CS-A1121

Projektityön dokumentti
Ohjelmoinnin peruskurssi Y2

14.01.2021-31.05.2021

Edward Daka – 705952

Kauppatieteiden kandidaattitutkinto, vsk 2019. 04.05.2021

1. **Yleiskuvaus**

   Toteutin rahanhallinnan ohjelman Pythonissa. Ohjelma lukee tilitapahtumat tiedostosta, erottelee tulot ja menot toisistaan sekä tekee selkeän kuvaajan menojen jakautumisesta pylväsdiagrammilla.

   Toteutettavan ohjelman osaa automaattisesti laskea yhteen samaan kauppaan tehdyt ostokerrat. Käyttäjä voi halutessaan myös ryhmitellä kaupat ja poistamaan tämän ryhmittelyn.

   Alustava tarkoitus oli tehdä *Keskivaikea*-versio sovelluksesta. Tämä ei rajallisen ajan aiheuttamista haasteista johtuen kuitenkaan onnistunut. Minulla oli haasteita luoda graafinen käyttöliittymä, joten päädyin lopulta toteuttamaan *Helpon*-version. Olennaisimpana erona on graafisen käyttöliittymän puuttuminen ja kuvaaja on alkuperäisestä suunnitelmasta poiketen pylväsdiagrammi, eikä piirakkadiagrammi.

2. **Käyttöohje**

   Ohjelma käynnistetään PyCharmissa menemällä ”main.py” tiedostoon ja painamalla **Run.** Tämän jälkeen tekstipohjainen käyttöliittymä tuottaa valikon, jossa on viisi eri toiminnallisuutta. Valikosta tehdään valinta syöttämällä numero 1-5.

   1. *Show me my Incomes* tuottaa tuloista kuvaajan. Mikäli käyttäjä on jo luonut ryhmittelyn, antaa käyttöliittymä vaihtoehdon näyttää kuvaaja ryhmiteltynä tai ilman syöttämällä y tai n (yes tai no). Muussa tapauksessa ohjelma tulostaa kuvaajan tuloista, jossa prosentuaalista määrää kuvataan pilareilla, jotka muodostuvat ”#”-symboleista.

   2. *Show me my Expenses* toimii täysin vastaavalla tavalla kuin 1. kohdan *Show me my Incomes*, mutta luo kuvaajan, jossa on esitetty menot.

   3. *How much more can I earn?* -valinnalla käyttäjä saa näkyviin KELA-laskurin. Laskuri on projektin yksilöivä piirre ja sen tarkoituksena on kertoa käyttäjälle, kuinka kauan hän voi tienata nykyisellä tulotasollaan, kunnes hänen täytyy ilmoittaa KELA:lle opintotukirajan ylittymisestä. Tässä käyttäjän on annettava erikseen kaksi syötettä, jotka kysyvät jo tienatun summan ja tukikuukaudet.

   4. *Group transactions* -toiminnolla voidaan ryhmitellä tilitapahtumia. Toiminto kysyy muodostetun ryhmän nimen. Tämän jälkeen se kysyy, onko kyseessä tulojen vai menojen ryhmittely. Tällä haetaan oikea listaus summista. Kun transaktion tyyppi tiedetään, voidaan syöttää tapahtumien nimiä, kunnes syötetään ”STOP”. Mikäli transaktion nimeä ei löydy listasta, saa käyttäjä tästä ilmoituksen.

   5. *Quit* -toiminnolla lopetetaan ohjelma. Ohjelma tulostaa poistuessa viestin, joka kiittää käyttäjää testauksesta.

