

Django Quickstart

- **Django-Installation in Debian-Linux**

1. Grundlagen-Pakete installieren

- Auf den Übungs-Servern (scilab-*) **bereits erledigt**
- Als User **root** ...

```
aptitude install ipython3 python3-pip gettext
# Unterstützung für MySQL, WSGI, Sprachunterstützung
aptitude install python3-psycopg2 python-sqlite
# Optional: Unterstützung für weitere DBMS (Postgres, SQLite)
aptitude install ipython3
# Optional: Die Python-Shell iPython
```

2. Django installieren (2 Möglichkeiten)

- Methode A: Als **Debian-Paket**
- Methode B: Mit **PIP** → wählbar, z.B. Django 4.2

– Wir benutzen hier Methode B (PIP)

Django Quickstart

Nicht empfohlen!

- **Alternative: Django-Installation als Debian-Paket**

- Als User root ...

```
aptitude show python-django python3-django  
# Paket-Infos zu "python-django" und "python3-django" anschauen  
  
aptitude install python3-django  
# "python-django" installieren (dauert ca. eine Minute)
```

- Es gibt noch diverse weitere Pakete

```
aptitude search python3-django  
# ca. 50 Erweiterungs-Pakete zu Django werden aufgelistet
```

- Nachteil: Debian-Pakete hier zum Teil nicht sehr aktuell

Django Quickstart

- **Hintergrund: Python Packet-Installer PIP**

- **PIP** ist ein Python-Werkzeug um Python-Pakete zu installieren
 - Funktioniert Unabhängig von den Debian-Paketen
 - Bietet sehr große Zahl von Paketen, meist sehr aktuelle Versionen
- ggf. zunächst **PIP** installieren: Als User root ...

```
aptitude install python3-pip  
# Werkzeug pip installieren
```

- Auf den Übungs-Servern (scilab-*) **bereits erledigt**

Django Quickstart

- **Hintergrund: Python Packet-Installer PIP**

- PIP-Pakete können an 2 Orten installiert werden ...

- ins **System** (als User root)
- in ein **Benutzer-Verzeichnis** (als regulärer Benutzer)

Ab jetzt am
Besten als regulärer
User

- Bei Bedarf zunächst **PIP** Paket-Liste aktualisieren

```
pip3 install --upgrade pip --user  
# als User (oder als root ohne --user ins System installieren)
```

Immer pip**3**
benutzen (nicht pip)

- Paket-Index von PIP (**Suche** nach Paketen)

- <https://pypi.org/search/>
- z.B. zu Django:
C

- Dokumentation zu PIP:

- https://pip.pypa.io/en/stable/user_guide/

Django Quickstart

- **Django-Installation mit PIP**

- **Django** durch PIP installieren: Als regulärer User ...

```
pip3 install Django==4.2 --user  
# Django Version 4.2.x in ~/.local/ installieren
```

- PIP installiert so die Pakete im Benutzer-Verzeichnis `~/.local/`
- Damit die Programme gefunden werden, muss `~/.local/bin` in den Programm-Suchpfad aufgenommen werden
 - Abhängig von der Shell
 - Für csh / tcsh: Eintrag `set path=( ~/.local/bin $path )`
 - systemweit in `/etc/csh.cshrc` (systemweit, als root)
 - oder benutzerlokal in `~/.cshrc`
 - evtl. direkt nach der Installation `'rehash'` um das neue Programm zu finden
 - Auf den Übungs-Servern (scilab-*) systemweit bereits erledigt
- Danach kann man als User `'django-admin'` aufrufen

Web-Application-Framework: Django

- **Django: Es gibt sehr Ausführliche Dokumentation**
 - Homepage: <https://www.djangoproject.com/>
 - Online-Doku Django 4.2: <https://docs.djangoproject.com/en/4.2/>
 - The Django Book: <https://django-book.readthedocs.io/>
 - Freies Online-eBook, Tutorial
 - Aktive Community
 - Viele (zentral und dezentral gepflegte) Module
 - z.B. Django-Snippets (Tricks & Erweiterungen)
 - <https://djangosnippets.org/>
 - z.B. Django im **Python Package Index** (<https://pypi.org/>)
 - <https://pypi.org/search/?q=Django>

Django Quickstart

- **Neues Projekt anlegen (als User)**

- `mkdir django ; cd django`
 - `#` unser Verzeichnis für unsere Django-Projekte
- `django-admin startproject test1`
 - `#` Wir legen ein neues Projekt "test1" an
- `cd test1 ; dir -R`

```
# Schauen wir uns an ...
test1
    manage.py          <-- wir sind hier (aktuelles Verzeichnis)
    test1              <-- unser Management-Programm
                        <-- das Config-Verzeichnis des Projekts
        __init__.py
        settings.py    <-- hier macht man Einstellungen
        urls.py
        wsgi.py
```

