CIJE-Tech High School

IoT

The Internet of Things

Updated for esp32

Includes 2 complete codes

The Center for Initiatives in Jewish Education (CIJE)

The Center for Initiatives in Jewish Education (CIJE) strengthens and enriches the quality of education in Jewish schools throughout the United States. CIJE is investing in our nation's future by providing beneficiary schools with cutting-edge technology, engaging curricula, and vital support so that students can acquire the skills they need to excel in our global society. Currently, CIJE has more than 225 beneficiary schools across the United States and programs which span grades K-12. CIJE's innovative programs are paving the way for the achievement and success of tomorrow's leaders and thinkers.

CIJE-TECH STEM PROGRAM: AN OVERVIEW

More than ten years ago, the Center for Initiatives in Jewish Education began the implementation of various STEM programs in elementary Jewish schools. The success of these programs brought about the initiation of the CIJE-Tech Principles in Engineering and Applied Engineering programs.

Goals:

The CIJE STEM education programs:

- Provides a challenging and rigorous program of study focusing on the application of STEM subjects.

- Offers courses and pathways for preparation in STEM fields and occupations.

- Bridges and connects in-school and out-of-school learning opportunities.

- Provides opportunities for student exploration of STEM related fields and careers.

- Prepares students for successful college and university STEM education.

To increase STEM learning, the CIJE-Tech programs include activities that improve student and teacher content knowledge and teacher pedagogical skills. Innovative strategies are used, including small group collaborative work and the use of hands-on activities and experiments to promote inquiry and curiosity. Learning is connected to the real world through an emphasis on the application of STEM subjects to everyday life, employment, and the surrounding environment.

The CIJE high school programs were approved as "d" Laboratory Science Courses by the University of California in 2015. The second-year course is approved at the more challenging honors level.

Center for Initiatives in Jewish Education

President   Jason Cury

Vice President, Professional Development   Faigy Ravitz

Director, Curriculum Development   Adam Jerozolim, M.E.

Coordinator, Innovative Programs   Orly Nadler, M.A.

CIJE STEM Specialists:

Christopher Auger-Dominguez	Katherine Owuor, Ph,D.
Dewain Clark, M.A.	Joseph Saltzman
Robert Jones	David Seay, M.A.
Yafa Lamm	Barbara Sehgal, M.A.
Aryeh Laufer	

IoT

The Internet of things

"When we talk about the Internet of Things, it's not just putting RFID tags on some dumb thing so we smart people know where that dumb thing is. It's about embedding intelligence, so things become smarter and do more than they were proposed to do."

– Nicholas Negroponte

Founder, MIT's Media Lab

Table of Contents

Introduction

The esp32 is a low-cost microcontroller, similar to Arduino, that has Wi-Fi capability built in. The chip is a fully functional processor, and in most ways, superior to the Arduino Uno. The Wi-Fi capabilities of the esp32 comes from a compact Wi-Fi antenna built right onto the chips silicon wafer. The chip comes with its own digital and analog pins. Various 3rd party breakouts boards can be used to access these pins and adds features to make the chip more useable; similar to the same way the Arduino UNO provides access to the Atmel chip in its center.

Internet Networks

Stations

Devices that connect to Wi-Fi network are called stations (STA). The source of the Wi-Fi is provided by an access point (AP), that acts as a hub for one or more stations.

Devices that connect to a network are called stations

Station
(ESP8266)

Access Point

Internet

Station (PC)

ESP8266 operating in the **Station** mode
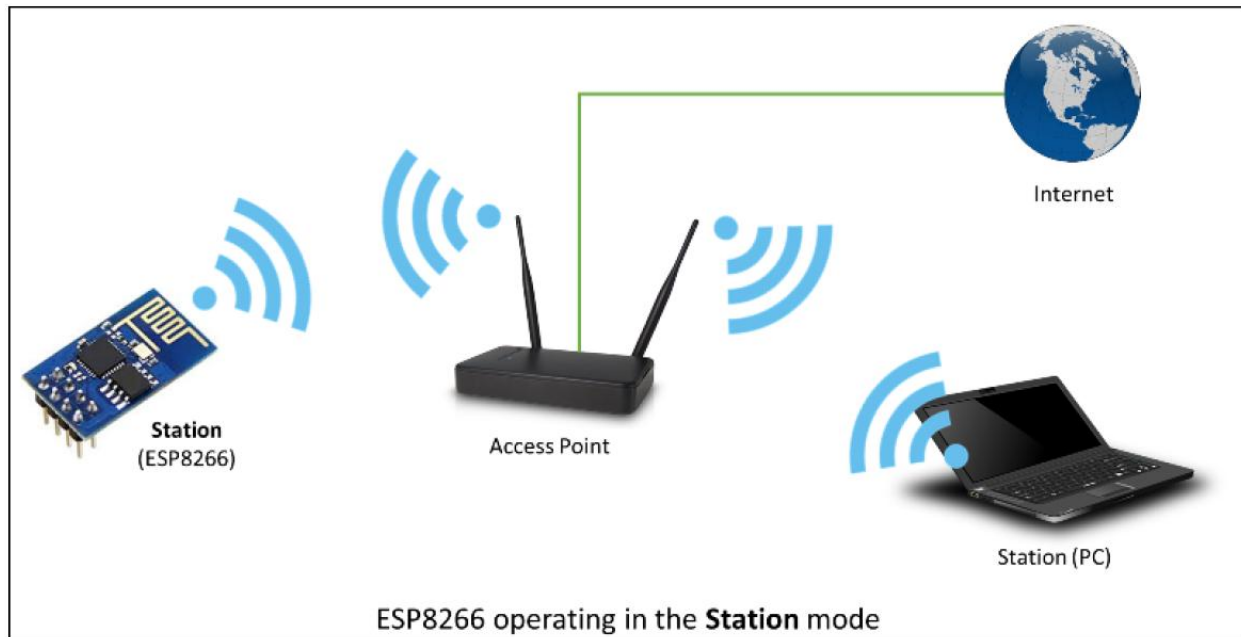
An access point is usually integrated with a router to provide access to the internet. The access point is recognized by a SSID (Service Set Identifier), that essentially is the name of network you select when connecting a device (station) to the Wi-Fi.

An Access Point serves as a bridge for Stations to connect to the internet

When the esp32 acts as an access points for other stations, since it is not wired to the internet, it is called a soft access point (soft-AP). Therefore, we can connect other stations to the esp32 to make a localized network, but none of the devices on the network will receive internet access through the network.

An Access Point that is not connected to the internet is called a Soft Access Point

ESP8266 operating in the **Soft Access Point** mode
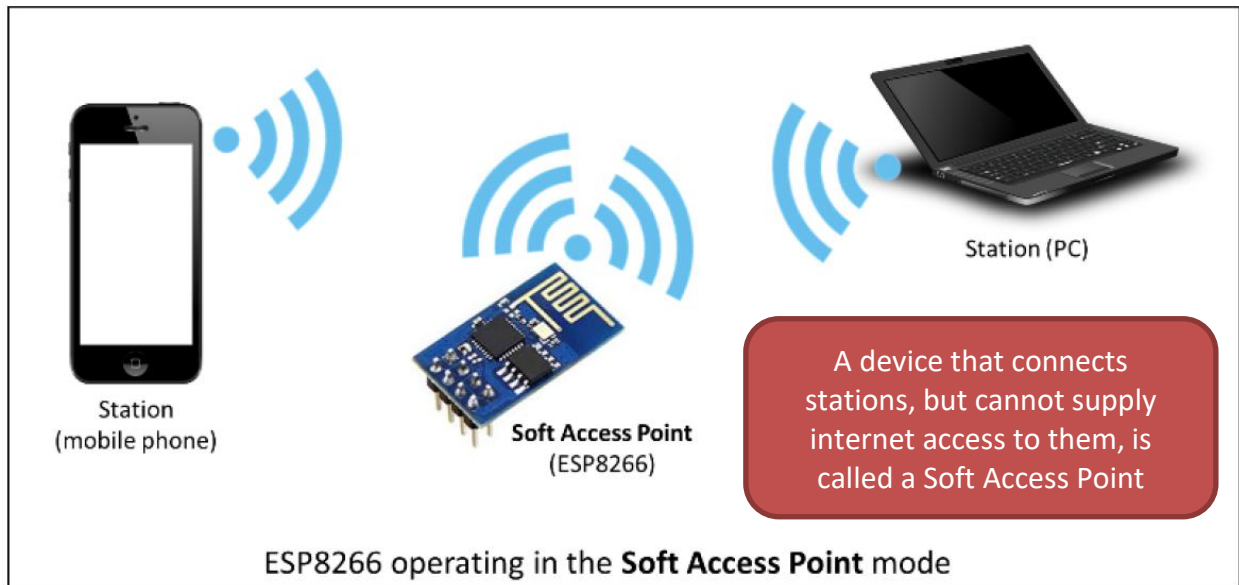
Servers

Servers provide functionality to other stations on a network. They might provide pieces of information, news, access to email, movies, databases, etc. A station (also referred to as a client) connects to servers to send and receive data.

