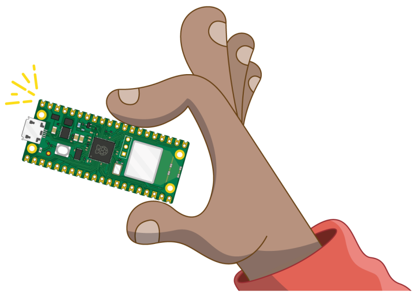
Projects

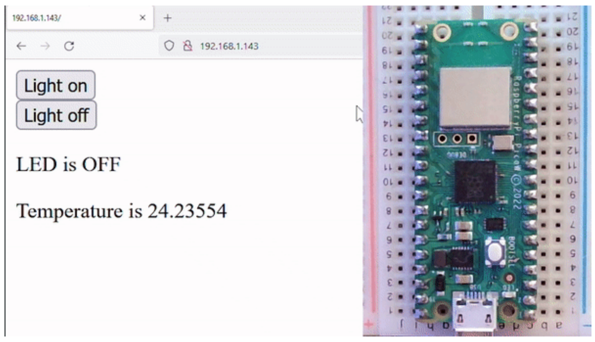
Getting started with your Raspberry Pi Pico W

Get started with your Raspberry Pi Pico W



Step 1 Introduction

Raspberry Pi Pico W is a new Raspberry Pi product that adds WiFi capability to the Raspberry Pi Pico, allowing you to connect the device to a WiFi network. In this guide, you will learn how to use a Raspberry Pi Pico W, how to connect it to a WiFi network, and then how to turn it into a web server to control digital outputs from a browser, and to receive sensor data.



LED is OFF

Temperature is 24.23554

WiFi enabled devices allow for easy communication between computers and the internet. The technology helped kick-start the Internet Of Things (IoT) revolution.

You will:

- Connect your Raspberry Pi Pico W to a WiFi hub
- Create a web server on your Raspberry Pi Pico W, to display a webpage
- Use your webpage to control the Raspberry Pi Pico W onboard LED and to receive temperature data from it

You will need:

- A Raspberry Pi Pico W and micro USB data cable
- A computer connected to your network
- The Thonny Python IDE

Install Thonny

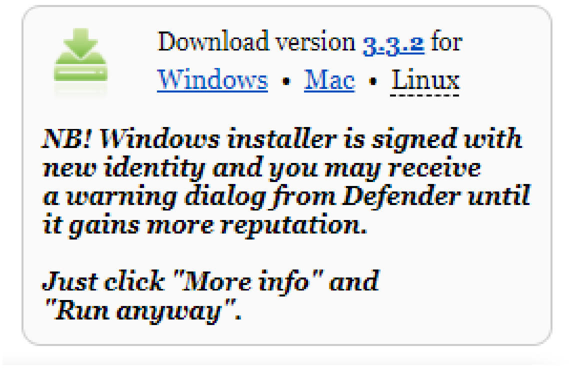
Install Thonny on a Raspberry Pi

- Thonny is already installed on Raspberry Pi OS, but may need to be updated to the latest version
- Open a terminal window, either by clicking the icon in the top left-hand corner of the screen or by pressing the Ctrl+Alt+T keys at the same time
- In the window, type the following to update your OS and Thonny

```
sudo apt update && sudo apt upgrade -y
```

Install Thonny on other operating systems

- On Windows, macOS, and Linux, you can install the latest Thonny IDE or update an existing version
- In a web browser, navigate to **thonny.org** (https://thonny.org/)
- In the top right-hand corner of the browser window, you will see download links for Windows and macOS, and instructions for Linux
- Download the relevant files and run them to install Thonny



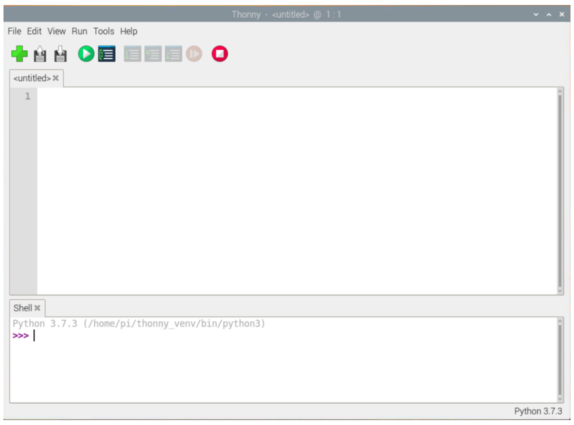
Download version 3.3.2 for
Windows • Mac • Linux

NB! Windows installer is signed with new identity and you may receive a warning dialog from Defender until it gains more reputation.

Just click "More info" and "Run anyway".

Opening Thonny

Open Thonny from your application launcher. It should look something like this:



```
Python 3.7.3 (/home/pi/thonny_venv/bin/python3)
>>> |
```

You can use Thonny to write standard Python code. Type the following in the main window, and then click the **Run** button (you will be asked to save the file).

