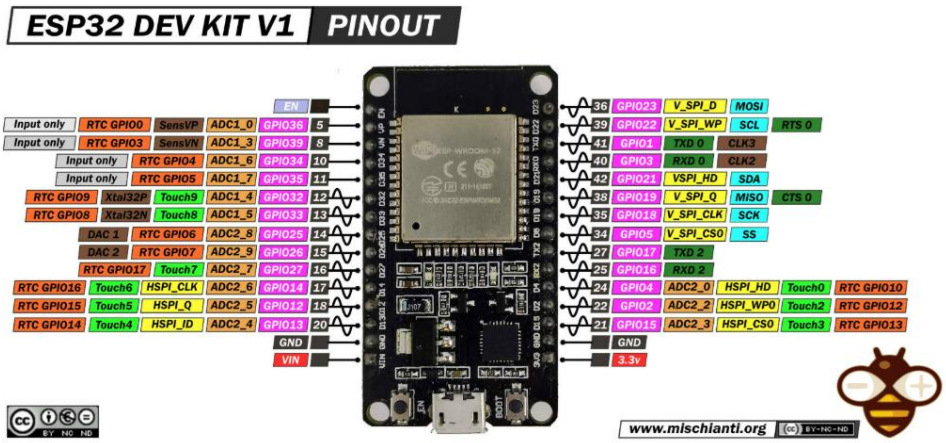
Experiment No.5: Introduction to IoT and Android Application development

Objective:	To introduce learners to the fundamentals of the Internet of Things (IoT) and Android-based application development.
Outcome:	▪ Learners explore the IoT-enabled ESP32 module and temperature sensor - DHT22 by interfacing them and interpreting the displayed results on a hyper terminal. ▪ Learners use an example web based IoT interface platform, 'Thingspeak', and display real-time temperature and/or humidity readings for remote access. ▪ Learners develop an understanding of the open-source Android development platform, MIT App Inventor, by creating a sample app that displays and announces a given message.
Tasks Problem Statement, Procedure, Observation , Discussion, and Report:	1) Introduction to the concept and technology of Internet of Things (IoT). 2) Introduction to ESP32. **Kindly Note: The Arduino IDE rigorously requires, regular updates. Hence it is very much advisable to ensure basic internet connectivity before proceeding with the activities.** Activity 1: Blink on board LED **Follow the steps below very carefully:** a) ESP 32 Dev Kit v1: Pinout  *ESP32 DOIT DEV KIT v1 pinout* b) The ESP32 can be programmed using different firmware and programming languages, like: Arduino C/C++ using the Arduino core for the ESP32 Espressif IDF (IoT Development Framework) Micropython

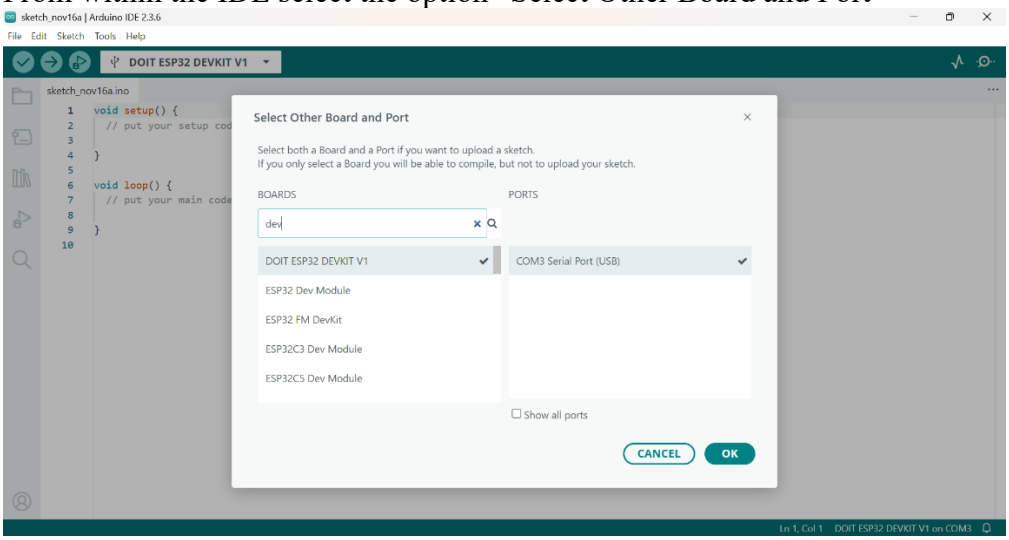
JavaScript etc.

