Älykäs reseptikirja

Dokumentaatio

23.4.2013

Jesse Rantala
294049
jesse.rantala@aalto.fi

Tietotekniikan koulutusohjelma

Vuosikurssi 2011

Yleiskuvaus

Ohjelma älykäs reseptikirja on tietorakenteiden hallinta ohjelma. Ohjelmassa on mahdollista tallentaa raaka-aineita varastoon, joka on ikäänkuin käyttäjän keittiö kaappeineen ja niiden sisältöineen. Ohjelmassa pystyy myös luomaan reseptejä, asettamalla niille tietyt raaka-aineet, valmistusohjeet, mahdollisesti sisällyttää toisia reseptejä jne. Käyttäjän on mahdollista tutkia ja muokata varastoaan sekä reseptikirjansa reseptejä.

Ohjelman päätarkoituksena on antaa käyttäjälle mahdollisuus tutkia, mitä reseptejä hän pystyy toteuttamaan omistamillaan raaka-aineillaan, mitkä reseptit eivät sisällä vaadittuja allergeeneja, mitkä reseptit sisältävät tiettyjä raaka-aineita, mistä resepteistä käyttäjältä puuttuu N kappaletta raaka-aineita tai muilla ehdoilla joita käyttäjä haluaa hauilleen asettaa. Näitä ehtoja voidaan asettaa samanaikaisesti, esim. etsitään reseptejä joita voidaan varaston raaka-aineilla toteuttaa, sekä niitä jotka eivät sisällä laktoosia ja joissa on oltava kanaa.

Ohjelma koostuu kahdeksasta luokasta: StoreIngredient, RecipeIngredient, Store, Recipe, Convertions, Recipebook, UI sekä LoadAndSave. Näistä neljä ensimmäistä on luokkia, jossa luodaan raaka-aineet, varastot ja reseptit. Ne yhdistetään käyttöliittymässä UI, luokkaan RecipeBook, jonka kautta voidaan hallinnoida eri luokissa luotuja olioita, tutkia niiden suhteita ja suorittaa kyselyitä. Luokan LoadAndSave kautta on mahdollista suorittaa tiedon (reseptien, varaston ja raaka-aineiden) latausta ja tallennusta. Luokassa Convertions, suoritetaan mahdollisia raaka-aineiden määrien yksiköidenmuunnoksia, vertailuja sekä yhteenlaskuja.

Omasta mielestäni palautettava ohjelma on jossain helpon ja keskivaikean vaatimuksien välimaastossa. Keskivaikeasta puuttuu lähinnä graafinen käyttöliittymä. Mutta tekstipohjaisesta käyttöliittymästä tuli laajempi kuin mitä sen tarvitsi, tehtävänannon mukaan, olla. Ohjelmaan on myös toteutettu joitain ylimääräisiä ominaisuuksia mm. varaston raaka-aineiden sisältämät (mahdolliset) päivämäärät.

Käyttöohje

Ohjelmalla pystytään toteuttamaan kaikki kohdassa **Yleiskuvaus** esitetyt asiat. Eli sillä pystytään hallinnoimaan ja muokkaamaan omaa varastoa (Store-luokka), joka kuvastaa käyttäjän keittiötä, eli sen kaappeja sisältöineen (raaka-aineet). Käyttäjä pääsee tallentamaan ohjelmaan reseptejä (ja raaka-aineita varastoon), manuaalisesti tai lataamalla ne tekstitiedostosta. Tietokannan sisältäessä tietoa (reseptejä ym.) voi käyttäjä suorittaa ns. kyselyitä, eli tutkia reseptikirjansa ja varastonsa sisältöä, tutkia niiden suhteita sekä muokata/poistaa tietoa mitä on sinne ladattu tai tallennettu.

Ohjelman käyttö käyttäjän toimesta toimii täysin (tekstipohjaisen)käyttöliittymä-luokan UI kautta. Ohjelman pääsee käynnistämään sen kautta, suorittamalla siinä Python Runin. Erillisiä help komentoja, neuvomaan ohjelman käytössä ei ole, sillä käyttöliittymä antaa suht tarkasti, tiedon siitä mitä voidaan suorittaa ja miten. Luokka UI antaa siis sen mukaa vaihtoehtoja, tai pyytää syötteitä, missä ”menussa” sillä hetkellä olet tai minkä toiminnon olet valinnut.

Esimerkiksi käynnistäessä ohjelman, se kysyy haluatko ladata vanhan varaston vai luoda uuden. Suoritettuasi jommankumman pääset MainMenuun. MainMenussa sinulla on erilaisia vaihtoehtoisia toimintoja, riippuen siitä mitä haluat tehdä. Pääset näihin vaihtoehtoihin, niiden toimintoihin/uusiin aukeaviin menuihin kirjoittamalla komentoriville, toimintoja vastaavat merkit. Alla esitettynä Main Menun näkymä, käyttäjän ladattua vanha varasto käyttöön:

```
Hello, and welcome to use Intellectual RecipeBook!
Made by: Jesse Rantala, jesse.rantala@aalto.fi

TYPE
 'new' - to create new store
 'load' - to load your already created store
 'exit' - to exit program
load
Give a name of the file from where to load, in form 'file_name.txt':
Store.txt
Store file loaded succesfully.

Welcome to store Oma Varasto.

Main Menu

What would you like to do?
TYPE
 'i' - to add ingredients manually to your store
 'lr' - to load recipe(s) from a file
 'li' - to load more ingredient(s) to your store from a file
 'r' - to add recipe to your recipebook
 'si' - to search ingredient from your store
 'sr' - to search recipe
 'c' - to check your store's dates
 'mr' - to modify already existing recipe
 'ms - to modify / change your store or its ingredients
 's' - to save changes
 'q' - to quit
```

Eli esimerkiksi kirjottaessasi komentoriville ”si”, pääset tutkimaan varastosi raaka-aineita, uuden ”menun” kautta, riippuen siitä mitä haluat ottaa selville.

