Learning Objectives: Bootstrap

Learners will be able to...

- **Add Bootstrap to an HTML document**
- **Position elements using the grid system**
- **Style text and elements with Bootstrap classes**
- **Serve static files in Django**

info

Make Sure You Know

You are comfortable working with basic HTML and CSS.

Limitations

This is only a quick introduction to Bootstrap. We will not dive deep into all of the ways in which it can be used.

Styling HTML

CSS

So far, our website is very basic. It is simple in structure, which is fine. Not every website needs to be complex. However, it looks bad. We should not be creating websites without any styling at all.

CSS or Cascading Style Sheets is used to style HTML documents. CSS is a powerful tool to make sites appealing and useful. We are first going to experiment with CSS to see what is possible. Then, we will style our site so that it looks better.

In terminal, enter the following command. This creates the file `practice.html`. We will use this document as a separate playground before altering our Django project.

```
touch practice.html
```

Next, set up a blank website. Click the link to open the newly created HTML file. This document will serve as a place to explore and experiment. In the `<body>` tag, add a “Hello, World!” message.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
  </head>
  <body>
    <h1>Hello, World!</h1>
  </body>
</html>
```

Click the link below to open `practice.html`. We should see the “Hello, World!” greeting.

Bootstrap

Instead of writing CSS in the traditional way, we are going to use the Bootstrap CSS framework. Bootstrap is to CSS as Django is to web development — much of the work is abstracted away by the framework. This means you can accomplish a lot with very little code. Go back to our HTML file with the following link.

Our practice site has a `<head>` section. This area is not used for adding visual elements to a site. Things like metadata and links to external tools go in this section. Add a link to the content delivery network (CDN) that hosts the Bootstrap code.

```
<head>
  <meta charset="utf-8">
  <link
      href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap
      .min.css" rel="stylesheet" integrity="sha384-
      1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3"
      crossorigin="anonymous">


</head>
```

Because Bootstrap is such a powerful tool that automates many tasks for us, it relies on the use of JavaScript to run properly. In the `<body>` of the page, add a `<script>` tag for Bootstrap.

```
<body>
  <h1>Hello, World!</h1>

  <script
      src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.m
      js" integrity="sha384-
      ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBO0LRn5q+8nbTov4+1p"
      crossorigin="anonymous"></script>


</body>
```

▼ **Code**

> Your complete code should look like this:
>
> ```
> <!DOCTYPE html>
> <html>
>   <head>
>     <meta charset="utf-8">
>     <link
>         href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap
>         .min.css" rel="stylesheet" integrity="sha384-
>         1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3"
>         crossorigin="anonymous">
>
>   </head>
>   <body>
>     <h1>Hello, World!</h1>
>
>     <script
>         src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bun
>         js" integrity="sha384-
>         ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBO0LRn5q+8nbTov4+1p"
>         crossorigin="anonymous"></script>
>
>   </body>
> </html>
> ```

Open the practice site with the link below. Then click the blue arrows in the shape of a circle. This is the refresh button for the browser inside Codio. Every time you make changes to the HTML file, you need to refresh the Codio browser to see them.

The image depicts two buttons from the Codio browser. The first

is a blue triangle. The second image shows two blue arrows in the shape of a cirle.

The text in the website should change. Even though all we did was add Bootstrap to an HTML document, changes were made. Bootstrap has default settings for HTML elements. This is the power of using a CSS framework like Bootstrap. So much of the work is already done for us, which eases the task of styling a website. We also have the ability to override these default settings.

Div Tags

Using `<div>` Tags

If you are familiar with CSS, this may seem a bit odd, but Bootstrap uses `<div>` tags with regularity. Let's replace our `<h1>` tag with a `<div>` for our "Hello, World!" message.

```
<div>
  Hello, World!
</div>
```

Open the website. Obviously the text no longer renders as a heading.

▼ **Did you notice?**

> We did not have to refresh the Codio browser since this is the first time the page has been loaded. Use the refresh icon for all subsequent changes.

However, we can add the styling we desire through the use of Bootstrap classes. In our `<div>`, add the class `h1`. This will do the exact same thing as using an `<h1>` tag. Even though Bootstrap makes use of `<div>` tags, you do not lose the ability to style your elements in the way you want.

```
<div class="h1">
  Hello, World!
</div>
```

Open the website and refresh to see the changes.

We can combine classes to get the exactly styling we desire. In addition to the `h1` class, add the `container` class to our `<div>`. This class adds margins on the left and right so there is some horizontal spacing for our content.

```
<div class="container h1">
  Hello, World!
</div>
```

Open the website and refresh to see the changes.

The new container for our content is a bit hard to see. Let's add a border so we can more easily see the container and margins.

```
<div class="container h1 border">
  Hello, World!
</div>
```

Open the website and refresh to see the changes.

challenge

Try this variation:

- Make the following changes to the border:

```
<div class="container h1 border border-4 rounded">
  Hello, World!
</div>
```

Open the website and refresh to see the changes.

▼ **What is happening?**

> The `border-4` sets the width of the border. You can use values from 1 to 5. The `rounded` class rounds the corners of the border. Click this link to see more information about borders in Bootstrap.

Colors

Bootstrap provides some built-in colors that can be used across a variety of classes. The basic colors are: primary, secondary, success, danger, warning, light, dark, and white.

You can add color to an element by taking the color name and adding it to the appropriate class. We can use the primary color (blue) as the background of our container by using `bg-primary`. We can then change the text color to white with `text-white`. Add this to our `<div>`.

```
<div class="container h1 bg-primary text-white">
  Hello, World!
</div>
```

Open the website and refresh to see the changes.

challenge

Try these variations:

- Give the container a red border, green background and red text.

▼ **Solution**

> Here is one possible solution:
>
> ```
> <div class="container h1 bg-success text-danger
>     border border-4 border-danger">
>   Hello, World!
> 
> ```
>
> The color gree is represented by “success”. So `bg-success` gives the container a gree background. “Danger” represents red. Use `text-danger` and `border-danger` for red text and red border. The default border width is not very large. Using `border-4` helps with visibility but is not required.

- Give the body of the page a yellow background. Have the container be transparent with light-blue text.

▼ **Solution**

> Here is one possible solution:
>
> ```
> <body class="bg-warning">
>   
>     Hello, World!
>   
> ```
>
> You are asked to make the background of the body a yellow color. That means you need to use `bg-warning` as the class for the body tag. Use `bg-transparent` for the background color of the container. Light blue is represented with “info”, so use `text-info` for light-blue text.

Open the website and refresh to see the changes.

The Grid System

Rows and Columns

Layout in Bootstrap is done through a combination of containers (like the `<div>` we saw on the last page) and a grid system made up of rows and columns. Bootstrap places rows on the page from top to bottom. Columns appear inside a row, and Bootstrap places them on the page from left to right.

Let's start by removing the yellow background from the `<body>`. Then create a `<div>` with the class `container`. Set the text for all of the rows and columns to white. We do not want the rows to touch the top of the page, so add some padding with `p-3` (increase the number to increase the padding). Finally, to make the text stand out a bit more, use `display-6` (decrease the number to increase the font size).

```
<body>
  <div class="container text-white p-3 display-6">

  </div>
```

Create a `<div>` and give it the class `row`. Inside this element, create another `div` and give it the class `col` for column. We want to see the boundaries of each column, so we are going to give it a background color.

```
  <div class="container text-white p-3 display-6">
    <div class="row">
      <div class="col bg-primary">Column 1</div>
    </div>
  </div>
```

Next, let's add a second row that has two columns. Create another `div` with the class `row`. Inside of it add two more `div` elements with the class `col`. Label them as columns 2 and 3, and give each one its own color.

```
<div class="container text-white p-3 display-6">
  <div class="row">
    <div class="col bg-primary">Column 1</div>
  </div>

  <div class="row">
    <div class="col bg-dark">Column 2</div>
    <div class="col bg-danger">Column 3</div>
  </div>
</div>
```

Open the website and refresh to see the changes.

challenge

Try this variation:

- Create a third row with three columns.

▼ **Solution**

> Here is one possible solution:
>
> ```
> 
>   
>     Column 1
>   
>
>   
>     Column 2
>     Column 3
>   
>
>   
>     Column 4
>     Column 5
>     Column 6
>   
> 
> ```
>
> Create another `div` with the class `row`. Inside, create three `div` elements with the class `col`. Give each column its own color and text.

Open the website and refresh to see the changes.

Column Width

You may have noticed that Bootstrap creates columns in a row with equal width. It is possible to create columns of unequal width. The Bootstrap grid system has a maximum width of 12 units. Bootstrap is responsive, which means the size of the unit depends on the size of your screen. You can set the width of a column by the number of units it occupies.

For example, our first row has only a single column. Bootstrap defaults to the full 12 units for the column's width. We can reduce the width by specifying that it should only span 10 units. Add a `-10` after the `col` class. This number represents the width of the column measured in units.

```
<div class="row">
  <div class="col-10 bg-primary">Column 1</div>
</div>
```

Column 1 should stop a bit short relative to the other columns. Open the website and refresh to see the changes.

Let's modify the second row so that the first column takes up one-third of the width and the second column takes up two-thrids of width. Since there are 12 total units, the fist column should take up 4 units and the second should take up 8.

```
<div class="row">
  <div class="col-4 bg-dark">Column 2</div>
  <div class="col-8 bg-danger">Column 3</div>
</div>
```

Open the website and refresh to see the changes.

challenge

Try this variation:

- Modify the columns in row 3 so that they take up 25%, 50%, and 25% respectively.

▼ **Solution**

> Here is one possible solution:
>
> ```
> 
>   Column 4
>   Column 5
>   Column 6
> 
> ```
>
> 25% of 12 is 3 and 50% of 12 is 6. So the columns should have widths of 3, 6, and 3 respectively.

Open the website and refresh to see the changes.

Components

List Group

Bootstrap provides pre-made components to help developers easily implement common features on websites. We are only going to look at a few of the choices. See the Bootstrap documentation for the full list of components.

The first component we are going to look at is the list group. Remove the code from the previous page in the `<body>` tag. In its place, add a `<div>` of with the class `container`. Just as before, provide some padding and increase the font size. This time, use the dark color for the text.

Inside the container, create a dive with the class `list-group`. We want each item in the list to be a link. Create an `<a>` tag for each item in the list. Use the classes `list-group-item` and `list-group-item-action`.

```
<div class="container text-dark p-3 display-6">
  <div class="list-group">
    <a href="#" class="list-group-item list-group-item-
    action">Link 1</a>
    <a href="#" class="list-group-item list-group-item-
    action">Link 2</a>
    <a href="#" class="list-group-item list-group-item-
    action">Link 3</a>
    <a href="#" class="list-group-item list-group-item-
    action">Link 4</a>
  </div>
</div>
```

▼ **Did you notice?**

> The symbol # was used for the link address. This is basically a link to nowhere.

Once the page loads, mouse over the various links. Notice how the background becomes slightly darker as you hover over each element in the list group. Click on the links as well. They do not take you anywhere, but you should see an animation each time you click a list element. All of this is automatically handled by `list-group-item-action` class.

challenge

Try this variation:

- Change the `<div>` class from `list-group` to:

```
<div class="list-group list-group-flush">
```

Open the website and refresh to see the changes.

Cards

Cards are flexible containers that provide lots of options for presenting information. Let's start by erasing the text in the `<body>` tag. Then set up a `<div>` with the class `container`. Use dark text and set the padding to 3.

```
<div class="container text-dark p-3">


</div>
```

Next, create a `<div>` with the class `card` inside the container. We want there to be space between the cards so they are not touching. Create a margin at the bottom of the card with the value of 2. Include an `img` tag so we can have an image in our card. Use the class `card-img-top` to place the image at the top of the card. Then add another `<div>` with the class `card-body`.

```
<div class="card mb-2">
  <img
  src="https://images.pexels.com/photos/3739926/pexels-
  photo-3739926.jpeg?
  auto=compress&cs=tinysrgb&w=1260&h=750&dpr=1"
  class="card-img-top">
  <div class="card-body">


  </div>
</div>
```

The body of a card is composed of two things — a title and text. Inside the card body, add a `<p>` tag with the class `card-title`. Also add the class `display-6` to make the text larger. Create another `<p>` tag with the class `card-text`. Add any text you want to appear in the card. Finally, we want a

button that links to another page. Add an `<a>` tag with the class `btn`. Add another class `btn-primary` to color the button blue. Use `#` as the link address.

```
<div class="card mb-2">
  <img
  src="https://images.pexels.com/photos/3739926/pexels-
  photo-3739926.jpeg?
  auto=compress&cs=tinysrgb&w=1260&h=750&dpr=1"
  class="card-img-top">
  <div class="card-body">
    <p class="card-title display-6">Card 1</p>
    <p class="card-text">The quick brown fox...</p>
    <a href="#" class="btn btn-primary">Go somewhere</a>
  </div>
</div>
```

Open the website and refresh to see the changes.

challenge

Try this variation:

- Add a second card with the title of “Card 2”, the text of “...jumped over the lazy dog.”, and the image URL of:

  ```
  https://www.pexels.com/photo/brown-and-white-short-coated-dog-lying-3978352/
  ```

Open the website and refresh to see the changes.

▼ **Solution**

> Here is one possible solution:

```
<div class="card">
  <img
  src="https://images.pexels.com/photos/3978352/pexels-photo-3978352.jpeg?auto=compress&cs=tinysrgb&w=1260&h=750&dpr=1"
  class="card-img-top">
  <div class="card-body">
    <p class="card-title display-6">Card 2</p>
    <p class="card-text">...jumped over the lazy dog.</p>
    <a href="#" class="btn btn-primary">Go somewhere</a>
  </div>
</div>
```

▼ **Images**

> Both of these images are in the public domain. Here are links to the fox and the dog.

Styling Django

Adding Bootstrap

Now that we have an idea about how Bootstrap works, let's style our Django project. Start by activating the `django` virtual environment.

```
conda activate django
```

The first thing we need to do is import Bootstrap into our project. Since we want Bootstrap accessible to every template, we are going to add it to the `base.html` file.

Our base template does not have the proper structure of an HTML document. Copy and paste the code below into the IDE. Visually speaking, nothing has changed with our website. The underlying structure has, which will help us to better organize our code.

```
<!DOCTYPE html>
<html>

    <head>
        <meta charset="utf-8">
    </head>

    <body>
        <header>
            <a href="{% url 'home' %}">Home</a> |
            <a href="{% url 'about' %}">About</a>
        </header>

        {% block content %} {% endblock content %}
    </body>

</html>
```

In the `<head>` tag, add the link to CDN that hosts the Bootstrap framework.

```
    <head>
        <meta charset="utf-8">
        <link
        href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap
        .min.css" rel="stylesheet" integrity="sha384-
        1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3"
        crossorigin="anonymous">


    </head>
```

Next, add the Bootstrap `<script>` tag in the body after the block template tag.

```
    <body>
        <header>
            <a href="{% url 'home' %}">Home</a> |
            <a href="{% url 'about' %}">About</a>
        </header>

        {% block content %} {% endblock content %}
        <script
        src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle
        js" integrity="sha384-
        ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBO0LRn5q+8nbTov4+1p"
        crossorigin="anonymous"></script>

    </body>
```

Bootstrap is now installed for our Django project. Open the terminal, enter the command to run the dev server, and open the website with the link. You should see some slight changes due to the default settings for Bootstrap.

```
python manage.py runserver 0.0.0.0:8000
```

Base Template

Our content is placed in the top-left corner of the page. To remedy this, add a `<div>` tag to the body of the page. The `<header>` and `<script>` tags should go inside this `<div>`. The new `<div>` should have the classes `container` and `p-3` so there are margins on left and right, as well as a bit of padding on the top.

```
    <body>
        <div class="container p-3">
            <header>
                <a href="{% url 'home' %}">Home</a> |
                <a href="{% url 'about' %}">About</a>
            </header>

            {% block content %} {% endblock content %}
            <script
        src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle
        js" integrity="sha384-
        ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBO0LRn5q+8nbTov4+1p"
        crossorigin="anonymous"></script>

        </div>
    </body>
```

Let's transform the `<header>` tag to make it visually more appealing. Add the `border-bottom` class to draw a horizontal line across the bottom of the header. Use `border-secondary` to darken the line a bit. Finally add a small bit of padding to the bottom of the header with `pb-1`.

