Python Curriculum

Part 03 - Data Containers and Repetitions (3/3)

Classes

```json
[
  {
    "poi": "Yggdrasil",
    "revenue": 790.2,
    "cost": 477.85,
    "visits": 53,
    "unique_visitors": 7
  },
  {
    "unique_visitors": 10,
    "revenue": 1700.65,
    "cost": 1500,
    "visits": 11,
    "poi": "Valhalla"
  },
  {
    "poi": "Asgard",
    "revenue": 3215.75,
    "cost": 2845.79,
    "visits": 265,
    "unique_visitors": 71,
    "poi_details": {
      "open_days": [
        1,
        2,
        3,
        4,
        5
      ],
      "lat": 0,
      "lon": 0,
      "wiki_link": "https://en.wikipedia.org/wiki/Asgard"
    }
  }
]
```

```python
'''norse_type.py'''
import json

class Norse:

    def __init__(self, data):
        self.data = data

    def to_json(self):
        return json.dumps(self.data)

with open('./norse.json', mode='r') as f:
    data = json.load(f)

n = Norse(data)

print(type(n))
print()  # empty new line
print(n.data)
print()
print(n.to_json())
```

```
% python norse_type.py
<class 'norse_type.Norse'>

[{'poi': 'Yggdrasil', 'revenue': 790.2, 'cost': 477.85, 'visits': 53, 'unique_visitors': 7}, {'unique_visitors': 10, 'reve

[{"poi": "Yggdrasil", "revenue": 790.2, "cost": 477.85, "visits": 53, "unique_visitors": 7}, {"unique_visitors": 10, "reve
```

I/O - File system (input)

```
'''norse_type.py'''
import json

def read_from(fname):
    with open(fname, mode='r') as f:
        return json.load(f)

class Norse:
    def __init__(self, data):
        if type(data) is str:
            self.data = read_from(data)
        else:
            self.data = data

    # ...

n = Norse('./norse.json')
# ...
```

I/O - File system (output)

```
'''norse_type.py'''
# ...

def write_to(data, fname):
    with open(fname, mode='w') as f:
        json.dump(data, f, indent=2)

class Norse:
    # ...

    def to_json(self, fname=''):
        if not fname:
            return json.dumps(self.data)

        return write_to(self.data, fname)

n = Norse('./norse.json')
n.to_json('./norse_processed.json')  # output to ./norse_processed.json
```