Jotkin MainMenun (tai jonkin muun alimenun) toiminnoista pyytää käyttäjältä syötteitä. Esimerkiksi, kun valitset Main Menussa ”i”:n eli haluat lisätä käsin raaka-aineita varastoon seuraa siitä:

```
Please type in the name of the ingredient you would like to add in your store:
Maito
Please type in the count of the ingredient you bought:
1.5l
Please type in the last date of use of your ingredient:
2013-04-25
Please type in the density of the ingredient:

Please type in the allergenic matter of the ingredient:
Laktoosi

Adding maito to your store.
```

Luokka UI, suorittaa syötteiden testausta, estäen viallisten syötteiden antamisen. Esimerkiksi:

```
Please type in the name of the ingredient you would like to add in your store:
Broileri
Please type in the count of the ingredient you bought:
1kilo
Not valid count. Valid units are liters, kilograms, deciliters, grams, milliliters,
teaspoon and tablespoon, or each with proper shortening. For count by pieces, feed only
number.
Please type in the count of the ingredient you bought:
```

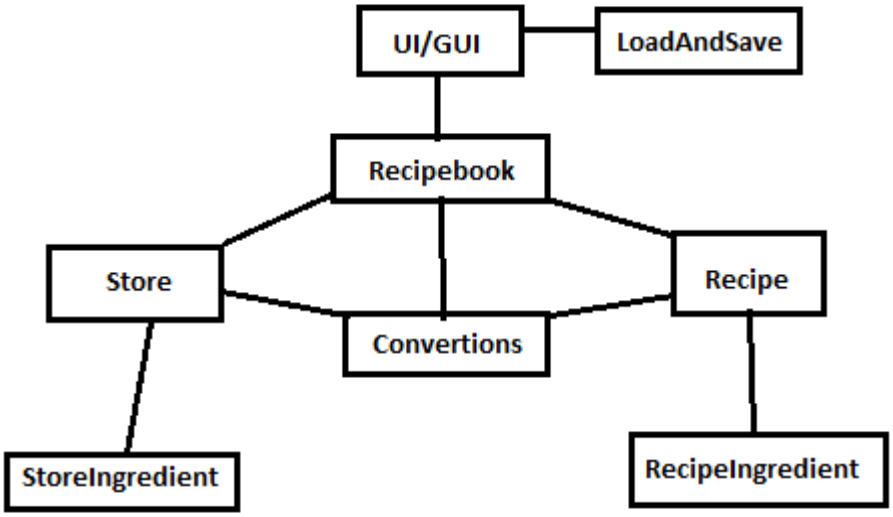
```
1kg
Please type in the last date of use of your ingredient:
2013-25-03
Given date not valid. Valid form: YYYY-MM-DD.
Please type in the last date of use of your ingredient:
2013-04-25
Please type in the density of the ingredient:

Please type in the allergenic matter of the ingredient:


Adding broileri to your store.
```

Huomaa, että antamalla tyhjän vastauksen (painamalla suoraan enter / antamalla whitespacen), tietyt kohdat (ne jotka sen hyväksyvät) asettavat tällöin vastaavat arvot defaulteiksi (useimmiten None, paitsi density = 1). Esim. allergeeniksi antaessa tyhjän, asettaa ohjelma allergeeniksi None.

Luokan UI funktiot ja niiden tehtävät on selitetty tarkemmin osiossa **Ohjelman rakenne**.

Ohjelman rakenne

Viereisessä kuvassa esitetään ohjelman käyttämien luokkien luokkakaavio. Siitä näkee luokkien väliset suhteet.



Rakenne on hierarkkinen jossa korkeammalla tasolla olevat luokat sisältävät alempien tasojen luokissa luotuja ominaisuuksia tai käyttävät niitä jollain tapaa.

Esim. RecipeIngredientissä luodaan reseptissä Recipe, käytettävät raaka-aineet. Luokkaan Recipe, kerätään siis eri RecipeIngredient-luokan olioita. Mahdollisia yksikön muunnoksia suoritetaan Recipe ja Convertions luokkien välillä. Recipe-luokan oliot talletetaan luokkaan Recipebook. Recipebookissa saatetaan käyttää yksikönmuunnoksia ja arvojen vertailuja, luokan Convertions kautta. Lopuksi käyttäjä pääsee käyttämään kaikkia Recipe-luokan olioita Recipebookissa, UI:n kautta.

Tämä sama em. hierarkkinen rakenne toteutuu toisessa haarassa, StoreIngredient-Store-Convertions-Recipebook-UI-LoadAndSave, samanaikaisesti toisen kanssa. Täten päästään hallinnoimaan luokkien Store (varasto) ja Recipe (resepti) välisiä suhteita ja niiden omia sekä yhteisiä ominaisuuksia luokassa Recipebook, ja mitä kaikkia käyttäjä hallinnoi/käyttää luokassa UI.

UI:ssa on myös mahdollista suorittaa lataus ja tallennus – operaatioita, jolloin kutsutaan luokkaa LoadAndSave. LoadAndSave on myös yhteydessä kaikkiin muihin ohjelman luokkiin,

käsitellessään ladattavaa/tallennettavaa aineistoa, mutta selvyyden vuoksi sitä ei piirretty kaavioon.

Luokat StoreIngredient ja RecipeIngredient olisi voitu yhdistää vain yhdeksi luokaksi Ingredient. Tästä lisää kohdassa **3 parasta ja 3 heikointa kohtaa**.

Seuraavana on esitelty ohjelman käyttämät luokat ja niiden funktioden tehtävät:

Luokka UI:

init(self):
Täällä asetetaan käytetävät luokat RecipeBook, LoadAndSave sekä Convert muuttujiin, eli self.system = RecipeBook(), self.loadsave = LoadAndSave() ja self.convertions = Convert(), eli luodaan luokkien oliot jotta voidaan käyttää niitä myöhemmin muissa funktioissa. Store-luokka on aluksi asetettu None:ksi, eli self.store = None, sillä funktiossa Start, siihen asetetaan vasta käyttäjän antama varasto.

Start(self):
Pyydetään käyttäjältä haluaako hän ladata vanhan vai luoda uuden varaston tai mahdollisesti lopettaa jo ohjelman.
Saatu storen arvo asetetaan self.store:ksi jolloin sitä pystytään käyttämään myöhemmin muissa funktioissa.

Exit(self):
Mahdollinen ohjelman lopetus, jos käyttäjä niin valitsee funktiossa Start.

Load(self):
Palauttaa varaston, jonka se on ladannut käyttäjän antamasta tiedostosta. Tätä funktiota kutsuu funktio Start, mikäli käyttäjä haluaa ladata vanhan varastonsa.

MainMenu(self):
Päävalikko jonka kautta käyttäjä pääsee valitsemaan eri alimenujen / toimintojen välillä. Nämä eri vaihtoehdot on esitelty alla.

LoadRecipe(self):
Täällä käyttäjä pääsee lataamaan reseptejä halutusta tekstitiedostosta ja ohjelma lisää ne annettuun RecipeBook-luokkaan.

LoadIngredient(self):
Täällä käyttäjä pääsee lataamaan raaka-aineita halutusta tekstitiedostosta ja ohjelma lisää ne annettuun Store-luokkaan.

AddIngredient(self):
Täällä käyttäjä pääsee lisäämään raaka-aineita manuaalisesti yksitellen varastoonsa.

AddRecipe(self):
Täällä käyttäjä pääsee lisäämään reseptejä yksitellen reseptikirjaansa.