```
        <header class="border-bottom border-secondary pb-1">
            <a href="{% url 'home' %}">Home</a> |
            <a href="{% url 'about' %}">About</a>
        </header>
```

Next, we are going to transform the links into buttons. Add the `btn` class to create a button. The `btn-primary` class makes the buttons blue. Since each link is a distinct button, we can remove the vertical bar (`|`) that was previously separating the links.

```
        <header class="border-bottom border-secondary pb-1">
            <a href="{% url 'home' %}" class="btn btn-
primary">Home</a>
            <a href="{% url 'about' %}" class="btn btn-
primary">About</a>
        </header>
```

Open the terminal, enter the command to run the dev server, and open the website with the link. The header now has blue buttons that take you back to the homepage or the about page. The rest of this links should work just as before.

```
python manage.py runserver 0.0.0.0:8000
```

Switch back to the terminal and stop the dev server with `Ctrl+C` on the keyboard. Then deactivate the virtual environment.

```
conda deactivate
```

Adding Images

Static Files

Start by activating the `django` virtual environment.

```
conda activate django
```

On a previous page, we created a card component with an image at the top. All we had to do was use an `<img>` tag and supply the URL of the image. Unfortunately, images do not work that way in Django. Instead, we need to have the images in our project and serve them as a static file.

▼ **What are static files?**

> Static files often refers to files like JavaScript, CSS, and images. For our purposes, we are talking about images when we say static files.

Preparing our Django project to serve static files is a multistep process. Let's start by modifying the `urls.py` file. Import `statticfiles_urlpatterns` from `django.contrib.staticfiles.urls`. Then, at the bottom, we are going to append to the `urlpatterns` list the return value from the `staticfiles_urlpatterns()` function.

```
from django.contrib import admin
from django.urls import path, include
from django.contrib.staticfiles.urls import
        staticfiles_urlpatterns

urlpatterns = [
    path("admin/", admin.site.urls),
    path("", include("languages.urls")),
]

urlpatterns += staticfiles_urlpatterns()
```

Open the terminal and run the Black formatter to make sure our code has a consistent style.

```
black languages/urls.py
```

Next, we need to modify the `settings.py` file. All the way at the bottom, we should already see the definition of the `STATIC_URLS` variable. This tells Django where to go to serve our static files. However, Django does not know where to find them. Creat the `STATICFILES_DIRS` variable and tell Django to look in the `assets` directory. This is where we are going to store our images.

```
# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/3.2/howto/static-files/

STATIC_URL = "/static/"
STATICFILES_DIRS = (
  os.path.join(BASE_DIR, 'assets'),
)
```

Open the terminal and run the Black formatter to make sure our code has a consistent style.

```
black beginner_project/settings.py
```

Adding Images

Our Django project is now set up to server static files, and it knows to look in the `assets` directory to find the static files. The problem is, however, the `assets` directory does not exist, nor do we have any images. Enter the following command in the terminal to make the `assets` directory.

```
mkdir assets
```

We are going to use logos for each of the programming languages as the images in the cards. These logos have already been uploaded to Codio. They reside in a hidden folder called `.guides`. The terminal command below copies all of the contents of the `.guides/img/logos` directory and pastes them into the `assets` directory. Run the following command.

```
cp .guides/img/logos/* assets/
```

challenge

Try this variation:

- If we properly created the `assets` directory and copied over the images, then the links below should load each programming language logo. Verify that each image loads as expected.

▼ **Images**

> Here are the images used for this project:
>
> - By www.python.org - www.python.org, GPL, Link
> - By The F# Software Foundation - https://fsharp.org/, Public Domain, Link
> - By Matthew Butterick - Racket's source code., LGPL, Link

Open the terminal and deactivate the `django` virtual environment.

```
conda deactivate
```

Card Components

Card Template

Instead of a list of links on the homepage, we are going to have a card component for each programming language. The structure of each card will not change, though the contents will. This is a good time to use a template. In the terminal, enter the command to create the `card.html` template in the `templates` directory.

