

# The Web, Crowds, and Clouds: Polling

## goals

- Incrementally develop a web app using Django frameworks and *Python*
- Learn how webpages are connected and accessed



## description of web App

Create an app as part of a website framework that lets people vote.

## Essential Questions

1. What makes the internet accessible to all?
2. How is abstraction in the programming language I am using managing complexity in my program?
3. How am I applying independent, cooperative, and collaborative strategies to find my own answers?

## essential Concepts

- Django Frameworks
- Cascading Style Sheets
- HTML
- Databases
- Conditionals and Modulo

## Resources

### Activity 3.2.1 Student Files

#### Part 1: Index View and Models

#### Part 2: Dynamic Sites

#### Part 3: Errors, Exceptions, and Style

## Website Poll App

At the end of this unit, you will create your own website to meet a need or solve a problem. Websites do many things for people, such as providing instant access to endless sources of information and collecting data from users around the globe. Examples of web applications within websites include polls, managing log-ins and passwords, forums, blogs, purchasing and checkout carts, and so much more.

In this activity, you will create a website that allows users to vote in polls on various questions that you create. You will create and personalize your own website. You will build a foundation to create websites in the future that meet the need of users other than yourself. You will host this site at a specific address on the internet so your friends and family—wherever they are, and whatever device they are using—will be able to enter the web address and actually use your polling site!

Initially, explanations of what the code is doing will be limited so that you can focus on getting a working polling website. As you develop each new feature in future activities, you will gain more understanding about how all the different pieces of code work together to create a worldwide presence on the web. Make sure you are being professional with your web development and that others whom you give your URL to know to be professional as well.

To create your website, you will use the Django web development framework within Cloud9. Django is a collection of *Python* code designed to make web development more convenient. You will need to follow each step carefully while creating toward the final poll app. As you get more familiar with the *Python* and Django processes, you will make more connections to the essential computer science concepts you have learned in this course. As you continue to develop each new feature in the following activities, you will gain more understanding as to how all the different pieces of the code are working together.

## Creating a Django Workspace in Cloud9

Cloud9 creates a directory structure and *Python* files so you don't have to! When you create a Django workspace, Cloud9 creates a directory containing the *Python* package with your workspace name. Using the workspace directory, you will create your website and **apps** within that website.

In this activity, the name of the workspace, project, and the app are similar, because you are only creating one app within this project and within the workspace. At the end of this unit, you will create one workspace with multiple apps in that one website workspace.

1. Navigate to Cloud9 and log in.

2. In your Cloud9 account, click **Create a new workspace**.
  - For Workspace name, use the naming convention set by your teacher.
    - Example: *pollsitelastnamefirstinitial*
    - Remember, Cloud9 uses only lowercase letters in workspace naming.
  - Under Choose a template, select **Django**.
  - Click **Create workspace**.

In your new workspace, you can see the Django directory structure when you click the **Workspace** tab on the left side of the Cloud9 interface. If you compare the folder names with your elbow partners' folder names or the images in this activity, you will see that they are all different. You all named your websites using *pollsitelastnamefirstinitial*, so everyone has their own name on their folders.

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

3. At the top of the Cloud9 window, click **Run Project**.
4. Your Django framework is now running on the Cloud9 server. You can share or view your website by entering your website's specific URL in any web browser. Click the web link that follows the text, "Your code is running at:"

After you click the **Run** button, a pop-up box appears (In this image it is green. The color may be different depending on the user settings and preferences). To view your website, click the URL link in the green box. Depending on your color theme in Cloud9, your interface may look a little different than the image below.

```
1 #!/usr/bin/env python
2 import os
3 import sys
4
5 if __name__ == "__main__":
6     os.environ.setdefault("DJANGO_SETTINGS_MODULE", "pollsite.settings")
7
8     from django.core.management import execute_from_command_line
9
10    execute_from_command_line(sys.argv)
11
```

Cloud9 Help  
Your code is running at <https://pollsite-projectleadtheway.c9users.io>

```
Applying auth.0001_initial... OK
Applying admin.0001_initial... OK
Applying admin.0002_logentry_remove_auto_add... OK
Applying contenttypes.0002_remove_content_type_name... OK
Applying auth.0002_alter_permission_name_max_length... OK
Applying auth.0003_alter_user_email_max_length... OK
Applying auth.0004_alter_user_username_opts... OK
Applying auth.0005_alter_user_last_login_null... OK
Applying auth.0006_require_contenttypes_0002... OK
Applying auth.0007_alter_validators_add_error_messages... OK
Applying sessions.0001_initial... OK
Performing system checks...

System check identified no issues (0 silenced).
April 10, 2017 - 19:00:21
Django version 1.9, using settings 'pollsite.settings'
```

- Click the URL. A new web page opens to display your first page. Now anyone with the web address can visit and see the page while the site is running.

As you begin to build your application web pages, visit them through your web browser often to see the how the changes you made in code impact the way the site looks and functions.

## It worked!

Congratulations on your first Django-powered page.

Of course, you haven't actually done any work yet. Next, start your first app by running `python manage.py startapp [app_label]`.

You're seeing this message because you have `DEBUG = True` in your Django settings file and you haven't configured any URLs. Get to work!

- Complete the three parts of the activity.

### Part 2: Index View and Models

### Part 3: Dynamic Sites

### Part 4: Errors, Exceptions, and Style

## Part 2: Index View and Models

### Creating a Polls App in Your Website

A website may have many apps, and you can reuse any app on different websites. The [Django framework](#) does a lot of the work of creating an app for you. It has a template of *Python* files and directories already created for making a polling app that you can use to ask questions and capture responses.

