HTTP Protocol:

- HTTP protocol is the foundation of data communication in world wide web. Different methods of data retrieval from specified URL are defined in this protocol.
- Here is a table of HTTP Methods:

HTTP Method	Function
Get	Sends data in unencrypted form to the server. Most common method.
Head	Same as GET, but without a response body.
Post	Used to send HTML form data to a server. Data received by the POST method is not cached by server.
Put	Replaces all current representations of the target resource with the uploaded content.
Delete	Removes all current representations of the target resource given by a URL.

- By default, the Flask route responds to the GET requests.

Starting a Flask app:

- A basic Flask app is structured as follows:

```
/home/my-flask-app/
|___ app.py
|___ templates/
     |___ base.html
|___ static/
     |___ style.css
```

- **/home/my-flask-app** would be the directory of your Flask application.
- **app.py** is your main file and is what runs the app. In order to start your application, you can run python app.py or python3 app.py depending on your Python version. This will open up your application at http://localhost:5000.
- The **templates/** directory contains all the HTML templates for what you will serve to the browser (i.e. what the user will see). Flask expects HTML files to be in a folder called templates/ so ensure your folder is named appropriately. If this is named improperly, render_template() will not work
- The **static/** directory contains all of the static, non-changing elements of the site (i.e. such as the CSS / JavaScript). If you choose to write CSS, you would place it in this folder.

- Your basic barebones app.py file will looks as follows:

```
# required imports.
# You may add more in the future; currently, we will only use
# render_template
from flask import Flask, render_template

# tells Flask that "this" is the current running app
app = Flask(__name__)

# setup the default route
# this is the page the site will load by default (i.e. like the home page)
@app.route('/')
def root():
    # tells Flask to render the HTML page called index.html
    return render_template('index.html')

# run the app when app.py is run
if __name__ == '__main__':
    app.run()
```

Getting Input From a Form:

- Flask can send data to app.py from the browser using an HTTP request. In order to receive data from the web page, Flask receives a GET request from a **<form></form>** block whenever the user presses the corresponding **<input type="submit"></input>** button.
  A block in a webpage that can send input to Flask looks as follows:

```
<form action="/add-comment">
        <input type="text" name="comment-input" placeholder="enter a new
        comment">
        <input type="submit" name="comment-submit" value="submit">
</form>
```

- **NOTE:** The GET request is not supposed to be used to modify data (this is a violation of the standard). Don't worry about this for now but in a later lab we will improve this example by refactoring this.
- Flask automatically sends a GET request to the endpoint provided in the action attribute inside the **<form></form>** tag. We can catch this particular endpoint in app.py by specifying it using **app.route('/add-comment')** or substituting **/add-comment** for any other endpoint you may decide to use. To get the input the user has submitted, we make use of a module of Flask called requests. Requests searches for any **<input></input>** tags inside the corresponding **<form></form>** tags and captures these values. To access them, in app.py we can using **request.args.get('comment-input')** as follows:

```
@app.route('/add-comment')
def add_comment():
    global comments
    comments.append(request.args.get('comment-input')
    ...
```

The **request.args.get()** command returns the value inside the input that matches the name attribute inside the **<input></input>** tags. Notice how comment-input matches the corresponding value in the HTML code. Additionally, notice how the variable comments is referred to using the global keyword. This is Python syntax for creating a global variable for the comments outside of the add_comment() function. Generally, this practice of using global variables is not recommended but we will refactor this later when we'll use an SQLite database instead of a global variable.

- To use send data from Flask to the browser, we can pass in additional parameters to the render_template() function. Consider the sample code:

```
@app.route('/')
def root():
    return render_template('home.html',data=posts, title="abbas")
```

The browser is now able to access these variables using the keys they were given.