بسم الله

django

AUTHENTICATION

HOSSEIN FORGHANI

MAKTAB SHARIF

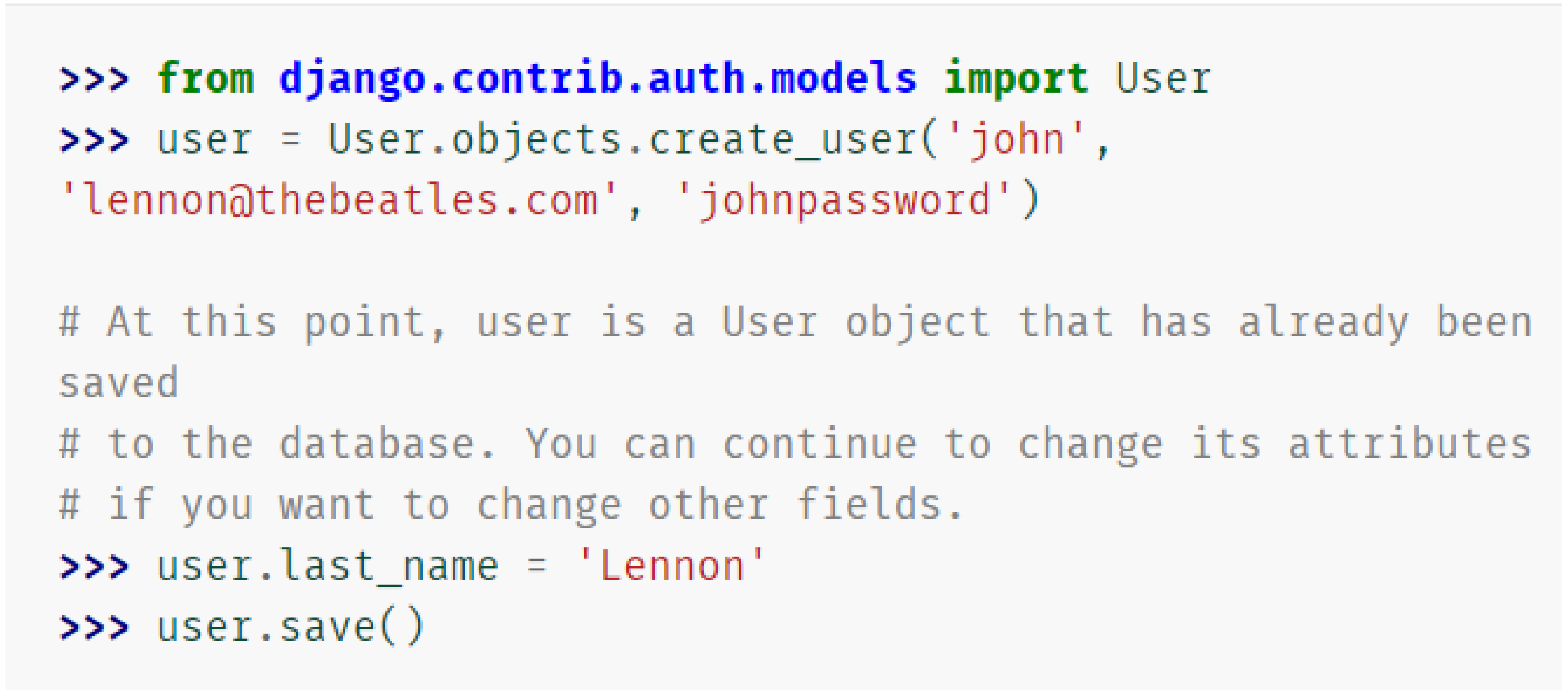
Contents

Installation

- By default, the required configuration is already included in the settings.py
- INSTALLED_APPS
  - django.contrib.auth
  - django.contrib.contenttypes
- MIDDLEWARE
  - SessionMiddleware
  - AuthenticationMiddleware

User objects

- Represent the people interacting with your site
- Creating users:



```
>>> from django.contrib.auth.models import User
>>> user = User.objects.create_user('john',
'lennon@thebeatles.com', 'johnpassword')

# At this point, user is a User object that has already been
saved
# to the database. You can continue to change its attributes
# if you want to change other fields.
>>> user.last_name = 'Lennon'
>>> user.save()
```

