Pico Project, general info

The end result of this project is a script with a working web server on your tiny Raspberry Pi Pico W.

So, this adventure starts (or started) with buying the Pico W and a connecting data wire. A Pico without the W cannot be used as it is missing the W from Wifi. And without Wifi there's no network connection.

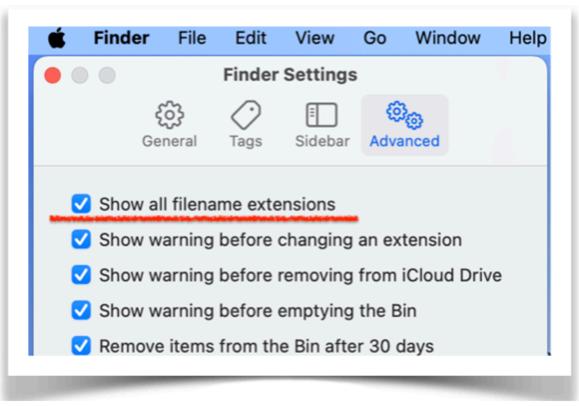
The idea behind this project is to do the research all by yourself or in small teams. Everything you need to know can be found online, some URL's are provided in this document. Experimenting, trial and error, it's all part of this adventure. If something doesn't work, do the research.

First things first: changing computer settings

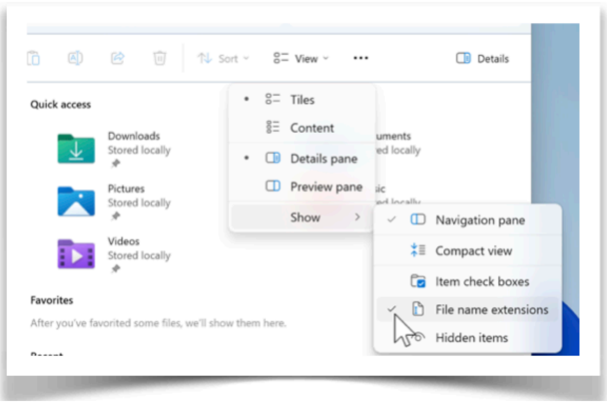
Some settings in the computer are nice for average users, but not for programmers. We have to change them first.

A nice explanation on how to do this can be found on howtogeek.com for instance. Search in Google or another search engine for *Windows 11* or *Mac show file extensions* and follow the steps to accomplish this.

If all is done right, from now on you see the file extension of all files. So a Word file now ends with .*docx* (not .odt). An image ends with .*png* or .*jpg* or so. Python files end with .*py*.



- ✓ Show all filename extensions
- ✓ Show warning before changing an extension
- ✓ Show warning before removing from iCloud Drive
- ✓ Show warning before emptying the Bin
- ✓ Remove items from the Bin after 30 days



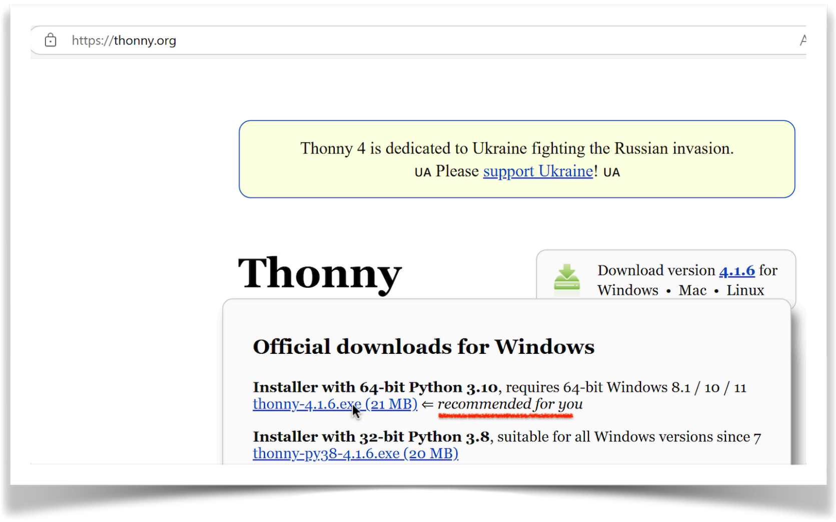
However Windows cannot open Python files without specific software installed. Mac offers to open it with TextEdit or Xcode. Don't do this, you'll install a simple editor later on.

It's **important** to know, that you cannot change a file extension manually and then expect the file to be different. If the file extension is .*jpg* it is a JPEG-file. If you want it be a .*png* file, you have to open it in some image editor and export it as PNG-file. Manually changing the extension leads to errors.