We shall be using Arduino IDE and C type programming language to code and program the ESP32.

c) The latest version of Arduino IDE viz. v 2.3.3 has already been installed on your computer terminals.

d) One also needs to install the CP210x USB to UART Bridge Virtual COM Port (VCP) drivers. This allows communication with the Arduino compatible boards, ESP 32 in this case, to communicate on the USB port.

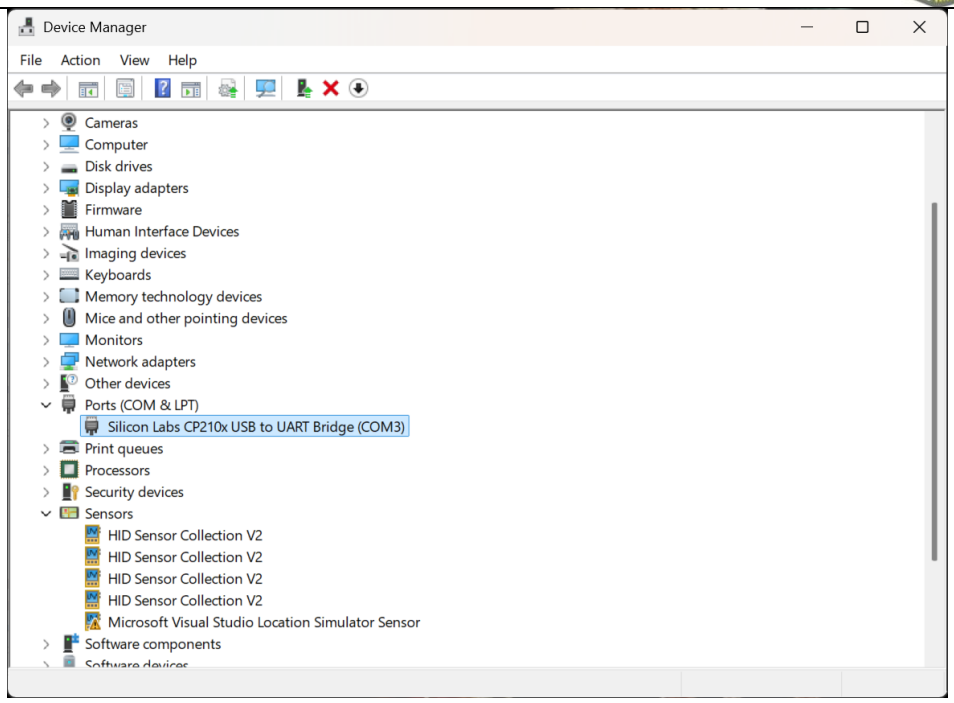
e) Configure the Arduino IDE on the terminal for ESP 32 board:
From within the IDE select the option "Select Other Board and Port"



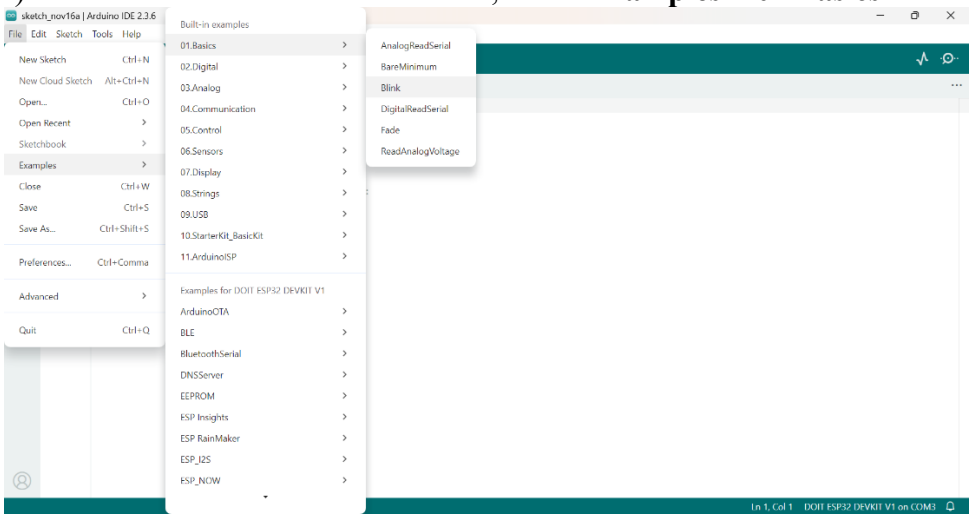
In the search menu, search for the board 'ESP 32 Dev Kit V1'. As shown select – DOIT ESP32 DEVKIT V1, as the board.

