Group 1: Use a webpage to make stuff happen in the real world

The challenge: Create an application (a Python web server program and any necessary HTML) which presents a web page to a user. Clicking the "start" link on the page should cause the ESP32 to do something (like turn on an LED, or display something on the OLED display). The link should lead to another page with a "stop" button that reverses the action and takes the user back to the first page.

Getting started:

1. Start by serving two pages which you can navigate back and forth between.
2. Make an action occur (such as turning an LED on or off) when a page is requested
3. See if you can control a smartplug with your web server, by sending an MQTT message with the topic `ESPURNA-BEEE8A/relay/0/set` with the payload 1 (on) or 0 (off).

Extra challenges:

- Talk to group 2 and see if you can make a web page where you type into a text box, and this text is displayed on an OLED screen

Group 2: Send data *from* a web page to the server

The challenge: Create an application (a Python web server program and any necessary HTML) which presents a web page containing a text box and a "submit" button to a user. When the user clicks "submit", the contents of the text box should be sent to the web server, and the server should present a page containing the submitted text (e.g., if the user is directed to type their name in the box, the next page could say, "hello, [NAME]").

Getting started:

1. Start by serving two pages which you can navigate back and forth between.
2. Insert an HTML form on the first page:

```
<form action="page_to_request_when_user_clicks_submit">
How much popcorn?
<input type="text" name="quantity" placeholder="Enter a number..."/>
<br/>
<input type="submit" value="Make popcorn!"/>
</form>
```

3. On the Python side, you can use the `request.args` variable to see what was passed in via the form. Use this to customize your response.

```
@app.route('/monitor')
def monitor(request):
    # We can use request.args to read the values sent from the HTML form, e.g.,
    n = request.args['quantity']
```

Extra challenges:

- Can you pass multiple pieces of information? What about a dropdown box and a text field or a number spinner? (See https://www.w3schools.com/html/html_form_elements.asp)
- Talk to group 1 and see if you can make a web page where you type into a text box, and this text is displayed on an OLED screen.

Group 3:

The challenge: Create an application which returns machine-readable data containing information about a sensor connected to the ESP32.

Take a look at http://worldtimeapi.org/api/timezone/America/New_York . Notice that you get a response, but it's not HTML. This is JSON (JavaScript Object Notation), which is a human-readable format for passing data around on the internet. We've talked before about how to *request* JSON with your ESP32; your task here is to figure out how to *produce* a JSON response (instead of HTML) using Microdot on the ESP32.

There is a very simple example here: https://microdot.readthedocs.io/en/latest/intro.html#json-responses

1. Set up the web server to return a JSON response for some URL (e.g., "data")
2. Instead of returning made-up data, figure out how to take a measurement with the ESP32 (e.g., from a thermocouple) and return that.

Extra challenges:

- Can you put multiple pieces of information into a single endpoint (e.g., return both the temperature and the humidity, or multiple sensors)?
- Can you set up multiple endpoints?
- Talk to group 4 and see if you can make a page which regularly updates information from a sensor connected to the ESP32

Group 4: Continuously updating a web page

(This one is best if you have some experience with JavaScript.)

The challenge: Make a web page which regularly updates (without refreshing the page) by making small requests to a separate endpoint and updating parts of the page. You can do this locally, without needing your ESP32.

1. Open up http://worldtimeapi.org/api/timezone/America/New_York in your web browser. Notice that you get a response, but it's not HTML. This is JSON (JavaScript Object Notation), which is a human-readable format for passing data around on the internet.
2. Create a web page (.html) which will display the time. Put the part that'll hold the time in a `<div>` or `<span>`, and give it an ID (e.g., `<div id=timebox>0:00</div>`).
3. You can use this Javascript code to send a request and process the response as JSON:

```
function doUpdate()
{
    // XMLHttpRequest will let us make a web request *inside of a web page*
    var xmlhttp = new XMLHttpRequest();

    // Set up some code to run when the request returns
    xmlhttp.onreadystatechange = function() {
        // If the request was successful,
        if (this.readyState == 4 && this.status == 200) {
            // Parse the raw text into a JSON object
            values = JSON.parse(this.responseText)

            // Look up the timebox, and set its content
            document.getElementById("timebox").textContent = "wooo, updates!";
        }
    };
    // Set up the request to the particular web page
    xmlhttp.open("GET", "http://worldtimeapi.org/api/timezone/America/New_York", true);

    // And now actually send the request
    xmlhttp.send();
}
```

4. You can schedule something to run after a specified interval using the `setTimeout()` function: `setTimeout(function_to_run, NUMBER_OF_MILLISECONDS)`

Extra challenges:

- Try pulling data from another API, such as the MBTA train arrival
- Talk to group 3 and see if you can make a page which regularly updates information from a sensor connected to the ESP32