- `python3 ./manage.py`
 - `chmod u+x ./manage.py` `#` Script vorher ggf. noch ausführbar machen
 - Evtl. in der ersten Zeile „python“ gegen „python3“ austauschen
 - `#` ab jetzt rufen wir dieses Script nur noch mit „./manage.py“ auf

Django Quickstart (falls SQLite)

- Django ist vorkonfiguriert für SQLite als Datenbank

SQLite benötigt keinen eigenen Datenbank-Server

- Die Daten werden in lokalen Dateien abgelegt
- Siehe <https://www.sqlite.org/docs.html>

- In einfachen Szenarien genügt SQLite zum Betrieb

- Geringe Last, geringe Anforderungen an Zuverlässigkeit, ...

- # Auszug aus **test1/settings.py**

```
DATABASES = { 'default': {  
    'ENGINE': 'django.db.backends.sqlite3',  
    'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),  
} }
```

- `./manage.py migrate`

- Aktualisiert / Erzeugt Datenbank-Schema

- `./manage.py createsuperuser`

- # Legt Admin-Account an (Name, Mailadresse und neues Passwort angeben)

Name der
SQLite-Datei

Django Quickstart (falls SQLite)

- Wie sieht die **SQLite**-Datenbank jetzt aus?

- `./manage.py dbshell`

- # nur interessehalber mal anschauen ...

- `.tables`

```
auth_group
auth_group_permissions
auth_permission
auth_user
auth_user_groups
auth_user_user_permissions
django_admin_log
django_content_type
django_migrations
django_session
```

- # Inhalt der Benutzerdatenbank ...

- `.schema auth_user`

- `SELECT * FROM auth_user;`

- Tipp: evtl. muss man als User root die sqlite3-Tools installieren

```
aptitude install sqlite3
```

```
# Kommando-Frontend für sqlite3 installieren
```

DB-Shell, ruft Shell zum konfigurierten DB-Backend auf (Aufruf unabhängig vom DB-Backend)

Die Kommandos in der DB-Shell hängen aber natürlich vom DBMS ab (hier: **SQLite**)

Tipp: Mit den Kommandos „.headers on“ und „.mode columns“ wird die Ausgabe lesbarer. (Kann man auch in ~/.sqliterc schreiben.)

Django Quickstart (falls **mySQL**)

- **Alternative: z.B. **mysql**-Datenbank**

- **EDIT test1/settings.py**

- # Editiere folgende Zeilen um die Datenbank anzubinden:

```
DATABASES = { 'default': {  
    'ENGINE': 'django.db.backends.mysql',  
    'NAME': 'test1',  
    'USER': 'lamp',  
    'PASSWORD': 'xxx_MEIN_PASSWORT_xxx', } }
```

- **mysql**

- **CREATE DATABASE test1;**
wir legen noch die o.g. DB an
ab jetzt benutzen wir statt "mysql" das Kommando **"./manage.py dbshell"**

- **./manage.py migrate**

- Aktualisiert / Erzeugt Datenbank-Schema
- Die SQL-Operationen kann man vorher mit „./manage.py sqlmigrate“ sehen

- **./manage.py createsuperuser**

- # Legt Admin-Account an (Name, Mailadresse und neues Passwort angeben)

Django Quickstart (falls **mySQL**)

- Wie sieht die **mysql**-Datenbank jetzt aus?

- `./manage.py dbshell`

- # nur interessehalber mal anschauen ...
`SHOW tables;`

```
+-----+
| Tables_in_test1 |
+-----+
| auth_group      |
| auth_group_permissions |
| auth_permission |
| auth_user      |
| auth_user_groups |
| auth_user_user_permissions |
| django_admin_log |
| django_content_type |
| django_migrations |
| django_session  |
+-----+
```

- # Inhalt der Benutzerdatenbank ...
`DESCRIBE auth_user;`
`SELECT * FROM auth_user \G`

DB-Shell, ruft Shell zum konfigurierten DB-Backend auf (Aufruf unabhängig vom DB-Backend)

Die Kommandos in der DB-Shell hängen aber natürlich vom DBMS ab (hier: **mySQL**)

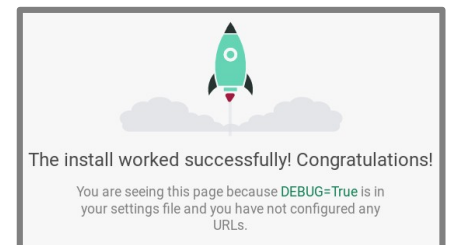
Django Quickstart

- **Start des Test-Servers auf Port 8000**

- `./manage.py runserver 0.0.0.0:8000`
 - `#` Wir starten die Web-Applikation
 - `#` Und können auf der Adresse des Servers mit einem Webbrowser
 - `#` darauf zugreifen, z.B.