> Servers are devices on a network that provide the functionality and data to the stations (clients).

ESP8266 operating as the **Server**
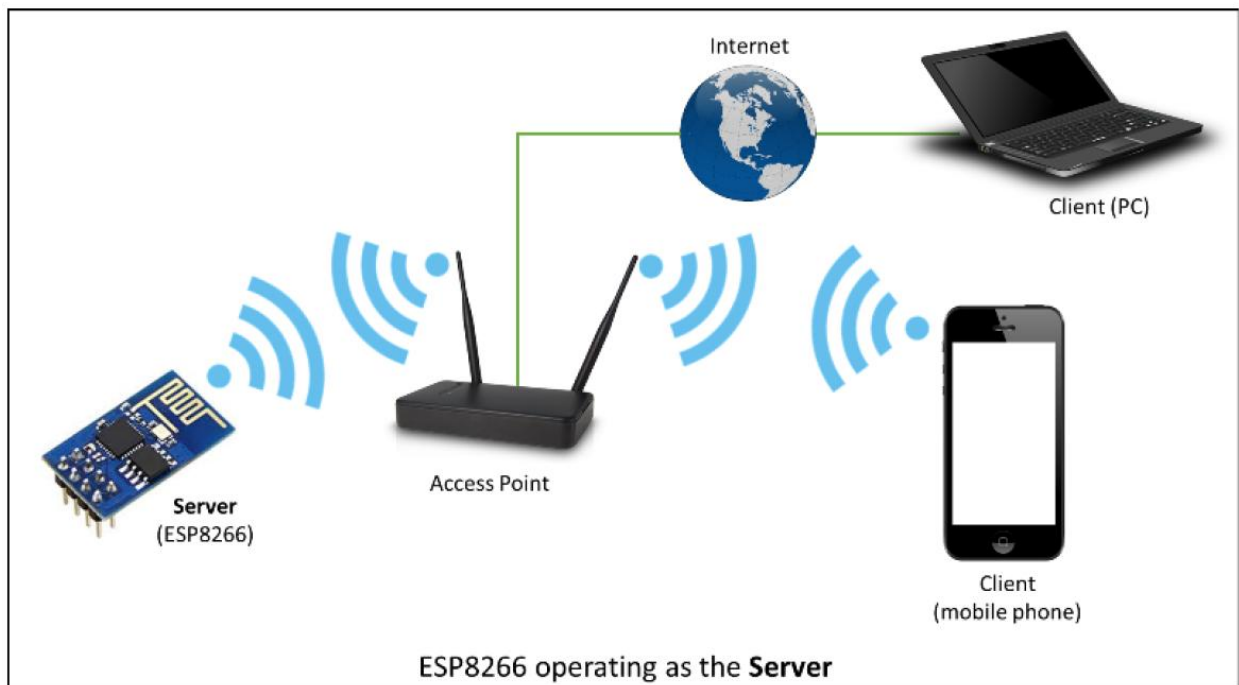
Clients

Some stations on a network are referred to as clients. Clients can access services provided by servers in order to send, receive and process data.

> A client is a device on a network that can access, send, and receive data from a server on the network

ESP8266 operating as the **Client**
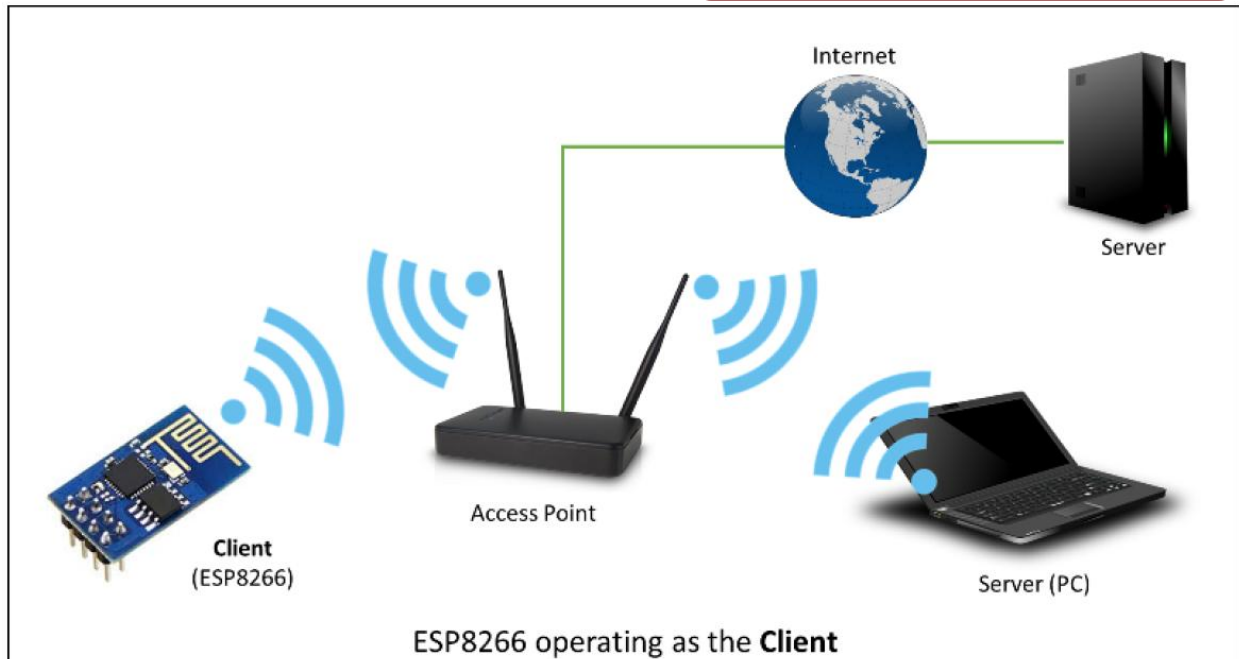
Client Secure

The Client Secure is a type of client, whereas the connection and data exchange with a server is done using a secure protocol. This prevents anyone from having access to the information that is shared between the client and the server. Secure applications have additional memory (and processing) overhead due to the need to run cryptography algorithms. The stronger the certificate's key, the more overhead is needed.

> Clients can interact with each other, and servers on the network, using encryption well.

ESP8266 operating as the **Client Secure**
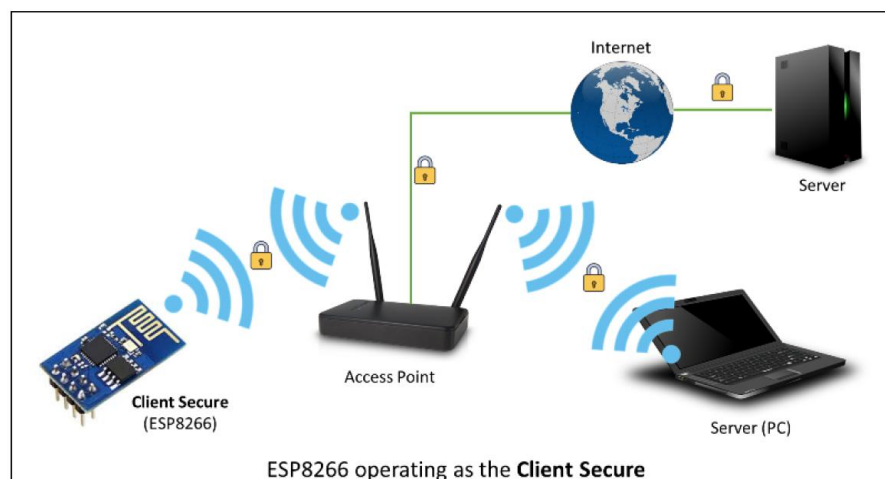
Finding your Device

Internet Protocol (IP) address

When computers and devices and servers are all linked together they form a network. Each device on a network has an address, called an Internet Protocol Address, or IP for short. These addresses are how information is directed to the correct device.

