Umeå Universitet

Släktträd på webben

Labbrapport – Uppgift 3.

Applikationsprogrammering i Python 7,5 HP.

Kristoffer Karlsson

CS-användare: kv14kkn

CAS-användare krka0050

2016 – 06 – 03

Handledare; Johan Eliasson, Alexander Sutherland

Innehåll

Problemspecifikation

Uppgiften går ut på att skapa ett släktträd där användaren får skriva in släktingar och ange *Förnamn, Efternamn, Född, Död (om aktuellt), Mamma* och *Pappa*. Jag valde att lägga till en fakta/kommentarsmöjlighet.

Uppgiften ska göras med ett gränssnitt via en webbsida och använda pythonscript via CGI-filer. Programmet ska kunna räkna ut relationerna *Förälder till*, *Barn till* och *Syskon till*, samt kommunicera med en databas (Sqlite3).

Det ska via websidan gå att lägga till, redigera och ta bort personer, samt gå att visa detaljerad information om en person. Sidan ska även kunna visa en lista över alla personer i en släkt.

Jag ville göra en användarvänlig sida som kan presentera användarens släktträd på ett smidigt sätt och kunna lagra träd för snabb åtkomst senare. Ett släktträd uppdatteras ibland och en användare vill inte ange alla släktingar varje gång. Jag ville även ha en tydlig översikt över vilka familjer som fanns i databasen.

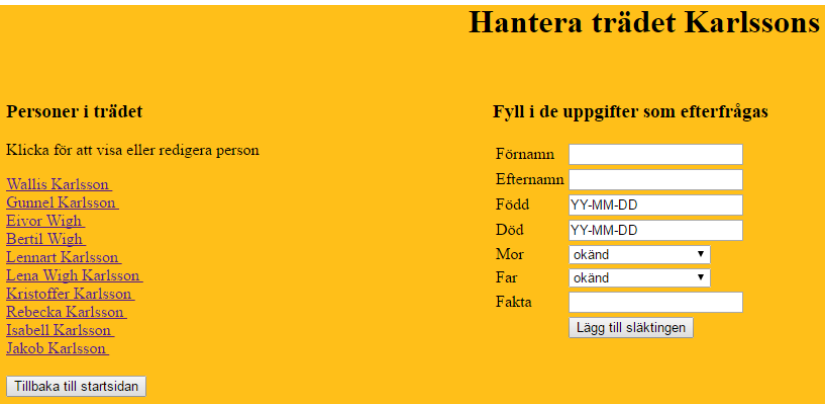
Originalinstruktionerna går hitta här https://www.cambro.umu.se/portal/site/57100VT16-1/page/246a177e-52f3-4145-9a88-ca8aa1bc0cfd

Åtkomst och användarhandledning

För att komma till webbsidan, använd länken

https://www8.cs.umu.se/~kv14kkn/5da000/familytree/

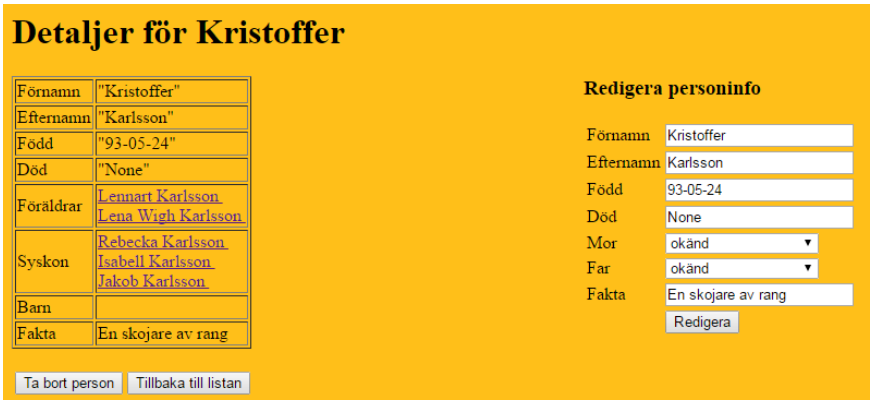
Börja med att skriva in eller klicka på ovanstående länk. Logga in med din CS-användare. Filen index.cgi exekveras då och skriver ut sidan. Ange sedan ditt familjenamn (OBS, du kan aldrig använda dig av Å, Ä eller Ö). Om du skapat ett träd tidigare eller om namnet redan används kommer du till det befintliga trädet, annars skapas ett nytt.