- **Web-Client-Zugriff auf den Test-Server**

- z.B. `http://scilab-0100.cs.uni-kl.de:8000/`
 - Fehlermeldung
- **EDIT** `test1/settings.py`
 - `#` Editiere folgende Zeilen um den Server-Hostnamen freizugeben, z.B.:
- Nochmal `http://scilab-0100.cs.uni-kl.de:8000/`
 - „*The install worked successfully!*“

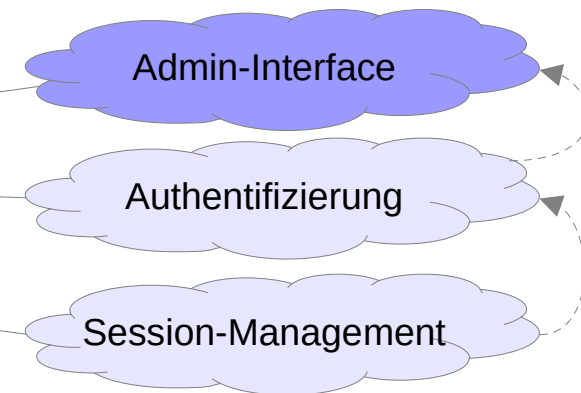


Django Quickstart

- Das **Admin-Interface** ist ein **Django-App**

- Man kann eigene **Apps** schreiben oder existierende nutzen
- Auszug aus `test/settings.py`

- `INSTALLED_APPS = [`
 `'django.contrib.admin',`
 `'django.contrib.auth',`
 `'django.contrib.contenttypes',`
 `'django.contrib.sessions',`
 `'django.contrib.messages',`
 `'django.contrib.staticfiles',`
 `]`



- Auszug aus `test1/urls.py`

- `urlpatterns = [`
 `path('admin/', admin.site.urls),`
 `]`

- Zugriff auf Admin-Interface

- z.B. <http://scilab-0100.cs.uni-kl.de:8000/admin/>

Django Quickstart

- Wir legen das Datenschema als App „Prüfungsamt“ an
 - ./manage.py startapp prüfungsamt
 - # wir legen ein neues Applikationsskelett an
Es gibt jetzt neue Unterverzeichnisse und Dateien, u.a.:
~ / test1 / prüfungsamt / models.py
 admin.py
 - EDIT prüfungsamt/models.py
 - # Modell neu anlegen (nächste Folie) -----
 - EDIT test1/settings.py
 - INSTALLED_APPS = (..., 'prüfungsamt', ...)
 - ./manage.py makemigrations
 - # Anpassung der Datenbank vorbereiten (→ prüfungsamt/migrations/0001_initial.py)
 - ./manage.py migrate
 - # Anpassung der Datenbank (Neue Tabellen anlegen → Wie sehen die aus?)
Hintergrund: Anzeige der Migrationen und der dazu benutzten SQL-Statements:
 - ./manage.py showmigrations # zeigt Status der Migrationen
 - ./manage.py sqlmigrate prüfungsamt 0001 # zeigt SQL-Statements

Django Quickstart

- **pruefungsamt/models.py**

```
from django.db import models
```

```
class Student(models.Model):  
    matnr = models.IntegerField(unique=True)  
    name = models.CharField(max_length=64)  
    hoert = models.ManyToManyField('Vorlesung', blank=True)
```

```
class Professor(models.Model):  
    persnr = models.IntegerField(unique=True)  
    name = models.CharField(max_length=64)
```

```
class Vorlesung(models.Model):  
    vorlnr = models.IntegerField(unique=True)  
    titel = models.CharField(max_length=128)  
    dozent = models.ForeignKey(Professor, null=True,  
                               on_delete=models.SET_NULL)
```

- **Hintergrund-Info:** Beachten Sie, dass ForeignKey / ManyToManyField als ersten Parameter eine Klasse oder deren Namen (String) haben kann. (Warum String?)

