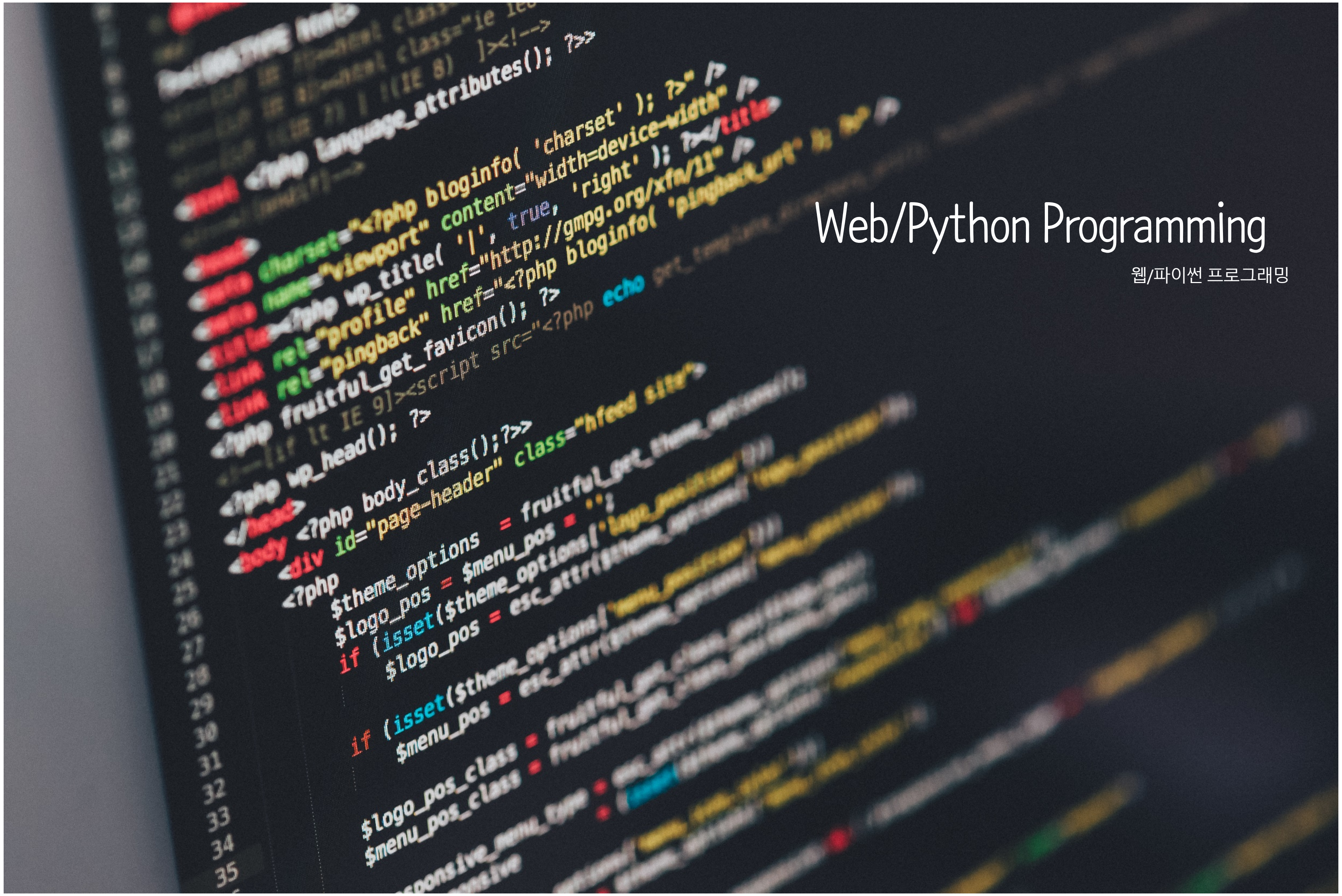


Web/Python Programming

웹/파이썬 프로그래밍

Today

- Let’s Build A Web Server
  - Python revisited

Building a web server

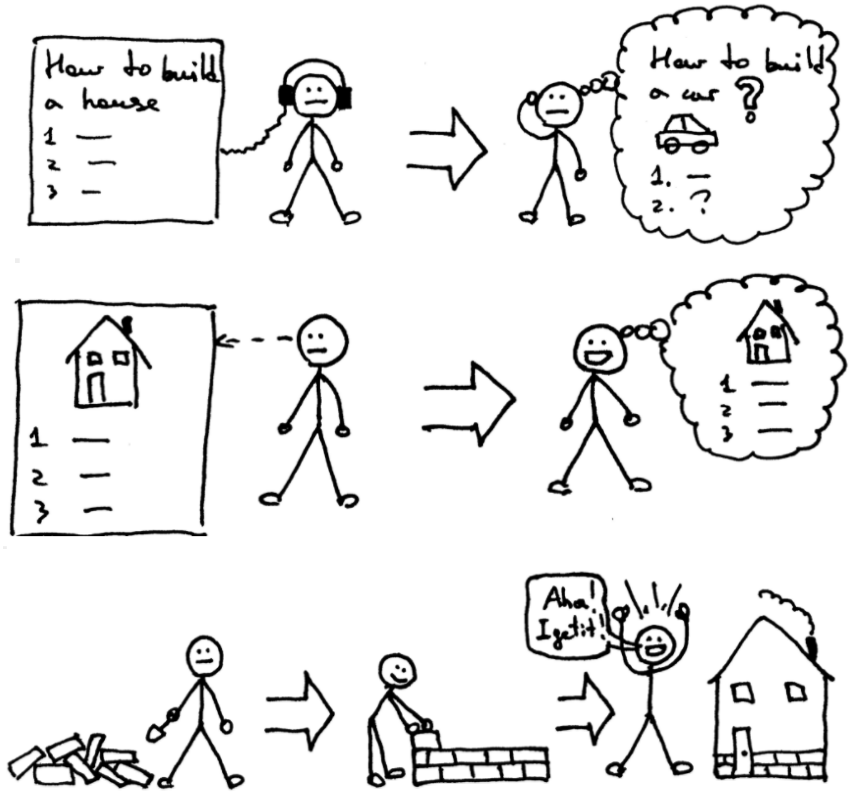
- Why?
  - In fact, web hosting companies provide such services…

