De-serialization

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.

BytesIo()

A stream implementation using an in-memory bytes buffer. It inherits BufferedIOBase. The buffer is discarded when the close() method is called.

```
import io
stream = io.BytesIO(json_data)
```

JSONParer()

This is used to parse json data to pythonnative data type.

```
from rest_framework.parsers import JSONParser
parsed_data = JSONParser().parse(stream)
```

Deserialization allows parsed data to be converted back into complex types, after first validating the incoming data.

Creating Serializer Object

```
serializer = StudentSerializer(data=parsed_data)
```

Validated Data

```
# Check if data is valid
is_valid = serializer.is_valid()

# Clean data after validation
valid_data = serializer.validated_data

# Validation error messages
errors = serializer.errors
```

Create Data/Insert Data

```
from rest_framework import serializers
from .models import Student

class StudentSerializer(serializers.Serializer):
    id = serializers.IntegerField()
    name = serializers.CharField(max_length=100)
    roll = serializers.IntegerField()
    city = serializers.CharField(max_length=100)
    def create(self, validated_data):
        return Student.objects.create(**validated_data)
```