

## Virtual Environment

pip install virtualenv

# តើក្នុង proj - python → រាយការ virtual env  
នៅ



តើ install នឹង → p1 នូវ

ឯករាជ្យ → នៅលើ venv នឹងនាំ django profle រាយការ  
ឬ ចារម្បាន នៅលើកណែនាំ folder នៃវត្ថុ week នឹងបានដោយទេ?

### Windows

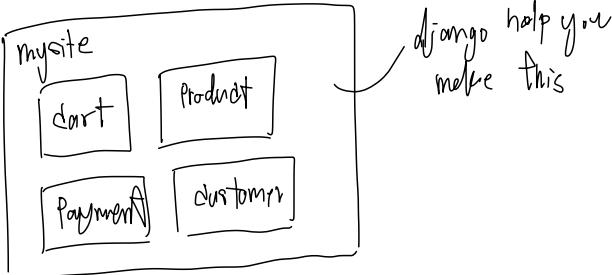
```
# Install virtualenv  
> pip install virtualenv  
  
# Create a virtual environment  
> py -m venv myvenv  
  
# Activate virtual environment  
> myvenv\Scripts\activate.bat
```

### Windows & MacOS

```
> pip install django
```

ត្រូវសរសៃរាយថា install ត្រូវបានដោយ command

```
> python -m django --version  
4.2.13
```



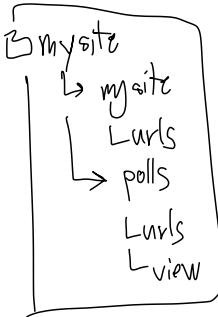
request path

http://localhost/polls/index

① คุณต้องหา polls URL ที่อยู่ใน request ของหน้าเว็บ  
แล้ว add ไปยัง function ของ view ที่ responsible ให้  
(คุณต้องมีmysite)  
② คุณต้อง search @file ของ project ในหน้า path นี้ว่ามีอะไร?

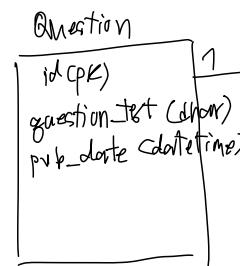
ที่ Link ของ polls → polls.url

๔. โครงสร้าง folder proj.

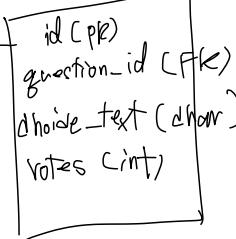


Database

one to many



choice



แบบฟอร์ม class

เราสามารถสร้าง models กัน โดยใน app polls ของเรารี้มี 2 models  
ได้แก่ Question และ Choice โดยเพิ่ม code ด้านล่างลงในไฟล์  
polls/models.py

```
from django.db import models

class Question(models.Model):
    question_text = models.CharField(max_length=200)
    pub_date = models.DateTimeField("date published")

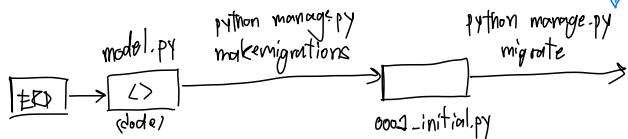
class Choice(models.Model):
    question = models.ForeignKey(Question, on_delete=models.CASCADE)
    choice_text = models.CharField(max_length=200)
    votes = models.IntegerField(default=0)
```

ก็จะต้องมี PK, PostgreSQL จะ  
ต้องมี id ของ PK  
PK ต้องมาต่อ

on delete=models.CASCADE

โดยจะลบตัวหนึ่ง → ลบตัวที่相连กัน

migration → รัน code migrate ที่จะสร้าง table ลง db



django

ไม่สามารถเขียน migration ลง db ได้

edit หรือ เรียก make migration  
จะหันมาดู django ที่ initialize  
แล้วจะ migrate ลง

insert

python manage.py shell  
from polls.models import Question, Choice

give instant @python shell → ex. รัน Question (question\_text, pub\_date, create\_by)