1. Click to open the [bash](#) tab at the bottom of the interface.

**Important:** Bash may also be called “the command line”, “the shell”, or “the command prompt”.

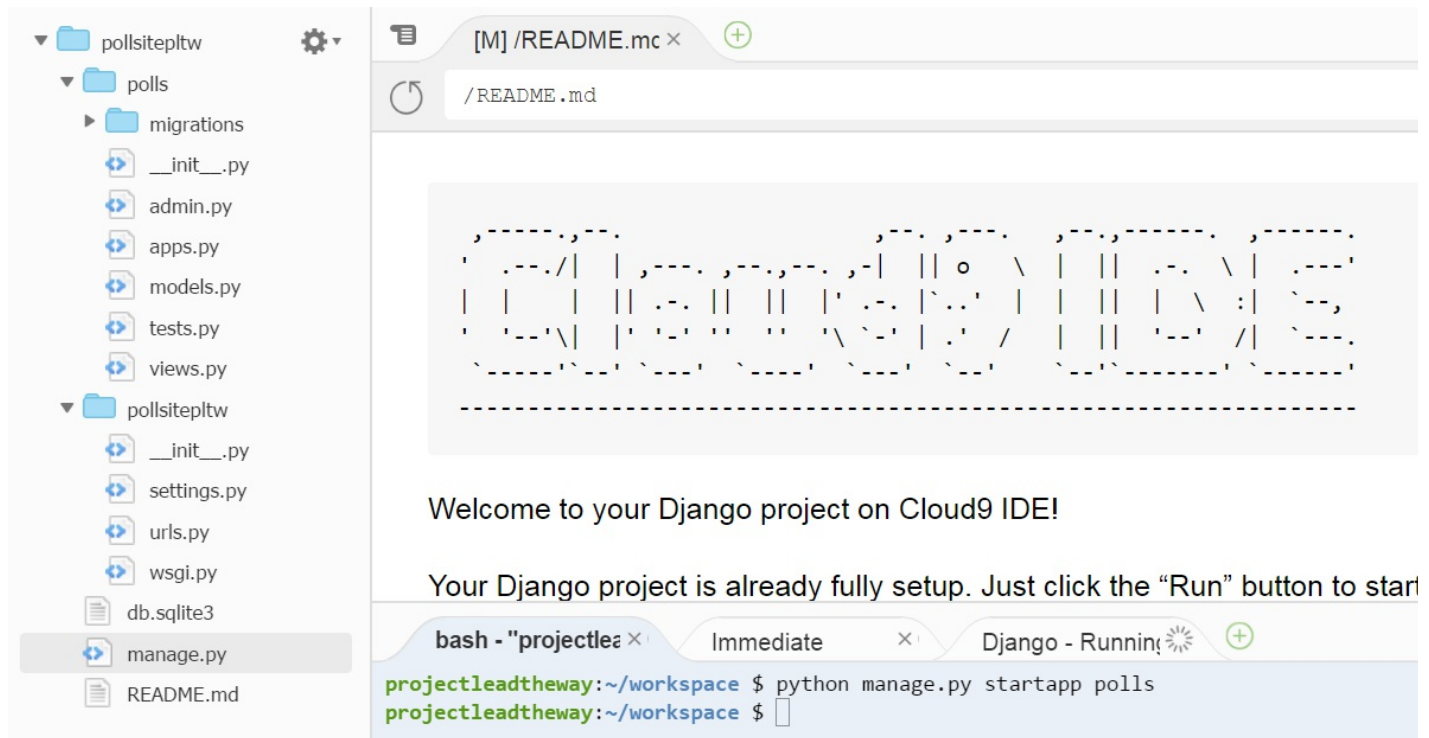
2. In the bash tab, enter the following command after the \$ and press Enter:

```
python manage.py startapp polls
```

This code creates the basic directory structure of the polls app, so you can focus on writing code rather than creating directories.

In your Cloud9 Workspace, you should now see:

- The top-level *pollsityourname*, which is the Cloud9 container or workspace.
- The second directory level folder *pollsityourname*, which contains the *python* package for your website.
- The second directory level folder *polls*, which contains the *Python* files for your polling app.



## Creating the Index View

Each type of view within a Django application has a specific purpose and page display. An index page is usually the default page people first visit on a website. You are now going to create an index view for your polls app. Later you will create other kinds of views, like a login page, a set of questions in a poll, a post on a web bulletin board, or a shopping cart page on a store's website.

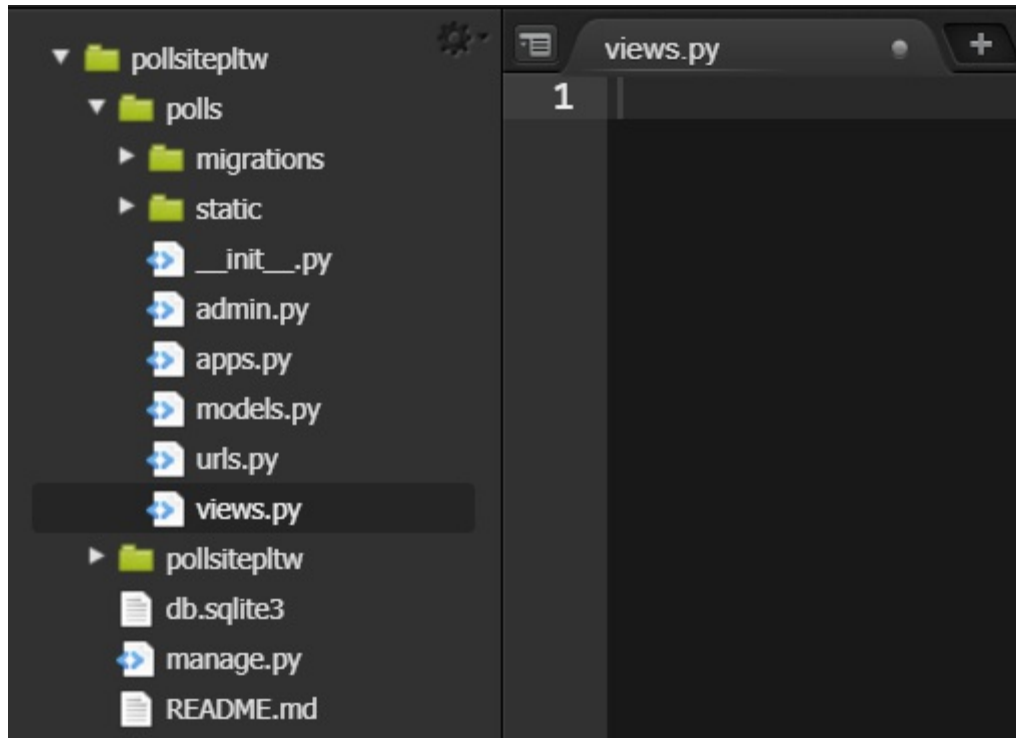
On the index page, you will add the message, "This is the index page for your polling site." so can you know the index is set up correctly and you are able to view it. The index page URL is what you send people to access your site.

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

3. In the directory structure, navigate to *polls/views.py* and double-click to open the file.

A new tab opens, *views.py*.