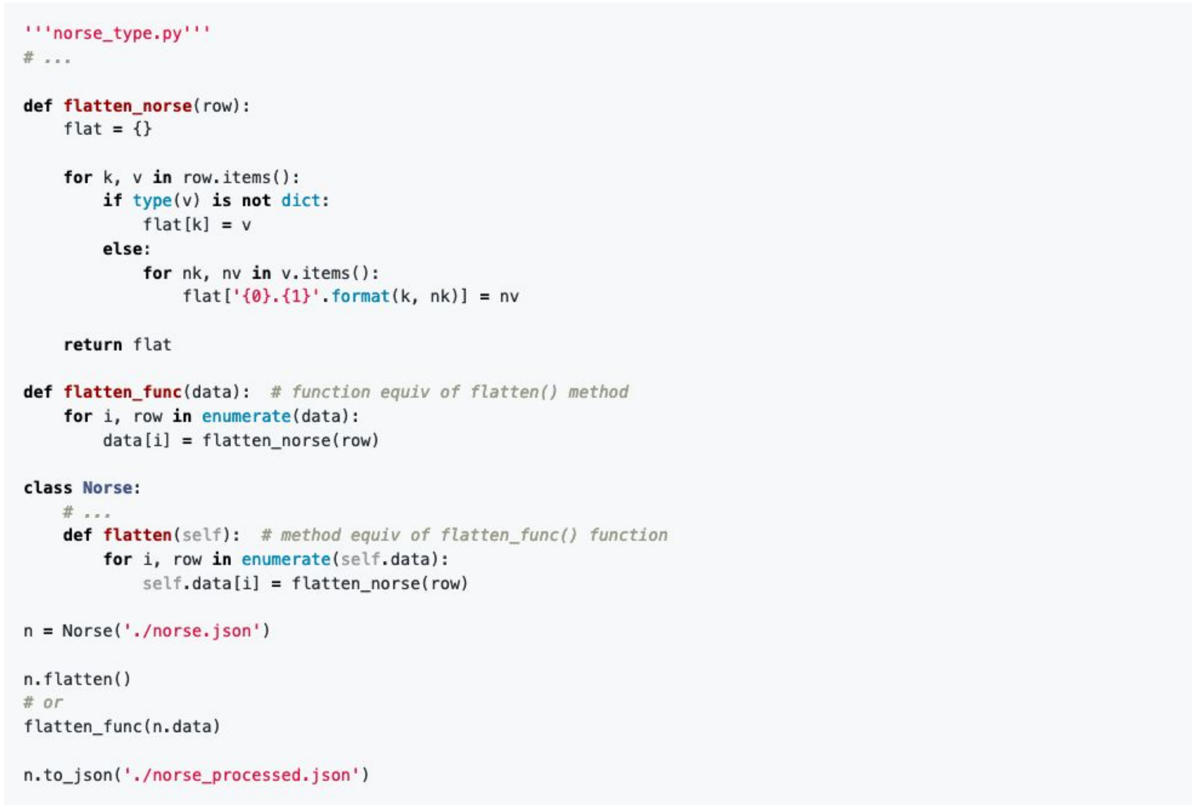
With context manager

```python
def read_from(fname):
    with open(fname, mode='r') as f:
        return json.load(f)
```

```python
def read_from(fname):
    try:
        f = open(fname, mode='r')
        return json.load(f)
    except:
        raise
    finally:
        try:
            f.close()
        except:
            pass
```

Without context manager

Classes or Functions?



```python
'''norse_type.py'''
# ...

def flatten_norse(row):
    flat = {}

    for k, v in row.items():
        if type(v) is not dict:
            flat[k] = v
        else:
            for nk, nv in v.items():
                flat['{0}.{1}'.format(k, nk)] = nv

    return flat

def flatten_func(data):  # function equiv of flatten() method
    for i, row in enumerate(data):
        data[i] = flatten_norse(row)

class Norse:
    # ...
    def flatten(self):  # method equiv of flatten_func() function
        for i, row in enumerate(self.data):
            self.data[i] = flatten_norse(row)

n = Norse('./norse.json')

n.flatten()
# or
flatten_func(n.data)

n.to_json('./norse_processed.json')
```

Statistics

```python
'''norse_type.py'''
# ...
import statistics as stats

STATS_KEYS = ['revenue', 'cost', 'visits', 'unique_visitors']

def transmute_stats(data):
    r = {}
    for key in STATS_KEYS:
        r[key] = [d[key] for d in data if d.get(key)]

    return r

class Norse:
    # ...

    def mean(self, column=''):
        ts = transmute_stats(self.data)
        if column:
            return stats.mean(ts.get(column, []))

        return {k: stats.mean(ts.get(k, [])) for k in STATS_KEYS}
```

