Bootstrap and React for Web Development

L A Liggett

2020-01-05

Contents

Chapter 1

Introduction

Chapter 2

HTML

2.1 HTML Properties

Commenting in HTML.

```
<!--
These are some comments.
-->
```

The head tag allows metadata to be labeled, the text of title for instance is typically listed in the tab or the status bar of the page in a browser.

```
<!DOCTYPE html>
<html>
    <head>
        <title>
            My Web Page!
        </title>
    </head>
</html>
```

The body specifies text for the page body.

```
<!DOCTYPE html>
<html>
    <body>
        Hello, world!
    </body>
</html>
```

Headings specifies header text of increasingly small sizes.

```
<!DOCTYPE html>
<html>
    <body>
        <h1>This is the largest headline</h1>
        <h2>This is also a large headline</h2>
        <h3>This is a slightly smaller headline</h3>
        <h4>This is an even smaller headline</h4>
        <h5>This is the second-smallest headline</h5>
        <h6>This is the smallest headline</h6>
    </body>
</html>
```

Unordered lists specify bullet points.

```
<!DOCTYPE html>
<html>
    <body>
        An Unordered List:
        <ul>
            <li>One Item</li>
            <li>Another Item</li>
            <li>Yet Another Item</li>
        </ul>
    </body>
</html>
```

Ordered lists number lines in increasing order.

```
<!DOCTYPE html>
<html>
    <body>
        An Ordered List:
        <ol>
            <li>First Item</li>
            <li>Second Item</li>
            <li>Another Item Here</li>
            <li>Fourth Item</li>
        </ol>
    </body>
</html>
```

The image tag refers to and inserts an image as an html attribute. The alt gives alternative code if the image is missing. The height and width sets the image size in number of pixels. When the image size is set to 50% sets the image size dynamically to 50% of the browser width or height.

```
<!DOCTYPE html>
<html>
```

```
    <body>
        <img src="cat.jpg" alt="cat" width="300" height="200">
        <img src="cat.jpg" alt="cat" width="50%" >
    </body>
</html>
```

Tables display data in a table format that can be styled in various ways. The `th` tag specifies the headings of each of the columns. The `td` tag specifies the data in each of the columns.

```
<!DOCTYPE html>
<html>
    <body>
        <table>
            <tr>
                <th>First Name</th>
                <th>Last Name</th>
                <th>Years in Office</th>
            </tr>
            <tr>
                <td>George</td>
                <td>Washington</td>
                <td>1789-1797</td>
            </tr>
            <tr>
                <td>John</td>
                <td>Adams</td>
                <td>1797-1801</td>
            </tr>
            <tr>
                <td>Thomas</td>
                <td>Jefferson</td>
                <td>1801-1809</td>
            </tr>
        </table>
    </body>
</html>
```

Tables can be styled within the header of the html document. Both the `th` and the `td` styles are defined together. `border-collapse` combines the borders of cells together.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Presidents</title>
        <style>
```

```
            table {
                border: 2px solid black;
                border-collapse: collapse;
                width: 50%;
            }

            th, td {
                border: 1px solid black;
                padding: 5px;
                text-align: center;
            }

            th {
                background-color: lightgray;
            }
        </style>
    </head>
    <body>
        <table>
            <tr>
                <th>First Name</th>
                <th>Last Name</th>
                <th>Years in Office</th>
            </tr>
            <tr>
                <td>George</td>
                <td>Washington</td>
                <td>1789-1797</td>
            </tr>
        </table>
    </body>
</html>
```

Forms can be created and labeled as such. The `placeholder` text is what is written within the form before anything is entered into it. The `name` is similar to a variable name and can be used to refer to the form and the data that is entered into it. The text within the button is the text that will appear on the button in the page.

```
<!DOCTYPE html>
<html>
    <body>
        <form>
            <input type="text" placeholder="Full Name" name="name">
            <button>Submit!</button>
        </form>
```

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

Text can be aligned and colord by specifying styles within the respective tags of text.

```
<!DOCTYPE html>
<html>
    <body>
        <h1 style="color:red;text-align:center;">Welcome to My Web Page!</h1>
        <h1 style="color:#4290f5;text-align:center;">Second heading</h1>
    </body>
</html>
```

Style elements can be separated from the actual body of the webpage. In this example every `h1` is styled within the style portion of the header.

```
<!DOCTYPE html>
<html>
    <head>
        <title>My Web Page!</title>
        <style>
            h1 {
                color: red;
                text-align: center;
            }
        </style>
    </head>
    <body>
        <h1>Welcome to My Web Page!</h1>
    </body>
</html>
```

Links to local pages or hyperlinks are included within the `<a>` tag.

```
<a href="about.html">About</a>
<a href="http://www.google.com">Google</a>
```

Links can also refer to locations on the same page.

```
<a href="section1">Section 1</a>
<h1 id="section1">Some stuff.</h1>
```

A newline can be inserted within the body of text by using `<br />`.

```
<a href="about.html">About</a><br />
```

2.1.1 Forms

Generic text input fields can be created with the text type.

```
<div>
<input name="name" type="text" placeholder="Name">
</div>
```

A password field is pretty similar to a text field, but the characters are obscured.

```
<input name="password" type="password" placeholder="Password">
```

Dropdown lists of the possible valid choices for a field can be used with `datalist`.

```
<input name="country" list="countries" placeholder="Country">
<datalist id="countries">
    <option value="Afghanistan">
    <option value="Albania">
    <option value="Algeria">
```

2.2 CSS

Commenting in CSS.

```
/*
These are some comments.
*/
```

CSS properties can be found here.

Instead of putting the css styles within the header of the html file, they can be included in a separate css file and referenced. In this example, the type of file being referenced is classified as a `stylesheet` and the code is within `styles.css`.

```
<!DOCTYPE html>
<html>
    <head>
        <link rel="stylesheet" href="styles.css">
    </head>
</html>
```

The code that goes within the css file is here, and it is simply the same code that was put into the style headers in the above example.

