CS-A1111 Ohjelmoinnin peruskurssi Y1 Tentti 27.11.2018

Tärkeitä ohjeita vastausten kirjoittamiseen: Kun kirjoitat ohjelmakoodia, käytä kahden ruudun levyisiä sisennyksiä. Jos sisennyksiä ei ole käytetty tai niistä ei saa selvää, vähennetään siitä pisteitä. Kirjoitettavaan ohjelmakoodiin ei tarvitse lisätä kommentteja. Missään tehtävässä tulostusta ei tarvitse muotoilla. Voit myös olettaa, että käyttäjän antama syöte on virheetöntä, ellei tehtävässä erikseen käsketä käsittelemään virhetilanteita.
Tentissä ei saa käyttää laskimia eikä lisämateriaalia. Opiskelijat, joiden äidinkieli ei ole suomi, saavat kuitenkin käyttää sanakirjaa, jos siinä ei ole merkintöjä (tentin valvoja tarkistaa sanakirjan). Nämä opiskelijat saavat halutessaan myös sekä suomen- että ruotsin- tai englanninkielisen kysymyspaperin.

Vastaa kurssin palautekyselyyn. Kyselyyn vastaamisesta saa 200 harjoitustehtäväpistettä. Linkki kyselyyn on lähetetty 23.11. sähköpostitse kurssille ilmoittautuneille.

1. a) (2 p)
Tee kurssin lopputesti A+-järjestelmässä. Tämän tentin pisteistä 2 tulee lopputestiin vastaamisesta. Testi on auki 5.12. klo 23:59 asti. Myös kesäkurssille osallistuneet voivat kirjautua Aalto-tunnuksilla syksyn kurssille A+-järjestelmässä ja tehdä testin siellä. Linkki A+-järjestelmään on MyCoursesissa kesäkurssin sivulla.

Kohdissa b, e ja f kerro, mitä annettu ohjelma tulostaa. Vastausta ei tarvitse perustella. Kohdissa c ja d kerro, mitä tehtävässä esitetty funktio tekee. Älä selitä funktion toimintaa käsky käskyltä, vaan selitä parilla lauseella, mikä on funktion tarkoitus (esimerkiksi: "funktio laskee ja palauttaa parametrina annetussa listassa olevien lukujen summan"). Funktioille annettavien parametrien luonne on selitetty kohdissa c ja d. Kaikki eri kohdissa annetut ohjelmat ja funktiot toimivat jollain tavalla eli niistä minkään suoritus ei esimerkiksi keskeydy siksi, että ohjelmassa olisi virhe.

b) (2 p)
```
def main():
    bacteria = 200
    if bacteria >= 100:
        print("Water quality is poor.")
    if bacteria < 10:
        print("Water quality is excellent.")
    else:
        print("Water quality is OK.")

main()
```

c) Funktiolle annetaan parametreina kaksi listaa, jotka sisältävät kokonaislukuja ja joissa on sama määrä alkioita. (4 p)
```
def mystery_func1(list1, list2):
    i = 0
    while i < len(list1):
        if list1[i] > list2[i]:
            list2[i] = list1[i]
        i += 1
    return list2
```

d) Funktiolle annetaan parametreina kaksi merkkijonoa, joissa molemmissa on vähintään yksi merkki. (4 p)
```
def mystery_func2(first, second):
    final = first[0]
    final += "."
    final += second[0]
    final += "."
    return final
```

e) (9 p)

```
def mystery1(lst, stp):
    nlst = []
    for elem in lst:
        if elem != stp:
            nlst.append(elem)
        else:
            return nlst

def mystery2(lst, value):
    nlst = []
    for elem in lst:
        if elem != value:
            nlst.append(elem)
    return nlst

def mystery3(lst):
    tota = 0
    for elem in lst:
        tota = tota + elem
    return tota

def mystery4(lst):
    if len(lst) > 0:
        return mystery3(lst) / len(lst)
    else:
        return -1

def main():
    values = [2, 3, -1, 0, 3, 0, 1, -2, 8, 1, 2, 3]

    lst_s = mystery1(values,8)
    print(lst_s)
    lst_c = mystery2(lst_s, 0)
    print(lst_c)
    a = mystery4(lst_c)
    print(a)

main()
```

f) (4 p)

```
def main():
    lst = [2, -1, 3, 0, 1, 0, -1, 2, 9, 1, 2, 3]
    var1 = 0
    count = 0
    index = 0
    sent = 9
    element = lst[index]

    while element != sent:
        if element > 0:
            var1 = var1 + element
            count += 1
        index += 1
        element = lst[index]

    if count > 0:
        print(var1, count, var1 / count)
    else:
        print(-1)

main()
```

