

# ECE 364 Lab 11 Handout

November 13

Name: \_\_\_\_\_ ID: ee364\_\_\_\_\_ Email: \_\_\_\_\_@purdue.edu

## Instructions

- You will work in the directory called Lab11.
- To submit, commit your files to SVN. ⚠ We will only grade the version of the file that is in SVN.
- ⚠ You must meet all base requirements in the syllabus to receive any credit.
- Use Python 3.4 (⚠ not 2.x, 3.3, 3.5, etc.) In PyCharm: File → Settings → Project Interpreter → Make sure that Python 3.4 (/usr/local/bin/python3.4) is selected.

⚠ Failure to meet all base requirements will result in zero credit. This includes submitting code that works with other versions of Python (e.g., 2.x,  $\leq 3.3$ ,  $\geq 3.5$ , etc., but not Python 3.4).

Rules: You may discuss the tools and metrics. You may use the web.

v4

---

## Web Page Outliner

This will get you started on the project. After this lab, you will be about 2/3 done with milestone #1.

### 0. Read this entire document—to the end—before you begin.

You are making three helper functions that will then be used by your web application. It all comes together at the end. Also, keep in mind: This is *similar*—but not the same—as the project milestone 1.

### 1. Web page outliner

1. Create a Python file called `wpo.py` in your Lab11 directory.
2. Make a function that fetches the HTML at a URL and returns a `str`.

```
def get_html_at_url(url, charset="UTF-8"):
```

Use the same method specified in the project description (<http://goo.gl/dk8u5S>).

Note that `urllib.request.urlopen(...)` returns a response. From the response, you can call `response.read()` to get a bytestring (raw bytes). That must be decoded (`s.decode(charset)`) to produce a `str`. The function signature above takes a variable called `charset` with default value of "UTF-8". This function must return a `str`.

3. Make a function that returns an `ElementTree` for a given HTML string and URL.

```
def make_etree(html, url):
```



To create the `ElementTree` object, use the `lxml` module as follows:

```
from lxml.html import HTMLParser, document_fromstring
parser = HTMLParser(encoding="UTF-8")
root = document_fromstring(html, parser=parser, base_url=url)
```

You may copy that code. This is snippet generic enough that no attribution is needed.

The root now contains the root of an `ElementTree` object.

To test:

```
>>> from wpo import get_html_at_url, make_etree
>>> html = get_html_at_url("http://alexquinn.org")
>>> root = make_etree(html, url)
<Element html at 0x7fc93525d0e8>
>>>
```

To get all of the nodes of a particular tag name (e.g., `<h2>`) you can use `root.find(...)` with a syntax for selecting nodes from HTML documents called XPath. (You can search for more info.)

```
>>> root.find("./h2")
<Element h2 at 0x7fc93525d318>
>>>
```

To iterate through all nodes use this code:

```
>>> for node in root.iter():
...     print(node.tag) # print tag name (e.g., "h2")
```

4. Make a function that, given an `ElementTree`, returns a textual outline of the headings (as a `str`).

```
def make_outline(etree):
```



You only need to outline based on the `h1`, `h2`, and `h3` tags. For each outline level (after `h1`), add four spaces. (Hint: You will use `root.iter(...)` but not `root.find(...)` for this lab.)

For example, if the `ElementTree` was created from a URL with the following HTML...

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Restaurants</title>
</head>
<body>
  <h1>Restaurants</h1>
  <p>...</p>
  <h2>Another Broken Egg</h2>
```

```

    <p>...</p>
    <h3>Address</h3>
    <p>...</p>
    <h2>Café Literato</h2>
    <p>...</p>
    <h3>Address</h3>
    <p>...</p>
</body>
</html>

```

... then your function should return the following:

```

Restaurants
  Another Broken Egg
    Address
  Café Literato
    Address

```

To get the text of a node in an `ElementTree`, use `node.text_content()`.

Note: There are two other ways of accessing text from an `ElementTree` node. `node.text` gets just the text associated with that element, ignoring any child elements. You don't usually want that. `node.itertext()` iterates all segments of text within the element and its descendants. `node.text_content()` is probably implemented using `node.itertext()`. (I did not verify this.)

Strip the string and replace all sections of whitespace with a single space (" "). Hint: Here's a good all-purpose snippet for normalizing whitespace:

```

>>> s = "  one    two three  four  "
>>> print(s)
    one    two three  four
>>> s = " ".join(s.split())    # normalize whitespace
>>> print(s)
one two three four
>>>

```

5. Make a web application with a `root_page()` mapped to "/" (relative URL). See the project description for more on this. You may use that starter code. You will need to create a directory called `template/` (inside your `Lab11/` directory) and a template HTML file called `root.html` in `template/`.

Here is the starter from the project page:

```

#!/usr/local/bin/python3.4
import flask

app = flask.Flask(__name__)

@app.route('/')
def root_page():
    return flask.render_template('root.html')

if __name__ == '__main__':
    app.run(host="127.0.0.1", port=os.environ.get("ECE364_HTTP_PORT", 8000),
            use_reloader=True, use_evalex=False, debug=True, use_debugger=False)
    # Each student has their own port, which is set in an environment variable.

```

```
# When not on ecegrid, the port defaults to 8000. Do not change the host,  
# use_evalex, and use_debugger parameters. They are required for security.  
# Credit: Alex Quinn. Used with permission. Preceding line only.
```

To test, open your browser and go to `http://localhost:12345` (where `12345` is your port). Type `echo $ECE364_HTTP_PORT` in bash to find your port. If you get an error (e.g., connection reset), make sure you have a `templates/` directory and that it contains a file called `root.html`.

6. In the body section of your HTML template, add a form that collects a URL from the user.

You will need a form element. The method attribute should be "GET". Leave the action attribute blank for now.

Inside the `<form>` element, you will need to add a text input:

```
<input type="text" name="url" size="40" value="">
```

You will also need a submit button, also inside your `<form>` element.

```
<input type="submit" value="Submit">
```

See the project description for more on this. You may use that starter code.

If you test your application again (`http://localhost:12345`), it should look like this:

Submit

7. Add an endpoint function called `outline_view()` at `"/outline"` (relative URL), connected to the form, that returns the outline of the page at the URL the user specified.

Your endpoint function should look for the URL the user specified with the name "url".

To get a URL parameter called "url", use this:

```
value = flask.request.args.get("url")
```

That returns the URL parameter called `url`. If no such parameter is not present, then it returns `None`. (An example of a URL with a parameter called "url" is below.)

You will have a function like this. You will use the functions you created in #1-5 to fetch the given URL and print an outline.


```
@route("/outline")  
def outline_view():
```

Normally the browser would compress all whitespace, so your outline wouldn't look right. To tell the browser to preserve whitespace, wrap the text in the `<pre></pre>` tag.

To test, open browse to `http://localhost:12345/outline?url=any URL`.

8. Edit `templates/root.html`. Set the action attribute of the form as shown below.

```
<form method="GET" action="{{ url_for('outline_view') }}">
```

The `{{  }}` will be expanded by the template engine (Jinja) at runtime to the outline URL (`"/outline"`).

9. Test your application by going to your application root page: `http://localhost:12345/`
10. Turn in everything inside your Lab11 directory, including `wpo.py` and `templates/`.