Creating web services was an interest of mine in the early days for fun, but these days creating web services to provide data to clients that perform machine learning algorithms is almost a necessity.

I found two popular REST API Frameworks for Python. Django and Flask.

In PyCharm, we can create a flask project. When created, pycharm creates a python file app.py with the following code:

```
fromflaskimportFlask

app=Flask(__name__)

@app.route('/')
defhello_world():
return'HelloWorld!'

if__name__=='__main__':
app.run()
```

When we run the program, pycharm shows the line below in the Run window:

* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)

When we click on the link http://127.0.0.1:5000, the browser shows the words "Hello World" on a web page. If we change the string 'HelloWorld' with:

```
return'<!doctypehtml><html><head></head><body>HelloThere!</body></html>'
```

, we get a web page with "HelloThere". Note this time, we sent back an HTML formatted string. The browser adhered to the tags. If we send:

```
return"""
'<!doctypehtml>
<htmlLang="en">

<head>
<metacharset="utf-8">
<title>LearningaboutHTML</title>
</head>
<h1>
CreatingwebpageswithHTML
</h1>
<p>
Buildingawebpageisasteptowardsbuildinggreatwebpages.
</p>
<ulHTMLElements>
<li>Head</li>
<li>H1forheader1</li>
<li>Pforparagraph</li>
```

```
</ul>

</html>'

"""
```

, we get something like below. It was copied from the web page:

Creating web pages with HTML

Building a web page is a step towards building great web pages.

- Head
- H1 for header1
- P for paragraph

From <http://127.0.0.1:5000/>

Note that with this tiny application, we created a web service that can display HTML on the client's browser. With REST, we would have to do a lot more to make our web service RESTful. This is not a trivial task. For a web service to be RESTfull, it has to support the POST,GET,PUT, and DELETE methods. It must also provide Resources, Request Verbs, Request Headers, Request Body, Response Body, and Response Status Code. So although the REST architecture has simplifield data exchange, the actual act of exchanging data requires a lot of design and code. Below is a snippet of a "RESTful" web service that returns JSON structure with data from a simple database table. Imagine a web service that produces data from a massive data storage. I a sure there are applications for that. Search continues…

```
fromflaskimportFlask,request

fromflask_restfulimportResource,Api
```

```
fromsqlalchemyimportcreate_engine
fromjsonimportdumps
fromflaskimportjsonify


db_connect=create_engine('sqlite:///chinook/chinook.db')
app=FLask(__name__)
api=Api(app)


classEmployees(Resource):
defget(self):
conn=db_connect.connect()#connecttodatabase
query=conn.execute("select*fromemployees")#Thislineperformsqueryandreturnsjsonre
sult
return{'employees':[i[0]foriinquery.cursor.fetchall()]}#Fetchesfirstcolumnthatis
EmployeeID


classTracks(Resource):
defget(self):
conn=db_connect.connect()
query=conn.execute("selecttrackid,name,composer,unitpricefromtracks;")
result={'data':[dict(zip(tuple(query.keys()),i))foriinquery.cursor]}
returnjsonify(result)
```

```
classEmployees_Name(Resource):

defget(self,employee_id):

conn=db_connect.connect()

query=conn.execute("select*fromemployeeswhereEmployeeId=%d"%int(employee_id))

result={'data':[dict(zip(tuple(query.keys()),i))foriinquery.cursor]}

returnjsonify(result)
```

```
api.add_resource(Employees,'/employees')#Route_1

api.add_resource(Tracks,'/tracks')#Route_2

api.add_resource(Employees_Name,'/employees/<employee_id>')#Route_3
```

```
if__name__=='__main__':

app.run(port='5002')
```

https://www.fullstackpython.com/flask.html

https://pythonprogramming.net/practical-flask-introduction/

Python Flask Tutorial: Full-Featured Web App Part 1 - Getting Started

Python Requests Tutorial: Request Web Pages, Download Images, POST Data, Read JSON, and More

https://www.fullstackpython.com/flask.html