SWCON104
Web & Python Programming

Web Crawling and Web Server Development

Department of Software Convergence

경희대학교
KYUNG HEE UNIVERSITY

Today

- Let’s Crawl and Build a Web Server using Python

Practice

- Practice_21_WebServerUsingPython

Crawling a web server

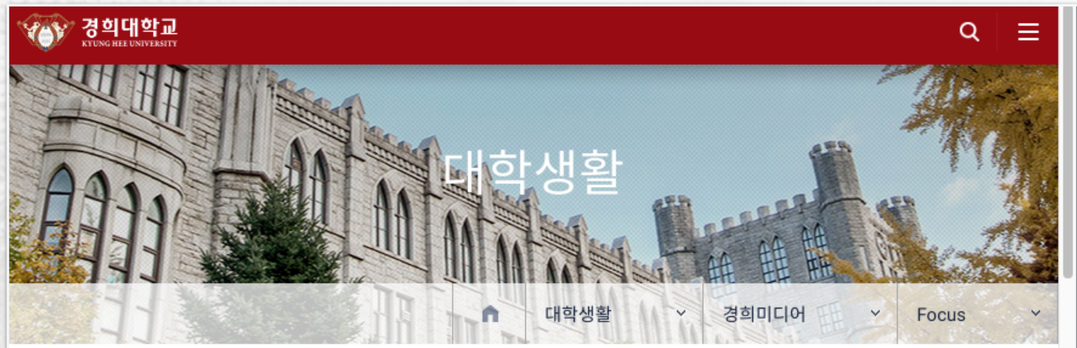
- Tools Installation
  - Requests: pip install requests

    » https://requests.readthedocs.io/en/latest/
  - Beautiful Soup: pip install bs4

    » https://www.crummy.com/software/BeautifulSoup/

경희대학교 대학생활 Focus 기사 제목 크롤링

- Site access and check source code
  - https://khu.ac.kr/kor/focus/detail.do?seq=2164725&page=1&pageSize=5



Focus

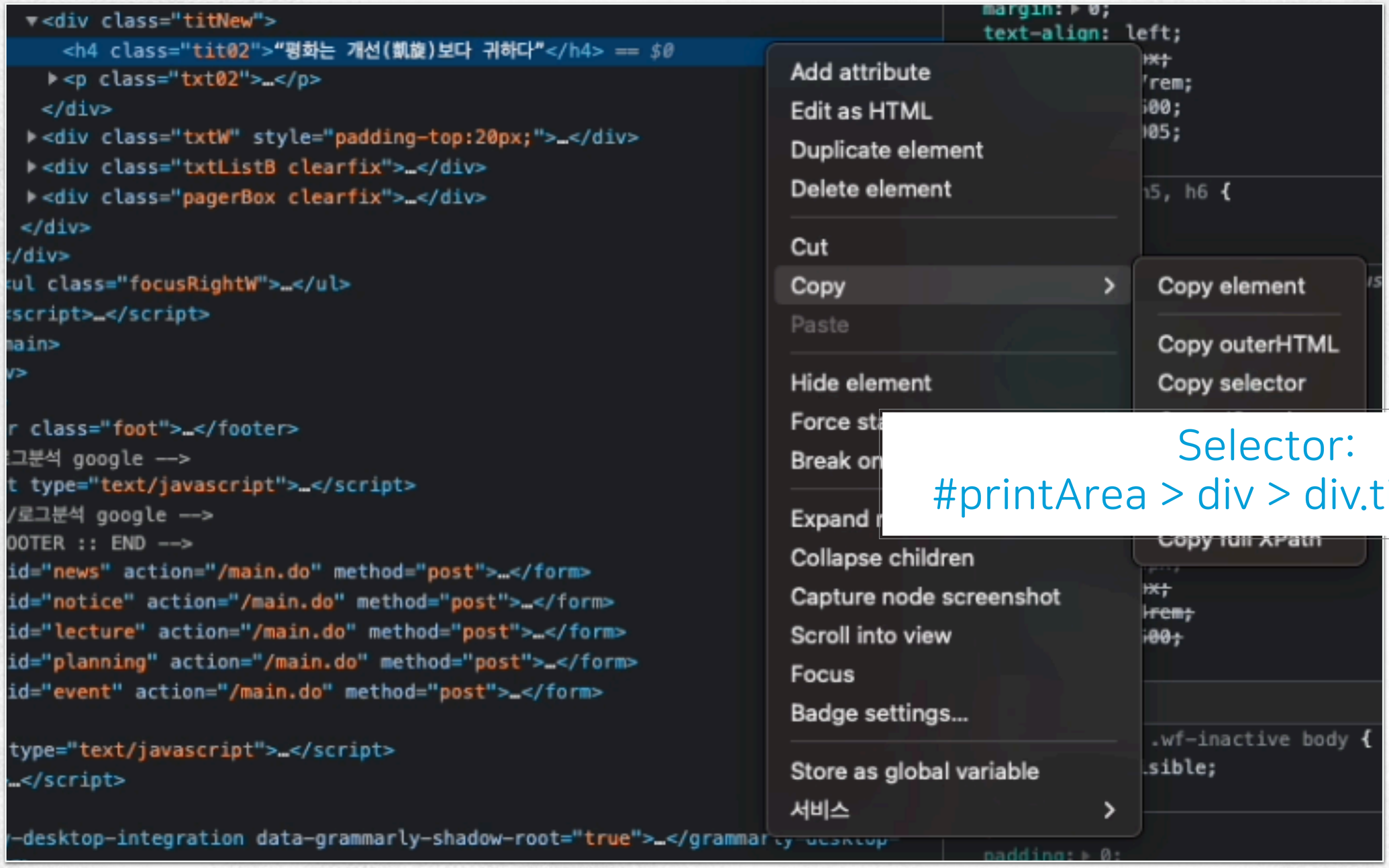
“평화는 개선(凱旋)보다 귀하다”

2022-10-27 교류/실천

경희학원이 제41회 유엔 세계평화의 날 기념 Peace BAR Festival·미원 서거 10주기 기념식의 일환으로 개최한 미원 서거 10주기 기념 심포지엄이 9월 21일(수) 평화의 전당에서 진행됐다.

경희대학교 대학생활 Focus 기사 제목 크롤링

- Check selectors & tags
  - https://khu.ac.kr/kor/focus/detail.do?seq=2164725&page=1&pageSize=5



경희대학교 대학생활 Focus 기사 제목 크롤링

- Create code and crawl target information
  - https://khu.ac.kr/kor/focus/detail.do?seq=2164725&page=1&pageSize=5

```python
import requests
from bs4 import BeautifulSoup

req = requests.get('https://khu.ac.kr/kor/focus/detail.do?seq=2164725&page=1&pageSize=5')
html = req.text
soup = BeautifulSoup(html, 'html.parser')
my_titles = soup.select(
    '#printArea > div > div.titNew > h4'
    )

for title in my_titles:
    print(title.text)
```

