CHAPTER 02 파이썬 웹 표준 라이브러리

# 파이썬 웹프로그래밍

# 시작하기전에

- **개발 환경**
  - Django 2.0
  - Python 3.6
  - Windows 10
  - 장고 공식 홈페이지
    https://docs.djangoproject.com/
    (오픈소스)

- **예제 파일 다운로드**
  - http://www.hanbit.co.kr/src/10104

# 이 책의 학습 목표

- CHAPTER 01  프로그래밍의 기본 기술에 대해 이해하기
- CHAPTER 02 파이썬 웹 라이브러리의 전체 구성 및 버전 2.x와 3.x의 달라진 점
- CHAPTER 03 Django 장고를 사용하여 웹 애플리케이션을 만드는 과정
- CHAPTER 04 장고의 기능 중에서 실제 프로젝트 개발을 위해 꼭 알아야 되는 6개 기능
- CHAPTER 05 클래스형 뷰를 사용하는 애플리케이션 만들기
- CHAPTER 06 웹 서버로 오래전부터 사용된 아파치와 차세대 웹 서버로 알려진 NGINX에 대해 알아보기
- CHAPTER 07 파이썬 개발자들이 많이 사용하는 PythonAnywhere 사이트에 대해 알아보기
- CHAPTER 08 아파치 웹 서버에서 장고 프로그램을 연동하기 위해 사용하는 mod_wsgi 프로그램 설정 및 실행
- CHAPTER 09 NGINX의 설치 방법과 장고 프로그램을 실행하기 위한 설정 방법

# Contents

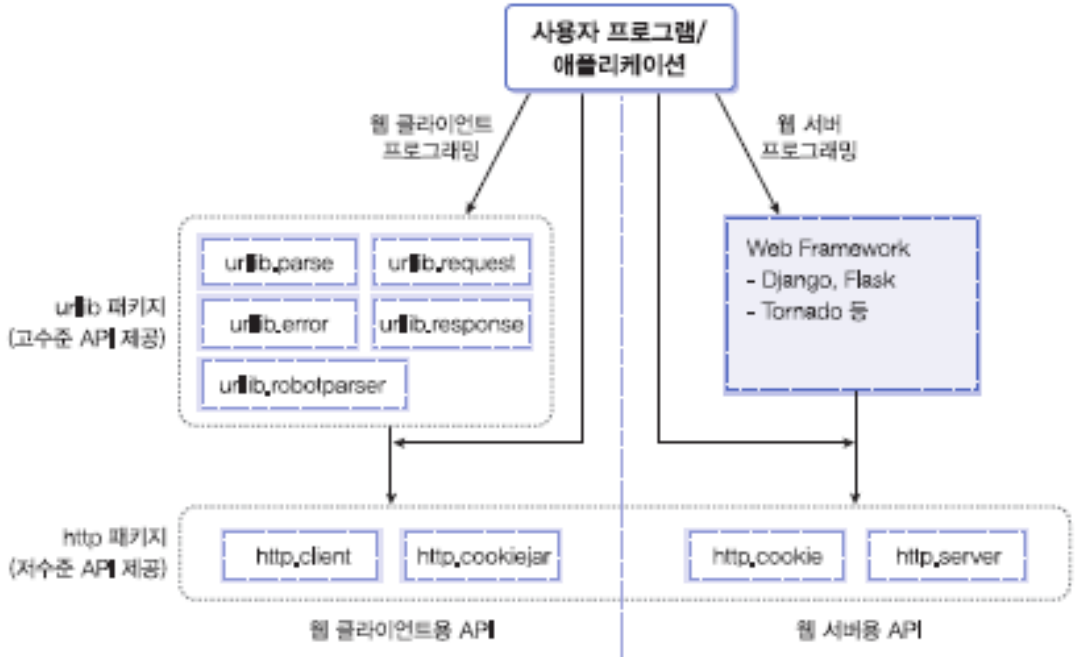
# Contents

# CHAPTER 02 파이썬 웹 표준 라이브러리

파이썬 웹 라이브러리의 전체 구성 및 버전 2.x와 3.x의 달라진 점

# SECTION 02 웹 라이브러리 구성

## 2.1 웹 라이브러리 구성




urllib 패키지에는 웹 클라이언트를 작성하는 데 사용되는 모듈들이 있으며, 가장 빈번하게 사용하는 모듈.

http 패키지는 크게 서버용과 클라이언트용 라이브러리로 나누어 모듈을 담고 있으며, 쿠키 관련 라이브러리도 http 패키지 내에서 서버용과 클라이언트용으로 모듈이 구분

# SECTION 02 웹 라이브러리 구성

표 **2-1** 파이썬 3.x에서 표준 라이브러리의 모듈 구성 변경사항

| 파이썬 3.x 모듈명 | 파이썬 2.x 모듈명 | | 파이썬 3.x에서의 변화 |
|---|---|---|---|
| urllib.parse | urlparse | urllib 일부 | 하나의 urllib 패키지로 모아서 모듈을 기능별로 나눴고, urllib 모듈은 기능에 따라 여러 모듈로 흩어졌습니다. |
| urllib.request | urllib2 대부분 | urllib 일부 | |
| urllib.error | urllib2 에러 부분 | urllib 일부 | |
| urllib.response | | urllib 일부 | |
| urllib.robotparser | robotparser | | |
| http.server | BaseHTTPServer | | 하나의 http 패키지로 모아 server와 client 모듈로 구분했습니다. |
| http.server | CGIHTTPServer | | |
| http.server | SimpleHTTPServer | | |
| http.client | httplib | | |
| http.cookies | Cookie | | 하나의 http 패키지로 모았습니다. |
| http.cookiejar | cookielib | | |
| html.parser | HTMLParser | | 하나의 html 패키지로 모았습니다. |
| html.entities | htmlentitydefs | | |

# SECTION 02_2웹 클라이언트 라이브러리

사용웹 클라이언트를 위한 파이썬 표준 라이브러리가 있지만, 실제 프로젝트에서는 외부 라이브러리인 requests, beautifulsoup4 등을 더 많이 사용하는 편입니다. 좀 더 간편하고 이해하기 쉬운 문법을 제공하기 때문

**Python Shell** 실행 파이썬 언어는 2가지 실행 방법을 제공합니다. 하나는 python 명령어로 파이썬 스크립트 파일 즉, *.py 파일을 실행하는 방법(> **python example.py**). 또 다른 방법은 파이썬 쉘 모드에서 라인 단위로 실행하는 것입 니다. 파이썬 쉘 모드로 진입하려면 python 명령어만 입력(>**python**

## 2.2.1 urllib.parse 모듈

```
C:₩RedBook₩ch2>python
Python 3.6.1 (v3.6.1:69c0db5, Mar 21 2017, 18:41:36) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> from urllib.parse import urlparse
>>> result = urlparse("http://www.python.org:80/guido/python.html;philosophy?overall=3#n10")
>>> result
ParseResult(scheme='http', netloc='www.python.org:80',
path='/guido/python.html', params='philosophy', query='overall=3', fragment='n10')
```

scheme : URL에 사용된 프로토콜을 의미합니다.

- netloc : 네트워크 위치. **user :password@host :port** 형식으로 표현되며, HTTP 프로토콜인 경우는 **host:port** 형식.
- path : 파일이나 애플리케이션 경로를 의미.
- params : 애플리케이션에 전달될 매개변수입니다. 현재는 사용하지 않음.
- query : 질의 문자열 또는 매개변수로, 앰퍼샌드(&)로 구분된 이름=값 쌍 형식으로 표현.
- fragment : 문서 내의 앵커 등 조각을 지정.

# SECTION 02_2웹 클라이언트 라이브러리

```
urlopen(url, data=None, [timeout])
```

- url 인자로 지정한 URL로 연결하고, 유사 파일 객체를 반환.
  url 인자는 문자열이거나, **Request** 클래스의 인스턴스가 올 수 있음.
- url에 **file** 스킴을 지정하면 로컬 파일을 열 수 있음.
- 디폴트 요청 방식은 GET이고, 웹 서버에 전달할 파라미터가 있으면 질의 문자열을 url 인자에 포함해서 보냄.
- 요청 방식을 POST로 보내고 싶으면 data 인자에 질의 문자열을 지정해주면 됨.
- 옵션인 timeout은 응답을 기다리는 타임아웃 시간을 초로 표시

| 사용 케이스 | 사용 방법 |
|---|---|
| URL로 GET/POST 방식의 간단한 요청 처리 | urlopen( ) 함수만으로 가능 |
| PUT, HEAD 메소드 등, 헤더 조작이 필요한 경우 | Request 클래스를 같이 사용 |
| 인증, 쿠키, 프록시 등 복잡한 요청 처리 | 인증/쿠키/프록시 해당 핸들러 클래스를 같이 사용 |

표 **2-2** urlopen() 함수 사용 방법

※ 교재의 상세 과정을 참고하여 실습을 진행합

## 2.2.3 urllib.request 모듈 예제

예제 2-7 urllib.request 모듈 예제 — 소스제공: ch2₩parse_image.p

```
C:₩Users₩shkim>cd C:₩RedBook₩ch2
C:₩RedBook₩ch2>notepad parse_image.py

from urllib.request import urlopen                                    ①
from html.parser import HTMLParser                                    ①

class ImageParser(HTMLParser):                                        ②
    def handle_starttag(self, tag, attrs):                           ③
        if tag != 'img':
            return
        if not hasattr(self, 'result'):
            self.result = []
        for name, value in attrs:                                     ④
            if name == 'src':                                         ④
                self.result.append(value)                             ④

def parse_image(data):                                                ⑤
    parser = ImageParser()
    parser.feed(data)                                                 ⑥
    dataSet = set(x for x in parser.result)                           ⑦
    return dataSet                                                    ⑧

def main():                                                           ⑨
    url = "http://www.google.co.kr"

    with urlopen(url) as f:                                           ⑩
        charset = f.info().get_param('charset')                       ⑪
        data = f.read().decode(charset)                               ⑪

    dataSet = parse_image(data)                                       ⑫

    print("₩n>>>>>>>>> Fetch Images from", url)
    print('₩n'.join(sorted(dataSet)))                                 ⑬

if __name__ == '__main__':                                            ⑭
    main()                                                            ⑭
```

**※ 교재의 상세 과정을 참고하여 실습을 진행합니다.**

## 2.2.4 http.client 모듈

| 순번 | 코딩 순서 | 코딩 예시 |
|---|---|---|
| 1 | 연결 객체 생성 | conn = http.client.HTTPConnection("www.python.org") |
| 2 | 요청을 보냄 | conn.request("GET", "/index.html") |
| 3 | 응답 객체 생성 | response = conn.getresponse( ) |
| 4 | 응답 데이터를 읽음 | data = response.read( ) |
| 5 | 연결을 닫음 | conn.close( ) |

표 **2-3** http.client 모듈 사용 시 코딩 순서

GET, POST 이외의 방식으로 요청을 보내거나, 요청 헤더와 바디 사이에 타이머 를 두어 시간을 지연시키는 등 urllib.request 모듈로는 쉽게 처리할 수 없는 경우 혹은 HTTP 프로토콜 요청에 대한 저수준의 더 세밀한 기능이 필요할 때는 http.client 모듈을 사용함



※ 교재의 상세 과정을 참고하여 실습을 진행합니다.

## 2.2.5 http.client 모듈 예제

예제 2-12 http.client 모듈 예제 — 소스제공: ch2₩download_image.py

```
C:₩Users₩shkim>cd C:₩RedBook₩ch2
C:₩RedBook₩ch2>notepad download_image.py

import os ------------------------------------------------------------------- ❶
from http.client import HTTPConnection
from urllib.parse import urljoin, urlunparse
from urllib.request import urlretrieve
from html.parser import HTMLParser ------------------------------------------

class ImageParser(HTMLParser): ---------------------------------------------- ❷
    def handle_starttag(self, tag, attrs): ---------------------------------- ❸
        if tag != 'img': ------------------------------------------------------
            return                                                            |
        if not hasattr(self, 'result'):                                      |
            self.result = []                                                  | ❹
        for name, value in attrs:                                             |
            if name == 'src':                                                 |
                self.result.append(value) ------------------------------------

def download_image(url, data): ---------------------------------------------- ❺
    if not os.path.exists('DOWNLOAD'): ------------------------------------- ❻
        os.makedirs('DOWNLOAD') -------------------------------------------

    parser = ImageParser() ------------------------------------------------- ❼
    parser.feed(data) -------------------------------------------------------
    dataSet = set(x for x in parser.result) ---------------------------------- ❽

    for x in sorted(dataSet) : --------------------------------------------- ❾
        imageUrl = urljoin(url, x) ------------------------------------------ ❿
        basename = os.path.basename(imageUrl)                                 |
        targetFile = os.path.join('DOWNLOAD', basename)                       |
                                                                              |
        print("Downloading...", imageUrl)                                     |
        urlretrieve(imageUrl, targetFile) ---------------------------------- ⓫

def main(): ------------------------------------------------------------------ ⓬
    host = "www.google.co.kr"

    conn = HTTPConnection(host) -------------------------------------------- ⓭
    conn.request("GET", '')                                                   |
    resp = conn.getresponse() ---------------------------------------------

    charset = resp.msg.get_param('charset') -------------------------------- ⓮
    data = resp.read().decode(charset) ------------------------------------
    conn.close()

    print("Wn>>>>>>>>> Download Images from", host)
    url = urlunparse(('http', host, '', '', '', '')) ----------------------- ⓯
    download_image(url, data) ---------------------------------------------- ⓰

if __name__ == '__main__': --------------------------------------------------
    main() ------------------------------------------------------------------ ⓱
```

※ 교재의 상세 과정을 참고하여 실습을 진행합니다.

# SECTION 02_3웹 서버라이브러리

## 2.3.1 간단한 웹 서버

예제 2-13 간단한 웹 서버 — 소스제공: ch2₩my_httpserver.py

```
C:₩Users₩shkim>cd C:₩RedBook₩ch2
C:₩RedBook₩ch2>notepad my_httpserver.py

from http.server import HTTPServer, BaseHTTPRequestHandler ---- 1

class MyHandler(BaseHTTPRequestHandler):
    def do_GET(self):
        self.send_response_only(200, 'OK')
        self.send_header('Content-Type', 'text/plain')
        self.end_headers()
        self.wfile.write(b"Hello World")

if __name__ == '__main__':
    server = HTTPServer(('', 8888), MyHandler) ---- 3
    print("Started WebServer on port 8888...")
    print("Press ^C to quit WebServer.")
    server.serve_forever() ---- 4
```

2 (do_GET 메서드 내부)

| 클래스명 | 주요 기능 |
|---|---|
| HTTPServer | • 웹 서버를 만들기 위한 클래스로, 서버 IP와 PORT를 바인딩함<br>• HTTPServer 객체 생성 시, 핸들러 클래스가 반드시 필요함 |
| BaseHTTPRequestHandler | • 핸들러를 만들기 위한 기반 클래스로, HTTP 프로토콜 처리 로직이 들어있음<br>• 이 클래스를 상속받아, 자신의 로직 처리를 담당하는 핸들러 클래스를 만듦 |
| SimpleHTTPRequestHandler | • BaseHTTPRequestHandler 클래스를 상속받아 만든 클래스<br>• GET과 HEAD 메소드 처리가 가능한 핸들러 클래스 |
| CGIHTTPRequestHandler | • SimpleHTTPRequestHandler 클래스를 상속받아 만든 클래스<br>• 추가적으로 POST 메소드와 CGI 처리가 가능한 핸들러 클래스 |

