Honors Enrichment Contract
CIS 440: Systems Design and Electronic Commerce
Dr. Joseph Clark
Fall 2014

Prepared by:
Hashim Al-Assi
December 8, 2014

What is Python?

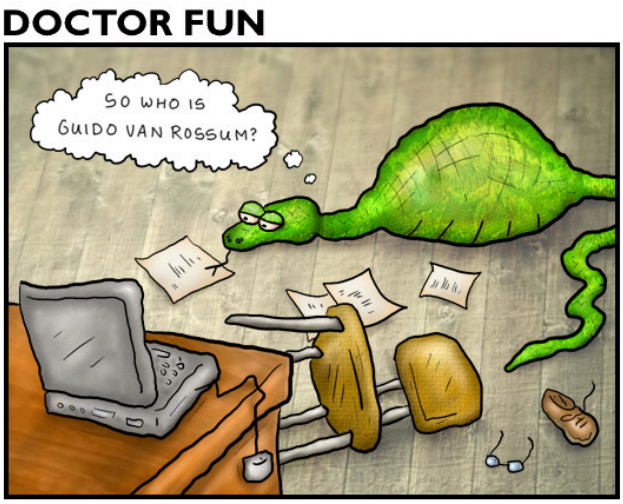
From Python's Executive Summary:

> Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.
>
> Often, programmers fall in love with Python because of the increased productivity it provides. Since there is no compilation step, the edit-test-debug cycle is incredibly fast. Debugging Python programs is easy: a bug or bad input will never cause a segmentation fault. Instead, when the interpreter discovers an error, it raises an exception. When the program doesn't catch the exception, the interpreter prints a stack trace. A source level debugger allows inspection of local and global variables, evaluation of arbitrary expressions, setting breakpoints, stepping through the code a line at a time, and so on. The debugger is written in Python itself, testifying to Python's introspective power. On the other hand, often the quickest way to debug a program is to add a few print statements to the source: the fast edit-test-debug cycle makes this simple approach very effective. (What is Python? Executive Summary)

Origin

The origin of Python is best described by the Foreword for "Programming Python" (1st ed.) where Guido van Rossum writes:

> Over six years ago, in December 1989, I was looking for a "hobby" programming project that would keep me occupied during the week around Christmas. My office (a government-run research lab in Amsterdam) would be closed, but I had a home computer, and not much else on my hands. I decided to write an interpreter for the new scripting language I had been thinking about lately: a descendant of ABC that would appeal to Unix/C hackers. I chose Python as a working title for the project, being in a slightly irreverent mood (and a big fan of Monty Python's Flying Circus). (Foreword for 'Programming Python' 1st ed.)



The Creator

The creator of Python is Guido van Rossum. On his personal home page he has a section devoted to the correct pronunciation of his name. He writes, “in Dutch, the ‘G’ in Guido is a hard G, pronounced roughly like the ‘ch’ in Scottish ‘loch’” (Guido’s Personal Home Page). He also has a sense of humor about it as he added, “However, if you're American, you may also pronounce it as the Italian ‘Guido’. I'm not too worried about the associations with mob assassins that some people have. :-)” (Guido’s Personal Home Page). On Guido van Rossum’s home page he keeps links to his resume, publications list, a brief bio, assorted writings, presentations, and interviews all about Python. He also has links to his new blog, his old blog, his Twitter (@gvanrossum), and his G+ profile. The Python community refers to him as the BDFL or the Benevolent Dictator For Life, which is a title straight from a Monty Python skit (Brief Bio).

From Guido’s Resume:
Guido van Rossum received a Master’s degree in Mathematics and Computer Science from the University of Amsterdam in 1982, and joined CWI as a researcher in the same year. While studying, he worked five years as a systems programmer at Amsterdam’s academic computer center, SARA. While at CWI he worked on both the Amoeba Project, a distributed operating system, and on the design and implementation of ABC, a programming language and environment for programming by non-expert users. From 1991 until 1998 he worked in the multimedia group at CWI, which still does most of their implementation work in Python. He was also a guest researcher at the U.S. national Institute of Standards and Technology (NIST) where most of his time was spent working on Python. In March of 1998 he began working for CNRI doing essentially the same work. For a brief period in 2000 he worked for BeOpen.com as Director of PythonLabs before transferring to Zope Corporation to hold the same position. In July of 2003 he began working for Elemental Security where he stayed for two years. In 2005, Google picked up Van Rossum where he has worked on various projects. At the beginning of last year, Van Rossum started working for Dropbox. While at Google and as part of his current contract at Dropbox, Van Rossum can spend 50% of his work time on Python. (Resume)

The Zen of Python

The vision for Python is best summed up by “The Zen of Python”, which states:

> Beautiful is better than ugly.
>
> Explicit is better than implicit.
>
> Simple is better than complex.
>
> Complex is better than complicated.
>
> Flat is better than nested.
>
> Sparse is better than dense.
>
> Readability counts.
>
> Special cases aren't special enough to break the rules.
>
> Although practicality beats purity.
>
> Errors should never pass silently.
>
> Unless explicitly silenced.
>
> In the face of ambiguity, refuse the temptation to guess.
>
> There should be one-- and preferably only one --obvious way to do it.
>
> Although that way may not be obvious at first unless you're Dutch.
>
> Now is better than never.
>
> Although never is often better than *right* now.
>
> If the implementation is hard to explain, it's a bad idea.
>
> If the implementation is easy to explain, it may be a good idea.
>
> Namespaces are one honking great idea -- let's do more of those! (The Zen of Python)

This philosophy alone drives the way Python is developed.

