# Flask Tutorial


파이썬 기반으로 작성된 마이크로 웹 프레임 워크(Micro Web Framework)

간단한 웹 사이트나 혹은 간단한 API 서버를 만드는 데에 특화.(API Server의 역할을 많이 함)

Micro Web Framework로 필요한 기능만 라이트하게 사용 가능.

# Flask 설치

# INDEX

# Flask 설치 및 간단 예제

$ pip install Flask  // 명령어를 이용해 설치

* Flask 공식 문서에서는 flask 프로젝트를 가상환경을 통한 설치를 안내하고 있음

예제)

```python
from flask import Flask

app = Flask(__name__)

@app.route('/')
def home():
    return 'Hello, World!'

if __name__ == '__main__':
    app.run(debug=True)
```

# 라우트(Route)

# 라우트(Route)

Flask에서 URL을 방문 할 때 route() 데코레이터를 사용하여 특정 함수가 바인딩 되게 하는 것

ex)

```python
from flask import Flask

app = Flask(__name__)

@app.route('/')
@app.route('/home')
def home():
    return 'Hello, World!'

@app.route('/guest')
def user():
    return 'Hello, guest!'

if __name__ == '__main__':
    app.run(debug=True)
```

@app.route('/')
@app.route('/home')
@app.route('/guest') 와 같이 작성 후,
아래에 트리거 함수 작성

* @app.route('/경로')의 형식으로 작성 시 반드시 '/'로 시작

# 동적 URL 및 HTML 렌더링

# 동적 URL

Flask에서 URL을 지정할 때 '<변수>'를 사용하는 것

ex)

```python
from flask import Flask

app = Flask(__name__)

@app.route('/')
@app.route('/home')
def home():
    return 'Hello, World!'

@app.route('/user/<user_name>/<int:user_id>')
def user(user_name, user_id):
    return f'Hello, {user_name}({user_id})!'

if __name__ == '__main__':
    app.run(debug=True)
```

'/user/<user_name>/<int:user_id>' 와 같이 <>를 이용하여 URL에서 변수 사용 가능

* <converter : variable_name>에서 converter 옵션
string(기본값) : /가 포함되지 않은 문자열
int
float
path : /가 포함된 문자열
uuid : UUID(범용고유식별자) 형식

# HTML 렌더링

뷰 함수에서 return하는 응답에 HTML을 직접 반환하게 하여 렌더링 하는 것

ex)

```python
from flask import Flask

app = Flask(__name__)

@app.route('/')
@app.route('/home')
def home():
    return '''
    <h1>이건 h1 제목</h1>
    <p>이건 p 본문 </p>
    <a href="https://flask.palletsprojects.com">Flask 홈페이지 바로가기</a>
    '''

@app.route('/user/<user_name>/<int:user_id>')
def user(user_name, user_id):
    return f'Hello, {user_name}({user_id})!'

if __name__ == '__main__':
    app.run(debug=True)
```

return에서 HTML이 직접 반환 됨

# Template (with. jinja2)

# Template

일관된 구조와 기능을 가지는 HTML 파일을 템플릿 파일로 구성하여 관리하는 기능

* jinja2라는 템플릿 엔진 사용

** 템플릿을 사용하기 위한 준비

1. 프로젝트 폴더 내에 templates 이름의 폴더 생성
2. templates 폴더 안에 템플릿으로 사용될 HTML 파일 작성
3. render_template(파일명) 을 이용해 HTML파일 렌더링

# Template 예제

```python
from flask import Flask, render_template
app = Flask(__name__)
@app.route('/')
def index():
    return render_template("index.html")

if __name__ == '__main__':
    app.run(debug=True)
```

http://{host}:{post}/ 에 접속하면 index.html 파일이 렌더링

# Template 활용 – 1. 템플릿에서의 변수 사용

1. render_template()에서 파일명에 이어 변수명을 인수로 추가 (쉼표를 이용해서 2개 이상의 변수 추가 가능)
ex) render_template("index.html", var_username)

