Viikkotehtävät 9 - Funktiot

HUOM! Kaikissa tehtävissä:

Tallenna omat funktiosi erilliseen tiedostoon: functions.py, ja lataa se jokaisen tehtävän kohdalla aina myös codePostin palautuslaatikkoon tehtävän lisäksi.

Muista aina ottaa omat funktiosi koodiisi mukaan **import**-komennolla!
esim. jos omat funktiot on tiedostossa **functions.py**, jossa on **greetings()** –niminen funktio, silloin:

```
from functions import *
greetings()
```

tai

```
import functions
functions.greetings()
```

Huom: Kun palautat tehtäviä, ei haittaa vaikka **functions.py** –tiedostossa on kaikkien tehtävien funktiot samassa tiedostossa. Funktioiden hienous on siinä, että niitä voi olla "varastossa" ylimääräisiäkin, koska ne eivät pelkällä olemassaolollaan tee mitään koodissa! **functions.py on tässä tapauksessa ikään kuin työkalupakki, josta eri tehtävät käyttää eri työkalua tarpeen mukaan eri tarkoituksiin.**

1. Tee funktio nimeltä ***show_personal_info()***, joka tulostaa henkilötiedot (nimi, kotipaikka, ammatti). Voit käyttää kuvitteellisia henkilötietoja tässä harjoituksessa.

   show_personal_info() -funktion **ei tule ottaa vastaan parametrejä, eikä sen tule palauttaa mitään, vaan se tulostaa ainoastaan kolme riviä tekstiä.**

   Kutsu funktiota ohjelmassasi.

   Esimerkki ohjelman toiminnasta:

```
Matti Meikäläinen
Sodankylä
Ohjelmistosuunnittelija
```

Tehtävän tiedostonimi = ***exercise9_1.py***

Tyypillinen koodimäärä***: 6-10 riviä*** *(sisältää myös funktioiden koodit, tyhjiä rivejä ja kommentteja ei lasketa)*

2. Tee funktio **count_seconds(hours, minutes, seconds)**, joka ottaa vastaan parametrit: tunnit, minuutit ja sekunnit. Funktion **tulee palauttaa** (eli **return**) vastauksena **yksi** kokonaisluku siitä, kuinka monta sekuntia annetut parametrit ovat yhteensä. Pyydä laskettavat luvut käyttäjältä, ja tulosta lopputulos.

   Älä ohjelmoi yhtään print() -kutsua funktion sisälle!

   Esimerkki ohjelman toiminnasta:

```
Anna tunnit:
3
Anna minuutit:
47
Anna sekunnit:
33
Yhteensä 13653 sekuntia.
```

Tehtävän tiedostonimi = ***exercise9_2.py***
Tyypillinen koodimäärä***: 7-16 riviä*** *(sisältää myös funktioiden koodit, tyhjiä rivejä ja kommentteja ei lasketa)*

Lisätehtävä:
Hyväksy käyttäjän antama formaatti ajasta: **“2h 45min 33sek”**

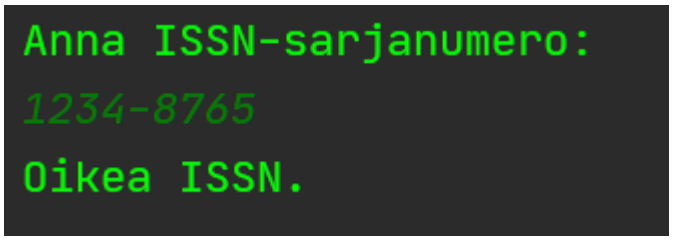
Vinkki: käytä tekstinkäsittelyfunktioita, osatekstiä tai split()-funktiota! **Palauta vain tämä versio perusversion sijasta, codePost osaa ottaa sen huomioon.**