Installing and running Thonny

Thonny is a very simple but proper editor for writing and executing Python code. When you install Thonny, you also install Python (version 3.10), so you do not have to install Python separately.

https://thonny.org/



https://thonny.org

Thonny 4 is dedicated to Ukraine fighting the Russian invasion.
UA Please support Ukraine! UA

Thonny

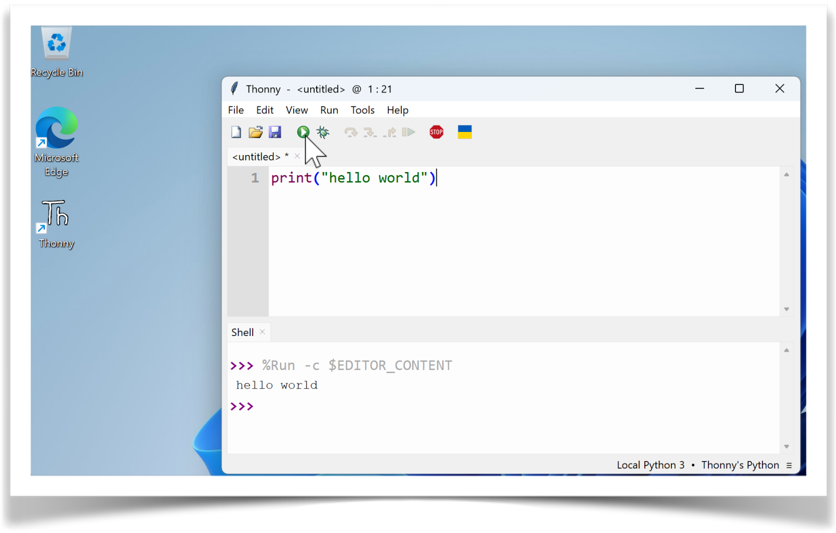
Download version 4.1.6 for Windows • Mac • Linux

Official downloads for Windows

Installer with 64-bit Python 3.10, requires 64-bit Windows 8.1 / 10 / 11
thonny-4.1.6.exe (21 MB) ⇐ *recommended for you*

Installer with 32-bit Python 3.8, suitable for all Windows versions since 7
thonny-py38-4.1.6.exe (20 MB)

Install Thonny on your computer and make sure that it works. Use the classic *Hello World* script to test Python.


Recycle Bin
Microsoft Edge
Thonny
Thonny - <untitled> @ 1 : 21
File Edit View Run Tools Help
<untitled> *
Shell
Local Python 3 • Thonny's Python

```
print("hello world")
```

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

Now that you have Python running in Thonny, you can build and run some of the applications made for the CS50python course in your own computer. Of course check50 and submit50 will not work, as they are predefined in the VS Code environment for CS50.

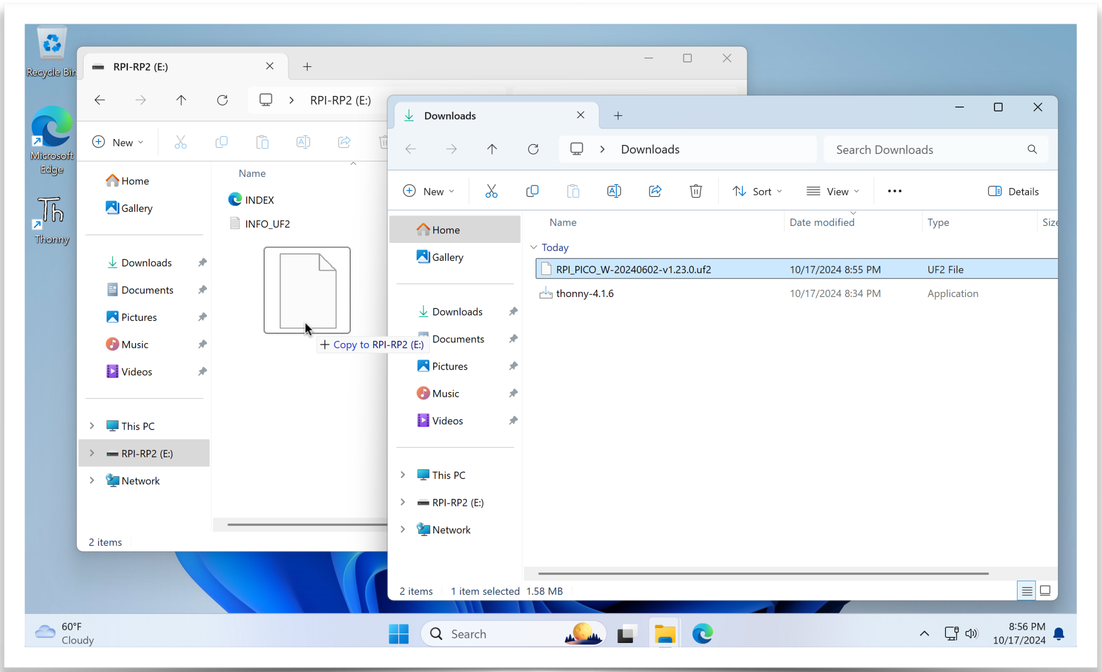
What does not work, is code that relies on external modules, such as *pyjokes* or *emoji*, as they have to be installed within Python *Pip install* does not work in Thonny.