```
h1 {
    color: blue;
    text-align: center;
}
```

Divisions define sections of the code that can be separated so it can be controlled in a particular manner. Font priorities are taken left to right if some fonts are not found.

```
<!DOCTYPE html>
<html>
    <head>
        <title>My Web Page!</title>
        <style>
            div {
                background-color: teal;
                width: 500px;
                height: 400px;
                margin: 30px;
                padding: 20px;
                font-family: Arial, sans-serif;
                font-size: 28px;
                font-weight: bold;
                border: 1px dotted black;
            }
        </style>
    </head>
    <body>
        <div>
            Hello, world!
        </div>
    </body>
</html>
```

Divisions and spans can be named and used to refer to different parts of the html document specifically.

```
<!DOCTYPE html>
<html>
    <head>
        <title>My Web Page!</title>
        <style>
            #top {
                font-size: 36px;
                color: red;
            }
```

```
            .name {
                font-weight: bold;
                color: blue;
            }
        </style>
    </head>
    <body>
        <div id="top">
            This is the <span class="name">top</span> of my web page.
        </div>
    </body>
</html>
```

Link styling can be done by adjusting colors and text decorations of links.

```
<style>
    a:link {
      color: blue;
      background-color: transparent;
      text-decoration: none;
    }

    a:visited {
      color: red;
      background-color: transparent;
      text-decoration: none;
    }

    a:hover {
      color: pink;
      background-color: transparent;
      text-decoration: underline;
    }

    a:active {
      color: orange;
      background-color: transparent;
      text-decoration: underline;
    }
</style>
```

Fonts can be imported from locations like google's hosted fonts and used directly to avoid problems with a browser not supporting them. The link to the fonts goes within the header portion of the html code.

```
<head>
    <link href="https://fonts.googleapis.com/css?family=Cormorant+Garamond|Proza+Libre&display=sw
</head>
```

The imported font families can then be used within the CSS directly.

```
body {
    font-family: Proza Libre, Cormorant Garamond
}
```

Nested elements can be styled in a grouped manner.

```
<style>
    ol li {
        color: red;
    }
</style>
<body>
    <ol>
        <li>list item</li>
        <li>second list item</li>
    </ol>
</body>
```

A similar use is to style the immediately nested child elements and none other using the > operator.

```
<style>
    ol > li {
        color: red;
    }
</style>
<body>
    <ol>
        <li>this will be colored</li>
        <ul>
            <li>this won't be colored</li>
        </ul>
        <li>this will also be colored</li>
    </ol>
</body>
```

Fields of particular types can be styled based on their type. These are examples of a text field that allows letters and numbers and a number field that only allows numbers.
The fields can then be styled based on the type of field that they are.

```
<style>
    input[type=text] {
    background-color: red;
}
</style>

<body>
    <input name="name" type="text" placeholder="First Name">
    <input name="name" type="number" placeholder="Age">
</body>
```

2.2.1 Selectors

CSS Selectors allow specific classes or elements to be selected and styled individually.

```
a, b /* Multiple element selector */
a b /* Descendant selector */
a > b /* Child selector */
a + b /* Adjacent sibling selector */
[a=b] /* Attribute selector */
a:b /* Pseudoclass selector */
a::b /* Pseudoelement selector */
```

Pseudo-classes allow for different styling effects depending on the state of the element.

```
<style>
    button {
        background-color: green;
    }
    button:hover {
        background-color: orange;
    }
</style>
<body>
    <button>Click</button>
</body>
```

Pseudo-elements are similar but select elements also allow things to be styled by placing information at the beginning of an item.
What is happening here is that there is a link amd before the link it says “Click here:”, and to the left of that the `\21d12` specifies a unicode arrow.

```
<style>
    a::before {
        content: "\21d2 Click here: ";
```

```
        font-weight: bold;
    }
</style>
<body>
    <a href="#">A link</a>
</body>
```

Text highlighting can also be controlled with pseudo-elements. Here the text color turns red and the highlight is in yellow when the text is highlighted.

```
<style>
    p::selection {
        color: red;
        background-color: yellow;
    }
</style>
<body>
    <p>This is some text</p>
</body>
```

2.2.2 Responsive Design

Media queries are CSS rules that are only used if certain properties are true. A commonly used property is screen size to adjust page layouts fore mobile. It is generally a good idea to design for mobile first and adjust properties to fit desktop, as this will ensure mobile gets the fastest performace.

Here the width of a column is being altered if the browser window is at least 768px in size. Altering the design in this way illustrates how development can be done "mobile-first", as the property is altered if a desktop is used instead of mobile.

```
@media only screen and (min-width: 768px) {
  /* For desktop: */
  .col-1 {width: 8.33%;}
}
```

Media queries can also control content what content gets printed. Here both paragraphs get displayed on the page, but only the first paragraph appears when the page is printed.

```
<style>
    @media print {
        .screen-only {
            display: none;
        }
    }
</style>
```

```
<body>
    <p>This gets printed</p>
    <p class="screen-only">This does not get printed</p>
</body>
```

The content of the page can also be changed using media queries.

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<style>
    @media (min-width: 500px) {
        h1::before {
            content: "Welcome to My Web Page!";
        }
    }

    @media (max-width: 499px) {
        h1::before {
            content: "Welcome!";
        }
    }
</style>
<body>
    <h1></h1>
</body>
```

2.2.3 Flexbox

Flexbox styling allows elements to be dynamically arranged to fit the screen. With a wide enough screen the div elements will all be in one row, but as the screen shrinks, they will move down into new rows.

