KAZAKH BRITISH TECHNICAL UNIVERSITY

WEB DEVELOPMENT

Lesson 12

Views

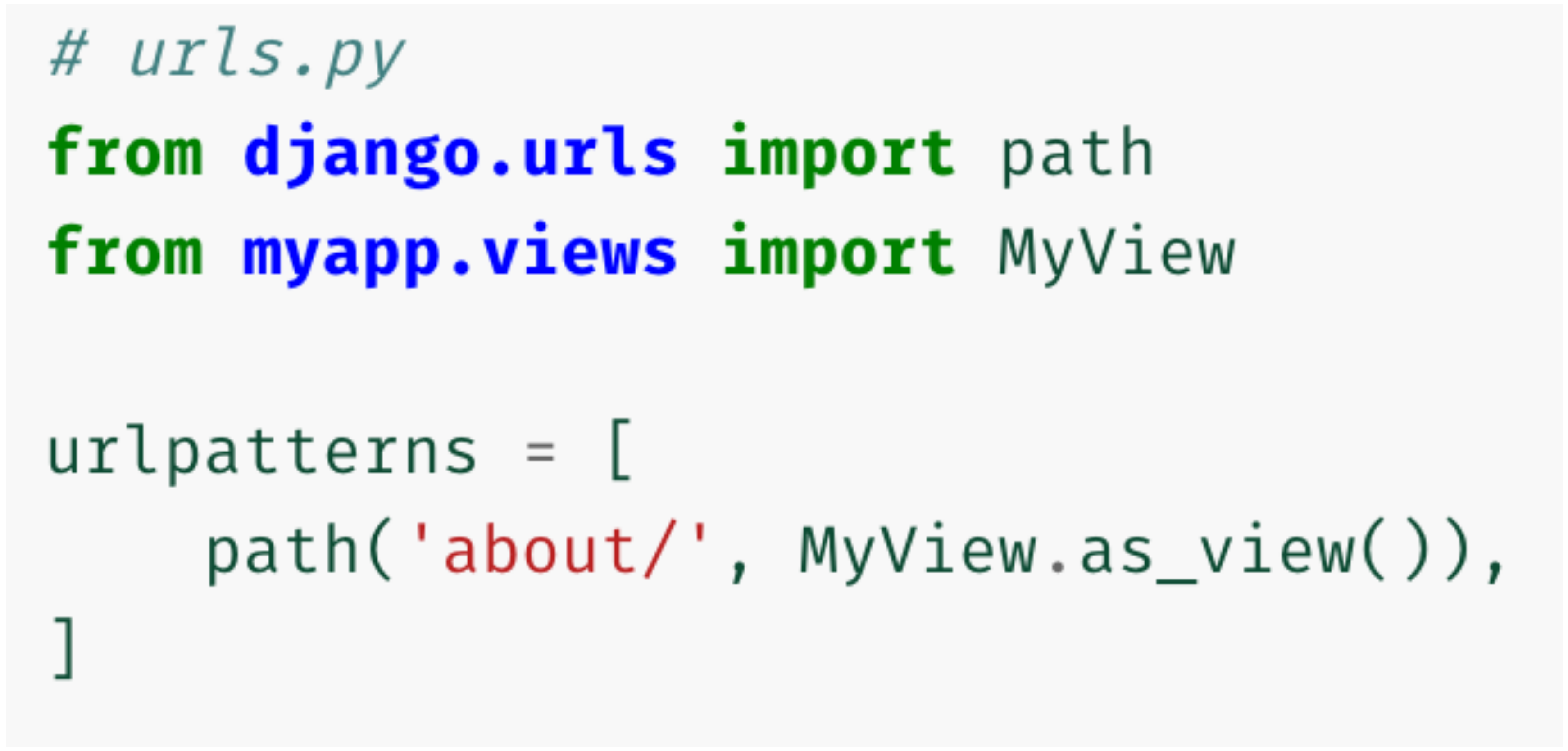
Function Based View

```python
from django.http import HttpResponse

def my_view(request):
    if request.method == 'GET':
        # <view logic>
        return HttpResponse('result')
```

Class Based View

```python
from django.http import HttpResponse
from django.views import View

class MyView(View):
    def get(self, request):
        # <view logic>
        return HttpResponse('result')
```

Class Based View



```python
# urls.py
from django.urls import path
from myapp.views import MyView

urlpatterns = [
    path('about/', MyView.as_view()),
]
```

django REST framework

https://www.django-rest-framework.org

DRF

- DRF leans heavily on object-oriented design and is designed to be easily extensible
- DRF builds directly off of Django CBVs. If you understand CBVs, DRF's design feels like an understandable extension of Django
- The serializer system is extremely powerful, but can be trivially ignored or replaced
- Authentication and Authorization are covered in a powerful, extendable way
- If you really want to use FBVs for your API, DRF has you covered there too

Few notes

- If you're implementing a read-only API, you might only need to implement GET methods.
- If you're implementing a read-write API, you should use the GET, POST, PUT, and DELETE methods.
- Relying on just GET and POST for all actions can be frustrating pattern for API users.
- By definition, GET, PUT, and DELETE are idempotent. POST and PATCH are not.
- PATCH is often not implemented, but it's a good idea to implement it if your API supports PUT requests.
- Django Rest Framework is designed around these methods, understand them and DRF itself becomes easier to understand.

Serialization and Deserialization

Creating a Serializer class

Using ModelSerializers

Questions?