Creating superusers

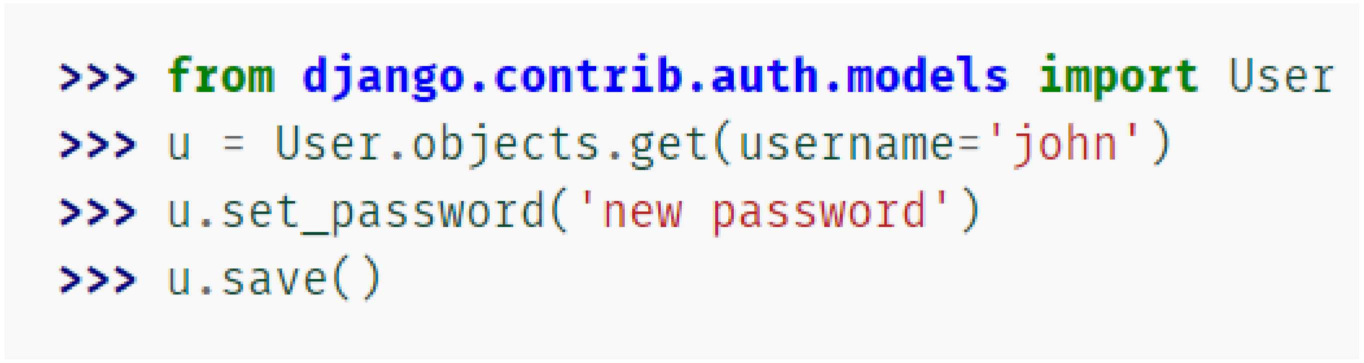
```
$ python manage.py createsuperuser --username=joe --
email=joe@example.com
```

- If you leave off the --username or --email options, it will prompt you for those values

Changing passwords

```
manage.py changepassword *username*
```

- Or programmatically:



```
>>> from django.contrib.auth.models import User
>>> u = User.objects.get(username='john')
>>> u.set_password('new password')
>>> u.save()
```

Authenticating users

```python
from django.contrib.auth import authenticate
user = authenticate(username='john', password='secret')
if user is not None:
    # A backend authenticated the credentials
else:
    # No backend authenticated the credentials
```

- request is an optional HttpRequest which is passed on the authenticate()

Permissions and Authorization

- Remind permissions “view”, “add”, “change”, and “delete” for each type of object in Django admin
- Permissions can be set also per specific object instance:

```
ModelAdmin.has_view_permission(request, obj=None)
```

```
ModelAdmin.has_add_permission(request)
```

```
ModelAdmin.has_change_permission(request, obj=None)
```

```
ModelAdmin.has_delete_permission(request, obj=None)
```

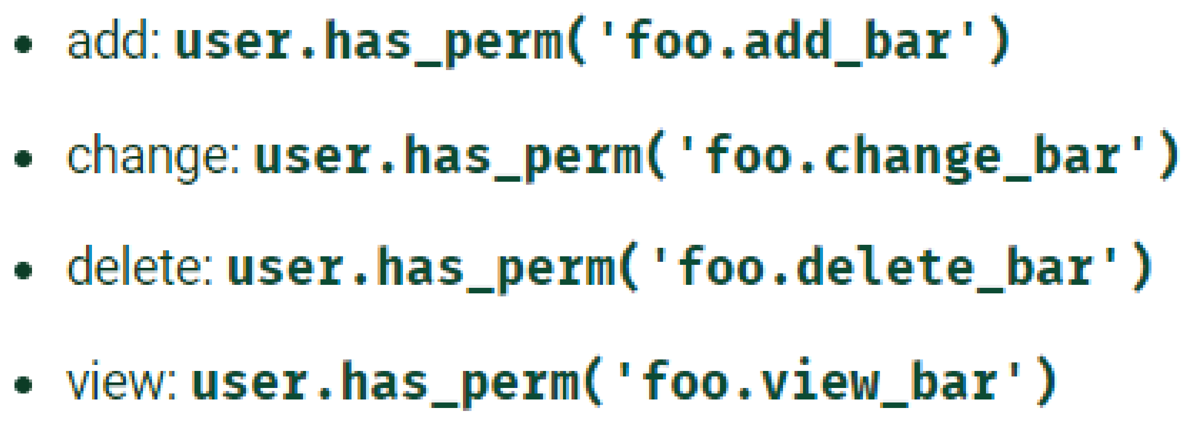
Managing Groups and Permissions

- User objects have two many-to-many fields: groups and user_permissions:

```
myuser.groups.set([group_list])
myuser.groups.add(group, group, ...)
myuser.groups.remove(group, group, ...)
myuser.groups.clear()
myuser.user_permissions.set([permission_list])
myuser.user_permissions.add(permission, permission, ...)
myuser.user_permissions.remove(permission, permission, ...)
myuser.user_permissions.clear()
```

Check Permissions

- Assuming you have an application foo and a model named Bar, to test for basic permissions you should use:



- add: `user.has_perm('foo.add_bar')`
- change: `user.has_perm('foo.change_bar')`
- delete: `user.has_perm('foo.delete_bar')`
- view: `user.has_perm('foo.view_bar')`

Custom Permissions

```
class Task(models.Model):
    ...
    class Meta:
        permissions = [
            ("change_task_status", "Can change the status of
tasks"),
            ("close_task", "Can remove a task by setting its
status as closed"),
        ]
```

Custom Permissions – cont.