Detta sker via filen addtable.cgi som körs när knappen trycks på. Filen skapar sedan nästa sida. Vill du se ett demoträd kan du skriva in ”Karlssons” i textrutan.

När du skapat/laddat trädet kommer du till ditt träd och presenteras för ett verktyg att bygga det med. Du lägger till personer med verktyget genom att skriva in *Förnamn*, *Efternamn*, *Född*, *Död*, *Mor*, *Far* och *Fakta*. Om du inte kan en uppgift; lämna rutan tom. För föräldrar gäller en speciell princip. För att ange förälder måste denne finnas inlagd sedan tidigare. Gör den inte det kan man lägga till den sedan när man redigarar personen. Personen sparas i en databas. Detta via filen addmember.cgi

Klickar man på knappen ”Tillbaka till startsidan” tas man tillbaka till startsidan via filen index.cgi.



För att visa personens detaljer och/eller redigera personen; klicka på ett av namnen till vänster på sidan. Då exekveras filen persondetail.cgi som bygger detaljsidan. Den hämtar all information om personen i databasen och räknar ut relationerna. Du erbjuds att redigera personen. Om detta görs körs filen modify.cgi. Du kan också välja att ta bor personen. Då körs filen deleteperson.cgi. Personen tas bort ur databasen, och du kommer tillbaka till föregående sida. Man kan också trycka på ”Tillbaka till listan” och skickas då tillbaka till byggsidan via goback.cgi utan att ändra något.

Algoritmbeskrivning

1. Programmet skriver ut en webbsida som ber användaren om ett trädnamn. Databasen skapas om den inte redan finns, samt en tabell som listar alla familjer om den inte redan finns. Här skrivs användarens familjenamn in, och en ny tabell för aktuell familj skapas, eller så laddas en tabell som redan finns och förnamn och efternamn på alla personer skrivs ut på nästa sida.
2. Sedan skriver programmet ut en webbsida som erbjuder användaren att skriva in nya släktingar. Släktingarna skrivs in i databasen. Föräldrar kan bara väljas om de finns inlagda i databasen sedan tidigare. Detta för att det ska fungera korrekt i databasen. Personens förälder får en id-representation som refererar till en annan person i databasen. Finns föräldern inte inlagd kan användaren ange ”okänd”. Id-representationen blir då 0. Det går sedan redigera detta.
3. När en släkting läggs in uppdatteras sidan och en uppdaterad lista av användarens släktingar skrivs ut och varje person blir en länk. Listan skapas genom att plocka ut namn och id på alla personer i aktuell familjs tabell i databasen.
4. Klickar användaren på en släkting plockas all personens information ut ur databasen och sparas som ett objekt.
    a. *Förnamn, Efternamn, Född, Död* och *Fakta* plockas ut direkt.
    b. *Föräldrar* plockas ut genom att plocka ut personer med samma id som personens far och personens mor.
    c. *Syskon* plockas ut genom att plocka ut de personer som har samma mor-id och far-id om föräldrarnas id inte är 0 eller samma som personens. Detta för att alla personer med okänd förälder har föräldra- id 0, och alla dessa är inte syskon. Man vill heller inte att personen ska vara syskon till sig själv.
    d. *Barn* plockas ut genom att kolla på alla personers mor-id och far-id. Om någon av dessa är samma som personens id så är dessa barn till personen.
5. På detaljsidan kan användaren även ta bort personen ur databasen. Användaren dirigeras sedan vidare till föregående sida.
6. Användaren kan också redigera personen. De flesta uppgifter är förifyllda och användaren behöver bara fylla i det som den vill ändra. Personen redigeras då i databasen, och användaren skickas tillbaka till startsidan.
7. Användaren kan också välja att gå tillbaka till trädet utan att ändra något.