Django Quickstart

- **Wir legen das Admin-Interface zur App „Prüfungsamt“ an**
 - EDIT pruefungsamt/admin.py
 - # Admin-Definition anlegen (→ nächste Folie)

Django Quickstart

- **pruefungsamt/admin.py**

```
from .models import *  
from django.contrib import admin
```

Import der Modell-Klassen
von **pruefungsamt** (.),
damit wir *Student*, *Professor*
und *Vorlesung* nutzen können

```
class Student_Admin(admin.ModelAdmin):  
    pass
```

„pass“ ist ein Platzhalter,
da wir hier (noch) nichts weiter
angeben wollen

```
admin.site.register(Student, Student_Admin)
```

```
class Professor_Admin(admin.ModelAdmin):  
    pass
```

```
admin.site.register(Professor, Professor_Admin)
```

```
class Vorlesung_Admin(admin.ModelAdmin):  
    pass
```

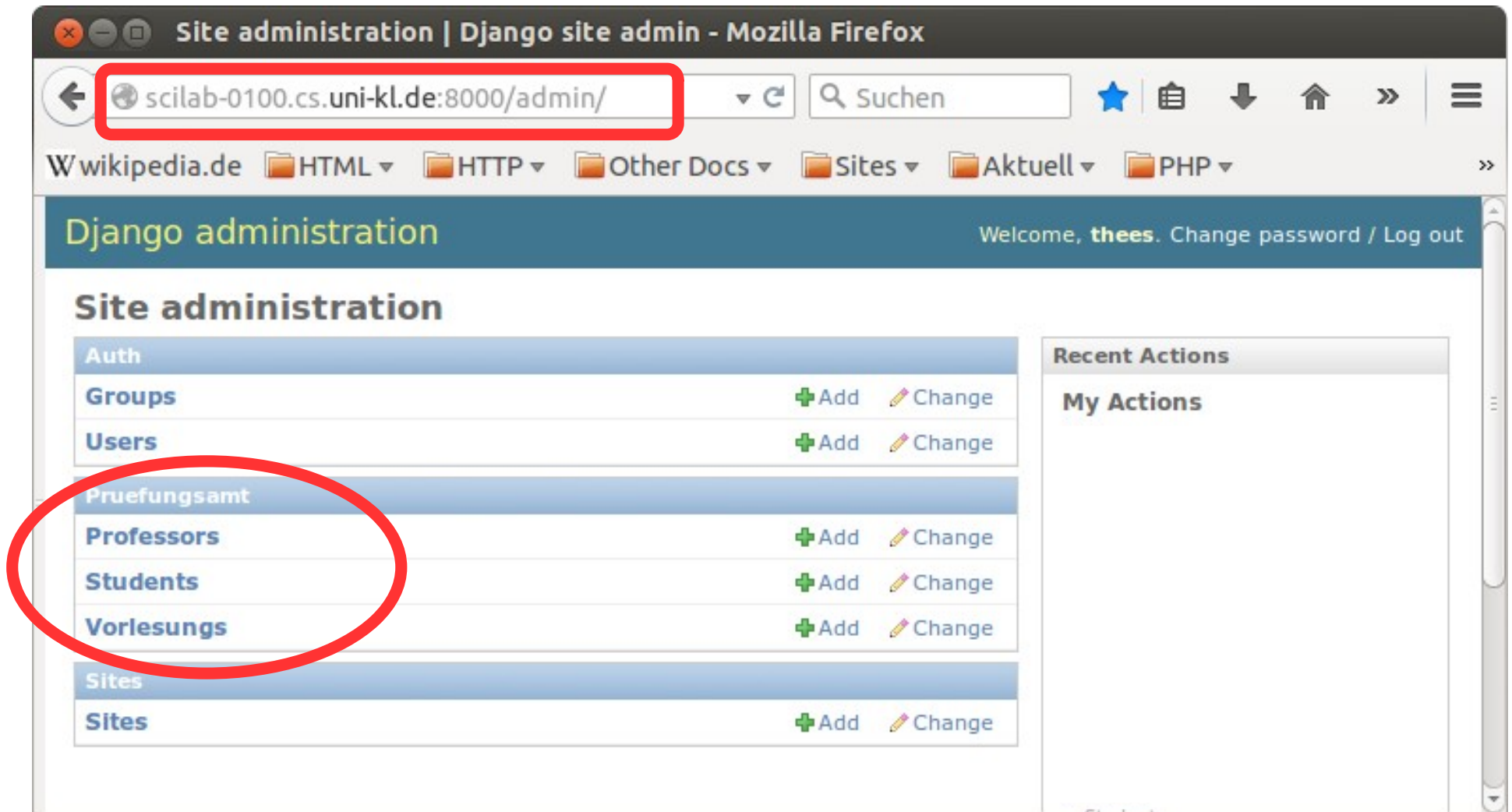
```
admin.site.register(Vorlesung, Vorlesung_Admin)
```

