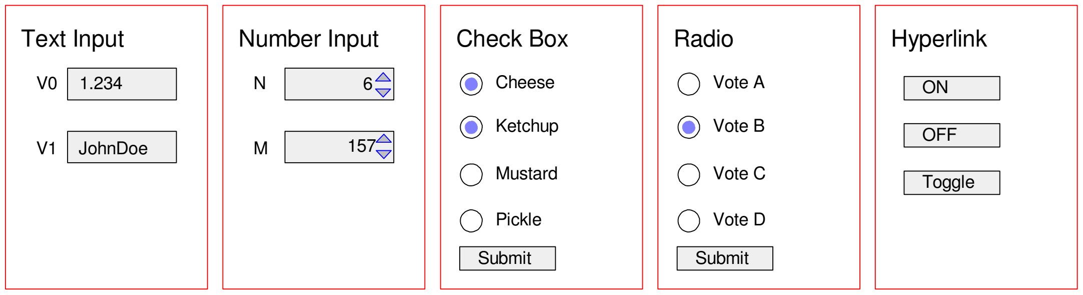
33. WiFi & AP Tags

Introduction:

In the last lecture, we looked at how to send data from a Pico to clients in AP mode. This allows clients the ability to see what's happening at the Pico's end. In this lecture, we'll look at having clients send data back to the Pico to control its actions.

The techniques covered in this lecture include

- Text Inputs: Send a text string back to the Pico
- Number Inputs: Send an integer back to the Pico
- Check Boxes: Give the client the option of checking one or more boxes.
- Radio Input: Give the client several options and allow them to check only one of these.
- Hyperlink: Send a different messages back to the Pico based upon which item is clicked on



AP tags allow for different ways to get data back to the host

The technique used here is to write a short html program which send back data from the client. The core of the code presented here is based upon code written at

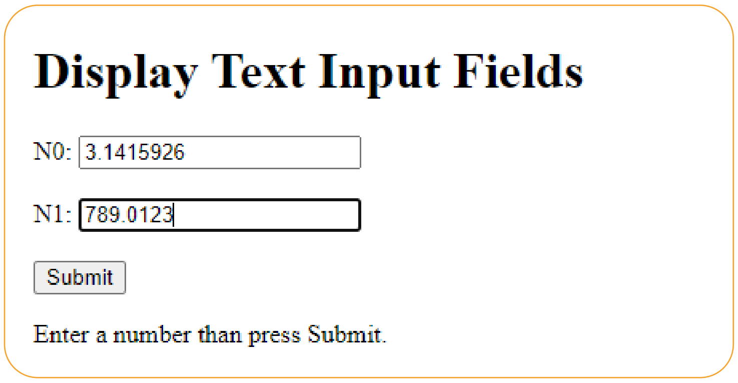
https://www.w3schools/tags/

which is an excellent site for learning about tags. In addition, you can write and test out your html code in interactive windows at

https://www.w3schools/tags/att_input_type

Text Inputs

With a text field, you can type in anything you like in a box (text, numbers, etc). This allows you to input commands, floating point numbers, etc. For example, generate the following display which outputs two numbers (actually, two text strings) when you press *Submit*



Text Inputs. Numbers, text, etc. can be input

html code: First, start with the html code to generate this display

```
<!DOCTYPE html>
<html>
<body>

<h1>Display Text Input Fields</h1>

<form action="/action_page.php">
  <label for="fname">N0: </label>
  <input type="text" id="N0" name="N0"><br><br>
  <label for="lname">N1: </label>
  <input type="text" id="N1=" name="N1"><br><br>
  <input type="submit" value="Submit">
</form>

<p>Enter a number than press Submit.</p>

</body>
</html>
```

33_text.html Code for generating two text input boxes

The main routine which uses this file is as follows:

First, a subroutine *web_page()* reads this file and converts it to a string. The main routine will then send this string over the WiFi network to any clients connected.

```
import network
import time
import socket

def web_page():
    f = open("33_text.html")
    x = f.read()
    x = x.replace('\r\n',' ')
    return(x)
```

Next, the AP network is created just like we did in the previous lecture.

```
ssid = 'Pico-Network'
password = 'PASSWORD'

ap = network.WLAN(network.AP_IF)
ap.config(ssid=ssid, password=password)
ap.active(True)

while ap.active() == False:
    pass
print('AP Mode Is Active, You can Now Connect')
print('IP Address To Connect to:: ' + ap.ifconfig()[0])

print('Channel',  ap.config('channel'))

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(('', 80))
s.listen(5)
```