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

- Confucius

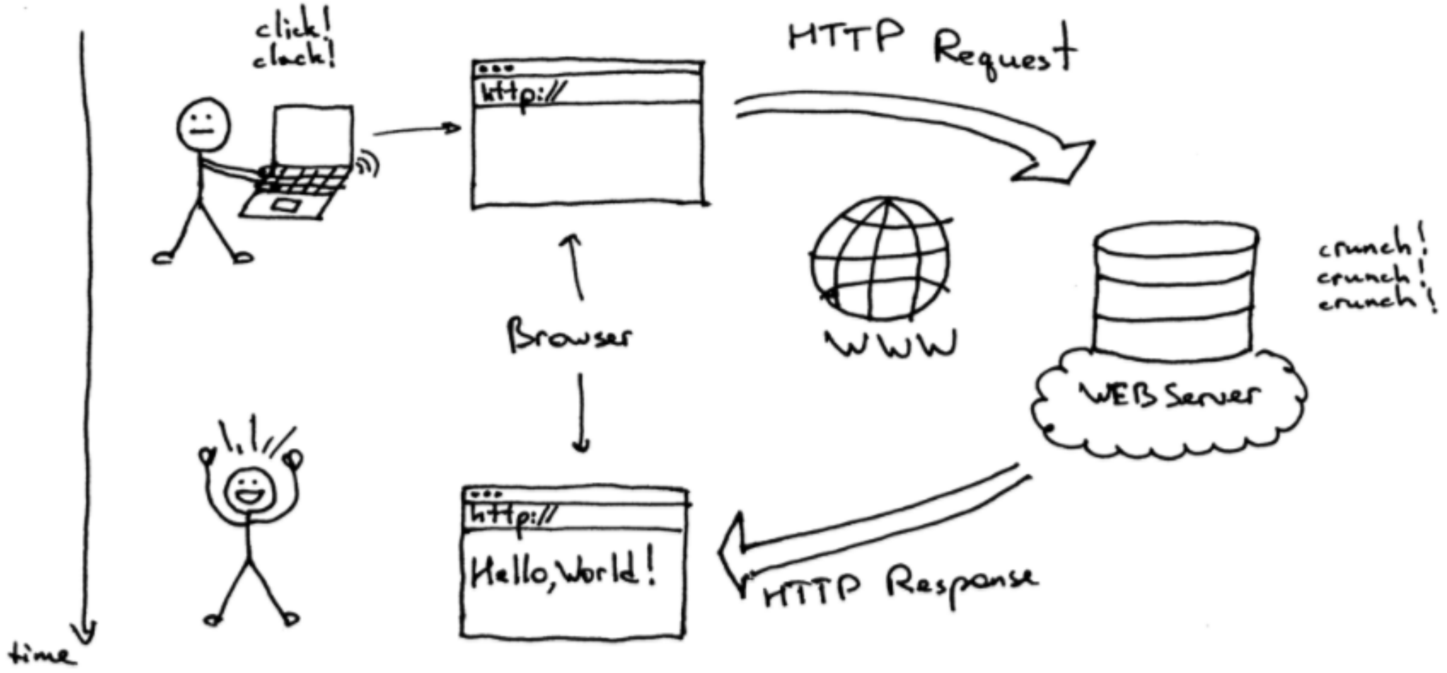
听而易忘
见而易记
做而易懂

– 孔子



What is a web server?

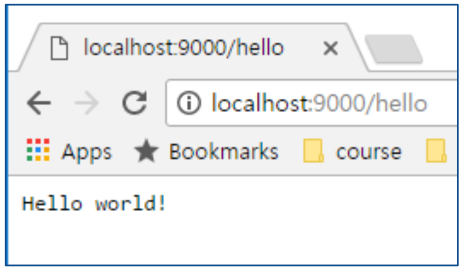
- Waits for a client to send a request
- When it receives a request, it generates a response and sends it back to the client



webserver1.py

Implementation

- A very simple implementation of a web server in Python
  - webserver1.py

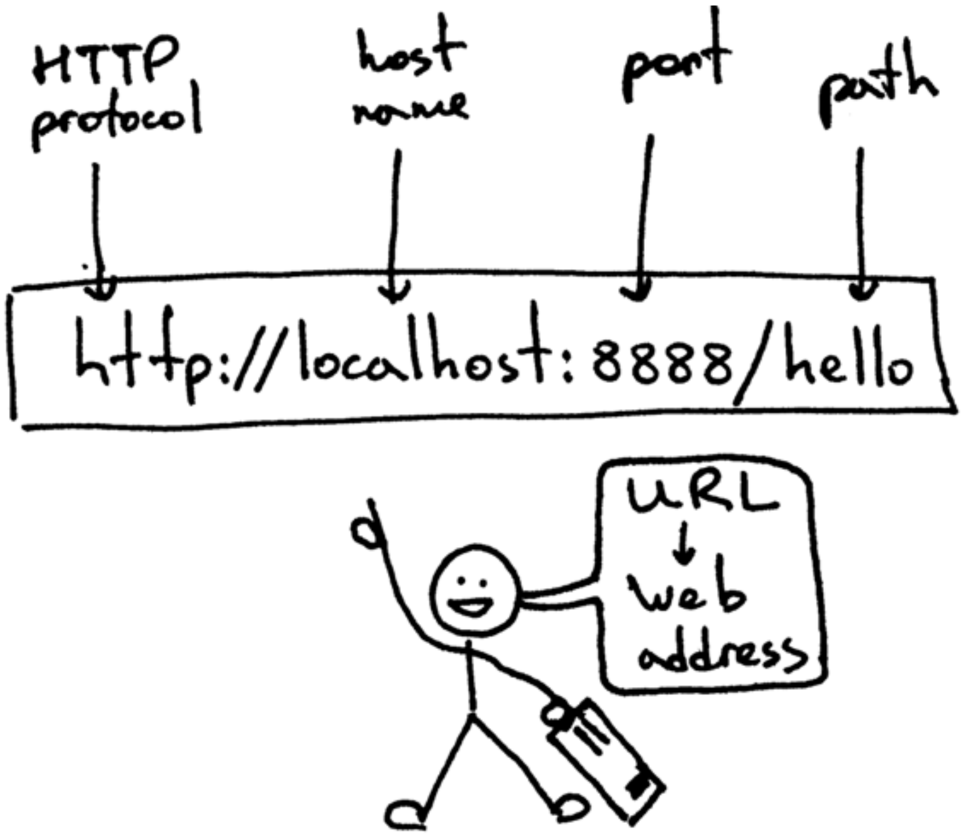


```
Hello world!
```

```
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.py

How it works?

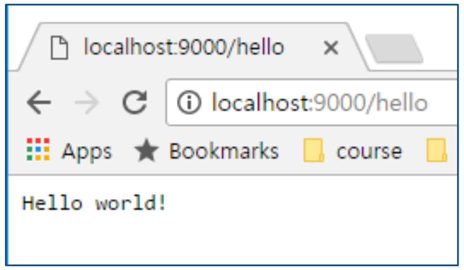


- The browser …
  - Establishes a TCP connection with the web server
  - Sends an HTTP request over the TCP connection to the server
  - Waits for the server to send an HTTP response back.
  - Receives the response and displays it.

TCP connection

webserver1.py

- Transmission Control Protocol
- How the client and the server establish a TCP connection?
- They use sockets


localhost:9000/hello

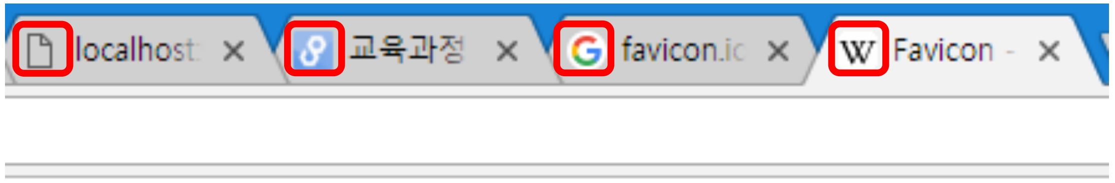
```
Hello world!
```

```
Serving HTTP on port 9000 ...