Django Quickstart

- **Danach jeweils den Web-Service starten ...**
 - `./manage.py runserver 0.0.0.0:8000`
 - `0.0.0.0` bedeutet, wir nehmen Verbindungen von jedem Host an
 - `8000` ist die Port-Adresse des Dienstes
- **Web-Client-Zugriff auf den Test-Server**
 - URL z.B. <http://scilab-0100.cs.uni-kl.de:8000/admin/>
 - Im Admin-Interface pflegen wir z.B. die Test-Daten ein

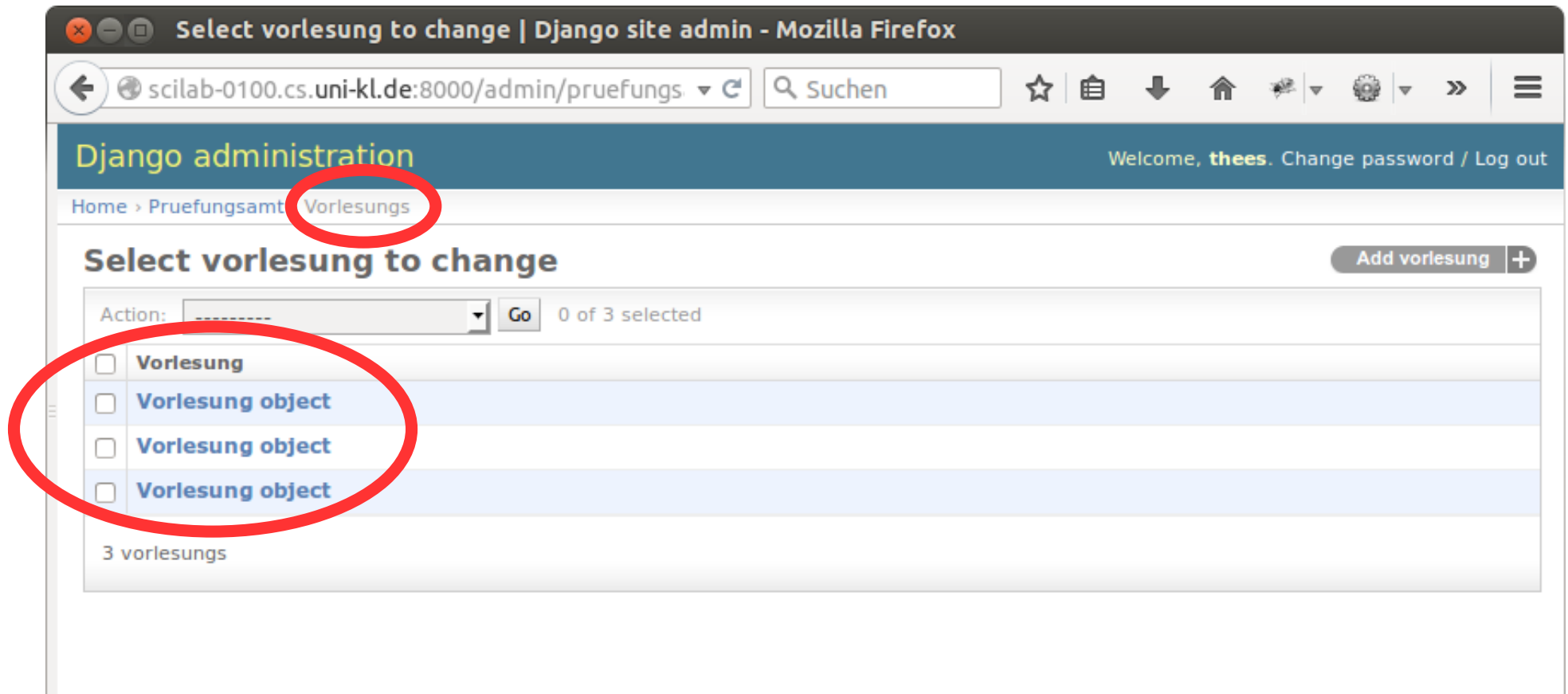
Django Quickstart

- Die 3 Klassen sind im Admin-Interface zugreifbar



Django Quickstart

- Die Liste der Vorlesungen ist aber noch „*unschön*“
 - Wir haben aber ja auch z.B. noch nicht gesagt, was wir an Attributen sehen wollen



Django Quickstart

- **pruefungsamt/admin.py (erweitert)**

```
from .models import *
from django.contrib import admin

class Student_Admin(admin.ModelAdmin):
    list_display = ('matnr', 'name',)
    filter_horizontal = ('hoert',)

admin.site.register(Student, Student_Admin)

class Professor_Admin(admin.ModelAdmin):
    list_display = ('name', 'persnr', )

admin.site.register(Professor, Professor_Admin)

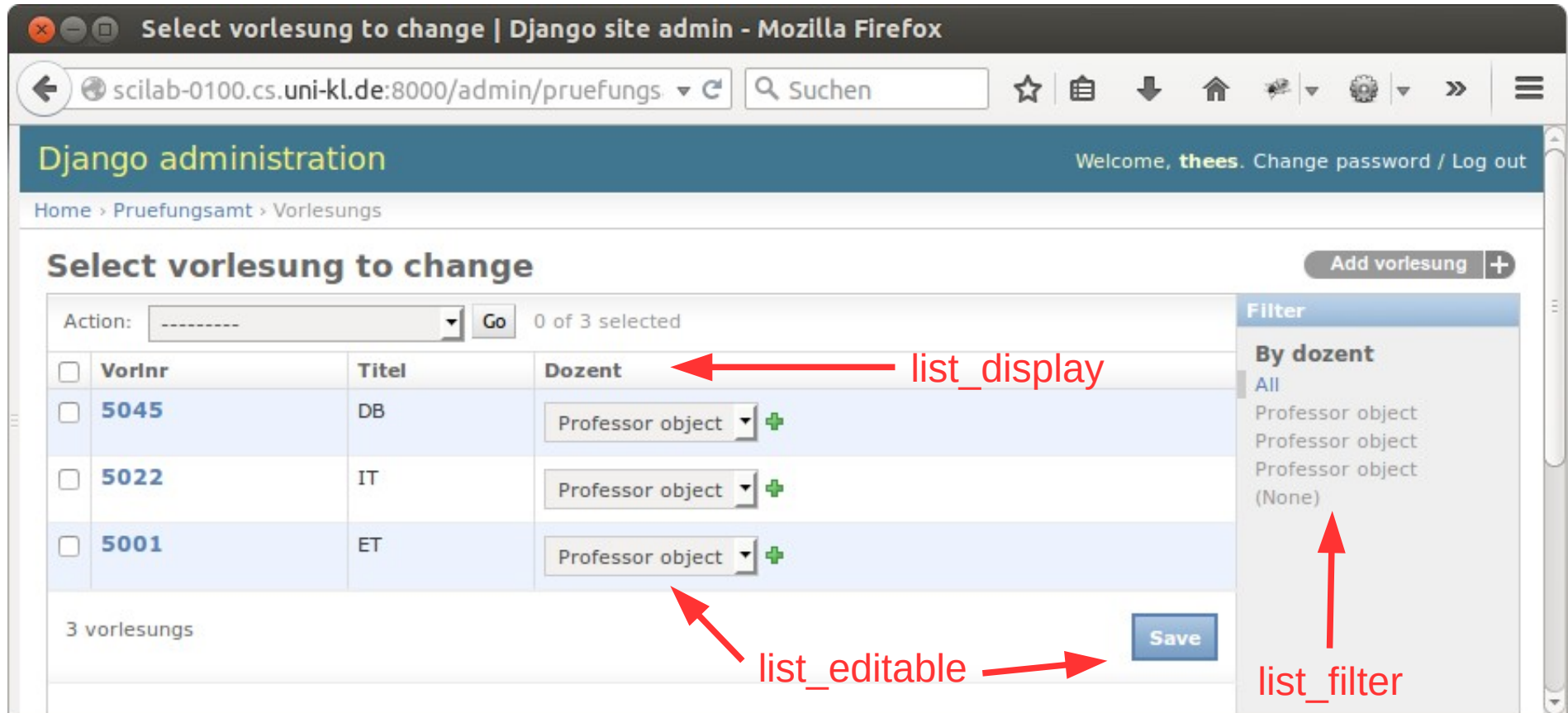
class Vorlesung_Admin(admin.ModelAdmin):
    list_display = ('vorlnr', 'titel', 'dozent',)
    list_filter = ('dozent',)
    list_editable = ('dozent',)

admin.site.register(Vorlesung, Vorlesung_Admin)
```



Django Quickstart

- Die Liste der Vorlesungen ist fast perfekt



- Objekte sollten noch benannt werden (statt „*Professor object*“)
 - Lösung: Jedes Objekt sollte eine String-Darstellung liefern

