Exercise set 7
Working with complex data structures
(**nested collections, internet data**)

1. Create the following dictionary in your application:

```
cafe = {
    "name": "Imaginary Cafe Ltd.",
    "website": "https://edu.frostbit.fi/sites/cafe/en",
    "categories": [
        "cafe",
        "tea",
        "lunch",
        "breakfast"
    ],
    "location": {
        "city": "Rovaniemi",
        "address": "Test address 22",
        "zip_code": "FI-96100"
    }
}
```

Print the contents of the dictionary as shown in the example below.

Example of the application running:

```
Imaginary Cafe Ltd.
Test address 22
FI-96100 Rovaniemi

https://edu.frostbit.fi/sites/cafe/en
Services: cafe, tea, lunch, breakfast
```

Filename of the exercise = ***exercise7_1.py***

Typical code amount : ***15-28 lines*** (empty lines/comments not included)

Tip:

Print the categories either by using a loop or using the join-function! All other information in the dictionary can be printed without a loop/join.

Don’t print the whole services list just by using ~~print(cafe['categories'])~~!

Check out Repetition statements and collections materials on how to handle dictionaries and lists, pages 66 - 83!

Also check out the guide "How to work with complex data collections and internet data"!

Note: The web address of the cafe gets underlined automatically when you run the application!

2. Create an application that contains a list called ***inventory***, which consists of other lists (**fruits**, **berries** and **vegetables**):

fruits: ***list 1***

- apple
- pear
- banana

berries: ***list 2***

- strawberry
- blueberry
- blackberry

vegetables: ***list 3***

- carrot
- kale
- cucumber

Print the contents of the **inventory** list by using a loop, that has another loop inside of it.

Note: Don't make three separate loops for each original list (fruits, berries, vegetables)! Instead, create just one loop that processes only the **inventory –**list, and have another loop inside that one!

Tips:
Firstly, create the three separate lists, and add the words described above into them. Then create another new list, called **inventory,** which contains the first lists as elements, for example:

inventory = [fruits, berries, vegetables]

After this, use a loop, that contains another loop! Check out Repetition statements –materials, page 72-74.

Example of the application running:

```
apple
pear
banana
strawberry
blueberry
blackberry
carrot
kale
cucumber
```

Filename of the exercise = ***exercise7_2.py***

Typical code amount : ***7-10 lines*** (empty lines/comments not included)

3. Copy the following list of dictionaries in your application:

```
shopcart = [
    {"name": "Beehive - lamp", "price": 999.9},
    {"name": "Malm - bed", "price": 169.9},
    {"name": "Moomin - mug set", "price": 59.9},
    {"name": "Nemo - divan", "price": 699.9},
    {"name": "Ritz - armchair", "price": 369.9}
]
```

Print the receipt of all purchases in a **loop** as well as the total price according to the example below.

Note: Calculate the sum in the loop, don't just print the number yourself!

Example of the application running:

```
Receipt:
 - Beehive - lamp
 - Malm - bed
 - Moomin - mug set
 - Nemo - divan
 - Ritz - armchair

Total sum: 2299.5 €.
Please come again!
```

Filename of the exercise = ***exercise7_3.py***

Typical code amount : ***14-22 lines*** (empty lines/comments not included)

Extra task: Calculate the amount of VAT (value added tax) of the total purchase as well (24%).

Note: the prices already include the VAT!

Tip: The VAT **is not calculated** by multiplying the price by 0.76. That would only mean we're calculating 76% of the price including VAT. ☺

The **VAT is also not calculated** by removing ***0.24 * total_price*** from the total, because it's the same thing as multiplying by 0.76 ☺

Tip: Think about the equation how VAT is added to a price without VAT. Then solve the equation, and you should find the formula on how you to get VAT out of a price that already has VAT.

Example:

If price without VAT is 100 €, the price with VAT would be 124 €. This means, the actual VAT is 24€.

Therefore, if you do this:
124 € * 0.24 = **29.76 €** , it is not correct.

4. Create an application that contains a **list** of **dictionaries:**

    - **0**:
        - name: ”Casablanca”
        - year: 1942
    - **1**:
        - name: ”Forrest Gump”
        - year: 1994
    - **2**:
        - name: ”Avatar”
        - year: 2009

    etc.

    Add your favorite movies into the collection until you have at least **6 movies in total**. (if you don't really watch movies that much, you can also use books or other media released on a certain year).

    After your **list of 6 movies or more** is completed, use a loop to go through all the items. Divide the movies into two lists: movies that were released before the year 2000, and movies released on year 2000 or later. Use two separate lists to store movies for each time period.

    Check out the example in Repetition statements –materials page 90!

    When you have divided all the movies into two lists, first print out the new movies and then the older movies according to the example below. You can use either two additional for-loops to do this or the join() –function to achieve this result.

