```
#1. Custom manager:

      class CustomManager(models.Manager):
            def my_custom_query(self):
                  return 'Really hard filtration'


      class MyModel(models.Model):
            field1 = models.CharField(
                  max_length=100,
            )
            custom_manager = CustomManager()

#2. Paginator:

      queryset = Person.objects.all()
      paginator = Paginator(queryset, per_page=10)
      page_number = 2
      print([x for x in paginator.get_page(2)])

#3. Pet:

      class Pet(models.Model):
            name = models.CharField(
                  max_length=40,
            )
            species = models.CharField(
                  max_length=40,
            )


      def create_pet(name: str, species: str):
            Pet.objects.create(
                  name=name,
                  species=species,
            )
            return f"{name} is a very cute {species}!"

#4. Artifact:

      class Artifact(models.Model):
            name = models.CharField(
                  max_length=70,
            )
            origin = models.CharField(
                  max_length=70,
            )
            age = models.PositiveIntegerField()
            description = models.TextField()
            is_magical = models.BooleanField(
                  default=False,
            )

      def create_artifact(name:str, origin:str, age:int, description:str,
is_magical:bool):
            Artifact.objects.create(
                  name=name,
                  origin=origin,
                  age=age,
```

```
                description=description,
                is_magical=is_magical,
            )
            return f"The artifact {name} is {age} yeaars old!"

      def delete_all_artifacts():
            Artifact.objects.all().delete()


#5.

      class Location(models.Model):
            name = models.CharField(
                  max_length=100,
            )
            region = models.CharField(
                  max_length=50,
            )
            population = models.PositiveIntegerField()
            description = models.TextField()
            is_capital = models.BooleanField(
                  default=False,
            )

            def __str__(self):
                  return f'{self.name} has a population of {self.population}'


      def show_all_locations():
            locations = Location.objects.all().order_by('-id')

            return '\n'.join(str(x) for x in locations)


      def new_capital():
            # Location.objects.filter(pk=1).update(is_capital=True)
            location = Location.objects.first()
            location.is_capital = True
            location.save()

      def get_capitals():
            return Location.objects.filter(is_capital=True).values('name')

      def delete_first_location():
            Location.objects.first().delete()


#6.

      class Car(models.Model):
            model = models.CharField(
                  max_length=40,
            )
            year = models.PositiveIntegerField()
            color = models.CharField(
                  max_length=40,
            )
            price = models.DecimalField(
                  max_digits=10,
```

```
                decimal_places=2,
            )
            price_with_discount = models.DecimalField(
                max_digits=10,
                decimal_places=2,
                default=0,
            )


      def apply_discount():
            cars = Car.objects.all()
            for car in cars:
                percentage_off = sum(int(x) for x in str(car.year)) / 100
                discount = car.price * percentage_off
                car.price_with_discount = car.price - discount
                car.save()


      def get_recent_cars():
            return Car.objects.first(year__gte=2020).values('model',
'price_with_discount')


      def delete_last_car():
            Car.objects.last().delete()

#7.

      class Task(models.Model):
            title = models.CharField(
                max_length=25,
            )
            description = models.TextField()
            due_date = models.DateField()
            is_finished = models.BooleanField(
                default=False,
            )

            def __str__(self):
                return f'Task - {self.title} needs to be done until
{self.due_date}!'


      def show_unfinished_tasks():
            unfinished_tasks = Task.objects.filter(is_finished=False)
            return '\n'.join(str(t) for t in unfinished_tasks)


      def complete_odd_tasks():
            for task in Task.objects.all():
                if task.id % 2 != 0:
                    task.is_finished = True
                    task.save()


      def encode_and_replace(text, task_title):
            tasks_with_matching_title = Task.objects.filter(title=task_title)
            decoded_text = ''.join(chr(ord(x) - 3) for x in text)
```

```
            for task in tasks_with_matching_title:
                  task.description = decoded_text
                  task.save()

            # decoded_text = ''.join(chr(ord(x) - 3) for x in text)
            #
Task.objects.filter(title=task_title).update(description=decoded_text)


#8.

