Exercise Book 2

Covering the materials of Chapters 7-8.
Topics: collection data structures, object oriented programming

In the following 4 lists you will find the country name, capital city name, area (in km^2) and population (in millions) data for 43 European countries respectively.

In [1]:

```
countries = ['Albania', 'Andorra', 'Austria', 'Belgium', 'Bosnia and Herzegovina', 'Bulgaria', 'Czech Republic', 'Denmark', 'United Kingdom', 'Estonia', 'Belarus', 'Finland', 'France', 'Greece', 'Netherlands', 'Croatia', 'Ireland', 'Iceland', 'Kosovo', 'Poland', 'Latvia', 'Liechtenstein', 'Lithuania', 'Luxembourg', 'Macedonia', 'Hungary', 'Malta', 'Moldova', 'Monaco', 'Montenegro', 'Germany', 'Norway', 'Italy', 'Portugal', 'Romania', 'San Marino', 'Spain', 'Switzerland', 'Sweden', 'Serbia', 'Slovakia', 'Slovenia', 'Ukraine']
capitals = ['Tirana', 'Andorra la Vella', 'Vienna', 'Brussels', 'Sarajevo', 'Sofia', 'Prague', 'Copenhagen', 'London', 'Tallin', 'Minsk', 'Helsinki', 'Paris', 'Athens', 'Hague', 'Zagreb', 'Dublin', 'Reykjavik', 'Prishtina', 'Warsaw', 'Riga', 'Vaduz', 'Vilnius', 'luxembourg', 'Skopje', 'Budapest', 'Valletta', 'Chisinau', 'Monaco', 'Podgorica', 'Berlin', 'Oslo', 'Rome', 'Lisbon', 'Bucharest', 'San Marino', 'Madrid', 'Berne', 'Stockholm', 'Belgrade', 'Bratislava', 'Ljubljana', 'Kiev']
areas = [28748, 468, 83857, 30519, 51130, 110912, 78864, 43077, 244100, 45100, 207600, 338145, 543965, 131957, 33933, 56500, 70283, 103000, 10887, 312683, 63700, 160, 65200, 2586, 25713, 93036, 316, 33700, 2, 13812, 357042, 323877, 301277, 92389, 237500, 61, 504782, 41293, 449964, 66577, 49035, 20250, 603700]
populations = [3.2, 0.07, 7.6, 10.0, 4.5, 9.0, 10.4, 5.1, 57.2, 1.6, 10.3, 4.9, 56.2, 10.0, 14.8, 4.7, 3.5, 0.3, 2.2, 37.8, 2.6, 0.03, 3.6, 0.4, 2.1, 10.4, 0.3, 4.4, 0.03, 0.6, 78.6, 4.2, 57.5, 10.5, 23.2, 0.03, 38.8, 6.7, 8.5, 7.2, 5.3, 2.0, 51.8]
```

Let's display the data stored in all lists:

In [2]:

```
print("Countries:")
print(countries)
print("----------")
print("Capitals:")
print(capitals)
print("----------")
print("Areas (in km2):")
print(areas)
print("----------")
print("Populations (in millions):")
print(populations)
```

```
Countries:
['Albania', 'Andorra', 'Austria', 'Belgium', 'Bosnia and Herzegovin
a', 'Bulgaria', 'Czech Republic', 'Denmark', 'United Kingdom', 'Esto
nia', 'Belarus', 'Finland', 'France', 'Greece', 'Netherlands', 'Croa
tia', 'Ireland', 'Iceland', 'Kosovo', 'Poland', 'Latvia', 'Liechtens
tein', 'Lithuania', 'Luxembourg', 'Macedonia', 'Hungary', 'Malta',
'Moldova', 'Monaco', 'Montenegro', 'Germany', 'Norway', 'Italy', 'Po
rtugal', 'Romania', 'San Marino', 'Spain', 'Switzerland', 'Sweden',
'Serbia', 'Slovakia', 'Slovenia', 'Ukraine']
----------
Capitals:
['Tirana', 'Andorra la Vella', 'Vienna', 'Brussels', 'Sarajevo', 'So
fia', 'Prague', 'Copenhagen', 'London', 'Tallin', 'Minsk', 'Helsink
i', 'Paris', 'Athens', 'Hague', 'Zagreb', 'Dublin', 'Reykjavik', 'Pr
ishtina', 'Warsaw', 'Riga', 'Vaduz', 'Vilnius', 'luxembourg', 'Skopj
e', 'Budapest', 'Valletta', 'Chisinau', 'Monaco', 'Podgorica', 'Berl
in', 'Oslo', 'Rome', 'Lisbon', 'Bucharest', 'San Marino', 'Madrid',
'Berne', 'Stockholm', 'Belgrade', 'Bratislava', 'Ljubljana', 'Kiev']
----------
Areas (in km2):
[28748, 468, 83857, 30519, 51130, 110912, 78864, 43077, 244100, 4510
0, 207600, 338145, 543965, 131957, 33933, 56500, 70283, 103000, 1088
7, 312683, 63700, 160, 65200, 2586, 25713, 93036, 316, 33700, 2, 138
12, 357042, 323877, 301277, 92389, 237500, 61, 504782, 41293, 44996
4, 66577, 49035, 20250, 603700]
----------
Populations (in millions):
[3.2, 0.07, 7.6, 10.0, 4.5, 9.0, 10.4, 5.1, 57.2, 1.6, 10.3, 4.9, 5
6.2, 10.0, 14.8, 4.7, 3.5, 0.3, 2.2, 37.8, 2.6, 0.03, 3.6, 0.4, 2.1,
10.4, 0.3, 4.4, 0.03, 0.6, 78.6, 4.2, 57.5, 10.5, 23.2, 0.03, 38.8,
6.7, 8.5, 7.2, 5.3, 2.0, 51.8]
```

