect metrics or audit)
            level: "info",
            // Whether or not to include metric events in the log output
            metrics: false,
            // Whether or not to include audit events in the log output
            audit: false
        }
    },

    // Customising the editor
    editorTheme: {
        projects: {
            // To enable the Projects feature, set this value to true
            enabled: false
        }
    }
}
```

4. Issue command to build and compose docker-nodered:

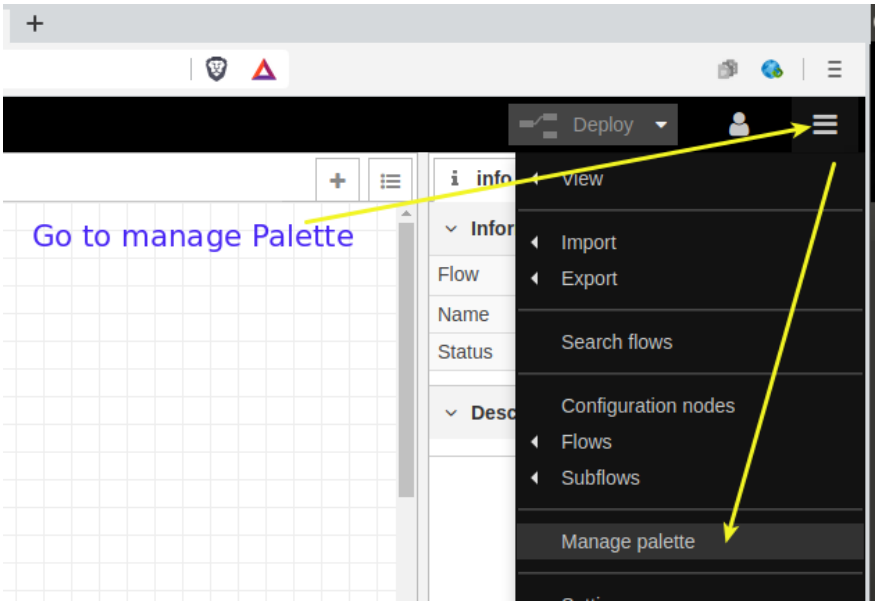
```
(base) fuzis@fsvivo:~/dockers/noderedockersecure$ sudo docker-compose up --build
ERROR: The Compose file './docker-compose.yml' is invalid because:
Unsupported config option for services.nodered: 'volume' (did you mean 'volumes'?)
(base) fuzis@fsvivo:~/dockers/noderedockersecure$ sudo docker-compose up --build
Creating network "noderedockersecure_default" with the default driver
Creating noderedsecure ... done
Attaching to noderedsecure
noderedsecure |
noderedsecure | > node-red-docker@1.0.0 start /usr/src/node-red
noderedsecure | > node $NODE_OPTIONS node_modules/node-red/red.js -v $FLOWS "--userDir"
"/data"
noderedsecure |
noderedsecure | 2 Jul 05:45:13 - [info]
noderedsecure |
noderedsecure | Welcome to Node-RED
noderedsecure | ===================
noderedsecure |
noderedsecure | 2 Jul 05:45:13 - [info] Node-RED version: v0.20.8
noderedsecure | 2 Jul 05:45:13 - [info] Node.js  version: v8.16.1
noderedsecure | 2 Jul 05:45:13 - [info] Linux 5.3.0-59-generic x64 LE
noderedsecure | 2 Jul 05:45:14 - [info] Loading palette nodes
noderedsecure | 2 Jul 05:45:14 - [warn] rpi-gpio : Raspberry Pi specific node set inactive
noderedsecure | 2 Jul 05:45:14 - [warn] rpi-gpio : Cannot find Pi RPi.GPIO python library
noderedsecure | 2 Jul 05:45:14 - [info] Settings file  : /data/settings.js
noderedsecure | 2 Jul 05:45:14 - [info] Context store  : 'default' [module=memory]
noderedsecure | 2 Jul 05:45:14 - [info] User directory : /data
noderedsecure | 2 Jul 05:45:14 - [warn] Projects disabled : editorTheme.projects.enabled=false
noderedsecure | 2 Jul 05:45:14 - [info] Flows file     : /data/flows.json
noderedsecure | 2 Jul 05:45:14 - [info] Creating new flow file
noderedsecure | 2 Jul 05:45:14 - [warn]
noderedsecure |
noderedsecure | ---------------------------------------------------------------------
noderedsecure | Your flow credentials file is encrypted using a system-generated key.
noderedsecure |
noderedsecure | If the system-generated key is lost for any reason, your credentials
noderedsecure | file will not be recoverable, you will have to delete it and re-enter
noderedsecure | your credentials.
noderedsecure |
noderedsecure | You should set your own key using the 'credentialSecret' option in
noderedsecure | your settings file. Node-RED will then re-encrypt your credentials
noderedsecure | file using your chosen key the next time you deploy a change.
noderedsecure | ---------------------------------------------------------------------
noderedsecure |
noderedsecure | 2 Jul 05:45:14 - [info] Server now running at http://127.0.0.1:1880/
```

```
noderedsecure | 2 Jul 05:45:14 - [info] Starting flows
noderedsecure | 2 Jul 05:45:14 - [info] Started flows
```

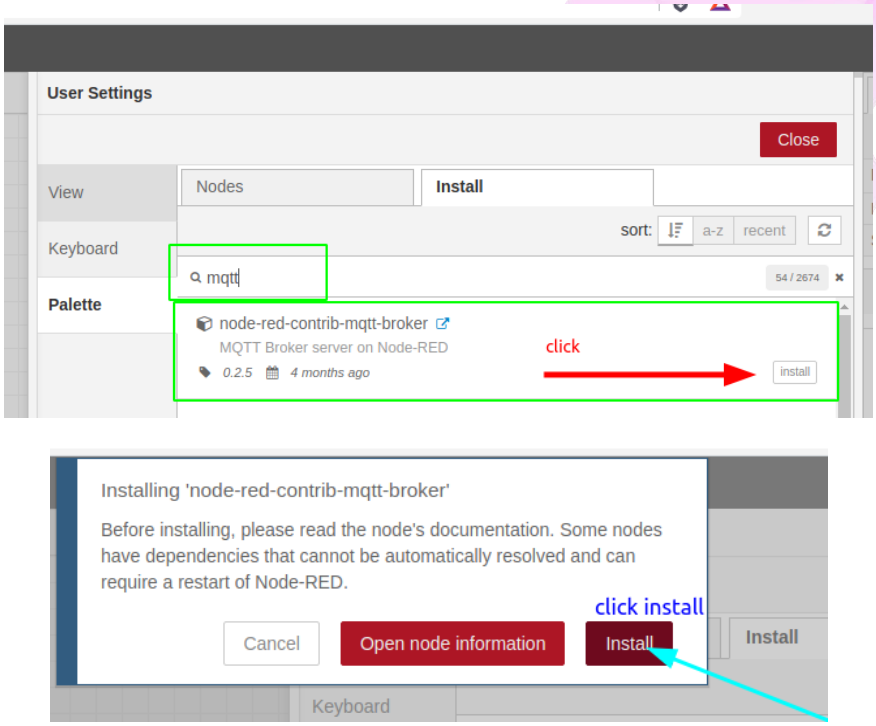
5. Open a browser and access to the url given: http://127.0.0.1:1880



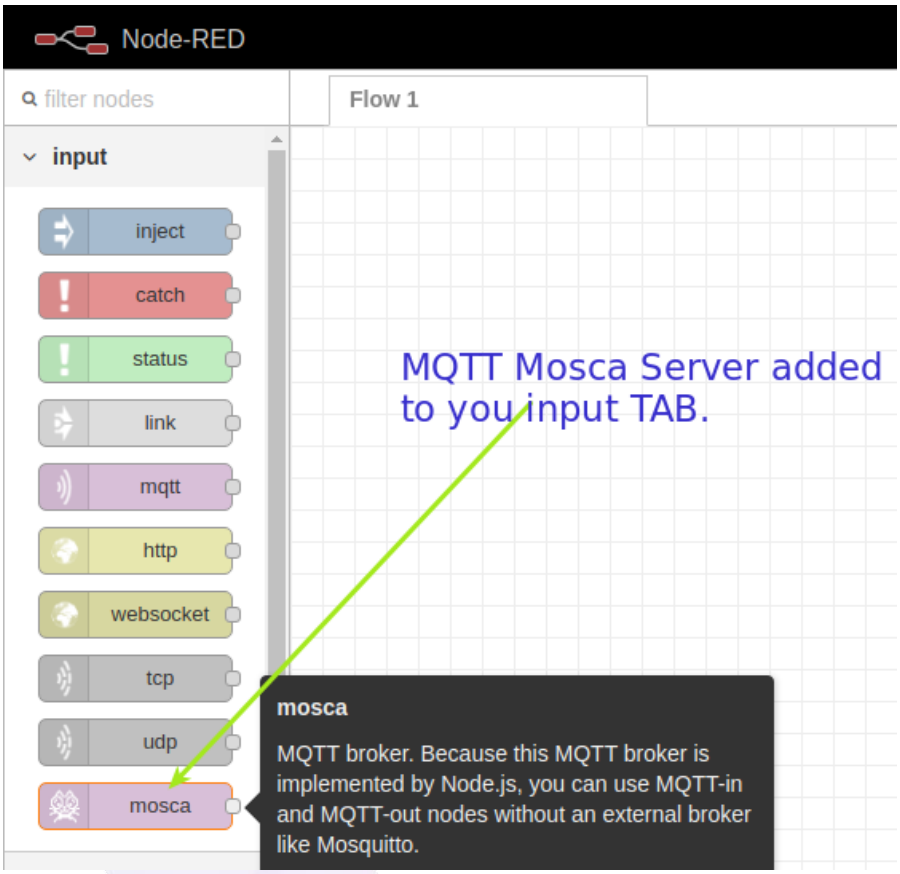
6. Once you login, go to the manage palette and Install MQTT broker as follows:



7. Click tab install and go to search box key in “mqtt”



8. Once installation completed, the Installed MQTT server name MOSCA can be found in the input TAB as shown below:



Note: MQTT serve port can be accessed from localhost:1883

[Step#04] Spin up Grafana + InfluxDB using docker image:

1. Use docker image available built by another user from docker-hub. Run the image as follows:

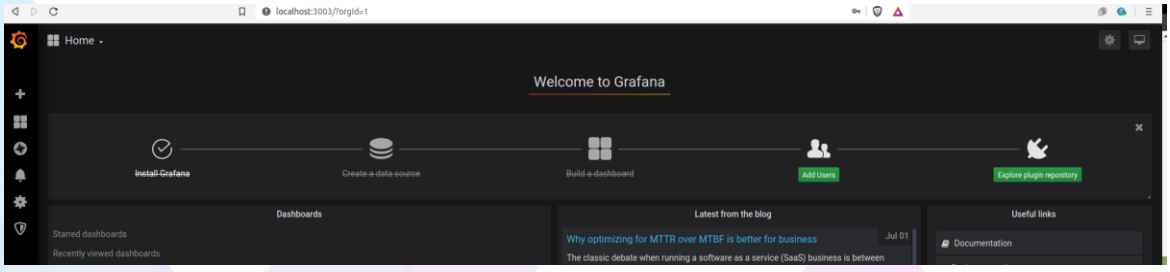
```
docker pull samuelebistoletti/docker-statsd-influxdb-grafana
```

```
docker run --ulimit nofile=66000:66000 \
  -d \
  --name docker-statsd-influxdb-grafana \
  -p 3003:3003 \
  -p 3004:8888 \
  -p 8086:8086 \
  -p 8125:8125/udp \
  samuelebistoletti/docker-statsd-influxdb-grafana:latest
```

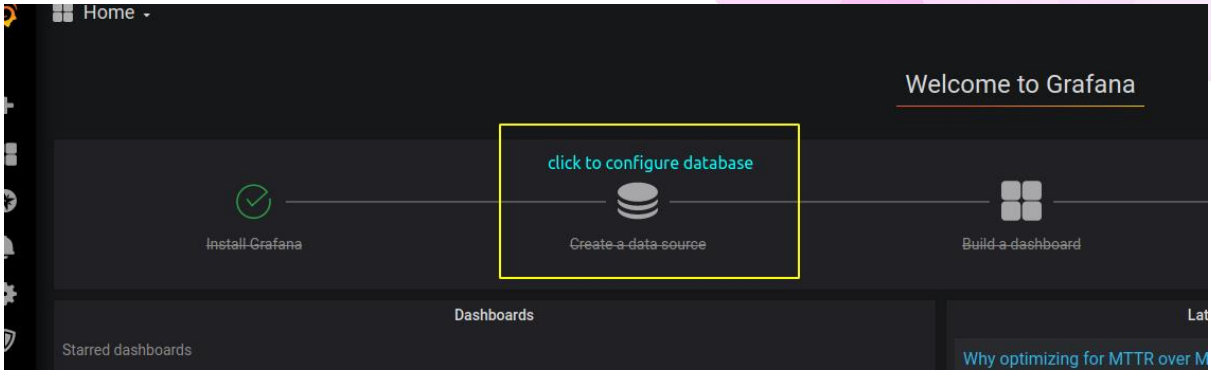
2. Run docker ps to check the status.

3. Now access to Grafana IoT visualization server: http://localhost:3003

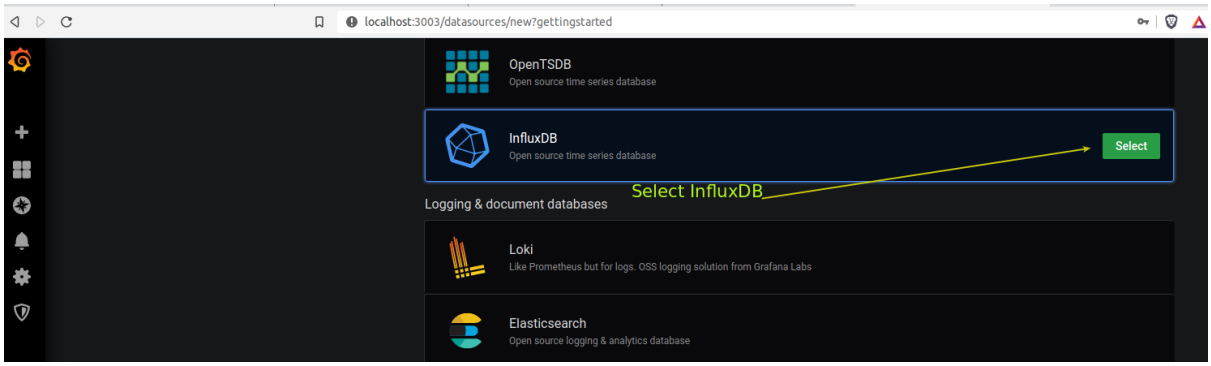
4. You should be able to see the Grafana Selcome Screen:

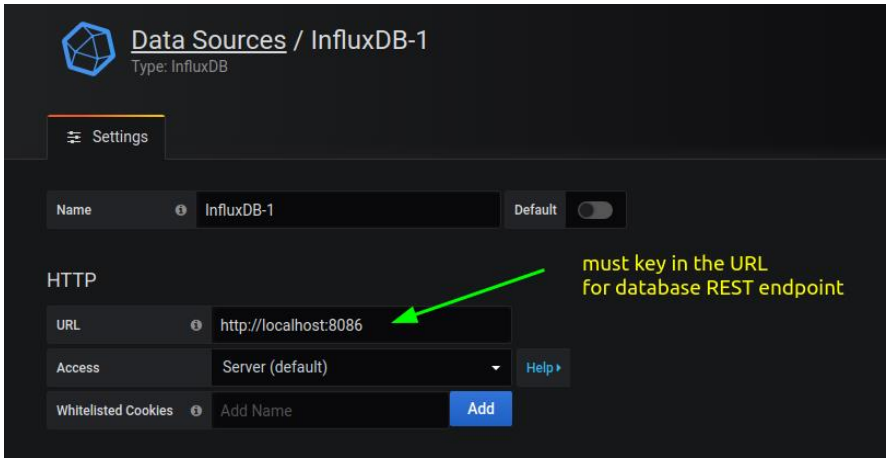


5. Now, configure the database for Grafana server.

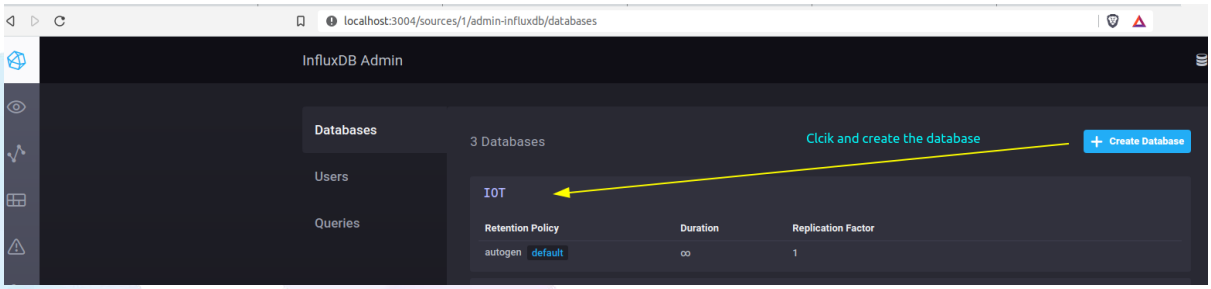


6. Select InfluxDB

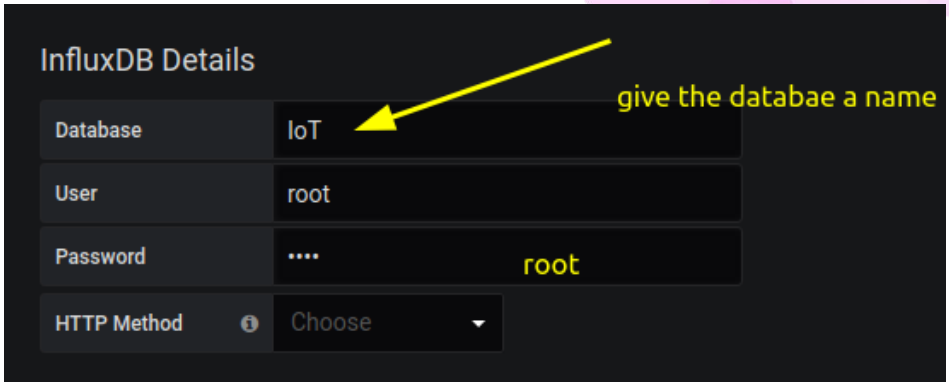


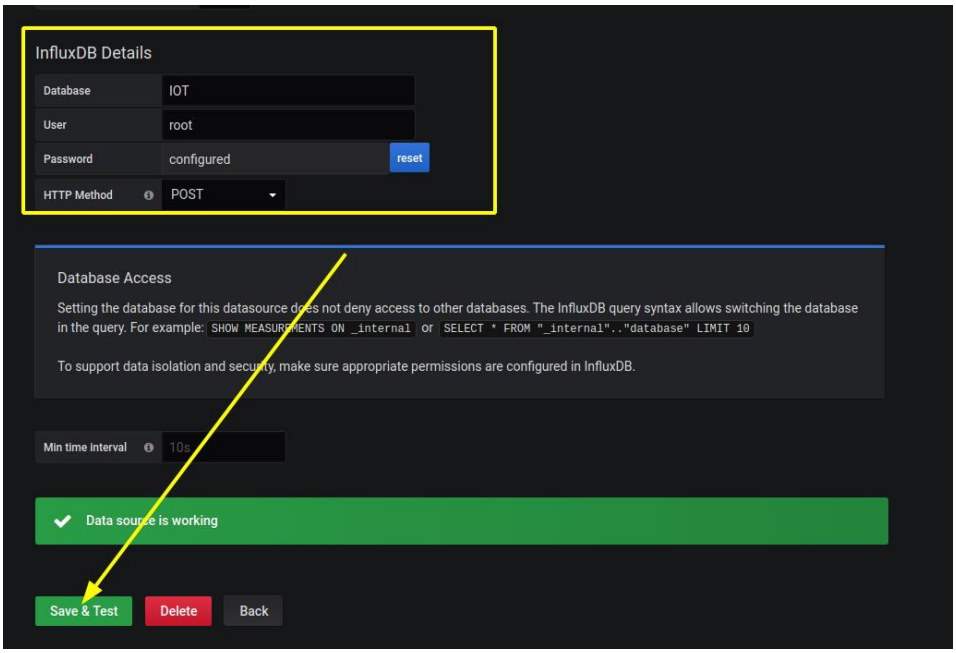


7. The database is not available yet on the InfluxDB server. You need to create it first using InfluxDB admin page. Open URL: http://localhost:3004 using another browser tab:



8. No go back to your Grafana configuration page and continue database setup.

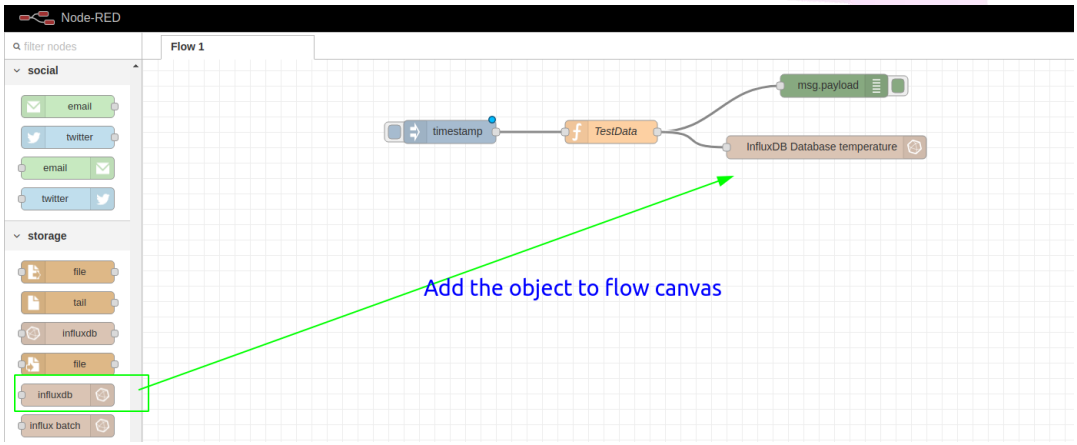




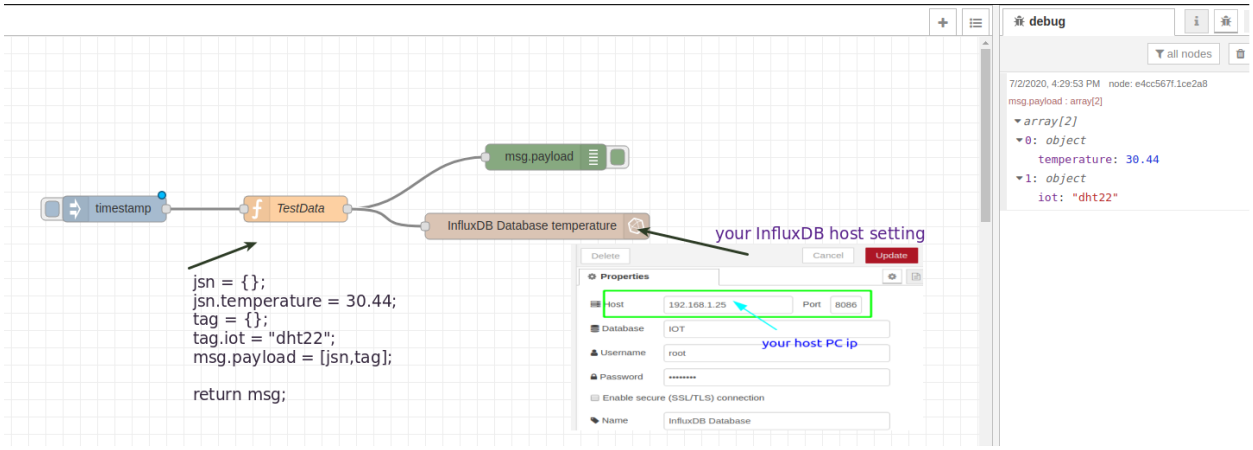
9. Save and test the database connection. You should be able to see the green alert bar "***Data source is working***"

[Step#05] Install InfluxDB

1. Now go to your NodeRED admin page and Install InfluxDB from Mage Palette. Create a flow:

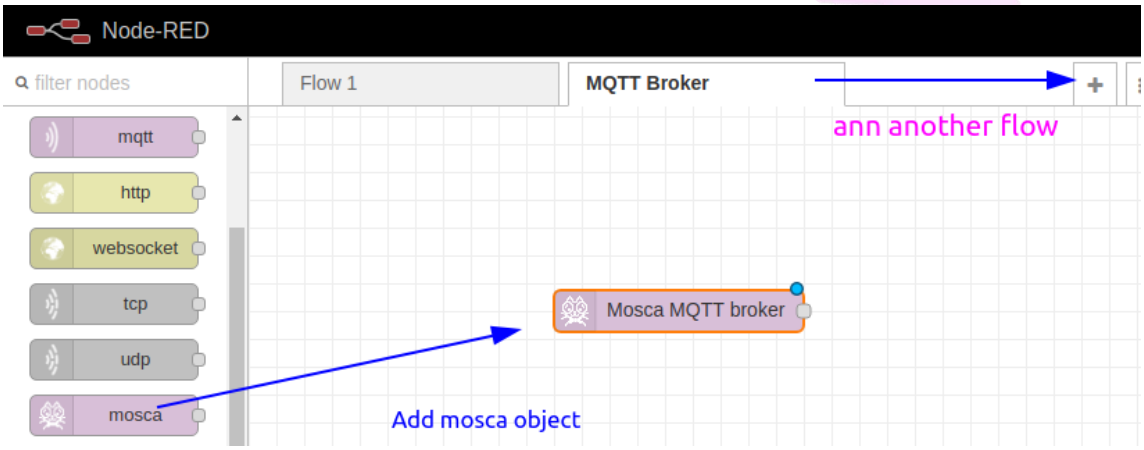


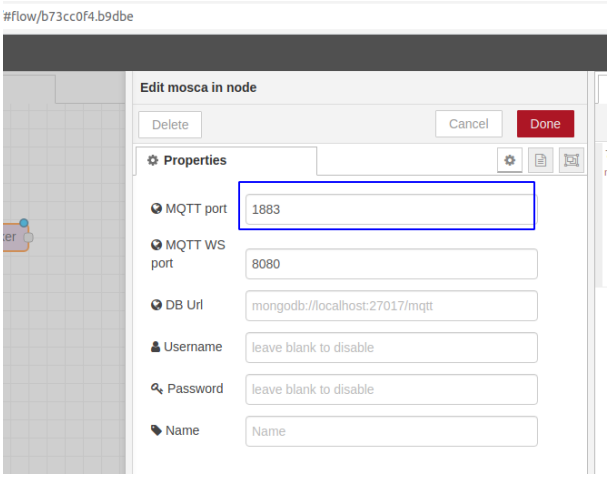
2. Configure the database and add the function script to send test data.



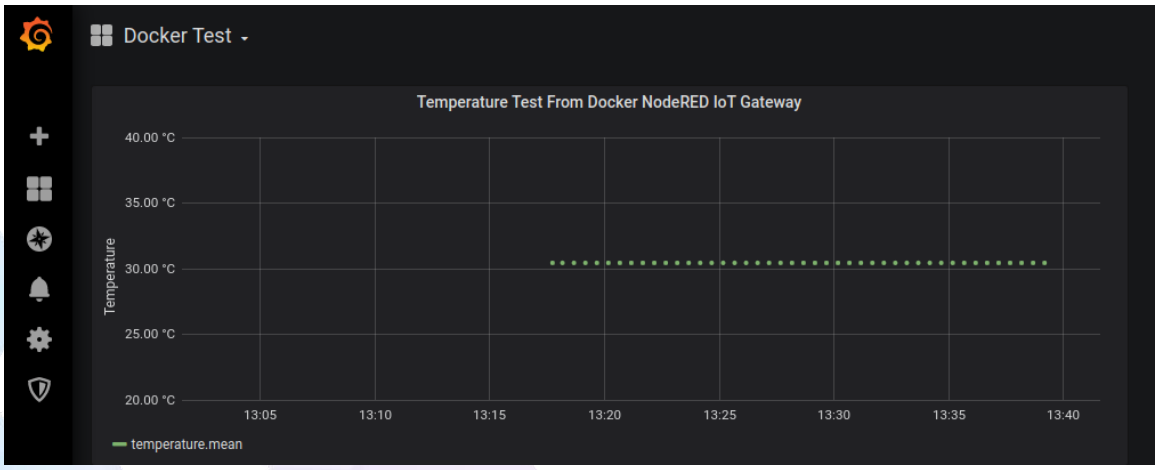

```
jsn = {};
jsn.temperature = 30.44;
tag = {};
tag.iot = "dht22";
msg.payload = [jsn,tag];

return msg;
```

3. Now enable MQTT broker by adding another flow on node-red.

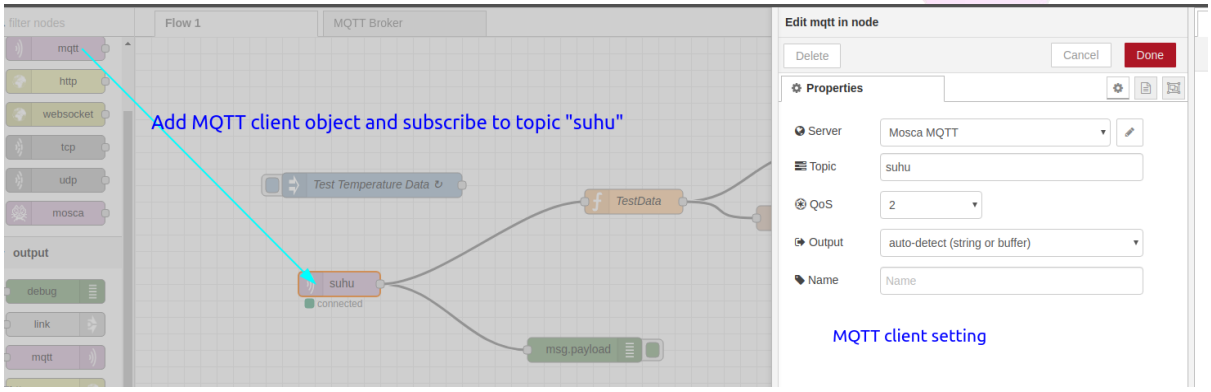




4. Click Done and enable Flow. Your PC now is able to receive messages from MQTT Clients like your ESP32 Iot device. Try send temperature data and see the result in Grafana Server:



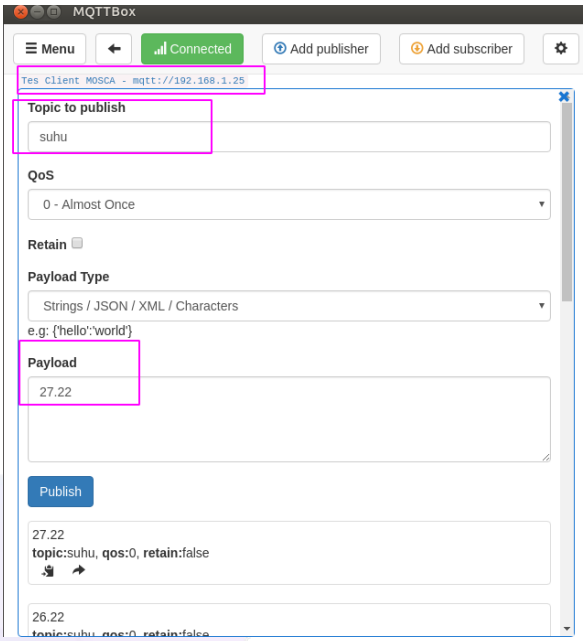
5. Now you can try to send Temperature data using MQTT client by publishing the data to Mosca server and subscribe back the data using NodeRed MQTT client. The data is then can be written to InfluxDB. Here is the updated Flow:



6. You need to change the JS script function to process temperature data arrives from MQTT client.

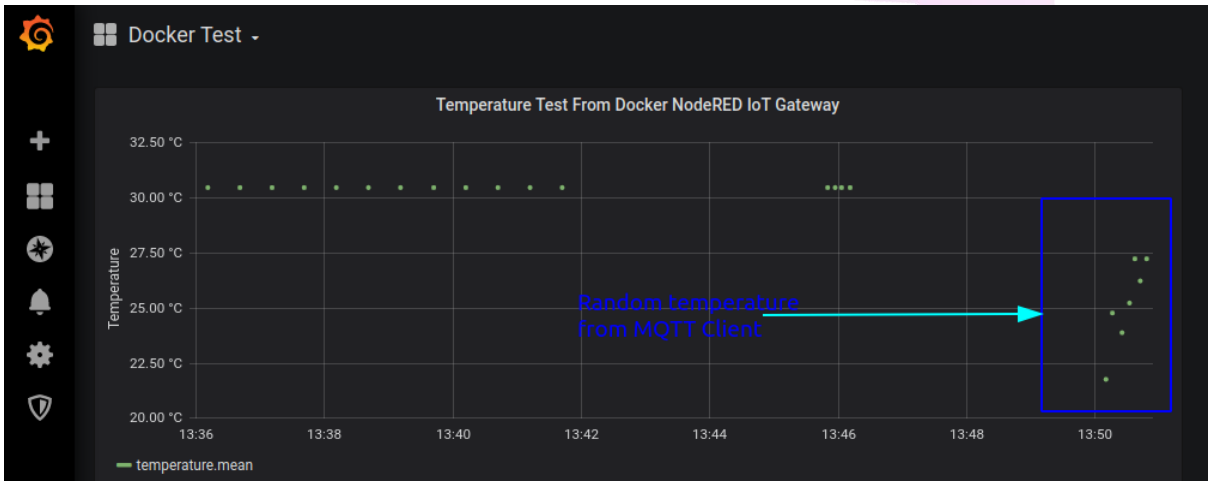
```
t = parseFloat(msg.payload);
jsn = {};
jsn.temperature = t;
tag = {};
tag.iot = "dht22";
msg.payload = [jsn,tag];

return msg;
```

7. Now you can use MQBOX Client to send test data.



Item 6 Using Grafana Visualization server

1. Your test data will be displayed immediately by the Grafana Visualization server.



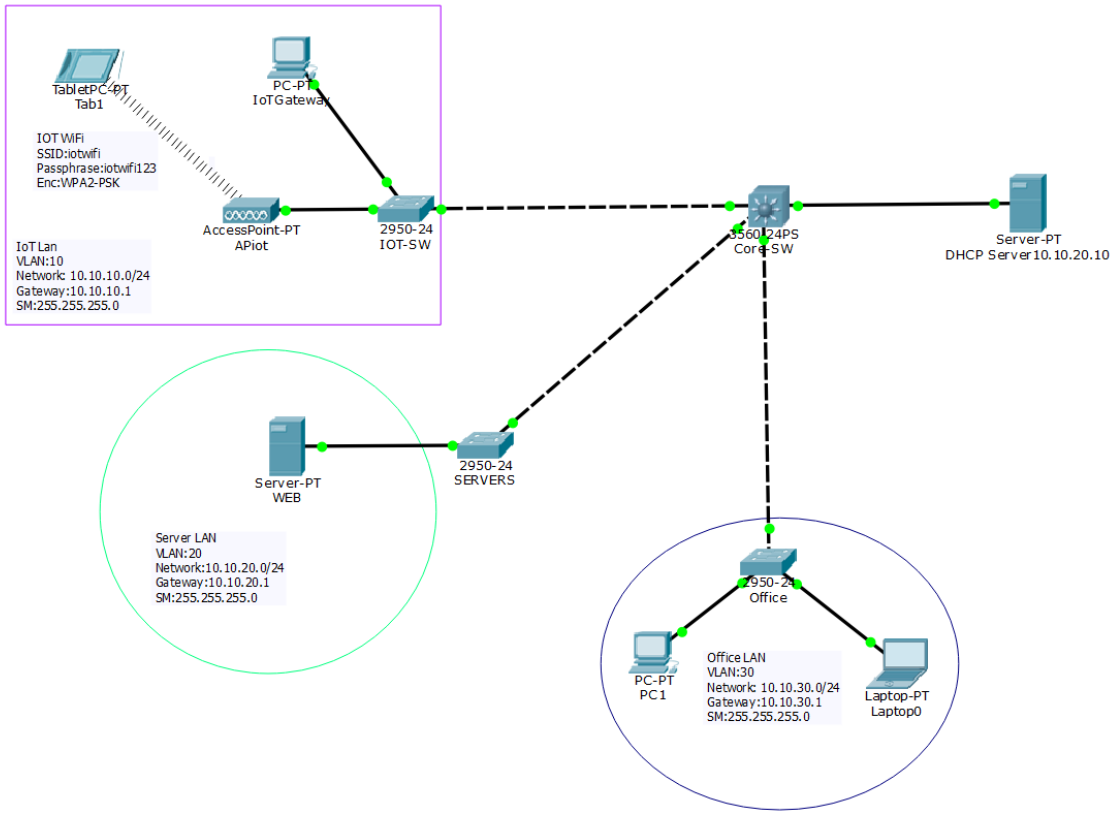
2. Now your IoT gateway that supports MQTT, InfluxDB and Grafana visualization server is complete.

Module 2: Basic IoT Network Design [1hr]

Objective: This lab exercise will guide you to build a basic reliable Local Area Network to Implement IoT technology. You will have to configure Core Switch, VLAN, DHCP Server and WiFi devices to make sure they can communicate with each other.

[Step#01] Network Design

1. Study below network design and service required

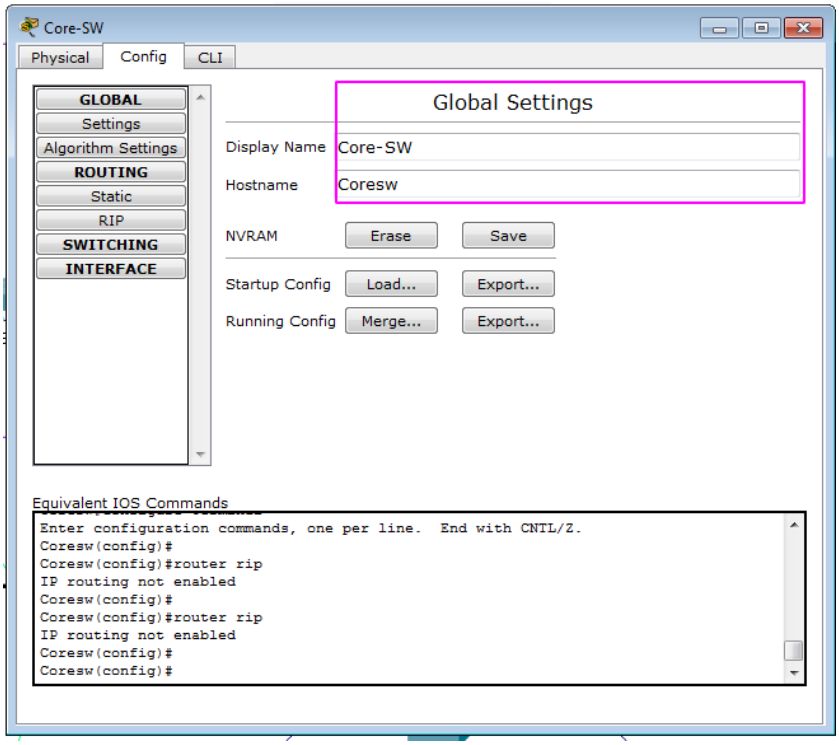


2. Launch the Cisco packet tracer and build the network as shown in Step1, ensure all the connections and devices are labelled.
3. Create VLAN Database on Core-SW, assign a unique VLAN Domain Name and VLAN password
4. Create VLANs (ids) and VLAN Name to be assigned to different subnetworks
5. Configure Core-SW and assign IP addresses to the VLAN interfaces. Remember, these interfaces are to be the gateways to all the corresponding subnetworks.
6. Configure Trunk ports on all connections between switches.
7. Configure all access switches to join the VLAN domain
8. Connect all devices to access switches and set their VLAN port to the correct VLAN id of the respective subnetwork.

9. Configure Wireless AP to serve Wi-Fi connection to IoT wireless devices.
10. Configure SSID and pass-phrase
11. Ensure Wireless AP is connected to the correct VLAN or network segment.
12. Configure DHCP and Web Servers.
13. Join the servers LAN port to the correct VLAN
14. Configure DHCP server IP address
15. Configure DHCP scopes to serve IP addresses to all subnetworks.
16. Configure Core-SW to allow DHCP request relay to each subnetwork.
17. Check that devices that use DHCP get the correct IP address
18. Check that the wireless devices are getting the correct IP addresses from DHCP server
19. Test all connections by sending PDU packets between each device. You are good once all PDU packets get the replies.

[Step#02] Build simple IoT Network

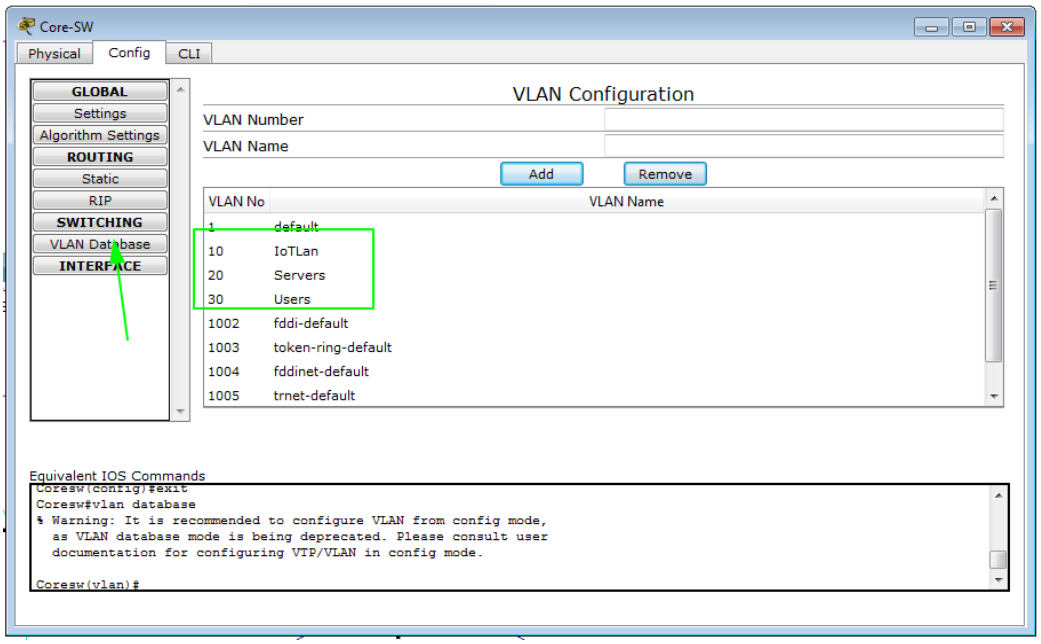
1. Install Cisco packet tracer 6.2 provided into your PC if you haven't got one.\
2. Make sure all the devices are added and connected as required by the network design.
3. Configure Core-SW and VLANs



4. Core-SW VLAN Database as VLAN Domain server

```
Coresw#
Coresw#show vtp status
VTP Version                 : 2
Configuration Revision      : 12
```

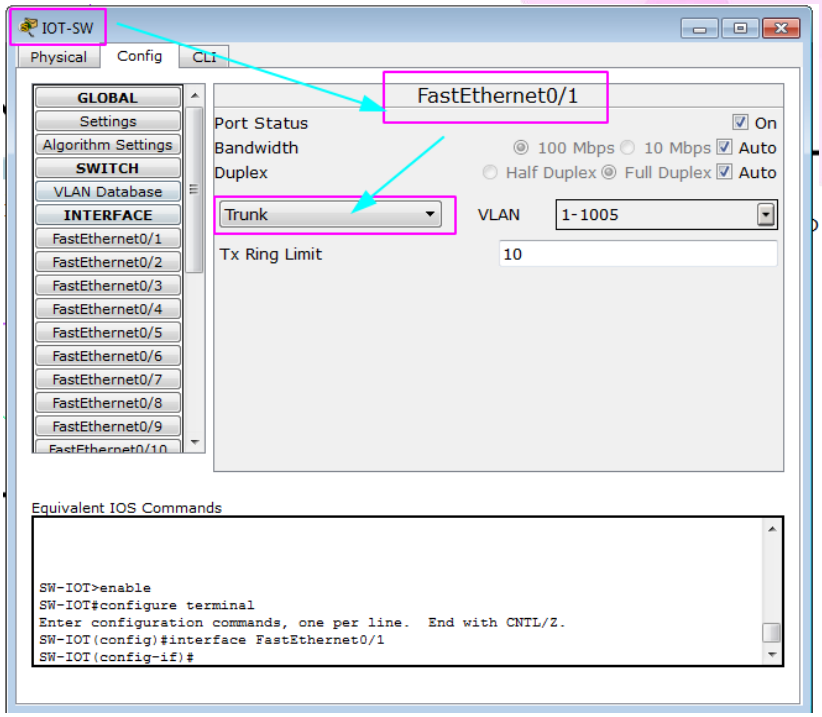
```
Maximum VLANs supported locally : 1005
Number of existing VLANs       : 8
VTP Operating Mode             : Server
VTP Domain Name                : iot
VTP Pruning Mode               : Disabled
VTP V2 Mode                    : Disabled
VTP Traps Generation           : Disabled
MD5 digest                     : 0x88 0x2D 0xD7 0xD0 0x0E 0xA5 0x40 0x56
Configuration last modified by 0.0.0.0 at 3-1-93 00:00:00
Local updater ID is 10.10.10.1 on interface Vl10 (lowest numbered VLAN interface
found)
```



5. Configure Core-SW VLAN interfaces

```
Coresw(config)#int Vlan 10
Coresw(config-if)#ip address 10.10.10.1 255.255.255.0
```

6. Configure trunk ports



7. Configure Access Switches to Join VLAN Domain

8. Configure VTP Database for switches

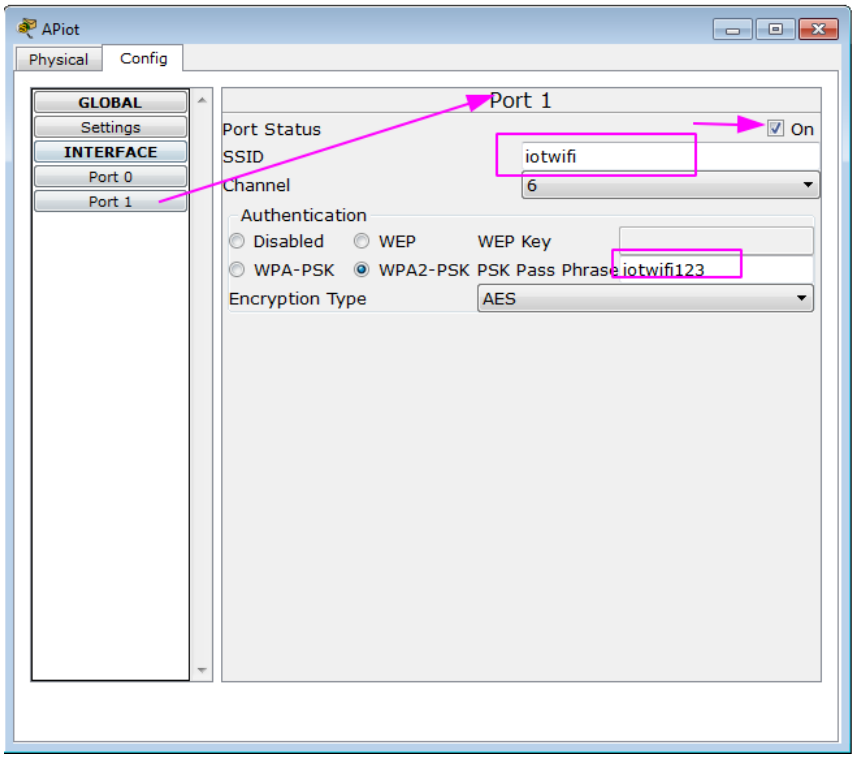
```
Switch#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
Switch(config)#vtp mode cli
Switch(config)#vtp mode client
Setting device to VTP CLIENT mode.
Switch(config)#vtp doma
Switch(config)#vtp domain iot
Changing VTP domain name from NULL to iot
Switch(config)#vtp pass
Switch(config)#vtp password iotwifi
Setting device VLAN database password to iotwifi
```

9. Checking VTP Status

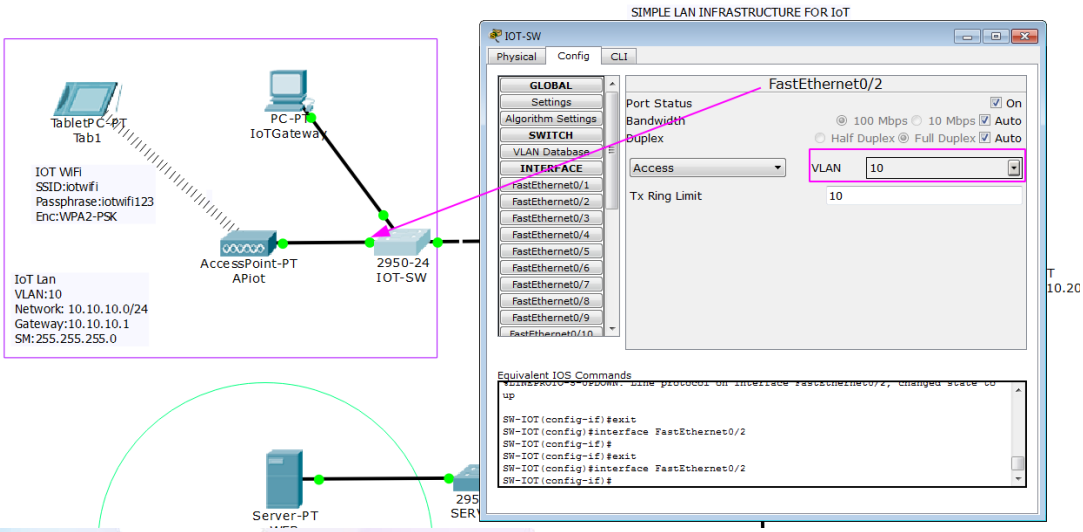
```
SERVERS#show vtp status
VTP Version                     : 2
Configuration Revision          : 12
Maximum VLANs supported locally : 255
Number of existing VLANs        : 8
VTP Operating Mode              : Client
VTP Domain Name                 : iot
```