Versions

There are two main versions of Python in use. Python 2.x is the legacy version and Python 3.x is the "present and future of the language" (Python2orPython3). The final 2.x version was 2.7, which came out in the middle of 2010. Guido van Rossum decided "to clean up Python 2.x properly, with less regard for backwards compatibility than in the case for new releases in the 2.x range" (Python2orPython3). The most drastic improvement is the better Unicode support with all text strings being Unicode by default. Several aspects of the core language have been adjusted to be easier for newcomers to learn and to be more consistent with the rest of the language. Most current Linux distributions and Macs are still using 2.x as default. (Python2orPython3)

Python 3.0 was released in 2008. Python 3.x is under active development and has seen over five years of stable releases. Version 3.3 was released in 2012 and version 3.4 was just released this year. Some of the less disruptive improvements in 3.0 and 3.1 have been backported to 2.6 and 2.7, respectively. However, there are several features that are only available in 3.x releases and won't be backported to the 2.x series. There are also several improvements to the standard library that will not be backported directly to Python 2. (Python2orPython3)

Codecademy still teaches Python version 2.7.3 because "Python3 is not fully backwards compatible with earlier versions and, as a result, hasn't yet seen wide adoption" ("What version of Python do you teach?").

Advantages

There are several advantages of using Python and as Python grows in popularity, the documentation becomes overwhelming. However, some key advantages include:

- Design philosophy emphasizes code readability
- Syntax allows programmers to express concepts in fewer lines of code than other major languages
- Supports multiple programming paradigm (object-oriented, imperative, functional programming)
- Large standard library
- Highly extensible through the use of modules and packages
- Interpreters available on most operating systems
- Open source development

Since some of these are hard to "show", I will focus on readability and syntax by going through some code examples.

Curly Braces vs. Indentation

The first example most beginning programmers should be familiar with is the concept of code blocks. Most languages use curly braces or brackets of some sort as block delimiters. One of the most debated issues in programming is how to style or arrange these curly brackets for maximum readability even though it makes no difference in how the program actually runs.

Ruby, a language similar to Python, uses keywords such as "end" for code blocks instead of brackets. This is common of most non-C derivative languages.

However, Python goes one step further and only uses one keyword at the beginning of the block and line indenting contains the rest. This is because whitespace indentation matters in Python. You can indent the first line as much or as little as you want, but all subsequent code in the block must remain consistent with the first line. Normal convention is to use 4 spaces instead of tabs. This feature is a huge advantage because it reduces ambiguity, which in turn increases efficiency. There is no more wasted time trying to figure out where blocks start and end. This also makes Python well suited for open source or other projects where programmers must collaborate on the same code. (Levin)

In an article titled "Python Programming Language Advantages", the author, Mike Levin, includes some great code comparisons (Levin).

C++ (Placement of curly brackets arbitrary and a matter of style):

```
1. void function functionname(arg)
2. {
3.     some code
4. }
```

…which can also be stated in this and other ways…

```
1. void function functionname(arg) {
2.     some code
3. }
```

Ruby (Uses keyewords instead of curly braces)

```ruby
def functionname(variable)
    some code
end
```

This is still allows matching end confusion to occur, especially with nested functions…

```ruby
def functionname(variable)
    def nestme
        more code
end
    some code
end
```

Python (Only one keyword, but indents matter)

```python
def functionname():
    some code
```

There is only one way to state this in Python. This is profound.

Duck-Typing

Duck-Typing is "the idea that it doesn't actually matter what type the data is, just whether or not you can do what you want with it" (Duck Typing in Python). Python doesn't require declaring and typing variables before using them. The first time a variable is used, it is declared and can continued to be used until you go out of scope. At that point it is destroyed. This obviously helps to make your program shorter and possibly easier to read.

A common drawback to duck-typing is the possibility of letting bugs in. To combat this, Python creates a forced-stop error if you try to do anything ambiguous with a variable.

An example Mike Levin shows is adding a string to an integer.

```python
'Ni'+3 # would fail
```

…because Ni is a string and 3 is an integer, so you might want concatenation or type conversion. But

```python
'Ni'*3 # will succeed
```

…because there is no such ambiguity and this can only mean Ni 3 times:

NiNiNi

If there is more than one possible meaning, Python will not do any auto-conversions. It will just raise a fatal-error so you know to find the mistake.

Lists

The last feature I thought was interesting from Mike Levine's article had to do with lists. He believes Python handles lists very well. He showed a quick example of how easy it is to add together two lists in Python.

I also feel the need to point out that:

```
1. ['Hello World', 'This is list 1'] + ['Hello World 2', 'This is list 2']
```

...becomes:

```
1. ['Hello World', 'This is list 1', 'Hello World 2', 'This is list 2']
```

...and likewise:

```
1. a = ['Hello World', 'This is list 1']
2. b = ['Hello World 2', 'This is list 2']
3. c = a + b
4. print c
```

...will also output:

```
1. ['Hello World', 'This is list 1', 'Hello World 2', 'This is list 2']
```

Yes, adding 2 lists together is that easy.

Other Examples

I also found some other great examples in an article titled, “5 Compelling Reasons to Learn Python as Your First Programming Language”.

A “for” loop on a list [Python]:

```python
items = [1,2,3,4]
for i in items:
    print (i)
```

An “each” call with a block on an array [Ruby]:

```ruby
items = [1,2,3,4]
items.each do |i|
    puts i
end
```

A “while” loop in Python:

```python
x=1
while x <=5:
    print "The number is:" + str(x)
    x += 1
```

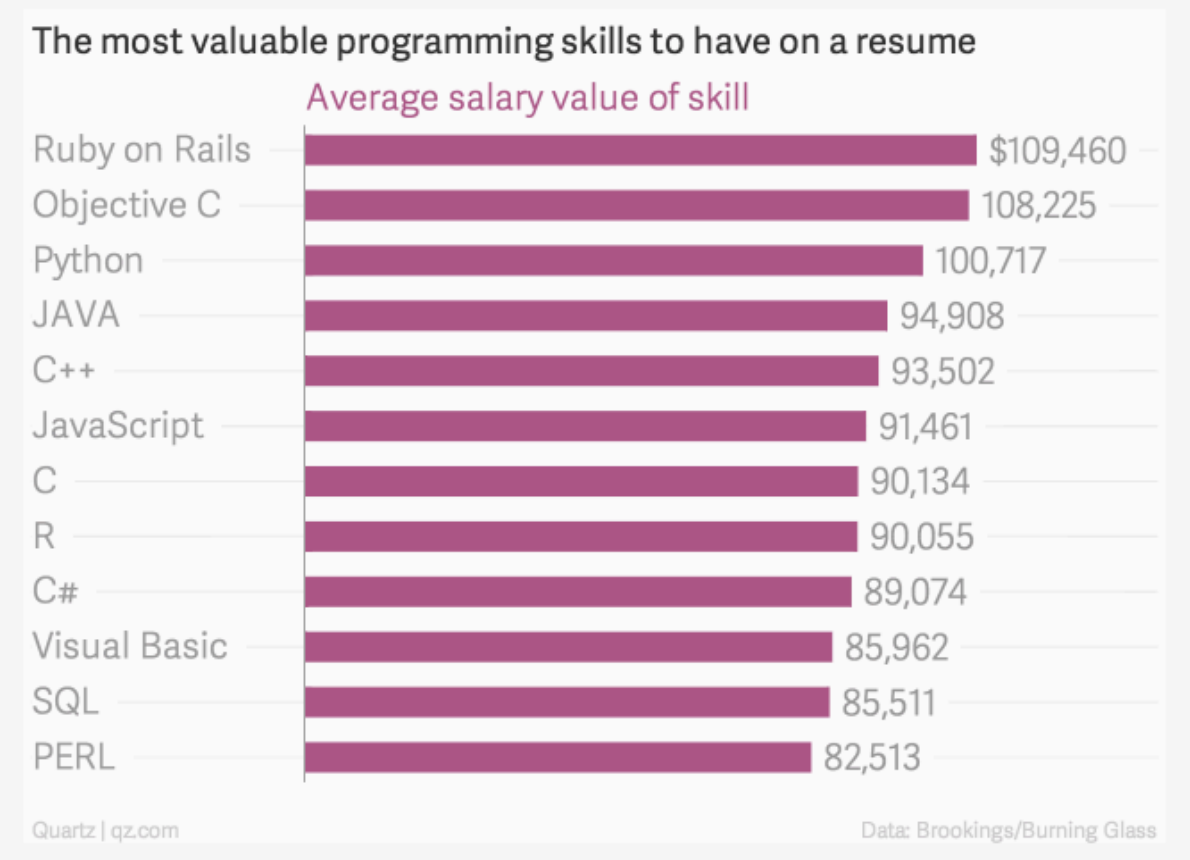
A “while” loop in PHP:

```php
<?php
$x=1;
while($x<=5)
    {
    echo "The number is: $x <br>";
x++;
}
?>
```

Why Should You Care?

Python is rapidly growing in popularity largely because it seems to follow the new trend in programming of simplifying syntax and increasing readability. There have been several articles and posts documenting Python's recent rise.

Quartz (qz.com) just posted an article on November 20, 2014, ranking the programming languages that will earn you the most money. The data was compiled by Burning Glass with Brookings Institution economist Jonathan Rothwell in July. The data was puled from thousands of American job ads. Python is one of the three languages that can help you earn over $100,000 in salary.



Just this past summer there was an article posted on both Javaworld.com and PCWorld.com claiming that Python has surpassed Java as the top learning language. The article by Joab Jackson detailed some interesting statistics:

> Python has surpassed Java as the top language used to introduce U.S. students to programming and computer science, according to a recent survey posted by the Association for Computing Machinery (ACM).
>
> Eight of the top 10 computer science departments now use Python to teach coding, as well as 27 of the top 39 schools, indicating that it is the most popular language for teaching introductory computer science courses.
>
> Python has been growing in popularity in the educational realm for at least the past few years, though this survey is the first to show it has eclipsed Java, which has been the dominant teaching language for the past decade.
>
> A number of universities, however, have switched to Python from Java, and others offer both -- Java for computer science students and Python to teach programming skills for noncomputer science majors.
>
> Python possesses a mix of qualities that makes it a good candidate for universities. It has a simpler syntax than Java or C++, allowing novices to start writing programs almost immediately. At the same time, it can be scaled up for heavy industrial usage -- it is widely used in the financial industry for data analysis, for instance. (Jackson)