> Each device on a network is given a local address, called a local IP, to be used internally.
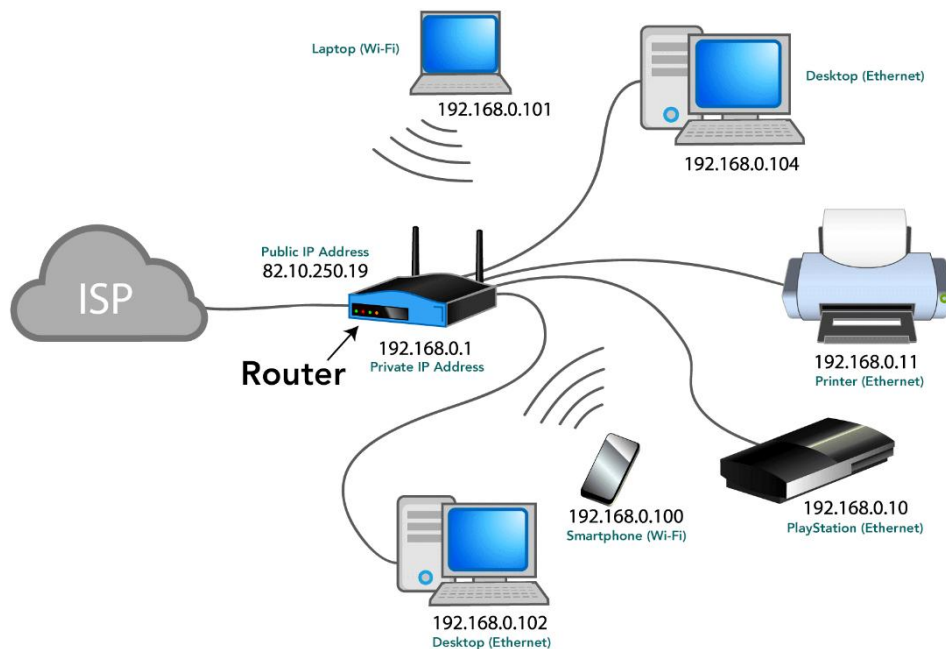> The network as a whole also receives a single IP address, called a Public IP Address, for the rest of the world to locate them.

Being there are only a limited number of global IP addresses (there are more devices in the world than addresses), each network only gets one **global IP**. Since your network can have multiple devices within it, like laptops and phones, the router's job is to assign each device a **local IP** address, one that the outside world cannot see. The router uses these local addresses to distribute the data to the correct device within the network.

So all the data is sent to your networks global IP address. The router then divides it up and distributes it accordingly using the internal local IP addresses.

Local Area Network (LAN)

It is also possible for a network of devices to exist independent of the internet. This could happen, for instance, if the internet connected to your house went down. The internal devices would still be able to communicate with each other using their local IP addresses, there would just be no global IP address associated with the router. This might also be done purposely when a secure network (e.g. – computers within the stock exchange) or a fast network (e.g. – gaming) is required.

Summary

The internet Service Provider (ISP) provides an address for each network, referred to as the global IP. Your local router provides a unique local IP address to each device. Data is received by your router using the global IP address, and is then distributed internally using the local IP addresses.
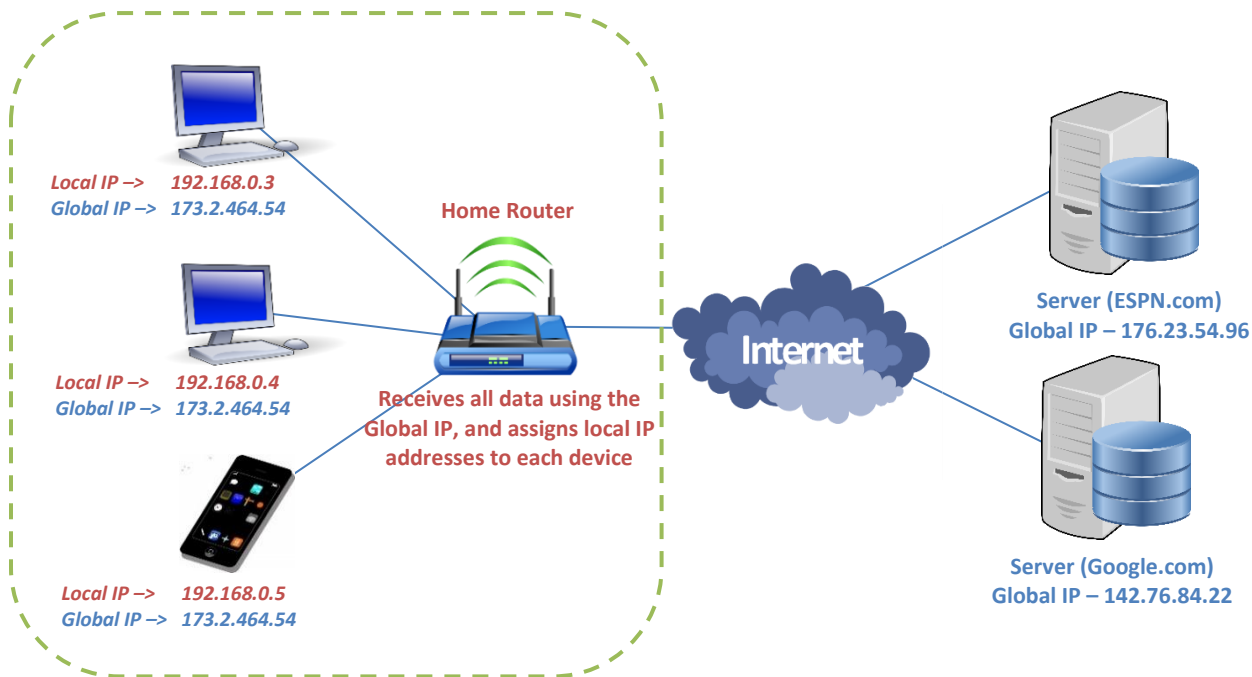
Each device can have two separate IP addresses, a local and a global. A local IP is assigned from your local network router (i.e. – the box in the closet). The global IP is assigned by your Internet Service Provider (ISP) (i.e. – Verizon, comcast, cable, etc.).

So each device within a network has a local IP address, but shares a global IP address. The devices are differentiated on the global network by the ports that they are assigned. The port number is usually listed after the IP address.
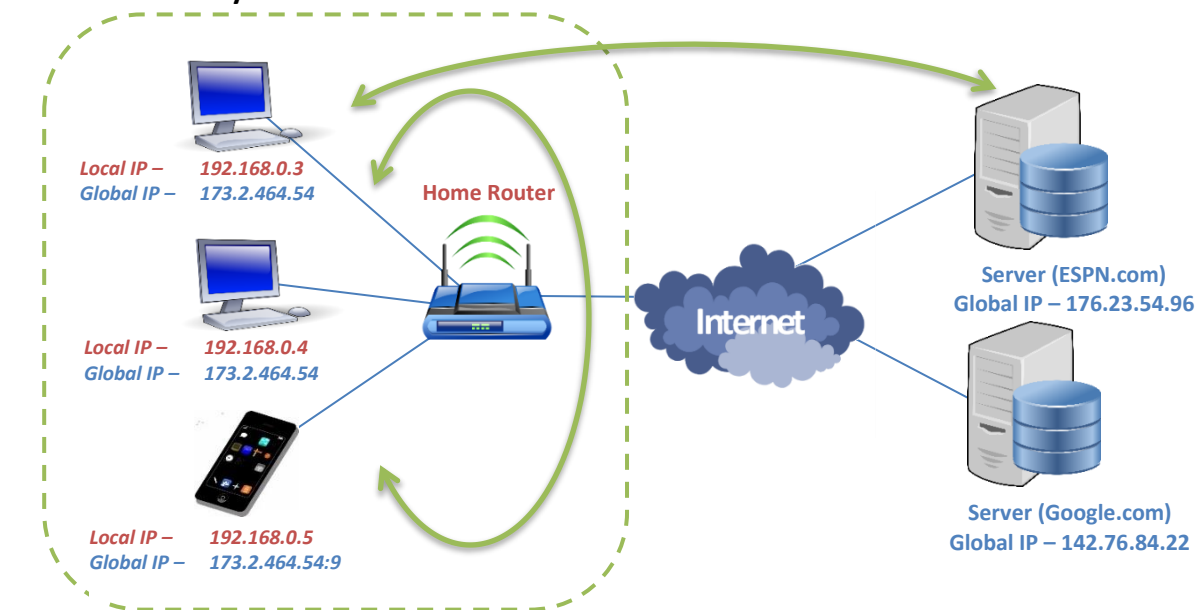
Limited IP addresses