```
touch templates/card.html
```

Create a `<div>` and set its class to `card`. Add an `<img>` tag and replace its source with a block template tag named `img-link`. Another `<div>` serves as the card body. The first `<p>` has the `card-title` class. Use `display-6` and `text-primary` to style the title. The contents of this element should be a block template tag named `language`. The next `<p>` is the card text. Add a block template tag named `description`. Finally, create an `<a>` tag whose link source is a block template tag named `language-link`. Set the class to `btn btn-primary` to make the link a blue button.

```
<div class="card">
    <img src="{% block img-link %}{% endblock img-link %}">
    <div class="card-body">
        <p class="card-title display-6 text-primary">{% block
        language %}{% endblock language %}</p>
        <p class="card-text">{% block description %}{% endblock
        description %}</p>
        <a href="{% block language-link %}{% endblock language-
        link %}" class="btn btn-primary">
          More info
        </a>
    </div>
</div>
```

Python Card

Now that the genearl card template is complete, we are going to make a template for each language. Each template should have block template tags that override content from the parent template. We need information for

an image, a the name of the language, a description, and a link. Let's start with Python. Open the terminal and use the `touch` command to create the `pythonCard.html` file in the `templates` directory.

```
touch templates/pythonCard.html
```

This template should extend the `card.html` template. In addition, we need to include `static` so we can display static files, or in our case images. Create a block template tag for `img-link`. Use the `static` template tag to tell Django we are using a static file. Then put the name of the file (`python-logo.png`) between quotes.

```
{% extends "card.html" %}
{% load static %}

{% block img-link %}
{% static 'python-logo.png' %}
{% endblock img-link %}
```

Return `Python` in the block template tag named `language`. The block template tag `description` has a short description of Python, while `language-link` returns a `url` template tag that links to the `python.html` template.

```
{% extends "card.html" %}
{% load static %}

{% block img-link %}
{% static 'python-logo.png' %}
{% endblock img-link %}

{% block language %}Python{% endblock language %}

{% block description %}
Python is general purpose programming language which emphasizes
        readability.
{% endblock description %}

{% block language-link %}
{% url 'python' %}
{% endblock language-link %}
```

F# Card

Open the terminal and create the `fsharpCard.html` file in the `templates` directory.

```
touch templates/fsharpCard.html
```

Repeat this process for F#. Update the `img-link` block with the correct image name. Change `langauge` to `F#`. Provide a quick summary of the language in `description`, and link to the `fsharp` template in `language-link`.

```
{% extends "card.html" %}
{% load static %}

{% block img-link %}
{% static 'f-sharp-logo.png' %}
{% endblock img-link %}

{% block language %}F#{% endblock language %}

{% block description %}
F# is a functional-first language from the ML family that runs
        on the .NET framework.
{% endblock description %}

{% block language-link %}
{% url 'fsharp' %}
{% endblock language-link %}
```

Racket Card

Open the terminal and create the `racketCard.html` file in the `templates` directory.

```
touch templates/racketCard.html
```

Again, do the same thing — update the information for the image name, language name, description, and link for Racket.

```
{% extends "card.html" %}
{% load static %}

{% block img-link %}
{% static 'racket-logo.png' %}
{% endblock img-link %}

{% block language %}Racket{% endblock language %}

{% block description %}
Racket descends from the Lisp family of languages and has an
        educational focus.
{% endblock description %}

{% block language-link %}
{% url 'racket' %}
{% endblock language-link %}
```

Positioning the Cards

Add a Container

Start by activating the `django` virtual environment.

```
conda activate django
```

We are going to replace the unordered list with three cards. In the `home.html` file, erase the list.

```
{% extends "base.html" %}

{% block content %}
<h1>Programming Languages</h1>

<p>This site provides a brief overview of several popular
        programming languages. Each page includes a quick
        summary of the language's history as well as a few
        examples of how it is used today.</p>

{% endblock content %}
```

In its place we are going to add a container and a row. For now, we are going to put all of the cards on one row. If you are creating your own website, feel free to place your cards across multiple rows.

```
{% extends "base.html" %}

{% block content %}
<h1>Programming Languages</h1>

<p>This site provides a brief overview of several popular
        programming languages. Each page includes a quick
        summary of the language's history as well as a few
        examples of how it is used today.</p>

<div class="container">
    <div class="row">

    </div>
</div>

{% endblock content %}
```

Since each card has its own HTML file, we can use the `include` template tag. Put each card inside a `<div>` with the class `col-4` so they are evenly spaced out across the grid.