```
VTP Pruning Mode            : Disabled
VTP V2 Mode                 : Disabled
VTP Traps Generation        : Disabled
MD5 digest                  : 0x88 0x2D 0xD7 0xD0 0x0E 0xA5 0x40 0x56
Configuration last modified by 0.0.0.0 at 3-1-93 00:00:00
```

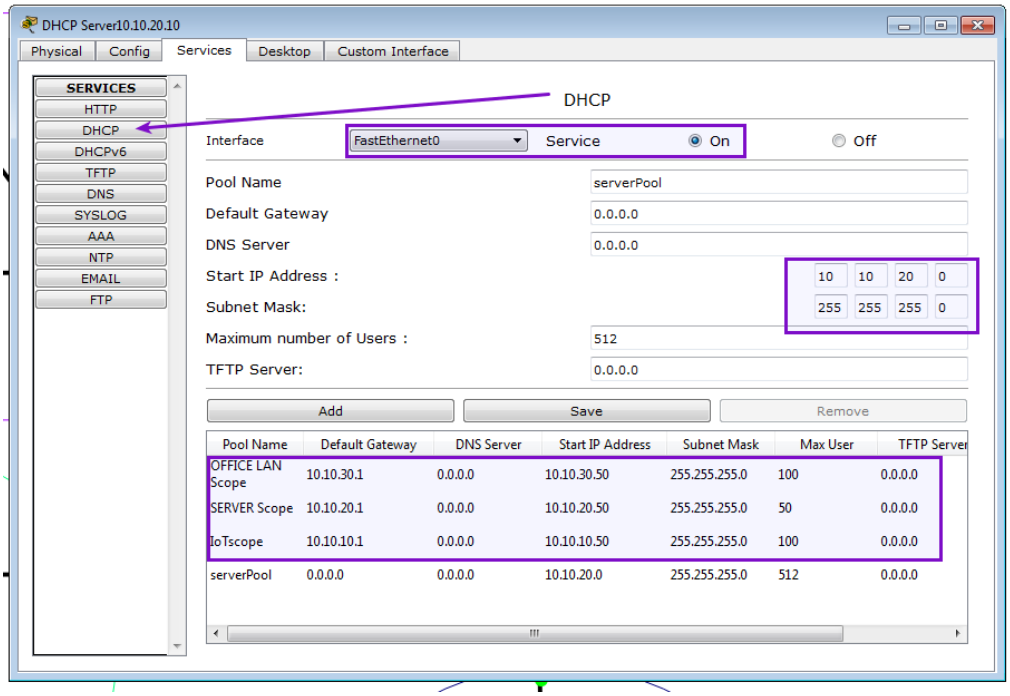
10. Configure Wireless AP



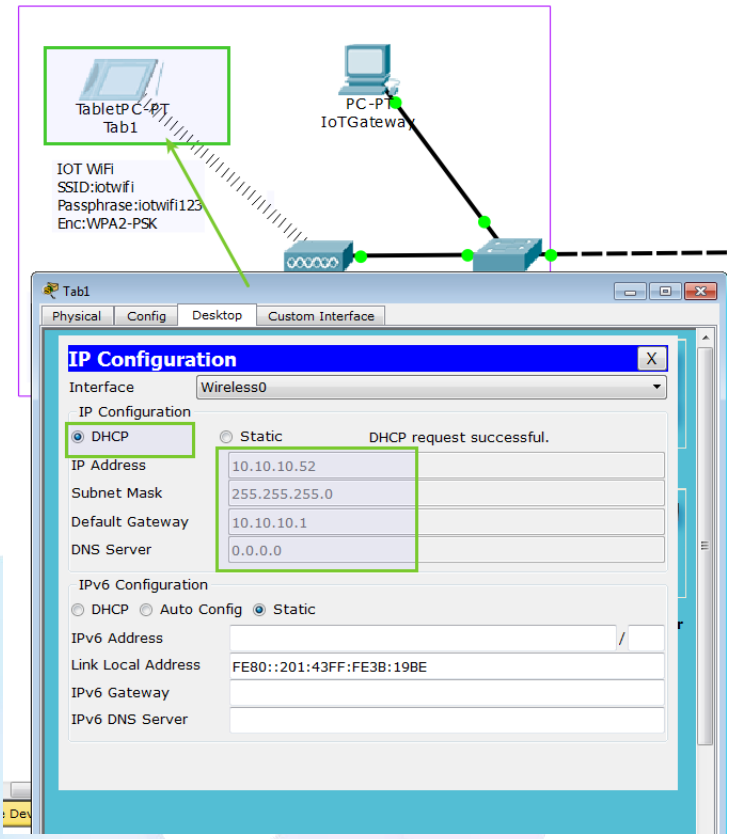
11. Ensure WirelessAP is connected to the correct VLAN.



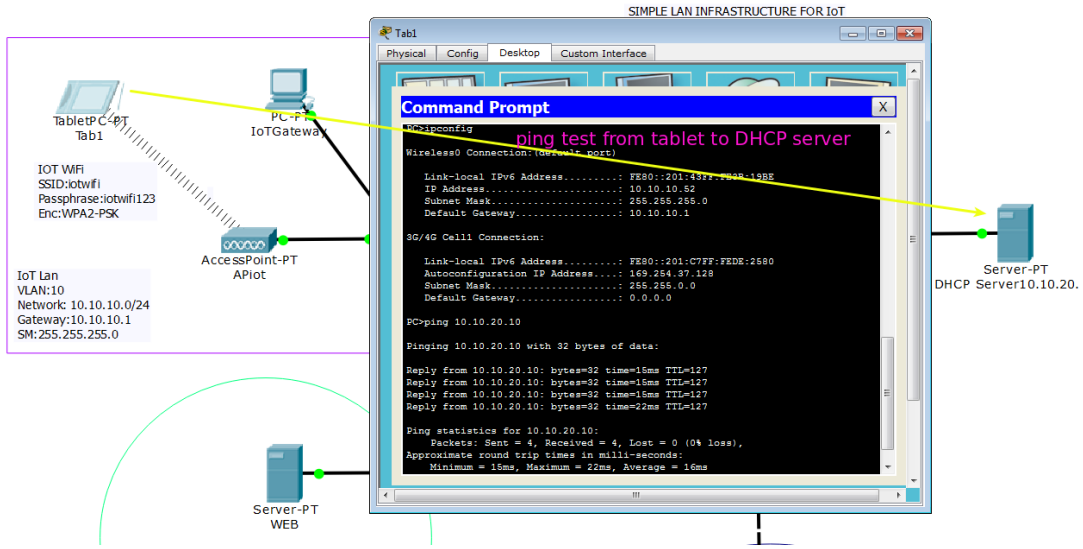
12. Configure DHCP Server and Subnet IP address scope



Checking device dhcp request status.



Successful ping test from Tab1(10.10.10.52) to DHCP Server (10.10.20.10). Please proceed with the ping test with other devices as well.



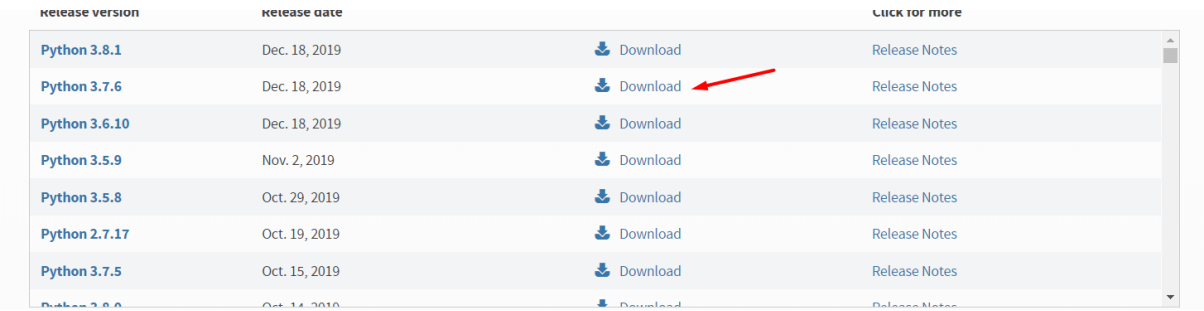
Topic 3: IoT Application Programming

Module 1: Getting Started with MicroPython [1hr]

Objective: In this lab we are going to install softwares used for micropython programming. MicroPython is a full Python compiler and runtime that runs on the bare-metal. You get an interactive prompt (the REPL) to execute commands immediately, along with the ability to run and import scripts from the built-in filesystem. uPyCraft is an IDE that works with Windows and Mac and designed with a simple interface which is convenient to use.

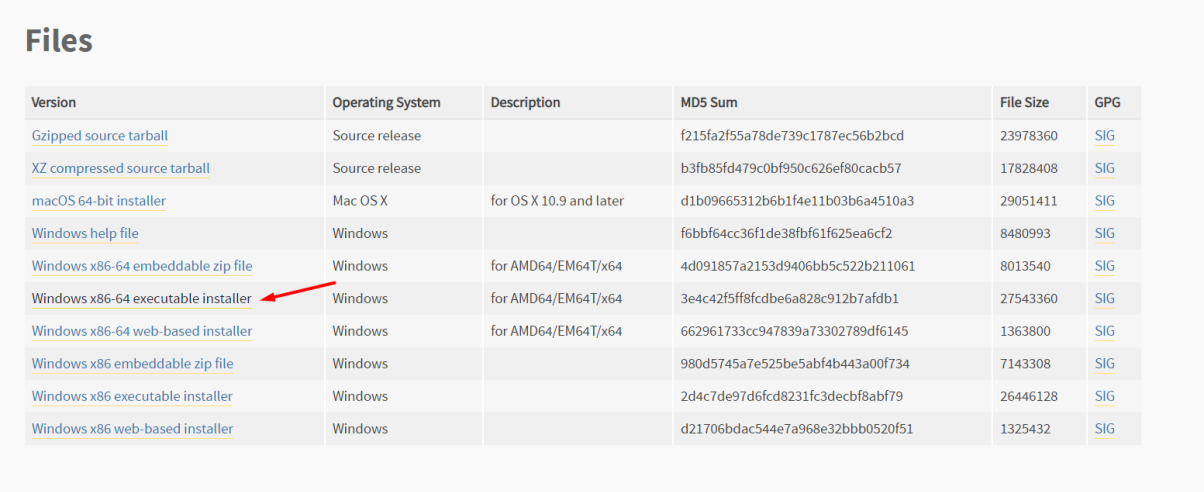
[Step#01] Install uPyCraft

1. To use uPyCraft we need to install python first
2. Go to https://www.python.org/downloads/release/python-382/



Release version	Release date		Click for more
Python 3.8.1	Dec. 18, 2019	Download	Release Notes
Python 3.7.6	Dec. 18, 2019	Download	Release Notes
Python 3.6.10	Dec. 18, 2019	Download	Release Notes
Python 3.5.9	Nov. 2, 2019	Download	Release Notes
Python 3.5.8	Oct. 29, 2019	Download	Release Notes
Python 2.7.17	Oct. 19, 2019	Download	Release Notes
Python 3.7.5	Oct. 15, 2019	Download	Release Notes

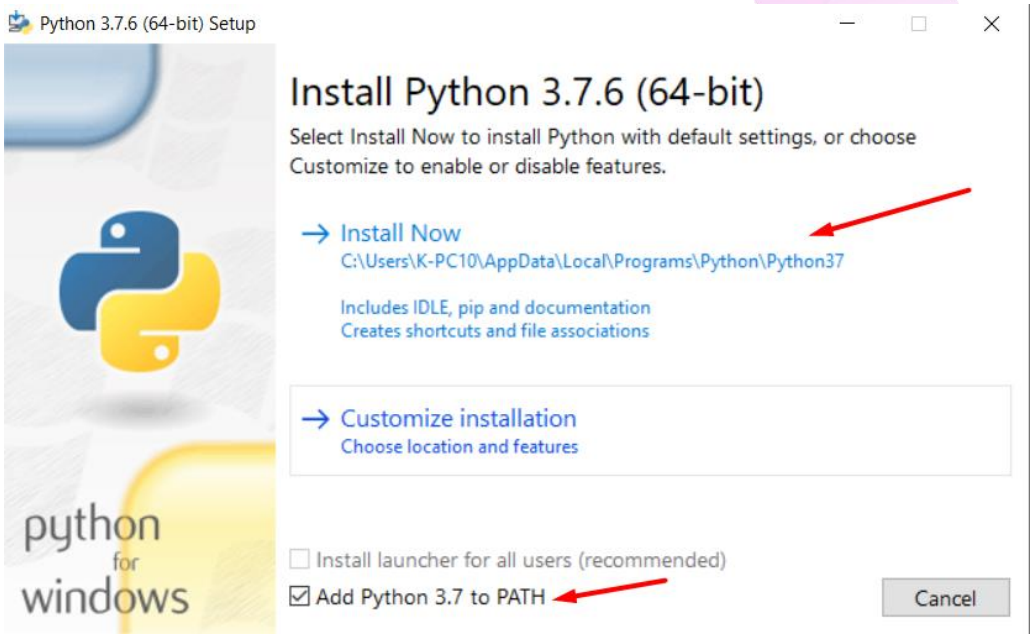
3. Scroll down and select executable file suitable for your OS version (64bit or 32bit(x86))



Files

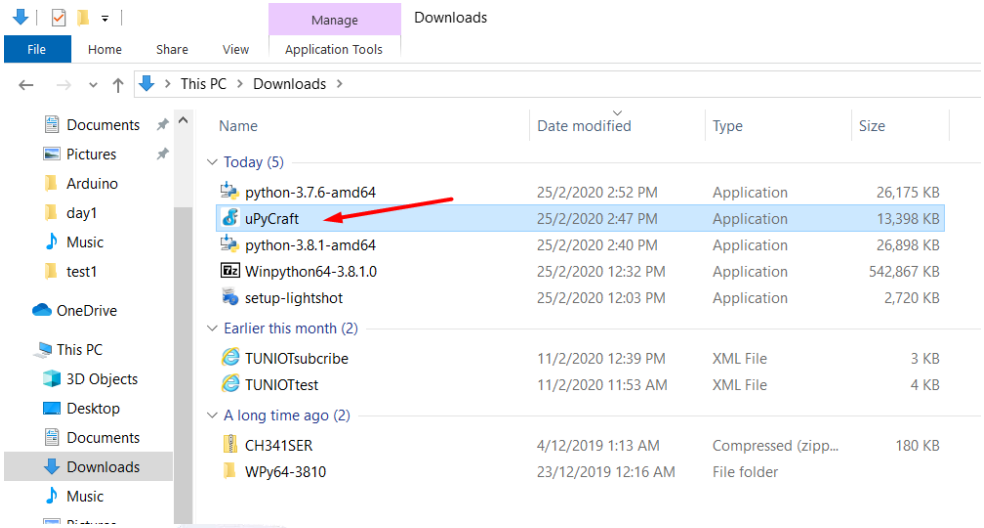
Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		f215fa2f55a78de739c1787ec56b2bcd	23978360	SIG
XZ compressed source tarball	Source release		b3fb85fd479c0bf950c626ef80cacb57	17828408	SIG
macOS 64-bit installer	Mac OS X	for OS X 10.9 and later	d1b09665312b6b1f4e11b03b6a4510a3	29051411	SIG
Windows help file	Windows		f6bbf64cc36f1de38fbf61f625ea6cf2	8480993	SIG
Windows x86-64 embeddable zip file	Windows	for AMD64/EM64T/x64	4d091857a2153d9406bb5c522b211061	8013540	SIG
Windows x86-64 executable installer	Windows	for AMD64/EM64T/x64	3e4c42f5ff8fcdbe6a828c912b7afdb1	27543360	SIG
Windows x86-64 web-based installer	Windows	for AMD64/EM64T/x64	662961733cc947839a73302789df6145	1363800	SIG
Windows x86 embeddable zip file	Windows		980d5745a7e525be5abf4b443a00f734	7143308	SIG
Windows x86 executable installer	Windows		2d4c7de97d6fcd8231fc3decbf8abf79	26446128	SIG
Windows x86 web-based installer	Windows		d21706bdac544e7a968e32bbb0520f51	1325432	SIG

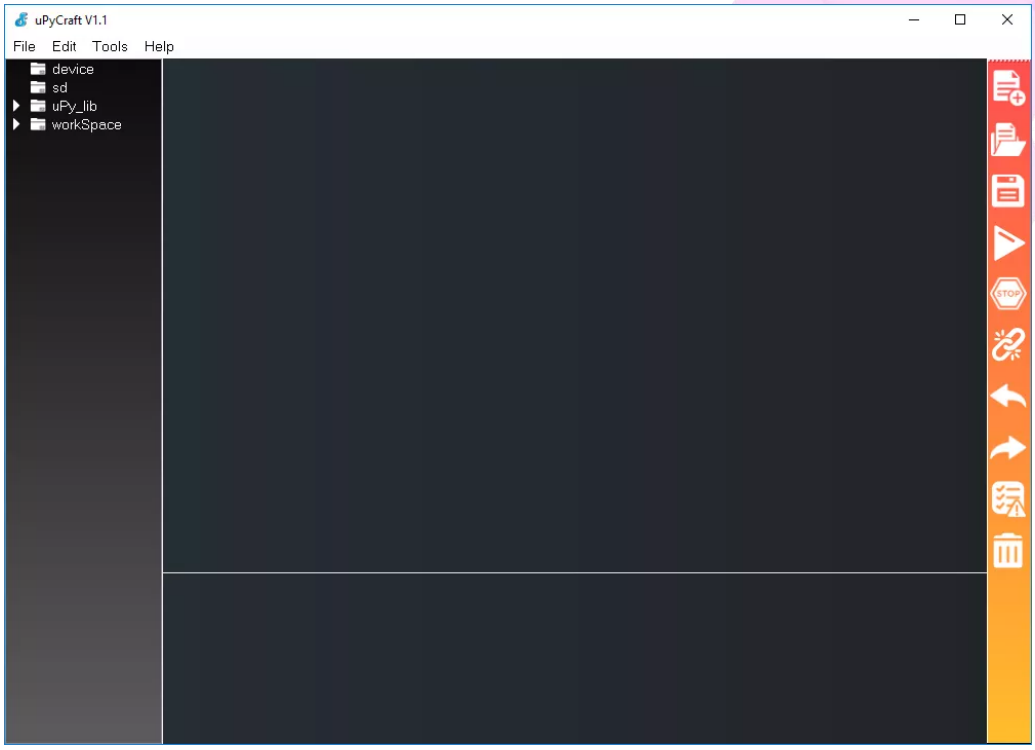
4. Install python - **check Add Python 3.8 to PATH !!!**



5. Download uPyCraft IDE for Windows. Go to this link : https://randomnerdtutorials.com/uPyCraftWindows

6. Launch uPyCraft IDE



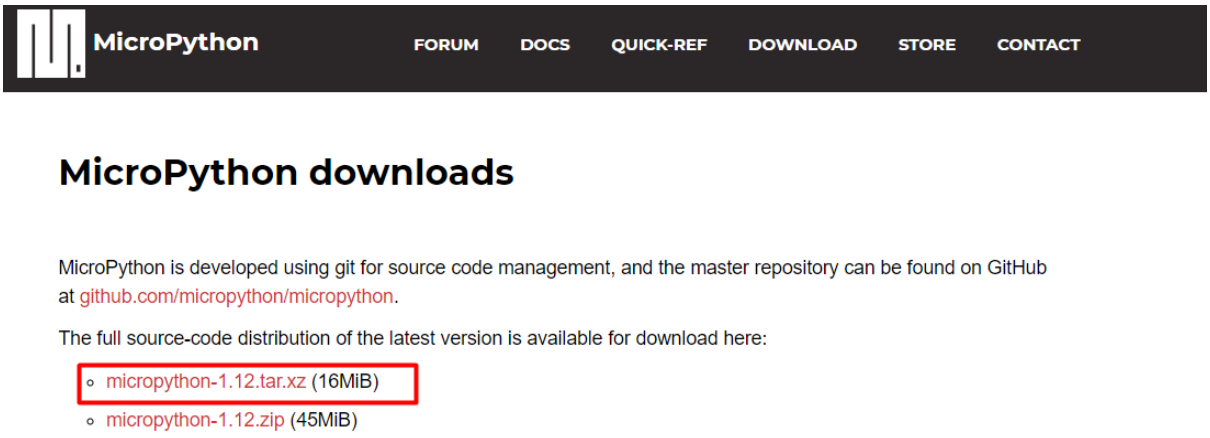


***If missing mscvr100.dll, install microsoft visual c++ 2010 redistributable package x86 and x64:

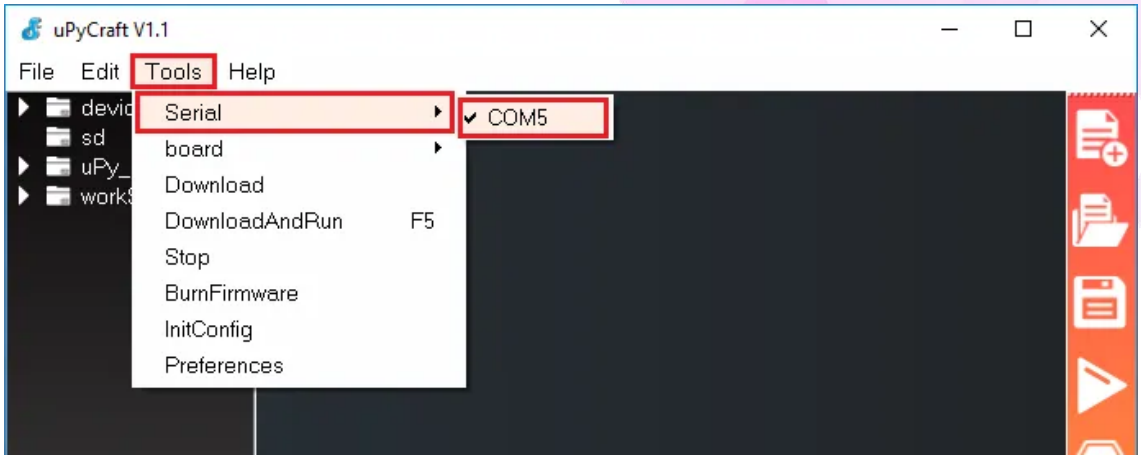
- https://www.microsoft.com/en-my/download/details.aspx?id=5555
- https://www.microsoft.com/en-us/download/details.aspx?id=14632

[Step#02] Flash MicroPython Firmware into ESP32/ESP8266

1. We'll be using this software to flash our ESP based boards with MicroPython firmware as well as to program the boards.
2. Download the latest version of MicroPython firmware for the ESP32. Go to https://micropython.org/download/#esp32.



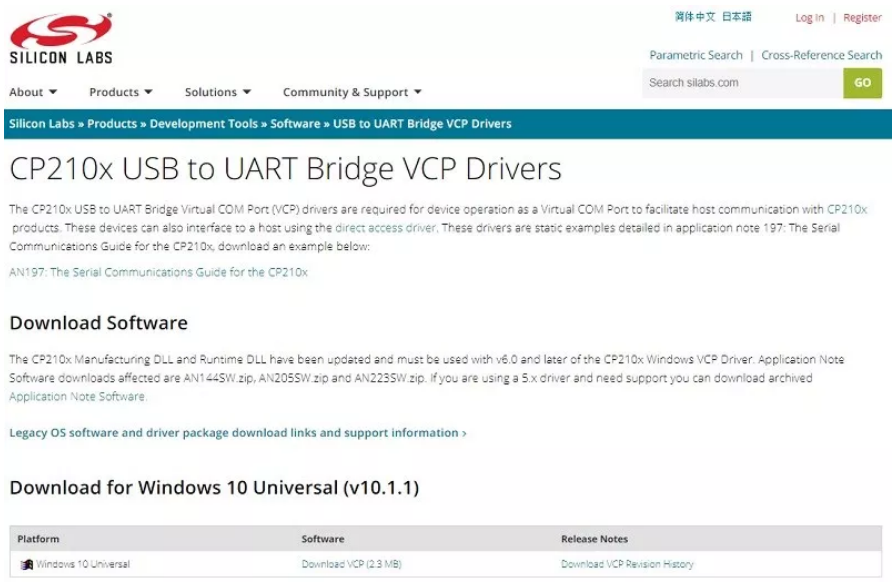
3. Go to Tools > Serial and select your ESP32 COM port (in our case it's COM5).



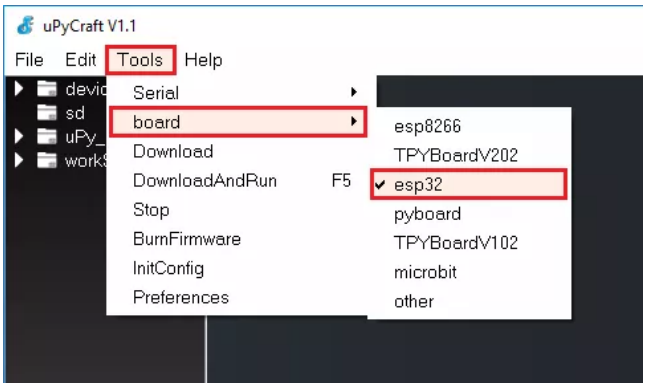
Important: if you plug your ESP32 board to your computer, but you can't find the ESP32 Port available in your uPyCraft IDE, it might be one of these two problems: USB drivers missing or USB cable without data wires.

If you don't see your ESP's COM port available, this often means you don't have the USB drivers installed. Take a closer look at the chip next to the voltage regulator on board and check its name.

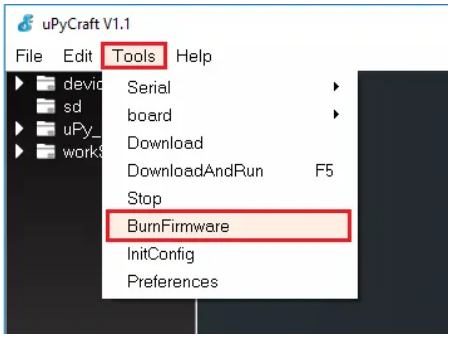
4. The ESP32 DEVKIT V1 DOIT board uses the CP2102 chip. download the CP2102 drivers on the Silicon Labs website.



5. Go to Tools > Board. To select the correct board which ours is esp32.

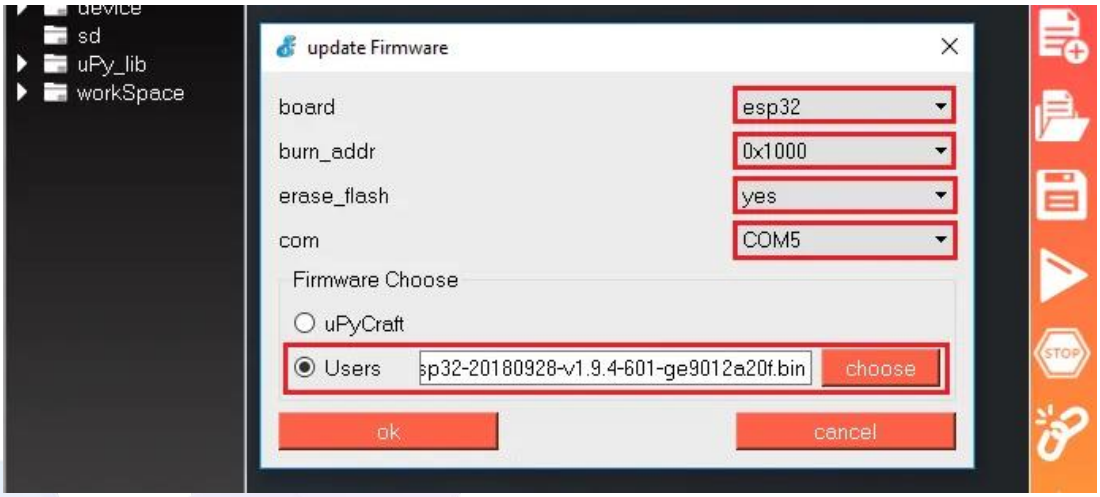


6. Finally, go to Tools > BurnFirmware menu to flash your ESP32 with MicroPython.

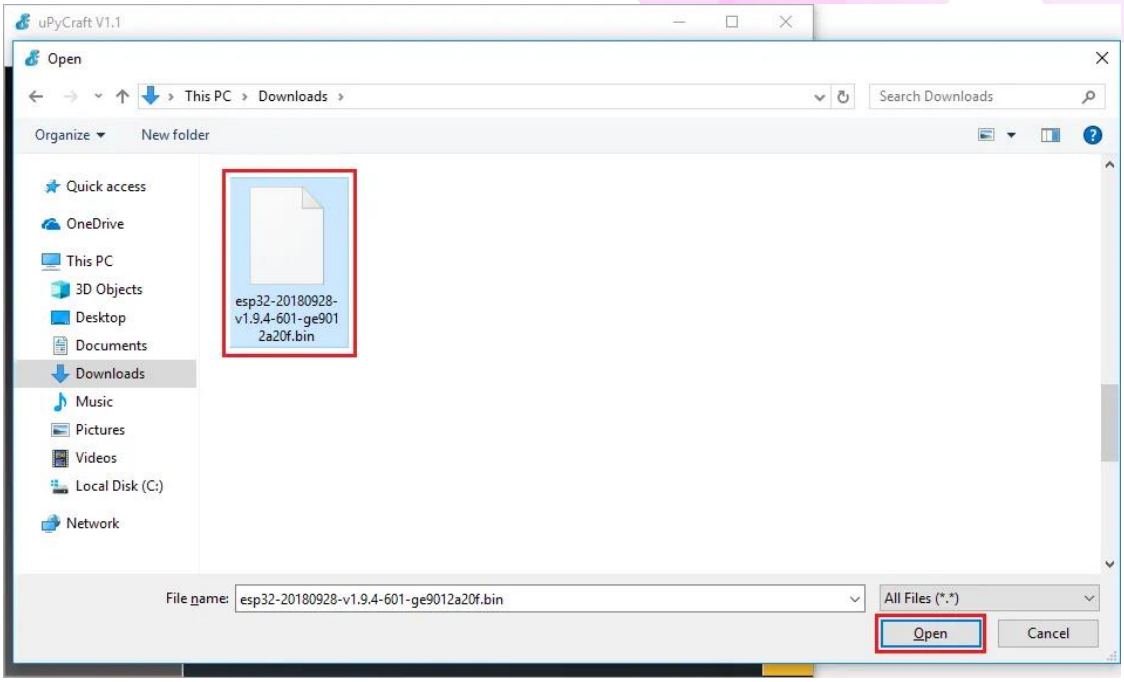


Select all these options to flash the ESP32 board:

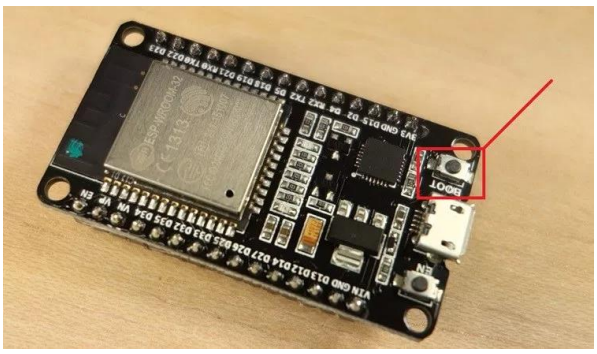
- ➔ board: esp32
- ➔ burn_addr: 0x1000
- ➔ erase_flash: yes
- ➔ com: COMX (in our case it's COM5)
- ➔ Firmware: Select "Users" and choose the ESP32 .bin file downloaded earlier



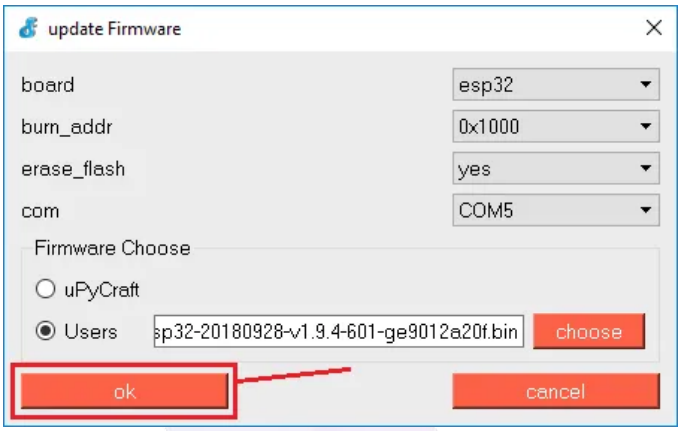
After pressing the “Choose” button, navigate to your Downloads folder and select the ESP32 .bin file:



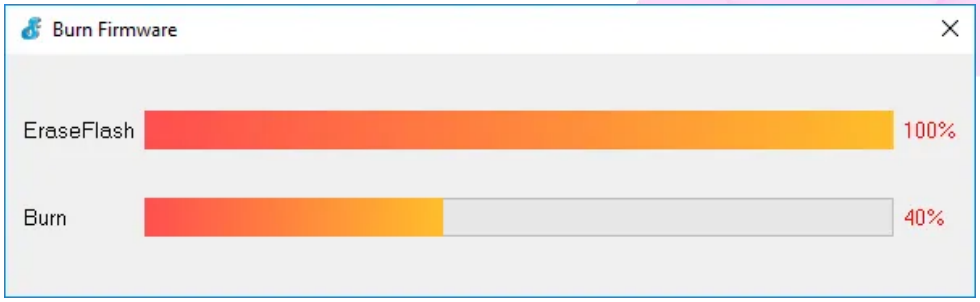
Having all the settings selected, hold-down the “BOOT/FLASH” button in your ESP32 board:



While holding down the “BOOT/FLASH“, click the “ok” button in the burn firmware window:



When the “EraseFlash” process begins, you can release the “BOOT/FLASH” button. After a few seconds, the firmware will be flashed into your ESP32 board.



Note: if the “EraseFlash” bar doesn’t move and you see an error message saying “erase false.“, it means that your ESP32 wasn’t in flashing mode. You need to repeat all the steps described earlier and hold the “BOOT/FLASH” button again to ensure that your ESP32 goes into flashing mode.

References:

1. https://randomnerdtutorials.com/flash-upload-micropython-firmware-esp32-esp8266/

Module 2: Basic MicroPython Programming [2hrs]

Objective: In this lab we are going to code using uPyCraft IDE. Throughout this lab, we will cover micropython syntax, element, comment, variable, data types and basic operators.

[Step#01] uPyCraft Familiarisation

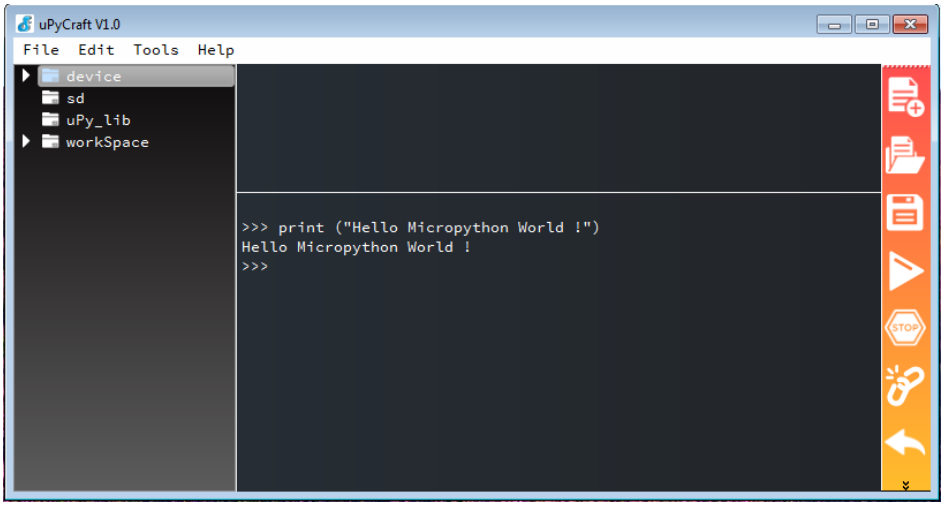
1. Let's execute an embedded program in a microcontroller, we start by using python REPL (read, evaluate, print loop).In the Shell, try several operations to see how it works.

2. After having the MicroPython firmware installed on your board and having the board connected to your computer through an USB cable, follow the next steps:

   i. Go to Tools > Board and select the board you're using.
   ii. Go to Tools > Port and select the com port your ESP is connected to.
   iii. Press the Connect button to establish a serial communication with your board.

   
   Connect/disconnect

   iv. The >>> should appear in the Shell window after a successful connection with your board. You can type the print command to test if it's working:



```
>>> 3+5
8
>>> 6-5
1
>>> 8*9
72
>>> 20/10
2.0
>>>
```

```
>>>
>>>
>>> 2==5
False
>>> 4==4
True
>>> 69874 != 65
True
>>> 3>2
True
>>>
```

```
>>> a = 10
>>> b = 12
>>> c = 20.6
>>> text = 'abcdef'
>>> d = True
>>>
>>>
>>> type(a)
<class 'int'>
>>> type(b)
<class 'int'>
>>> type(b)
<class 'int'>
>>>
>>> type(c)
<class 'float'>
>>> type(text)
<class 'str'>
>>>
```

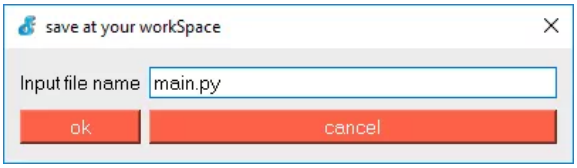
```
>>> number = 1
>>>
>>> while number <= 10:
...     print(number)
...     number = number +1
...
...
1
2
3
4
5
6
7
8
9
10
>>>
```

For this exercise, press Shift+Enter to execute.

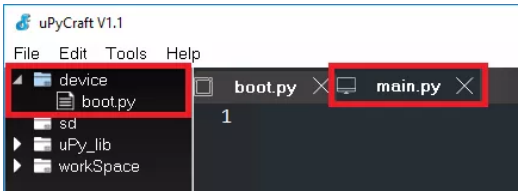
3. Creating the main.py file on your board.
   I. Press the “New file” button to create a new file.

   II. Press the “Save file” button to save the file in your computer.

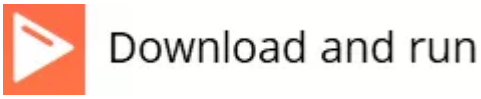
   III. A new window opens, name your file main.py and save it in your computer.



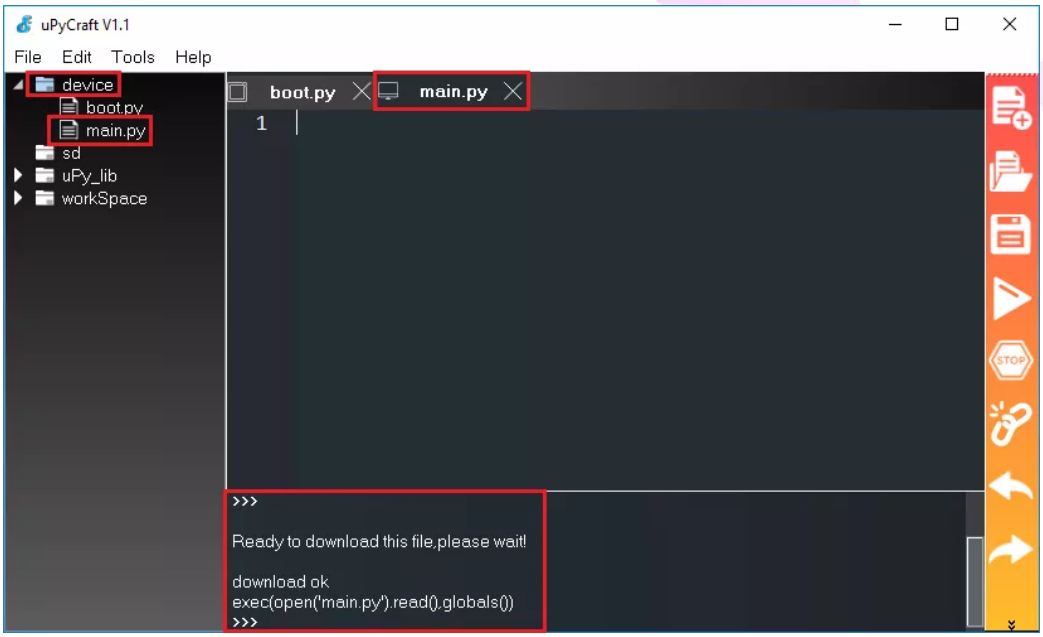
   IV. After that, you should see the following in your uPyCraft IDE (the boot.py file in your device and a new tab with the main.py file)



   V. Click the “Download and run” button to upload the file to your ESP board.



VI. The device directory should now load the main.py file. Your ESP has the file main.py stored.



4. Uploading the blink LED script.
   I. Code to the Editor on the main.py file.
   II. Press the “Stop” button to stop any script from running in your board.

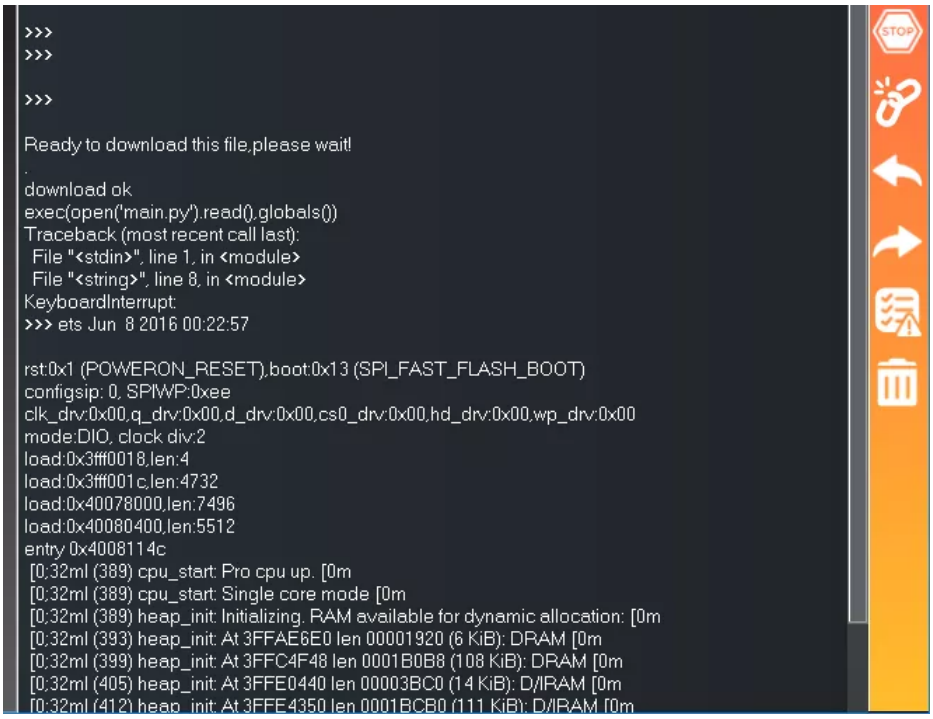
   III. Click the “Download and Run button” to upload the script to the ESP32 or ESP8266.

   IV. You should see a message saying “download ok” in the Shell window.

5. Testing the script
   I. Press the “Stop” button

II. Press the on-board ESP32/ESP8266 EN (ENABLE) or RST (RESET) button to restart your board and run the script from the start:

III. If you're using an ESP32, your Terminal messages should look something as shown in the following figure after a EN/RST button press:



6. print() and sleep()

print() and sleep()

- Import the **sleep** class from the **time** module.

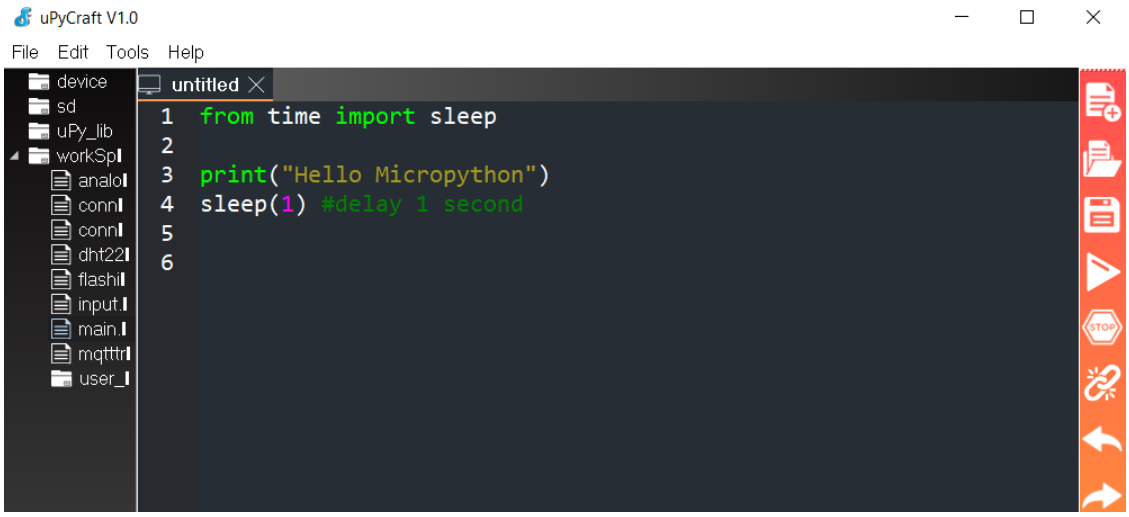
```
from time import sleep
```

- To delay, use sleep(time_in_second). For example:

```
sleep(1)
```

- To print any value use print(val). For example:

```
print("Hello Micropython")
```

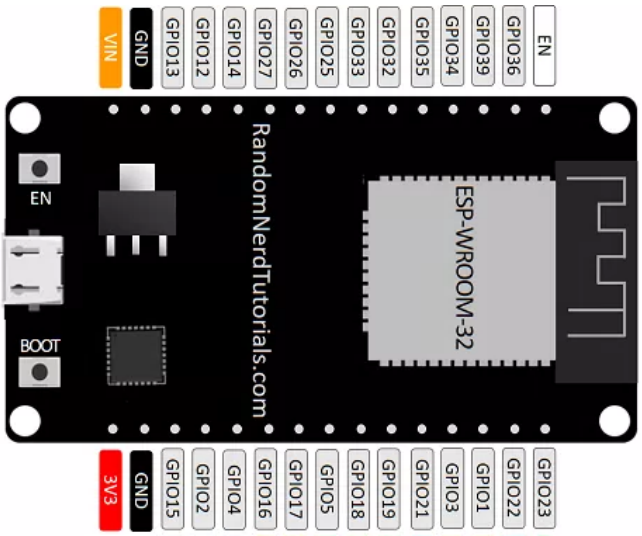


References:

1. https://randomnerdtutorials.com/flash-upload-micropython-firmware-esp32-esp8266/

Module 3: ESP32 Programming [3hr]

Objective: In this lab we are going to code ESP32 microcontroller using uPyCraft IDE. Throughout this lab, we will cover ESP32 digital input and output, analog input, DHT sensor and PWM.



[Step#01] ESP32 Digital Output Pin

1. We start with ESP32 digital output.

Digital Outputs

- You start by importing the **Pin** class from the **machine** module.

```
from machine import Pin
```

- To set a GPIO on or off, first you need to set it as an output. For example:

```
led = Pin(5, Pin.OUT)
```

- To control the GPIO, use the value() method on the Pin object and pass 1 or 0 as argument. For example, the following command sets a Pin object (led) to HIGH:

```
led.value(1)
```

- To set the GPIO to LOW, pass 0 as argument:

```
led.value(0)
```

2. Prepare all the components needed and build the following circuit:

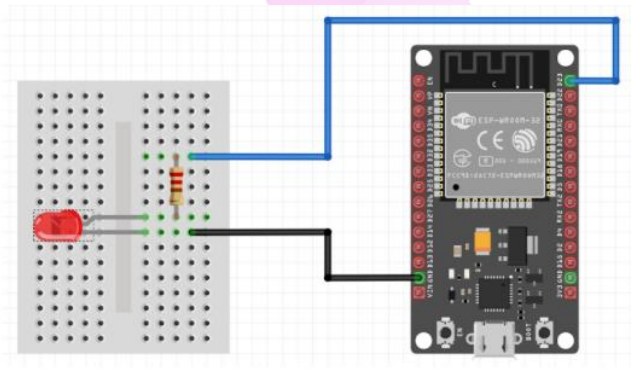
Blinking LED

What you'll need:

1) LED in any color (1 unit)
2) Resistor 330 Ohm (1 unit)
3) Male to Female jumper wire (2 unit)
4) ESP32 (1 unit)
5) Breadboard (1 unit)
6) USB Micro B cable (1 unit)

Instructions:

The LED will light up for 2 seconds and turn off for 1 second.



3. Write the program into uPyCraft IDE, in the main.py and upload it into ESP32 board.

*main.py

```
1
2  import time
3  from machine import Pin
4
5
6  led1 = Pin(23,Pin.OUT)
7
8  while True:
9     led1.value(1)
10    time.sleep(2)
11    led1.value(0)
12    time.sleep(1)
13
14
```

2. Prepare all the components needed and build the following circuit:

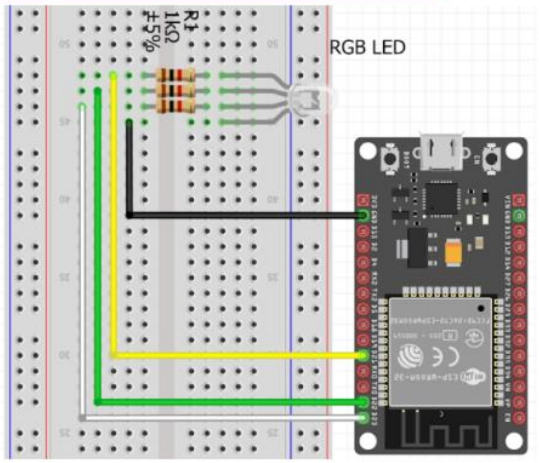
RGB LED Runnning Light

What you'll need:

1) RGB LED (1 unit)
2) Resistor 1K ohm (3 unit)
3) Male to Female jumper wire (5 unit)
4) ESP32 (1 unit)
5) Breadboard (1 unit)
6) USB Micro B cable (1 unit)

Instructions:

The LED will light up in GREEN for 1 second follows by RED and lastly with BLUE



4. Write the program into uPyCraft IDE, in the main.py and upload it into ESP32 board.

main.py

```
1
2  import time
3  from machine import Pin
4
5  led1 = Pin(21,Pin.OUT)
6  led2 = Pin(22,Pin.OUT)
7  led3 = Pin(23,Pin.OUT)
8
9  while True:
10   led1.value(1)
11   led2.value(0)
12   led3.value(0)
13   time.sleep(1.5)
14   led1.value(0)
15   led2.value(1)
16   led3.value(0)
17   time.sleep(1.5)
18   led1.value(0)
19   led2.value(0)
20   led3.value(1)
21   time.sleep(1.5)
22
```

[Step#02] ESP32 Digital Input Pin

1. We continue with ESP32 digital input.

Digital Inputs

- You start by importing the **Pin** class from the **machine** module.

```
from machine import Pin
```

- To get the value of a GPIO, first you need to create a Pin object and set it as an input. For example:

```
button = Pin(4, Pin.IN)
```

- Then, to get is value, you need to use the value() method on the Pin object without passing any argument. For example, to get the state of a Pin object called button, use the following expression:

```
button.value()
```

2. Prepare all the components needed and build the following circuit:

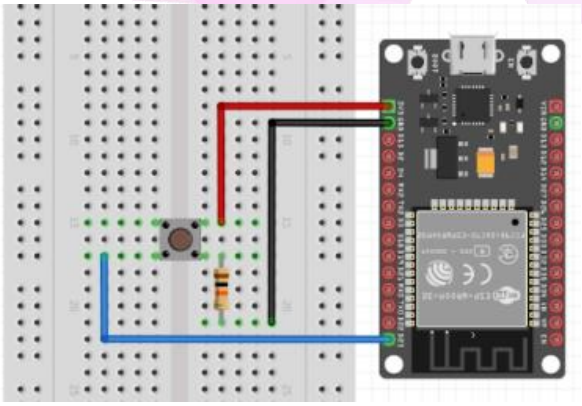
Button - Digital Input

What you'll need:

1) Contact switch or button (1 unit each)
2) Resistor 10K ohm (1 unit)
3) Male to Female jumper wire (3 unit)
4) NodeMCU (1 unit)
5) Breadboard (1 unit)
6) USB Micro B cable (1 unit)

Instructions:

The button will write a value of 1 (ON) when pushed and 0 (OFF) when released.
Show your result at the terminal



3. Write the program into uPyCraft IDE, in the main.py and upload it into ESP32 board.

*main.py

```
1
2  import time
3  from machine import Pin
4
5
6  button = Pin(23,Pin.IN)
7
8  while True:
9    print(button.value())
10
11
```

4. Prepare all the components needed and build the following circuit:

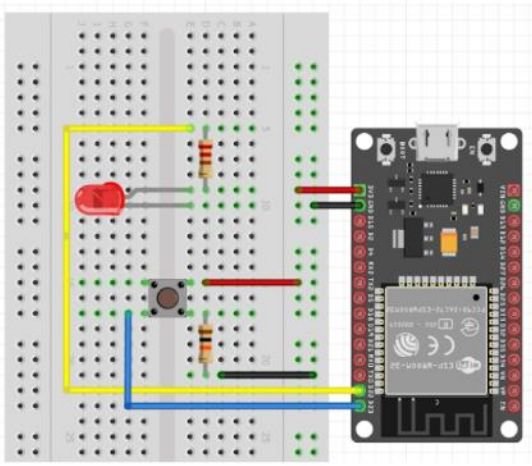
Digital Input & Digital Output

What you'll need:

1) Contact switch or button (1 unit)
2) LED in any color (1 unit)
3) Resistor 10K ohm (1 unit)
4) Resistor 220 ohm (1 unit)
5) Male to Female jumper wire (3 unit)
6) ESP32 (1 unit)
7) Breadboard (1 unit)
8) USB Micro B cable (1 unit)

Instructions:

The button will write a value of 1 (ON) when pushed and will also lights up the LED



5. Write the program into uPyCraft IDE, in the main.py and upload it into ESP32 board.

*main.py input.py

```
1
2  import time
3  from machine import Pin
4
5
6  while True:
7    print(button.value())
8     button = Pin(23, Pin.IN)
9     print(button.value())
10    led = Pin(22, Pin.OUT)
11    led.value(button)
```

[Step#03] ESP32 - DHT sensor

1. We continue with DHT Sensor.

DHT Sensor

- Import the **DHT22** class from the **dht** module.

```
from dht import DHT11
```

- To get the value of a DHT sensor, first you need to create a DHT11 object and set at which pin it connected. For example:

```
sensor = DHT11(Pin(2))
```

- Then, to get is value, you need to use the temperature() or humidity() method on the DHT11 object without passing any argument. For example, to get the reading of a DHT11 object called sensor, use the following expression:

```
sensor.temperature() or sensor.humidity()
```

2. Prepare all the components needed.

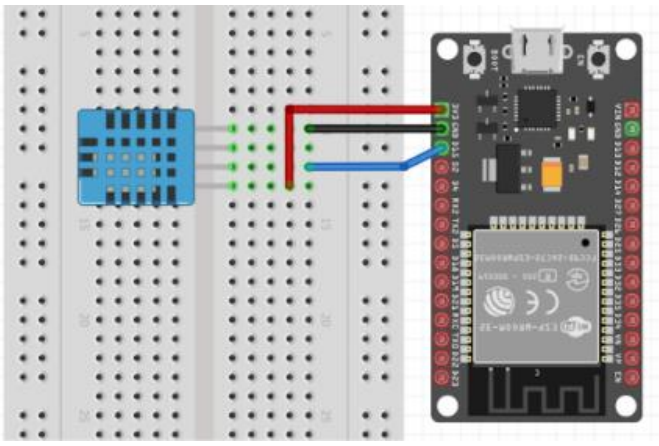
DHT Sensor

What you'll need:

1) DHT11 (1 unit)
2) Male to Female jumper wire (3 unit)
3) ESP32 (1 unit)
4) Breadboard (1 unit)
5) USB Micro B cable (1 unit)

Instructions:

DHT11 will read room temperature and result can be seen on terminal



3. Write the program into uPyCraft IDE, in the main.py and download it into ESP32 board.

*dht22.py

```
import time
from dht import DHT22
from machine import Pin
dht22 = DHT22(Pin(15))


while True:
  time.sleep(3)
  dht22.measure()
  print(dht22.temperature(),dht22.humidity())

```

[Step#04] ESP32 Analog Input

1. We start with ESP32 analog input.

Analog Readings – ESP32

- To read analog inputs, import the ADC class in addition to the Pin class from the machine module.

```
from machine import Pin, ADC
```

- Then, create an ADC object called pot on GPIO 34.

```
pot = ADC(Pin(34))
```

- The following line defines that we want to be able to read voltage in full range. This means we want to read voltage from 0 to 3.3V.

```
pot.atten(ADC.ATTN_11DB)
```

- Read the pot value and save it in the pot_value variable. To read the value from the pot, simply use the read() method on the pot object.

```
pot_value = pot.read()
```

2. Prepare all the components needed and build the following circuit:

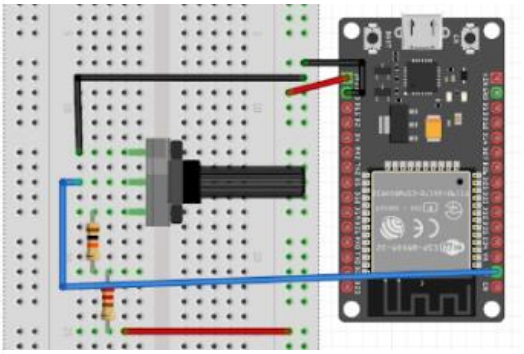
Analog Input

What you'll need:

1) Potentiometer/Variable Resistor (1 unit)
2) Male to Female jumper wire (3 unit)
3) ESP32 (1 unit)
4) Breadboard (1 unit)
5) USB Micro B cable (1 unit)

Instructions:

Analog reading can be seen on terminal



3. Write the program into uPyCraft IDE, in the main.py and upload it into ESP32 board.

*analogInput.py

```
import time
from machine import Pin, ADC


pot = ADC(Pin(34))
pot.atten(ADC.ATTN_11DB)
pot_value = pot.read()


while True:

  if pot_value >1400:
    print(pot_value)
    print("lebih dari 1400")

  else:
    print(pot_value)
    print("kurang dari 1400")

  time.sleep(2)
  pot_value = pot.read()

```

[Step#05] ESP32 PWM

1. We continue with ESP32 PWM.

PWM – ESP32

- The range() function has the following syntax:

```
range(start, stop, step)
```

  - Start: a number that specifies at which position to start. We want to start with 0 duty cycle;
  - Stop: a number that specifies at which position we want to stop, excluding that value. The maximum duty cycle is 1023, because we are incrementing 1 in each loop, the last value should be 1023+1. So, we'll use 1024.
  - Step: an integer number that specifies the incrementation. By default, incrementation is 1.

- In each for loop, we set the LED's duty cycle to the current duty_cycle value:

```
led.duty(duty_cycle)
```

- After that, the duty_cycle variable is incremented by 1.

2. Prepare all the components needed and build the following circuit:

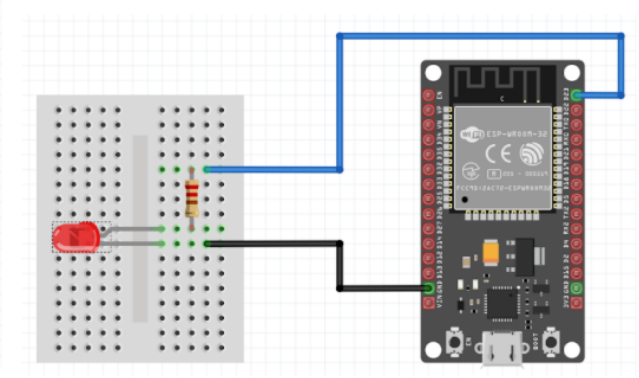
PWM

What you'll need:

1) LED (1 unit)
2) Male to Female jumper wire (3 unit)
3) ESP32 (1 unit)
4) Breadboard (1 unit)
5) potentiometer
6) USB Micro B cable (1 unit)

Instructions:

Dim the brightness of an LED by changing the duty cycle over time.



3. Write the program into uPyCraft IDE, in the main.py and upload it into ESP32 board.

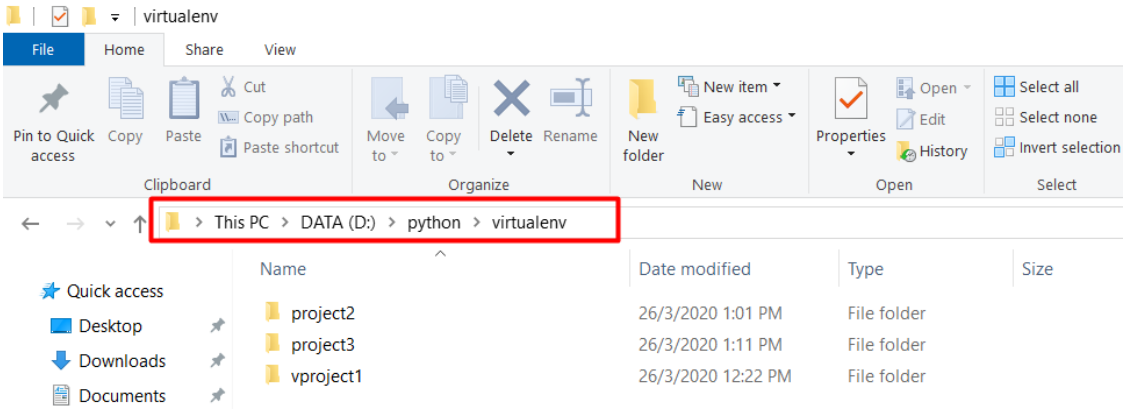
Topic 4: Web Apps Development for IoT

Module 1: Getting Started With Web Development [3hrs]

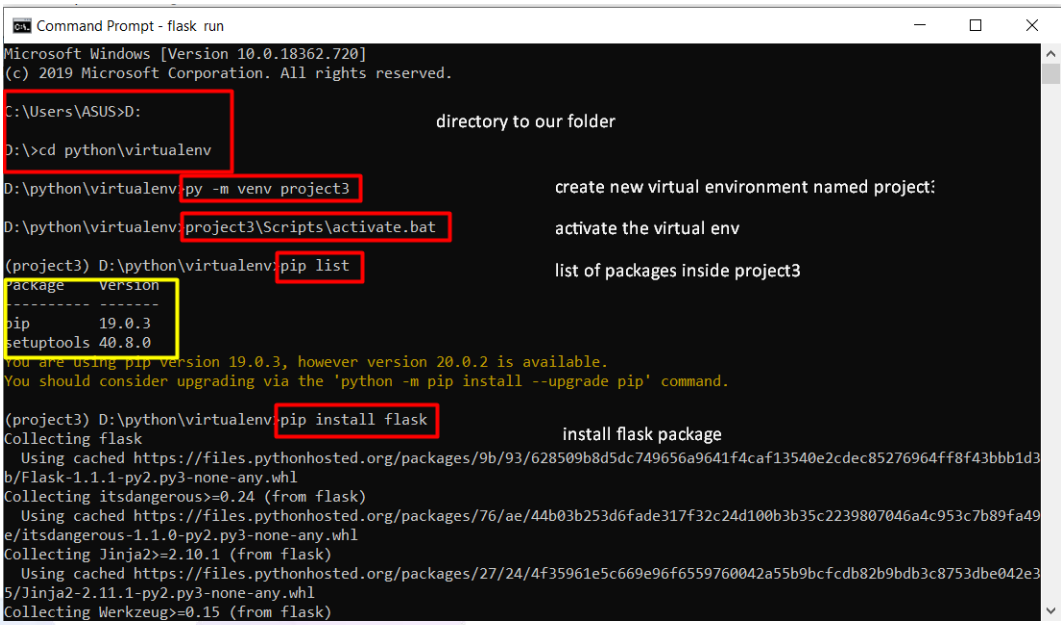
Objective: In this lab we are going to create a virtual environment. So we will install the flask package inside this virtual environment. As we do not install into our main python system, so every project is not going to be affected with the packages updates etc.

[Step#01] Creating virtual project environment

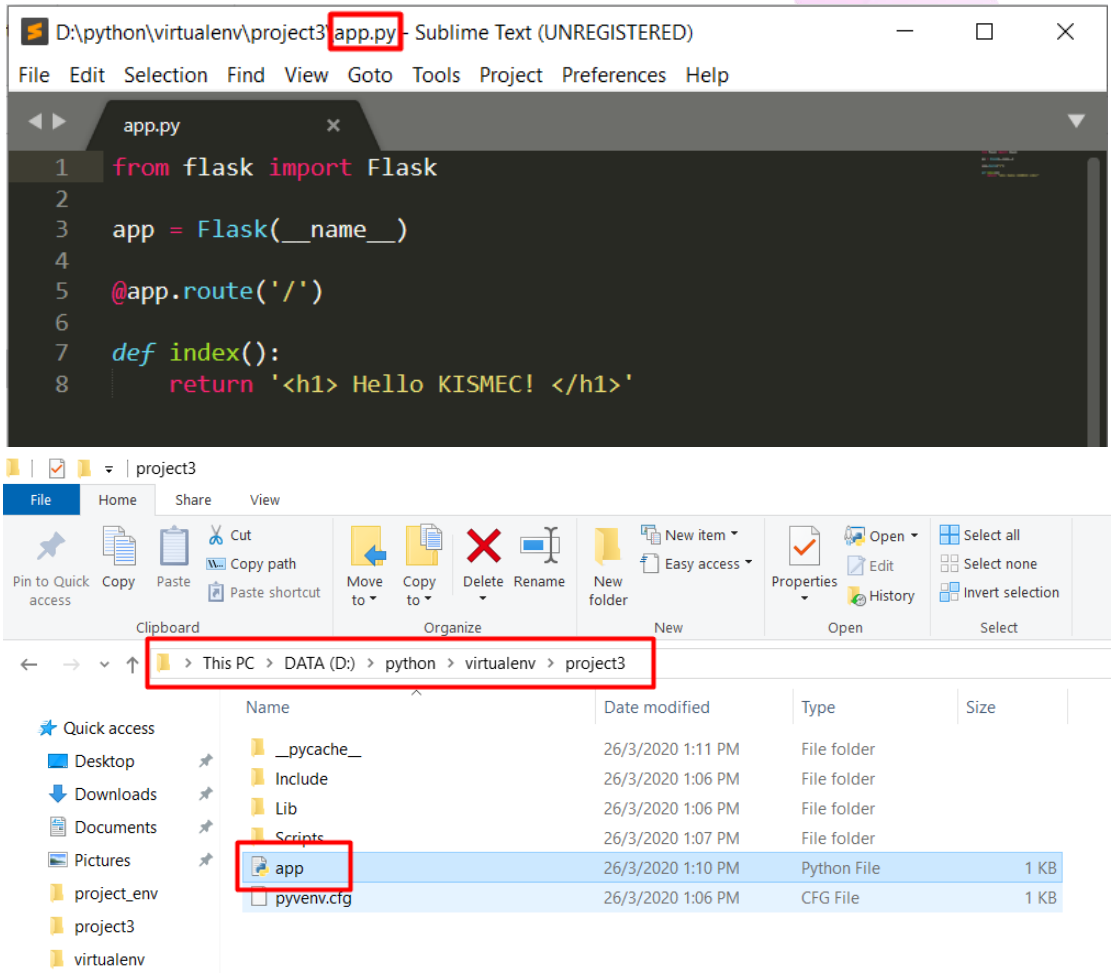
1. Create a new folder to save all our virtual project.



2. Create virtual environment, activate it and install flask inside

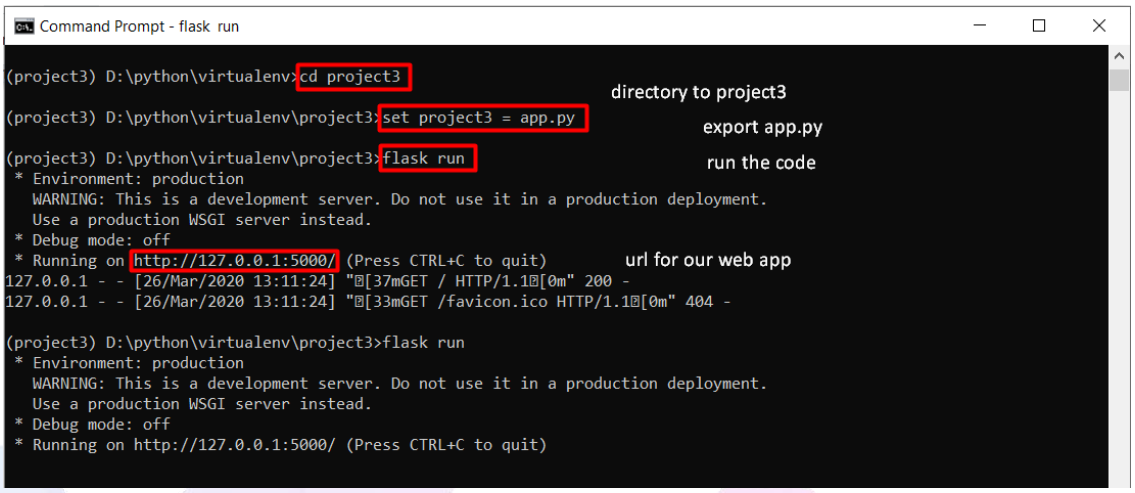


3. Code our first web app and saves as app.py, type all files inside project3 folder.



4. Export and run

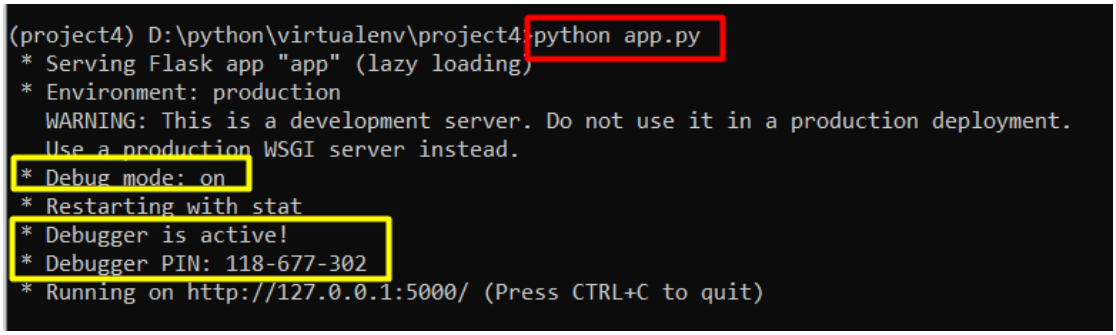
   * *change set project3 = app.py to set FLASK_APP = app.py*



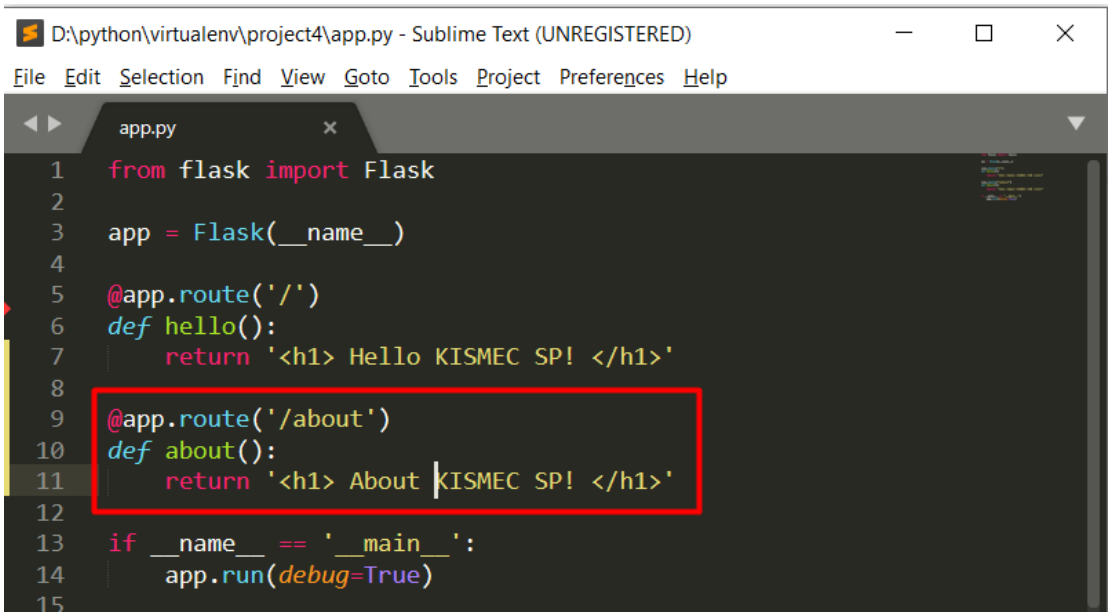
5. Go to http://127.0.0.1:5000/ in your browser

6. To run in debug mode, append this code. In debug mode you don't need to stop if you change your code. Save your new codes and refresh the page only.

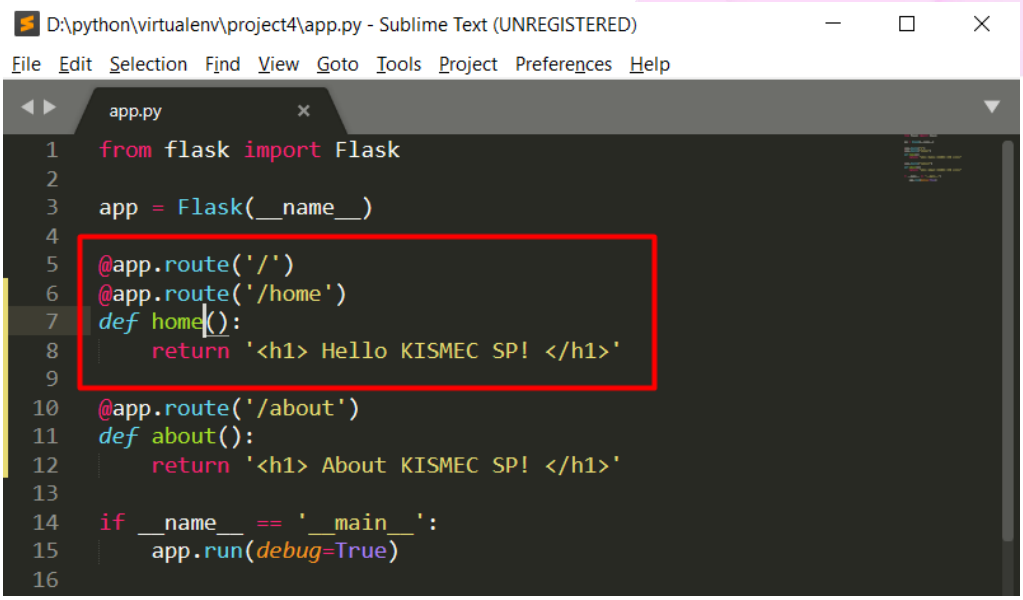
7. Run your code - python app.py



8. We want to create page with a route

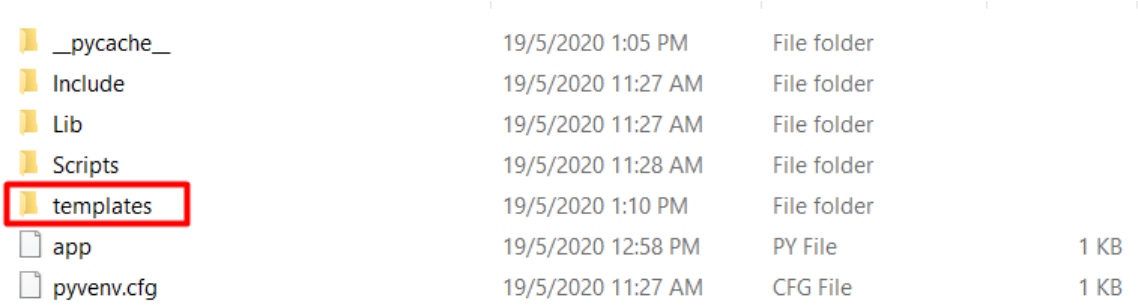


9. Route home + no route

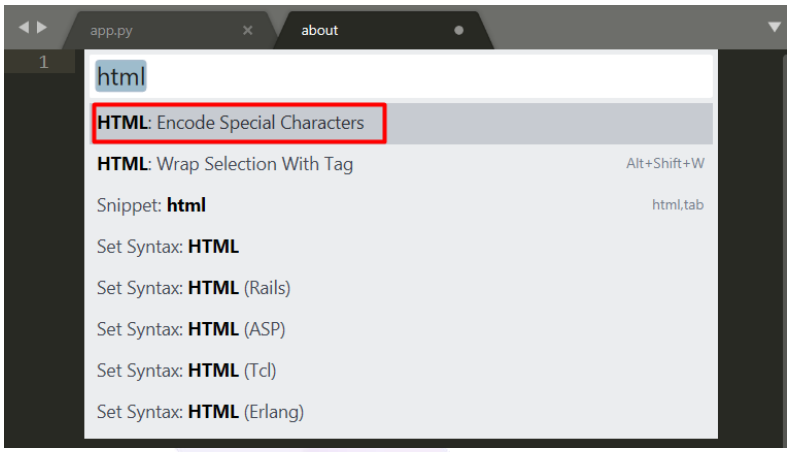


[Step#02] Using template

1. Create new folder and name it “templates”



2. Create new html file and save in templates folder
3. Press "Ctrl-shift-P" and type HTML. Select "HTML: Encode Special Characters".



4. Code in html language

