Tentti: T-106.1210 Ohjelmoinnin peruskurssi, osa 1

Tenttipäivä: 23.5. 2012

EI APUVÄLINEITÄ

Yleistä

Tentissä on kaksi tehtävää. Ensimmäinen tehtävä on tärkeämpi: sillä pyritään varmistamaan, että jokaisella tämän kurssin läpäisevällä opiskelijalla on vähintäänkin auttavat taidot ohjelmakoodin lukemisessa ja kirjoittamisessa. Toinen tehtävä liittyy aiheisiin, jotka ovat hyödyllisiä mutta eivät tämän kurssin kannalta aivan välttämättömimpiä. Sen suorittamalla voi korottaa kurssiarvosanaa.

Tentistä tulee arvosanaksi jokin kolmesta vaihtoehdosta: hylätty, hyväksytty tai +1. Kunhan ensimmäisestä tehtävästä ei tule hylättyä arvosanaa, niin tentistä pääseen hyväksytysti läpi. Arvosanan +1 ja korotuksen kurssiarvosanaan saa ratkaisemalla myös jälkimmäisen tehtävän.

Inspiraatiota!

Tehtävä 1

Ohjelmassa matkustellaan julkisilla kulkuneuvoilla teekkarimaailmassa, jossa matkustelu tapahtuu Espoossa ja Helsingissä. Muita kaupunkeja ei ole (sinne ei tarvi mennä). Ykkösalue tarkoittaa jomman kumman kaupungin sisäistä matkaa ja kakkosalue matkaa(matkoja) kummankin kaupungin alueella. Kertalipulla ei ole voimassaoloaikaa. Se on siis todellakin kertalippu. Joka kerta kulkuneuvoon noustaessa on joko oltava voimasa oleva kausilippu, ostettava kertalippu tai ostettava lippu kuljettajalta.

Tutustu huolellisesti luokkiin TravelCard, TicketMachine ja CardReader. Luokka TravelCard on matkakortti. Kortilla voi olla rahaa lippujen ostoa varten sekä kausilippuja, joita voi ostaa jomman kumman kaupungin sisäiseen matkustamiseen, tai molemmissa matkustamiseen. Espoon kausilipulla ei voi matkustaa Helsingissä ja päinvastoin. Huomattavaa on myös että kausilipun voimassaolo alkaa ostohetkestä. Kertalipun voi ostaa joko yhden kaupungin sisäisenä tai kaupungista toiseen. Kausilipusta pidetään tallessa kaupunki ja loppumispäivä, kertalipusta kaupunki ja ostohetki (tarkastajia varten). Kertalipullakaan Espoon lipulla ei voi matkustaa Helsingissä, jne. Luokka CardReader on julkisen kulkuneuvon kortinlukija. Tässä versiossa jokaisella pysäkillä luodaan uusi lukulaite, jolle annetaan senhetkinen aika. Kortinlukijan ainoa toiminto on matkan maksu, johon on erilliset metodit kausilipun tarkistukseen ja kertalipun ostamiseen kortilla. Luokka TicketMachine on kortinlatausautomaatti. Siinä kortille voi ladata kausilipun ja rahaa lipun ostamiseksi kulkuvälineessä (arvo). Lisäksi koodista löytyy esimerkkejä luokkien käytöstä.

Toteuta luokan CardReader metodit **`check_period ja use_value`** siten, että ne toimivat kuvatulla tavalla. Toteutustapa on muuten vapaa.

- Tutustu annettuihin luokkiin huolella, jotta niiden toiminta selviää. Muista myös esimerkkitapaukset.
- **`check_period`**: Jos kortilla on voimassa oleva kausi pysäkin sjaintikaupunkiin,

palautetaan teksti: OK: valid period found. Jos voimassa olevaa kautta ei löytynyt, palautetaan teksti: Buy ticket

- **use_value**: Jos ei käytä kausilippua, joutuu ostamaan kertalipun: Yritetään ottaa kortilta lipun hintaa vastaava summa rahaa. Jos onnistuu, lisätään kortille uusi kertalippu, johon menee talteen pysäkin kaupunki ja ostohetki. Jos osto onnistui, palautetaan teksti: ticket bought, money left: X. (x:n paikalle kortille jäänyt rahamäärä). Jos osto ei onnistunut, palautetaan teksti: Buy ticket from the driver.
- Käytä hyväksesi valmiita metodeja.

