the Model

Persist Data

What is a model?

A model is the place where information is stored. It contains all the important fields and behaviors of this information.

Models exist only on app level and there in the file models.py.

Since an app may have more than one model (e.g. articles and comments), there will be several classes in models.py.

Basically, each model is mapped to exactly one database table.

Example for a model

A blog article could be a model.
The article has a headline, a subline, content, an image, an author, and so on.
All these attributes are used in a model.
Technically, a model is just a Python class that inherits from models.Model
and is defined in models.py

```
class Article(models.Model):
    headline = models.CharField(max_length=100)
    subline = models.CharField(max_length=200)
    article_img = models.CharField(max_length=40)
    content = models.TextField()
    pub_date = models.DateTimeField(auto_now_add=True)
```

SQL

A database table is reated based on the model definition. The SQL for this model looks like this

```
CREATE TABLE blog_articles (
    "id" serial NOT NULL PRIMARY KEY,
    "headline" varchar(100) NOT NULL,
    "subline" varchar(200) NOT NULL,
    "article_img" varchar(40) NOT NULL,
    "content" TEXT NOT NULL,
    "pub_date" DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

Migrations

In order to transfer the model definition into the, i.e. to form an SQL statement from it, which creates (or changes) the corresponding database table, so-called migrations are necessary.

Database changes are NEVER made manually!

Migrations are files that are created based on the model definition and are then executed.

Advantage of migrations

Database tables always correspond exactly to models (classes)

Migrations are in version control and can be executed at any time, for example when another developer installs the project locally.

Migrations are abstract, meaning they can be run on any DB supported by Django (ORM).

Create a migration

Migrations are created app-specifically using the manage.py management tool and the makemigrations subcommand, optionally followed by the name of the Django App

```
python manage.py makemigrations pets
python manage.py makemigrations
```

in pets/migrations a file has now been created :

0001_initial.py

the prefix 0001 is a consecutive number and is incremented automatically

Example of a Migration

```
from django.db import migrations, models
class Migration(migrations.Migration):
    initial = True
    dependencies = [
    ]
    operations = [
        migrations.CreateModel(
            name='FirstModel',
            fields=[
                ('id', models.AutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
                ('title', models.CharField(max_length=100)),
                ('text', models.TextField(verbose_name='Meldungstext')),
            ],
        ),
    ]
```

Migration files are changed manually only in rare exceptions!

SQL

To view the generated SQL that would create a migration, we can use the sqlmigrate subcommand.

Mandatory here are the specification of the app (pets) and the number of the migration (is the file prefix of the migration file).

```
python manage.py sqlmigrate pets 0001
```

Transfer migration to database

With makemigrations we have created only one migration file! To convert this into an SQL statement, we use the migrate subcommand of the management tool manage.py

```
python manage.py migrate pets
python manage.py migrate
```

Now a new table has been created in the database (or modified if it already existed).