⚠ **Warning**

I will be closing off free access to my Python 2 (https://learnpythonthehardway.org/book/) and Python 3 (https://learnpythonthehardway.org/python3/") at Midnight on July 8th, 2017. If you need this book then please purchase it (http://bit.ly/buypython). If you need a free book then I recommend you read my Learn Ruby The Hard Way (https://learnrubythehardway.org/book/) book instead. You can read more about my decision at the blog (https://blog.learncodethehardway.com/2017/07/07/learn-python-3-the-hard-way-officially-released/) and you can email me at help@learncodethehardway.org if you need help with this. My sincere apologies to anyone that is inconvenienced by this decision.

Exercise 50: Your First Website

These final three exercises will be very hard and you should take your time with them. In this first one you'll build a simple web version of one of your games. Before you attempt this exercise you *must* have completed Exercise 46 successfully and have a working `pip` installed such that you can install packages and know how to make a skeleton project directory. If you don't remember how to do this, go back to Exercise 46 and do it all over again.

Installing lpthw.web

Before creating your first web application, you'll first need to install the "web framework" called `lpthw.web`. The term "framework" generally means "some package that makes it easier for me to do something." In the world of web applications, people create "web frameworks" to compensate for the difficult problems they've encountered when making their own sites. They share these common solutions in the form of a package you can download to bootstrap your own projects.

In our case, we'll be using the `lpthw.web` framework, but there are many, many, *many* others you can choose from. For now, learn `lpthw.web`, then branch out to another one when you're ready (or just keep using `lpthw.web` since it's good enough).

Using `pip`, install `lpthw.web`:

```
$ sudo pip install lpthw.web
[sudo] password for zedshaw:
Downloading/unpacking lpthw.web
  Running setup.py egg_info for package lpthw.web

Installing collected packages: lpthw.web
  Running setup.py install for lpthw.web

Successfully installed lpthw.web
Cleaning up...
```

This will work on Linux and Mac OSX computers, but on Windows just drop the `sudo` part of the `pip install` command and it should work. If not, go back to Exercise 46 and make sure you can do it reliably.

⚠ **Warning**

Other Python programmers will warn you that `lpthw.web` is just a fork of another web framework called `web.py` and that `web.py` has too much "magic." If they say this, point out to them that Google App Engine originally used `web.py` and not a single Python programmer complained that it had too much magic, because they all worked at Google. If it's good enough for Google, then it's good enough for you to get started. Then, just get back to learning to code and ignore their goal of indoctrination over education.

Make a Simple "Hello World" Project

Now you're going to make an initial very simple "Hello World" web application and project directory using `lpthw.web`. First, make your project directory:

```
$ cd projects
$ mkdir gothonweb
$ cd gothonweb
$ mkdir bin gothonweb tests docs templates
$ touch gothonweb/__init__.py
$ touch tests/__init__.py
```

You'll be taking the game from Exercise 43 and making it into a web application, so that's why you're calling it `gothonweb`. Before you do that, we need to create the most basic `lpthw.web` application possible. Put the following code into `bin/app.py`:

```
import web

urls = (
  '/', 'index'
)

app = web.application(urls, globals())

class index:
    def GET(self):
        greeting = "Hello World"
        return greeting

if __name__ == "__main__":
    app.run()
```

Then run the application like this:

```
$ python bin/app.py
http://0.0.0.0:8080/
```

However, if you did this:

```
$ cd bin/   # WRONG! WRONG! WRONG!
$ python app.py  # WRONG! WRONG! WRONG!
```

Then you are doing it *wrong*. In all Python projects you do not `cd` into a lower directory to run things. You stay at the top and run everything from there so that all of the system can access all the modules and files. Go reread Exercise 46 to understand a project layout and how to use it if you did this.