Code
webCrawling_1.py

Output

```
drsungwon~$ python webCrawling_1.py
"평화는 개선(凱旋)보다 귀하다"
```

Building a web server

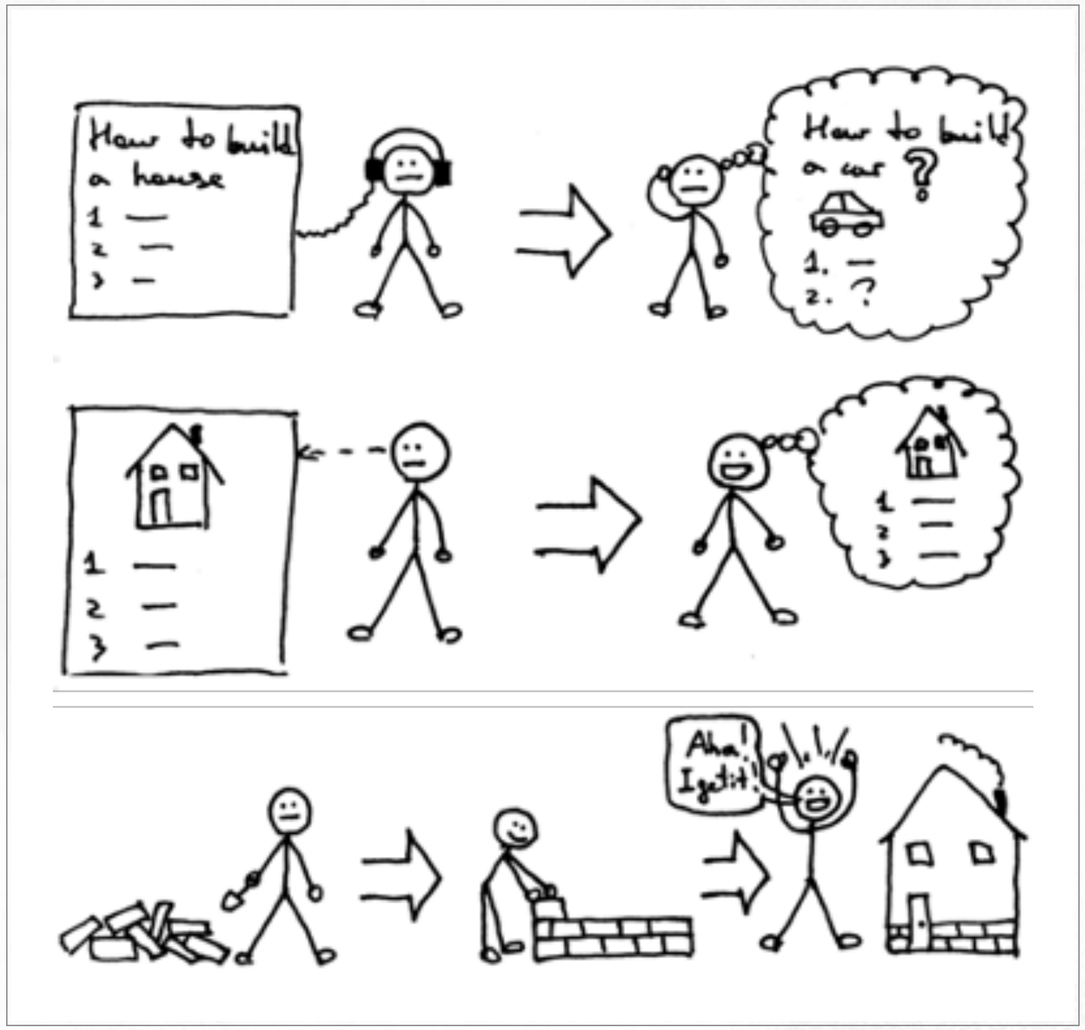
- Why?
  - Understands conventional web services in view of developer
  - Develops HTML/CSS/JS web service server using python
  - Develops server for backend and micro services

"I hear and I forget
I see and I remember
I do and I understand"

Confucius

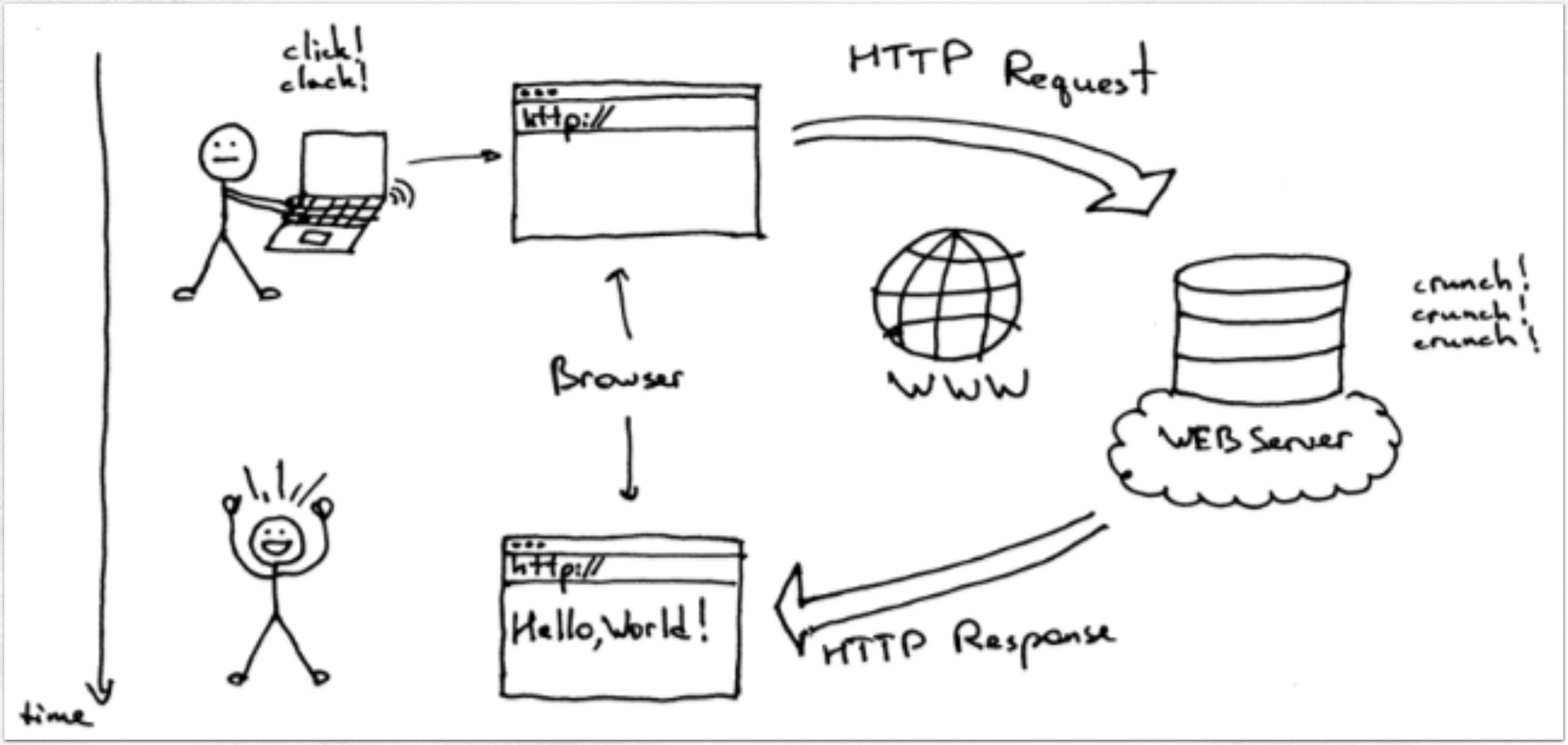
听而易忘
见而易记
做而易懂

孔子



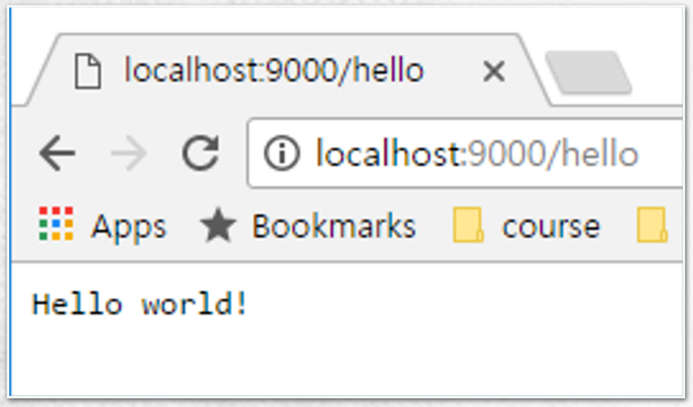
What is a web server?

