

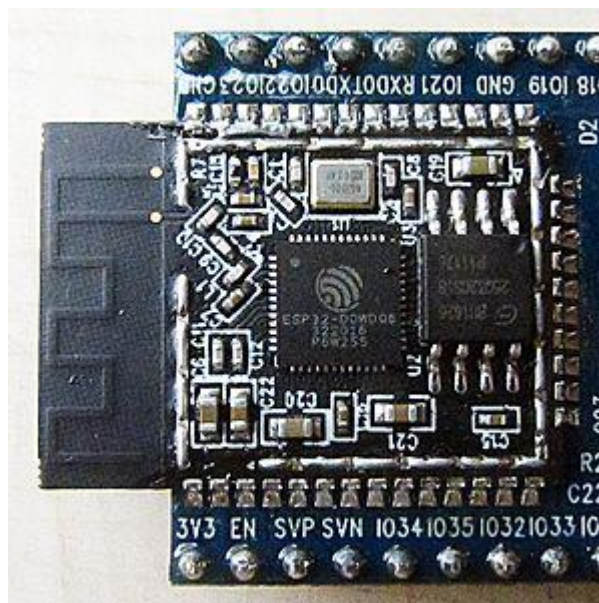
Wi-Fi 接続…通常のマイコンにはない無線機能

《何ができる?》ESP32 の大きな特徴に無線機能がある。

無線接続:

- Wi-Fi 802.11 b/g/n/e/i
- Bluetooth v4.2 BR/EDR と BLE

このうち、Wi-Fi は LAN 接続もインターネット接続も可能なので、センサーの情報をクラウドにアップしたり、構内・校内で PC やスマホとのやりとりが手軽にできる。



Wi-Fi ライブラリ

Wi-Fi 接続のライブラリ関数は、現在は Arduino の標準のライブラリとなりパッケージのインストールは不要で、スケッチの最初に<Wi-Fi.h>ファイルのインクルードを記述しておけば各種ライブラリが利用可能となる。

```
#include <WiFi.h>
```

利用可能なクラス

- ・ WiFi クラス…Wifi 無線接続に関するクラス。
- ・ IPAddress クラス…自分の IP アドレス/サブネットマスク、ゲートウェイ IP アドレスに関するクラス。
- ・ Server クラス…シンプルな http サーバーを立てることができるクラス
- ・ Client クラス…ブラウザなどと同じような http クライアントアクセスが行えるクラス。
- ・ UDP クラス…UDP プロトコルを使って通信するクラス。

```
WiFi.begin(ssid, pass);
```

指定した **ssid** と **pass** で Wi-Fi に接続する

【パラメータ】

ssid: 接続したい Wi-Fi ネットワークの SSID (Service Set Identifier)
pass: WPA(Wi-Fi Protected Access) 認証のためのパスワード

【戻り値】

WL_CONNECTED : ネットワークにつながった
WL_IDLE_STATUS: ネットワークにつながっていないが電源オンである

《サンプル①》 Wi-Fi の接続を行って、接続後自身の IP アドレスをシリアル表示する。

	シリアルモニタ
<pre> 1 #include <WiFi.h> 2 3 // 接続先のSSIDとパスワード 学内CampusIoT 4 const char ssid[] = "CampusIoT-WiFi"; 5 const char passwd[] = "0b8b413f2c0fa6aa90e085e9431abbbf1fa1b2bd2db0ecf4ae9ce4b2e87da770c"; 6 7 void setup() { 8 Serial.begin(115200); 9 // WiFi接続シーケンス 10 WiFi.begin(ssid, passwd); 11 Serial.print("WiFi connecting..."); 12 while(WiFi.status() != WL_CONNECTED) { 13 Serial.print("."); 14 delay(100); 15 } 16 Serial.print(" connected. "); 17 Serial.println(WiFi.localIP()); 18 } 19 20 void loop() { 21 } 22 </pre>	<pre> WiFi connecting..... connected. 10.101.0.x ↑ 1行で表示される。xは任意。 「.」の数は接続されるまでず っと繰り返し表示される。 </pre>

演習では、上記サンプルを動かした後、http サーバーに書きかえてみる。

上記サンプル①では、4 行目と 5 行目で学内の接続先 WiFi の接続情報をセットしている。

従来はソースに SSID とパスワード（WiFi のパスワード）を入力していたが、パスワードそのものを文字列として平文でソースに入力すると、リバースエンジニアリングによって文字列を解読される可能性がある。