Tehtävästä saa palautteena jonkin seuraavista:

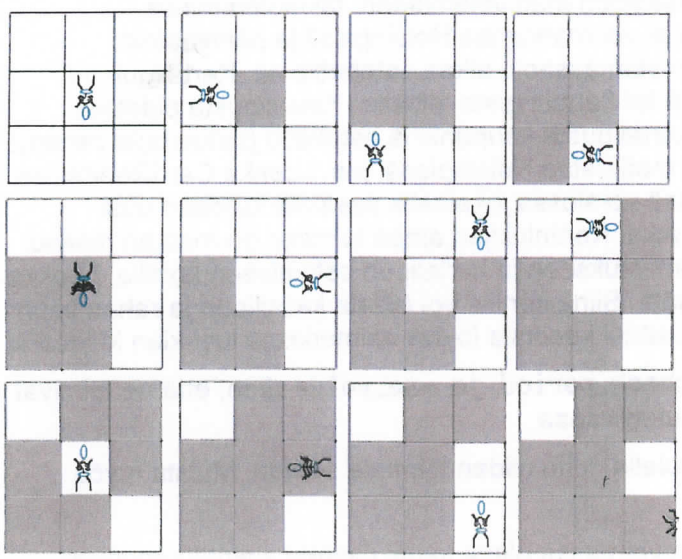
- Hylätty: metodien toteutus on selvästi puutteellinen ja toimimaton. Vastauksen perusteella ei saa käsitystä, että vastaaja osaa lukea/kirjoittaa ohjelmakoodia.
- Hyvä: vastaus kelpaa, vaikka siinä onkin joitain puutteita tai virheitä.
- Erinomainen: vastaus on käytännöllisesti katsoen täysin oikein.

Arvosanan kannalta on täysin sama saatko hyvän vai erinomaisen mutta on varmaan kiva tietää onnistuneensa.

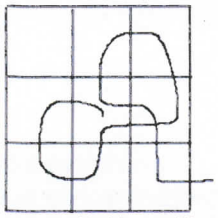
Yksittäisistä pikku- ja pilkkuvirheistä ei sakoteta mutta yritä silti kirjoittaa koodi niin täsmällisesti kuin pystyt. Kokonaisuus ratkaisee. Arvostelu tässä tehtävässä ei tule olemaan valtavan ankaraa.

Tehtävä 2

Myös tämä on täydennystehtävä. Tehtävässä liikutellaan ruudukossa (laudalla) ns. Langtonin muurahaista. Ruudukon kaikki ruudut ovat aluksi valkoisia ja vaihtavat väriä mustan ja valkoisen välillä muurahaisen edetessä. Muurahaisen liikkumissäännöt ovat yksinkertaiset: valkoisessa ruudussa muurahainen kääntää kulkusuuntansa vasemmalle, mustassa ruudussa oikealle, siirtyy sitten edessä olevaan ruutuun ja kääntää vielä lähtöruutunsa värin. Ohjelma loppuu, kun kohderuutu on ruudukon ulkopuolella (muurahainen tipahtaa). Alla muurahaisen kulku 3x3 ruudukossa. Liikkeelle lähtö on tässä keskeltä ja suunta on ensin pohjoiseen. Siitä muurahainen siirtyy vasemmanpuoleiseen ruutuun. Oikealla muurahaisen polku ruudukossa.



Luokka **Board** on käytettävä ruudukko. Ruudukko on esitetty taulukkona, jonka alkiot ovat sanakirjoja. Sanakirjassa on aina kaksi tietoa: ruudun väri (0 – valkoinen, 1 - musta) ja onko muurahainen siinä ruudussa (0 – ei, 1 – on). Luokka **Ant** on muurahainen. Se pitää itse kirjaa kulkusuunnastaan, joka menee ilmansuuntien mukaan: **N**orth, **E**ast, **S**outh, **W**est. Muurahaista luotaessa sen voi sijoittaa halumaansa kohtaan ruudukossa (ruudukko,



rivi, sarake, kulkusuunta). Merkkigrafiikassa valkoinen tyhjä ruutu on esitetty välilyöntinä, valkoinen ruutu, jossa on muurahainen, on X, musta tyhjä ruutu on # ja musta ruutu muurahaisella on *. Tee muurahaisen liikkumisesta vastaava **move** metodi:

- Luokkamuuttujana on annettu sanakirja, josta voi hakea oikeat muutokset rivin ja sarakkeen numeroihin sekä uuden kulkusuunnan, kun tietää kulkusuunnan ja lähtöruudun värin.
- Toteuta metodiin muurahaisen liikkuminen ruudusta toiseen sääntöjen mukaan.
- Laudalta putoaminen lopettaa ohjelman.