- Or you can also create permissions directly:

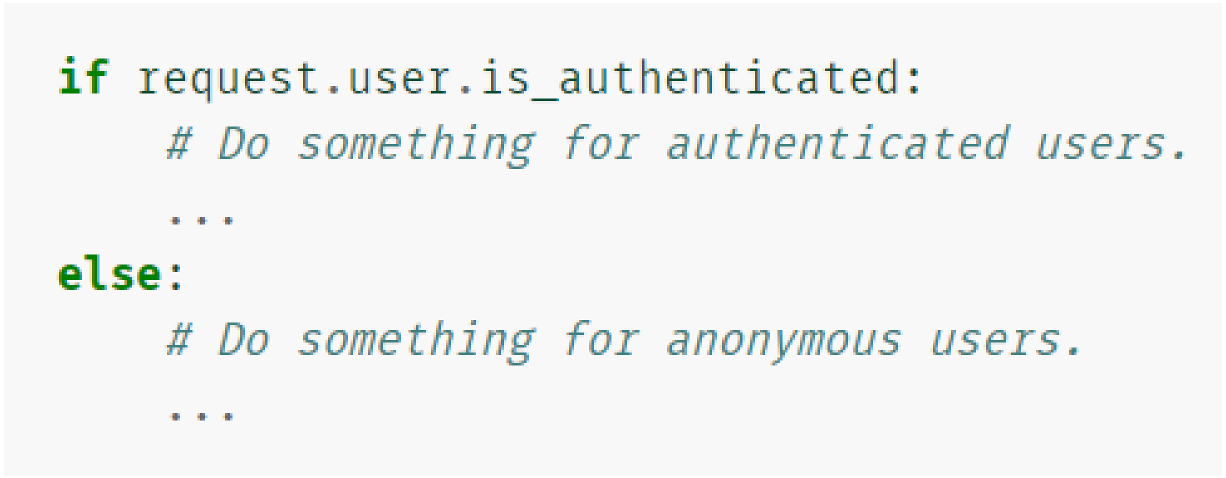
```
from myapp.models import BlogPost
from django.contrib.auth.models import Permission
from django.contrib.contenttypes.models import ContentType

content_type = ContentType.objects.get_for_model(BlogPost)
permission = Permission.objects.create(
    codename='can_publish',
    name='Can Publish Posts',
    content_type=content_type,
)
```

Permission Caching

```python
from django.contrib.auth.models import Permission, User
from django.contrib.contenttypes.models import ContentType
from django.shortcuts import get_object_or_404

from myapp.models import BlogPost

def user_gains_perms(request, user_id):
    user = get_object_or_404(User, pk=user_id)
    # any permission check will cache the current set of permissions
    user.has_perm('myapp.change_blogpost')

    content_type = ContentType.objects.get_for_model(BlogPost)
    permission = Permission.objects.get(
        codename='change_blogpost',
        content_type=content_type,
    )
    user.user_permissions.add(permission)

    # Checking the cached permission set
    user.has_perm('myapp.change_blogpost')  # False

    # Request new instance of User
    # Be aware that user.refresh_from_db() won't clear the cache.
    user = get_object_or_404(User, pk=user_id)

    # Permission cache is repopulated from the database
    user.has_perm('myapp.change_blogpost')  # True
```

Authentication in Web Requests

- request.user attribute on every request
- If the current user has not logged in, this attribute will be set to an instance of AnonymousUser