CheckDates(self):
Täällä käyttäjä voi tarkistaa varastonsa raaka-aineiden päiväykset, ja suorittaa mahdolliset poistot mikäli löytyy vanhentuneita tuotteita.

ModifyStore(self):

Täällä käyttäjä pääsee muokkaamaan varastoaan. Toisin sanoen hän voi poistaa raaka-aineita, vähentää tietyn raaka-aineen määrää, muuttaa varaston nimeään tai vaihtaa varastonsa toiseen (ladata uuden tai luoda kokonaan uuden).

ModifyRecipe(self, recipe):
Täällä käyttäjä voi muokata tiettyä reseptiä, joka on aikaisemmin käyttäjältä pyydetty joko MainMenussa (tai mahdollisesti AddRecipessä, jos on koitettu lisätä reseptikirjassa valmiiksi ollutta reseptiä). Eli hän voi tehdä seuraavaa muokattavaan reseptiin: lisää/poistaa raaka-aineita, lisätä/poistaa valmistusohjeen, lisää/poistaa toisia reseptejä, poistaa reseptin reseptikirjasta, vaihtaa reseptin nimen ja vähentää tietyn raaka-aineen määrää reseptissä.

IngredientSearch(self):
Täällä suoritetaan käyttäjän haluamat haut varastonsa sisällöstä.

RecipeSearch(self):
Täällä suoritetaan käyttäjän haluamat haut reseptikirjan resepteistä.

Save(self):
Täällä käyttäjä voi tallentaa varastonsa, raaka-aineensa sekä reseptinsä erillisiin tekstitiedostoihin.

Quit(self):
Täällä käyttäjä voi sammutaa ohjelman suorituksen, ja mahdollisesti tallentaa varastonsa, raaka-aineensa sekä reseptinsä erillisiin tekstitiedostoihinsa.

Luokka LoadAndSave:

load_store(self):
Täällä ladataan käyttäjän antama varaston tiedot sisältämä tekstitiedosto.

load_recipes(self):
Täällä ladataan käyttäjän antama reseptien tiedot sisältämä tekstitiedosto.

load_ingredients(self):
Täällä ladataan käyttäjän antama raaka-aineiden tiedot sisältämä tekstitiedosto.

save_store(self):
Täällä tallennetaan käyttäjän käyttämä varasto tekstitiedostoon.

save_ingredients(self):
Täällä tallennetaan käyttäjän käyttämän varaston sisältämät raaka-aineet tekstitiedostoon.

save_recipes(self):
Täällä tallennetaan käyttäjän käyttämän reseptikirjan sisältämät reseptit tekstitiedostoon.

Luokka LoadAndSaveError(Exception):
Tätä luokkaa kutsutaan jos jokin lataus ei onnistu luokassa LoadAndSave.

Luokka Recipebook:

init(self):
Asetetaan self.recipes ja self.stores tyhjiksi listoiksi, sekä alustetaan self.convertions,

luokaksi Convert.

get_recipes(self):
Täällä haetaan ja palautetaan kaikki reseptikirjan sisältämät reseptit.

add_recipe(self, recipe):
Täällä reseptikirjaan lisätään haluttu resepti, recipe.

add_store(self, store):
Täällä reseptikirjaan lisätään haluttu store. Eli lisätään se listaan self.store.
Huom. tämän voisi myös tehdä kuten luokassa UI, eli alustaa self.store aina vain tietyksi varastoksi, sillä niitä ei ole käytössä kuin vain yksi kerrallaan joka tapauksessa.

add_other_recipe_to_other(self, where, toadd, count):
Täällä tutkitaan onko toiseen reseptiin lisättävä resepti, reseptikirjassa. Jos ei ei tehdä lisäystä. Jos on suoritetaan lisäys.

enough_ingredients(self, recipe, store):
Täällä tutkitaan onko varastossa, store, tarpeeksi raaka-aineita (onko reseptin vaatimat raaka-aineet, sekä onko vaatimia raaka-aineita tarpeeksi), jotta voidaan valmistaa annettu resepti, recipe. Palauttaa True tai False.

can_prepare(self, store):
Täällä tutkitaan yllä olevaa funktiota apuna käyttäen, kaikki reseptikirjan reseptit, ja palautetaan käyttäjälle lista niistä jotka voidaan valmistaa varaston raaka-aineilla.

recipes_without(self, allergenic):
Täällä haetaan ne reseptit, joissa ei ole annettua allergeenia, allergenic, ja palautetaan käyttäjälle listana.

store_in_date(self, store):
Täällä tutkitaan onko annetun varaston raaka-aineet vanhentuneita, kutsuen luokan Store funktiota check_dates().

modified_recipe(self, recipe):
Täällä lisätään reseptikirjassa jo ollut, mutta muutettu resepti, recipe, reseptikirjaan ja poistetaan vanha. Poistot ja lisäykset tehdään yhtälailla jokaiselle reseptille, joihin muutettu resepti on sisältynyt.

delete_recipe(self, recipe):
Täällä poistetaan annettu resepti, recipe, reseptikirjasta, sekä jokaisesta reseptistä, johon se on sisällytetty.

recipes_without_n(self, store, count):
Täällä haetaan ne reseptit, joiden raaka-aineista varastosta saa puuttua enintään count:n verran kyseisiä aineita, ja palautetaan löydetyt reseptit listana.

recipes_with_ingredients(self, ingredientlist):
Täällä haetaan ne reseptit, joissa on annetun ingredientlistin määräämät raaka-aineet, ja palautetaan löydetyt reseptit listana.

Luokka Recipe:

init(self, name):
Täällä asetetaan self.name:ksi reseptille annettu nimi, name. Alustetaan self.ingredients ja self.recipes tyhjiksi sanakirjoiksi. Näihin lisätään myöhemmin pareina ensimmäiseen raaka-aineet ja niiden määrät, sekä jälkimmäiseen reseptiin sisältyvät reseptit ja niiden kertoimet. Täällä alustetaan myös self.ilist, self.rlist sekä self.comment tyhjiksi listoiksi. Ensimmäiseen lisätään reseptin käyttämät raaka-aineet, toiseen siihen sisältyvät toiset reseptit sekä viimeiseen valmistusohje.

get_name(self):
Palauttaa reseptin nimen.

get_comment(self):
Palauttaa reseptin valmistusohjeen.

get_ingredients(self):
Palauttaa reseptin käyttämät raaka-aineet.

get_count_of_ingredient(self, ingredient):
Palauttaa haetun raaka-aineen, ingredient, määrän reseptissä.

add_ingredient(self, ingredient, count):
Lisää reseptiin raaka-ainetta, ingredient, count:n verran.

add_comment(self, comment):
Lisää reseptiin annetun valmistusohjeen.

add_other_recipe(self, recipe, count):
Lisää reseptiin toisen reseptin ohjetta, recipe, count:n verran.

get_allergenics(self):
Palauttaa kaikki allergeenit jota reseptissä on.

recipe_has_allergenic(self, allergenic):
Palauttaa True tai False, sen mukaan onko reseptissä annettua allergeenia, allergenic.

recipe_has_ingredient(self, ingredient):
Palauttaa True tai False, sen mukaan onko reseptissä annettua raaka-ainetta, ingredient.

get_information(self):
Palauttaa reseptin tiedot. Eli nimen, raaka-aineet ja niiden määrät, valmistusohjeen sekä mahdolliset sen sisältämät allergeenit.

