**Edd Mann** </> Developer

# Understanding Python WSGI with Examples

17 Feb 2014

Coming from a strongly PHP background, initially looking at the web-development landscape whilst delving into Python seemed a little confusing. As Python was not developed for the web from an offset, a specification was accepted called PEP 333 which standardised the required interface between Web servers and Python Web Frameworks/Applications. Despite the additional complexity, the manner in which middle-ware applications can be integrated, along with the server choice add possibilities that I find hard to locate a comparable in PHP.

## Basic Example

Simply put a WSGI (Web Sever Gateway Interface) compliant application must supply a callable (function, class) which accepts a 'environ' dictionary and 'start_response' function. For a familiar PHP comparison, you can think of the 'environ' dictionary as a combined '$_SERVER', '$_GET' and '$_POST', with extra processing required. This callable is expected to invoke the 'start_response' function with the desired response-code/header-data, and then return a byte iterable with the response body.

```
def app(environ, start_response):
    start_response('200 OK', [('Content-Type', 'text/html')])
    return [b'Hello, world!']

if __name__ == '__main__':
    try:
        from wsgiref.simple_server import make_server
        httpd = make_server('', 8080, app)
        print('Serving on port 8080...')
        httpd.serve_forever()
    except KeyboardInterrupt:
        print('Goodbye.')
```

Using the single-threaded WSGI reference implementation provided with Python is a great choice for experimenting with these lower-level concepts. You will notice that as the example is written for Python 3 we must return an iterable (in this case a list) with declared 'byte' content inside.

## Post Example

Now that we are familiar with the basic structure of a WSGI compliant application we can now experiment with a more practical example. Below we provide the client with a simple form which posts a supplied 'name' for the application to greet accordingly.

```
import cgi

form = b'''
<html>
    <head>
        <title>Hello User!</title>
    </head>
    <body>
        <form method="post">
            <label>Hello</label>
            <input type="text" name="name">
            <input type="submit" value="Go">
        </form>
    </body>
</html>
'''

def app(environ, start_response):
    html = form

    if environ['REQUEST_METHOD'] == 'POST':
        post_env = environ.copy()
        post_env['QUERY_STRING'] = ''
        post = cgi.FieldStorage(
            fp=environ['wsgi.input'],
            environ=post_env,
            keep_blank_values=True
        )
        html = b'Hello, ' + post['name'].value + '!'

    start_response('200 OK', [('Content-Type', 'text/html')])
    return [html]

if __name__ == '__main__':
    try:
        from wsgiref.simple_server import make_server
        httpd = make_server('', 8080, app)
        print('Serving on port 8080...')
        httpd.serve_forever()
    except KeyboardInterrupt:
        print('Goodbye.')
```

Although somewhat verbose we have been able to create a simple web application which handles supplied POST data using the CGI modules 'FieldStorage' class. These are the very simplified building blocks used in popular frameworks such as Flask and Django.

Developer at MyBuilder
Three Devs and a Maybe podcast co-host
All ramblings can be found in the Archive