↳ สร้าง q1 = Question (..., ..., ..., ...)  
q1.save()

update

↳ q1.question\_text = "กี่วัน"  
↳ q1.save() ทำแล้ว

delete → q1.delete()

query

→ Question.objects.all() → ตรวจสอบ  
↳ return contains คืน list ฟังก์ชันนี้ ไม่มี [] → ต้องมีสิ่ง (๑๒) นี้อยู่  
showing Attributes Only It . ๑๗

เรามาลองใช้ API ของ Django ในการ query ข้อมูล (SELECT query)

```

>>> Question.objects.all()
<QuerySet [<Question: What is up?>, <Question: Hello world?>]

>>> Question.objects.filter(question_text__icontains="Hello") WHERE, text like "Hello"
<QuerySet [<Question: Hello world?>]>

>>> Question.objects.filter(question_text__startswith="What") ที่ question_text ที่เริ่มต้นด้วย What
<QuerySet [<Question: What is up?>]>

>>> Choice.objects.filter(question_id=1) django รัน question_id คือ primary key
<QuerySet [<Choice: Yes>, <Choice: No>, <Choice: OK>]>

>>> Choice.objects.filter(question_id=1, choice_text="Yes") 2 condition ที่ question_id=1 และ choice_text="Yes"
<QuerySet [<Choice: Yes>]>

>>> Question.objects.first()
<Question: What is up?>

>>> Question.objects.last()
<Question: Hello world?>
  
```

ที่ต้อง case sensitive

question\_text\_\_icontains = "Hello"

question\_text\_\_startswith = "What"

choice\_id = 1

choice\_text = "Yes"

View details

url → scenario → view

∴ .py → Python Script

{ } → Python echo

```
ass Blog(models.Model):
    title = models.CharField(max_length=100)
    content = models.TextField()
    like = models.IntegerField(default=0)
    created_by = models.ForeignKey("blogs.Author", on_delete=models.PROTECT)
    created_date = models.DateTimeField(auto_now_add=True)
    categories = models.ManyToManyField("blogs.Category")
# https://docs.djangoproject.com/en/5.0/topics/db/examples/many_to_many/

def __str__(self):
    return f"{self.title} ({self.like})"

ass Comment(models.Model):
    comment = models.CharField(max_length=200)
    comment = models.ForeignKey("blogs.Blog", on_delete=models.CASCADE)
    created_date = models.DateTimeField(auto_now_add=True)
```

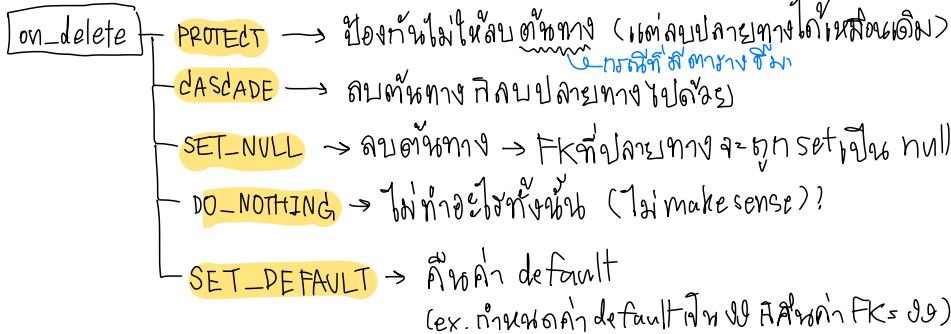
กูนີ້ໃຫຍ່ນ models → make migrations  
migrate ໂອດມີຢັງທີ່ກະບຽນ DB ດີເລີກ

ໄດ້ມີຂໍ້ມູນ \_id ລາຍລຳ, django ໂດຍ

(fk) ຕົວກິທຸນຂອງ on\_delete

ຈົດຕັ້ງຂໍ້ມູນຂອງ (Authors)

ໃນ current timestamp ບໍ່ໄດ້ໃຫຍ່



DATE - TIME

[ကျမ်းမြေစာမျက်နှာ]

DateTime module มีทั้งหมด 5 class หลัก

1. date - มี attributes ได้แก่ year, month, และ day
  2. time - มี attributes ได้แก่ hour, minute, second, microsecond, และ tzinfo
  3. datetime - คือการรวม date และ time และมี attributes ได้แก่ year, month, day, hour, minute, second, microsecond, and tzinfo
  4. timedelta - เป็นระยะเวลา (microsecond) ซึ่งเป็นส่วนต่างของ 2 date, time หรือ datetime **ผลลัพธ์จะอยู่ใน datetime 2 อัน**
  5. tzinfo - เป็น object สำหรับเก็บข้อมูล time zone

ต้อง import module datetime ของภาษา Python  
 import datetime as dt  
 $d_1 = dt.date(2024, 10, 1)$  ผลลัพธ์จะเป็นวันที่  
 $t_1 = dt.time(10, 10, 0)$   $\rightarrow 10:10$  ผลลัพธ์จะเป็นเวลา  
 $dt_1 = dt.datetime(2024, 10, 1, 10, 10, 0)$   
 ↴ # คือ ต้องใส่ ส่วนของวันที่ ไม่ใส่ก็ได้, default=0

Naire datetime → datetime នៃវត្ថុទិន្នន័យ

aware datetime } → ~~using~~ timezone

```
dt_2 = dt.datetime(2024, 10, 5)
```

$$dt_2 - dt_1 = \underline{\text{timedelta}} \ (\text{days} = 4)$$

(ex) `dt1 + timedelta2 (days=10)`

```
> from zoneinfo import ZoneInfo
> from datetime import datetime

> dt1 = datetime(2015, 5, 21, 12, 0)
> print(dt1)
15-05-21 12:00:00
> dt2 = datetime(2015, 12, 21, 12, 0, tzinfo = ZoneInfo(key='Asia/Bangkok'))
> print(dt2)
15-12-21 12:00:00+07:00

> print("Naive Object :", dt1.tzname())
Naive Object : None
> print("Aware Object :", dt2.tzname())
Aware Object : +07

> now_aware = dt1.replace(tzinfo=ZoneInfo(key='UTC'))
> print(now_aware)
15-05-21 12:00:00+00:00
```

timezone.makeaware(dt1)

↳ available by default

enum → fig ការណែនាំ មួយ ឱ្យ គិតអរគុណភាពរូប (String)  
ឬ ឱ្យការពារជាបញ្ជី (dict) នៃវត្ថុនេះ charField

# WEEK 4

## Making Queries

```
from datetime import date
from django.db import models

class Blog(models.Model):
    name = models.CharField(max_length=100)
    tagline = models.TextField()

    def __str__(self):
        return self.name

class Author(models.Model):
    name = models.CharField(max_length=200)
    email = models.EmailField()

    def __str__(self):
        return self.name

class Entry(models.Model):
    blog = models.ForeignKey(Blog, on_delete=models.CASCADE)
    headline = models.CharField(max_length=255)
    body_text = models.TextField()
    pub_date = models.DateField(default=date.today)
    mod_date = models.DateTimeField(default=date.today)
    authors = models.ManyToManyField(Author)
    number_of_comments = models.IntegerField(default=0)
    number_of_pingbacks = models.IntegerField(default=0)
    rating = models.IntegerField(default=5)

    def __str__(self):
        return self.headline
```

class = table → 1 ท่ากัน



[CREATING OBJ.]

### Creating objects

ใน Django จะใช้หลักการตั้งค่า model class จะเปรียบเสมือน database table และ instance ของ class นั้นๆ จะเปรียบเสมือน 1 record ใน table

การสร้าง instance ของ class model ก็สามารถทำได้ง่ายๆ ดังนี้

NOTE: ให้เปิด Django shell ขึ้นมา (`python manage.py shell`) และพิมพ์คำสั่งดังนี้

จาก `Blog.objects.create(Fieldname="")`

```
>>> from blogs.models import Blog
>>> b = Blog(name="Beatles Blog", tagline="All the latest B")
>>> b.save()
```