Luokka Store:

init(self, name):
Asetetaan self.ingredients tyhjäksi listaksi. Tähän tallennetaan varaston sisällään pitämät raaka-aineet. Self.name asetetaan annetuksi nimeksi, name.

get_name(self):
Palauttaa varaston nimen.

add_ingredient(self, ingredient):
Lisää annetun raaka-aineen, ingredient, varastoon.

get_rid_of(self, ingredient):
Poistaa annetun raaka-aineen, ingredient, varastosta.

get_ingredients(self):
Palauttaa merkkijonona varaston raaka-aineet yhdistettynä niiden määriin.

get_ingredients_without_counts(self):
Palauttaa saman kuin ylempi, mutta ilman määriä.

get_count_of_ingredient(self, ingredient):
Palauttaa annetun raaka-aineen, ingredient, lukumäärän varastossa.
check_dates(self):
Tarkistaa varaston raaka-aineiden päiväykset käyttämällä hyödykseen StoreIngredient-luokan funktiota is_expired. Vanhentuneen raaka-aineen löytyessä, kysyy käyttäjältä haluaako että kyseinen raaka-aine poistetaan.

ingredients_without(self, allergenic):
Palauttaa kaikki varaston raaka-aineet, joista ei löydy annettua allergeenia, allergenic.

Luokka RecipeIngredient:

init(self, name, density, allergenic):
Asetetaan annetut arvot name, density ja allergenic, muuttujiin self.name, self.allergenic ja self.density. Jos annettu tiheys, density, on None, asetetaan se arvoon 1.

get_name(self):
Palauttaa raaka-aineen nimen.

get_density(self):
Palauttaa raaka-aineen tiheyden

is_allergenic(self):
Palauttaa True tai False sen mukaan onko raaka-aineessa allergisoivaa tekijää.

get_allergenic_matter(self):
Palauttaa raaka-aineen allergisoivan tekijän.

Luokka StoreIngredient:

init(self, name, date, count, density, allergenic):
Asetetaan annetut arvot name, date, density ja allergenic, muuttujiin self.name, self.date, self.allergenic ja self.density. Jos annettu tiheys, density, on None, asetetaan se arvoon 1.

get_name(self):
Palauttaa raaka-aineen nimen.

get_expire_date(self):
Palauttaa raaka-aineen vanhenemis päivämäärän.

get_count(self):
Palauttaa raaka-aineen määrän.

is_allergenic(self):
Palauttaa True tai False sen mukaan onko raaka-aineessa allergisoivaa tekijää.

get_allergenic_matter(self):
Palauttaa raaka-aineen allergisoivan tekijän.

get_density(self):
Palauttaa raaka-aineen tiheyden

is_expired(self):
Palauttaa True tai False sen mukaan onko raaka-aine vanhentunut.

Luokka Convertions:

init(self):
Täällä on määritelty self.counts ja self.units, jotka käsittävät sallitut merkistöt, mitä reseptikirjassa käytetään.

check_time_count(self, a):
Täällä tarkistetaan käyttäjältä luokassa UI pyydetyt arvot, ovatko ne positiivisia lukuja.

check_count(self, a):
Täällä tarkistetaan onko käyttäjältä luokassa UI pyydetty raaka-aineen määrä, oikeaa muotoa. Eli löytyykö sen yksikkö init:ssä määritetystä self.units listasta.

a_enough_for_b(self, a, b):
Täällä palautetaan True tai False sen mukaan onko a suurempi tai yhtä suuri kuin b. Esimerkiksi kun tarkistetaan pystyykö varasto tekemään jotain reseptiä, tarkistetaan niiden samojen raaka-aineiden määrät täällä, eli riittääkö varaston raaka-aineen määrä, a, reseptin raaka-aineen määrään, b.

add_a_to_b(self, a, b):
Täällä lisätään a:n arvo b:hen. Eli esimerkiksi kun sisällytetään toinen resepti toiseen, lisätään niiden samat raaka-aineet yhteen täällä.

subtract_from_a(self, a, count):
Täällä vähennetään a:sta määrä count. Eli esimerkiksi kun käyttäjä haluaa vähentää varastonsa jonkin raaka-aineen määrää, se toteutetaan täällä.

b_times_a(self, a, b):
Täällä kerrotaan a b:llä. Eli esimerkiksi kun sisällytetään toisen reseptin ohjetta kolminkertaisesti toiseen reseptiin, suoritetaan sisällytettävän reseptin raaka-aineiden määrien kertomisia kolmella täällä.

convert_a_to_b(self, a, counta, b):
Täällä on varsinainen yksikön muunnos. Eli kun halutaan muuttaa raaka-aineen a määrä, counta, samaan yksikköön toisen raaka-aineen määrän, b, kanssa kutsutaan tätä funktiota. Palaute on counta muutettuna samaan yksikköön b:n kanssa.

kilograms_to_liters(self, count, ingredient):
Tätä kutsutaan kun halutaan muuttaa raaka-aineen, ingredient, määrä, count, kilogrammoista litroiksi.

liters_to_kilograms(self, count, ingredient):
Tätä kutsutaan kun halutaan muuttaa raaka-aineen, ingredient, määrä, count, litroista kilogrammoiksi.

liters_to_desiliters(self, count):
Tätä kutsutaan kun halutaan muuttaa määrä, count, litroista desilitroiksi.

kilograms_to_grams(self, count):
Tätä kutsutaan kun halutaan muuttaa määrä, count, kilogrammoista grammoiksi.

teaspoon_to_liters(self, count):
Tätä kutsutaan kun halutaan muuttaa määrä, count, teelusikoista litroiksi.

tablespoon_to_liters(self, count):
Tätä kutsutaan kun halutaan muuttaa määrä, count, ruokalusikoista litroiksi.

Algoritmit

Ohjelmassa ei käytetä mitään suuria tai tunnettuja algoritmeja, vaan se on lähinnä useiden tietorakenteiden (listojen ja sanakirjojen) samanaikaista hallintaa ja luokkien välisten funktioiden käsittelyä. Keskeisimmät tehtävät mitä tietokannassa tehdään, koskee jollain tapaa reseptien etsimistä, muokkausta tai varaston ja sen raaka-aineiden käsittelyä.