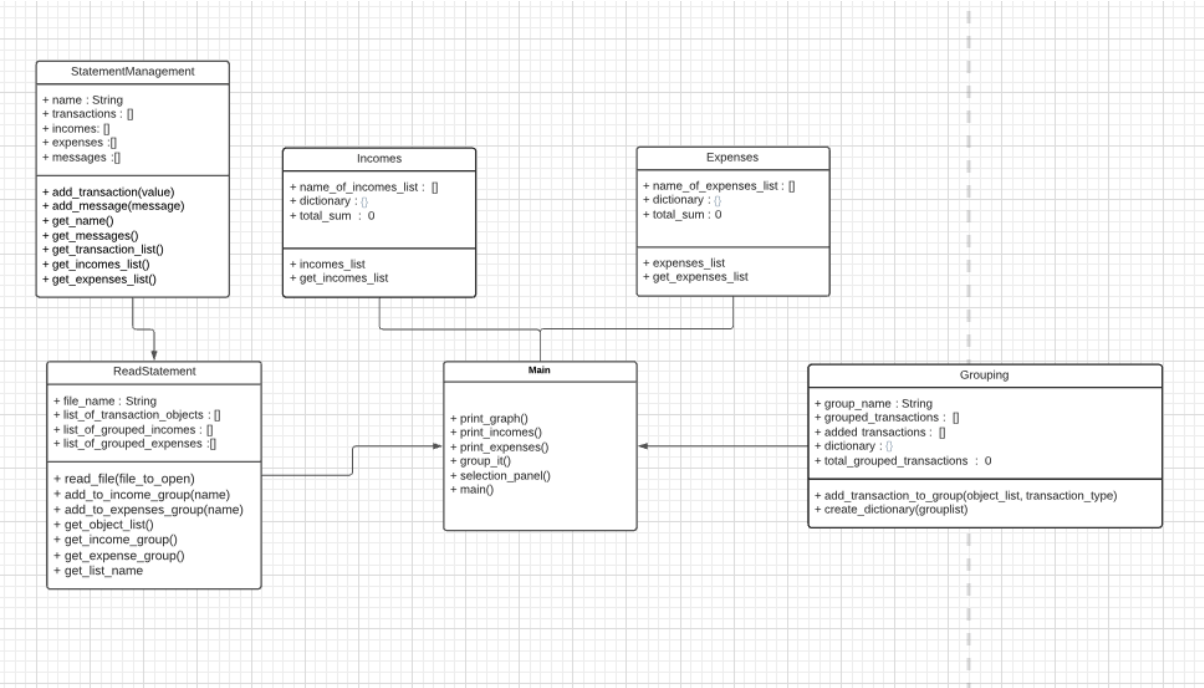
3. **Ulkoiset kirjastot**

Olen hyödyntänyt projektissa vain Pythonin standard-kirjastoa, ja sieltä tarkemmin ottaen csv. ja random -kirjastoja. Niillä on käsitelty tiedoston lukemista.

4. **Ohjelman rakenne**

Toteutunut ohjelma noudattaa pitkälti projektisuunnitelmaani. Ohjelmiston keskeisimmät osat ovat main.py ja luokat, joilla käsitellään tilitapahtumia sisältävää csv. -tiedostoa.

Luokkia ohjelmassa ovat *readstatement, statementmanagement, incomes, expenses ja grouping.*



Readstatement mallintaa tiedoston lukemista. Se luo valitusta tiedostosta olion, luoden sille listan tiedoston sisältäväistä transaktioista, sekä mahdollisista tulo- ja menoryhmistä.

- *read_file* lukee tiedoston ja lisää tietoja *StatementManagementin* luomista transaktio-olioista. *Read_file* käsittelee myös transaktion lisäämän viestin. Esimerkiksi mikäli viestinä on ”Oma siirto” ei tätä lasketa tuloihin.
- *add_to_income_group* lisää olion *list_of_grouped_incomes* -listaan, jonne kootaan kaikki tuloihin luokiteltavat ryhmät.
- *add_to expenses_group* tekee vastaavan toimenpiteen, mutta menojen osalta.
- *get_object_list* palauttaa listan transaktio-olioista. Tätä on hyödynnetty ohjelmassa käymällä listaa läpi ja hakemalla sieltä eri toimintoihin olennaisia olioita.
- *get_income_group* palauttaa list_of_grouped_incomes listan.

- *get_expenses_group* tekee vastaavan toimenpiteen, mutta menojen osalta.
- *get_list_name* palauttaa oliotiedoston nimen *file_name.*

Statementmanagement hoitaa yksittäisen tilitapahtuman tietojen käsittelyn. Se luo tilitapahtumalle olion sen nimen perusteella. Se siis ryhmittelee esimerkiksi K-Market Jätkäsaaren alle kaikki sen nimiset tilitapahtumat. Se luo tilitapahtuman oliolle, tiedot siihen sisältyvistä tuloista ja menoista, sekä sen sisältämästä viestistä.

Luokasta on mahdollisuus noutaa eri get_ funktioilla tilitapahtuman tietoja, kuten edellä mainitut olioon kuuluvat tulot, menot ja viestit.

- *add_transaction* lisää transaktion listaan. Se ottaa huomioon summan perusteella onko kyseessä tulo vai meno. Se huomioi myös onko kyseessä sisäinen siirto, tarkastamalla onko transaktion viestinä ”Oma siirto.”.
- *add_message* lisää viestin transaktion viestilistaan *messages*.
- *get_name* palauttaa transaktion nimen, esim. K-Market Jätkäsaari.
- *get_messages* palauttaa transaktion viestilistan *messages*.
- *get_transactions_list* palauttaa transaktioiden listan *transactions* eli samannimisen tapahtuman summat.
- *get_incomes_list* palauttaa listan transaktioon liittyvistä kuluista eli *incomes*.
- *get_expenses_list* palauttaa listan transaktioon liittyvistä menoista eli *expenses*.

Incomes käsittelee tilitapahtuman tuloja. Se muodostaa tuloista tulostusta varten sanakirjan.

- *incomes_list* luo sanakirjan annetun transaktiolistan perusteella. Se käy läpi listalla olevat transaktiot ja lisää niiden nimet ja tätä vastaavat summat sanakirjaan. Lopuksi se palauttaa luodun sanakirjan.
- *get_total_incomes* palauttaa kokonaissumman transaktioista.

Expenses käsittelee tilitapahtuman menoja. Se muodostaa menoista sanakirjan, ja palauttaa sen main.py:lle.

- *expenses_list* toimii vastaavalla tavalla kuin *incomes_list*. Se luo sanakirjan sille annetun transaktiolistan perusteella.