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<style>
    .container {
        display: flex;
        flex-wrap: wrap;
    }

    .container > div {
        background-color: springgreen;
    }
</style>
<body>
    <div class="container">
        <div>Some stuff</div>
        <div>Some stuff</div>
```

```
        <div>Some stuff</div>
    </div>
</body>
```

2.2.4 Grid Styling

In the following example a grid system is being used for the items being displayed. the grid creates the first two colums as 200px and the last column is automatically sized to fill the rest of the remaining screen.

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<style>
    .grid {
        background-color: green;
        display: grid;
        grid-column-gap: 20px;
        gird-row-gap: 10px;
        grid-template-columns: 200px 200px auto;
    }

    .grid-item {
        background-color: white;
    }
</style>
<body>
    <div class="grid">
        <div class="grid-item">1</div>
        <div class="grid-item">2</div>
        <div class="grid-item">3</div>
    </div>
</body>
```

Chapter 3

Bootstrap

3.1 Setup

The bootstrap stylesheet `<link>` can be used directly `stackpath.com` by including the following reference in the `<head>` before any other listed stylesheets.

```
<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.mi
```

Here is an alternative link to the bootstrap CSS file.

```
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.cs
```

The `charset` and `viewport` meta tags are often required for proper bootstrap responsive behaviors, and should be included when using the bootstrap css. The `viewport` line is a responsive meta tag that ensures proper rendering and touch zooming for mobile devices.

```
<!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
```

3.2 Columns

Bootstrap styles a page as 12 columns. In the following code, columns that are 3/12 columns wide are used.

```
<style>
</style>
<body>
    <div class="container">
        <div class="row">
            <div class="col=3">
```

```
                This is stuff
            </div>
            <div class="col=3">
                This is stuff
            </div>
            <div class="col=3">
                This is stuff
            </div>
        </div>
    </div>
</body>
```

Bootstrap can also style elements to take different amounts of the 12 total columns depending on the screen size. What is happening in the following code is that the columns being listed take 3/12 columns if the screen is large, as defined by the bootstrap CSS, and they take 6/12 columns if the screen is small.

```
<style>
    .row > div{
        padding: 20px;
        background-color: teal;
        border: 2px solid black;
    }
</style>
<body>
    <div class="container">
        <div class="row">
            <div class="col-lg-3 col-sm-6">
                This is a section.
            </div>
            <div class="col-lg-3 col-sm-6">
                This is a section.
            </div>
            <div class="col-lg-3 col-sm-6">
                This is a section.
            </div>
        </div>
    </div>
</body>
```

Chapter 4

Sass

4.1 Intro

Sass can be installed with npm.

```
npm install -g sass
```

Sass is an extension to CSS that adds functionality, including the addition of variables. Sass converts its code into CSS that can then be used by the browser. Below is an example of setting a Sass variable `$color`.

```
$color: blue;

ul {
    font-size:14px;
    color: $color;
}
```

A Sass file is then converted to CSS by running Sass and specifying the output.

```
sass variables.scss variables.css
```

CSS files can be automatically recompiled if any changes are detected using Sass. Here the `variables.scss` file is being monitored for changes and recompiled to `variables.css` whenever changes are detected.

```
sass --watch variables.scss:variables.css
```

Github pages actually will automatically compile scss files into css files when a scss file is comitted to a github repository.

4.2 Nesting

Styles can be applied to divisions or items within other divisions when using sass. In the below example, the code within a sass file will style only those paragraphs that are nested within a `div` blue and only the `ul` wihin `div` as green. Anything outside of a `div` will not be styled by this scss code.

```
div {
    font-size: 18px;

    p {
        color: blue;
    }

    ul {
        color: green;
    }
}
```

4.3 Inheritance

Sass uses inheritance to create generic governing rules that can then be extended by other elements. This can be useful for similar elements that share a number of properties but then have a couple of different properties.
In the following example, a `%message` group is created with a number of different styles, then `.success` extends `%message` and thereby inherits all of the included styles but then also has a green background.

```
%message {
    font-family: sans-serif;
    font-size: 18px;
    font-weight: bold;
    border: 1px solid black;
    padding: 20px;
    margin: 20px;
}

.success {
    @extend %message;
    background-color: green;
}
```

Chapter 5

Flask

5.1 Intro

There seems to be a nice full flask tutorial here where a blog is created.

Installation can be done through anaconda.

```
conda install -c conda-forge flask
```

Flask code is generally stored within a file called `application.py`. Below is a general framework of the flask code that resides in this file.
Line 3 is creating a new flask web application.
Flask applications are designed around routes. On line 5, what is happening is that the code is referring to navigation to the `/` or home directory. The two lines below this route give the code of what to do when a user navigates to that home directory.

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def index():
    return "Hello, world!"
```

Flask then needs to be told how to import it, by setting the `FLASK_APP` environment variable. If the above code is put into a file called `application.py`, just export that filename. It may be useful if `application.py` is always going to be the main filename to just add this code to the bashrc.

```
export FLASK_APP=application.py
```

If flask is run in debug mode, an app will update anytime a change is made to

the underlying code.

```
export FLASK_ENV=development
```

The web application can then be run by running a flask webserver.

```
flask run
```

5.2 Routes

Instead of just using the main route, new routes can be created that when navigated to can have different functionality. In the following example, the route can be accessed at this address: `http://127.0.0.1:5000/david`

```
@app.route("/david")
def david():
    return "Hello, David"
```

The route can take the URL addess information dynamically and use it as a variable. In the following example, the text in the URL is being read in as a `string`, and the value is being assigned to the `name` variable.

```
@app.route("/<string:name>")
def hello(name):
    name = name.capitalize()
    return "Hello, {}!".format(name)
```

HTML code can also be included in the python code and can be returned and interpreted as HTML. In the following example the text that is getting returned and displayed is being styled as header text on the webpage.

```
@app.route("/<string:name>")
def hello(name):
    name = name.capitalize()
    return "<h1>Hello, {}!</h1>".format(name)
```

Routes can be referred to by a link which can also pass variables.

```
<a href='{{ url_for('details', title=book.title, author=book.author) }}', method='GET'
```

The variables that get passed can then be used in `application.py` and passed to another html file.