Tärkein tehtävänannossa annettu tehtävä ohjelman toiminnalle, on ohjelmasta reseptien etsiminen käyttäjän antamin ehdoin. Ohjelmassani tänne pääsee Main Menun kautta, avaamalla reseptien etsintämenun (komento 'sr' – to search recipe), ja täällä valitsemalla kohdan ('m' – modified search), jolloin käyttäjältä kysytään ehdot (allergeeni, raaka-aine, n-raaka-ainetta, jotka reseptin raaka-aineista voi varastosta puuttua sekä reseptit jotka varaston raaka-aineilla on mahdollista valmistaa [viimeinen on siis sama kuin, n = 0, toiseksi viimeisessä ehdossa]), minkä mukaan hän haluaa suorittaa hakuja.

Kun käyttäjä on tullut tähän toimintoon modified search, suorittaa ohjelma while-looppia, jonka sisältä eri ehdot löytyvät, niin kauan kunnes käyttäjä antaa komennoksi 's', eli hän haluaa suorittaa haun antamillaan ehdoilla. Tämäntyyppistä while-loopin käyttöä, luokassa UI käytetään paljon.

Oletetaan että käyttäjä syöttää jokaiselle ensimmäisestä kolmesta ehdosta jonkin arvon. Ensimmäiseen ehtoon, eli ”anna jokin allergeeni mitä et halua reseptin sisältvän”, annetaan ”laktoosi”. Tällöin ohjelma kutsuu luokan RecipeBook funktiota, recipes_without(allergenic) arvolla laktoosi. Sitten ohjelma käy lineaarisesti läpi RecipeBook luokan listan resepteistä ja tutkii jokaisen kohdalla onko siinä kyseistä allergeenia. Reseptit joista ei allergeenia löytynyt palautetaan listana luokalle UI, joka ottaa sen talteen muuttujaan.

Seuraavaksi annetaan ehdoksi että resepteissä täytyy olla kananmunia, sekä kinkkua, joka näyttää käyttöliittymän tulosteissa tältä:
TYPE

```
 'allergenic' - to search for recipes without allergenic
 'ingredient' - to get recipes with certain ingredient(s)
 'n' - to get recipes without N ingredient(s)
 'prepare' - to search for recipes you can prepare with your store
 's' - to execute search with given limits
ingredient
Give the ingredient you would like recipe to include.
Kananmunat
Give another ingredient you would like recipe to include, or type 'done' if you don't
want more ingredients to limit your search.
Kinkku
Give another ingredient you would like recipe to include, or type 'done' if you don't
want more ingredients to limit your search.
done
```

Luokan UI tallennettua saadut ingredient-arvot muuttujaan (tyhjään listaan), antaa se kyseisen listan parametrina luokan RecipeBook funktiolle recipes_with_ingredients(ingredientlist). Tässä funktiossa, ohjelma käy taas läpi kaikki reseptit reseptikirjassa, käy jokaisen reseptin omat raaka-aineet läpi, sekä jokaisen reseptiin sisällytetyn reseptin raaka-aineet läpi, tallentaen aineet tyhjään listaan. Kun reseptissä olevat kaikki raaka-aineet on tallennettu listaan, tutkitaan kuinka monta raaka-ainetta käyttäjän antamista aineista löytyy tältä listalta. Jos tämä tutkittu määrä on sama kuin käyttäjän antaman listan alkioiden määrä, on reseptissä kaikki vaaditut raaka-aineet. Kaikki raaka-aineet jotka toteuttavat vaaditun ehdon palautetaan listana luokalle UI, jolloin luokka UI, suorittaa ensimmäisen ehdon ja toisen ehdon antamien listojen ”leikkauksen”. Eli jäljelle jääneeseen listaan jää ne reseptit, jotka löytyvät molemmista kahden ehdon palauttamista erillisistä listoista.

(Tämän em. koko ehdon antamien reseptien etsimisen olisi voinut toteuttaa helpommin samalla tavalla kuin toteutettaessa ensimmäistä ehtoa, eli etsiessä reseptejä ilman tiettyä allergeenia, ja käyttämällä luokan Recipe funktiota has_ingredient kuten ensimmäisessä ehdossa funktiota has_allergenic. Tässä on ohjelmoitu turhan hankalasti, kun olisi ollut jopa valmiit funktiot, ehdon helpompaan toteutukseen. **Tähän on viitattu kohdasta Ohjelman tunnetut puutteet ja viat.**)

Seuraavassa ehdossa, haetaan reseptit, joiden raaka-aineista varastosta voi puuttua N-määrän aineita. Tämä on toteutettu kuten edeltävä kohta (toteutetulla hankalalla tavalla), mutta lopussa ei tutkita onko käyttäjän antaman raaka-ainelistan alkioiden määrä sama kuin niiden ja reseptin raaka-ainelistan yhteisten aineiden määrä. Tässä tutkitaan onko varaston raaka-ainelistan ja reseptin raaka-ainelistan (= reseptin omat raaka-aineet sekä reseptiin sisällytettyjen reseptien raaka-aineet) yhteisten aineiden lukumäärän, reseptin raaka-ainelistasta vähentämä määrä sama kuin vaadittuna ehtona annettu luku N. Reseptit jotka toteuttavat ehdot palautetaan luokalle UI, ja toteutetaan sama listojen ”leikkaus”, jossa luokan UI tallentamaan listaan, resepteistä jotka täyttävät käyttäjän antamat ehdot, jää jäljelle ainoastaan ne reseptit, jotka täyttävät tosiaan kaikki kolme suoritettua ehtoa. Eli ne reseptit jotka on löytynyt kaikista kolmesta luokalle UI palautetusta listasta.

Tietorakenteet

Käytin ohjelmassa lähinnä Pythonin sisäänrakennettuja listoja sekä sanakirjoja, tietorakenteina. Tiedon varastoimiseen kyseisessä tietorakenne-ohjelmassa oli hyvinkin tärkeää käyttää nimenomaan dynaamisia rakenteita, sillä on hankala arvioida paljonko reseptejä ym. dataa käyttäjä tulee tallentamaan.

Jos olisi käytetty esim. staattisia taulukoita, olisi se saattanut hyvinkin pian täyttyä jolloin data olisi jouduttu luomaan uuteen suurempaan taulukkoon, mikä on hyvin raskas operaatio. Toinen

vaihtoehto olisi ollut että taulukosta olisi tehty valmiiksi hyvinkin suuri, jolloin siinä olisi hyvin paljon tyhjää tilaa, mutta se tila olisi silti jatkuvasti varattuna muistissa, mikä ei myöskään ole hyvä vaihtoehto.