Tehtävästä saa palautteena jonkin seuraavista:

- Puutteellinen: vastaus on selvästi puutteellinen tai väärin
- Sinnepäin: vastauksessa on jotain oikeaa, mutta myös jotain oleellista puuttuu
- Oikein: vastauksessa on kaikki pyydetyt osiot toimivasti toteutettuina (ainoastaan tästä saa +1 kurssiarvosanaan)

Arvosanakorotus edellyttää siis, että kaikki tehtävänannossa pyydetyt asiat ovat kunnossa. Tässäkään tehtävässä ei kuitenkaan muotoseikoista sakoteta. Jos et muista varmasti, miten jokin tietty asia Pythonissa kirjoitetaan, voit yrittää paikata tätä selittämällä sen, mitä koodisi tekee.

TEHTÄVÄN 1 KOODI:

```
from datetime import datetime, timedelta

class TravelCard(object):
    '''
    Matkakorttia esittävä luokka. Kortilla voi olla kausilippu sekä rahaa
    ladattuna. Kausilippu on voimassa, jos sille löytyy loppuaika, joka on
    suurempi kuin lipunostohetki.
    '''
    def __init__(self, name):
        self.name = name
        self.euros = 0 #kortilla oleva raha
        self.periods = [] #ei kausilippua
        self.single_tickets = [] #Kertalippuja voi olla useampia

    def add_period(self, endtime, towns):
        '''
        Kausilipun lisäys korttiin. Talteen laitetaan loppumispäivä ja
        kaupungit, joissa se on voimassa.
        '''
        for town in towns:
            self.periods.append((town, endtime))

    def add_money(self, euros):
        '''
        Rahan lisäys kortille
        '''
        self.euros += euros

    def take_money(self, euros):
        '''
        Kortin rahamäärää pienennnetään lipun hinnan verran. Palauttaa jäljelle
        jääneen rahamäärän. Jos rahaa ei ole tarpeeksi, palauttaa -1.
        '''
        if self.euros >= euros:
            self.euros -= euros
            return self.euros
        return -1

    def add_ticket(self, region, bought_time):
        '''
        Lisää kertalipun listaan. Tietoina voimassaoloalue ja ostohetki.
        '''
        self.single_tickets.append((region, bought_time))

    def read_card(self):
        '''
        Palauttaa kortin tiedot: kausiliput ja kertaliput
        '''
        return [self.periods, self.single_tickets]


class TicketMachine(object):
    def load_period(self, card, towns, endtime):
        '''
        Lataa annetulle kortille kausilipun, joka loppuu annettuna aikana ja
        voimassa annetuissa kaupungeissa.
        '''
        card.add_period(endtime, towns)

    def load_value(self, card, euros):
        '''
        Lataa annetulle kortille arvoa annetun euromäärän.
        '''
        card.add_money(euros)
```

```
class CardReader(object):
    '''
    Kulkuneuvossa oleva kortinlukija. Luotaessa annetaan kortinlukuhetkeä
    kuvaava aika, lippuhinnat alueittain sekä kaupunki, minkä alueella
    senhetkinen pysäkki on.
    '''
    def __init__(self, town, time_now, ticket_prices):
        self.time_now = time_now
        self.prices = ticket_prices
        self.town = town # pysäkin sijaintikaupunki

    def check_period(self, card):
        '''
        Matkustetaan kausilipulla. Parametrina tulee korttiolio, joka toimii
        maksuvälineenä. Kausilipulla matkustettaessa
        aluetta ei tarvitse antaa. Jos kortilla on voimassa oleva kausilippu
        pysäkin sijaintikaupungissa, palautetaan teksti: OK: valid period found.
        Jos kautta ei löytynyt, palautetaan teksti:
        Buy ticket.
        '''
        # Toteuta tämä

    def use_value(self, card, trip_region):
        '''
        Maksetaan matka matkakortilta. Parametrina tulee korttiolio, josta
        maksu otetaan sekä tieto lipun alueesta (1 vai 2). Yrittää ottaa
        kortilta lipun hintaa vastaavan summan. Jos otto onnistui, lisätään
        kortille ostetun lipun alue ja ostohetki. Palautetaan teksti:
        Ticket bought, money left: x.xx
        x:t ovat lipulle jäänyt rahamäärä. Jos rahaa ei ollut tarpeeksi lippua
        varten, palautetaan: Buy ticket from the driver
        '''
        # Toteuta tämä


    def find_price(self, trip_region):
        '''
        Hakee lipun hinnan halutun alueen perusteella. Palauttaa lipun hinnan.
        '''
        if trip_region:
            for region, price in self.prices:
                if region == trip_region:
                    return price
```

```
def main():
    region_1 = 1 # kaupungin sisäinen
    region_2 = 2 # molempien kaupunkien alueella
    ticket_prices = ((region_1, 1.84), (region_2, 3.47))

    card1 = TravelCard('Teemu Teekkari')
    card2 = TravelCard('Brian Kottarainen')

    ticket_automat = TicketMachine()
    # datetime(vuosi, kk, päivä)
    ticket_automat.load_period(card1, ['Espoo'], datetime(2012, 3, 20))
    ticket_automat.load_period(card2, ['Espoo', 'Helsinki'],
                               datetime(2012, 4, 3))
    ticket_automat.load_value(card1, 4) # rahamäärä euroina
    ticket_automat.load_value(card2, 5) # rahamäärä euroina

    # luo kortinlukijoita. (pysäkin sijaintikaupunki, ajanhetki(vvvv,kk,p,t,m),
    # lippujen hinnat) ja rahasta liput
    bus_103 = CardReader('Espoo', datetime(2012, 3, 12, 16, 45), ticket_prices)
    print 'Teemu (Otaniemi-Keilaniemi):', bus_103.check_period(card1)
    print 'Brian (Otaniemi-Kamppi):', bus_103.check_period(card2)
    metro = CardReader('Helsinki', datetime(2012, 3, 12, 17, 20), ticket_prices)
    print 'Brian (Kamppi-Hakaniemi):', metro.check_period(card2)
    tram_3T = CardReader('Helsinki', datetime(2012, 3, 12, 19, 1),
                         ticket_prices)
    print 'Brian (Hakaniemi-Keskusta):', tram_3T.check_period(card2)
    print 'Teemu (Hakaniemi-Keskusta):', tram_3T.check_period(card1)
    print 'Teemu (Hakaniemi-Keskusta):', tram_3T.use_value(card1, region_1)
    bus_102T = CardReader('Helsinki', datetime(2012, 3, 12, 19, 21),
                          ticket_prices)
    print 'Brian (Kamppi-Olari):', bus_102T.check_period(card2)
    print 'Teemu (Kamppi-Otaniemi):', bus_102T.use_value(card1, region_2)

if __name__ == '__main__':
    main()
```