Django Quickstart

- **Wir ergänzen die Schema-Objekte:**

- um eine Methode `__str__`, die einen lesbaren Text liefert
- um eine **Meta-Klasse**, die Zusatzinformationen liefert
 - z.B. Name der Klasse in der Darstellung (singular / plural)
 - z.B. Standard-Reihenfolge bei Queries

- **pruefungsamt/models.py**

```
class Professor(models.Model):
    persnr = models.IntegerField(unique=True)
    name    = models.CharField(max_length=64)

    def __str__(self):
        return "%s [%s]" % (self.name, self.persnr)

class Meta:
    verbose_name        = 'Professor'
    verbose_name_plural = 'Professoren'
    ordering              = ('name', 'persnr',)
```

Django Quickstart

- Die Liste der Vorlesungen ist jetzt fertig

The screenshot shows the Django administration interface in a Mozilla Firefox browser window. The address bar shows the URL `scilab-0100.cs.uni-kl.de:8000/admin/pruefungs`. The page title is "Select vorlesung to change | Django site admin - Mozilla Firefox".

The main content area is titled "Django administration" and shows the breadcrumb "Home > Pruefungsamt > Vorlesungen". The page is titled "Select vorlesung to change" and includes a button "Add vorlesung +".

Below the title, there is a form with an "Action:" dropdown, a "Go" button, and a status "0 of 3 selected".

The table below shows the list of lectures:

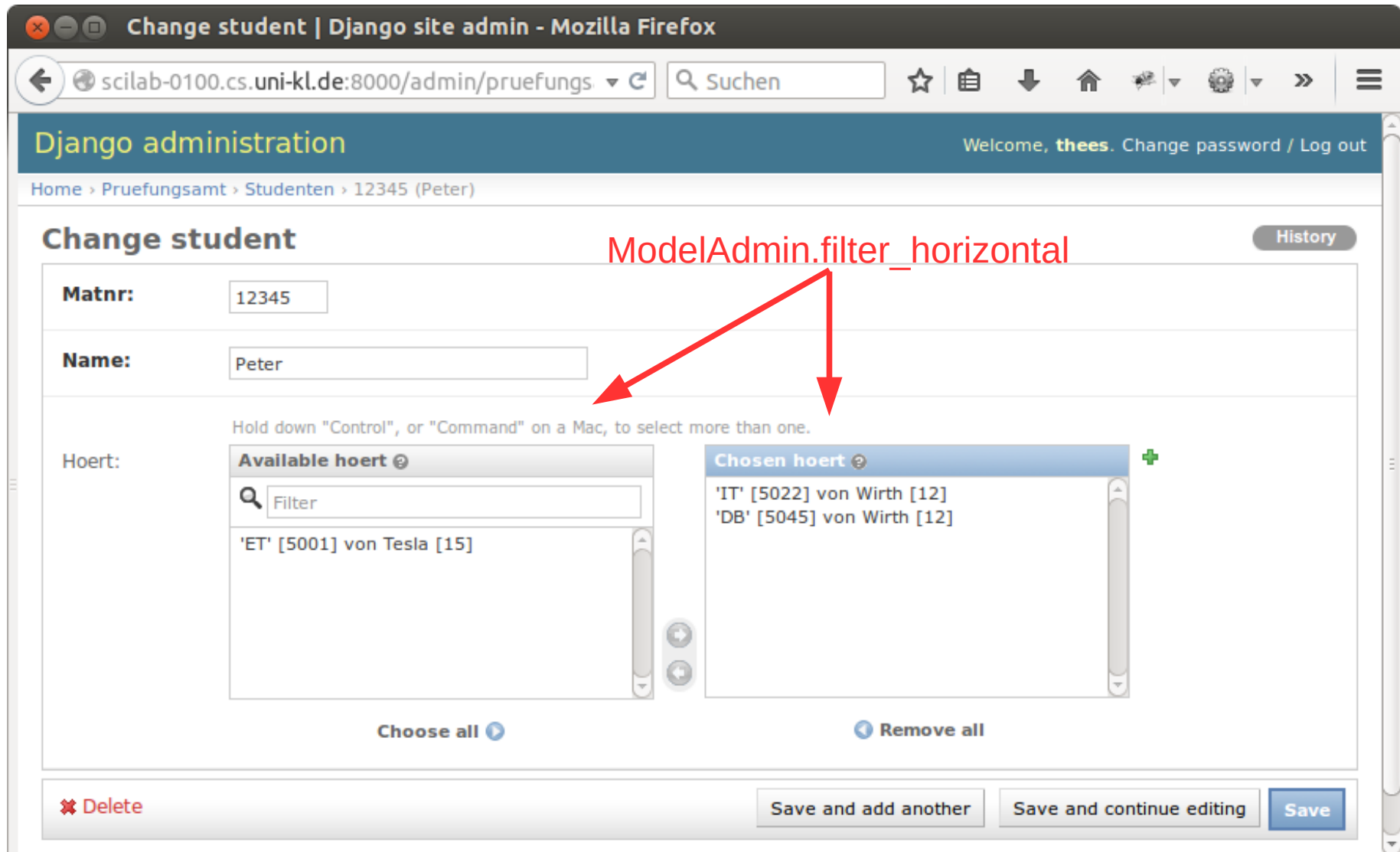
☐	Vorlnr	Titel	Dozent
☐	5001	ET	Wirth [12] +
☐	5022	IT	Wirth [12] +
☐	5045	DB	Tesla [15] +

