KAZAKH KBTU BRITISH
TECHNICAL
UNIVERSITY

BACKEND FRAMEWORK (DJANGO)

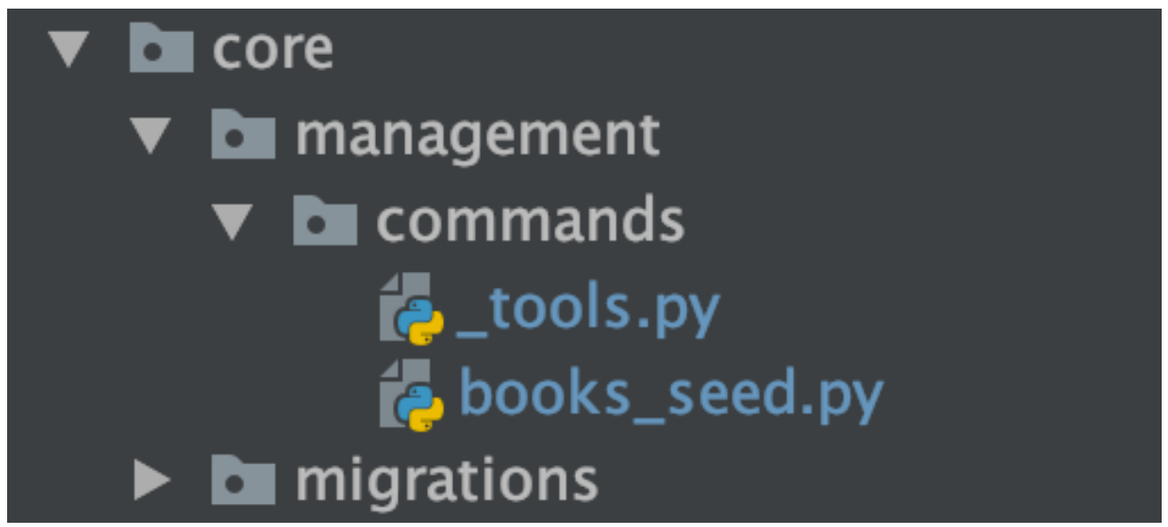
Lesson 11

Django Management Command

commands executed from the command line
using the **`manage.py`** script

Django Management Command

- Folder structure - **`app/management/commands`**
- All files from this folder will be available, except files which started from “_” not available
- In this example, the **`books_seed`** command will be available, but **`_tools`** will not



books_seed.py

- `Command` class which extends from `django.core.management.base.BaseCommand`
- `handle` — function will be executed, when runs from command line
- `stdout.write` — printing message to console
- `add_arguments` — add argument to command
  - required arguments
  - optional arguments
  - flag arguments

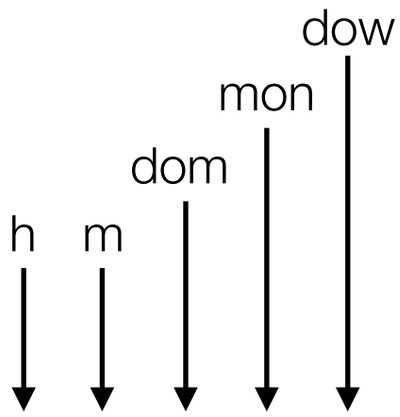
Available Styles

```python
from django.core.management.base import BaseCommand


class Command(BaseCommand):
    help = 'Show all available styles'

    def handle(self, *args, **kwargs):
        self.stdout.write(self.style.ERROR('error - A major error.'))
        self.stdout.write(self.style.NOTICE('notice - A minor error.'))
        self.stdout.write(self.style.SUCCESS('success - A success.'))
        self.stdout.write(self.style.WARNING('warning - A warning.'))
        self.stdout.write(self.style.SQL_FIELD('sql_field - The name of a model field in SQL.'))
        self.stdout.write(self.style.SQL_COLTYPE('sql_coltype - The type of a model field in SQL.'))
        self.stdout.write(self.style.SQL_KEYWORD('sql_keyword - An SQL keyword.'))
        self.stdout.write(self.style.SQL_TABLE('sql_table - The name of a model in SQL.'))
        self.stdout.write(self.style.HTTP_INFO('http_info - A 1XX HTTP Informational server response.'))
        self.stdout.write(self.style.HTTP_SUCCESS('http_success - A 2XX HTTP Success server response.'))
        self.stdout.write(self.style.HTTP_NOT_MODIFIED('http_not_modified - A 304 HTTP Not Modified server response.'))
        self.stdout.write(
            self.style.HTTP_REDIRECT('http_redirect - A 3XX HTTP Redirect server response other than 304.'))
        self.stdout.write(self.style.HTTP_NOT_FOUND('http_not_found - A 404 HTTP Not Found server response.'))
        self.stdout.write(
            self.style.HTTP_BAD_REQUEST('http_bad_request - A 4XX HTTP Bad Request server response other than 404.'))
        self.stdout.write(self.style.HTTP_SERVER_ERROR('http_server_error - A 5XX HTTP Server Error response.'))
        self.stdout.write(self.style.MIGRATE_HEADING('migrate_heading - A heading in a migrations management command.'))
        self.stdout.write(self.style.MIGRATE_LABEL('migrate_label - A migration name.'))
```

Periodic tasks




```
0 3 * * * /home/venv/bin/python /home/demo/manage.py report
```

The example above will execute the **`report`** every day at 3 a.m.