4. To create your page views in the *polls/views.py* file, define that page's function in *views.py*. By copying and pasting the code below into your *polls/view.py* file, you create what will display on your index page.

```
from django.http import HttpResponse

def index(request):
    return HttpResponse("This is the index page for your polling site.
```

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

5. Save the file after the code has been added.

**Important:** Changes do not take effect if you do not save. If you have many *python* file tabs open while you are working, be sure to save all of them. To see the updates, refresh the browser tab that displays your website after you save the *python* files.

## Creating an HTTP Response for When Your Index Is Called

After a web page receives a request to access its content, a server responds with an HTTP response message.

The *HttpResponse* code specifies that if a request is made of the index page (someone visits that

web address), an HTTP response message is provided. For example, the HTTP response for your site could say, “This is the index page of your polling site.” You have created a placeholder message to yourself that lets you know you are looking at the index page of your website. However, to make it visible, you need to add this page view to the list of recognized URLs for your site.

6. Create a new file in the polls folder and name it *urls.py*. To begin, right-click on the **polls** folder and select **New File**.

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

7. In the *polls/urls.py* file, include the following code and save the file:

```
from django.conf.urls import url
from . import views
urlpatterns = [
    url(r'^$', views.index, name='index'),
]
```

**Important:** There are now two URL config files—one under the top-level directory of *pollsite\_pltw* (*urls.py*) and a second *urls.py* under the polls folder (*polls/urls.py*). Make sure you are in the polls folder to open the *urls.py*.

8. In the directory structure, open *pollsitepltw/urls.py*.
9. In the site *pollsitepltw/urls.py* file, add code that points the index page of your project to the *poll/urls.py* you just created.
  - Add “include” to the two parts of the code as shown below.
    - The first “include” is after the import statement.
    - The second “include” adds a new function in the *urlpatterns*.

```
from django.conf.urls import url, include
from django.contrib import admin
urlpatterns = [
    url(r'^polls/', include('polls.urls')),
    url(r'^admin/', admin.site.urls),
]
```

**Important:** Make sure your indents are consistent so you do not get an error.

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.



Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

You now have wired an index view into the `urls` config file. This means that visitors to your URL will receive an HTTP response from your server containing the text that you just added: “This is the index page for your polling site.”

10. If it is not already running, run your project and view the website URL.

You will see a “Page not found” error. If you see a different error, make sure your web page is running and all the files have been saved.

**Important:** You don’t need to stop and rerun your project every time you make changes to your code. Saved changes are automatically updated in the website. You will need to refresh your website page in the browser to see the updates though.

11. In your browser address bar, add “/polls/” to the end of the URL.
  - Example -
  - `https://pollsitelastnamefirstintial-YourUserName.c9users.io/polls/`
12. The web page should display your placeholder message, which indicates you are on the index page of your polling site. Be sure the URL you use has your workspace and user name.



## PLTW DEVELOPER’S JOURNAL

1. Why did you need to add “/polls/” at the end of the web address?
2. Is Django a programming language?
3. What is an index page?

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

## Creating Models - Poll Questions and Choices

In your example poll, you will ask a user questions and allow them to see how many other users

voted for each answer choice. To do this, you need to create models in the database.

A model is a *Python* class that contains the data you are storing. This could be user login credentials or items in a shopping cart. Models define your database layout. In the polling site, the database model is based on presenting questions and associated answer choices and then storing the selections the users make.

13. Explore additional information on [Models in the Django Framework](#).

In your polls app, you will create two models: *Question* and *Choice*.

- A *Question* has two fields: question text and a publication date.
- A *Choice* has two fields: the text of the choice and a vote tally.
- Each *Choice* is associated with a *Question*.

14. Navigate to *polls/models.py* and add the following code:

```
# Create your models here.
class Question(models.Model):
    question_text = models.CharField(max_length=200)
    pub_date = models.DateTimeField('date published')
class Choice(models.Model):
    question = models.ForeignKey(Question, on_delete=models.CASCADE)
    choice_text = models.CharField(max_length=200)
    votes = models.IntegerField(default=0)
```

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

## Activating Models and Migrations

To be able to access your models, you need to tell the website server that the polls app is installed. A **migration** is a command in the Django framework that applies changes to the database of the website server.

15. Navigate to the *pollsitepltw/settings.py* file to add your polls app to the list of installed apps.
16. Add 'polls.apps.PollsConfig', as the first line in the list of *INSTALLED\_APPS*.

```
'polls.apps.PollsConfig',
```

**Important:** Make sure that you copy the comma at the end of the line and check the indentation levels.

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

17. Save the *settings.py* file.

**Important:** *INSTALLED\_APPS* is a list. You have initialized lists in this manner before in *Python*.



### PLTW DEVELOPER'S JOURNAL

1. What does the code within brackets mean following *INSTALLED\_APPS*?
2. At what index can the element you added to *polls.apps.PollsConfig* be referenced by the program?
3. What *Python* statement would you use to access the element?

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

18. To tell your app that you added a new model to your code, you will now make a migration. In a bash tab, enter the following command:

```
python manage.py makemigrations polls
```

**Important:** If you do not save the *settings.py* and *models.py* files, Django will not know they are there and will return an error message.

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

You have now migrated your new models, and Django named this migration *001\_initial.py*.

## Setting Up the Database for the Classes

Structured Query Language (SQL) is a standard computer language for database management

and data manipulation. SQL is used to get, put, update, and otherwise modify data. SQL operates at a lower level of abstraction that you do not need to worry about right now, because with a simple command in Django, your database will be set up without you needing to learn SQL.

19. Enter this command in bash to set up your Question/Response database:

```
python manage.py sqlmigrate polls 0001
```

Django sets up tables in your database named "*polls\_choice*" and "*polls\_question*".

20. Explore the explanations of what the Django framework does when you enter the *sqlmigrate* command in bash.

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

## Django Admin Portal

There are several ways to set up your data for testing before users enter any of their own data. For now, you will set up and use the Django Admin web portal, a [user interface](#) (UI) for managing your data.

You need to create a Django admin account (also known as a superuser), username, and password to manage the database in the portal. Remember to store your user credentials in a secure location so that you will be able to look them up in case you forget them.

### Setting Up the Django Admin Portal for Your Website

21. To create your admin account, enter the following command in bash:

```
python manage.py createsuperuser
```

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

**Important:** As you type a password, be aware that your password will be hidden. Django will return messages if your password does not meet the required security criteria.