```
app.py | about
1 html

type html and press tab
```

```
app.py | about
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title></title>
5  </head>
6  <body>
7
8  </body>
9  </html>
```

5. Import render_template

```
app.py | about
1  from flask import Flask, render_template
2
3  app = Flask(__name__)
4
5  @app.route('/')
6
7  def index():
8      return '<h1> Hello World!!! </h1>'
9
10 @app.route('/about')
11
12 def about():
13     return render_template('about.html')
14
15 if __name__ == '__main__':
16     app.run(debug=True)
```

6. Pass in keyword arguments to the template, like in the example with *my_string*. Render value to the template

```python
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/')

def index():
    return '<h1> Hello World!!! </h1>'

@app.route('/about')

def about():
    return render_template('about.html', my_string="Passed value")

if __name__ == '__main__':
    app.run(debug=True)
```

7. Take the value from flask app

```html
<!DOCTYPE html>
<html>
<head>
    <title>About</title>
</head>
<body>
    <h1>About Page</h1>
    <h2>{{my_string}}</h2>
</body>
</html>
```

Reference:

- https://realpython.com/primer-on-jinja-templating/
- https://jinja.palletsprojects.com/en/2.11.x/templates/#template-inheritance

There are a few kinds of delimiters. The default Jinja delimiters are configured as follows:

- {% ... %} for Statements
- {{ ... }} for Expressions to print to the template output
- {# ... #} for Comments not included in the template output
- # ... ## for Line Statements

app.py

```python
from flask import Flask, render_template

app = Flask(__name__)

my_list = ['hello','hi','assalamualaikum']


@app.route('/')

def index():
    return '<h1> Hello World!!! </h1>'

@app.route('/about')

def about():
    return render_template('about.html', my_string="Passed value", my_list=my_list)

if __name__== '__main__':
    app.run(debug=True)
```

about.html

```html
<!DOCTYPE html>
<html>
<head>
    <title>About</title>
</head>
<body>
    <h1>About Page</h1>
    <h2>{{my_string}}</h2>
    <h2>{{my_list[1]}}</h2>
</body>
</html>
```

app.py about.html

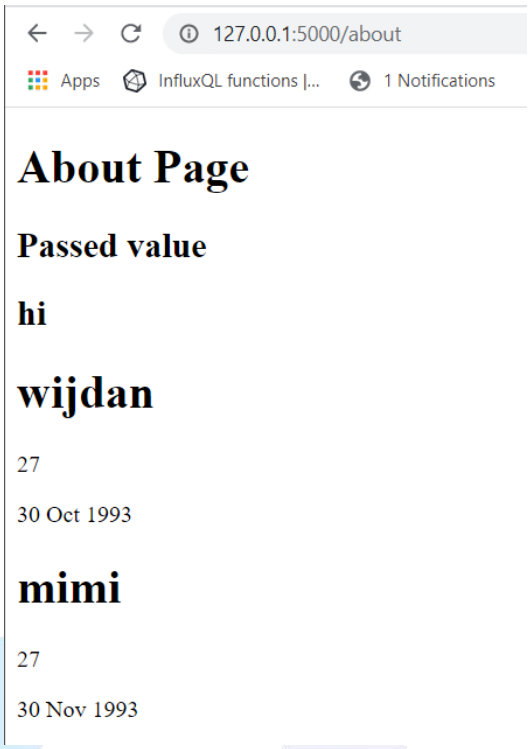
```python
from flask import Flask, render_template

app = Flask(__name__)

my_list = ['hello','hi','assalamualaikum']

my_dict = [
    {

        'name':'wijdan',
        'age':27,
        'dateofbirth':'30 Oct 1993'
    },
    {

        'name':'mimi',
        'age':27,
        'dateofbirth':'30 Nov 1993'
    }
]

@app.route('/')

def index():
    return '<h1> Hello World!!! </h1>'

@app.route('/about')

def about():
    return render_template('about.html', my_string="Passed value", my_list=my_list, my_dict=my_dict)

if __name__ == '__main__':
    app.run(debug=True)
```

Create a for loop:

app.py | about.html

```
<!DOCTYPE html>
<html>
<head>
    <title>About</title>
</head>
<body>
    <h1>About Page</h1>
    <h2>{{my_string}}</h2>
    <h2>{{my_list[1]}}</h2>

    {% for n in my_dict %}

        <h1>{{n.name}}</h1>
        <p>{{n.age}}</p>
        <p>{{n.dateofbirth}}</p>
    {% endfor %}
</body>
</html>
```

The output would be:



References:

1. https://blog.miguelgrinberg.com/post/designing-a-restful-api-with-python-and-flask

Module 2: HTML and Jinja Templating for Web Application [3hrs]

Objective:
In this lab we will learn how to create web pages that give structure and layout to your web application using HTML codes and JINJA templating.

[Step#01] Create a HTML page

1. Create a directory in your project directory to place the HTML template files.

   hint: templates file must be placed in "/static/templates"

2. Create the base template file and give it a name as "base.html".

   hint: base.html must be a valid standard html file with minimum html, head and body tags.

3. Write HTML code to structure your base.html
   a. Define header block
   b. Define content block
   c. Define footer block
4. Write HTML code for index.html
   a. write HTML Jinja code to inherit main.html

      hint: use "extend" to inherit main.html into your index.html

   b. Write HTML Jinja code to display content for index.html
   c. Write an html code to welcome visitors to your web page.
   d. Edit flask @app.route for "/" to return the template as response.

      hint: return render_template("index.html")

   e. Re-launch flask app and use an internet browser to access to index.html, i.e. point the browser the the web application root end-point "/"
   f. Try to change the content of your index.html, and notice the changes.
5. Create another html file in the template directory and name it "dashboard.html"
   a. Copy index.html and rename it to "dashboard.html".

b. Use Jinja code to dynamically parse data to the web application to be rendered in your dashboard.html

hint: use double curly brackets "{{}}" as a placeholder to display data in your Jinja block content.

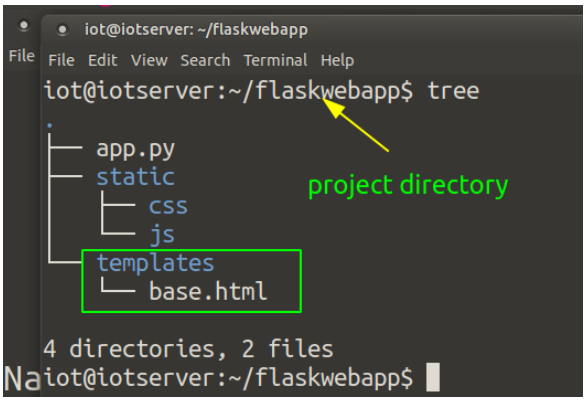
hint: create a flask route to accept dynamic queries from the user and use the value received via client request to be re-assigned to a variable to be used with the "render_template" function.

c. Change variable values in your flask code and pass it as a parameter for the "render_template" function along with "dashboard.html".

d. Reload the web app and notice the changes.

[Step#02] Create a Flask Project

1. Create a directory structure to place your flask project and it should follow below tree structure:
2. Name you project directory as you like (example "/flaskwebapp") and create the "templates" directory as shown in directory tree structure below:



3. Create the base.html in the templates directory:

4. Now edit “base.html” with your preferred text editor or an IDE software, and enter the following content:

```
<!DOCTYPE html>
<html>

<head>
   <title>Flask Template Example</title>
</head>

<body>
   {% block header %}
   <h1>Flask Template Example</h1>
   {% endblock %}

   {% block content %}
   {% endblock %}

</body>

</html>
```

5. Test you base template by editing you app.py flask application:

```
from flask import Flask, render_template
app = Flask(__name__)

@app.route('/')
def myapp():
  return render_template("base.html")

if __name__ == '__main__':
  app.run(host='0.0.0.0', debug=True)
```

Note that you must import **render_template function from the flask module.*

6. Run your flask application:

```
iot@iotserver:~/flaskwebapp$ python3 app.py
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
```

```
* Debug mode: on
* Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 252-203-194
```

7. Create a new html file called “index.html”

```
iot@iotserver:~/flaskwebapp/static$ touch index.html
```

8. Edit the file using your preferred text editor or IDE. “index.html” content:

```
{% extends "base.html" %}

  {% block header %}
  <h1>Welcome to My Flask Main Page !</h1>
  {% endblock %}

  {% block content %}
  <br>
  <h1>Hello {{visitor}} !</h1>
  {% endblock %}
```

** Notice the directive {% extends “base.html” %}, this will cause the jinja template engine to inherit “base.html” code and combine it within your index.html code.*

** Notice the place holder “{{visitor}}”, this will tell jinja to render any value that the variable visitor contains into the page at the <h1> tag location.*

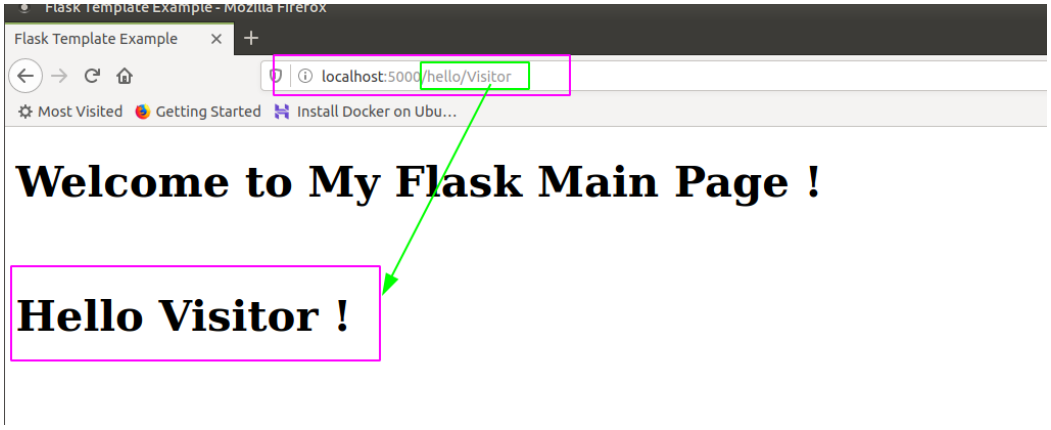
9. Edit your flask application code “app.py”:

```
from flask import Flask, render_template
app = Flask(__name__)

@app.route('/')
def myapp():
  return render_template("index.html")
```

```
@app.route('/hello/<visitor>')
def hello(visitor):
  return render_template("index.html", visitor=visitor)

if __name__ == '__main__':
  app.run(host='0.0.0.0', debug=True)
```

** Notice that the new endpoint **"/hello"** is added to flask application code and the endpoint can accept additional input query from the user and parsed the value to variable **"visitor"** to be used by python and jinja template engine.*

10. Save the file, flask will automatically reload the code.
11. Open your internet browser and point to the new endpoint "/hello/*<put your name here>*"



** Notice the changes on the page.*

** The page is now showing your "index.html" which contains the greeting "Hello {{visitor}} !", along with it is the original greeting "Welcome to My Main Page !" which is coded in the "main.html".*

** This shows that your jinja template engine has combined the main.html and index.html into one page and response to the request.*

** Notice also, the new endpoint can accept another input in the form of query parameters by extending the endpoint /hello with parameter input "/yourname".*

The input is rendered after the word "Hello" in your index.html.

12. Create a Dashboard page. In the templates directory copy your index.html then paste and rename it to "dashboard.html"

```
iot@iotserver:~/flaskwebapp/templates$ cp index.html dashboard.html
iot@iotserver:~/flaskwebapp/templates$ ls
base.html  dashboard.html  index.html
```

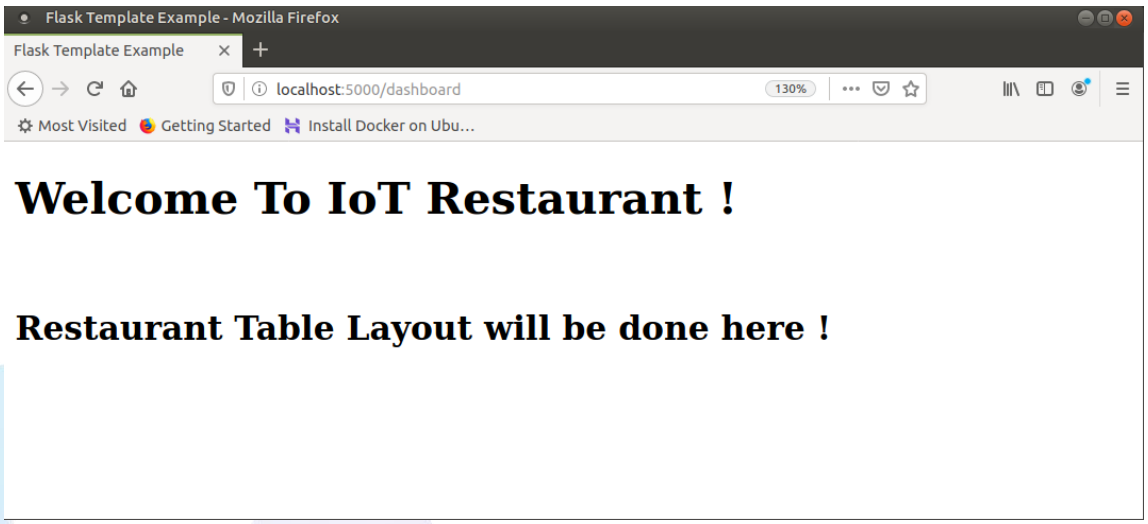
13. Then edit the dashboard.html with your preferred text editor or IDE.

```
{% extends "base.html" %}

  {% block header %}
  <h1>Welcome To IoT Restaurant !</h1>
  {% endblock %}

  {% block content %}
  <br>
    <div>
    <h2>Restaurant Table Layout will be done here ! <h2>
    </div>
  {% endblock %}
```

14. Create another endpoint to access the dashboard.html via the internet browser. Add the code snippet below to your flask application code.

```
@app.route('/dashboard')
def dashboard():
  return render_template("dashboard.html")
```

** You should get a result similar to the picture below.*



That’s it… you have learnt from this lab the jinja templating engine…! To make your page look more modern and familiar to users, you need to use “CSS” codes so that the page will be laid out correctly and responsively to various screen sizes.

Module 3: Web Application Page Styling using CSS [3hrs]

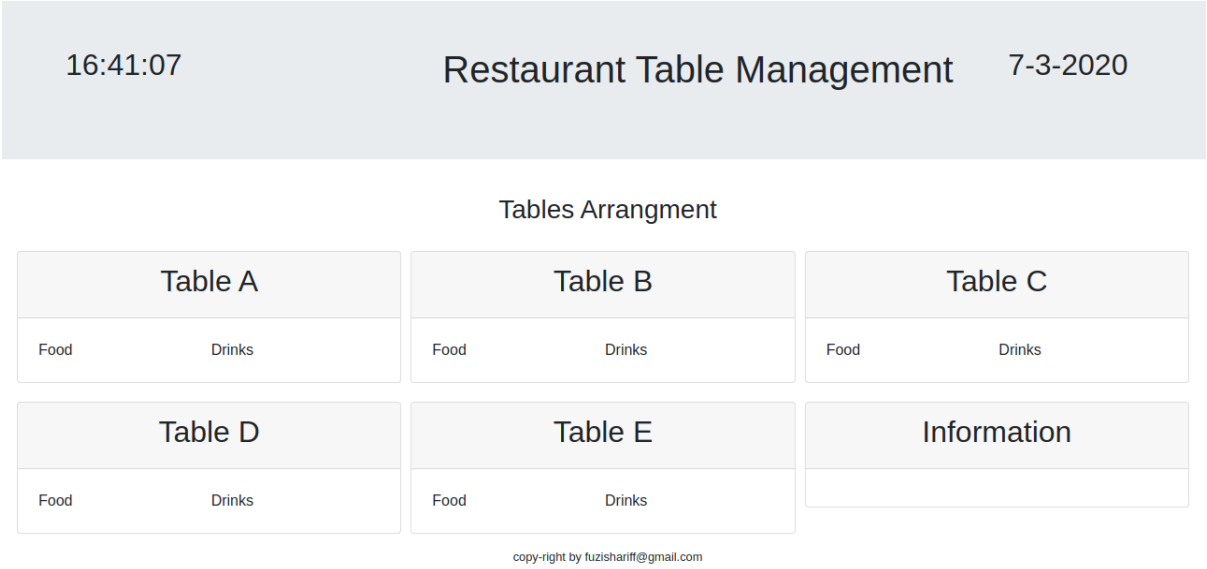
Objective: In this lab we are going to develop a dashboard for restaurant table management. CSS code must be used to style the html page in order to achieve the proper page layout that can represent a typical restaurant table layout.

Requirements:

1. Html coding knowledge.
2. Complete Lab 1
3. Complete Lab 2

Use Case:

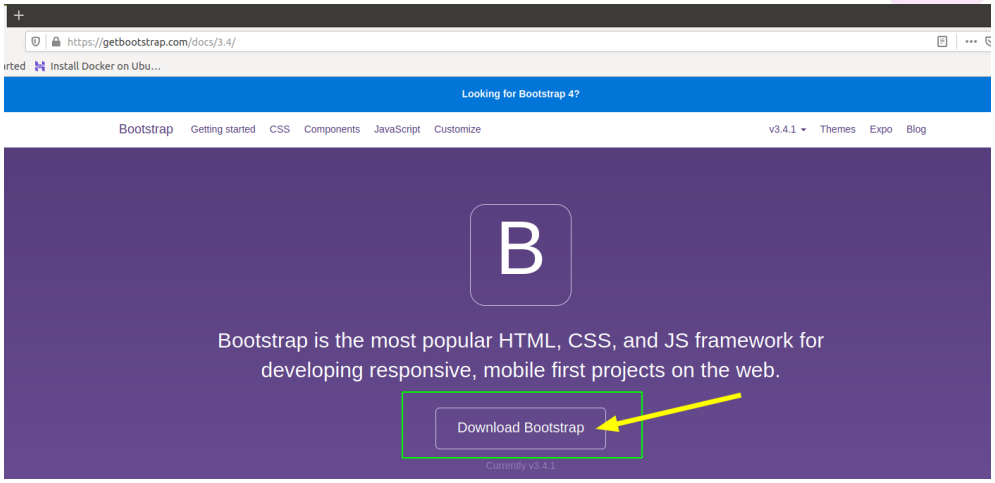
The Customer needs to display table layout on a web application dashboard that represents their restaurant table layout. The screenshot shown below is the expected appearance of the dashboard.

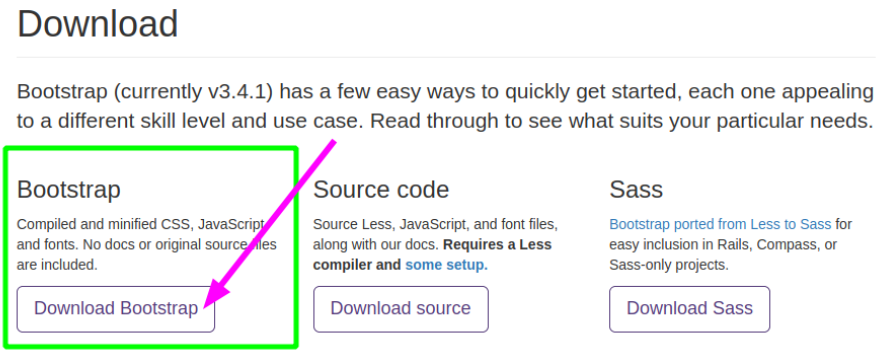


[Step#01] Integrating Bootstrap

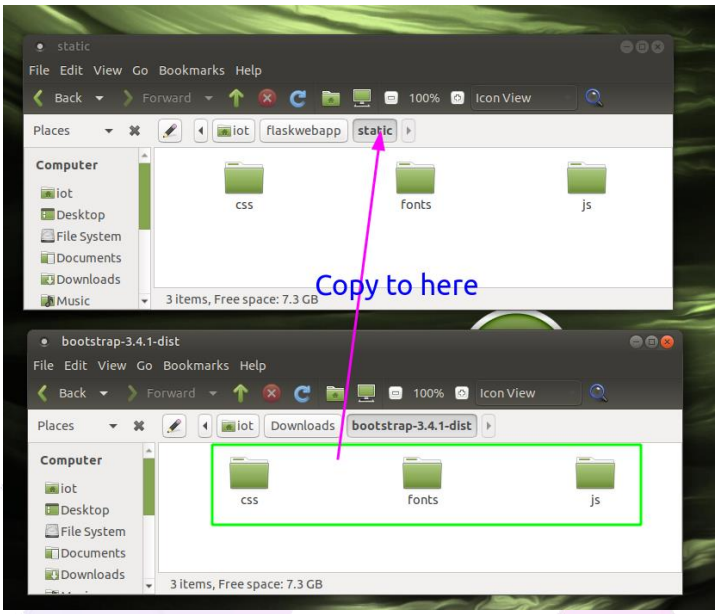
1. Copy or include Bootstrap codes into the web application, recommended in *"***base.html***"*, so that each page will load it automatically.
2. Create a html file to represent the table layout page, example **"tables.html".**
3. Structure the page content using the jinja templating engine.

4. Build page using html div elements to represent rows and columns.
5. Assign each DIV element with ID.
6. Add html element to display each table id in the corresponding table div element.
7. Use CSS styling available in Bootstrap to create responsive column layout. Make it 3 columns.
8. Go to Bostraps webpage and download the latest library.





9. Extract the file and copy the content to your static directory



10. Modify base.html to include the CSS and JS files that come with the Bootstrap library. [ref:jinja template inheritance]

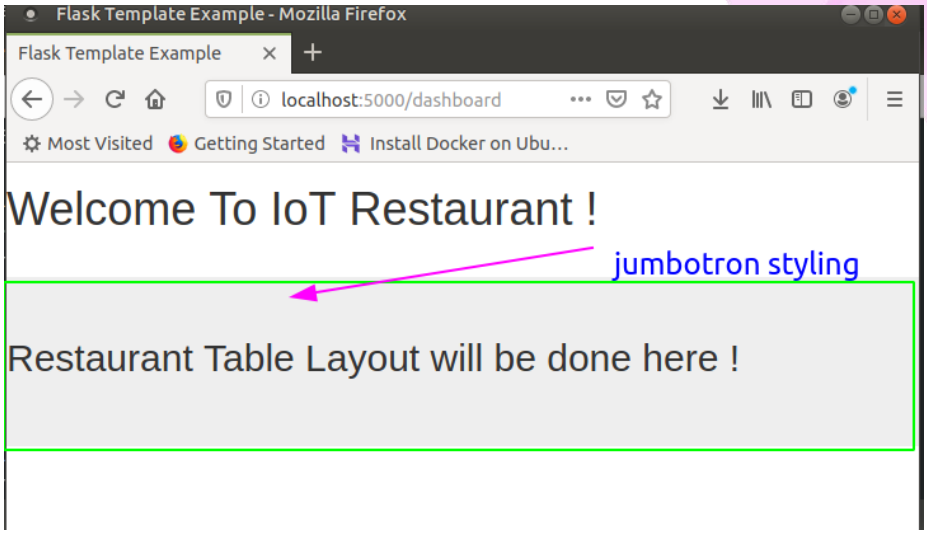
base.html:

```
<!DOCTYPE html>
<html>

<head>
   <title>Flask Template Example</title>
   <link href="{{ url_for('static', filename='css/bootstrap.min.css') }}" rel="stylesheet">
   <link href="{{ url_for('static', filename='css/main.css') }}" rel="stylesheet">
   <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>
</head>

<body>
   {% block header %}
   <h1>Flask Template Example</h1>
   {% endblock %}

   {% block content %}
   {% endblock %}

</body>

</html>
```

11. To test your Bootstrap setup is correct, modify your “dashboard.html” file to use one of the available classes, such as “jumbotron”.

```
{% extends "base.html" %}

   {% block header %}
   <h1>Welcome To IoT Restaurant !</h1>
   {% endblock %}

   {% block content %}
   <br>
   <div class="jumbotron">
      <h2>Restaurant Table Layout will be done here ! <h2>
      </div>
   {% endblock %}
```

12. Save the file and see the difference:



[Step#02] Implementing CSS Styling

Create another html file in the templates directory and name it “tables.html”.
We can make use of the class available in the bootstrap library called “grid system”.
The class ability to build tables like layout which control the size and layout of html “DIV” elements.
It consists of class “row” and “column”

In the restaurant table layout we need 2 rows and in each row there will be 3 columns.

Entire row can accommodate 12 columns with 1 unit width.
So if we want to create only 3 columns in a row, we need to span each column by 4 unit widths to occupy all 12 unit widths across a row.

So we need to use two classes:

1. row
2. col-sm-4

Step 3:
Structure “tables.html” by inheriting from “dashboard.html” and layout html code to use bootstrap grid system.

Step 4, 5, 6, 7:

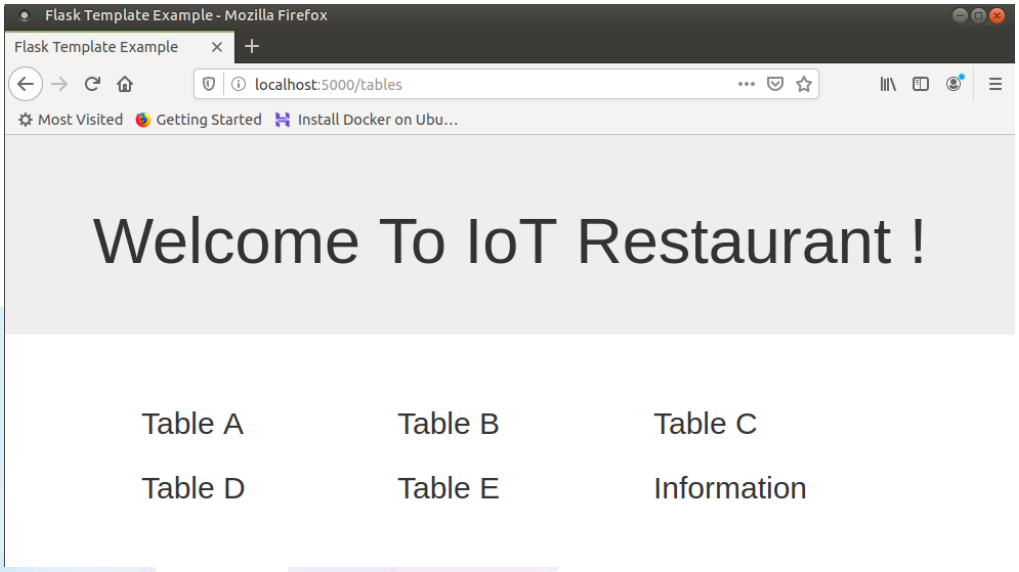
Enter the following codes and save the file.

```
{% extends "dashboard.html" %}
{% block content %}

  <br>
  <div class="container">

    <div class="row">
        <div class="col-sm-4" id="TA"><h2>Table A</h2></div>
        <div class="col-sm-4" id="TB"><h2>Table B</h2></div>
        <div class="col-sm-4" id="TC"><h2>Table C</h2></div>
    </div>

    <div class="row">
        <div class="col-sm-4" id="TD"><h2>Table D</h2></div>
        <div class="col-sm-4" id="TE"><h2>Table E</h2></div>
        <div class="col-sm-4"id="Info">
                <h2>Information</h2>
            </div>
    </div>
    </div>
{% endblock %}
```

Update app.py to add the endpoint for table.html

```
@app.route('/tables')
def tables():
  return render_template("tables.html")
```

Save app.py and check the page by using your internet browser.



Module 4: Web Application JavaScript and jQuery / Ajax [3hrs]

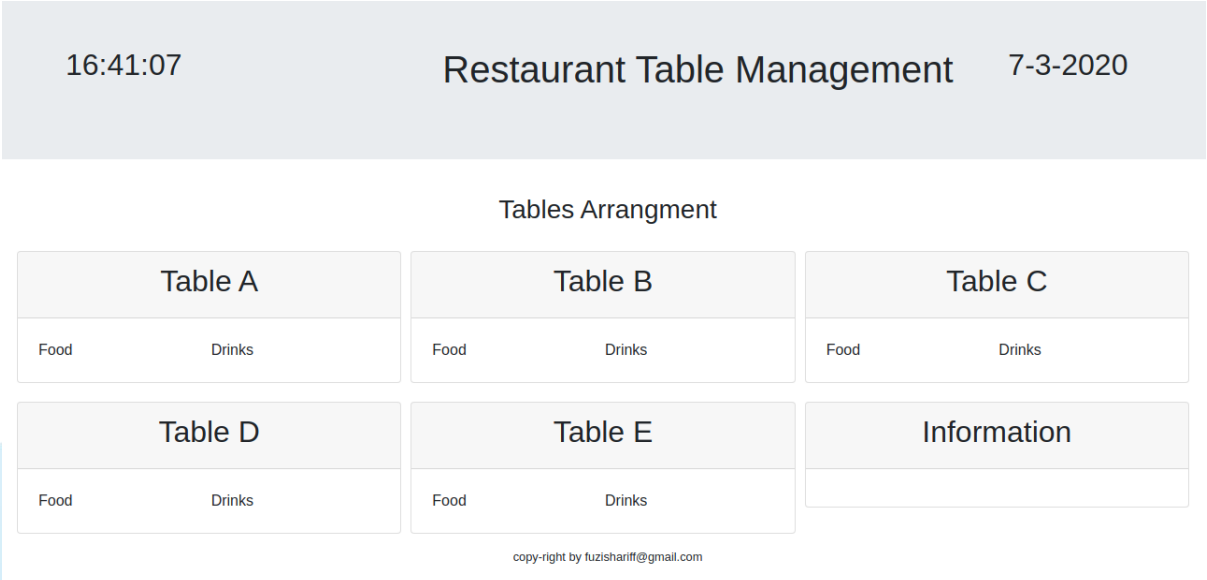
Objective: In this lab we are going to develop a dashboard for restaurant table management.

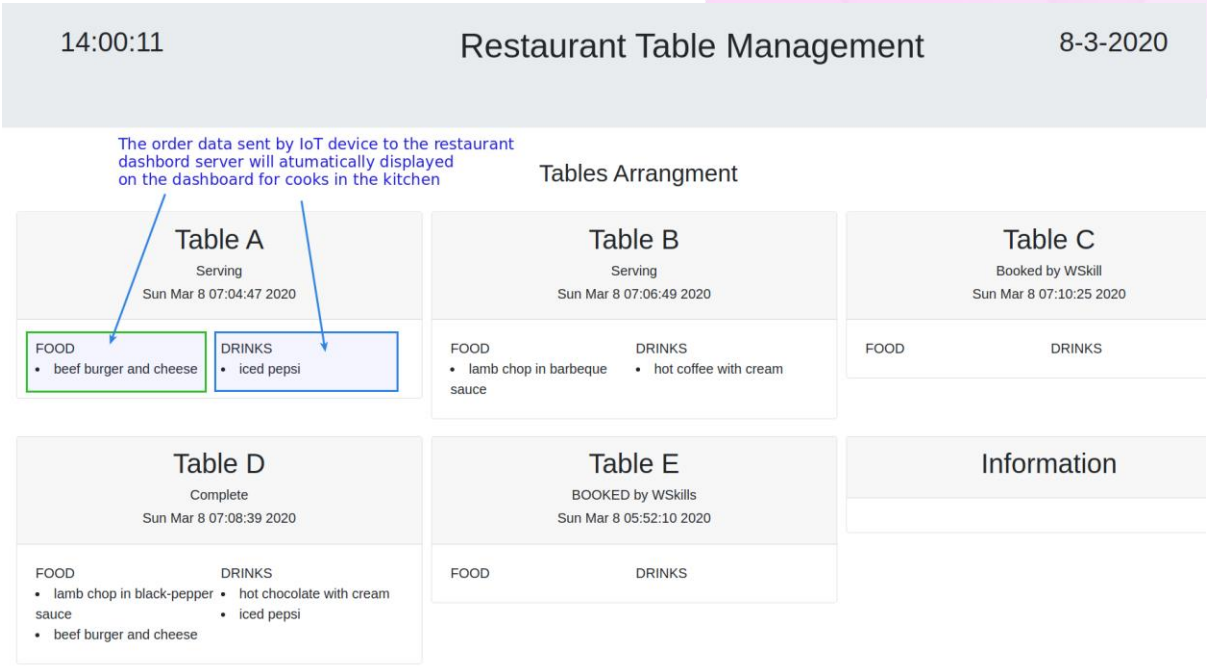
Use Case:
You are given a requirement for a restaurant to develop an ordering system using an electronic menu for its customer. At the dining table, customers will find an electronic menu controlled by a microcontroller with a menu linked to its touchpads.

Your job is to develop an embedded application such that, whenever a customer touches the selected touchpad designated with a menu set, the microcontroller automatically places the order onto the restaurant dashboard. The customer can make repeated orders by touching different touchpads designated with other sets of menus, or repeatedly touches the same touchpad to make multiple orders.

The microcontroller relies on Wi-Fi connection available in the restaurants to connect to the dashboard server that manages the order. Cooks in the kitchen will use the information to cook the meals based on the orders displayed on the dashboard.

The dashboard developer has given the endpoints details and methods to use in order for the IoT to be able to send data successfully to the back-end system. It used REST technology to accomplish this data communication process.





[Step#01] Connecting to Wi-Fi

1. The below endpoint format that we will use to access the restaurant web application API.

http://<servername_ip>:8000/status/<table_ID>

2. Here is the list of REST Endpoints.

No	End Point (URI)	methods	Data (JSON)
1	http://<servername_ip>:8000/table/<table_ID>	POST	data: {“food”:[“name food of menu”], “Drinks”:[“name of drink”] } Successful Response: { "Status": "idle", "drinks": ["hot chocolate with cream"], "food": ["lamb chop in black-pepper sauce"], "status": "Booked by WSkill", "table": "C", "time": "Sun Mar 8 07:10:25 2020" }
2	http://<servername_ip>:8000/status/<table_ID>	POST	Data: {"status":"BOOKED by WSkills" } Successful response:

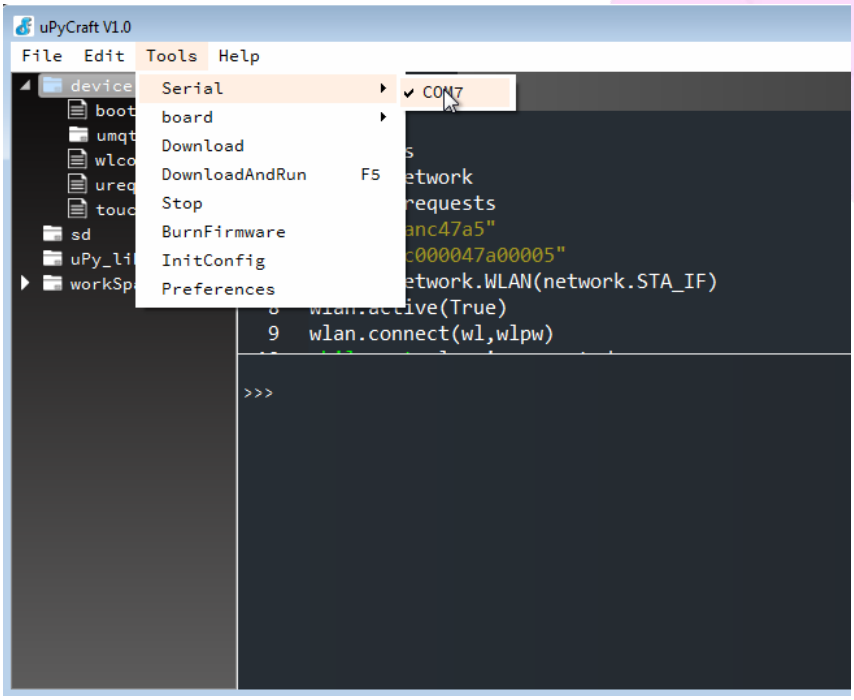
			Updated: table C, BOOKED by WSkills
3	http://<servername_ip>:8000/mainpage	GET	Successful response: Table layout HTML page

4	http://<servername_ip>:8000/tables	GET	Successful response:
			JSON data:{
			"A": {
			"drinks": [
			"iced pepsi"
			],
			"food": [
			"beef burger and cheese"
			],
			"status": "Serving",
			"table": "A",
			"time": "Sun Mar 8 07:04:47 2020"
			},
			"B": {
			"drinks": [
			"hot coffee with cream"...}

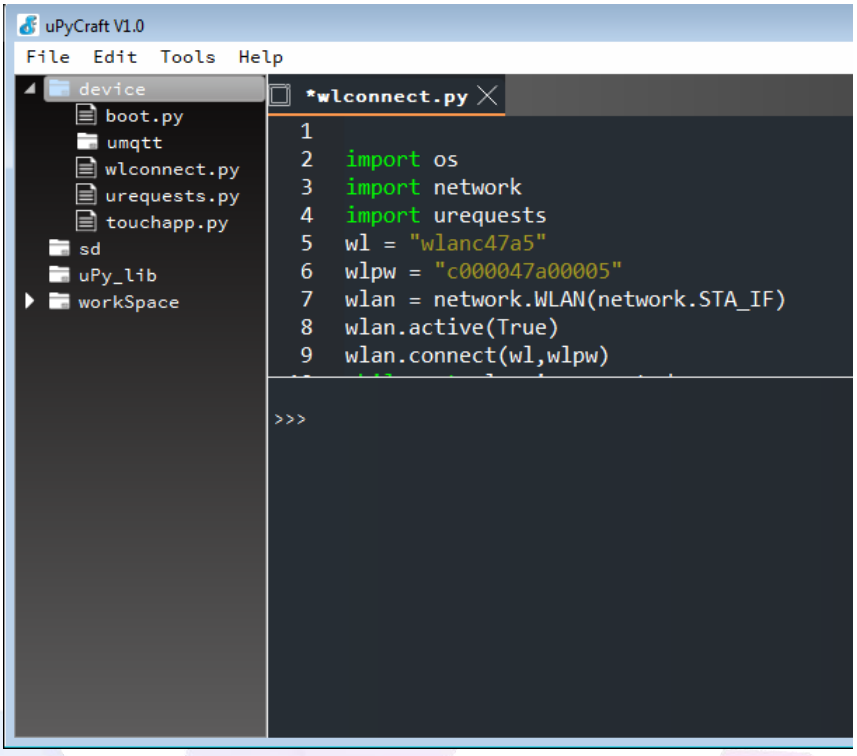
5	http://<servername_ip>:8000/menu	GET	application/JSON response with menu database.
			{
			"dnr-1": {
			"drinks": "Iced Fresh Pineapple Juice",
			"food": "Grilled Lamb Chop in Black Pepper Sauce with Baked Potato Salad",
			"price": 22.5
			},
			"dnr-2": {
			"drinks": "Iced Kasturi Lime Juice with Asam Boi",
			"food": "Spicy Fried Rice with Grilled Beef in Percik Sauce ",
			"price": 19.5
			}, …… }

3. Now let's start by programming the microcontroller to connect to WiFi and perform some http requests from the dashboard server by using urequest.py library.
4. Connect your microcontroller to the USB port and allow it to boot properly.

5. Make sure the uPyCraft IDE is connecting to the correct com port. In this case COM7

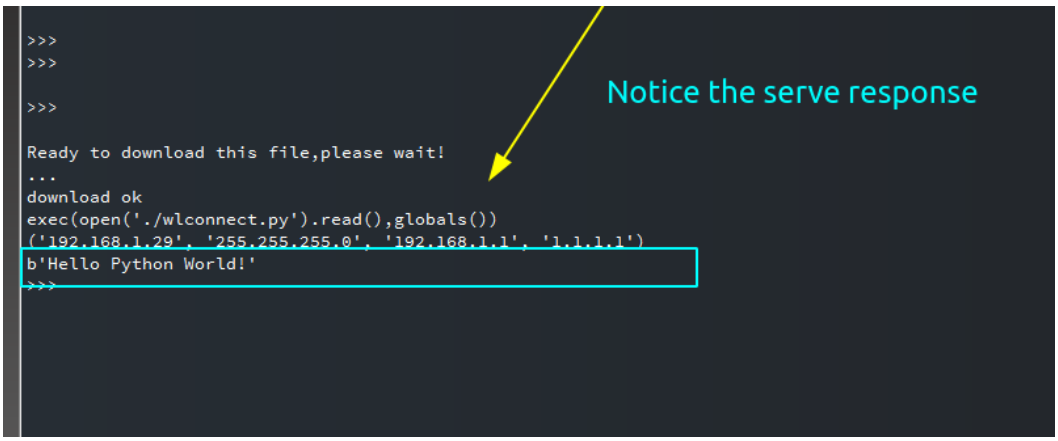


6. Click on the device, list of older programs may exist…
7. Create a new python file and give it a name wlconnect.py.
8. Ensure that you have the necessary libraries copied to the device directory as well…
9. In this case we need “urequests.py”.
10. Now start editing the “wlconnect.py”



Let test a HTTP requests to a sample web application server

```
import os
import network
import urequests
wl = "wlanc47a5"
wlpw = "c000047a00005"
wlan = network.WLAN(network.STA_IF)
wlan.active(True)
wlan.connect(wl,wlpw)
while not wlan.isconnected:
  pass
print(wlan.ifconfig())

websvr = "http://192.168.1.23:8000"

rsps = urequests.get(websvr)
print(rsps.content)
```



11. Congratulations ! You have successfully programmed an embedded application using an IoT device.

[Step#02] Implementing REST API

1. Now let us test the REST endpoint related to restaurant application….
2. Let's try to access the data from the server using HTTP / GET requests from one of the end-points. Use this endpoint: "http://<servername_ip>:8000/tables"

```
>>> uri = websvr + "tables"
>>> uri
'http://192.168.1.23:8000tables'
>>> uri = websvr + "/tables"
>>> uri
'http://192.168.1.23:8000/tables'
>>> rsps = urequests.get(uri)
>>> print(rsps.content)
b'{\n  "A": {\n    "drinks": [\n      "iced pepsi"\n    ], \n    "food": [\n      "beef burger and cheese"\n    ], \n
"status": "Serving", \n    "table": "A", \n    "time": "Sun Mar  8 07:04:47 2020"\n  }, \n  "B": {\n    "drinks": [\n
"hot coffee with cream"\n    ], \n    "food": [\n      "lamb chop in barbeque sauce"\n    ], \n    "status":
"Serving", \n    "table": "B", \n    "time": "Sun Mar  8 07:06:49 2020"\n  }, \n  "C": {\n    "Status": "idle", \n
"drinks": [\n      "hot chocolate with cream"\n    ], \n    "food": [\n      "lamb chop in black-pepper sauce"\n
], \n    "status": "BOOKED by WSkills", \n    "table": "C", \n    "time": "Sun Mar  8 14:17:37 2020"\n  }, \n  "D":
{\n    "drinks": [\n      "hot chocolate with cream", \n      "iced pepsi"\n    ], \n    "food": [\n      "lamb chop in
black-pepper sauce", \n      "beef burger and cheese"\n    ], \n    "status": "Complete", \n    "table": "D", \n
"time": "Sun Mar  8 07:08:39 2020"\n  }, \n  "E": {\n    "drinks": [\n      "iced pepsi"\n    ], \n    "food": [\n
"beef burger and cheese"\n    ], \n    "status": "BOOKED by WSkills", \n    "table": "E", \n    "time": "Sun Mar
8 05:52:10 2020"\n  }, \n  "information": ""\n}\n'
>>>
```

3. From the example above, we use micropython to build the endpoint based on its existing variable values in memory…

   uri = websvr + "tables"

   Here uri will be our endpoint variable, we concatenate the value server address in variable websvr and test the value:

   http://192.168.1.23:8000tables

   This URI value seems to be correct, but there is a slight mistake. It is missing one "/" before the parameter "tables"

   It is easily fixed in the code follows :

   uri = websvr + "/tables"

   The code below re-execute REST request to the server:

   rsps = urequests.get(uri)

4. After execution, REPL prompted with no error, this means that the REST request has been successful.

The "rsps" variable is the variable corresponding to the request object, now it holds the data being sent by the server response.

```
print(rsps.content)
```

This command prints the content or data available in rsps object as shown below:

```
b'{\n  "A": {\n    "drinks": [\n      "iced pepsi"\n    ], \n    "food": [\n      "beef burger and cheese"\n    ], \n
"status": "Serving", \n    "table": "A", \n    "time": "Sun Mar  8 07:04:47 2020"\n  }, \n  "B": {\n    "drinks": [\n
"hot coffee with cream"\n    ], \n    "food": [\n      "lamb chop in barbeque sauce"\n    ], \n    "status":
"Serving", \n    "table": "B", \n    "time": "Sun Mar  8 07:06:49 2020"\n  }, \n  "C": {\n    "Status": "idle", \n
"drinks": [\n      "hot chocolate with cream"\n    ], ...
>>>
```

5. The data is a large string object that can be processed as a JSON object.
   Now let's try to post some data according to REST API endpoint and data structure format.

   Let say, the customer wants to order as set for:

   Food: "Fish and Chips with Salads"
   Drink: "hot coffee latte"

   The the JSON data format show look like as follows:

```
{
"food":["fish and chips with salads"],
"drinks":["hot coffee latte"]
}
```

6. Now we need to create a python object to hold this data.

```
menu = {"food":["fish and chips with salads"],
"drinks":["hot coffee latte"]}
```

7. To achieve this we can use python built in function "dict", the function that builds a dictionary object which has a similar structure as JSON data.

```
>>> menu = dict(food = ["fish and chips with salads"], drinks = ["hot coffee latte"])
>>>
>>> menu
{'drinks': ['hot coffee latte'], 'food': ['fish and chips with salads']}
>>>
```

8. It looks like “food” and “drinks” switch positions, but luckily it doesn’t matter to python.

   Now the data is ready to be posted to the server. What we need now is the correct server address and end-point that will be able to process the data..

   Checking the REST end-points of the web application we the end point is:
   http://<servername_ip>:8000/table/<table_ID>

9. So let's build the URI for this end-point, and assign it to a variable.
   We choose table A, since the Table is available...


```
>>> menu = dict(food = ["fish and chips with salads"], drinks = ["hot coffee latte"])
>>>
>>> menu
{'drinks': ['hot coffee latte'], 'food': ['fish and chips with salads']}
>>> uri
'http://192.168.1.23:8000/tables'
>>> websvr
'http://192.168.1.23:8000'
>>> uri = websvr + "/table" + "/A"
>>> uri
'http://192.168.1.23:8000/table/A'
>>>
```

10. With the python code shown above , now the “uri” object holds the correct end-point value that we need.

    'http://192.168.1.23:8000/table/A'

11. Next we will post the menu data to the endpoint using micro python “urequests” REST client library.
    Another piece of information is required to inform the web application server that we want the data to be processed as a JSON application. Or else, it assumes XML/HTML which is designed for human visualization.

So the information is fed to header parameter of urequests object like shown below:

```
headers = {'content-type': 'application/json'}
```

12. Now the complete instruction to use for micropython will be as follows:

```
>>> rps = urequests.post(uri, json=menu, headers = headers)
```

13. Lets run the command.... You should receive confirmation from the web application server the response like shown in the screenshot below.

