일단 만드는 장고

<오리엔테이션1>

웹 프레임워크 : 웹서비스를 쉽게 만들어주는 기계
우리가 배울 것은 상상한 것을 구현하는 것, 그 도구가 장고다.
우리가 만들 것 : 개발할 프로젝트 그림 그리기, 핵심 장고 개념 학습, 실제로 구현

<오리엔테이션2>

장고는 파이썬 기반으로 한 웹 프레임워크 -> 웹서비스를 쉽게 만들어주는 기계. 사용 방법이 정형화됨. 쓰는 문법만 쓴다. 따라서 핵심이 되는 것만 알아도 오케이

스텝2에서 : 프로젝트 그림 그리기,
URL 중요! 개념 어떻게 만들어지는지 알아갈 것,
이쁘게 보여주기 위한 파일은 어떻게 다루고 어떻게 보여줄지!
Bootstrap 웹서비스 꾸밈 요소 (쉽게 꾸미는 것 도와줌)
장고만의 강력한 기능인 템플릿 상속
실제로 만들어볼 것!

~~스텝3에서 : 그림 그리기(대나무숲), 로그인, 댓글 등등 기능~~
~~데이터베이스 구축, 장고를 위한 최소한의 데이터베이스~~
~~사용자의 입력(사진, 글 ...) 어떻게 받아들이는지 3가지~~
~~저장된 내용 목록으로 갖고옴~~
~~댓글~~
~~로그인 아웃~~
~~개발자 블로그 완성~~

사전지식 : 기초 파이썬, html, css 정도는 알아야 함
사전준비 : 파이썬 설치, 크롬, VSCode, 깃(선택)
유의사항 : 실습의 중요성 (눈으로 보기만 하면 안 됨. 같이 따라 치는 게 너무너무 중요함)
확장의 중요성 -> 수업 개념에서 멈추는 게 아니라 다른 곳에서는 어디서 쓸 수 있을까 계속 생각하라! -> **스스로 시나리오 만들고 구현하는 것이 목적!!**

Chapter 2

파이썬 지식 중 중요도

- 자료형(Dictionary)
- 예외처리 너무너무 중요!
- 클래스와 객체
- 모듈, 패키지, 라이브러리

장고 개발을 하기 위해 어느정도 파이썬 지식이 있어야 하는가?

*웹 프레임워크 = 웹서비스를 **쉽게** 만들어주는 기계이다!
어떤 기계가 쓰기 쉬운가? -> 사용법이 **정형화** 되어 있는 기계!
*쓰이는 문법만 주구장창 쓰인다. **정형화** 되어 있다! 그러나 매우 기초적인 부분은 알아야 한다!

딕셔너리 자료형

Dictionary? 사전!

왜 쓰일 수 밖에 없는가..?
=> 무작위로 데이터가 널부러져있다고 하자. 만약 특정 조건으로 그룹을 나눈다고 한다면 어떻게 나눌 수 있을까?
=> A: **대응!**

딕셔너리는 데이터들을 **대응**시켜주는 자료형!

왜 사전인가?
=> **탐색의 기준**으로서 **찾고자 하는 특정 값**을 얻는 것 => **키워드 + 찾고자 하는 값**

탐색의 기준 : Key
기준에 대응되는 찾고자 하는 값 : Value
===== 사전적 의미 =====

실무에서는 언제 쓰는가?
2 X N 표를 통해 데이터를 표현하고 싶을 때 딕셔너리를 쓴다! 라고 생각해도 무방하다

코드로서 표현 : Val = {Key1 : Value1, Key2 : Value2, ...}
이때, Key값은 중복 되어도 변해서도 안 된다!
Print(Val[Key1]) => Value1이 출력될 것이다
Val[Key3] = Value3 => 새로운 Key, Value가 등록될 것이다
// Q : 딕셔너리에서는 무조건 1대1 값인가?

예외처리

사용자가 있는 경우에는 예외처리가 매우 중요하다! 혼란을 야기할 수 있기 때문.

파이썬의 오류?