- Waits for a client to send a request
- When it receives a request, it generates a response and sends it back to the client



Simple web server: webserver1.py

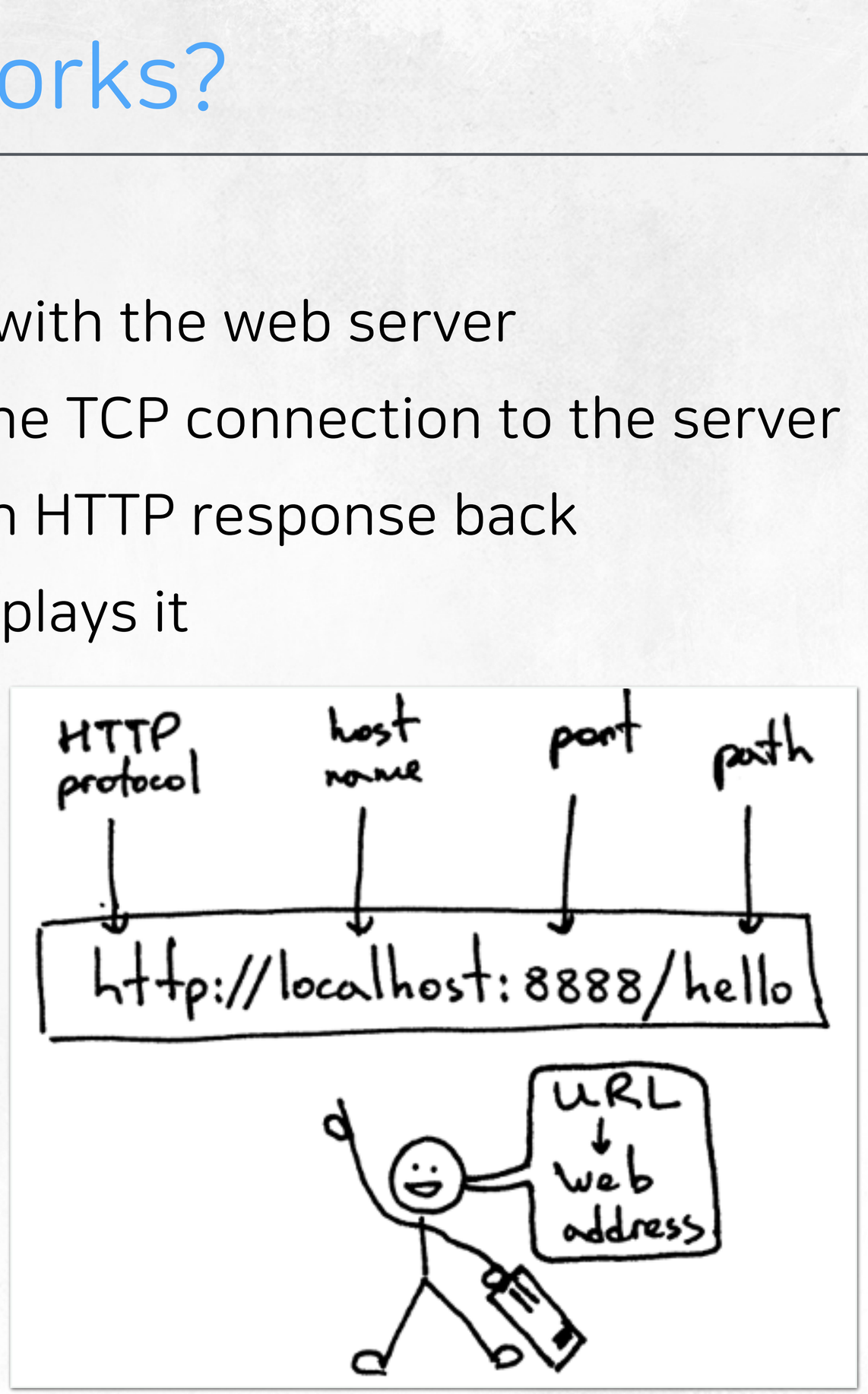
- A very simple implementation of a web server in Python




```
import socket

HOST, PORT = '', 9000

listen_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
listen_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
listen_socket.bind((HOST, PORT))
listen_socket.listen(1)
print ('Serving HTTP on port', PORT, '...')
while True:
    client_connection, client_address = listen_socket.accept()
    request = str(client_connection.recv(1024),'utf-8')
    print (request)

    http_response = """\
HTTP/1.1 200 OK

Hello world!
"""
    client_connection.sendall(bytes(http_response,'utf-8'))
    client_connection.close()
```

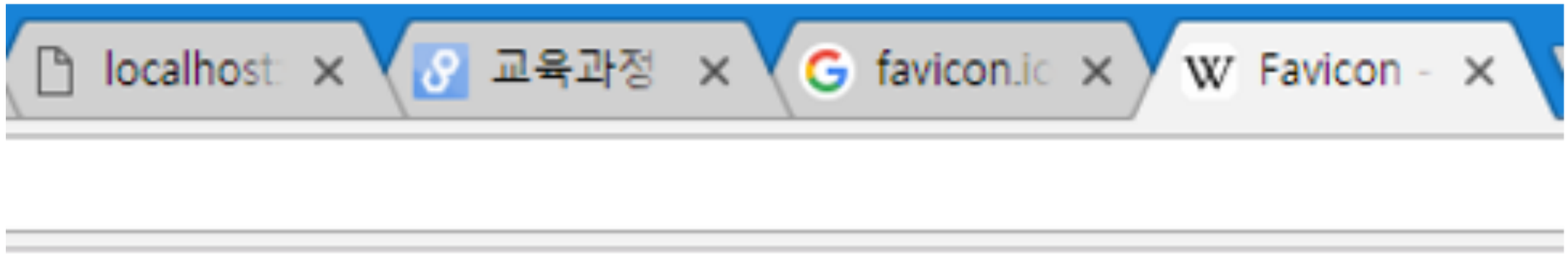
webserver1: How it works?

- The browser:
  - Establishes a TCP connection with the web server
  - Sends an HTTP request over the TCP connection to the server
  - Waits for the server to send an HTTP response back
  - Receives the response and displays it



webserver1: favicon

- A favicon (short for favorite icon)
  - a shortcut icon, website icon, tab icon, URL icon, bookmark icon
  - a file containing one or more small icons, associated with a particular website or webpage



webserver1: favicon

```
Serving HTTP on port 9000 ...
```

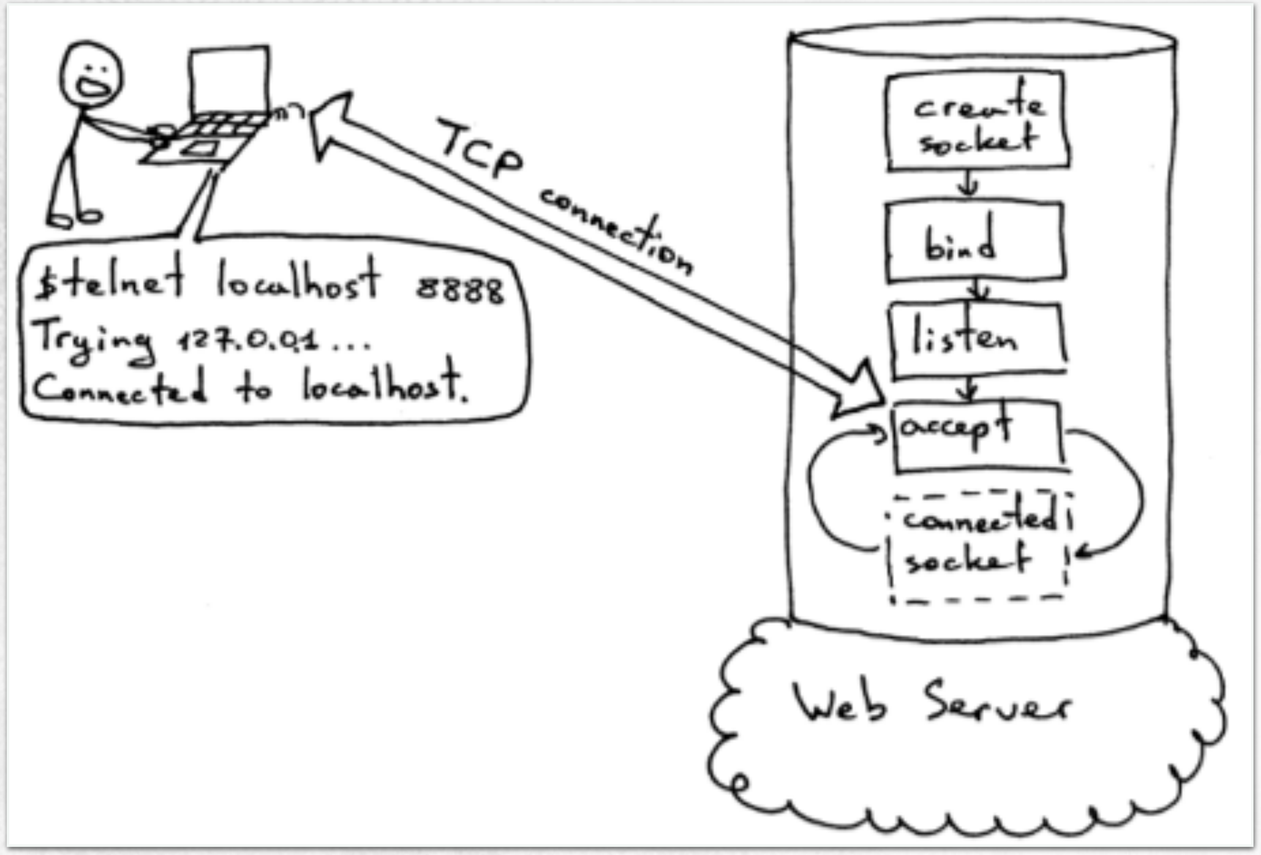
GET request comes in twice:
once for /hello and another for /favicon.ico

