CHECKLIST

Advanced Form Validation

Objectives:

1. How do we determine whether a user submitted a valid email address to our server?
2. What is *regular expression*, or *regex*?

Another common validation that needs to be performed is checking whether an email address is
something known as a **regular expression** or **regex**.

A regex is a sequence of characters that defines a search pattern. It can be used to match a strir
example, every email has a series of alphanumeric characters followed by an @ symbol followed
characters followed by a "." and finally another series of alphanumeric characters. You don't nee
this point, but understanding what they are and what they are used for is definitely important. Tl
email address based on the above criteria looks something like this:

```
r'^[a-zA-Z0-9.+_-]+@[a-zA-Z0-9._-]+\.[a-zA-Z]+$'
```

Let's create a simple application similar to the basic_validation example where we can input an e

Let's start with a basic index.html page:

/advanced_validation/templates/index.html

```
<!--Advanced Validation Example-->
  <h3>Enter a Valid(Any) Email!</h3>
  {% with messages = get_flashed_messages() %}
    {% if messages %}
      {% for message in messages %}
        <p>{{message}}</p>
      {% endfor %}
    {% endif %}
  {% endwith %}
  <form action='/process', method='post'>
    <label for="email">Email</label>
    <input id="email" name="email" type="text">
    <button>Submit</button>
  </form>
```

And don't forget our server.py!

/advanced_validation/server.py

```
# import Flask
from flask import Flask, render_template, redirect, request, session, flash
# the "re" module will let us perform some regular expression operations
import re
# create a regular expression object that we can use run operations on
EMAIL_REGEX = re.compile(r'^[a-zA-Z0-9.+_-]+@[a-zA-Z0-9._-]+\.[a-zA-Z]+$')
app = Flask(__name__)
app.secret_key = "ThisIsSecret!"
@app.route('/', methods=['GET'])
def index():
    return render_template("index.html")
@app.route('/process', methods=['POST'])
def submit():
    if len(request.form['email']) < 1:
        flash("Email cannot be blank!")
    # else if email doesn't match regular expression display an "invalid email address"
    else:
        flash("Success!")
    return redirect('/')
if __name__=="__main__":
    app.run(debug=True)
```

Now let's modify our validation conditional to include a case for an invalid email!

```
@app.route('/process', methods=['POST'])
def submit():
    if len(request.form['email']) < 1:
        flash("Email cannot be blank!")
    elif not EMAIL_REGEX.match(request.form['email']):
        flash("Invalid Email Address!")
    else:
        flash("Success!")
    return redirect('/')
```

Let's see what is going on here:

- We are using if/else statements just like we did before to perform validations
- We are using the EMAIL_REGEX object that we created and running the .match() method can be found. If the argument matches the regular expression, a match object instance is r

Multiple form validations

Let's say that you're validating multiple fields and if there are error messages, you want to redire error messages, you want to redirect the user to another url, say to "/success". Furthermore, say flash error message into different labels/buckets. This can be accomplished by structuring your

```
@app.route("/process", methods=['POST'])
def submit():
    # adding validation rules
    if len(request.form['email']) < 1:
        flash("Email cannot be blank!", 'email')
    elif not EMAIL_REGEX.match(request.form['email']):
        flash("Invalid Email Address!", 'email')
    if len(request.form['name']) < 1:
        flash("Name cannot be blank!", 'name')
    elif len(request.form['name']) <= 3:
        flash("Name must be 3+ characters", 'name')

    if len(request.form['comment']) < 1:
        flash("Comment cannot be blank!", 'comment')
    elif len(request.form['comment']) > 100:
        flash("Comment must be less than 100 characters", 'comment')

    if '_flashes' in session.keys():
        return redirect("/")
    else:
        return redirect("/success")
```

Other Useful Validation Tools:

str.isalpha() -- Returns a boolean that shows whether a string contains only alphabetic charact (https://docs.python.org/3.6/library/stdtypes.html#str.isalpha).

Other string methods that may be useful can be found in the Python Docs here (https://docs.python.org/3.6/library/stdtypes.html#string-methods).

time.strptime(string, format) -- Changes a string to a time using the given format. Documenta (https://docs.python.org/3.6/library/time.html#time.strptime)

More information on Flash

http://flask.pocoo.org/docs/0.12/patterns/flashing/ (http://flask.pocoo.org/docs/0.12/patterns/fl