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
```

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

favicon

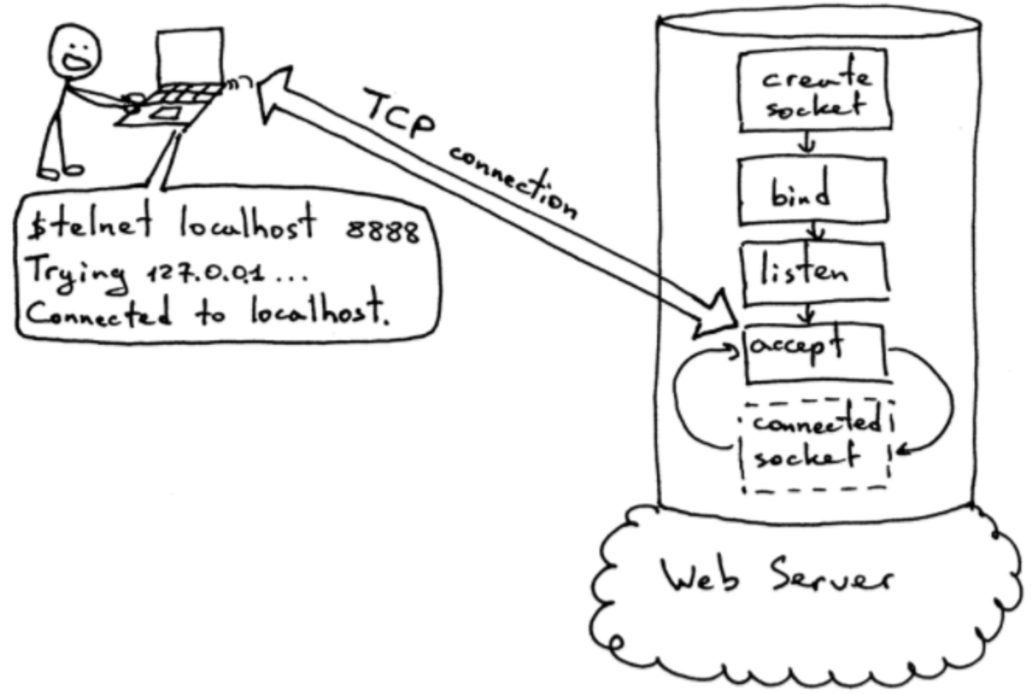
webserver1.py



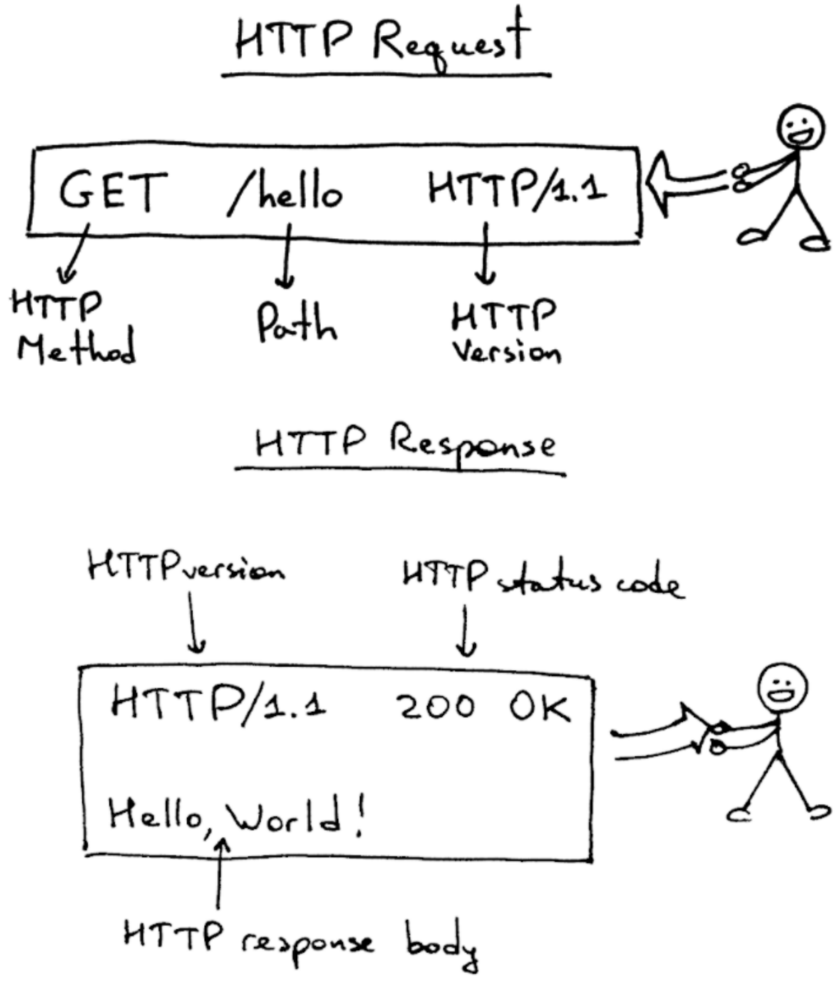
- A favicon (short for favorite icon)
  - aka. a shortcut icon, website icon, tab icon, URL icon, bookmark icon
  - A file containing one or more small icons, associated with a particular website or webpage.
- How to set favicon? Explained later

Web server – stage by stage

webserver1.py



```
$telnet localhost 8888
Trying 127.0.0.1 ...
Connected to localhost.
```

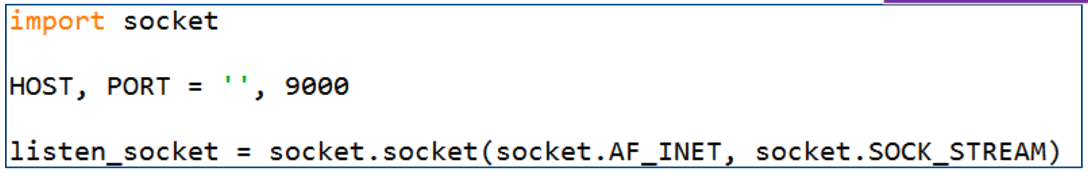


```
GET /hello HTTP/1.1
```

```
HTTP/1.1 200 OK

Hello, World!
```

Code Explanation

webserver1.py

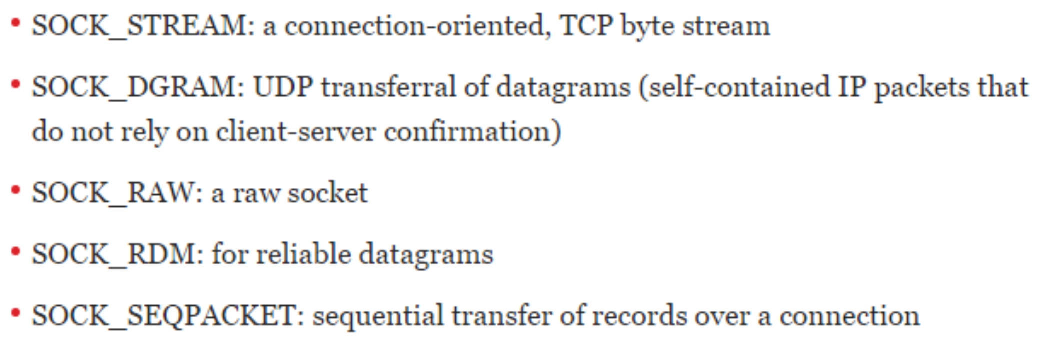