**PLTW DEVELOPER'S JOURNAL** Record your superuser account information. Using the same information for all the website apps you create will help you remember the information.

22. Make sure your project is running and then view the website by adding “/admin/” to the end of the http url. If your website page is still open, you can just replace “/polls/” with “/admin/”.
23. Log in using the credentials that you created. You will be taken to a page that is titled Django administration.

Secure | <https://pollsite-projectleadtheway.c9users.io/admin/>

Django administration

Username:

Password:

Log in

## Django administration

### Site administration

#### AUTHENTICATION AND AUTHORIZATION

Groups

[+ Add](#) [Change](#)

Users

[+ Add](#) [Change](#)

## Data Types: Registering Your Models

Before you can create and edit the actual Questions and Choices, you need to tell Django what data types to expect as inputs and what data types you want to be returned. To view and alter your poll data within the admin portal, you will go back to Cloud9 and add some code to *polls/admin.py*.

**Important:** As you create your own website apps in Django, you may run into errors when you start trying to add objects in the admin view. If you get one of these errors, you will need to go back to your *Python* files to make updates.

24. In the Cloud9 directory structure, open the *polls/admin.py* file, add the following code, and save.

```
from django.contrib import admin
from .models import Question
# Register your models here.
admin.site.register(Question)
```

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

25. Refresh your Django administration browser page. You should see the *POLLS* app and *Questions* class added to your UI.



**PLTW DEVELOPER'S JOURNAL** What do you think the purpose of the *admin.py* file is?

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

26. Under *POLLS*, click **+ Add** for the Questions data. Respond to the prompts:
- For Question text, our first example poll question will be, "What has been your favorite activity in CSE so far?"
  - For Date published, select **Today** and **Now**. These shortcut links make modifying the database easier for administrators.
27. Click **SAVE**. A Questions page opens where you can see your newly created Question.

**Important:** If you get a table error message while trying to add objects, use the following code in bash to sync databases:

```
python manage.py migrate --run-syncdb
```

28. Create a second question: "What is your favorite programming language?"
29. Navigating to your index page, example - <https://pollsitelastnamefirstintial-YourUserName.c9users.io/polls/>

The site will not show anything different. You may have created the models, a super user, and made some posts, but you need to direct the code how and when to get and display those posts.

30. Continue to the next part of the activity.

## Part 2: Dynamic Sites

## Part 3: Dynamic Sites

### Creating Dynamic Web Pages with Variables, Modulo, and Conditionals

Currently, you have only one view for your website that displays the same message regardless of what content is added to your database. This view is found in the *views.py* file. In this part of the activity, you will allow your user to view information from your database, making your website behave dynamically.

First, you will modify the index page for your poll, so that it shows a list of all of the Questions in your database. To test this functionality, you need to make sure that you have multiple Questions in your database.

1. Check in your admin portal to make sure you have two questions. Add another question if you do not have at least two.
2. Open the *polls/views.py* file and add the following *import* statement at the top of your code:

```
from .models import Question
```

The code statement allows the *views.py* file access to your Question data type defined in your *models.py* file.

3. Replace the body of the *index()* definition with the following code:

```
latest_question_list = Question.objects.order_by('-pub_date')[:5]
output = ', '.join([q.question_text for q in latest_question_list])
return HttpResponse(output)
```

The code sets a new *output* variable and replaces the temporary message you created while testing your index page.

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

4. Save your code and visit your site with “/polls/” at the end of the URL.
  - Verify that you see a comma separated list of the text from each of your Questions.



What is your favorite activity in CSE so far?, What is your favorite programming language?

- If your code does not run, make sure that your indentation levels are correct.

Your website can now access the Questions database to display the questions on your page. Right now, you have the questions presented as a list, in order of publication date. As you move along, you will make the website more attractive and user friendly.

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

## Making a Dynamic Website

In your final poll app, each question will be displayed on its own page. You will now add a feature that shows each question's number. You will also add suffixes of ordinal numbers (st, nd, rd, th) to the question numbers.

At the conclusion of setting up your conditional, you will see the following outputs:

- On page 1, "You're looking at the 1st question".
- On page 2, "You're looking at the 2nd question".
- On page 3, "You're looking at the 3rd question".
- On page 4, "You're looking at the 4th question".

Using ordinal indicators makes reading the page number more natural. To add the ordinal indicator suffixes, you need to use, combine, and switch between different data types (*int* and *str*). You will also use formatting operators to place the values from variables within a string.

5. In *polls/urls.py*, add a call to the *urlpatterns* list. Add:

```
url(r'^(?P<question_id>[0-9]+)/$', views.detail, name='detail'),
```

The provided code allows a user to view your URL with different numbers at the end. The details for each question are on their own page. That is why you are calling it the "detail" view.

**Error Alert:** Adding the code above will direct the program to look for a *detail* function in the *views.py* file. You will get an error until you define a *detail* function in the *views.py* file.

6. In *polls/views.py*, create a new function called "detail", as shown below.

```
def detail(request, question_id):  
    return HttpResponse("You're looking at question number %s." % ques
```

**Important:** *question\_id* is an argument to the detail function and will contain a string value at runtime.

7. Switch to your website tab and add `/polls/1` at the end of the web address.

Do you see a placeholder for your first question?

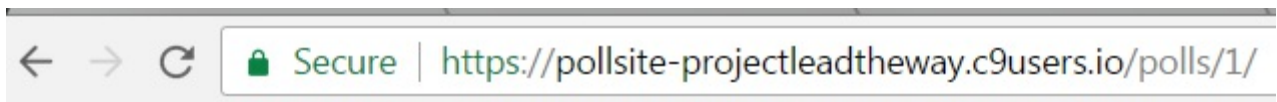
So far, the code you've added to your website has been to modify a list, call functions, or define functions. You are about to modify the `detail()` function in `polls/views.py` to dynamically change text based on the numerical value of the `question_id`. As you do this, you will use some of the control flow statements you used before, like conditionals and loops. Take a look at the `HttpResponse` that's returned when a user visits the `/polls/1` page.