Finally, use your web browser and go to `http://localhost:8080/` and you should see two things. First, in your browser you'll see `Hello, world!`. Second, you'll see your Terminal with new output like this:

```
$ python bin/app.py
http://0.0.0.0:8080/
127.0.0.1:59542 - - [13/Jun/2011 11:44:43] "HTTP/1.1 GET /" - 200 OK
127.0.0.1:59542 - - [13/Jun/2011 11:44:43] "HTTP/1.1 GET /favicon.ico" - 404 
```

Those are log messages that `lpthw.web` prints out so you can see that the server is working, and what the browser is doing behind the scenes. The log messages help you debug and figure out when you have problems. For example, it's saying that your browser tried to get `/favicon.ico` but that file didn't exist so it returned `404 Not Found` status code.

I haven't explained the way *any* of this web stuff works yet, because I want to get you setup and ready to roll so that I can explain it better in the next two exercises. To accomplish this, I'll have you break your lpthw.web application in various ways and then restructure it so that you know how it's setup.

What's Going On?

Here's what's happening when your browser hits your application:

1. Your browser makes a network connection to your own computer, which is called `localhost` and is a standard way of saying "whatever my own computer is called on the network." It also uses port 8080.
2. Once it connects, it makes an HTTP request to the `bin/app.py` application and asks for the `/` URL, which is commonly the first URL on any website.
3. Inside `bin/app.py` you've got a list of URLs and what classes they match. The only one we have is the `'/', 'index'` mapping. This means that whenever someone goes to `/` with a browser, `lpthw.web` will find the `class index` and load it to handle the request.
4. Now that `lpthw.web` has found `class index` it calls the `index.GET` method on an instance of that class to actually handle the request. This function runs and simply returns a string for what `lpthw.web` should send to the browser.
5. Finally, `lpthw.web` has handled the request and sends this response to the browser, which is what you are seeing.

Make sure you really understand this. Draw up a diagram of how this information flows from your browser, to `lpthw.web` , then to `index.GET` and back to your browser.

Fixing Errors

First, delete line 11 where you assign the `greeting` variable, then hit refresh in your browser. You should see an error page now that gives you lots of information on how your application just exploded. You know that the variable `greeting` is now missing, but `lpthw.web` gives you this nice error page to track down exactly where. Do each of the following with this page:

1. Look at each of the `Local vars` outputs (click on them) and see if you can follow what variables it's talking about and where they are.
2. Look at the `Request Information` section and see if it matches anything you're already familiar with. This is information that your web browser is sending to your `gothonweb` application. You normally don't even know that it's sending this stuff, so now you get to see what it does.
3. Try breaking this simple application in other ways and explore what happens. Don't forget to also look at the logs being printed into your Terminal as `lpthw.web` will put other stack traces and information there too.

Create Basic Templates

You can break your `lpthw.web` application, but did you notice that "Hello World" isn't a very good HTML page? This is a web application, and as such it needs a proper HTML response. To do that you will create a simple template that says "Hello World" in a big green font.

The first step is to create a `templates/index.html` file that looks like this:

```
$def with (greeting)


<html>
    <head>
        <title>Gothons Of Planet Percal #25</title>
    </head>
<body>


$if greeting:
    I just wanted to say <em style="color: green; font-size: 2em;">$greeting<
$else:
    <em>Hello</em>, world!


</body>
</html>
```

If you know what HTML is, then this should look fairly familiar. If not, research HTML and try writing a few web pages by hand so you know how it works. This HTML file, however, is a *template*, which means that `lpthw.web` will fill in "holes" in the text depending on variables you pass in to the template. Every place you see `$greeting` will be a variable you'll pass to the template that alters its contents.

To make your `bin/app.py` do this, you need to add some code to tell `lpthw.web` where to load the template and to render it. Take that file and change it like this:

```python
import web

urls = (
  '/', 'Index'
)

app = web.application(urls, globals())

render = web.template.render('templates/')

class Index(object):
    def GET(self):
        greeting = "Hello World"
        return render.index(greeting = greeting)

if __name__ == "__main__":
    app.run()
```

Pay close attention to the new `render` variable and how I changed the last line of `index.GET` so it returns `render.index()` passing in your `greeting` variable.

Once you have that in place, reload the web page in your browser and you should see a different message in green. You should also be able to do a `View Source` on the page in your browser to see that it is valid HTML.