```
import socket

HOST, PORT = '', 9000

listen_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

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

- Socket families:
  - AF_INET: IPv4 protocols
  - AF_INET6: IPv6 protocols
  - AF_UNIX: UNIX domain protocols

- Socket types:



- SOCK_STREAM: a connection-oriented, TCP byte stream
- SOCK_DGRAM: UDP transferral of datagrams (self-contained IP packets that do not rely on client-server confirmation)
- SOCK_RAW: a raw socket
- SOCK_RDM: for reliable datagrams
- SOCK_SEQPACKET: sequential transfer of records over a connection

Code Explanation

webserver1.py

```
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,'...')
```

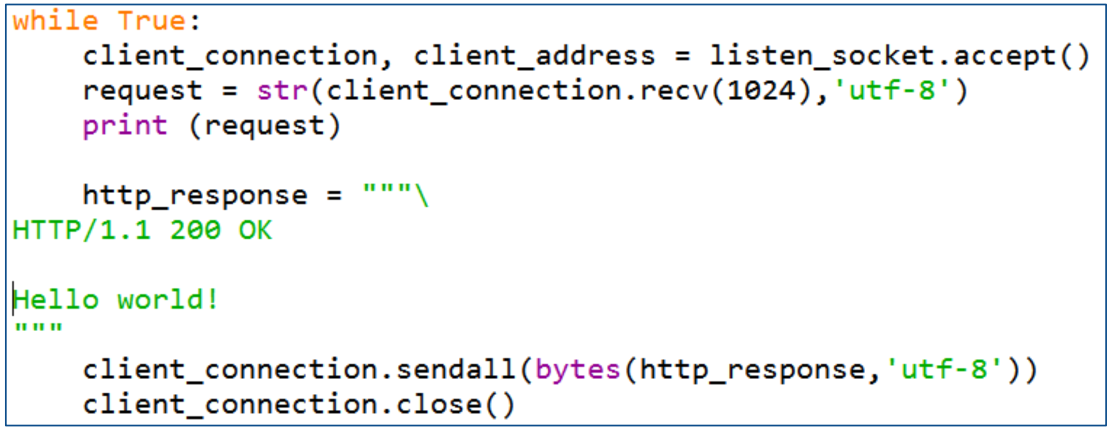
- Setting socket options
- <socket_object>.setsockopt(level, option_name, value)

- Level: the categories of options.
  - SOL_SOCKET: socket-level options
  - IPPROTO_IP: protocol numbers
  - Available options are determined by your OS and whether using IPv4/IPv6

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

Handling a server request

webserver1.py

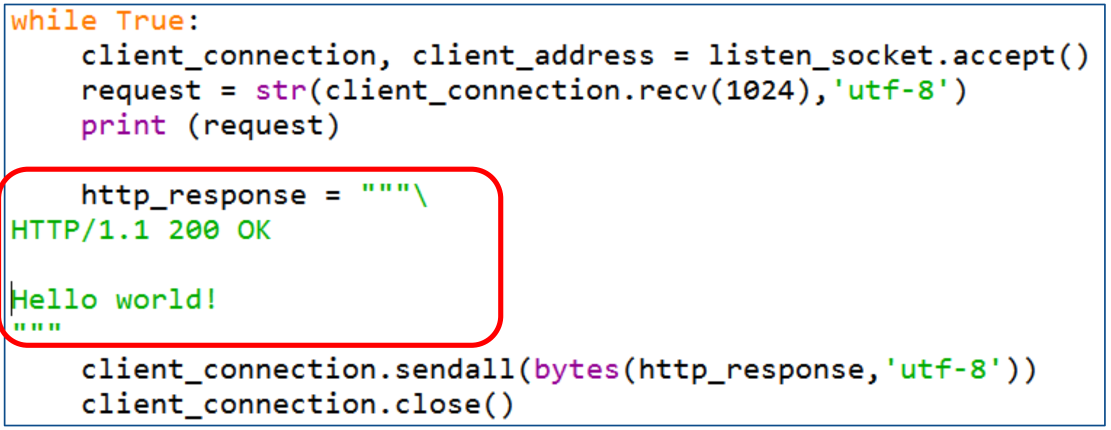


```
while True:
    client_connection, client_address = listen_socket.accept()
    request = str(client_connection.recv(1024),'utf-8')
    print (request)

    http_response = """\
HTTP/1.1 200 OK

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

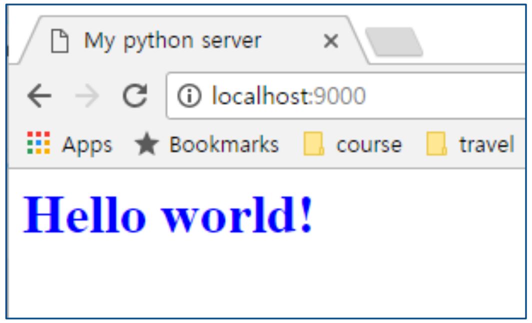
- When request is made,
- The server accepts the request,
- Sends data (a response)
- Data
  - First line: a status line (protocol, protocol version, message number, and status)
  - Two new line characters (“\n\n”) to distinguish the protocol information from the page content
  - Then the rest of the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

Send HTML tags

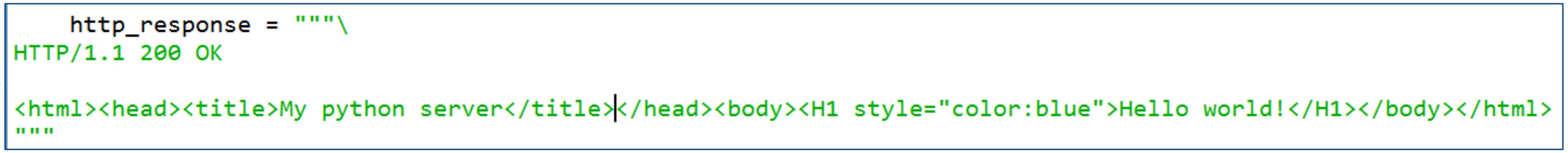
webserver1.py



```
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()
```



Hello world!



```
    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.py

Send HTML files

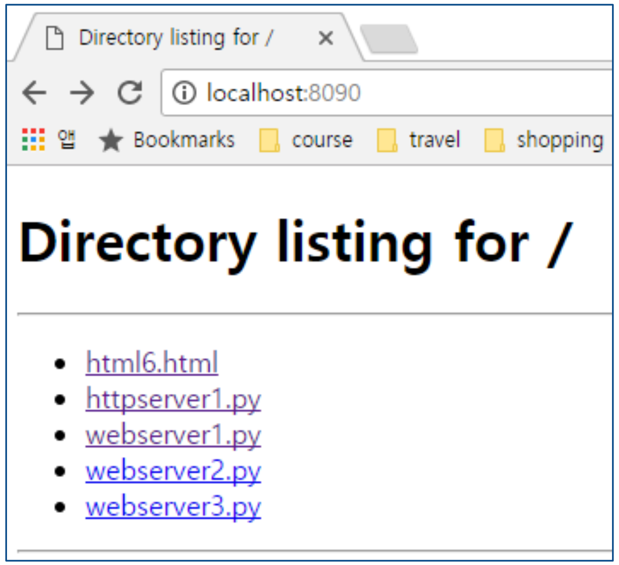
```
while True:
    client_connection, client_address = listen_socket.accept()
    request = str(client_connection.recv(1024),'utf-8')
    print (request)

    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()
```