```
@app.route('/details/<string:title>/<string:author>', methods=['GET', 'POST'])
def details(title, author):
    return render_template('details.html', title=title, author=author)
```

5.3 Templates

Instead of embedding HTML within values that get returned by python code, HTML files themselves can be served by the python code. Flask can look for HTML files and it will look in a subdirectory of the main directory called `templates`. So in the following example if there is an HTML file called `index.html` within a subdirectory called `templates`, that html file gets served up by the following code.

```
@app.route("/")
def index():
    return render_template("index.html")
```

Variables can also be passed from python to the html files that exist within the `templates` directory in order to dynamically alter the HTML content.
In the following code, the `headline` variable is defined in python and then getting passed to HTML, and it is common that the varible names are just kept the same, though in the return function the second `headline` refers to the python variable and the first to the html variable just as is python methods.

```
@app.route("/")
def index():
    headline = "Hello thar"
    return render_template("index.html", headline=headline)
```

Now this `headline` variable can be used in HTML like so. The language being used is Jinja code. This functionality can be helpful in using the very same HTML code but allow it to perform differently.

```
<body>
    <h1>{{ headline }}</h1>
</body>
```

5.4 Jinja

Jinja brings programming logic gates to HTML. The code must be put into an html file within the `templates` directory and used by flask as shown in the above templates section.

```
<body>
    {% if new_year %}
        <h1>Happy New Year!</h1>
    {% else %}
        <h1>Go back to work</h1>
    {% endif %}
</body>
```

Here is an example of a simple counter.

```
{% for i in p %}
    {{ loop.index }}
{% endfor %}
```

For loops work pretty similar to python as well. In the following code `names` is a python list getting passed from `application.py`, and the list items are rendered as a `ul` in HTML.

```
<ul>
    {% for name in names %}
        <li>{{ name }}</li>
    {% endfor %}
</ul>
```

Routes can be referred to by the name of a method within them from HTML. If the following code exists within the `application.py` file:

```
@app.route("/more")
def more():
    return render_template("more.html")
```

The above method `more()` just refers to another html file, and this html file can be used from html like so, where the jinja function `url_for` finds the route that contains the `more()` method, and then uses that URL as the link in the `href`.

```
<a href="{{ url_for('more') }}">See more...</a>
```

5.5 User Handling

This seems like a decent guide about user authentication and handling.

Chapter 6

SQL

6.1 Intro

SQL stands for structured query language that is designed to facilitate accessing data that is structured into table form. PostgreSQL is a version of SQL that is used in these notes.

6.2 Installation

The method I can easily get to work is by installing through `apt`.
It seems also that for some reason the install only works if the version number is specified.

```
sudo apt-get install postgresql-9.6 postgresql-client libpq-dev
```

conda install -c conda-forge psycopg2 flask-sqlalchemy flask-migrate

Both the default database user and default database are called postgres, so switch to that user and start `psql`.

```
sudo -u postgres psql
```

Then start a server.

```
psql
```

I have not been able to get the installation working through a conda install but here is the reference to postgresql.

```
conda install -c conda-forge postgresql
```

To quit `psql` run `\q`.

6.3 Starting Local Database

First set a password for local use.

```
sudo -u postgres psql
postgres=# \password
Enter new password:
Enter it again:
postgres=# \q
```

Initially work with localhost to create a database.

```
psql -U postgres -W  -h localhost
create database airplanes;
\q
```

Export the location to the newly created database and connect or connect directly.

```
export DATABASE_URL="postgresql://localhost/airplanes";
```

or

```
psql -U postgres -W  -h localhost postgres://localhost/airplanes
```

6.4 Heroku

Install Heroku.

```
sudo snap install --classic heroku
```

Then running any command should prompt to login to Heroku but the `login` command is specific for this.

```
heroku login
```

A sample app from heroku can be cloned to test app deployment.

```
git clone https://github.com/heroku/python-getting-started.git
cd python-getting-started
```

From within the app directory, create the app on heroku. This will create a new app with a random name, but a name can instead be passed to manually set the app name.

```
heroku create
```

The app code can then be committed to the heroku server.

```
git push heroku master
```

Pushing the code will deploy the app, but an instance of the app must be then be run.

```
heroku ps:scale web=1
```

The app can now be visited on the web.

```
heroku open
```

Logs can be viewed of page requests here in a way that will constantly update with each new request.

```
heroku logs --tail
```

The `Procfile` is a file within the root directory of the application that declares the command that should be executed when the app is started. Typically it contains the following.

```
web: gunicorn gettingstarted.wsgi --log-file -
```

6.4.1 Heroku Dynos

Heroku has some introduction to using postgresql with a deployed app. And some more extensive information as well. It also looks like CS50 has some instructions as well.

By default heroku apps are deployed to a single dyno which is like a container that runs the commands specified in the `Procfile`. Using a free account will allow a single dyno to be used that will sleep after a half an hour of inactivity. The number of dyno containers can be scaled up and down. If scaled to zero, no containers will run the app.

```
heroku ps:scale web=0
```

The number of free dyno hours is set to 550 per month and with a verified credit card on file will be set to 1000. The remaining time available for a dyno to be running for an app can be checked.

```
heroku ps -a <app-name>
```

To run the heroku test app locally first setup Django to use local assets.

```
python manage.py collectstatic
```

The app can then be run locally at `http://localhost:5000`.

```
heroku local web
```

Changes can be committed in a similar manner as is done with github.

```
git add .
git commit -m 'message'
git push heroku master
```

6.4.2 Heroku Databases

The current addons including databases available to an app can be viewed.

```
heroku addons
```

The `DATABASE_URL` can be displayed.

```
heroku config
```

More extensive information about the app can also be displayed, includeing the postgres version (PG Version)

```
heroku pg
```

As long as postgresql is installed locally, it is possible to connect to the remote database.

```
heroku pg:psql
```

It is also possible to connect with a database that has just been initiated, where the URI is listed in the `Config Vars` section of the Settings, under `DATABASE_URL`.