```
uri = websvr + "/table" + "/A"
>>> uri
'http://192.168.1.23:8000/table/A'
>>> menu
{'drinks': ['hot coffee latte'], 'food': ['fish and chips with salads']}
>>> headers
{'content-type': 'application/json'}
>>> rps = urequests.post(uri, json=menu, headers=headers)
>>> rps.content
b'{\n  "drinks": [\n    "hot coffee latte"\n  ], \n  "food": [\n    "fish and chips with salads"\n  ], \n  "status": "idle", \n  "table": "A", \n  "time":
""\n}\n'
>>>
```

14. Response from successful POST requests to end-point for ordering a dinner set of the restaurant web application. We can now check how it appears on the dashboard.

15. We also need to send another piece of JSON data to set the table status as being served…

    The endpoint should be:

```
http://<servername_ip>:8000/status/<table_ID>
```

16. Let's set the status as "Serving now..."

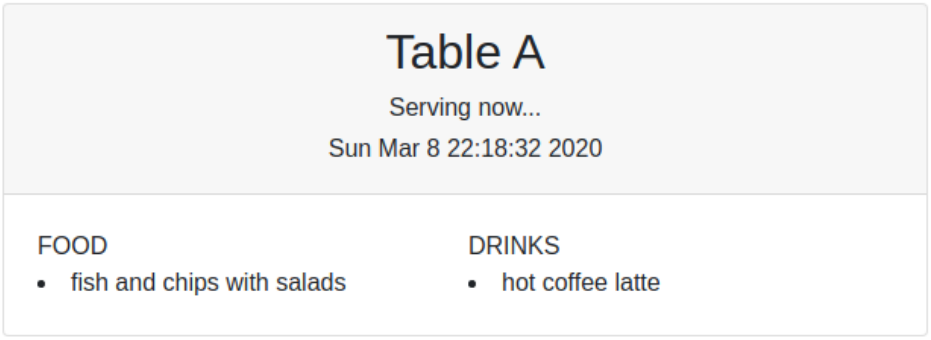
    Our JSON data should be:

```
Data:
{
"status":"Serving now..."
```

```
}
```

```
uri = websvr + "/status" + "/A"
>>> uri
'http://192.168.1.23:8000/status/A'
>>> tblsts = dict(status = "Serving now..."
... )
>>> tblsts
{'status': 'Serving now...'}
>>> rps = urequests.post(uri, json = tblsts , headers=headers)
>>> rps.content
b'Updated: table A, Serving now...'
>>>
```

17. So now let's take a look on the dashboard again.



As you can see, the information about the menu being ordered and the status of the table being served are displayed correctly for human visualization.

18. Congratulations ! Your IoT is capable of using REST data communication protocol to work as required by the web application API.

19. Finally you need to put these commands or codes into a python function or module so that it can be called by other python functions such as a function that runs when touchpad is activated.

    Alright… now lets us put all the pieces of code together to perform as one embedded application in an IoT device.

20. Let's begin with the menu codes:
    You can be creative in this task, just imagine your favorite food…
    Here are four suggested menus that water your mouth…!

SET ID	Servings	Category
snk-1	food: Pineapple Shrimp Sandwich with Cheese and Pepper drinks: Hot black coffee	Snack - I'm not hungry

lch-1	food:BBQ Chicken Maryland and Steamed Butter Rice and Curry Gravy drinks: Iced lemon tea	Lunch - I'm hungry
dnr-1	food:Grilled Lamb Chop in Black Pepper Sauce with Baked Potato Salad drinks:Iced Fresh Pineapple Juice	Dinner - I'm very hungry
dnr-2	food:Spicy Fried Rice with Grilled Beef in Percik Sauce drinks:Iced Kasturi Lime Juice with Asam Boi	Dinner - I'm very hungry

21. We must assume that the developer of the web application will be able to provide us with this data via its API interface and the way to access to the resources by suing the endpoint:

    http://<servername_ip>:8000/menu

    We can test the API service from IoT itself…

    This is how we do it…

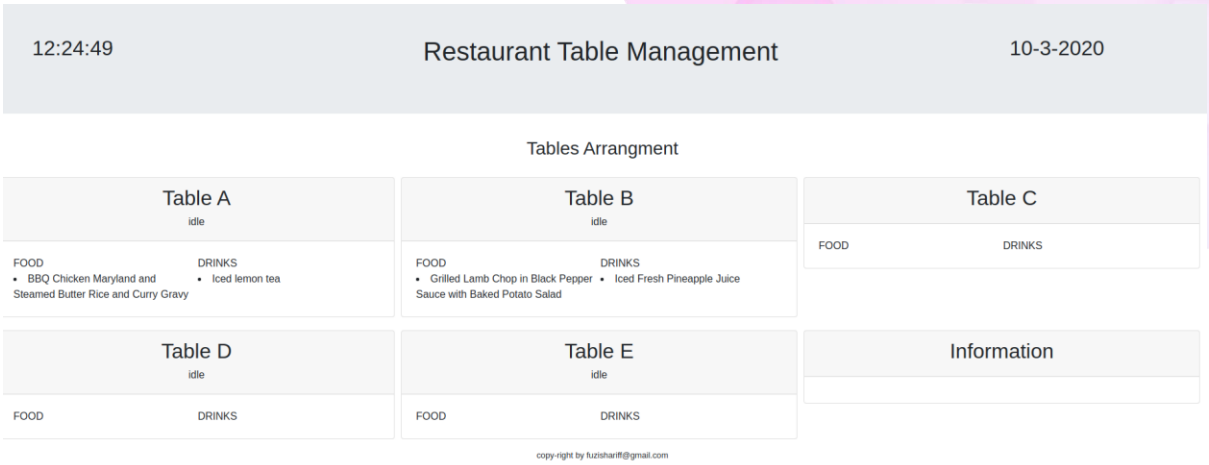
```
>>> websvr
'http://192.168.1.23:8000'
>>> rps = urequests.get(websvr + "/menu")
>>> rps.content
b'{\n  "dnr-1": {\n    "drinks": [\n      "Iced Fresh Pineapple Juice"\n    ], \n    "food": [\n      "Grilled Lamb Chop in Black Pepper Sauce with Baked Potato Salad"\n    ], \n    "price": 22.5\n  }, \n  "dnr-2": {\n    "drinks": [\n      "Iced Kasturi Lime Juice with Asam Boi"\n    ], \n    "food": [\n      "Spicy Fried Rice with Grilled Beef in Percik Sauce "\n    ], \n    "price": 19.5\n  }, \n  "lch-1": {\n    "drinks": [\n      "Iced lemon tea"\n    ], \n    "food": [\n      "BBQ Chicken Maryland and Steamed Butter Rice and Curry Gravy"\n    ], \n    "price": 17.5\n  }, \n  "snk-1": {\n    "drinks": [\n      "Hot black coffee"\n    ], \n    "food": [\n      "Pineapple Shrimp Sandwich with Cheese and Pepper"\n    ], \n    "price": 9.5\n  }\n}\n'
>>> menus=json.loads(rps.content)
>>> menus
{'snk-1': {'drinks': ['Hot black coffee'], 'price': 9.5, 'food': ['Pineapple Shrimp Sandwich with Cheese and Pepper']}, 'dnr-1': {'drinks': ['Iced Fresh Pineapple Juice'], 'price': 22.5, 'food': ['Grilled Lamb Chop in Black Pepper Sauce with Baked Potato Salad']}, 'dnr-2': {'drinks': ['Iced Kasturi Lime Juice with Asam Boi'], 'price': 19.5, 'food': ['Spicy Fried Rice with Grilled Beef in Percik Sauce ']}, 'lch-1': {'drinks': ['Iced lemon tea'], 'price': 17.5, 'food': ['BBQ Chicken Maryland and Steamed Butter Rice and Curry Gravy']}}
>>>
>>> menus.keys()
dict_keys(['snk-1', 'dnr-1', 'dnr-2', 'lch-1'])
>>> menus['lch-1']
{'drinks': ['Iced lemon tea'], 'price': 17.5, 'food': ['BBQ Chicken Maryland and Steamed Butter Rice and Curry Gravy']}
```

22. Now we can see that we can access each menu information available on the web application server via REST API using an IoT microcontroller. This means that this information can be used to place orders on restaurant applications by using python codes that will respond to customer input such as a touchpad.

    Here is the combined code into one IoT touchpad order application utilizing two touchpad sensors as inputs…

```
#This application is for embedded program to utilize touchpad sensors for restaurant ordering system
#author fuzi shariff
#email: fuzishariff@gmail.com

import wlconnect   #your separate module to connect to WiFi
import urequests
from machine import TouchPad, Pin
from utime import sleep
led = Pin(15,Pin.OUT)
tch = TouchPad(Pin(4))
tch.config(2000)
tch2 = TouchPad(Pin(13))
tch2.config(2000)
import json

websvr ="http://192.168.1.23:8000"
headers = {"content-type":"application/json"}

def getMenu():
   rps = urequests.get(websvr+"/menu")
   menus = json.loads(rps.content)
   rps.close()
   return(menus)

def orderMenu(tableID, menuKey):
  menus = getMenu()
  rps = urequests.post(websvr + "/table/" + tableID, json=menus[menuKey], headers=headers)
  print(rps.content)
  rps.close()

while True:
  d = tch.read()
  if d < 490:
    led.on()
    print("You touched ESP32 sensor !")
    menulis = getMenu()
    orderMenu("A",'lch-1')
    sleep(1)
  led.off()
  sleep(0.2)

