템플릿

표현과 처리의 분리

- python코드와 HTML 코드를 물리적으로 구분할 필요가 있음
- python : 브라우저의 요청에 응답, 출력할 데이터 준비
- 템플릿 : python이 생성한 데이터와 결합될 HTML
- 이들을 결합한 최종 HTML 생성(렌더링, 바인딩)은 flask가 담당

python에서 HTML 생성

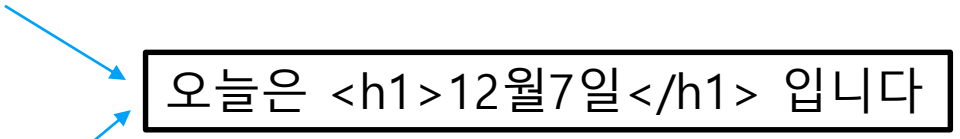
```
html = "오늘은 <h1" +  LocalDate.now() + "</h1> 입니다"

return html
```

Controller (Data생성)

```
render_template("template.html",
                now= LocalDate.now())
```

```
오늘은 <h1>{{now}}</h1> 입니다
```

외부 HTML(template)



진자 (Jinja2)

- 템플릿 엔진이란 HTML(Markup)과 데이터를 결합한 결과물을 만들어 주는 도구
- 전달받은 데이터를 이용하여 동적으로 HTML 생성
- 서버에서 HTML을 동적으로 렌더링
- 기존 HTML 문법에 {% %} 구분관 {{}} 표현식을 사용해 파이썬 연동
- flask 모듈의 render_template 함수 import 해서 호출
- template 파일은 templates 폴더에 저장

```
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/tmpl1')
def tmpl1():
    return render_template('tmpl1.html')
```

templates/tmpl1.html

```
<h1>Hello World!</h1>
<h2>Welcome to FlaskApp!</h2>
```

템플릿 표현식 literal

- {{}}표현식으로 다양한 표현신을 사용할 수 있음
- 산술 연산 가능 {10 + 5}
- list, (), {} 타입의 복합 자료 구조 사용 가능

tmpl0.html

```
{{20}}  <br/>
{{3+4}}  <br/>
{{"hello"}}  <br/>


{{[10, 20, 40]}}  <br/>

{{  {"name":"이순신", "age":62}  }}  <br/>
```

```
@app.route('/tmpl0')
def tmpl0():
    return render_template("tmpl0.html")
```

템플릿 변수

- key/value 파라미터 기능을 이용해 템플릿으로 전달할 변수 정의
- 여러개의 변수 전달가능, msg="hello", id=45
- list, tuple , dictionary 타입 모두 전달 가능

tmpl2.html

```
<body>    {{msg}}  </body>
```

```
@app.route('/tmpl2')
def tmpl2():
    return render_template("tmpl2.html", msg="Hello World")
```

```
@app.route('/tmpl3')
def tmpl3():
    return render_template("tmpl3.html",
   lst=["ckt", 53], tpl=('ckt', 53), obj={"name":"ckt", "age":53})
```

tmpl3.html

```
<body>
  {{lst[0]}} // {{lst[1]}}  <br/>
  {{tpl[0]}} // {{tpl[1]}}  <br/>
  {{obj.name}} // {{obj.age}}  <br/>
</body>
```

객체 리스트 전달

- 딕셔너리 타입을 배열로 정의해서 레코드 집합 타입으로 데이터 전달
- 가장 빈번하기 사용하는 구조

```
@app.route('/tmpl4')
def tmpl4():

    objs = [{"name":"홍길동", "age":40},
            {"name":"이순신", "age":50},
            {"name":"임꺽정", "age":60}            ]

    return render_template("tmpl4.html", objs = objs)
```

tmpl4.html

```
<body>
    {{objs[0].name}} //  {{objs[0].age}}  <br/>
    {{objs[1].name}} //  {{objs[1].age}}  <br/>
    {{objs[2].name}} //  {{objs[2].age}}  <br/>
</body>
```

Escape

- 표현식을 출력할 때 <,>는 escape 문자로 변환
  - <는 &lt로 변환, >는 &gt로 변환됨
- 따라서 <h1>Hello</h1>출력시 큰 글씨로 나오지 않고 원본 데이터가 그대로 출력됨
- HTML tag 효과를 그대로 적용하기 위해 | safe 사용

```
@app.route('/escape')
def escape():
    return render_template("escape.html",
title="<h1>Hello</h1>")
```

127.0.0.1:8080/escape

<h1>Hello</h1>

escape.html

```
<body>
    {{title}}  <br/>
    {{title | safe}}  <br/>
  </body>
```

Hello

view-source:127.0.0.1:8080/escape

자동 줄바꿈

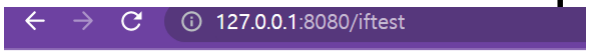
```
1  <body>
2    &lt;h1&gt;Hello&lt;/h1&gt;  <br />
3    <h1>Hello</h1>  <br />
4  </body>
```

if / else 제어문

- {% %} 구문을 사용해 하나의 키워드로 이루어진 if, else, elif, endif 제어문 사용 가능

```
@app.route("/iftest")
def home_page():
    username ="이순신"
    return render_template("iftest.html", username=username, score=87)
```

```
{% if username == "": %}
  <h1>Hello User!</h1>
{% else %}
  <h1>Hello {{ username }}!</h1>
{% endif %}
<h1>Score {{score}} :
{% if score >= 90 %}
     A+
{% elif score >= 80 %}
     B+
{% else  %}
     C+
{% endif %}
</h1>
```



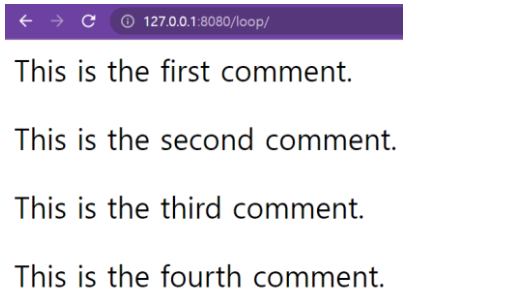
Hello 이순신!

