Weekend: Welcome to Flask; The Internet Virtual Environment

Welcome to Flask

Flask is a web framework that has tools that allow us to create web applications.

It belongs to a category of web frameworks known as a **micro-framework**. This means Flask contains very few dependencies out of the box. It allows the developer to take the driver's seat and have full creative control over the application.

It comes packed with a robust core that is also supplemented by many other third party extensions that can be picked to fit our different needs.

Flask has two main dependencies:

1. **Werkzeug** - This provides flask with _routing_,_debugging_ and _web server_ capabilities.
2. **Jinja 2** - This provides template support for flask.

Web Clients and Servers

You're about to learn your first Python web framework! Before we begin, let's address several language independent concepts that apply to the internet in general.

In this lesson, we will walk through details of how the web works; From your browser, known as the **client**, to a **web server** and back again.

IP Addresses and the Internet

First, let's talk about the **Internet**.

When we use the term Internet, we refer to millions of networks that are connected to allow devices (computers, laptops, tablets, phones, etc) to communicate with one another, and access content.

Similar to the address of your home, the device you use to browse the Internet also has an address called an **Internet Protocol** address, or **IP address** for short. This address is assigned by an **Internet Service Provider (ISP)**.

Similarly, the server for a website you visit also has an IP address. IP addresses are represented with a set of four groups of numbers separated by periods, like this:

_198.185.159.145_

You may also see an additional number after an IP Address that's separated from the others by a colon, like this:

_198.185.159.145:43_

This additional number is a **port** number. A port number is like an apartment number or box number after your mailing or home address. It indicates an even more specific location information will travel to at the particular address.

Communicating Between Addresses

To recap, we've covered two addresses so far: Your device's address and a website's address. Next, let's walk through how your address and the website's address interact when you visit a website on your device.

Client-Server Process

Requests

First, in order to access a website you must make a request from a client. The client is usually a web browser, like Chrome, Firefox, Safari, or Internet Explorer. Essentially,anything that facilitates your interaction with the web and web addresses. And making a request usually looks like entering a web address in the browser's _URL bar_, clicking a link, or submitting a form. Anything that triggers a navigational change to a new location on the internet.

When you create a request, the client constructs a message (similar to mailing something from one address to another). This message must be formatted using a strict set of rules, or protocol, known as **Hypertext Transfer Protocol (HTTP)**.

Part of this message includes the web address that you are sending your request to: For example, when we click a link to visit this area of the website, the web address _http://moringaschool.com/_ is included.

Continuing with the metaphor of mailing a letter, Hypertext Transfer Protocol (HTTP) is similar to the formatting rules of addressing an envelope. You can't place the address just anywhere, or format it in any manner you please. There is a protocol for what information is required (street address, city, state, zip), the format it is written in, and where these pieces of information are placed on the envelope. Deviation from this protocol will result in your message, or "letter", not reaching its intended destination.

HTTP has similar standards. It requires certain kinds of information, placed and formatted in a specific manner in order for the client to reach its destination. For instance, when the request message for http://moringaschool.com/ leaves the client, the ISP is the first stop on the round trip to collect the Moringa School's web content from the server where it is stored.

Before the location of the server can be found, the IP address must be identified using the web address provided in the client's request message. This is the job of the **Domain Naming System** or **DNS servers**.

There are hundreds of DNS servers that house databases with the single purpose of resolving web addresses to IP addresses. In our example, a DNS server matches _http://moringaschool.com/_ to the IP address: _198.54.116.26_. Once resolved to an IP address,

your request message is sent to its identified destination, 198.54.116.26, or, the server that contains our Moringa School's website.

The web server receives the message and determines the resource that is being located or acted upon in the request. In our example, the request to view _http://moringaschool.com/_ will gather the resource for displaying the main Courses page. This resource is an an HTML document. Depending on the website, other resources may need to be gathered too, such as images, stylesheets, scripts, videos, etc.

Responses

When the content is collected, the server returns it in a **response** back to the client. The response is interpreted by the client and you see the _http://moringaschool.com/_ main page displayed in your browser window.