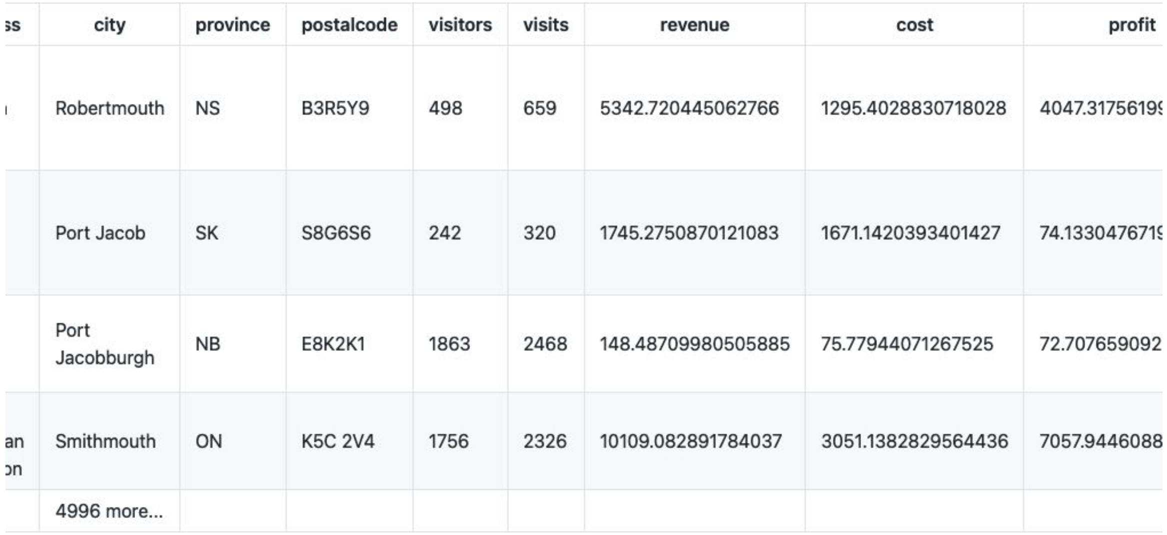
```
>>> from norse_type import Norse
>>> n = Norse('./norse.json')
>>> n.mean('visits')
109.66666666666667
>>> n.mean()
{'revenue': 1902.2, 'cost': 1607.88, 'visits': 109.66666666666667, 'unique_visitors': 29.333333333333332}
```

Pandas

```python
'''norse_pandas.py'''
import pandas as pd

df = pd.read_json('./norse.json')
print('Means:')
print(df.mean())
print('\nMedians:')
print(df.median())
print('\nStandard deviations:')
print(df.std())
```

```
% python norse_pandas.py
Means:
revenue            1902.200000
cost               1607.880000
visits              109.666667
unique_visitors      29.333333
dtype: float64

Medians:
revenue            1700.65
cost               1500.00
visits               53.00
unique_visitors      10.00
dtype: float64

Standard deviations:
revenue            1225.271400
cost               1187.650425
visits              136.151876
unique_visitors      36.115555
dtype: float64
```

```python
'''poi_stats.py'''
import pandas as pd
import requests

data_url = 'https://raw.githubusercontent.com/EQWorks/python-curriculum/03/main/data/poi_stats.json'
with requests.get(data_url) as r:
    data = r.json()

df = pd.DataFrame.from_dict(data)
df['profit'] = df['revenue'] - df['cost']
df.to_csv('./poi_stats.csv')
```



ss	city	province	postalcode	visitors	visits	revenue	cost	profit
ı	Robertmouth	NS	B3R5Y9	498	659	5342.720445062766	1295.4028830718028	4047.3175619
	Port Jacob	SK	S8G6S6	242	320	1745.2750870121083	1671.1420393401427	74.1330476719
	Port Jacobburgh	NB	E8K2K1	1863	2468	148.48709980505885	75.77944071267525	72.707659092
an on	Smithmouth	ON	K5C 2V4	1756	2326	10109.082891784037	3051.1382829564436	7057.9446088
	4996 more...							

```python
# ...
df['profit'] = df['revenue'] - df['cost']
df['profit_margin'] = df['profit'] / df['revenue']
df['avg_revenue'] = df['revenue'] / df['visitors']
df['avg_visits'] = df['visits'] / df['visitors']
df.to_csv('./poi_stats.csv')
```

	cost	profit	profit_margin	avg_revenue	avg_visits
5	1295.4028830718028	4047.3175619909634	0.7575387115249703	10.728354307354952	1.32329317269076З
3	1671.1420393401427	74.13304767196564	0.042476425764422385	7.211880524843423	1.322314049586777
35	75.77944071267525	72.7076590923836	0.4896564023934589	0.07970322050727796	1.324745034889962 5
7	3051.1382829564436	7057.944608827594	0.6981785276054866	5.756880917872459	1.324601366742596 9

Questions?