factoring out the data model

Daniel Jackson
responding to a simple GET

@app.route('subjects/<sid>/reviews/<id>', methods=['GET', 'POST'])
def show_review (sid, rid):
    cur = g.db.execute(
        '''select users.first, created, rating, subjects.name, content
        from reviews, subjects, users
        where reviews.id = ? and users.id = reviews.by and subjects.id = ?
        ''' , [rid, sid])
    row = cur.fetchone()
    review = dict(by=row[0], created=row[1], rating=row[2], about=row[3], content=row[4])
    return render_template('review.html', review=review)

<div class=reviewheader>
    <span class=by>{{ review.by }}</span>
    <span class=date>{{ review.created() }}</span>
    {% if review.rating != 0 %}
        <span class=starwidget value="{{ review.rating }}"></span><br>
    {% endif %}
    <span class=subject>{{ review.about() }}</span>
</div>
<div class=reviewcontent>
    {{ review.content }}
</div>

> what’s wrong with this?
recall MVC architecture

controller actions in __init__.py

view templates & CSS

model model classes SQL queries DB
separating out model

```python
@app.route('subjects/<sid>/reviews/<id>', methods=['GET', 'POST'])
def show_review (sid, rid):
    review = Model.get_review_data(sid, rid)
    return render_template('review.html', review=review)

class Model:
    @staticmethod
def get_review_data(sid, rid):
        cur = g.db.execute(
            '''select users.first, created, rating, subjects.name, content
            from reviews, subjects, users
            where reviews.id = ? and users.id = reviews.by and subjects.id = ?
            ''', [rid, sid])
        row = cur.fetchone()
        review = dict(by=row[0], created=row[1], rating=row[2], about=row[3], content=row[4])
        return review
```

› now data model is factored out
› but look at dictionary constructor!
introducing objects

```python
@app.route('subjects/<sid>/reviews/<id>\'', methods=['GET', 'POST'])
def show_review (sid, rid):
    review = Review.get_review_by_id(rid)
    subject = Subject.get_subject_by_id(sid)
    review.by = subject.first
    return render_template('review.html', review=review)

class Review:
    def __init__(self, created, rating, content):
        self.created = created
        self.rating = rating
        self.content = content

    @staticmethod
    def get_review_by_id(id):
        cur = g.db.execute('''select created, rating, content from reviews where reviews.id = ?''', [id])
        row = cur.fetchone()
        return Review(row[0], row[1], row[2])

class Subject:
    ...

> now controller is object-oriented
```
problems with this style

class Review:
    def __init__(self, created, rating, content):
        self.created = created
        self.rating = rating
        self.content = content

@staticmethod
def get_review_by_id(id):
    cur = g.db.execute(''select created, rating, content from reviews where reviews.id = ?'''', [id])
    row = cur.fetchone()
    return Review(row[0], row[1], row[2])

look at Review class
  › all boiler-plate: just converting between row & object
  › every model class, every get, every update
what's going on: the data cycle

update request

request.form

<table>
<thead>
<tr>
<th>id</th>
<th>rating</th>
<th>content</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>5</td>
<td>yummy!</td>
</tr>
</tbody>
</table>

database table

<table>
<thead>
<tr>
<th>id</th>
<th>by</th>
<th>content</th>
<th>rating</th>
<th>about</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>yummy!</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>neat</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

response

<table>
<thead>
<tr>
<th>id</th>
<th>rating</th>
<th>content</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>5</td>
<td>yummy!</td>
</tr>
</tbody>
</table>