Finally comes the main loop. The bare minimum for this main loop

- Waits for a response from a client (*s.accept()*)
- Once received, the response is printed (saved in variable *request*),
- The web page is refreshed (*conn.send()*), and
- The connection is closed

```
while(1):
    conn, addr = s.accept()
    print('Got a connection from %s' % str(addr))
    request = conn.recv(1024)
    request = request.decode('utf-8')
    print(request)
    response = web_page()
    conn.send(response)
    conn.close()
```

With this minimal setup for the main loop, the response in the shell window looks like the following:

```
Got connection from ('192.168.4.17', 59475)

GET /favicon.ico HTTP/1.1
Host: 192.168.4.1
Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/127.0.0.0
Safari/537.36
Accept:
image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8
Referer: http://192.168.4.1/action_page.php?N0=123.456&N1=789.012
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
```

This contains a lot of information I really don't care about. To pull out the responses, the following steps are taken:

- First, look for the string *php?*. This marks the start of the response.
- Look for a carriage return and line feed (*/r/n*). This marks the end of the response.
- Next, look for a question mark. This indicates the end of a text box and the start of the next
- Keep looking for question marks until there are none left.

This is optional, but it separates the responses so you can do something with them. The resulting code looks like this:

```
while(1):
    conn, addr = s.accept()
    print('Got a connection from %s' % str(addr))
    request = conn.recv(1024)
    request = request.decode('utf-8')
    n = request.find('php?')+4
    request = request[n:]
    n = request.find('\r\n')
    request = request[0:n]
    msg = []
    while(request.find('&')>0):
        n = request.find('&')
        msg.append(request[0:n])
        request = request[n+1:]
    msg.append(request)
    for i in range(0,len(msg)):
       print('msg[',i,'] = ', msg[i])
    response = web_page()
    conn.send(response)
    conn.close()
```

Main loop with each field pulled out and stored in *msg[]*

The shell window then shows the data each time you press *submit*

```
Got connection from ('192.168.4.17', 59482)
msg[ 0 ] =  N0=123.456
msg[ 1 ] =  N1=789.012
```

Shell window shows the data passed from the client

If you want, you can pull out the numbers and convert the text string to a floating point number using the *float()* command.

Note that spaces are interpreted as + symbols. If you type in

- N0: *The quick red fox jumped*
- N1: *over the lazy dog's head*

you get

```
Got connection from ('192.168.4.17', 55722)
msg[ 0 ] =  N0=The+quick+red+fox+jumped
msg[ 1 ] =  N1=over+the+lazy+dof%27s+head
```

Note that

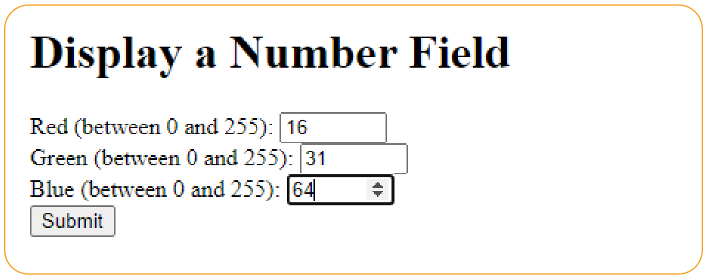
- spaces are converted to + signs
- The quote is replaces with its ascii value (%27)

Just so you know...

Note: This basic structure is used for all of the following tags. The only thing that changes is which html file is read in within the *web_page()* subroutine.

Number Fields

Number fields allow you to input an integer over a certain range. The number can be either typed into the box or the up-down arrows can be used to select a given number. Once happy with the values, the numbers can be sent to the Pico using the *Submit* button.



Display a Number Field

Red (between 0 and 255): 16
Green (between 0 and 255): 31
Blue (between 0 and 255): 64

Display for inputting data using a number field.