A Simple HTTP server

httpserver1.py

- A very simple implementation of a HTTP server in Python
  - httpserver1.py


Directory listing for /

localhost:8090

Bookmarks course travel shopping

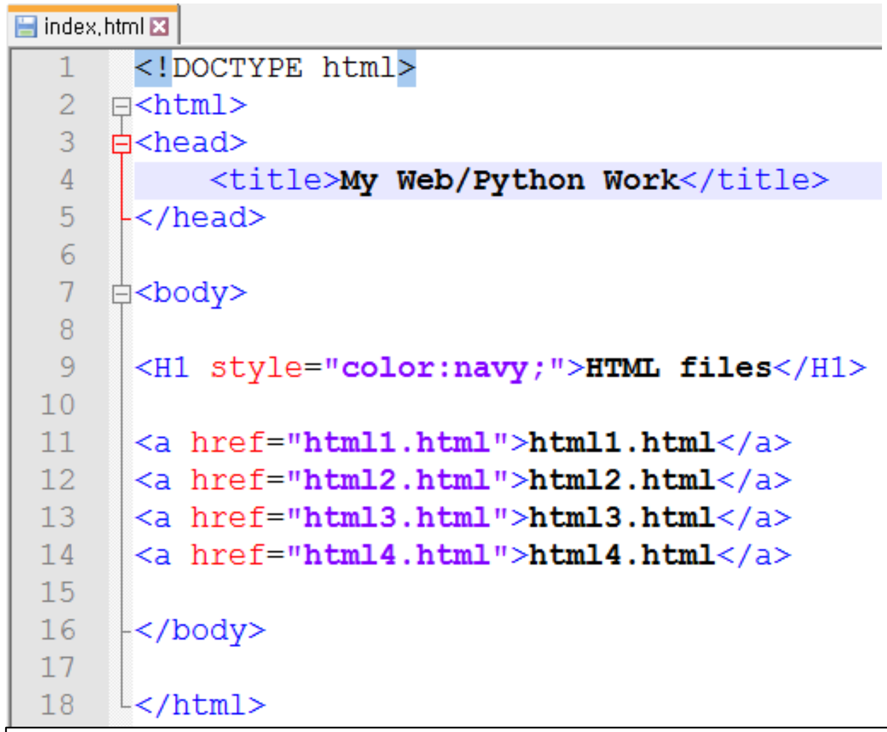
Directory listing for /

- html6.html
- httpserver1.py
- webserver1.py
- webserver2.py
- webserver3.py

```
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()
```

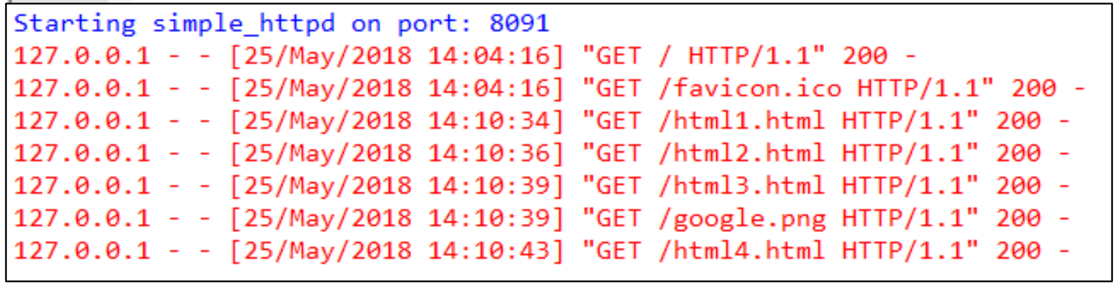
httpserver2.py

Make an Index Page



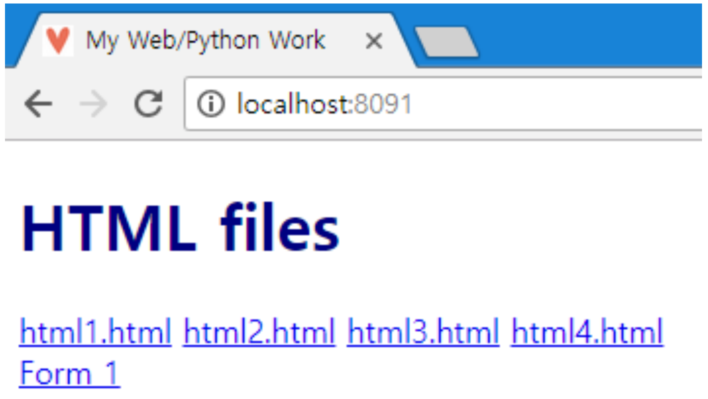
index.html

```
<!DOCTYPE html>
<html>
<head>
    <title>My Web/Python Work</title>
</head>

<body>

<H1 style="color:navy;">HTML files</H1>

<a href="html1.html">html1.html</a>
<a href="html2.html">html2.html</a>
<a href="html3.html">html3.html</a>
<a href="html4.html">html4.html</a>

</body>

</html>
```



```
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 -
```

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



HTML files

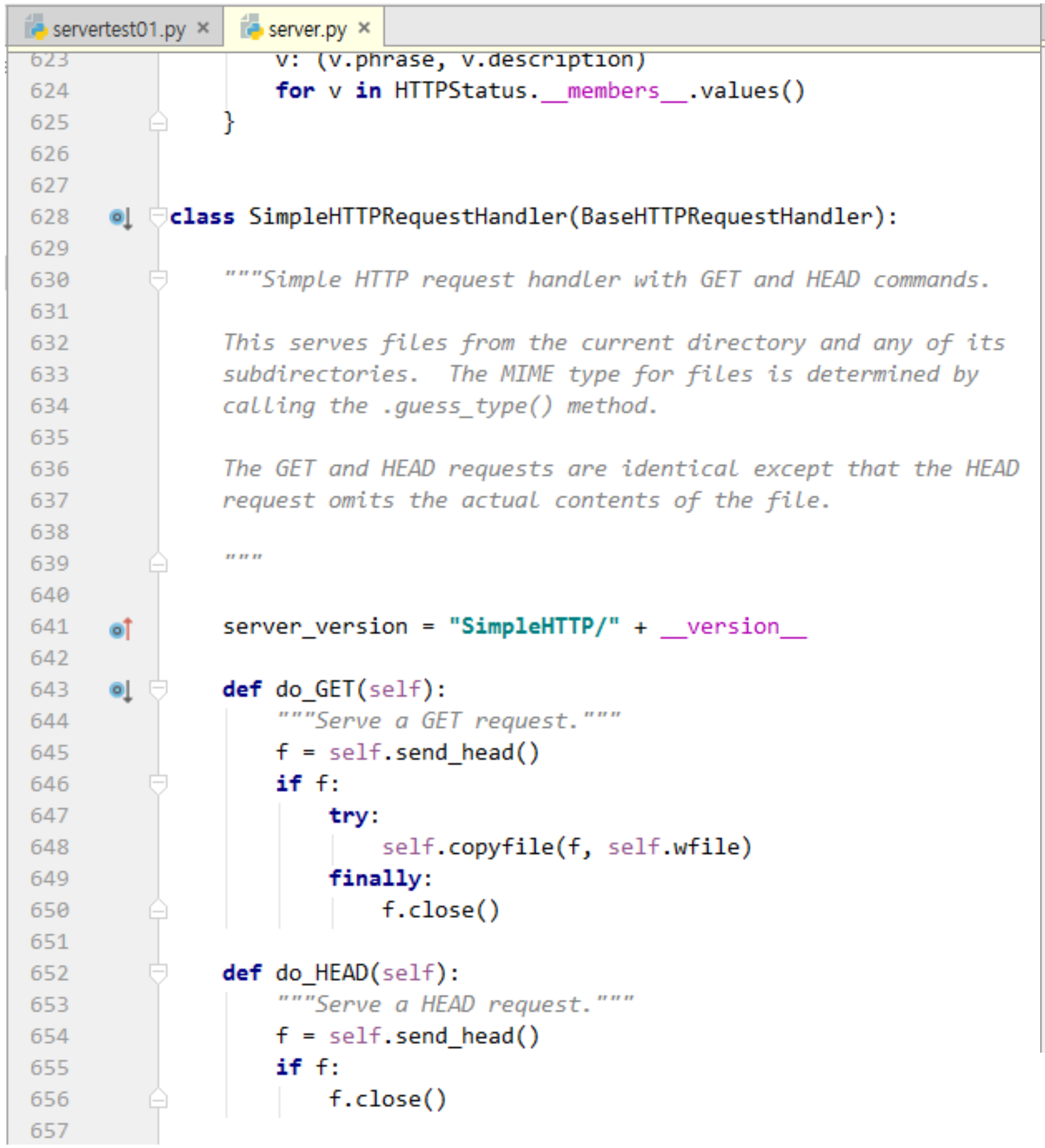
html1.html html2.html html3.html html4.html
Form 1