ซึ่ง Django จะเป็นการ generate SQL command `INSERT` เมื่อเราสั่ง `save()`

### Saving changes to objects

ในการบันทึกการแก้ไข record ที่มีอยู่ใน database เล็กๆ ไปใช้ `save()` เช่น เผชากัน

```
>>> b.name = "New name"
>>> b.save()
```

ซึ่ง Django จะเป็นการ generate SQL command `UPDATE` เมื่อเราสั่ง `save()`

## JUPITER NOTEBOOK

# #Quiz Week 6 របៀប setup JUPITER របស់ខ្លួន

- ↳ ត្រូវ run python, និងកែល dell, run នូវការអនុវត្តន៍
- ↳ ត្រូវចាត់បញ្ជាប់ run តាមការគាំទ្រ #
- ↳ នូវការទូទាត់ក្នុង markdown.

## RETRIEVING OBJ.

### Retrieving objects

ការ SELECT មួយឯកតារាង database នៃការងារ Django នឹង API ដើម្បីធ្វើនាយករាយ ដែលមានការរាយការណ៍ query ដើម្បីទូទាត់ពីទីតាំងទៅទំនាក់ទំនង។

A QuerySet represents a collection of objects from your database. It can have zero, one or many filters. Filters narrow down the query results based on the given parameters. In SQL terms, a QuerySet equates to a SELECT statement, and a filter is a limiting clause such as WHERE or LIMIT.

នៅទីនេះ API នៃ Django ធ្វើការរួមទៅ Manager នៃ class models.Model ដែលបានធ្វើឡើង Manager មានចំណាំ .objects ដែលជាដំឡើងនៃការ SELECT មួយឯកតារាងនៅរាយ។

```
>>> Entry.objects.all() # SELECT + FROM entry;
```

### Retrieving specific objects with filters

ពីនេះបានផ្តល់ព័ត៌មាន filter conditions ណាន៉ែង (ដើម្បីកែតារា SELECT នៅក្នុង query)

ការពិនិត្យចំណាំនៃសមតាតរួមទៅការ QuerySet នៃ blog entries នាពី 2010

```
>>> Entry.objects.filter(pub_date__year=2010)
```

```

• date, year, month, day, week, week_day

-- Entry.objects.filter(pub_date__year=2005)
-- Entry.objects.filter(pub_date__year__gte=2005)
SELECT ... WHERE pub_date BETWEEN '2005-01-01' AND '2005-12'
SELECT ... WHERE pub_date >= '2005-01-01';

• isnull

-- Entry.objects.filter(pub_date__isnull=True)
SELECT ... WHERE pub_date IS NULL;

• regex

-- Entry.objects.get(title__regex=r'(An|The) +')
SELECT ... WHERE title ~ '^An?The +'; -- PostgreSQL

HINT: តាមការកែលព័ត៌មាន SQL query អំពីការសមារភាពការរាយដោយប្រើ .query

q = Entry.objects.filter(headline__in=('a', 'b', 'c'))
print(q.query)

```

```

HINT: តាមការកែលព័ត៌មាន SQL query អំពីការសមារភាពការរាយដោយប្រើ .query

q = Entry.objects.filter(headline__in=('a', 'b', 'c'))
print(q.query)

```

- exact
- iexact → = រាយការ close insensitive
- contains

```
-- Entry.objects.filter(headline__contains='Lennon')
SELECT ... WHERE headline LIKE '%Lennon%'
```

- icontains

```
-- Entry.objects.filter(headline__icontains='Lennon')
SELECT ... WHERE headline ILIKE '%Lennon%';
```

- startswith
- endswith
- in

```
-- Entry.objects.filter(headline__in=['a', 'b', 'c'])
SELECT ... WHERE headline IN ('a', 'b', 'c');
```

- gt, gte, lt, lte

```
SELECT ... WHERE id > 4;
```

- range

```
-- Entry.objects.filter(pub_date__range=(start_date, end_date))
SELECT ... WHERE pub_date BETWEEN '2005-01-01' AND '2005-03'
```