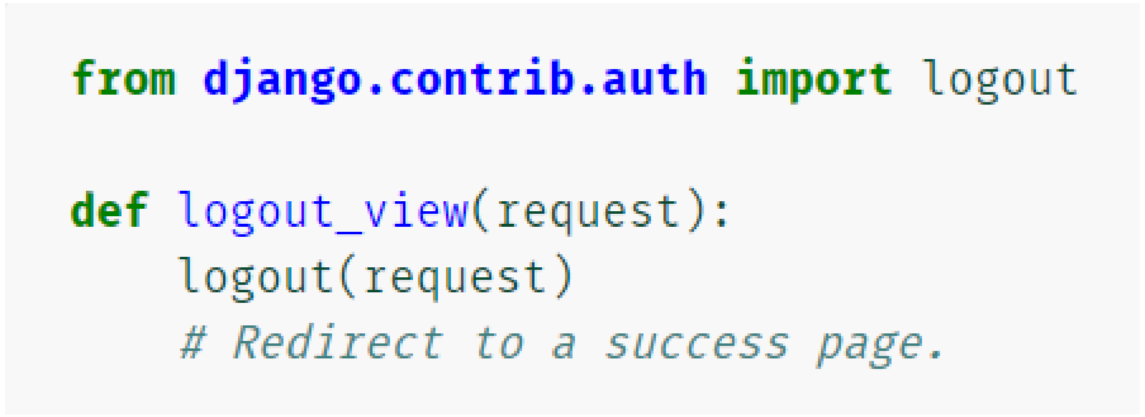
```python
if request.user.is_authenticated:
    # Do something for authenticated users.
    ...
else:
    # Do something for anonymous users.
    ...
```

How to Log a User in

- To log a user in, from a view, use login()
- Session data will be retained

```
from django.contrib.auth import authenticate, login

def my_view(request):
    username = request.POST['username']
    password = request.POST['password']
    user = authenticate(request, username=username,
password=password)
    if user is not None:
        login(request, user)
        # Redirect to a success page.
        ...
    else:
        # Return an 'invalid login' error message.
        ...
```

How to Log a User out



```
from django.contrib.auth import logout

def logout_view(request):
    logout(request)
    # Redirect to a success page.
```

- doesn’t throw any errors if the user wasn’t logged in
- Session will be completely cleaned out

Limiting Access to Logged-in Users

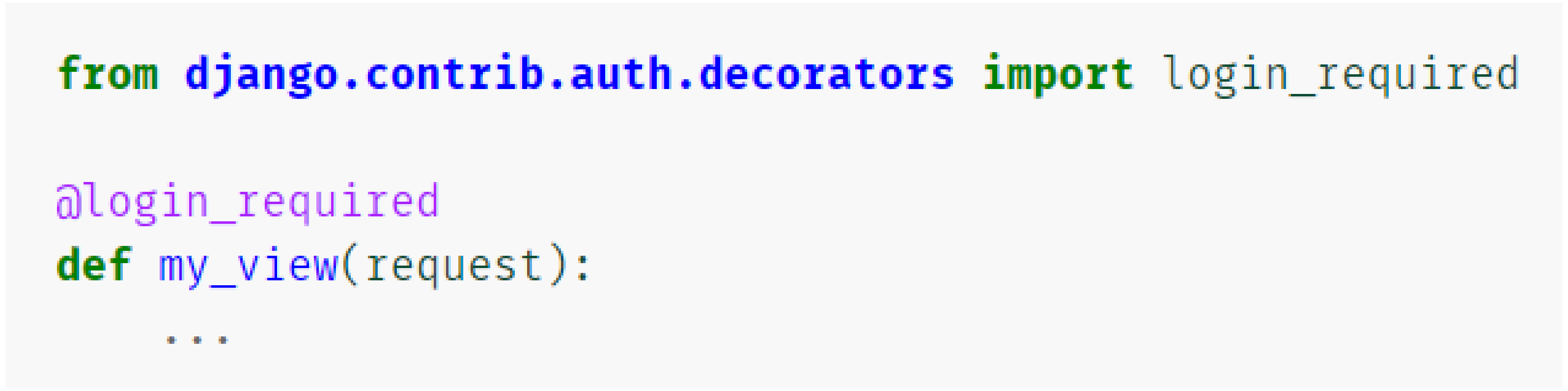
- The raw way:

```
from django.conf import settings
from django.shortcuts import redirect

def my_view(request):
    if not request.user.is_authenticated:
        return redirect('%s?next=%s' % (settings.LOGIN_URL, 
request.path))
    # ...
```

- You can also display an error message

Limiting Access to Logged-in Users – cont.