  d = tch2.read()
  if d < 490:
    led.on()
    print("You touched ESP32 sensor !")
    menulis = getMenu()
    orderMenu("B",'dnr-1')
    sleep(1)
  led.off()
  sleep(0.2)
```

24. The dashboard application displayed the orders as shown below:



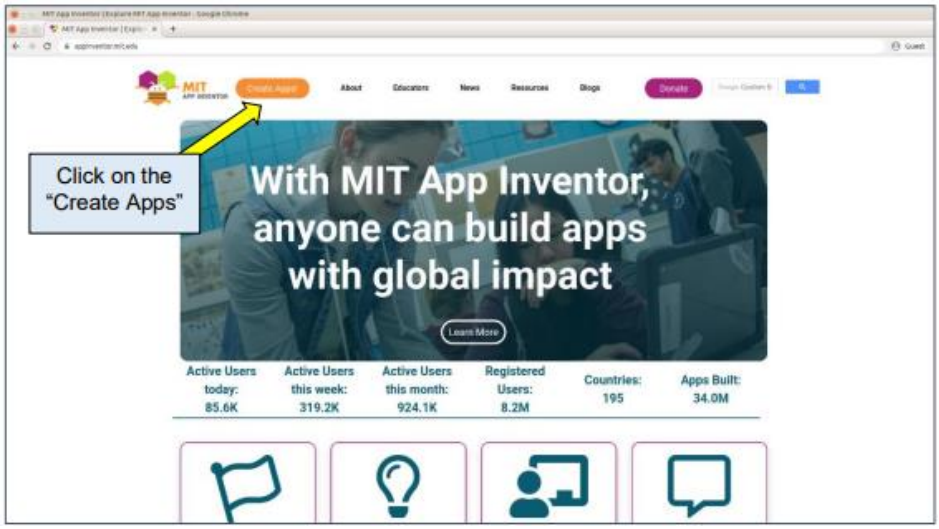
Topic 5: Mobile Apps Development for IoT

Module 1: Getting Started with MIT App Inventor [2hrs]

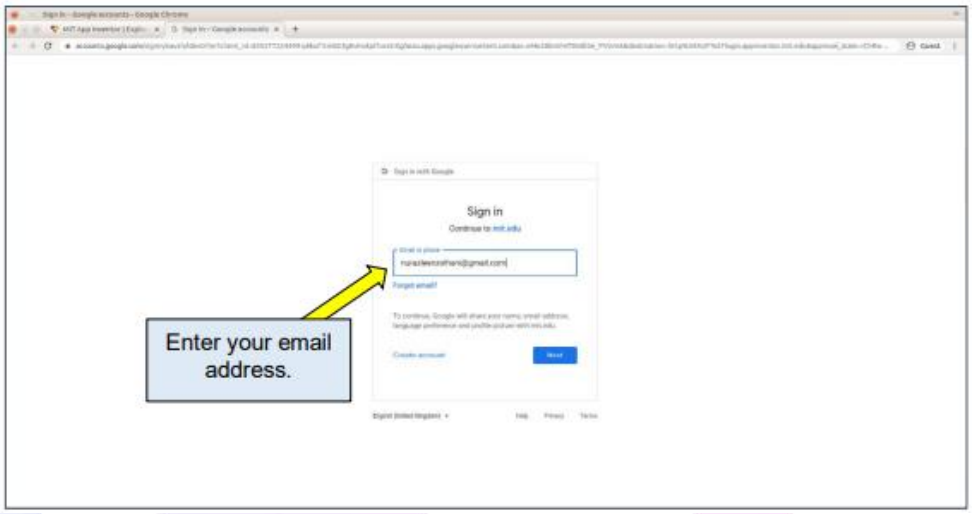
Objective: In this lab we are going to walk through the MIT App Inventor and getting familiar with the layout, menus and panels available at the web application. MIT App Inventor is a web based application that allows user to create Android apps.

[Step#01] Create Your 1st App

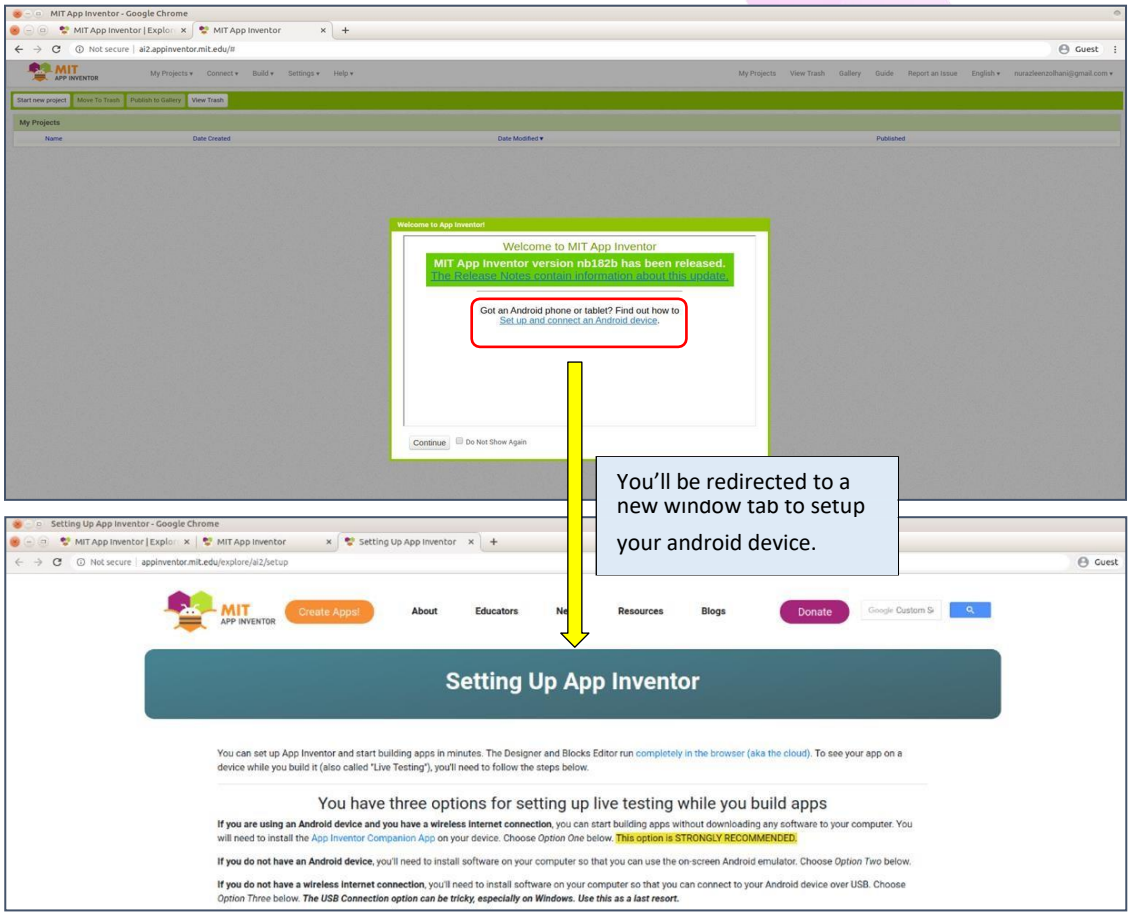
1. Launch your preferred web browser and typed in the following URL to access the MIT App Inventor https://appinventor.mit.edu/
2. At the homepage, click on the **Create Apps!** button.



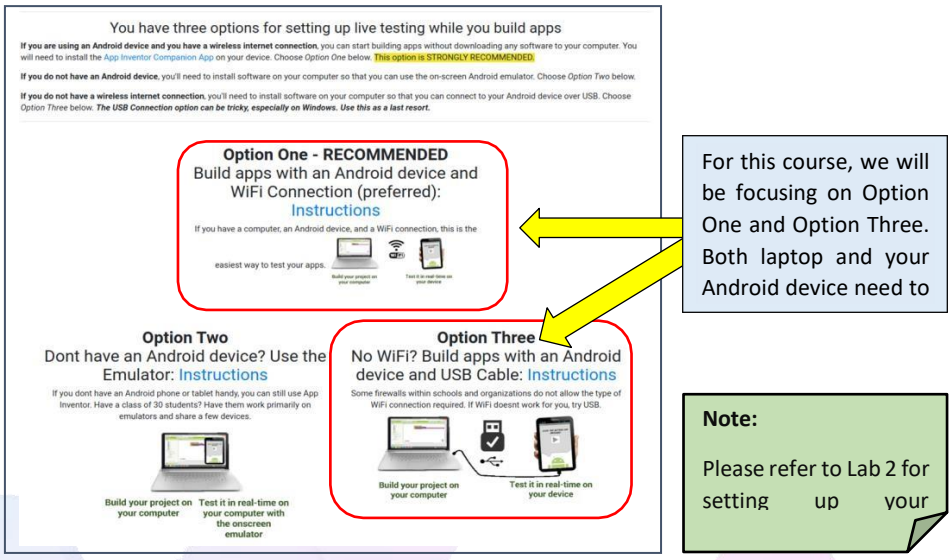
3. Then, you will be redirected to Sign-in to your Google Account. Key-in your email address and make sure you have the access to your email.



4. After signing-in to your Google Account, you will be redirect to the workspace in building your Android app. Here, you will be prompt with a pop-up windows asking you whether you want to setup your android device.

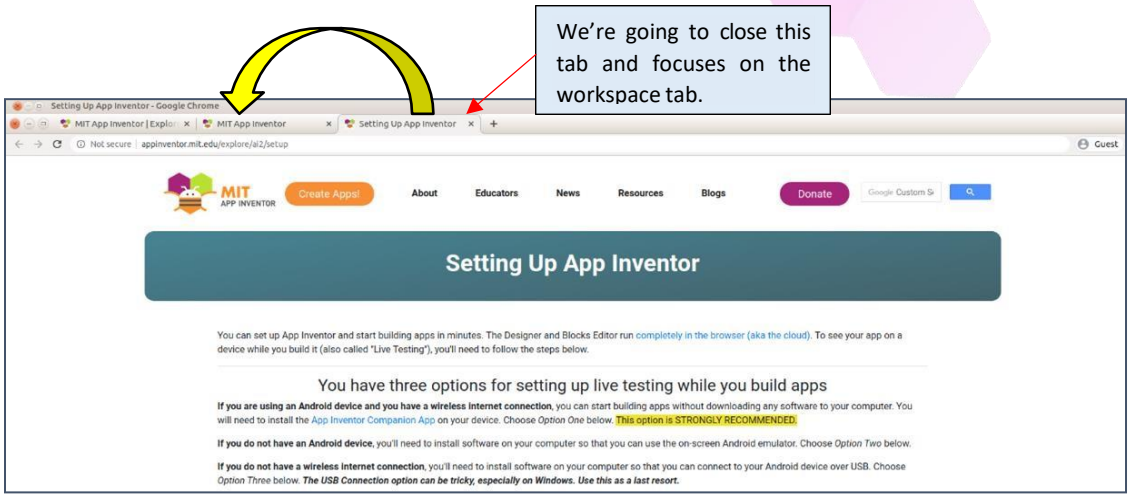


5. There are 3 options available to connect or setup your android device. If you have an Android device, you can opt for Option One or Option Three. If you don't have an

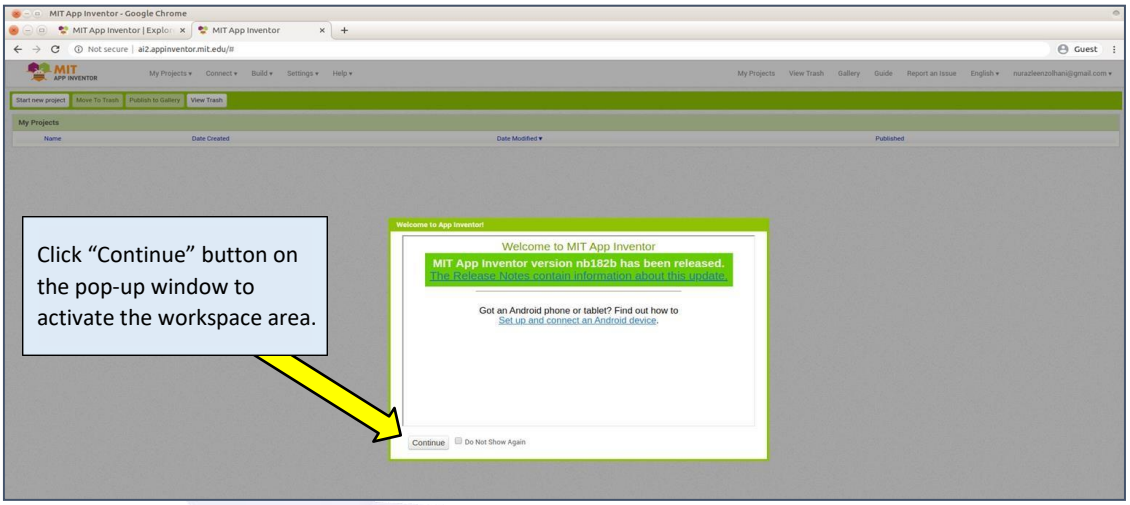


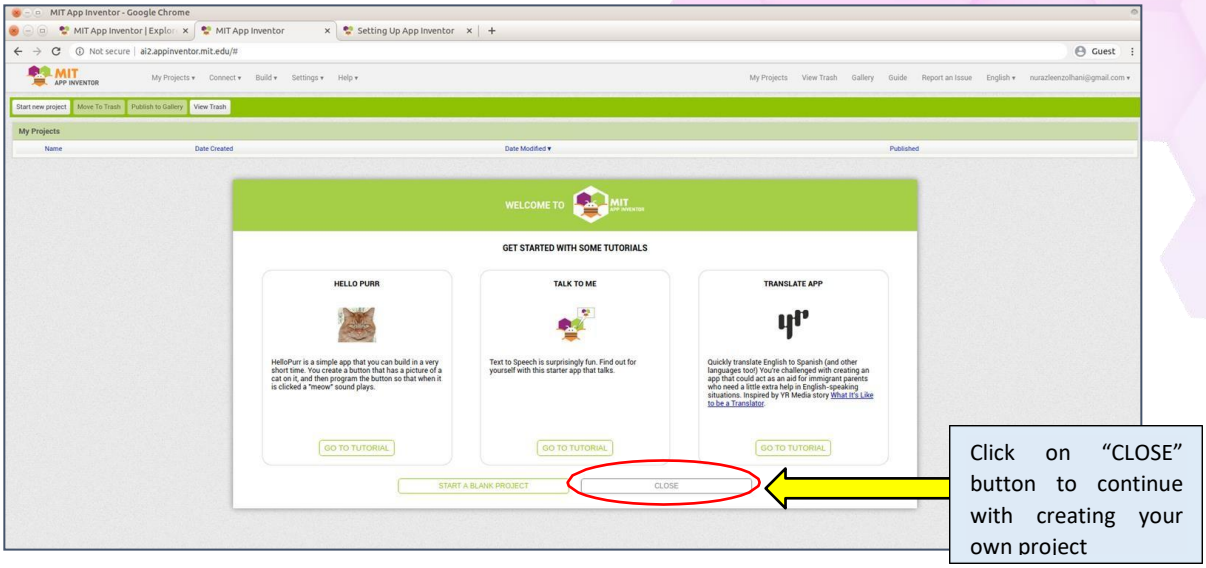
Android device, you can opt for Option Two which uses an Android Emulator.

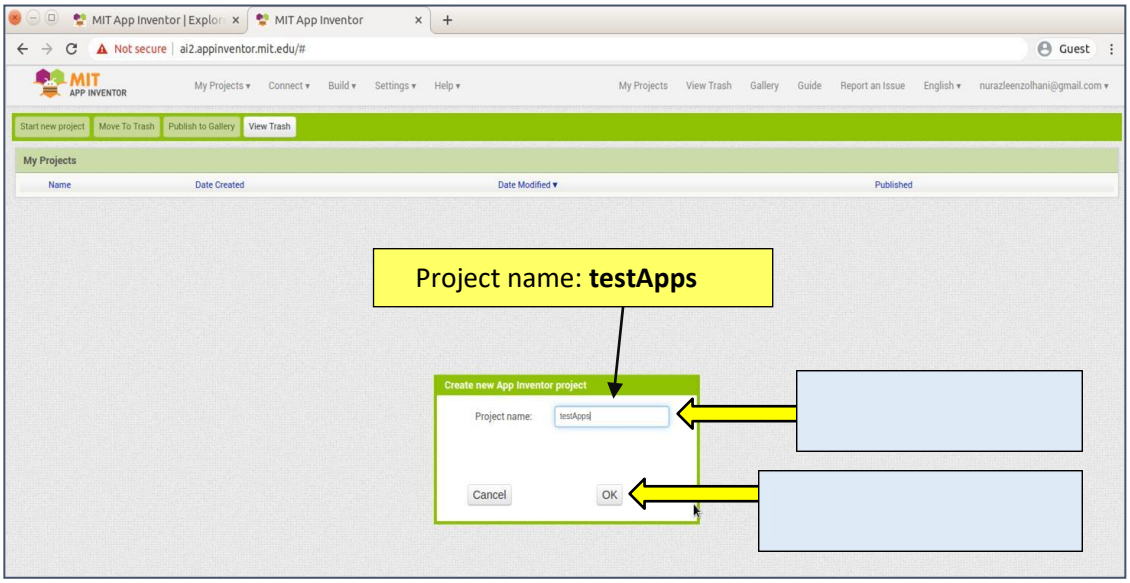
6. After you have setup your Android device, you can close the current tab and return back to the workspace tab as shown in the figure below.

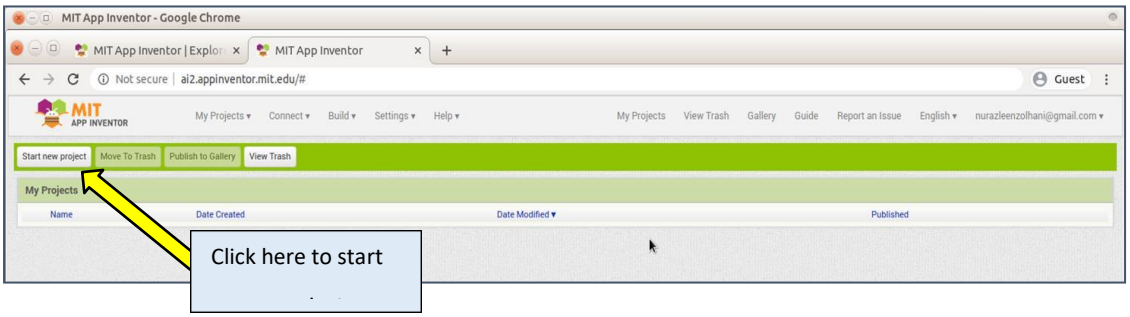


7. Once you've clicked on the "Continue", you will be presented with a welcome pop-up window. You can start building Android apps based on these tutorials or you can explore the MIT App Inventor on your own. In this lab, we're going to create our own project and familiarize with the layout, menus and panels in MIT App Inventor.
8. Then click on the **"Start new project"** button and you will be prompt with a pop-up window to key-in the name of the project.

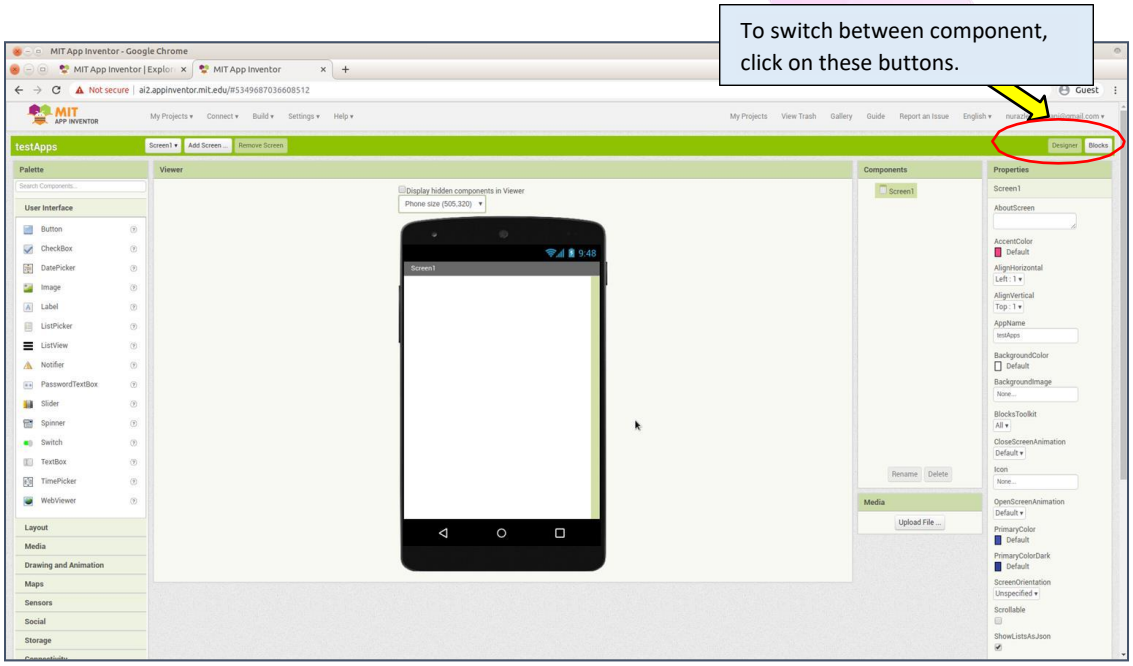



WELCOME TO
GET STARTED WITH SOME TUTORIALS
HELLO PURR
TALK TO ME
TRANSLATE APP
START A BLANK PROJECT
CLOSE
Click on "CLOSE" button to continue with creating your own project

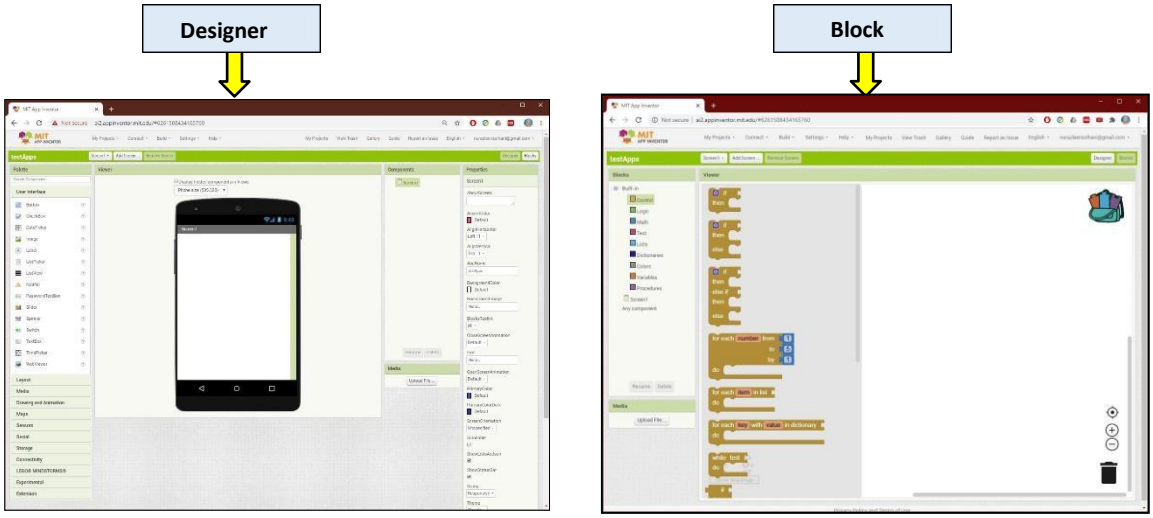

Project name: testApps
Create new App Inventor project
Project name:
testApps
Cancel
OK


Click here to start

9. Then, you will be redirected to the MIT App Inventor design layout as shown in figure below. By default, you'll be presented with the Designer component. There are two types of component, which are **Designer** and **Block**. You can switch between components by clicking on the button at the top right side.

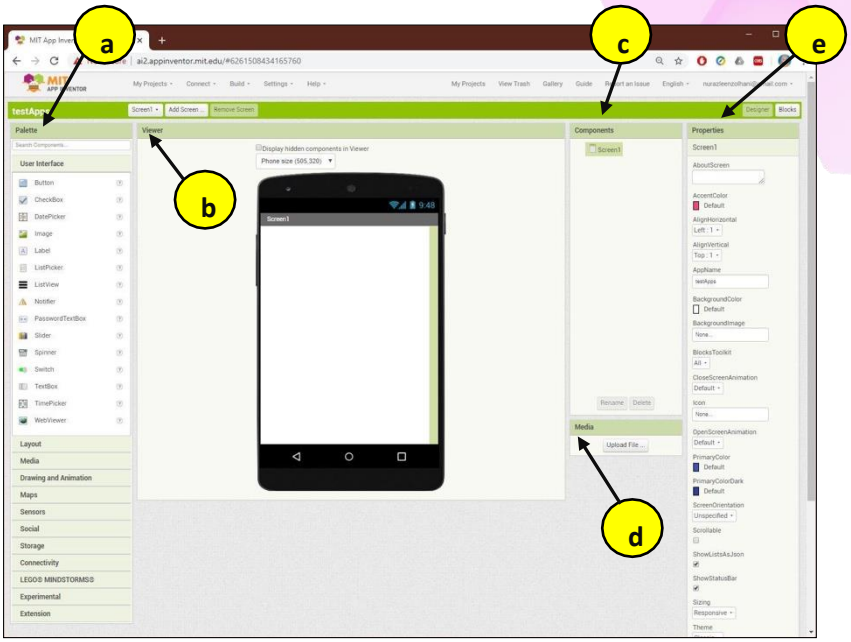


10. The MIT App Inventor Designer lets you design your apps by using the drag and drop method. Meanwhile, the MIT App Inventor Blocks lets you code your program by arranging the blocks of code.

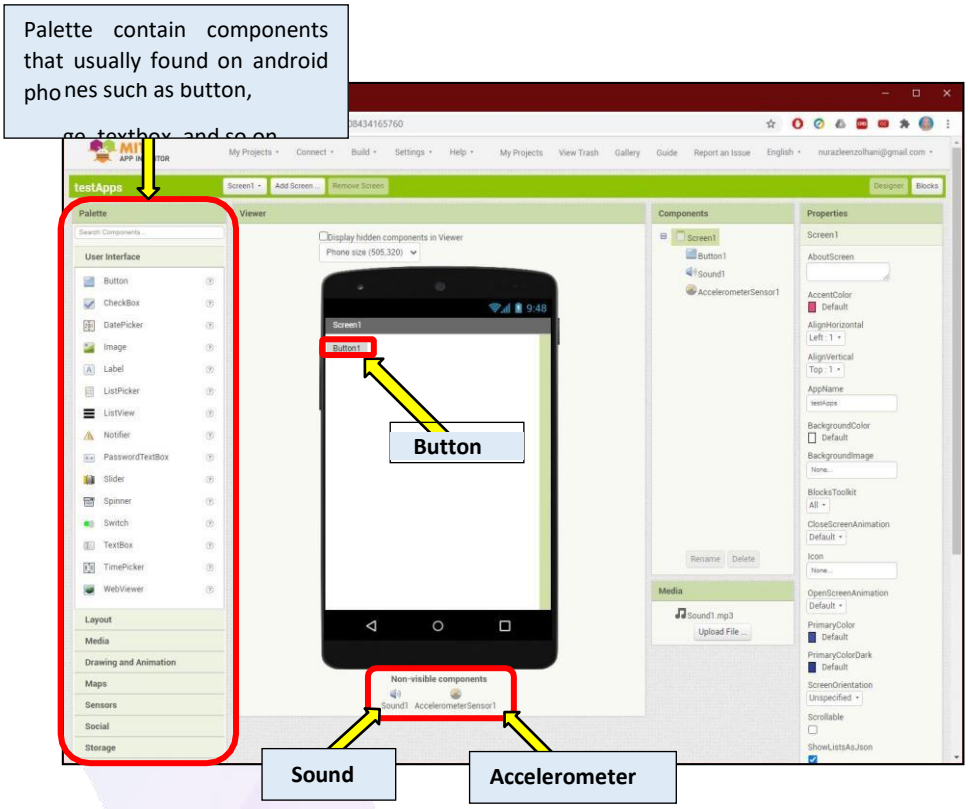


11. There are five (5) windows in the Designer of MIT App Inventor Designer.
    a. **Palette:** holds the components you can use in your program; separated into sub lists
    b. **Viewer:** shows components mapped out to what the app will look like
    c. **Components List:** lists components in the app
    d. **Media:** Allows developer to upload audio and pictures.

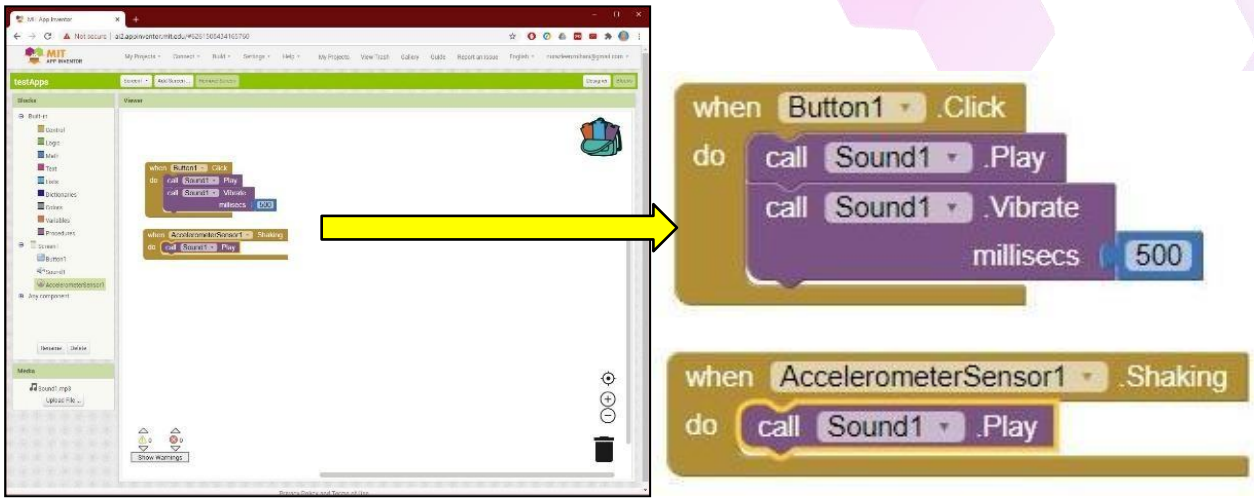
e. **Properties:** Showing the selected component.



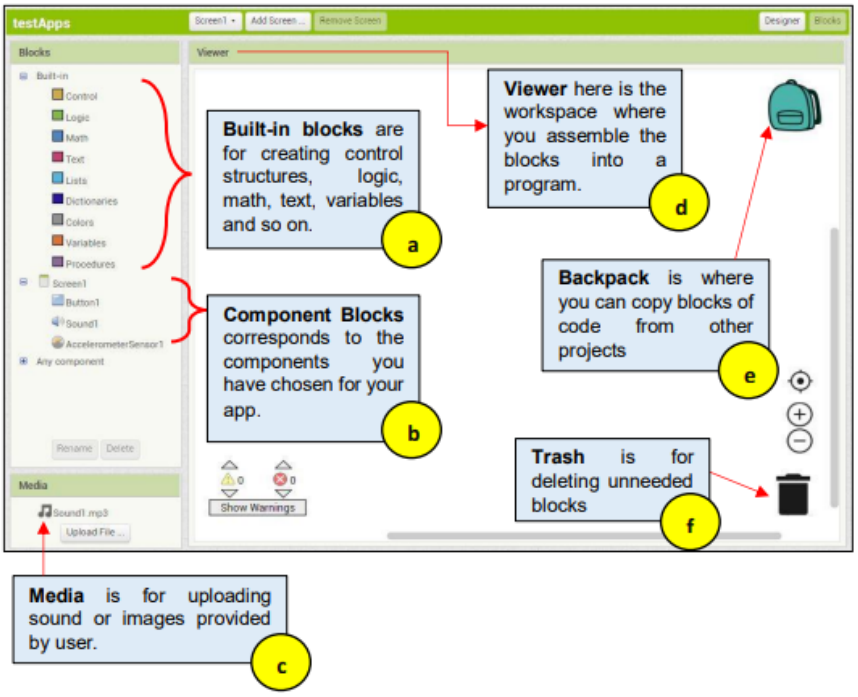
12. Figure below shows the screen design with a **Button**, a **Sound** and a **Sensor** which is the Accelerometer. All these components can be find at the **palette** on the left side of the MIT App Inventor Designer window.



13. Then, to specify how the components works, you can easily code them at the MIT App Inventor Block window as show in the figure below. The figure shows blocks of code showing the action of the Button and the Accelerometer sensor on the phone.



14. In the MIT App Inventor Block, there are 6 components that are important for us to remember which are
    a. Built-in blocks
    b. Component Blocks
    c. Media
    d. Viewer
    e. Backpack
    f. Trash



References:

1. http://appinventor.mit.edu/explore/sites/all/files/teachingappcreation/unit1/MagicTrickHa ndout.pdf
2. https://appinventor.mit.edu/explore/library
3. https://appinventor.mit.edu/explore/ai2/tutorials
4. https://www.programwithappinventor.org/
5. https://www.amazon.com/Learning-MIT-App-Inventor-Hands-On/dp/0133798631/

Module 2: Setting Up Connection for MIT App Inventor [1hr]

Objective: In this lab we are going to go through the steps needed in setting up the connection from the MIT App Inventor 2 to our Android device. There are three (3) options to setup your connection which are via **WiFi**, via **USB** cable, and lastly via an Android **Emulator**. For this lab we're going to focus on connecting using the AI Companion app that you can download from Google Play Store and connecting it via WiFi and USB cable to the MIT App Inventor 2.

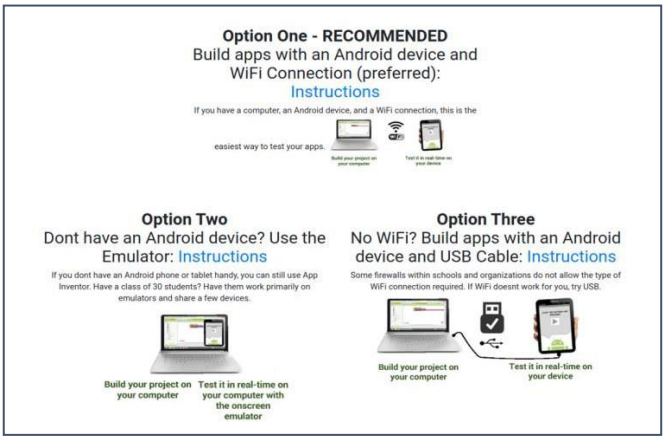
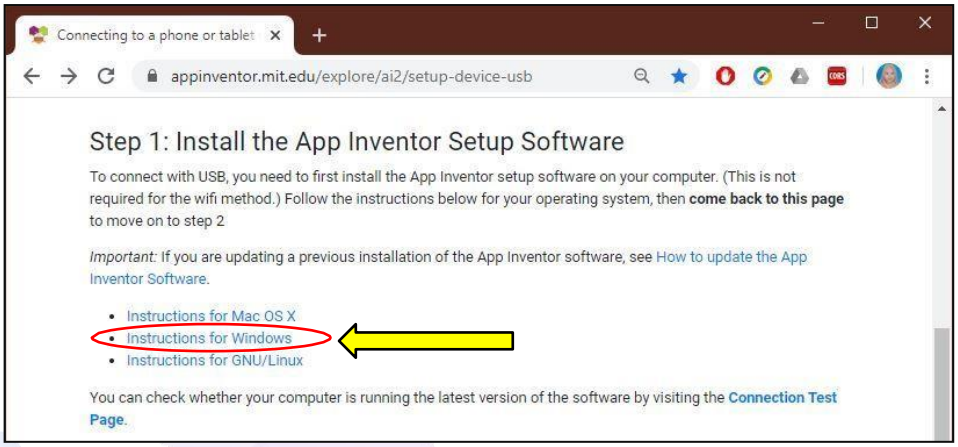


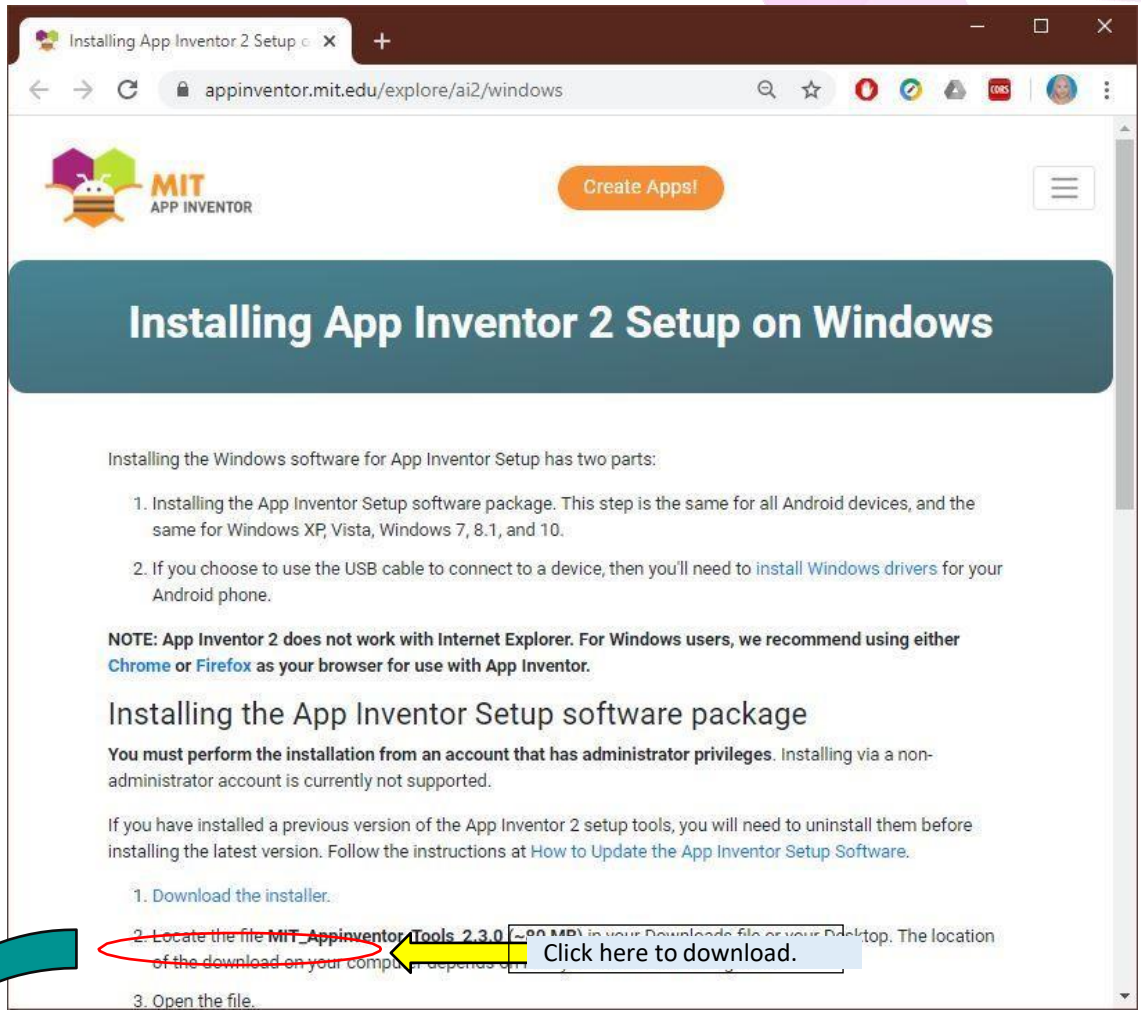
Figure 1: Connections between MIT App Inventor and your Android Device

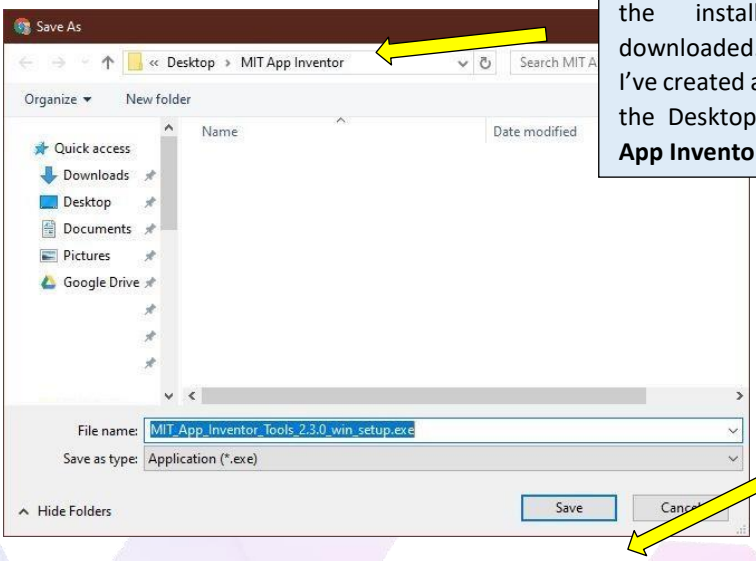
[Step#01] Connecting to Phone using USB

1. Go to https://appinventor.mit.edu/explore/ai2/setup-device-usb and scroll until you see a link to install App Inventor 2 for Windows as shown in figure below.



2. Then, click on the hyperlink **"Download the installer".** Once you've clicked on the hyperlink, a window will pop-up prompting the location to save the installer.

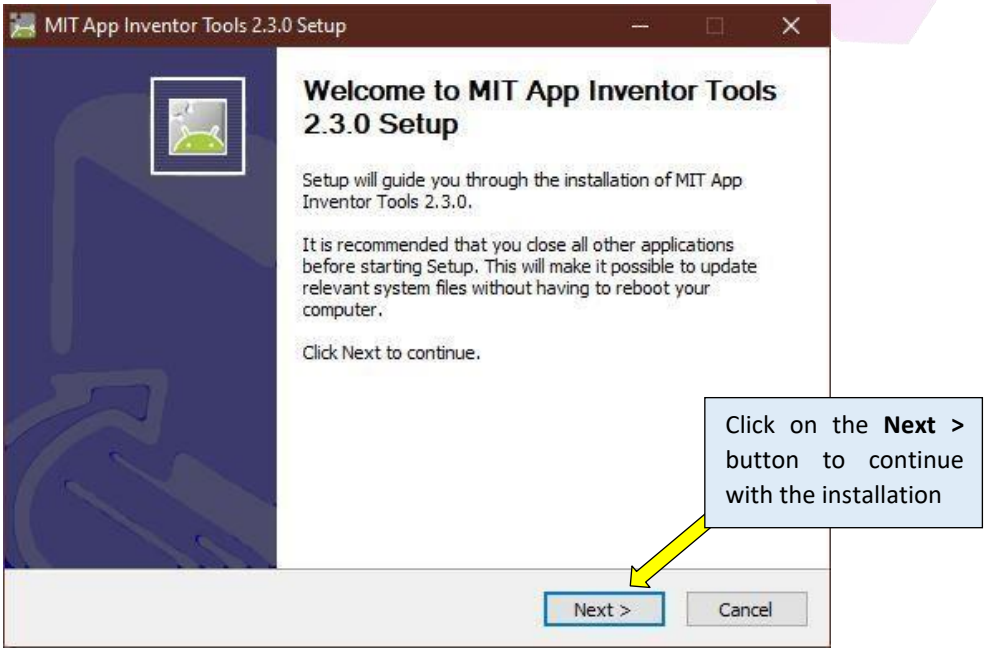




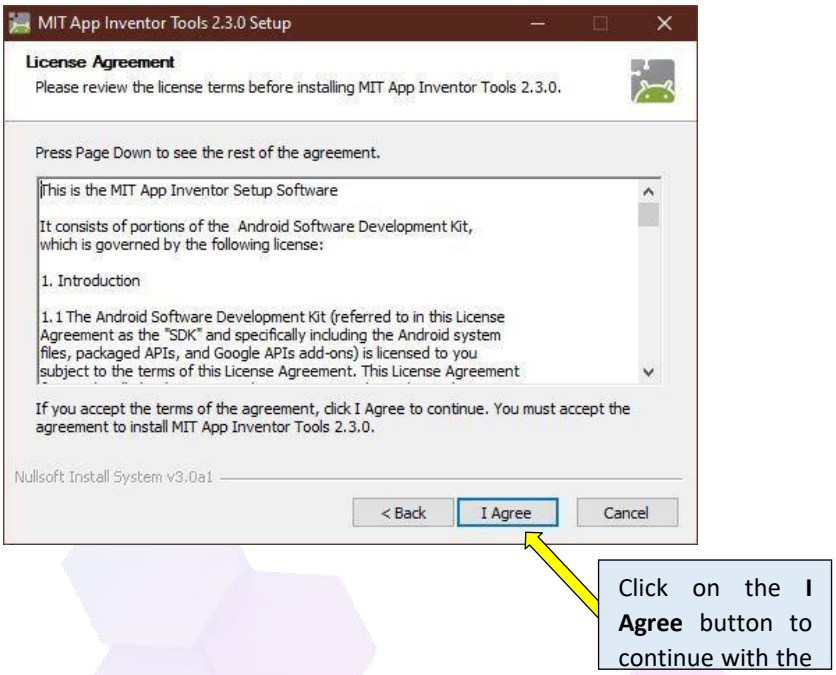
You can set the location for the installer to be downloaded. In this case I've created a folder over at the Desktop named **"MIT App Inventor"**

Click on the button **"Save"** to continue downloading the installer

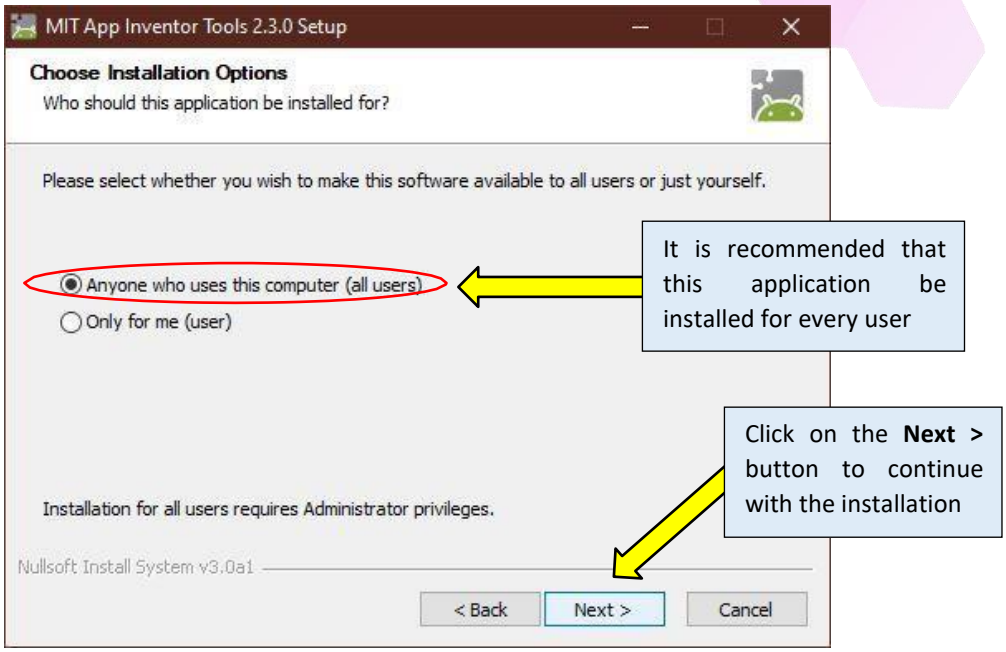
3. Once you have located the installer, right-click on it and **"Run as administrator".** Then, allow it to run.
4. After you have run the installer, a setup window will pop-up as shown in figure below. Click on the button **"Next >"** to continue with the installation.



5. Then you'll be presented with the License Agreement for MIT App Inventor Tools. Read carefully before agreeing. If you agreed with the agreement, click on the **"I Agree"** button to continue with the installation.

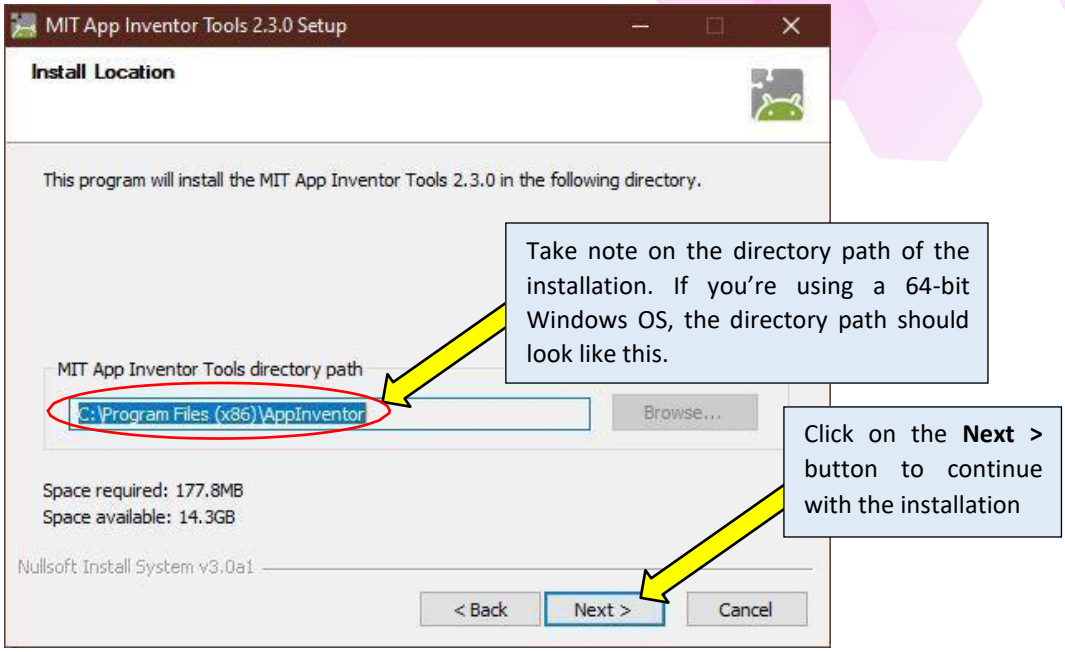


6. Next, you need to choose whether to make the application or software accessible to all users or only for the current user. It is recommended that the software to be accessible to all users.

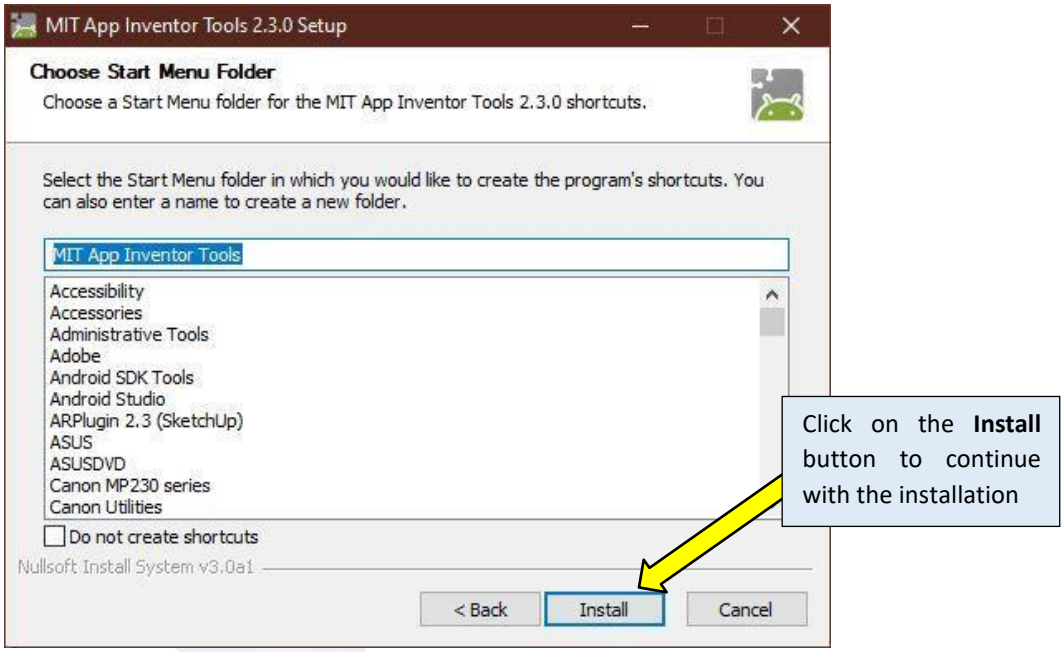


7. After that, you can choose the components to be installed. As you can see, by default the MIT App Inventor Tools have been checked. You are left to choose either to create a Desktop Icon or not.

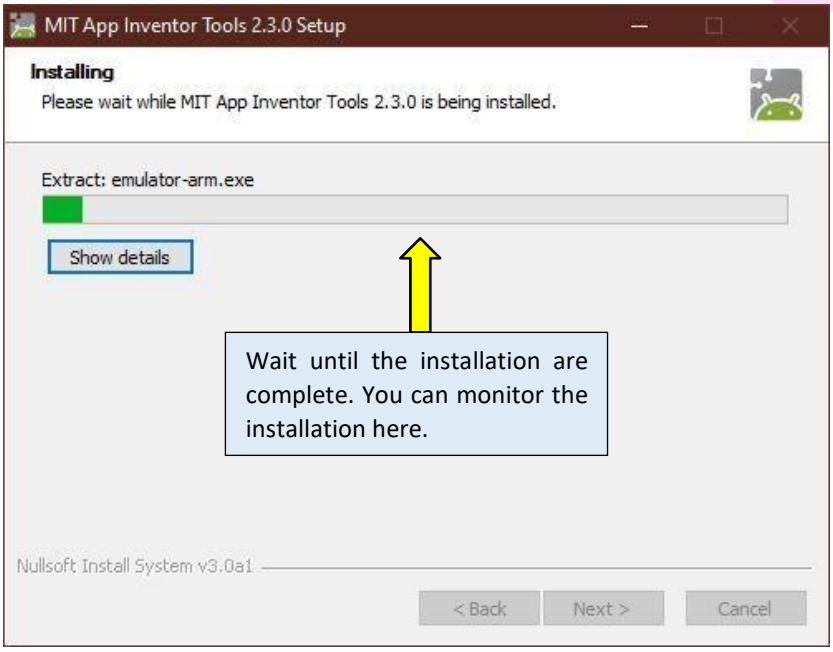
8. It is important for you to remember the directory path of the MIT App Inventor Tools.



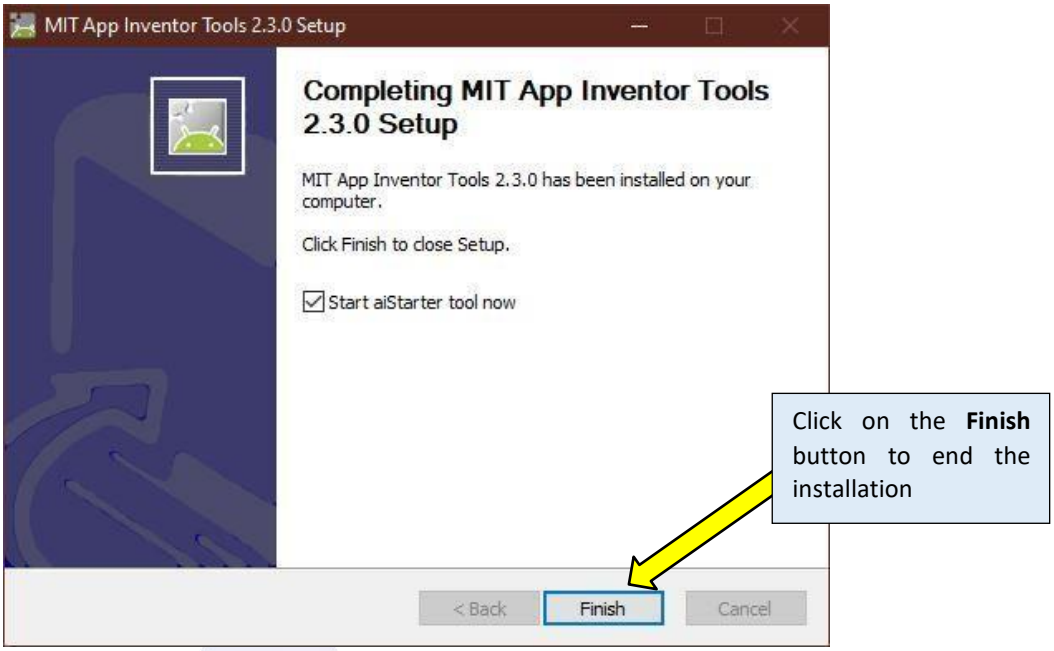
9. Now, all that is left is for you to do is to install this applications on your laptop or workstation. You can change the name of the folder for the MIT App Inventor Tools shortcut if you want but by default the name of the folder is already given to you.



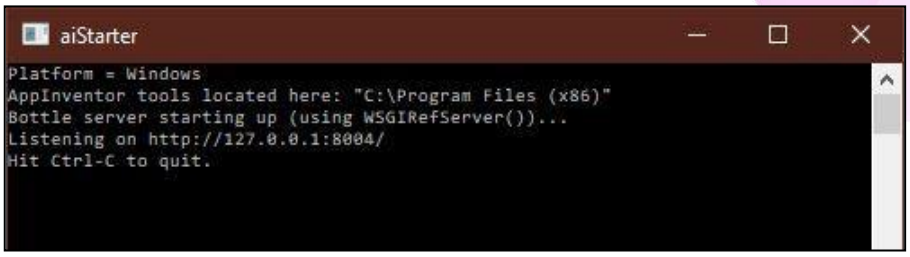
10. Sit back, relax and wait until the installation finished. You can monitor the progress of the installation from the installation window as shown in the figure below.

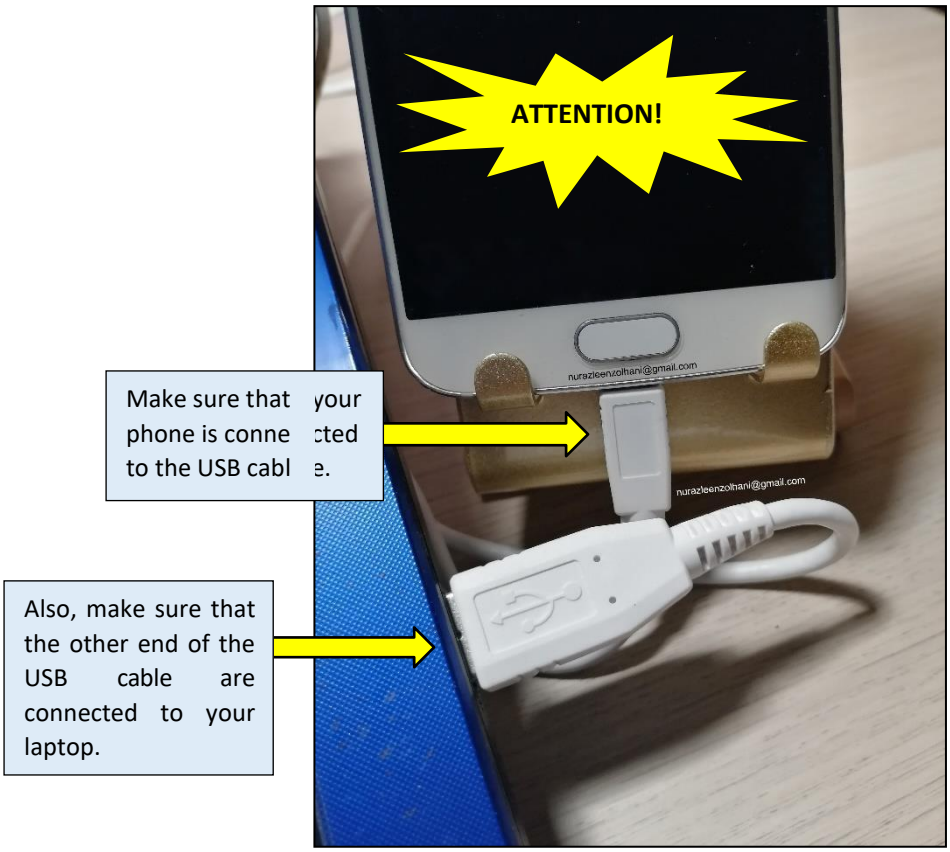


11. Lastly, click on the Finish button and the **aiStarter tool** will start.

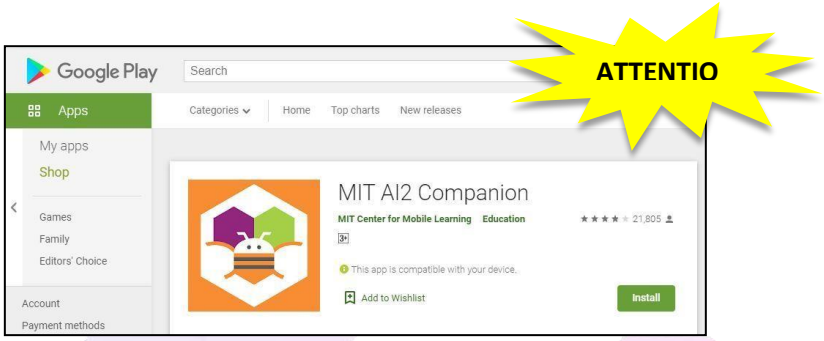


12. Then, a window will pop-up looking like the figure below. This is the tools to connect to the MIT App Inventor 2.



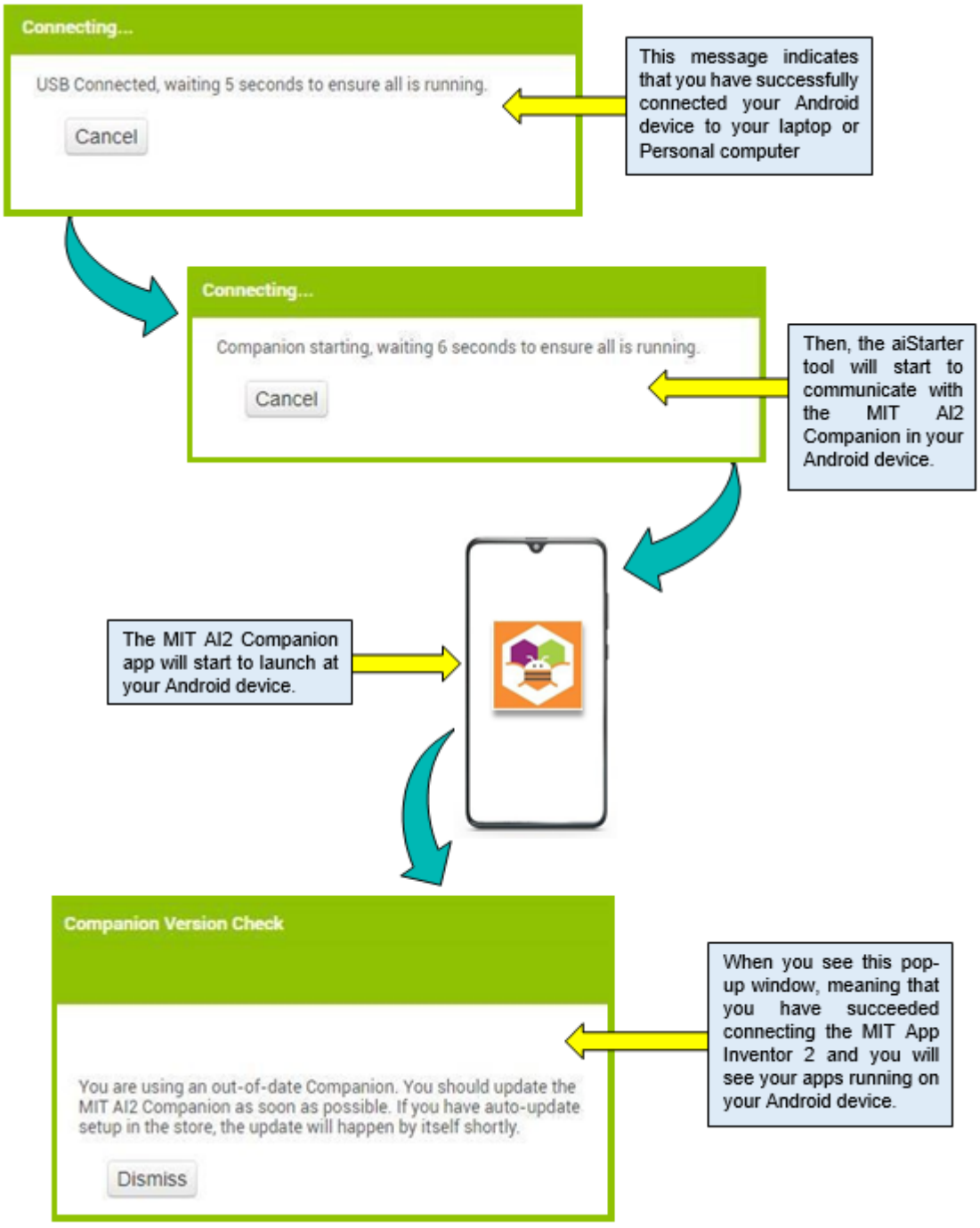


** Besides that, make sure that you **"Allow USB Debugging" at your phone**. As well as downloading the MIT AI2 Companion app over at the Google Play Store as shown in the figure below.*



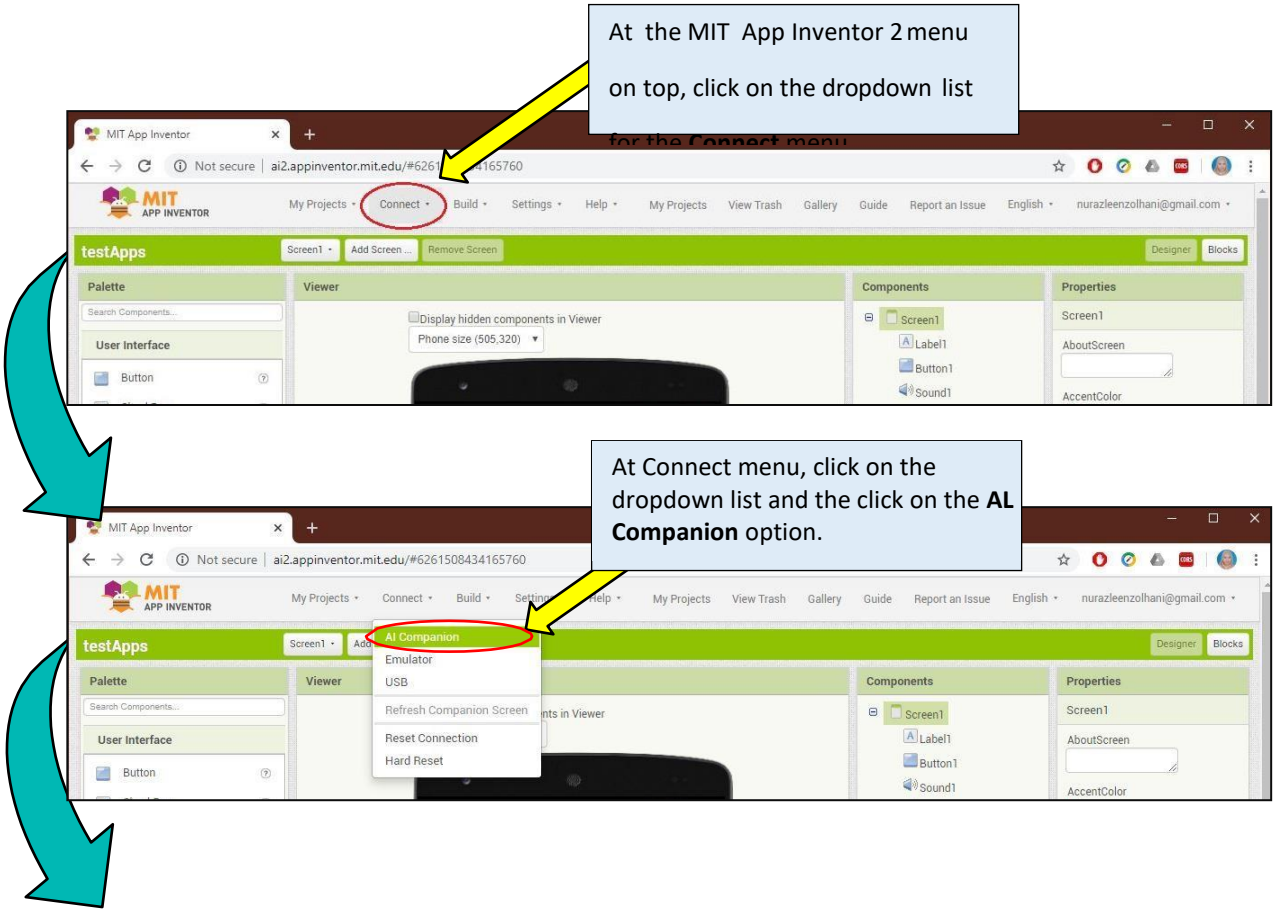
13. Next step, go to the MIT App Inventor 2 web application, and click on the second menu which the Connect menu.

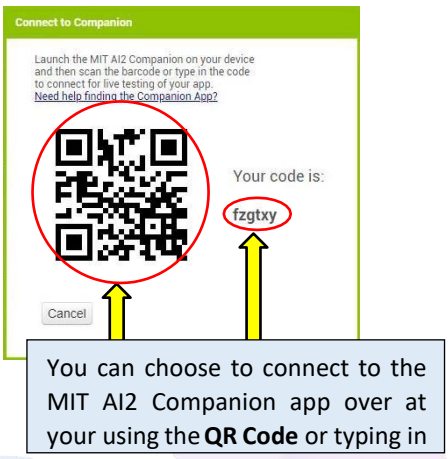
14. Once the MIT App Inventor 2 managed to connect via USB, another window will pop- up notifying that it has successfully connect via USB.



[Step#02] Connecting to Phone via Wi-Fi

1. For this step, please make sure that your phone and your laptop have internet connection.
2. Go to the MIT App Inventor 2 web application, and click on the second menu which the Connect menu.