For this project, we're not going to run Python in the computer, but in the Raspberry Pi Pico W.

Installing and connecting the Pico

For connecting the Pico with your computer, you need a proper USB data cable: USB A (normal male USB) to B (smaller male) or USB C (male) to B (male).

https://www.kiwi-electronics.com/nl/raspberry-pi-pico-wh-10940
https://www.kiwi-electronics.com/nl/microusb-kabel-usb-a-naar-micro-b-1-meter-76

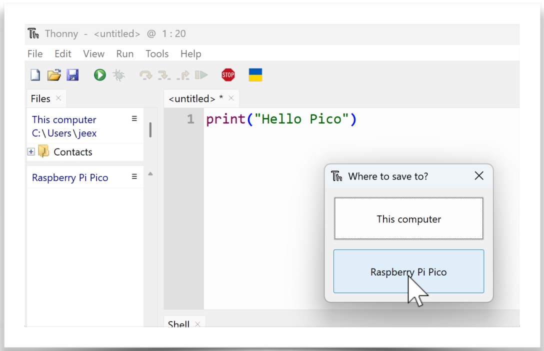


The Pico W is designed for educational use. Best place to start is here:

https://projects.raspberrypi.org/en/projects/get-started-pico-w/1
If you installed the proper *.uf2* file on your Pico W, the *network* package and *machine* package are installed automatically.

IMPORTANT: read and watch the instructions accurately. You can **skip** the *picozero* part.

The simplest way of testing if the Pico works, is by making a small program in Thonny and run it.



```
print("Hello Pico")
```

Write a one-liner. Press Ctrl + S or use the menu in Thonny to Save the file.

Then store it on the Raspberry Pi Pico with the filename **test.py**.

Now press on the green Run button. If it shows MPY: soft reboot and prints the line of text, Python runs in your Pico.

Important: if you call the file **boot.py** or **main.py**it will startup automatically at startup, best not use these names.

Onboard stuff

The Pico has a small LED on board. You can make this light up or blink with a few lines of code.

There are many examples about how to make the onboard LED of the Pico light up. Follow one or many of them for this assignment. A lot of them address externally connected LED's. That works the same, only the *physical address* or *port* or *pins* of the LED is different.

Finding the correct *physical address* of the onboard LED is an issue. Some resources say 25, others say 1, the proper address however is "LED". So, a string, not a number, see below code:

```
import machine
led = machine.Pin("LED", machine.Pin.OUT)
```

Onboard the Pico Pi has a temperature sensor. Not for measuring the room temperature, but for measuring the temperature of the Pico itself. For that you also need the built in machine library, to be specific the ADC class.

```
adc = machine.ADC(4)
```

More info about this class can be found in the many tutorials about the Pico Pi.

Time is another onboard thing. Sort of. You can use the time library to get the current unix timestamp and put that in a more readable format, with date and time.

```
import time
timestamp = time.time() # for utc timestamp
localtime = time.localtime() # for a list with time info
```

Connect to Wifi and internet

For connecting your Pico Pi W to your local Wifi, you need some background information about network connections and Wifi.

People often use the word *Wifi* when they mean *internet* and vice versa. The Wifi is an *internal* wireless network at your home or at school, also called WLAN.

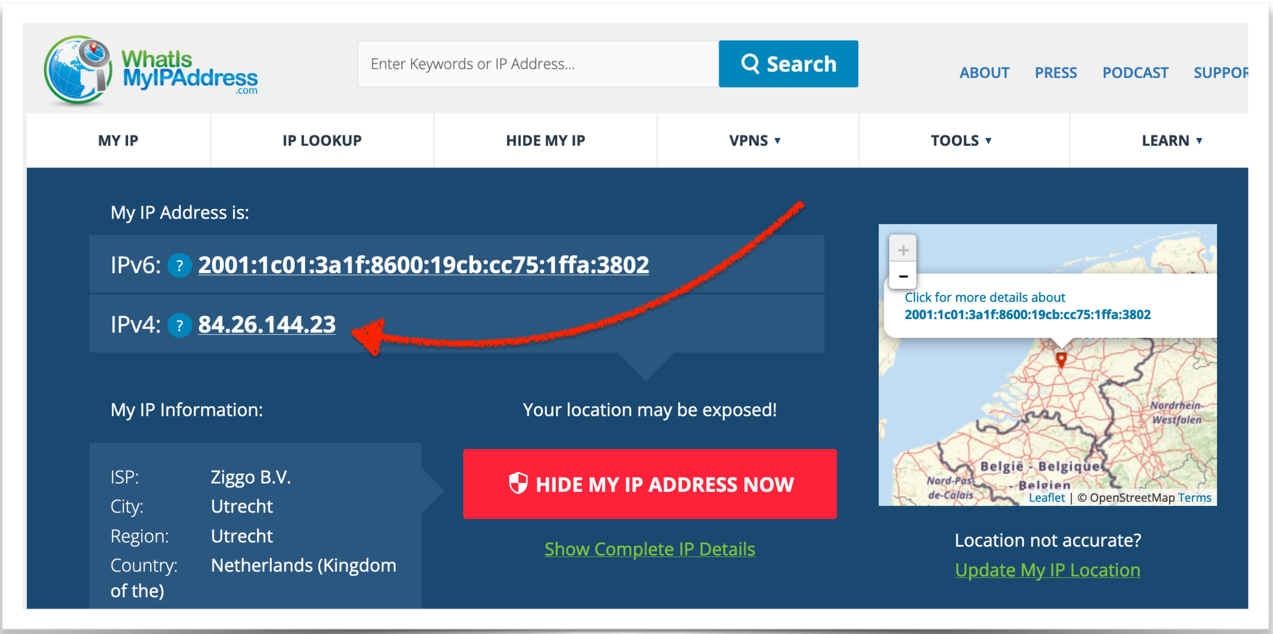
Being connected to the local internal network does not automatically mean that you're connected to the internet. Somewhere in your home (or school) the local network is linked to the outside network, called internet, by a router.

If that outside connection fails, you're not connected to the internet, while the internal network, such as the Wifi might still function properly.

So, in a way you're connected to two networks: local and internet. That is why a Wireless connection (Wifi) can show two different IP addresses:
• the internal IP address of your device within the local network
• the external IP address of your local network to the internet

Testing if Wifi works is one step. After being connected to the Wifi you also need to test if the internal network is connected to the outside world.

The internal IP address most of the time starts with 192.168...
The external IP address can be checked by visiting https://whatismyipaddress.com/



My IP Address is:

IPv6: 2001:1c01:3a1f:8600:19cb:cc75:1ffa:3802

IPv4: 84.26.144.23

My IP Information:

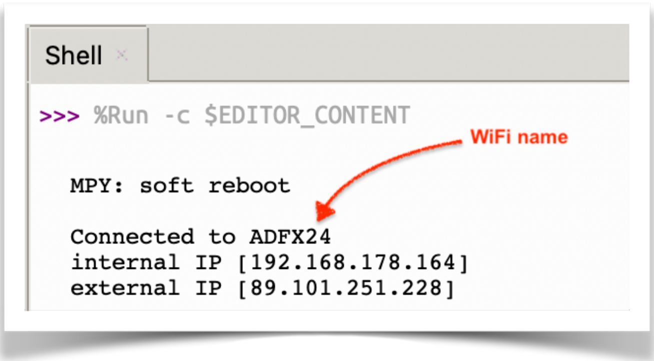
Your location may be exposed!

ISP:	Ziggo B.V.
City:	Utrecht
Region:	Utrecht
Country:	Netherlands (Kingdom of the)

Click for more details about 2001:1c01:3a1f:8600:19cb:cc75:1ffa:3802

Location not accurate?

https://projects.raspberrypi.org/en/projects/get-started-pico-w/2



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

  MPY: soft reboot

  Connected to ADFX24
  internal IP [192.168.178.164]
  external IP [89.101.251.228]
```

Of course you need the *access credentials* for your local network. In above example the name (**SSID**) of the Wifi is ADFX24. There's a **password** necessary to connect to this Wifi. For the Pico you can use the same Wifi connection as with your laptop or smartphone. So you do need the SSID and password of your local Wifi.

Web server

For a web server on the Pico two approaches are possible: using *socket* or using *uasyncio*. Both packages are available on Micropython. As the ***socket*** is the simplest, we use the simpler ***socket*** for this example.

More about socket in general can be found here:
https://www.tutorialspoint.com/unix_sockets/what_is_socket.htm

This part will not work without a working Wifi connection.

Make sure that the function of your Wifi-connection returns the *internal* IP address. Socket needs this IP address.

Your main() then may now look like:

```
def main():
    local_ip = Wifi_connect()
    if local_ip is None:
        sys.exit(1)

    valid_internet = check_internet()
    server(local_ip) # this is to call the server function
```

https://projects.raspberrypi.org/en/projects/get-started-pico-w/3

The skeleton of your server() function will look like this. One thing is very important to add: the **KeyboardInterrupt** exception.

