WiFi with the ESP8266 and Arduino
Transmit analog values to the ESP8266

Arduino installation for the ESP8266

See
http://weigu.lu//microcontroller/tips_tricks/wemos_tips_tricks/index.html

http://staff.ltam.lu/feljc/creativelab/snyder2/snyder2_firmware_1.pdf

Revision of server basics: switching ON / OFF

See http://staff.ltam.lu/feljc/electronics/arduino/WiFi_01.pdf

This describes how to switch a LED by using the HTTP protocol.
Basically the WEMOS server generates an access point displaying a web page with a link to switch the LED on and another one to switch it off.

The reaction of the server is defined in the setup part:

```
…
server.on("/", handleRoot);
server.on("/ledon", ledon);
server.on("/ledoff", ledoff);
server.begin();
…
```

where ledon and ledoff are the functions to switch the LED on or off.

Now the LED can be switched on and off by typing

```
192.168.4.1/ledon       or
192.168.4.1/ledoff
```

in a browser connected to the accesspoint with IP 192.168.4.1.

But the interaction is always binary, and cannot very well be used to control a robot:
http://staff.ltam.lu/feljc/creativelab/snyder2/snyder2_firmware_1.pdf

A simple solution: use HTTP requests

The HTTP request syntax normally used for queries like a web search allows the transmission of any values along a HTTP connection.

So a PWM value could for example be sent to the server by typing

http://192.168.4.1/other?pwm=200

in a browser (or more sophisticatedly by moving a slider sending its values by using Javascript in the web page).

The Arduino part can be the same as described above, we only have to add the reaction to the request.

This has to be done in the **setup funtion**:

```
void setup() {
      pinMode(led, OUTPUT);
      ...
      WiFi.softAP(ssid, password);
      WiFi.softAPConfig(IPaddr, IPaddr, IPmask);

      ...
      server.on("/", handleRoot);
      server.on("/ledon", ledon);
      server.on("/ledoff", ledoff);
      server.on("/other",handleArg );
      server.begin();
      ...
}
```

Now we have to define the function handleArg reacting to the request:

```
void handleArg() {

    String pwmarg = server.arg("pwm");

    if (pwmarg!= ""){
        int pwmvalue = pwmarg.toInt();
        analogWrite(pwmpin, pwmvalue);
        server.send(200, "text/plain", "");
        }
     }
```

This solution is working fine and it is simple, but as it is based on HTTP, it is really slow!

The HTTP protocol has much overhead slowing down the communication. For example the connection is always opened and cloesed on every request.

A better solution: use websockets

The overhead of a websocket communication is much lesser than that of HTTP.
For example the connection channel stays open as long as the communication exists, and there is no metacommunication like headers to be sent. This makes the communication much faster.

The establishment of a websocket communication follows two steps:

- Contact the server (the ESP8266) via HTTP on port 80
- Upgrade the communication to a websocket communication on port 81.

This is done by using a HTML page, like we did in the HTTP example, but we have to use some javascript for the websocket communication:

```
<!DOCTYPE html>
<html>

<head>
  <meta http-equiv="content-type" content="text/html;charset=utf-8" />
    <title>LED Control</title>
    <STYLE> input{width: 50%}  </STYLE>

    <script type = "text/javascript">

      // ------------------------------------------------
      // Connection
      var connection = new WebSocket('ws://'+location.hostname+':81/',
      ['arduino']);

      connection.onopen = function () {
            connection.send('Connect ' + new Date());
      };

      connection.onerror = function (error) {
          console.log('WebSocket Error ', error);
      };

      connection.onmessage = function (e) {
          console.log('Server: ', e.data);
      };

      connection.onclose = function(){
          console.log('WebSocket connection closed');
      };

      // ------------------------------------------------

      function send_data() {
          // get slider value
          var dat = document.getElementById('slider_01').value;

          // make string
          var rstr = dat.toString();
          console.log('dat: ' + rstr);

          // and send it
          connection.send(rstr);
      }
      //-------------------------------------------------
    </script>
</head>

<body>
        <header> <h1>LED Control</h1> </header>
        <input class="enabled" id="slider_01" type="range" orient="horizontal"
min="0" max="1023" step="1" oninput="send_data();" value="0">
</body>
</html>
```

The visible part of the html page (the body) is very short, it consists of a header and a slider named slider_01, with a value range from 0...1023.
When the slider is moved, the javascript function send_data() is called.

