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 ohjelma 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 aika 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 5 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 ”#”-symbolista.

   2. *Show me my Expenses* toimii täysin vastaavalla tavalla kuin 1. 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 kolme syötettä, jotka kysyvät jo tienatun summan, 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.

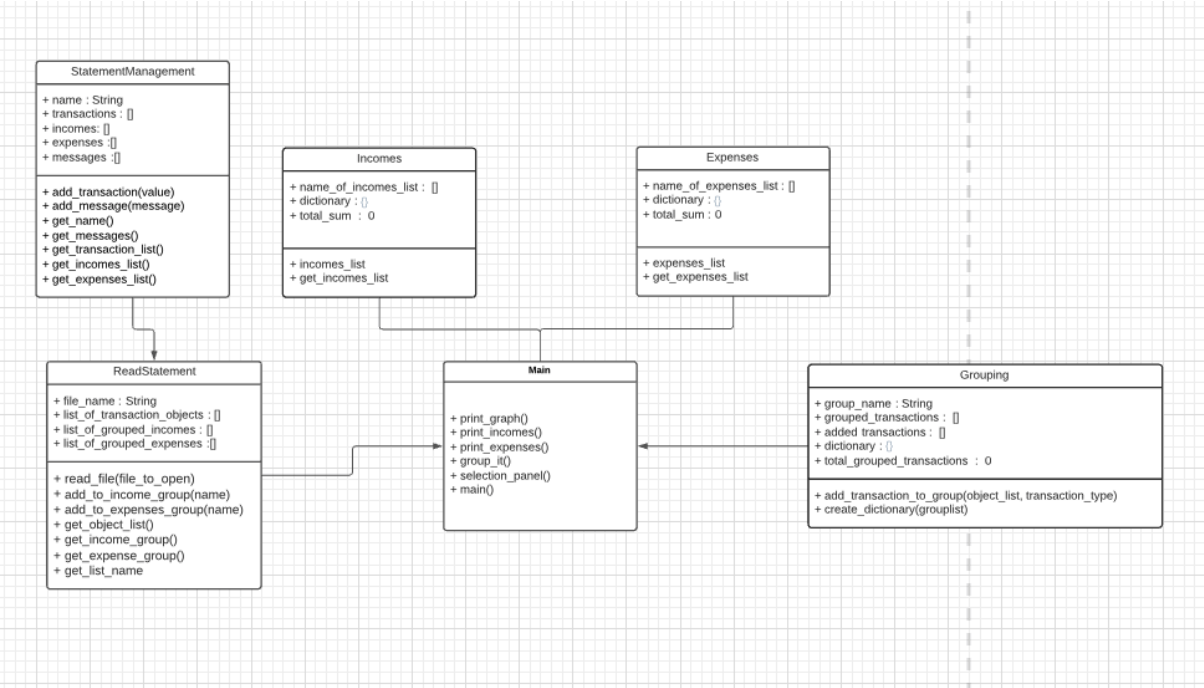
3. **Ulkoiset kirjastot**

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

4. **Ohjelman rakenne**

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

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



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

- *read_file* lukee tiedoston ja lisää tietoja *StatementManagementin* luomista transaktio olioista. Käsittelee myös transaktion lisäämän viestin mm. mikäli viestinä on ”Oma siirto” ei tätä lasketa tuloihin.
- *add_to_income_group* lisää olion *list_of_grouped_incomes* listaan. Tänne kootaan kaikki tuloihin luokiteltavat ryhmät.
- *add_to expenses_group* tekee vastaavan toimenpiteen, mutta menoihin
- *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 menoihin*.

- *get_list_name* palauttaa oliotiedoston nimen *file_name.*

Statementmanagement hoitaa yksittäisen tilitapahtuman tietojen käsittelyn. Se luo tilitapahtumalle olion sen nimen perusteella. Eli se ryhmittelee esim. 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 viestilistaa *messages*
- get_name palauttaa transaktion nimen esim. K-Market Jätkäsaari.
- get_messages palauttaa transaktion viestilistan messages
- get_transactions_list palauttaa transaktioiden lista 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 sanakirjan, 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 onko kyseessä tulot ja menot, joita ollaan ryhmittelemässä. 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 tekstipohjaien 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

- *Main* funktio suorittaa ohjelman aloituksen, se 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 ryhmitelty tai ryhmittelemätön
- *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* voisi yhdistää yhdeksi funktioksi. Tällöin poistettaisiin päällekkäisyyksi, sillä funktiot toimivat lähes samalla tavalla. Yhdellä funktiolla saavuttaisi siistimmän ja yksinkertaisen koodin.
2. *Print_incomesin ja Print_expensesin* sisällä voisi yhdistää tulojen ja menojen olio muodostusta yhdeksi funktioksi, näissä on ylimääräistä toistoa joka voisi olla yhdessä funktiossa.

5. **Algoritmit**

Rahan seuranta sovellus ei ole vaatinut monimutkaisten algoritmien rakentamista, matemaattinen laskenta perustuu pitkälti peruslaskutoimituksiin, kuten kerto-, jako -, yhteen -ja vähennyslaskuun.

