Remember to check existance parameter before using it !

Homework 2: A Django Calculator

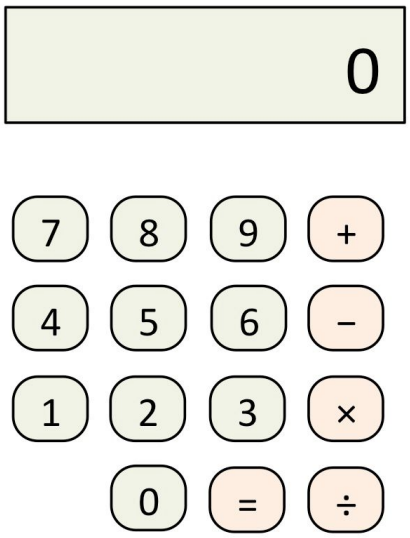
Due date: September 13, 2018 at 11:59pm

This assignment introduces Django, a popular web framework written in Python, which you will use for the next series of homeworks. In this homework you will build a working 4-function calculator, implementing the calculator functions by sending web requests to a Django application running on the Django development web server.

The learning goals for this assignment are to:

- Demonstrate a basic understanding of HTML and CSS, as with Homework 1.
- Learn how requests are routed and processed in a typical MVC web application.
- Demonstrate an understanding of typical data interactions between a web client and a web server, including the difference between HTTP GET and POST parameters and the use of HTML forms as input to a stateless web application.
- Demonstrate thorough, manual validation of HTTP request parameters by a web application.
- Gain hands-on experience with Django, a production-quality web framework.

Part 1: Building a calculator webpage



To start, design the user interface for a simple calculator. Your solution should roughly resemble the image to the right:

- The calculator will contain 15 clickable buttons: a button for each digit 0 to 9, and **+**, **-**, ×, ÷, and **=**.[1]
- Buttons must have rounded corners.
- Digit buttons must be a different color than other buttons.
- Your calculator should display a single integer value that the user may not directly edit.
- When the user hovers over a button, its color must change.

Part 2: Implement your calculator

This section describes the behavior of your calculator, which is a simple four-function calculator:

- A “current value” is displayed at all times, as a single integer. This value is initially zero (0).
- The calculator performs integer math (e.g. 9 ÷ 4 = 2).
- If the user attempts to divide by 0 or send malformed input, reset the state and display an error message until the next button is clicked.
- There is no operator precedence. Execute the operations in the order they are encountered.

Here are a few examples of the expected behavior of the calculator; you may use these as test cases:

[1] The HTML character entities for these symbols are `&plus;` `&minus;` `&times;` `&divide;` and `&equals;`.

Button Pressed	2	4	+	1	9	×	2	0	-	1	0	=
Display	2	24	24	1	19	43	2	20	860	1	10	850

Button Pressed	6	8	-	9	7	=	1	5	÷	1	3	=
Display	6	68	68	9	97	-29	1	15	15	1	13	1

Additional requirements

Your solution must also follow these requirements:

- You must have some design (or basic theme) using CSS and HTML. CSS must be in one (or more) static file(s). You may include images, also in one or more static files.
- You must write any CSS and HTML content yourself, and your application must run on the web server. The web browser must simply submit requests to the web server (after each button click) and display the response. You may not use any external libraries (such as Bootstrap or jQuery) or use JavaScript for this homework assignment. The client must not perform any processing of the calculator data.
- Your server-side web application must be *stateless*: the server may not store any data between web requests. To meet this requirement your server will need to send extra data (in addition to the displayed calculator value) to the client with each response. The client will need to resubmit that extra data with its next request. You may not use sessions, store data in-memory or use a database at the web server.
- The empty URL (e.g. http://localhost:8000/ for the Django development server running on port 8000) must route to your application's main page in an initial (zero) state.
- You may not use the Python `eval()` function in the implementation of your calculator.
- Your calculator must be able to support numbers up to ±1,000,000. We will not test with numbers beyond that range. You do not need to handle overflow or outputs wider than -1,000,000.
- Your application should run with Django 2.1.x, which the graders will use to evaluate your work.
- Your application should not crash as a result of any input sent to the web server or as a result of any user actions. Note that the user can send arbitrary data (malformed or otherwise) to the server, and you must anticipate and validate these requests.
- You must run your solution (as viewed from a browser) through the W3 HTML validator (http://validator.w3.org/) and CSS validator (http://jigsaw.w3.org/css-validator/) and have no errors; avoid inline CSS; and use semantically correct tags. For example, use the `<button>` tag for buttons, not the `<div>` or the `<image>` tag.
- In your `README.md` file, explain why your app generates either GET or POST requests. Also cite any external resources used and any additional notes you would like to convey to your grader.

Evaluating your solution

As usual, for substantial credit *your solution must clearly demonstrate the learning goals for this assignment*, which are described above in the introduction. We will grade your work approximately as follows:

Committing your work [10pt]

As with the previous homework, we will evaluate your version control use. Please remember that good version control use typically means (1) incremental, modular commits with (2) descriptive and useful commit messages.

Specification fulfillment [20pt]

Validation [20pt]

We will test your Django application with a variety of inputs and requests, including requests that cannot be sent by the buttons on your calculator. Your calculator must validate all input and generate an appropriate response.

Routing and configuration [10pt]

General requirements, appropriate use of technologies [40pt]

Turning in your work

Your submission should be turned in via Git and should consist of a Django project/application inside your `homework/2` directory. Name your project **webapps** and your application **calculator**. E.g.:

```
[YOUR-ANDREW-ID]/homework/2/
     webapps/
         settings.py
         urls.py
         [etc.]
     calculator/
         static/
         templates/
         models.py
         views.py
         [etc.]
     manage.py
     README.md
```

Other well-organized directory structures are fine, as long as we can easily find your work and your solution's components are in appropriate locations.