When you interact again with the site, by clicking a link, or navigating to a different area, the whole process begins again.

Although you're probably most familiar with making web requests through your web browser, know that any program can actually be a web client. For instance, a program installed on your computer that automatically checks for and downloads updates, an app running on your tablet, or even one web server making a request to another web server are all clients. Each of these must use the same HTTP protocol.

Terminology

- **Client**: A computer program that sends a request to another program for data or services (e.g., a web browser is a client that sends requests to a web server).
- **Server or Web Server**: A computer program or device that uses HTTP to distribute information to a client.
- **Internet Protocol (IP) Address**: A set of numbers separated by periods that uniquely identifies a device on the Internet.
- **Port number**: Part of the addressing information used to identify the sender/receiver of requests at an IP address (e.g., 43 is the port number with this IP address: 198.185.159.145:43).
- **Internet Service Provider (ISP)**: A company that provides access to the Internet and assigns an IP address to the connecting device.
- **Hypertext Transfer Protocol (HTTP)**: A protocol that defines how requests are formatted, transmitted and processed between web clients and web servers.
- **Domain Naming System (DNS) Servers**: The "address book" of the Internet; servers that maintain all domain names and translate them to Internet Protocol (IP) addresses.
- **Web Resource**: The target of a web address; Examples of web resources include files, documents, images, videos, stylesheets, scripts.

Additional Resources

To further explore any of these concepts, visit the following articles linked in this lesson:

- **Web Server (https://en.wikipedia.org/wiki/Client%E2%80%93server_model)**
- **Client (https://en.wikipedia.org/wiki/Client_(computing))**
- **IP Address (https://en.wikipedia.org/wiki/IP_address)**
- **Internet Service Provider (ISP) (https://en.wikipedia.org/wiki/Internet_service_provider)**
- **Hypertext Transfer Protocol (HTTP) (https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol)**
- **Domain Naming System (DNS) (https://en.wikipedia.org/wiki/Domain_Name_System)**

Uniform Resource Locator (URL)

As we discussed in the previous lesson, when you navigate to http://www.moringaschool.com/, your browser makes a request to a web server. One of the most important parts of that request is the address you're visiting. That address is technically called the **URL**, short for **Uniform Resource Locator**, or **URI**, short for **Uniform Resource Identifier**. (The difference is largely semantics.) In this lesson, we'll explore the URL in detail.

Required URL elements

Some elements must be included in the URL for a request to be successful. Here is an example of a URL.

```
http://www.moringaschool.com/about
```

Let us break it down to see each individual element.

1. Scheme

The **scheme** defines how the client locates or acts upon the requested resource. In our example, the client will make a request using the _http://_ scheme. Other schemes include _ftp_, _mailto_, and _file_. The scheme is always followed by a colon; for the http scheme, a colon and two slashes (://).

2. Host

The **host** provides the details of where the requested resource is located. The host in a URL is either the domain name for the IP address or the IP address itself. In our example, the host is _www.moringaschool.com_.

3.Path

The path consists of one or more segments separated by slashes and provides the name for identifying the resource requested. In our example, the path indicates the resource: _/about_. This directs to the _About_ page of the Moringa school website.

Optional URL elements

Not all elements in a URL are required. Here are two that are optional.

1. ? Query

There may be times where the URL contents must provide additional details for a resource to be identified.

For example, if we use the search functionality on _https://www.python.org_, the server will need to know the input that we enter in order to know what search results to return. When we enter "python3" as our search parameter and press "Search", the request is made to this URL: _https://www.python.org/search/?q=python3_

After the path in the URL, we see a `?` symbol which indicates the beginning of a **query string**.

You'll see our search parameter has a **key** of _search_ and a **value** of _internet_. Query parameters are separated by the `&` symbol when there is more than one.

With this information, the server will use the value to query the database for only the lessons that have the term "internet" in them.

2. # Fragment

Another optional URL element is called a fragment.

Here is an example of a URL containing a fragment:

https://docs.python.org/3.6/tutorial/#the-python-tutorial

Fragments begin with a `#` symbol and contain information that's typically processed by the client, rather than the server. In this example, "the-python-tutorial" is the id of a div in the HTML document that is returned. The HTML page is extremely long but by adding the fragment, the browser will position the display of the page precisely at that element when displayed.