Em. syistä, dynaamiset tietorakenteet olivat ainut järkevä vaihtoehto kyseiseen ohjelmaan.

Tiedostot

Ohjelma pystyy käsittelemään .txt-päätteisiä tekstitiedostoja, latauksessa ja tallennuksessa. Näitä tiedostoja hallinnoidaan luokan LoadAndSave kautta. Se pystyy lataamaan erillisistä tekstitiedostoista varaston tiedot (itse varasto, eli sen sisältämät raaka-aineet), reseptikirjan reseptit (Recipe-olioita) sekä varastoon lisättäviä tai sen sisältämiä raaka-aineita erikseen ilman yhdistämistä tiettyyn varastoon. Nämä yhdistetään latauksen suorituksessa UI:n kautta luokkaan RecipeBook, jotta käyttäjä voi päästä käyttämään ladattua dataa. Ohjelma myös pystyy tallentamaan tietoa, joka on vain latauksen vastakkainen operaatio, jolloin ohjelma luo ohjelman LoadAndSave-luokan määrittelemän ja hyväksymän tekstitiedoston, tallennettavasta tiedosta (varasto, raaka-aineet tai reseptit).

```
#Store
Oma Varasto
#Ingredient
Maito
4l
2013-03-25
Laktoosi
1.5
#Ingredient
Kananmunat
12
2013-03-29
None
1
#Ingredient
Leipa
1kg
2013-03-27
Gluteeni
```

Tekstitiedostoista on mallit lähdekoodin mukana, mistä voi nähdä mallin miten tieto on niissä esitetty. Esimerkkinä viereinen kuvitteellinen Store_esimerkki.txt, jossa siis on ladattavissa valmis varasto:

Eli tiedoston aloittava #Store, on merkitsemässä että ladattava tiedosto on varasto. Sen jälkeisellä rivillä on varaston nimi. Tämän jälkeen esitetään varastosta löytyvät raaka-aineet kertomalla ensin että seuraava on raaka-aine eli #Ingredient. Sen jälkeen tulee eri riveillä järjestyksessä sen nimi, määrä, viimeinen käyttöpäivämäärä, allergeeni sekä tiheys. Näin esitetään koko varasto ohjelman hyväksymässä muodossa.

Vastaavasti pelkkä raaka-aine tiedosto, ladattaessa varastoon lisää raaka- aineita, on vastaava kuin Store tiedosto, ilman tiedoston aloittavaa #Store:a sekä varaston nimeä.

Reseptitiedostoissa uuden reseptin aloittaa #Recipe, josta seuraavalla rivillä on sen nimi. Raaka-aineet luetellaan kuten Storessa ja Ingredientissä, mutta ilman päiväystä. Sen lisäksi reseptissä voi olla #Comment, joka ilmaisee että seuraavalta riviltä alkaa valmistusohje (comment). Resepteissä voi myös olla #Other Recipe, joka kertoo että seuraavalta riviltä saadaan reseptiin sisällytettävän reseptin nimi, ja sitä seuraavalta riviltä määrä monta sisällytettävää reseptiä reseptiin kuuluu. Huomattavaa on että, sisällyttäessä toista reseptiä toiseen, on loogisesti sisällytettävän reseptin löydyttävä reseptikirjasta. Eli tiedon latauksessa (ja ylipäätään) jos haluaa sisällyttää toisen reseptin toiseen täytyy sisällytettävän reseptin olla määritetty tiedostossa ennen sitä johon se sisällytetään.

Testaus

Testauksen pyrin suorittamaan siten, kuten suunnitelmassa kerrottiin, että ensin testasin (aina kun uusia ominaisuuksia luotiin) uudet luodut funktiot ja ominaisuudet erikseen testi-luokassa. Jos ne toimivat testi-luokassa erikseen, kokeilin niitä tekstipohjaisen käyttöliittymän kautta, kokeillen niiden yhteentoimivuutta muiden funktioiden kanssa, suorittaen ensin monia eri toimintoja, sitten

vasta testattavat toiminnot ja sen jälkeen tarkistaa että ohjelma toimii oikein vielä niiden suorittamisen jälkeen. Ennen kuin tekstipohjainen käyttöliittymäni oli valmis, suoritin uusien ominaisuuksien testaamista ja yhteentoimivuutta muiden kanssa, testi-luokassa.

Palautettava ohjelma on läpäissyt kaikki suorittamani testit, ja on toiminut siten kuten olen alunperin suunnitellut sen toimivan. Pyrkimykseni on ollut suorittaa testit kattavasti, ja kaikki vaihtoehdot läpikäyden, mutta on aina mahdollista, että jokin asia on jäänyt huomaamatta, ja ohjelma tekee jotain mikä ei ole toivottavaa.

Ohjelman tunnetut puutteet ja viat

Ohjelma pystyy lisäämään reseptejä toisiin resepteihin. Tämä voi tapahtua kuitenkin vain yhdellä tasolla. Esim. reseptiin 1, on sisällytetty reseptin 2 raaka-aineet. Luodaan uusi resepti 3, johon halutaan sisällyttää reseptin 1 raaka-aineet, tällöin resepti 3 saa vain ja ainoastaan reseptin 1 alkuperäiset raaka-aineet, eikä niiden lisäksi reseptin 2 raaka-aineita. Ohjelmassa siis oletetaan että ei suoriteta em. kolmen tai useamman reseptin hierarkkioita vain ainoastaan on mahdollista suorittaa kahden reseptin alkuperäisten raaka-aineiden yhdistämisiä.

Kun esim. reseptiin 1 sisällytetään resepti 2, ei reseptiä 1 tutkittaessa nähdä että se sisältää toisen reseptin. Toisin sanoen reseptin 2, raaka-aineet ainoastaan lisätään reseptiin 1, ja ne näkyvät reseptin 1 valmistusohjeessa, mutta ei näy että ne ovat reseptin 2 raaka-aineita. Jos resepteillä on samoja raaka-aineita, em. aineiden määrät vain lisätään yhteen ja näytetään yhteenlaskettu määrä.