IP version 4 contains four 8-bit numbers (0-255). This leads to a limited number of potential addresses
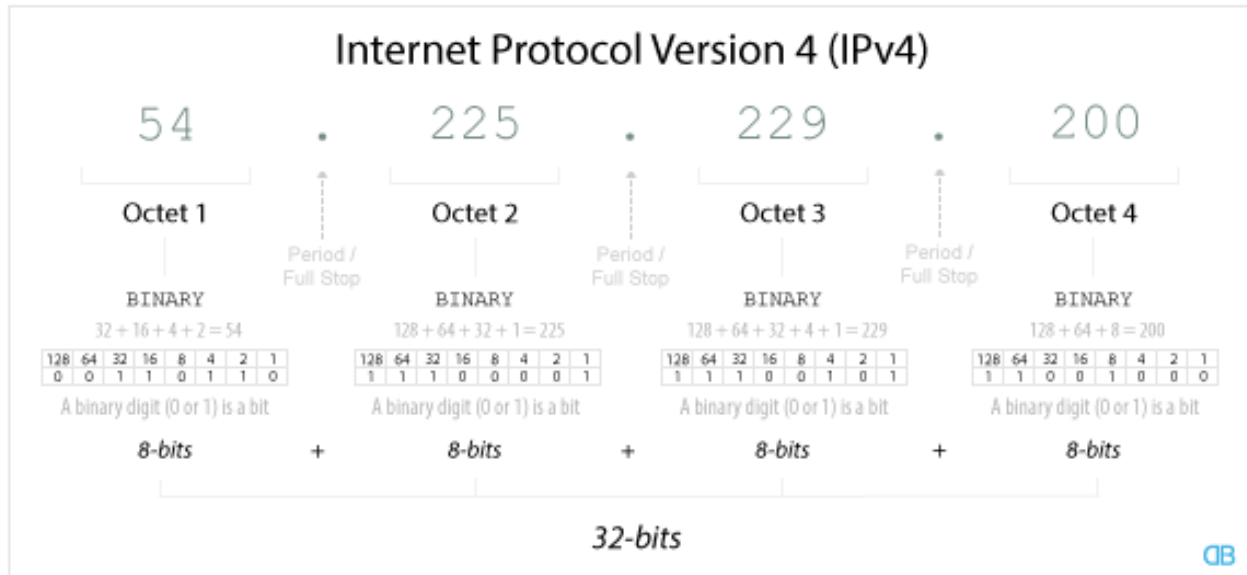
=> 255 x 255 x 255 x 255
=> 255^4
= 4,228,250,625
(approx. 4.3 billion)

Since 2016 there has already been more than 20 billion devices connected to the internet. This includes every computer, laptop, smartphone, smartwatch, smart appliances, cars, airplanes, etc.
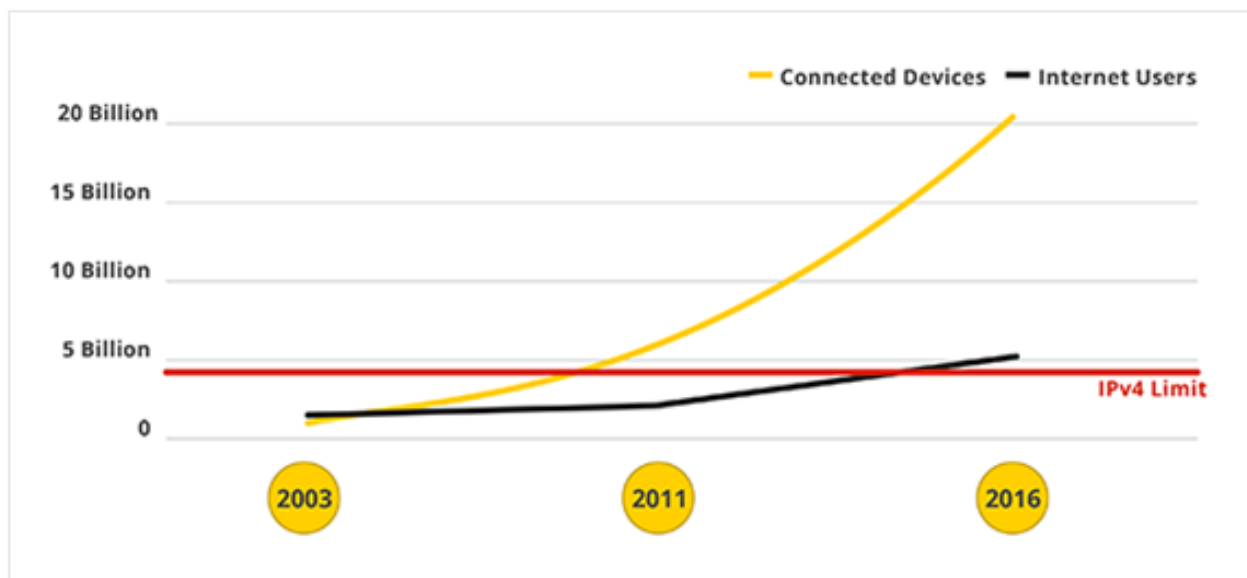
To cope with this growth, IP version 6 was created with eight 16-bit numbers. This leads to many more combinations of IP addresses.
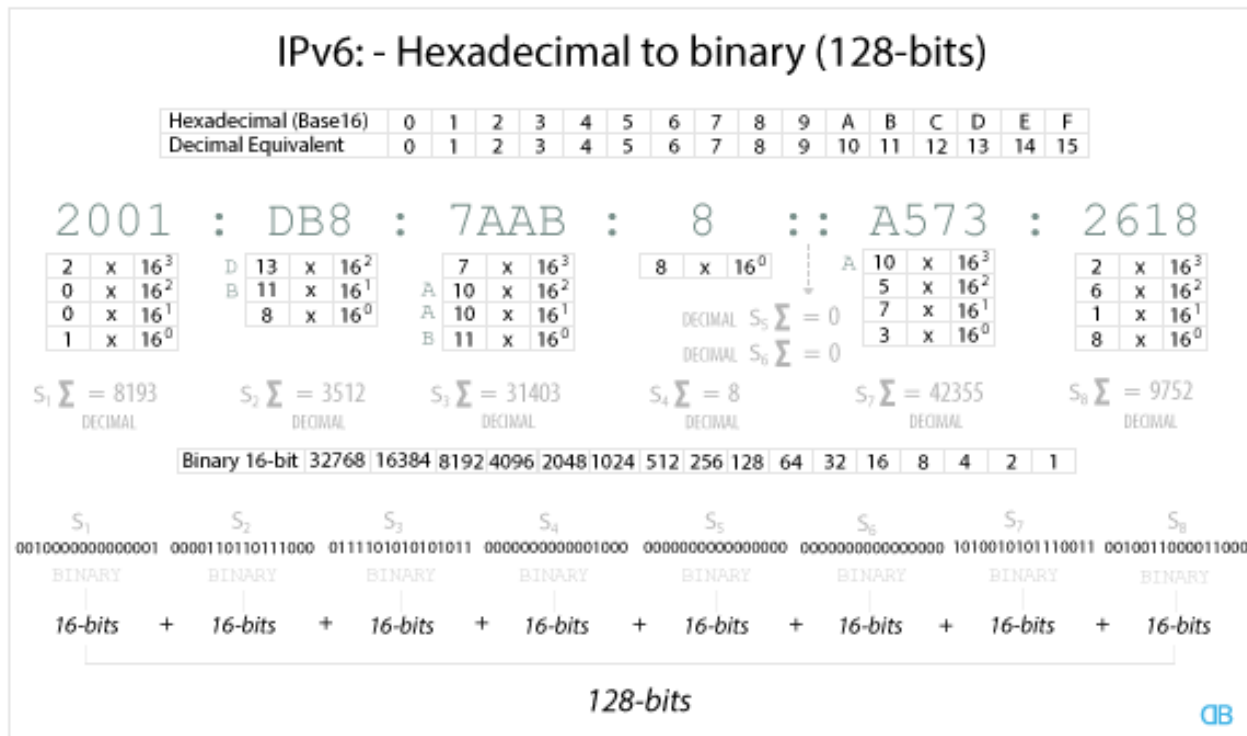
=> 2^{16} x 2^{16} x 2^{16} x 2^{16} x 2^{16} x 2^{16} x 2^{16} x 2^{16}

=> $(2^{16})^8$

=> 2^{128}

= 340,282,366,920,938,000,000,000,000,000,000,000,000

Security

In general, when two devices talk over a network they encrypt their data, making it virtually impossible for someone to read that data, or infuse their own data into either device.
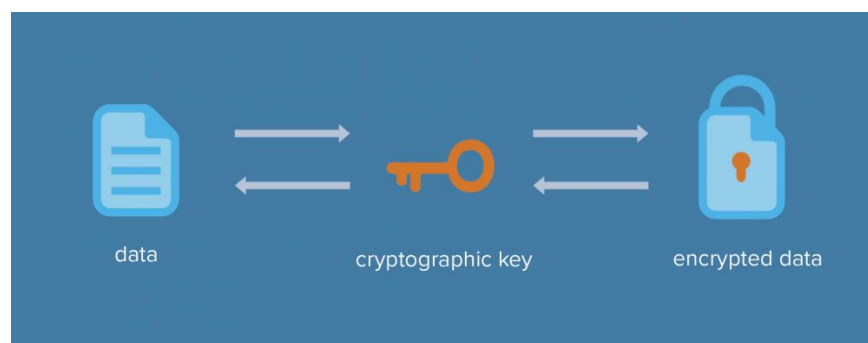
The data remains encrypted the entire time it is being transferred from the sender to the receiver.