នូវការ filter នៅក្នុងការរាយការណ៍ទៅការសមតាត

Exclude → រាយការណ៍ទៅការសមតាត

នូវការកែលចំណាំ

```
>>> one_entry = Entry.objects.get(pk=1)
>>> one_entry = Entry.objects.filter(pk=1).first()
>>> one_entry = Entry.objects.filter(pk=1)[0]
>>> # នៅទី 3 នរណ៍ថ្មីត្រូវបានដោឡូង
```

ត្រូវការកែល obj. នៅក្នុង → នូវការកែតារា condition នៃការ return នូវការកែតារា

# Field Lookups

#00 หัวข้อ

QuerySet API ของ Django ช่วยให้เขียน query ซึ่งบุกที่เกี่ยวข้องกับตารางเดียวที่ relationship กันได้ร่างง่าย โดย Django จะไปจัดการเรื่องการ generate SQL JOINs ให้ฟรีๆ นั่นเอง

ยกตัวอย่างเช่น ถ้าเราต้องการที่จะดู blog Entry ทั้งหมดของ Blog ที่มี name = "Beatles Blog"

```
>>> Entry.objects.filter(blog__name='Beatles Blog')  
>>> Entry.objects.filter(blog__name__contains='Beatles Blog')
```

สังเกตว่าเราต้องมาใช้ field foreign key มาก่อนแล้วก็ field ของตารางที่ต้องการไปโดยต้อง underscore ไว้ เช่น blog\_\_name - และอีกหนึ่งอย่างที่สำคัญคือ lookup type ให้เลือก

หากต้องการดูblog query ข้อมูลไปต่อไป ก็ต้องมาดู

```
Blog.objects.filter(entry__authors__name='Lennon')
```

```
Blog.objects.filter(entry__authors__name__isnull=True)
```

Filters can reference fields on the model

ในการตั้งค่าต่อๆ กันของ relation ของ field ใน model กับ field อื่นใน model เดียวกัน เราสามารถใช้ F expressions ได้()

```
>>> from django.db.models import F  
>>> Entry.objects.filter(number_of_comments__gt=F('number_o  
>>> Entry.objects.filter(authors__name=F('blog__name')) # si
```

join ทำสิ่งที่ต้อง? แบบ?

```
Blog.objects.filter(entry__authors__name='Lennon')  
Blog.objects.filter(entry__authors__name__isnull=True)
```

blog ที่มี field entry ไม่ join ไม่ได้ต้อง  
ต้อง one-to-many อยู่

# FILTER

## Filters can reference fields on the model

ในกรณีที่เราต้องการเมริยอนเทียบค่าของ field ใน model กับ field อื่นใน model เดียวกัน เราสามารถใช้ F expressions ได้ F()

```
>>> from django.db.models import F  
>>> Entry.objects.filter(number_of_comments__gt=F("number_o  
>>> Entry.objects.filter(authors__name=F("blog__name")) # si
```

โปรดอย่ารบกวน F อย่างไร  
3 Field

โดย Django นั้น support การใช้ +, -, \*, / ร่วมกับ F() ด้วย เช่น

```
>>> Entry.objects.filter(number_of_comments__gt=F("number_o  
>>> Entry.objects.filter(rating__lt=F("number_of_comments"))
```

## Complex lookups with Q objects

Keyword argument ที่ส่งเข้าไปใน method filter() ทุกด้วยจะถูกเอา去 generate เป็น SELECT ... WHERE ... AND ... สมอ เช่น

```
-- Entry.objects.filter(headline__contains='Lennon', pub_da  
SELECT * FROM entry WHERE headline LIKE '%Lennon%' AND pub_d
```

ในกรณีที่เราต้องการทำ query ที่ขั้นซ้อน อาจจะต้องการใช้ OR หรือ NOT ร่วมด้วย เราจะต้องใช้ Q objects

กรณี OR

```
Entry.objects.filter(Q(headline__startswith="Who") | Q(headline__st  
# SELECT ... WHERE headline LIKE 'Who%' OR headline LIKE 'W
```

กับกันสองอย่าง OR

กรณี NOT

```
Entry.objects.filter(Q(headline__startswith="Who") | ~Q(pub.  
# SELECT ... WHERE headline LIKE 'Who%' OR pub_date NOT BET
```