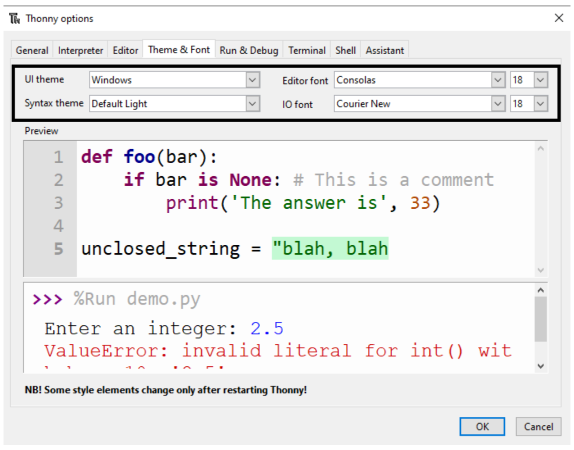
```
print('Hello World!')
```

Change the theme and font in Thonny

Thonny allows you to change the theme and font of the software. This feature means that you can increase the font size and change the background and text colours to suit your needs.

To change the theme and font:

- Click on Tools -> Options.
- Click on the 'Theme & Font' tab.
- Click on the drop down boxes for each option until you find the settings that best suit your needs.



UI theme: Windows
Syntax theme: Default Light
Editor font: Consolas 18
IO font: Courier New 18

```
def foo(bar):
    if bar is None: # This is a comment
        print('The answer is', 33)

unclosed_string = "blah, blah
```

```
>>> %Run demo.py
 Enter an integer: 2.5
 ValueError: invalid literal for int() wit
```

NB! Some style elements change only after restarting Thonny!

- Press OK when you are finished.

Warning: Stick to simple, clean fonts. If you use a handwriting style font then it can make it difficult to read and debug.

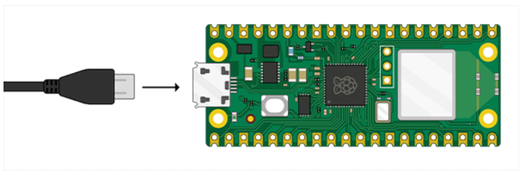
Step 2 Set up your Raspberry Pi Pico W

Connect your Raspberry Pi Pico W and set up MicroPython.

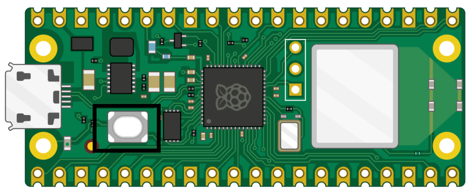
MicroPython is a version of the Python programming language for microcontrollers, such as your Raspberry Pi Pico W. MicroPython lets you use your Python knowledge to write code to interact with electronics components.

Download the latest version of Raspberry Pi Pico W firmware at **https://rpf.io/pico-w-firmware** (https://rpf.io/pico-w-firmware)

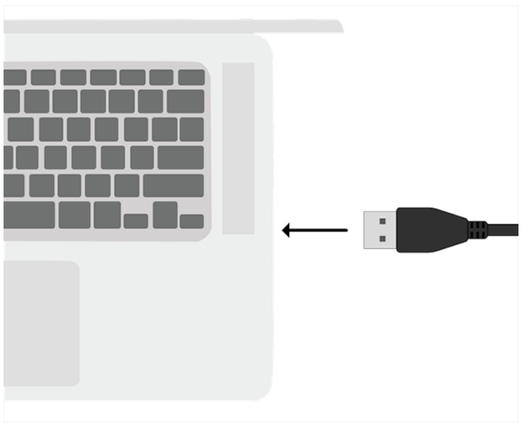
Connect the small end of your micro USB cable to the Raspberry Pi Pico W.



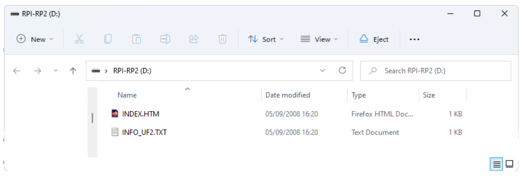
Hold down the **BOOTSEL** button on your Raspberry Pi Pico W.



Connect the other end to your desktop computer, laptop, or Raspberry Pi.



Your file manager should open up, with Raspberry Pi Pico being show as an externally connected drive. Drag and drop the firmware file you downloaded into the file manager. Your Raspberry Pi Pico should disconnect and the file manager will close.

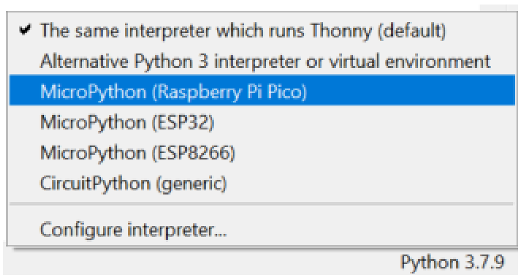


Name	Date modified	Type	Size
INDEX.HTM	05/09/2008 16:20	Firefox HTML Doc...	1 KB
INFO_UF2.TXT	05/09/2008 16:20	Text Document	1 KB

Open the Thonny editor.

