Viikkotehtävät 7
Monimutkaisten tietorakenteiden käsittely
(**sisennetyt kokoelmat , internet-data**)

1. Tee ohjelmasi koodiin seuraavanlainen dictionary:

```
cafe = {
    "name": "Imaginary Cafe Oy",
    "website": "https://edu.frostbit.fi/sites/cafe",
    "categories": [
        "cafe",
        "tea",
        "lunch",
        "breakfast"
    ],
    "location": {
        "city": "Rovaniemi",
        "address": "Testikuja 22",
        "zip_code": "96200"
    }
}
```

Tulosta dictionaryn sisältö alla olevan esimerkin mukaisesti.
HUOM! Noudata kirjoitusasuja ***täsmällisesti!***

Esimerkki ohjelman toiminnasta:

```
Imaginary Cafe Oy
Testikuja 22
96200 Rovaniemi

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

Tehtävän tiedostonimi = ***exercise7_1.py***

Tyypillinen koodimäärä***: 15-28 riviä*** *(tyhjiä rivejä ja kommentteja ei lasketa)*

Vinkki:
Tulosta kategoriat silmukassa tai käyttämällä join-funktiota! **Kaikki muu tieto voidaan tulostaa dictionarysta ilman silmukkaa** tai join() – funktiota (ks. materiaalit).

Huom: Älä tulosta koko services-listaa sellaisenaan! (eli ei tällä tavalla: ~~**print(cafe['services'])**~~

Huom myös: dictionaryä varten ei tarvitse tehdä silmukkaa tulostamista varten (vaikka dictionary on toisen dictionaryn sisällä). Ks. materiaalit sivu 85!

Ks. materiaalit kuinka kokoelmia, listoja ja dictionaryjä käytetään, sivut 71-90!

Ks. lisäksi materiaali Osa 3.5: ”Monimutkaiset tietorakenteet ja niiden hyödyntäminen”

Huom: Kahvilan web-osoite alleviivautuu PyCharmissa automaattisesti kun ajat ohjelman!

2. Tee ohjelma, jossa on lista nimeltä ***inventory***, joka koostuu muista listoista (**fruits**, **berries** ja **vegetables**):

fruits: ***lista 1***

- apple
- pear
- banana

berries: ***lista 2***

- strawberry
- blueberry
- blackberry

vegetables: ***lista 3***

- carrot
- kale
- cucumber

Tulosta **inventory**-listan sisältö allekkain käyttäen silmukkaa, **jonka sisällä on toinen silmukka.**

Huom: Älä tee kolmea erillistä for-silmukkaa jokaiselle alkuperäiselle listalle (fruits, berries, vegetables); **sen sijaan** tee vain yksi silmukka, jonka sisällä on toinen silmukka. Käsittele silmukassa ainoastaan **inventory-**listaa, älä muita listoja.

Vinkki:
Tee ensi kolme erillistä listaa, joiden sisälle laitetaan tehtävänannon mukaiset sanat. Tee lopuksi uusi lista **inventory**, joka sisältää aiemmat listat:

inventory = [fruits, berries, vegetables]

Tämän jälkeen käytä silmukkaa, jonka sisällä on toinen silmukka!
Ks. Toistolauseet -materiaali, sivu 80-82.

Esimerkki ohjelman toiminnasta:

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

Tehtävän tiedostonimi = ***exercise7_2.py***

Tyypillinen koodimäärä***: 7-10 riviä*** *(tyhjiä rivejä ja kommentteja ei lasketa)*

3. Luo ohjelmaasi seuraavanlainen lista:

```
shopcart = [
    {"name": "Lokki-valaisin", "price": 349.9},
    {"name": "Stockholm-matto", "price": 129.9},
    {"name": "Malm-lipasto", "price": 49.9},
    {"name": "Vienna-divaanisohva", "price": 799.9},
    {"name": "Ritz-nojatuoli", "price": 369.9}
]
```

Tulosta ostosten kuitti **silmukassa** kokonaishinnan kanssa alla olevan esimerkin mukaisesti.

Huom: Laske tuotteiden summa silmukassa, älä tulosta summaa itse sellaisenaan!

Huom: voit käsitellä tuotteiden nimet ja hinnat yhdessä ja samassa silmukassa, eli ei tarvitse kahta erillistä for-silmukkaa!

Esimerkki ohjelman toiminnasta:

```
Kuitti ostoksista:
 - Lokki-valaisin
 - Stockholm-matto
 - Malm-lipasto
 - Vienna-divaanisohva
 - Ritz-nojatuoli

Yhteensä 1699.5 €.
Tervetuloa uudelleen!
```

Tehtävän tiedostonimi = ***exercise7_3.py***

Tyypillinen koodimäärä***: 14-22 riviä*** *(tyhjiä rivejä ja kommentteja ei lasketa)*

Lisätehtävä: Laske kuittiin hinnasta myös ALV –osuus (24%) euroina.

Huom: Tuotteiden hinnat sisältävät jo ALV:n.

Vinkki:
oikea ratkaisu tässä tapauksessa **ei ole** kertoa summa 0.76:lla. Se tarkoittaa silloin vain sitä, että otetaan 76% ALVillisesta hinnasta! ☺

ALVia **ei myöskään lasketa** vähentämällä ***0.24 * total_price*** kokonaishinnasta, koska se on sama asia kuin kertoisi summan 0.76:lla ☺

Vinkki:
Mieti yhtälöä / kaavaa, jolla ALV lisätään sellaiseen hintaan, jossa ei vielä ole ALVia. Ratkaise tämän jälkeen tämä yhtälö siten, että miten kokonaishinnasta saadaan ALV pois.

Esimerkki:

Jos hinta ilman ALVia on 100 €, silloin hinta ALV:n kanssa on 124 €. Tämä tarkoittaa sitä, että varsinainen ALV on tässä tapauksessa 24€.

Joten, jos lasketaan näin:
124 € * 0.24 = **29,76€**, mikä ei ole oikea ALVin määrä!

Lisävinkki ALV:iin: Kertolaskun sijaan, pyri löytämään jakolaskua hyödyntävä ratkaisu!

4. Tee ohjelma, jossa on seuraavanlainen **lista**, joka koostuu **dictionaryistä** (**dict**):

   - **0**:
     - name: ”Komisario Palmun erehdys”
     - year: 1960
   - **1**:
     - name: ”Kauas pilvet karkaavat”
     - year: 1996
   - **2**:
     - name: ”Mies vailla menneisyyttä”
     - year: 2002

   jne.

   Lisää kokoelmaan vaikkapa suosikkielokuviasi siten, että lopulta listassa on **vähintään 6 elokuvaa**! (Mikäli et juuri katso elokuvia, voit käyttää myös esimerkiksi kirjoja tai muita julkaisuvuoteen sidottuja teoksia.)

   Kun **elokuvalista** on valmis, käy se läpi silmukalla. Jaottele elokuvat kahteen uuteen eri listaan: 2000-luvun elokuviin, sekä sitä vanhempiin elokuviin. Toisin sanoen, käytä kahta erillistä listaa, jotka pitävät kirjaa vanhoista ja uusista elokuvista.

   Tätä tehtävää varten ks. Toistolauseet –materiaali, sivu 100.

   Kun koodisi on jakanut elokuvat kahteen listaa, tulosta näiden kahden listan sisältö alla olevan esimerkin mukaisesti. Voit tehdä tämän joko tekemällä kaksi uutta for-silmukkaa, tai hyödyntää join() -funktiota lopputuloksen saavuttamiseksi.

Esimerkki ohjelman toiminnasta:

```
Seuraavat elokuvat on julkaistu 2000-luvulla:
Mies vailla menneisyyttä

Seuraavat elokuvat on julkaistu ennen vuotta 2000:
Komisario Palmun erehdys, Kauas pilvet karkaavat
```

Tehtävän tiedostonimi = ***exercise7_4.py***

Tyypillinen koodimäärä***: 35-55 riviä*** *(tyhjiä rivejä ja kommentteja ei lasketa)*

Vinkkejä: tee molempia aikakausia varten (vanhat elokuvat ja uudet elokuvat) oma listansa, esim. new_movies = [] ja old_movies = [].

Tämän jälkeen käy elokuvat silmukassa läpi, ja tarkista vuosiluku if-lauseella. Aseta elokuva jompaankumpaan listaan vuosiluvun perusteella (joko new_movies tai old_movies).

esim. jos elokuva on vanhempi kuin 2000-luku:
old_movies.append(movie_name)

Ohjelman lopuksi voit tulostaa molemman listan sisällöt vuorotellen sopivien otsikoiden alle. (joko tekemällä kaksi peräkkäistä erillistä for-silmukkaa tai hyödyntämällä muita menetelmiä)

Esimerkki (ei toimiva koodi, vain hahmotelma)

```
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. Tee ohjelma, joka lataa ajan tasalla olevat säätiedot internetistä, ja ilmoittaa käyttäjälle missä tällä hetkellä tuulee Suomessa eniten ja vähiten.

   Lataa ohjelmassa data tästä osoitteesta (koodiesimerkki alempana):
   https://edu.frostbit.fi/api/weather/

   Huom: data päivittyy päivittäin klo 9:00 ja 21:00, ja se sisältää keskiarvot viimeisen 12h ajalta.

   Osoitteen data koostuu **dictionaryistä**, jotka ovat **listassa**. Esimerkki yksittäisestä datasta:

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

   Datakenttien selitteet:

   snow = keskimääräinen lumen korkeus (cm)

   rain = keskimääräinen sademäärä (mm)
   - ***0.3 - 0.9 mm*** = vähäinen sade
   - ***1 - 4.4 mm*** = sadetta
   - ***yli 4.5 mm*** = runsas sade

   wind = keskimääräinen tuulen nopeus, metriä per sekunti (m/s)

   area = säämittausalue, joko lapland, middle tai south

Voit aloittaa tällä koodilla. Internetistä ladatut tiedot ovat tässä tapauksessa **weather** –muuttujassa:

```
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)
```

Esimerkki lopputuloksesta (tulokset riippuvat aina päivän datasta):

Esimerkki ohjelman toiminnasta:

```
Tänään eniten tuulee sijainnissa: Helsinki, 5.8 m/s
Tänään vähiten tuulee sijainnissa: Enontekiö, 1.1 m/s
```

Tästä tehtävästä annetaan enemmän pisteitä kuin muista perustehtävistä!

Vinkki: Heikoin tuuli voi tuntu aluksi hieman kinkkiseltä ratkaista. Tämä johtuu siitä, että jos alkuarvo heikoimmalla tuulella on 0, emme voi vain verrata kaupungin tuulta siihen, koska mikään tuuli ei ole vähemmän kuin 0. 😊

Sen sijaan voimme tehdä ehtolauseen, joka on jotain tähän tyyliin:
if wind < weakest_wind or weakest_wind == 0:
=> aseta uusi arvo heikoimmaksi tuuleksi

MIksi tämä toimii? Jos weakest_wind on tasan 0, se tarkoittaa sitä että sille ei ole vielä asetettu minkään tuulen arvoa. Joten heti ensimmäisellä kierroksella käy niin, että weakest_wind-muuttujan ensimmäiseksi arvoksi asetetaan ensimmäinen tuuliarvo. Tämän jälkeen, vertailu wind < weakest_wind toimii aivan ok. 😊

Lisätehtävä: tulosta myös jokaisen mittausalueen (area) keskimääräinen tuulennopeus. Pyöristä keskiarvo yhteen desimaaliin. Esimerkki lopputuloksesta:

```
Keskimääräinen tuuli, Lappi: 1.6 m/s
Keskimääräinen tuuli, Maan keskiosa: 3.3 m/s
Keskimääräinen tuuli, Etelä-Suomi: 4.7 m/s
```

Tehtävän tiedostonimi = ***exercise7_5.py***

Tyypillinen koodimäärä***: 14-32 riviä*** *(tyhjiä rivejä ja kommentteja ei lasketa)*

Lisätehtäviä!