Example of the application running:

```
These movies have been released in year 2000 or later:
Avatar

These movies have been released before the year 2000:
Casablanca, Forrest Gump
```

Filename of the exercise = ***exercise7_4.py***

Typical code amount : ***35-55 lines*** (empty lines/comments not included)

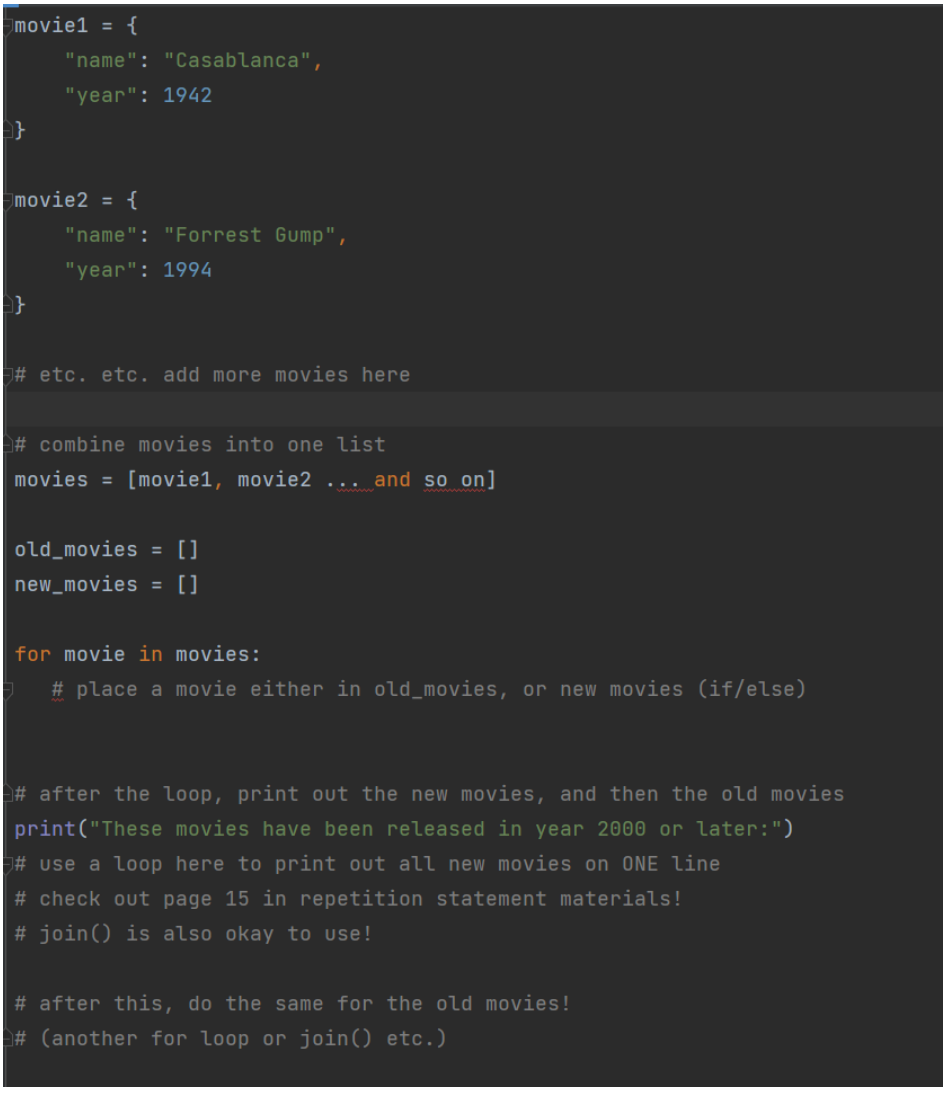
Tips: Create a separate list for both time periods (e.g. new_movies = [] and old_movies = [])

After this, loop through all the movies, and check the year with an if-statement. Place the movie in one of the new lists (new_movies or old_movies) based on the release year of the movie.

for example: if the movie is before the year 2000:
old_movies.append(movie_name)

In the end of the application, you can loop through each list (new_movies and old_movies) either by using two additional loops or other methods.

Example (not actual working code, just a sketch):



```
movie1 = {
    "name": "Casablanca",
    "year": 1942
}

movie2 = {
    "name": "Forrest Gump",
    "year": 1994
}

# etc. etc. add more movies here

# combine movies into one list
movies = [movie1, movie2 ... and so on]

old_movies = []
new_movies = []

for movie in movies:
    # place a movie either in old_movies, or new movies (if/else)


# after the loop, print out the new movies, and then the old movies
print("These movies have been released in year 2000 or later:")
# use a loop here to print out all new movies on ONE line
# check out page 15 in repetition statement materials!
# join() is also okay to use!

# after this, do the same for the old movies!
# (another for loop or join() etc.)
```

5. Create an application that downloads the current weather data from the internet, and informs the user which city in Finland has the strongest and weakest wind at the moment.

   The data can be downloaded here (code example further below):
   https://edu.frostbit.fi/api/weather/

   Note: the data updates daily approximately at 9:00 and 21:00, and it contains the average values from last 12 hours.

   The data in the address consists of a **list** of **dictionaries**.