```
{% extends "base.html" %}

{% block content %}
<h1>Programming Languages</h1>

<p>This site provides a brief overview of several popular
        programming languages. Each page includes a quick
        summary of the language's history as well as a few
        examples of how it is used today.</p>

<div class="container">
    <div class="row">
        <div class="col-4">{% include "pythonCard.html" %}</div>
        <div class="col-4">{% include "fsharpCard.html" %}</div>
        <div class="col-4">{% include "racketCard.html" %}</div>
    </div>
</div>

{% endblock content %}
```

Viewing the Site

The styling of our Django project is now complete. Open the terminal and run the dev server.

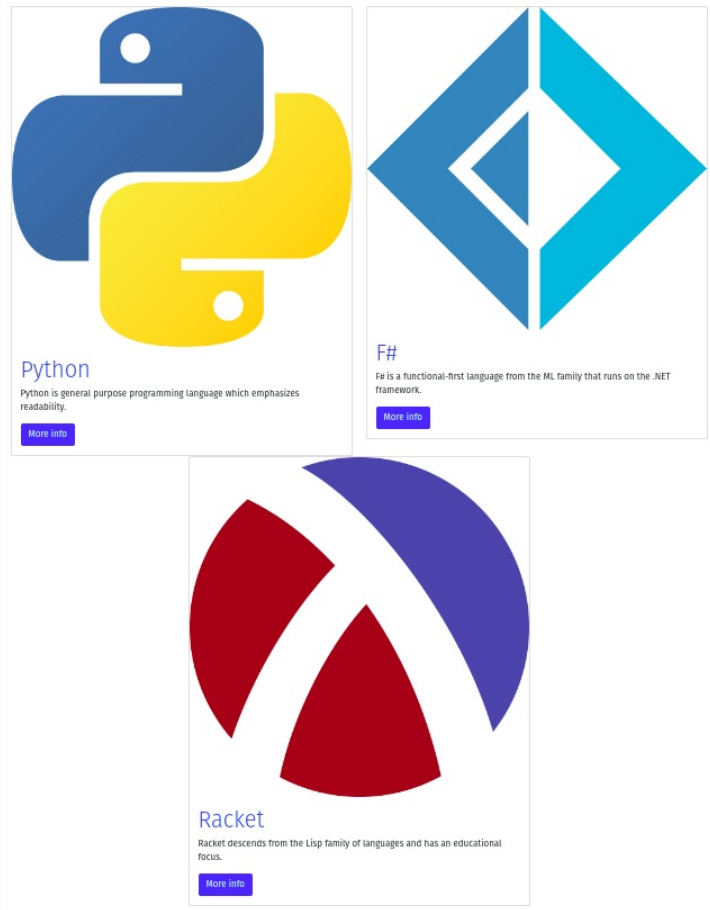
```
python manage.py runserver 0.0.0.0:8000
```

Open the website. You should now see three cards appear in a single row in place of the unordered list. Test the links on the each card to make sure each programming language page loads as expected.

challenge

Try this Variation:

- Create a grid with two rows. Two cards are on the first row and one card is on the second. The cards should be centered in the container.



The image depicts the three cards spread across two rows. The first row has two cards that take up half of he grid each. The second row has one card. The third card has the same width as the other two cards. The third card is centered in the second row.

Make your changes to the `home.html`. Because you are only changing HTML files, you can leave the dev server running. Once your changes are done, go back to the website and click the icon with blue arrows to refresh the Django project.

▼ **Solution**

> Here is one possible solution:
>
> ```
> 
>     
>         {% include "pythonCard.html"
>         %}
>         {% include "fsharpCard.html"
>         %}
>     
>     
>         
>         {% include "racketCard.html"
>         %}
>         
>     
> 
> ```
>
> All of the cards should have the same width, `col-6`. The first row is easy, because Django places the components for you. However, the second row is a bit tricky. We want the third card to be centered. Create columns on either side of the third card. Since cards have a width of 6, the first and third columns on row 2 need a width of `col-3`.