Huom: Hyvä arvosana ei välttämättä tarvitse kaikkien lisätehtävien tekemistä. Voit valita itsellesi mieluisat tehtävät!

6. Luo ohjelmaasi seuraavanlainen kokoelma, jossa on useita ravintoloita (vähintään 5 kpl tai enemmän):

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: "Tornio"**
- **2:**
  - **name: ”Snacksy Oy”**
  - **rating: 3.2**
  - **reservations: False**
  - **services:**
    - **”lunch”**
    - **”dinner”**
    - **"night"**
  - **price_level : 2**
  - **location: "Oulu"**

jne.

Lisää myös muitakin ravintoloita, joko oikeita tai kuvitteellisia. Voit käyttää kuvitteellisia arvosteluarvoja tässä harjoituksessa.

Pyydä tämän jälkeen käyttäjältä kysymyksiä, minkälaisen ravintolan hän haluaisi valita (numero- ja kyllä/ei -kysymyksiä).

Esim.

Tervetuloa ravintolahakuun!
Monenko tähden ravintolan haluat vähintään? (1-5)
Minkä hinta-tason ravintolan haluat maksimissaan? (1-5)
Haluatko tehdä etukäteen varauksen? (k/e)
Mihin kellonaikaan haluat ruokailla? (0 – 23)

Kun ohjelma on kysynyt nämä kysymykset, näytä lista niiden ravintoloiden nimistä, jotka vastaavat hakua. Mikäli yksikään ravintolaa ei vastaa hakua, tulosta ”**Valitettavasti sopivaa ravintolaa ei löytynyt!**”

services –kentän tietoja tulee hyödyntää kellonajoissa tällä logiikalla:

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

Eli jos käyttäjä syöttää vaikkapa klo 13, täytyy sen ravintolan **services** –listasta löytyä sana "lunch".

Tehtävän tiedostonimi = ***exercise7_6.py***

Tyypillinen koodimäärä***: vaihtelee, n. 50-120 riviä*** *(tyhjiä rivejä ja kommentteja ei lasketa)*

7. Tee ohjelma, joka selvittää missä kaupungissa on Suomessa tilastollisesti ollut eniten liukastumisvaroituksia.

   Liukastumisvaroituksia voi ladata seuraavasta internet-rajapinnasta (API):

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

   Näytä lisäksi myös 5 viimeisintä liukastumisvaroitusta aikaleiman perusteella (kaupunki + päivämäärä + kellonaika).

Tehtävän tiedostonimi = ***exercise7_7.py***

Tyypillinen koodimäärä***: 15-30 riviä*** *(tyhjiä rivejä ja kommentteja ei lasketa)*

8. Tee ohjelma, joka pyytää käyttäjältä vuosiluvun (2017 – 2020) ja laskee Oulun alueen terveyspalveluiden keskimääräisen jonotusajan kyseiseltä vuodelta.

   Data löytyy seuraavasta osoitteesta:

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

   Huom:

   Jätä huomiotta sellaiset rivit, joista aikaleima puuttuu (null). Huomioi myös, että yksittäisen datan keskimääräinen jonotusaika on doctor_queuen ja nurse_queuen välinen keskiarvo.

Tehtävän tiedostonimi = ***exercise7_8.py***

Tyypillinen koodimäärä***: 12-30 riviä*** *(tyhjiä rivejä ja kommentteja ei lasketa)*