Python has also recently won Programming Language of the Year in 2007 and 2010. Programming Language of the Year is awarded by TIOBE, which specializes in assessing and tracking the quality of software. TIOBE uses a community index to determine the popularity of programming languages. They describe their index as:

> The TIOBE Programming Community index is an indicator of the popularity of programming languages. The index is updated once a month. The ratings are based on the number of skilled engineers world-wide, courses and third party vendors. Popular search engines such as Google, Bing, Yahoo!, Wikipedia, Amazon, YouTube and Baidu are used to calculate the ratings. It is important to note that the TIOBE index is not about the *best* programming language or the language in which *most lines of code* have been written. The index can be used to check whether your programming skills are still up to date or to make a strategic decision about what programming language should be adopted when starting to build a new software system. (TIOBE Index for December 2014)

Demo

From *OpenTechSchool's* "Websites with Python Flask":

Setting Up (using Mac OSX)

Mac OSX has Python built in, which means the Terminal can be used as an interpreter.

To check which version of Python is currently installed use the *python* command:

```
hashims-mbp:~ hashimza$ python
Python 2.7.5 (default, Mar  9 2014, 22:15:05)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.0.68)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

The next step is to install any additional packages that may be needed. Python comes with a lot of built-in functionality, but it doesn't always have everything you may need. Python can be extended by adding new modules, which come bundled up in "packages". The *pip* command uses the Python Packages Index to install new packages.

The first package this demo needs is Virtualenv. Virtualenv is very useful if you plan on working on more than one python project at a time. It makes it easy to use different packages and even different versions of packages in each of your Python projects.

To install Virtualenv use the *pip* command again:

```
hashims-mbp:~ hashimza$ pip install virtualenv
Requirement already satisfied (use --upgrade to upgrade): virtualenv in /usr/loc
al/lib/python2.7/site-packages
Cleaning up...
```

If you are using Mac OSX, Virtualenv is usually already installed.

Now that Virtualenv is installed, the next step is to set up a "virtual environment".

To create a new directory use the *mkdi**r*** command.

```
hashims-mbp:~ hashimza$ mkdir Scorpion
```

Use the *cd* command to change to the newly created directory.

```
hashims-mbp:~ hashimza$ cd Scorpion
hashims-mbp:Scorpion hashimza$
```

Once inside the new directory, use virtualenv to create a new virtual environment under the 'scorpionvenv' directory. Since I'm using Python 2.7.5, I made sure to tell the interpreter which version of Python to use.

```
hashims-mbp:Scorpion hashimza$ virtualenv --python=python2 scorpionvenv
```

The next step is to set up the terminal session by using the *activate* command. It is important to remember that the next time you start up a new terminal you will need to repeat this activation step. You will notice "(scorpionvenv)" is now at the beginning of every line to indicate that the virtual environment is active. The *deactivate* command can be used to exit the virtual environment.

```
hashims-mbp:Scorpion hashimza$ source scorpionvenv/bin/activate
(scorpionvenv)hashims-mbp:Scorpion hashimza$
```

The final step in the step up process is to install our framework, Flask. Once again the *pip* command can be used.

```
(scorpionvenv)hashims-mbp:Scorpion hashimza$ pip install flask
```

Hello World!

Now that we have everything setup it is time to begin creating a simple static "Hello World!" website.

The first step is to create a new Python app file using any preferred text editor. I saved my file as "techdemo.py".

Inside the file add the code:

techdemo.py

File Path : ~/Documents/flasktest/techdemo.py

techdemo.py (no symbol selected

```
1  from flask import Flask
2  app = Flask(__name__)
3
4  @app.route('/')
5  def hello_world():
6      return 'Hello World!'
7
8  if __name__ == '__main__':
9      app.run()
```

To better understand this code I will go through each step.

```
from flask import Flask
app = Flask(__name__)
```

The first line imports the Flask library.
The second line creates a new website in a variable called *app*.

```
@app.route('/')
def hello_world():
    return 'Hello World!'
```

The @ is called a "decorator" and it is used to make function definitions stand out. Flask uses *route()* to say that if the browser requests the address /, then the app should *route* that request to the *hello_world* function. The address / is also the default or home address. The *hello_world* function returns the string "Hello World!" which will be sent to the web browser to be displayed.

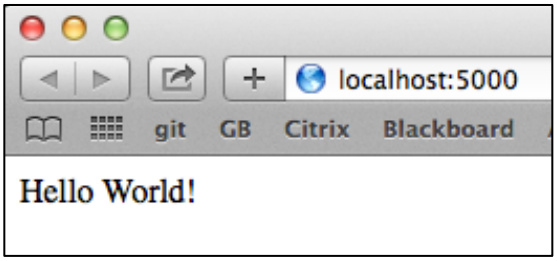
```
if __name__ == '__main__':
    app.run()
```

This says if this script is run directly then start the application.

Now we can run the app in the terminal using the *python* command:

```
(scorpionvenv)hashims-mbp:Scorpion hashimza$ python techdemo.py
 * Running on http://127.0.0.1:5000/
```

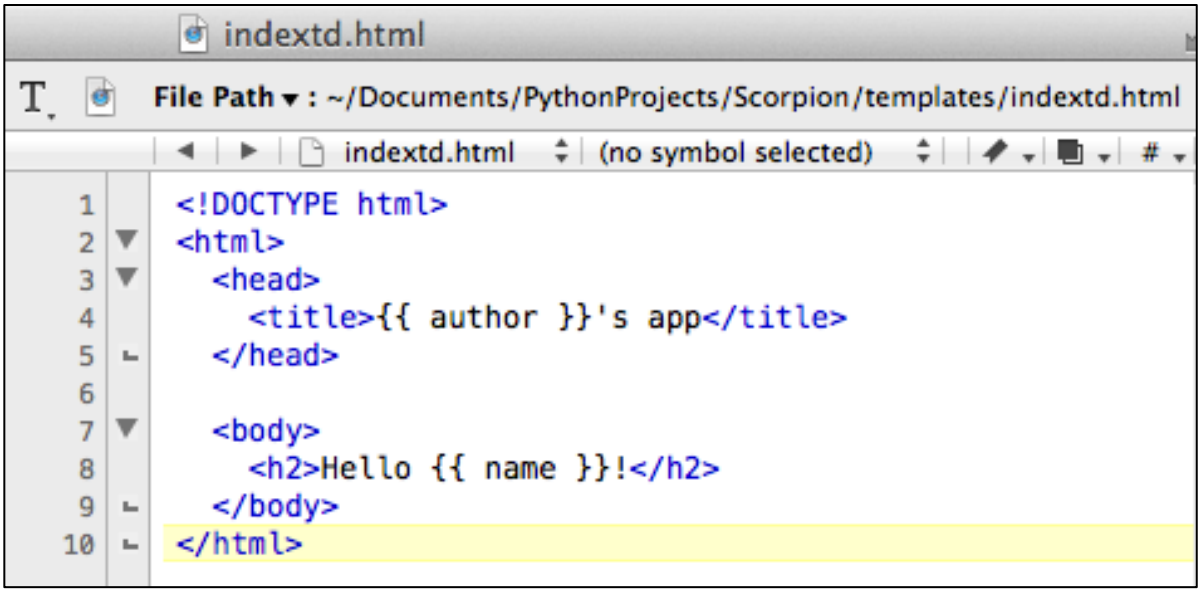
This shows the app is running on 127.0.0.1 (localhost) on port 5000. If you type this into the address on your browser it should bring up the "Hello World!" web site.



Templates

So the page is pretty basic at this point. We can change it by adding some simple HTML. Flask uses a special directory called *templates*, which allows you to put dynamic application data into HTML. This allows you to manipulate the content and display it nicely on any web browser.

We will start by creating a Templates folder and placing it on the same level as our techdemo.py file. Inside of the Templates folder create a file called indextd.html and add the following code:



This should look familiar for the most part, but it does contain two pieces of Flask's templating language. Flask uses double curly braces to show where variables will be displayed. When Flasks displays this template to the browser it will replace {{ author }} and {{ name }} with whatever you assign them in the actual application. This allows the template to separate what the content of the page should be from the actual data that will be used as content. This makes writing dynamic web pages a lot easier.

In order to serve this template we have to modify our flask app. Add the following code:

techdemo.py

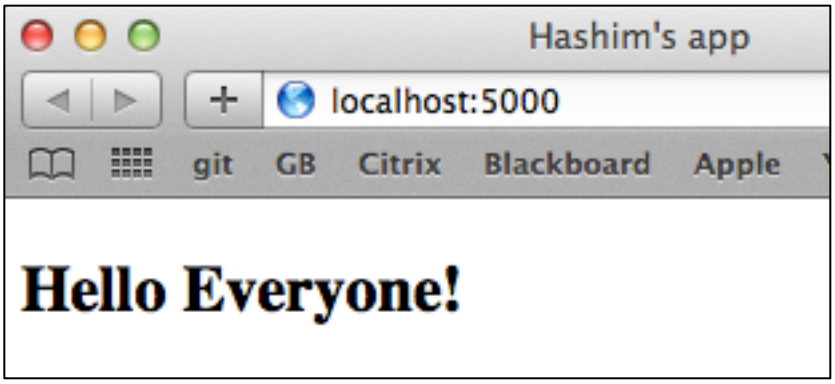
File Path : ~/Documents/PythonProjects/Scorpion/techdemo.py

techdemo.py | hello_world

```
1  from flask import Flask, render_template
2  app = Flask(__name__)
3
4  @app.route('/')
5  def hello_world():
6      author = "Hashim"
7      name = "Everyone"
8      return render_template('indextd.html', author=author, name=name)
9
10 if __name__ == "__main__":
11     app.run()
```

The first bit of code we added was importing a new function from flask called *render_template*.

Next, I assigned my name to the author variable and "Everyone" to the name variable. Instead of returning some predefined text ("Hello World!"), we return the result of calling *render_template*. To return *render_template* we have to give it the file name of the template we wish to use (indextd.html) and pass in the variable names the template should know about (author and name) and what their values should be (in this case they are also author and name). The page should now look like this:

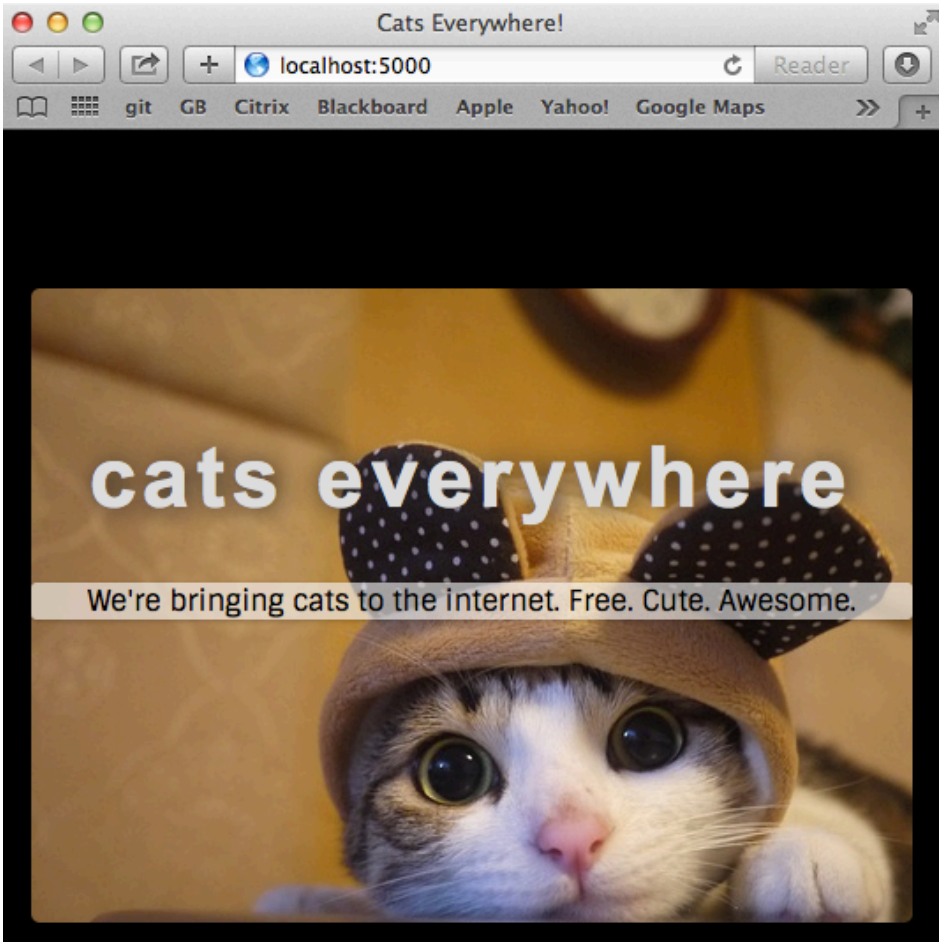


You will notice my name replaces the title and "Everyone" replaces "World".

Part of the tutorial I am following was about learning how to make fun/creative web pages. To help with this they supplied an HTML template to use. The HTML template is very simple, but dramatically changes the page. They also included all of the CSS within a <style> tag so you did not have to create a separate file. If you would like to use the template, add the following code to your indextd.html file:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Cats Everywhere!</title>

    <link     href='http://fonts.googleapis.com/css?family=Sintony:400,700'
        rel='stylesheet' type='text/css'>

    <style type="text/css">
      body
      {
        background-color:#000;
      }

      h1
      {
        font-size:48px;
        margin-top:0;
        font-family:Arial, sans-serif;
        text-shadow:2px 0 15px #292929;
        letter-spacing:4px;
        text-decoration:none;
        color:#DDD;
      }

      #banner
      {
        width:500px;
        height:200px;
        text-align:center;
        background-image:url(http://i.imgur.com/MQHYB.jpg);
        background-repeat:no-repeat;
        border-radius:5px;
        margin:90px auto auto;
        padding:80px 0;
      }


      .lead
      {
        background-color:rgba(255,255,255,0.6);
        border-radius:3px;
        box-shadow:rgba(0,0,0,0.2) 0 1px 3px;
        font-family:Sintony, sans-serif;
      }
    </style>

  </head>

  <body>
    <div id="banner">
      <h1>cats everywhere</h1>
      <p class="lead">We're bringing cats to the internet. Free.     Cute. Awesome.</p>
    </div>
    <div id="emailform">

    </div>
  </body>
</html>
```

The page should now look like this:



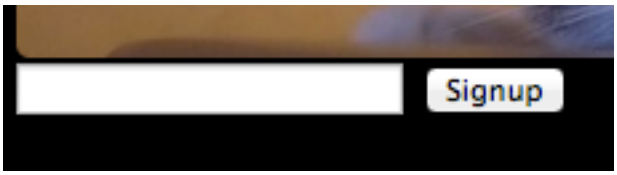
Forms

To add some functionality and turn this basic landing page into a functioning form we need to add some HTML and a new route in our Flask app to accept the data posted by the form.

To add an HTML form to our web page include the following code just above the closing </body> tag:

```
    </div>
        <form action="/signup" method="post">
            <input type="text" name="email"></input>
            <input type="submit" value="Signup"></input>
        </form>
  </body>
</html>
```

This adds a form that will collect an email address.



The action attribute of the form says it will be posted to */signup*. However, we currently do not have any code for this URL. To add this code we need to import two more objects from Flask. We need *request* to get the data and *redirect* to redirect the browser once we are done. This will be added to the top of our Python code.

techdemo.py

File Path : ~/Documents/PythonProjects/Scorpion/techdemo.py

techdemo.py (no symbol selected)

```
1  from flask import request, redirect
2  from flask import Flask, render_template
3  app = Flask(__name__)
```

Next, we need to add a new route for */signup*.

```
@app.route('/signup', methods = ['POST'])
def signup():
    email = request.form['email']
    print("The email address is '" + email + "'")
    return redirect('/')
```

We use the same decorator as before. When the browser requests */signup* it will accept the HTTP POST method, which is mentioned in the HTML form element as *method="post".*

In the *signup* method we retrieve the email address using the *request* object, which contains the form data. In the HTML we used *name="email",* which means that in the request object we can use *request.form["email"].*

When you submit the form, you should be able to see the address you supplied printed out on the terminal.

```
^C(scorpionvenv)hashims-mbp:Scorpion hashimza$ python techdemo.py
 * Running on http://127.0.0.1:5000/
127.0.0.1 - - [10/Dec/2014 03:30:31] "GET / HTTP/1.1" 200 -
The email address is 'halassi@asu.edu'
127.0.0.1 - - [10/Dec/2014 03:30:45] "POST /signup HTTP/1.1" 302 -
127.0.0.1 - - [10/Dec/2014 03:30:45] "GET / HTTP/1.1" 200 -
```

Finally, after printing the email address we still need a response to send back to the web browser. A *redirect* response tells the browser to go to another page and in this case sending it back to the home page ('/') makes the most sense at this time.

The form will be left aligned. To fix this add the code below just before the closing </style> tag:

```
    #form
    {
      text-align:center;
    }
  </style>
```

For this styling to work, you must also add an id to the HTML form element:

```
  </div>
      <form id="form" action="/signup" method="post">
          <input type="text" name="email"></input>
          <input type="submit" value="Signup"></input>
      </form>
</body>
```

Storing Information

The final feature we will add to our web page is the ability to store information temporarily. The information will be stored until the browser closes. We currently don't have anywhere to store the information, but we can change that by adding a list. To create an empty list add the following code right after the app = Flask(__name__) line:

```
3   app = Flask(__name__)
4
5   email_addresses = []
6
```

Next, in the signup() function, we can add the email address to the list by using the name of the list and ".append". Now when you submit the form, it prints out the entire list every time. You must quit out of your previous session using Control C, before the changes will be applied.

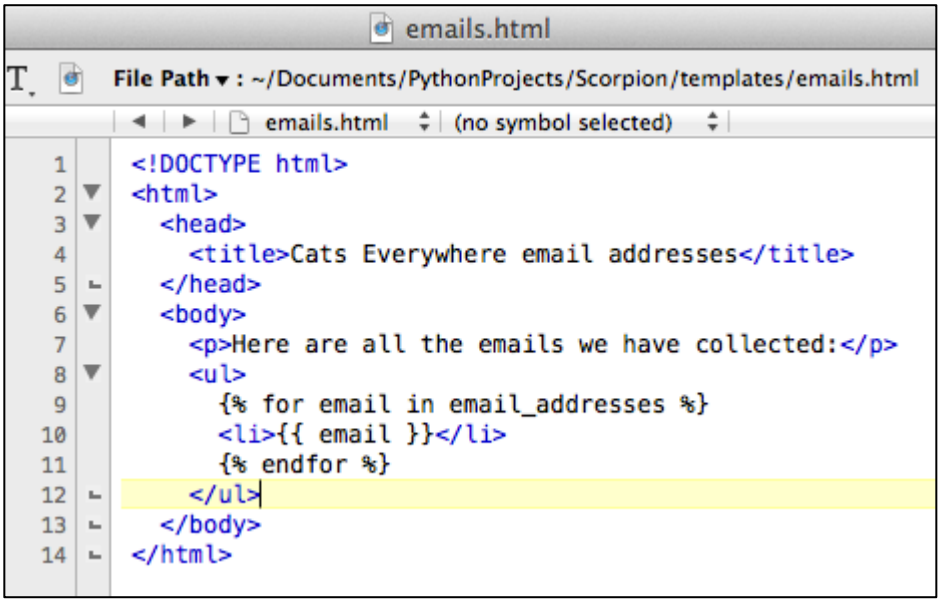
```
^C(scorpionvenv)hashims-mbp:Scorpion hashimza$ python techdemo.py
 * Running on http://127.0.0.1:5000/
[u'halassi@asu.edu']
127.0.0.1 - - [10/Dec/2014 03:49:51] "POST /signup HTTP/1.1" 302 -
127.0.0.1 - - [10/Dec/2014 03:49:51] "GET / HTTP/1.1" 200 -
[u'halassi@asu.edu', u'hashimza@yahoo.com']
127.0.0.1 - - [10/Dec/2014 03:50:11] "POST /signup HTTP/1.1" 302 -
127.0.0.1 - - [10/Dec/2014 03:50:11] "GET / HTTP/1.1" 200 -
[u'halassi@asu.edu', u'hashimza@yahoo.com', u'Joseph.W.Clark@asu.edu']
127.0.0.1 - - [10/Dec/2014 03:50:59] "POST /signup HTTP/1.1" 302 -
127.0.0.1 - - [10/Dec/2014 03:50:59] "GET / HTTP/1.1" 200 -
```

Since it can be a hassle to read the email addresses off the terminal every time you need them, you can create a new route and page to display the addresses nicely for you.