MALLIAJO:

```
Teemu (Otaniemi-Keilaniemi): OK: valid period found
Brian (Otaniemi-Kamppi): OK: valid period found
Brian (Kamppi-Hakaniemi): OK: valid period found
Brian (Hakaniemi-Keskusta): OK: valid period found
Teemu (Hakaniemi-Keskusta): Buy ticket
Teemu (Hakaniemi-Keskusta): Ticket bought, money left:2.16
Brian (Kamppi-Olari): OK: valid period found
Teemu (Kamppi-Otaniemi): Buy ticket from the driver
```

TEHTÄVÄN 2 KOODI:

```
from time import sleep

class Ant(object):
    directions = {}
    directions['N', 'left'] = (0, -1, 'W')
    directions['N', 'right'] = (0, 1, 'E')
    directions['E', 'left'] = (-1, 0, 'N')
    directions['E', 'right'] = (1, 0, 'S')
    directions['S', 'left'] = (0, 1, 'E')
    directions['S', 'right'] = (0, -1, 'W')
    directions['W', 'left'] = (1, 0, 'S')
    directions['W', 'right'] = (-1, 0, 'N')

    def __init__(self, territory, row, column, direction):
        self.territory = territory
        if not self.territory.is_inside(row, column):
            self.row = 0
            self.column = 0
        else:
            self.row = row
            self.column = column
        self.direction = direction

        self.territory.place_ant(row, column)

    def move(self):
        # Toteuta tämä


class Board(object):

    def __init__(self, rows, columns):
        self.rows = rows
        self.columns = columns

        self.board = [None] * rows
        for row in range(self.rows):
            self.board[row] = [None] * columns
            for column in range(self.columns):
                self.board[row][column] = {}
                self.board[row][column]['color'] = 0
                self.board[row][column]['ant'] = 0
        self.draw()

    def draw(self):
        print '--' * (self.columns + 2)
        for row in range(self.rows):
            print '|',
            for column in range(self.columns):
                if self.board[row][column]['ant']:
                    if self.board[row][column]['color']:
                        print '*',
                    else:
                        print 'X',
                elif self.board[row][column]['color']:
                    print '#',
                else:
                    print ' ',
            print '|'
        print '--' * (self.columns + 2)
        sleep(1)

    def get_color(self, row, column):
        return self.board[row][column]['color']

    def move_out(self, row, column):
        if self.board[row][column]['color']:
            self.board[row][column]['color'] = 0
        else:
            self.board[row][column]['color'] = 1
        self.board[row][column]['ant'] = 0

    def place_ant(self, row, column):
        self.board[row][column]['ant'] = 1
        self.draw()

    def is_inside(self, row, column):
        if row < 0 or row >= self.rows:
            return False
        if column < 0 or column >= self.columns:
            return False
        return True

if __name__ == '__main__':
    territory = Board(7, 7)
    ant = Ant(territory, 3, 3, 'N')
    ant.move()
```

MALLIAJO: