Learning Objectives: Templates

Learners will be able to...

- **Identify the advantages of Django templates**
- **Extend and override a base template**
- **Use template tags to augment templates**
- **Create URL patterns for new templates**
- **Create class-based views**

info

Make Sure You Know

You are comfortable with Python, specifically inheritance. You can also familiar with basic HTML and interacting with the terminal.

Limitations

This Django project does not make use of models and a database. All templates are only text. There is no CSS.

Creating a Template

Template Directory

Start the `django` virtual environment.

```
conda activate django
```

By default, Django expects templates to reside in each separate Django app. If we were to do this, we would need a `templates` directory in the `languages` app. Then a `languages` directory in the `templates` directory. Finally, we could save our HTML files. It would look something like this:

```
└── pages
  ├── templates
    ├── pages
      ├── home.html
```

If this sounds or looks cumbersome, that's because it is. We are going to opt for a more simple way to store our templates. Instead of having a `templates` directory in each Django app, we are going to have a single `templates` directory for the entire Django project. In the terminal, enter the following command to create the `templates`.

```
mkdir templates
```

▼ **Did you notice?**

> We did not create the `templates` directory inside of the `beginner_project` directory. Instead, it will reside at the same level as the `beginner_project` and `languages` directories. If run the `tree` command in the terminal, this is what you should see:

```
.
├── beginner_project
│   ├── asgi.py
│   ├── __init__.py
│   ├── __pycache__
│   │   ├── __init__.cpython-310.pyc
│   │   ├── settings.cpython-310.pyc
│   │   ├── urls.cpython-310.pyc
│   │   └── wsgi.cpython-310.pyc
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
├── db.sqlite3
├── languages
│   ├── admin.py
│   ├── apps.py
│   ├── __init__.py
│   ├── migrations
│   │   └── __init__.py
│   ├── models.py
│   ├── __pycache__
│   │   ├── __init__.cpython-310.pyc
│   │   ├── urls.cpython-310.pyc
│   │   └── views.cpython-310.pyc
│   ├── tests.py
│   ├── urls.py
│   └── views.py
├── manage.py
└── templates
    └── home.html
```

Next, we need to tell Django to look in this newly create directory to find our templates. Doing this requires a change to our `settings.py` file. Open the file with the link below to direct Django to the `templates` directory in `beginner-project` (our Django project). We are only making a single change to the line that starts with `"DIRS":`.

```
TEMPLATES = [
    {
        "BACKEND":
        "django.template.backends.django.DjangoTemplates",
        "DIRS": [BASE_DIR / 'templates'],
        "APP_DIRS": True,
        "OPTIONS": {
            "context_processors": [
                "django.template.context_processors.debug",
                "django.template.context_processors.request",
                "django.contrib.auth.context_processors.auth",

        "django.contrib.messages.context_processors.messages",
            ],
        },
    },
]
```

▼ **Did you notice?**

> Our new setting uses the constant `BASE_DIR`, which refers to the Django project. In our case, that would be the `beginner_project` directory.

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

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

Home Template

We are ready to make our first template. Create the file `home.html` inside the `templates` directory.

```
touch templates/home.html
```

```
<h1>Home Page</h1>
```

We are not yet ready to run the dev server and see our Django project. We previously used function-based views to display content to the user. While this is a valid way to create a Django project, modern Django development uses class-based views. Our next step is to convert our function-based view into a class-based view.

Deactivate the virtual environment.

```
conda deactivate
```

Class-Based Views

What are Class-Based Views?

We saw how we can use a function-based view to display information in Django. Over time, developers found that there are certain tasks (like displaying a template) that are so common that Django should have a pre-made function to perform these tasks. These are called generic function-based views. However, modifying functions to meet the needs of each developer is a bit cumbersome.

You may remember one of the features of object-oriented development is inheritance. Django introduced class-based views to offer more flexibility to its users. Django has generic class-based view for the common tasks. The developers inherit from the generic view and extend the class to meet their specific needs.

The end result is short and concise class-based views. All of the real work happens in the parent class, which is abstracted away. Let's convert our function-based view into a class-based view.