3. Then, at the menu choose **"AI Companion"**. Then a pop-up window with a code as well QR Code will appear.





4. Over at your phone, launch your MIT AI2 Companion app, and in this lab we are going to scan the QR Code generated by the MIT App Inventor 2 web

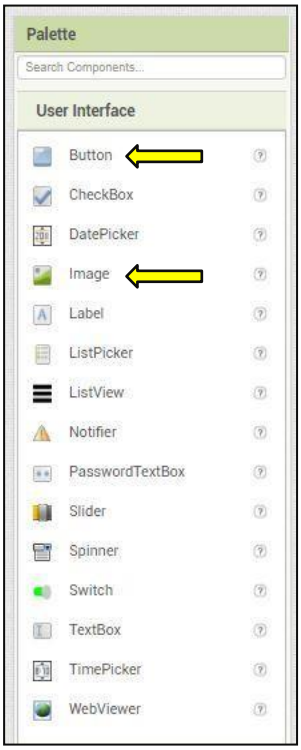
References:

a. http://appinventor.mit.edu/explore/sites/all/files/teachingappcreation/unit1/MagicTrickHa ndout.pdf
b. https://appinventor.mit.edu/explore/library
c. https://appinventor.mit.edu/explore/ai2/tutorials
d. https://www.programwithappinventor.org/
e. https://www.amazon.com/Learning-MIT-App-Inventor-Hands-On/dp/0133798631/

Module 3: Building Your First App using MIT App Inventor [3hrs]

Objective: In this lab we are going to go through the steps for building your first app using the MIT App Inventor. We will go in depth in designing the screen layout and coding the functionality of the components on the MIT App Inventor Designer.

[Step#01] User Interface



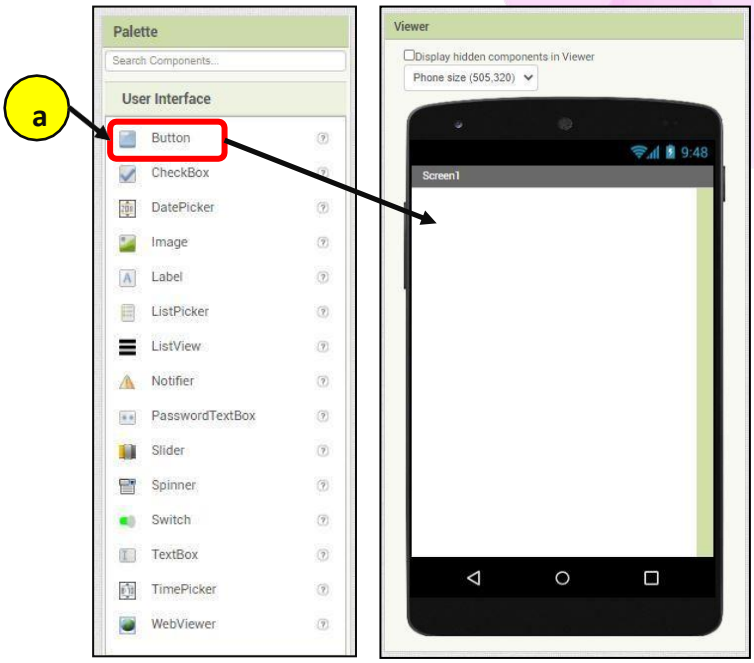
The first group of the component in the palette is the **User Interface**. In the User Interface you can see components that are usually visible on the screen an android phone. In this part of lab, we will utilize two components from the User Interface which are **Button**, and **Image**.

***Note**: From this point onwards, this lab guide will continue from the previous lab guide which is Lab 2: Setting up Connection for MIT App Inventor 2. Please make sure that you have followed and completed Lab 1 and Lab 2 step by step guide before starting with this lab guide.

You can choose the size of your display screen in the drop down list. There are 3 options which are Phone size, Tablet size and Monitor size.

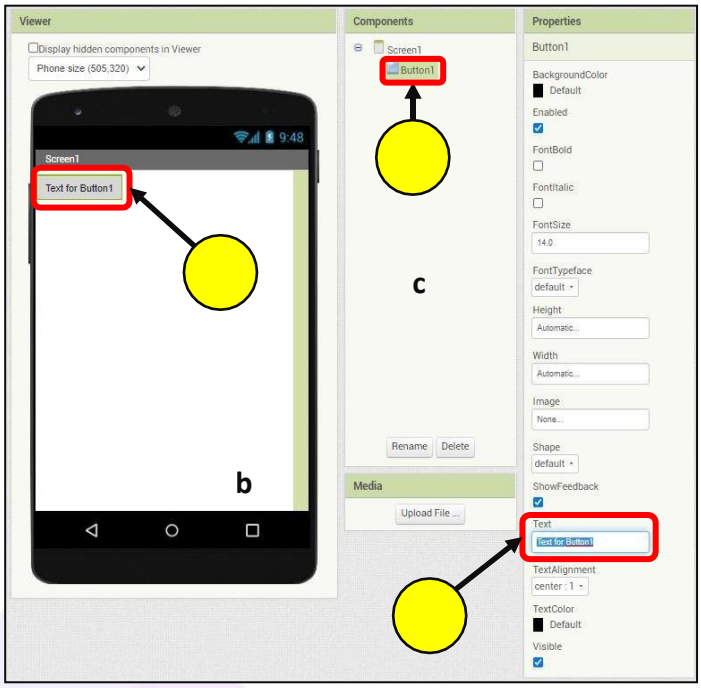
Once you have open your MIT App Inventor 2, follow the following steps.

a. ***Drag and drop*** the Button components on the Viewer at the screen of the android phone. Also, make sure that you have selected the screen to be a phone screen. Unless, you have a tablet with you then you can set the view to be in tablet mode.
b. Once you have dropped the button on the screen, we will need to ***change the text on the button*** from Text for Button 1 to Button. This can only be done at the Properties window on the right side of the window. As mention in Lab 1 guide, the properties window is responsible in changing the property of the components on the screen.



c. Take note that, once you have dragged the components on the screen, you can see a tree view list of the components available on the screen as shown in the figure below. Be sure to select the components that you wanted to change the property.

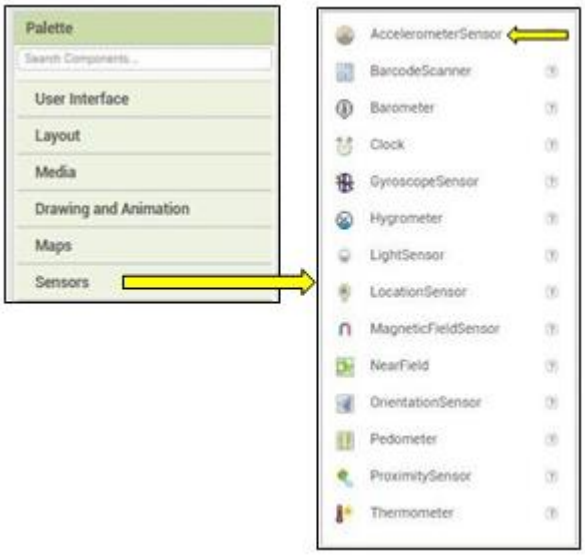
d. Change the text appearing on the Button by changing it at the Text Property. As mention previously, please change it from ***Text for Button 1*** to ***Button.***



e. The end result should look like figure below.

[Step#02] Sensors

Next, we will be adding a Sensor on our android app. The sensors component can be found on the palette as show in figure on the right. The sensor listed are sensors that are commonly found on an android phone.

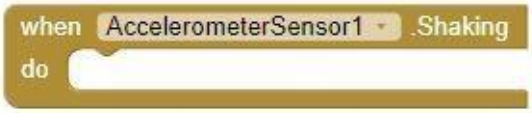


In this lab guide, we will be adding an **AccelerometerSensor** on our screen.

We will be implementing a code as show below which will do an event when the **AccelerometerSensor** detects a shaking motion.

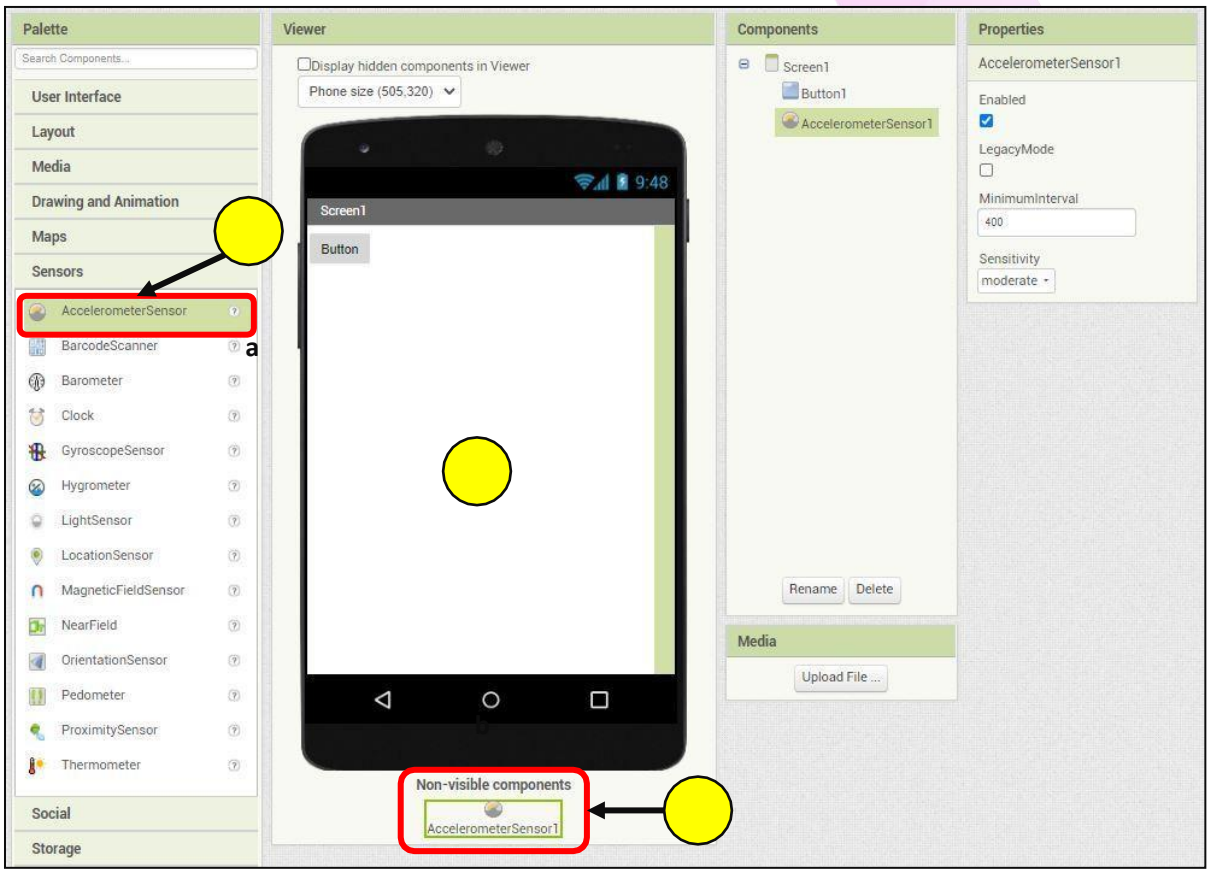
The steps for adding the **AccelerometerSensor** are as follows:

a. Select the **AccelerometerSensor** from the Sensors component list.
b. Drag it across the screen and drop it on



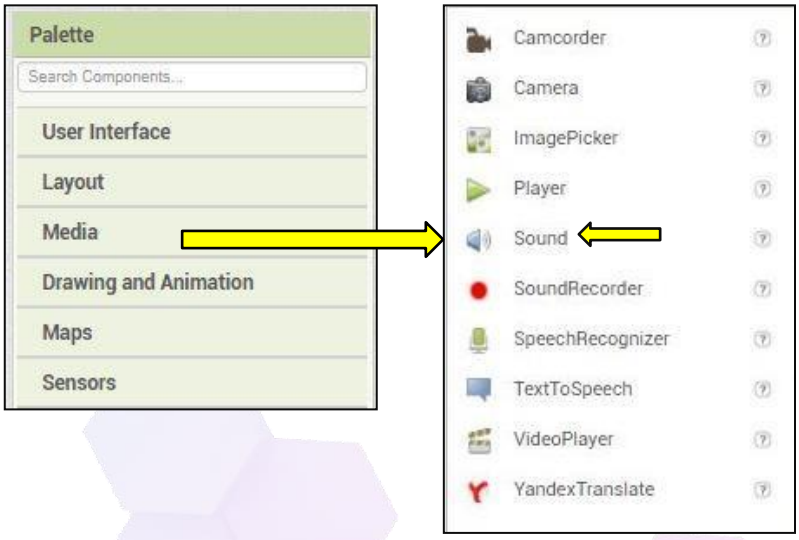
the screen.

c. Take note that, the **AccelerometerSensor** is an invisible component on the android apps. It will only appear at the bottom part of the viewer.



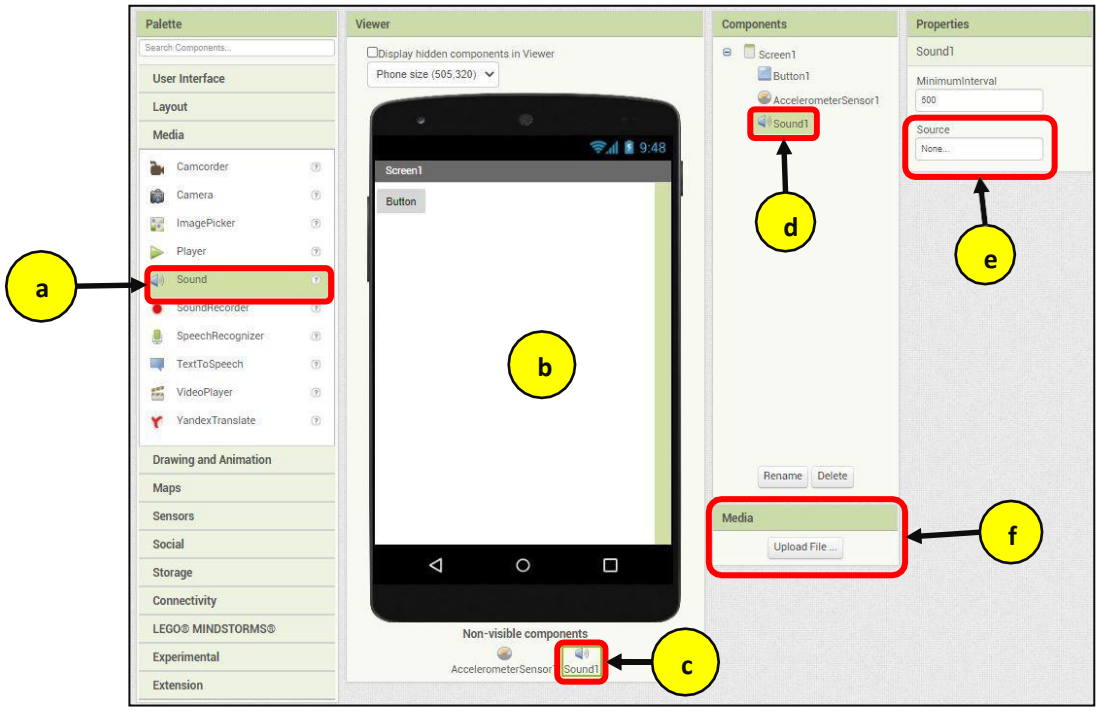
[Step#03] Media

Then, we will be adding a Sound on our app. The Sound component can be found on the Palette at the Media group component as shown in figure below.



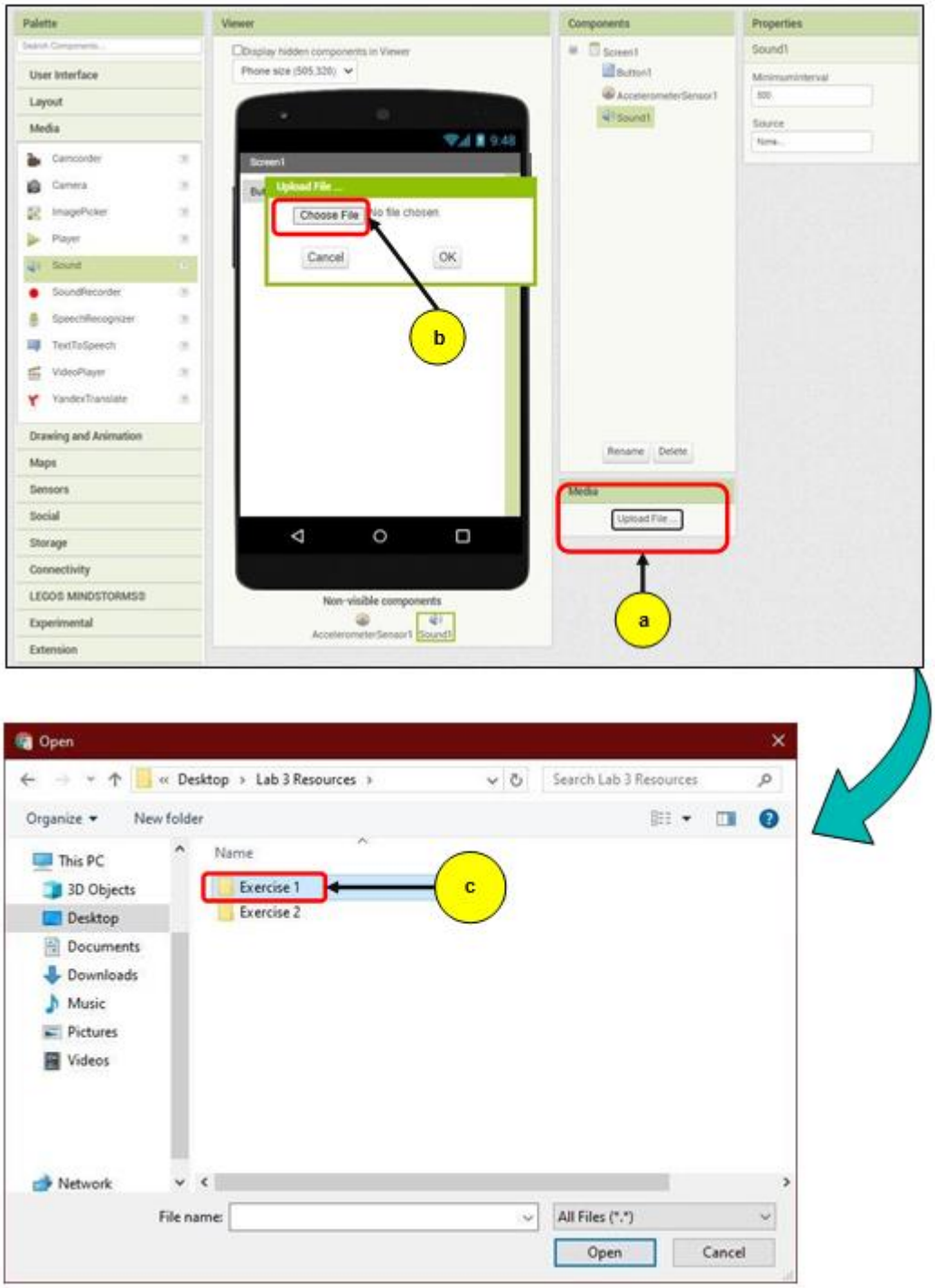
The steps for adding the **Sound** on the MIT App Inventor 2 are as follows:

a. Select the **Sound** from the Media component list.
b. Drag it across the screen and drop it on the screen.
c. Take note that, the **Sound** is an invisible component on the android apps. It will only appear at the bottom part of the viewer
d. The select the Sound1 from the Component tree view list.
e. Notice that the Source at the Properties of the Sound1 is **None**.
f. To change this, we need to upload a sound file to the MIT App Inventor 2 server.

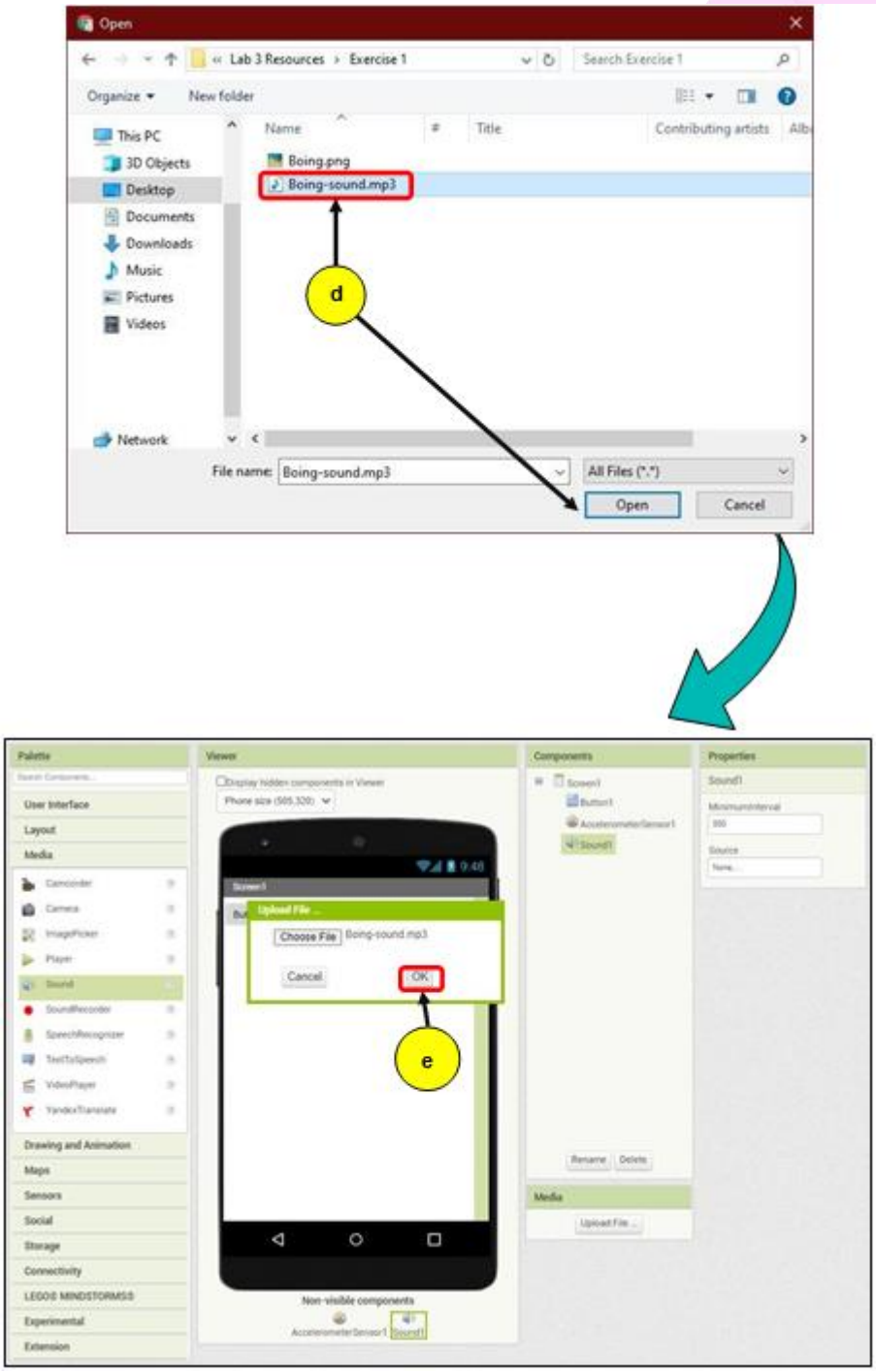


The following steps will show you ways to upload a media file onto the MIT App Inventor 2.

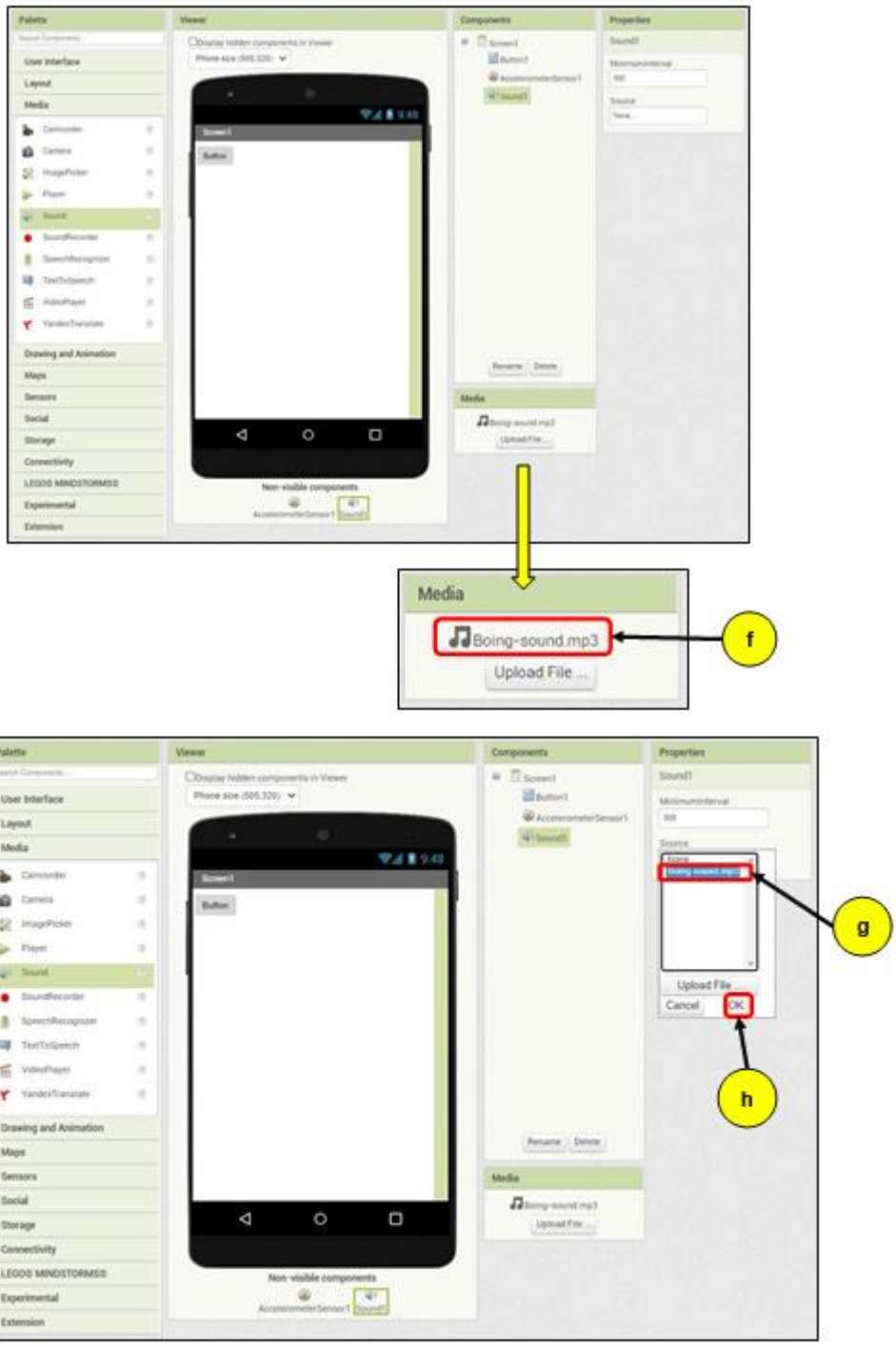
a. Go to the Media at the bottom right of the MIT App Inventor and click on the Upload File button.
b. Then, a prompt for Upload file will appear at the centre of the window. Click on Choose File button.
c. A window will pop-up. Choose the folder Exercise 1. Double-click to open.
d. Choose the ***Boing-sound.mp3*** file and click **Open**.
e. You will be returning back to the MIT App Inventor 2 window. Click OK.
f. Take note that, at the Media there is a file titled ***Boing-sound.mp3*** in the list.
g. Lastly, go to the Properties for Source and choose the file ***Boing-sound.mp3.***
h. Click OK to append the file to the Source.


Palette
Viewer
Components
Properties
User Interface
Layout
Media
Drawing and Animation
Maps
Sensors
Social
Storage
Connectivity
LEGOS MINDSTORMS
Experimental
Extension
Screen1
Upload File ...
Choose File
No file chosen
Cancel
OK
b
Non-visible components
AccelerometerSensor1
Sound1
Button1
Rename
Delete
Media
Upload File ...
a

Open
Desktop
Lab 3 Resources
Search Lab 3 Resources
Organize
New folder
This PC
3D Objects
Desktop
Documents
Downloads
Music
Pictures
Videos
Network
Name
Exercise 1
Exercise 2
c
File name:
All Files (*.*)
Open
Cancel


Open
Lab 3 Resources › Exercise 1
Search Exercise 1
Organize
New folder
This PC
3D Objects
Desktop
Documents
Downloads
Music
Pictures
Videos
Network
Name
Title
Contributing artists
Boing.png
Boing-sound.mp3
d
File name: Boing-sound.mp3
All Files (*.*)
Open
Cancel

Palette
Viewer
Components
Properties
Upload File ...
Choose File
Boing-sound.mp3
Cancel
OK
e
Non-visible components

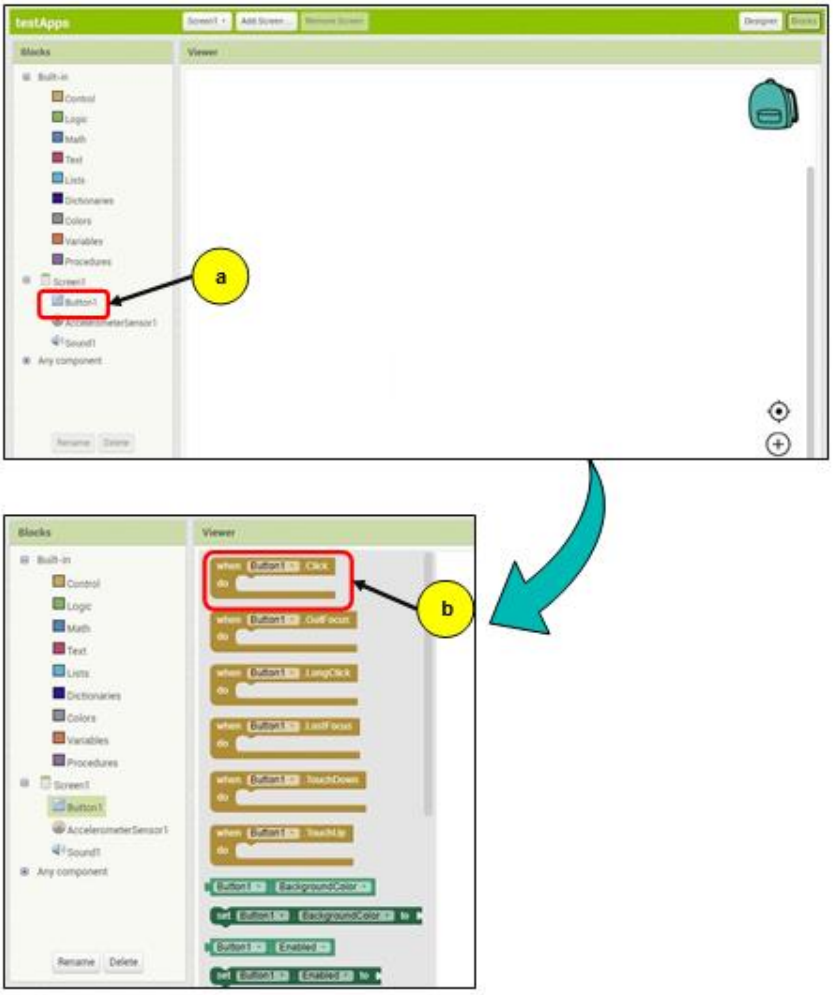


[Step#04] Implementing code in MIT App Inventor

After designing your android screen, we will then proceed with coding the android apps at the MIT App Inventor Blocks. The following steps will show you ways to code by arranging the block at the MIT App Inventor 2 Blocks. Make sure that you have open the MIT App Inventor 2 Blocks.

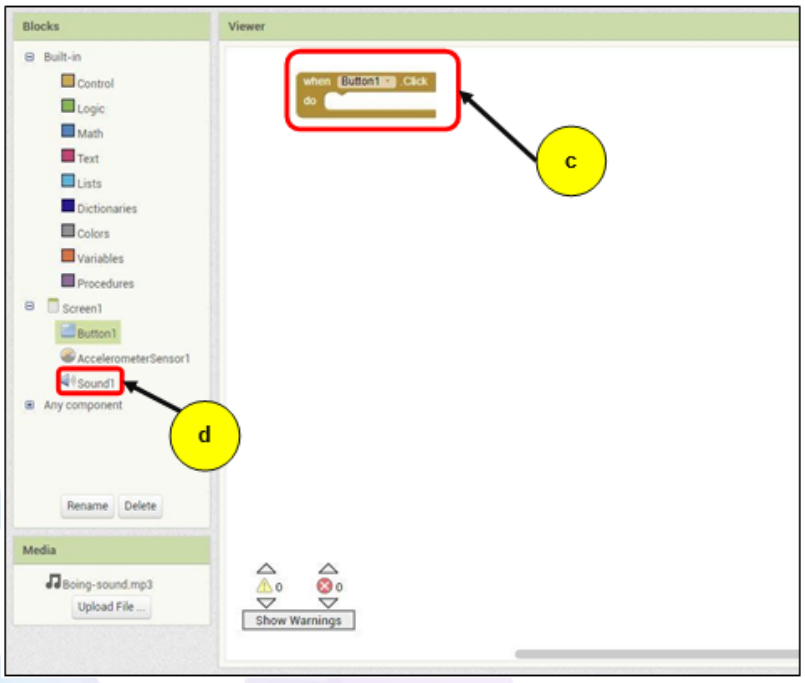
a. Firstly, select the ***Button1*** at the ***Component Blocks***.

b. Then, select code block for the activity ***when Button1.Click*** *(see figure below).*

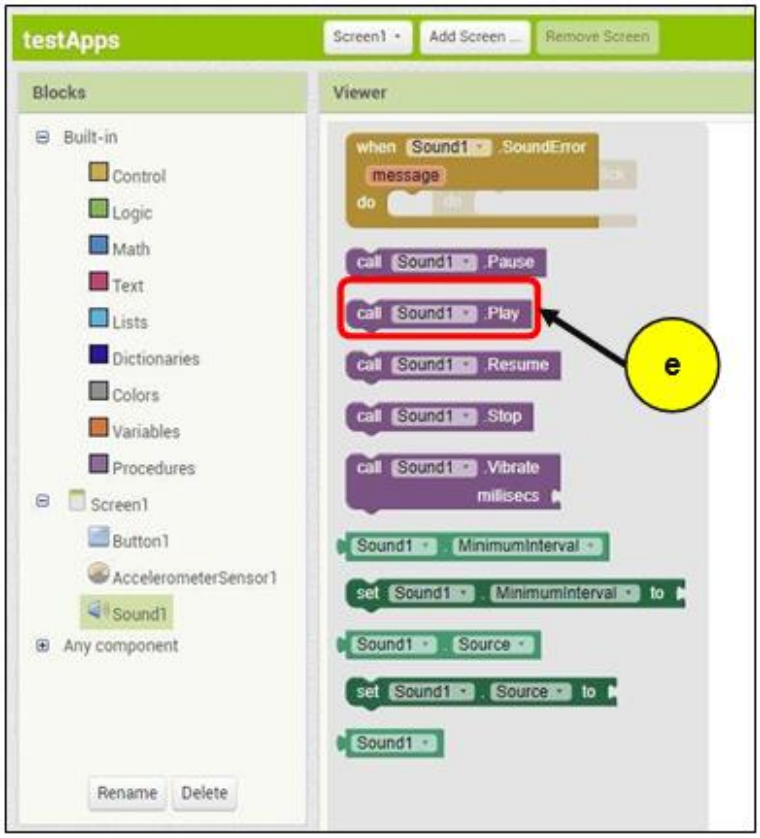


c. Drag and drop it on the **Viewer**.

d. After that, select ***Sound1*** at the ***Component Blocks***.



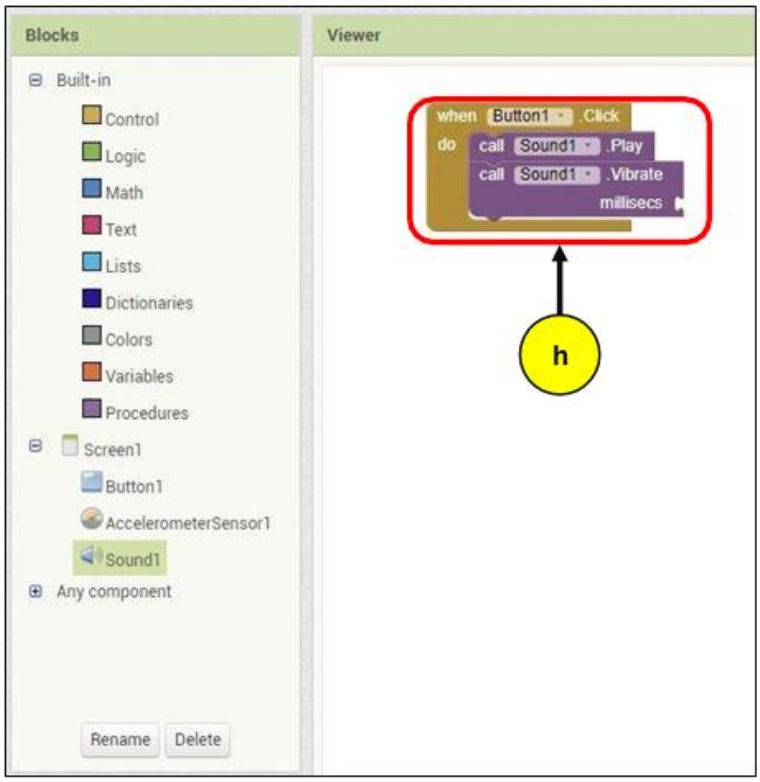
e. Select the block for ***call Sound1.Play*** *(see figure below).*



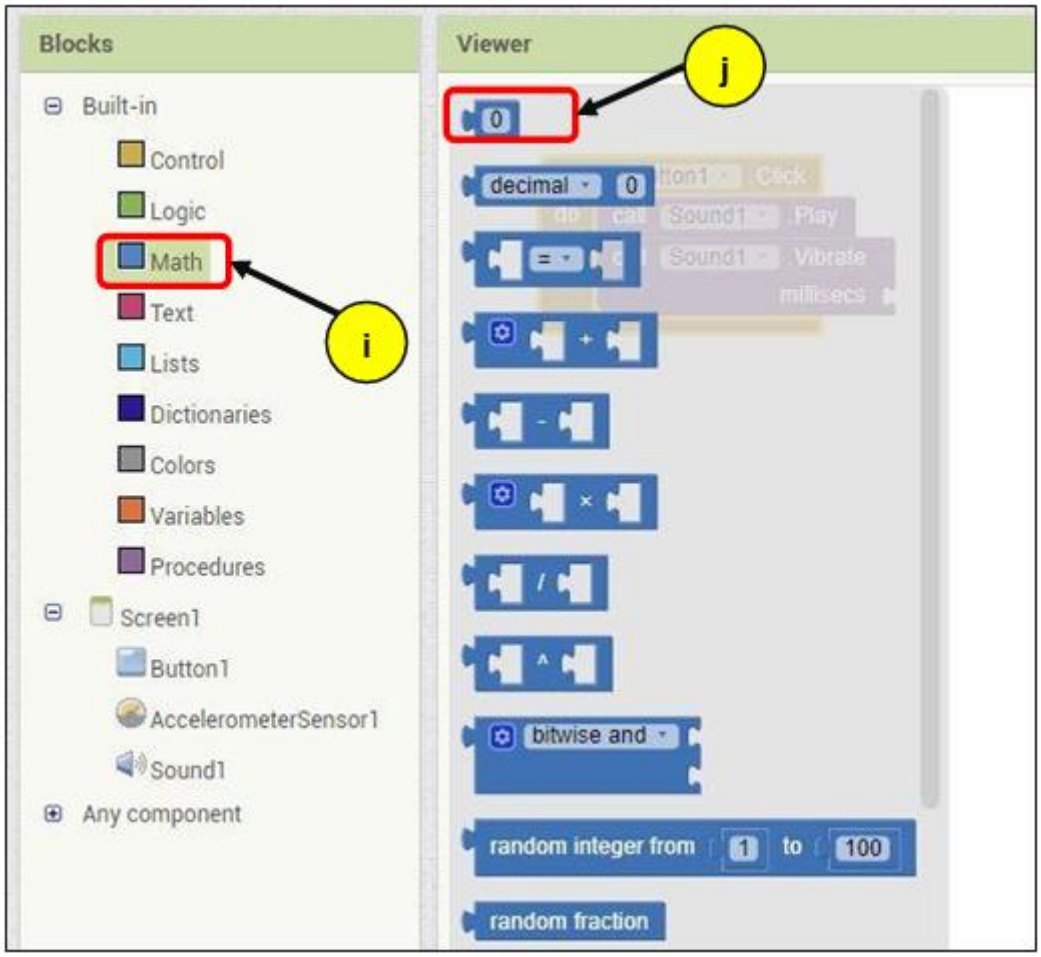
f. Drag and drop it on the **Viewer** and arrange it under the ***when Button1.Click*** block *(see figure below).*

g. Then, at the same block (Sound1) select the block **call Sound1.Vibrate millisecs** *(see figure below).*

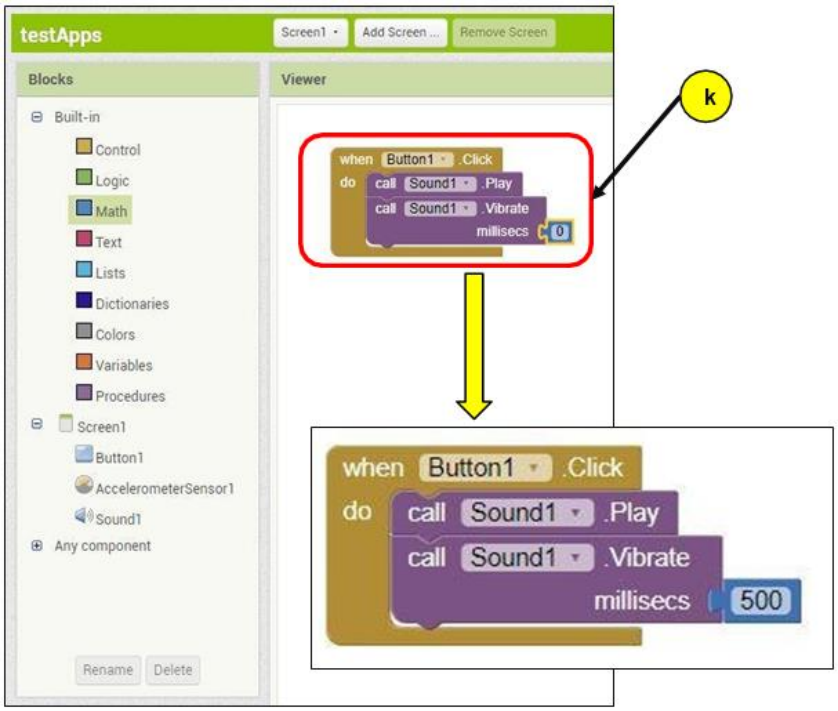
h. Arrange the block under the ***call Sound1.Play*** block *(see figure below).*



i. Then, choose the **Math** block at the **Built-in Blocks**.
j. Select the number block.

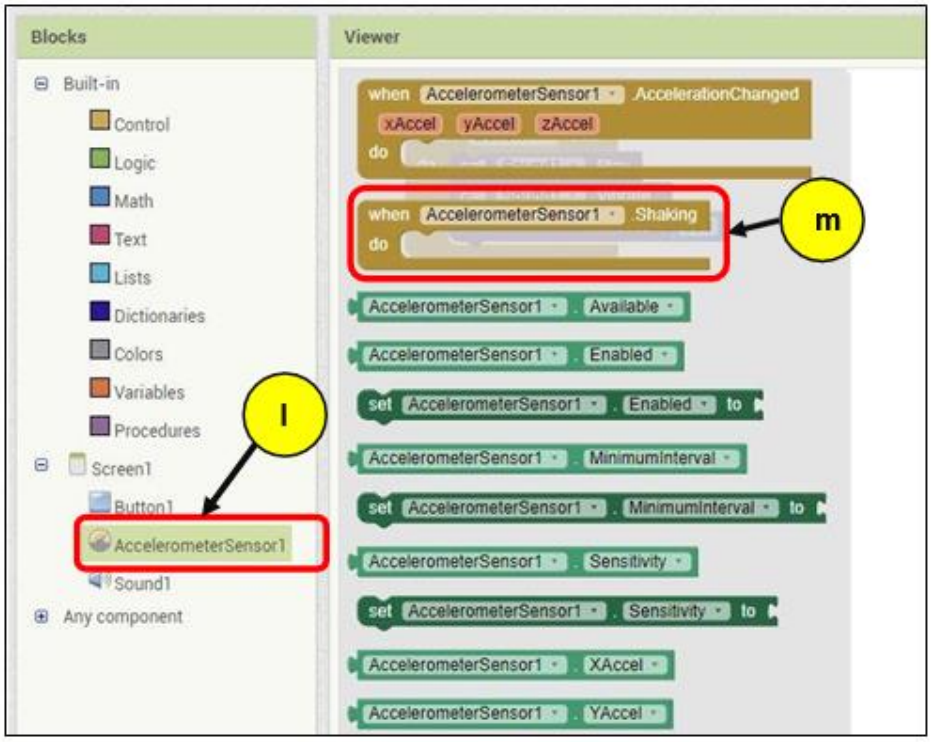

Blocks
Built-in
Control
Logic
Math
Text
Lists
Dictionaries
Colors
Variables
Procedures
Screen1
Button1
AccelerometerSensor1
Sound1
Any component
i
Viewer
j
0
decimal
0
bitwise and
random integer from
1
to
100
random fraction

k. Append it at the call **Sound1.Vibrate millisecs** block and change the number to **500**.

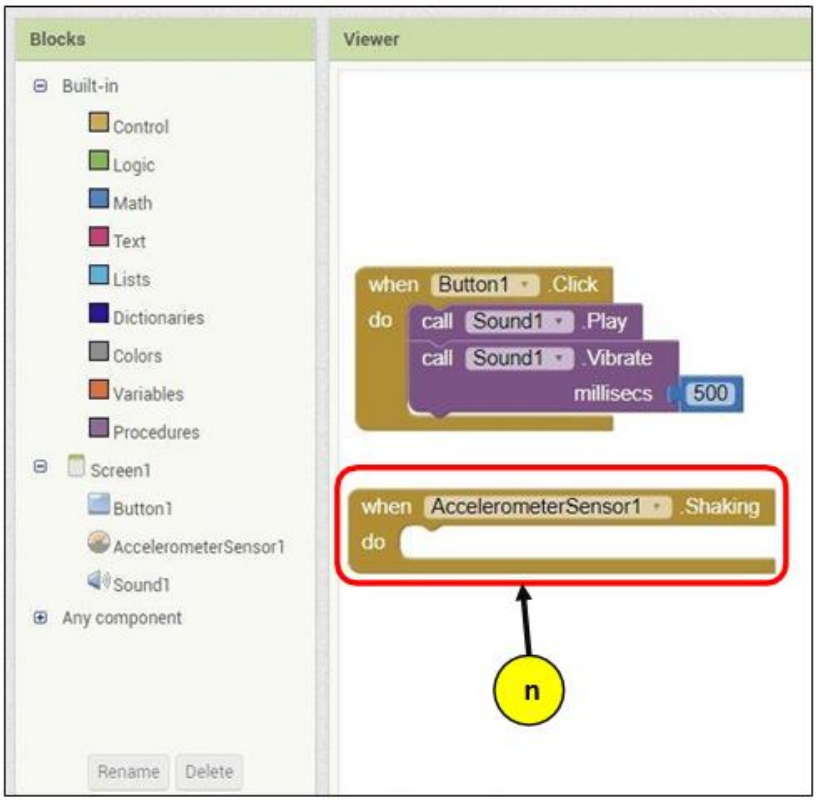


l. Select the ***AccelerometerSensor1*** at the ***Component Blocks***.

m. Then, choose the block ***when AccelerometerSensor1.Shaking***.



n. Drag and drop the block at the **Viewer**. You can place the block anywhere on the Viewer but it is best to place it under the Button1 block (*see figure below).*

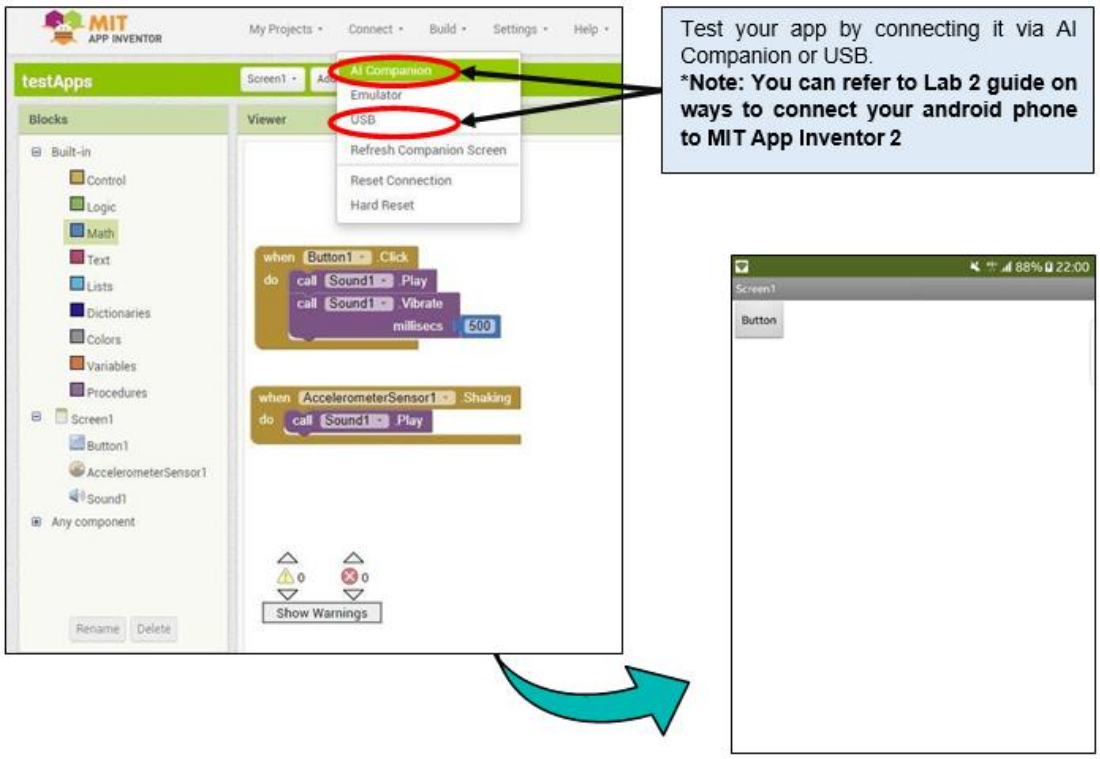


o. As in previous step, select ***Sound1*** at the ***Component Blocks***.

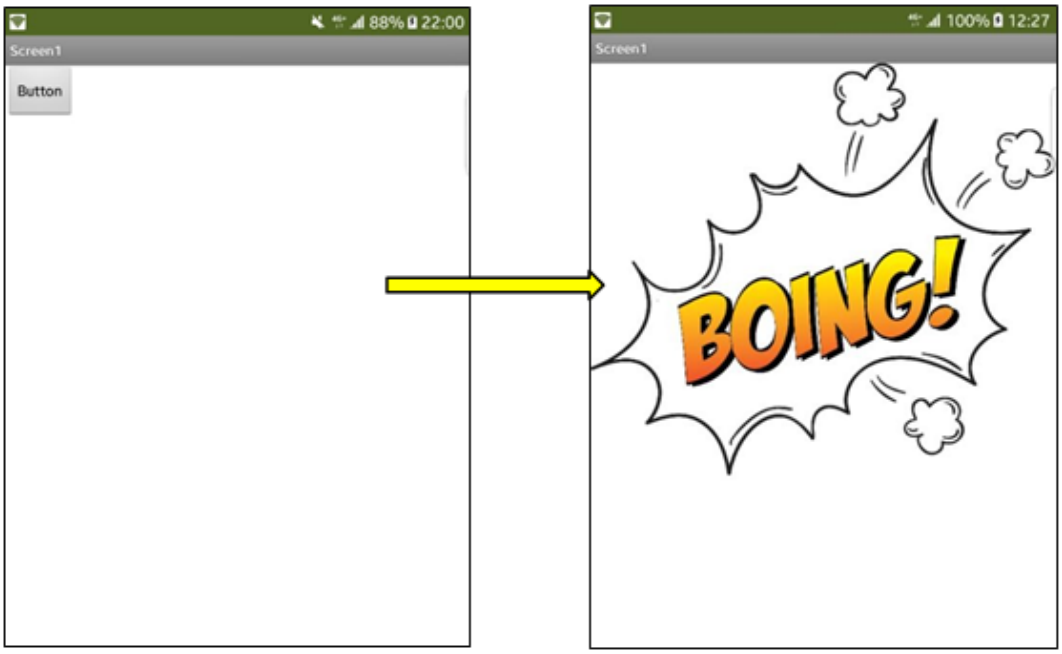
p. Then, select the block for ***call Sound1.Play*** *(see figure below).*

q. Arrange the block under the ***when AccelerometerSensor1.Shaking*** *(see figure below).*

r. Lastly, connect your Android Phone via WiFi (AI Companion) or USB and test the app!



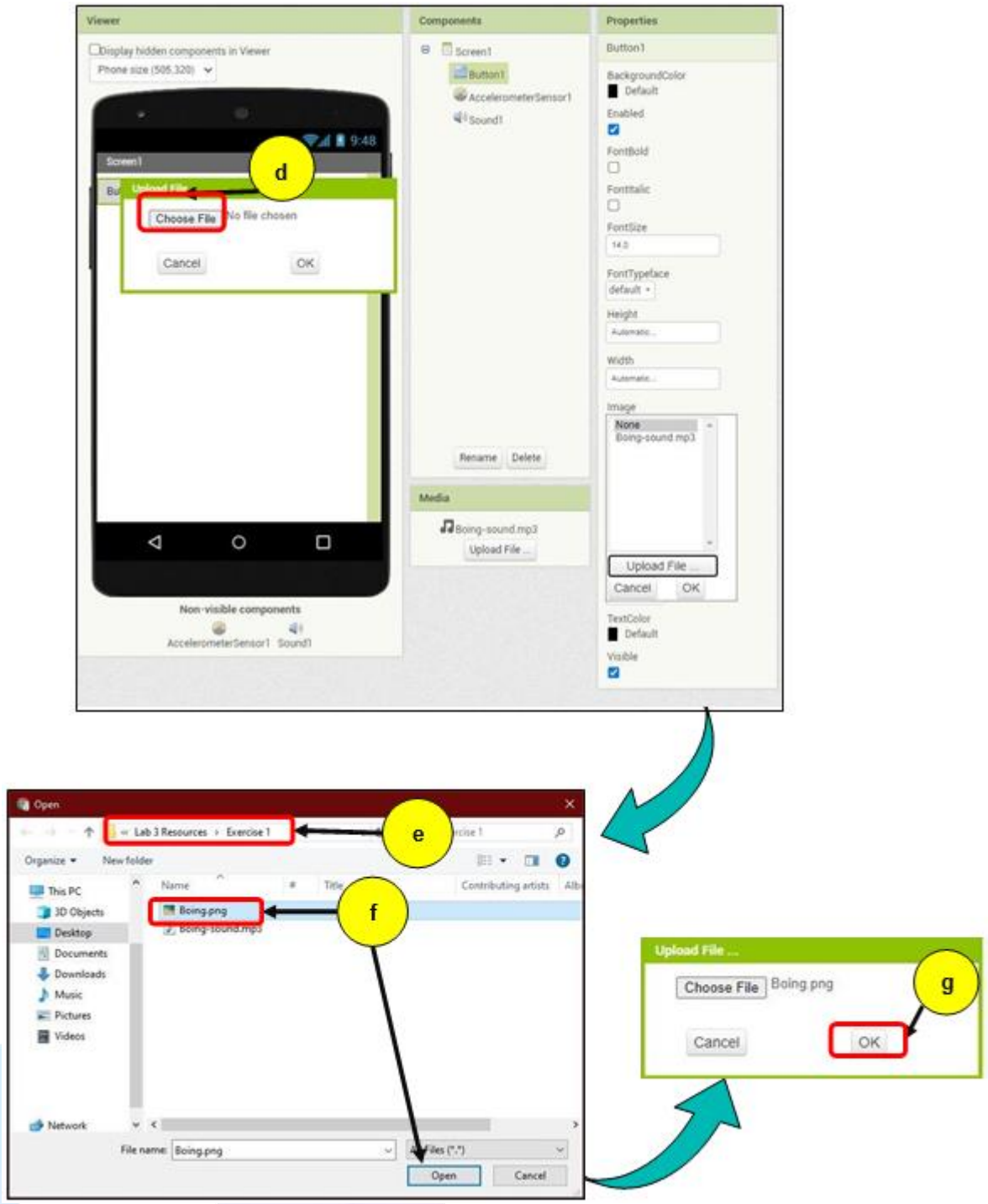
[Step#05] Adding Image to Button



A

a. Make sure that you have selected Button1 at the Components.
b. Then, at the Properties, go to Image. As you can see that the Image is currently None