## Variables in Conditionals

8. Review information about formatting operators.

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

9. To modify the `detail()` function, change the `HttpResponse` to read, "You're looking at the 1st question."
10. Check the `/polls/1` page to see the message. Is it what you expected? Keep modifying until it reads like:



11. Add conditionals that will compare the `question_id` variable and return the correct `HttpResponse` based on what page the viewer is looking at. Here is a sample conditional for the first question:

```
def detail(request, question_id):  
    if question_id == 1 :  
        return HttpResponse("You're looking at the %sst question." % c
```

12. Save, run, and visit the `/polls/1/` page in your browser. You will see an error message.

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

13. In `polls/views.py`, add the following line of code under your `def detail(request, question_id)`, which will change `question_id` from a `str` to an `int`.

```
question_id = int(question_id)
```

14. Use conditionals to create an `HttpResponse` for question number 2. If you view it, you will notice that it does not display the way people would say it in natural language.
15. Add `elif` and `else` clauses in a chained conditional statement to correctly name the numerical formatting for numbers in your `HttpResponse`.

The English language uses 2nd (not 2st) and 3rd (not 3st ).

16. To test your function, temporarily assign other values (such as 2, 3, or 4) to `question_id`. You will address question numbers greater than 4 later in this activity.
17. Use Modulo to help address the pattern for identifying numbers.

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.



## PLTW DEVELOPER'S JOURNAL

1. What if your poll had more than four questions? What if it had 50 questions?
2. What computer science concepts could you use to get the correct suffix (st, nd, rd, th) attached to the correct question number so your messages read correctly?

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

## Hypertext Markup Language (HTML)

When you view the `/polls/` address, the presentation of your questions is very basic (in a comma-separated list) instead of laid out in a more professional way.

To modify how the page is presented and change the comma-separated list into an unordered (bulleted instead of numbered) list, you need to use **HTML**. HTML stands for Hypertext Markup Language. It is a web development language that describes the structure of a website. When you enter a URL in your browser address bar, the data coming from the server to your computer often contains HTML.

The HTML tells your browser how to display information using specific tags. Tags have keywords enclosed in left and right angle brackets (< >) that “mark up” how to display the words contained within the opening and closing tags.

In this part of the activity, you will use *Python* to dynamically generate these HTML tags as *Python* strings, which will change the appearance of your website. Using HTML tags in strings allows you to use any *Python* code you like to manipulate these strings including conditionals and loops.

18. Explore more information on [Python strings](#).

## HTML for a Professional Appearance

Right now the look of your polls questions is not very professional, because they are presented in a simple comma-separated list. The HTML code provided in the next step will format the questions in a bulleted list. The <ul> and <li> tags create a more professional looking bulleted list. You will learn more about HTML in future activities.

19. Open the *polls/views.py* file and replace the code in the *index()* function with the following code:

```
def index(request):
    latest_question_list = Question.objects.order_by('-pub_date')[:5]
    output = "<ul>"
    for question in latest_question_list:
        output = output + '<li><a href="/polls/' + str(question.id) + ' '
    output = output + "</ul>"
    return HttpResponse(output)
```

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

20. In your web browser, navigate to the */polls/* URL and view the index page to your website. Your questions should be formatted in a bulleted list now.



## Adding Choices and Voting

Now you will modify your site so that it allows the user to vote using a web form. First, you will add a

voting page and then a page where users can view the results of the poll. To add the voting page, complete the following steps.

21. In *polls/urls.py*, add your new view, be sure to check the indents and make sure you save both files after you add the code.

```
url(r'^(?P<question_id>[0-9]+)/vote/$', views.vote, name='vote'),
```

**Error Alert:** Adding the code above will direct the program to look for a *vote* function in the *views.py* file. You will get an error until you define a *vote* function in the *views.py* file.

22. In *polls/views.py*, add the following function definition at the end of your code for a voting view.

```
def vote(request, question_id):  
    return HttpResponse("You're voting on question %s." % question_id)
```

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

23. To verify that your new voting view is recognized by the web server, visit the URL for your site with */polls/1/vote* at the end.

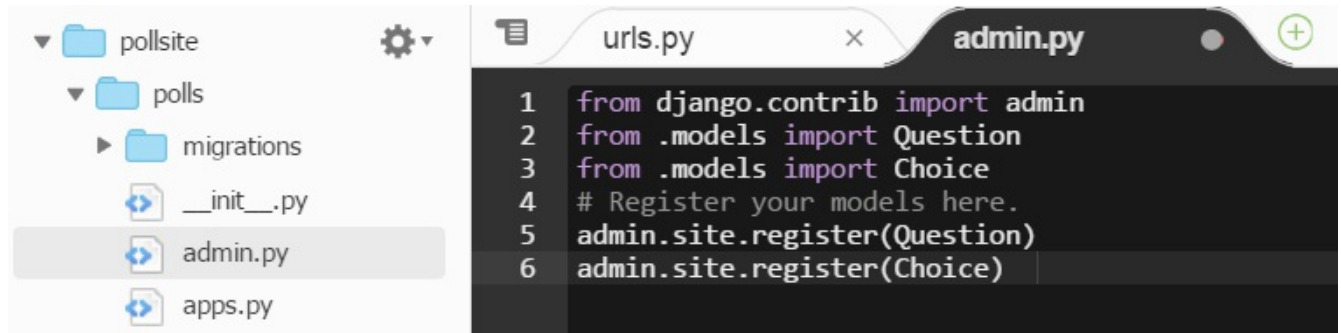


You're voting on question 1.

In Part 1 of this activity, you created Questions. You now need to create Choices for your poll.

24. In *polls/admin.py*, add your new model Choices using the following code in the appropriate places of the file:

```
from .models import Choice  
  
admin.site.register(Choice)
```