Creating a Class-Based View

Start by activating the `django` virtual environment.

```
conda activate django
```

Use the link to open the `views.py` file in our `languages` app. Erase the contents of the file. Instead, we are going to import the generic `TemplateView` so we can inherit from it. Then transform `homePageView` into a class (capitalization changes). All we have to do is give the `template_name` attribute a value of the HTML file to render.

```
from django.views.generic import TemplateView

class HomePageView(TemplateView):
  template_name = 'home.html'
```

▼ **Did you notice?**

> Our `HomePageView` class only has a single line of code. The core logic of rendering the template view is handled by the parent class (a generic class-based view). We do not need to worry about or even see that code. All you need to do is inherit from the generic class-based view and make changes so the child class-based view works with your use case. This is the main benefit of class-based views.

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

```
black languages/views.py
```

Now that we are using a class-based view, we need to make some changes. How we render the class-based view in the `urls.py` file in the Django app needs to be updated. Invoke the `as_view()` method (inherited from the generic class-based view) on our `HomePageView` class.

```
from django.urls import path
from .views import HomePageView

urlpatterns = [
  path('', HomePageView.as_view(), name='home'),
]
```

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

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

Enter the following command in the terminal to launch the dev server. Click the link to open the website. You should see the text `Home Page` appear in bold.

```
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
```

About Page

Creating the About Template

Start by activating the `django` virtual environment.

```
conda activate django
```

Like most webpages, our site is going to have an about page. Create the `about.html` file inside of the `templates` directory.

```
touch templates/about.html
```

Open the newly created template and add a tile with `<h1>` tags. We will provide more content to these templates later. For now, this will be a placeholder.

```
<h1>About Page</h1>
```

Updating the Views and URLs

Since we have a new template, we need to update the views and URL patterns to accommodate the `about.html` template. Open the `views.py` file and add a class-based view that inherits from `TemplateView` and sets the `template_name` attribute to `about.html`.

```
from django.views.generic import TemplateView

class HomePageView(TemplateView):
  template_name = "home.html"

class AboutPageView(TemplateView):
  template_name = 'about.html'
```

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

```
black languages/views.py
```

Now open the `urls.py` file so we can update the URL patterns to accept the about page template. We need to import the `AboutPageView` you just created. Then add a new URL path to the list. This path has the pattern `/about` and redirects to the `AboutPageView`. That means when a user goes to your site and adds `/about` to the end of the URL, they will the about template.

```
from django.urls import path
from .views import HomePageView, AboutPageView

urlpatterns = [
  path('about/', AboutPageView.as_view(), name='about'),
  path("", HomePageView.as_view(), name="home"),
]
```

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

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

Enter the following command in the terminal to launch the dev server. The click the link to open the website. Once the home page loads, go up to the address bar for the Django project and add `/about` to the URL. The press `Enter` or `Return` on your keyboard. You should see the text `About Page`.

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

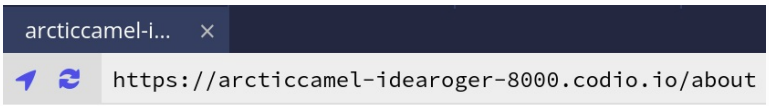

Image depicts the address bar for the Django project with "/about" added to the end.

Important, Codio creates a domain for every project. The domain is a collection of random words like `arcticcamel-idearoger`. Your domain name will be different than the one in the image.

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

```
conda deactivate
```

Base Template

Creating the Base Template

Like object-oriented programming, the real power of templates is the ability to extend a parent. Our templates for the home and about pages do not extend anything. They are standalone HTML files. We are going to create a base template from which all of our other templates will inherit.

Start by activating the `django` virtual environment.

```
conda activate django
```

Create the `base.html` file so that it resides in the `templates` directory so that it resides with the other templates. Then open the newly created file.

```
touch templates/base.html
```

We are going to give our base template a header. In it, we will provide links to the home and about pages. This means that no matter what page we are viewing, there will be two links at the top — one for the home page and one for the about page.

Django uses a special templating language that mixes non-HTML code inside an HTML document. This special syntax is called template tags. You can see the complete list of template tags here. Template tags begin with a `{%` and end with a `%}`. In the code snippet below, the `url` template tag is used to provide links to the home and about pages.

What is really important to understand is the use of the `block` template tag. This is used to provide an area that can be overridden by the child templates. You will notice too that the `block` tag has the name `content`. That means we can reference this specific block and override it. Currently, the content block is empty.