Object **R**elational **M**apper

interaction wit DB

Database access optimization

=/EQUAL and !=/NOT EQUAL queries: **exact, iexact**

```
Project.objects.get(id__exact=1)

Project.objects.get(id=1)
```

```
Project.objects.filter(name__exact="Project 1")

Project.objects.filter(name="Project 1")


Project.objects.filter(name__iexact="project 1")
```

```
WHERE <field>=<value>
```

!= or NOT EQUAL query with **exclude()** and **Q** objects

```
from django.db.models import Q

Project.objects.exclude(status="not finished")

Project.objects.exclude(tasks_count__gte=20)

Project.objects.exclude(~Q(tasks_count__gte=20))
```

```
WHERE NOT <field>=<value>
```

AND queries

```
from django.db.models import Q

Project.objects.filter(creator_id=1, status="finished")
```

```
Project.objects.filter(Q(creator_id=1) &
                       ~Q(status="finished"))
```

```
WHERE <field_1> AND <field_2>
```

OR queries: **Q()** objects

```
from django.db.models import Q


Project.objects.filter(Q(creator_id=1) |
                       ~Q(status="finished"))
```

```
Project.objects.filter(Q(creator_id=1) |
                       Q(creator_id=2))
```

```
WHERE <field_1> OR <field_2>
```

IS and **IS NOT** queries: **isnull**

```
Project.objects.filter(description=None)

Project.objects.filter(description__isnull=True)

Project.objects.filter(description__isnull=False)
```

```
WHERE <field_1> IS NULL / IS NOT NULL
```

Django **IN** queries

```
Project.objects.filter(creator_id__in=[1,2,3])
```

```
Project.objects.filter(status__in=[“new”, “in progress”])
```

```
WHERE <field_1> IN (1, 2, 3)
```

Django **LIKE** queries

contains, icontains, startswith, istartswith, endswith, iendswith

```
Project.objects.filter(name__contains="ject")
```

```
Project.objects.filter(name__istartswith="pro")
```

```
WHERE <field_1> LIKE %PATTERN%
```

Django **GREATER THAN** and **LESSER THAN** queries

gt, gte, lt, lte

```
Project.objects.filter(tasks_count__gt=10)
```

```
Project.objects.filter(tasks_count__lte=10)
```

```
WHERE <field_1>  “>”, “<=” <value>
```

Django date and time queries with field lookups

range, date, year, month, day, week, week_day, time, hour, minute, second

```
Project.objects.filter(created_at__year=2019)
```

```
Project.objects.filter(created_at__hour=10,
                       created_at__minute=25,
                       created_at__second=12)
```

```
WHERE <field_1>=<value>
```

order_by() and **reverse()**

```
Project.objects.all().order_by(“name”)

Project.objects.all().order_by(“-name”)
```

```
ORDER BY <field_1> DESC / ASC
```

LIMIT queries

```
# LIMIT 3
Project.objects.all()[:3]
```

```
# LIMIT 4 OFFSET 6
Project.objects.all()[6:10]
```

```
ORDER BY <field_1> DESC / ASC
```

Aggregation queries

Generating aggregates over a QuerySet

`aggregate()`

```
from django.db.models import Avg, Max, Min, Sum

Project.objects.aggregate(Avg('status'))

Project.objects.aggregate(max_point=Max('points'))

Project.objects.aggregate(tasks_count=Count('tasks'))
```

Generating aggregates for each item in a QuerySet

`annotate()`

```
from django.db.models import Count

Project.objects.annotate(tasks_count=Count('tasks'))

Task.objects.annotate(documents_count=Count('documents'))
```

Combining multiple aggregations

```
from django.db.models import Count

Project.objects.annotate(Count('tasks'), Count('members'))
```

Joins and aggregates

Joins and aggregates

```
from django.db.models import Max, Min, Count

Project.objects.annotate(tasks_count=Count('tasks'))

Project.objects.annotate(
      min_priority=Min('tasks__priority'),
      max_priority=Max('tasks__priority')
)


Project.objects.aggregate(
      min_priority=Min('tasks__priority'),
      max_priority=Max('tasks__priority')
)
```

Group By query

Group By query

`values()`

```
from django.db.models import Count

Project.objects.values('name').annotate(Count('id'))

Project.objects.values('creator').annotate(Count('id'))
```

F expressions

```
from django.db.models import F

Project.objects.filter(members_count__lt=F('tasks_count'))

Project.objects.update(members_count=F('members_count') + 1)
```

Django Debug Toolbar

https://django-debug-toolbar.readthedocs.io/en/1.8/installation.html

Related Objects.
Eager and lazy loading

- **`select_related`** — ForeignKey, OneToOne objects (**SQL JOIN**)
- **`prefetch_related`** — set of objects, ManyToMany, ForeignKey (**Python JOIN**)

Questions?