そのため、今年度よりハッシュ化されたパスワードを利用する。

ハッシュ化されたパスワードは「Arduino core for ESP32 WiFi chip」ライブラリ（ESP32 コアライブラリと呼ばれる libaries フォルダに入らないコア部分のライブラリ）が最新であれば問題なく動作する。以下の課題では、次に示すコードを利用してほしい。

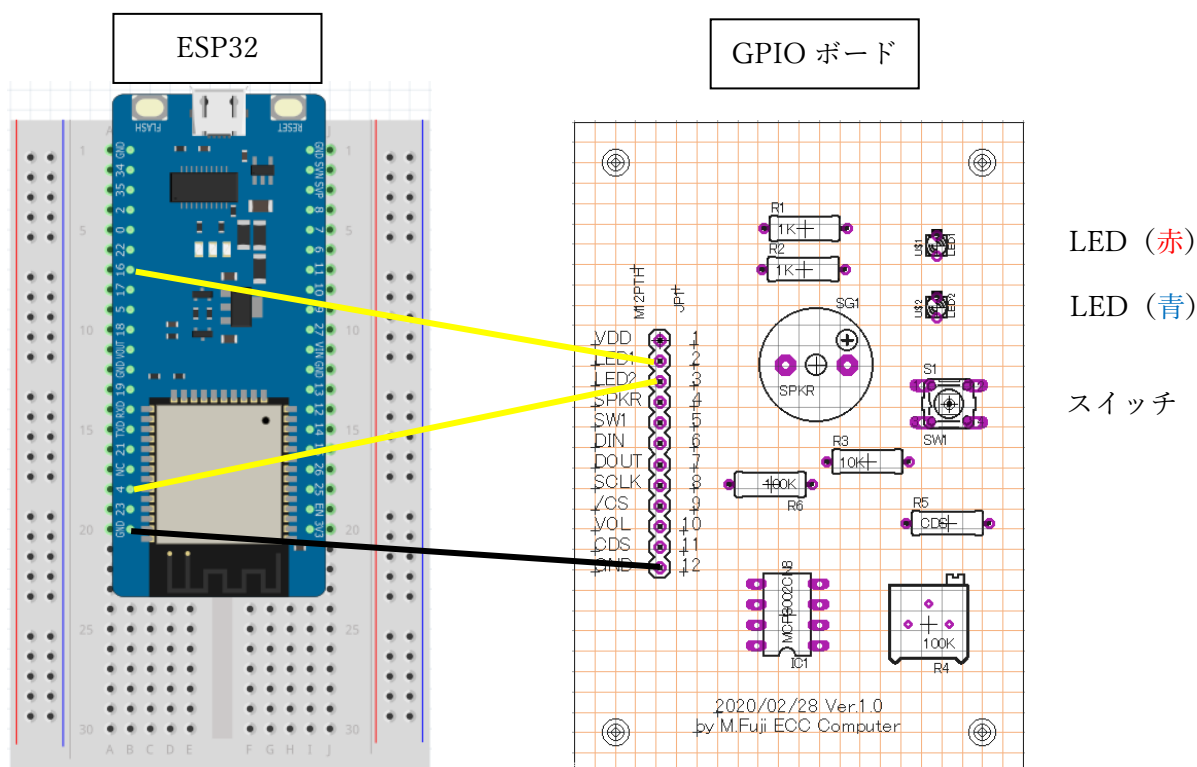
学内マイコン/IoT 演習用 WiFi の接続設定情報となるコード
<pre> // 接続先の SSID とパスワード 学内 CampusIoT const char ssid[] = "CampusIoT-WiFi"; const char passwd[] = "0b8b413f2c0fa6aa90e085e9431abbbf1fa1b2bd2db0ecf4ae9ce4b2e87da770c"; </pre>

第9回 演習

【第9回の目標】

ESP32 ボードで Wi-Fi 接続を行う。Wi-Fi 接続を行った後 http サーバーを実行してブラウザからアクセスできるようにする。

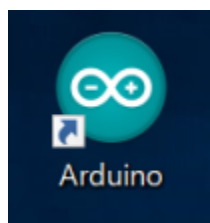
【1. ESP32 と電子工作部品との接続】



上記実体配線図を元に、5 本のジャンパーコードの配線を行う。接続するピンは以下の一覧の通り。


- ① ジャンパーコード (黄) ESP32 の I016 - GPIO ボード LED1 (2 番ピン)
- ② ジャンパーコード (黄) ESP32 の I04 - GPIO ボード LED2 (3 番ピン)
- ③ ジャンパーコード (黒) ESP32 の GND - GPIO ボード GND (12 番ピン)