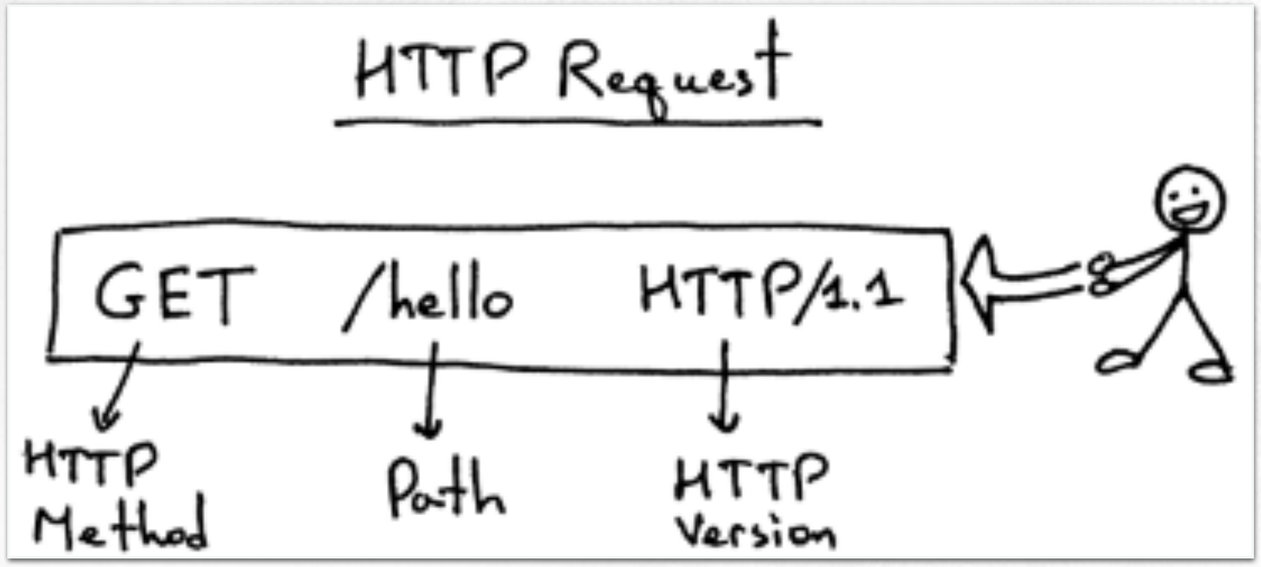
```
GET /hello HTTP/1.1
Host: localhost:9000
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/66.0.3359.181 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/a
png,*/*;q=0.8
Accept-Encoding: gzip, deflate, br
Accept-Language: ko-KR,ko;q=0.9,en-US;q=0.8,en;q=0.7


GET /favicon.ico HTTP/1.1
Host: localhost:9000
Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/66.0.3359.181 Safari/537.36
Accept: image/webp,image/apng,image/*,*/*;q=0.8
Referer: http://localhost:9000/hello
Accept-Encoding: gzip, deflate, br
Accept-Language: ko-KR,ko;q=0.9,en-US;q=0.8,en;q=0.7
```

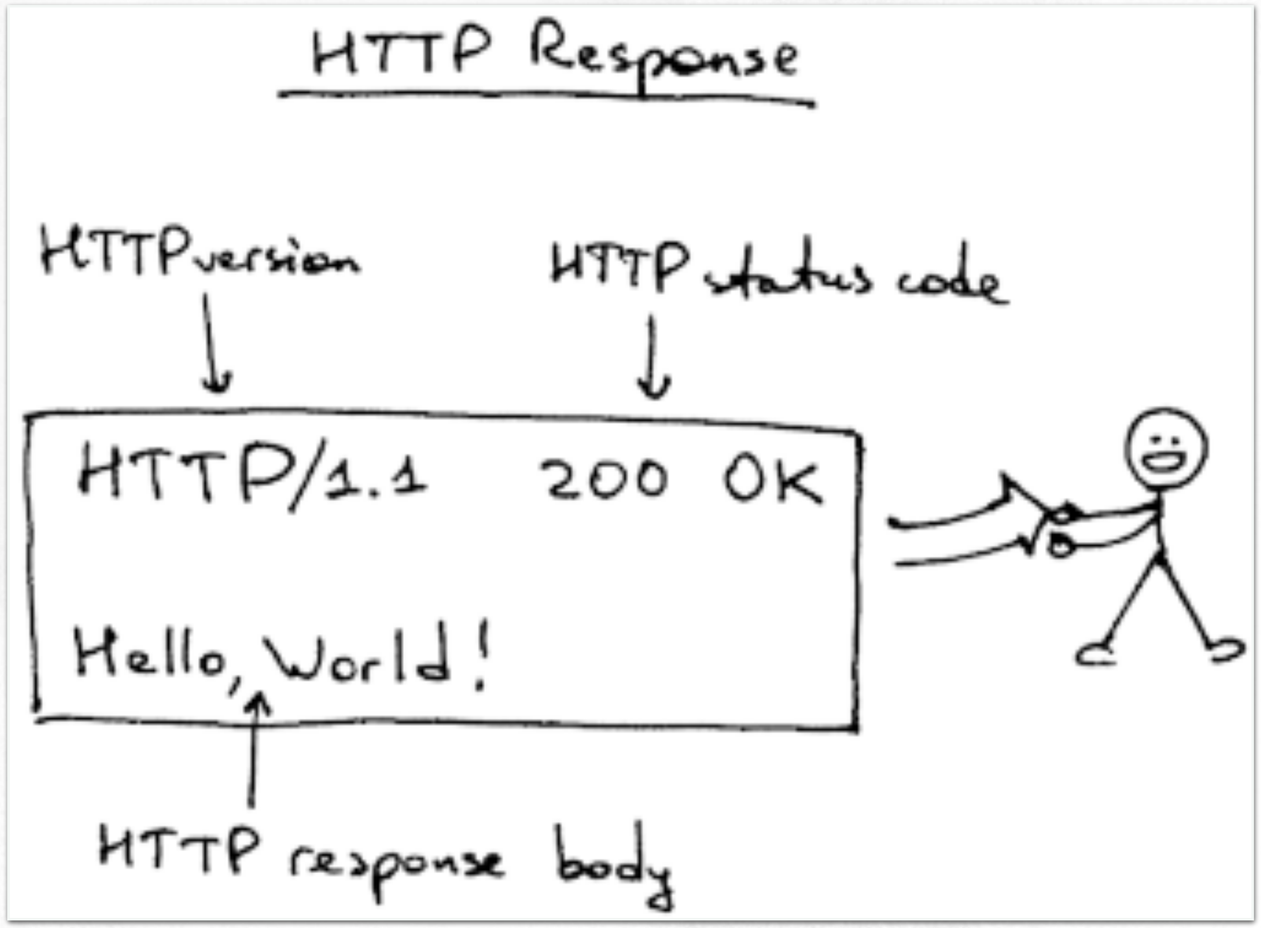
webserver1: TCP connection

- TCP is Transmission Control Protocol
- Client and the server establish a TCP connection under HTTP

webserver1: stage by stage







webserver1: code explanation

- Import socket module
- HOST = ‘’ # service apply to any host name
- PORT = 9000 # open port, 80 or 8080 are common to avoid conflict
- <variable> = socket.socket(<family>, <type>)

- Socket: programming interface to TCP and UDP
- Socket families:
  - AF_INET: IPv4 protocols
  - AF_INET6: IPv6 protocols
  - AF_UNIX: UNIX domain protocols

webserver1: code explanation

- Setting socket options
  - <socket_object>.setsockopt(level, option_name, value)

- Level: the categories of options.
  - SOL_SOCKET: socket-level options
  - IPPROTO_IP: protocol numbers
  - Available options are determined by your OS and IP version

- Binding the port to the socket
- Tell the computer to wait and to listen on that port

