Lab 23: Dynamic web programming

Apr 26, 2018

Main Event

1. Anytime calendar

Update the calendar you made in the last lab such that the month and year are provided by the user. Specifically, you should add a form to the page that has two options representing the month and year, respectively. The form should also contain a submit button that when clicked, sends the form data back to the page (the current route) via a GET request. When generating the year options, you are free to use whatever date range you like.

The first time the page is loaded no user information will have been sent. In this case, it is okay to display the current calendar.

Solution:

```
import flask
import datetime
import calendar

app = flask.Flask(__name__)

def mkform(today):
    tbl = [ '<form>' ]

    tbl.append('Month:<select name="month">')
    for (i, j) in enumerate(calendar.month_name):
        if i > 0:
            s = 'selected' if i == today.month else ''
            tbl.append('<option value="{0}" {2}>{1}</option>'.format(i
```

```
    tbl.append('</select>Year:<select name="year">')
    for i in range(today.year - 50, today.year + 50):
        s = 'selected' if i == today.year else ''
        tbl.append('<option value="{0}" {1}>{0}</option>'.format(i, s)
    tbl.append('</select><input type="submit">')

    tbl.append('</form>')

    return '\n'.join(tbl)

def mkcal(today, current):
    cal = calendar.Calendar(5)

    tbl = [ '<table cellpadding="10">' ]

    # month heading
    month = today.strftime('%B')
    row = '<tr><th colspan="7">{0} {1}</th>'.format(month.upper(), toc
    tbl.append(row)

    # week days heading
    row = ''
    for i in cal.iterweekdays():
        row += '<th>{0}</th>'.format(calendar.day_name[i][:3])
    row = '<tr>{0}</tr>'.format(row)
    tbl.append(row)

    # the calendar!
    for week in cal.monthdatescalendar(today.year, today.month):
        tbl.append('<tr>')
        for day in week:
            color = ''
            if day.month == today.month:
                entry = day.day
                if entry == today.day and current:
                    color = 'bgcolor="yellow"'
            else:
                entry = ' '
            row = '<td {0} align="right">{1}</td>'.format(color, entry
            tbl.append(row)
        tbl.append('</tr>')

    tbl.append('</table>')
```

```
    return '\n'.join(tbl)

@app.route('/', methods=['GET'])
def f1():
    args = flask.request.args
    now = datetime.datetime.now()

    if 'year' in args and 'month' in args:
        ftime = args['year'] + args['month'].zfill(2)
        today = datetime.datetime.strptime(ftime.zfill(2), '%Y%m')
        current = today.month == now.month and today.year == now.year
    else:
        current = True

    if current:
        today = now

    return mkform(today) + mkcal(today, current)

app.run(host='0.0.0.0', threaded=True)
```

2. Dynamic multiplication

Update the multiplication table you made in the last lab such that the number of rows and the number of columns are provided by the user. In this case, the user should communicate with the page through input text's: one for the number of rows, and one for the number of columns. The form should also have a submit button that sends the user request back to the page using a GET request.

Solution:

```
import flask

app = flask.Flask(__name__)

def valid_arguments(args):
    if 'rows' in args and 'cols' in args:
        return args['rows'].isdigit() and args['cols'].isdigit()
    return False

def mkform(args):
```

```
    r = args['rows'] if 'rows' in args else ''
    c = args['cols'] if 'cols' in args else ''

    form = [
        '<form method="GET" action="/">',
        'Rows: <input type="text" name="rows" value="{0}">'.format(r),
        'Columns: <input type="text" name="cols" value="{0}">'.format(
        '<input type="submit" value="Multiply!">'
    ]

    return '\n'.join(form)

@app.route('/', methods=['GET'])
def times():
    args = flask.request.args
    if not valid_arguments(args):
        return mkform(args)

    rows = int(args['rows']) + 1
    cols = int(args['cols']) + 1

    html = mkform(args)
    table = [
        '<table cellpadding="10">',
        '<tr><td colspan="{0}" align="center">{1}</td></tr>'.format(cc
    ]

    for row in range(rows):
        table.append('<tr>')
        for column in range(cols):
            if not row:
                if not column:
                    entry = ' '
                else:
                    entry = '<b>{0}</b>'.format(column)
            elif not column:
                entry = '<b>{0}</b>'.format(row)
            else:
                entry = row * column
            table.append('<td>{0}</td>'.format(entry))
        table.append('</tr>')

    table.append('</table>')
```

```
    return '\n'.join(table)

app.run(host='0.0.0.0', threaded=True)
```

3. Build your own blog

The objective of this exercise is to write a program that allows you to blog. The application will have two routes: one that displays existing posts and allows the user to post something new, and another that adds posts to the system.

User interaction

Define the following route:

```
@app.route('/')
def display():
    # your code here
```

The function should build a form that includes the following elements:

- Input text for the author name.
- Input text for the subject of the post. By default, the box created for input text is not very long. The size is fine for user names, but not necessarily for subjects. You can change that using the `size` attribute:

  ```
  <input type="text" name="subject" size="50">
  ```

- A textarea in which the user can type their message.
- A password field.
- A submit button.

The form should direct the input to `/add` via a POST request.

Adding posts to the system

Posts are essentially just entries in a CSV file. Populating that CSV file should happen in the `add` route; thus, you should define the following:

```
@app.route('/add', methods=['GET', 'POST'])
def save():
```

```
    # your code here
```

The `save` function, should add the information that is available from the POST'ed data (created by the form you just made) to a CSV file on your system *if the password is what you expect*. You likely do not want to include the password in what is saved! This is perhaps the only instance in which opening a file in append mode is suggested.