```
psql <URI>
```

6.5 Database Hosting With Heroku

1. Navigate to https://www.heroku.com/, and create an account if you don't already have one.
2. On Heroku's Dashboard, click "New" and choose "Create new app."
3. Give your app a name, and click "Create app."
4. On your app's "Overview" page, click the "Configure Add-ons" button.
5. In the "Add-ons" section of the page, type in and select "Heroku Postgres."
6. Choose the "Hobby Dev - Free" plan, which will give you access to a free PostgreSQL database that will support up to 10,000 rows of data. Click "Provision."
7. Now, click the "Heroku Postgres :: Database" link.
8. You should now be on your database's overview page. Click on "Settings", and then "View Credentials." This is the information you'll need to log into your database. You can access the database via Adminer, filling in the server (the "Host" in the credentials list), your username (the "User"), your password, and the name of the database, all of which you can find on the Heroku credentials page.
9. Use the URI found in the Settings page of Credentials to link to the database using `export DATABASE_url="URI_URL"`

6.6 Creating a Database

Note that reserved words are all being capitalized below, but this is not required, but rather a stylistic choice.

To create a new database, first run `psql`.

```
create database airplanes;
```

Refer to that database.

```
export DATABASE_URL="postgresql://localhost/airplanes";
```

List current databases.

```
\list
```

Connect to a new database;

```
\connect airplanes
or
\c airplanes
```

Below is the general syntax to create a PostgreSQL database.
The `id` is being used in a manner similar to an index that will just number each of the items.
The next three lines are all different data, and are of types `VARCHAR` and `INTEGER`.
The `NOT NULL` aspect will cause the server to reject the entry if some data is added to the database but that value is not included.

```
CREATE TABLE flights (
    id SERIAL PRIMARY KEY,
    origin VARCHAR NOT NULL,
    destination VARCHAR NOT NULL,
    duration INTEGER NOT NULL
);
```

Display the currently created databases.

```
\d
```

Insert data into the flights database.

```
INSERT INTO flights (origin, destination, duration) VALUES ('New York', 'London', 415);
```

Select all the data from flights.

```
SELECT * FROM flights;
```

Select only the origin and destination columns from flights.

```
SELECT origin, destination FROM flights;
```

Select only the data in flights where the id is 3.

```
SELECT * FROM flights WHERE id = 3;
```

Select only the data in flights that have an origin of New York.

```
SELECT * FROM flights WHERE origin = 'New York';
```

Boolean logic data selection from flights.

```
SELECT * FROM flights WHERE destination = 'Paris' AND duration > 500;
```

Average a column of data.

```
SELECT AVG(duration) FROM flights;
```

Combine boolean logic with calculations performed on the data.

```
SELECT AVG(duration) FROM flights WHERE origin = 'New York';
```

Count columns of matching data.

```
SELECT COUNT(*) FROM flights;
```

Select matching data from a list.

```
SELECT * FROM flights WHERE origin IN ('New York', 'Lima');
```

Wildcards can be used to search for substrings.

```
SELECT * FROM flights WHERE origin LIKE '%a%';
```

SQL uses `LIMIT` to function like `HEAD`.

```
SELECT * FROM flights LIMIT 2;
```

Order data in ascending order `asc` is ascending and `desc` is for descending.

```
select * from flights order by duration asc;
```

Data can be grouped together as in pandas.
The following code selects the origin column, counts the number of identical origin entries and then adds this count to a new count column.

```
select origin, count(*) from flights group by origin;
```

Grouped data counts can be immedately filtered when searching.

```
SELECT origin, COUNT(*) FROM flights GROUP BY origin HAVING COUNT(*) > 1;
```

6.7 Updating Data

This will change the duration of the flight from New York to London to 430.

```
UPDATE flights
SET duration = 430
WHERE origin = 'New York'
AND destination = 'London';
```

Delete a set of matching data from a database.

```
DELETE FROM flights
WHERE destination = 'Tokyo';
```

6.8 Join/Merge

Inner join is the default join and will only include data that is matched.

Foreign keys can be used to reference the data within another table so that data does not need to be continually repeated. For instance if a number of flights are all heading to New York, the city id could be set as number 1 and then flight destination ids could be just set to 1.
As an example, the following table can be created that has passenger names, and in the `flight_id` column, the `flights` database is reference to get the actual flight locations.
When one database references another as in the following code, the referenced column is typically the `PRIMARY KEY`, thought this can be modified if necessary.

```
CREATE TABLE passengers(
    id SERIAL PRIMARY KEY,
    name VARCHAR NOT NULL,
    flight_id INTEGER REFERENCES flights);
```

It might be helpful to add some data to the above database for testing, so here is some data.

```
INSERT INTO passengers (name, flight_id) VALUES ('Alice', 1);
INSERT INTO passengers (name, flight_id) VALUES ('Bob', 1);
INSERT INTO passengers (name, flight_id) VALUES ('Charlie', 2);
INSERT INTO passengers (name, flight_id) VALUES ('Dave', 2);
INSERT INTO passengers (name, flight_id) VALUES ('Erin', 4);
INSERT INTO passengers (name, flight_id) VALUES ('Frank', 6);
INSERT INTO passengers (name, flight_id) VALUES ('Grace', 6);
```

In order to reference the id data from the `flights` database for instance and match that up with the data found in the `passengers` database, the databases should can be joined on the id columns that correspond with each other, in an analogous manner as is performed in pandas.
The following code selects the `origin` and `destination` columns from the `flights` database, and the `name` column from the `passengers` database, after the two databases have been joined on their respective flight id columns.

```
SELECT origin, destination, name FROM flights JOIN passengers ON passengers.flight_id =
```

Data can be directly selected when merging two databases together like the following selection of on the data corresponding with the passenger named `Alice`.

```
SELECT origin, destination, name FROM flights JOIN passengers ON passengers.flight_id =
```

Left/right joining of databases can be performed that unlike inner join will include all of the data in the left or right database.

```
SELECT origin, destination, name FROM flights left JOIN passengers ON passengers.flight
```

6.9 Indexing

Indexing allows a subset of one database (sort of like a pandas series) to be used to select data from a database (analogous to a DataFrame).
In the following code, within the parentheses the `flight_id` column is being selected to create a "series", and the ids are being grouped by `passengers`, and only those flights have more than one passenger are included within the "series". This series of flight ids is then used to filter the original `flights` "DataFrame".

```
select * from flights where id in (select flight_id from passengers group by flight_id
```

6.10 SQL Injection

User input should be escaped or sanitized to prevent a user from inputting SQL code into input fields and have it be directly executed as SQL.
As an example if a username and password field are presented to the user, and the input is checked against a SQL database, it might look something like the following.

```
SELECT * FROM users WHERE (username = 'Bill') AND (password = '12345')
```

The problem with the above code is that SQL code could theoretically be entered into the fields like the following.
If `1', OR '1' = '1'` is entered into the password field, now the SQL query would looks something like the following, which evaluates as True and may allow somebody access to the database.

```
SELECT * FROM users WHERE (username = 'Bill') AND (password = '1', OR '1' = '1')
```

6.11 Transactions and Race Conditions

This refers to a challenge if multiple requests simultaneously are made to the same data within a database and an attempt to modify the database occurs. The problem is that if the requests conflict in some manner the database can

be improperly altered. One solution to this is transactions, where one user essentially checks-out the database and can modify it, and the second user can only modify the database after the first user as completed interacting with the database.

6.12 SQLAlchemy

SQLAlchemy is a python library that can be used to interact with SQL data. The library can be installed like this:

```
conda install -c conda-forge sqlalchemy sqlalchemy-utils
```

SQLAlchemy can be used to interface with an SQL database.

```
from sqlalchemy import create_engine
engine = create_engine(os.getenv("DATABASE_URL"))
```

Users should also be permitted to interact with the SQL database individually within sessions. To do this, a scoped session variable can be created and used to individually interact with the SQL database.

```
from sqlalchemy.orm import scoped_session, sessionmaker
db = scoped_session(sessionmaker(bind=engine))

# this is an example then of interacting with the database
flights = db.execute("SELECT origin, destination, duration FROM flights").fetchall()
for flight in flights:
    print(f"{flight.origin} to {flight.destination}, {flight.duration} minutes.")
```

CSV files can be used to store the information for a database, and inserted into an SQL database.
Assuming `flights.csv` looks something like this:

```
Paris,New York,540
Tokyo,Shanghai,185
Seoul,Mexico City,825
Mexico City,Lima,350
Hong Kong,Shanghai,130
```

The following code will then read `flights.csv` and add in the flights to the database.
`:origin`, `:destination`, and `:duration` define placeholders within the database into which data will be inserted. This placeholder syntax also provides a layer of safety as it helps to prevent users from entering SQL code into a field and directly running SQL.

```
def main():
    f = open("flights.csv")
    reader = csv.reader(f)
```

```
    for origin, destination, duration in reader:
        db.execute("INSERT INTO flights (origin, destination, duration) VALUES (:origi
                    {"origin": origin, "destination": destination, "duration": duratio
        print(f"Added flight from {origin} to {destination} lasting {duration} minutes
    db.commit()
```

Here are some links that may be helpful in setting up an sql database and using it with SQLAlchemy. RealPython Location of Postgresql Database Location of Postgresql Database CS50 Project1 Help Reddit CS50 SQLAlchemy Help

6.13 Object-Relational Mapping

Python classes can be used to control the function of a SQL database. Typically a Python class can be used to control the functionality in a single SQL database.

Below is an example of a Python class that controls some of the function of an SQL database.
`__tablename__` refers to the `flights` sql database.
`db.Model` defines some interactivity with SQLAlchemy and the SQL database.
`primary_key=True` indicates that this column is used as the primary way to identify the data.
`db.ForeignKey("flights.id")` references the `flights.id` column within the `Flight` class.

```
from flask_sqlalchemy import SQLAlchemy

db = SQLAlchemy()

class Flight(db.Model):
    __tablename__ = "flights"
    id = db.Column(db.Integer, primary_key=True)
    origin = db.Column(db.String, nullable=False)
    destination = db.Column(db.String, nullable=False)
    duration = db.Column(db.Integer, nullable=False)
    passengers = db.relationship("Passenger", backref="flight", lazy=True)

class Passenger(db.Model):
    __tablename__ = "passengers"
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String, nullable=False)
    flight_id = db.Column(db.Integer, db.ForeignKey("flights.id"), nullable=False)
```

Creating a class in the above manner will allow modification of an SQL database using just python with SQLAlchemy. This avoids the requirement of using SQL commands explicitly within the code.
The following is an example of adding flight data to a database using the Flight class.

```
db.init_app(app)
flight = Flight(origin=origin, destination=destination, duration=duration)
db.session.add(flight)
db.session.commit()
```

A SQL database can also be queried with python instead of SQL commands.
The first command below is how a SQL database can be queried with SQL syntax and the second is with python using SQLAlchemy.

```
# using SQL
flights = db.execute("SELECT origin, destination, duration FROM flights").fetchall()

# using SQLAlchemy
flights = Flight.query.all()
```

Filtering can also be done with similar code.

```
flights = Flight.query.filter_by(origin="Paris").first()
```

Results can be counted.

```
flights = Flight.query.filter_by(origin="Paris").count()
```

Data can be retrieved by its ID, where the below code gets the row with ID 28.

```
Flight.query.get(28)
```

Data within a table can be modified.
The following code gets the row with an ID of 6 then updates its duration to 280.

```
flight = Flight.query.get(6)
flight.duration = 280
```

Data can be deleted from a table.

```
flight = Flight.query.get(6)
db.session.delete(flight)
```

Data from a table can be retrieved and ordered.