- 문법 에러(파싱 에러) : 아예 문법 자체가 틀림, 오타, … 구동 불가 치명적 오류!, SyntaxError 이후 문장 => 이때 발생한 오류의 이름
- 예외 : 실행 중에 발생하는 오류, 실행 자체를 멈추진 않는 사소한 오류, (0으로 나눔, 자료형 안 맞음, ..)

파이썬 오류를 모두 잡는 것도 중요하지만 핸들링하는 것도 중요

- Try, except => try 이하 명령 일단 실행함. 이때 발생할 오류를 except 이하로 핸들링하면 프로그램은 그대로 작동시키며 오류 메세지 출력 가능!
  이때, except 이하에 아무것도 안 적어도 된다. 그러면 프로그램은 어떤 오류가 발생하던 실행시켜라
- '' + finally => 예외가 발생하든 안 하든 최종적으로 적을 코드를 finally 이하에 적는다

*오류 핸들링 의의: 프로그램을 **멈춤 없이 실행하기 위해**서 쓴다!!

객체와 클래스

객체지향 프로그램 : 파이썬 자바 C++ ...

객체, 클래스 개념 생겨난 이유 : 세상의 만물을 프로그래밍 하고 싶어서
=> 세상에 있는 모든 **대상**을 관찰 => 모든 **객체**를 관찰
(**객체**를 프로그래밍해야 모든 **다른 대상**을 프로그래밍 할 수 있기 때문)

이 세상에 있는 모든 객체 구성 요소 => **상태 + 동작**

프로그래밍에 적용시키면?

- 상태 : 변수
- 동작 : 함수

문제) 만물은 한 두개가 아니다. 너무 많다
=> 변수와 함수를 여러 개 만들 수 있어야 한다 => 한번에 여러 개 정의할 수 있는 방법 => **객체지향프로그래밍 (변수, 함수를 클래스로 정의하고, 이를 틀로서 달고나처럼 마구 찍어내는 것)**

예시) 판다 세 마리 만들래 => 판다 클래스 정의 => 판다 3번 생성

모듈, 패키지, 라이브러리

모듈 : 파이썬으로 정의된 파일 (함수, 변수 … 가장 작은 단위)
a.py (모듈)에 있는 내용을 b.py 에 쓰고 싶다면? Import a, a.sum(1,2)

패키지 : 모듈의 집합, 모듈의 계층 단위 (사전적 정의)
4개의 파이썬 파일이 있다고 가정할 때, 이들이 큰 폴더 안에, 3개가 또 다른 폴더 안에 있다고 볼 때, 3개를 감싸고 있는 폴더가 패키지이다 (import **a**.get)

라이브러리 : 미리 준비된 모듈, 패키지
파이썬 라이브러리
- 파이썬 스텐다드 라이브러리 (기본 제공, 내장 함수, …)
- 파이썬 패키지 인덱스 라이브러리 (사람들이 만든 모듈, 패키지)

pip: 파이썬 패키지 관리자 (패키지 설치, 검색, 버전 지정, 조회, 제거 …)
+ pip freeze : 프로젝트에 설치되어 있는 패키지 명 및 버전 조회 가능!

웹서비스란? (중요한 기반 지식)

웹?

www : world wide web => 정보의 그물망, 정보를 받아들이는 수순이 다름! 하이퍼 링크를 통해 원하는 정보가 있는 위치로 비 순차적으로 이동 가능 => 자원이 그물처럼 엮임
결과적으로 URL, HTTP, HTML 제공

URL : 정보 자원이 어디 있는지를 나타내는 표식 => 우리가 원하는 정보가 위치한 위치정보

HTTP: 정보로 접근하고 통신하게 해주는 약속 => URL 통해 위치 알았다면 그 정보자원에게 요청할 때 통신하는 규약,

HTTP 요청

- GET : 갖다 줘! (유튜브에 get요청 -> 유튜브 정보 갖다 줘 -> 정보 전달 받음)
- POST : 데이터를 처리해 줘! (블로그 글 읽고 댓글 작성 -> 댓글 등록해줘 요청)

HTML : 응답으로서 정보 자원, 사용자에게 **정보 알려주는** 매체, 다른 정보 자원과 **연결** 매개체

Server : 정보를 URL로 미리 간직하고, 경우에 따라 HTML도 미리 준비, 간직하고 있는 URL로 HTTP요청이 들어오면 거기에 대한 응답을 해주는 것!
예시) 구글 서버 컴퓨터에서 URL에 대한 HTML준비 해놓고 www.google.com입력 하자마자 나의 웹 브라우저로 HTML 제공!