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

{% block content %} {% endblock content %}
```

▼ **Did you notice?**

> The name used in the `url` template tag to designate `home.html` is `home`. This is the optional name from the URL pattern.

Extending the Base Template

Now that we have `base.html` which serves as the parent template, we need to update the home and about templates to inherit from the base template. Open `home.html` and add an `extends` template tag so that it inhertis from `base.html`. Use the `block content`template tags to override the `content` block from `base.html`. This is where we are going to put the information specific to the `home.html` page. For now, it is just a title.

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

{% block content %}
<h1>Home Page</h1>
{% endblock content %}
```

We are going to do something similar for the `about.html` page. It should inherit from `base.html` though an `extends` template tag. Similarly, it will override the `content` block with the title for the about page.

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

{% block content %}
<h1>About Page</h1>
{% endblock content %}
```

Enter the following command in the terminal to launch the dev server. The click the link to open the website. You should now see the links for the home and about pages in the header. Enter `/about` into the address bar for the Django project, and the header will be on the about page too. Click the links in the header to verify they work as expected.

```
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
```

Customizing the Home and About Templates

info

Customizing Your Webpage

If you are going to customize the content of this Django project, now is the time where you are going to deviate from the content below. You will still have the same basic idea, but the content of the webpage will align with a topic that interests you.

Home Template

The time has come to put actual content into our website. This example is going to describe a few programming languages. The home template will list out all of the languages and provide a link to their page.

▼ **Did you notice?**

> We did not start the `django` virtual environment. That is because we are only editing HTML files and we are not running the dev server (see below for an explanation). The virtual environment has Django and Black installed, neither of which we are using on this page. So activating the virtual environment is not really required.

Open `home.html`. Replace the title with one that corresponds to the topic of the website. Then give a brief explanation of what the site does and howto use it. The explanation should be set inside a pari of `<p>` tags.

```
{% 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. Click on a language to
        learn more about it.</p>

{% endblock content %}
```

Next, we are going to create an unordered list of three programming languages — Python, F# (pronounced f-sharp), and Racket. Use the `url` template tag to provide a link to a template for each programming language.

```
{% 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>

<ul>
  <li><a href="{% url 'python' %}">Python</a></li>
  <li><a href="{% url 'fsharp' %}">F#</a></li>
  <li><a href="{% url 'racket' %}">Racket</a></li>
</ul>
{% endblock content %}
```

About Template

The about page is simply going to state the purpose of this page, which is learning how to use templates and URL patterns in Django. Open the about template with the link below. The title does not need to change. Add a quick description in a paragraph tag.

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

{% block content %}
<h1>About Page</h1>

<p>This page is an exercise in learning how Django displays
        templates through URL pattern matching.</p>
{% endblock content %}
```

Important, we are not going to run the dev server at this moment. Doing so would generate an error because the templates for the three programming languages do not yet exist. We will create them on the next page.

Programming Language Templates

info

Customizing Your Webpage

If you are customizing your own pages, feel free to follow the same format of title, image, short text. If you are comfortable with HTML, add more complexity to your pages. Add more than three pages if you desire. Just be sure that you are updating the appropriate files to accommodate these pages.

General Language Template

The page for each programming language will have the same general structure, but the information will be different. This is the perfect case for creating another template. In the terminal, create the `language.html` template which resides in the `templates` directory.

```
touch templates/language.html
```

Open the new file and use the `include` template tag to place the contents of `base.html` in this file. Create an `<h1>` tag with a block template tag. This is where we will place the name of the programming language. Create `<h2>` tags for the “History” and “Uses” subtitles. Each section should have a block template tag for their specific information.

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

<h1>{% block language-name %}{% endblock language-name %}</h1>

<h2>History</h2>
{% block history %}{% endblock history %}

<h2>Uses</h2>
{% block uses %}{% endblock uses %}
```

Python

Open the terminal and create the `python.html` page in the `templates` directory with the command below.

