Tentti: T-106.1210 Ohjelmoinnin peruskurssi, osa 1

Tenttipäivä: 13.12. 2011

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 käsitellään matkatoimiston lentoja. Tutustu huolellisesti luokkiin Flight ja TravelAgency. Flight-luokka kuvaa yksittäistä lentoa, kohteesta toiseen. Luokka TravelAgency kuvaa matkatoimistoa, jossa on lentoja useitten eri kohteitten välillä. Kahden kohteen välillä voi olla useampia eri hintaisia ja eri numerolla olevia lentoja. Matkatoimistosta voi etsiä suoria lentoja annettujen paikkojen välillä ja etsiä halvimman suoran lennon annetulla välillä. Lisäksi koodista löytyy esimerkkejä luokkien käytöstä.

Toteuta luokan TravelAgency metodit search_flights ja search_cheapest_flight siten, että ne toimivat kuvatulla tavalla. Toteutustapa on muuten vapaa.

- Tutustu annettuihin luokkiin huolella, jotta niiden toiminta selviää. Muista myös esimerkkitapaukset.
- search_flights etsii kaikki suorat lennot parametrina annettujen paikkojen välillä ja palauttaa löytyneet lennot(Flight-oliot) listana (tyhjä lista, jos yhtään lentoa ei löydy)
- search_cheapest_flight etsii parametrina saatujen kohteitten väliltä halvimman suoran lennon (Flight-olio) ja palauttaa sen (None, jos yhtään lentoa ei ole ko. välillä)
- Tarvittavista listan metodeista on esimerkkejä annetussa koodissa. Muita ei pitäisi tarvita.

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

Tässäkin tehtävässä tutustut annettuun koodiin ja täydennät sitä. Tutustu luokkiin BankAccount ja Bank sekä testiohjelmaan ja sen testiajoon. BankAccount kuvaa pankkitiliä ja Bank pankkia, jossa voi olla useita erilaisia tilejä. Tehtävänäsi on luoda luokka CheckingAccount, joka kuvaa sekkitiliä. Tililuokan metodien lisäksi siinä on oltava metodit sekkien lunastamiseen sekä sekin tietojen kyselyyn. Lunastetut sekit täytyy varastoida jollain tavalla, jotta niitä pystyy tutkimaan myöhemmin. Luokka käyttää hyväkseen perintää.

- process_check saa parametreina sekin numeron, nostajan nimen ja summan
- Sekin lunastus tarkoittaa sekissä olevan summan nostamista tililtä ja sekin tietojen laittamista talteen.
- check_info saa parametrina katsottavan sekin numeron. Jos sekki on jo lunastettu se palauttaa kyseisen sekin tiedot listana([nostajan nimi, summa]) ja jos sekkiä ei ole lunastettu, palautuu teksti 'not cashed yet'

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.

Muista käydä täyttämässä palautelomake. Aikaa on 20.12. asti.

TEHTÄVÄN 1 KOODI

```
class Flight:
    def __init__(self, flight, origin, destination, price):
        self.flight = flight
        self.origin = origin
        self.destination = destination
        self.price = price

    def get_origin(self):
        return self.origin

    def get_destination(self):
        return self.destination

    def get_price(self):
        return self.price

    def __str__(self):
        return "{0}: {1}-{2}, price: {3}".format(self.flight, self.origin,
                                                 self.destination, self.price)

class TravelAgency:

    def __init__(self):
        self.flights = []

    def count_destinations(self):
        destinations = []
        for flight in self.flights:
            if flight.get_destination() not in destinations:
                destinations.append(flight.get_destination())
        return len(destinations)

    def add_flight(self, flight, origin, destination, price):
        self.flights.append(Flight(flight, origin, destination, price))

if __name__ == '__main__':
    otatours = TravelAgency()
    otatours.add_flight(1, 'helsinki', 'tukholma', 159)
    otatours.add_flight(2, 'helsinki', 'oulu', 99)
    otatours.add_flight(3, 'helsinki', 'riika', 199)
    otatours.add_flight(4, 'helsinki', 'tukholma', 342)
    otatours.add_flight(5, 'tampere', 'lontoo', 429)
    otatours.add_flight(6, 'tampere', 'helsinki', 49)
    otatours.add_flight(7, 'helsinki', 'lontoo', 249)
    otatours.add_flight(8, 'tampere', 'riika', 219)
    otatours.add_flight(9, 'tampere', 'tukholma', 69)
    otatours.add_flight(10, 'helsinki', 'lontoo', 499)

    print 'Flights from Helsinki to Lontoo:'
    for flight in otatours.search_flights('helsinki', 'lontoo'):
        print flight

    print '\nCheapest flight from helsinki to Lontoo:'
    print otatours.search_cheapest_flight('helsinki', 'lontoo')

    print '\nFlights from Helsinki to Timbuktu:'
    for flight in otatours.search_flights('helsinki', 'timbuktu'):
        print flight

    print '\nNumber of destinations in Otatours:'
    print otatours.count_destinations()
```