c. Click on Upload File

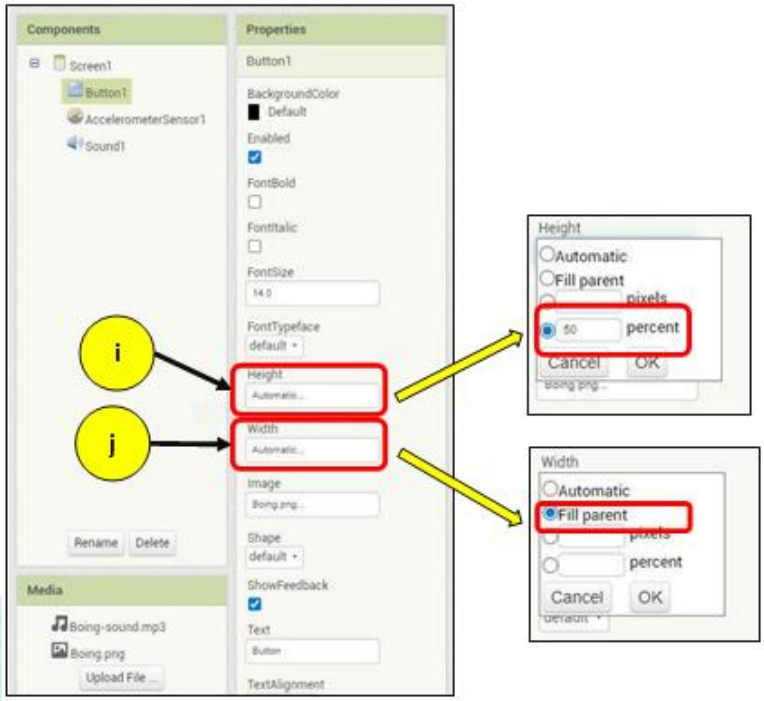
d. Then, a prompt for Upload file will appear at the centre of the window. Click on Choose File button.
e. A window will pop-up. The image will be in a subfolder named Exercise 1 in the Lab 3 Resources folder (the same folder we added the Sound1 source).
f. Choose the ***Boing.png*** file and click **Open.**
g. You will be returning back to the MIT App Inventor 2 window. Click **OK** to upload the file.



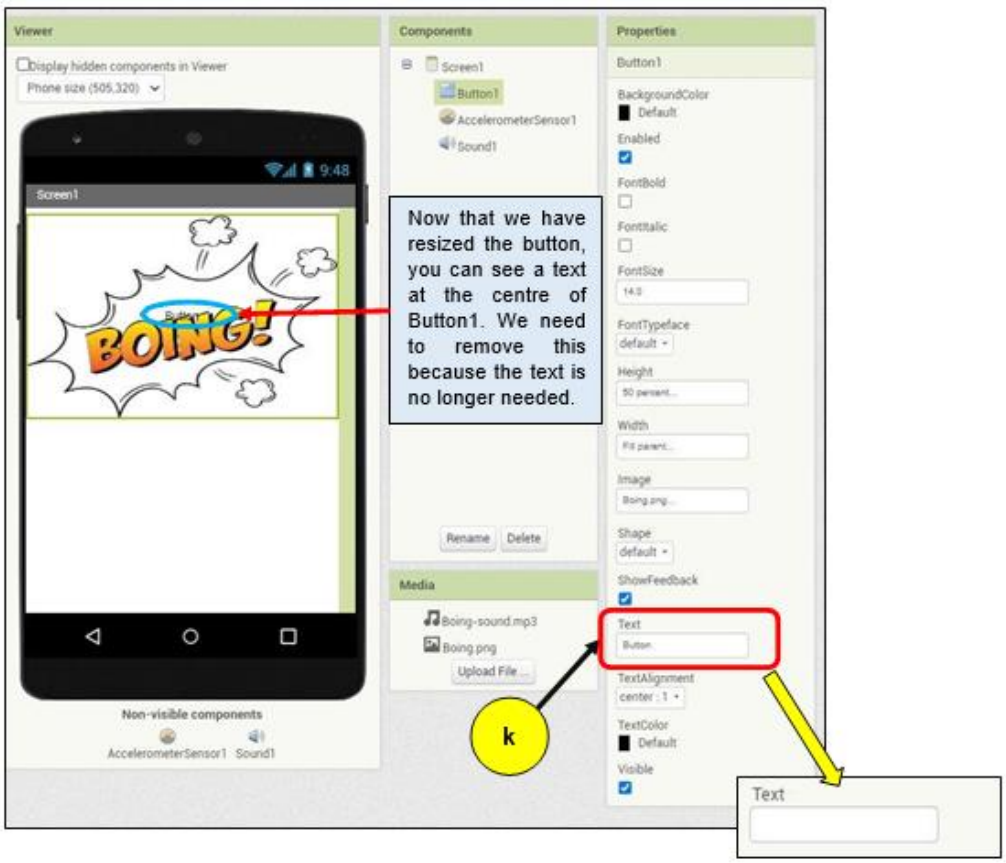
h. As you can see now, the Image is currently set to **Boing.png**

i. Now, we are going to change the Height of Button1. Currently, the height of Button1 is being set to Automatic. Change this to **50 percent.**

j. Next, we need to change the Width of Button1. Set it to **Fill parent**.



k. Then, we need to remove the text “Button” on Button1.



Since our phone are still connecting to the MIT App Inventor 2, we can simply Refresh the Interface of our AI Companion.

References:

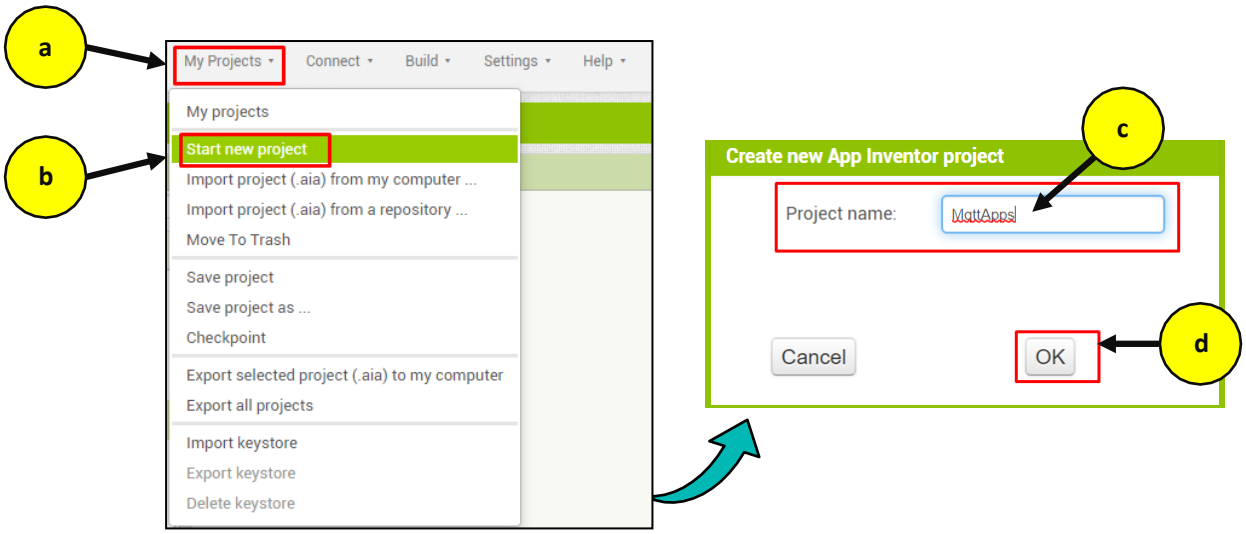
1. http://appinventor.mit.edu/explore/sites/all/files/teachingappcreation/unit1/MagicTrickHa ndout.pdf
2. https://appinventor.mit.edu/explore/library
3. https://appinventor.mit.edu/explore/ai2/tutorials
4. https://www.programwithappinventor.org/
5. https://www.amazon.com/Learning-MIT-App-Inventor-Hands-On/dp/0133798631/

Module 4: Developing Internet of Things App using MIT App Inventor [3hrs]

Objective: In this lab we are going to go through the steps of creating a MQTT apps. MQTT is the most commonly used Internet of Things Communication Protocols. We will be implementing this protocols in our apps.

[Step#01] Create a new project

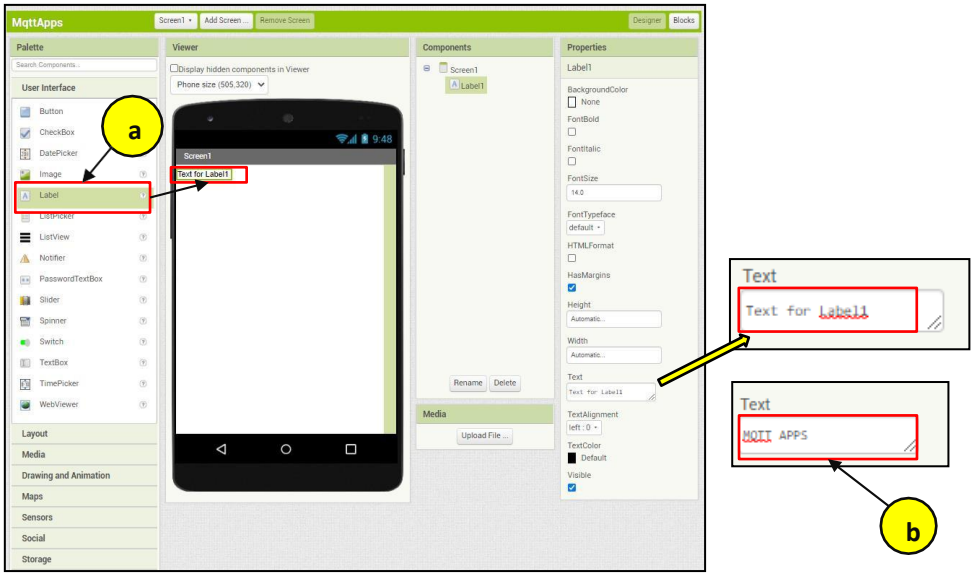
a. Go to My Projects.
b. Select Start new project
c. A pop-up will appear. Typed in the project name as **"MqttApps".**
d. Then, click **OK**.



[Step#02] Designing the Broker Settings

a. At the **Palette** under the **User Interface** group components, choose the **Label** and then drag and drop it onto **Screen1**

b. Change text at the Label1 from ***Text for Label1*** to ***MQTT APPS***



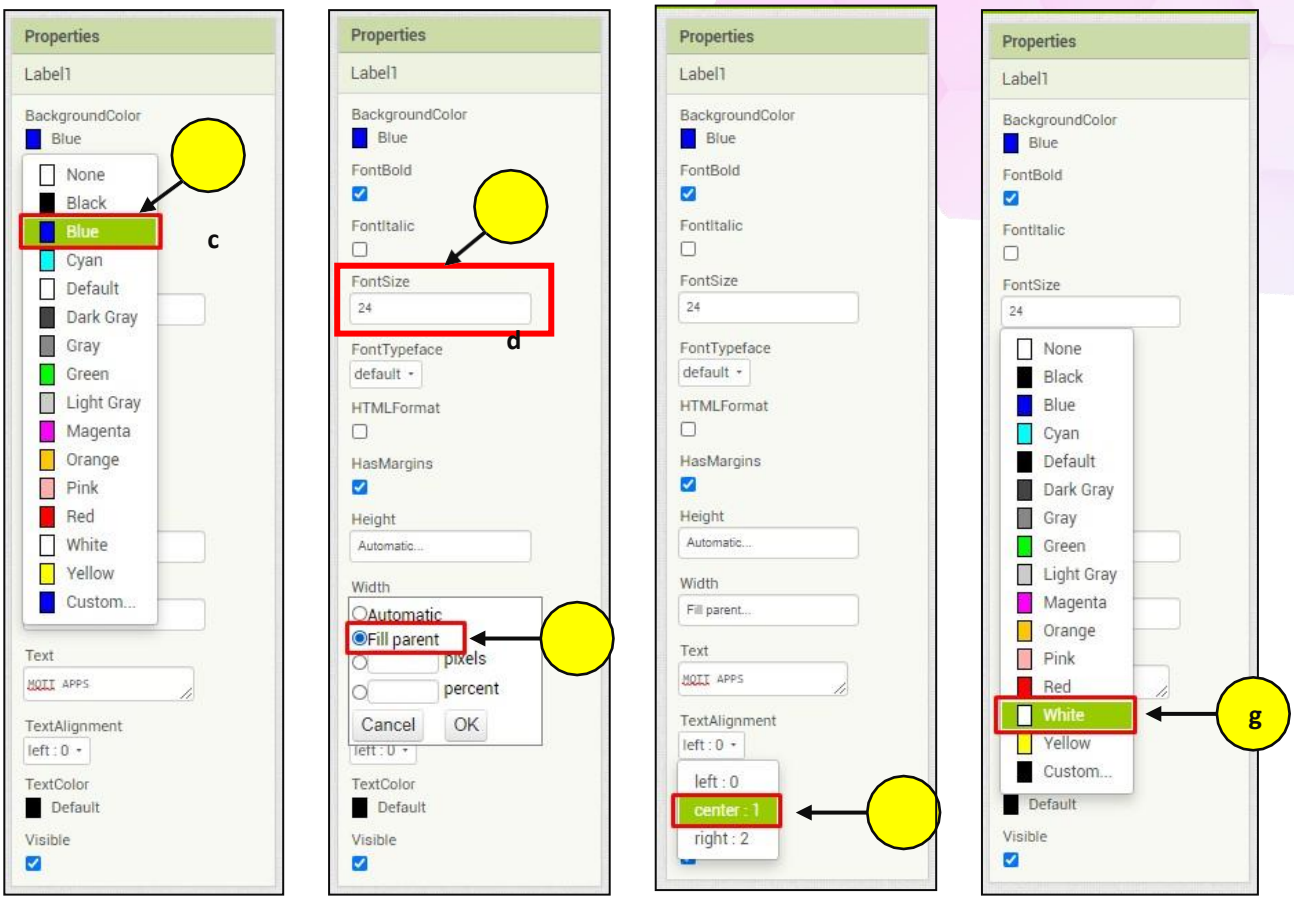
c. Then, change the **BackgroundColor** to ***Blue***

d. Next, change the **FontSize** to ***24***.

e. After that, change the **Width** of Label1 to ***Fill parent***.

f. To align the text at the centre of the screen, change the **TextAlignment** to ***center:1***

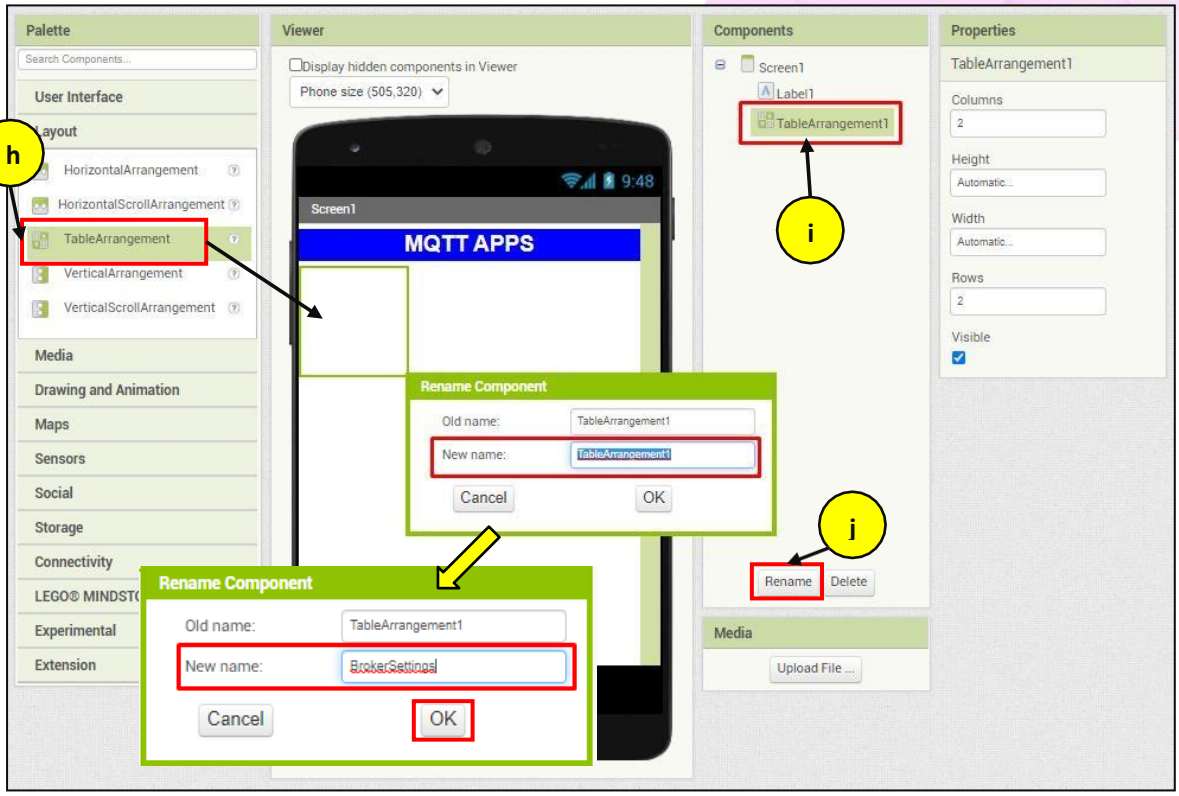
g. Lastly, change the **TextColor** to ***White***.



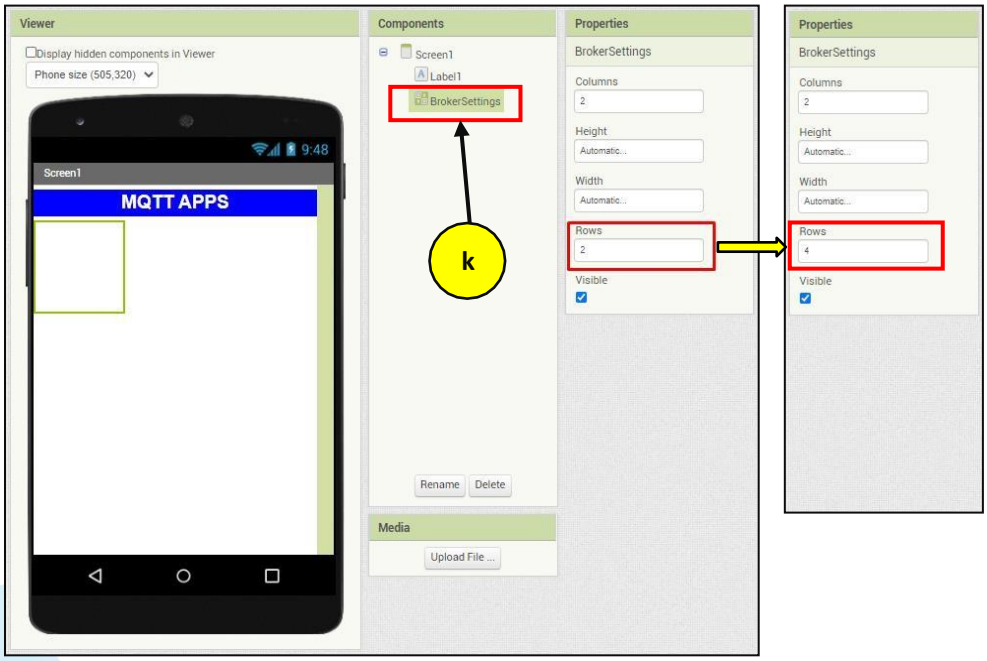
h. Then, go to the **Palette** and expand the **Layout** group component. Choose the **TableArrangement layout** and drag and drop it on **Screen1**

i. Select the **TableArrangement1** at the **Components**.

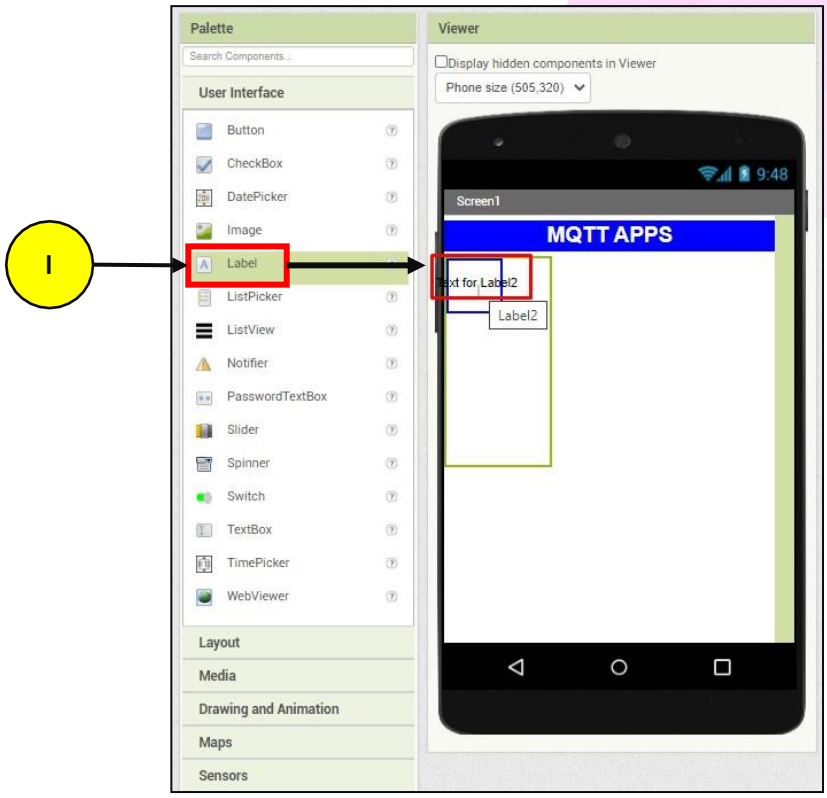
j. Click on the button Rename and change it to ***BrokerSettings***.



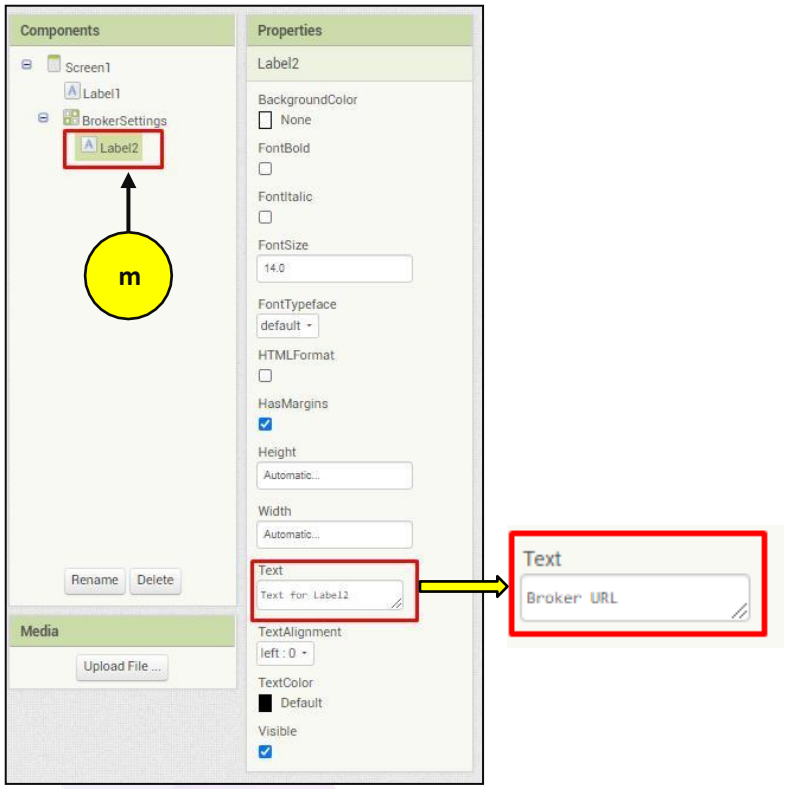
k. After that, select the **BrokerSettings layout** at the **Components** and change the **Rows** number from ***2 to 4***



l. At the Palette, select a Label at the User Interface. Drag and drop it on the first row, first column to the BrokerSettings layout.

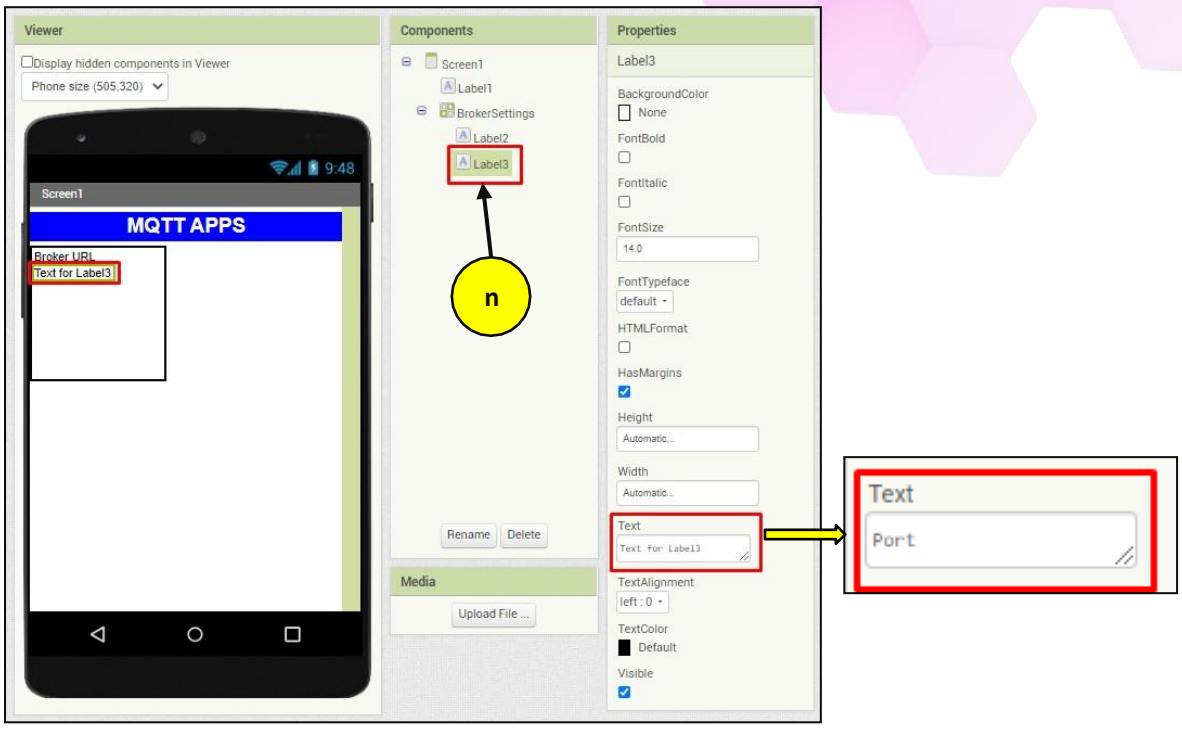


m. Then, select the **Label2** underneath the **BrokerSettings** and change the **Text** at the **Properties** to ***Broker URL***

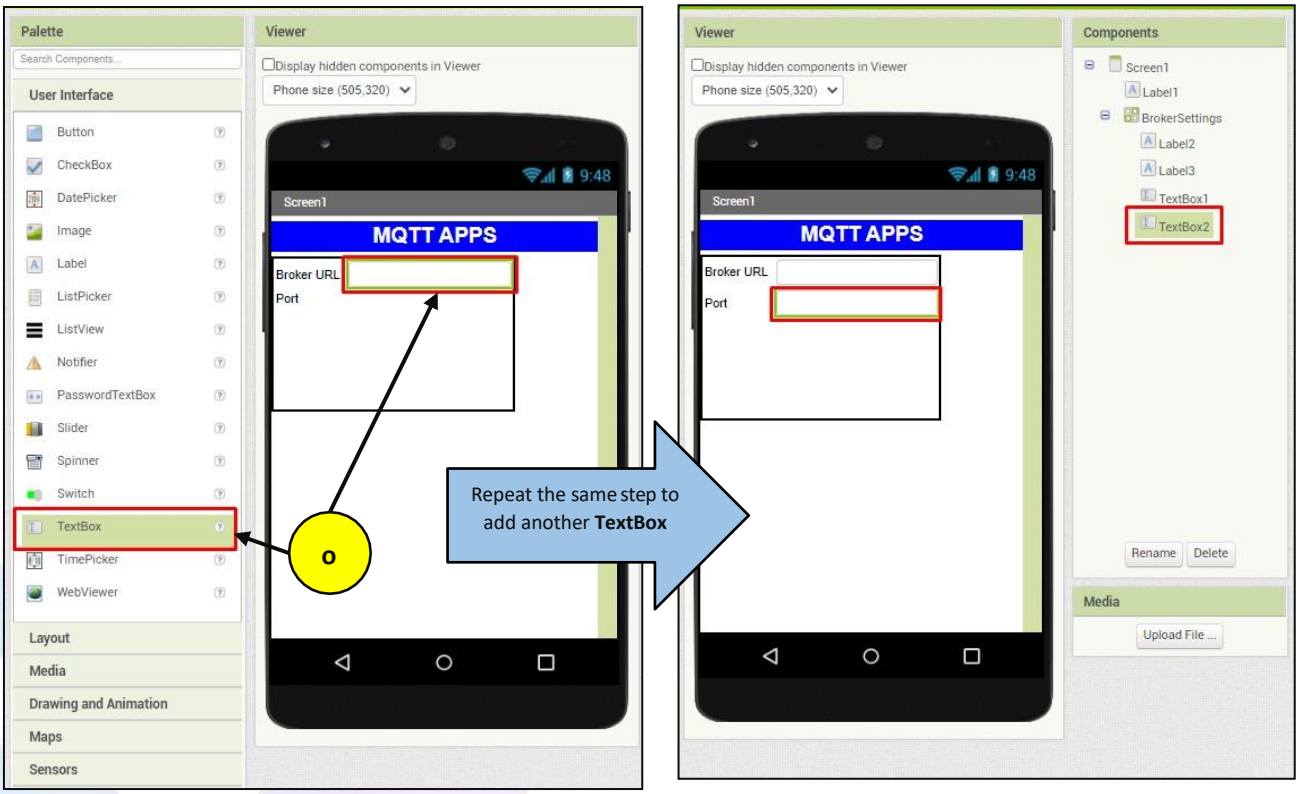


n. Add another **Label** underneath the Broker URL label (Drag and drop it on the ***second row, first***

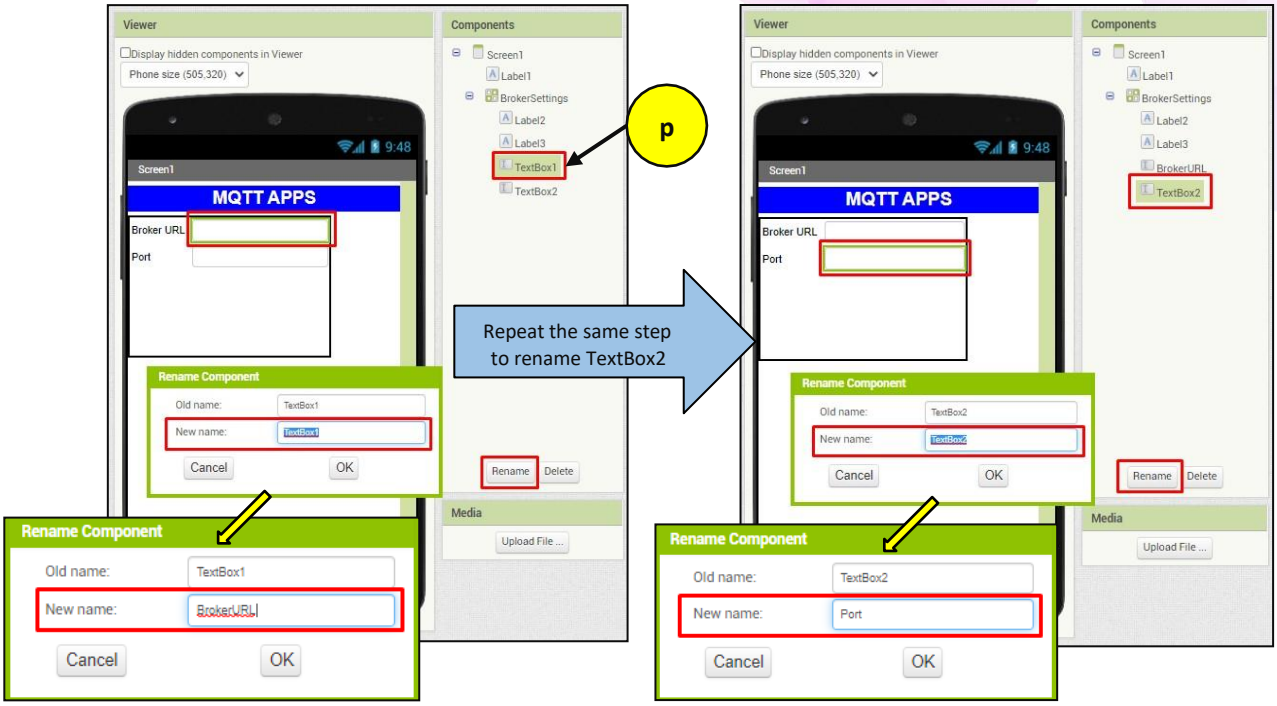
column). Also, change the Text from Text for Label3 to Port. (*see figure below*)



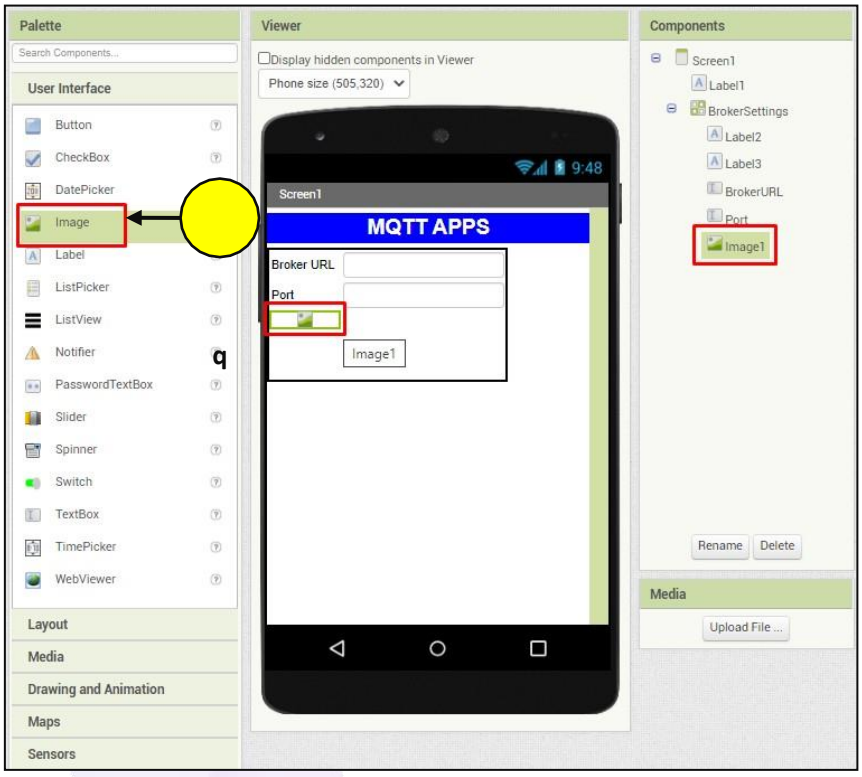
o. Next, add a TextBox at the column next to the Broker URL label. Repeat the same steps, except this time add it to the column next to the Port label.



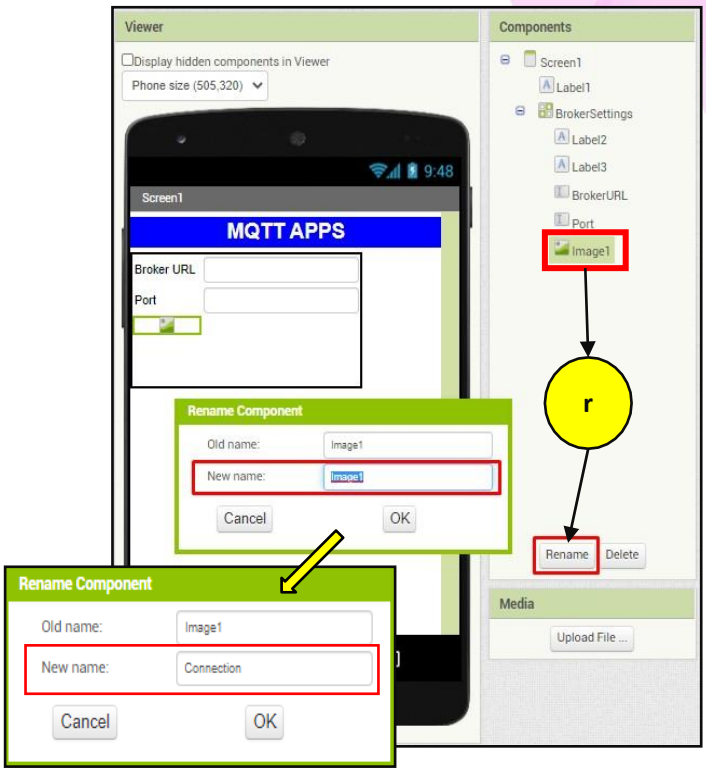
p. Rename the TextBox1 to BrokerURL. Repeat the same step for TextBox2 underneath it except renaming will be change to Port.



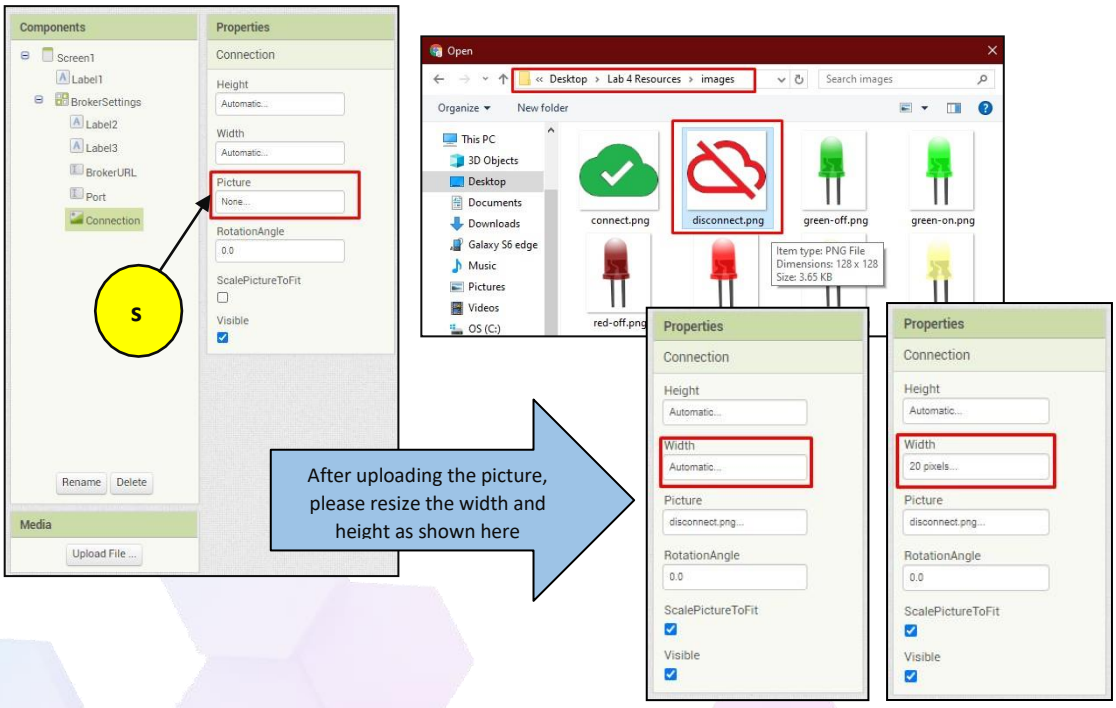
q. After that, on the third row of **BrokerSettings layout**, we will be adding an Image.



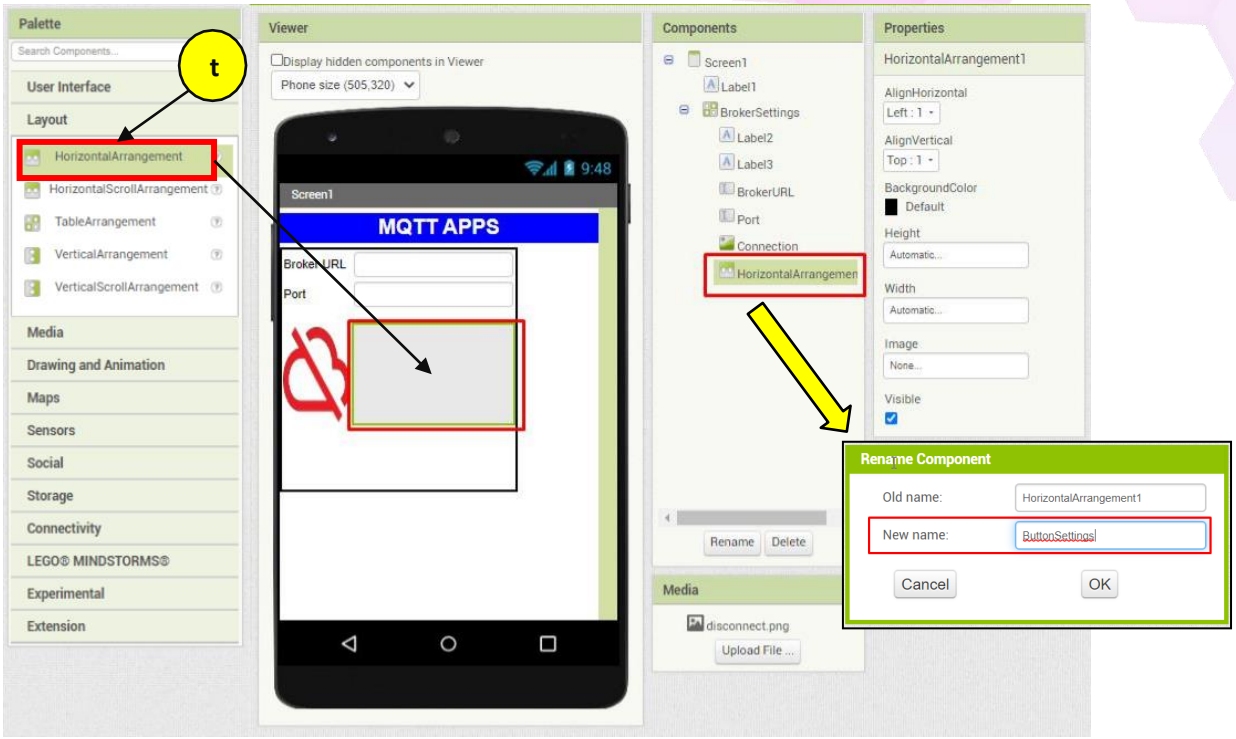
r. Then, rename Image1 at the Components to Connection



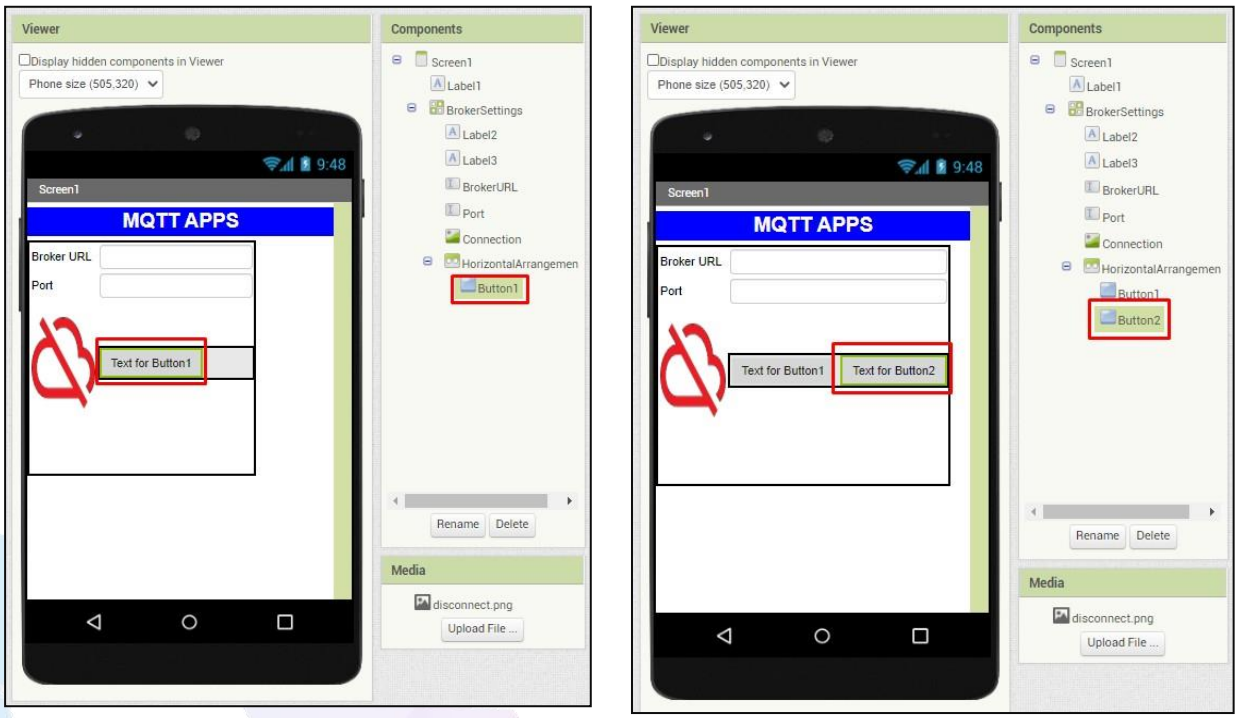
s. Next, add a Picture to the image. You will be given a folder named “Lab 4 Resources”. In the file, there are several images being provided to you. For this image, choose the disconnect.png file. Also, change the Width and Height of the picture as show in figure below.



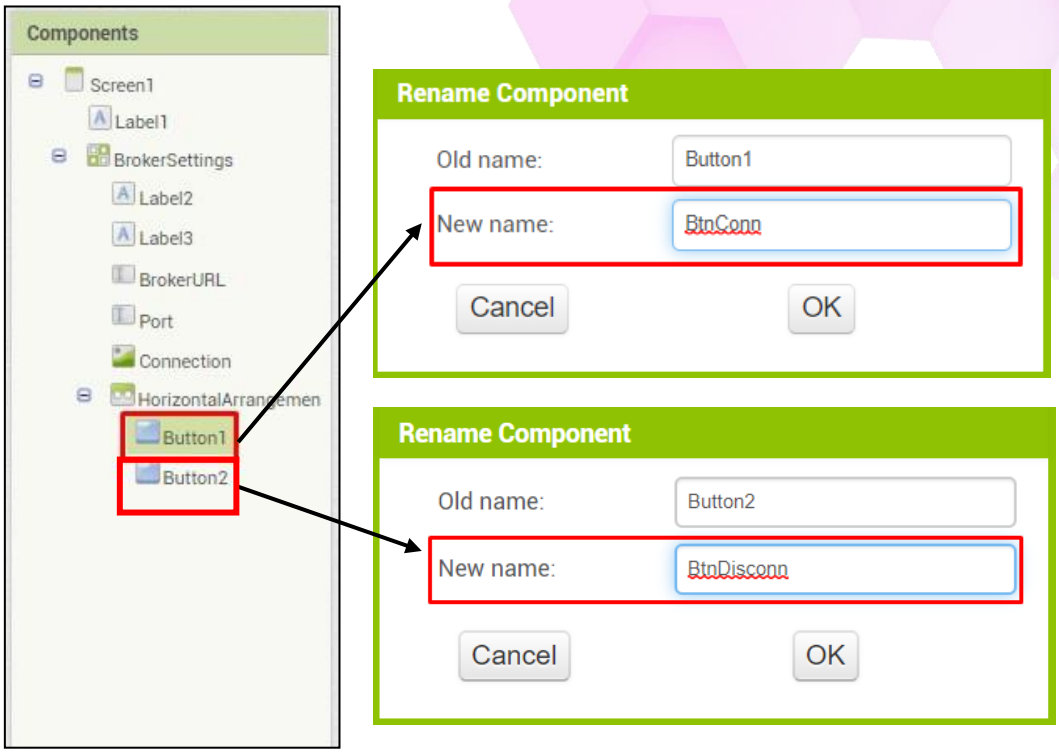
t. Next, we will add a HorizontalArrangement layout to the column next to the Connection Image. Also, we will be renaming it to **ButtonSettings** (*see figure below)*



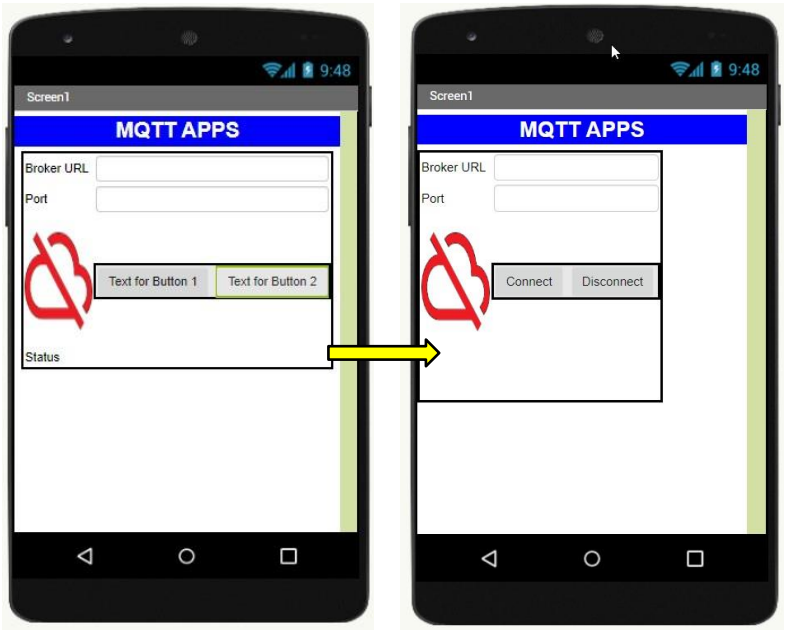
u. Then, in the **ButtonSettings**, we will be adding **two (2) buttons**



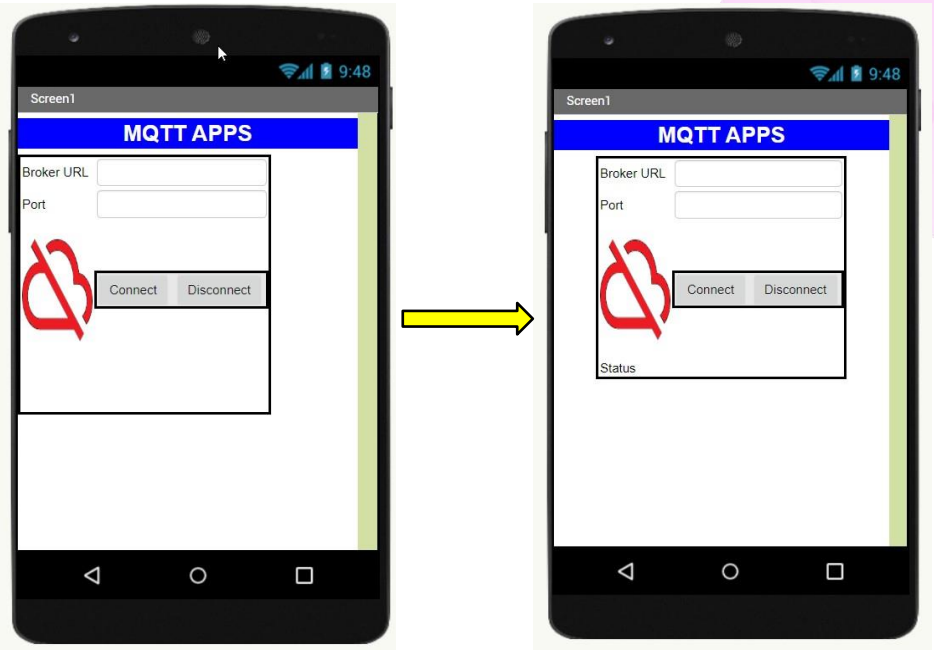
v. Rename each respective button as shown in figure below.

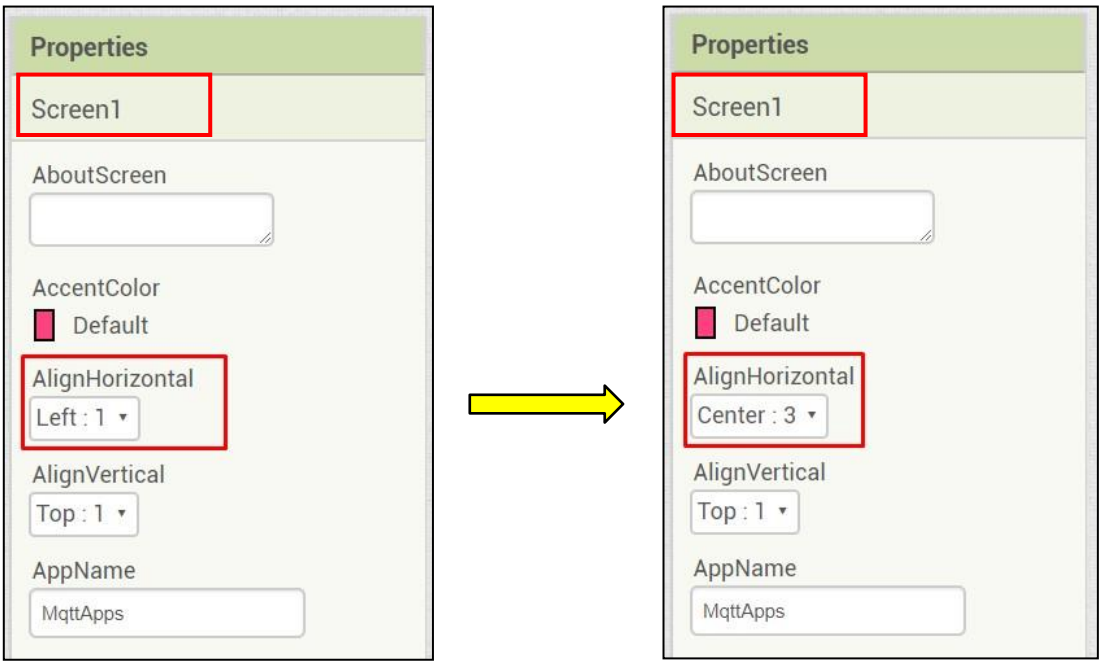


w. Also, we will be changing the Text on the both button as **Connect** and **Disconnect** respectively (*see figure below*)

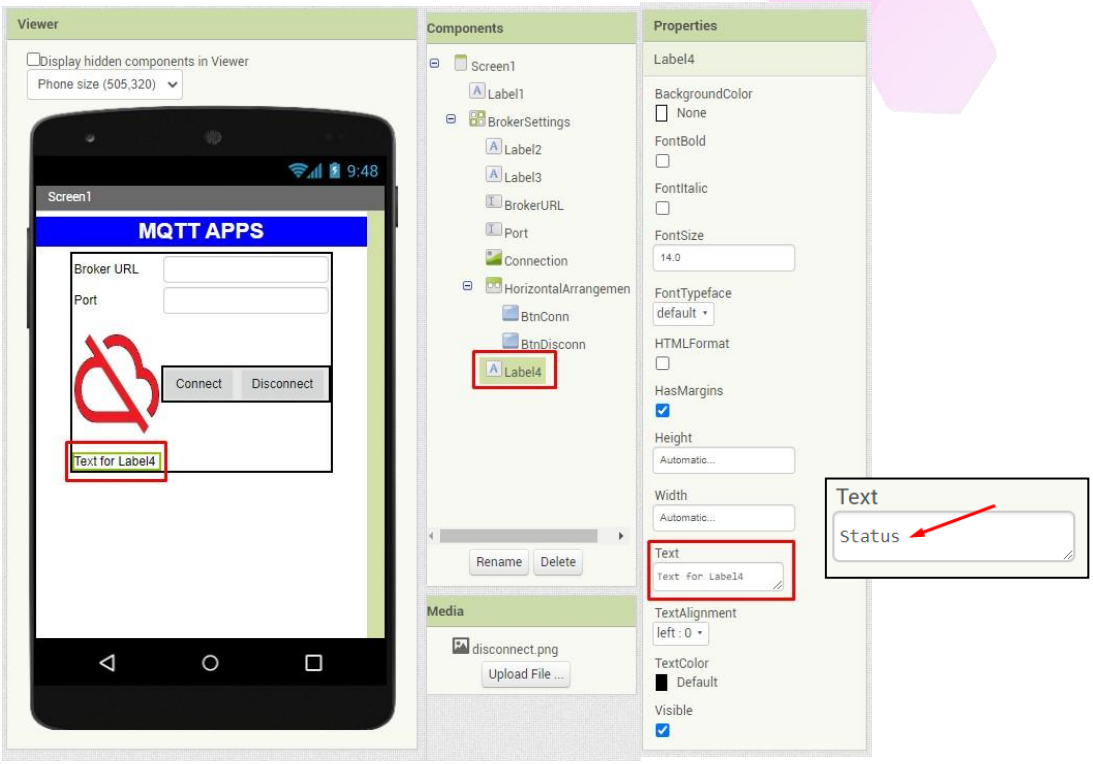


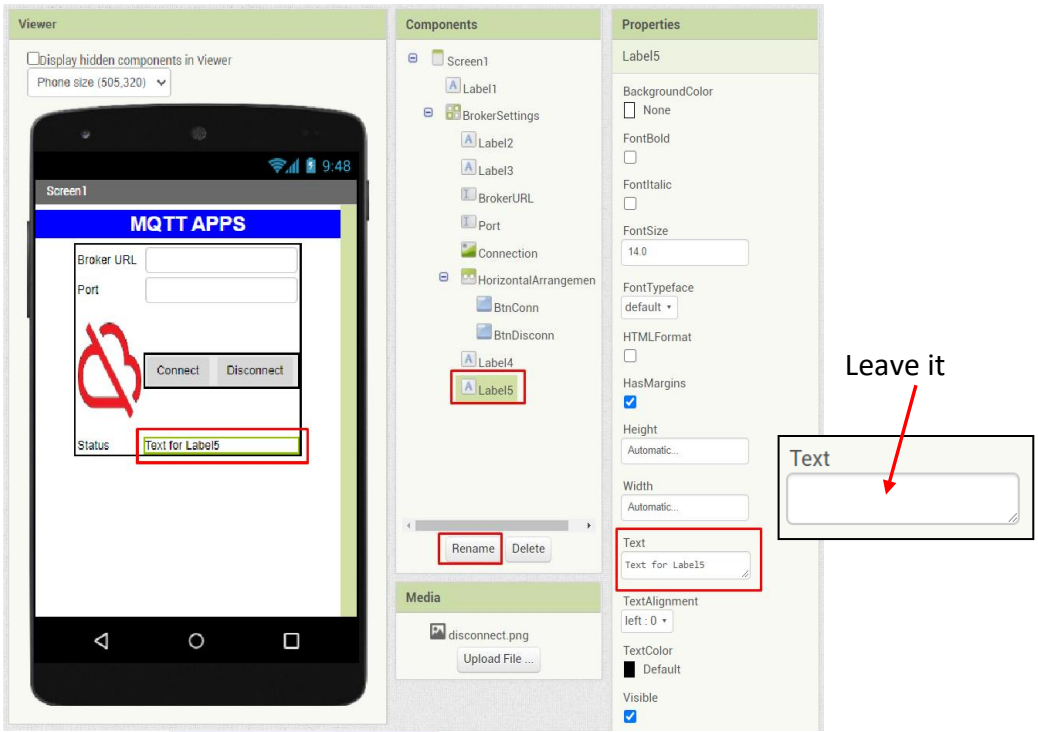
x. Next, we will align the **BrokerSettings layout** to the centre. To do this, simply change the alignment at the Component of Screen1 from **AlignHorizontal: Left** to **AlignHorizontal: Centre**


9:48
Screen1
MQTT APPS
Broker URL
Port
Connect
Disconnect
9:48
Screen1
MQTT APPS
Broker URL
Port
Connect
Disconnect
Status


Properties
Screen1
AboutScreen
AccentColor
Default
AlignHorizontal
Left : 1
AlignVertical
Top : 1
AppName
MqttApps
Properties
Screen1
AboutScreen
AccentColor
Default
AlignHorizontal
Center : 3
AlignVertical
Top : 1
AppName
MqttApps

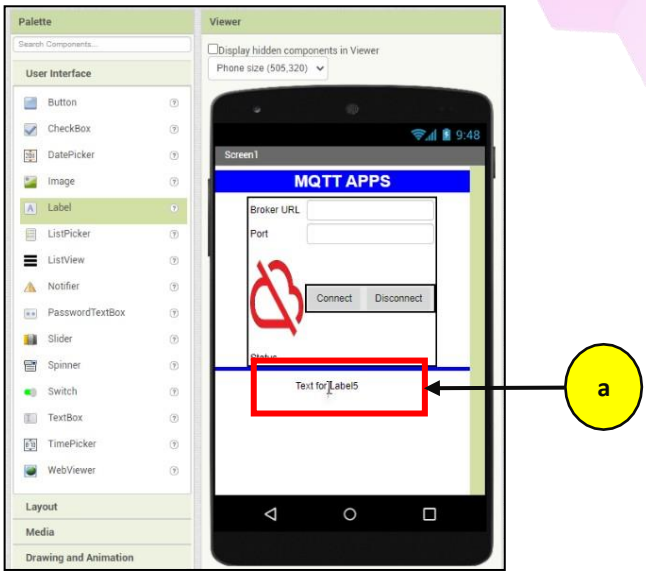
y. Lastly, we will add another **two (2) label** at the last row and column of the **BrokerSettings**. See figure below for configuration at the Properties for both label





[Step#03] Designing LED Status

a. Add a **Label** underneath the **BrokerSettings layout**.



b. Change the **BackgroundColor** to ***Magenta.***

c. ***Checked*** on the checkbox for **FontBold.**

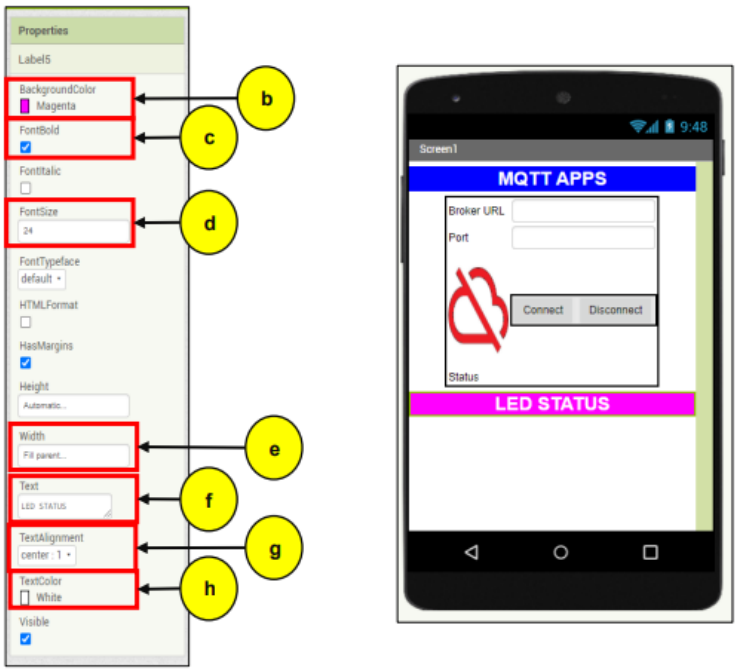
d. Then, change the **FontSize** to ***24.***

e. Resize the label **Width** to ***Fill parent.***

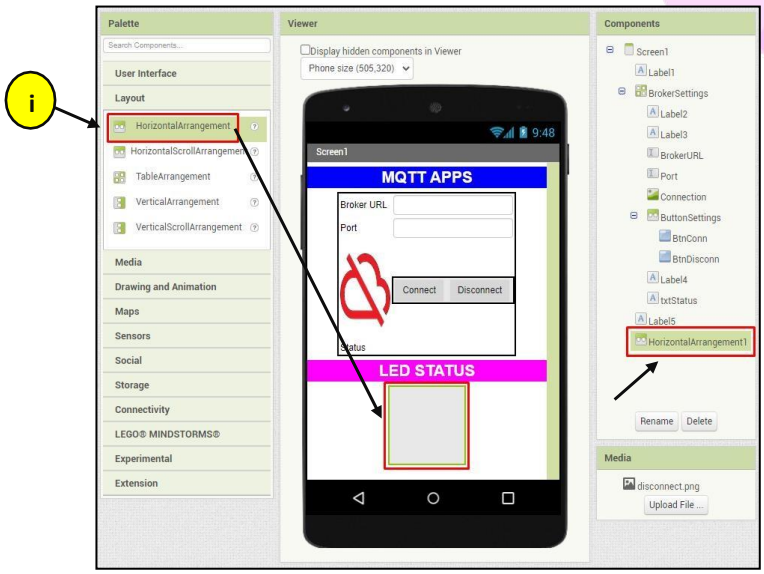
f. Change the **Text** to **"LED STATUS".**

g. Then, we are going to change the **TextAlignment** to ***centre: 1***

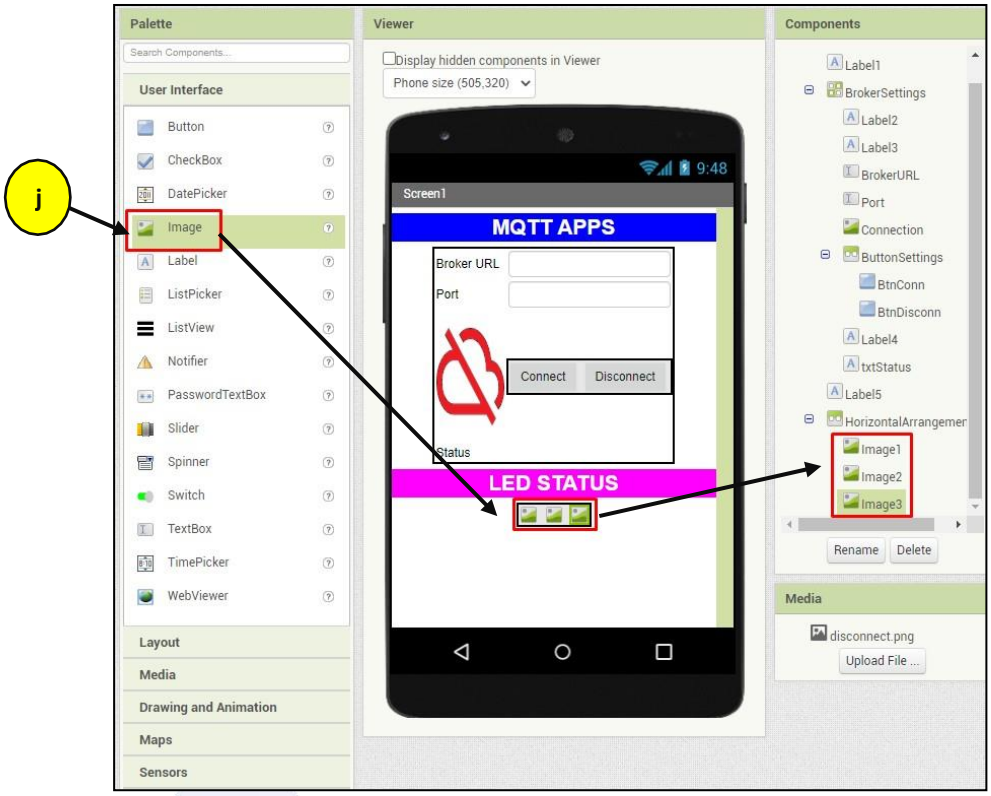
h. Lastly change the **TextColor** to ***White***.



i. Then, we will add a HorizontalArrangement Layout underneath the Label5



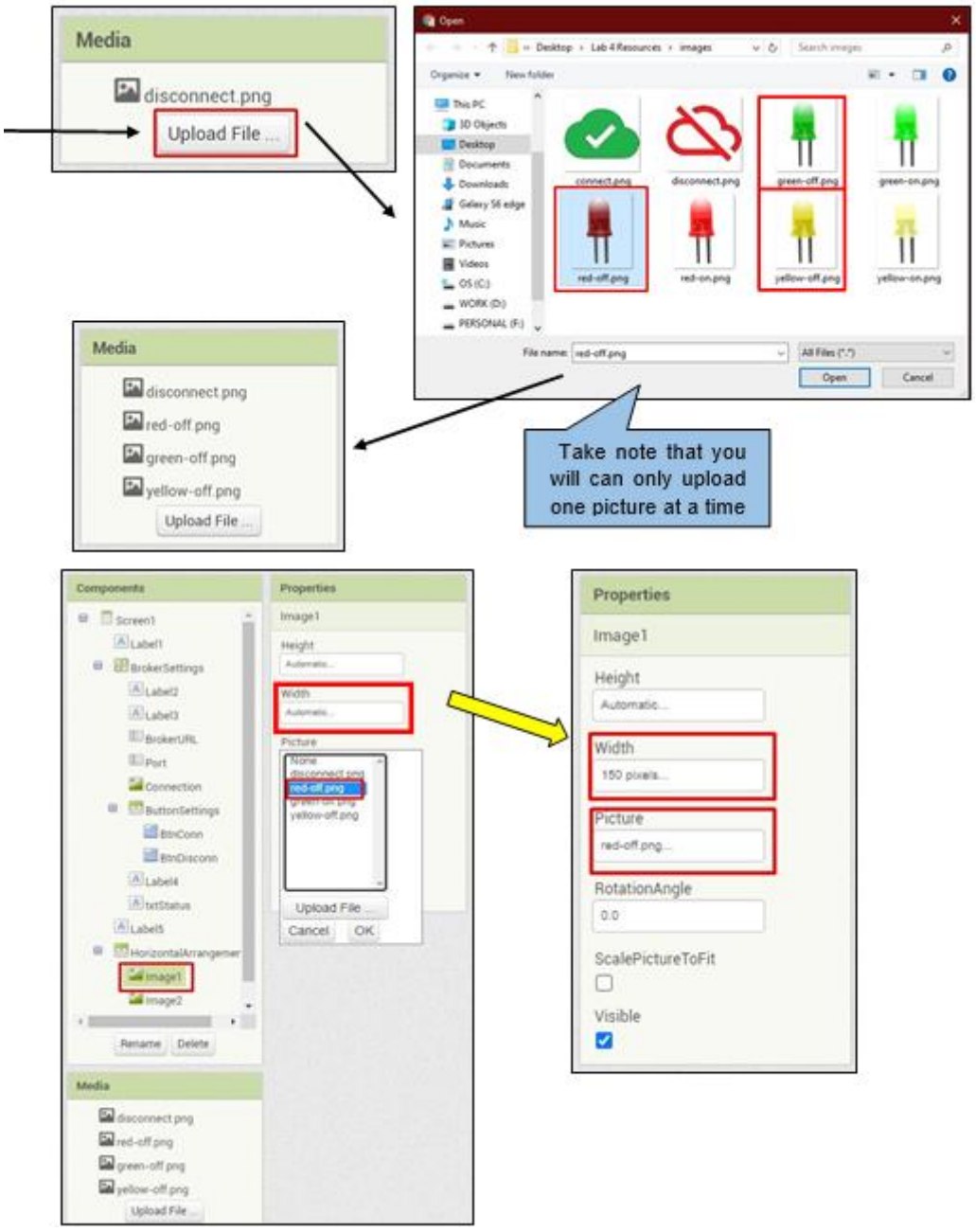
j. Next, add three (3) Image from the User Interface and add it into the **HorizontalArrangement1** as shown in figure below.



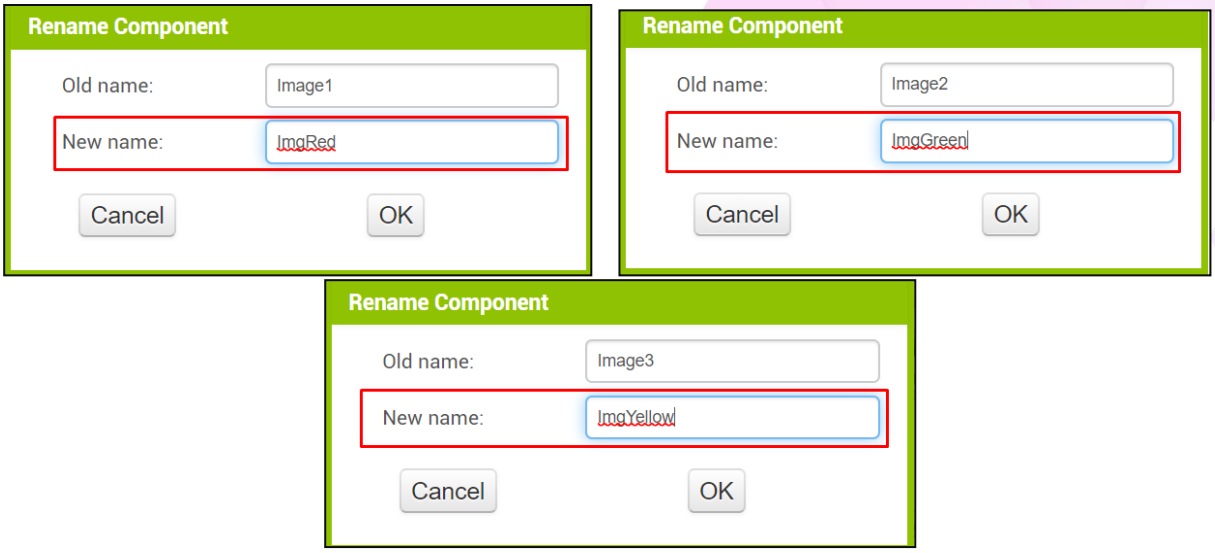
k. We will be adding source of Pictures for each of the images. We will be uploading 3 pictures from the Lab 4 Resources which are red-off.png, green- off.png and yellow-off.png.

Assign the pictures sources as follows:
Image1: ***red-off.png***
Image2: ***green-off.png***
Image3: ***yellow-off.png***

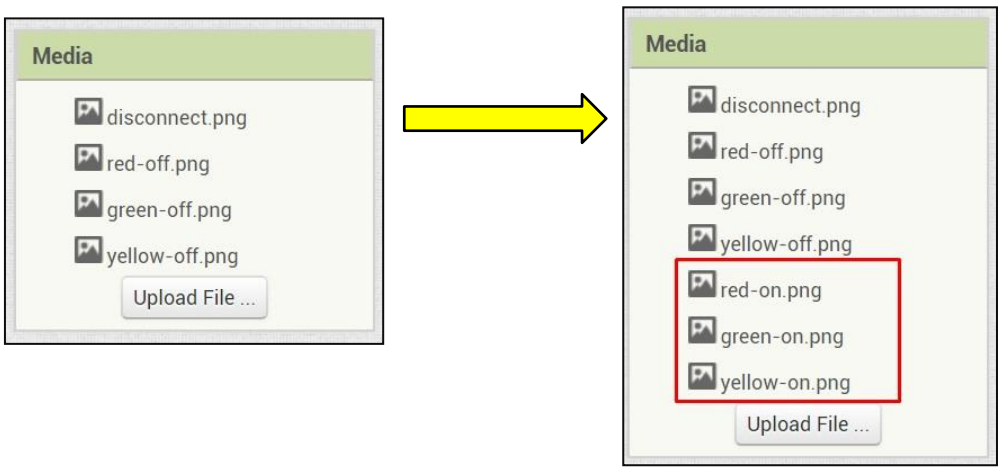
Also, we will need to resize the width of the pictures (see figure below)



l. Then, we will be renaming Image1, Image2, and Image3 as shown in the figure below:



m. Lastly we will be uploading the remaining three (3) images which are red- on.png, green-on.png and yellow-on.png.

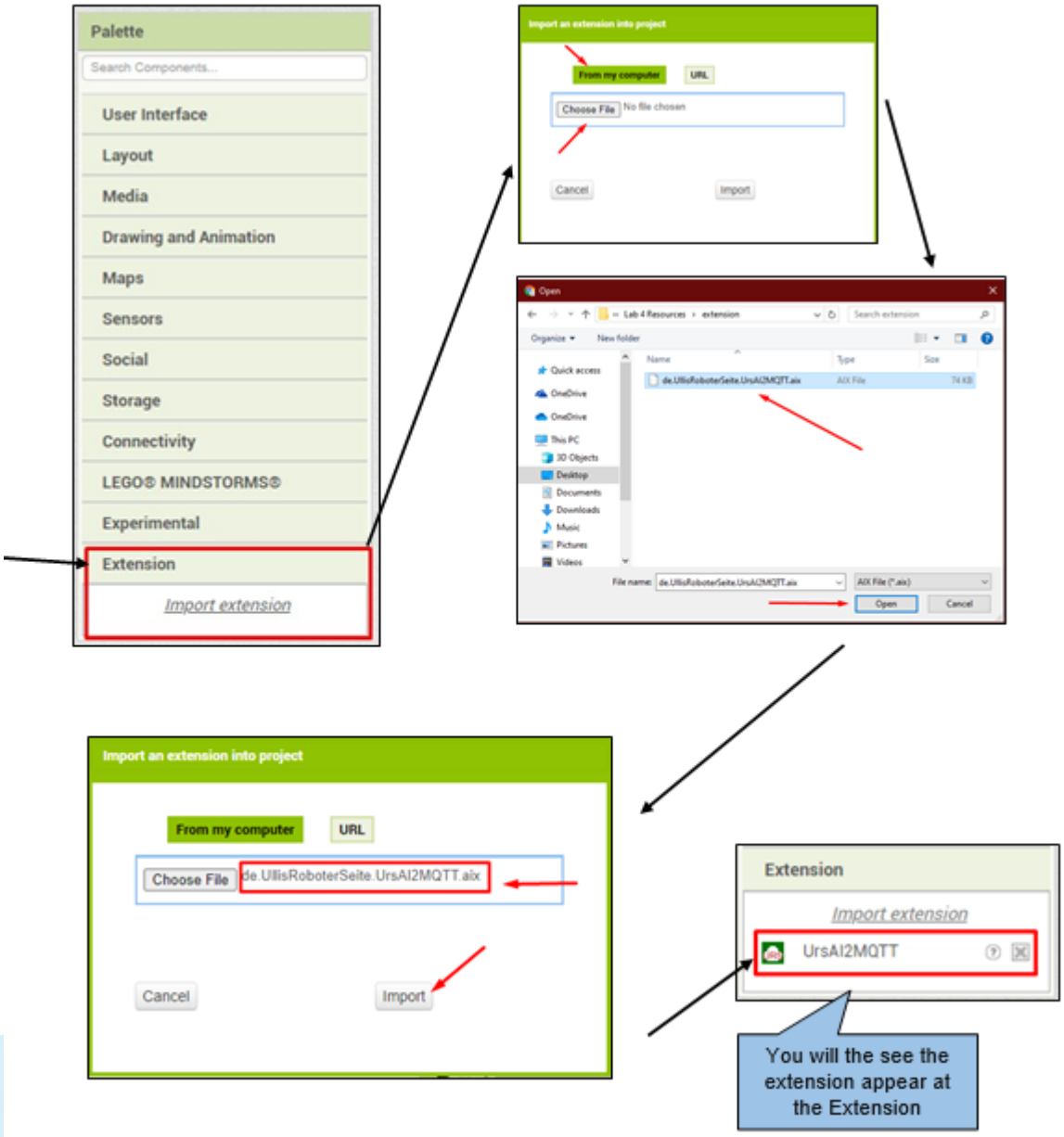


[Step#04] Coding the Connection Settings

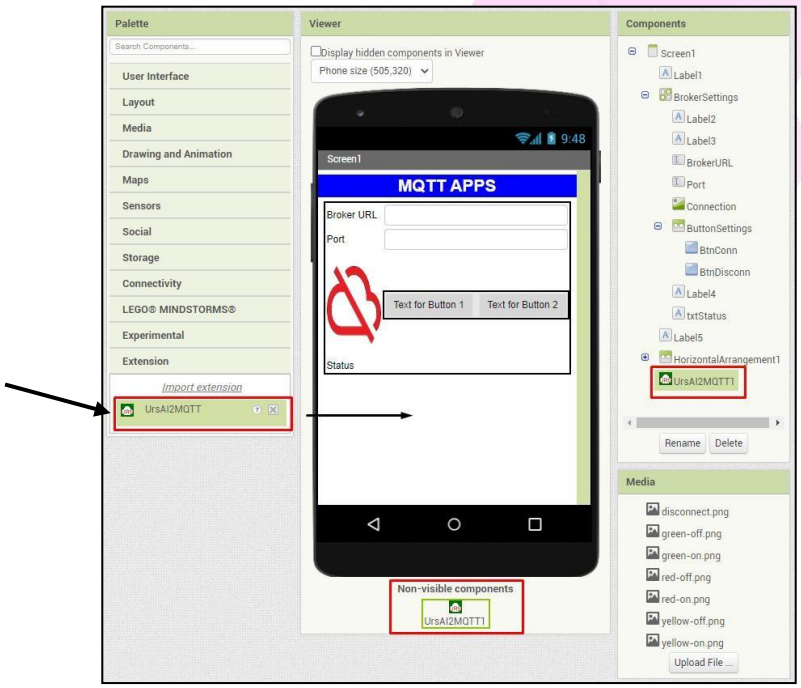
We will be using an extension from https://ullisroboterseite.de/android-AI2-MQTT- en.html . The extension will be provided in the Lab 4 Resources. We will need to upload the extension to the MIT App Inventor2 server.

Based on the guide from https://ullisroboterseite.de/android-AI2-MQTT-en.html

a. First, import the extension into the Palette. Go to Extension and click on *Import extension.* Choose the extension in the Lab 4 Resources.

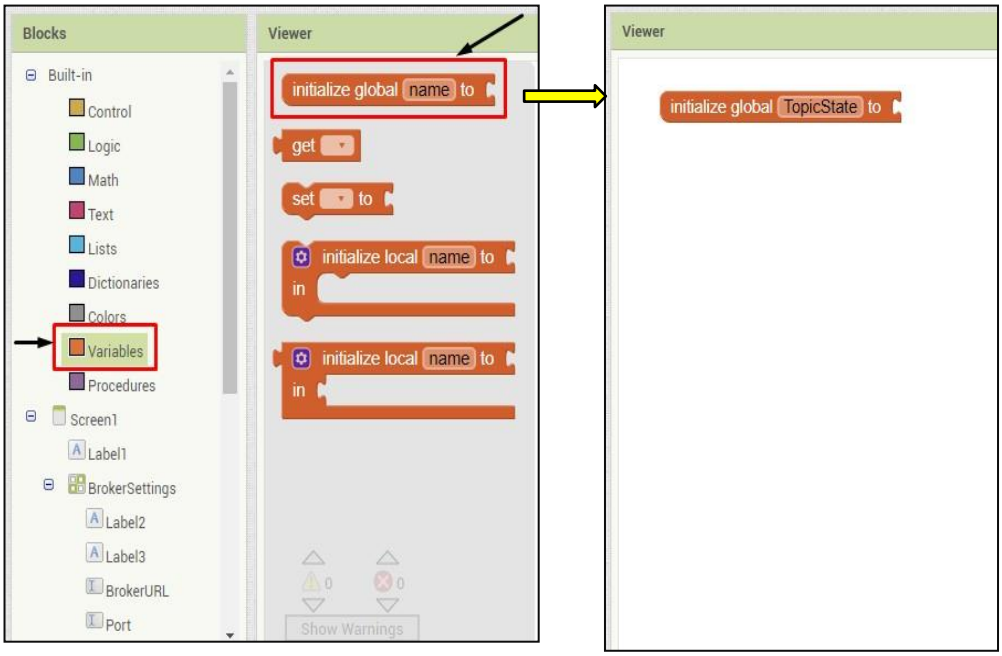


b. Then, drag and drop the extension to Screen1. As you can see the extension is a Non-visible components. Meaning, the extension does not appear on the screen.

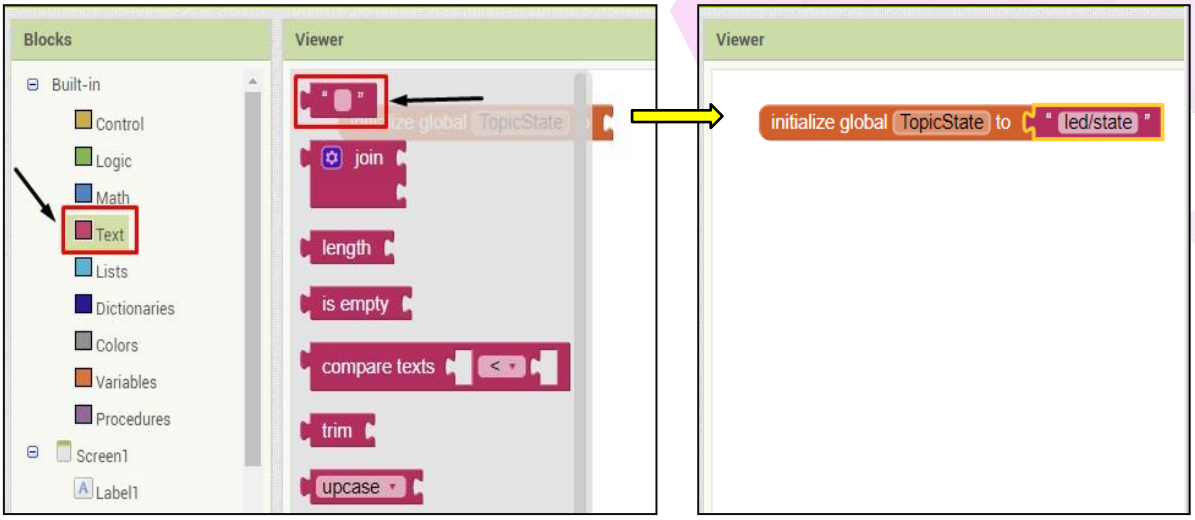


c. Now, switch over to the MIT App Inventor Blocks

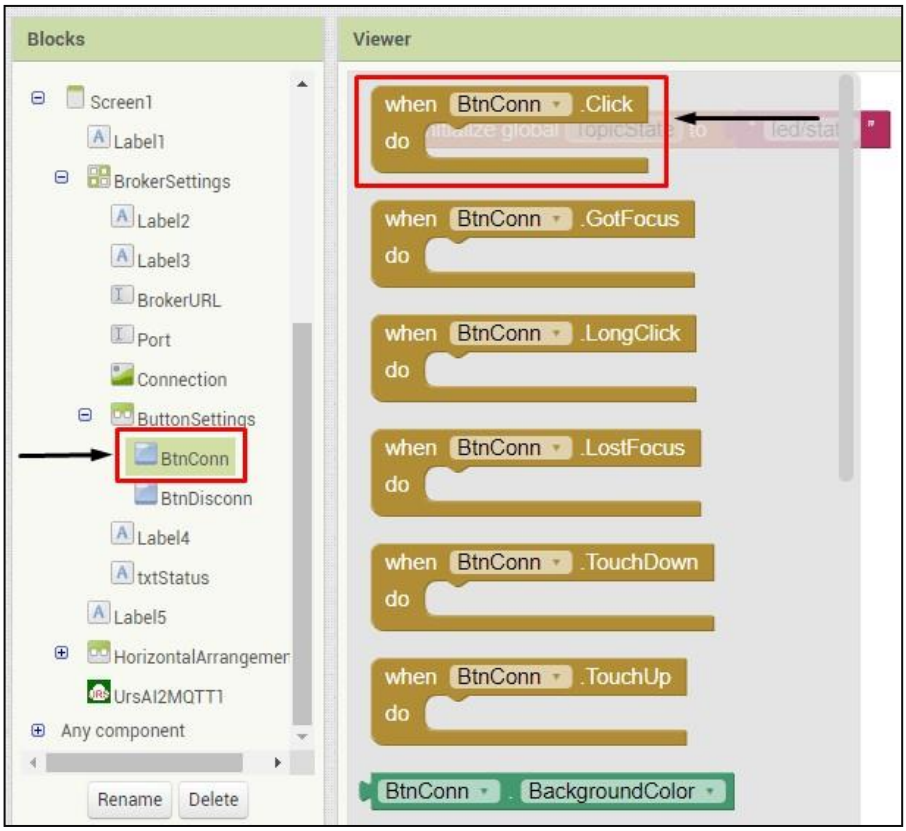
d. Then, at the Built-in blocks, select the ***Variable*** blocks. Choose the ***initialize global name to*** block and drag and drop it onto the Viewer. Change the name to ***TopicState***.



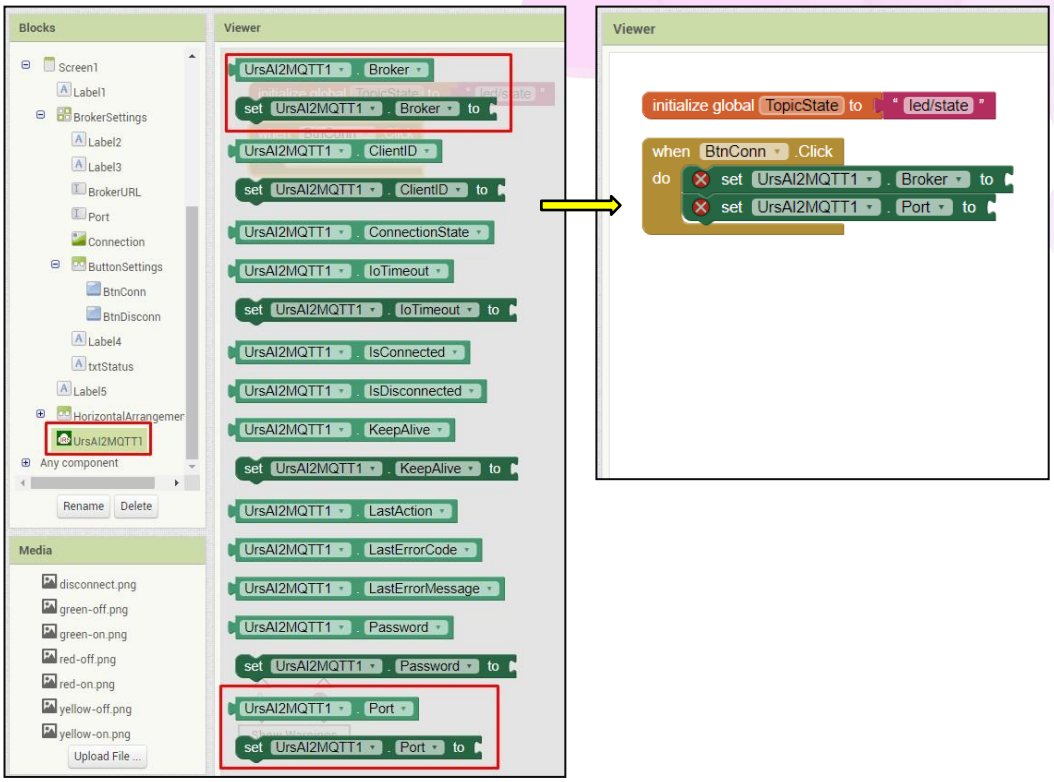
e. After that, go to ***Text*** blocks and choose the ***String*** block. In the String block typed in ***"led/state"*** and attached it to the ***initialize global block*** that we add previously.



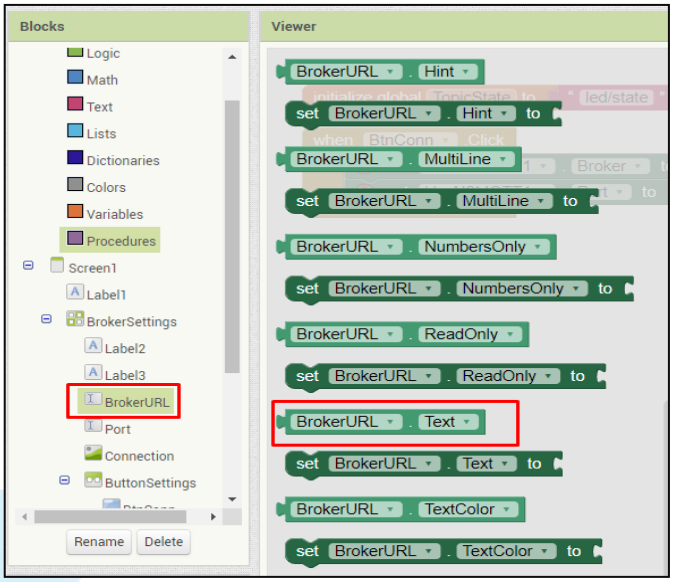
f. Next, go to the ***Components*** block and select the BtnConn component. Choose the ***when BtnConn.Click block*** and drag then drop it to the Viewer