The javascript part resides in the header.

It creates a Websocket object and has functions reacting to opening and closing of a connection, and to an error or a message.

The send_data function reads the slider value, converts it to a text string and sends it over the websocket connection.

Remember that this is the HTML page that the ESP8266 must send at the beginning of any connection. It can be located in the ESP8266 memory as a big string. (or as a file in the SPIFFS file system that can be updated over the air, but this is a bit more complicated, so we stay with the simple example).

To store the web page as a raw string, it has to be declared in a special form:

```
String html = R"***(
<!DOCTYPE html>
<html>
      . . .
      . . .
</html>
)***";
```

It is no bad idea to put it in an extra tab.

A simple websocket program

This program is a demo example that provides a WiFi access point and a web page with a slider.

A tablet can connect to the web page and control the pwm value for a LED or a motor with the slider.

First we define pwmpin, libraries, ssid, password and IP address :

```
#define pwmpin 13                  // =D7 on board
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#include <WebSocketsServer.h>

ESP8266WebServer server(80);
WebSocketsServer webSocket(81);

const char* ssid = "myAccesspoint";
const char* password = "myAccesspoint";
IPAddress IPaddr (192, 168, 168, 168);
IPAddress IPmask(255, 255, 255, 0);
```

The **setup function**, besides defining the output, calls several functions:

```
void setup() {

  pinMode(pwmpin, OUTPUT);
      delay(10);
```

```
        Serial.begin(115200);
        Serial.println();
        Serial.print("Configuring access point...");

  startWiFi();
  startWebSocket();
  startServer();
}
```

The **loop function** only listens for web server events:

```
void loop() {
  webSocket.loop();
  server.handleClient();
}
```

We still have to define the functions called in the setup program.

The **startWiFi** function creates an access point:

```
void startWiFi(){

  WiFi.softAP(ssid, password);
  WiFi.softAPConfig(IPaddr, IPaddr, IPmask);
  IPAddress myIP = WiFi.softAPIP();

  Serial.print("Access Point ");
  Serial.println(ssid);
  Serial.print("AP IP address: ");
  Serial.println(myIP);
}
```

The Serial.print statements are useful for debugging purposes.

The **startWebSocket** starts a websocket and defines a function **webSocketEvent** that is called when a websocket event is happening. (This will be defined later).

```
void startWebSocket(){
  webSocket.begin();
  webSocket.onEvent(webSocketEvent);
  Serial.println("WebSocket server started.");
}
```

The **startServer function** starts a HTTP server and defines the functions that are called when different HTTP requests are done. Here we need only “/”, but in a more complex program there might be others like a reaction to buttons etc, see http://staff.ltam.lu/feljc/electronics/arduino/WiFi_01.pdf

```
void startServer() {
  server.onNotFound(handleNotFound);
  server.on("/", handleRoot);

  server.begin();
  Serial.println("HTTP server started.");
}
```

The **HTTP event functions** define reaction to root and handle not found:

```
void handleRoot() {
  server.send(200, "text/html", html);
}

void handleNotFound(){ /
    server.send(404, "text/plain", "404: File Not Found");
}
```

The handleRoot function sends the **html page** text string defined as described above.

The **webSocketEvent function** handles websocket events.

The reaction is different according to the type of the event. CONNECTED and DISCONNECTED events give us a message, but the one we need is the TEXT event, as our message is just a string containing the value that is sent.

```
void webSocketEvent(uint8_t num, WStype_t type, uint8_t * payload, size_t lenght) {

  switch (type) {

    case WStype_DISCONNECTED:
        Serial.printf("[%u] Disconnected!\n", num);
        break;

    case WStype_CONNECTED: {
        IPAddress ip = webSocket.remoteIP(num);
        Serial.printf("[%u] Connected from %d.%d.%d.%d url: %s\n",
             num, ip[0], ip[1], ip[2], ip[3], payload);
        }
        break;

    case WStype_TEXT:
        int pwm = atol((const char *)payload);
        analogWrite(pwmpin, pwm);
        Serial.println(pwm);
        break;
        }
}
```

With this, we can control the PWM value over WiFi, and the reaction is really fast.

The sketch can be found here:

http://staff.ltam.lu/feljc/electronics/arduino/8266/WiFi_set_PWM_websocket_01.zip

With CSS the slider can be embellished.