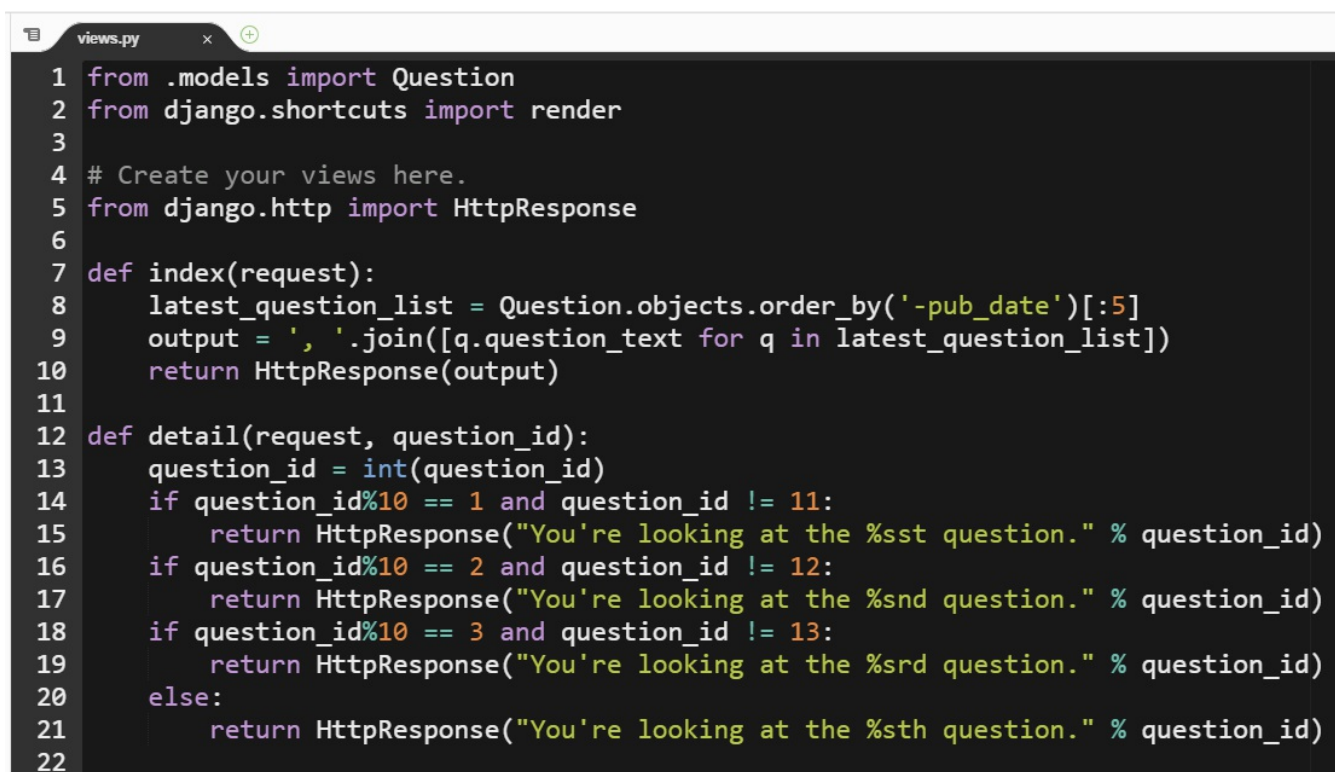
25. Navigate to the admin portal for your site (your URL with `/admin/` at the end) and add a few answer choices to each question.

**Important:** If “Choices” does not show up under POLLS, what bash command did you forget to do?

You may have trouble knowing which choice is attached to which question. Click a question object to see the question, so you know what you are attaching choices to.

You will not be able to view your questions or choices in the web page until you deal with the conditionals that direct you to the right page. It is a good idea to create a local variable that will store the `question_id` value and that will be tested in the conditional to determine which question to display.

26. Compare your `/polls/views.py` file with the example version below:



## Creating a Local Variable

Each time the program runs, it checks the variable `question_id` to see whether it meets the criteria of

the first condition, and if it does not, then it moves to the next part of the conditional. If *question\_id* does not meet any of the *elif* conditionals in the chained conditional, then it moves to the *else* part of the conditional.

At the end of each conditional, there is an *HttpResponse* with the appropriate ordinal indicator (st, nd, rd, th) message if the conditional is true. Rather than return an *HttpResponse* for each conditional, you are going to create a new variable called *output*, which stores the string from the correct conditional and passes that to the *HttpResponse* at the end of the function.

By creating the variable *output* before the conditionals, the variable's **scope** allows it to be accessed anywhere in the function.

27. Define a variable *output* as a blank string inside the *detail()* function before the chained conditional. This way the variable's scope allows it to be accessed anywhere inside or outside the conditional, but within the function.

```
output = ""
```

28. Modify your current detail view so that, instead of returning an *HttpResponse* directly from within your conditionals, it stores the strings you created within those *HttpResponses* to the output variable. This way you can modify output before creating an *HttpResponse* with it later. Example for the first conditional:

```
if question_id % 10 == 1 and question_id != 11:
    output = "You're looking at the %sst question." % question_id
```

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

29. Now modify your remaining conditionals.
30. As the last line of your detail view function, add a return statement:

```
return HttpResponse(output)
```

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

31. Navigate back to your pages and check that the correct ordinal identifiers still appear on each page.



# Coding for Non-existent Pages

Right now, your web browser will direct you to any page you enter and put the correct suffix in your message, but the page is not connected to the items in your database yet (Questions and Choices). If a user attempts to navigate to a question that does not already exist in the database, they will likely see a page of code that makes no sense to them. To be professional, you would prefer to give them a standard error message.

32. Add the following import to *polls/views.py*. This displays an appropriate error (404), if the Question is not found.

```
from django.shortcuts import get_object_or_404
```

33. To store the data from the Question that the user is requesting in a variable identified by *question* (or return a 404 error if the question is not found), add the following to your *detail()* function call:

```
question = get_object_or_404(Question, pk=question_id)
```

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

## Applying HTML Formatting

You are now going to format your questions using HTML tags. The HTML tags provide basic formatting structures like headers, paragraphs, and forms. You will use HTML to display the message on the page formatted the way you want it to look. To do this, you need to define another variable in your *polls/views.py detail()* function.

34. After your chained conditional in *detail()*, but before the return statement, add the following line of code to customize the detail view with actual information from the database.

```
output = "<h1>" + question.question_text + "</h1><p>" + output + "</p>"
```

You're taking the output variable, concatenating it with HTML formatting, and redefining the output variable to output the updated format.



**PLTW DEVELOPER'S JOURNAL** Where do you think *question.question\_text* is coming from?



Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

35. View your website for the poll questions to see the updated view based on the HTML you added.
36. Take a moment to compare the code you added in Cloud9 with the way your website looks. Match up and record how the output on the web page lines up with the code you added.

```
output = "<h1>" + question.question_text + "</h1><p>" + output + "</p>"
```

← → ↻ Secure | <https://pollsite-projectleadtheway.c9users.io/polls/1/>

## What is your favorite activity in CSE so far?

This is the 1st question.

37. Visit [w3schools](https://www.w3schools.com/html/html_form_elements.asp) or other internet search results to help you determine what purpose the h1 and p tags serve.



**PLTW DEVELOPER'S JOURNAL** Record a few HTML tags that you might use in the future for formatting your websites and apps.