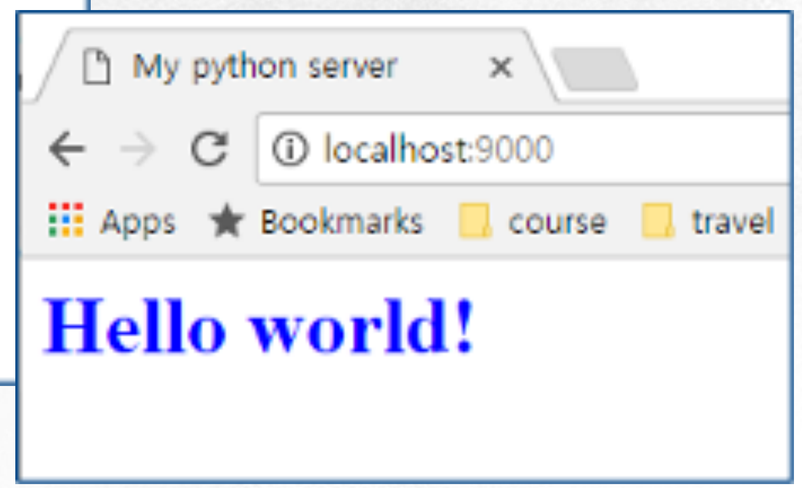
webserver1: handling a server request

- When request is made,
- The server accepts the request, and
- Sends data (a response)
  - Data
    - » First line: a status line (protocol, protocol version, message number, and status)
    - » Two new line characters (“₩n₩n”) to distinguish the protocol information from the page content
    - » Then the rest of the data
- Close the server socket

```
while True:
    client_connection, client_address = listen_socket.accept()
    request = str(client_connection.recv(1024),'utf-8')
    print (request)

    http_response = """\
HTTP/1.1 200 OK

Hello world!
"""
    client_connection.sendall(bytes(http_response,'utf-8'))
    client_connection.close()
```

webserver1: handling a server request

```
while True:
    client_connection, client_address = listen_socket.accept()
    request = str(client_connection.recv(1024),'utf-8')
    print (request)

    http_response = """\
HTTP/1.1 200 OK

Hello world!
"""
    client_connection.sendall(bytes(http_response,'utf-8'))
    client_connection.close()
```




```
    http_response = """\
HTTP/1.1 200 OK

<html><head><title>My python server</title></head><body><H1 style="color:blue">Hello world!</H1></body></html>
"""
```

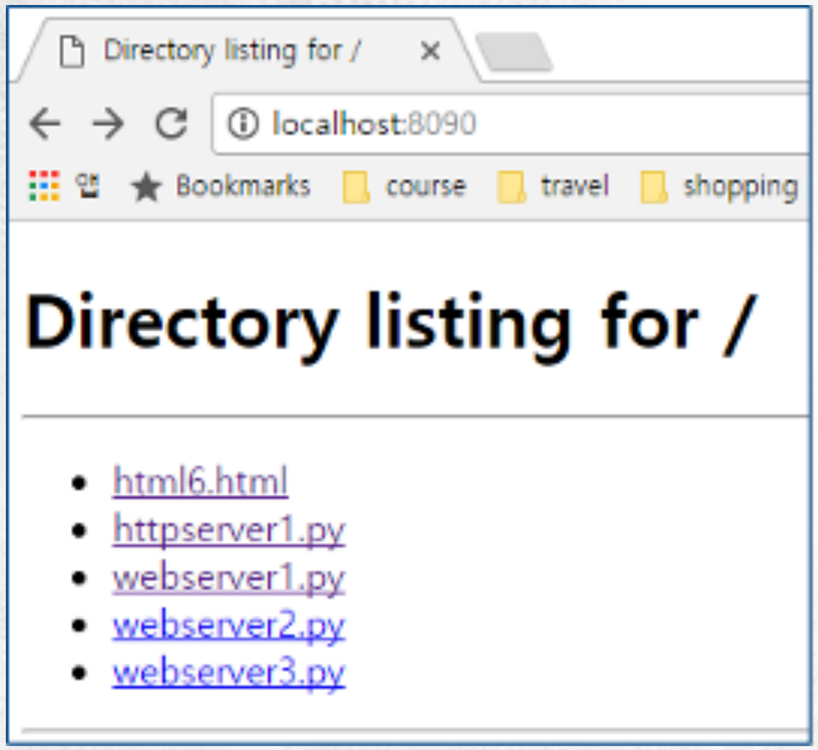
webserver1: send HTML files

```
while True:
    client_connection, client_address = listen_socket.accept()
    request = str(client_connection.recv(1024),'utf-8')
    print (request)

    http_response = """\
HTTP/1.1 200 OK

Hello world!
"""
    client_connection.sendall(bytes(http_response,'utf-8'))
    client_connection.close()
```

```
while True:
    client_connection, client_address = listen_socket.accept()
    request = str(client_connection.recv(1024),'utf-8')
    print (request)

    http_response = "HTTP/1.1 200 OK\n\n"
    file = open('html6.html','r+b')
    client_connection.sendall(bytes(http_response,'utf-8'))
    client_connection.sendfile(file)

    file.close()
    client_connection.close()
```

httpserver1: simple HTTP server

- A very simple implementation of a HTTP server in Python



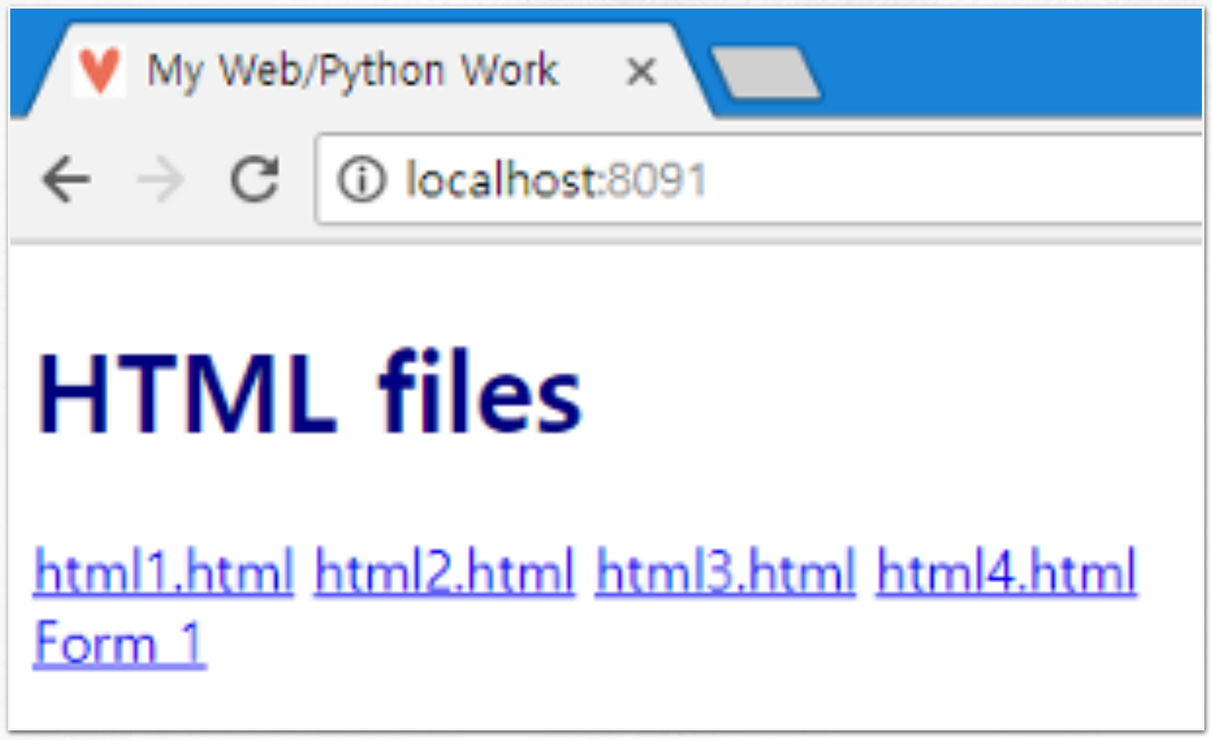

```
from http.server import SimpleHTTPRequestHandler, HTTPServer

port = 8090
server_address = ('', port)
httpd = HTTPServer(server_address, SimpleHTTPRequestHandler)
print("Starting simple_httpd on port: " + str(httpd.server_port))
httpd.serve_forever()
```

httpserver2: make an index page

- index.html
- Link to other pages
- Save an favicon.ico  to the same folder

index.html

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title>My Web/Python Work</title>
5  </head>
6
7  <body>
8
9  <H1 style="color:navy;">HTML files</H1>
10
11 <a href="html1.html">html1.html</a>
12 <a href="html2.html">html2.html</a>
13 <a href="html3.html">html3.html</a>
14 <a href="html4.html">html4.html</a>
15
16 </body>
17
18 </html>
19
```