**※ 교재의 상세 과정을 참고하여 실습을 진행합니다.**

## 2.3.2 HTTPServer 및 BaseHTTPRequestHandler 클래스

앞 절에서 설명한 것처럼 우리가 원하는 웹 서버를 만들기 위해서는 기반 클래스를 임포트하거나 상 속받아야 함.
이처럼 기반이 되는 클래스가 바로 **HTTPServer** 및 **BaseHTTPRequestHandler** 클래스

```
C:₩RedBook₩ch2>python my_httpserver.py
```

```
C:₩RedBook₩ch2>python my_httpserver.py
Started WebServer on port 8888...
Press ^C to quit WebServer.
```

그림 **2-4** 웹 서버(my_httpserver.py) 실행 화면

```
http://127.0.0.1:8888/
```


그림 **2-5** 웹 서버(my_httpserver.py) 접속 결과

## 2.3.3 SimpleHTTPRequestHandler

```
C:₩RedBook₩ch2>python -m http.server 8888
```

```
C:₩RedBook₩ch2>python -m http.server 8888
Serving HTTP on 0.0.0.0 port 8888 (http://0.0.0.0:8888/) ...
```

그림 **2-6** Simple 웹 서버 실행 화면


디렉토리 리스트가 나오는 것은 SimpleHTTPRequestHandler의 do_GET() 메소드가 디렉토리 리스트를 반환하도록 구현되어 있기 때문

## 2.3.4 CGIHTTPRequestHandler 클래스

SimpleHTTPRequestHandler 클래스와 유사하게 CGIHTTPRequestHandler 클래스가 미리 구현되어 있어서 필요할 때 즉시 웹 서버를 실행할 수 있습니다. CGIHTTPRequestHandler 클래스에는 do_POST() 메소드가 정의되어 있어서 POST 방식을 처리할 수 있습니다. 물론 SimpleHTTPRequestHandler 클래스를 상속받고 있어서, GET 및 HEAD 방식도 처리합니다.

```
C:\Users\shkim>cd C:\RedBook\ch2\cgi-server
C:\RedBook\ch2\cgi-server>python -m http.server 8888 --cgi
```

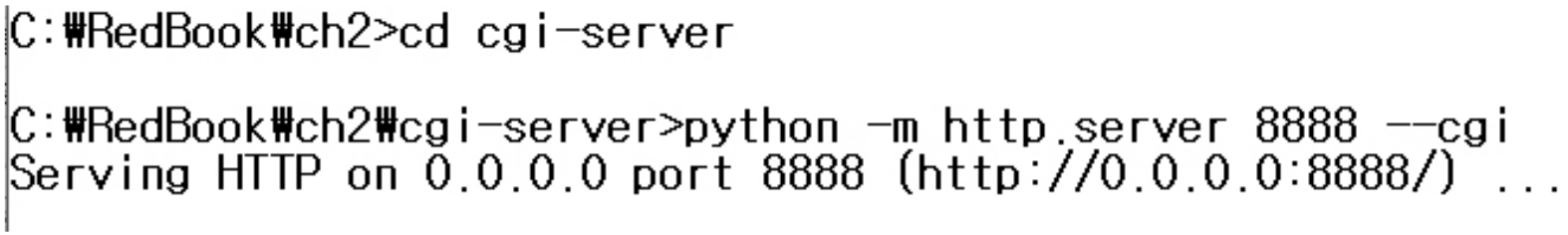



그림 **2-8** CGI 웹 서버 실행 화면

## 2.3.5 xxxHTTPServer 모듈 간의 관계(파이썬 2.× 버전만 해당)

모든 HTTP 웹 서버는 BaseHTTPServer 모듈의 HTTPServer 클래스를 사용하여 작성하고, 웹 서버에 사용되는 핸들러는 BaseHTTPServer 모듈의 BaseHTTPRequestHandler를 상속받아 작성. BaseHTTPServer 모듈이 기본이 되고, 이를 확장한 모듈이 SimpleHTTPServer 모듈이며, CGIHTTPServer 모듈은 SimpleHTTPServer 모듈을 확장하여 작성