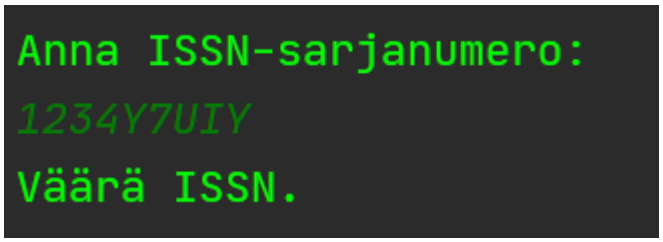
3. Tee funktio nimeltä **magazine_serial_check(serial)**, joka vastaanottaa parametrina vain ISSN-sarjanumeron (tekstinä), ja tarkistaa onko se oikeassa muodossa. Jos sarjanumero on oikeassa muodossa, tulosta "**Oikea ISSN**". Jos sarjanumero ei ole oikeassa muodossa, tulosta "**Väärä ISSN**". Voit tulostaa tekstin joko suoraan funktiossa, tai palauttaa tulostettava tekstin muuttujaan.

   Oikeassa muodossa oleva ISSN-sarjanumero sisältää kahdeksan numeroa, joiden keskellä on viiva.

   Esimerkiksi **0781-2078**, on oikeassa muodossa oleva ISSN-numero. **1564XABCD** taas ei ole ISSN-numero.

   Esimerkkejä ohjelman toiminnasta:





   Tehtävän tiedostonimi = ***exercise9_3.py***
   Tyypillinen koodimäärä***: 10-24 riviä*** *(sisältää myös funktioiden koodit, tyhjiä rivejä ja kommentteja ei lasketa)*

   Taustatietoa: ISSN-sarjanumeroa (International Standard Serial Number) käytetään sellaisten julkaisujen yksilöimiseen, joita julkaistaan jatkuvasti samalla nimellä (esim. **lehdet**). ISSN-numero on ainutlaatuinen maailmanlaajuisesti.

Vinkki: Pythonin tekstifunktioista on tässä paljon hyötyä.

Yksi lähestymistapa on tarkistaa ensin, onko keskimmäinen merkki viiva (indeksi 4). Tämän jälkeen voit poistaa keskellä olevan viivan (string.replace()), ja katsoa sen jälkeen onko ISSN-numero tasan kahdeksan merkkiä pitkä, ja lopuksi tarkistaa koostuuko se vain numeroista! (ks. tekstifunktiot aiemmissa materiaaleissa).

Tämän rakenteen voi tehdä monella tavalla, helpoin lienee **if-elif-elif-else** -rakenne.

Pieni lisätehtävä: Palauta (return) funktiosta Boolean, joka ilmoittaa onko annettu ISSN-numero oikeassa muodossa (True/False). Tässä tapauksessa älä tulosta lopputulosta itse funktiossa, vaan vasta ohjelmakoodissa.

4. Tee funktio nimeltä **show_numbered_list(title, data),** joka ottaa vastaan parametrina kaksi parametria: ***otsikkotekstin*** sekä **listan tapahtuman osallistujien nimistä**. Funktion tulee ensin tulostaa annettu otsikko, sekä sen jälkeen numeroitu lista osallistujista.

   Huom: funktion sisälle tuleva koodi on melkoisen lyhyt, **ja siihen ei tarvitse yhtään if-lausetta sisälle!**

   Käytännössä funktion tulee tehdä <u>vain ja ainoastaan</u> seuraavaa:

   - tulosta title -muuttuja (oli se mikä tahansa)
   - tämän jälkeen funktiossa tulee olla silmukka, joka tulostaa jokaisen data-listan -elementin kerrallaan rivinumeron kanssa (oli listassa sisältönä mitä tahansa).

   Huom: funktion sisälle ei tule yhtään if-lausetta sisälle!

   Kutsu samaa funktiota kolme kertaa:

   1. Tulosta lista alkuperäisessä järjestyksessä (otsikko: **"Ilmoittautumisjärjestys:"**)
   2. Järjestä lista aakkosjärjestykseen ja tulosta se (otsikko: **"Aakkosjärjestys etunimen perusteella:"**)
   3. Järjestä lista aakkosjärjestykseen sukunimen perusteella (ks. vinkit alempaa) ja tulosta se (otsikko: **"Aakkosjärjestys sukunimen perusteella:"**)

   Pyydä osallistujalista yhdellä rivillä pilkulla eroteltuna (ks. vinkit alempaa):