Grouping hoitaa tapahtumien ryhmittelyn. Se ottaa huomioon ollaanko ryhmittelemässä tuloja vai menoja. Se palauttaa ryhmitellyn tiedon main.py:lle.

- *add_transaction_to_group* lisää transaktion ryhmään. Se tarkistaa onko kyseessä meno vai tulo ja ilmoittaa mikäli käyttäjän antamaa transaktioita ei löydy transaktiolistalta.

- *create_dictionary* luo sanakirjan ryhmien olioista, lisäten sanakirjaan niiden nimen ja nimeä vastaavan summan.

Main.py suorittaa ohjelman käytännön toiminnot ja luo tekstipohjaisen käyttöliittymän. Main.py sisältää seitsemän eri funktiota: *main, selection_panel, print_graph, print_incomes, print_expenses, group_it* ja *kelacalculator*. Main.py kutsuu luokkia ja saa palautteena luokkien tuottamia tietorakenteita.

- *Main* -funktio suorittaa ohjelman aloituksen, toivottaa käyttäjän tervetulleeksi ja pyytää avattavan tiedoston nimeä. Se antaa virheviestin, mikäli tiedosto on väärän niminen. Tämän jälkeen se suorittaa valintakomentoja. Se myös hoitaa ohjelman lopetuksen ja tulostaa lopetusviestin.
- *Selection_panel* tulostaa valintapaneelin ja tarkistaa että annettu valinta on oikeanlainen. Lopuksi se palauttaa käyttäjän tekemän valinnan *Main* -funktion suoritettavaksi.
- *print_graph* tulostaa tekstipohjaisen kuvaajan. Se laskee ja lajittelee sanakirjan tiedot suuruusjärjestykseen ja hoitaa laskutoimitukset, tulostaen niistä saadut tulokset.
- *print_incomes* hyödyntää *incomes* -luokkaa, luoden siihen olion ja samalla se luo tulostuksessa hyödynnettävän sanakirjan. Luokka huomioi onko käyttäjä jo luonut ryhmittelyn ja tätä kautta antaa vaihtoehdon valita joko ryhmitellyn tai ryhmittelemättömän.
- *print_expenses* toimii täysin vastaavalla tavalla kuin *print_incomes*.
- *group_it* luo käyttäjän haluaman ryhmittelyn. Se pyytää tiedot ryhmän nimestä, transaktion tyypistä ja transaktion nimestä.
- *Kelacalculator* laskee nykyisen tulotason perusteella paljonko käyttäjä voi ansaita tiettyjen tukikuukausien ja jo ansaitun summan perusteella.

Vaihtoehtoiset mallit

1. *Print_incomes* ja *Print_expenses* voitaisiin yhdistää yhdeksi funktioksi. Tällöin poistettaisiin päällekkäisyyksiä, sillä funktiot toimivat lähes samalla tavalla. Yhdellä funktiolla saavutettaisiin siistimpi ja yksinkertaisempi koodi.
2. *Print_incomesin ja Print_expensesin* sisällä voisi yhdistää tulojen ja menojen oliomuodostusta yhdeksi funktioksi. Näissä on ylimääräistä toistoa, joka voisi olla yhdessä funktiossa.

5. **Algoritmit**

Rahanseurantasovellus ei ole vaatinut monimutkaisten algoritmien rakentamista. Matemaattinen laskenta perustuu pitkälti peruslaskutoimituksiin, kuten kerto-, jako-, yhteen- ja vähennyslaskuun.

Nämä ovat erityisen tärkeitä, kun luodaan tekstipohjaista kuvaajaa tuloista. Tällöin tarvitaan tieto siitä, montako prosenttia kyseinen tilitapahtuma on kokonaistuloista ja menoista. Tätä kautta voidaan ladata oikea määrä kuvaajan palkkeja esimerkiksi **Tulostukseen käytetystä algoritmista.**

Aloitetaan käymällä läpi jokainen sanakirjaan lisätty avain. Jokaisen sanakirjan avaimen kohdalla, otetaan sitä vastaava arvo. Aloitetaan laskemalla kuinka ison osan tehtävä kuvaaja muodostaa kokonaissummasta.

$$\boldsymbol{percentage\ of\ whole}(\%) = \frac{\boldsymbol{amount\ of the\ transaction}}{\boldsymbol{total\ amount\ of\ incomes\ or\ expenses}} * \mathbf{100}$$

Tästä voidaan sen jälkeen laskea, paljonko palkkeja tulostetaan. Yksi prosentti vastaa puolta palkkia. Päädyin tähän testattuani erilaisia palkkimääriä. Liian suuri palkkimäärä esim. yksi palkki = yksi prosentti, olisi tuottanut keskimääräiseen tiliotetapahtuma tiedostoon turhan suuren variaation. Monissa tilitapahtumissa etenkin menot ovat jakautuneet tasaisesti. Tulot voivat olla jakautuneet erittäin epätasaisesti, palkan ollessa suurin tulon lähde.

Mikäli olisi valinnut pienemmän palkkimäärän, ei kuvaaja olisi ollut liiemmin hyödyllinen, sillä 12:ta prosenttia ja 16:ta prosenttia voisi kuvastaa sama palkkimäärä. Testausten tuloksena puoli palkkia yhtä prosenttia kuvastamassa oli paras tasapaino.

$$\boldsymbol{percentage\ of\ pillars} = \boldsymbol{round}(\frac{\boldsymbol{percentage\ of\ whole}}{\mathbf{2}})$$

Lopuksi tämä osuus kerrotaan *print* -toiminnossa pilarin symbolilla ”#”, joka tulostaa tarvittavat pilarit.

Tiedoston lukeminen :

```
Tiedoston lukeminen :

1. for infomation in csv file
   name = valitsee nimen riviltä 5
   if name not in listaus jo lisätyistä nimistä
      names.append(nimi)

      2. luodaan tilitapahtumalle oli ja lisätään se olio_listaan

      3.  messages = valitsee viestin riviltä 8
          messages_split = Muokkaa viestin oikeaan muotoon
          Jos viesti on tyhjä se ohitetaan, muuten se lisätään tilitapahtumaan

      4. Kun nimi ja viesti on luotu listätään summa
          amount = valitsee summan riviltä 2

      5  Lisää summan listaan ja lisää sen transaktioon.
          self.list_of_transaction_objects[len(self.list_of_transaction_objects) - 1].add_transaction(
                          amount)
      6. Jos lisäys epäonnistuu heittää ohjelma virheen
```

Transaktion lisääminen :

```
Transaktion lisääminen :

1. transactions.lisää(summa) // Lisätään arvo transaktiolistaan

2. Tarkistetaan onko transaktiolla jo viestiä if len(self.__messages) == tyhjä = 0, jos ei niin lisätään arvon perusteella tuloihin tai menoihin.

3. Tarkistetaan viesti self.__viestinsisältö[len(self.__viestinsisältö) - jonosta yksi aiempi = 1] == "Omasiirto", jos on Omasiirto niin ohitetaan transaktio.
```

Transaktion ryhmittely :

```
Transaktion ryhmittely :

1. Kunnes käyttäjä syöttää "STOP" niin jatketaan transaktion kysymistä

2. Tarkistetaan onko syötetty transaktio, jo listalla if name == lisätty transaktio:
    2.2 Jos on niin lisätään lisättyjen transaktioiden listalle added.transactions.lisää(nimi)
    2.3 Tarkistetaan onko lisättävä tapahtuma tulo vai meno, ja haetaan sitä kautta listalle
            transaction_type = TOSI TAI EPÄTOSI
    2.4 Lisätään transaktio ryhmiteltyihin transaktioihin
```

Sanakirjaan ryhmittely :

```
Sanakirjaan ryhmittely :

    1. Hakee transaktion tulojen/menojen listan ja lukee sen
          for tulo/meno in tulojen/menojen lista
              sum = income + sum
          Lisätään self.totaalisummaan = summa
       2. Lisätään loopin lopussa tulevat tilitapahtumat tapahtuman nimenä olevan avaimen vastineeksi.
              self.dictionary[nimi] = sum
       3. Palautetaan luotu sanakirja
              return.luotusanakirja
```

6. Tietorakenteet

Keskeiset tietorakenteen joihin tietoa varastoitiin ja joiden avulla tietoa käsiteltiin, olivat sanakirjat ja listat. Niiden käyttö oli mielestäni luonteva valinta käsitellä tilitapahtumia, sillä oliot luotiin nimen perusteella ja saman nimiset tilitapahtumat voitiin niputtaa saman olion alle ja summat kerätä tätä kautta yhteen listaan.

Sanakirjaa hyödynnettiin kuvaajan tulostuksessa. Sanakirja toimii ohjelmassa periaatteessa kahden listan yhdistelmänä; se yhdistää tilitapahtuman nimen ja sitä vastaavan summan. Tällä tavalla on helppo poimia tulostettavan tiedon nimi ja summa.

Kelacalculator -funktiossa käytin sanakirjaa varastoimaan muuttumatonta tietoa, eli Kelan sivuilta poimitut tukikuukaudet ja niitä vastaavat tulotasot. Tämä auttoi saamaan helposti tiedon paljonko tietyllä tukikuukausien määrällä voi tienata.

Listojen käytön sijaan olisi voinut käyttää pelkkää sanakirjaa, johon tallentaisi tiedot esim. olioihin kuuluvista tuloista. Tällä olisi voinut vähentää listojen määrää, yhdistellen olennaisia tietoja yhteen rakenteeseen.

7. Tiedostot

Ohjelman toiminto perustuu pitkälti csv. -tiedoston lukemiseen. Tarkemmin ottaen Osuuspankin tilitapahtumatiedoista luotuun csv. -tiedostoon. Tiedosto koostuu kymmenestä sarakkeesta, joita ovat *Kirjauspaiva;Arvopaiva;Maara Euroa;Laji;Selitys;Saaja/Maksaja;Saajan tilinumero ja pankin BIC;Viite;Viesti;Arkistointitunnus*. Kuten edellä olevasta listauksesta näkee, erotellaan csv. -tiedoston tiedot toisistaan ”;” merkillä.

Ohjelma lukee tiedoston csv. -kirjaston csv.reader -toiminnon avulla ja valikoi oikeat rivit row[] -toiminnolla. Valtaosa sarakkeista on ohjelman toiminnon kannalta epärelevantteja. Halutut sarakkeet ovat *Maara Euroa,Saaja/Maksaja ja Viesti*.

Ohjelman toimintaa on testattu Tilitapahtumat.csv nimisellä tiedostolla, joka sisältää omasta Osuuspankin verkkopankistani noudetun tilitapahtumatiedoston.

Tätä tiedostoa tulisi hyödyntää ohjelman testauksessa.

8. **Testaus**

Jaoin suunnitelmassani testauksen kolmeen keskeiseen vaiheeseen, joilla oli mielestäni tärkein merkitys suunnitelman toteutuksen kannalta.

1. Tiliotteen lukeminen
    - Aluksi rakensin *try* ja *except* -toiminnot valitessani luettavaa tiedostoa. Lisäsin myös herjan siitä, että mikäli tiedosto ei pääty ”csv” kirjaimiin niin tästä huomautetaan.
    - *Debug* -toiminnon avulla tutkin ja testailin miten *readstatement* tiedosto kerää tietoja eri listoihin.
2. Summien erotteleminen tuloiksi ja menoiksi
    - Tämä oli myös kriittistä ohjelman toiminnallisuuden kannalta, sillä näin varmistettiin, ettei luokkiin tule ylimääräisiä tietoja, eikä niistä myöskään puutu oleellisia tietoja. Molemmissa luokissa tuotetaan oleelliset sanakirjat, joita ilman tekstipohjaista kuvaajaa on vaikea tuottaa.
    - Tätä testasin myös *debug* -toiminnolla ja muokkaamalla tiedoston summia ja nimikkeitä ”Tilitapahtumat.csv”-tiedostossa ja katsoen miten ohjelma reagoi muutoksiin.
3. Ryhmittelyn toiminta
    - Tätä testailin luomalla nimikkeitä ja koittamalla yhdistää tilitietoja uusiksi tilitiedoiksi. Ryhmittelyn haasteena oli miettiä mikä on paras tapa luoda luetuista tilitapahtumista uusi tilitapahtuma. Tässä päätin luodan oman *grouping*olion, muodostetuille ryhmille, eli kyseessä ei olisi *statementmanagement* tilitapahtuma. Tämä oli mielestäni helpoin tapa erotella ryhmät ja yksittäiset tilitapahtumat.
    - Ajoin erilaisia ryhmiä *debuggerin* läpi, ja katsoin miten ryhmien tiedot muodostuvat olioihin ja miten niistä saisi helpoiten muodostettua uuden tekstipohjaisen kuvaajan, joka huomioi ryhmittelyn, tai joka ei huomioi ryhmittelyä.

Testauksen ytimessä oli miettiä ”Mitä jos?”. Mitä jos käyttäjä syöttää numeron sijaan kirjanjonon? Mitä jos käyttäjä valitsee liian suuren tai pienen numeron? Mitä jos käyttäjä ylipäätään ei aja ohjelmaa oletetulla tavalla?

Näitä varten ohjelmaan on muodostettu erilaisia tarkistimia, jotka tarkistavat käyttäjän syötteen ja varmistavat, että syöte on yhdenmukainen haluttujen ehtojen kanssa. Mikäli syöte on virheellinen, ohjelma tulostaa auttavan virheviestin, jolla pyritään auttamaan käyttäjää syöttämään ehtojen mukainen syöte.

9. **Ohjelman tunnetut puutteet ja viat.**

    1. Ohjelma ei huomioi, mikäli ryhmittelyssä syötettävät tulot loppuvat.

        - Mikäli siis syöttää kaikki tulot valmiiksi yhteen, ei ohjelma osaa huomioida, että tilitapahtumia ei enää ole ja täten automaattisesti pysäyttää *grouping* -toiminnon.

        - Tähän voisi lisätä tarkisteen, joka katsoisi onko ryhmässä jo kaikki menot tai tulot, jota tiedostossa on.

    2. Mikäli tilitapahtuma on jo ryhmitelty, ei ohjelma ymmärrä varoittaa tästä lisäysvaiheessa.

        - Lisäsin syötteen, joka ilmoittaa tulostusvaiheessa, että näin on päässyt tapahtumaan. Löysin vian sen verran myöhään, etten ehtinyt koodata tähän toimivaa korjausta.

        - Mahdollinen korjaus sisältäisi tarkisteen, joka tarkistaa jo tehdyistä ryhmistä ja työn alla olevasta ryhmästä, löytyykö tapahtuma mahdollisesti niistä.

        - Ryhmittely antaa myöskin syöttää ryhmän nimeksi tyhjän, tämän voisi myös korjata tarkisteella.

    3. Järjestelmä ei huomioi *Kelacalculator* -funktiossa, millaista tuloa tuloissa on.

        - Ohjelma laskee tulorajoihin huomioitaviksi tuloiksi kaikki, jotka ovat tulot sanakirjassa.

