Assessment: Flask 2

Download code <../flask-2.zip>

Warning: Assessments

Remember, assessments are meant to be completed **by you**, not as a shared exercise with friends or other members of your cohort.

All code submitted should be **written by you**. If you incorporate code from elsewhere, it must be clearly specified.

Your solution must be submitted by email by Sunday evening at 9PM.

Please do not put your assessment on GitHub.

Part 1: Conceptual

Answer the following questions inside the ***conceptual.md*** file.

Part 2: Practice Problems

Solve the following problems **in JavaScript**. Make sure the tests pass for each of these.

snakeToCamel

Python programmers write variable names in ***snake_case***, where each word is lowercased and joined by underscores. JavaScript programmers write variable names in ***camelCase***, where the initial word is lowercase, and other words are capitalized and jammed together.

Given a variable name in ***snake_case***, return a string with that variable name written in ***camelCase***.

For example:

```
snakeToCamel("awesome_sauce"); // "awesomeSauce"
snakeToCamel("a_man_a_plan"); // "aManAPlan"
snakeToCamel("HOW_ABOUT_NOW?"); // "HOWABOUTNOW?"
```

Star Out Grid

If we have a grid like this:

```
A B C
D * E
F G H
```

We should take every row and every column that contains a star (`*`) and star-out that row and column. So that grid would turn into:

```
A * C
* * *
F * H
```

A grid will be defined as an array-containing-arrays, so that original grid would be represented like this:

```
[ ['A', 'B', 'C'], ['D', '*', 'E'], ['F', 'G', 'H'] ]
```

Write a function that, given a NxM grid, like the one above, returns a grid with all cells in a column or row originally containing a star turned into stars. Values which have been transformed into stars do not affect their rows and columns. You should do this *in-place* – by changing the original grid, not by creating a new one. Your function should return the grid.

For example, for the input above, it should return:

```
[ ['A', '*', 'C'], ['*', '*', '*'], ['F', '*', 'H'] ]
```

Part 3: Lucky Nums App

You should build a small full-stack application with a Flask backend API and a HTML/JS frontend.

It will be for a service where users provide some data about themselves, and receive a lucky number and facts about that number and their birth year.

> **Warning: Write Good Code!**
>
> While this is a small app, a lot of what we covered this week was around making good choices for names and helper functions to make more complex apps understandable.
>
> Write good code for this; this is part of what we'll be assessing for.

Task 1: Build Backend API

We've given you a Flask app with one route already defined, for an HTML page. For now, you should ignore that route — we're going to build the API before the front-end.

This API needs one endpoint:

POST /api/get-lucky-num

This route needs a JSON body with the following information:

- name: name of user *(required)*
- email: email of user *(required)*
- year: birth year *(required, must be between 1900 and 2000, inclusive)*
- color: their favorite color *(required and must be one of "red", "green", "orange", "blue")*

This route should return JSON.

If the user failed to provide valid data for all fields, this should return JSON like this:

```
{
  "errors": {
    "color": [
      "Invalid value, must be one of: red, green, orange, blue."
    ],
    "name": [
      "This field is required."
    ]
  }
}
```

(there should be an entry for only the invalid fields; your error messages don't have to match ours exactly)

If they provided valid data, they should get a response like this:

```
{
  "num": {
    "fact": "67 is the number of throws in Judo.",
    "num": 67
  },
  "year": {
    "fact": "1950 is the year that nothing remarkable happened.",
    "year": "1950"
  }
}
```

The lucky number should be a random number from 1-100 (inclusive).

The random facts for the number and the year should come from the numbersapi API: http://numbersapi.com/ <http://numbersapi.com/>

Note: test your API backend with Insomnia before moving on!

Task 2: Make a very simple front-end

In most real full-stack apps, there is a different server for serving the API and for serving the front-end JS and HTML. In this example, though, we'll have you use the same server for both.

We've already provided for you the route and HTML template for the form, along with a stub JS file. **Do not edit our HTML file**.

Edit the JS:

- on submission of the form, it should stop the normal form processing
- instead, it should gather up that form data into JSON and call the backend API
- if the backend API returns errors, it should put the error messages in the ***<b id="fieldname-err">*** elements corresponding to those fields with errors returned
- if the backend API does not return errors, it should put the following message into the result div:

  ```
  Your lucky number is [num] ([num-fact]).
  Your birth year ([year]) fact is [year-fact].
  ```

  (this doesn't need any special formatting; plain text is fine)

There's no need to add any CSS or fanciness to the form (it shouldn't have drop-downs for the year or color or anything like that).

Extra-Credit: Use WTForms for validation

You could solve this by hand-validating the JSON sent to the API endpoint.

For a challenge, do this with WTForms. **Don't make the form itself with WTForms**; this should still just be the HTML we provided.

However, you should:

- make a ***FlaskForm*** of the fields and validation you need
- figure out how to use flask-wtf to validate the JSON received

> **Hint**
>
> Turn off CSRF For that Form!
>
> WTForms normally expects a CSRF field, but users of our API won't have a token to send. Learn how to have WTForms ignore this (search flask-wtf for ***csrf_enabled***)

For a hint on "how could I validate JSON", hover over this:

How Can I Validate JSON with WTForms?

When you instantiate your form in the route, you normally don't have to pass anything to it (flask-wtf automagically passes in Flask's ***request.form***).

However, you can instantiate with arguments for things like "Here's a Python object to get default vqlues from", or "Here's a dictionary to get values from". See https://wtforms.readthedocs.io/en/stable/forms.html#the-form-class <https://wtforms.readthedocs.io/en/stable/forms.html#the-form-class> for help.

Solution

You can view Our Solution <solution/index.html> (password required)