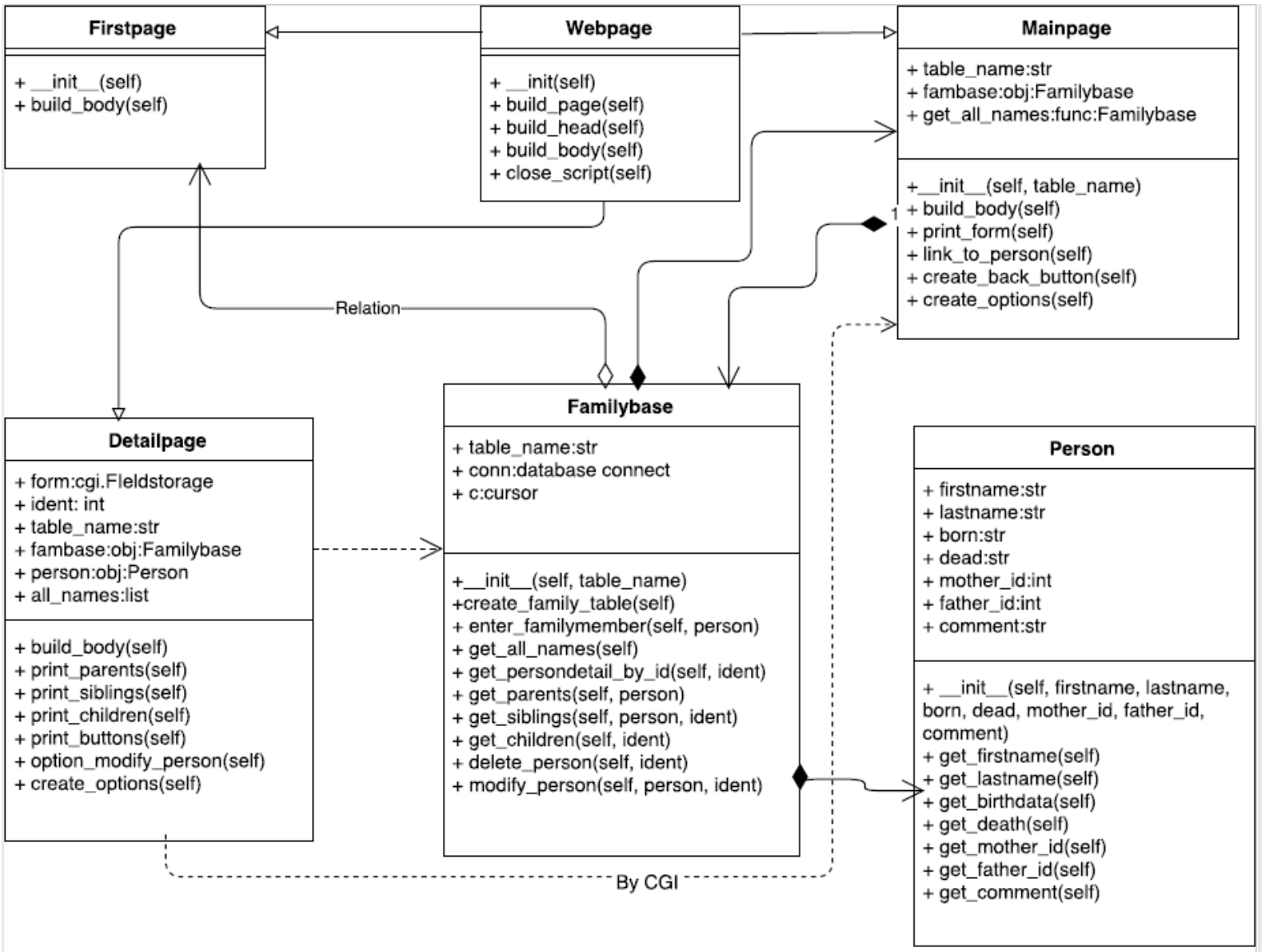
Systembeskrivning

Klasser:

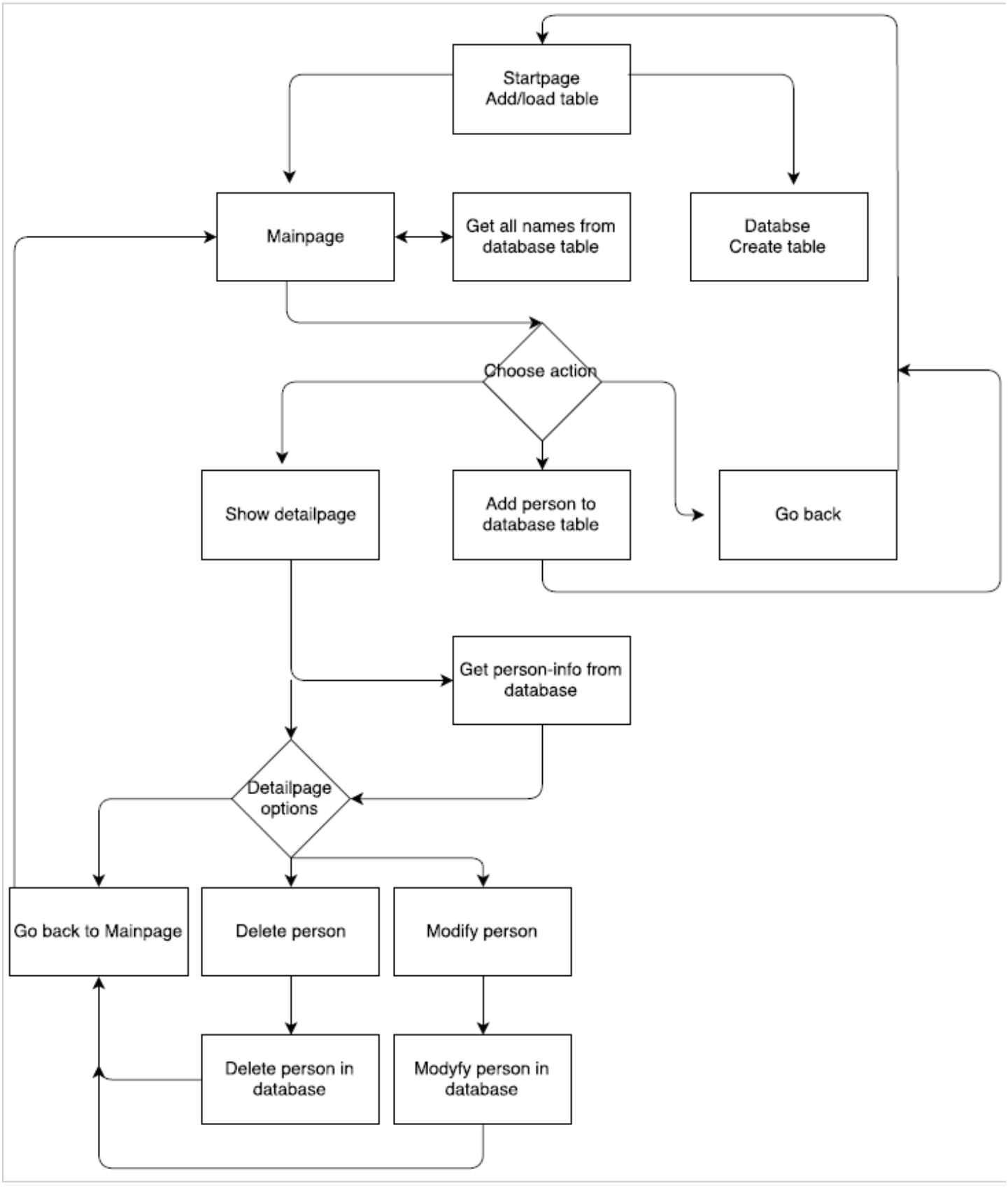
- **Webpage**
  - En tom websida som skriver ut sidans head och stänger scriptet. De andra websidorna ska ärva denna och förändra dess funktion och utseende; build_body().
- **Firstpage**
  - Hit kommer man från CGI-filen index.cgi
  - Ärver från Webpage och förändrar build_body funktionen
  - Tar in input från användaren i ett HTML-form. Inputen är ett tabellnamn/familjenamn
  - Inputen i form:et skickas till cgi-filen addtable.cgi som kontaktar klassen Familybase där familjenamnet skrivs in i databasen, samt skapar en ny tabell för familjen, om den inte finns sedan innan.
  - Användaren skickas sedan till klassen Mainpage.
- **Mainpage**
  - Skriver ut alla personer i en familj med hjälp av Familybase-klassen.
  - Tar input från användaren i form av personinformation som den skickar till Familybase-klassen via CGI-filen addmember.cgi.
  - Addmember.cgi skickar till Mainpage igen.
  - Användaren kan klicka på en knapp för att komma tillbaka till Firstpage-klassen. Då kommer man till index.cgi som skriver ut Firstpage-klassen.
  - Användaren kan klicka på en personlänk. Då körs persondetail.cgi som kontaktar Detailpage-klassen
- **Detailpage**
  - Körs via persondetail.cgi och skriver ut all data som finns för användaren, samt räknar ut syskon, föräldrar och barn om sådana finns. Detta sker med hjälp av Familybase-klassen
  - Anändaren kan välja att
    - Modifiera – då körs modifyperson.cgi – Familybase kontaktas.
    - Ta bort – då körs deleteperson.cgi – Familybase kontaktas.
    - Gå tillbaka till Mainpage – då körs goback.cgi. Mainpage körs
    - Vid alla tre tas användaren tillbaka till Mainpageklassen.