httpserver2.py

Explanation using Inheritance

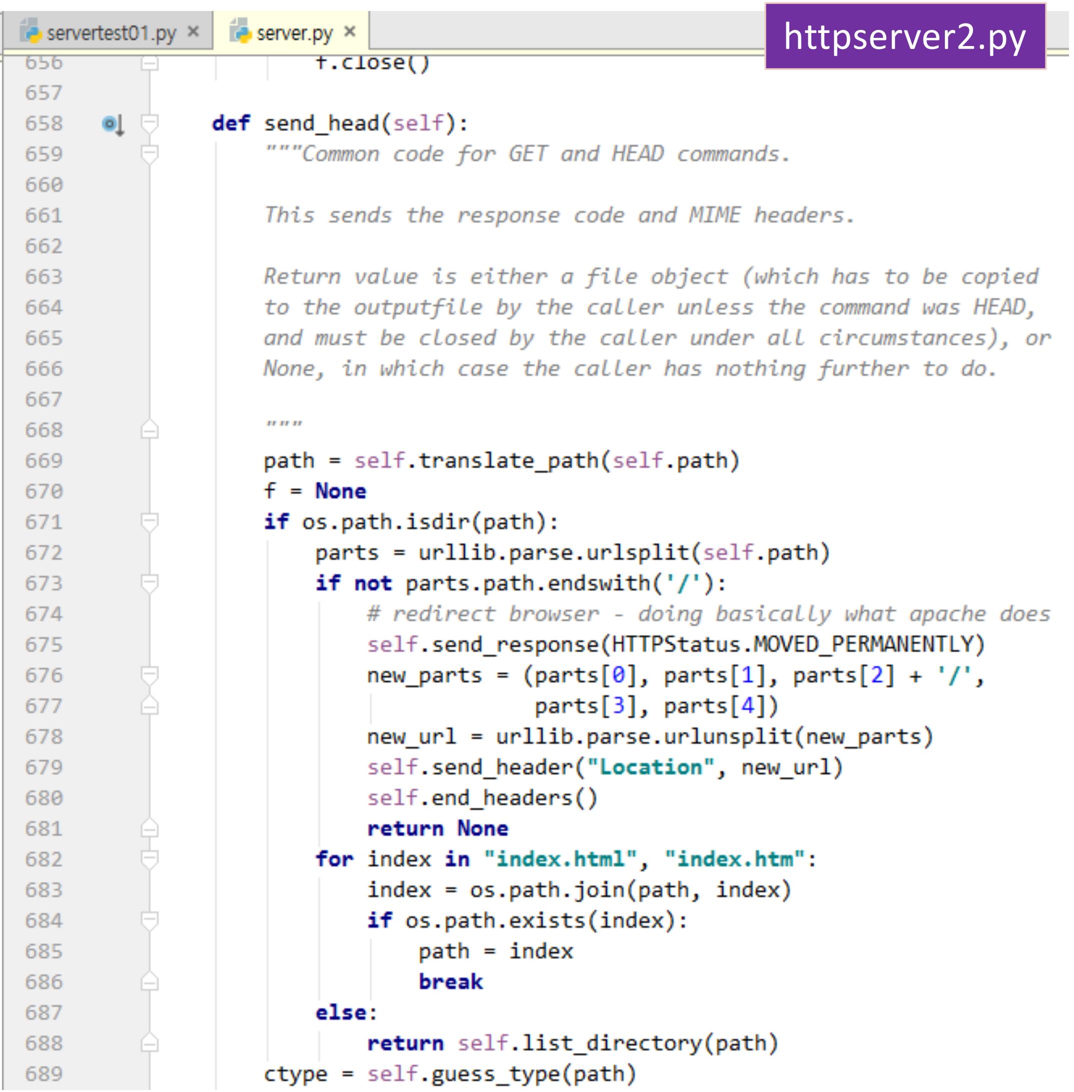
- A very simple implementation of a HTTP server in Python
  - httpserver2.py

```
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()
```



```python
        v: (v.phrase, v.description)
        for v in HTTPStatus.__members__.values()
    }


class SimpleHTTPRequestHandler(BaseHTTPRequestHandler):

    """Simple HTTP request handler with GET and HEAD commands.

    This serves files from the current directory and any of its
    subdirectories.  The MIME type for files is determined by
    calling the .guess_type() method.

    The GET and HEAD requests are identical except that the HEAD
    request omits the actual contents of the file.

    """

    server_version = "SimpleHTTP/" + __version__

    def do_GET(self):
        """Serve a GET request."""
        f = self.send_head()
        if f:
            try:
                self.copyfile(f, self.wfile)
            finally:
                f.close()

    def do_HEAD(self):
        """Serve a HEAD request."""
        f = self.send_head()
        if f:
            f.close()