The index position of the elements in the lists ties the information for each country together:

In [3]:

```
for idx in range(len(countries)):
    print("Name: %s, Capital: %s, Area: %d km2, Population: %.2f millions" % (co
untries[idx], capitals[idx], areas[idx], populations[idx]))
```

```
Name: Albania, Capital: Tirana, Area: 28748 km2, Population: 3.20 mi
llions
Name: Andorra, Capital: Andorra la Vella, Area: 468 km2, Population:
0.07 millions
Name: Austria, Capital: Vienna, Area: 83857 km2, Population: 7.60 mi
llions
...
Name: Slovakia, Capital: Bratislava, Area: 49035 km2, Population: 5.
30 millions
Name: Slovenia, Capital: Ljubljana, Area: 20250 km2, Population: 2.0
0 millions
Name: Ukraine, Capital: Kiev, Area: 603700 km2, Population: 51.80 mi
llions
```

Task 1: List of dictionaries

Storing the data in 4 separate lists is not comfortable. Construct a list of dictionaries programatically:

- each item in the list is a dictionary;
- each dictionary contains the relevant information for a single country.

The result should be like the following:

```
[
    {
        'country': 'Albania',
        'capital': 'Tirana',
        'area': 28748,
        'population': 3.2
    },

    ...

    {
        'country': 'Ukraine',
        'capital': 'Kiev',
        'area': 603700,
        'population': 51.8
    }
]
```

In [4]:

```
dataset = []
for idx in range(len(countries)):
    dataset.append({
        'country': countries[idx],
        'capital': capitals[idx],
        'area': areas[idx],
        'population': populations[idx]
    })
print(dataset)
```

```
[{'country': 'Albania', 'capital': 'Tirana', 'area': 28748, 'populat
ion': 3.2}, {'country': 'Andorra', 'capital': 'Andorra la Vella', 'a
rea': 468, 'population': 0.07}, {'country': 'Austria', 'capital': 'V
ienna', 'area': 83857, 'population': 7.6},
...,
{'country': 'Slovakia', 'capital': 'Bratislava', 'area': 49035, 'pop
ulation': 5.3}, {'country': 'Slovenia', 'capital': 'Ljubljana', 'are
a': 20250, 'population': 2.0}, {'country': 'Ukraine', 'capital': 'Ki
ev', 'area': 603700, 'population': 51.8}]
```

Task 2: Population density

Calculate the population density for each country (in people / km^2 unit) and extends each country with this information.

The result should be like the following:

```
[
    {
        'country': 'Albania',
        'capital': 'Tirana',
        'area': 28748,
        'population': 3.2,
        'density': 111.31209127591485
    },

    ...

    {
        'country': 'Ukraine',
        'capital': 'Kiev',
        'area': 603700,
        'population': 51.8,
        'density': 85.80420738777539
    }
]
```

In [5]:

```
for item in dataset:
    item['density'] = item['population'] * 1e6 / item['area']
print(dataset)
```

```
[{'country': 'Albania', 'capital': 'Tirana', 'area': 28748, 'populat
ion': 3.2, 'density': 111.31209127591485}, {'country': 'Andorra', 'c
apital': 'Andorra la Vella', 'area': 468, 'population': 0.07, 'densi
ty': 149.57264957264957}, {'country': 'Austria', 'capital': 'Vienn
a', 'area': 83857, 'population': 7.6, 'density': 90.63047807577185},

...,
{'country': 'Slovakia', 'capital': 'Bratislava', 'area': 49035, 'pop
ulation': 5.3, 'density': 108.08606097685326}, {'country': 'Sloveni
a', 'capital': 'Ljubljana', 'area': 20250, 'population': 2.0, 'densi
ty': 98.76543209876543}, {'country': 'Ukraine', 'capital': 'Kiev',
'area': 603700, 'population': 51.8, 'density': 85.80420738777539}]
```

Task 3: Highest density

Find the country with the highest population density.

In [6]:

```
max_idx = 0

for idx in range(1, len(dataset)):
    if dataset[idx]['density'] > dataset[max_idx]['density']:
        max_idx = idx

print(dataset[max_idx])
```

```
{'country': 'Monaco', 'capital': 'Monaco', 'area': 2, 'population':
0.03, 'density': 15000.0}
```

Task 4: Object oriented approach

Task A): define a class named `Country`, which can store a country's name, capitaly city, area and population. Construct a list of *objects*, where each object is an instance of the `Country` class.

Task B): add a `density()` method to the `Country` class, which calculates the population density for that country dynamically. Find the country with the highest population density.

In [7]:

```
class Country():
    def __init__(self, name, capital, area, population):
        self.name = name
        self.capital = capital
        self.area = area
        self.population = population

    def density(self):
        return self.population * 1e6 / self.area

dataset2 = []
for idx in range(len(countries)):
    dataset2.append(Country(countries[idx], capitals[idx], areas[idx], populations[idx]))

max_idx = 0
for idx in range(1, len(dataset2)):
    if dataset2[idx].density() > dataset2[max_idx].density():
        max_idx = idx

print(dataset2[max_idx].name)
```

```
Monaco
```