2. a) Ryhmä opiskelijoita on lähdössä ekskursiolle. Vaihtoehtona on kulkea matka joko junalla, bussilla tai henkilöautolla. Kirjoita Python-ohjelma, joka tutkii, mikä em. vaihtoehdoista on kustannuksiltaan halvin. Ohjelma kysyy käyttäjältä ryhmän koon, yhden hengen juna- ja bussilippujen (meno-paluulippu) hinnat sekä edestakaisen automatkan kilometrit ja sen, kuinka monta henkilöautoa ryhmän kuljettamiseen tarvitaan. Automatkan kustannuksiksi oletetaan 20 snt / km jokaista tarvittavaa henkilöautoa kohti siitä riippumatta, montako henkilöä autoissa matkustaa. Ohjelma tulostaa, onko halvinta kulkea matka junalla, bussilla vai henkilöautoilla, kun koko ryhmän kustannukset lasketaan yhteen. Lisäksi ohjelman pitää tulostaa edullisimman vaihtoehdon hinta (koko ryhmän yhteishinta). Jos kaksi tai kolme halvinta vaihtoehtoa on täsmälleen samanhintaisia, riittää että ohjelma tulostaa niistä yhden (ei ole väliä minkä). (10 p.)

   b) Kodinkoneliikkeen kaikkien myyjien kuukauden aikana myymien tuotteiden hintojen summat on tallennettu listaan niin, että listan kukin alkio (desimaaliluku) kertoo yhden myyjän kuukauden kokonaismyynnin eli tämän myyjän kuukauden aikana myymien tuotteiden hintojen summan euroina. Kirjoita Python-funktio `under_average(sales, difference)`, joka saa ensimmäisenä parametrina edellä kuvatun listan ja toisena parametrina desimaaliluvun. Funktio palauttaa arvonaan tiedon siitä, kuinka monella myyjällä kokonaismyynti on vähintään toisena parametrina annetun luvun verran pienempi kuin liikkeen kaikkien myyjien kokonaismyyntien keskiarvo. Jos esimerkiksi listassa olevien lukujen keskiarvo on 15000.0 ja funktion toinen parametri on 2000.0, funktio palauttaa tiedon siitä, kuinka monella liikkeen myyjällä kokonaismyynti on 13000 euroa tai vähemmän. Kirjoita vain pyydetty funktio, älä ohjelman muita osia. (20 p)

3. Kuntoilija on kerännyt tiedot liikuntaharjoittelustaan tekstitiedostoon niin, että yhdellä rivillä on aina ensin yhden harjoituksen päivämäärä, sitten harjoituksen kesto minuuteissa, harjoituksen taso (rasittavuutta kuvaava kokonaisluku asteikolla 1-10) ja lopuksi urheilulaji. Eri tiedot on erotettu toisistaan kaksoispisteellä. Tiedoston rivit voisivat näyttää esimerkiksi seuraavilta:

```
7.6.2018:30:8:running
8.6.2018:60:6:gym
9.6.2018:90:3:running
11.6.2018:45:5:running
```

   Kirjoita Python-ohjelma, joka laskee tiedostosta kaikkien niiden harjoitusten yhteisajan, joissa on käyttäjän haluama urheilulaji ja joiden taso on vähintään käyttäjän antama alaraja. Ohjelma pyytää käyttäjältä tiedot sisältävän tiedoston nimen, halutun urheilulajin ja vähimmäistason. Ohjelman tulostaa lasketun yhteisajan. Esimerkiksi jos käyttäjä antaa urheilulajiksi running ja tason alarajaksi 5, ohjelma tulostaisi yllä olevassa esimerkkitapauksessa 75 minuuttia.

   Ohjelman on käsiteltävä seuraavat virhetilanteet:
   - Annetun nimistä tiedostoa ei ole olemassa tai tiedoston lukeminen ei onnistu jostain muusta syystä.
   - Tiedoston jollain rivillä ajan tai tason paikalla ei ole kokonaisluku.

   Näissä tapauksissa ohjelma ilmoittaa käyttäjälle, millainen virhe on sattunut (siis joko että tiedoston lukeminen ei onnistu tai että tiedostossa on virheellinen luku), ja lopettaa toimintansa. Ohjelman ei tarvitse jatkaa rivien lukemista virheellisen rivin jälkeen. Voit myös olettaa, että tiedoston jokaisella rivillä on täsmälleen neljä toisistaan kaksoispisteellä erotettua osaa. Ohjelman ei tarvitse osata käsitellä esimerkiksi sellaisia virhetilanteita, joissa rivi on tyhjä tai ei sisällä päivämäärän lisäksi muuta tekstiä. (20 p)

4. Kirjoita Python-kielellä luokka `Creature` kuvaamaan eräässä tietokonepelissä toimivaa pelihahmoa. `Creature`-oliolla on oltava seuraavat kentät:
   - `__name` pelihahmon nimi.
   - `__force` pelihahmon voima, joka vaikuttaa siihen, miten hahmo menestyy taisteluissa. Voimaa kuvataan kokonaislukuna (mitä suurempi luku, sitä enemmän hahmolla on voimaa).
   - `__gun` kentän arvo on `True`, jos pelihahmolla on hallussaan ase, ja muuten `False`. **JATKUU**

Määrittele luokkaan seuraavat metodit. (Jos metodin kuvauksessa ei ole kerrottu mitään metodin palauttamasta arvosta, metodin ei tarvitse palauttaa mitään.)