        - Tähänkin voisi lisätä tarkisteen, joka suodattaisi muut tulot, paitsi sellaiset, joiden viestissä on ”Palkka”.

    4. Syöttäessä kirjainta huomioi onko kyseessä iso/pieni kirjain.

        - Syötteen on oletava täysin samassa muodossa, kuin mitä pyydetään tai ohjelma ei tunnista syötettä oikeaksi.

        - Tähän voisi käyttää muunninta kuten lower() tai upper(), jolla muuttaisi käyttäjän syötteen tai tilitapahtumat yhdenmukaiseen muotoon. Tulostusta varten voisi käyttää muuntimia saadakseen ulkoasun samanlaiseksi.

10. **3 parasta ja 3 heikointa kohtaa**

    Parhaat:

1. Pääsääntöisesti ohjelma toimii ja siitä tulostettu tekstipohjainen kuvaaja on selkeä ja antaa hyvän kuvan tulojen ja menojen jakautumisesta.
2. Tarkisteet antavat mahdollisuuden jatkaa käyttöä virheellisestä syötteestä huolimatta. Tällä tarkoitan, ettei esimerkiksi koko ryhmittelyprosessia täydy aloittaa alusta, vaikka syöttää keskikohdassa ohjelmaa virheen. Ohjelma myös pyrkii välttämään turhien syötteiden antamisen. Esimerkiksi mikäli henkilö ei ole tehnyt ryhmittelyä, ei ohjelma edes ehdota tätä, kun luodaan kuvaajaa tuloista tai menoista. Vasta kun henkilö on tehnyt ryhmittelyn, voi hän valita näytetäänkö tiedosto ryhmiteltynä vai ryhmittelemättömänä.
3. Koodi on pyritty pitämään helppolukuisena ja yksinkertaisena. Toiminnot on jaoteltu selkeästi eri luokkiin ja tarvittaviin funktioihin. Koodirivit on ryhmitelty jättäen välejä, jotka helpottavat koodin ymmärtämistä ja luovat jaottelun siitä, mikä osa tekee mitäkin.

Heikkoudet:

1. Main.py:ssä, joudutaan luomaan *Incomes* ja *Expenses* oliot. Tämä loi koodiin toisteisuutta, kun tehdään valinta siitä, luodaanko kuvaaja ryhmiteltynä vai ryhmittelemättömänä. En oikein keksinyt tapaa välttää tätä. Yksi mahdollisuus olisi luoda main.py:hyn funktio, joka loisi oliot. Tätä funktiota voitaisiin aina kutsua tarvittaessa, kun luodaan tuloista tai menoista kuvaaja.
2. CSV. -tiedoston oikein lukeminen voi olla vaikeaa, mikäli tiedosto on muussa muodossa kuin Osuuspankilla. Koska minulla on rajallinen määrä tilitapahtumatiedostoja, on vaikeaa testata lukemisen onnistuvuutta. Osuuspankilla tieto on aina tietyllä sarakkeella, joten mikäli sarakkeiden paikkaa muuttaa, muuttuu myös luettava tieto.
3. Vioissa mainitut pienet puutteet, jotka johtuvat tarkistimien puutteesta. Juuri tarkistimissa mietin olisiko ollut jotain keskitetty ratkaisua, jolla olisi voinut luoda valtaosan tarkisteista. Jouduin ohjelmassa turvautumaan paljon ifiin ja while loopiin, mutta jäin lopussa pohtimaan olisiko nämäkin voinut keskittää jotenkin.

   Ylipäätään nuo viat ja puutteet osion kohdat voisi kaikki merkitä projektin heikkouksiksi.

11. Poikkeamat suunnitelmasta ja toteutunut työjärjestys ja aikataulu

Alkuperäinen suunnitelma, 26.2.2021

Alkuperäinen suunnitelmani oli jaettu kolmeen vaiheeseen:

Aloitus 26.2.2021 – 26.3.2021(Käytetty aika n. 30 h)

Alkuperäinen suunnitelmani oli toteuttaa *keskivaikea* -toteutus, johon liittyy graafinen käyttöliittymä PyQt5:sella. Kuten aiemmin todettu tämä on ohjelman suurin poikkeus alkuperäisestä suunnitelmasta.

Huomasin PyQt:n käytön odotettua haastavammaksi, ja käyttöliittymän luonnin opettelu vaati huomattavasti aikaa ja panostusta. Sattumalta juuri 3. periodin aikana minulle oli kertynyt kuusi kurssia päällekkäin. En aikarajoitteesta johtuen ehtinyt perehtymään tarpeeksi graafisen käyttöliittymän luontiin, joten ensimmäiseen checkpointiin 26.3.2021 päätin aloittaa *helpon* -vaatimusten tekemisen. Sain viikolla 10 tehtyä olennaisimmat osat *helpon* -vaatimuksesta ja viimeistelin näitä palautusta varten.

Checkpointiin mennessä sain luotua tekstipohjaisen version ohjelmasta. Ohjelma kykeni lukemaan tiedoston ja luomaan tekstipohjaisen kuvaajan tuloista ja menoista. Nämä ominaisuudet loivat keskeisen osan ohjelman vaatimuksista.