Alla esimerkkitulosteet kahden eri reseptin tiedoista:

On luotu resepti Taikina:

```
Taikina

Ingredients:

2dl jauhot

2 kananmunat

1dl sokeri

0.2kg voi

Preparation:
vatkaa...

Recipe includes following allergenics:
gluteeni, laktoosi
```

Joka on sisällytetty reseptiin Herkku Letut kaksi kertaa, eli Herkku Lettuja valmistettaessa tehdään kaksi Taikinaa. Tästä johtuen Herkku Letusta löytyvät Taikinan raaka-aineiden määrät ovat kaksinkertaisia verrattuna Taikina-ohjeeseen:

```
Herkku Letut

Ingredients:

4.0dl jauhot

4.0 kananmunat
```

```
3dl maito

2 salainen ainesosa

2.0dl sokeri

0.4kg voi

Preparation:
lisaa......

Recipe includes following allergenics:
laktoosi, gluteeni
```

Reseptiä muokatessa, jos haluaa poistaa tiettyjä raaka-aineita, tai vähentää tietyn raaka-aineen määrää, suorittaa ohjelma vähennyksen tai poiston ainoastaan vaaditun reseptin omien raaka-aineista, eikä osaa ottaa huomioon siihen sisällytetyn reseptin raaka-aineita näissä operaatioissa. Esim. jos ylläolevaa reseptiä Herkku Letut muokattaisiin, siten että koitettaisiin poistaa kananmunat, ei ohjelma osaisi niitä poistaa sillä ne eivät ole Herkku Letun omia raaka-aineita vaan Taikinan raaka-aineita.

Luokassa Recipebook, voisi olla alustettu self.store, yhtälailla kuin luokassa UI, yhdeksi luokaksi, eikä tallentaa sitä listaan. Tämä siitä syystä, että reseptikirjalla on käytössä vain yksi varasto kerrallaan, sekä listassakin on vain yksi varasto kerrallaan.

Ohjelmassa on myös mahdollista tällä hetkellä saada sekaannuksia aikaan lyömällä turhia whitespaceja ennen varsinaisia syötteitä, koska näitä syötteitä funktioille annettaessa parametreiksi, on jäänyt tämä whitespacejen karsinta pois. Se olisi helposti korjattavissa, lisäämällä tietokantaan syötteet vasta kun niistä on karsittu turhat whitespacet alusta ja lopusta strip()-komennolla. Tätä stringien strip() komentoa käytetäänkin jo syötteiden hyväksyttävyyden testauksessa hyödyksi.

Ohjelmassa on myös muutamia turhan hankalasti ohjelmoituja kohtia, jotka olisi muutettavissa suht nopeasti ja helposti, mutta ajan vähyyden vuoksi en niitä kerennyt tekemään. Esimerkkinä tällaisesta hankalasta ohjelmoinnista on esitetty kohdassa **Algoritmit**.

3 parasta ja 3 heikointa kohtaa

Ohjelmani kolme heikointa kohtaa ovat ehdottomasti seuraavat:

- loin erikseen Ingredient luokat varastoille ja resepteille
- luokka Convertionsista tuli hiukan hankalasti toteutettu, pitkä ja vaikeaselkoinen
- yleisestikin koodin vaikelukuisuus, ja paikoittaiset hankalat toteutukset, etenkin luokat UI, RecipeBook ja Convertions

Ensimmäinen näistä mainituista heikoista kohdista olisi ollut helposti vältettävissä, luomalla yksi Ingredient-luokka, ja antamalla sille samat funktiot ja parametrit kuin tämänhetkisellä RecipeIngredientillä. Toisin sanoen oltaisiin voitu poistaa StoreIngredient-luokka, luoda varastoon (sekä resepteihin) lisättävät raaka-aineet Ingredient-luokassa, ja ottaa raaka-aineen määrä talteen erilliseen sanakirjaan, yhdistäen siinä raaka-aineet ja niiden määrät, samalla tavalla kuin resepteillä. Samoin olisi saatu lisättyä raaka-aineiden päiväykset, erilisseen sanakirjaan Store-luokassa. Tällöin oltaisiin vältytty kahdelta erilliseltä Ingredient-luokalta. Syy miksen näin tehnyt, oli se, että olin ehtinyt koodata jo suhteellisen pitkälle kahden erillisen luokan taktiikalla, eikä minulla ollut enää aikaa korjata sitä.

Toiseen heikkoon kohtaan olisi varmasti voinut kehittää toimivamman toteutuksen kuin > 100 riviä if-lauseita, mutta en keksinyt muuta, joten päädyin tähän. Huomattavaa on, että ratkaisu toimii, mutta ei ole helposti luettavaa eikä tehokastakaan.

Kolmas heikko kohta on osittain mainittu jo kohdissa **Ohjelman tunnetut puutteet ja viat** sekä **Algoritmit**. Ajanpuuttellisuuden ja ryhtymisestä liian nopeasti koodaamaan suunnittelun sijaan takia, on koodi osittain hankalasti toteutettua tai/ja hankalasti toisen osapuolen luettavissa.

Paras kohta ohjelmassani on mielestäni käyttöliittymä. Vaikka se saattaa olla hiukan sekava ja sen koodi vaikealukuista, on se kuitenkin toimiva, ja se antaa käyttäjälle paljon mahdollisuuksia ohjelman erilaiseen käyttämiseen ja tarjoaa monia eri toimintoja. Luokka UI, oli tarkoitus olla lähinnä edeltäjä graafiselle käyttöliittymälle, ja ikäänkuin pohjapiirustus sille. Jos UI:n olisi päässyt toteuttamaan graafisesti olisi sen käyttö ollut selkeämpää ja helpompaa.

Poikkeamat suunnitelmasta

Hyvin pitkälle pitäydyin suunnitelmassani. Erillisiä Ingredient-luokkia en toteuttanut, mikä oli alunperin tarkoitus tehdä. Syyt on mainittu kohdassa **3 parasta ja 3 heikointa kohtaa**.

Toteutusjärjestelmä oli aikalailla suunnitelman mukainen. Tiedon lataus ja tallennus oli nopeammin toteutettu kuin uskoin, mutta tekstipohjaiseen käyttöliittymään taas meni hyvin paljon enemmän aikaa. Hyvin pitkälti sen korjaaminen, jatkuva suunnittelu ja testaaminen kesti kauan.

Graafiseen käyttöliittymään en ehtinyt koskaan paneutua tarpeeksi, että olisin saanut sen deadlineen mennessä valmiiksi. Tähän vaikutti muun ohjelman korjaamisen, ja etenkin tekstipohjaisen käyttöliittymän, viemä aika, muut koulutehtävät sekä ylipäätään (varsinkin loppupuolella) rajallinen aika ohjelman tekemiseen.

Kaiken kaikkiaan olin aika pitkälti hyvällä mallilla jo checkpointin aikoihin, joka oli alkuperäinen suunnitelmanikin, jolloin minulla oli lähes toimiva tekstipohjainen käyttöliittymä, lataus ja tallennus sekä tietenkin valmiina ohjelman alemman hierarkian luokat. Sen jälkeen ei ohjelmoinnille löytynyt tarpeeksi enää aikaa, ja aika jonka ehdin käyttämään meni ohjelman muokkaamiseen ja korjaamiseen.

Toteutunut työjärjestys ja aikataulu

Kuten edellisessä kappaleessa mainitsin, oli työjärjestys lähes yhtäpitävä suunnitelmani kanssa. Olin ehtinyt aloittaa ohjelmoinnin jo 18.2. ja suunnitelmademon palautuksen aikoihin oli minulla alemman tason luokat (RecipeIngredient, StoreIngredient, Store ja Recipe) jollain tapaa aluillaan.

Sen jälkeen aloitin ohjelman tarkemman hahmoittelun, miettiä mahdollisen käyttöliittymän toimintaa, ja miten luokkien pitäisi toimia tämän kanssa. Tein alemman tason luokat valmiiksi siten että pidin niitä jo toimivina ja yhdistin niitä luokassa RecipeBook, ja testasin niiden yhteistä toimivuutta.

Tämän jälkeen oli mahdollista alkaa toteuttaa tekstipohjaista käyttöliittymää UI. Se helpotti testaamista, kun pystyi testata helposti vaihtelevilla syötteillä ja arvoilla, antamalla itse niitä manuaalisesti. Kun tekstipohjainen käyttöliittymä oli karkeasti luonnosteltu ja sen mukainen mitä toivoin olevan, aloitin lataus ja tallennus – operaatioiden tekemisen. Niiden valmistumisen myötä

oli nopeampaa ja helpompaa suorittaa testejä. Tässä vaiheessa minulla oli checkpoint, ja ohjelmani karkea versio oli valmis.

Checkpointin jälkeen ryhdyin testaamaan ohjelmaa monilla eri syötteillä ja hienosäätämään sen ominaisuuksia. Tässä menikin yllättävän kauan, ja ajatukseni graafisen käyttöliittymän tekemisestä venyi ja venyi. Lopulta yleisen ajan puutteen vuoksi, jäi graafinen käyttöliittymä hyvin karkeaan vaiheeseen ja kesken, ja keskityin enimmäkseen ohjelman helppojen vaatimuksien sekä yleisen käytettävyyden testaamiseen ja korjaamiseen.

Arvio lopputuloksesta

Lopulliseen ja palautettavaan ohjelmaan, olen jossain määrin tyytyväinen ja jossain määrin pettynyt. Olin toivonut, että saisin tehtyä graafisen käyttöliittymän, oppiakseni graafista ohjelmointia ja muutenkin ohjelman kivemman toteutuksen puolesta. Kohdan **3 parasta ja 3 heikointa kohtaa** heikot kohdat olisi täytynyt pystyä korjaamaan jo työn alussa, ja paremmalla suunnitelulla ne olisikin palautettaessa paremmat. Ryhdyin ehkä liian nopeasti ohjelmoimaan ilman tarkempaa suunnitelmaa.

Ohjelman toimivuuden kannalta olen ihan tyytyväinen, sillä ainakin omien lukuisten testien myötä, ohjelmassa ei pitäisi tulla odottamattomia erroreita, ja virhetilanteet ylipäätään täytyisivät olla vältetty. On toki mahdollista, että jotkin asiat ovat jääneet huomaamatta. Ohjelma on kuitenkin suorittanut lähes kaikki suunnittelemani ominaisuudet niitä testatessani kuten sen pitääkin, joten siihen olen tyytyväinen.

Koodista olisi voinut koittaa tehdä siistimpää, selkeämpää ja tehokkaampaa. Ajan puutteellisuuden vuoksi jäi nämä tekemättä. Käyttöliittymästäkin olisi voinut koittaa saada jollain tapaa järkevämmän ja selkeämmän. Tekstipohjainen käyttöliittymä jäi varsinaiseksi käyttöliittymäksi, kun alunperin sen oli tarkoitus olla lähinnä graafisen käyttöliittymän pohjana ja suunnitelmana. Jos käyttöliittymän olisi päässyt toteuttamaan graafisesti, olisi siitä saanut ehdottomasti selkeämmän. Mutta toisaalta käyttöliittymästä tuli monipuolinen sekä liikkuminen lukuisten eri menujen välillä ja niiden monet eri ominaisuudet toimivat kuten pitää, joten se vähän paikkaa tuota sekavuutta.

Ohjelmaa pystyy muokkaamaan, mutta se saattaa olla hankalaa, lukuisten luokkien ja niiden yhteisten ominaisuuksien ja toimintojen takia. Muutoksia tehdessä saattaa ne joutua tehdä moneen eri luokkaan ja funktioon. Senkään suhteen ohjelma ei ole täysin ihanteellinen. Mutta toisaalta kun kyseessä on tietokanta-ohjelma, on ne yleisestikin hankalasti muokattavia, ja siksi niiden tekemisessä suunnitteluvaihe on hyvin tärkeä.

Uskon että koitan vielä tulevaisuudessa viimeistellä ohjelmalleni graafisen käyttöliittymän, sillä se parantaisi ohjelman käytettävyyttä hyvin paljon. Jos kerkeän niin saattaa se olla esittelyvalmiina jo ohjelman esittelytilaisuudessa.

Viitteet

En juurikaan käyttänyt mitään lähteitä ohjelmani teossa, vaan hyvinkin pitkälle ratkaisin kaiken itse, jo omaavillani tiedoilla ja taidoilla. Kuitenkin kuten suunnitelmassani mainitsin välillä kätevä apuri on Pythonin oma dokumentaatio, etenkin Pythonin sisäänrakennetuista ominaisuuksista. Se onkin jäänyt ainoaksi lähteeksi ohjelmani teossa.

Seuraava on lainattu kirjoittamastani teknisestä suunnitelmasta koskien viitteitä ja lähteitä: ”Suurimpana apuna Python-ohjelmoinneissa olen aina käyttänyt, ja tulen käyttämään, Pythonin omaa dokumentaatiota, sivustolla http://docs.python.org/2.7/contents.html. Muuten olen ohjelmoinneista tähän mennessä selvinnyt hyvin pitkälti ilman muuta materiaalia, enkä ole ainakaan suunnitellut käyttäväni muuta tässäkään projektissa. Joskus hyviä neuvoja löytyy sivustolta http://stackoverflow.com/, jossa on usein ohjelmoijia etsimässä ratkaisuja samantyyppisiin ongelmiin kuin itse. Usein etsin vain miten jokin asia teknisesti tehdään Pythonissa tai mitkä ovat mahdollisia tapoja tehdä. Tietyt komennot, sekä valmiit kirjastot, ja niiden käyttö löytyy kätevästi Pythonin dokumentaatiosta.”

Liitteet

Dokumentaation yhteydessä on lähdekoodini, erilliset valmiit tekstitiedostot, jotka sisältävät valmiita ohjelmaan ladattavia reseptejä, varastoja sekä raaka-aineita ja pari rikkinäistä tiedostoa, jotka ohjelman pitäisi huomata olevan väärin muodostettuja (eli nostaa LoadAndSaveError:n luokassa LoadAndSave).