Also select the corresponding COM (communication port for serial communication over the USB port) port. – COM3 as above.

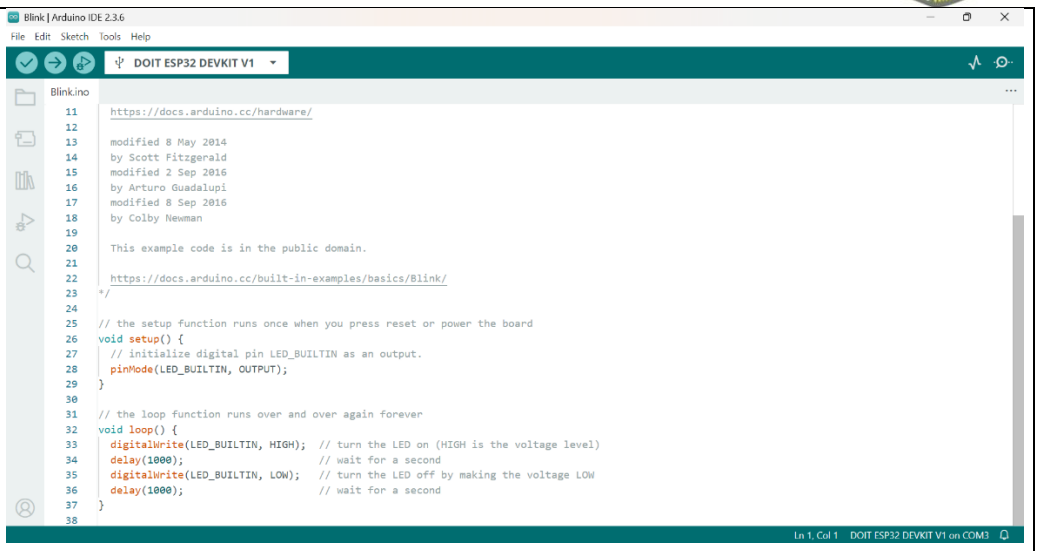
(This may differ from terminal to terminal and may require one to verify from the 'Device Manager', in Windows for obtaining the right serial port assigned to the board connected from the various available)
Search – Device Manager in the Windows, Search bar and locate the right COM port as shown, from the – Ports (COM & LPT) menu.



f) Now from the File menu in the IDE, select **Examples > 01 Basics > Blink**



As seen below the example code file Blink.ino, made available in public domain, gets opened:



Few points to be noted:

- Arduino Files of code or program are text files, called as '**sketch**' and are saved by the default extension **.ino**
- Every Arduino program or sketch consists of two functions:

setup ( ) {

}

and

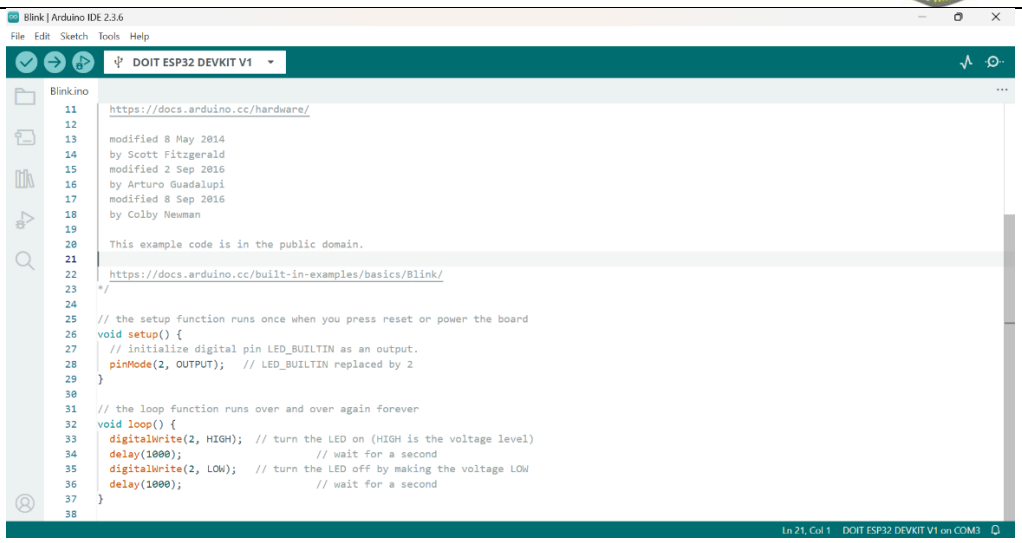
loop ( ) {

}

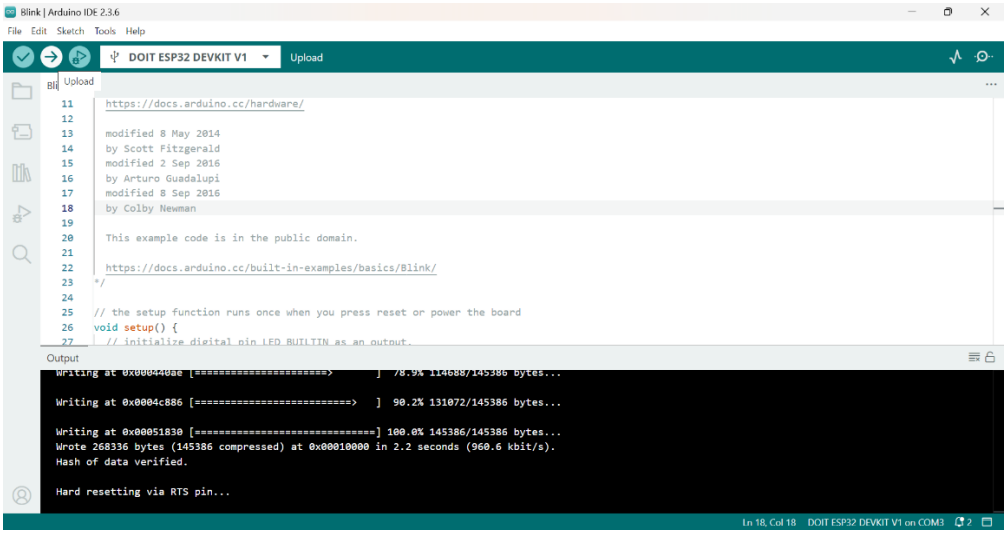
- Upon compilation and during execution, the setup () function executes only once, whereas the loop () function executes continuously. Thus the setup () function is expected to contain the initialisation code, whereas the loop () function contains the body of the program functionality.
- Arduino programs have a C – type structure. Thus the body of the function is enclosed in brace brackets {…..} and every statement ends with a semicolon ;

g) With the board connected, right board and port selected in the terminal and the above example i.e. Blink.ino being opened, make the following changes in the program before compilation and upload on the board.

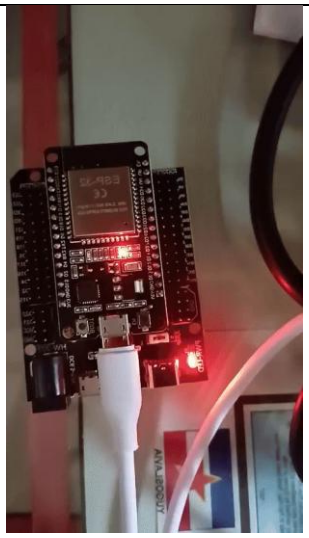
Replace **LED_BUILTIN** by the ESP 32's built-in LED's I/O port pin – 2



Verify (Compile) the code by clicking on the 'tic' icon at the top left of the IDE.
Upon receiving no errors
Upload the code by clicking on the 'right arrow' icon.



As seen and reflected in the code the in-built LED blinks at the rate decided by the code's delay of 1000 milli seconds in the OFF and ON period.



h) Change the delay period and repeat the process. Observe and verify the changes visually.