   Example of a single item of data:

```
{
    "location": "Inari",
    "snow": 2.4,
    "rain": 0.3,
    "wind": 2.1,
    "area": "lapland"
}
```

   Data field explanations:

   snow = average snow level (height), in centimeters (cm)

   rain = average amount of rain (mm)
   - ***0.3 - 0.9 mm*** = slightly raining
   - ***1 - 4.4 mm*** = raining
   - ***over 4.5 mm*** = pouring

   wind = average wind speed, meters per second (m/s)

   area = the area of observation, either ***lapland, middle*** or ***south.***

You can begin with this code. In this example, the data from the internet resides in the "**weather" –**variable:

```
import json
import urllib.request
url = "https://edu.frostbit.fi/api/weather/"
req = urllib.request.Request(url)
raw_data = urllib.request.urlopen(req).read().decode("UTF-8")
weather = json.loads(raw_data)
```

Example of the application running (actual results depend on the time and day):

```
Strongest wind today at location: Helsinki, 6.6 m/s
Weakest wind today at location: Inari, 1.6 m/s
```

Tip: The weakest wind might be a bit tricky. If your starting value of weakest wind is **0**, you can't just compare to it with a simple if-statement, because no wind will be less than 0. ☺

Instead you might need an if-statement something more like this:
if wind < weakest_wind or weakest_wind == 0:
=> set new weakest wind value

Why this works? If weakest_wind is exactly 0, it means it doesn't have any starting value, and this will place the first value as the current weakest wind. After this, the comparison wind < weakest_wind works fine. 😊

Extra task: print also the average wind speed of every area of observation. Round the result into one decimal. Example of the outcome:

```
Average wind, Lapland: 2.0 m/s
Average wind, Middle part of Finland: 3.6 m/s
Average wind, Southern Finland: 5.1 m/s
```

Filename of the exercise = ***exercise7_5.py***

Typical code amount : ***14-32 lines*** (empty lines/comments not included)

Extra exercises!

Note: A good grade does not necessarily require doing all extra exercises! You can choose exercises that interest you most!

6. Create an application that has the following collection of multiple restaurants (at least 5 restaurants or more):

   restaurants
   - **0:**
     - **name: ”North Delish”**
     - **rating: 4.5**
     - **reservations: True**
     - **services:**
       - **”lunch”**
       - **”dinner”**
     - **price_level : 5**
     - **location: "Rovaniemi"**
   - **1:**
     - **name: ”Food Galore”**
     - **rating: 3.8**
     - **reservations: False**
     - **services:**
       - **”breakfast”**
       - **”lunch”**
     - **price_level : 3**
     - **location: "London"**
   - **2:**
     - **name: ”Snacksy Ltd”**
     - **rating: 3.2**
     - **reservations: False**
     - **services:**
       - **”lunch”**
       - **”dinner”**
       - **"night"**
     - **price_level : 2**
     - **location: "Berlin"**

   etc. (you can use your imagination here!)

   Add also other restaurants, either real or imaginary ones. You can use imaginary ratings etc. in this exercise.

Finally, ask the user a series of questions of the type of restaurant the user is looking for (numeric and yes/no –questions).

For example:

Welcome to restaurant search!
Which star rating at least for the restaurant? (1-5)
What is the maximum price level you're looking for? (1-5)
Would you like to make a reservation before hand? (y/n)
In what time would you like to arrive? (0 – 23)

After the application has asked all the questions, show a list of restaurants that meet all the criteria. If none of the restaurants match, print
"No matching restaurants found, unfortunately!"

Use the services –data by using this logic:

```
# breakfast = 6-10
# lunch = 11-16
# dinner = 17-24
# night = 0-5
```

For example, if the user inputs "13", the **services** of the restaurant has to include "**lunch**".

Filename of the exercise = ***exercise7_6.py***

Typical code amount : varies, approximately ***50-120 lines*** (empty lines/comments not included)

7. Create an application that determines which city in Finland has statistically the most warnings of slippery weather conditions.

   You can download the slippery condition warning from the following internet-API:

   https://liukastumisvaroitus-api.beze.io/api/v1/warnings

   Print also the latest 5 slippery weather warnings based on the timestamp (city + date + time).

Filename of the exercise = ***exercise7_7.py***

Typical code amount : ***15-30 lines*** (empty lines/comments not included)

8. Create an application that asks the user for a year between 2017 and 2021. Based on the year, calculate the average queue time in the city of Oulu on that particular year.

   The data can be found here:

   https://api.ouka.fi/v1/chc_waiting_times_monthly_stats?order=year.desc,month.desc

   Note:

   Ignore all data that is missing the timestamp (null).

   The total queue time in this dataset is the average of ***doctor_queue*** and ***nurse_queue***.

   Also: the **chc**-field contains either the area or the name of the service in Finnish. This field is not needed when calculating the queue-times!

Filename of the exercise = ***exercise7_8.py***

Typical code amount : ***12-30 lines*** (empty lines/comments not included)