Esimerkki ohjelman toiminnasta:

```
Syötä tapahtuman osallistujat pilkulla eroteltuna:
Matti Korhonen, Timo Virtanen, Tuula Salminen, Juha Mäkinen, Ritva Heikkilä

Ilmoittautumisjärjestys:
1. Matti Korhonen
2. Timo Virtanen
3. Tuula Salminen
4. Juha Mäkinen
5. Ritva Heikkilä

Aakkosjärjestys etunimen perusteella:
1. Juha Mäkinen
2. Matti Korhonen
3. Ritva Heikkilä
4. Timo Virtanen
5. Tuula Salminen

Aakkosjärjestys sukunimen perusteella:
1. Heikkilä Ritva
2. Korhonen Matti
3. Mäkinen Juha
4. Salminen Tuula
5. Virtanen Timo
```

Tehtävän tiedostonimi = ***exercise9_4.py***
Tyypillinen koodimäärä***: 12-18 riviä*** *(sisältää myös funktioiden koodit, tyhjiä rivejä ja kommentteja ei lasketa)*
Tyypillinen koodimäärä, **funktiossa oleva koodi: *3-8 riviä***

Huom: show_numbered_list() -funktioon ei tule yhtään if-lausetta!

Hahmotelma odotetusta ohjelman rakenteesta:

```
# import oma funktiotiedosto

# pyydetään tapahtuman osallistujat yhdellä rivillä, pilkulla eroteltuna
# muutetaan käyttäjän antama teksti listaksi

# otetaan nimistä ylimääräiset välilyönnit pois (nimen alusta ja lopusta)
# ks. tehtävänannon vinkit tähän liittyen!

# vaihe 1: kutsutaan show_numbered_list -funktiota, parametreina otsikko ja osallistujalista

# vaihe 2: järjestetään osallistujat-lista sorted() -funktiolla aakkosjärjestykseen

# vaihe 3: kutsutaan show_numbered_list -funktiota, parametreina otsikko ja osallistujalista

# vaihe 4: järjestetään osallistujat aakkosjärjestykseen sukunimen perusteella
# ks. tehtävänannon vinkit tähän liittyen!

# vaihe 5: kutsutaan show_numbered_list -funktiota, parametreina otsikko ja osallistujalista

# HUOM! funktiossa ei tehdä mitään muuta kuin tulostetaan otsikko ja
# sen jälkeen listan sisältö. Eli ei yhtään if-lausetta funktioon!
```

Vinkkejä:
Jos käyttäjä syöttää vaikkapa tällaisen tekstin: ***Matti Meikäläinen, Tiina Testaaja, Kaisa Koekäyttäjä,*** saadaan siitä lista näin:

```
# pyydetään numerot yhtenä tekstinä käyttäjältä
people_string = input("Syötä tapahtuman osallistujat pilkulla eroteltuna:\n")

# muutetaan käyttäjän antama teksti listaksi
people = people_string.split(",")

# otetaan nimistä ylimääräiset välilyönnit pois (nimen alusta ja lopusta)
people = [p.strip() for p in people]

# people-muuttuja on tässä vaiheessa lista, joka koostuu henkilöiden kokonimistä
```

Ennen kuin lista voidaan järjestää sukunimen perusteella, pitää meidän ensin muuttaa se sellaiseen muotoon, että jokainen nimi on muotoa "Sukunimi Etunimi". Se onnistuu kätevästi näin:

```
# muutetaan listan henkilöt niin, että sukunimi tulee ennen etunimeä.
# tämä on ns. list comprehension -ominaisuus Pythonissa.
# Logiikka: jokainen nimi vuorollaan muutetaan listaksi
# välilyönnin perusteella, eli tämän vuoksi jokainen nimi on lista,
# jossa on kaksi elementtiä, etunimi ja sukunimi.
# sitten elementtien järjestys muutetaan toisin päin käyttämällä
# reversed-funktiota- lopuksi uusi järjestys muutetaan takaisin
# kokonaiseksi nimeksi .join() -funktion avulla. Lopputuloksena
# people lista koostuu nimistä, joissa sukunimi tulee ensin.

people = [" ".join(reversed(p.split(" "))) for p in people]
```