```
def server(ip):
    # connect and stuff
    # IMPORTANT LINE TO MAKE CONNECTION RESTART POSSIBLE
    name_of_your_connection.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    # etc...

    while True: # because it waits for requests from browsers
        try:
            # client stuff
            # request stuff, where your script gets information from the browser

            # response stuff, where your script sends info back to the browser

        except KeyboardInterrupt:
            # IMPORTANT SO YOU CAN STOP THE SERVER WITH Ctrl+C
            print(f"server {ip} closed by: Ctrl+C")
            sys.exit(1)

        except Exception as e:
            print(f"server {ip} closed by: {e}")
            sys.exit(1)
```

This exception makes it possible to stop the script at any time with Ctrl+C (or Ctrl+Z). Of course you need to import *sys* in the top of your program and call *sys.exit(1)* after the interrupt. Otherwise it will only stop by unplugging the Pico.

Another practical thing about above script is the code in line 4.

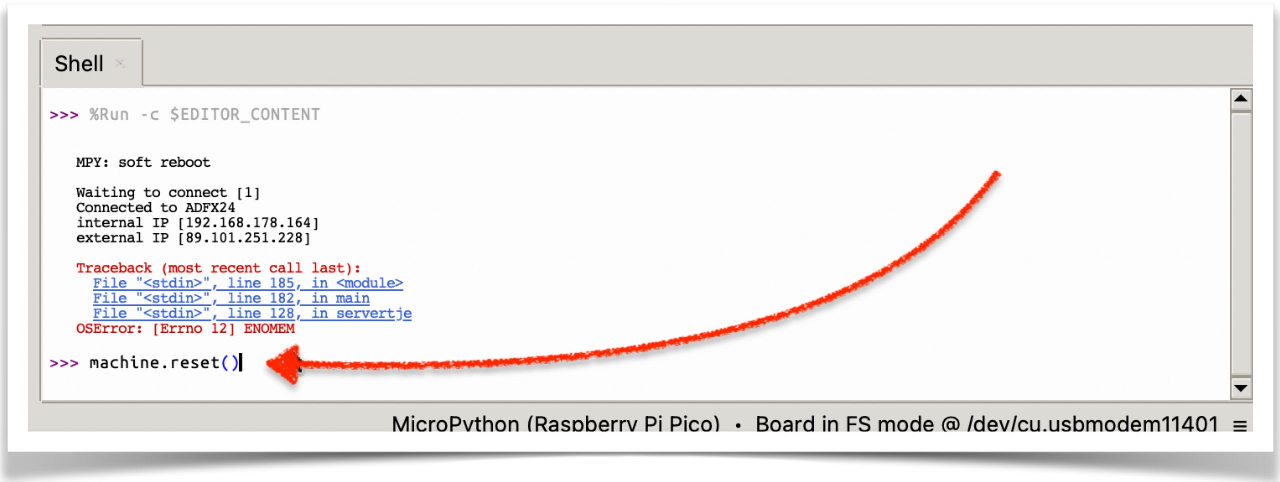
```
name_of_your_connection.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
```

This takes care of clearing used addresses and it prevents irritable errors when restarting the server.

Sometimes your server will not restart properly, like in below example. In that case type the following command in the shell:

```
machine.reset()
```

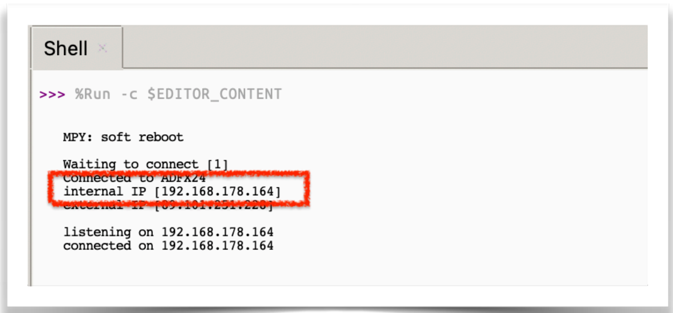
Then **wait for 10 seconds** before restarting the server with the green *Run current script* button.


Shell
MicroPython (Raspberry Pi Pico) • Board in FS mode @ /dev/cu.usbmodem11401

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

  MPY: soft reboot

  Waiting to connect [1]
  Connected to ADFX24
  internal IP [192.168.178.164]
  external IP [89.101.251.228]

  Traceback (most recent call last):
    File "<stdin>", line 185, in <module>
    File "<stdin>", line 182, in main
    File "<stdin>", line 128, in servertje
  OSError: [Errno 12] ENOMEM
>>> machine.reset()
```

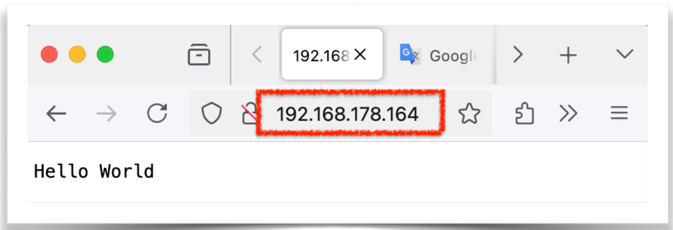
Another pointer: do not call your scripts *main.py* or *boot.py* or *app.py* as these startup automatically when the Pico boots.

Follow the steps in https://projects.raspberrypi.org/en/projects/get-started-pico-w/3**.**



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

  MPY: soft reboot

  Waiting to connect [1]
  Connected to ADFX24
  internal IP [192.168.178.164]
  external IP [89.101.251.228]

  listening on 192.168.178.164
  connected on 192.168.178.164
```

After following above instructions without adding any HTML, your web server can send a simple line of text to the browser:



```
Hello World
```

Web server: HTML

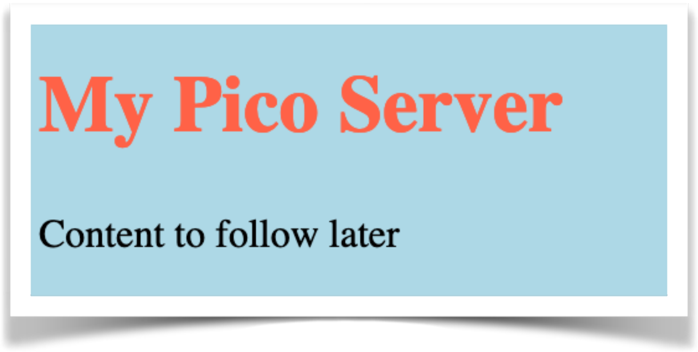
Once your web server is running, you can have it serve a proper web page. Simply by replacing the text “Hello World” from the previous example by proper HTML text.

https://randomnerdtutorials.com/raspberry-pi-pico-w-http-requests-micropython/
https://www.w3schools.com/html/html_basic.asp

Place your HTML-code in a function like:

```
def my_html():
    return """<!DOCTYPE html>
        insert your html here
    """
```

Feel free to add some styling to your HTML using CSS.



My Pico Server

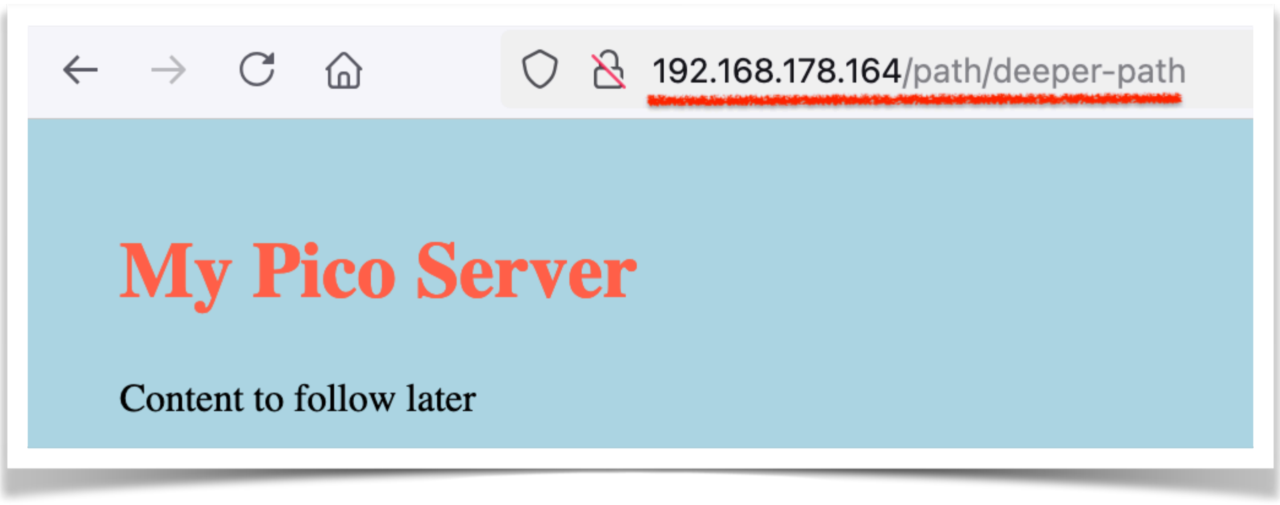
Content to follow later

https://www.codecademy.com/article/html-inline-styles

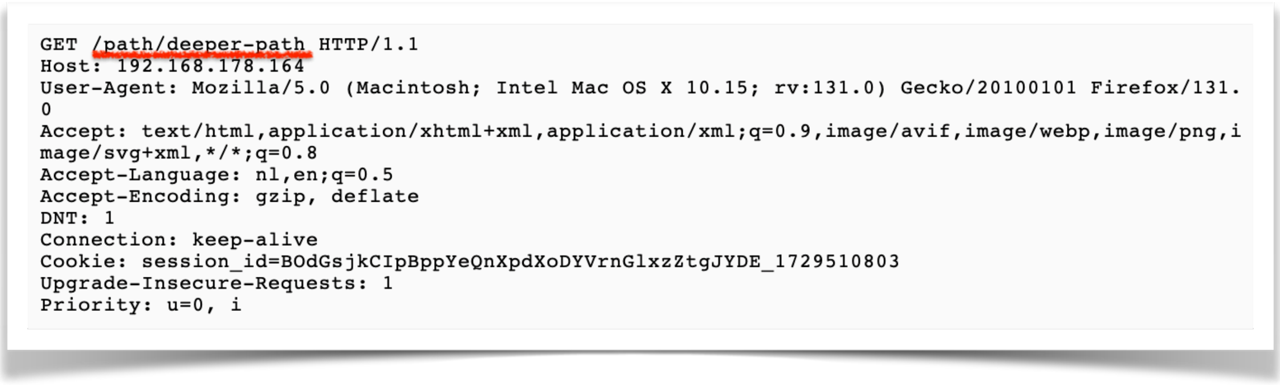
Web server: paths, args & cookies

Websites have paths. A website's home address looks like cpnits.com, while a path could be like *cpnits.com/progress/* where */progress* is the path inside the website to a different page.

if you ask your browser for *cpnits.com/progress/* the server receives this as the so called *request*. The *path* is a part of the so called *request headers*, received by the server from the browser.



These *request headers* contain all sorts of information, such as the type of computer used by the visitor (client), the type of browser etc.

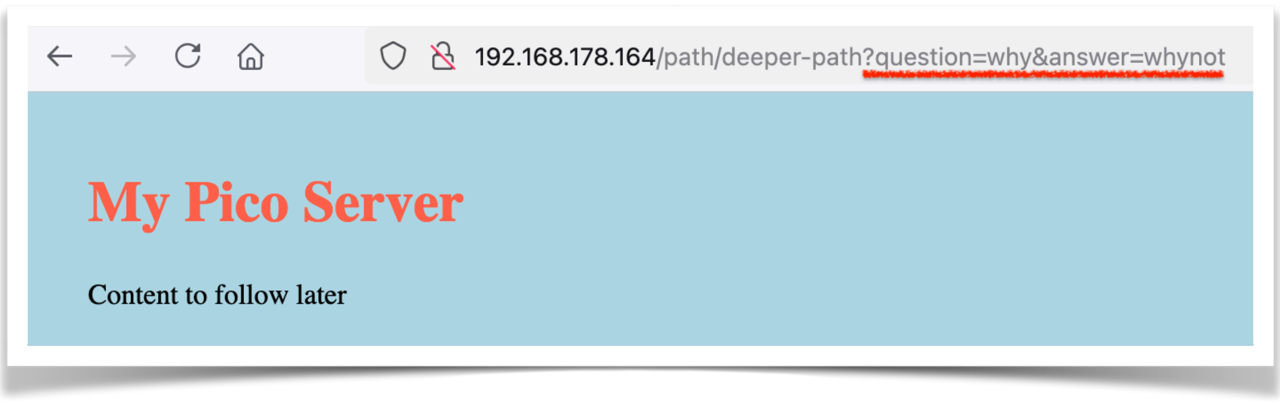


