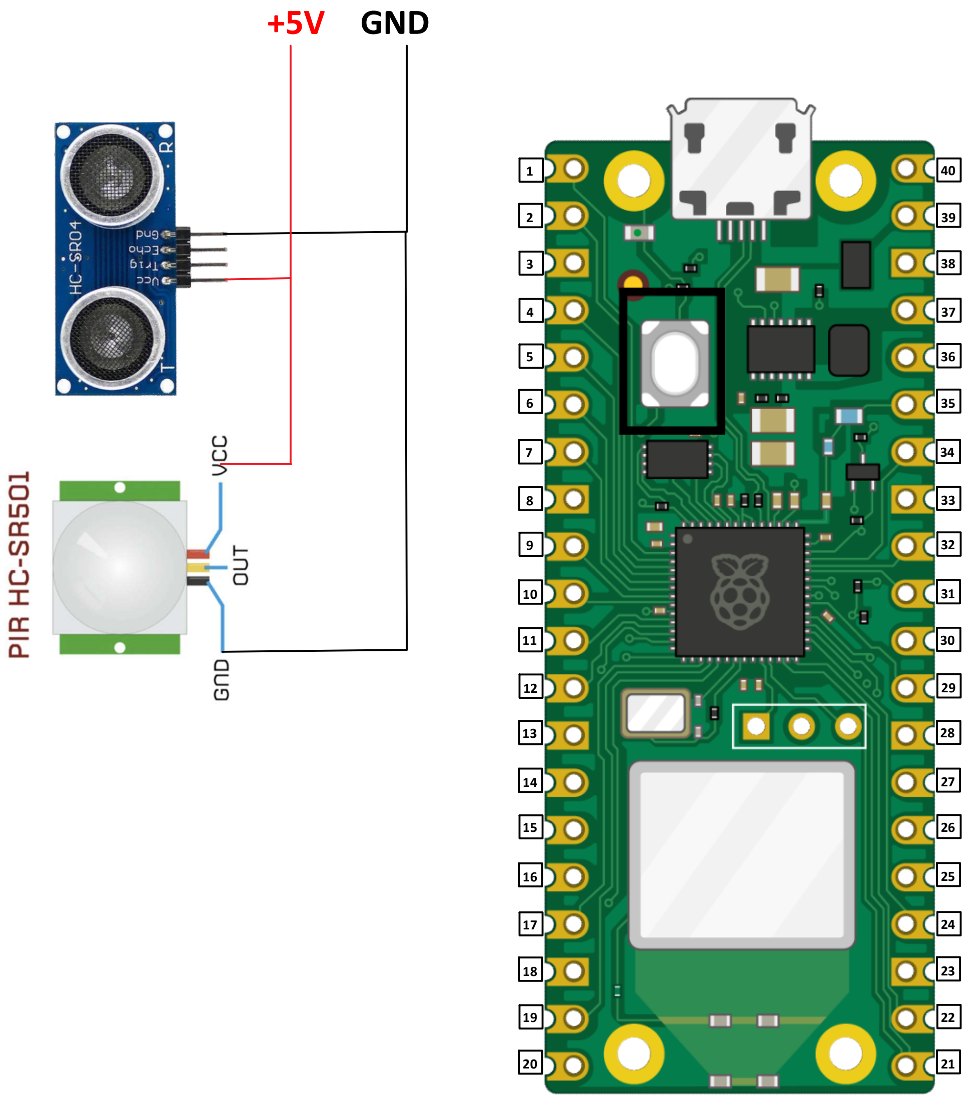
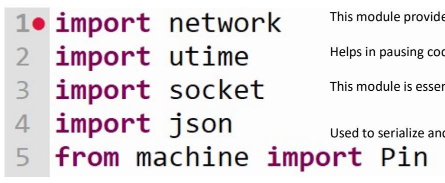


Code Explanation

This section imports necessary modules



```
import network
import utime
import socket
import json
from machine import Pin
```

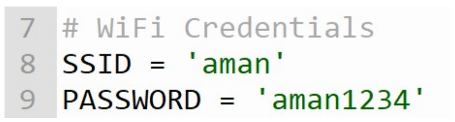
This module provides functionalities to connect to Wi-Fi networks

Helps in pausing code execution.

This module is essential for creating and interacting with network sockets, effectively enabling the Raspberry Pi Pico to run a lightweight web server.

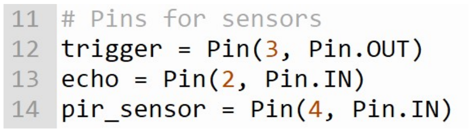
Used to serialize and deserialize JSON data, especially useful when sending sensor data to the client.

