CS1520 Recitation:

Flask 3: Template Inheritance / Datamodel

Jeongmin Lee
Template Inheritance

- Template inheritance allows you to build a base “skeleton” template that contains all the common elements of your site and defines blocks that child templates can override.
Base Template

- `base.html` defines a simple HTML skeleton document that you might use for a simple two-column page.
- The `{% block %}` tags define four blocks that child templates can fill in.
- All the `block` tag does is tell the template engine that a child template may override those placeholders in the template.
<!DOCTYPE html>
<html lang="en">
<head>
    {% block head %}
    <link rel="stylesheet" href="style.css" />
    <title>{% block title %}{% endblock %} - My Webpage</title>
    {% endblock %}
</head>
<body>
    <div id="content">
        {% block content %}{% endblock %}
    </div>
    <div id="footer">
        {% block footer %}
        {% endblock %}
    </div>
</body>
</html>
Child Template

- Child.html: The \{% extends \}\% tag is the key here.
- It tells the template engine that this template “extends” another template.
- When the template system evaluates this template, it first locates the parent.
- The extends tag should be the first tag in the template.
- Everything before it is printed out normally and may cause confusion.
Note that since the child template doesn’t define the footer block, the value from the parent template is used instead.
You can access templates in subdirectories with a slash:
{% extends "layout/default.html" %}
Welcome to my awesome homepage.
Super Blocks

- It’s possible to render the contents of the parent block by calling super.
- This gives back the results of the parent block.
Install SQLAlchemy

- PIP: pip install flask-sqlalchemy
- Then, run python and test: `import sqlalchemy`
First Code

- Import SQLAlchemy class and create db first.
- Create a Flask application object and set URI for the database to be used.

```python
from flask import Flask, render_template
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///students.sqlite3'
```
Create an object of SQLAlchemy class with application object as the parameter.

This object contains helper functions for ORM operations. It provides a parent Model class using which user defined models are declared.

```python
db = SQLAlchemy(app)
class Students(db.Model):
    id = db.Column('student_id', db.Integer, primary_key = True)
    name = db.Column(db.String(100))
    city = db.Column(db.String(50))
    addr = db.Column(db.String(200))
    pin = db.Column(db.String(10))

    def __init__(self, name, city, addr, pin):
        self.name = name
        self.city = city
        self.addr = addr
        self.pin = pin
```
Create db!

db.create_all()
Session Objects

- Session object of SQLAlchemy manages all persistence operations of the data (ORM) objects.
- CRUD (remember!)
  - Create
  - Read
  - Update
  - Delete
Session Objects

- Session object of SQLAlchemy manages all persistence operations of the data (ORM) objects.
- CRUD (remember!)
  - Create & Update: `db.session.add(model object)`
  - Read: `model.query.all()`
  - Delete: `db.session.delete(model object)`
• Add some students.

```python
a = Students(id="0123", name="James Dean", city="Pittsburgh")
b = Students(id="0125", name="Lily Cory", city="Greensburgh")
db.session.add(a)
db.session.add(b)
db.session.commit()
```
Delete student

```python
a = Students(id="0123", name="James Dean", city="Pittsburgh")
db.session.delete(a)
db.session.commit()
```
Query student

```python
a = Students.query.filter_by(id="0125").first
a.id
a.name
```
Questions?