Erityisen tärkeitä nämä ovat, 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. Esim. tulostukseen käytetystä algoritmistä.

Aloitetaan käymällä läpi, jokainen sanakirjaan lisätty avain. Jokaisen sanakirjan avaimen kohdalla, otetaan sitä vastaava arvo. Aloitetaan laskemalla kuinka iso osa kuvaajista

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

Tästä voidaan sen jälkeen laskea, paljonko palkkeja tulostetaan. Yksi prosentti vastaa puolta palkkia. Päädyin tähän testattuani erilaisia palkkimääriä. Liian suuri palkkimäärä esim. 1 palkki = 1 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 % ja 16% voi olla sama palkkimäärä. Testausten tuloksena 0,5 palkkia = 1% 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.

Sanallinen kuvaus käytetyistä algoritmeista, eli siitä miten ohjelma suorittaa tarvittavat tehtävät. Esim. miten tarvittava matemaattinen laskenta tapahtuu? (kaavat mukaan) Miten algoritmisi löytää lyhimmän tiereitin kahden kaupungin välille? Miten toteutun pelin tekoäly toimii? Kaavioita tms. voi käyttää apuna tarpeen mukaan. Mitä muita ratkaisuvaihtoehtoja olisi ollut? Perustele valintasi: Vertaa toteutusta johonkin toiseen ratkaisuun, ja selittäkää miksi päädyit juuri tähän.

Tässä kohdassa on siis tarkoitus selostaa ne periaatteet, joilla ongelmat on ratkaistu, ei sitä, miten algoritmit koodataan. Siis ei luokkien tai metodien kuvauksia tai muitakaan Pythoniin tai ohjelmakoodiin liittyviä seikkoja tänne. Pseudokoodiesitys keskeisimmistä ei-tunnetuista algoritmeista on kuitenkin hyvä olla sanallisen kuvauksen tukena. **HUOM!** Jokaisessa työssä on aina algoritmeja, toiset ehkä yksinkertaisempia kuin toiset, moni aivan itse alusta saakka keksittyjä. Kuvaa tässä niistä muutama kaikkein olennaisin.

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 tätä kautta yhteen listaan.

Sanakirjaa hyödynnettiin kuvaajan tulostuksesta. 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 tukikuukaudella voi tienata.

Minkälaiset kokoelmatyypit/tietorakenteet soveltuvat parhaiten ohjelmassa tarvittavan tiedon varastoimiseen ja käsittelyyn? Miksi? Mitä muita valintamahdollisuuksia olisi ollut? Käytittekö muuttuvatilaisia (mutable) vai muuttumattomia (immutable) rakenteita? Jos käytit Pythonin valmiita tietorakenteita, ei niiden tarkkaa määrittelyä tarvitse esittää. Jos taas ohjelmoit itse jonkin tietorakenteen, on sen toimintatapa selostettava.

7. **Tiedostot**

   Ohjelman toiminto perustuu pitkälti csv.tiedoston lukemiseen. Tarkemmin ottaen Osuuspankin tilitapahtumattiedoista luotuun csv-tiedostoon. Tiedosto koostuu 10 sarakkeesta, joita ovat *Kirjauspaiva;Arvopaiva;Maara Euroa;Laji;Selitys;Saaja/Maksaja;Saajan tilinumero ja pankin BIC;Viite;Viesti;Arkistointitunnus.* Kuten edelläolevasta 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 OP:n verkkopankistani noudetun tilitapahtumatiedoston.

   Tätä tiedostoa voi hyödyntää ohjelman testauksessa.

8. **Testaus**

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

   1. Tiliotteen lukeminen

      - Aluksi rakensi try -ja except toiminnot valitessaan 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 ja ei huomioi.

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 käyttäjä syöttämään ehtojen mukaisen syötteen.

Kertokaa miten ohjelmaa testattiin ja kuinka se vastasi suunnitelmassa esitettyä.

Läpäiseekö ohjelma kaikki suunnitelmassa esitetyt testit? Kuinka ohjelmaa testattiin sitä rakennettaessa? Oliko testauksen suunnittelussa jotain olennaisia aukkoja? Yksikkötestausta harjoiteltiin kurssin alkupuolella. Jos teit yksikkötestejä jollekin koodin osalle, niin kerro testauksesta tässä.

9. **Ohjelman tunnetut puutteet ja viat.**
    1. Ohjelma ei huomio mikäli ryhmittelyssä syötettävät tulot loppuvat.
        - Eli mikäli 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 lisäys vaiheessa.
        - Lisäsin syötteen, joka ilmoittaa tulostus vaiheessa että näin on päässyt tapahtumaan. Löysin vian sen verran myöhään, etten ehtinyt koodata tähän toimivaa korjausta.
        - Mahdollinen korjaus sisältäsi tarkisteen, joka tarkistaa jo tehdyt ryhmät ja työn alla olevaa ryhmää, löytyykö tapahtuma mahdollisesti niistä.
    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”.