***웹 브라우저**는 서버까지 http통신 보내기 + html 로 받아 가공하여 보여주는 도구

웹서비스

사용자 : 컴퓨터로 원하는 것을 갖다 주는, 원하는 걸 할 수 있는 웹 페이지
프로그래머 : 사용자가 원할 만한 정보 + 위치를 html, url로 준비. 사용자가 요청할 때 언제든 그 위치에 응답을 보낼 수 있는 프로그램

웹 프레임워크란?

프레임워크 : 웹 서비스를 쉽게 만들어주는 기계
+ 복잡한 문제를 해결하거나 서술하는 데 사용되는 **구조.** 간단히 **뼈대, 골조, 프레임워크**라 한다.

프로그래밍 언어로 웹 만들 수는 있어 => 맨날 비슷한 거 만드는 게 귀찮았음 => 프레임워크 제작!

대부분 웹 서비스 기능 : 서버 + 로그인, 로그아웃 + 게시판 + 사진, 글 + 댓글 ...
'' 설계 : 데이터 베이스와 상호작용(등록, 삭제) + 보이는 HTML 관장 + 웹서비스 내부 동작, 원리 담당
=> **정형화** 되어 있다
=> 웹 프레임워크는 굳이 비슷한 코드 작성 안 해도 되게끔 **미리** 만들어 놓은 웹 개발의 기능 단위, 설계 단위의 코드 집합

라이브러리와 프레임워크의 차이?

- **프레임워크 : 명확한 목적**을 달성하기 위해 이미 설계까지 만들어진 구조이자 뼈대 (장고는 프레임워크) 잘 지키며 따라해도 웹 만들어짐
- **라이브러리 :** 연장, 도구의 모음 / 필요에 따라 그때 그때 갖다 써야 함

MVC, MTV(둘이 같은 것 의미)

보편적으로는 MVC라 쓰고, 장고 안에서 만큼은 MTV패턴 이라 한다
이는 설계 원칙을 다음 세 가지로 나눈 것을 의미한다!

1. DB와 상호작용 하는 부분
2. 사용자들 눈에 보이는 부분
3. 내부 동작의 논리를 담당하는 부분

설계 원칙 = 디자인 패턴/ **즉, 장고의 디자인 패턴 = MTV 패턴**

M : model => 데이터베이스와 상호작용 담당
V : view => 사용자 인터페이스 담당, 사용자와 직접 상호작용 하는 부분
C : Controller => 사용자가 상호작용, 혹은 데이터베이스에 일이 있을 때 웹서비스의 내부에서 어떻게 동작해야 할지에 대한 논리를 나타내는 부분

M : model => DB와 상호작용 담당
T : Template => 사용자와 상호작용 하는 부분, MVC 중 View에 해당
V : View => 웹 서비스 내부 동작의 논리 담당, MVC 중 Controller에 해당

예시) 인스타

1. 브라우저에 Instagram.com 입력
2. 해당 화면을 갖다 달라는 GET 요청 감
3. **Controller**가 해당 URL 있는지 확인 후 해당 HTML을 **View**로 보여줌
4. 인스타그램 사용자가 글 씀
5. 글 처리해달라고, DB에 등록해달라고 POST 요청 감
6. **Controller**가 DB의 특정 위치에 저장하겠다는 논리 실행
7. DB와 상호작용하는 **Model**을 통해 등록

⇨ 각각 MVC에서 무엇을 담당하는지 알아 두자

개발 환경 셋팅

가상 환경 : 독립적인 개발환경 만들어주는 파이썬 내장 기능

만약 가상 환경 없이 그냥 설치되었다면?
=> 장고 개발 환경은 내 컴퓨터 전체가 될 것이다.
=> 설치한 패키지는 내 컴퓨터 전체에 영향을 미치게 될 것이다.
=> 여러 개의 프로젝트 진행 시 문제가 생길 수 있음!
(사용하면 안 되는 패키지가 있을 수도, 필요한 패키지의 버전이 다를 수도 있기 때문)