```
GET /path/deeper-path HTTP/1.1
Host: 192.168.178.164
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:131.0) Gecko/20100101 Firefox/131.
0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,i
mage/svg+xml,*/*;q=0.8
Accept-Language: nl,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Connection: keep-alive
Cookie: session_id=BOdGsjkCIpBppYeQnXpdXoDYVrnGlxzZtgJYDE_1729510803
Upgrade-Insecure-Requests: 1
Priority: u=0, i
```

The path is: */path/deeper-path* as requested by the browser to the server.

The server can then decide to show a different HTML-page with other contents or functionality. You can place hyperlinks in your HTML containing paths like these.

Arguments
With the browser, you can also send URL arguments. These are added to the server address and path, like:

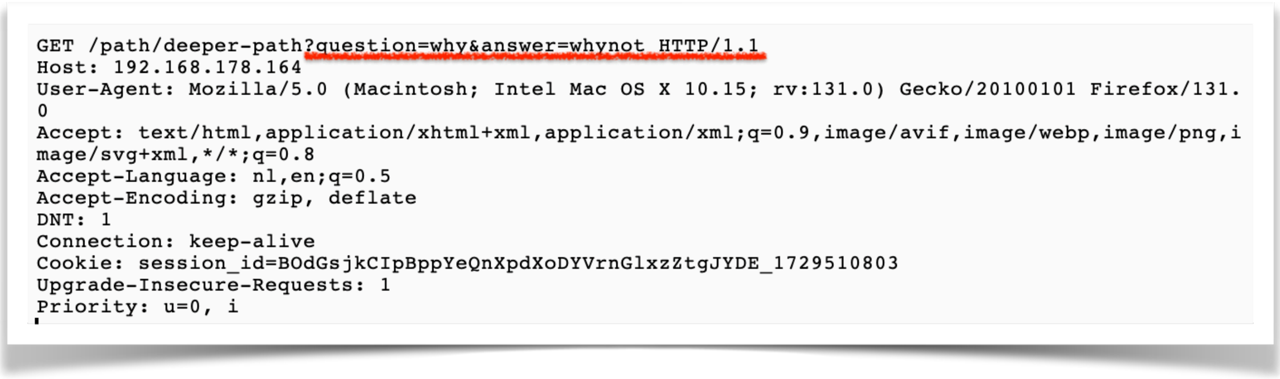


My Pico Server

Content to follow later

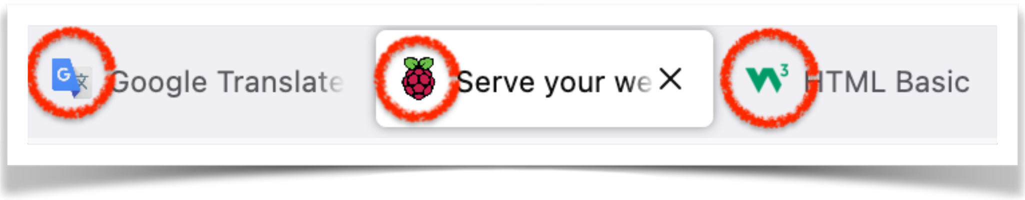
These *key-value* arguments always start with a **?**. Different key-value arguments are separated by a **&**. Like in above example, where each argument is a *key-value* pair using **=** as separator between key and value. Above example contains two key-value arguments:

- question=why
- answer=why not

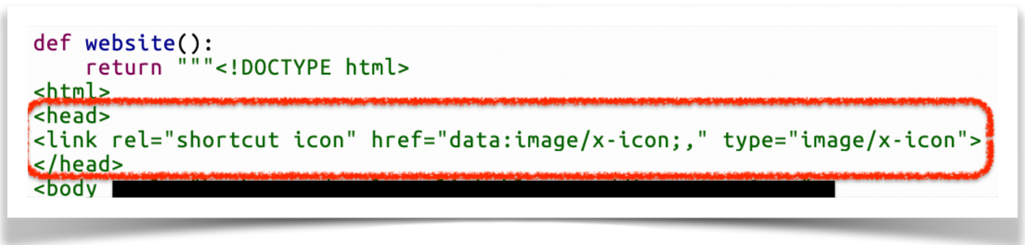