- The login_required decorator:



```
from django.contrib.auth.decorators import login_required

@login_required
def my_view(request):
    ...
```

- This does exactly same as the previous page

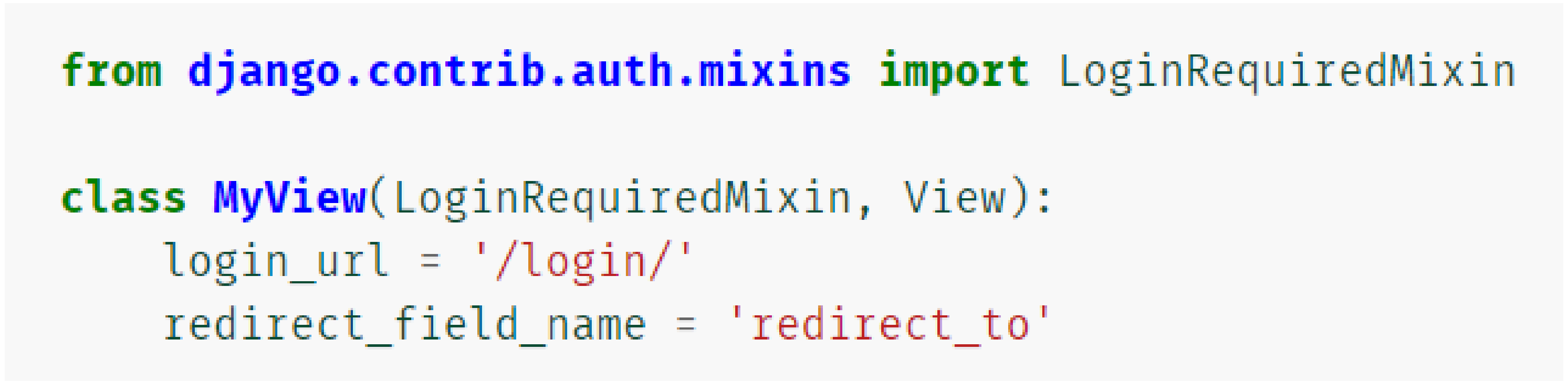
login_required Arguments

- Takes 2 optional arguments:
- redirect_field_name: default is “next”
- login_url: default is “settings.LOGIN_URL”
  - Default value of settings.LOGIN_URL is /accounts/login/

$$\underbrace{\text{/accounts/login/}}_{\text{login\_url}}\text{?}\underbrace{\text{next}}_{\text{redirect\_field\_name}}\text{=/polls/2/}$$

The LoginRequired Mixin

- When using class-based views
- This mixin should be at the leftmost position in the inheritance list



```python
from django.contrib.auth.mixins import LoginRequiredMixin

class MyView(LoginRequiredMixin, View):
    login_url = '/login/'
    redirect_field_name = 'redirect_to'
```

Limiting Access to Logged-in Users that Pass a Test

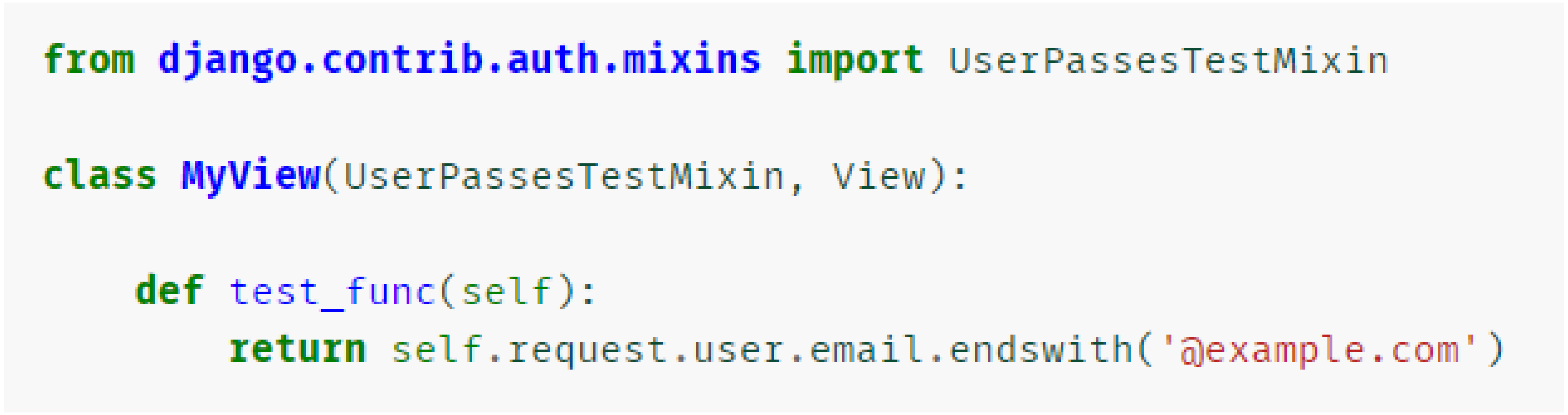
```python
from django.shortcuts import redirect

def my_view(request):
    if not request.user.email.endswith('@example.com'):
        return redirect('/login/?next=%s' % request.path)
    # ...
```