If you remove the fragment, the position returns to the default top of the document.

Terminology

- **Uniform Resource Identifier (URI)/Uniform Resource Locator (URL)**: The web address which specifies the location of the requested web resources.
- **Scheme**: The part of the URL that indicates the protocol to be used in communication (e.g. http://).
- **Host**: The part of the URL that contains the domain name.
- **Path**: The part of the URL that contains the resource name.
- **Query string**: An optional part of a URL that contains parameters for querying a database; begins with the `?` symbol; often used in a search request.

- **Fragment**: An optional part of a URL that contains details for where the browser should display the information (e.g. on a particular div).

Additional Resources

Uniform Resource Locator (https://en.wikipedia.org/wiki/Uniform_Resource_Locator)

HTTP Requests

Let's take a closer look at HTTP. In this lesson, we'll examine the organization and data of HTTP requests.

HTTP request-response cycle

Client HTTP Requests

When the client triggers a new HTTP request, the request includes four elements that will be sent to the server:

1. Method
2. Path
3. Headers
4. Body

HTTP Method

HTTP methods, also known as _verbs_, indicate the kind of action that the client is requesting be done in the server. The two most frequently used methods are the ones that are supported by all browsers: **GET** and **POST**. A request with a GET method indicates that a resource is being _retrieved_. A request with a POST method indicates that the resource is being _changed_ - _possibly added_, _updated_ or _deleted_.

An example of a GET request would be when we navigate to the Moringa School immersive page. The client requests the resource at the URL, _http://www.moringaschool.com/_ and the / resource(s) is retrieved.

An example of a POST request would be when we click submit on a form to add our email to a mailing list. The message would post data to the server by adding our information to the mailing list resource.

Path

The path identifies the web resource that should be retrieved (GET) or acted upon (POST). In our example, _http://www.moringaschool.com/_, the resource _/immersive_ is known as the path.

Headers

Request headers are part of the request message protocol and provide the server with more information about the client, the server and the request.

Here are some examples of header fields that are included in a request message:

- Host: This is the host or domain name of the server. In our example: www.moringaschool.com.
- User-agent: This is the software acting on behalf of the user to make the request; in our example, this would be our browser (Chrome, Firefox, etc)
- Accept-language: This includes the human languages that are acceptable for the response.

Body

The body contains data beyond the contents of the URL and headers that needs to be transmitted to the server. For example, if a user submits a form with data, the resulting POST request would need to include all of the data inputted so the server could store the information. This data is delivered in the body of the request.

Terminology

- **HTTP Method**: The kind of action that the client is requesting to be done in the web server, also known as a verb. Most frequently used HTTP methods: GET and POST.
- **GET**: A request method that retrieves information from the server but does not change anything on the server. Example: request to see a homepage for a site.
- **POST**: A request method that acts upon the resource by adding, updating or deleting information on the server. Example: submitting a form to join a mailing list which adds your name to the list.
- **HTTP Request Header**: The first lines of an HTTP request message that include information about the client, server and the request.
- **HTTP Request Body**: Data that needs to be transmitted to the server in the HTTP request message (like data from a submitted form).

Additional Resources

List of all HTTP Header Fields (https://en.wikipedia.org/wiki/List_of_HTTP_header_fields)

HTTP Responses

After a server receives an HTTP request, it will respond with a message that consists of the following three elements:

1. Status
2. Headers
3. Body

Status

The first line of a response message is the status which includes a status code and reason phrase. The HTTP status code is a three-digit number that indicates how the request was processed (or not). Each code is also accompanied by a reason phrase which is not read by the client but is intended for humans to provide a brief description of the status code.

Below is a list of the five status code classes determined by the first digit of the code. This lesson shows the most common codes. To see a complete list, visit HTTP Status Codes.

1xx Informational

Status codes beginning with 1 indicate the request was received. This class of code is rarely used.

2xx Success

Status codes beginning with 2 indicate the request was received and handled successfully.