```

httpserver2.py

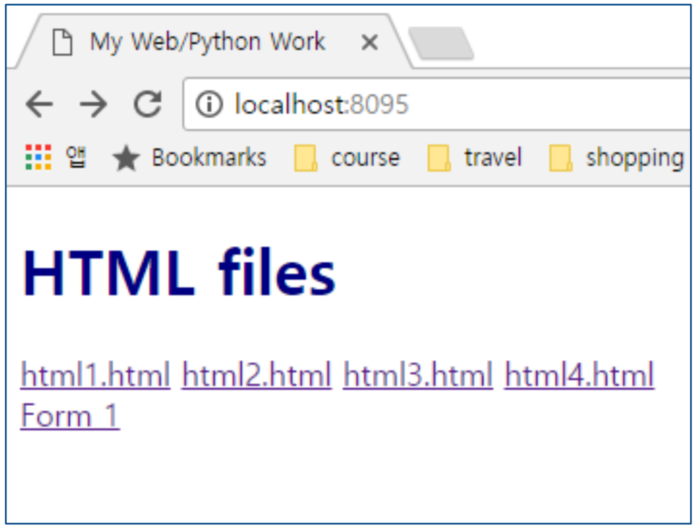


```python
            f.close()

    def send_head(self):
        """Common code for GET and HEAD commands.

        This sends the response code and MIME headers.

        Return value is either a file object (which has to be copied
        to the outputfile by the caller unless the command was HEAD,
        and must be closed by the caller under all circumstances), or
        None, in which case the caller has nothing further to do.

        """
        path = self.translate_path(self.path)
        f = None
        if os.path.isdir(path):
            parts = urllib.parse.urlsplit(self.path)
            if not parts.path.endswith('/'):
                # redirect browser - doing basically what apache does
                self.send_response(HTTPStatus.MOVED_PERMANENTLY)
                new_parts = (parts[0], parts[1], parts[2] + '/',
                             parts[3], parts[4])
                new_url = urllib.parse.urlunsplit(new_parts)
                self.send_header("Location", new_url)
                self.end_headers()
                return None
            for index in "index.html", "index.htm":
                index = os.path.join(path, index)
                if os.path.exists(index):
                    path = index
                    break
            else:
                return self.list_directory(path)
        ctype = self.guess_type(path)
```

httpserver3.py

HTML Form Exercise

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

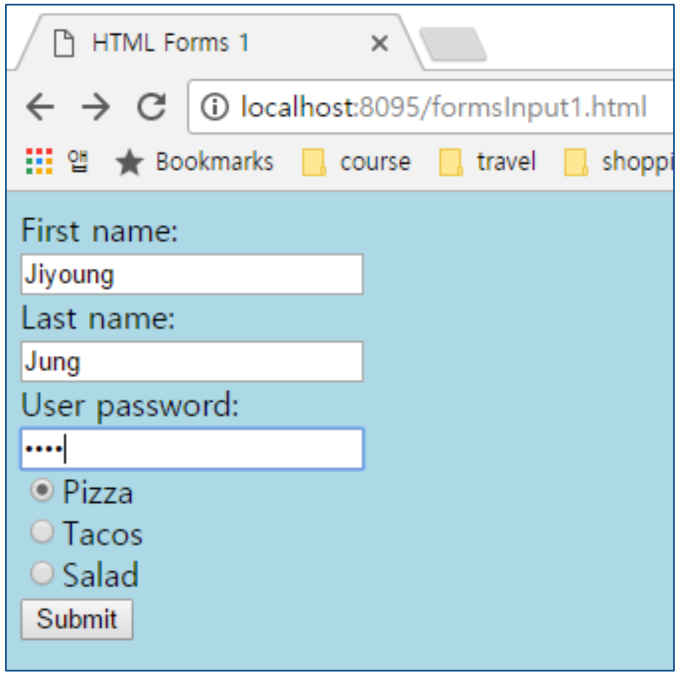

My Web/Python Work

localhost:8095

Bookmarks course travel shopping

HTML files

html1.html html2.html html3.html html4.html Form 1


HTML Forms 1

localhost:8095/formsInput1.html

Bookmarks course travel shoppi

First name:

Jiyoung

Last name:

Jung

User password:

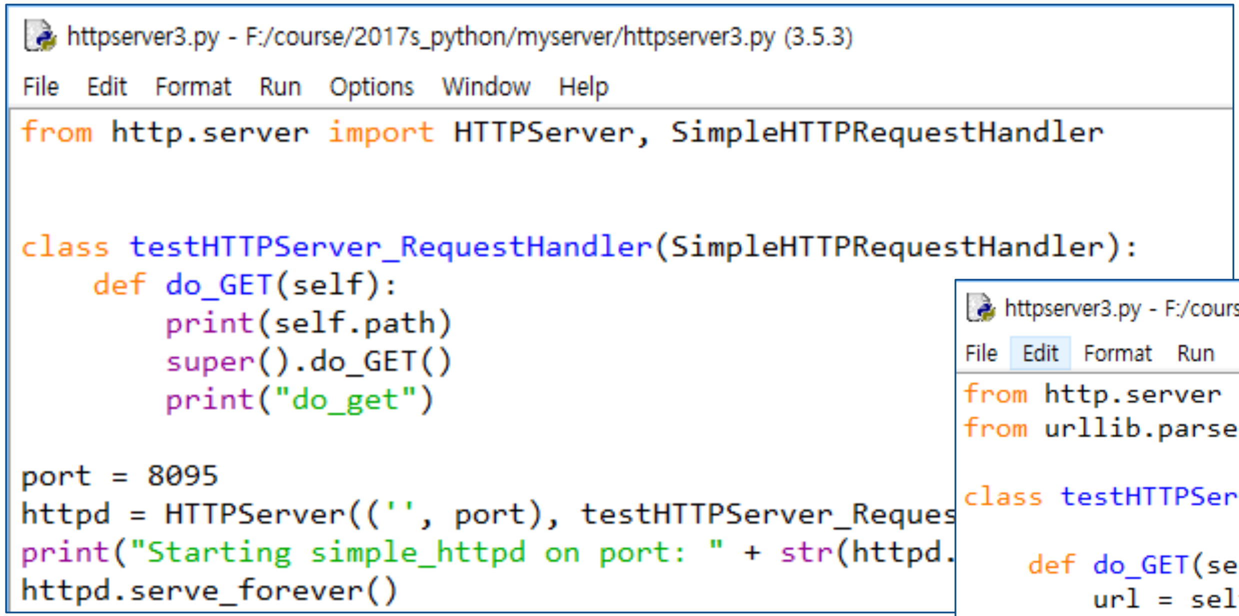
••••

[✓ Pizza] Tacos Salad

Submit

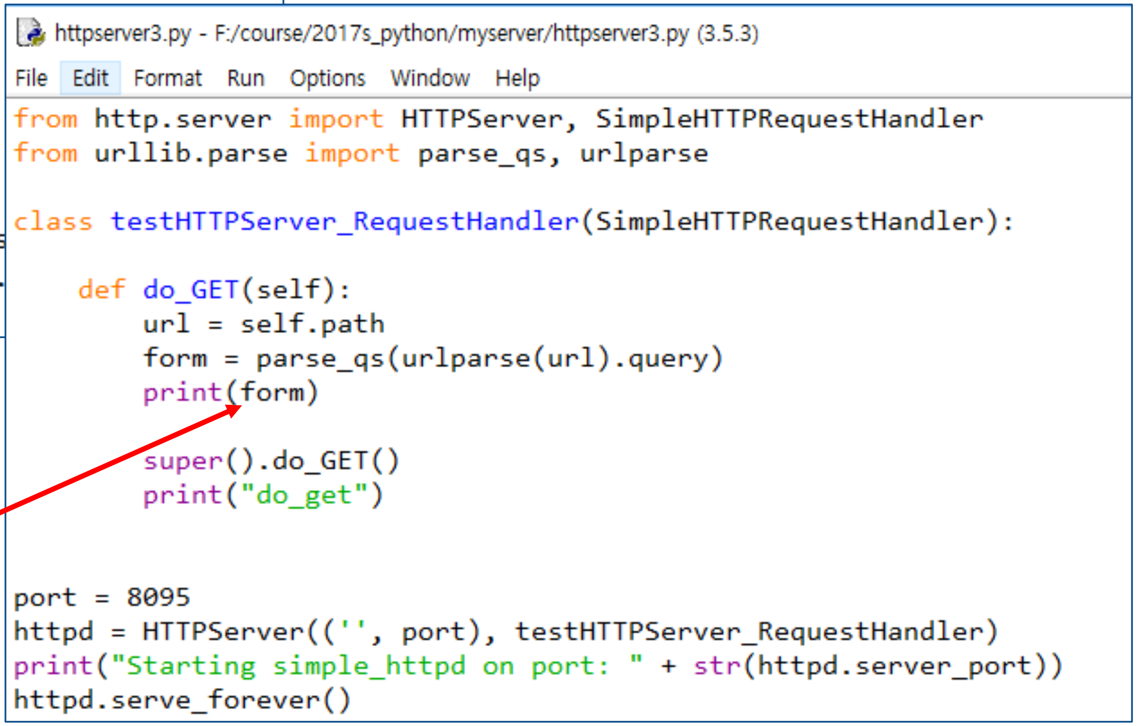
httpserver3.py

Fetch user input



httpserver3.py - F:/course/2017s_python/myserver/httpserver3.py (3.5.3)

```
from http.server import HTTPServer, SimpleHTTPRequestHandler


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

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



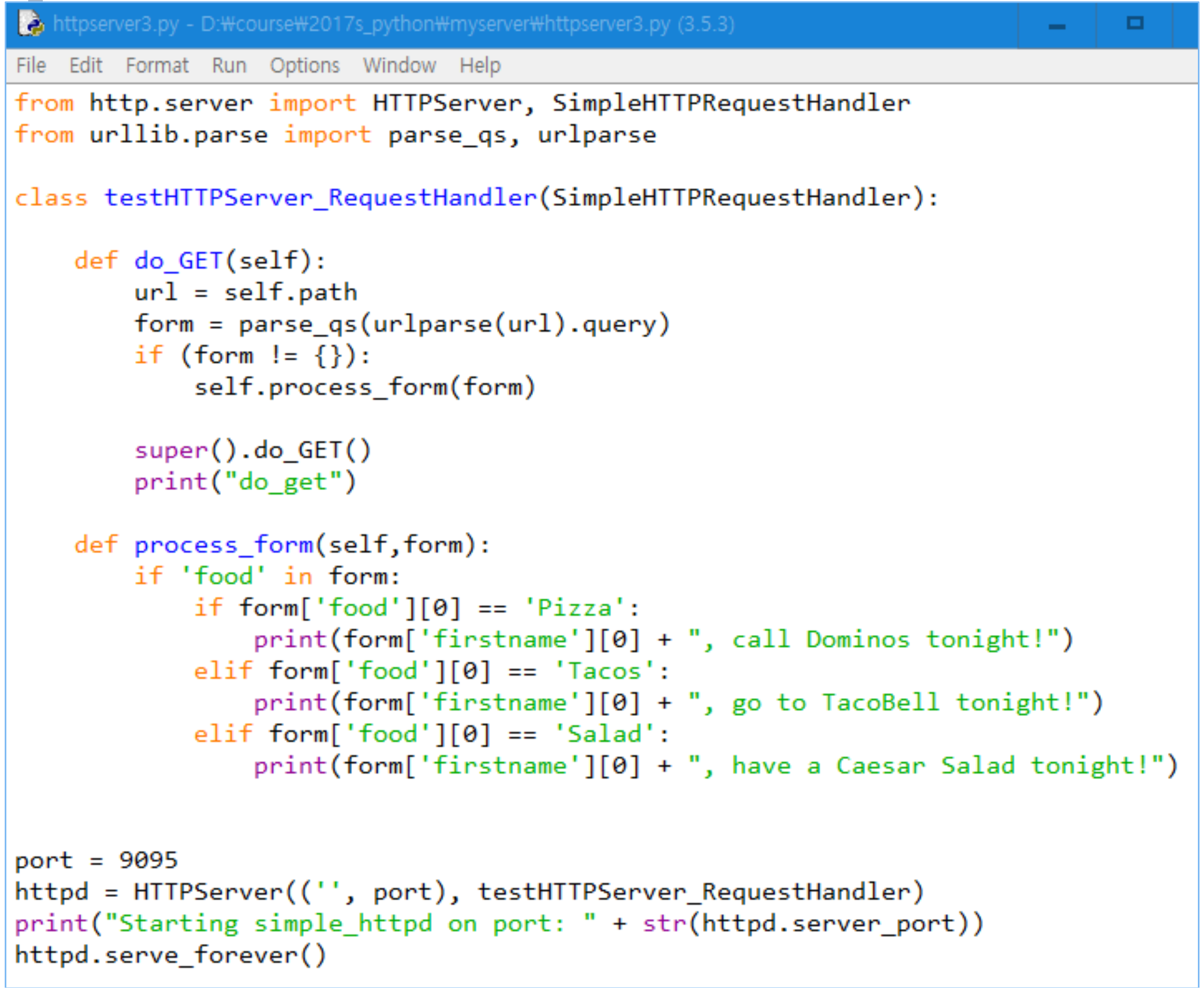
httpserver3.py - F:/course/2017s_python/myserver/httpserver3.py (3.5.3)

```
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)
        print(form)

        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()
```

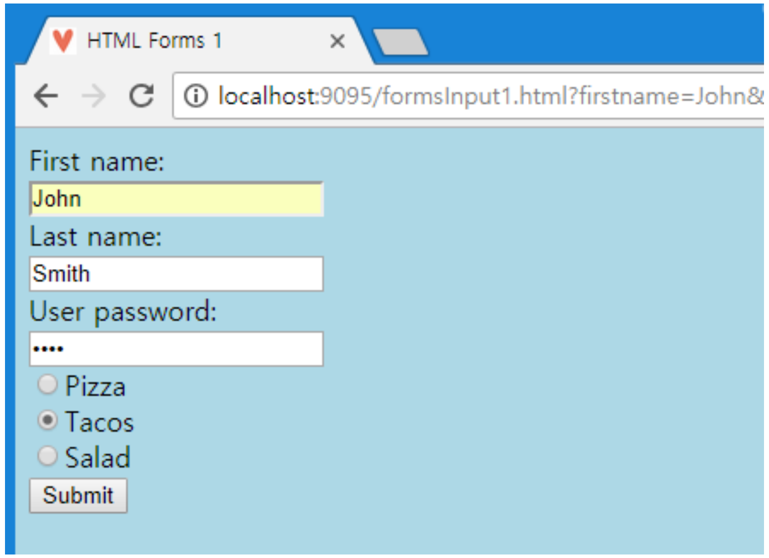
Empty dictionary!

httpserver3.py

Process Form Input


httpserver3.py - D:₩course₩2017s_python₩myserver₩httpserver3.py (3.5.3)
File Edit Format Run Options Window Help

```python
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()
```



First name: John
Last name: Smith
User password: ••••
○ Pizza
● Tacos
○ Salad
Submit

```
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
```