## Connecting Questions and Choices

At this point, your database has the questions and the answer choices, while the website uses HTML to display everything when you go to specific addresses. Now it is time to connect the answer choices to the specific questions, so your users can vote on them. Adding an HTML form to your view file will allow a user to see the answer choices associated with each question (as well as a button to submit a vote).

38. In the *detail()* function, just before the return statement, add the following code and verify that the indents show all this code inside the *detail()* and that the appropriate code is indented within the *for* loop.

```
output = output + '<form action = "/polls/' + str(question_id) + '/vote'
count = 1
for choice in question.choice_set.all():
    output = output + '<input type="radio" name = "choice" id="choice'
    output = output + '<label for="choice' + str(count) + '">' + choic
    count += 1
    output = output + '<input type="submit" value = "Vote" /></form>'
```

39. Comment the code in Cloud9.

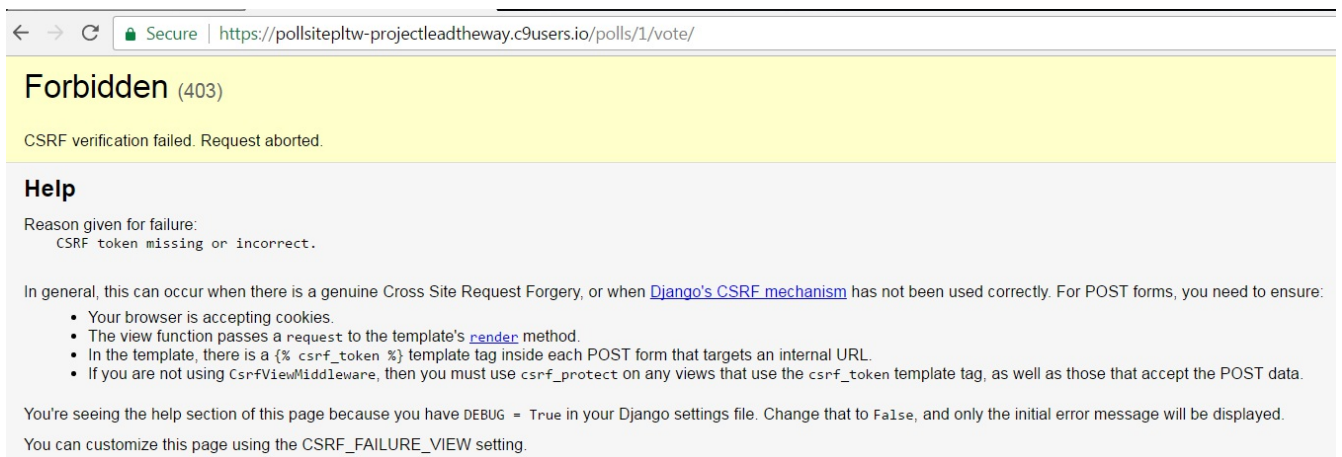
40. Navigate to your URL with `/polls/1` at the end. You should see something like:



A screenshot of a web browser window. The address bar shows a secure connection to `https://pollsite-projectleadtheway.c9users.io/polls/1/`. The main content area displays a large heading "What is your favorite activity in CSE so far?". Below the heading, it says "This is the 1st question." There are two radio button options: "Choice A" and "Choice B". At the bottom of the form is a "Vote" button.

41. Try to vote.

You get an error because of something called a cross-site request forgery (CSRF).



A screenshot of a web browser window showing a 403 Forbidden error. The address bar shows `https://pollsitepltw-projectleadtheway.c9users.io/polls/1/vote/`. The error message is "Forbidden (403)" with the subtext "CSRF verification failed. Request aborted." Below this is a "Help" section titled "Reason given for failure:" with the message "CSRF token missing or incorrect." It then explains that this can occur when there is a genuine Cross Site Request Forgery, or when Django's CSRF mechanism has not been used correctly. It lists three bullet points: "Your browser is accepting cookies.", "The view function passes a request to the template's `render` method.", and "In the template, there is a `{% csrf_token %}` template tag inside each POST form that targets an internal URL." It also mentions that if not using `CsrfViewMiddleware`, one must use `csrf_protect` on any views that use the `csrf_token` template tag, as well as those that accept the POST data. At the bottom, it says "You're seeing the help section of this page because you have `DEBUG = True` in your Django settings file. Change that to `False`, and only the initial error message will be displayed. You can customize this page using the `CSRF_FAILURE_VIEW` setting."

CSRF errors are not the only types of errors you will get. Depending on user input, other errors might occur. To help limit errors from user input, add exception handling to your website.

42. Continue to the final part of the activity to see how to address these errors.

### Part 3: Errors, Exceptions, and Style

# Part 4: Errors, Exceptions, and Style

## Cross-site Request Forgery

Cross-site request forgery (CSRF) is a type of malicious activity that uses code hidden from a user to try to accomplish a task through a website. Some servers address the cybersecurity concern by making sure that Get requests are only getting content, not trying to change data on the server.

**Important:** The example provided here is based on adding the CSRF token to the polls app, but can be applied to other apps in Django through similar methods.

1. If you get an error because of cross-site request forgery (CSRF), you will need to import *csrf\_exempt* from Django to help bypass the cybersecurity concern.
2. In the *imports* section of your *polls/views.py* file, add the following import statement.

```
from django.views.decorators.csrf import csrf_exempt
```

3. In *polls/views.py*, add “@csrf\_exempt” on its own line immediately above each of your function definitions.
4. Along with the other import statements at the top of the *views.py* file, add another import from *.models* to import Choice in addition to Question.
5. Add *HttpResponseRedirect* as another import within the from *django.http* statement.

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

6. In *polls/views.py*, add the following results function to test the site and get confirmation that the vote was cast.

```
@csrf_exempt
def results(request, question_id):
    response = "You're looking at the results of question %s."
    return HttpResponseRedirect(response % question_id)
```

**Important:** This is a temporary message much like the other ones you have used to help you keep track of what page you are on. Eventually, this will display a list of choices for the question accompanied by a list of choices for each vote choice.

Refer to your downloadable resources for this material. Interactive

7. You need to register the URL pattern for the results pages in *polls/urls.py* as you did with the detail, vote, and index views. Add the following line of code to *polls/urls.py*:

```
url(r'^(?P<question_id>[0-9]+)/results/$', views.results, name='result
```

**Note:** For more information on CSRF in Django, view the [Django documentation on CSRF](#). The Django site mentions templates, which will be used in later activities.