Keskivaihe 26.3.2021 – 26.4.2021 (Käytetty aika n. 34 h)

Keskivaiheen tavoitteet olivat saada koodi valmiiksi ja varmistaa että ohjelma täyttää vähintään *helpon* -vaatimukset. Lopuksi tavoite oli tarkastella graafisen käyttöliittymän mahdollista toteutusta ennen lopetusvaihetta.

Viikolla 13 parantelin keskivaiheessa tulostusta ja loin tarkisteita, jotka varmistivat sujuvan tiedoston lukemisen ja kuvaajan tulostamisen. Tämän lisäksi loin *Kelacalculator* -funktion, joka oli projektini yksilöivä ominaisuus, sekä loin ryhmittelyn tekevän luokan *Grouping*.

Aikani ei enää viikon 16 lopussa riittänyt perehtymään graafisen käyttöliittymän toteutukseen, joten hylkäsin sen implementoinnin ohjelmaan.

Lopetus 26.4.2021 – 7.5.2021 (Käytetty aika n. 8 h):

Loppuosiossa pääasiassa testailin debugilla ja eri ajoilla ohjelman toimintoja. Tätä kirjoittaessani korjasin vielä useita pieniä vikoja ja parantelin ohjelmaa aktiivisesti varmistaen sujuvamman testauksen.

12. Arvio lopputuloksesta

Olen kokonaisuudessaan lopputulokseen tyytyväinen. Mielestäni ohjelma saavuttaa olennaiset tavoitteet, joita projektilta on haluttu, etenkin *helpon* -vaatimusten osalta.

Ohjelman luominen on ollut mielekästä ja erilaisten vaihtoehtojen sekä toteutustapojen pohtiminen on kehittänyt ohjelmistokehitystaitojani.

Ohjelmiston kriittisin osa on tiedoston lukeminen. Ohjelman luonnissa on tehty tiettyjä oletuksia tiedoston muodon suhteen. Tässä on huomioitava, että mikäli luettavaa tiedostoa muokkaa tarpeeksi, se saattaa ajaa lukemisen virheeseen tai tuottaa kuvaajan väärällä tavalla. Eli niin sanotusti ”*Garbage in, grabage out*”. Kehitetympi antaa paremmat virheviestit, mikäli tiedostossa on todella monessa rivissä virheitä.

Tällä hetkellä tiedosto käsittelee vain yksittäiset virheet tulostamalla niistä virheviestin. Ohjelmaa voisi jatkokehittää suuntaan, joka antaa esim. numeerisen palautteen paljonko virheitä on. Tähän voi hyödyntää yksinkertaista i+= laskuria.

Toinen jatkokehityksen kohde on *Kelacalculator* -toiminto niputtaa tulot yhteen, joten mikäli siellä on esim. Mobilepay maksuja, lasketaan myös nämä tuloihin mukaan, vaikka näillä ei ole todellista vaikutusta tuloihin. Tähän voisi asettaa tarkistimen, joka ei huomioisi muita tilitapahtumia, kuin esim. ”Palkka” nimisellä viestillä olevat, valitettavasti aikarajoitteista johtuen tämän implementoiti jäi ulos projektista.

Myös ryhmittelyä voisi parantaa, kuten aiemmissa osioissa onkin mainittu. Ohjelmaan ei ole tehty tarkistetta, jonka vuoksi mikäli henkilön tilitapahtumassa ei ole tuloja tai menoja ja henkilö lähtee ryhmittelemään tuloja, on tätä varten tehty tarkistin kuvaajaa luotaessa, mutta tätä ryhmittelyä ei kuitenkaan estetä. Oletusarvona on, että henkilö ensin tarkistaa tulot ja menot ja vasta sitten ryhmittelee. Tällöin käyttäjä ei lähtisi ryhmittelemään tuloja tai menoja, joita ei ole ollenkaan.

Ohjelman rakenne tekee laajentamisesta mahdollista. Ohjelmaa on koitettu pitää mahdollisimman yksinkertaisena, jolloin uusien luokkien tai funktioiden integrointi onnistuu. Etenkin *readstatement* ja *statementmanagement* on mahdollista kutsua kattavasti erilaisia tietoja tilitapahtumista ja luettavasta tiedostosta, joka helpottaa jo olemassa olevien tietojen hyödyntämistä uusien ominaisuuksien luonnissa.

13. Viitteet

The Python Standard Library — Python 3.9.2 documentation. (2021). Retrieved 26 February 2021, from https://docs.python.org/3/library/

PyQt5 Reference Guide — PyQt v5.15 Reference Guide. (2021). Retrieved 26 February 2021, from https://www.riverbankcomputing.com/static/Docs/PyQt5/

Matthes, E. (2020). *Python Crash Course, 2nd Edition* (2nd ed.). San Francisco: No Starch Press Inc.