Add the new route using the following code:

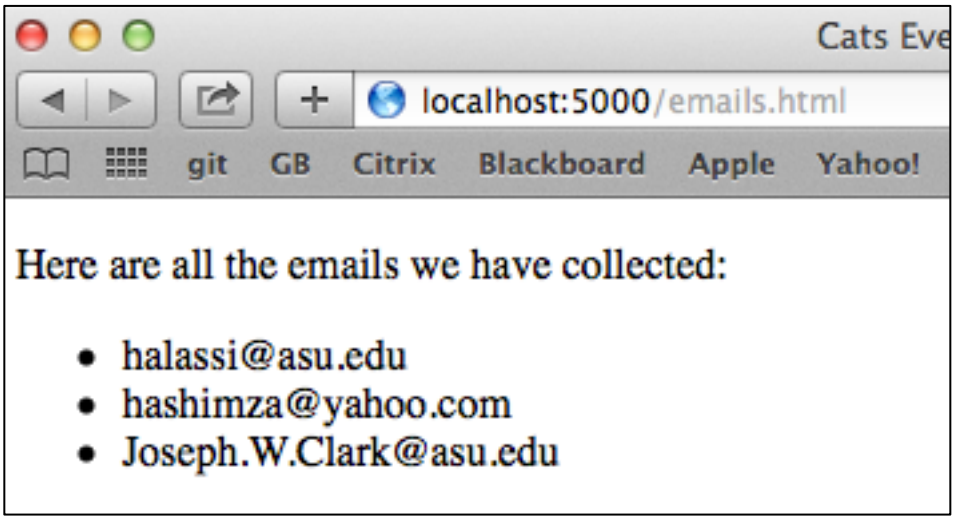
```
@app.route('/emails.html')
def emails():
    return render_template('emails.html', email_addresses=email_addresses)
```

Next, create a new "email.html" page under the Templates folder using:



You will notice that this page uses some more Templating language in the form of a “for” loop. The loop is encased in {% %} and requires an “endfor” statement.

Now visit localhost:5000/emails.html to see the nice list of emails that have been collected.



Remember to reboot your browser session for the changes to take place.

Congratulations! You have successfully completed this demo.

Works Cited

"Duck Typing in Python." *Voidspace*. Web. 09 Dec. 2014. Retrieved from http://www.voidspace.org.uk/python/articles/duck_typing.shtml

"Foreword for 'Programming Python' (1st ed.)." *Python.org*. Web. 09 Dec. 2014. Retrieved from https://www.python.org/doc/essays/foreword/

"Guido van Rossum – Brief Bio." *Python.org*. Web. 09 Dec. 2014. Retrieved from https://www.python.org/~guido/bio.html

"Guido's Personal Home Page." *Python.org*. Web. 09 Dec. 2014. Retrieved from https://www.python.org/~guido/

"Guido van Rossum – Pictures of Me." *Python.org.* Web. 09 Dec. 2014. Retrieved from https://www.python.org/~guido/pics.html

"Guido van Rossum - Resume." *Python.org*. Web. 09 Dec. 2014. Retrieved form https://www.python.org/~guido/Resume.html

Jackson, Joab. "Python bumps off Java as top learning language." *Javaworld.com*. Web. 09 Dec. 2014. Retrieved from http://www.javaworld.com/article/2452940/learn-java/python-bumps-off-java-as-top-learning-language.html

Levin, Mike. "Python Programming Language Advantages." (12 Jan. 2011). *The Daily Journal of a Tech Geek*. Web. 09 Dec. 2014. Retrieved from http://mikelev.in/2011/01/python-programming-language-advantages/

Nisen, Max. "These programming skills will earn you the most money." *Quartz | qz.com.* Web. 09 Dec. 2014. Retrieved from http://qz.com/298635/these-programming-languages-will-earn-you-the-most-money/

"PEP 20 – PEP20 – The Zen of Python." *Python.org.* Web. 09 Dec. 2014. Retrieved from https://www.python.org/dev/peps/pep-0020/

"Python2orPython3." *Python.org*. Web. 09 Dec. 2014. Retrieved from https://wiki.python.org/moin/Python2orPython3

"TIOBE Index for December 2014." *TIOBE.com*. Web. 09 Dec. 2014. Retrieved from http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

Toscano, Nick. "5 Compelling Reasons to Learn Python as Your First Programming Language." (3 Mar 2013). *SkilledUp.com*. Web. 09 Dec. 2014. Retrieved from http://www.skilledup.com/articles/reasons-to-learn-python/

"Websites with Python Flask - Setup." *OpenTechSchool.* Web 09 Dec. 2014. Retrieved from http://opentechschool.github.io/python-flask/core/setup.html

"What is Python? Executive Summary." *Python.org*. Web. 09 Dec. 2014. Retrieved from https://www.python.org/doc/essays/blurb/

"What version of Python do you teach?." *Codecademy.com*. Web. 09 Dec. 2014. Retrieved from http://help.codecademy.com/customer/portal/articles/1403551-what-version-of-python-do-you-teach-