Look at the text in the bottom right-hand corner of the Thonny editor. It will show you the version of Python that is being used.

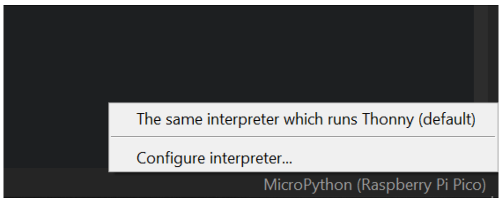
If it does **not** say 'MicroPython (Raspberry Pi Pico)' there, then click on the text and select 'MicroPython (Raspberry Pi Pico)' from the options.


✓ The same interpreter which runs Thonny (default)
Alternative Python 3 interpreter or virtual environment
MicroPython (Raspberry Pi Pico)
MicroPython (ESP32)
MicroPython (ESP8266)
CircuitPython (generic)
Configure interpreter...
Python 3.7.9

Debug:

I don't know if the firmware is installed and cannot connect to my Pico

Make sure your Raspberry Pi Pico W is connected to your computer with a micro USB cable. Click on the list in the bottom right-hand corner of your Thonny window. A pop-up menu will appear, which lists the available interpreters.


The same interpreter which runs Thonny (default)
Configure interpreter...
MicroPython (Raspberry Pi Pico)

If you cannot see Pico in the list (as shown in the picture), you need to reconnect your Raspberry Pi Pico W while holding the BOOTSEL button to mount it as a storage volume, and then reinstall the firmware by following the instructions in the section above.

Firmware is installed but I still cannot connect to my Pico

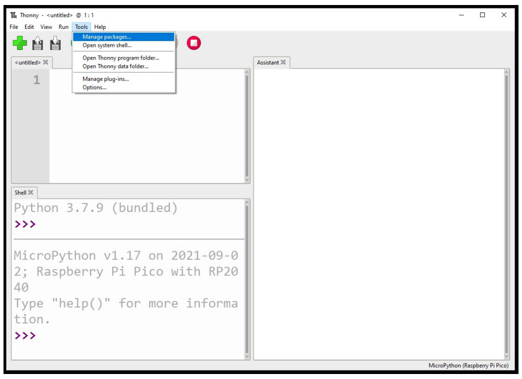
You may be using the wrong kind of micro USB cable. Your current micro USB cable may be damaged, or designed to only carry power to devices and not transfer data. Try swapping your cable for another if nothing else has worked.

If your Pico still won’t connect after trying all these things, it may **itself** be damaged and unable to connect.

For newcomers to Raspberry Pi Pico, `picozero` is a MicroPython library that’s beginner-friendly.

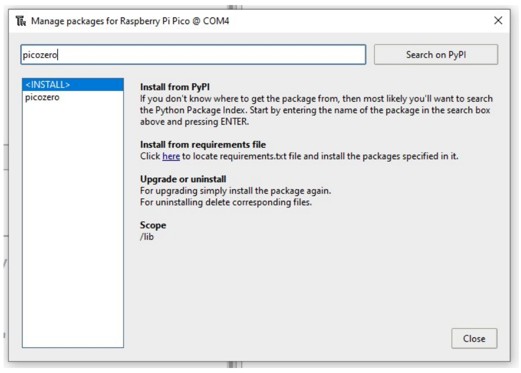
To complete the projects in this path, you need to install the `picozero` library as a Thonny package.

In Thonny, choose **Tools** > **Manage packages**.



```
Python 3.7.9 (bundled)
>>>

MicroPython v1.17 on 2021-09-02; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>>
```

In the pop-up ‘Manage packages for Raspberry Pi Pico’ window, type `picozero` and click **Search on PyPi**.



Install from PyPI
If you don't know where to get the package from, then most likely you'll want to search the Python Package Index. Start by entering the name of the package in the search box above and pressing ENTER.

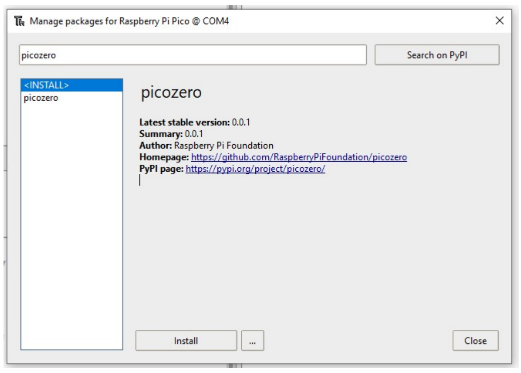
Install from requirements file
Click here to locate requirements.txt file and install the packages specified in it.

Upgrade or uninstall
For upgrading simply install the package again.
For uninstalling delete corresponding files.

Scope
/lib

Click on **picozero** in the search results.

Click on **Install**.



picozero

Latest stable version: 0.0.1
Summary: 0.0.1
Author: Raspberry Pi Foundation
Homepage: https://github.com/RaspberryPiFoundation/picozero
PyPI page: https://pypi.org/project/picozero/

When installation has completed, close the package window, then quit and reopen Thonny.

If you have difficulties installing the `picozero` library in Thonny, you can download the library file and save it to your Raspberry Pi Pico W.