- **Familybase**
  - Hanterar all databaskommunikation.
  - Har kopplingar till alla andra klasser.
- **Person**
  - Skapar ett personobjekt.
  - Har kontakt med Familybase, Detailpage (indirekt via Familybase och även via CGI vid redigering av person) och Mainpage (via CGI).

Klassdiagram



Flödesschema



Lösningens begränsningar.

- **Cgi.FieldStorage() –** Jag ställdes inför problemet att FieldStorage bara verkade kunna plocka fram ett värde en gång, och eftersom jag i vissa CGI-filer behövde mitt table name i flera klasser blev jag tvungen att ta in all input från FieldStorage i de CGI-filerna. Detta var inte en så snygg lösning, men det var tvunget för att få det att fungera.

addmember.cgi

```
#!/usr/local/bin/python
# -*- coding: iso-8859-1 -*-
import cgi, cgitb
from mainpagefile import Mainpage
from familybasefile import Familybase
from personfile import Person
cgitb.enable()
"""Imports the users input and creats a person object with the person-class.
The object is being sent to the database in the Familybase-class. Then the
webpage is being created with updated info."""

form = cgi.FieldStorage()
table_name = str(form.getvalue('tree_name'))
firstname = str(form.getvalue('firstname'))
lastname = str(form.getvalue('lastname'))
born = str(form.getvalue('born'))
death = str(form.getvalue('death'))
mother_id = int(form.getvalue('mother_id'))
father_id = int(form.getvalue('father_id'))
comment = str(form.getvalue('comment'))
person = Person(firstname, lastname, born, death, mother_id, father_id, comment)
fambase = Familybase(table_name)
fambase.enter_familymember(person)
page = Mainpage(table_name)
page.build_page()
```

- **Teckenkodning och Å, Ä Ö –** Jag försökte ge mig på att byta teckenkodning och print till enc_print enligt vad vi fått lära oss under kursen. Men jag fick det ej at fungera. Koden hängde upp sig när jag använde strängformatverktyget %s. Kanske skulle detta problem gå att lösa på ett annat vis och jag vet att jag skulle kunna klura ut det om jag haft någon dag till på mig. Blev nu tvungen att välja bort detta för att hinna med labbrapport och en duglig layout på mina sidor.