2. 템플릿(HTML 파일)에서 해당 변수를 사용할 때는 {{ 변수명 }}의 형식으로 불러와 사용

+ 템플릿 내에서 변수를 파이썬 함수처럼 filter 기능으로 처리하는 방법도 있음. ex) {{ 변수명 | 필터 }}

# Template 활용 – 2. 템플릿에서의 if 조건문

```
{% if template_variable < 20 %}
  <p>{{ template_variable }}은 20보다 작다.</p>
{% elif template_variable > 20 %}
   <p>{{ template_variable }}은 20보다 크다.</p>
{% else %}
   <p>{{ template_variable }}은 20이다.</p>
{% endif %}
```

{% if 조건문 %} ~ {% endif %} 형식으로 작성
elif와 else문도 파이썬에서 직접 사용하는 방식으로 사용

# Template 활용 – 3. 템플릿에서의 for문

```
<ul>
{% for x in range(10) %}
    <li>{{ x }}</li>
{% endfor%}
</ul>
```

파이썬에서 for문을 사용하는 것처럼 {% for 반복문 %} ~ {% endfor %}의 형식으로 작성

```
<ul>
{% for key, value in template_dict | dictsort %}
    <li>{{ key }} : {{ value  }}</li>
{% endfor%}
</ul>
```

딕셔너리 사용 및 filter 기능을 이용한 dictsort 예시 ( .items() 붙이지 X )

# Template 활용 – 4. 템플릿에서의 상속

* footer, header 등 레이아웃의 일관성을 유지하기 위해 사용

```
<html>
  <head>
    <title>내 웹사이트</title>
  </head>
  <body>
  {% block content %}{% endblock %}
  </body>
</html>
```

parent.html

1. 부모 템플릿에서 자식 템플릿의 내용이 담길 위치에 {% block content%}{% endblock %}을 작성

# Template 활용 – 4. 템플릿에서의 상속

* footer, header 등 레이아웃의 일관성을 유지하기 위해 사용

```
{% extends "base.html"  %}
{% block content %}
    <p>자식 템플릿에 포함될 내용</p>
{% endblock %}
```

child.html

2. 자식 템플릿에서 {% extends “부모템플릿이름” %} 작성

3. 자식 문서에 포함될 내용을 {% block content%} ~ {% endblock %}의 형식으로 작성

# GET, POST 요청 처리

# GET, POST 요청 처리

@app.route(“/”, methods = [“GET”, “POST”]) 와 같이 route() 데코레이터의 methods를 이용하여 GET, POST 요청 처리 구분

(method를 지정하지 않으면 기본값으로 “GET” 요청)

*** POST 요청 예시 ***

```
@app.route('/register', methods=["POST"])
    def post_example():
```

# GET, POST 요청 처리 – Example

```python
from flask import Flask, render_template, url_for, flash, redirect
from forms import RegistrationForm
# RegistationForm은 외부 .py 파일로 간단한 form 양식

app = Flask(__name__)
app.config["SECRET_KEY"] = '123123123sdasdasdasdae'

@app.route('/')
def home():
    return render_template('layout.html')

@app.route('/register', methods=["GET", "POST"])
def register():
    form = RegistrationForm()
    if form.validate_on_submit():
        # 알람 카테고리에 따라 부트스트랩에서 다른 스타일을 적용 (success, danger)
        flash(f'{form.username.data} 님 가입 완료!', 'success')
        return redirect(url_for('home'))
    return render_template('register.html', form=form)

if __name__ == '__main__':
    app.run(debug=True)
```

# GET, POST 요청 처리 – Example

```python
@app.route('/register', methods=["GET", "POST"])
def register():
    form = RegistrationForm()
    if form.validate_on_submit():
        # 알람 카테고리에 따라 부트스트랩에서 다른 스타일을 적용 (success, danger)
        flash(f'{form.username.data} 님 가입 완료!', 'success')
        return redirect(url_for('home'))
    return render_template('register.html', form=form)
```

* 클라이언트가 /register url로 접속만 했을 시
GET 요청 -> render_template('register.html', form=form)을 통해 html 파일 불러오기

* 클라이언트가  form 등으로 POST 요청을 보냈을 시
POST 요청 -> {form.username.data}와 POST 요청에서의 정보를 가져온 후 요청 처리

# THE END