The html code for this is as follows:

```
<!DOCTYPE html>
<html>
<body>

<h1>Display a Number Field</h1>

<form action="/action_page.php">
  <label for="red">Red (between 0 and 255):</label>
  <input type="number" id="red" name="r" min="0" max="255">
  <br>

  <label for="green">Green (between 0 and 255):</label>
  <input type="number" id="green" name="g" min="0" max="255">
  <br>

  <label for="blue">Blue (between 0 and 255):</label>
  <input type="number" id="blue" name="b" min="0" max="255">
  <br>
  <input type="submit">

</form>

</body>
</html>
```

33_Number.html Code for three number fields

'The main routine is the same as before, only with the *web_page()* routine pulling in a different file from your Pi-Pico:

```
def web_page():
    f = open("33_Number.html")
    x = f.read()
    x = x.replace('\r\n',' ')
    return(x)
```

The main routine remains unchanged except for the file you read in

The result in the shell window shows the data returned from the client each time you press *Submit*

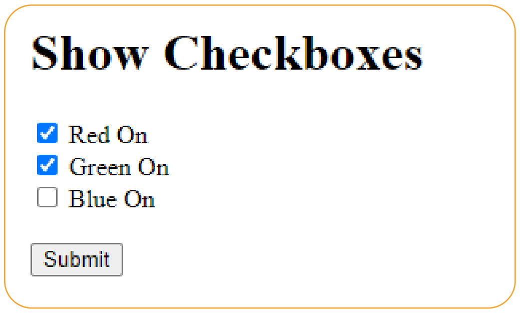
```
--------------------
Got a connection from ('192.168.4.17', 57050)
msg[ 0 ] =  r=16
msg[ 1 ] =  g=31
msg[ 2 ] =  b=64
--------------------
Got a connection from ('192.168.4.17', 57051)
msg[ 0 ] =  r=35
msg[ 1 ] =  g=255
msg[ 2 ] =  b=88
--------------------
Got a connection from ('192.168.4.17', 64852)
msg[ 0 ] =  r=35
msg[ 1 ] =  g=255
msg[ 2 ] =  b=88
--------------------
Got a connection from ('192.168.4.17', 57053)
msg[ 0 ] =  r=0
msg[ 1 ] =  g=0
msg[ 2 ] =  b=0
```

Each time you press *Submit*, you get a new message from the client

What you do with this data is up to you.

Check Box

A Check Box lets you select any number of items from a list and send those to the Pico.



Example of a check-box. Press Submit for which colors to turn on

The html code for a checkbox is:

```
<!DOCTYPE html>
<html>
<body>

<h1>Show Checkboxes</h1>

<form action="/action_page.php">
  <input type="checkbox" id="r" name="color1" value="Red">
  <label for="color1"> Red On</label><br>
  <input type="checkbox" id="g" name="color2" value="Green">
  <label for="color2"> Green On</label><br>
  <input type="checkbox" id="b" name="color3" value="Blue">
  <label for="color3"> Blue On</label><br><br>
  <input type="submit" value="Submit">
</form>

</body>
</html>
```

33_CheckBox.html Code for three checkboxes

The main routine remains unchanged (save for the name of the html file loaded in *web_page()*). The data passed shows up in the shell window. Note

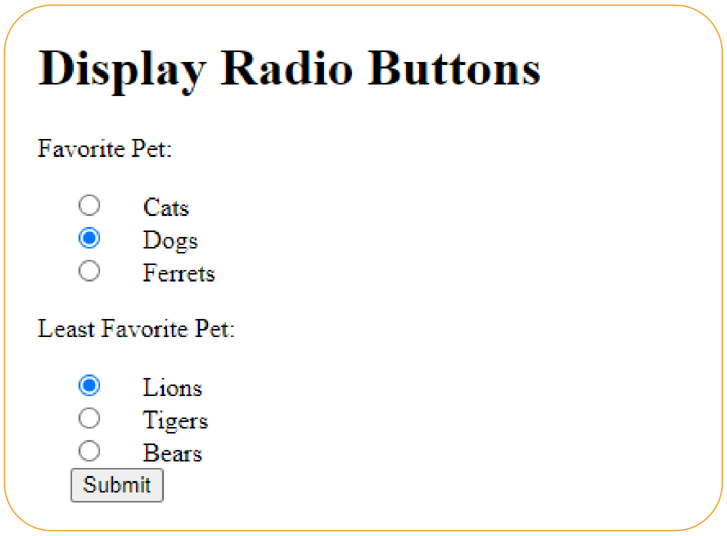
- If no buttons are checked, you get a null response (first entry)
- If multiple buttons are checked, you get multiple responses (last entry)

```
Got a connection from ('192.168.4.17', 59069)
msg[ 0 ] = 
-----
Got a connection from ('192.168.4.17', 59071)
msg[ 0 ] =  color1=Red
-----
Got a connection from ('192.168.4.17', 59073)
msg[ 0 ] =  color2=Green
-----
Got a connection from ('192.168.4.17', 57509)
msg[ 0 ] =  color1=Red
msg[ 1 ] =  color2=Green
msg[ 2 ] =  color3=Blue
```

Responses from a CheckBox tag

Radio Buttons

Radio Buttons let you select only one option. If you click on another item, it deselects the previously selected item.



