4강. 관리자 페이지 활용하기

`python manage.py createsuperuser`

계정명 : jiyoonbaekbaek

이메일 : 내꺼

비밀번호 : 네이버 비번이랑 동일

사이트 확인하기

`python manage.py runserver`

서버주소/admin ⇒ 관리자 페이지

Post 모델을 편리하게 admin에서 관리하기

how?

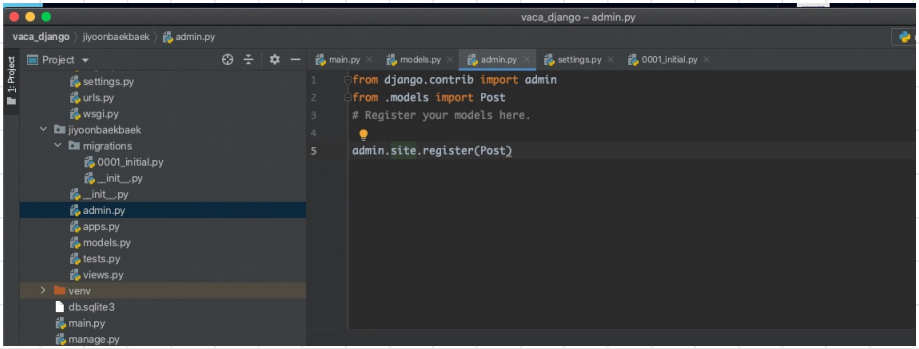
jiyoonbaekbaek 앱 내의 admin.py 접속

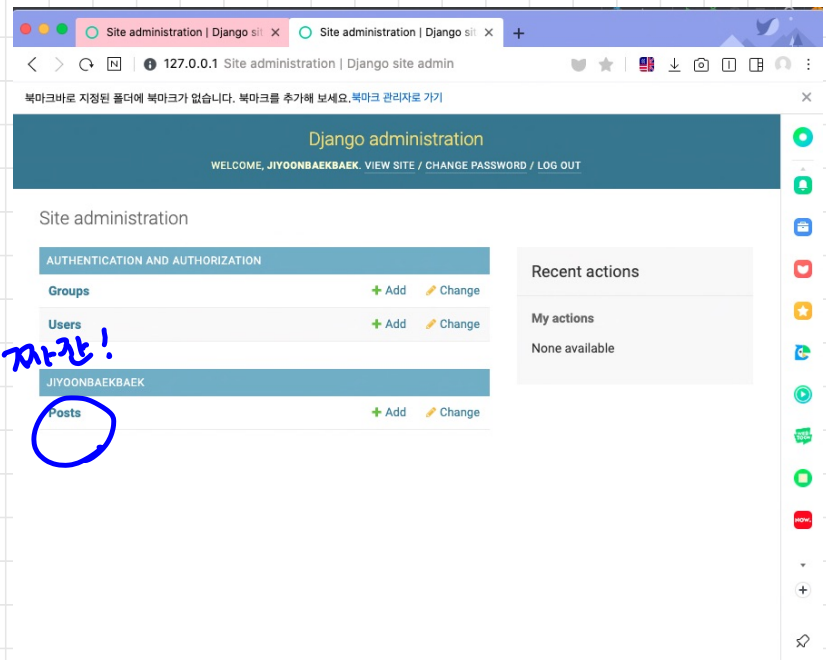
admin.py : 관리자 관련 파일

모델을 관리자 페이지에서 관리하고 싶으면 해당 내용 여기에 추가하면 됨

같은 경로

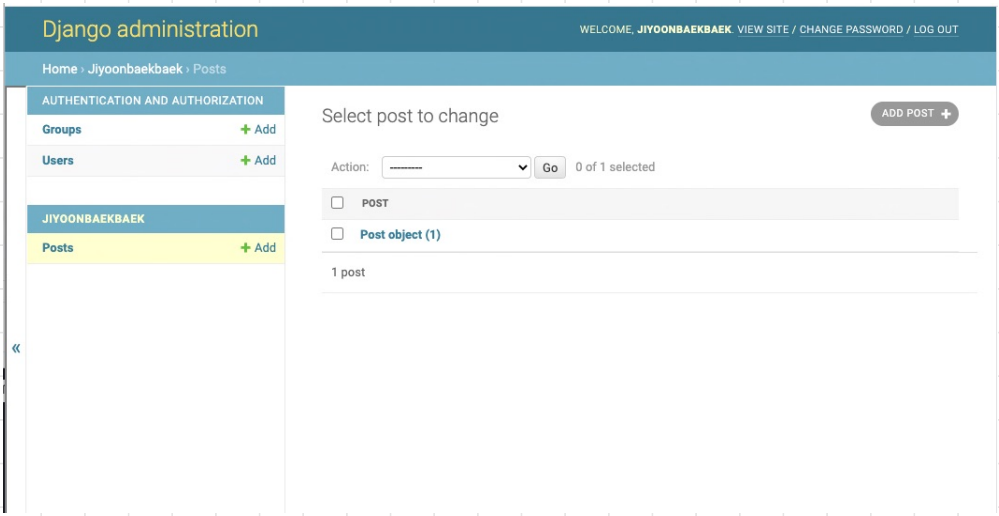
```
from .models import Post
admin.site.register(Post)
```


vaca_django – admin.py
vaca_django
jiyoonbaekbaek
admin.py
Project
settings.py
urls.py
wsgi.py
jiyoonbaekbaek
migrations
0001_initial.py
__init__.py
__init__.py
admin.py
apps.py
models.py
tests.py
views.py
venv
db.sqlite3
main.py
manage.py
main.py
models.py
admin.py
settings.py
0001_initial.py
from django.contrib import admin
from .models import Post
# Register your models here.
admin.site.register(Post)


Site administration | Django sit
Site administration | Django sit
127.0.0.1 Site administration | Django site admin
북마크바로 지정된 폴더에 북마크가 없습니다. 북마크를 추가해 보세요. 북마크 관리자로 가기
Django administration
WELCOME, JIYOONBAEKBAEK. VIEW SITE / CHANGE PASSWORD / LOG OUT
Site administration
AUTHENTICATION AND AUTHORIZATION
Groups + Add Change
Users + Add Change
JIYOONBAEKBAEK
Posts + Add Change
Recent actions
My actions
None available
까간!