```
Flight.query.order_by(Flight.origin.desc()).all()
```

Table data can be retrieved when it does not match a boolean expression.

```
Flight.query.filter(
    Flight.origin != "Paris").all()
```

SQL substring querying can be performed with SQLAlchemy.

```
Flight.query.filter(
    Flight.origin.like("%a%")).all()
```

SQL search can be performed using a list of values.

```
Flight.query.filter(
    Flight.origin.in_(
        ["Tokyo", "Paris"])).all()
```

Boolean expressions can be used.

```
Flight.query.filter(
    or_(Flight.origin == "Paris",
        Flight.duration > 500)).all()
```

Tables can be joined together.

```
db.session.query(Flight, Passenger).filter(
    Flight.id == Passenger.flight_id).all()
```

6.14 APIs

API stands for application programming interface.
JSON stands for Javascript object notation.

There are a number of different HTTP request types.
GET: retrieve a resource
POST: create a new resource
PUT: replace a resource
PATCH: update a resource
DELETE: delete a resource

The python library `requests` enables requesting from python. Below is a get request which grabs the html code from google.com.

```
import requests
res = requests.get("https://www.google.com/")
print(res.text)
```

API calls should return status codes that classify the status of the request.
Status codes are as follows:
200: OK
201: Created
400: Bad Request
403: Forbidden
404: Not Found
405: Method Not Allowed
422 Unprocessable Entity

```
res = requests.get("https://api.fixer.io/latest?base=USD&symbols=EUR")
if res.status_code != 200:
    raise Exception("ERROR: API request unsuccessful.")
    data = res.json()
    print(data)
```

An API can be designed to return a JSON object that contains some of the information from the site.
In flask the `application.py` can use the following code to accomplish this.

```
from flask import jsonify

@app.route("/api/flights/<int:flight_id>")
def flight_api(flight_id):
    """Return details about a single flight."""

    # Make sure flight exists.
    flight = Flight.query.get(flight_id)
    if flight is None:
        return jsonify({"error": "Invalid flight_id"}), 422

    # Get all passengers.
    passengers = flight.passengers
    names = []
    for passenger in passengers:
        names.append(passenger.name)
    return jsonify({
            "origin": flight.origin,
            "destination": flight.destination,
            "duration": flight.duration,
            "passengers": names
        })
```

Chapter 7

Javascript

7.1 Intro

FYI Ctrl+Shift+I in chrome opens the console. In the console commands can be entered and variables can be used that have been defined in the page. Variables can also be modified from this console as well.

The `<script>` tags within the `<head>` is where javascript can be put.

```
<head>
    <script>
        alert('Hello');
    </script>
</head>
```

7.2 Event Handling

There a number of event handlers in Javascript, and here are a few:

```
onclick
onmouseover
onkeydown
onkeyup
onload
onblur
```

The following code creates a button that runs the `hello()` function only when the button is clicked.

```
<head>
    <script>
        function hello() {
```

```
            alert('Hello!');
        }
    </script>
</head>
<body>
    <button onclick="hello()">Click Here</button>
</body>
```

When forms are submitted it appears that by default they will resubmit the page. This functionality can simply be suppressed, or a function can be run when the form is submitted.

```
<form onsubmit="return false">
<form onsubmit="return getUsername()">
```

Javascript can be used to change the contents of the html within a page.
In the following code, the `querySelector` searches for the first `h1` tag on the page, and then uses the `innerHTML` method to modify the contents within that tag.

```
<style>
    <script>
        function hello() {
            document.querySelector('h1').innerHTML = 'Goodbye!';
            }
    </script>
</style>
<body>
    <h1>Welcome!</h1>
    <button onclick="hello()">Click Here!</button>
</body>
```

`querySelector` can also select by id or class.

```
document.querySelector('tag')
document.querySelector('#id')
document.querySelector('.class')
```

Here is an example that uses the `id` to control its contents.

```
<style>
    <script>
        counter = 0;

        function count() {
            counter++;
            document.querySelector('#counter').innerHTML = counter;
        }
```

```
    </script>
</style>
<body>
    <h1 id="counter">0</h1>
    <button onclick="count()">Click Here!</button>
</body>
```

Template literals allow variables to be used within a string.

```
if (counter % 10 === 0) {
    alert(`Counter is at ${counter}!`);
}
```

7.2.1 External javascript

The necessary javascript can be separated out of a page and included in a `something.js` file and then referenced within an html file.

```
<head>
    <script src="something.js"></script>
</head>
```

It appears that flask's handling of static external javascript files is a bit nuanced. At least when running locally, flask will serve static javascript files within a `static` directory existing within the root directory. The following will reference the `layout.js` file located within the `static` directory.

```
<script src = "{{ url_for('static', filename = 'layout.js') }}"></script>
```

7.3 Variables

`const` can not be redefined.
`let` has block scope so it only exists within a block defined by `{}`.
`var` exists within the function in which it is defined.

7.4 Event Listeners

Event listeners can be run only when the DOM is loaded, and can listen for events within the html document. This can be a helpful way of offloading some of the work from the html onto javascript.
Below is an example of using javascript to control the function of a button using a callback function.
In the following code, the `addEventListener` takes two input variables, the first is `DOMContentLoaded` which triggers the listener to start processing after the page has loaded. For the second parameter, whatever should be triggered is

passed. Below, the second parameter is an entire function being passed in, and it selects the first button, and runs the count method on a click event.

```
<head>
    <script>
        document.addEventListener('DOMContentLoaded', function() {
            document.querySelector('button').onclick = count;
        });

        let counter = 0;

        function count() {
            counter++;
            document.querySelector('#counter').innerHTML = counter;

            if (counter % 10 === 0) {
                alert(`Counter is at ${counter}!`);
            }
        }
    </script>
</head>
<body>
    <h1 id="counter">0</h1>
    <button>Click Here!</button>
</body>
```

An event listener can be used to pull the contents of a field when a button is pressed.