Display Radio Buttons

Favorite Pet:

- ○ Cats
- ● Dogs
- ○ Ferrets

Least Favorite Pet:

- ● Lions
- ○ Tigers
- ○ Bears

Radio Buttons: Only one can be selected from the list

The html code for a radio button is

```
<!DOCTYPE html>
<html>
<body>

<h1>Display Radio Buttons</h1>

<form action="/action_page.php">
  <p>Favorite Pet:</p>
  <input type="radio" id="cats" name="like" value="Cats">
  <label for="cats">Cats</label>
  <br>
  <input type="radio" id="dogs" name="like" value="Dogs">
  <label for="dogs">Dogs</label>
  <br>
  <input type="radio" id="ferret" name="like" value="Ferrets">
  <label for="ferret">Ferrets</label>
  <br>
  <p>Least Favorite Pet:</p>
  <input type="radio" id="lion" name="dislike" value="Lions">
  <label for="lion">Lions</label>
  <br>
  <input type="radio" id="tiger" name="dislike" value="Tigers">
  <label for="tiger">Tigers</label>
  <br>
  <input type="radio" id="bear" name="dislike" value="Bears">
  <label for="bear">Bears</label>
  <br>
  <input type="submit" value="Submit">
</form>

</body>
</html>
```

33_Radio.html Radio button code

The selected items are returned each time you press the Submit button in the shell window

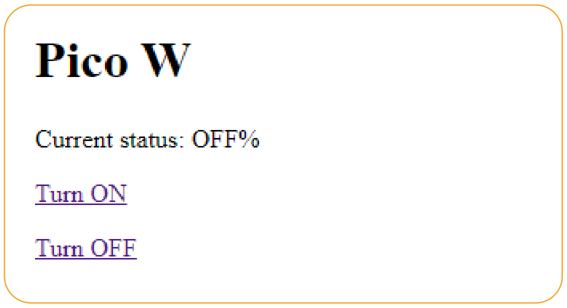
```
>> %Run -c $EDITOR_CONTENT

MPY: soft reboot
AP Mode Is Active, You can Now Connect
IP Address To Connect to:: 192.168.4.1
Channel 3
-----
Got a /favicon response from ('192.168.4.17', 56946)
msg[ 0 ] =  like=Cats
msg[ 1 ] =  dislike=Lions
-----
Got a /favicon response from ('192.168.4.17', 56951)
msg[ 0 ] =  like=Cats
msg[ 1 ] =  dislike=Tigers
-----
Got a /favicon response from ('192.168.4.17', 56954)
msg[ 0 ] =  like=Ferrets
msg[ 1 ] =  dislike=Lions
-----
Got a /favicon response from ('192.168.4.17', 56957)
msg[ 0 ] =  like=Dogs
msg[ 1 ] =  dislike=Lions
-----
Got a /favicon response from ('192.168.4.17', 56959)
msg[ 0 ] =  like=Ferrets
msg[ 1 ] =  dislike=Lions
```

Note: You can get multiple reads each time you press *Submit.* If used as a voting machine, you might need to add some code to detect these duplicate readings.

Hyperlink

A fourth way to have a client talk to the Pico is through hyperlinks. Hyperlinks normally take you to another web page, but they can also be used to send data back to the host. This requires slightly different coding for the web server routine as well as the main routine.

As an example, use hyperlinks to turn on and off the LED attached to GP16:



Pico W

Current status: OFF%

Turn ON

Turn OFF

33_Hyperlink.html. The client can turn on an off an led by clicking on the hyperlinks

The html code for this is as follows:

```
<!DOCTYPE html>
<html>

<head> <title>Pico W</title> </head>
<body> <h1>Pico W</h1>
<p>Current status: bbbbb /p>
<p><a href="http://aaaaa/light_on">Turn ON</a></p>
<p><a href="http://aaaaa/light_off">Turn OFF</a></p>
</body>

</html>
```

33_Hyperlink.html. Code for setting up two hyperlinks that send data back to the host

This uses some tricks of the previous lecture:

- aaaaa is the IP-address of the host
- bbbbb is the status of the LED (ON or OFF)

The *web_page()* routine in Python reads this file and replaces *aaaaa* and *bbbbb* with parameters that are passed to it.

```
def web_page(ip_address, OnOff):
    f = open("33 Hyperlink.html")
    x = f.read()
    x = x.replace('\r\n',' ')
    x = x.replace('aaaaa',ip_address)
    x = x.replace('bbbbb', OnOff)
    return(x)
```

Subroutine which reads *33_Hyperlink.html* and replaces aaaaa with the IP address and bbbbb with the LED's status

Down in the main loop of the main routine, more stuff goes on...