PyQt5 tutorial 2021 — Create GUI applications with Python and Qt. (2021). Retrieved 26 February 2021, from https://www.learnpyqt.com

Kurssimateriaali Y2. (2021). Retrieved 26 February 2021, from https://plus.cs.aalto.fi/y2/2021/toc/

1. Tietokoneista ja ohjelmista — Ohjelmoinnin peruskurssi Y1 0.0.1 documentation. (2021). Retrieved 26 February 2021, from https://grader.cs.aalto.fi/static/y1/moniste/tietokoneista_ja_ohjelmista.html

14. **Liitteet**

Tulojen ja menojen tulostus :

```
Welcome to the money tracker !

Enter the name of the statement file:
Tilitapahtumat.csv

What information would you like to see ?
1. Show me my Incomes
2. Show me my Expenses
3. How much more can I earn ?
4. Group transactions ?
5. Quit

Insert the number for the corresponding information:
1

Names of giver:                          | Percentages

Accountor Finago                         | ###################################      788.0 € (69.67%)
KELA/FPA                                 | #############                            252.8 € (25.16%)
MobilePay: Emmi Suominen                 | ###                                      52.0 € (5.18%)

What information would you like to see ?
1. Show me my Incomes
2. Show me my Expenses
3. How much more can I earn ?
4. Group transactions ?
5. Quit

Insert the number for the corresponding information:
2

Names of receivers:                      | Percentages

Saunalahti                               | ############                             -86.2 € (23.87%)
K-Citymarket Ruoholahti                  | ##########                               -72.9 € (20.18%)
Verkkokauppa.com                         | ######                                   -39.9 € (11.05%)
MobilePay: Emmi Suominen                 | #####                                    -36.0 € (9.97%)
K-Market Jätkäsaari                      | ####                                     -28.9 € (8.01%)
K-Market Ruoholahti                      | ###                                      -25.2 € (6.97%)
Lakrids by Johan Bülow                   | ##                                       -15.0 € (4.15%)
MobilePay: VR-Yhtymä                     | ##                                       -12.0 € (3.32%)
Raan Khao Gaeng                          | ##                                       -11.9 € (3.29%)
Spotify                                  | #                                        -10.0 € (2.77%)
Tokmanni Ruoholahti                      | #                                        -7.7 € (2.13%)
MobilePay: Johanna Daka                  | #                                        -6.0 € (1.66%)
MobilePay: Anni Koskinen                 | #                                        -4.5 € (1.25%)
Sale Kaskenkatu                          |                                          -2.6 € (0.72%)
Parkman                                  |                                          -2.4 € (0.67%)

What information would you like to see ?
1. Show me my Incomes
2. Show me my Expenses
3. How much more can I earn ?
4. Group transactions ?
5. Quit

Insert the number for the corresponding information:
```

Tilitapahtumien ryhmittely ja sen tulostus :

```
Insert the number for the corresponding information:
4

In this part of the program, you can group transactions.
What is the name of your group ?
Palkka + KELA
Are the added transactions expenses or incomes ?
incomes
You may finish adding by typing 'STOP'
What transaction would you like to add ?
Accountor Finago
What transaction would you like to add ?
KELA/FPA
What transaction would you like to add ?
STOP
Would you like to make more groups ? (y/n)
n

What information would you like to see ?
1. Show me my Incomes
2. Show me my Expenses
3. How much more can I earn ?
4. Group transactions ?
5. Quit

Insert the number for the corresponding information:
1

Do you want to show the transaction grouped or not ? (y/n)
y
Names of giver:                          | Percentages

Palkka + KELA                            | ################################################   952.8 € (94.82%)
MobilePay: Emmi Suominen                 | ###                                                52.8 € (5.18%)

What information would you like to see ?
1. Show me my Incomes
2. Show me my Expenses
3. How much more can I earn ?
4. Group transactions ?
5. Quit

Insert the number for the corresponding information:
1

Do you want to show the transaction grouped or not ? (y/n)
n
Names of giver:                          | Percentages

Accountor Finago                         | ###################################                788.0 € (69.67%)
KELA/FPA                                 | #############                                      252.8 € (25.16%)
MobilePay: Emmi Suominen                 | ###                                                52.8 € (5.18%)

What information would you like to see ?
1. Show me my Incomes
2. Show me my Expenses
3. How much more can I earn ?
4. Group transactions ?
5. Quit
```

Kelalaskurin käyttö :

```
/usr/local/bin/python3.8 /Users/edwarddaka/PycharmProjects/edward-daka-y2-2021-rahan-seuranta/code/main.p

Welcome to the money tracker !

Enter the name of the statement file:
Tilitapahtumat.csv

What information would you like to see ?
1. Show me my Incomes
2. Show me my Expenses
3. How much more can I earn ?
4. Group transactions ?
5. Quit

Insert the number for the corresponding information:
3

I'll help you calculate how much more you can earn
before you'll need to contact KELA regarding your student benefits.

How many benefit months do you have during the year ?
9
How much have you already earned during the year ?
2500
With your current earnings of 1004.76 € you'll need to check your student benefits in 10 months.

What information would you like to see ?
1. Show me my Incomes
2. Show me my Expenses
3. How much more can I earn ?
4. Group transactions ?
5. Quit
```