Or use this shortcut:

```python
from django.contrib.auth.decorators import user_passes_test

def email_check(user):
    return user.email.endswith('@example.com')

@user_passes_test(email_check)
def my_view(request):
    ...
```

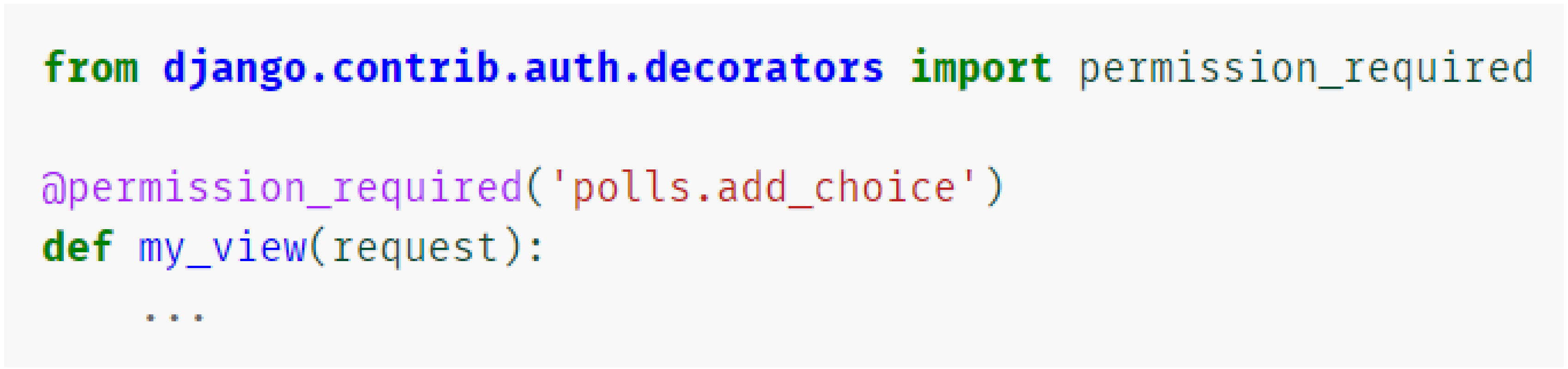
UserPassesTestMixin

- For class-based views:



```python
from django.contrib.auth.mixins import UserPassesTestMixin

class MyView(UserPassesTestMixin, View):

    def test_func(self):
        return self.request.user.email.endswith('@example.com')
```

The permission_required Decorator



```
from django.contrib.auth.decorators import permission_required

@permission_required('polls.add_choice')
def my_view(request):
    ...
```

- May also take an iterable of permissions
- If the raise_exception parameter is given, the decorator will raise PermissionDenied the 403 (HTTP Forbidden) view instead of redirecting to the login page

The PermissionRequiredMixin Mixin

- For class-based views:

```
from django.contrib.auth.mixins import PermissionRequiredMixin

class MyView(PermissionRequiredMixin, View):
    permission_required = 'polls.add_choice'
    # Or multiple of permissions:
    permission_required = ('polls.view_choice',
'polls.change_choice')
```

AccessMixin

- UserPassesTestMixin and PermissionRequiredMixin are subclasses of AccessMixin
- AccessMixin has following things to override:
  - login_url (default: settings.LOGIN_URL)
  - permission_denied_message (default=‘’)
  - redirect_field_name (default: ‘next’)
  - raise_exception (default: True)
  - get_login_url()
  - …

Authentication Views

Using the Views

- The easiest way is to include the provided URLconf in django.contrib.auth.urls in your own URLconf:

```
urlpatterns = [
    path('accounts/', include('django.contrib.auth.urls')),
]
```