Ohjelman tulostus:

```
Flights from Helsinki to Lontoo:
7: helsinki-lontoo, price: 249
10: helsinki-lontoo, price: 499

Cheapest flight from helsinki to Lontoo:
7: helsinki-lontoo, price: 249

Flights from Helsinki to Timbuktu:

Number of destinations in Otatours:
5
```

TEHTÄVÄN 2 KOODI:

```
class BankAccount(object):
    account_number = 0


    def __init__(self, balance=0.0):
        self.balance = balance
        BankAccount.account_number += 1
        self.number = BankAccount.account_number #account number
        self.type = 'bank account'  #account type

    def get_number(self):
        return self.number

    def deposit(self, amount):
        self.balance += amount

    def withdraw(self, amount):
        if self.balance < amount:
            return 0.0
        else:
            self.balance -= amount
            return amount

    def get_balance(self):
        return self.balance

    def transfer(self, amount, to_account):
        if self.withdraw(amount):
            to_account.deposit(amount)
            return True
        else:
            return False

    def __str__(self):
        return '{0:d} {1:s} {2:.2f}'.format(self.number, self.type, self.balance)
```

```
class Bank(object):

    def __init__(self):
        self.accounts = []

    def new_bank_account(self, initial_balance):
        self.accounts.append(BankAccount(initial_balance))

    def new_checking_account(self, initial_balance):
        self.accounts.append(CheckingAccount(initial_balance))

    def remove_account(self, number):
        found = None
        for account in self.accounts:
            if account.get_number() == number:
                found = account
                break
        if found:
            self.accounts.remove(found)
            return True
        else:
            return False

    def list_accounts(self):
        accounts = ''
        for account in self.accounts:
            accounts += '{:s}\n'.format(account)
        return accounts

    def get_account(self, number):
        for account in self.accounts:
            if account.get_number() == number:
                return account
        return None
```

```
if __name__ == '__main__':
    otabank = Bank()
    otabank.new_bank_account(800)
    otabank.new_checking_account(1200)
    print otabank.list_accounts()
    otabank.get_account(1).deposit(200)
    print otabank.get_account(1).get_balance()
    print otabank.get_account(1).withdraw(300)
    print otabank.get_account(1).get_balance()
    otabank.get_account(1).transfer(50, otabank.get_account(2))
    print otabank.list_accounts()
    otabank.get_account(2).deposit(500)
    print otabank.get_account(2).get_balance()
    otabank.get_account(2).process_check(3001, 'donald duck', 40)
    otabank.get_account(2).process_check(2996, 'john clark', 300)
    otabank.get_account(2).process_check(3002, 'john wayne', 200)
    print otabank.get_account(2).get_balance()
    print otabank.get_account(2).check_info(2996)
    print otabank.get_account(2).check_info(3003)
```

Ohjelman tulostus:

```
1 bank account 800.00
2 checking account 1200.00

1000
300
700
1 bank account 650.00
2 checking account 1250.00

1750
1210
['john clark', 300]
not cashed yet
```