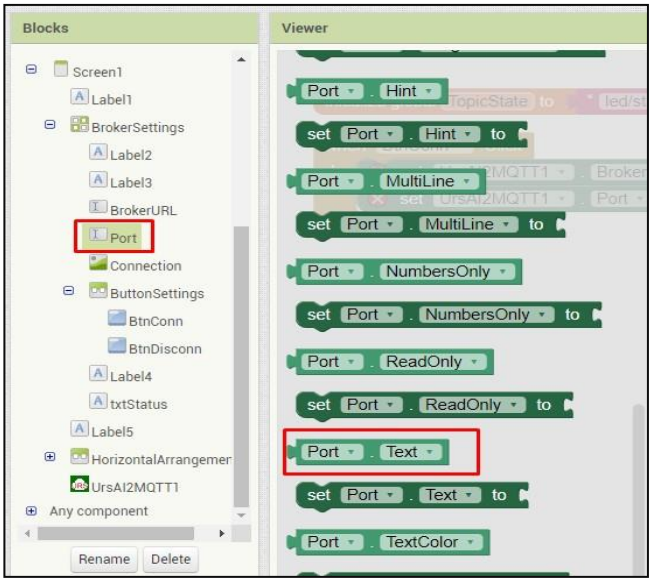


g. Go to the ***UrsAI2MQTT1 Extension*** and select the ***set UrsAI2MQTT1.Broker block*** and the ***set UrsAI2MQTT1.Port block***. Then, arrange these blocks to append under the ***when BtnConn.Click block***

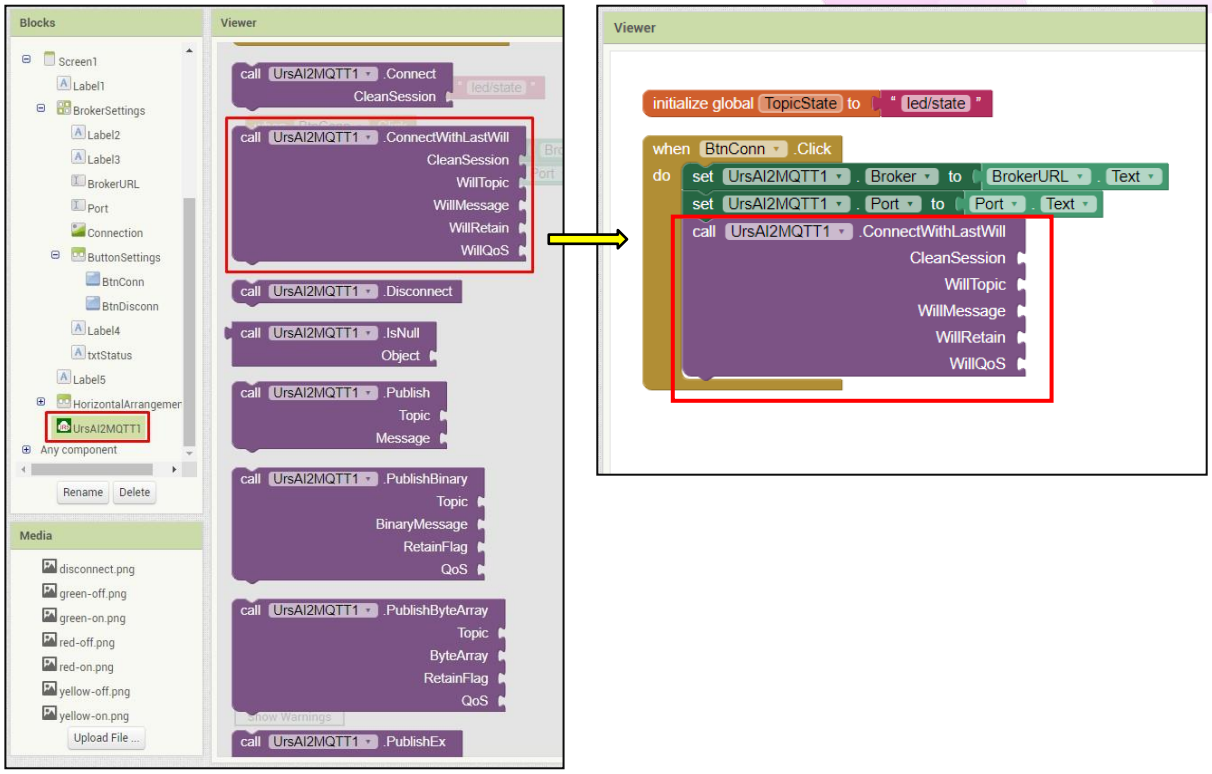


h. Then, go to the ***Component block*** and select the ***TextBox BrokerURL*** and ***TextBox Port***. Choose the ***block BrokerURL.Text*** and ***Port.Text*** and append it at the blocks we set in the previous step.

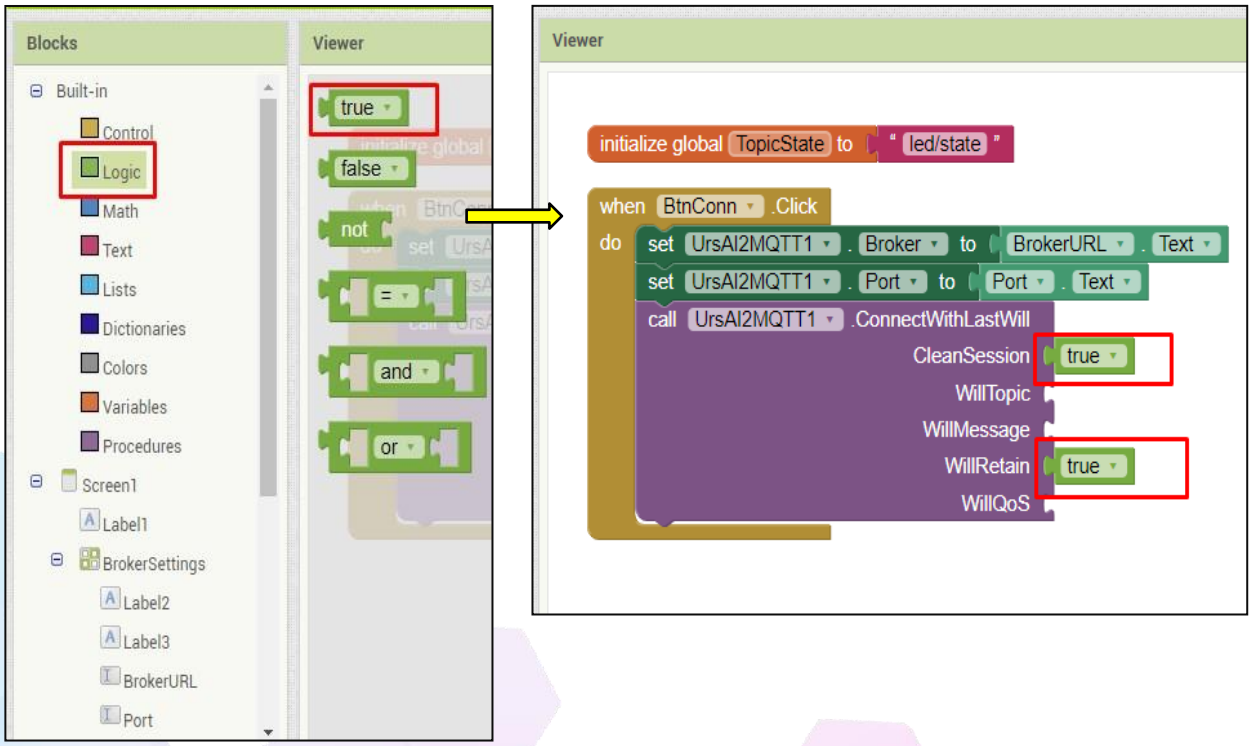




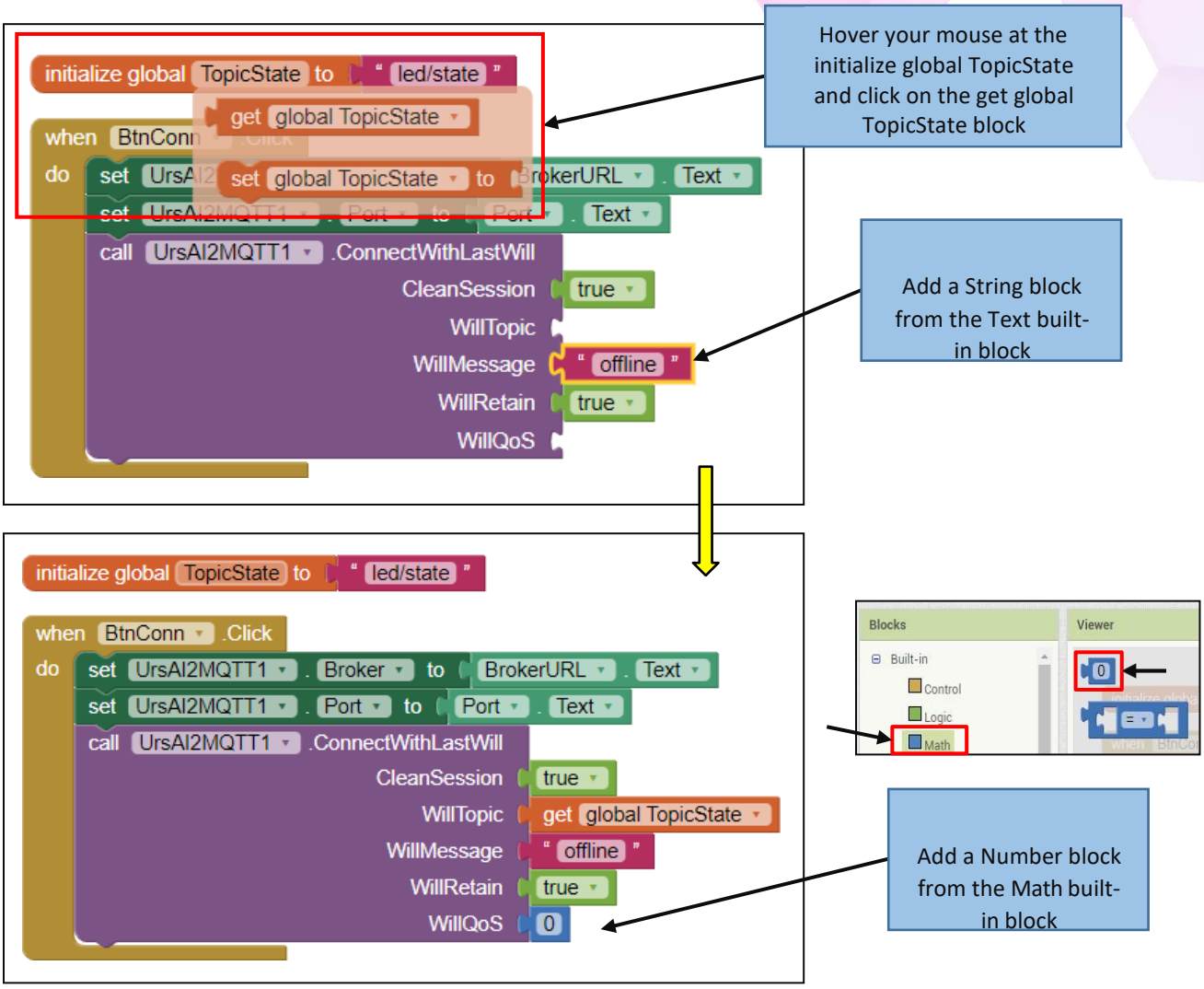
i. After that, go to the ***UrsAI2MQTT1*** extension component and select the ***call UrsAI2MQTT1.ConnectWithLastWill*** block. Append it under the Broker and Port settings.



j. Then, go to ***Logic built-in block*** and select the ***Boolean true block*** and append it at the ***call UrsAI2MQTT1.ConnectWithLastWill*** block as shown in the figure below.

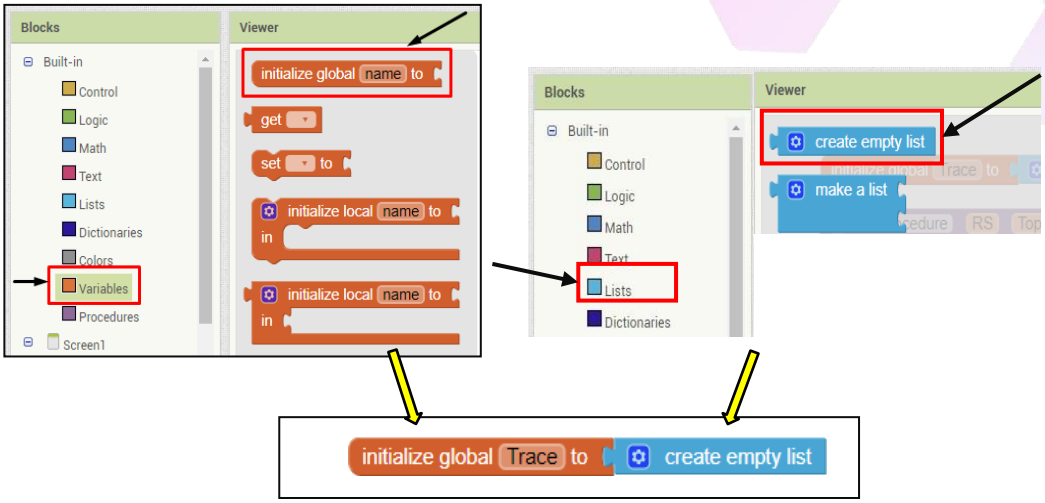


k. Now, we will configure the ***UrsAI2MQTT1.ConnectWithLastWill*** as shown in the figure below.

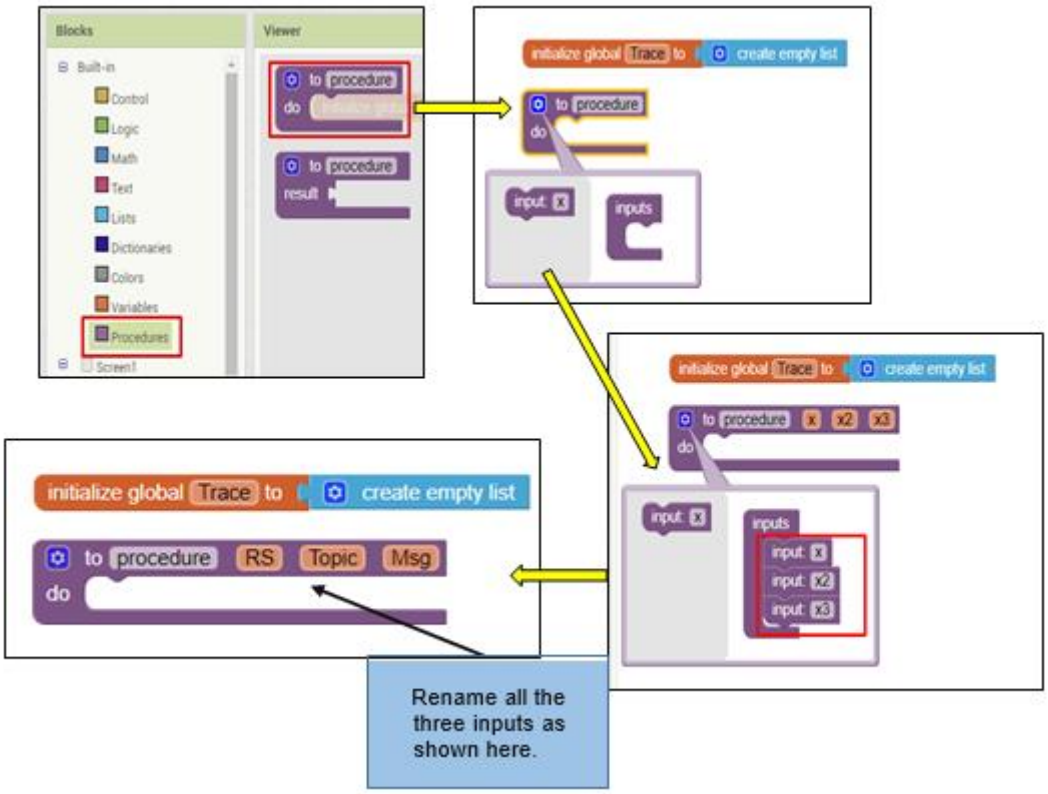


[Step#05] Coding the Topic Settings

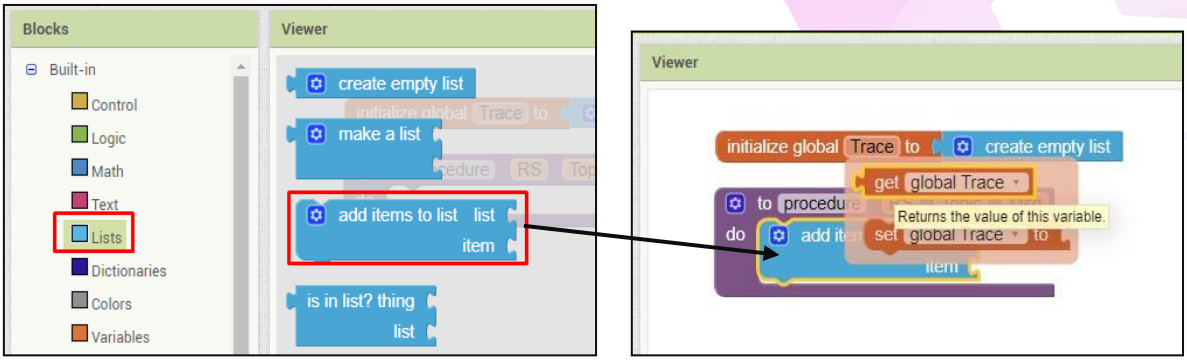
a. Then, go to Variables built-in block and select ***initialize global name to*** block. Next, select create empty list block from the Lists built-in block.



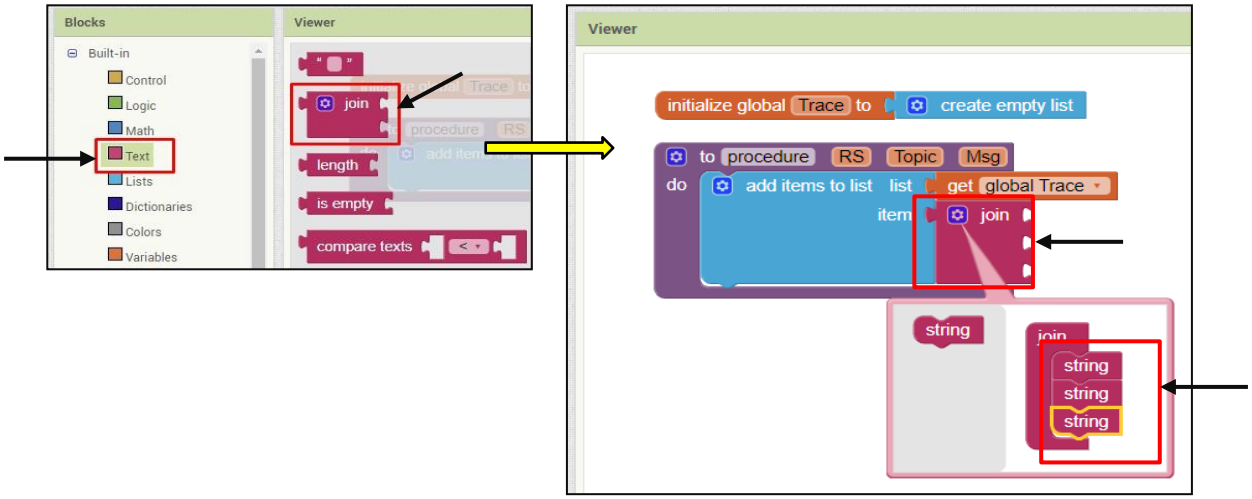
b. Next, go to the ***Procedures built-in blocks*** and the ***to procedure do block***. Drag and drop to the Viewer. The click at the Setting button at the ***to procedure do block.*** And add 3 inputs to the block as shown in figure below.



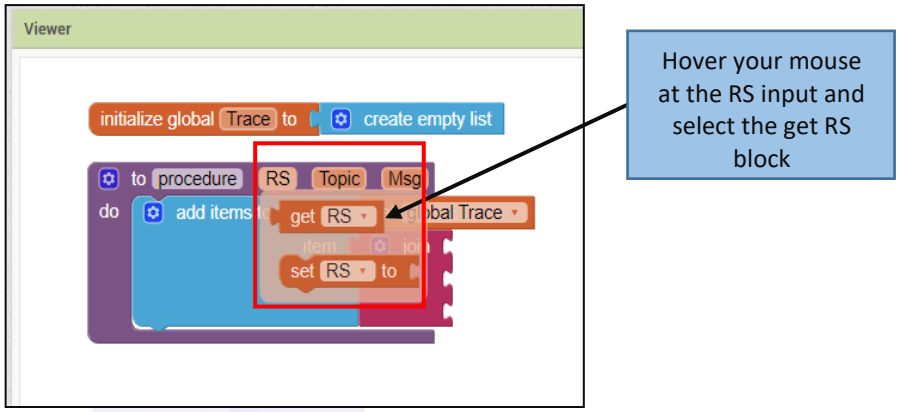
c. After that, go to the Lists block and select the add items to list block. And append in underneath the procedure block we add in the previous step.

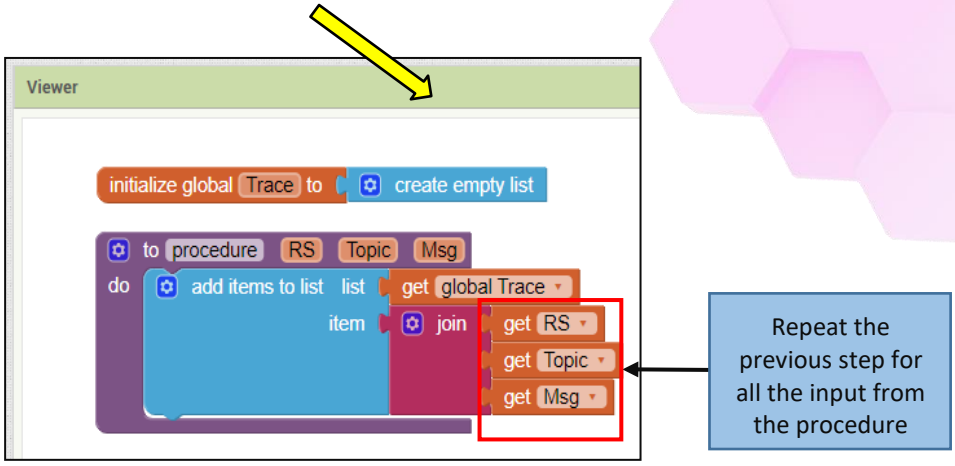


d. Go to the ***Text block*** and select the ***join block***. Append it to the ***add items to list block at the item***. Set the string at the join block to three string as shown in the figure below.

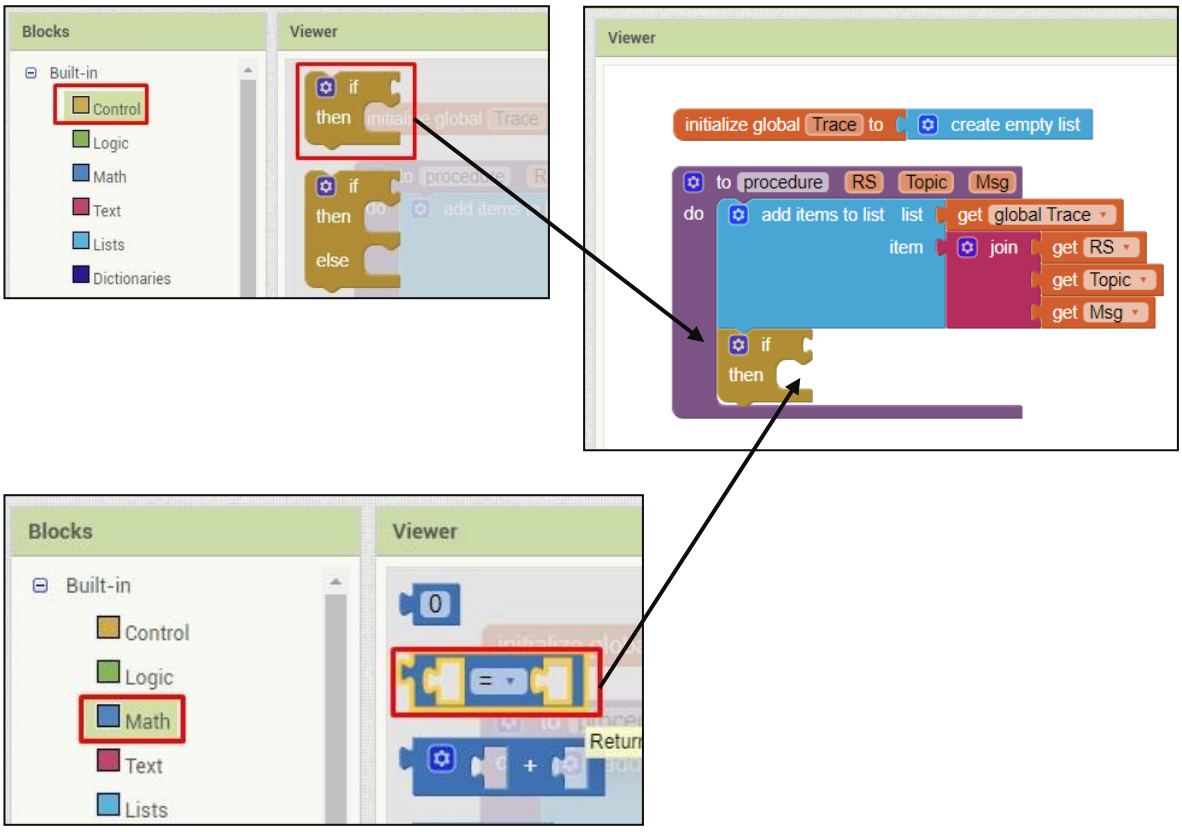


e. Then, we will need to set the input from the procedure to the join block as show in figure below.

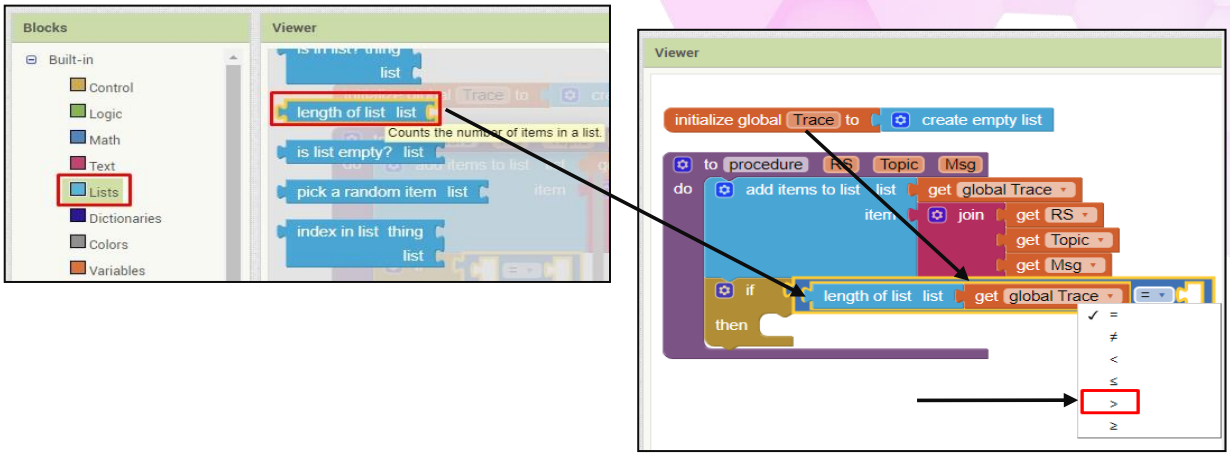




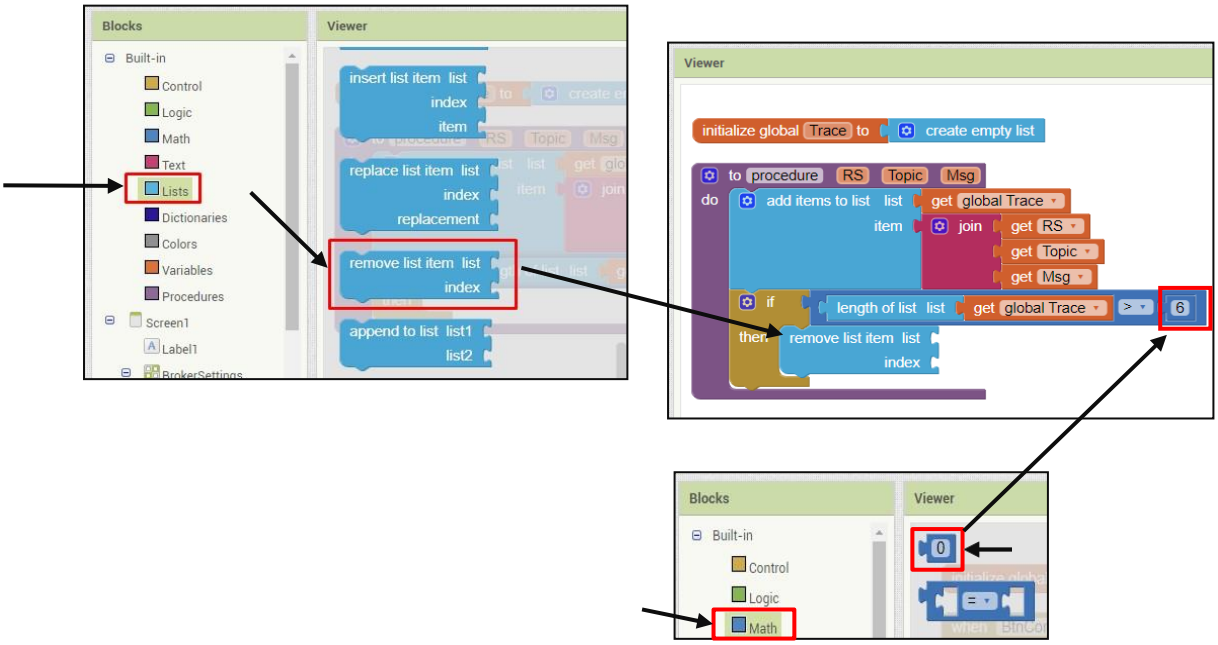
f. Next, go to the ***Control built-in block*** and choose the ***if then block***. Drag and drop it underneath the ***add items to list block.*** Then, go to the ***Math built-in block*** and select the ***comparator block*** as shown in the figure below and append it to the ***if then block***.



g. After that, add the go ***to Lists built-in block*** and select the ***length of list block*** to the ***comparator block*** as shown in figure below. Next to the length of list, append ***the get global Trace*** (hover to the initialize global Trace and choose get global Trace block. Make sure to choose the ***comparator value*** as shown the figure below.



h. Then, at the Lists built-in block, choose ***the remove list item block*** and attached under ***the comparator block***. Also, add a number to the comparator block as shown in figure below.

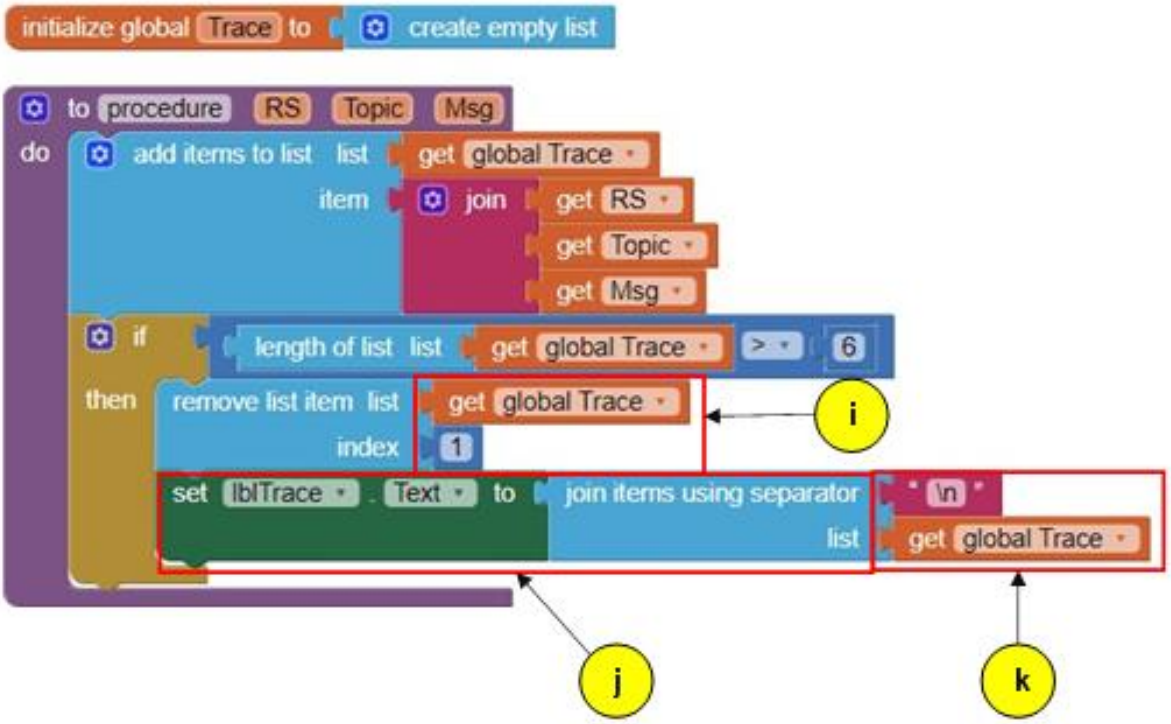


i. At the ***remove list item block***, append the ***get global Trace block*** at the list and append a ***number block*** at the index.

j. Next, add a ***set lblTrace.Text block*** underneath the ***remove list item block***. Then, append the ***join items using separator block*** to the ***set lblTrace.Text block.***

k. Lastly, append a String block with input "\n" to the the ***join items using separator block*** as well as the ***get global Trace block.***

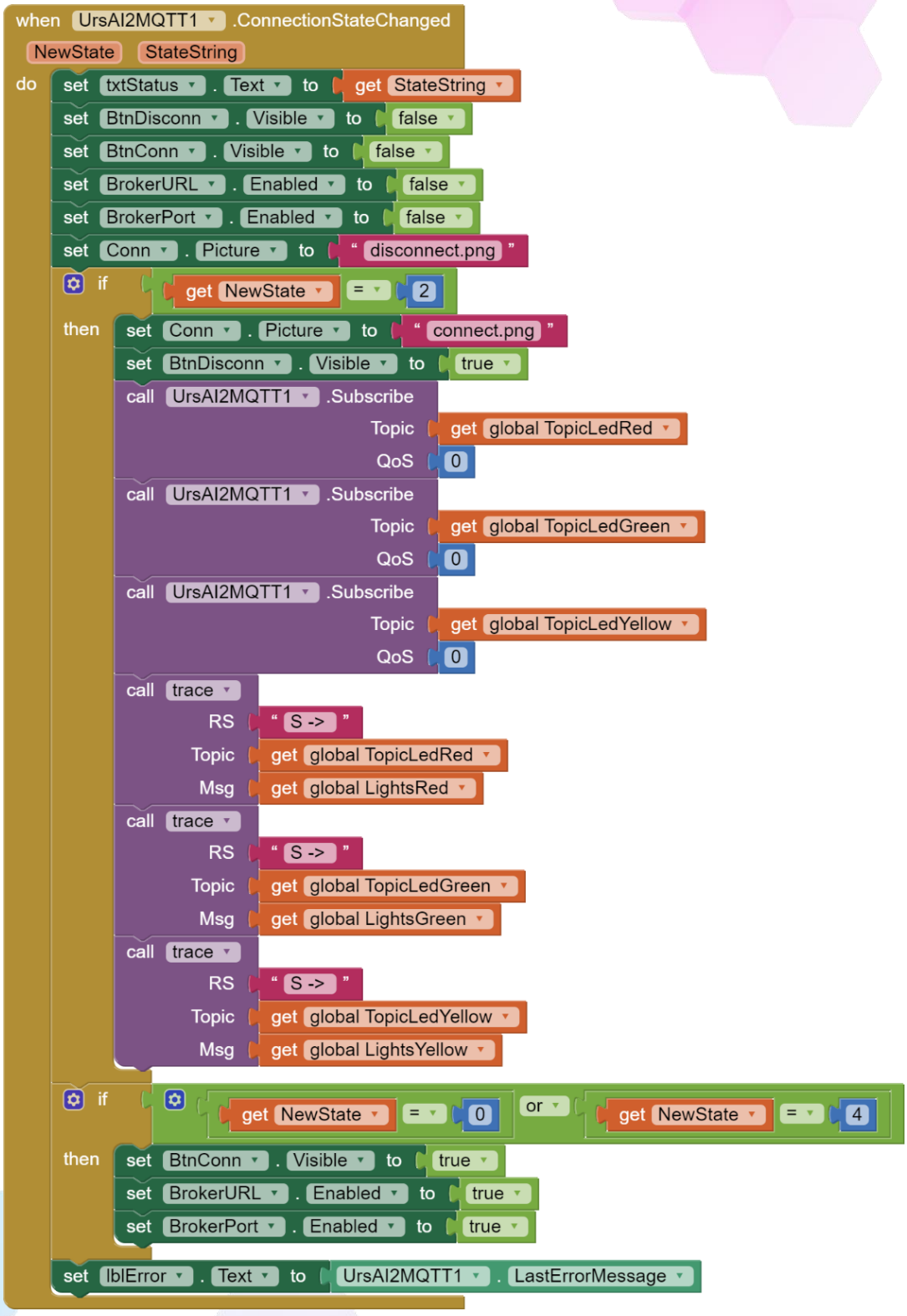
l. The end result should look something like the figure below



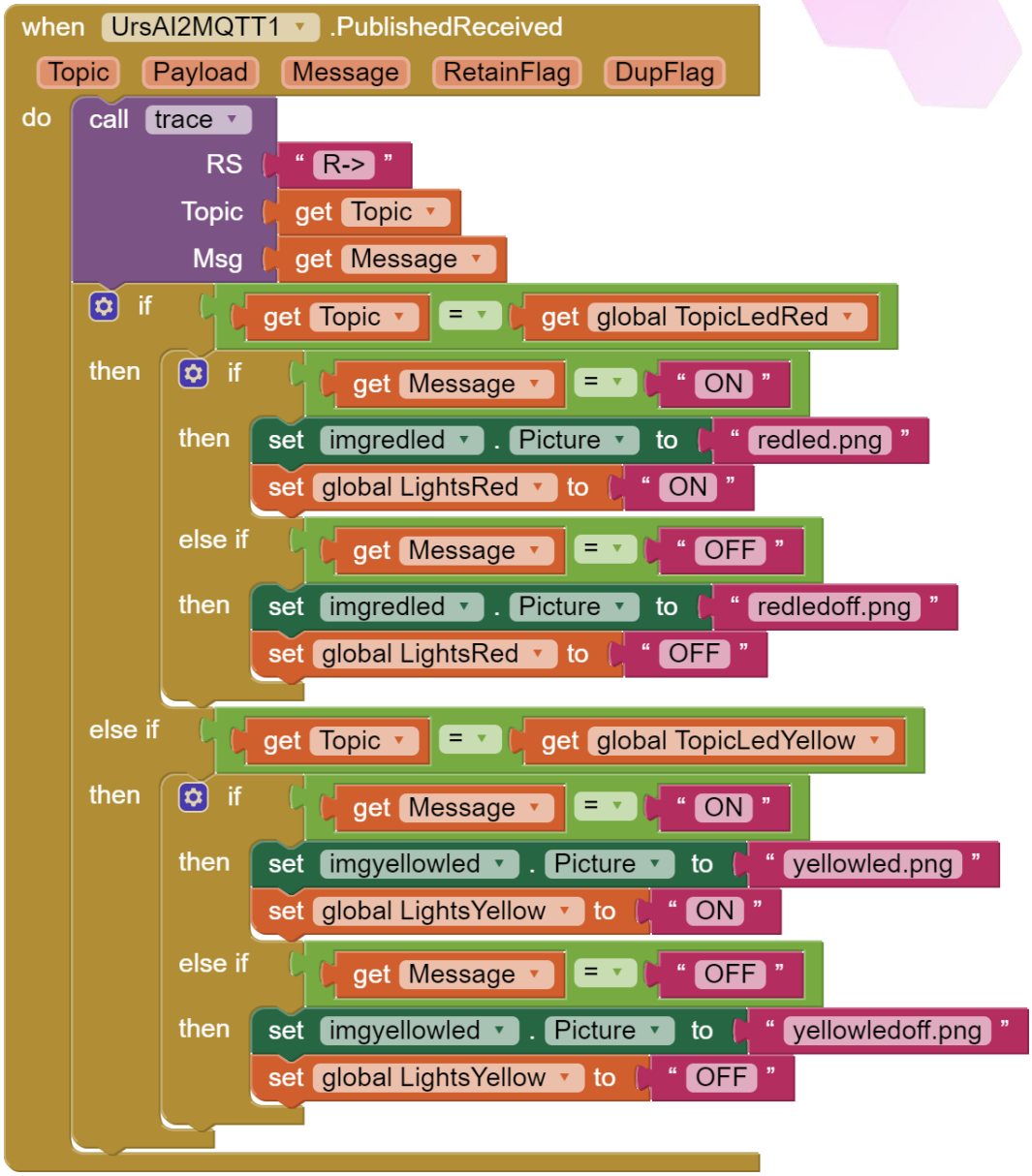
[Step#06] Coding the Connection Change Settings

a. Add a initialize global as follows:

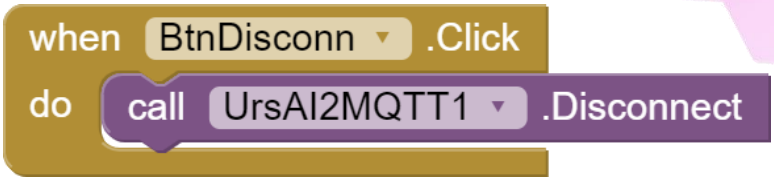
b. Then, add the following blocks.



[Step#07] Coding the Received Payload



[Step#08] Coding the Disconnect Button



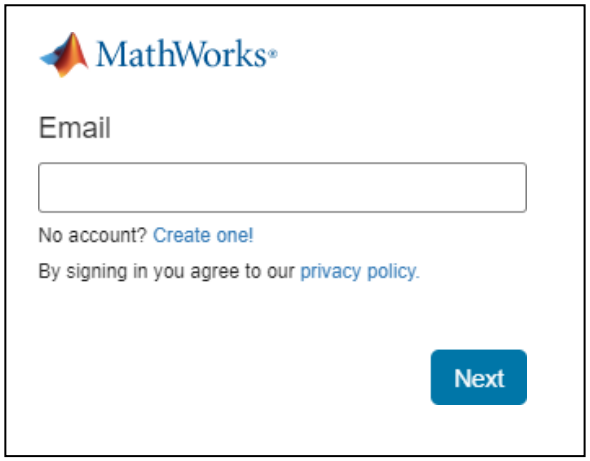
References:

1. http://appinventor.mit.edu/explore/sites/all/files/teachingappcreation/unit1/MagicTrick Handout.pdf
2. https://appinventor.mit.edu/explore/library
3. https://appinventor.mit.edu/explore/ai2/tutorials
4. https://www.programwithappinventor.org/
5. https://www.amazon.com/Learning-MIT-App-Inventor-Hands-On/dp/0133798631/

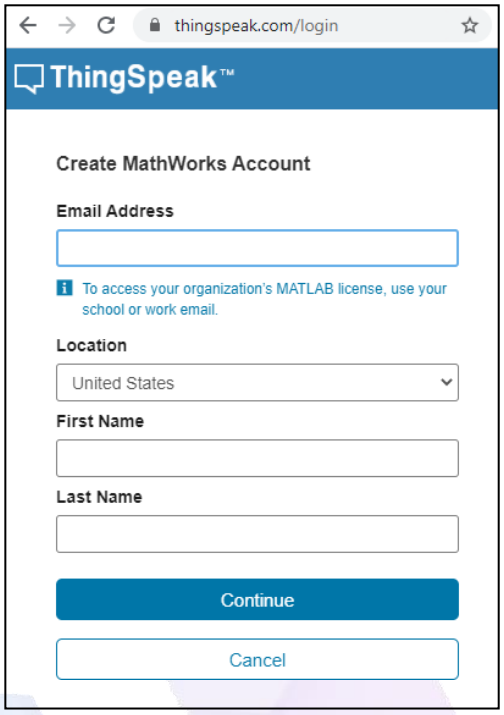
Module 5: Data Visualization using MIT App Inventor [3hrs]

Objective: In this lab we are going to go through steps by steps on creating a data visualization. We will be using a visualization chart from ThingSpeak and a MQTT Client Desktop to simulate data transfer. In this lab we will be utilising the MQTT Key instead of the API Key.

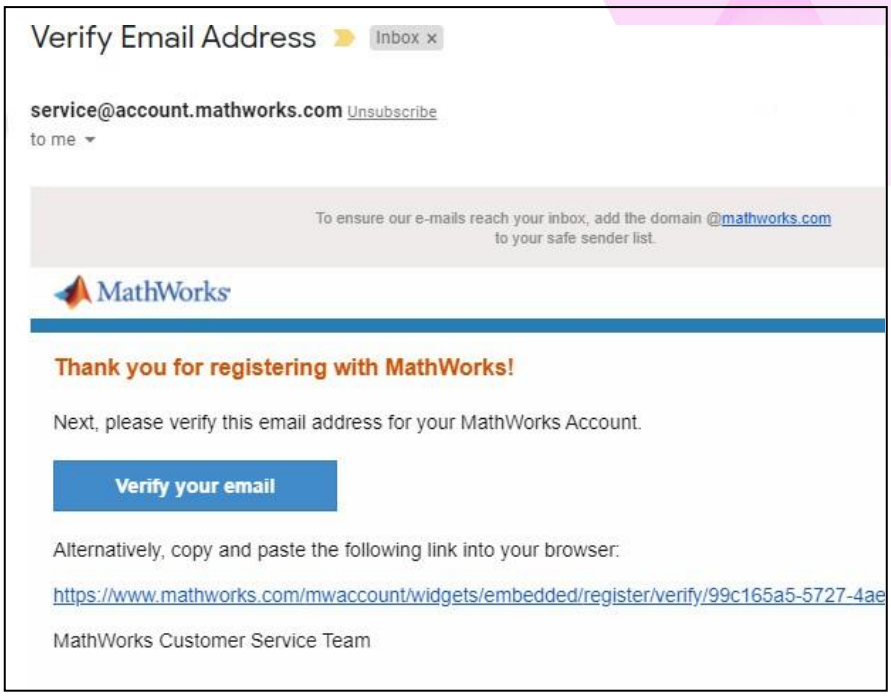
1. Create an account at ThingSpeak. Go to this link https://thingspeak.com/login



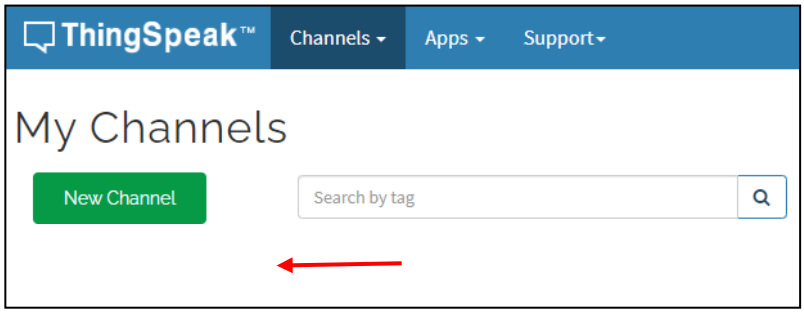
2. Fill in your details in the fields below



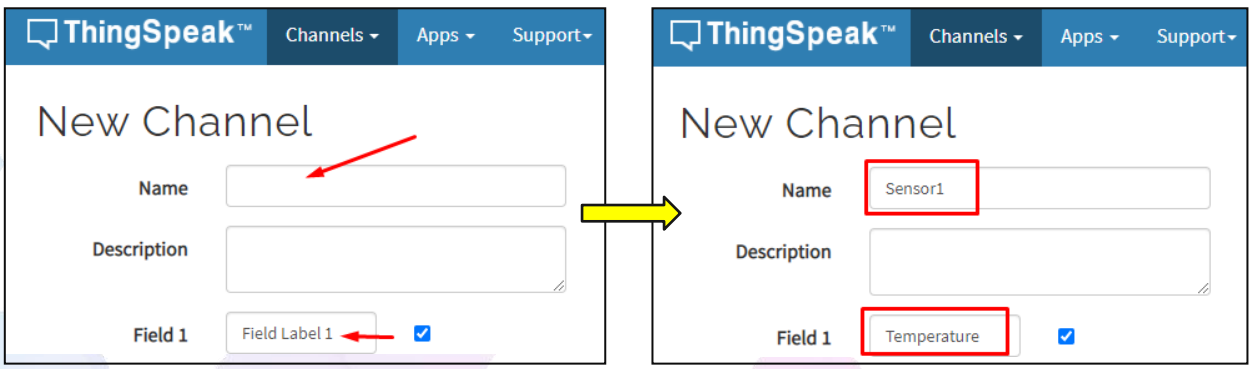
3. You will receive a verification email from mathworks. Click on Verify your email button.



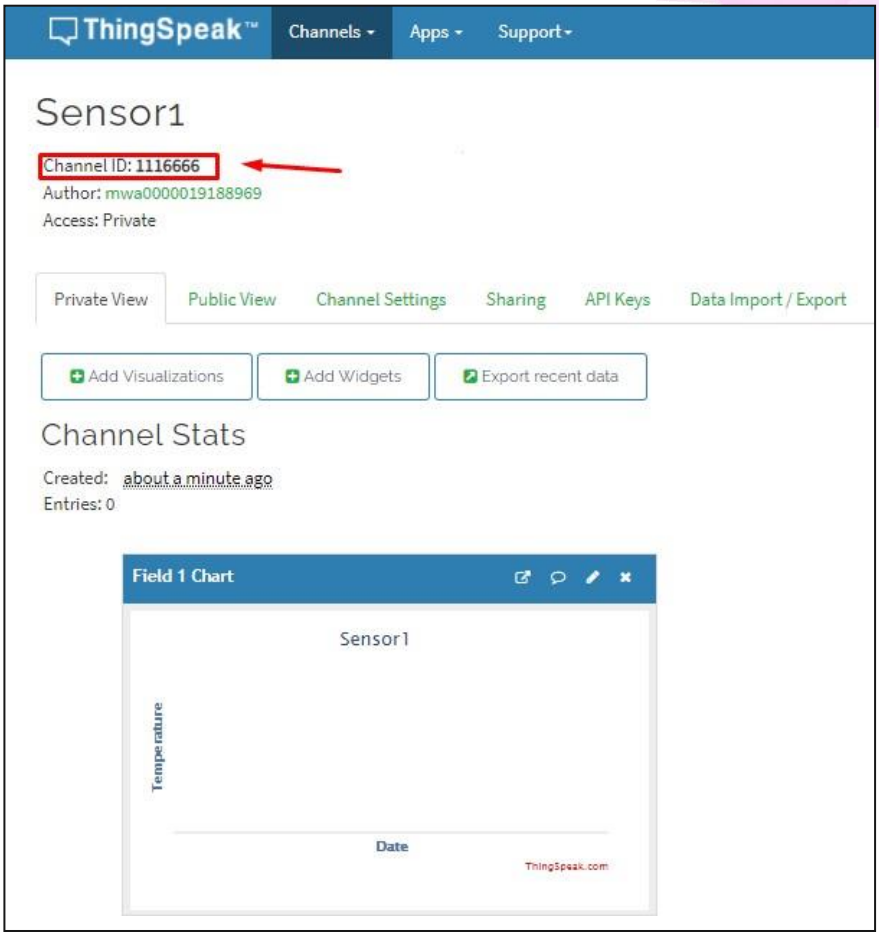
4. Once, you have verified your email, you will be prompt to keyed in your Password.
5. After that you will be brought to your channels as shown in figure below. Click on **New Channel** button.



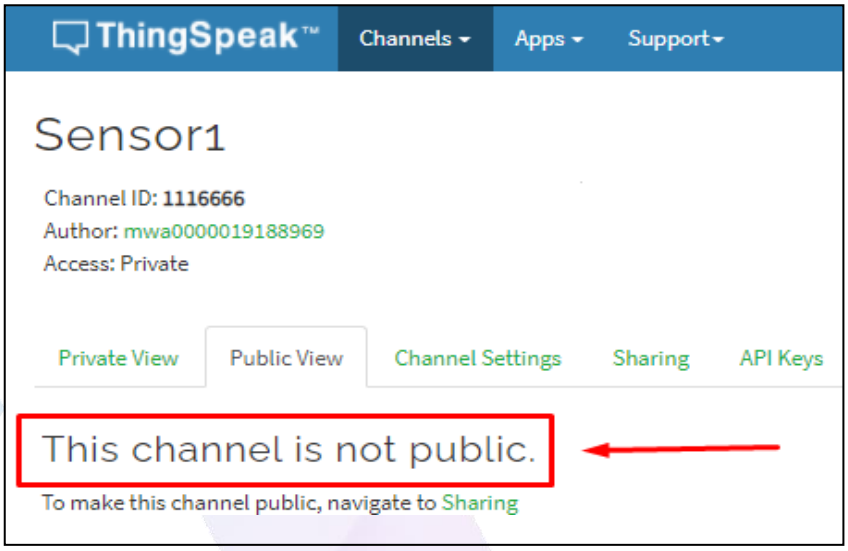
6. Fill in the name as Sensor1, and change the Field Label 1 to Temperature and scroll down to Save Channel.



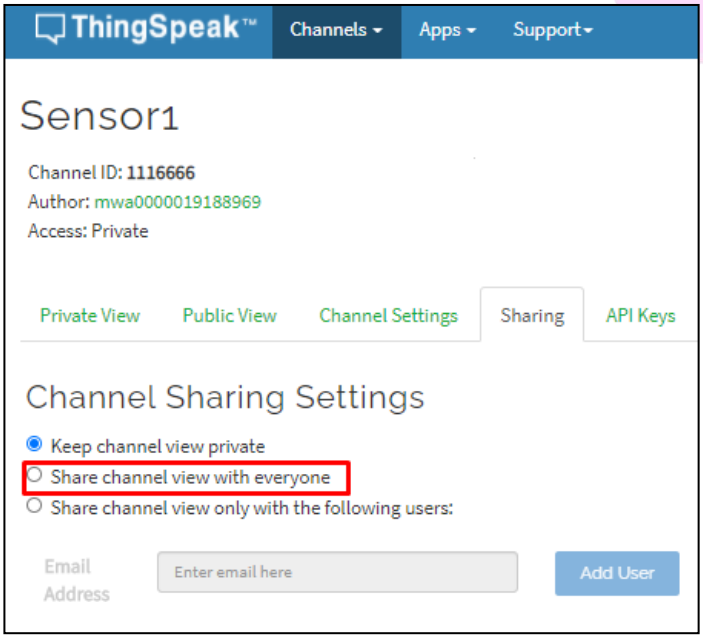
7. Your Channel will look something like this. Please take note of the channel ID.



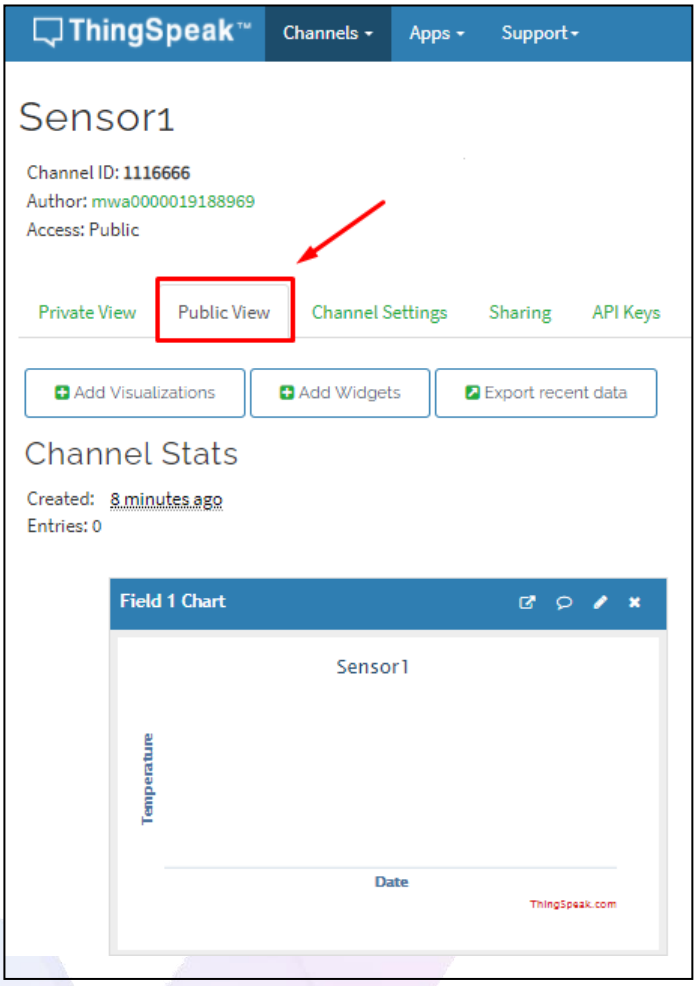
8. Now, change the access to the channel to be publically available.

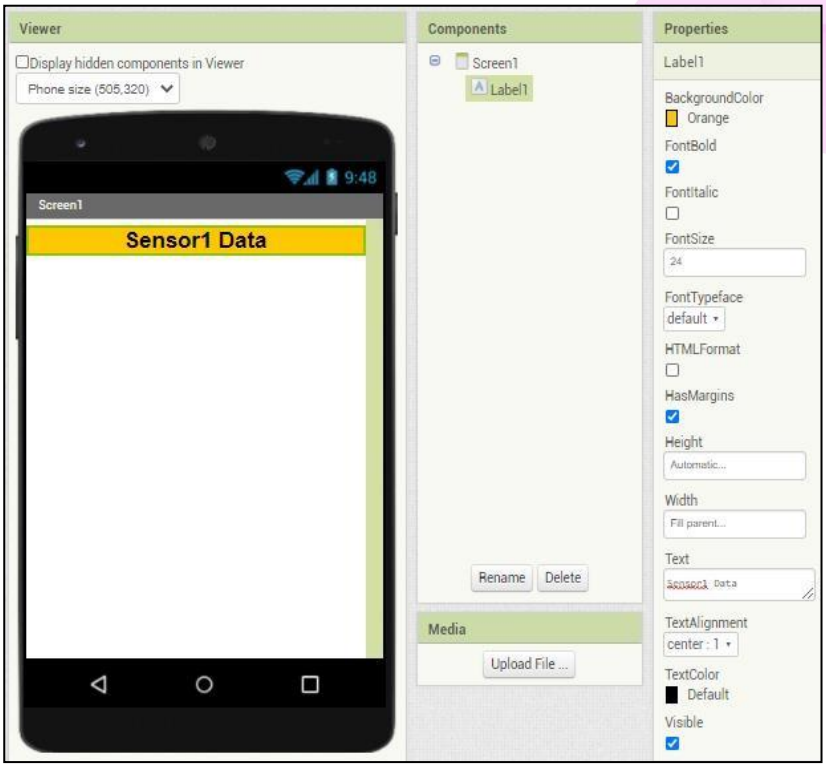


9. Go to Sharing tab and choose the Share channel view with everyone.

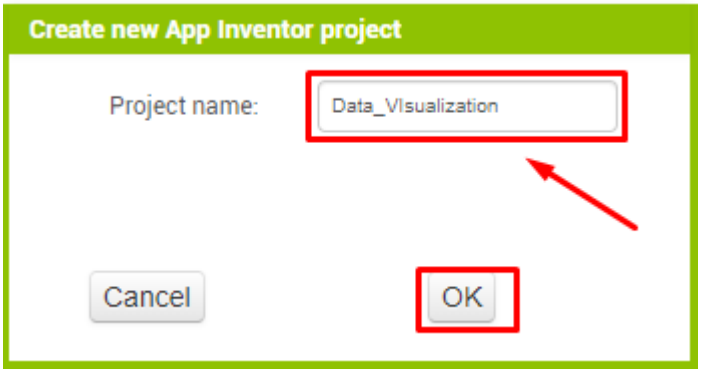


10. Now, the channel is ready for the public to Publish and Subscribe.



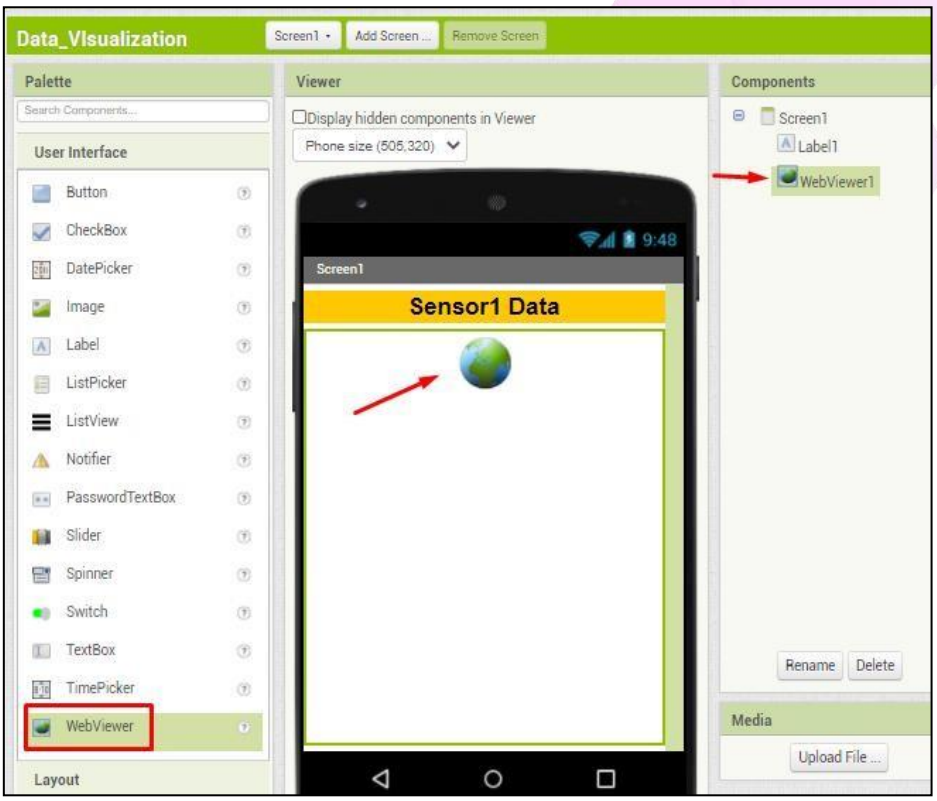


11. Let's create our app. Login to your MIT App Inventor account and Start new project called Data_Visualization. Click OK to continue.

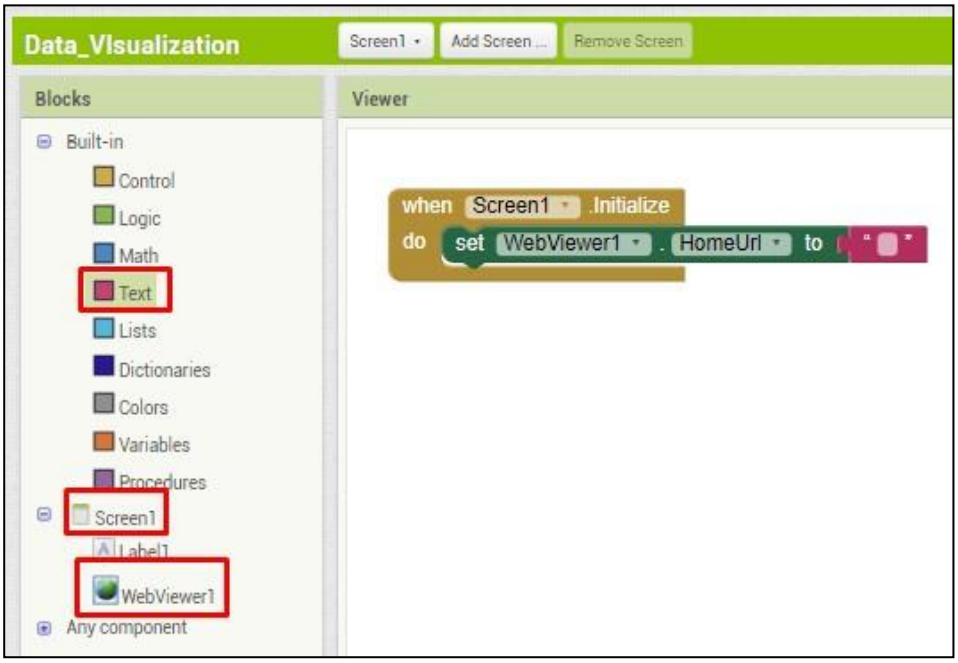


12. Add a label to Screen1. Change the following Properties for Label 1:
    a. BackgroundColor: **Orange**
    b. FontBold: **checked**
    c. FontSize: **24.0**
    d. Width: **Fill parent**
    e. Text: **Sensor1 Data**
    f. TextAlignment **center:1**

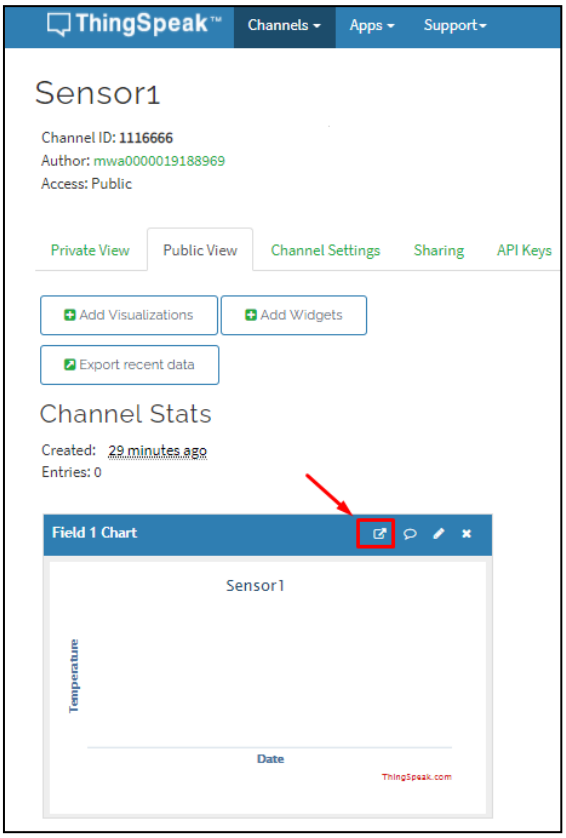
13. Then, add a WebViewer as shown in figure below.



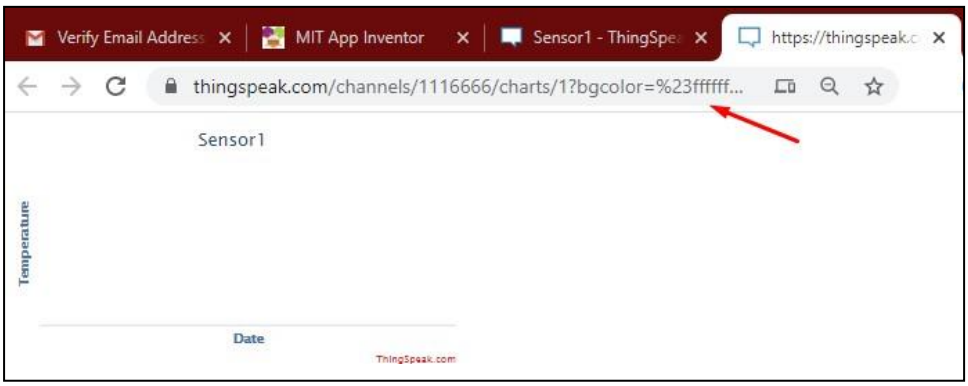
14. Then, go to the MIT App Inventor Blocks and add the following blocks.



15. Go back to your ThingSpeak account and click on Field 1 Chart.

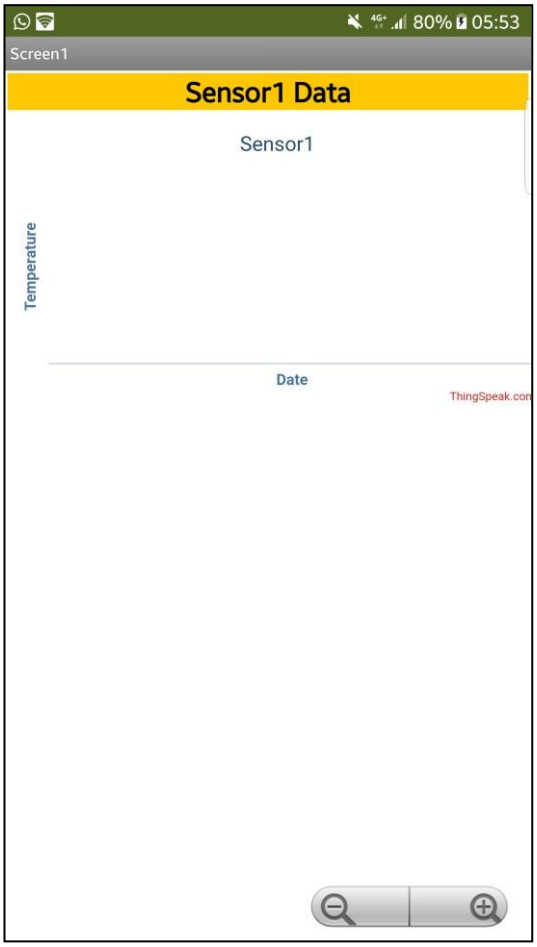


16. A new tab will open the Field 1 Chart as show in figure below. Copy the URL to the Chart.

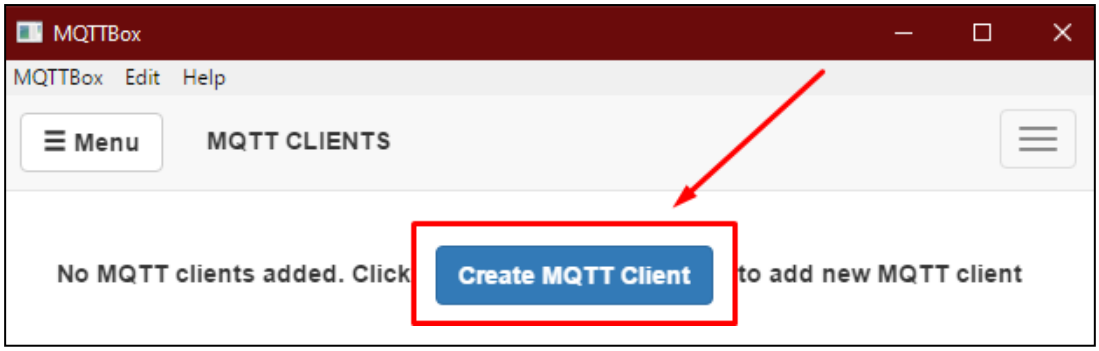


17. Next, paste the URL in the String block

18. Lastly, we will test it in our Android device. As you can see there are no chart being plot because there is no data being sent to ThingSpeak.

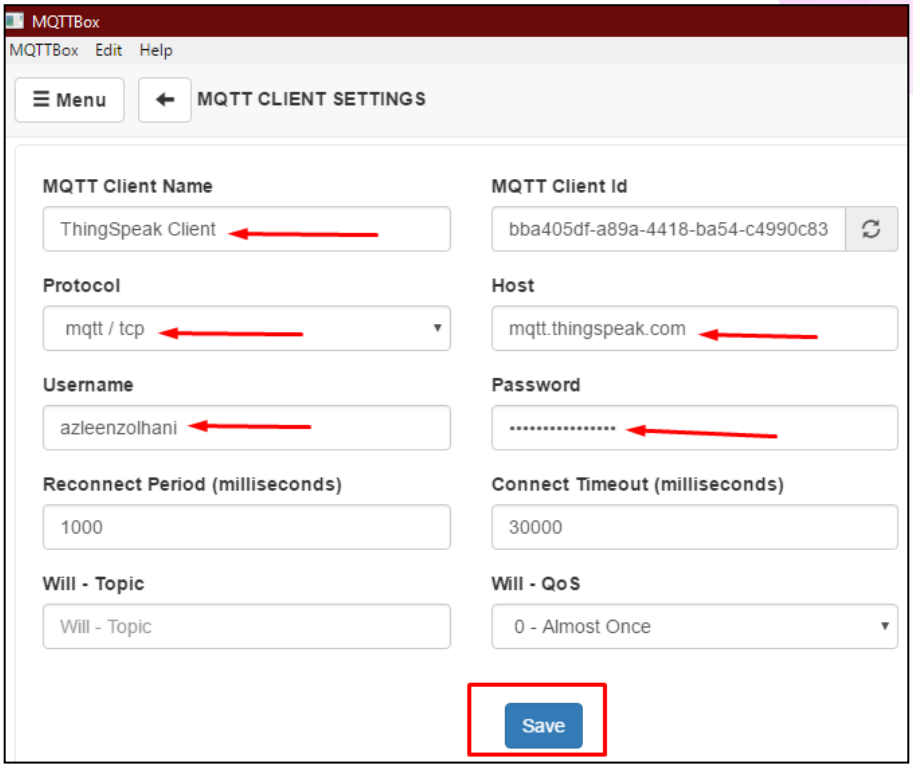


19. Now, we will use MQTTBox to transfer data to ThingSpeak. Launch your MQTTBox. And click on Create MQTT Client

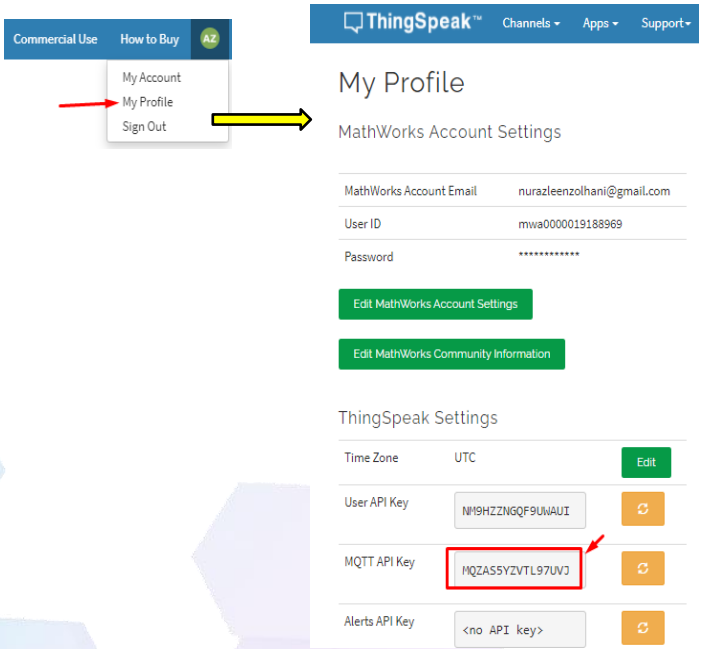


20. Change the following settings:
    a. MQTT Client Name: ThingSpeak Client
    b. Protocol: mqtt/tcp

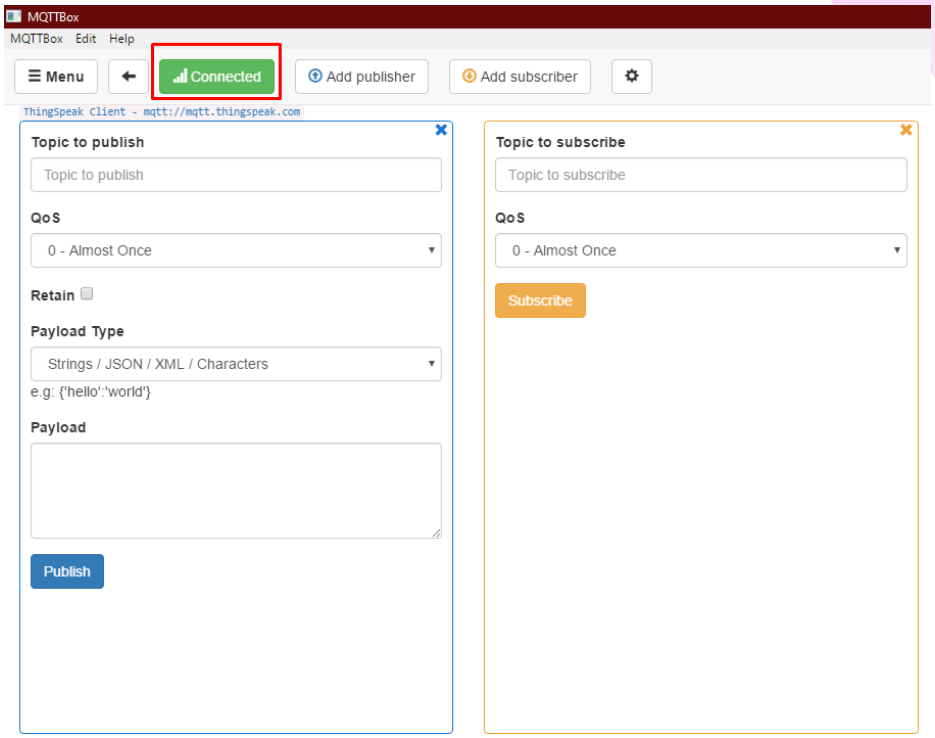
c. Host: mqtt.thingspeak.com
d. Username: <yourname> It can be anything
e. Password:<MQTT API Key>



21. Now, we will use MQTTBox to transfer data to ThingSpeak. Launch your MQTTBox. And click on Create MQTT Client
22. To retrieve the MQTT API Key, go to ThingSpeak, click My Profile and scroll down till you see MQTT API Key as shown in figure below. Copy and paste the key as Password

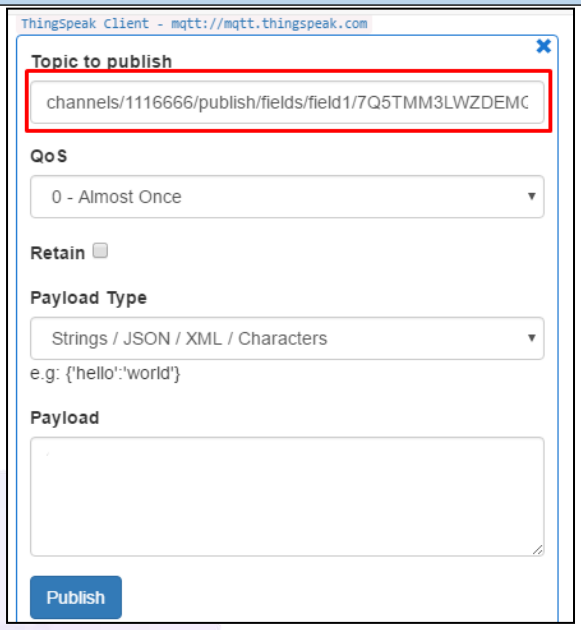


23. Make sure that your MQTTBox is successfully connected to the ThingSpeak broker.

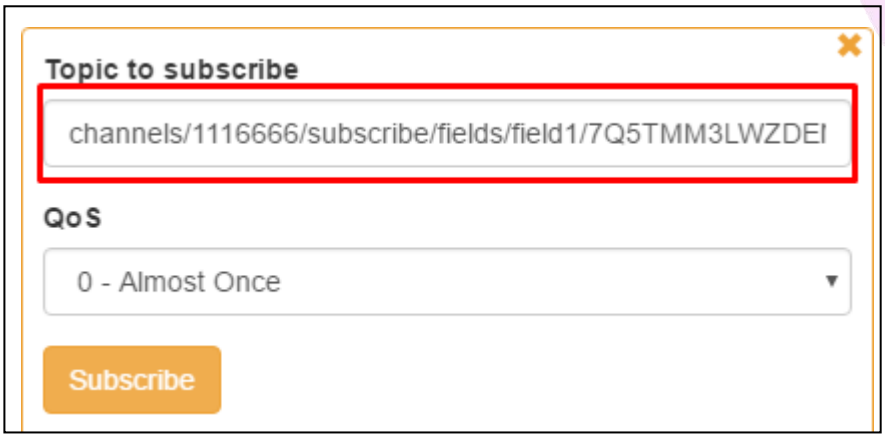


24. Now, to test we will be publishing and subscribing data at the MQTTBox. To publish data to a ThingSpeak broker we must follow the following settings:

channels/<channelID>/publish/fields/field<fieldnumber>/<apikey>



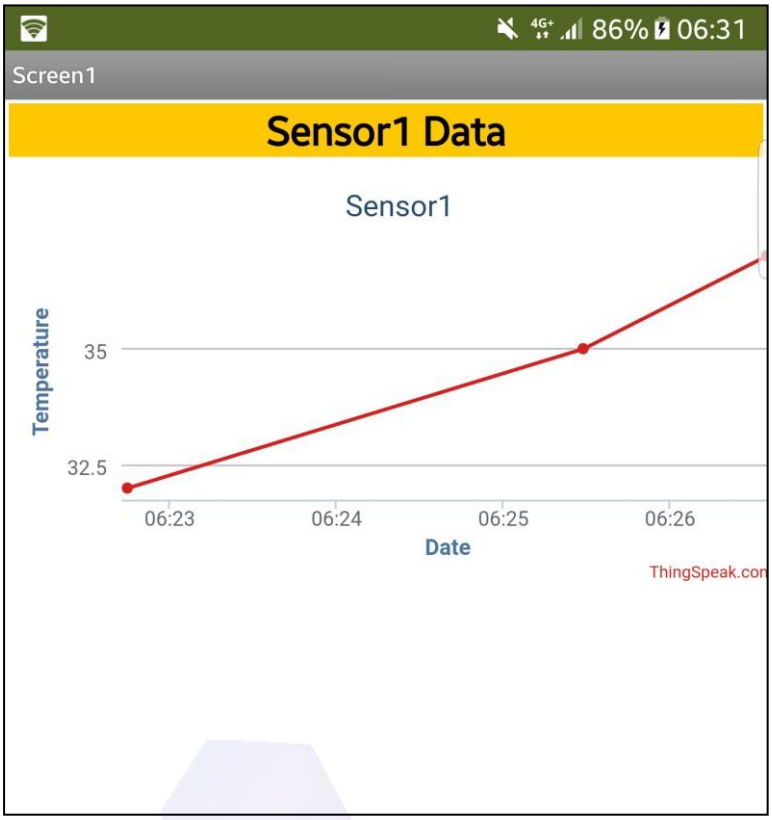
channels/<channelID>/subscribe/fields/field<fieldnumber>/<apikey>



25. We will publish three (3) data which are 32, 35 and 37.

26. Make sure that we received a data 32, 35 and 37 at the subscriber.

Now, check at your AI Companion, you should be able to see the chart.



References:

1. http://appinventor.mit.edu/explore/sites/all/files/teachingappcreation/unit1/MagicTrickHa ndout.pdf
2. https://appinventor.mit.edu/explore/library
3. https://appinventor.mit.edu/explore/ai2/tutorials
4. https://www.programwithappinventor.org/
5. https://www.amazon.com/Learning-MIT-App-Inventor-Hands-On/dp/0133798631/
6. https://www.mathworks.com/help/thingspeak/use-desktop-mqtt-client-to-publish-to-a- channel.html