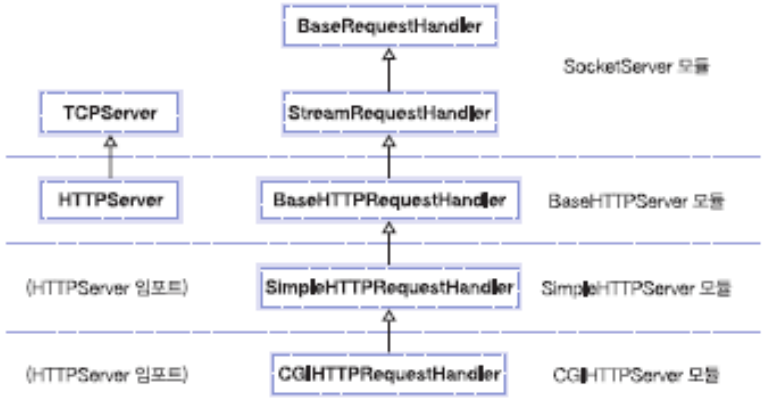



그림 **2-10** xxxHTTPServer 모듈의 클래스 간 상속도


그림 **2-11** 웹 서버용 모듈의 test( ) 함수 간 호출 관계

# SECTION 2.4 CGI/WSGI 라이브러리

## **2.4.1** CGI 관련 모듈

웹 서버가 사용자의 요청을 애플리케이션에 전달하고 애플리케이션의 처리 결과를 애플 리케이션으로부터 되돌려받기 위한, 즉 웹 서버와 애플리케이션 간에 데이터를 주고받기 위한 규 격을 CGICommon Gateway Interface라고 함

## 2.4.2 WSGI 개요

앞 절에서 설명한 CGI 방식은 요청이 들어올 때마다 처리를 위한 프로세스가 생성되는 방식이라 서, 짧은 시간에 수천, 수만의 다량 요청을 받으면 서버의 부하가 높아져서 프로세스가 멈추거나 다운될 수도 있습니다. 이러한 CGI의 단점을 해결하고, 파이썬 언어로 애플리케이션을 좀 더 쉽게 작성할 수 있도록 웹 서버와 웹 애플리케이션 간에 연동 규격을 정의한 것이W SGI 규격