페이지 가독성을 위해 커스토마이징하고 싶다면,

<before>



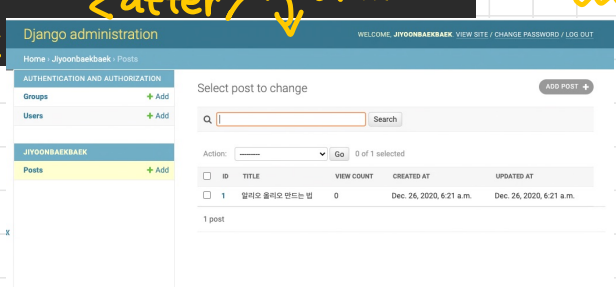
admin.py의 코드를 이와 같이 바꿔야 함

```python
from django.contrib import admin
from .models import Post
# Register your models here.

@admin.register(Post)
class PostAdmin(admin.ModelAdmin):
    list_display = (
        'id',
        'title',
        'view_count',
        'created_at',
        'updated_at'
    ) #list_display : 원하는 컬럼들을 보여줌 "이러한 정보들을 테이블 형태로 보고 싶어"

    search_fields = (
        'title',
    ) #search_fields : "title 로 검색을 가능하게 할꺼야 "
```

데코레이터 사용!

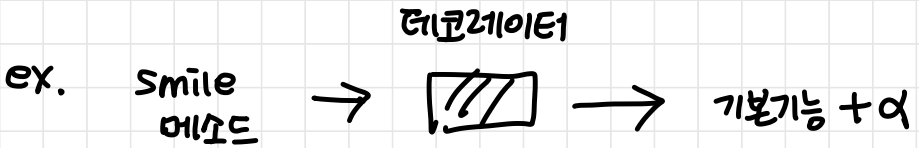
이 외에도 list_filter, list_display_links도 있음!

<after> 이제 CRUD가 가독성있게 admin에서 가능



* 데코레이터

; 보강해주는 역할을 하는 함수 또는 클래스



smile 메소드

```
>>> def deco(func):    # 데코레이터 함수, 그냥 줄여서 '데코레이터'라고도 함
        def df():
            print('emoticon!')      # 추가된 기능
            func()       # 원래 갖고 있던 기능
            print('emoticon!')      # 추가된 기능
        return df        # 보강된 기능의 함수를 반환
>>> smile = deco(smile)      # smile 함수 전달하고 반환 결과를 smile에 저장
>>> smile()      # 기능이 보강된 smile 함수 호출
emoticon!
^_^
emoticon!
```

α

반환 결과를 smile에 저장 (함수 이름도 일종의 변수여서 옮겨붙일 수 있다!)

기능 보강을 하고자 하는 함수에 인자가 있다면?

```
>>> def adder_deco(func):    # 데코레이터 함수
        def ad(*args):   # 전달 인자를 튜플로 묶는다.
            print(*args, sep = ' + ', end = ' ')
            print("= {0}".format(func(*args)))
        return ad

>>> adder2 = adder_deco(adder2)
>>> adder2(3, 4)
3 + 4 = 7
>>> adder3 = adder_deco(adder3)
>>> adder3(1, 2, 3)
```

함수선언기 호출기 호출기

∴패킹

언패킹

언패킹

튜플 패킹, 언패킹 활용!

- 선언하는 위치에 * ; 패킹
- 호출하는 위치에 * ; 언패킹(분리)

@ 기반으로 더 간편하게 데코레이터를 쓸 수 있다!

```
>>> def deco(func):  # 데코레이터 함수
        def df():
            print('emoticon!')
            func()
            print('emoticon!')
        return df    # 보강된 기능의 함수를 반환
>>> def smile(): # 웃는 얼굴 출력
        print("^_^")

>>> smile = deco(smile)
>>> smile()
```

@deco "이어서 내가 정의하는 함수는 deco 라는 데코레이터를 통과시킬꺼야. 그리고 smile은 이 기능이 보강된 함수를 참조하도록 할꺼야"

```python
from django.contrib import admin
from .models import Post
# Register your models here.

@admin.register(Post)
class PostAdmin(admin.ModelAdmin):
    list_display=(
        'id',
        'title',
        'view_count',
        'created_at',
        'updated_at'
    ) #list_display : 원하는 컬럼들을 보여줌 "이러한 정보들을 테이블 형태로 보고 싶어"

    search_fields=(
        'title',
    ) #search_fields : "title 로 검색을 가능하게 할꺼야 "
```

admin.register(Post)
: models.py에 있는 모델 Post를 관리자 페이지에 등록시켜 줌

1) PostAdmin이 ModelAdmin을 상속받아서 가독성을 위해 원하는 기능 사용

2) admin.register(Post) 데코레이터를 통해 PostAdmin 클래스가 모델 Post를 관리자 페이지에 가독성 있게 등록시켜 주도록 함