따라서 가상 환경 만들어 위와 같은 어려움을 배제하자

가상환경 만들기 (Windows기준)
Python -m venv myvenv => myvenv라는 폴더 하에 자동으로 가상환경이 만들어짐

가상환경 실행하기 (Windows기준)
Source myvenv/Scripts/activate => ~하에 있는 activate라는 파일을 실행하겠다
올바르게 실행시켰다면 명령어창 위에 소괄호로 뜬다

가상환경 끄기
Deactivate => 소괄호 사라진다

개발 환경 셋팅 실습

파이썬 설치 시 Add Python 3.8 to PATH 해야 함 (Windows 기준)

1. 프로젝트 진행하고자 하는 폴더 열기
2. Git bash here => Code . => 이 폴더에서 VSCode 열어라
3. 터미널 열어 깃 bash 로 지정 (리눅스 기반이기 때문, 절대 다수 서버 호스팅하는 경우 리눅스 기반임)
4. Python –version => 설치된 파이썬 버전 확인 가능 (파이썬 설치 여부 확인 가능)
5. Python -m venv myvenv => 가상환경 만들기
6. Source myvenv/Scripts/activate => 가상환경 실행 (소괄호 확인, 가상환경 실행 여부 확인 가능)
7. Pip install django => 장고 패키지 설치 (4개)
8. Pip freeze => 설치된 패키지 목록 확인
9. **Django-admin startproject myproject (*암기)** => myproject라는 이름의 장고프로젝트 만들 것이다

Django 뜯어보기 1

프로젝트 생성시 만들어진 python 파일들의 코드를 알 필요는 없다. 기능이 뭔지 알아보자!
(init, manage, settings, urls) , "Settings는 좀 중요함." 그러나 외울 필요는 없다 쓰다 보면 알 것

__init__.py
"이게 위치한 곳이 패키지이구나, basicsettings 폴더가 패키지입니다" 알려주는 파일, 일종의 약속

Urls.py
각종 url들을 등록하고 관리하는 파일. 어떤 url에 대한 요청이 들어왔을 때 어떤 동작을 수행할지.
예시) 홈페이지, 로그인 창, 결제 창 등을 '/' 기준으로 관리 ...

Manage.py (*중요)
어떻게 활용할지가 중요. 여러가지 기능을 제공 "아~ 이런 기능을 제공하는구나 정도"

1. 서버 켜기
2. 장고의 작은 구성 단위, 프로젝트를 구성하는 요소인 application 만들기
3. 데이터베이스와 상호작용 위한 마이그레이션 (?)
4. 관리자 계정 만들기

Settings.py
각각의 항목을 아는 것이 중요

Django 뜯어보기 2

Manage.py (*중요)

어떻게 활용할지가 중요. 여러가지 기능을 제공 "아~ 이런 기능을 제공하는구나 정도"

1. 서버 켜기
   서버 실행 : python manage.py runserver => http://127.0.0.1:8000/ (8000은 포트 번호)
   서버 끄기 : ctrl + c

2. 장고의 작은 구성 단위, 프로젝트를 구성하는 요소인 application 만들기

   게시판, 구매, 장바구니 기능, 독립적인 3가지 기능을 각각 묶어서 관리하는 게 좋음 => 각각 다른 앱으로 구성 => 각 어플 묶어 쇼핑몰 프로젝트 완성!
   *특정 기능 하는 어플리케이션 만들고 어플이 모여 웹사이트 만든다.