## Programming an Exception

Sometimes users may think they selected an answer, but did not. Or a user enters “five” instead of “5”. These user inputs can cause a program to appear not to function, even though it functions properly with the correct types of inputs. You can handle these situations by programming an exception. Once you have programmed that exception, you can add instructions to follow when that exception occurs.

### Exception Handling in the Polling App

8. In *polls/views.py*, replace the temporary *vote()* function with the following code to handle voting exceptions that occur when no answer choice vote was selected:

```
@csrf_exempt
def vote(request, question_id):
    question = get_object_or_404(Question, pk=question_id)
    try:
        user_choice = question.choice_set.get(pk=request.POST['choice'])
    except (KeyError, Choice.DoesNotExist):
        return HttpResponse("No choice selected.")
    else:
        user_choice.votes += 1
        user_choice.save()
    return HttpResponseRedirect("/polls/" + str(question_id) + "/resul
```

9. Visit your website URL with */polls/1* at the end and test the following:
- ☐ Select one of the answer choices.
  - ☐ Click the **Vote** button.
  - ☐ Verify that you are taken to the results page and see, “You're looking at the results of question 1.”
10. If you tested the code and it worked as expected, replace the temporary *results()* function with the following.

```
@csrf_exempt
def results(request, question_id):
    question = get_object_or_404(Question, pk=question_id)
    output = '<h1></pre>' + question.question_text + '</h1><ul>'
    for choice in question.choice_set.all():
        output = output + '<li>' + choice.choice_text + ' >>> ' + str(
    output = output + '</ul>'
    return HttpResponse(output)
```

Using the try, except, else pattern, it is possible to make a programs execute functions only when the input is appropriate. Otherwise, it is possible to use loops or conditionals to direct users toward acceptable input, without stopping the program from running.

## CSS

You can customize the appearance of your site as you see fit. For now, you will walk through the process of styling your page views with a **Cascading Style Sheet**(CSS) file. Developers use CSS to connect different HTML tags to different kinds of consistent appearances, or styles. By using CSS, you have the ability to quickly update the way all pages look by changing one file instead of several. If you set up the site with a blue background and later decide you would rather have green, it is a fast fix in a single location.

### Setting Up a CSS for a Website

**Important:** The example goes through adding the CSS to the Polls app, but the process would be the same in other apps, just with different names and file references.

11. In your polls directory, create a new directory folder named “static”.
12. Within the static folder, create a new directory folder named “polls”.

The naming may seem repetitive, but if you had multiple apps in your site, you would use this directory to distinguish which app this CSS is formatting. Currently, you are creating a CSS for the polls app of this website.

13. Download **style.css**.
14. Upload it in the polls directory folder in the static folder.



**PLTW DEVELOPER’S JOURNAL** What do you notice about the color scheme of the .css file?

You may notice that the color scheme used in the *style.css* file is different. That is because you are in a different file type, and .css files have different color schemes than .py files do. This color convention can help you identify at a glance what programming language you are working in.

15. Add an import statement for static in the *polls/views.py* file:

```
from django.contrib.staticfiles.template_tags.staticfiles import static
```

16. In each of the view functions in *polls/views.py*, append and concatenate the following code to the front of your output string before its return as an *HttpResponse*.

```
'<head><link rel="stylesheet" href="' + static('polls/style.css') + '"
```

17. Test your site to make sure the stylistic elements have been applied.

```
10 # Create your views here.
11 @csrf_exempt
12 def index(request):
13     latest_question_list = Question.objects.order_by('-pub_date')[:5]
14     output = '<head><link rel="stylesheet" href="' + static('polls/style.css') + '"></head> <ul>'
15     for question in latest_question_list:
16         output = output + '<li><a href="/polls/' + str(question.id) + '/'>' + question.question_text + '</a></li>'
17     output = output + "</ul>"
18     return HttpResponse(output)
```

```
55 @csrf_exempt
56 def results(request, question_id):
57     question = get_object_or_404(Question, pk=question_id)
58     output = '<h1>' + question.question_text + '</h1><ul>'
59     for choice in question.choice_set.all():
60         output = output + '<li>' + choice.choice_text + '</li>' + str(choice.votes)
61     output = '<head><link rel="stylesheet" href="' + static('polls/style.css') + '"></head>' + output + '</ul>'
62     return HttpResponse(output)
```

```
20 @csrf_exempt
21 def detail(request, question_id):
22     question = get_object_or_404(Question, pk=question_id)
23     question_id = int(question_id)
24     output = ""
25     if question_id % 10 == 1 and question_id != 11:
26         output = ("You're looking at the %sst question." % question_id)
27     elif question_id % 10 == 2 and question_id != 12:
28         output = ("You're looking at the %snd question." % question_id)
29     elif question_id % 10 == 3 and question_id != 13:
30         output = ("You're looking at the %srd question." % question_id)
31     else:
32         output = ("You're looking at the %sth question." % question_id)
33     output = "<h1>" + question.question_text + "</h1><p>" + output + "</p>"
34     output = output + '<form action="/polls/' + str(question_id) + '/vote/" method="post">'
35     count = 1
36     for choice in question.choice_set.all():
37         output = output + '<input type="radio" name="choice" id="choice' + str(count) + '" value="' + str(choice.id) + '" />'
38         output = output + '<label for="choice' + str(count) + '">' + choice.choice_text + '</label><br />'
39         count = count + 1
40     output = '<head><link rel="stylesheet" href="' + static('polls/style.css') + '"></head>' + output + '<input type="submit" value =
```

## Debugging Tips

Your pages should look like this:

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

18. If your site does not look like the images above, try tracing your steps, as follows:
- Start with your *index()* function and page first. After that is working, move on to other pages. The index page URL is the one with only */polls/* at the end.
  - You do not need to break up or separate the code provided in this section to make

the styles work.

- ☐ Check that functions, classes, and file names are spelled and capitalized the same in all the files that they appear.
- ☐ What does the append look like in other places where you added HTML?
- ☐ Are you appending where the style is not likely to be overridden—before the output is returned?
- ☐ Did you name all the files and folders exactly as described in the directions?
- ☐ Did you put the folders in the correct places in the directory?

## Conclusion

1. Identify a few community businesses or organizations that you might like to work with to create a website for the end-of-unit problem.
2. How were functions used in this activity?
3. How were lists used in this activity?
4. How did you interpret and respond to the essential questions? Capture your thoughts for future conversations.