First, instead of responding to every message received from a client, only *favicon* messages are responded to. All other messages are ignored. (note: might be a good idea to do this with previous programs too)

```
while(1):
    flag = 0
    while(flag == 0):
        conn, addr = s.accept()
        request = conn.recv(1024)
        request = request.decode('utf-8')
        if(request.find('favicon') > 0):
            flag = 1
        else:
            response = web_page(IP_Address, OnOff[LED.value()] )
            conn.send(response)
            conn.close()
```

Main loop: Keep looking for messages until you see one with *favicon* in its content

Once you find this message, locate the string which starts with *Referer:*

```
Referer:  //https:/192.168.4.1/light_on
```

Keep the message past this point and the end of the line (carriage return, line feed):

```
    n = request.find('Referer:')+9
    request = request[n:]
    n = request.find('\r\n')
    request = request[0:n]
```

At this point, the message will look something like

```
//https:/192.168.4.1/light_on
```

Start stripping out everything to the left of the backslashes until you run out of back slashes. To avoid infinite loops, only repeat ten times (a little inefficient but works).

```
    for i in range(0,10):
        n = request.find('/')+1
        if(n>0):
            request = request[n:]
    print(request)
```

At this point, you have the message from the hyper link: either *light_on* or *light_off*. Based upon what you receive, you can turn on and off the LED:

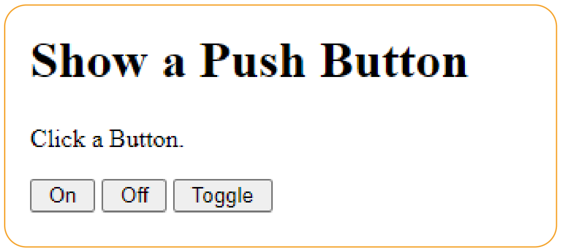
```
    if(request == 'light_on'):
        LED.value(1)
    if(request == 'light_off'):
        LED.value(0)
    response = web_page(IP_Address, LED.value() )
    conn.send(response)
    conn.close()
```

The net results is

- Every time you click on Turn_ON, the LED turns on
- Every time you click on Turn_OFF, the LED turns off

Hyperlink with Buttons

Finally, using hyperlinks, clicking on an item such as a button can return data to the host. If using buttons, you get the same results as the previous section but with better looks:



Using hyperlinks, data is sent back when you press a button (file 33_button.html)

The html code for this uses the button tags along with a hyperlink. The routine *web_page()* will again replace *aaaaa* with the ip-address of the host.

```
<!DOCTYPE html>
<html>
<body>

<h1>Show a Push Button</h1>

<p>Click a Button.</p>
<form>
<p><a href="http://aaaaa/light_on"><input type="button" value=" On
"></a>
<a href="http://aaaaa/light_off"><input type="button" value=" Off
"></a>
<a href="http://aaaaa/light_toggle"><input type="button" value="
Toggle "></a>
</p>
</form>

</body>
</html>
```

html code for the push button example

When you click on a button, the message *light_on, light_off,* or *light_toggle* is sent back to the host.

```
>>> %Run -c $EDITOR_CONTENT

MPY: soft reboot
AP Mode Is Active, You can Now Connect
IP Address To Connect to:: 192.168.4.1
Channel 3

light_on
light_off
light_toggle
light_on
light_off
light_toggle
```

Clicking on a button sends a single command back to the main routine

Summary:

In this lecture, techniques for having a client in AP mode send data back to the host were presented. This allows you to

- Connect to your Pico even if there is no internet present
- Input binary numbers, turning an LED on or off from your cell phone or browser,
- Input floating point numbers, allowing you to vary the brightness of an LED, speed of a motor, etc, and
- Select from several options using various tags.

Many more tags exist and are presented in w3schools. The ones presented here are the ones I was able to get to work with a Pi-Pico. With some effort, you can probably get the other ones to work as well.

References:

- https://www.w3schools/tags/att_input_type