## WEEK5 Making Queries (Advanced??)

Aggregate function → total number of books

annotate → new column first  
intuitive DB

Aggregation → sum, count var

SQL query 2 min  
 num\_employees -- gt = F("num\_chairs")  
 greater than  
 Tomcat

```
In [9]: Book.objects.filter(publisher__name='Penguin Books').aggregate(avg_rate=Avg('rating'))
Out[9]: {'avg_rate': 4.3400000000000001}

In [11]: Book.objects.filter(publisher__name='Oxford University Press').aggregate(avg_rate=Avg('rating'))
Out[11]: {'avg_rate': 4.246153846153847}

In [13]: pubs = Publisher.objects.annotate(num_books=Count("book")).values()

Out[13]: <QuerySet [{"id": 1, "name": "Penguin Books", "num_books": 20}, {"id": 2, "name": "Oxford University Press", "num_books": 39}]>

In [16]: Publisher.objects.annotate(avg_rating=Avg('book__rating')).values()
Out[16]: <QuerySet [{"id": 1, "name": "Penguin Books", "avg_rating": 4.3400000000000001}, {"id": 2, "name": "Oxford University Press", "avg_rating": 4.246153846153847}>

In [22]: Author.objects.annotate(avg_auth=Avg('book__rating')).filter(avg_auth__gt=4.2).values()
Out[22]: <QuerySet [{"id": 22, "name": "Joseph Heller", "age": 76, "avg_auth": 4.3}, {"id": 4, "name": "Jane Austen", "age": 41, "avg_auth": 4.4}, {"id": 6, "name": "Herman Melville", "age": 72, "avg_auth": 4.4}, {"id": 2, "name": "George Orwell", "age": 46, "avg_auth": 4.425}, {"id": 7, "name": "Leo Tolstoy", "age": 82, "avg_auth": 4.4}, {"id": 20, "name": "Emily Bronte", "age": 30, "avg_auth": 4.333333333333333}, {"id": 1, "name": "F. Scott Fitzgerald", "age": 44, "avg_auth": 4.3999999999999995}, {"id": 18, "name": "Ray Bradbury", "age": 91, "avg_auth": 4.5}, {"id": 11, "name": "Gabriel Garcia Marquez", "age": 87, "avg_auth": 4.25}, {"id": 9, "name": "Fyodor Dostoevsky", "age": 59, "avg_auth": 4.34}, {"id": 16, "name": "Mary Shelley", "age": 53, "avg_auth": 4.5}, {"id": 12, "name": "Aldous Huxley", "age": 69, "avg_auth": 4.4}]>
```

In [ ]:

publisher(s)  
publisher book-set, all

1 author of same book

1 author of same book set

to column of authors

```
b.book_set.filter(name__startswith="The").values_list("id", flat=True)
6, 9, 15, 18>
```

list

## Many-to-many relationships

```
In [25]: from books.models import Book, Author  
a1 = Author.objects.get(pk=1)  
a2 = Author.objects.get(pk=2)  
  
book = Book.objects.get(pk=10)  
book.authors.add(a1, a2)  
  
book.authors.all()
```

หน้าที่ที่ต้องทำ

```
Out[25]: <QuerySet [
```

```
In [28]: a1.book_set.add(book)
```

หน้าที่ที่ต้องทำ

```
In [29]: a1.book_set.all()
```

```
Out[29]: <QuerySet [<Book: Book object (2)>, <Book: Book object (10)>, <Book: Book object (21)>, <Book: Book object (31)>, <Book: Book object (51)>]>
```

สามารถทำการ ยกเลิก ความสัมพันธ์ ได้โดยใช้ `remove()` หรือ `clear()` ผู้ดูแลระบบสามารถลบความสัมพันธ์ทั้งหมด

```
>>> book = Book.objects.get(pk=10)  
>>> book.authors.remove(a1)  
>>> book.authors.all()  
<QuerySet [>>> book.authors.clear()  
>>> book.authors.all()  
<QuerySet []>
```

} บล็อก clear ผู้ดูแล.  
คุณต้อง bridge entity.