These allow interaction with the Raspberry Pi Pico's hardware pins, which is essential for sensor interfacing.



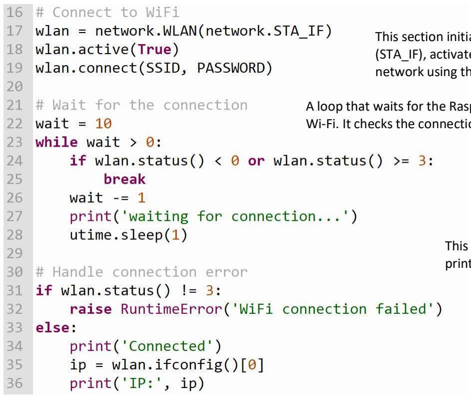
```
# WiFi Credentials
SSID = 'aman'
PASSWORD = 'aman1234'
```

These are your WiFi credentials. The script will attempt to connect to the network with this SSID using the provided password.



```python
# Pins for sensors
trigger = Pin(3, Pin.OUT)
echo = Pin(2, Pin.IN)
pir_sensor = Pin(4, Pin.IN)
```

In this section all the pins of sensors are defined

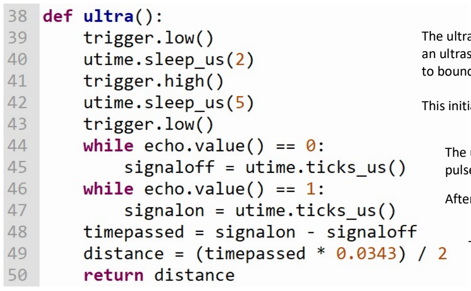


```python
# Connect to WiFi
wlan = network.WLAN(network.STA_IF)
wlan.active(True)
wlan.connect(SSID, PASSWORD)

# Wait for the connection
wait = 10
while wait > 0:
    if wlan.status() < 0 or wlan.status() >= 3:
        break
    wait -= 1
    print('waiting for connection...')
    utime.sleep(1)

# Handle connection error
if wlan.status() != 3:
    raise RuntimeError('WiFi connection failed')
else:
    print('Connected')
    ip = wlan.ifconfig()[0]
    print('IP:', ip)
```

This section initializes the Wi-Fi interface in station mode (STA_IF), activates it, and then attempts to connect to the network using the credentials provided earlier.

A loop that waits for the Raspberry Pi Pico to successfully connect to the Wi-Fi. It checks the connection status and waits for up to 10 seconds.

This block checks if the device is connected successfully and prints its IP address. If not connected, it raises a runtime error.



```python
def ultra():
    trigger.low()
    utime.sleep_us(2)
    trigger.high()
    utime.sleep_us(5)
    trigger.low()
    while echo.value() == 0:
        signaloff = utime.ticks_us()
    while echo.value() == 1:
        signalon = utime.ticks_us()
    timepassed = signalon - signaloff
    distance = (timepassed * 0.0343) / 2
    return distance
```

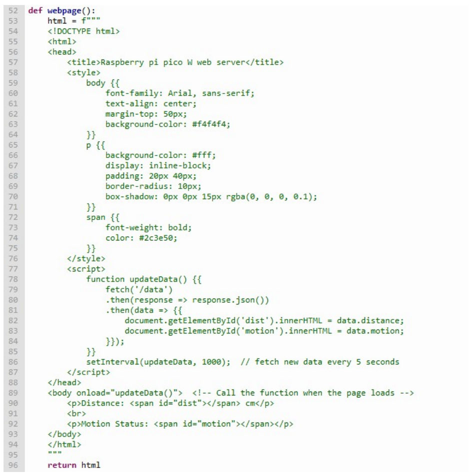
The ultra function calculates the distance to an object using an ultrasonic sensor. It measures the time it takes for a pulse to bounce back and converts this time to a distance.

This initial segment is responsible for generating a short ultrasonic pulse.

The ultrasonic sensor works by sending out a sound pulse and then listening for the echo or return pulse.

After detecting the start of the echo, this loop waits for the echo to end

This line computes the distance to the object that reflected the ultrasonic pulse:
- timepassed * 0.0343 calculates the total distance traveled by the ultrasonic pulse. Sound travels at approximately 343 meters per second (or 0.0343 cm per microsecond).
- Since the pulse travels to the object and then back to the sensor, we divide by 2 to get only the distance to the object.

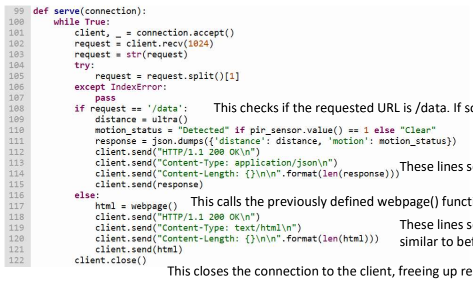


```python
def webpage():
    html = f"""
    <!DOCTYPE html>
    <html>
    <head>
        <title>Raspberry pi pico W web server</title>
        <style>
            body {{
                font-family: Arial, sans-serif;
                text-align: center;
                margin-top: 50px;
                background-color: #f4f4f4;
            }}
            p {{
                background-color: #fff;
                display: inline-block;
                padding: 20px 40px;
                border-radius: 10px;
                box-shadow: 0px 0px 15px rgba(0, 0, 0, 0.1);
            }}
            span {{
                font-weight: bold;
                color: #2c3e50;
            }}
        </style>
        <script>
            function updateData() {{
                fetch('/data')
                .then(response => response.json())
                .then(data => {{
                    document.getElementById('dist').innerHTML = data.distance;
                    document.getElementById('motion').innerHTML = data.motion;
                }});
            }}
            setInterval(updateData, 1000);  // fetch new data every 5 seconds
        </script>
    </head>
    <body onload="updateData()">  <!-- Call the function when the page loads -->
        <p>Distance: <span id="dist"></span> cm</p>
        <br>
        <p>Motion Status: <span id="motion"></span></p>
    </body>
    </html>
    """
    return html
```

The webpage function creates an HTML page to display sensor data. The HTML includes embedded JavaScript to fetch updated data from the server every second.

The <body> tag contains the visible parts of the HTML document. The onload attribute ensures the updateData function is called as soon as the webpage is loaded.

The content in the body displays the distance (from the ultrasonic sensor) and the motion status (from the PIR sensor). The actual values are filled in dynamically using the JavaScript function mentioned earlier.



```python
def serve(connection):
    while True:
        client, _ = connection.accept()
        request = client.recv(1024)
        request = str(request)
        try:
            request = request.split()[1]
        except IndexError:
            pass
        if request == '/data':
            distance = ultra()
            motion_status = "Detected" if pir_sensor.value() == 1 else "Clear"
            response = json.dumps({'distance': distance, 'motion': motion_status})
            client.send("HTTP/1.1 200 OK\n")
            client.send("Content-Type: application/json\n")
            client.send("Content-Length: {}\n\n".format(len(response)))
            client.send(response)
        else:
            html = webpage()
            client.send("HTTP/1.1 200 OK\n")
            client.send("Content-Type: text/html\n")
            client.send("Content-Length: {}\n\n".format(len(html)))
            client.send(html)
        client.close()
```

The serve function contains the core logic of the web server. It waits for client requests, processes the requests, and sends appropriate responses.

This checks if the requested URL is /data. If so, it will respond with the sensor data

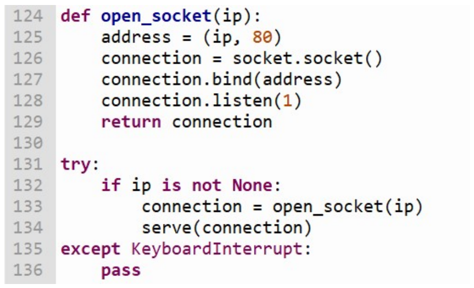
This line creates a JSON string containing the distance and motion status values

These lines send an HTTP response back to the client.

This calls the previously defined webpage() function to get the HTML content for the main page.

These lines send an HTTP response back to the client, similar to before, but this time sending HTML content.

This closes the connection to the client, freeing up resources and allowing the server to handle other incoming connections.



```python
def open_socket(ip):
    address = (ip, 80)
    connection = socket.socket()
    connection.bind(address)
    connection.listen(1)
    return connection

try:
    if ip is not None:
        connection = open_socket(ip)
        serve(connection)
except KeyboardInterrupt:
    pass
```

This function sets up a socket on port 80 (standard HTTP port) and binds it to the IP address of the Raspberry Pi Pico.

This block ties everything together:

- If there's a valid IP (meaning we're connected to Wi-Fi), it opens a socket.
- The serve function then takes over to handle web requests.
- The script runs indefinitely unless interrupted by a keyboard action (e.g., Ctrl+C)