```
touch templates/python.html
```

Open the newly created file and copy in the HTML example. This is a very basic page with a title, section titles, and two short paragraphs. Each new template should extend the language template and override the blocks named `language-name`, `history`, and `uses`.

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


{% block language-name %}Python{% endblock language-name %}


{% block history %}
<p><a href="https://www.python.org/">Python</a> was invented by
        Guido van Rossum and released in 1991. Python is an
        interpreted scripting language with dynamic typing.
        Developers enjoy using Python due to its readability
        (though you do need to pay attention to indentation),
        flexibility, and its comprehensive standard library.</p>
{% endblock history %}


{% block uses %}
<p>Python is one of the most popular programming languages.
        Developers can use it for general purpose scripting, web
        development, scientific computing, artificial
        intelligence and machine learning, graphical application
        development (both desktop and mobile), as well as game
        development. Python is popular in education due to its
        more beginner-friendly syntax. The Raspberry Pi project
        adopted Python as their main programming language for
        its users.</p>
{% endblock uses %}
```

F

The next programming language in our list is F#. Open the terminal and create its HTML page with the terminal command below.

```
touch templates/fsharp.html
```

Open the newly created file and copy/paste the following HTML.

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


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


{% block history %}
<p><a href="https://fsharp.org/">F#</a> (pronounced f-sharp) is
        a strongly-typed, functional programming language that
        runs on the .NET framework. The language was born out of
        a project led by Don Syme at Microsoft Research in 2005.
        Today, F# is officially governed by the F# Software
        Foundation, which is independent of Microsoft. The
        language now also works across the Windows, Mac, and
        Linux platforms.</p>
{% endblock history %}


{% block uses %}
<p>F# is a general purpose programming language. Often times, it
        will be used as a replacement for C#, another
        programming language from Microsoft. Developers have
        also used F# for web programming (it can compile to
        JavaScript), application development, and a variety of
        tasks that involve business analytics.</p>
{% endblock uses %}
```

Racket

The final programming language page is for the language Racket. Open the terminal and create its HTML page with the `touch` command.

```
touch templates/racket.html
```

Open the newly created file and copy/paste the following HTML.

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


{% block language-name %}Racket{% endblock language-name %}


{% block history %}
<p><a href="https://racket-lang.org/">Racket</a> is a general
        purpose programming language derived from the Lisp
        family. Racket first appearedin 1995. It was created by
        Matthias Felleisen and a group of academics as a
        language for novice programmers. Racket also has a
        robust macro system which lends itself to creating
        domain-specific languages.</p>
{% endblock history %}


{% block uses %}
<p>Given the academic nature to Racket's inception, it is no
        surprise that Racket is most popular in the educational
        setting. In addition to university-level courses, the
        Racket community has worked hard to address Coputer
        Science education for K12 students. Projects like
        Bootstrap World and the Pyret language come from the
        Racket community.</p>
{% endblock uses %}
```

Updating the Views and URL Patterns

info

Customizing Your Webpage

If your webpage has custom content, be sure that you are making the appropriate substitutions regarding file name, view names, imports, etc.

Class-Based Views

Start by activating the `django` virtual environment.

```
conda activate django
```

Open the `views.py` file and add the class-based views for each of the new HTML files. Just like before, the views should inherit from the `TemplateView` class. Give the `template_name` attribute the value of the corresponding HTML file.

```
from django.views.generic import TemplateView

class HomePageView(TemplateView):
    template_name = "home.html"

class AboutPageView(TemplateView):
    template_name = "about.html"

class PythonPageView(TemplateView):
    template_name = 'python.html'

class FsharpPageView(TemplateView):
    template_name = 'fsharp.html'

class RacketPageView(TemplateView):
    template_name = 'racket.html'
```

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

```
black languages/views.py
```

URL Patterns

The last thing we need to do is link these new class-based views to a URL pattern. Add the new paths to the `urlpatterns` list. Remember, the value for `name` needs to match the value used in the `url` template tags.

```
from django.urls import path
from .views import HomePageView, AboutPageView, FsharpPageView,
        PythonPageView, RacketPageView

urlpatterns = [
    path("about/", AboutPageView.as_view(), name="about"),
    path('fsharp/', FsharpPageView.as_view(), name='fsharp'),
    path('python/', PythonPageView.as_view(), name='python'),
    path('racket/', RacketPageView.as_view(), name='racket'),
    path("", HomePageView.as_view(), name="home"),
]
```

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

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

Enter the following command in the terminal to launch the dev server. The click the link to open the website. Test the links on the main page to make sure each new pages loads as expected.

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

On each new page, you should see a tile, section headings, and two short paragraphs. There is no styling, so the site is not very appealing. Adding CSS with Bootstrap is our next topic.

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

```
conda deactivate
```