Activity 2: Interfacing the ambient temperature and humidity sensor – DHT 22 to ESP 32, and display the temperature and humidity on PC hyper terminal.

a) The pre-requisite for interfacing sensors and relating to their readings is a careful study of their data sheets. Download and refer to the data sheet of DHT 22, to identify its functional details and measurement specifications.

b) The **DHT11 and DHT22** are popular, low-cost digital sensors used to measure ambient **temperature and relative humidity** in hobbyist electronics projects. They use a single-wire digital interface for communication and consist of a capacitive humidity sensor and a thermistor (NTC) for temperature measurement.

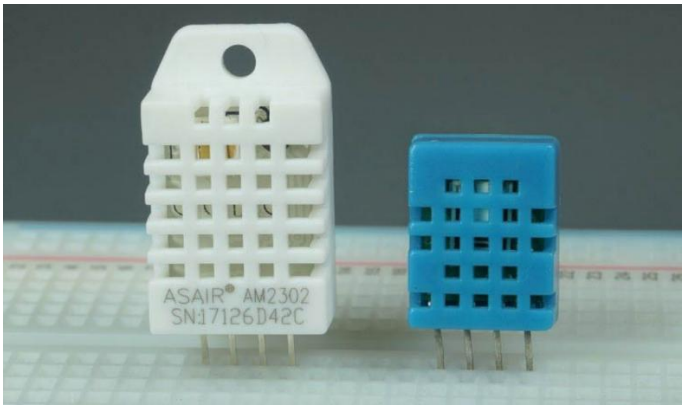


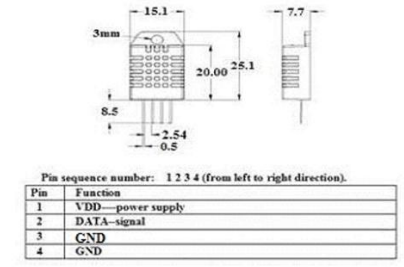
Figure: Sensors – DHT22 (AM2302) and DHT11

c) **Key Differences: DHT11 vs. DHT22**

The DHT22 is the more advanced version, offering higher accuracy and a wider measurement range at a slightly higher cost.

Parameter	DHT11	DHT22 (AM2302)
Humidity Range	20% to 80% RH	0% to 100% RH
Humidity Accuracy	±5% RH	±2% to 5% RH
Temperature Range	0°C to 50°C	-40°C to 80°C
Temperature Acc.	±2°C	±0.5°C
Sampling Rate	1 Hz (once per second)	0.5 Hz (once per two seconds)
Cost	Lower	Higher
Resolution	1°C / 1% RH	0.1°C / 0.1% RH

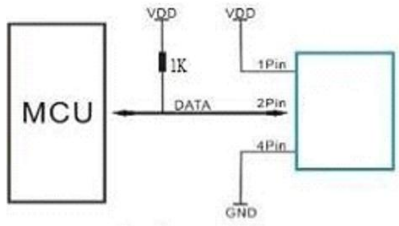
d) DHT 22 Pinout and connections:



Pin sequence number: 1 2 3 4 (from left to right direction).

Pin	Function
1	VDD—power supply
2	DATA–signal
3	GND
4	GND

Standard AM2302's dimensions as above



NOTE: Make the connections for DHT 22 as shown above i.e.
+Ve or Pin 1 is connected to Vcc on the base board
- Ve or Pin 3 is connected to GND on the base board
OUT or Pin 2 is connected to D4 on the base board
KINDLY GET THE CONNECTIONS CHECKED FROM MENTORS !!