Score 87 : B+

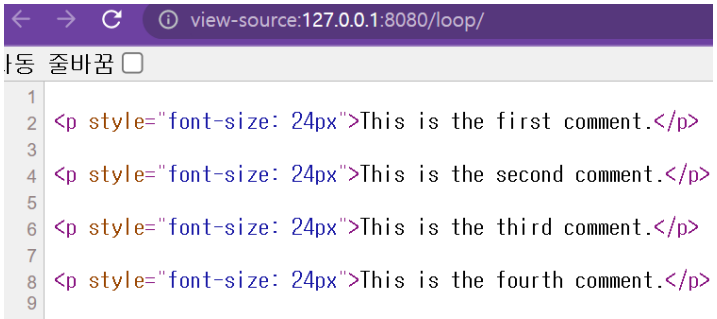
iftest.html

Loop

- {% %} 구문을 사용해 for, endfor 키워드를 사용해 반복 구문을 처리할 수 있음
- {% for 생성변수 in 파이썬전달변수%}
- 해당 블록은 원소수 만큼 반복됨




```
@app.route('/loop/')
def loop():
    comments = ['This is the first comment.',
                'This is the second comment.',
                'This is the third comment.',
                'This is the fourth comment.'
                ]

    return render_template('loop.html', comments=comments)
```



loop.html

```
{% for comment in comments %}
<p style="font-size: 24px">{{ comment }}</p>
{% endfor %}
```

Loop/ if

- 특별 변수 loop.index(고정) 반복문에서 반복문에 대한 index 정보를 리턴함
  - index 시작값은 1,
  - index0(시작값0), length(원소갯수), first(첫번째?), last(마지막?) 사용가능
- set 키워드를 사용해 템플릿 내에서 변수를 정의해서 사용할 수 있음

```
@app.route('/loopif/')
def loopif():
    comments = ['This is the first comment.',
                'This is the second comment.',
                'This is the third comment.',
                'This is the fourth comment.']
    return render_template("loopif.html", comments=comments)
```

loopif.html

```
{% for comment in comments %}
    {% if loop.index % 2 == 0 %}
       {% set bg_color = '#e6f9ff' %}
    {% else %}
       {% set bg_color = '#eee' %}
    {% endif %}
   <div style="background-color:{{bg_color}}">
      <p style="font-size: 24px">{{ comment }}</p>
   </div>
{% endfor %}
```

Loop : 객체 리스트

- 딕셔너리 타입을 배열로 정의(객체리스트)해서 레코드 집합 타입으로 데이터를 전달하는 경우도 loop에서 사용가능
- 가장 빈번하기 사용하는 구조
- loop안에서 생성한 변수가 딕셔너리 타입

1. 이순신 // lee // 서울시
2. 홍길동 // hong // 인천
3. 김유신 // kim // 제주도

```
@app.route('/loopobjs/')
def loopobjs():
    users = [
        {"name":"이순신", "userid": "lee", "address":"서울시"},
        {"name":"홍길동", "userid": "hong", "address":"인천"},
        {"name":"김유신", "userid": "kim", "address":"제주도"}    ]
    return render_template("loopobjs.html", users=users)
```

loopobjs.html

```
<ol>
{% for user in users %}
    <li>   {{user.name}}  // {{user.userid}}  //
{{user.address}}    </li>
{% endfor %}
</ol>
```

HTML 중복 처리

- 상단 header/menu나 하단 footer 처럼 많은 페이지에서 반복되는 부분을 별도의 템플릿으로 분리하고 include 문을 사용해 결합
- 개별 템플릿은 내부적으로는 랜더링된 결과를 리턴(표현식 사용가능)

include.html

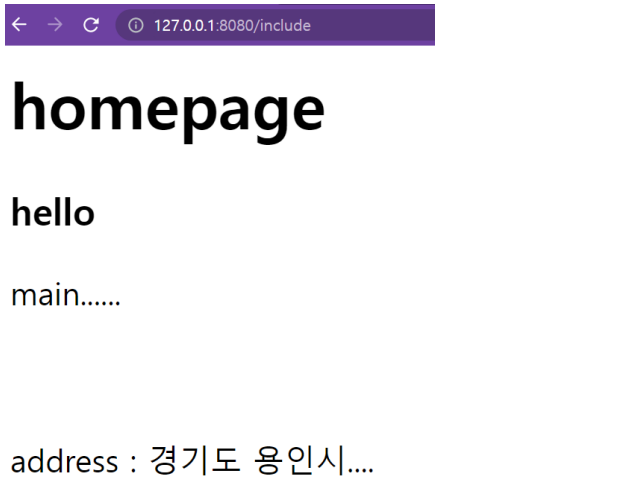
```
<html>

{% include "header.html" %}

main..... <br/>
<br/><br/><br/>
{% include "footer.html" %}

</html>
```

header.html

```
<h1> homepage </h1>
<h3>{{title}}</h3>
```

footer.html

```
address : 경기도 용인시....
```




```
@app.route('/include')
def include():
    return render_template("include.html", title="hello")
```