- **Relationsprolem –** Personer med bara en förälder kommer inte bli syskon till den förälderns andra barn. Detta är ett medvetet val och kan ändras genom ett utbyte i SQL-satsen (AND mot OR). Men jag gillade detta bättre eftersom man inte kan veta om den personen verkligen är helsyskon med de andra. Den kan tillexempel ha en annan pappa än de andra barnen. Användaren kan ange detta som en anteckning tills vidare.

```
def get_siblings(self, person, ident):
    """ Takes out a persons siblings using ident from the person and puts
    it in a list of dictionarys. Siblings is everyone that has the same
    father and mother ident. It checks so the sibling is not the person
    itself and if the siblings has parents ident = 0 it does not add it.
    0 is default if parents are unknown."""

    sibling_list = []
    self.c.execute('''SELECT ident, firstname, lastname, mother_id,
    father_id FROM ''' + self.table_name + ''' WHERE mother_id = ? AND
    father_id = ?
    ''', (person.get_mother_id(), person.get_father_id()))
```

- **Familjetabeller –** Jag valde inledningsvis att skapa en tabell där alla familjenamn lagras för att man skulle kunna få en bra översikt över vilka familjer som finns i

databasen, och att varje familj lagras som en egen tabell. Detta gör tyvärr att varje familj blir isolerad och släktträdet kan inte expandera ut bland användare. Detta skulle kunna lösas genom en förändring i databasens struktur där man har två eller tre tabeller. En där familjenamnen lagras, precis som nu. En där alla personer i hela databasen lagras och har en egenskap som är familjenamnets id. Sedan relationerna antingen så som jag har det nu att man refererar till en förälder i samma tabell genom id eller i en annan tabell som bara hanterar föräldrarelationer. Man skulle då kunna visa föräldrar och syskon till andra familjeträd. Dock hade det blivit ett problem i mina rullgardinsmenyer där man väljer föräldrar. De hade blivit otroligt långa efter ett tag.

Problem och reflektioner.

Jag har lärt mig otroligt mycket under denna uppgift och hade jag börjat idag hade jag gjort en del annorlunda. Största skillnaden hade nog återfunnits i databasen. Jag hade gärna sett att vi fått övningsuppgifter för att lära oss rätt tänk. Jag känner mig även osäker på klassdiagram. Det hade varit givande om vi fått en större feedback på tidigare uppgifter för att bättre förstå de olika typerna av klassrelationer.

Jag har använt mig av person-objekt i den mån jag tyckt det var nödvändigt. Jag fick inspiration av Johan Eliasson att göra dessa personobjekt, men föredrog vid de flesta utplock ur databasen att göra en lista av dictionaries som innehöll den personinformation jag behövde för uppgiften. Det var ofta samma information (id, förnamn och efternamn), vilket gör att jag kanske borde ha gjort en ny klass som skapade ett objekt av den informationen. Anledningen till att jag inte skapade ett person-objekt i min Personklass dessa gånger var att jag ansåg det onödigt att ta ut mer information i databasen än jag behövde.

Jag har försökt vara noga med att hålla mig till rätt antal tecken på varje rad, men när jag skrivit länkar till olika CGI-filer har de dragit över 80 tecken och det har inte gått att radbryta dem eftersom det då blir ett avbrott i länken.

Tack till:

- Johan Eliasson – Föreläsare på kursen Applikationsprogrammering i Python för bra föreläsningar där jag fått mycket av den kunskap jag använt. Jag har även fått bra hjälp med tips och vägledning fram till lösningar på problem som jag haft. Tillexempel har jag använt mig av arv på webbsidorna enligt Johans modell och select-taggen för att ange föräldrar.
- Youtube-profilen Sentdex som har bra och pedagogiska videos. Jag lärde mig mycket av hans genomgång av databaser. https://www.youtube.com/channel/UCfzlCWGWYyIQ0aLC5w48gBQ
- http://www.w3schools.com/html/ - Som har bra och pedagogiska genomgångar om allt som rör HTML och CSS.