      class HotelRoom(models.Model):
            ROOM_CHOICES = (
                  ('Standard', 'Standard'),
                  ('Deluxe', 'Deluxe'),
                  ('Suite', 'Suite'),
            )
            room_number = models.PositiveIntegerField()
            room_type = models.CharField(
                  choices=ROOM_CHOICES,
            )
            capacity = models.PositiveIntegerField()
            amenities = models.PositiveIntegerField()
            price_per_night = models.DecimalField(
                  max_digits=8,
                  decimal_places=2,
            )
            is_reserved = models.BooleanField(
                  max_length=20,
                  default=False,
            )

            def __str__(self):
                  return f'{self.room_type} room with number {self.room_number}
costs {self.price_per_night}$ per night!'


      def get_deluxe_rooms():
            deluxe_rooms = HotelRoom.objects.filter(room_type="Deluxe")
            even_id_deluxe_rooms = []

            for room in deluxe_rooms:
                  if room.id % 2 == 0:
                        even_id_deluxe_rooms.append(str(room))

            return '\n'.join(even_id_deluxe_rooms)


      def increase_room_capacity():
            rooms = HotelRoom.objects.all().order_by("id")
            previous_room_capacity = None
            for room in rooms:
                  if not room.is_reserved:
                        continue
                  if previous_room_capacity:
                        room.capacity += previous_room_capacity
                  else:
                        room.capacity += room.id
```

```
                previous_room_capacity = room.capacity
                room.save()


      def reserve_first_room():
            first_room = HotelRoom.objects.first()
            first_room.is_reserved = True
            first_room.save()


      def delete_last_room():
            last_room = HotelRoom.objects.last()
            if last_room.is_reserved:
                  last_room.delete()

#9.
      from django.db.models import F


      class Character(models.Model):
            CLASS_NAME_CHOICES = (
                  ('Mage', 'Mage'),
                  ('Warrior', 'Warrior'),
                  ('Assassin', 'Assassin'),
                  ('Scout', 'Scout')
            )

            name = models.CharField(
                  max_length=100,
            )
            class_name = models.CharField(
                  max_length=100,
                  choices=CLASS_NAME_CHOICES,
            )
            level = models.PositiveIntegerField()
            strength = models.PositiveIntegerField()
            dexterity = models.PositiveIntegerField()
            intelligence = models.PositiveIntegerField()
            hit_points = models.PositiveIntegerField()
            inventory = models.TextField()


def update_characters():
    # characters = Character.objects.all()
    # for character in characters:
    #     if character.class_name == 'Mage':
    #         character.level += 3
    #         character.intelligence -= 7
    #     elif character.class_name == 'Warrior':
    #         character.hit_points = character.hit_points / 2
    #         character.dexterity += 4
    #     else:
    #         character.inventory = 'The inventory is empty'
    #     character.save()

    Character.objects.filter(class_name='Mage').update(
        level=F('level') + 3,
        intelligence=F('intelligence') - 7
    )
```

```
    Character.objects.filter(class_name='Warrior').update(
        hit_points=F('hit_points') * 0.5,
        dexterity=F('dexterity') + 4
    )
    Character.objects.filter(class_name__in=['Assassin', 'Scout']).update(
        inventory='The inventory is empty',
    )


def fuse_characters(first_character:Character, second_character:Character):
    fusion_name = first_character.name + ' ' + second_character.name
    fusion_level = (first_character.level + second_character.level) // 2
    fusion_class = "Fusion"
    fusion_strength = (first_character.strength + second_character.strength) * 1.2
    fusion_dexterity = (first_character.dexterity + second_character.dexterity) *
1.4
    fusion_intelligence = (first_character.intelligence +
second_character.intelligence) * 1.5
    fusion_hit_points = (first_character.hit_points + second_character.hit_points)
    if first_character.class_name in ["Mage", "Scout"]:
        fusion_inventory = "Bow of the Eleven Lords, Amulet of Eternal Wisdom"
    else:
        fusion_inventory = "Dragon Scale Armor, Excalibur"

    Character.objects.create(
        name=fusion_name,
        level=fusion_level,
        class_name=fusion_class,
        strength=fusion_strength,
        dexterity=fusion_dexterity,
        intelligence=fusion_intelligence,
        hit_points=fusion_hit_points,
        inventory=fusion_inventory,
    )
    first_character.delete()
    second_character.delete()


def grand_dexterity():
    Character.objects.update(dexterity=30)


def grand_intelligence():
    Character.objects.update(intelligence=40)


def grand_strength():
    Character.objects.update(strength=50)


def delete_character():
    Character.objects.filter(inventory='The inventory is empty').delete()
```