【2. Arduino スケッチのサンプルプログラムをプログラム実行】



デスクトップのアイコンをダブルクリックして
Arduino IDE を起動する。

【課題】プロジェクト名「kad20_WifiServerLedOn」

まずは、《サンプル①》を打ち込んで、動作確認をする。 シリアルモニタを開き IP アドレスが取得できているか確認する。シリアル表示される IP アドレスは、Web サーバーのアクセス先になるため覚えておく。

IP アドレスが表示されたら以下の様書き加える。赤矢印の 5 か所を書き加えて、《サンプル①》と同じ動作になるか確認。

```
1 #include <WiFi.h>
2
3 // 接続先のSSIDとパスワード 学内CampusIoT
4 const char ssid[] = "CampusIoT-WiFi";
5 const char passwd[] = "0b8b413f2c0fa6aa90e085e9431abbbf1fa1b2bd2dk
6
7 const int ledPin = 16; //赤色LED
8 WiFiServer server(80);
9
10 void setup()
11 {
12     delay(1000);
13     Serial.begin(115200);
14     pinMode(ledPin, OUTPUT); // set the LED pin mode
15
16     // WiFi接続シーケンス
17     Serial.print("Connecting...");
18     WiFi.begin(ssid, passwd);
19     while (WiFi.status() != WL_CONNECTED) {
20         delay(100);
21         Serial.print(".");
22     }
23     Serial.println("connected");
24     Serial.println(WiFi.localIP());
25
26     server.begin();
27 }
28
```

《サンプル①》と同じ IP アドレスが表示されたら、http サーバーとしてアクセスできる状態は整った。8 行目のグローバル変数の宣言のような構文は、C++のクラスのコンストラクタ宣言で、WiFiServer クラスの server という名前のインスタンスが作られている。コンストラクタの引数に 80 として初期化され、80 番ポートを Listen ポートとした http サーバーが ESP32 ボード上で準備された状態になる。

```
8 WiFiServer server(80);
```

その後、26 行目の WiFiServer#begin() メソッドで実際にサーバーが開始される(#はインスタンスメソッドの説明記号)。

ただし、http のリクエスト&レスポンスについての記述が無く、<http://<IP アドレス>/>にアクセスしても反応はない。

次に、loop() 関数内のスケッチを加える。以下のソースを loop() 関数内に貼り付けて、マイコンに書き込む(※loop 関数宣言行と最後の波かっこは貼り付けないように注意)。**★テキストファイルを渡します**

```
void loop() {
  WiFiClient client = server.available();

  if (client) {
    Serial.println("new client");
    String currentLine = "";
    while (client.connected()) {
      if (client.available()) {
        char c = client.read();
        Serial.write(c);
        if (c == '\n') {
          if (currentLine.length() == 0) {
            client.println("HTTP/1.1 200 OK");
            client.println("Content-type:text/html");
            client.println();
            client.println("<!DOCTYPE html>");
            client.println("<html>");
            client.println("<head>");
            client.println("<meta name='viewport' content='initial-scale=1.5'>");
            client.println("</head>");
            client.println("<body>");
            client.println("<h1>ESP-WROOM-32</h1>");
            client.println("<h2>Wi-Fi http Server </h2>");
            client.println("</body>");
            client.println("</html>");
            client.println();
            //WiFiServer のレスポンス
            digitalWrite(ledPin, HIGH);
            delay(1000);
          }
        }
      }
    }
  }
}
```

kad20-loop-only.txt

```

        break;
    } else {
        currentLine = "";
    }
    } else if (c != '\r') {
        currentLine += c;
    }
    }
}
client.stop();
Serial.println("client disconnected");
digitalWrite(ledPin, LOW);
}
}