How it works

A device that wants to receive data creates a set of two keys. Although different, they operate in tandem so that one key can "lock" the data while the other can "unlock" it. The "locking" key is made public, so that anyone can encrypt, i.e. – lock, data they wish to send over. They data is

then sent over a network. Even if someone were to intercept the data, they would not be able to open it since the "unlock" key remains private. Since the two keys are different, just having the public "locking" key will not help you create an "unlocking" key. Once the data is received by the recipient, they can use the corresponding "unlock" key to decrypt the data.
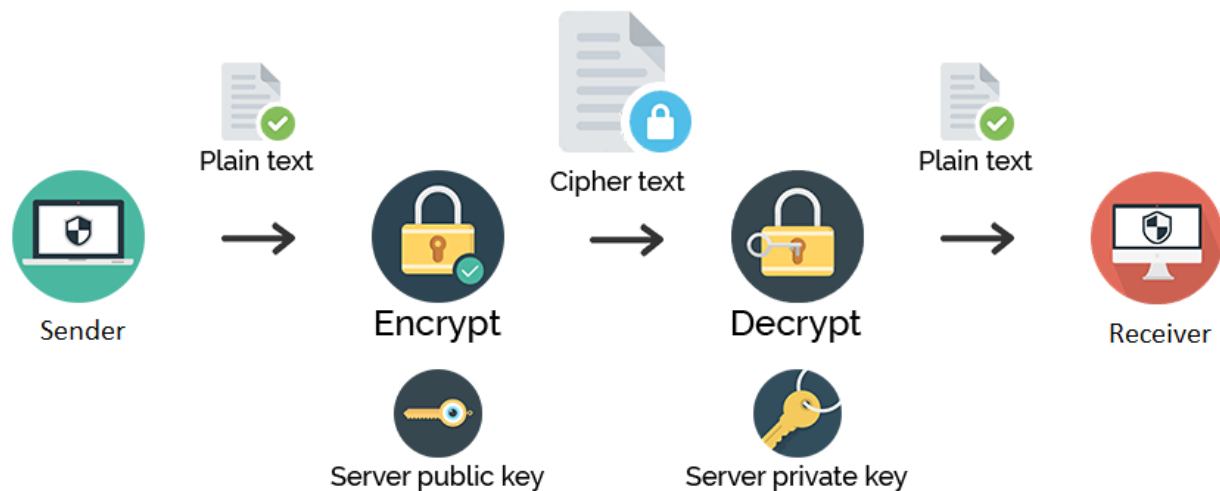
The above example merely touches on the basics of data encryption. There are many more aspects of data encryption such as the size and complexity of the 'keys', the amount of keys, encrypting the encryption keys, private "locking" keys, etc.

What is important to take away is that encryption involves constantly running complex algorithms on all data coming into and out of a device to simultaneously encrypt and decrypt data. It also involves a complete and universally accepted standard for how to encrypt and decrypt data. Every device on a network must be designed to use the same system and processes.

IoT and Security

With the sudden and quick growth of internet connected devices, there is a tremendous lack of security in IoT related devices. The main reasons for this include:

- Devices want to be able to talk to all other devices, and no standard protocol for security has been setup

- To ease the process of connecting a device to a network, the cost and setup associated with security is often pushed to the side

- To keep the costs and sizes of devices down, smaller and simpler processors are being used. Encryption requires a significant amount of additional memory and processing to enable their cryptographic algorithms.

Awareness

We are living in a time where devices are being connected to the internet as unpreceded rates. This opens up new avenues for hackers to gain access to these devices an use them in nefarious ways.

GROWTH IN THE INTERNET OF THINGS
THE NUMBER OF CONNECTED DEVICES WILL EXCEED 50 BILLION BY 2020
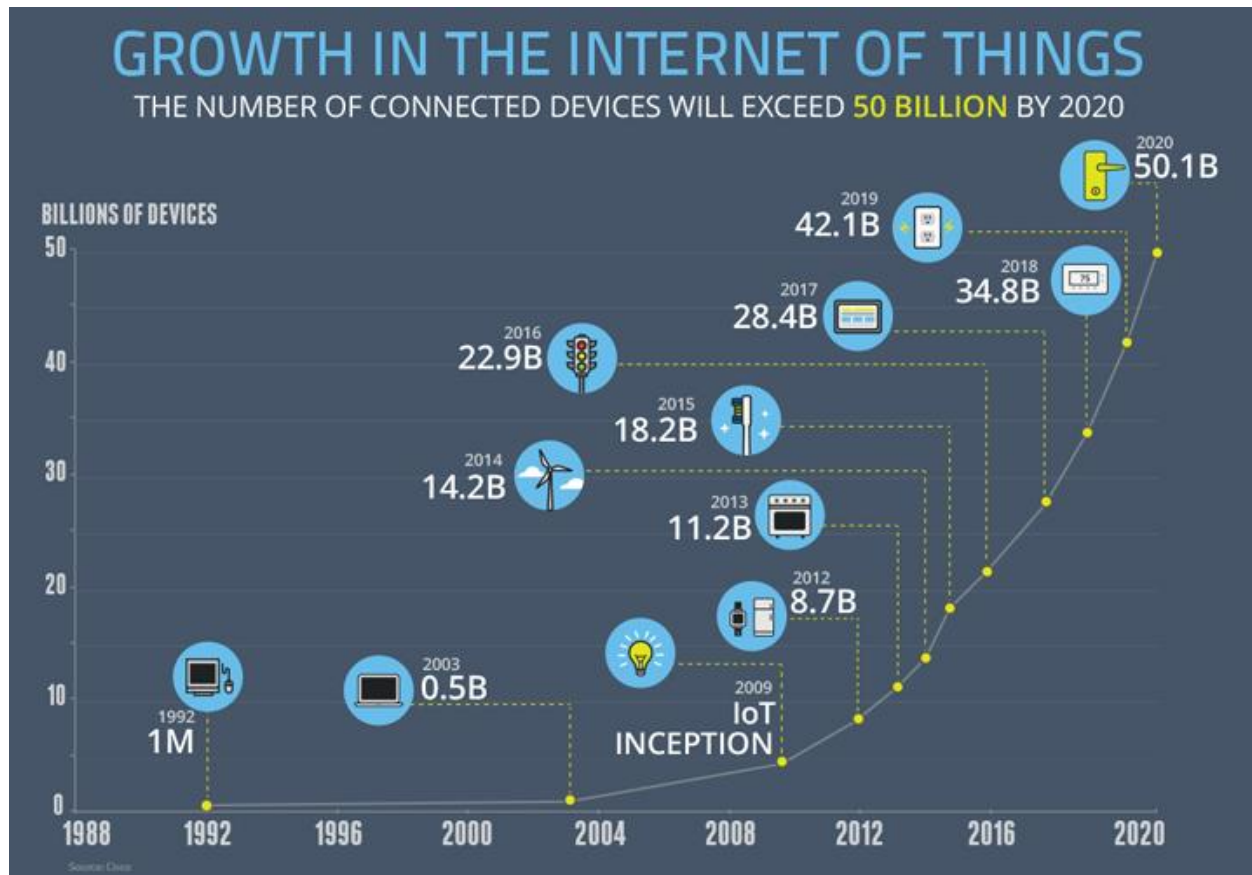
Some examples of insecure IoT devices being hacked include

- In 2016, the largest DDoS (Distributed Denial of Service, where high amounts of devices attempt to contact a server at one time, overloading and shutting it down) attack was launched on service provider Dyn, using a swarm of IoT devices. This lead to huge portions of the internet going down, including Twitter, the Guardian, Netflix, Reddit, and CNN. The devices infected included things like digital cameras and DVR players.

- IoT devices, like pacemakers and defibrillators, are used to monitor and control patients' heart functions and prevent heart attacks. A vulnerability occurred in the transmitter that reads the device's data and remotely shares it with physicians. The FDA said hackers could control a device by accessing its transmitter. Once in, they could deplete the battery or administer incorrect pacing or shocks.

Often we find ourselves in situations where data encryption is either not yet available or not yet implemented. Even in situations where the transfer of data is encrypted, the storage and end user for the data is not encrypted. For example, when sending an email using Gmail, the transfer of the data is encrypted, but once the email resides on Gmail's servers, it is no longer encrypted. This allows the owner of the server to have access to your data. This is common in almost all large online tech companies, including amazon, google, etc. While not necessarily a bad thing, it is important to keep in mind who has access to your data when you share it over the internet. And while we trust some large firms to keep our data secure, there are also many smaller companies that we entrust that do not have the resources to secure our data e.g. – a small doctor's office, small online retailer, online registration for local events, etc.

esp32

Introduction

The esp32 is a microcontroller designed to be an Internet of Things (IoT) platform.