Huom! Tee kaikki tulostaminen funktion sisällä! Eli jos people -lista sisältää nyt oikeassa järjestyksessä olevat nimet, sen jälkeen kutsutaan vain funktiota:

```
show_numbered_list("Tähän otsikko jonka haluat", people)
```

Tämän jälkeen muutetaan **people**-listaa tarvittavalla tavalla (esim. muutetaan järjestystä sort():lla), ja kutsutaan funktiota uudestaan samalla tavalla:

```
show_numbered_list("Jokin toinen otsikko tällä kertaa!", people)
```

Toisin sanoen, tämän tehtävän tarkoitus on demonstroida sitä, kuinka yhtä ja samaa funktiota voi käyttää monta kertaa eri tilanteissa! **Huom!** Funktion koodiin ei tule yhtään if-lausetta!

5. Tee tilavuuslaskuri, jolla voi laskea seuraavat tilavuudet:

1. **laatikon tilavuus**
   -> tee uusi funktio: ***box_volume(width, height, depth)***
   - kaava = leveys * korkeus * syvyys
   - palauta funktiossa laskutoimituksen tulos pyöristettynä kahteen desimaaliin (return). **Älä tulosta funktiossa mitään**.

2. **pallon tilavuus**
   -> tee uusi funktio: ***ball_volume(radius)***
   - kaava = (4 * π * säde^3) / 3
   - palauta funktiossa laskutoimituksen tulos pyöristettynä kahteen desimaaliin (return). **Älä tulosta funktiossa mitään**.

3. **putken tilavuus**
   -> tee uusi funktio: ***pipe_volume(radius, length)***
   - kaava = π * säde^2 * pituus
   - palauta funktiossa laskutoimituksen tulos pyöristettynä kahteen desimaaliin (return). **Älä tulosta funktiossa mitään**.

Huom: tee jokaiselle laskutoimitukselle oma funktio! **Älä tulosta yhdessäkään funktiossa mitään**, vaan palauta funktiosta luku (return), joka tulostetaan vasta varsinaisessa ohjelmassa!

Pyydä aluksi käyttäjältä numero, jonka perusteella valitaan haluttu laskutoimitus (vaihtoehtona luvut 0 – 3).

1 = laatikko, **2** = pallo ja **3** = putki, **0** = lopetetaan ohjelman käyttäminen

Kysy tämän jälkeen käyttäjältä laskutoimitukseen tarvittavat luvut, kutsu sopivaa funktiota, ja tulosta lopputulos käyttäjälle.

Aja ohjelmaa kunnes käyttäjä ei halua enää jatkaa (valitsemalla vaihtoehdon 0).

Esimerkkejä ohjelman toiminnasta:

```
Valitse toimenpide (1-3), 0 lopettaa ohjelman:
1
Anna laatikon leveys:
3
Anna laatikon korkeus:
5
Anna laatikon syvyys:
7
Laatikon tilavuus: 105 m3
```

```
Valitse toimenpide (1-3), 0 lopettaa ohjelman:
2
Anna pallon säde:
7
Pallon tilavuus: 1436.76 m3
```

```
Valitse toimenpide (1-3), 0 lopettaa ohjelman:
3
Anna putken säde:
6
Anna putken pituus:
9
Putken tilavuus: 1017.88 m3
```

```
Valitse toimenpide (1-3), 0 lopettaa ohjelman:
0
Kiitos ohjelman käytöstä!
```

Tehtävän tiedostonimi = ***exercise9_5.py***
Tyypillinen koodimäärä***: 25-50 riviä*** *(sisältää myös funktioiden koodit, tyhjiä rivejä ja kommentteja ei lasketa)*

Tästä tehtävästä tulee enemmän pisteitä kuin tavanomaisesta perustehtävästä!

Lisätehtäviä!

Huom: Hyvä arvosana ei vaadi kaikkien lisätehtävien tekemistä. Tärkeintä on, että jokaiselle löytyy jokin mielenkiintoinen vaihtoehto!

6. Lottorivi

Tee funktio, joka tuottaa satunnaisen lottorivin. Lottorivissä on 7 eri numeroa väliltä 1-40.