```

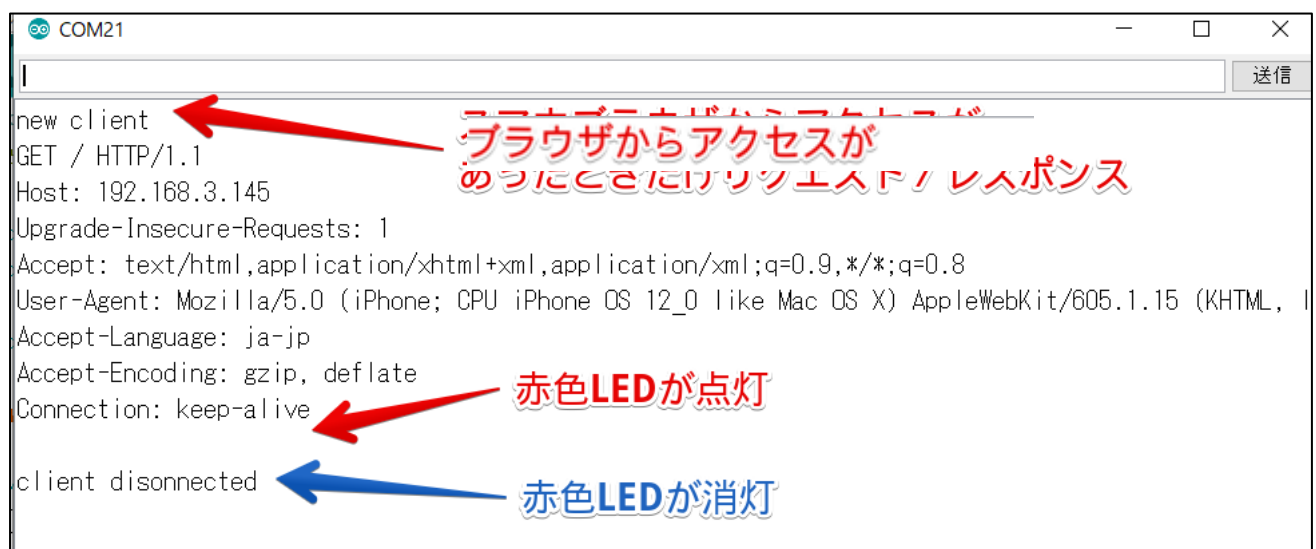
loop()関数直下に、WifiClient クラスの client インスタンスが宣言され、server.available() メソッドでサーバーとやりとりされる http クライアントオブジェクトが代入される。

```

28 void loop() {
29   WiFiClient client = server.available();
30
31   if (client) {
32     Serial.println("new client");

```

重要ポイントは、後ほど接続確認を行う PC ブラウザでアクセスがあったときのみ client オブジェクトが代入され、アクセスが無いときは、client インスタンスは null のまま 31 行の if 文の中のブロックはすり抜ける。31 行の if 文の中で http リクエスト／レスポンスを受けるとシリアルモニタの表示は以下のようになる。



【課題】の動作確認

Arduino のスケッチを書き込んだら、シリアルモニタに表示される IP アドレスを使った URL で、スマホやタブレットで接続確認する。

①シリアルモニタでもし以下のように出たら http サーバーの URL は「 <http://10.101.0.44> 」となる。

```
Connecting.....connected  
10.101.0.44
```

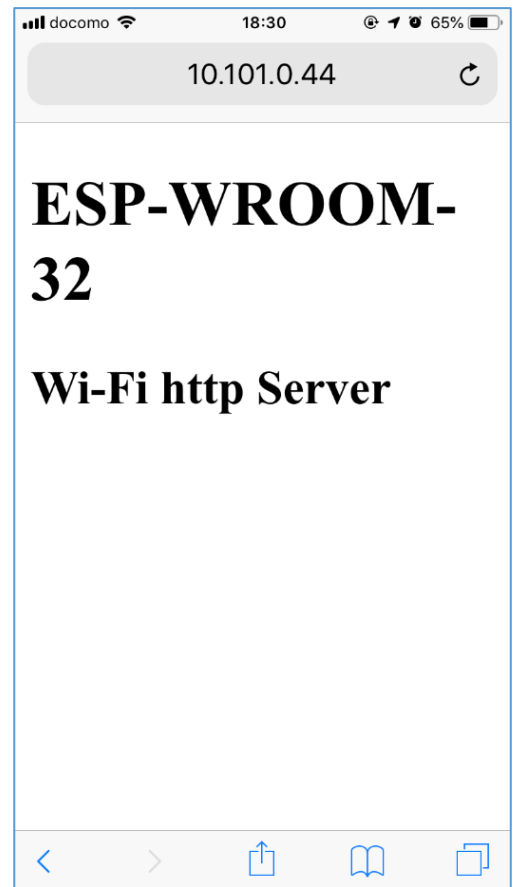
②PC のブラウザを開き、①で確認した URL を打ち込むと、右図のような画面になれば OK!