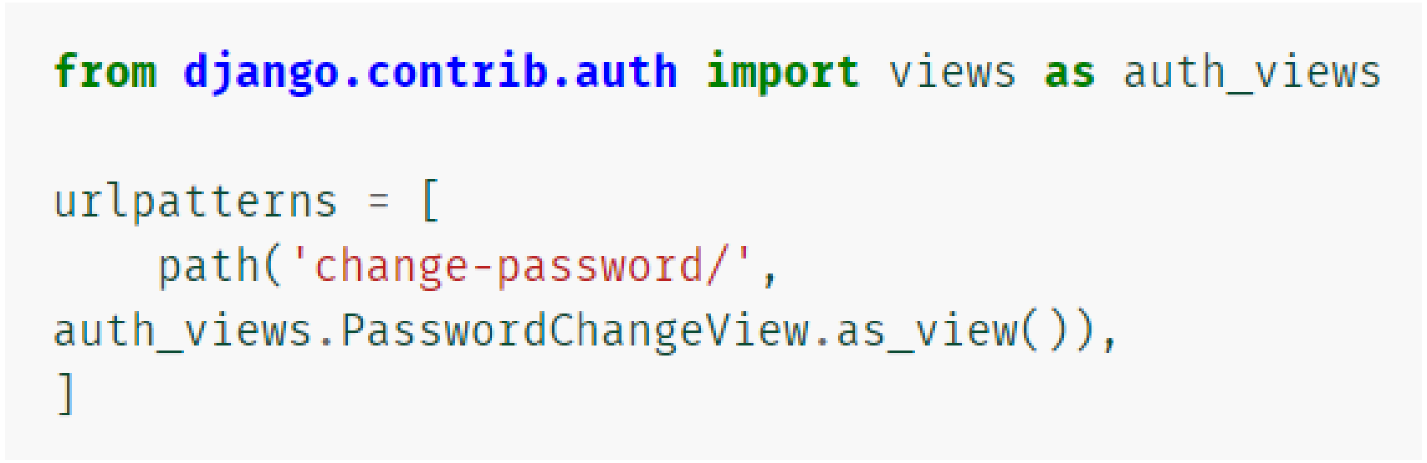
Authentication URL’s

- This will include the following URL patterns::

```
accounts/login/ [name='login']
accounts/logout/ [name='logout']
accounts/password_change/ [name='password_change']
accounts/password_change/done/ [name='password_change_done']
accounts/password_reset/ [name='password_reset']
accounts/password_reset/done/ [name='password_reset_done']
accounts/reset/<uidb64>/<token>/ [name='password_reset_confirm']
accounts/reset/done/ [name='password_reset_complete']
```

Changing URL's

- If you want more control over your URLs:



```
from django.contrib.auth import views as auth_views

urlpatterns = [
    path('change-password/',
auth_views.PasswordChangeView.as_view()),
]
```

Optional Arguments of Views

- For example if you want to change the template name a view uses:

```
urlpatterns = [
    path(
        'change-password/',
        auth_views.PasswordChangeView.as_view(template_name='change-
password.html'),
    ),
]
```

Authentication Views

- All views are class-based:
  - LoginView
  - LogoutView
  - PasswordChangeView
  - PasswordChangeDoneView
  - PasswordResetView
  - PasswordResetDoneView
  - PasswordResetConfirmView
  - PasswordResetCompleteView

LoginView

- Attributes:
  - template_name (default: registration/login.html)
  - redirect_field_name
  - authentication_form (default: AuthenticationForm)
  - extra_context
  - redirect_authenticated_user (default: False)
- If login is successful, the view redirects to the URL specified in next
- If next isn't provided, it redirects to settings.LOGIN_REDIRECT_URL (which defaults to /accounts/profile/)

LoginView – cont.

- It’s your responsibility to provide the html for the login template , called registration/login.html by default
- This template gets passed four template context variables:
  - form
  - next
  - site
  - site_name

LogoutView

- Attributes:
  - next_page (default: settings.LOGOUT_REDIRECT_URL)
  - template_name (default: registration/logged_out.html)
  - redirect_field_name: (default: ‘next’)
  - extra_context
- Template context:
  - title
  - site
  - site_name

A Helper Function

```
redirect_to_login(next, login_url=None, redirect_field_name='next')
```

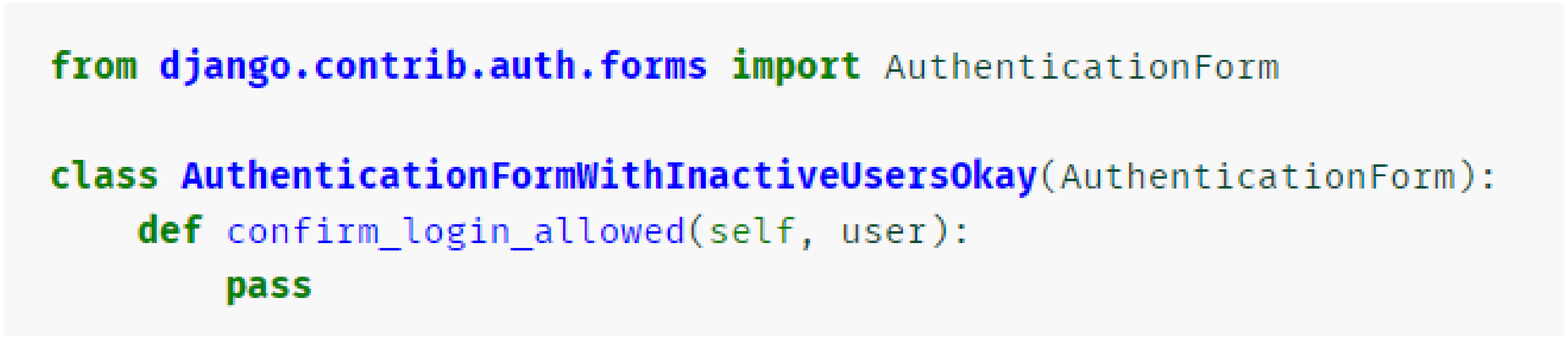
- Redirects to the login page, and then back to another URL after a successful login