The esp32 is programmed using the Arduin IDE and is very similar to coding a regular Arduino Uno. However, due to its small size, attaching circuitry can be difficult. To facilitate project development, many esp32 breakout boards have been created, very similar to the Arduino family or circuit board.
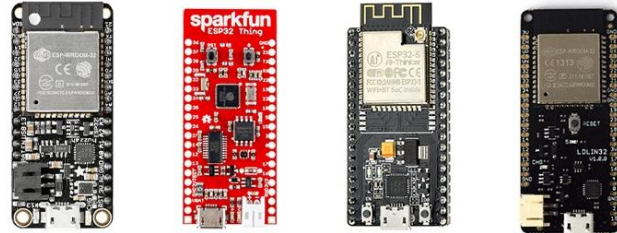
This curriculum will focus on the Acebott breakout version of the esp32, a board designed to be very similar to the standard Arduino Uno.
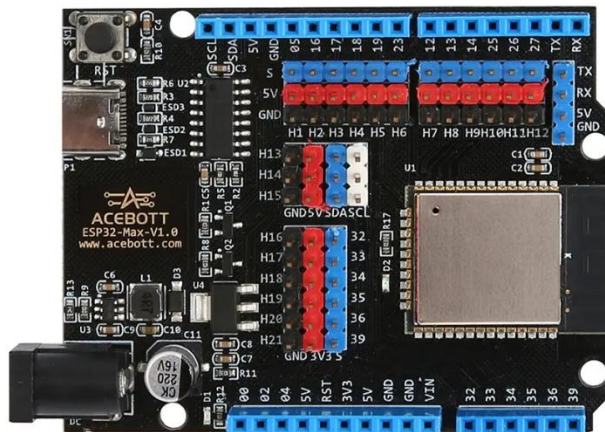
esp32 vs Arduino Uno

	esp32	Arduino
Vin	$7 - 15v$	$7 - 12v$
analogRead	15 pins @ 0 - 3.3v	6 pins @ $0 - 5v$
analogRead Values	0 - 4095	0 - 1025
Digital pins	25 @ 3.3v	14 @ 5v
Max Digital Pin Current	40mA	20mA
analog Write	2 PWM pins: 25 and 26	6 PWM pins: 3, 5, 6, 9, 10, 11
Built in LED	D2	D13

Coding microcontrollers with the Arduino IDE

The Arduino IDE is a versatile compiler, meaning it takes code written in one language and converts it into another. In most cases it is taking high level code written in text and compiling it into binary coding for a microcontroller to use. Although we generally code Arduino boards using the Arduino IDE, we can set up the Arduino IDE to compile code for a variety of microcontrollers. In this case we will set up the Arduino IDE to code the esp32 microcontroller.

Setting up the Arduino IDE for coding
We can use the familiar Arduino IDE to code the esp32, even though it is not an Arduino board.

> **The following 7 steps must be performed to configure the Arduino IDE to code the esp32**

1. Go to File → Preferences.

2. Select the "Additioanl URL's" icon in the corner.

3. Add the following URL's to the board manager:

> http://arduino.esp8266.com/stable/package_esp8266com_index.json
> https://www.arduino.me/package_esp32_index.json
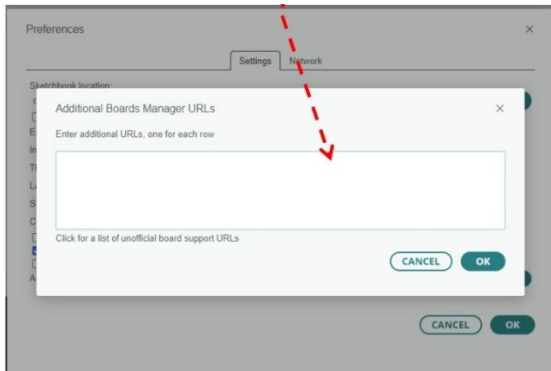
4. Select OK until you return to the Arduino main screen.

5. Search for "esp32" in the BOARDS MANAGER's search bar and install it.

6. After installation is complete, close and reopen the Arduino IDE.
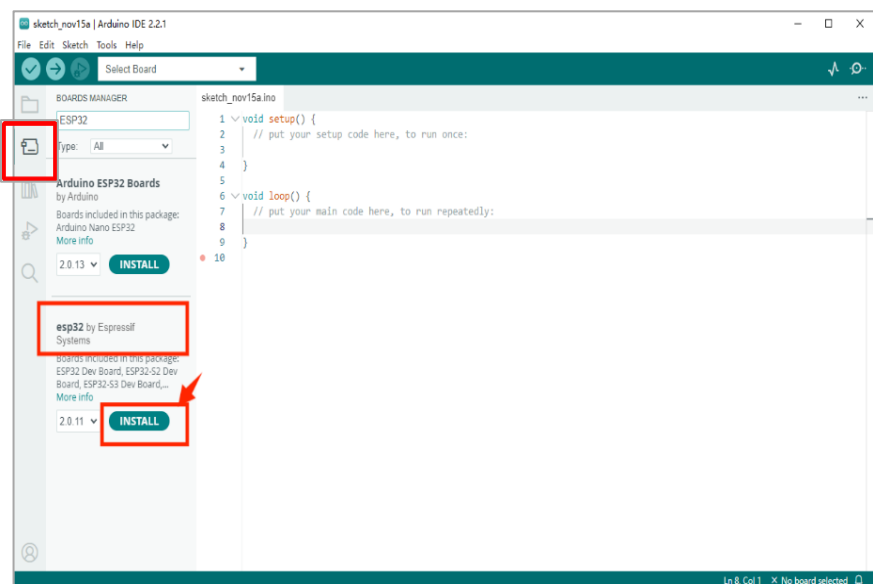
7. Select Tools → Board → esp32 → esp32 Dev Module

You're ready to code the esp32 using the Arduino IDE!

Note: You can increase the uploaed speed under the tools menu if you are not experinceing any upload errors.

In some cases where the board is not recognized, and additoanl driver may need to be installed:

> Windows: https://acebottteam.github.io/acebott-docs-master/getting%20started/Arduino/Download%20CH340%20Driver%20on%20Windows%20System.html
>
> Mac: https://acebottteam.github.io/acebott-docs-master/getting%20started/Arduino/Download%20CH340%20Driver%20on%20MAC%20System.html

HTTP

Standardizing Communication

HTTP stands for **H**yper**T**ext **T**ransfer **P**rotocol. This is a basis for data communication on the internet. The data communication starts with a request sent from a client (e.g. – your computer) and ends with the response received from a web server (e.g. – google.com). Each time two computers communicate with each other using the HTTP protocol, a simple sequence is followed.

1. A website URL starting with "http://" is entered in a web browser from a computer (client). The browser can be a Chrome, Firefox, Edge, Safari, Opera or anything else.

2. Browser sends a request to the web server that hosts the website.

3. The web server then returns a response as a HTML page or any other document format to the browser.

4. Browser displays the response from the server to the user.

The HTTP protocol requires a minimum of information to ensure consistent communication. The HTTP request might contain:

```
HTTP Request Structure
1 GET /home.html HTTP/1.1
2 Host: www.yoursite.com
```

- o The version of the HTTP protocol your computer is using

- o The name of the website it would like to see

- o The page within the website it's looking for

The HTTP response might contain:

```
HTTP Response Structure
1 HTTP/1.1 200 OK
2 Date: Sun, 28 Jul 2013 15:37:37 GMT
3 Server: Apache
4 Last-Modified: Sun, 07 Jul 2013 06:13:43 GMT
5 Transfer-Encoding: chunked
6 Connection: Keep-Alive
7 Content-Type: text/html; charset=UTF-8
8 Webpage Content
```

- o Confirmation that the message was received

- o Time and date of the response

- o Language and protocol that the website is coded in

- o Followed by the actual content of the website

HTML – Building a Website

Tags

Like Arduino and its loops, HTML uses sections indicated by tags between angle brackets < >. The opening and closing tags are the same, except the closing tag has a forward slash (/) in it.

```
<html>
<head>
<title>
    Document title belongs here
</title>
</head>
<body>
    Text and other contents of the page belong here
</body>
</html>
```