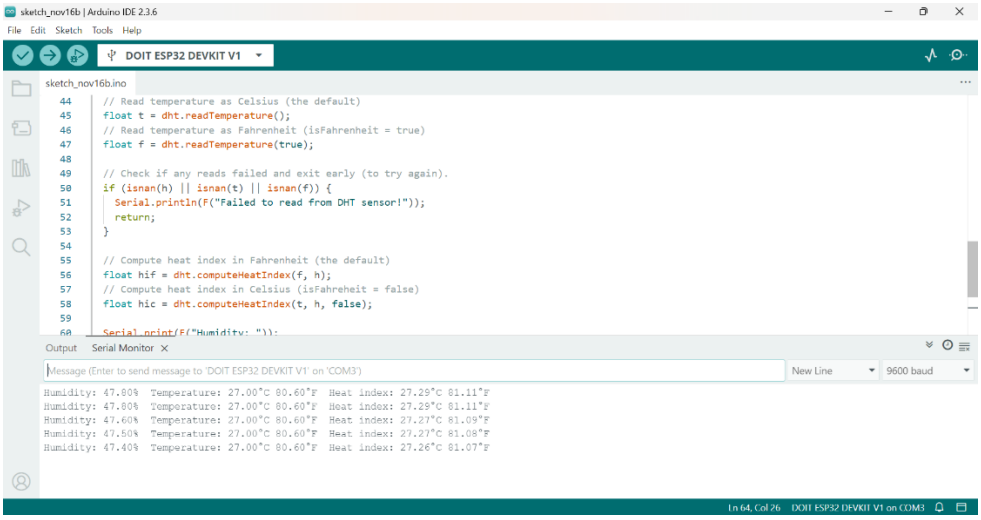
e) Install the following libraries from **Tools>Manage Libraries**
i) DHT 22 and
ii) Adafruit Unified Sensor
(if not appearing as installed)

f) From the File menu, initiate new sketch and use the following code in the new sketch:

```
//*****************************************************************
// Example testing sketch for various DHT humidity/temperature sensors written by ladyada
// REQUIRES the following Arduino libraries:
// - DHT Sensor Library: https://github.com/adafruit/DHT-sensor-library
// - Adafruit Unified Sensor Lib: https://github.com/adafruit/Adafruit_Sensor

#include "DHT.h"

#define DHTPIN 4     // Digital pin connected to the DHT sensor
// Feather HUZZAH ESP8266 note: use pins 3, 4, 5, 12, 13 or 14 --
// Pin 15 can work but DHT must be disconnected during program upload.

// Uncomment whatever type you're using!
//#define DHTTYPE DHT11   // DHT 11
#define DHTTYPE DHT22   // DHT 22  (AM2302), AM2321
//#define DHTTYPE DHT21   // DHT 21 (AM2301)

// Connect pin 1 (on the left) of the sensor to +5V
// NOTE: If using a board with 3.3V logic like an Arduino Due connect pin 1
// to 3.3V instead of 5V!
// Connect pin 2 of the sensor to whatever your DHTPIN is
// Connect pin 4 (on the right) of the sensor to GROUND
// Connect a 10K resistor from pin 2 (data) to pin 1 (power) of the sensor

// Initialize DHT sensor.
// Note that older versions of this library took an optional third parameter to
// tweak the timings for faster processors.  This parameter is no longer needed
// as the current DHT reading algorithm adjusts itself to work on faster procs.
DHT dht(DHTPIN, DHTTYPE);

void setup() {
  Serial.begin(9600);
  Serial.println(F("DHTxx test!"));

  dht.begin();
}

void loop() {
  // Wait a few seconds between measurements.
  delay(2000);

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

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

```
  // Compute heat index in Fahrenheit (the default)
  float hif = dht.computeHeatIndex(f, h);
  // Compute heat index in Celsius (isFahreheit = false)
  float hic = dht.computeHeatIndex(t, h, false);

  Serial.print(F("Humidity: "));
  Serial.print(h);
  Serial.print(F("%  Temperature: "));
  Serial.print(t);
  Serial.print(F("°C "));
  Serial.print(f);
  Serial.print(F("°F  Heat index: "));
  Serial.print(hic);
  Serial.print(F("°C "));
  Serial.print(hif);
  Serial.println(F("°F"));
}

//*************************************************************
```

g) Verify the code and ensure no errors. Upload the code and observe the readings on the serial terminal, provided in the IDE. Select the baud rate for 9600 baud, as specified in the code.
The output should appear similar as this:



Discussion, and Report
Answer the following questions:

1. How can we verify the LED blink frequency to its exact value as decided by the time period in the code ?
2. How can we rely upon or verify the readings obtained.

Result:

- LED blink basic activity completed : Yes/No
- LED blink with changes in delay demonstrated: Yes/No

	• Temperature and Humidity readings displayed on hyper-terminal: Yes/No **Faculty Signature & Date: _____**
Conclusions:	