At the bottom of the table, it says "3 Vorlesungen" and there is a "Save" button.

On the right side, there is a "Filter" sidebar with the title "By dozent". The filter options are: "All", "Tesla [15]", "Urlauber [20]", "Wirth [12]", and "(None)".

Django Quickstart

- Und auch die Editier-Seite ist fertig



Django Quickstart

- **Schema-Zugriff mit der Python-Shell (1)**

- Man kann die Schema-Objekte in eigenen Programmen oder gar interaktiv in einer Python-Shell zugreifen
 - Zu letzterem rufen wir das Django-Kommando „shell“ auf
 - Wenn `ipython` installiert ist, wird dieses als Shell benutzt (mehr Komfort)
- `./manage.py shell`

```
from pruefungsamt.models import *  
  
s = Student()  
s.name = 'Tester'  
s.matnr = 123123  
s.save()
```

- Das neue Objekt ist jetzt dauerhaft in der Datenbank abgelegt
 - Wir könne es z.B. im Admin-Interface sehen

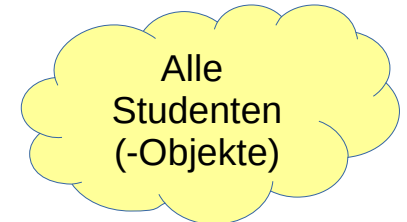
Django Quickstart

- **Schema-Zugriff mit der Python-Shell (2)**

- `./manage.py shell`

- `from pruefungsamt.models import *`
- `Student.objects.all()`

```
[<Student: 12345 (Peter)>, <Student: 25403 (Jonas)>, <Student: 26120 (Buche)>, <Student: 27103 (Fauler)>, <Student: 123123 (Tester)> ]
```



- Solche Query-Sets werden von Django in SQL-Queries umgesetzt

- Hintergrund: Wie sieht die SQL-Anfrage dazu aus?

```
qs = Student.objects.all()
str(qs.query)
```

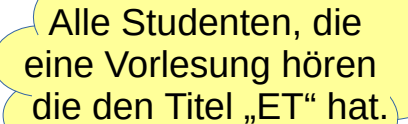
```
SELECT
  `pruefungsamt_student`.`id`,
  `pruefungsamt_student`.`matnr`,
  `pruefungsamt_student`.`name`
FROM `pruefungsamt_student`
ORDER BY (`pruefungsamt_student`.`name` ASC ,
          `pruefungsamt_student`.`matnr` ASC )
```

Django Quickstart

- **Schema-Zugriff mit der Python-Shell (3)**

- `./manage.py shell`

- `from pruefungsamt.models import *`
 - `Student.objects.filter(hoert__titel = 'ET')`
`[<Student: 25403 (Jonas)>, <Student: 26120 (Buche)>]`

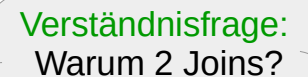


Alle Studenten, die eine Vorlesung hören die den Titel „ET“ hat.

- So kann man komplexe Anfragen stellen

- Hintergrund: Wie sieht die SQL-Anfrage dazu aus?

```
SELECT
    `pruefungsamt_student`.`id`,
    `pruefungsamt_student`.`matnr`,
    `pruefungsamt_student`.`name`
FROM `pruefungsamt_student`
INNER JOIN `pruefungsamt_student_hoert`
    ON (`pruefungsamt_student`.`id` = `pruefungsamt_student_hoert`.`student_id`)
INNER JOIN `pruefungsamt_vorlesung`
    ON (`pruefungsamt_student_hoert`.`vorlesung_id` = `pruefungsamt_vorlesung`.`id`)
WHERE `pruefungsamt_vorlesung`.`titel` = 'ET'
ORDER BY (`pruefungsamt_student`.`name` ASC ,
          `pruefungsamt_student`.`matnr` ASC )
```



Verständnisfrage:
Warum 2 Joins?