Installing picozero offline

Installing picozero offline

If you do not have access to the internet on the computer you are connecting to you Raspberry Pi Pico, or you do not have permissions to install packages with Thonny, you can still use the picozero library.

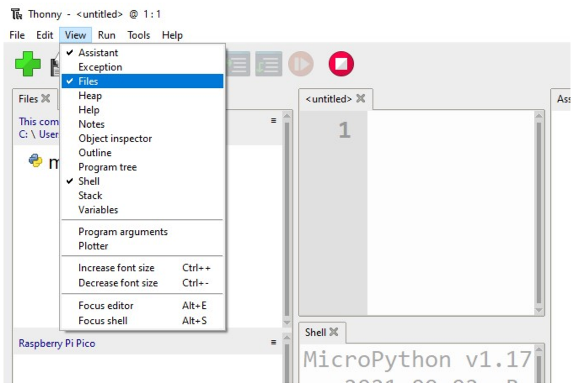
You can use another internet connected computer to download the file you need and then store the file on a USB flash memory stick.

1. Go to the `picozero.py` file in the **picozero GitHub repository** (https://raw.githubusercontent.com/RaspberryPiFoundation/picozero/master/picozero/picozero.py?token=GHSAT0AAAAAABRLTKWZCT53CGKBFHMJGE54YSC762A) using a web browser.
2. Right click on the picozero page, and choose to **Save page as**.

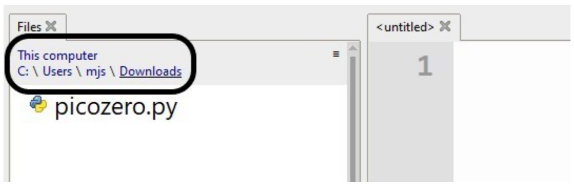
3. Choose a download location, and keep the file name the same - `picozero.py`
   Option 1 - Transfer files using Thonny file manager

4. On the computer connect your Raspberry Pi Pico using a microUSB cable.

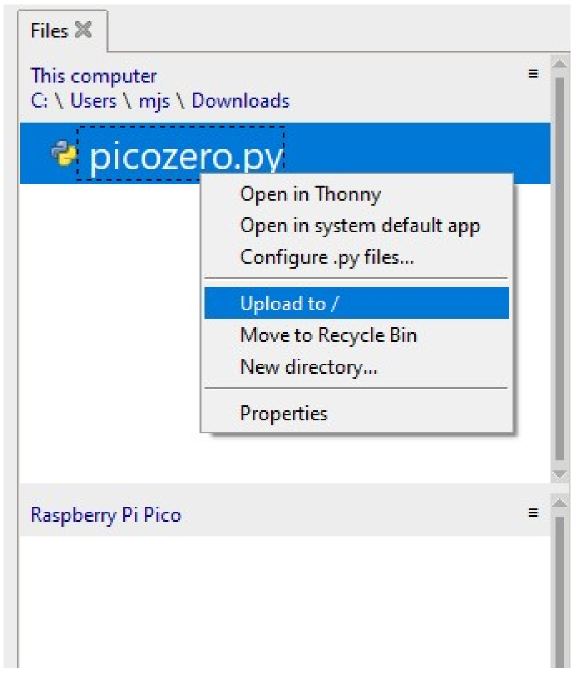
5. Load Thonny from your application menu, then from the **View** menu, choose to see files.



1. Use the path to navigate to the directory where you saved the `picozero.py` file.



1. Right click on the `picozero.py` and select **Upload to /** from the menu.



]

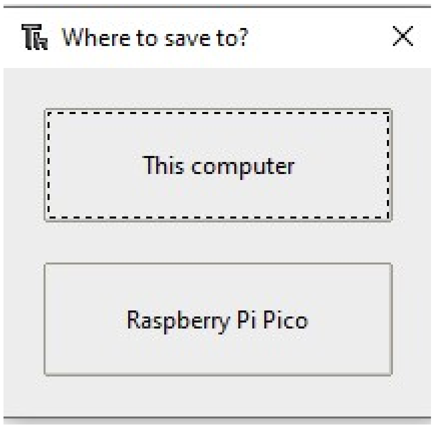
1. You should now see a new copy of the `picozero.py` library on the Raspberry Pi Pico.

Option 2 - Copy and paste the file using Thonny

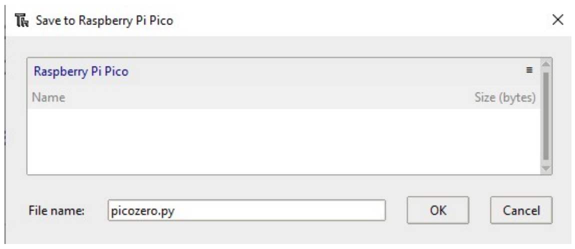
1. Select all the text in the `picozero.py` file by pressing **Ctrl + a** on your keyboard, then copy it by pressing **Ctrl + c**.

2. Open Thonny, click in the **untitled** tab and press **Ctrl + v** to paste the contents of `picozero.py` into the file.

3. Use **Ctrl + s** to save the file, and when prompted choose to save it to **Raspberry Pi Pico**



1. Name the file `picozero.py` and then click on the **OK** button.



File name: picozero.py

Step 3 Connect your Raspberry Pi Pico W to a WLAN

Here, you will learn to use MicroPython to connect your Raspberry Pi Pico W to a wireless local area network (WLAN), more commonly known as a WiFi network.

```
Shell
MicroPython v1.18-659-g7afd6452f-dirty on 2022-06-15; Raspberry Pi Pico W with RP2040
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT
 Waiting for connection...
 Waiting for connection...
 Waiting for connection...
 Connected on 192.168.1.143
>>>
```

Passwords need to be kept securely and privately. In this step, you will add your WiFi password into your Python file. Make sure you don't share your file with anyone that you wouldn't want to tell your password to.

To connect to a WiFi network, you will need to know your service set identifier (SSID). This is the name of your WiFi network. You will also need your WiFi password. These can usually be found written on your wireless router, although you should have changed the default password to something unique.

In Thonny, import the packages you will need to connect to your WiFi network, read the onboard temperature sensor, and light the onboard light-emitting diode (LED).

web_server.py

```python
import network
import socket
from time import sleep
from picozero import pico_temp_sensor, pico_led
import machine
```

Save this code now, and choose the option to save to **This computer**

Next, set up your Raspberry Pi Pico W to use the onboard LED, and additionally add in the SSID and password for your network.

web_server.py

```python
ssid = 'NAME OF YOUR WIFI NETWORK'
password = 'YOUR SECRET PASSWORD'
```

Now, begin to build a function to connect to your WLAN. You need to set up a `wlan` object, activate the wireless, and provide the object with your `ssid` and `password`.

web_server.py

```python
def connect():
    #Connect to WLAN
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    wlan.connect(ssid, password)
```

If you've ever connected a device to a WiFi network, you will know that it doesn't happen instantly. Your device will send requests to your WiFi router to connect, and when the router responds, they will perform what is called a handshake to establish a connection. To do this with Python, you can set up a loop that will keep sending requests each second until the connection handshake has been performed.

web_server.py

```python
def connect():
    #Connect to WLAN
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    wlan.connect(ssid, password)
    while wlan.isconnected() == False:
        print('Waiting for connection...')
        sleep(1)
```

Now print out your WLAN configuration, and test it all. You'll need to call your function. Keep all your function calls at the bottom of your file, so they are the last lines of code that are run. Because the WiFi connection can stay up, even when you stop the code, you can add a `try/except` that will reset the Raspberry Pi Pico W when the script is stopped.

web_server.py

```python
def connect():
    #Connect to WLAN
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    wlan.connect(ssid, password)
    while wlan.isconnected() == False:
        print('Waiting for connection...')
        sleep(1)
    print(wlan.ifconfig())

try:
    connect()
except KeyboardInterrupt:
    machine.reset()
```

Test: Save and run your code. You should see some output in the shell that looks something like this, although the specific IP addresses will be different.

```
Waiting for connection...
Waiting for connection...
Waiting for connection...
Waiting for connection...
Waiting for connection...
('192.168.1.143', '255.255.255.0', '192.168.1.254', '192.168.1.254')
```

The Raspberry Pi Pico W won't connect

1. Make sure that you are using the correct SSID and password.
2. If you are on a school or work WLAN, unauthorised devices might not be permitted access to the WiFi.

3. Unplug your Raspberry Pi Pico W from your computer to power it off, then plug it back in. This can be a problem when you have connected once, and then try to connect again.

You don’t need all the information provided by `wlan.ifconfig()`. The key information you need is the IP address of the Raspberry Pi Pico W, which is the first piece of information. You can use an **fstring** to output the **IP address**. By placing an `f` in front of your string, variables can be printed when they are surrounded by `{}`.

web_server.py

```python
def connect():
    #Connect to WLAN
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    wlan.connect(ssid, password)
    while wlan.isconnected() == False:
        print('Waiting for connection...')
        sleep(1)
    ip = wlan.ifconfig()[0]
    print(f'Connected on {ip}')


try:
    connect()
except KeyboardInterrupt:
    machine.reset()
```

You can now return the value for the IP address of your Raspberry Pi Pico W, and store it when you call your function.

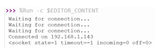
web_server.py

```python
def connect():
    #Connect to WLAN
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    wlan.connect(ssid, password)
    while wlan.isconnected() == False:
        print('Waiting for connection...')
        sleep(1)
    ip = wlan.ifconfig()[0]
    print(f'Connected on {ip}')
    return ip


try:
    ip = connect()
except KeyboardInterrupt:
    machine.reset()
```

Save your project

Step 4 Open a socket

In this step, you will use the connection to your WLAN to open a socket.



```
>>> %Run -c $EDITOR_CONTENT
 Waiting for connection...
 Waiting for connection...
 Waiting for connection...
 Connected on 192.168.1.143
 <socket state=1 timeout=-1 incoming=0 off=0>
```

A socket is the way a **server** can listen for a **client** that wants to connect to it. The webpage you are currently looking at is hosted on Raspberry Pi Foundation servers. These servers have an open socket that waits for your web browser to make a connection, at which point the contents of the webpage are sent to your computer. In this case, your server is going to be your Raspberry Pi Pico W and the client will be a web browser on another computer.

To open a socket, you need to provide the IP address and a port number. Port numbers are used by computers to identify where requests should be sent. For instance, port `80` is normally used for web traffic; Stardew Valley uses port `24642` when you're playing a multiplayer game. As you are setting up a web server, you will be using port `80`.

Create a new function that can be called to open a socket. It should be above your `try`/`except`. Start by giving the socket an IP address and a port number.

web_server.py

```python
def open_socket(ip):
    # Open a socket
    address = (ip, 80)


try:
    connect()
except KeyboardInterrupt:
    machine.reset()
```

Now create your socket, and then have it listen for requests on port `80`. Don't forget to call your function at the bottom of your code.

web_server.py

```python
def open_socket(ip):
    # Open a socket
    address = (ip, 80)
    connection = socket.socket()
    connection.bind(address)
    connection.listen(1)
    print(connection)

try:
    ip = connect()
    open_socket(ip)
except KeyboardInterrupt:
    machine.reset()
```

Test: Run your code, and you should see an output that looks something like this.

```
>>> %Run -c $EDITOR_CONTENT
Waiting for connection...
Waiting for connection...
Waiting for connection...
Waiting for connection...
Waiting for connection...
Connected on 192.168.1.143
<socket state=1 timeout=-1 incoming=0 off=0>
```

`socket state=1` tells you that your socket is working.

Lastly, replace your `print` with a `return` and then store the returned socket connection as a variable.

web_server.py

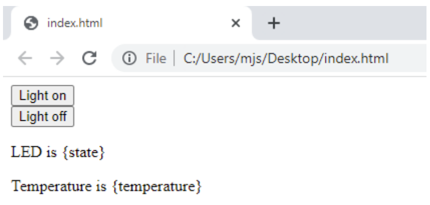
```
def open_socket(ip):
    # Open a socket
    address = (ip, 80)
    connection = socket.socket()
    connection.bind(address)
    connection.listen(1)
    return connection


try:
    ip = connect()
    connection = open_socket(ip)
except KeyboardInterrupt:
    machine.reset()
```

You now have your Raspberry Pi Pico W listening for connections to its IP address on port `80`. This means that it is ready to start serving HTML code, so that a connected web browser can see a webpage.

Save your project

Step 5 Create a webpage

In this step, you will create a webpage that the web server, running on your Raspberry Pi Pico W, can send to a client web browser. You're going to test the webpage on your computer first though, to make sure it displays as it should. In the next step, you can add the code to your Python script, so that your Raspberry Pi Pico W can serve the webpage.



LED is {state}

Temperature is {temperature}

A webpage can be as simple as some text, formatted in such a way that a web browser will render it and provide some interactivity. Although Thonny is not designed to write HTML, it can be used for this purpose. However, you can use your preferred text editor if you like, be that VSCode, TextEdit, or Notepad.

In your text editor or in Thonny, create a new file. You can call it whatever you like, but `index.html` is the standard name for the first page that a user interacts with. Make sure you add the `.html` file extension. If using Thonny, make sure to save to **This computer**.

There is some standard HTML code that you will need to include to begin with.

index.html

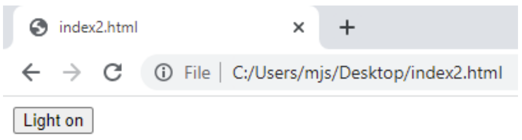
```html
<!DOCTYPE html>
<html>
<body>
</body>
</html>
```

Next, you can create a button that will be used to turn the onboard LED on or off.

index.html

```html
<!DOCTYPE html>
<html>
<body>
<form action="./lighton">
<input type="submit" value="Light on" />
</form>
</body>
</html>
```

Save your file and then find it in your file manager. When you double click the file, it should open in your default web browser. Here is what the webpage looks like in Google Chrome.



Add a second button to turn the LED off.

index.html

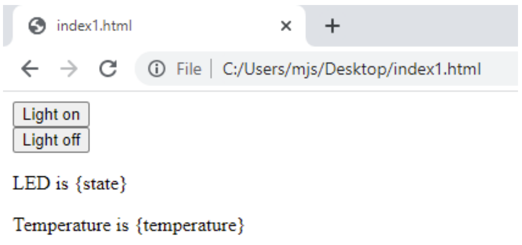
```
<!DOCTYPE html>
<html>
<body>
<form action="./lighton">
<input type="submit" value="Light on" />
</form>
<form action="./lightoff">
<input type="submit" value="Light off" />
</form>
</body>
</html>
```

To finish off the webpage, you can add in some extra data, such as the state of the LED and the temperature of your Raspberry Pi Pico W.

index.html

```
<!DOCTYPE html>
<html>
<body>
<form action="./lighton">
<input type="submit" value="Light on" />
</form>
<form action="./lightoff">
<input type="submit" value="Light off" />
</form>
<p>LED is {state}</p>
<p>Temperature is {temperature}</p>
</body>
</html>
```

Your webpage should look like this:



LED is {state}

Temperature is {temperature}

Now that you have a working webpage, you can add this code into your Python script. You'll need to switch back to your Python code in Thonny first.

Create a new function called `webpage`, that has two parameters. These are `temperature` and `state`.

web_server.py

```
def webpage(temperature, state):
    #Template HTML
```

You can now store all your HTML code that you have written and tested in a variable. Using **fstrings** for the text means that the placeholders you have in the HTML for `temperature` and `state` can be inserted into your string.

web_server.py

```
def webpage(temperature, state):
    #Template HTML
    html = f"""
            <!DOCTYPE html>
            <html>
            <form action="./lighton">
            <input type="submit" value="Light on" />
            </form>
            <form action="./lightoff">
            <input type="submit" value="Light off" />
            </form>
            <p>LED is {state}</p>
            <p>Temperature is {temperature}</p>
            </body>
            </html>
            """
```

Lastly, you can return the `html` string from your function.

web_server.py

```
def webpage(temperature, state):
    #Template HTML
    html = f"""
            <!DOCTYPE html>
            <html>
            <form action="./lighton">
            <input type="submit" value="Light on" />
            </form>
            <form action="./lightoff">
            <input type="submit" value="Light off" />
            </form>
            <p>LED is {state}</p>
            <p>Temperature is {temperature}</p>
            </body>
            </html>
            """
    return str(html)
```

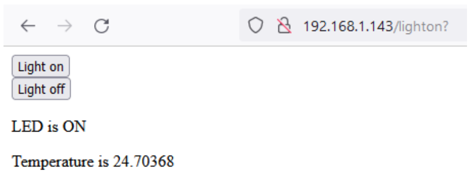
Save your project

You can’t test this code yet, as your program is not yet serving the HTML. That will be tackled in the next step.

The simple HTML code you have just written will be stored used in your MicroPython script and served to the browser of any computers that connect to it over your network, just like a webpage stored on any other server in the world. An important difference is that only devices connected to your WiFi network can access the webpage or control your Raspberry Pi Pico W. This page is a very simple demonstration of what is possible. To learn more about HTML coding and creating websites, see some of our other projects on this site!

Step 6 Serve your webpage

In this step, you will start up your web server so that a client can connect to it, and control your LED and read the temperature.



LED is ON

Temperature is 24.70368

Create a function that will start your web server, using the `connection` object you saved as a parameter. The `state` and `temperature` variables need to be set for your HTML data. The state is going to start as being set to `'OFF'`, and the temperature to `0`, which means you should also ensure that the LED is off when the server starts.

web_server.py

```python
def serve(connection):
    #Start a web server
    state = 'OFF'
    pico_led.off()
    temperature = 0
```

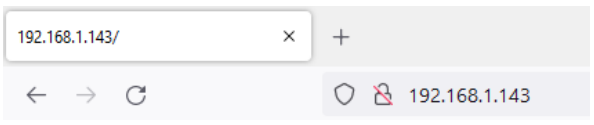
When your web browser asks for a connection to your Raspberry Pi Pico W, the connection needs to be accepted. After that, the data that is sent from your web browser must be done in specific chunks (in this case, 1024 bytes). You also need to know what request your web browser is making — is it asking for just a simple page? Is it asking for a page that doesn't exist?

You want to keep the web server up and listening all the time, so that any client can connect to it. You can do this by adding a `while True:` loop. Add these five lines of code so that you can accept a request, and `print()` to see what the request was. Add a call to your `serve` function in your calls at the bottom of your code.

web_server.py

```python
def serve(connection):
    #Start a web server
    state = 'OFF'
    pico_led.off()
    temperature = 0
    while True:
        client = connection.accept()[0]
        request = client.recv(1024)
        request = str(request)
        print(request)
        client.close()


try:
    ip = connect()
    connection = open_socket(ip)
    serve(connection)
except KeyboardInterrupt:
    machine.reset()
```

Test: Run your program and then type in the IP address into a web browser's address bar on your computer.



You should see something like this in the shell output in Thonny.

```
>>> %Run -c $EDITOR_CONTENT
Waiting for connection...
Waiting for connection...
Waiting for connection...
Connected on 192.168.1.143
b'GET / HTTP/1.1\r\nHost: 192.168.1.143\r\nUser-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:101.0) Gecko/20100101
Firefox/101.0\r\nAccept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8\r\nAccept-
Language: en-GB,en;q=0.5\r\nAccept-Encoding: gzip, deflate\r\nConnection: keep-alive\r\nUpgrade-Insecure-Requests: 1\r\n\r\n'
b'GET /favicon.ico HTTP/1.1\r\nHost: 192.168.1.143\r\nUser-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:101.0)
Gecko/20100101 Firefox/101.0\r\nAccept: image/avif,image/webp,*/*\r\nAccept-Language: en-GB,en;q=0.5\r\nAccept-Encoding: gzip,
deflate\r\nConnection: keep-alive\r\nReferer: http://192.168.1.143/\r\n\r\n'
```

Next, you need to send the HTML code you have written to the client web browser.

web_server.py

```
def serve(connection):
    #Start a web server
    state = 'OFF'
    pico_led.off()
    temperature = 0
    while True:
        client = connection.accept()[0]
        request = client.recv(1024)
        request = str(request)
        print(request)
        html = webpage(temperature, state)
        client.send(html)
        client.close()


try:
    ip = connect()
    connection = open_socket(ip)
    serve(connection)
except KeyboardInterrupt:
    machine.reset()
```

Refresh your page when you've run the code again. Click on the buttons that are displayed. In Thonny, you should then see that there are two different outputs from your shell.

```
b'GET /lighton? HTTP/1.1\r\nHost: 192.168.1.143\r\nUser-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:101.0)
Gecko/20100101 Firefox/101.0\r\nAccept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8\r\nAccept-Language: en-
GB,en;q=0.5\r\nAccept-Encoding: gzip, deflate\r\nConnection: keep-alive\r\nReferer: http://192.168.1.143/\r\nUpgrade-
Insecure-Requests: 1\r\n\r\n'
```

and

```
b'GET /lightoff? HTTP/1.1\r\nHost: 192.168.1.143\r\nUser-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:101.0)
Gecko/20100101 Firefox/101.0\r\nAccept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8\r\nAccept-Language: en-
GB,en;q=0.5\r\nAccept-Encoding: gzip, deflate\r\nConnection: keep-alive\r\nReferer: http://192.168.1.143/lighton?
\r\nUpgrade-Insecure-Requests: 1\r\n\r\n'
```

Notice that you have `/lighton?` and `lightoff?` in the requests. These can be used to control the onboard LED of your Raspberry Pi Pico W.

Split the request string and then fetch the first item in the list. Sometimes the request string might not be able to be split, so it's best to handle this in a `try`/`except`.

If the first item in the split is `lighton?` then you can switch the LED on. If it is `lightoff?` then you can switch the LED off.

web_server.py

```
def serve(connection):
    #Start a web server
    state = 'OFF'
    pico_led.off()
    temperature = 0
    while True:
        client = connection.accept()[0]
        request = client.recv(1024)
        request = str(request)
        try:
            request = request.split()[1]
        except IndexError:
            pass
        if request == '/lighton?':
            pico_led.on()
        elif request =='/lightoff?':
            pico_led.off()
        html = webpage(temperature, state)
        client.send(html)
        client.close()
```

Run your code again. This time, when you refresh your browser window and click on the buttons, the onboard LED should turn on and off.

You can also tell the user of the webpage what the state of the LED is.

web_server.py

```python
def serve(connection):
    #Start a web server
    state = 'OFF'
    pico_led.off()
    temperature = 0
    while True:
        client = connection.accept()[0]
        request = client.recv(1024)
        request = str(request)
        try:
            request = request.split()[1]
        except IndexError:
            pass
        if request == '/lighton?':
            pico_led.on()
            state = 'ON'
        elif request =='/lightoff?':
            pico_led.off()
            state = 'OFF'
        html = webpage(temperature, state)
        client.send(html)
        client.close()
```

Now when you run the code, the text for the state of the LED should also change on the refreshed webpage.

Lastly, you can use the onboard temperature sensor to get an approximate reading of the CPU temperature, and display that on your webpage as well.

web_server.py

```python
def serve(connection):
    #Start a web server
    state = 'OFF'
    pico_led.off()
    temperature = 0
    while True:
        client = connection.accept()[0]
        request = client.recv(1024)
        request = str(request)
        try:
            request = request.split()[1]
        except IndexError:
            pass
        if request == '/lighton?':
            pico_led.on()
            state = 'ON'
        elif request =='/lightoff?':
            pico_led.off()
            state = 'OFF'
        temperature = pico_temp_sensor.temp
        html = webpage(temperature, state)
        client.send(html)
        client.close()
```

Test: You can hold your hand over your Raspberry Pi Pico W to increase its temperature, then refresh the webpage on your computer to see the new value that is displayed.

Step 7 What next?

If you have a look at **Introduction to Raspberry Pi Pico: LEDs, buzzers, switches, and dials** (https://projects.raspberrypi.org/en/pathways/pico-intro), you can find plenty of ideas for how to improve on your Raspberry Pi Pico web server.

In the **LED firefly** (https://projects.raspberrypi.org/en/projects/led-firefly) project, you can learn how to wire up external LEDs to a Raspberry Pi Pico, then you could use a web interface to change the pattern your firefly blinks with.

In the **Party popper** (https://projects.raspberrypi.org/en/projects/party-popper) project, you can learn how to use a switch to control an RGB LED and a buzzer. You could use a web form with this project to control the colour of the LED and the tone that the buzzer plays.

In **Beating heart** (https://projects.raspberrypi.org/en/projects/beating-heart), you learn to use a potentiometer to control a pulsing LED, but you could also use your browser to control the pulse rate, and the potentiometer could be used to change the appearance of your webpage.