This may have flown by you very fast, so let me explain how a template works:

1. In your `bin/app.py` you've added a new variable, `render`, which is a `web.template.render` object.

2. This `render` object knows how to load `.html` files out of the `templates/` directory because you passed that to it as a parameter.
3. Later in your code, when the browser hits the `index.GET` like before, instead of just returning the string `greeting`, you call `render.index` and pass the greeting to it as a variable.
4. This `render.index` method is kind of a *magic* function where the `render` object sees that you're asking for `index`, goes into the `templates/` directory, looks for a page named `index.html`, and then "renders" it, or converts it.
5. In the `templates/index.html` file you see the beginning definition that says this template takes a `greeting` parameter, just like a function. Also, just like Python this template is indentation sensitive, so make sure you get them right.
6. Finally, you have the HTML in `templates/index.html` that looks at the `greeting` variable and, if it's there, prints one message using the `$greeting`, or a default message.

To get deeper into this, change the greeting variable and the HTML to see what effect it has. Also create another template named `templates/foo.html` and render that using `render.foo()` instead of `render.index()` like before. This will show you how the name of the function you call on `render` is just matched to an `.html` file in `templates/`.

Study Drills

1. Read the documentation at http://webpy.org/ (http://webpy.org/) which is the same as the `lpthw.web` project.
2. Experiment with everything you can find there, including their example code.

3. Read about HTML5 and CSS3 and make some other .html and .css files for practice.
4. If you have a friend who knows Django and is willing to help you, then consider doing Exercises 50, 51, and 52 in Django instead to see what that's like.

Common Student Questions

I can't seem to connect to `http://localhost:8080/` .

Try going to `http://127.0.0.1:8080/` instead.

What is the difference betwee `lpthw.web` and `web.py` ?

No difference. I simply "locked" `web.py` at a particular version so that it would be consistent for students, then named it `lpthw.web` . Later versions of `web.py` might be different from this version.

I can't find `index.html` (or just about anything).

You probably are doing `cd bin/` first and then trying to work with the project. Do not do this. All of the commands and instructions assume you are one directory above `bin/` , so if you can't type `python bin/app.py` then you are in the wrong directory.

Why do we assign `greeting=greeting` when we call the template?

You are not assigning to `greeting` . You are setting a named parameter to give to the template. It's sort of an assignment, but it only affects the call to the template function.

I can't use port 8080 on my computer.

You probably have an anti-virus program installed that is using that port. Try a different port.

After installing `lpthw.web` I get `ImportError "No module named web"` .

You most likely have multiple versions of Python installed and are using the wrong one, or you didn't do the install correctly because of an old version of `pip` . Try uninstalling `lpthw.web` and reinstalling it. If that doesn't work make triple sure you're using the right version of Python.

Gitter LCodeTHW Community Chat

Zed sometimes hangs out in the Learn Code The Hard Way community chat rooms at http://bit.ly/lcthwchat (http://bit.ly/lcthwchat) If you have a quick question or just want to hang out with other people working on the books then join in.

$ 29. 99

BUY THE PYTHON 2 COURSE
(HTTPS://SHOP.LEARNCODETHEHARDWAY.ORG/ACCESS/BUY/2/)

PRE-ORDER THE PYTHON 3 COURSE INSTEAD
(HTTPS://SHOP.LEARNCODETHEHARDWAY.ORG/ACCESS/BUY/9/)

OR, YOU CAN READ LEARN PYTHON THE HARD WAY FOR FREE

(HTTPS://LEARNPYTHONTHEHARDWAY.ORG/BOOK/) RIGHT HERE, VIDEO LECTURES NOT INCLUDED.

Other Buying Options

BUY ON AMAZON (HTTP://BIT.LY/AMZNLPTHW)

BUY A HARD COPY FROM THE PUBLISHER (HTTP://BIT.LY/INFORMITLPTHW)

BUY A HARD COPY FROM BARNES & NOBLE (HTTP://BIT.LY/BNLPTHW)

Previous (ex49.html) Next (ex51.html)

HOME (/) **ABOUT (HTTPS://LEARNCODETHEHARDWAY.ORG/ABOUT/)** **CONTACT (HTTPS://LEARNCODETHEHARDWAY.ORG/CONTACT/)**

(https://twitter.com/lzsthw)