Kuvaa tässä osiossa kaikki tuntemasi puutteet ja viat ohjelmassasi. Kerro miten korjaisit nämä ongelmat jos jatkaisit projektia. *Mitä vähemmän assistentti löytää puutteita kohdista, joiden väität toimivan sen parempi.* Ole siis rehellinen. Lisäksi hyvin jäsennetyt kehitystarpeet kertovat perehtymistä ongelmaan ja sen ratkaisuun, sekä kriittisestä oman työn arvioinnista.

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

    Parhaat:

    1. Tulostettu tekstipohjainen kuvaaja on selkeä ja antaa hyvän kuvan tulojen/menojen jakautumisesta.
    2. Tarkisteet antavat mahdollisuuden jatkaa käyttöä virheellisestä syötteestä huolimatta. Tällä tarkoitan, ettei esim. koko ryhmittely prosessia täydy aloittaa alusta, vaikka syöttää keskikohdassa ohjelmaa virheen. Ohjelma myös pyrkii välttämään turhien syötteiden antamisen, esim. 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 mikä osa tekee mitäkin.

    Heikkoudet:

        1. Main.py:ssä, joudutaan luomaan Incomes ja Expenses oliot. Tämä loi koodiin toisteisuutta, kun tehdään valinta luodaanko kuvaaja ryhmiteltynä vai ryhmittelemättömänä. Tähän en oikein keksinyt tapaa välttää tuota, 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 OP:lla. Koska minulla on rajallinen määrä tilitapahtumatiedostoja, on vaikeaa testata lukemisen onnistuvuutta. OP:lla tieto on aina tietyllä sarakkeella, joten mikäli sarakkeiden paikkaa muuttaa, muuttuu myös luettava tieto.
        3. Vioissa mainitut pienet puutteet, jotka johtuvat tarkistimien puutteesta

11. **Poikkeamat suunnitelmasta ja Toteutunut työjärjestys ja aikataulu**

12. Alkuperäinen suunnitelma, 26.2.2021

Alkuperäinen suunnitelmani oli jaettu kolmeen vaiheeseen:

Aloitus 26.2.2021 – 26.3.2021(Käytetty aika n. 30h) : 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 6 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. 34h) :

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. 8h)

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

13. Arvio lopputuloksesta

Olen lopputulokseen tyytyväinen. Mielestäni ohjelma saavuttaa olennaiset tavoitteet, jota projektilta on haluttu, etenkin helpon vaatimusten osalta.

Ohjelman kriittinen funktio on lukea tiedosto, ohjelman luonnissa on tehty tiettyjä oletuksia tiedoston muodon suhteen. Mikäli luettavaa tiedostoa muokkaa tarpeeksi, se saattaa ajaa lukemisen virheeseen tai tuottaa kuvaajan väärällä tavalla.

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.

Toinen kohta on mikäli ei ole tuloja tai menoja, olen tehnyt tähän tarkistimen kun kuvaajaa koitetaan luoda, mutta mikäli henkilö lähtee ryhmittelemään tuloja, ei tätä estetä. Tähänkin voisi rakentaa tarkisteen.

Mutta oletusarvona on, että henkilö ensin tarkistaa tulot ja menot ja vasta sitten ryhmittelee. Tällöin käyttäjä ei lähtisi ryhmittelemään tuloja/menoja joita ei ole ollenkaan.

Mielestäni koodia olisi voinut vielä virtaviivaistaa kattamaan edellä mainitut ongelmat. Etenkin print_incomes ja print_expenses funktioissa olevaa toisteisuutta olisi voinut minimoida yhdistämällä olioiden luonnin yksittäiseen funktioon.

Ohjelman rakenne tekee laajentamisesta mahdollista. Ohjelmaa on koitettu pitää mahdollisimman ja yksinkertaisena, jolloin uusien luokkien tai funktioiden integrointi onnistuu. Olioiden käytöllä saadaan uusien luokkien käyttöön helposti jo valmiina olevaa tietoa.

14. 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

15. **Liitteet**

Liitteeksi tulee mahdollisten muiden liitteiden lisäksi ainakin tekstipohjaisissa ohjelmissa laittaa **muutama havainnollinen ajoesimerkki**, jotka on kätevä tehdä unixympäristöissä script-ohjelmalla. Script-ohjelma käynnistyy kirjoittamalla script ja päättyy painamalla CRTL-D tai unix shellissa exit. Scriptin ollessa käynnissä se tallentaa tiedostoon kaiken ruudulle ilmestyvän sekä käyttäjän kirjoittaman syöteen. Graafisissa töissä ajoesimerkkejä ei vaadita, mutta muutama todellinen kuva ohjelman käytöstä joko erillisenä liitteenä tai käyttöohjeen yhteydessä ei tekisi pahaa. (Kuvia voi Linux- tai Windows ympäristöissä napsia painamalla Alt+PrintScreen-nappuloita, (Riippuen käyttöjärjestelmästä voit antaa joko tallenettavan kuvan nimen heti tai kuva on leikepöydällä, mistä sen voi tallentaa jonkin piirto-ohjelman kautta.)

##