Capstone project (**Generating an API with Python)**

Instructions:

1. Code is provided for the various functionalities. Align them properly to get correct output.

2. Record the screenshot with reflecting the execution of flask server and save them with your folder.
3. Upload all the code files and the screenshots at the git

1. Create a folder with <your_nm>+"api" for eg. "cloudapi".

2. Install python, Flask and Flask-RESTful.

3. Create and initialize the file as api.py

4. Import the following:

- Flask
- Resource, Api, reqparse from flask_restful

```
from flask import Flask
from flask_restful import Resource, Api, reqparse
```

Initialize the API with the following code:

```
app = Flask(#####)
api = Api(app)EMPLOYEES = {}if __name__ == "#######":
  app.run(debug=False)
```

Create mocked data.

```
EMPLOYEES = {
  '1': {'name': 'Mark', 'age': 28, 'spec': 'devops'},
  '2': {'name': 'Jane', 'age': 32, 'spec': 'php'},
  '3': {'name': 'Peter','age':41, 'spec': 'python'},
  '4': {'name': 'Kate', 'age': 27, 'spec': 'sales'},
}
```

Create EmployeesList class and route

```
class EmployeesList(Resource):
  def get(self):
      return EMPLOYEES

  def post(self):
      parser.add_argument("name")
     parser.add_argument("age")
  parser.add_argument("spec")
  args = parser.parse_args()
  employee_id = int(max(EMPLOYEES.keys())) + 1
  employee_id = '%i' % employee_id
  EMPLOYEES[#########] = {
    "name": args[#####],
    "age": args[#####],
    "spec": args[#####],
  }
  return EMPLOYEES[employee_id], 201
```

add a route that will be used as an URL to call the data from this class.

```
api.####_resource(EmployeesList, '/employees/')
```

add the below code before the class:

```
parser = reqparse.RequestParser()
```

Check the code by running the flask server & record the screenshots for Get and post:
1. with browser
2. with POSTMAN

Perform the CRUD operations and test with POSTMAN

create another class and other endpoints.

Define Employee class and route

```
class Employee(Resource):
  def get(self, employee_id):
      if employee_id not in #########:
        return "Not found", 404
```

```
       else:
        return ########[employee_id]
  def put(self, #######):
      parser.add_argument("name")
       parser.add_argument("age")
       parser.add_argument("spec")
       args = parser.parse_args()
       if employee_id not in EMPLOYEES:
         return "Record not found", 404
       else:
         employee = EMPLOYEES[employee_id]
       employee["name"] = args["name"] if args["name"] is not None
else employee ["name"]
    employee ["age"] = args[#####] if args["age"] is not None else
employee ["age"]
    employee ["spec"] = args["spec"] if args[#####] is not None
else employee ["spec"]
    return employee, 200

  def delete(self, employee_id):
      if employee_id not in ########:
        return "Not found", 404
       else:
        del EMPLOYEES[employee_id]
       return '', 204

api.add_resource(Employee, '/employees/<employee_id>')
```

Test all the endpoint with POSTMAN and record the screen shots for the outputs

____