```
Starting simple_httpd on port: 8091
127.0.0.1 - - [25/May/2018 14:04:16] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [25/May/2018 14:04:16] "GET /favicon.ico HTTP/1.1" 200 -
127.0.0.1 - - [25/May/2018 14:10:34] "GET /html1.html HTTP/1.1" 200 -
127.0.0.1 - - [25/May/2018 14:10:36] "GET /html2.html HTTP/1.1" 200 -
127.0.0.1 - - [25/May/2018 14:10:39] "GET /html3.html HTTP/1.1" 200 -
127.0.0.1 - - [25/May/2018 14:10:39] "GET /google.png HTTP/1.1" 200 -
127.0.0.1 - - [25/May/2018 14:10:43] "GET /html4.html HTTP/1.1" 200 -
```

httpserver2: explanation using inheritance

- A very simple implementation of a HTTP server in Python

```
from http.server import HTTPServer, SimpleHTTPRequestHandler

class testHTTPServer_RequestHandler(SimpleHTTPRequestHandler):
    def do_GET(self):
        super().do_GET()
        print("do_get")

port = 8095
httpd = HTTPServer(('', port), testHTTPServer_RequestHandler)
print("Starting simple_httpd on port: " + str(httpd.server_port))
httpd.serve_forever()
```

httpserver2: explanation using inheritance

- Standard HTTP handler in Python

servertest01.py × server.py ×

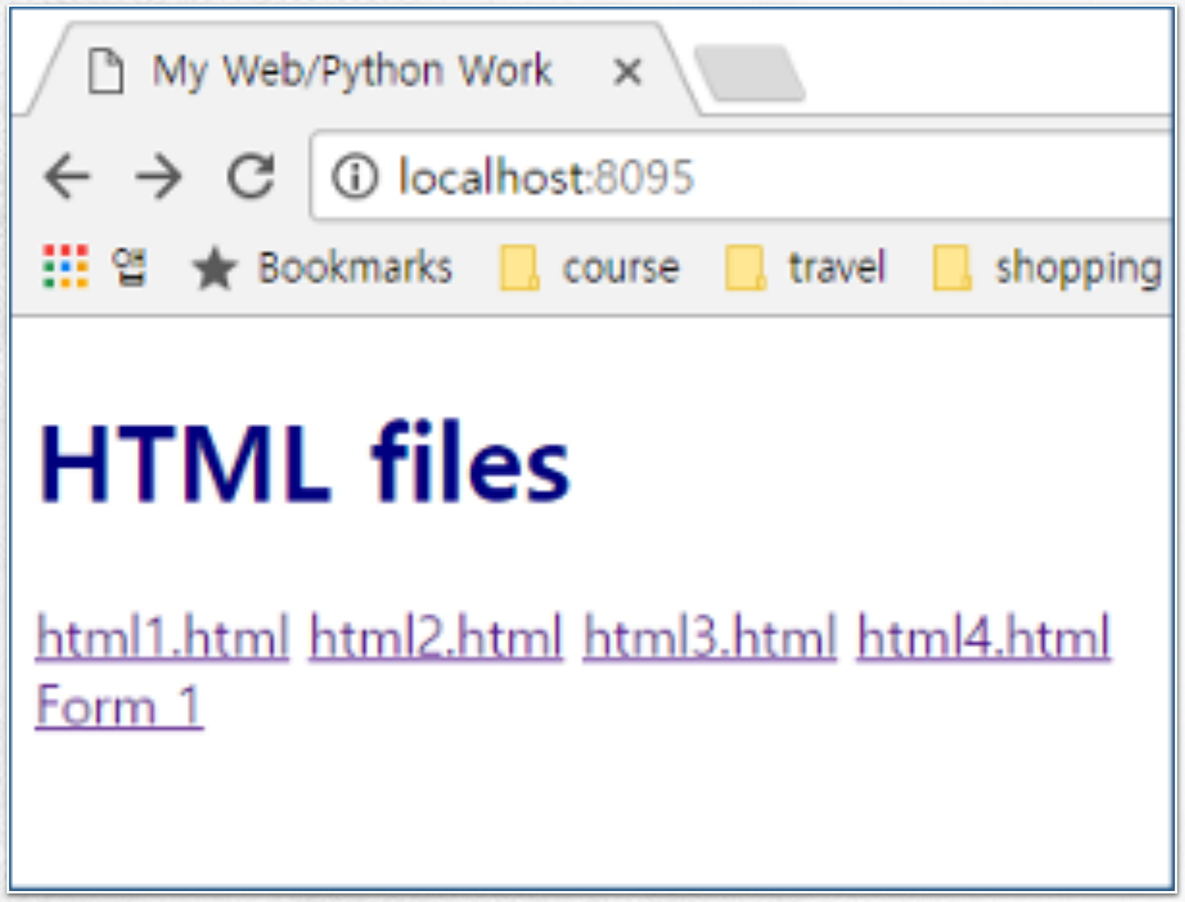
```
623             v: (v.phrase, v.description)
624             for v in HTTPStatus.__members__.values()
625         }
626
627
628     class SimpleHTTPRequestHandler(BaseHTTPRequestHandler):
629
630         """Simple HTTP request handler with GET and HEAD commands.
631
632         This serves files from the current directory and any of its
633         subdirectories.  The MIME type for files is determined by
634         calling the .guess_type() method.
635
636         The GET and HEAD requests are identical except that the HEAD
637         request omits the actual contents of the file.
638
639         """
640
641         server_version = "SimpleHTTP/" + __version__
642
643         def do_GET(self):
644             """Serve a GET request."""
645             f = self.send_head()
646             if f:
647                 try:
648                     self.copyfile(f, self.wfile)
649                 finally:
650                     f.close()
651
652         def do_HEAD(self):
653             """Serve a HEAD request."""
654             f = self.send_head()
655             if f:
656                 f.close()
657
```

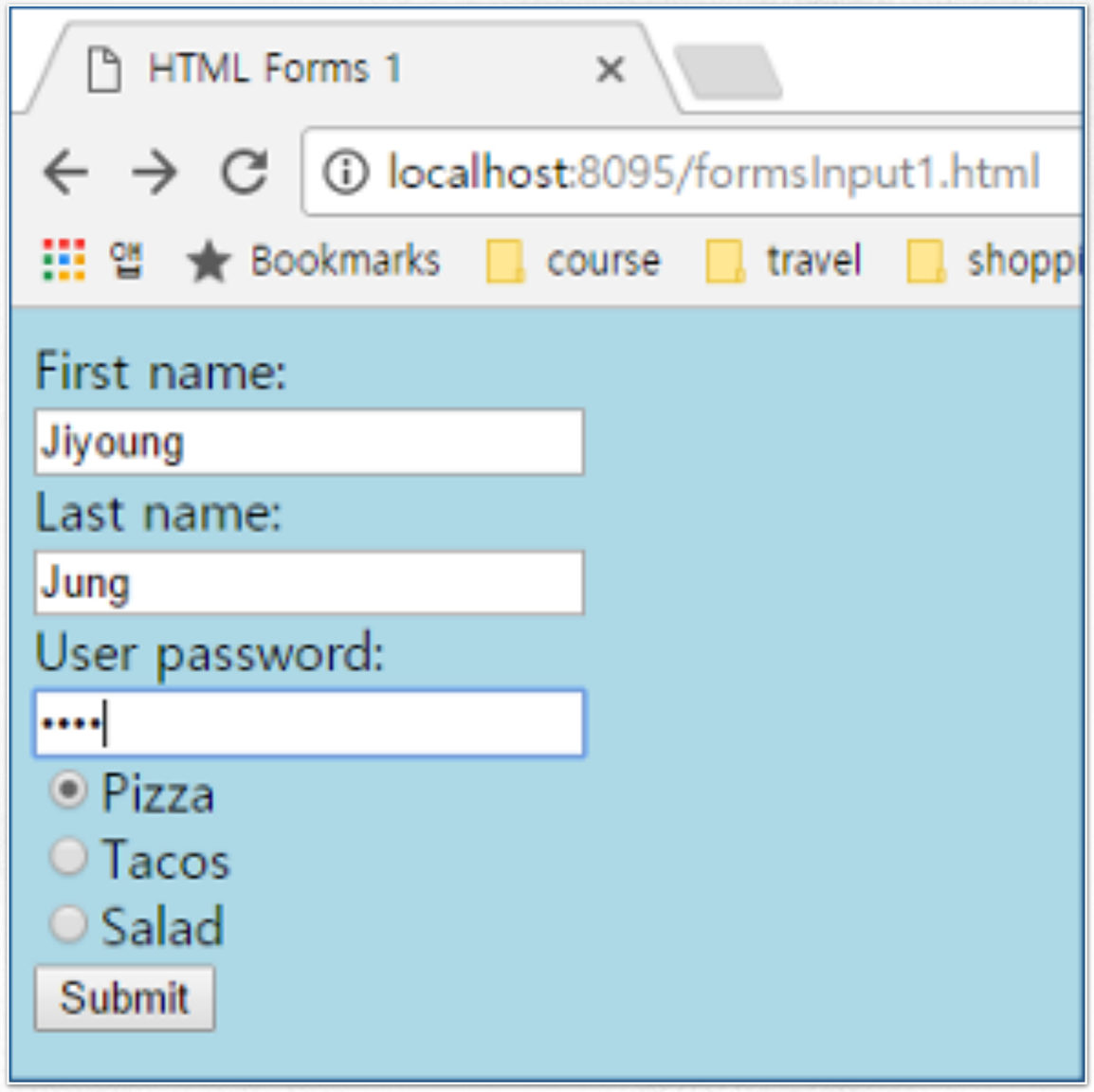
servertest01.py × server.py ×

