## 3. Approaches to create Request Body and Parameters - 2

### JSON Library

➔ Used when you need to ensure the payload is strictly formatted as a JSON string before sending it (e.g., interacting with APIs requiring strict JSON payloads)

```
import json, pytest, requests
from dataclasses import dataclass, asdict
BASE_URL = "http://localhost:3000/students"   #Global variable
student_id = None   #Global variable
request_headers = {"Content-Type": "application/json"}
```

**Test to create student using JSON library**

```
def test_createStudentUsingJsonLibrary():
  global student_id
  request_body = {
     "name": "Scott",
     "location": "France",
     "phone": "123456",
     "courses": ["C", "C++"]
  }
  response = requests.post(BASE_URL,data=json.dumps(request_body),headers=request_headers)
  assert response.status_code == 201, "Status code is not 201"
  response_body = response.json()
  assert response_body["name"] == "Scott", "Name is not correct"
  assert response_body["location"] == "France", "Location is not correct"
  assert response_body["phone"] == "123456", "Phone is not correct"
  assert response_body["courses"][0] == "C", "Course 1 should be C"
  assert response_body["courses"][1] == "C++", "Course 2 should be C++"
  student_id = response_body["id"]
  print(response.json())
```

### Custom Python class

➔ Used when data is complex, and you want to encapsulate related properties in a reusable structure (e.g., creating and sending user profile details).

**Test to create student using Python class**

```
def test_createStudentUsingPythonClass():
  class Student:
     def __init__(self, name, location, phone, courses):
        self.name = name
        self.location = location
        self.phone = phone
        self.courses = courses
  student = Student("Scott", "France", "123456", ["C", "C++"])
  global student_id
  request_body = student.__dict__   → You converted the object to a dictionary
  response = requests.post(BASE_URL,json=request_body)
                                          or
  response = requests.post(BASE_URL,data=json.dumps(request_body),headers=request_headers)
  assert response.status_code == 201, "Status code is not 201"
  response_body = response.json()
  assert response_body["name"] == student.name, "Name is not correct"
  assert response_body["location"] == student.location, "Location is not correct"
  assert response_body["phone"] == student.phone, "Phone is not correct"
  assert response_body["courses"] == student.courses, "Courses are not correct"
  student_id = response_body["id"]
  print(response.json())
```

**@dataclass decorator in dataclass Python**

- ➔ Used when data has a fixed structure, and you want to benefit from type hints and easy conversions to a dictionary (e.g., creating structured payloads for APIs like registration or booking systems).

**Test to create student using Dataclass**

```
def test_createStudentUsingDataclass():
  @dataclass
  class Student:
     name: str
     location: str
     phone: str
     courses: list
  student = Student("Scott", "France", "123456", ["C", "C++"])
  global student_id
  request_body = student._ _dict_ _
  response = requests.post(BASE_URL,json=request_body)
                                          or
  response = requests.post(BASE_URL, json=asdict(student))
                                          or
  response = requests.post(BASE_URL,data=json.dumps(request_body),headers=request_headers)
  assert response.status_code == 201, "Status code is not 201"
  response_body = response.json()
  assert response_body["name"] == student.name, "Name is not correct"
  assert response_body["location"] == student.location, "Location is not correct"
  assert response_body["phone"] == student.phone, "Phone is not correct"
  assert response_body["courses"] == student.courses, "Courses are not correct"
  student_id = response_body["id"]
  print(response.json())
```

**Note**

- ➔ **Use @dataclass:** When your class is mainly for storing data.

- **Use custom class:** When your class has complex logic, custom behavior, or needs inheritance.

## External json file

- Used when data is static, predefined, or reusable across multiple requests (e.g., configuration data or bulk data uploads).
- Create **body.json file** in the Package

### body.json

```json
{
    "name": "Scott",
    "location": "France",
    "phone": "123456",
    "courses": [
      "C",
      "C++"
    ]
  }
```

### Test to create student using Externalfile

```python
def test_createStudentUsingExternalfile():
  global student_id
  with open("./body.json", "r") as file:
     request_body = json.load(file)
  response = requests.post(BASE_URL,json=request_body)
                                        or
  response = requests.post(BASE_URL,data=json.dumps(request_body),headers=request_headers)
  assert response.status_code == 201, "Status code is not 201"
  response_body = response.json()
  assert response_body["name"] == "Scott", "Name is not correct"
  assert response_body["location"] == "France", "Location is not correct"
  assert response_body["phone"] == "123456", "Phone is not correct"
  assert response_body["courses"][0] == "C", "Course 1 should be C"
  assert response_body["courses"][1] == "C++", "Course 2 should be C++"
  student_id = response_body["id"]
  print(response.json())
```

## Pytest fixture to delete created record by any of above ways

- After each test, we can also delete the created student record by using the student_id.
- We can specify below fixture in the same module where we specify any of above request codes.

### pytest fixture to delete created record

```python
@pytest.fixture(autouse=True)
def delete_student():
  yield
  if student_id:
     response = requests.delete(f"{BASE_URL}/{student_id}")
```

```
    assert response.status_code == 200
    print("student deleted")
```

**Note**

- ➔ In Python, if you assign a value to a variable inside a function, Python treats it as a local variable by default. To modify a global variable inside a function, you must declare it with the global keyword.

**test_ParametersDemo.py**

```
import requests
def test_path_params():
```

**path parameter**

```
  country = "India"
  response = requests.get(f"https://restcountries.com/v2/name/{country}")
  assert response.status_code == 200
  print(response.json())
def test_query_params():
  headers = {
    "Content-Type": "application/json",
    "x-api-key": "reqres-free-v1"
  }
```

**query parameters**

```
  query_params = {
    "page": 2,
    "id": 5
  }
  response = requests.get("https://reqres.in/api/users", params=query_params, headers=headers)
  assert response.status_code == 200, "Expected status code 200"
  print(response.json())
```