```
GET /path/deeper-path?question=why&answer=whynot HTTP/1.1
Host: 192.168.178.164
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:131.0) Gecko/20100101 Firefox/131.
0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,i
mage/svg+xml,*/*;q=0.8
Accept-Language: nl,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Connection: keep-alive
Cookie: session_id=BOdGsjkCIpBppYeQnXpdXoDYVrnGlxzZtgJYDE_1729510803
Upgrade-Insecure-Requests: 1
Priority: u=0, i
```

Hints:
- the path and arguments are always in the first element in the header list, as generated in above code example.
- there's an irritating thing about browsers and websites: they always send extra requests for a so called *favicon.png*. This is the image as shown in the browser tab.



You can add a few lines of code to your HTML to prevent this request from being sent:

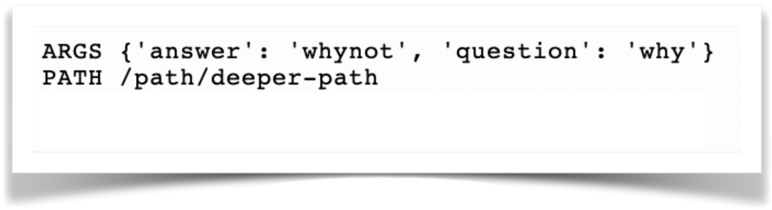


```
def website():
    return """<!DOCTYPE html>
<html>
<head>
<link rel="shortcut icon" href="data:image/x-icon;," type="image/x-icon">
</head>
<body [redacted]
```

In the end, the calls and output of the functions for path and arguments could look like this:

```
path = get_path(headers)
print('ARGS', args)
args = get_args(headers)
print('PATH', path)
cookies = get_cookies(headers)
print('COOKIES', cookies)
```

where you of course must build the functions *get_path()* and *get_args()*. This would create output like:



```
ARGS {'answer': 'whynot', 'question': 'why'}
PATH /path/deeper-path
```

Cookies are included in a thing called *headers*. Getting the cookies is much like getting the arguments. Just print() all the lines in the *reader* object of the request, and look for the line starting with ***Cookie***. See the image on the previous page. Also cookies are key-value pairs, connected with **=**. In above example the cookie is *session_id=BOdgs…*, where the cookie name is *session_id* and the cookie value is the long string starting with *BOdgs*.

By now, you've learned from scratch how a browser sends a request and how a server receives and crunches that request into a response. A lot of professional web programmers around the world have never done anything like this and lack the basic knowledge about this process. Kudos for you.