③リロードボタンを押すたびに、赤色 LED が点灯・消灯すれば正常に動作している。

① 接続が確認できたら、先生の実習のチェックをしやすくするために、QR コードを作っておいてください。下記は <https://m.qrqrq.com/> サイト

学生は、PC（実習室のデスクトップ PC または自分のノート PC）のブラウザで動作確認します。

講師は、CampusIoT の WiFi の設定を済ませたスマホまたはタブレットで QR コードを読みアクセス確認します。



【課題】 プロジェクト名「kad21_WifiLedOnOffServer」

次に、右図のような、LED をオン・オフさせることができるローカル Web サービスを作成する。

Arduino スケッチの宣言部分と `setup()` 関数部分は、前の課題と同じでよい。`loop()` 関数の内容を以下に示すようにする。

コピー＆ペーストすればよい。

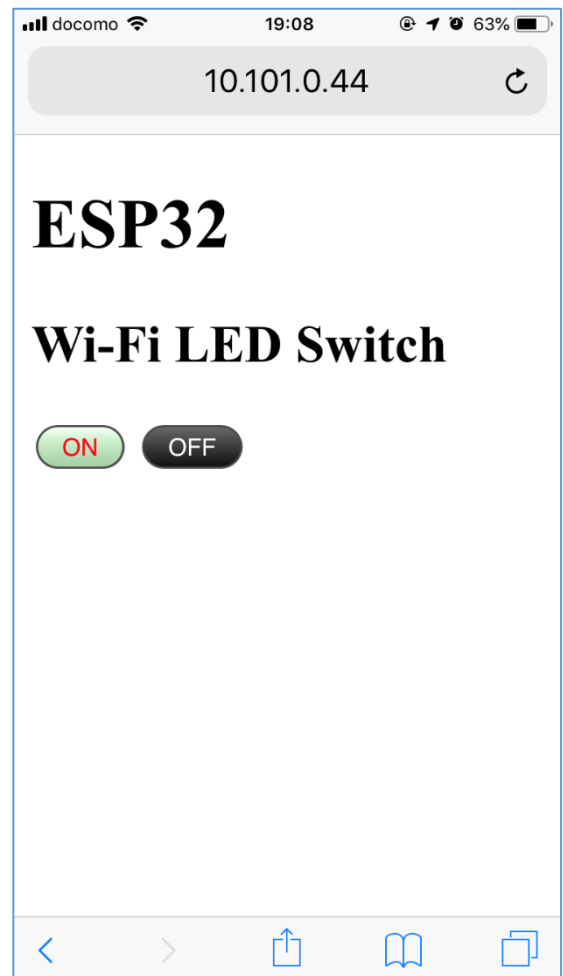
☆テキストファイルを渡します

【課題】の動作確認

Arduino のスケッチを書き込んだら、シリアルモニタに表示される IP アドレスを使った URL で、PC ブラウザで接続確認する（講師はスマホやタブレットで確認）。

① Safari や Chrome などスマホ or タブレットのブラウザを開き、上記 URL を打ち込むと、右図のような画面になれば OK！

② [ON] ボタンをタップすれば赤色 LED が点灯。[OFF] ボタンをタップすれば赤色 LED が消灯すれば OK！



```
void loop() {
  WiFiClient client = server.available();

  if (client) {
    Serial.println("new client");
    String currentLine = "";
    while (client.connected()) {
      if (client.available()) {
        char c = client.read();
        Serial.write(c);
        if (c == '\n') {
          if (currentLine.length() == 0) {
            client.println("HTTP/1.1 200 OK");
          }
        }
      }
    }
  }
}
```



```

    client.println("Content-type:text/html");
    client.println();

    client.println("<!DOCTYPE html>");
    client.println("<html>");
    client.println("<head>");
    client.println("<meta name='viewport' content='initial-scale=1.5'>");
    client.println("</head>");
    client.println("<body>");
    client.println("<form method='get'>");
    client.println("<h1>ESP32</h1>");
    client.println("<h2>Wi-Fi LED Switch</h2>");
    client.println("<input type='submit' name=btn value='ON'
style='background-color:#88ff88; color:red;'>");
    client.println("<input type='submit' name=btn value='OFF'
style='background-color:black; color:white;'>");
    client.println("</form>");
    client.println("</body>");
    client.println("</html>");
    client.println();
    break;
} else {
    currentLine = "";
}
} else if (c != ' r') {
    currentLine += c;
}

if (currentLine.endsWith("GET /?btn=ON")) {
    digitalWrite(ledPin, HIGH);
}
if (currentLine.endsWith("GET /?btn=OFF")) {
    digitalWrite(ledPin, LOW);
}
}
}

client.stop();

