Jake Gadaleta | Programing with a Network in Mind

Introduction

This paper will be taking a Python 3 (3.8.6) point of for all examples unless expressly stated otherwise.

In this paper we will look at a network with 2 separate and unique view points. The first viewpoint will be that of an end user, someone who wants to take information from the internet and use code to be able to preform some task based on that data that has been downloaded. The second person is that of a content creator / server manager / basically someone who wants to host their own content. I have decided to take an in-depth look into both viewpoints independently at first to build up the basics and then combine the 2 together in a final section and an overview of how to build a full product.

I would like to take a quick note and say that the code that I use in the following examples may not be the absolute best ways to solve a given problem instead using the simplest code that I can in order to achieve the explanations.

If you have questions while reading this please feel free to reach out to me and I will do my best to explain jg9902@desales.edu and I will try to get back to you in as timely as a manner as I can.

The End User

This is for the person who wishes to load `.csv` files directly into an analysis tool, or those who want to look through a set of quizlets relating to class in order to get a study sheet for the entire semester, or you want to cycle through wikipedia pages to pull brief definitions on a number of topics. We can even take this as far as to generate our own pseudo-apis to automate the flow of data.

The TL;DR of this is that its called Web Scraping and involves making http connections to download files to be analyzed through code.

Requests

The main package that we use to connect to a site and pull down its contents is called `requests`. Starting in python 3.5 (Sept. 2015) a version of `requests` has been bundled into each release. Prior to the bundling `requests` was consistently the most popular package in the pyPI repo

One of the main reasons that `requests` gained popularity was the simplicity of its usage. The following code snippet outlines how to import and then use requests to pull down a csv stored in a github repo.

```python
import requests

# at this point req will hold all information about the url
req =
requests.get("https://raw.githubusercontent.com/gadzygadz/Networking_Presen
tation/main/example.csv")

# to filter down to just get the text you can look to the text property
text = req.text
```

At the end of this example we text now has whatever text could be pulled from that url in this case it pulls down a bunch of information pertaining to countries.

`requests` is also the way that some apis talk to a server in python a popular one just returns back your public ip address. It has been shown here.

```
>>> requests.get('https://api.ipify.org').text
'147.106.162.61'
```

We can also pull down full websites html, to do this I call very similar code to what I used to grab my IP but it is difficult to read

```
>>> requests.get('https://docs.python.org/3/').text
'\n<!DOCTYPE html>\n\n<html xmlns="http://www.w3.org/1999/xhtml">\n
<head>\n    <meta charset="utf-8" /><title>3.9.0 Documentation</title>\n
<link rel="stylesheet" href="_static/pydoctheme.css" type="text/css" />\n
<link rel="stylesheet" href="_static/pygments.css" type="text/css" />\n
\n    <script id="documentation_options" data-url_root="./"
src="_static/documentation_options.js"></script>\n    <script
src="_static/jquery.js"></script>\n    <script src="_static/underscore.js">
</script>\n...
```

no-one wants to read or parse through that information so we can use a secondary package in order to help us handle just that.

Beautiful Soup (bs4)

Beautiful Soup is a Markup Language (html, xml) parser. When paired with requests we can better pull information out of websites. Because Beautiful Soup doesn't exist in the standard library it must be installed using pip

```
# <python> may have to be swapped out with whatever your python command
actually is
python -m pip install bs4
```

after the installation finishes you are good to use it, to load it in we simply do the following

```
import requests
import bs4
```

so lets say I wanted to load in that same file and pull out all the links. using Beautiful Soup it's really easy.

```python
req = requests.get('https://docs.python.org/3/')
soup = bs4.BeautifulSoup(req.text, "html")

# html stores links using an <a></a> tag
links = soup.find_all('a')

# to see the actual link you need to treat it like a dictionary of json
like objects

for index in links:
    print(index['href'])
```

using these tools in tandem means that we can consistently search through webpages to generate useful data and load them into python themselves in order to do some operations to them.

Other Languages

We went fairly in-depth when it comes to python but this is also possible on most modern programming languages as follows is a few example in some popular programing languages.

```javascript
// Node
require "http"
let request = http.get("https://jake.dev/a.csv", function(response) {
  // csv parser
}
```

```java
// Java
import java.io.BufferedInput;
import java.net.URL;
var csv = new BufferedInput(new URL("https://jake.dev/a.csv"));
```

```csharp
// C#
using System.Net;
var csv = new WebClient("https://jake.dev/a.csv", "my.csv");
```

```rust
// Rust
let mut csv = request::get("https://jake.dev/a.csv");
```

```ruby
# Ruby
require "open-uri"
```

```
open("https://jake.dev/a.csv") do |file|
  # whatever you want to do
end
```

```
# R
library (RCurl)
data <- read.csv(getURL("https://jake.dev/a.csv"))
# naturally can load http files without RCurl
```

Content Creator

This section is for those who want to create and host their own site. In this section we will be discussing theory behind the Model View Controller (MVC) pattern, doing a practical showcase using the `flask` framework.

Model View Controller

At some point in web development you will approach a situation where a login system or IP ban_list system and raw javascript will no longer cut it. MVC comes in as a solution allowing you to embed server side logic before users site can be generated. At times each step of the MVC can be separated into their own files but in the coming examples using `flask` they are all regulated to their own functions.

Model

The model represents the shape of the data. Commonly this is formatted as a object of some kind, this object can often be stored as a "classic" object as defined in true Object Oriented Programming (OOP) or in a dictionary like JSON object. Generally these objects will contain all pertinent information to the page.

View

The view is the what is generated for the user to see. It's primary goal is to format the information in a way that would be understandable to user. It pulls that data that was generated by the model uses that to integrate into itself. Often this will include some "template"

Template

Many web frameworks offer templates that allow you to write pseudo-html that includes logic from the programming language. Some popular ones are `razor` which is used by `.NET` and `jiniga` which `flask` uses

Controller

The controller is where your logic is. You can do any regular programing things that the specific language has the capabilities of doing. Here you can manipulate your model or set up how data is bundled to be moved into the next view.