Examples

- **200**: OK
- **201**: Created

3xx Redirection

Status codes beginning with 3 indicate that additional action is required to complete processing the request.

Examples

- **301**: Moved permanently
- **302**: Moved temporarily

4xx Client errors

Status codes beginning with 4 indicate that something was wrong about the request.

Examples

- **400**: Bad Request (the request used invalid syntax)
- **401**: Unauthorized (you have to log in)
- **403**: Forbidden (you're logged in, but not allowed to make this request)
- **404**: Not Found
- **422**: Unprocessable (the request had valid syntax but the instructions it contained were invalid)

5xx Server errors

Status codes beginning with 5 indicate that something went wrong on the server side, such as a bug in the code or a server that went down.

Examples

- **500**: Internal Server Error (usually a bug in the server code)
- **502**: Bad Gateway (the server sent the request to another server and got an invalid response)

- **503**: Service Unavailable (the server is overloaded or down for maintenance)

Headers

Like request headers, response headers include additional protocol providing more details about the HTTP response.

Some examples of response header fields include:

- Server: The name of the server where the response is coming from.
- Content-Language: The human language(s) of the content.
- Content-Length: How long the response body is in 8-bit bytes.

Body

The response body includes all of the content for the resource requested. You'll often hear it referred to as the **payload**. When we issue a GET request for _http://www.moringaschool.com/_, we expect the body to include an HTML document with the table of contents. This resource may need additional resources such as images, CSS, script files, etc. For each resource, a new request is made by the client and a new response is issued for the resource from the server.

Once there is a response to all of the requests, the / resource is displayed in the browser.

Terminology

- **Status code**: First line of the response message from the server consisting of a three-digit number indicating the status of the request. Example: 200 indicates that the request was successfully processed.
- **Status reason**: The human language interpretation of the status code, not read by the client but intended for humans.
- **Payload**: The data that was requested in the original request message that is not protocol. Example: the payload for a request for the main page of a website would be the actual HTML document content.

Overview

Status Codes **1xx** Informational

2xx Success

3xx Redirection

4xx Client errors

5xx Server errors

Additional Resources Complete List of HTTP Status Codes may be found **here (https://developer.mozilla.org/en-US/docs/Web/HTTP/Response_codes)** .

Setting up Flask

The most convenient way to install Flask is using a **virtual environment**.

A virtual environment is a private copy of the Python interpreter that allows us to install packages privately. This means all the third party modules will only be available in that environment. This is very useful because it prevents the packages from cluttering our system and also prevents conflicts.

Creating a Virtual Environment.

Let us create a simple `Hello World` application.

Create a new directory and name it *Hello-Flask*. Inside it let us create a new virtual environment for our application so that we can install Flask.

Python3.6 comes with a tool called `venv` that allows us to create a virtual environments.

```
$ python3.6 -m venv --without-pip virtual
```

Above we used the tool to create a virtual environment and called it `virtual`. We can give our virtual environment any name provided it is easy to remember.

Activating and Deactivating the virtual environment

To activate the virtual environment we run the following command:

```
$ source virtual/bin/activate
```

We know the virtual environment is active when the terminal responds with the name of your virtual environment before your computer's name. For the name `virtual`, the terminal would return this:

```
(virtual) james@james-P43SJ:~/Desktop/exerciseFiles/Hello-Flask$
```

Next, we are going to download the latest version of pip in virtual our environment. We will be using pip to install flask and any other packages that we will need.

```
$ curl https://bootstrap.pypa.io/get-pip.py | python
```

To deactivate we simply use the `deactivate` keyword

```
(virtual) james@james-P43SJ:~/Desktop/exerciseFiles/Hello-Flask $ deactivate

james@james-P43SJ:~/Desktop/exerciseFiles/Hello-Flask $
```

Installing Flask

Let us first get back to our virtual environment with the `source virtual/bin/activate` command.

```
$ source virtual/bin/activate
```

Now, in the virtual environment, we will install Flask to the project using the following command:

```
$ pip install flask
```

Here we use the pip tool to install Flask to our project. In the next lesson, we will create a basic Flask application that displays `Hello World` to the browser.