```

```
Serial.println("client disconnected");  
}  
}
```

【課題】のシリアルモニタ画面

Wi-Fi 接続後、ブラウザからアクセスして[ON]ボタンをタップしてLEDを点灯、[OFF]ボタンをタップしてLEDを消灯するところまでの流れ。

```
Connecting.....connected  
10.101.0.44 ← Wi-Fi 接続でき、IP アドレスを表示  
new client  
GET / HTTP/1.1 ← 最初「 http://10.101.0.44 」にアクセスした  
Host: 10.101.0.44  
Upgrade-Insecure-Requests: 1  
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8  
User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 12_0 like Mac OS X) AppleWebKit/605.1.15  
(KHTML, like Gecko) Version/12.0 Mobile/15E148 Safari/604.1  
Accept-Language: ja-jp  
Accept-Encoding: gzip, deflate  
Connection: keep-alive  
  
client disconnected  
new client ← [ON]ボタンをタップして submit した  
GET /?btn=ON HTTP/1.1 ← [ON]ボタンの属性 name=btn value='ON'によりクエリができた  
Host: 192.168.3.145  
Connection: keep-alive  
Upgrade-Insecure-Requests: 1  
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8  
User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 12_0 like Mac OS X) AppleWebKit/605.1.15  
(KHTML, like Gecko) Version/12.0 Mobile/15E148 Safari/604.1  
Referer: http://192.168.3.145/  
Accept-Language: ja-jp  
Accept-Encoding: gzip, deflate  
  
client disconnected  
new client ← [OFF]ボタンをタップして submit した  
GET /?btn=OFF HTTP/1.1 ← [OFF]ボタンの属性 name=btn value='OFF'によりクエリができた  
Host: 192.168.3.145  
Connection: keep-alive
```

```
Upgrade-Insecure-Requests: 1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 12_0 like Mac OS X) AppleWebKit/605.1.15
(KHTML, like Gecko) Version/12.0 Mobile/15E148 Safari/604.1
Referer: http://10.101.0.44/?btn=ON
Accept-Language: ja-jp
Accept-Encoding: gzip, deflate

client disconnected
```

スケッチにある[ON][OFF]のボタンが submit される際にできるクエリ文字列が URL に追加される。

[ON]ボタン submit なら「btn=ON」が、[OFF]ボタンなら「btn=OFF」が、URL 文字列の後に「?」をつけた後ろに追加されてリクエストされる。

```
53 |         client.println("<input type='submit' name=btn value='ON' style='backgroun");
54 |         client.println("<input type='submit' name=btn value='OFF' style='backgroun");
```

currentLine 文字列変数は、ブラウザからのリクエストで送られてくる文字が改行されるまで追加したもの。currentLine 文字列変数が、クエリ文字列で終わるとき、その内容に応じて LED をオン・オフする。

```
67 |         if (currentLine.endsWith("GET /?btn=ON")) {
68 |             digitalWrite(ledPin, HIGH);
69 |         }
70 |         if (currentLine.endsWith("GET /?btn=OFF")) {
71 |             digitalWrite(ledPin, LOW);
72 |         }
```

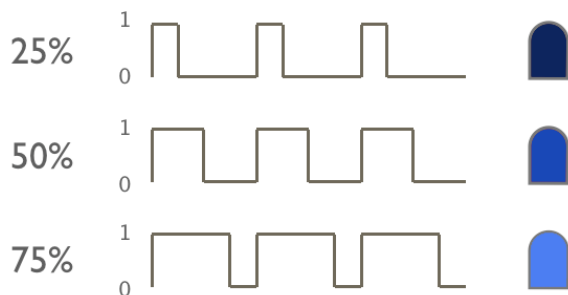
【課題】 プロジェクト名「kad22_WifiLedPwmServer」

最後に、右図のような、青色 LED の明るさを PWM で変化させることができるローカル Web サービスを作成する。

5 個のラジオボタンと [Send] ボタンを submit 用にする。ラジオボタンの属性は「`name='pwm'`」にする。

[Send] ボタンの文字の色は青にしておくこと。

25, 50, 75%のラジオボタンを選択し [Send] ボタンを押すと以下の様に青色 LED の輝度が変化する。



100%を選択し [Send] ボタンを押すと最大 (255) の明るさになり、0%を選択すると消えるようにする。