Vinkki: hyödynnä kokoelmaa (listaa), ja pidä kirjaa jo arvotuista numeroista, ettei vahingossa arvota useampaa samaa numeroa samaan riviin. Voit tarkistaa onko uusi arvottu luku listassa ehtolauseella (**if number in lotto_numbers**: => arvo uusi luku)

Arvo samalla myös lisänumerot! Voit käyttää samaa funktiota, tai tehdä erillisen funktion lisänumeroille.

Tulosta lopuksi arvottu lottorivi käyttäjälle.

Tehtävän tiedostonimi = ***exercise9_6.py***

7. Aiemmat harjoitukset funktioiden avulla

Valitse jokin seuraavista aiemmista harjoituksista, ja tee siitä sellainen versio, että ohjelman logiikka on sijoitettu joko yhteen tai useampaan funktioon. Voit tehdä myös monta aiempaa harjoitusta funktioilla, lisäpisteitä tulee maksimissaan 3:sta aiemmasta tehtävästä.

Vaihtoehdot:

- **Viikkotehtävä 3 – 3**
- **Viikkotehtävä 4 – 2**
- **Viikkotehtävä 6 – 2**
- **Viikkotehtävä 6 – 3**
- **Viikkotehtävä 7 – 5**
- **Lisätehtävä 3 – 7 tai 3 – 8**
- **Lisätehtävä 4 – 7**
- **Lisätehtävä 5 – 8**
- **Lisätehtävä 6 – 8**

Tehtävän tiedostonimi: sama kuin tehtävässä, josta uusi versio on tehty. Palauta tehtävä tehtävän alkuperäiseen palautuslaatikkoon.

Huom: joissakin tehtävissä palautuslaatikon testit eivät välttämättä aina hoksaa funktiolla tehtyä ratkaisua, joten älä välitä pisteistä liikaa. Ohjaaja arvioi funktio-pohjaiset palautukset erikseen käsin.

8. Valuuttamuunnin

Tee funktio nimeltä "**convert_money**", joka ottaa vastaan muunnettavan summan, alkuperäisen valuutan sekä valuutan johon summa muutetaan. Hyväksyttävät valuutat ovat € (euro), $ (dollari), £ (punta) sekä kr (Ruotsin kruunu).

Funktion tehtävä on muuntaa annettu rahasumma haluttuun valuuttaan valuuttakurssien mukaisesti.

Käytä näitä valuuttakursseja hyväksi muunnoksessa (voit myös hakea ajantasaiset kurssit internetistä):

- 1 € = 1.1 $ (dollari)
- 1 € = 0.9 £ (punta)
- 1 € = 11.8 kr (Ruotsin kruunu)

Pyydä ensin käyttäjältä rahasumma sekä valuutan yksikkö. Kysy tämän jälkeen mihin valuuttaan hän haluaa muuntaa rahasumman. Hyödynnä tämän jälkeen tekemääsi funktiota, ja anna käyttäjälle muunnettu rahasumma.

Lisätehtävä: Hae päivän valuuttakurssit jostain internet-rajapinnasta!

Tehtävän tiedostonimi = ***exercise9_8.py***

9. Rekursiot

Rekursiivinen funktio on sellainen, joka kutsuu tarvittaessa myös itseään. Muista: rekursiota ei ole pakko käyttää yksinkertaisissa tapauksissa. Esimerkiksi aiemmin koodipajassa tekemämme Fibonacci-esimerkki oli täysin mahdollista tehdä pelkillä silmukoillakin.

Esimerkiksi Fibonacci-harjoitus, jonka teimme aiemmin luennoilla, voidaan tehdä rekursiolla näin (ks. kohta Fibonacci Numbers):
https://www.python-course.eu/python3_recursive_functions.php

Tässä lisätehtävässä, tee Pythonilla rekursiivinen funktio, joka tulostaa annetun kansion sisällön, sekä kaikki sen alikansiot ja niiden sisällöt mukaan lukien.

Aina kun tulostetaan kansion nimi, tulostetaan perään kolme pistettä. Lisäksi jokainen kansion taso merkitään ylimääräisellä viivalla, esim.