   어플 만들기 : python manage.py startapp dashboard(프로젝트 이름) => 프로젝트 생성됨

   (*중요) Application 만든 후 **settings.py에** 만든 application을 **등록**해야 함
   Settings.py => INSTALLED_APPS 리스트 안에 등록
   - 'dashboard(프로젝트 이름)' 만 써서 등록하기
   - 'dashboard.apps.DashboardConfig' 까지 써서 등록 (dash~폴더 안에 apps.py 안에 있는 클래스 이름으로 등록

3. 데이터베이스와 상호작용 위한 마이그레이션 (?)
4. 관리자 계정 만들기

Models.py : DB와 상호작용
Views.py : 장고 웹서비스 내 동작 관할, 논리를 입력
Templates 폴더 : 사용자에게 보여지는 부분 관리 (html 들어감)

Django 뜯어보기 2

Manage.py (*중요)

1. 서버 켜기
2. 장고의 작은 구성 단위, 프로젝트를 구성하는 요소인 application 만들기
3. 데이터베이스와 상호작용 위한 마이그레이션 (?)

데이터베이스 초기화 OR 데이터베이스 변경사항 있을 시 (마이그레이션)
Python manage.py migrate

4. 관리자 계정 만들기

Python manage.py createsuperuser => admin(관리자이름) => 1234(password) => 완성
Python manage.py runserver => http://127.0.0.1:8000/admin => 관리자 페이지로 이동

Settings.py "외울 필요는 없지만 어떤 역할인지 정도는 알아라"

- **BASE_DIR :** 프로젝트 주소(위치), 직접 건드릴 일 없음
- **SECRET_KEY :** 암호화 필요할 때 사용하는 문자열 (외부에 노출하면 안 됨)
- ***DEBUG = True :** 어떻게 서버를 킬 것인지 결정.
예시) 존재하지 않는 서버 url 입력해도 개발자들이 보기 편하게 정보 제공. 배포시 False 해야!
- **INSTALLED_APP :** 어플리케이션 적어두기. 만든 외부 패키지들을 적어준다 (pip 포함하는 것인가?)
- **DATABASES :** 어떤 DB 쓸 것이며 어디에 있는지 씀. 실제 데이터베이스와 연결해주는 플러그
- **Internalization :** 언어, 시간 … (LANGUAGE_CODE = 'ko-kr' / TIME_ZONE = 'Asia/Seoul' 해보기)
- **STATIC_URL :** html, css, js … 우리 웹서비스에서 미리 준비한 정적인 대상들의 위치

Hello World 1

본격적 장고 개발

1. Python -m venv myvenv : 가상환경 생성
2. Python source myvenv/Scripts/activate : 가상환경 실행
3. Pip install Django : 가상환경 내 장고 설치
4. Pip freeze : 잘 다운 받아졌는지 확인 (설치된 패키지 목록 확인)
5. Django-admin startproject helloworld : 가상환경 내 프로젝트 생성
6. Cd helloworld : 폴더 이동
7. Python manage.py runserver : 서버 실행
8. 웹브라우저로 해당 화면으로 이동하여 서버 구동 확인
9. Python manage.py startapp myapp : 장고의 작은 단위인 앱 만들기
10. Settings.py => INSTALLED_APPS 내 'myapp' OR 'myapp.apps.MyappConfig' 입력하여 등록
    클래스 이름으로 지정해주는 것이 안정적이다. (왜??)
11. Myapp 내 templates 폴더 생성 후 index.html 생성 : Helloworld 열 때 필요한 html 만듦
12. <h1>처음으로 Django로 만든 페이지~</h1> : html 작성
13. Views.py => 함수 작성 => 어떤 요청이 들어오면 index.html 보여줘라
    def home(request):
        return render(request, 'index.html')
    요청과 함께 index.html을 찍어서 보내줘라 (랜더링 해줘라) 라는 뜻
14. Urls.py => path 이하 첫번째 요소 주소로 요청 들어오면, 두번째 요소를 실행해라
    Urlpatterns = [
        Path('admin/', admin.site.urls),
        Path('/foo', myapp.views.count),
        Path(' ', myapp.views.home, name='hellow_world'), #url이름 지정,, 별 영향X
    ]
    => http://127.0.0.1:8000/foo 접속시 count 함수 실행하라
    => http://127.0.0.1:8000 접속시 myapp폴더 내 views.py 내의 home 함수 실행하라
    => index.html 불러온다 => <h1>처음으로 Django 로 만든 페이지~</h1> 뜬다