<html> <head> <body>
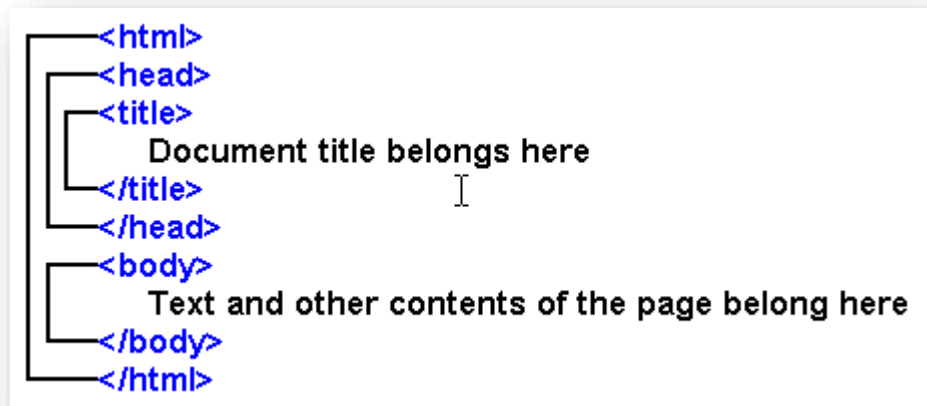
The entire code is contained within <html> tags, with two subsections, <head> and <body>. The basic tag structure of a website looks like this:

```
<html>
    <head>
    </head>
    <body>
    </body>
</html>
```
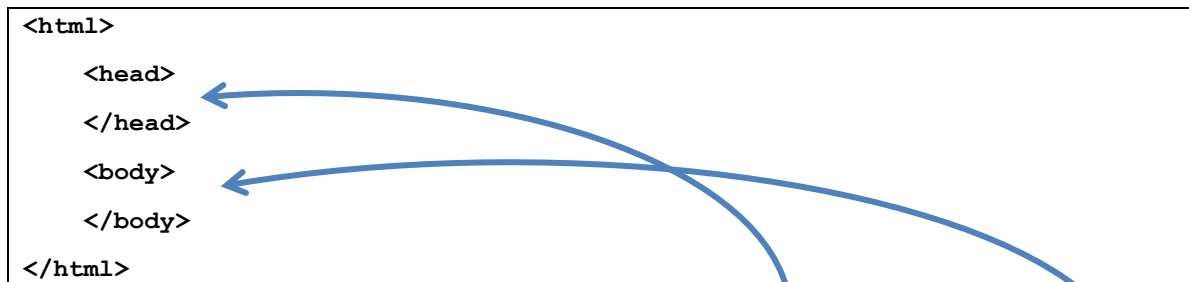
Things that do not appear on the page are placed between the <head> tags. Things that are to be visible on the web page, or apply to the web page content, are placed between the <body> tags.

\<title\>

The title tag, \<title\>, is placed in the \<head\> section of the HTML page and will display text in the top bar of the web browser. This tag is intended to display the web page title.

```
<html>

    <head>

        <title> Center for Initiatives in Jewish Education </title>

    </head>

    <body>

    </body>

</html>
```

\<h1\> \<p\>

The text of the website goes between the \<body\> tags. The format of the text is controlled using the headings tags, e.g.: \<h1\>, or the standard paragraph tag, \<p\>. Any text typed between these tags will be formatted accordingly.
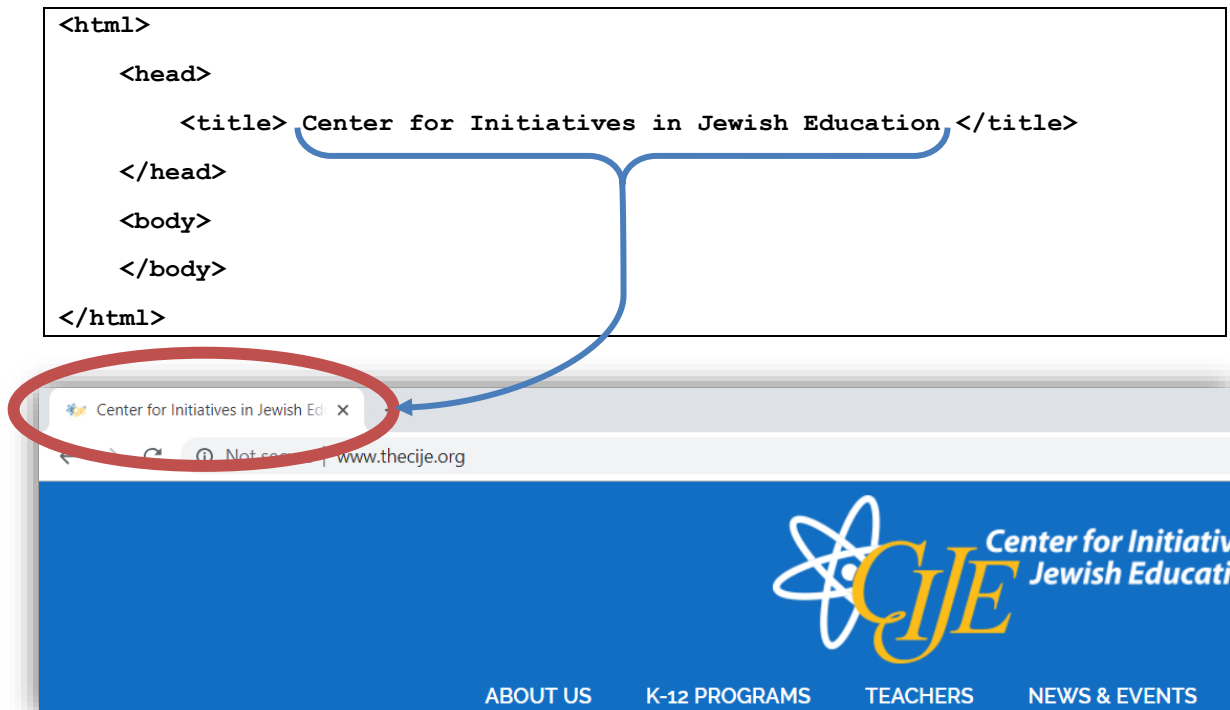
Below is an example code demonstrating different heading and paragraph tags:

For example:
Code placed between the \<h2\> and \</h2\> tags will be formatted like a second level header

```
<!DOCTYPE html>

<html>

    <head>
        <title>Title</title>
    </head>

    <body>
        <h1> H1 Heading </h1>
        <h2> H2 Heading </h2>
        <h3> H3 Heading </h3>
        <h4> H4 Heading </h4>
        <h5> H5 Heading </h5>
        <h6> H6 Heading </h6>
        <p> Regular Text </p>
    </body>

</html>
```

Code   Website

H1 Heading

H2 Heading

H3 Heading

H4 Heading

H5 Heading

H6 Heading

Regular Text

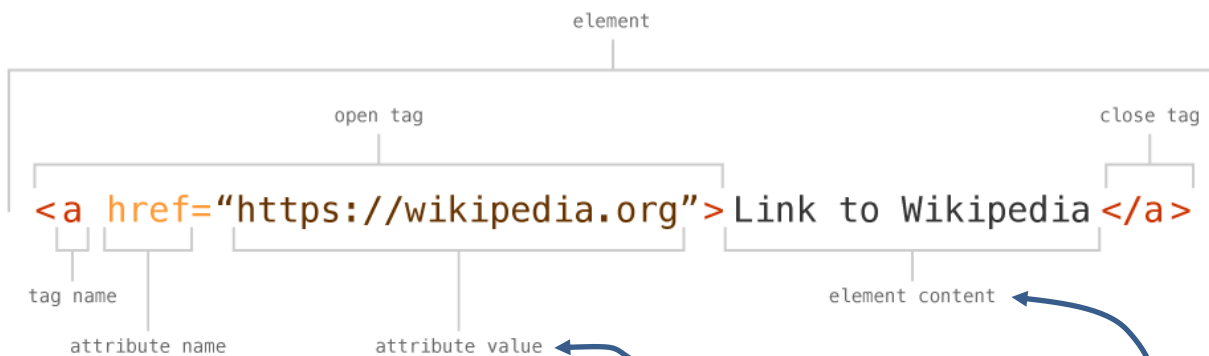An example of different format types used in a webpage:

```
<!DOCTYPE html>

<html>

    <head>

        <title> Testy Web Page </title>

    </head>

    <body>

        <h1> Hello from Arduino! </h1>

        <p> A web page from the Arduino server </p>

    </body>

</html>
```

Testy Web Page × +

← → C G .../Desktop/testy.html

Hello from Arduino!

A web page from the Arduino server

<a> (links)

The <a> tag is used to insert a link in your html webpage. The link will update the URL, or address of the page you are on. It can contain a completely new URL, and thus forward you to a new site, or simply amend your current URL, keeping you on the same page, but with a slightly different URL.

element

open tag   close tag

`<a href="https://wikipedia.org">Link to Wikipedia</a>`

tag name   element content

attribute name   attribute value

➢ All the information fits within the <a> tags

➢ The `href=` portion contains what the link will update the URL to. Note that this information is contained within the opening <a> tag.