Pictures …
- Flower.jpg
- Pear.jpg
- Vacation …
- - Prague1.jpg
- - Prague2.jpg
- TODO.txt

Voit hyödyntää seuraavaa Python-moduulia tässä harjoituksessa:

```
import os

for x in os.listdir("C:\\kansion_nimi\\toinen_kansio"):
    print(x)
```

Huom: Tätä operaatiota varten on Pythonissa olemassa valmiitakin työkaluja, mutta tämän harjoituksen ideana on toteuttaa kyseinen rekursiivinen operaatio itse.

Tehtävän tiedostonimi = ***exercise9_9.py***

Hyödyllinen Google-haku:
"python 3 traverse directory contents with recursion"

10. Edistyneemmät funktiotekniikat

- **Tee lambda, joka palauttaa Booleanin riippuen siitä, onko annettu kokonaisluku parillinen**
    - **Vinkki:** Lambdan voi tallentaa muuttujaan!

- **Tee seuraavaksi kokoelma, jossa on satunnaisesti parittomia ja parillisia lukuja**

- **Käytä filter():ä siten, että suodatat kokoelmasta parittomat luvut käyttämällä aiemmin tekemääsi lamdbaa**
    - filter() palauttaa ns. filter objectin, joka pitää kääntää vielä listaksi käyttämällä list() -funktiota

- **Käytä map()- funktiota siten, että käsittelet kokoelman aiemmin tekemälläsi lambdalla**
    - map() –palauttaa ns. map objectin, joka pitää kääntää vielä listaksi käyttämällä list() -funktiota

Lopputuloksena pitäisi olla kaksi uutta listaa, joista toinen sisältää parittomat luvut, ja toinen sisältää listan True ja False –arvoja (eli onko parillinen vai ei). **Tulosta nämä listat lopuksi ohjelmassa (suora print() -kutsu riittää).**

Lisätietoa:

filter() ja map() voivat vaikuttaa samanlaisilta ominaisuuksilta, mutta niiden suorittama operaatio on päinvastainen. Filter suodattaa tiedosta halutut tiedot lambdan perusteella. Lopputulos tässä tapauksessa on alkuperäinen data muokattuna siten, että lambdan mukaiset muuttujat on siivottu pois. Map sen sijaan kerää talteen kaikki arvot jotka lambda palauttaa. Kumpaa tekniikkaa tarvitaan riippuu siitä, halutaanko muokata alkuperäistä tietoa (filter) vai halutaanko kerätä tietoa listan jokaisesta elementistä (map).

Huom: muista myös list comprehension kun tarvitset kokoelmien suodatusta!

Tehtävän tiedostonimi = ***exercise9_10.py***

11. Monimutkainen tiedon järjestäminen (custom sort)

Tee ohjelmassasi seuraava lista (joka koostuu dictionaryistä):

- **0:**
  - **city : ”Rovaniemi”**
  - **company: ”Sampokeskus”**
- **1:**
  - **city : ”Oulu”**
  - **company: ”Myllyoja”**
- **2:**
  - **city : ”Rovaniemi”**
  - **company: ”Rinteenkulma”**
- **3:**
  - **city : ”Rovaniemi”**
  - **company: ”Revontulikeskus”**
- **4:**
  - **city : ”Oulu”**
  - **company: ”Valkea”**
- **5:**
  - **city: ”Oulu”**
  - **company: ”Ideapark”**

Tee Python –ohjelma, jossa on funktio nimeltä **city_company_sort(data)**, joka suodattaa listan siten, että tiedot laitetaan aakkosjärjestykseen ensin kaupungin, ja vasta sitten ostoskeskuksen nimen perusteella. Tämän voi ratkaista usealla eri tavalla!

Tehtävän tiedostonimi = ***exercise9_11.py***

Lopputuloksen tulisi olla tässä tapauksessa:
Oulu: Ideapark
Oulu: Myllyoja
Oulu: Valkea
Rovaniemi: Revontulikeskus
Rovaniemi: Rinteenkulma
Rovaniemi: Sampokeskus

Vinkki: Asiaa voi Googlettaa esim. ”python 3 sort list of dictionaries by multiple keys”