Python JSON

Python has a built in package called json, which is used to work with json data.

dumps(data) – This is used to convert python object into json string.

Example:-

To use json package First we have to import it.

```
import json
python_data = {‘name’: ‘Sonam’, ‘roll’:101 }
json_data = json.dumps(python_data)
print(json_data)
{“name” : “Sonam”, “roll” : 101}
```

Python JSON

loads(data) – This is used to parse json string.

Example:-

```
import json
json_data = {“name” : “Sonam”, “roll” : 101}
parsed_data = json.loads(json_data)
print(parsed_data)
{‘name’ : ‘Sonam’, ‘roll’ : 101}
```

Serializers

In Django REST Framework, serializers are responsible for converting complex data such as querysets and model instances to native Python datatypes (called serialization) that can then be easily rendered into JSON, XML or other content types which is understandable by Front End.

Serializers are also responsible for deserialization which means it allows parsed data to be converted back into complex types, after first validating the incoming data.

- Serialization
- Deserialization

Serializer Class

A serializer class is very similar to a Django Form and ModelForm class, and includes similar validation flags on the various fields, such as required, max_length and default.

DRF provides a Serializer class which gives you a powerful, generic way to control the output of your responses, as well as a ModelSerializer class which provides a useful shortcut for creating serializers that deal with model instances and querysets.

How to Create Serializer Class

- Create a separate seriealizers.py file to write all serializers

```
from rest_framework import serializers
class StudentSerializer(serializers.Serializer):
    name = serializers.CharField(max_length=100)
    roll = serializers.IntegerField()
    city = serializers.CharField(max_length=100)
```

models.py

```
from django.db import models
class Student(models.Model):
   name = models.CharField(max_length=100)
   roll = models.IntegerField()
   city = models.CharField(max_length=100)
```

Run makemigrations and migrate command

ID	NAME	ROLL	CITY
1	Sonam	101	Ranchi
2	Rahul	102	Ranchi
3	Raj	103	Bokaro

JSON Data

ID	NAME	ROLL	CITY
1	Sonam	101	Ranchi
2	Rahul	102	Ranchi
3	Raj	103	Bokaro

Model Object 1

Model Object 2

Model Object 3

Serialization

The process of converting complex data such as querysets and model instances to native Python datatypes are called as Serialization in DRF.

- Creating model instance stu

```
stu = Student.objects.get(id = 1)
```

- Converting model instance stu to Python Dict / Serializing Object

```
serializer = StudentSerializer(stu)
```

Serialization

- Creating Query Set

  stu = Student.objects.all()

- Converting Query Set stu to List of Python Dict / Serializing Query Set

  serializer = StudentSerializer(stu, many=True)

serializer.data

This is the serialized data.

serializer.data

JSONRenderer

This is used to render Serialized data into JSON which is understandable by Front End.

Importing JSONRenderer

```
from rest_framework.renderers import JSONRenderer
```

Render the Data into Json

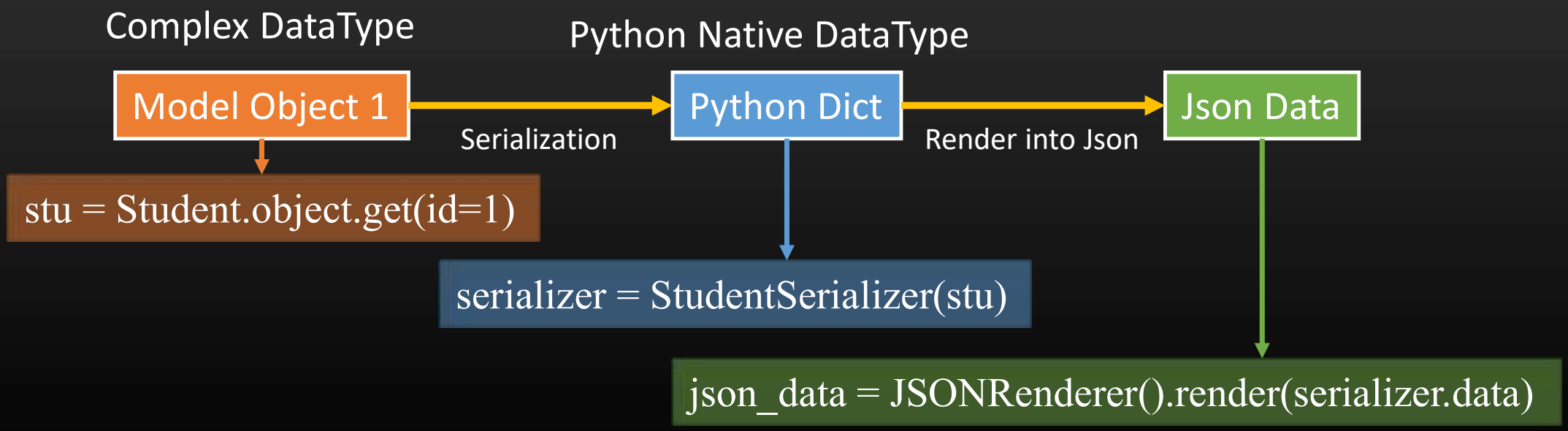
```
json_data = JSONRenderer().render(serializer.data)
```

ID	NAME	ROLL	CITY
1	Sonam	101	Ranchi
2	Rahul	102	Ranchi
3	Raj	103	Bokaro

Model Object 1

Model Object 2

Model Object 3



JsonResponse()

JsonResponse(data, encoder=DjangoJSONEncoder, safe=True, json_dumps_params=None, **kwargs)

An HttpResponse subclass that helps to create a JSON-encoded response. It inherits most behavior from its superclass with a couple differences:

- Its default Content-Type header is set to *application/json*.
- The first parameter, *data*, should be a *dict* instance. If the safe parameter is set to False it can be any JSON-serializable object.
- The encoder, which defaults to django.core.serializers.json.DjangoJSONEncoder, will be used to serialize the data.
- The safe boolean parameter defaults to True. If it's set to False, any object can be passed for serialization (otherwise only dict instances are allowed). If safe is True and a non-dict object is passed as the first argument, a TypeError will be raised.
- The json_dumps_params parameter is a dictionary of keyword arguments to pass to the json.dumps() call used to generate the response.