

Examples

Running a simple HTTP server

Python 2.x ≥ 2.3

```
python -m SimpleHTTPServer 9000
```

Python 3.x ≥ 3.0

```
python -m http.server 9000
```

Running this command serves the files of the current directory at port 9000 .

If no argument is provided as port number then server will run on default port 8000 .

The `-m` flag will search `sys.path` for the corresponding `.py` file to run as a module.

If you want to only serve on localhost you'll need to write a custom Python program such as:

```
import sys
import BaseHTTPServer
from SimpleHTTPServer import SimpleHTTPRequestHandler

HandlerClass = SimpleHTTPRequestHandler
ServerClass = BaseHTTPServer.HTTPServer
Protocol = "HTTP/1.0"

if sys.argv[1:]:
    port = int(sys.argv[1])
else:
    port = 8000
server_address = ('127.0.0.1', port)

HandlerClass.protocol_version = Protocol
httpd = ServerClass(server_address, HandlerClass)

sa = httpd.socket.getsockname()
print "Serving HTTP on", sa[0], "port", sa[1], "..."
httpd.serve_forever()
```


Serving files


Assuming you have the following directory of files:

Documents library

files

Name

 factory.py

 facade.py

You can setup a web server to serve these files as follows:

Python 2.x ≥ 2.3

```
import SimpleHTTPServer
import SocketServer

PORT = 8000

handler = SimpleHTTPServer.SimpleHTTPRequestHandler
httpd = SocketServer.TCPServer(("localhost", PORT), handler)
print "Serving files at port {}".format(PORT)
httpd.serve_forever()
```

Python 3.x ≥ 3.0

```
import http.server
import socketserver

PORT = 8000
```

```

handler = http.server.SimpleHTTPRequestHandler
httpd = socketserver.TCPServer(("", PORT), handler)
print("serving at port", PORT)
httpd.serve_forever()

```

The `SocketServer` module provides the classes and functionalities to setup a network server.

`SocketServer`'s `TCPServer` class sets up a server using the TCP protocol. The constructor accepts a tuple representing the address of the server (i.e. the IP address and port) and the class that handles the server requests.

The `SimpleHTTPRequestHandler` class of the `SimpleHTTPServer` module allows the files at the current directory to be served.

Save the script at the same directory and run it.

Run the HTTP Server :

Python 2.x ≥ 2.3

```
python -m SimpleHTTPServer 8000
```

Python 3.x ≥ 3.0

```
python -m http.server 8000
```

The `-m` flag will search `'sys.path'` for the corresponding `'.py'` file to run as a module.

Open `localhost:8000` in the browser, it will give you the following:

Directory listing for /

- [facade.py](#)
- [factory.py](#)
- [server.py](#)

Basic handling of GET, POST, PUT using `BaseHTTPRequestHandler`

```

# from BaseHTTPServer import BaseHTTPRequestHandler, HTTPServer # python2
from http.server import BaseHTTPRequestHandler, HTTPServer # python3
class HandleRequests(BaseHTTPRequestHandler):
    def _set_headers(self):
        self.send_response(200)
        self.send_header('Content-type', 'text/html')
        self.end_headers()

    def do_GET(self):
        self._set_headers()
        self.wfile.write("received get request")

    def do_POST(self):
        '''Reads post request body'''
        self._set_headers()
        content_len = int(self.headers.getheader('content-length', 0))
        post_body = self.rfile.read(content_len)
        self.wfile.write("received post request:<br>{}".format(post_body))

    def do_PUT(self):
        self.do_POST()

host = ''
port = 80
HTTPServer((host, port), HandleRequests).serve_forever()

```

Example output using `curl` :

```

$ curl http://localhost/
received get request%

$ curl -X POST http://localhost/
received post request:<br>%

$ curl -X PUT http://localhost/
received post request:<br>%

```

```
$ echo 'hello world' | curl --data-binary @- http://localhost/  
received post request:<br>hello world
```

Programmatic API of SimpleHTTPServer

What happens when we execute `python -m SimpleHTTPServer 9000` ?

To answer this question we should understand the construct of `SimpleHTTPServer` (<https://hg.python.org/cpython/file/2.7/Lib/SimpleHTTPServer.py>) and `BaseHTTPServer` (<https://hg.python.org/cpython/file/2.7/Lib/BaseHTTPServer.py>) .

Firstly, Python invokes the `SimpleHTTPServer` module with `9000` as an argument. Now observing the `SimpleHTTPServer` code,

```
def test(handlerClass = SimpleHTTPRequestHandler,  
        serverClass = BaseHTTPServer.HTTPServer):  
    BaseHTTPServer.test(handlerClass, serverClass)  
  
if __name__ == '__main__':  
    test()
```

The test function is invoked following request handlers and `ServerClass`. Now `BaseHTTPServer.test` is invoked

```
def test(handlerClass = BaseHTTPRequestHandler,  
        serverClass = HTTPServer, protocol="HTTP/1.0"):  
    """Test the HTTP request handler class.  
  
    This runs an HTTP server on port 8000 (or the first command line  
    argument).  
  
    """  
  
    if sys.argv[1:]:  
        port = int(sys.argv[1])  
    else:  
        port = 8000  
    server_address = ('', port)  
  
    handlerClass.protocol_version = protocol  
    httpd = serverClass(server_address, handlerClass)  
  
    sa = httpd.socket.getsockname()  
    print "Serving HTTP on", sa[0], "port", sa[1], "..."  
    httpd.serve_forever()
```

Hence here the port number, which the user passed as argument is parsed and is bound to the host address. Further basic steps of socket programming with given port and protocol is carried out. Finally socket server is initiated.

This is a basic overview of inheritance from `SocketServer` class to other classes:

```
+-----+  
| BaseServer |  
+-----+  
      |  
      v  
+-----+ +-----+  
| TCPServer |----->| UnixStreamServer |  
+-----+ +-----+  
      |  
      v  
+-----+ +-----+  
| UDPServer |----->| UnixDatagramServer |  
+-----+ +-----+
```

The links <https://hg.python.org/cpython/file/2.7/Lib/BaseHTTPServer.py> and <https://hg.python.org/cpython/file/2.7/Lib/SocketServer.py> are useful for finding further information.

Syntax

Parameters