```
656                 f.close()
657
658         def send_head(self):
659             """Common code for GET and HEAD commands.
660
661             This sends the response code and MIME headers.
662
663             Return value is either a file object (which has to be copied
664             to the outputfile by the caller unless the command was HEAD,
665             and must be closed by the caller under all circumstances), or
666             None, in which case the caller has nothing further to do.
667
668             """
669             path = self.translate_path(self.path)
670             f = None
671             if os.path.isdir(path):
672                 parts = urllib.parse.urlsplit(self.path)
673                 if not parts.path.endswith('/'):
674                     # redirect browser - doing basically what apache does
675                     self.send_response(HTTPStatus.MOVED_PERMANENTLY)
676                     new_parts = (parts[0], parts[1], parts[2] + '/',
677                                  parts[3], parts[4])
678                     new_url = urllib.parse.urlunsplit(new_parts)
679                     self.send_header("Location", new_url)
680                     self.end_headers()
681                     return None
682                 for index in "index.html", "index.htm":
683                     index = os.path.join(path, index)
684                     if os.path.exists(index):
685                         path = index
686                         break
687                 else:
688                     return self.list_directory(path)
689             ctype = self.guess_type(path)
```

httpserver3: HTML form exercise

- Edit index.html to link a form page (formsInput1.html)
- Check how the server receives user input values



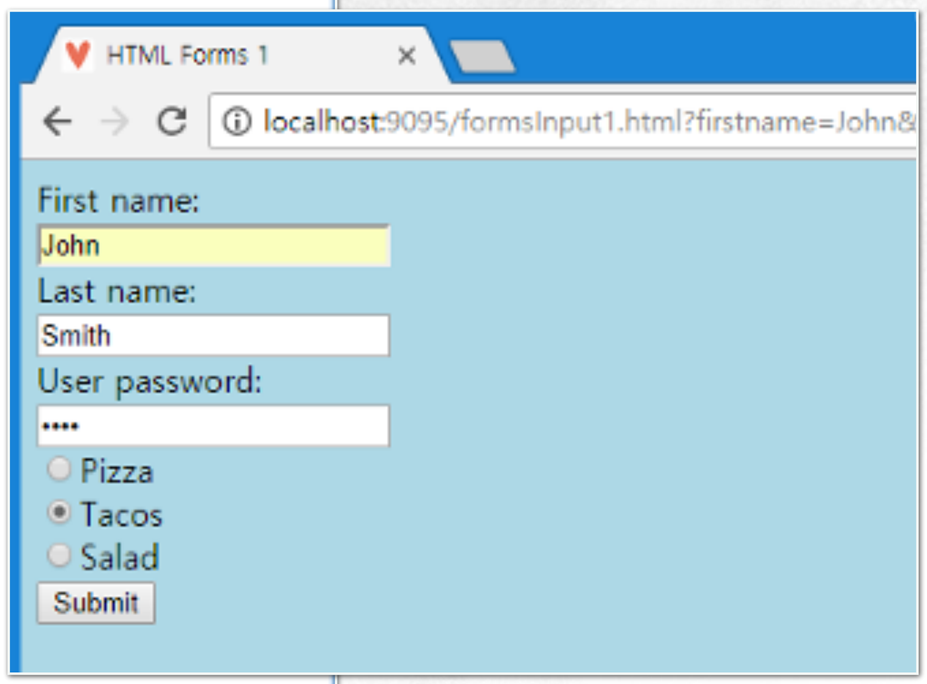


httpserver3: fetch and precess user input

httpserver3.py - D:₩course₩2017s_python₩myserver₩httpserver3.py (3.5.3)

File Edit Format Run Options Window Help

```
from http.server import HTTPServer, SimpleHTTPRequestHandler
from urllib.parse import parse_qs, urlparse

class testHTTPServer_RequestHandler(SimpleHTTPRequestHandler):

    def do_GET(self):
        url = self.path
        form = parse_qs(urlparse(url).query)
        if (form != {}):
            self.process_form(form)

        super().do_GET()
        print("do_get")

    def process_form(self,form):
        if 'food' in form:
            if form['food'][0] == 'Pizza':
                print(form['firstname'][0] + ", call Dominos tonight!")
            elif form['food'][0] == 'Tacos':
                print(form['firstname'][0] + ", go to TacoBell tonight!")
            elif form['food'][0] == 'Salad':
                print(form['firstname'][0] + ", have a Caesar Salad tonight!")


port = 9095
httpd = HTTPServer(('', port), testHTTPServer_RequestHandler)
print("Starting simple_httpd on port: " + str(httpd.server_port))
httpd.serve_forever()
```




```
Starting simple_httpd on port: 9095
John, go to TacoBell tonight!
127.0.0.1 - - [25/May/2018 14:36:27] "GET /
formsInput1.html?firstname=John&lastname=Sm
ith&password=dddd&food=Tacos HTTP/1.1" 200
-
do_get
```

Frameworks (3rd party softwares)

Welcome to Flask's documentation. Get started with Installation and then get an overview with the Quickstart. There is also a more detailed Tutorial that shows how to create a small but complete application with Flask. Common patterns are described in the Patterns for Flask section. The rest of the docs describe each component of Flask in detail, with a full reference in the API section.

Flask depends on the Jinja template engine and the Werkzeug WSGI toolkit. The documentation for these libraries can be found at:

```python
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello_world():
    return "<p>Hello, World!</p>"
```

Frameworks (3rd party softwares)

Latest version: 20.1.0

gunicorn

gunicorn

Gunicorn 'Green Unicorn' is a Python WSGI HTTP Server for UNIX. It's a pre-fork worker model. The Gunicorn server is broadly compatible with various web frameworks, simply implemented, light on server resources, and fairly speedy.

View source | Download

QUICKSTART

Read the quickstart guide to get started using Gunicorn.

DEPLOYMENT

Learn how to deploy the Gunicorn server.

COMMUNITY

Get in touch with the community.

DOCUMENTATION

Read the documentation to learn more about Gunicorn.

Frameworks (3rd party softwares)

Latest version: 20.1.0

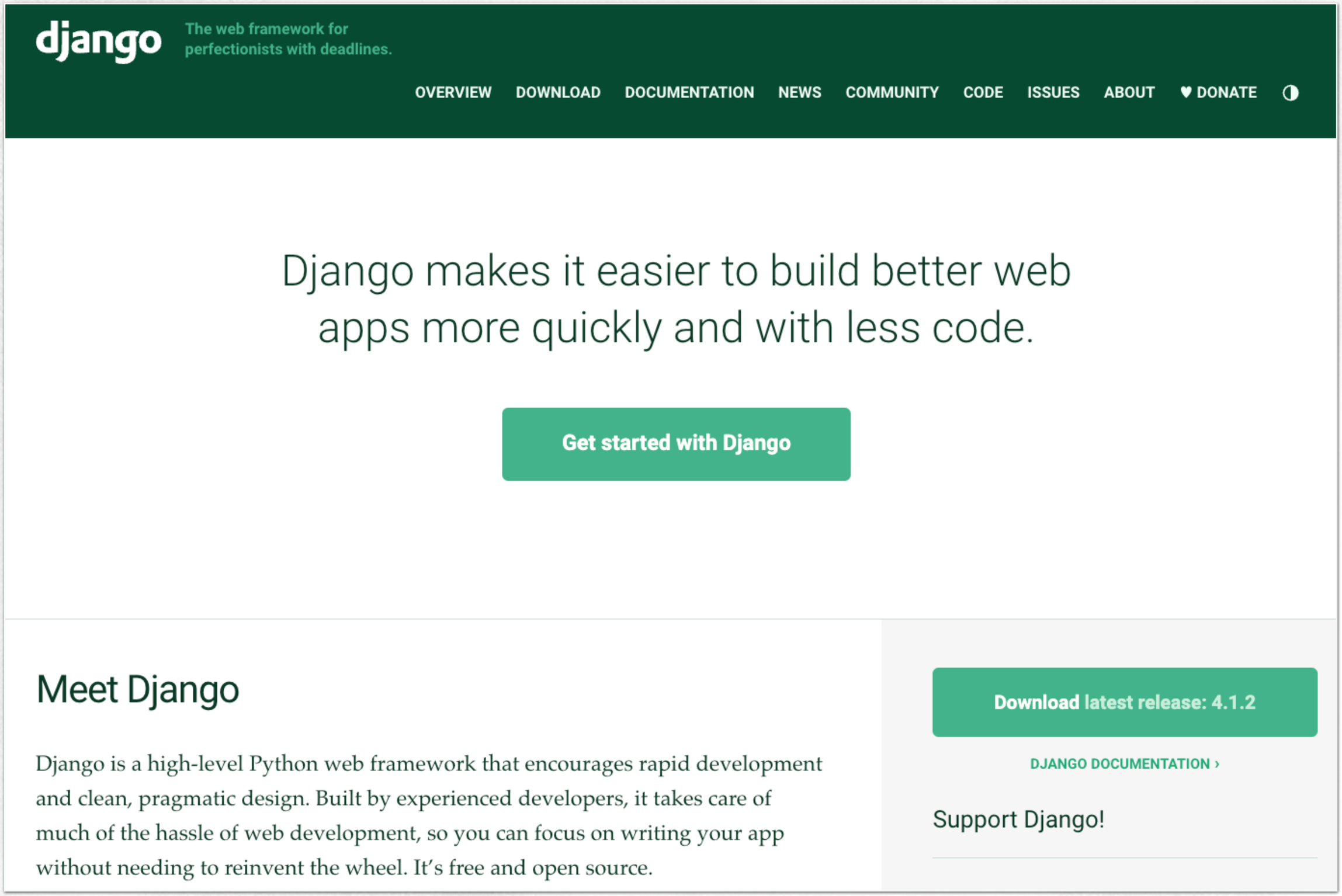
gunicorn

gunicorn

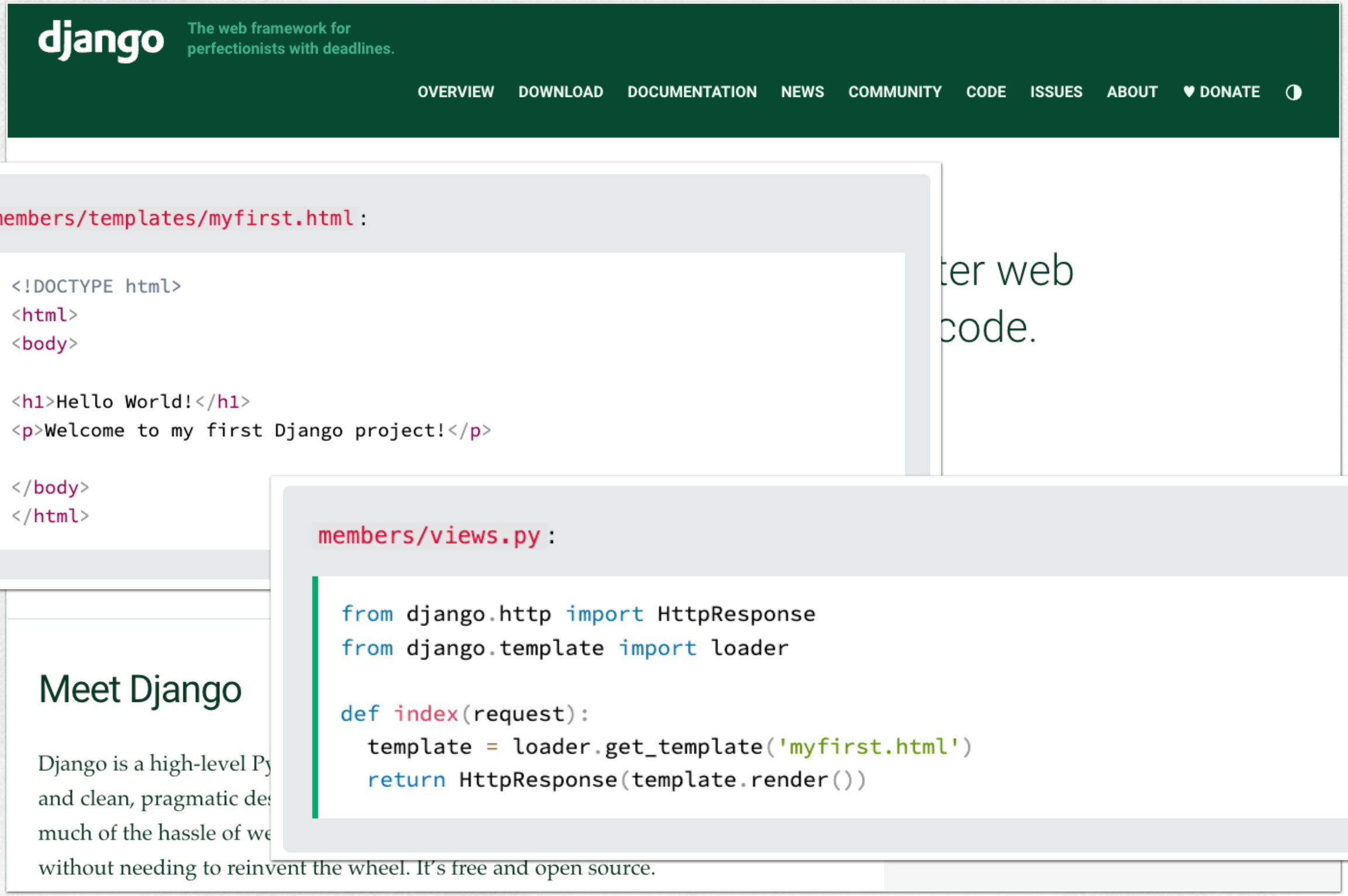
Gunicorn 'Green Unicorn' is a Python WSGI HTTP Server for UNIX. It's a pre-fork worker model. The Gunicorn server is broadly compatible with various web frameworks, simply implemented, light on server resources, and fairly speedy.

```
$ pip install gunicorn
$ cat myapp.py
  def app(environ, start_response):
      data = b"Hello, World!\n"
      start_response("200 OK", [
          ("Content-Type", "text/plain"),
          ("Content-Length", str(len(data)))
      ])
      return iter([data])
$ gunicorn -w 4 myapp:app
[2014-09-10 10:22:28 +0000] [30869] [INFO] Listening at: http://127.0.0.1:8000 (30869)
[2014-09-10 10:22:28 +0000] [30869] [INFO] Using worker: sync
[2014-09-10 10:22:28 +0000] [30874] [INFO] Booting worker with pid: 30874
[2014-09-10 10:22:28 +0000] [30875] [INFO] Booting worker with pid: 30875
[2014-09-10 10:22:28 +0000] [30876] [INFO] Booting worker with pid: 30876
[2014-09-10 10:22:28 +0000] [30877] [INFO] Booting worker with pid: 30877
```

Frameworks (3rd party softwares)



Frameworks (3rd party softwares)



members/templates/myfirst.html :

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

<h1>Hello World!</h1>
<p>Welcome to my first Django project!</p>

</body>
</html>
```

members/views.py :

```
from django.http import HttpResponse
from django.template import loader

def index(request):
  template = loader.get_template('myfirst.html')
  return HttpResponse(template.render())
```

Thank you

경희대학교
KYUNG HEE UNIVERSITY