Along with the information provided in the post, you should also write the current timestamp to the CSV file:

```
> import time
> t = time.time()
> print(t)
1512536727.39614
```

A timestamp denotes the number of seconds that have passed since January 1970. When storing information, its representation as a float makes it more convenient to work with than formatted time.

Once the addtion has been made, you can redirect the user (back) to the form via a Flask redirect:

```
def save(): # from above
    # ...
    url = flask.url_for('display')
    return flask.redirect(url)
```

The function `url_for` takes the name of the function you want run. Internally, Flask will build the URL that corresponds to that route and direct the current user to it. For a more detailed example of what's going, checkout the Flask documentation.

Displaying existing posts

Update your `/` route (`display` function) to *also* show your posts. That is, along with the form that the function already produces, it should produce a table inclusive of the posts in your post CSV file.

It is probably cleanest to do this in an HTML table where a row (`<tr>`) consists of cells (`<td>`) consisting of information from the current CSV "row". You can use the datetime module to convert the timestamp into something pretty:

```
> import datetime
> x = datetime.datetime.fromtimestamp(t) # t from previous fragment
> y = x.strftime('%Y-%m-%d %H:%M:%S')
> print(y)
2017-12-06 09:05:27
```

The string that is given to `strftime` tells the datetime object how to format itself. See the Python documentation for an outline of other formats you can specify.

Share your blog with friends!

If you started your Flask application with with the special “all-interface” host IP

```
app.run(host='0.0.0.0') # you may have additional parameters
```

then anyone on campus (or with access to the NYU VPN) can see your blog.

1. Find your computers IP address. Let’s assume, for example, it’s `1.2.3.4`.
2. Start your Flask application.
3. Tell a friend to visit your website: `http://1.2.3.4:5000/`. Your friend should be able to see all of your posts. They can only add posts if they know your password.

Solution:

```
import csv
import time
import datetime
from pathlib import Path

import flask

app = flask.Flask(__name__)

pw = 'haha'
blog = Path('blog.csv')

def entry_cell(entry):
    stamp = datetime.datetime.fromtimestamp(float(entry['time']))
    stamp = stamp.strftime('%Y-%m-%d %H:%M')
```

```
    cell = [
        '<tr bgcolor="#c0c0c0">',
        '<td>{0}</td>'.format(entry['author']),
        '<td>{0}</td>'.format(stamp),
        '</tr>',
        '<tr><td colspan="2">{0}</td></tr>'.format(entry['subject']),
        '<tr><td colspan="2">{0}</td></tr>'.format(entry['post']),
    ]

    return '\n'.join(cell)

@app.route('/')
def display():
    form = """
<form method="POST" action="/add">

<table width="100%">
<tr><td>Author</td><td><input type="text" name="author"></td></tr>
<tr><td>Subject</td><td><input type="text" name="subject"></td></tr>
<tr><td>Entry</td><td><textarea name="post"></textarea></td></tr>
<tr><td>Password</td><td><input type="password" name="pw"></td></tr>
<tr><td colspan="2"><input type="submit" value="Post!"></td></tr>
</table>

</form>
"""

    page = [
        '<table width="100%"><tr>',
        '<td>' + form + '</td>',
    ]

    if blog.exists():
        posts = [ '<td><table width="100%">' ]

        with blog.open() as fp:
            reader = csv.DictReader(fp)
            for entry in reader:
                posts.append(entry_cell(entry))

        posts.append('</table></td>')
        page.extend(posts)
```

```
    page.append('</tr></table')

    return '\n'.join(page)

@app.route('/add', methods=['GET', 'POST'])
def save():
    args = flask.request.form

    if args and args['pw'] == pw:
        if not blog.exists():
            header = ['time', 'author', 'subject', 'post']
            blog.write_text(','.join(header) + '\n')

        with blog.open('a') as fp:
            print(time.time(),
                  args['author'],
                  args['subject'],
                  args['post'],
                  sep=',',
                  file=fp)

    url = flask.url_for('display')
    return flask.redirect(url)

app.run(host='0.0.0.0', debug=True)
```

Additional Practice

1. Advanced blogging

Add new forms to your blog to alter what it displays. As you create more posts, this can help you make sense of what has been written. Doing these exercises will also make the next lab more enjoyable.

- Display only posts from a selected author. Have `display` create an additional form that contains an option and a submit button. The contents of the option (each “select”) should be the unique authors of all blog posts. Along with the author names, the option should have an additional select signifying that all posts should be displayed.
- Sort posts by subject. Assuming your posts are displayed in a table, add a header to that table if you have not done so already. For the “subject” head,

make the text a link (back to the same route) that provides a variable letting `display` know that posts should be ordered by subject; for example:

```
<a href="/?sort=subject">Subject</a>
```

When the link is followed, and Flask subsequently executes `display`, you should recognize that the option has been requested and display the posts accordingly.

- Allow the user to specify a range of dates by which the posts are filtered. There are lots of ways to do this:

  1. Use two option blocks for start and end times.
  2. Take the user to another page (route) that displays several month calendars (use your existing code!). Each day is presented with a checkbox allowing the user to select that day. Submit sends the minimum and maximum dates to the `/` route, where `display` takes them into account when selecting messages to display.