Built-in Forms

- You can override each form and use in your views:
  - AdminPasswordChangeForm
  - AuthenticationForm
  - PasswordChangeForm
  - PasswordResetForm
  - SetPasswordForm
  - UserChangeForm
  - UserCreationForm

Change Login Policy

- For example, to allow all users to log in regardless of "active" status:



```python
from django.contrib.auth.forms import AuthenticationForm

class AuthenticationFormWithInactiveUsersOkay(AuthenticationForm):
    def confirm_login_allowed(self, user):
        pass
```

Change Login Policy – cont.

- Or to allow only some active users to log in:

```
class PickyAuthenticationForm(AuthenticationForm):
    def confirm_login_allowed(self, user):
        if not user.is_active:
            raise ValidationError(
                _("This account is inactive."),
                code='inactive',
            )
        if user.username.startswith('b'):
            raise ValidationError(
                _("Sorry, accounts starting with 'b' aren't welcome
here."),
                code='no_b_users',
            )
```

Authentication Data in Templates

```
{% if user.is_authenticated %}
    <p>Welcome, {{ user.username }}. Thanks for logging in.</p>
{% else %}
    <p>Welcome, new user. Please log in.</p>
{% endif %}
```

Permissions in Templates

- To check if the logged-in user has any permissions in the foo app:

```
{% if perms.foo %}
```

- To check if the logged-in user has the permission foo.add_vote:

```
{% if perms.foo.add_vote %}
```

- It is possible to also look permissions up by {% if in %} statements:

```
{% if 'foo' in perms %}
    {% if 'foo.add_vote' in perms %}
        <p>In lookup works, too.</p>
    {% endif %}
{% endif %}
```

Extending the Existing User Model

Extending the Existing User Model

- There are two ways to extend the default User model:
- 1: Add a model (profile model) for additional fields and put a one-to-one to User model
- 2: Extending AbstractUser and overriding the default user model

Profile Model

```python
from django.contrib.auth.models import User

class Employee(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    department = models.CharField(max_length=100)
```

```python
>>> u = User.objects.get(username='fsmith')
>>> freds_department = u.employee.department
```

Adding Profile Model to Admin

```python
from django.contrib import admin
from django.contrib.auth.admin import UserAdmin as BaseUserAdmin
from django.contrib.auth.models import User

from my_user_profile_app.models import Employee

# Define an inline admin descriptor for Employee model
# which acts a bit like a singleton
class EmployeeInline(admin.StackedInline):
    model = Employee
    can_delete = False
    verbose_name_plural = 'employee'

# Define a new User admin
class UserAdmin(BaseUserAdmin):
    inlines = (EmployeeInline,)

# Re-register UserAdmin
admin.site.unregister(User)
admin.site.register(User, UserAdmin)
```

Substituting a Custom User Model

- Some kinds of projects may have authentication requirements for which Django's built-in User model is not always appropriate
- For instance, on some sites it makes more sense to use an email address as your identification token instead of a username
- If you're starting a new project, it's highly recommended to set up a custom user model, even if the default User model is sufficient for you:

```
from django.contrib.auth.models import AbstractUser

class User(AbstractUser):
    pass
```

Substituting a Custom User Model – cont.

- Override the default user model by providing this setting:

```
AUTH_USER_MODEL = 'myapp.User'
```

- Do this before creating any migrations or running manage.py migrate for the first time
- Also, register the model in the app's admin.py:

```
from django.contrib import admin
from django.contrib.auth.admin import UserAdmin
from .models import User

admin.site.register(User, UserAdmin)
```

Substituting a Custom User Model – cont.

- Then reference to settings.AUTH_USER_MODEL everywhere your want to reference to user model:

```
from django.conf import settings
from django.db import models

class Article(models.Model):
    author = models.ForeignKey(
        settings.AUTH_USER_MODEL,
        on_delete=models.CASCADE,
    )
```

References

- https://docs.djangoproject.com/en/3.1/topics/auth/default/
- https://docs.djangoproject.com/en/3.1/topics/auth/customizing/

Any Question?