## 2.4.3 WSGI 서버의 애플리케이션 처리 과정

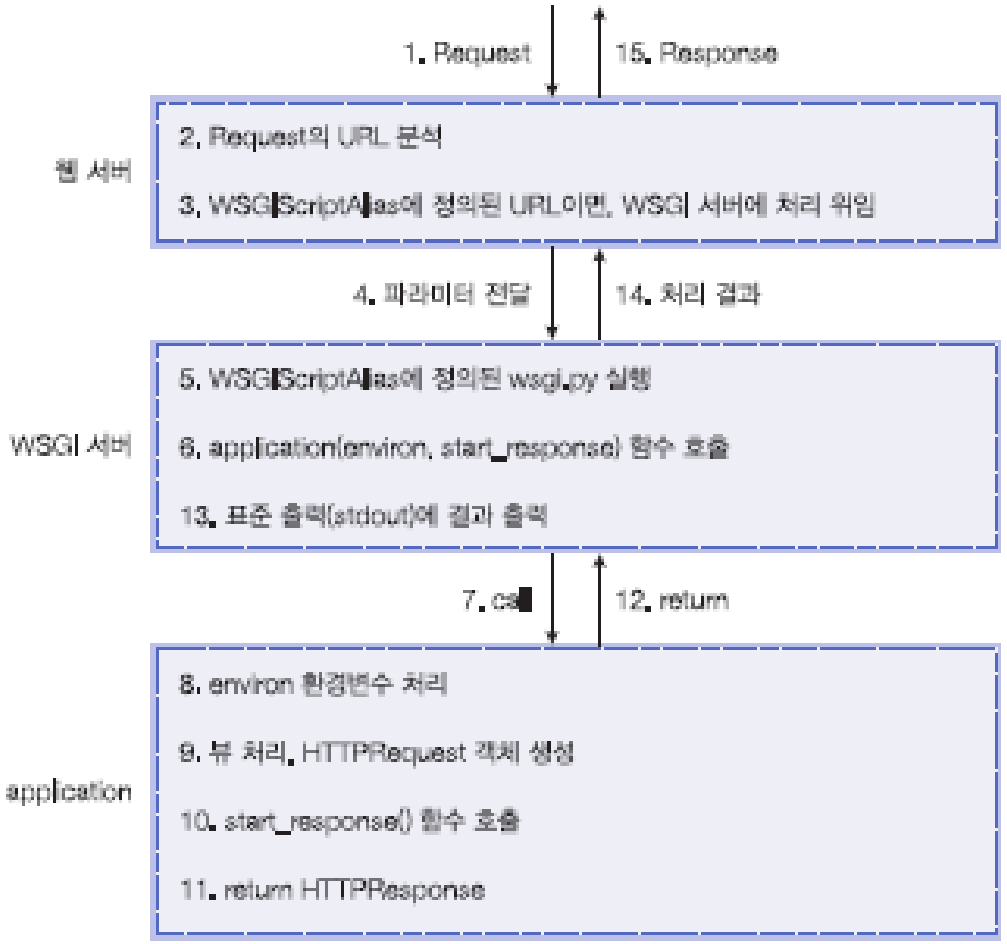



그림 **2-12** WSGI 애플리케이션의 처리 순서

## 2.4.4 wsgiref.simple_server 모듈

이 모듈은 WSGI 스펙을 준수하는 웹 서버(일명, WSGI 서버)에 대한 참조reference 서버, 즉 개발자에게 참고 가 될 수 있도록 미리 만들어 놓은 WSGIServer 클래스와 WSGIRequestHandler 클래스를 정의.
장고의 runserver도 이들 클래스를 사용하여 만든 테스트용 웹 서버

예제 2-16 WSGI 서버 — 소스제공: ch2₩wsgi-server₩my_wsgiserver.py

```
C:₩Users₩shkim>cd C:₩RedBook₩ch2₩wsgi-server
C:₩RedBook₩ch2₩wsgi-server>notepad my_wsgiserver.py

from wsgiref.simple_server import make_server

def my_app(environ, start_response):

    status = '200 OK'
    headers = [('Content-Type', 'text/plain')]
    start_response(status, headers)

    response = [b"This is a sample WSGI Application."]

    return response

if __name__ == '__main__':
    print("Started WSGI Server on port 8888...")
    server = make_server('', 8888, my_app)
    server.serve_forever()
```

## 2.4.5 WSGI 서버 동작 확인

```
C:\Users\wshkim>cd C:\RedBook\ch2\wsgi-server
C:\RedBook\ch2\wsgi-server>python my_wsgiserver.py
```

```
C:₩RedBook₩ch2₩wsgi-server>python my_wsgiserver.py
Started WSGI Server on port 8888...
```

그림 **2-14** WSGI 서버(my_wsgiserver.py) 실행 화면

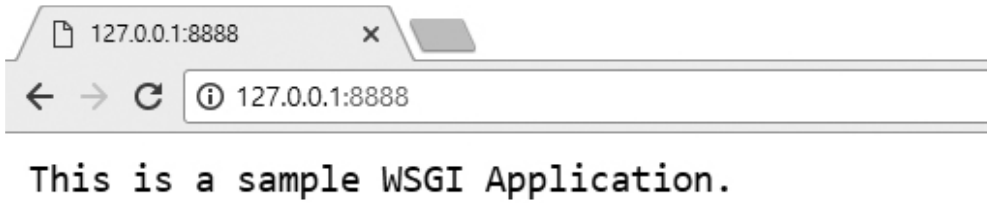



그림 **2-15** WSGI 서버에서 my_app( ) Application 실행 결과