- `__init__(self, name1, initial_force)` luo uuden `Creature`-olion. Luotavan pelihahmon nimi ja hahmon alkuperäinen voima on annettu parametreina. Jos viimeinen parametri on negatiivinen, pelihahmon voimaksi asetetaan 0. Uudella pelihahmolla ei ole hallussaan asetta.
- `get_name(self)` palauttaa pelihahmon nimen.
- `get_force(self)` palauttaa pelihahmon voiman (sitä kuvaavan kokonaisluvun).
- `has_gun(self)` palauttaa arvon `True`, jos pelihahmolla on hallussaan ase ja muuten arvon `False`.
- `pick_gun(self)` pelihahmo ottaa aseen haltuunsa eli metodi muuttaa olion kenttien arvoja sopivasti siten, että hahmolla on metodin suorituksen jälkeen hallussaan ase.
- `give_up_gun(self)` muuttaa olion kenttien arvoja sopivasti siten, että hahmolla ei ole metodin suorituksen jälkeen hallussaan asetta.
- `eat(self, amount)` kuvaa pelihahmon syömistä. Syödyn ravinnon määrä grammoina (kokonaisluku) annetaan metodin parametrina. Syöminen vaikuttaa siten, että pelihahmon voima kasvaa yhdellä jokaista syötyä grammaa kohti. Jos esimerkiksi pelihahmo syö 200 grammaa, sen voima kasvaa 200:lla. Metodi muuttaa pelihahmon voimaa vain siinä tapauksessa, että parametrina annettu määrä on positiivinen.
- `move(self, steps)` metodi kuvaa pelihahmon liikkumista. Liikuttavien askelten määrä on annettu parametrina. Jokainen liikuttu askel pienentää pelihahmon voimaa yhdellä. Liikkuminen ei kuitenkaan saa viedä voimaa negatiiviseksi. Jos esim. hahmon voima on 2000 ja parametrina on annettu 2500, niin hahmo liikkuu vain 2000 askelta. Metodi palauttaa liikuttujen askelten määrän.
- `fight(self, another_creature, level)` pelihahmo haastaa tappeluun parametrina annetun toisen pelihahmon (toisen `Creature`-olion). Viimeisenä parametrina annetaan kokonaisluku 1, 2 tai 3, joka kertoo, kuinka vakavasta tappelusta on kysymys. Tappelun voittaja määräytyy seuraavasti: jos toisella tappelijoista on hallussaan ase ja toisella ei, niin se, jolla on ase, voittaa tappelun. Muussa tapauksessa tappelun voittaja on se, jolla on suurempi voima. Jos kummankin hahmon tilanne aseen suhteen on sama ja kummallakin on yhtäsuuri voima, tappelun voittaa nykyinen `Creature`-olio (`self`). Jos nykyinen hahmo voittaa tappelun, hänen voimansa kasvaa joko 3000:lla, 6000:lla tai 9000:lla sen mukaan, onko tappelun kovuus 1, 2 vai 3 (mitä suurempi kovuus, sitä suurempi voiman kasvu). Jos hän häviää tappelun, hänen voimansa vähenee samalla luvulla ja lisäksi hän menettää aseensa, jos hänellä on sellainen. Voima voi mennä häviön seurauksena myös negatiiviseksi. Metodi ei muuta parametrina saadun vastustajan voimaa tai asetietoa mitenkään (se olisi kyllä järkevää, mutta tekisi tehtävästä vähän vaikeamman). Metodi palauttaa arvon `True`, jos nykyinen pelihahmo voittaa tappelun, ja arvon `False`, jos nykyinen pelihahmo häviää tappelun. Jos tappelun kovuutta kuvaava viimeinen parametri ei ole 1, 2 tai 3, metodi ei tee mitään muuta kuin palauttaa arvon `False`.
- `__str__(self)` palauttaa merkkijonon, joka sisältää pelihahmon nimen, voiman ja joko tekstin "the creature has a gun" tai "the creature does not have a gun" sen mukaan, onko hahmolla tällä hetkellä hallussaan ase vai ei.

Kirjoita lisäksi pääohjelma, joka luo kaksi `Creature`-oliota ja ottaa molempien haltuun aseen. Sen jälkeen ohjelman pitää yrittää liikuttaa ensiksi luotua pelihahmoa 500 askelta ja tulostaa, montako askelta tämä oikeasti liikkui. Seuraavaksi toiseksi luodun pelihahmon pitää haastaa ensiksi luotu pelihahmo tappeluun. Pääohjelman pitää tulostaa, voittiko toinen pelihahmo (haastaja) tappelun. Lopuksi ohjelman pitää tulostaa molempien hahmojen tiedot (nimi, voima ja tieto siitä, onko hahmolla hallussaan ase). Pääohjelman ei tarvitse kysyä mitään käyttäjältä, vaan voit päättää olioiden luonnissa ja metodien kutsuissa tarvittavat tiedot itse. (25 p)