➢ The information between the tags is what is displayed on the site

   o To display a clickable button on the site, you can use the <button> tag, with the information between the tags the actual words that will appear on the button

** <button> click here for wikipedia </button> **

click here for wikipedia

Tags Summary
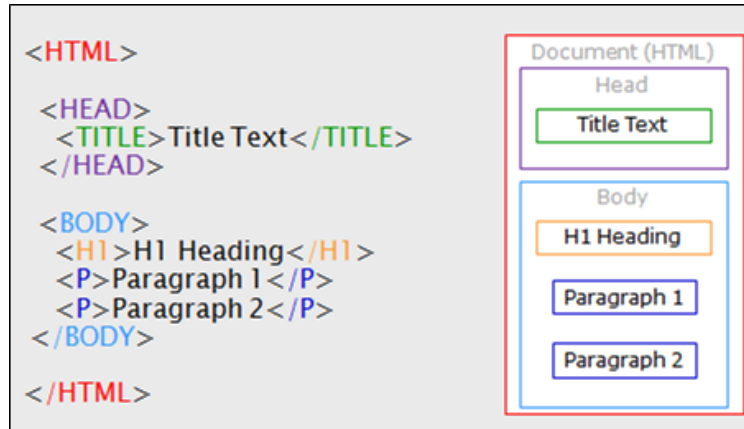
Writing HTML in Arduino

In order to host the server on your esp32, we need to follow certain syntax developed by the esp32 library. All HTML coding must be within one single string. We can use the html+= to continuously code and add each line the string, and then send that string to the browser.

```
String html = "<!DOCTYPE html>"; //web page is made using HTML
html += "<html>";
  html += "<head>";
      html += "<title> Ethernet Tutorial </title>";
  html += "</head>";
  html += "<body>";
      html += "<h1> A Webserver Tutorial </h1>";
      html += "<h2> Observing State Of Switch </h2>";
      html += "<h2> Switch is:   </h2>";
      if (digitalRead(8))    {
      html += "<h3> ON </h3>";
      }
      else    {
      html += "<h3> OFF </h3>";
      }
  html += "</body>";
html += "</html>";
```

> When using the Arduino IDE, All HTML coding must be added to one long string variable

Display analogRead Data on Website

The following example, when placed in the <body> tags, will display the output from the analogRead pin on your website.

```
int sensorReading = analogRead(A0);
html += "analog input is";
html += String(analogRead(39));
```

Display "Button State" on a website

The following code would print different text on the website based on a button press:

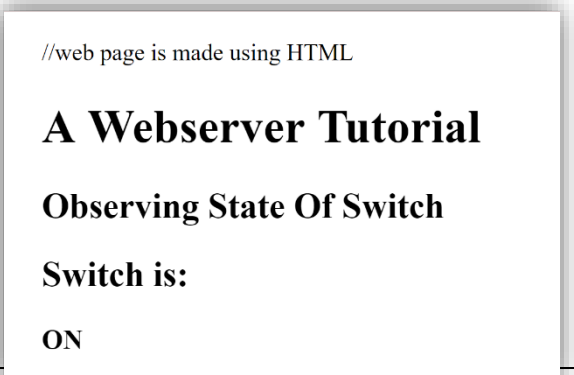**NOTE: The HTML is incorrectly formatted as it's not within the `String html` command**

```
<!DOCTYPE html> //web page is made using HTML

<html>

   <head>

   </head>

   <body>

     <h1>A Webserver Tutorial </h1>

     <h2>Observing State Of Switch</h2>

     <h2>Switch is:   </h2>

       if (digitalRead(8)== HIGH) {

       <h3> ON </h3>   }

       else  {

       <h3> OFF </h3> }

   </body>

</html>
```
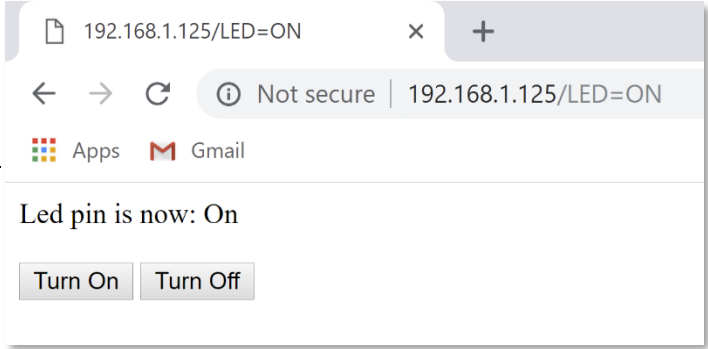
//web page is made using HTML

A Webserver Tutorial

Observing State Of Switch

Switch is:

ON

Turning on a light with a "button"

The following example, when placed in the <body> tags, will append the URL of your site with either an "LED=ON" or "LED=OFF" when one of the buttons are pushed

```
html += "<html>";
  html += "Led pin is now: ";
    if (value == HIGH)
      html += ("On");
    else
      html += ("Off");
  html += ("<br><br>");
  html += ("<a href=\"/LED=ON\"><button>Turn On </button></a>");
  html += ("<a href=\"/LED=OFF\"><button>Turn Off </button></a><br />");
html += ("</html>");
```

192.168.1.125/LED=ON

Not secure | 192.168.1.125/LED=ON

Apps   M Gmail

Led pin is now: On

Turn On   Turn Off

The example code on the following pages will discuss how to operate the LED based on the updated URL suffixes.

IMPORTANT: when coding, a backslash \ must precede any " within the <a> tags for Arduino

Full Example Code - Posting information on a webpage

```
#include <WiFi.h>

const char* ssid = "enter between quotes";
const char* password = "enter between quotes";

WiFiServer server(80);

void setup() {
 Serial.begin(9600);
 WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
 Serial.println("WiFi connected");
 server.begin();
 Serial.println("Server started");
 Serial.print("The URL to connect: http://");
 Serial.print(WiFi.localIP());
 Serial.println("/");
}

void loop() {
  // Check if a client (browser) has connected
  WiFiClient client = server.available();
  if (!client) {
    return;
  }

    if (client && client.connected()) {
        String html = "HTTP/1.1 200 OK\r\n";
        html += "Content-type:text/html\r\nConnection: close\r\n\r\n";
        html += "<!DOCTYPE html>";
        html += "<html>";
        html += "<head><meta http-equiv='refresh' content='3'></head>";
        html += "<body>";
        html += "<h1>";
        html += "Analog Value: ";
        html += String(analogRead(39));
        html += "</h1>";
        html += "</body>";
        html += "</html>";
        client.print(html);
        client.stop();
    }
}
```

Enter the username and password for the Wi-Fi network you will be using

If your network assigns a "port" to each device, enter the port your Arduino is on here. By default, all "servers" are set to 80

While the Wifi is NOT connected, the Arduino will print dots on the screen until the connection is established

The esp32 is "hosting" the information for anyone to come and see. It is the "server"

Open the serial monitor to see your device's IP address. This is what you type into a browser to see the website hosted on your "server"

Checking to see if any browsers asked to see our website. Client = TRUE if someone asked

If no browsers opened, Client is NOT TRUE, so return to top of the loop

When the server sends information to your browser (the client), it includes what type of information and what language it will be sending

The server will "serve", or refresh the page, every 3 seconds, regardless of the readings

The last section of our code is the html of our website

Full Example Code - Controlling Arduino from a webpage

```cpp
#include <WiFi.h>

const char* ssid = "enter between quotes";
const char* password = "enter between quotes";

WiFiServer server(80);
int ledPin = 2;

void setup() {
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("WiFi connected");
  server.begin();
  Serial.print("Use this URL to connect: http://");
  Serial.print(WiFi.localIP());  Serial.println("/");
}

void loop() {
  // Check if a client has connected
  WiFiClient client = server.available();
  if (!client) {
    return;  //go back to top of the void loop
  }

  // Read the URL line from the client (browser)
  String request = client.readString();
  Serial.println(request);

  if (request.indexOf("/on") > 0) {
    digitalWrite(ledPin, HIGH);
  }
  if (request.indexOf("/off") > 0) {
    digitalWrite(ledPin, LOW);
  }
//******* WEBSITE **************
  String html = "HTTP/1.1 200 OK\r\n";
  html += "Content-type:text/html\r\nConnection: close\r\n\r\n";
  html += "<!DOCTYPE html>";

  html += "<html>";
  html += "<body>";
  html += "<h1> esp32 LED Control</ h1>";
  html += "<p>";
  html += "<a href = '/on'><button> ON</ button></ a>;";
  html += "<a href = '/off'><button> OFF</ button></ a>";
  html += "</ p>";
  html += "</ body>";
  html += "</ html> ";

  client.print(html);
  client.stop();
}
```

Checking to see if any browsers asked to see our website

If no browsers opened, return to top of the loop

The "client", or browser, sends a string of information (the URL) back to your server (esp32) when you click on a link

"indexOf" checks if a certain piece of information is contained within the string (the URL) from the browser. Here it's checking if the word "on" is the data. If it is, it will turn on (HIGH) pin 2.

ESP32 LED Control

ON: OFF

Based on what you click on, the "buttons" on the site will return different strings of data