```
<head>
    <script>
        document.addEventListener('DOMContentLoaded', function() {
            document.querySelector('#form').onsubmit = function() {
                const name = document.querySelector('#name').value;
                alert(`Hello ${name}!`);
            };
        });
    </script>
    <title>My Website</title>
</head>
<body>
    <form id="form">
        <input id="name" autocomplete="off" autofocus placeholder="Name" type="text">
        <input type="submit">
    </form>
</body>
```

CSS styling can be changed using javascript event listeners.

```
document.querySelector('#red').onclick = function() {
    document.querySelector('#hello').style.color = 'red';
};
```

Event listeners can be generalized and used for multiple elements using the `dataset` property.
In the following code, `document.querySelectorAll('.color-change')` selects all of the buttons that belong ot hte `color-change` class.
The `forEach` method loops through the array of all of the buttons.
Then the onclick method uses `button.dataset.color` to reference the `data-color="red"` part of each button to set the color of the text defined by `id="hello"`.
The `data-color="red"` of each button can be set to anything else that follows the pattern `data-*` and then can be referenced with `button.dataset.*`.

```
<head>
    <script>
        document.addEventListener('DOMContentLoaded', function() {

            // Have each button change the color of the heading
            document.querySelectorAll('.color-change').forEach(function(button) {
                button.onclick = function() {
                    document.querySelector('#hello').style.color = button.dataset.color;
                };
            });

        });
    </script>
    <title>My Website</title>
</head>
<body>
    <h1 id="hello">Hello!</h1>
    <button class="color-change" data-color="red">Red</button>
    <button class="color-change" data-color="blue">Blue</button>
    <button class="color-change" data-color="green">Green</button>
</body>
```

7.4.1 Arrow Functions

Arrow functions are new to ES6 and abbreviate the manner in which functions are created.
Below a function is created that creates the Hellow World alert.

```
() => {
    alert('Hello world!');
```

```
}
```

Below is a function that takes as input the variable `x`.

```
x => {
    alert(x);
}
```

An even more succinct manner of writing a function is below.

```
x => x * 2;
```

Below is an example of the above code re-written to use arrow functions.

```
<script>
    document.addEventListener('DOMContentLoaded', () => {

        // Have each button change the color of the heading
        document.querySelectorAll('.color-change').forEach(button => {
            button.onclick = () => {
                document.querySelector('#hello').style.color = button.dataset.color;
            };
        });

    });
</script>
```

7.5 Local Storage

Local storage allows variables to be stored within the browser of a user and recurrently accessed by javascript even between sessions.
In the following example, the value for counter is being retrieved if it can be, or if not it is set to zero when the page is first loaded.
Then if the `counter` variable is ever modified, the new value will be stored.

```
<script>
    // Set starting value of counter to 0
    if (!localStorage.getItem('counter'))
        localStorage.setItem('counter', 0);

    document.querySelector('#counter').innerHTML = counter;
    localStorage.setItem('counter', counter);
<script>
```

All variables stored in local storage can be cleared at once.

```
localStorage.clear()
```

Chapter 8

Hackernews

8.1 Setup

First make sure create react app is installed. The project here follows this tutorial. There are lots of other good looking tutorials like The React Handbook, and others at gitconnected.

```
npm i -g create-react-app
```

Then create a new directory for the app.

```
create-react-app hacker-news-clone
```

Change into the newly created directory and then create a file to handle environmental variables.

```
cd hacker-news-clone
touch .env
```

Within the `.env` file refer to the `src` folder. This will allow dependencies to be more easily imported. Add the following to the `.env` file.

```
NODE_PATH=src
```

Make a components directory within `src` to hold all of the components for the project.

```
mkdir -p src/components/App
```

Make a services directory within `src` to add additional functionality to the app and reference other site APIs.

```
mkdir src/services
```

Make a styles directory within `src` to add styles that can be used across the app.

```
mkdir -p src/styles
```

Make a store directory within `src` to add styles that will add Redux function.

```
mkdir -p src/store
```

Make a utils directory within `src` for shared functions across the app.

```
mkdir -p src/utils
```

Now move `App.js` to components just to keep the components bundled together. Rename `App.js` to index so that it can be imported from the mycomponents app.

```
mv src/App*js src/components/App/
mv src/components/App/App.js src/components/App/index.js
mv src/logo.svg src/components/App/
```

Delete the css files because style components will be used instead.

```
rm src/*css
```

Remove the imports of the css files in `src/components/App/index.js`.

```
import './App.css';
```

And remove the import within `src/index.js`.

```
import './index.css';
```

Now create some styles to be used throughout the app.

```
mkdir src/styles
touch src/styles/globals.js
touch src/styles/palette.js
```

The js files above contain routine code that can be copied from the author's github page. Alternatively, here is the code for `global.js`.

```
import { injectGlobal } from 'styled-components';
import { colorsDark } from './palette';

const setGlobalStyles = () =>
  injectGlobal`
    * {
      box-sizing: border-box;
    }
    html, body {
```

```
      font-family: Lato,Helvetica-Neue,Helvetica,Arial,sans-serif;
      width: 100vw;
      overflow-x: hidden;
      margin: 0;
      padding: 0;
      min-height: 100vh;
      background-color: ${colorsDark.background};
    }
    ul {
      list-style: none;
      padding: 0;
    }
    a {
      text-decoration: none;
      &:visited {
        color: inherit;
      }
    }
  `;

export default setGlobalStyles;
```

And here is the code for `palette.js`.

```
export const colorsDark = {
  background: '#272727',
  backgroundSecondary: '#393C3E',
  text: '#bfbebe',
  textSecondary: '#848886',
  border: '#272727',
};

export const colorsLight = {
  background: '#EAEAEA',
  backgroundSecondary: '#F8F8F8',
  text: '#848886',
  textSecondary: '#aaaaaa',
  border: '#EAEAEA',
};
```