2020



Eksamensprojekt

**Max Hansen**

Odense tekniske Gymnasium

6. maj 2020

Programmering - B

**Økonomi**

Denne synopsis indeholder udførelsen af et program

der skal hjælpe med at give et overblik over brugerens

økonomi.

# Titelblad

## Vejleder

Søren Præstegaard - SPR

## Fag

Programmering B

## Klasse og skole

3.D - Odense Tekniske Gymnasium

## Antal sider

X A4-sider

X sider à 2400 tegn

## Dato for udlevering

11-03-2020

## Dato for aflevering

07-05-2020

# Indledning

Økonomi er et vigtigt begreb i dagligdagen som alle mennesker skal holde styr på, det kan til tider være svært at have et overblik hvor mange penge man har haft på forskellige datoer. Jeg arbejder derfor med at lave et program som har en grafisk brugerflade, som ved hjælp af en database kan hjælpe brugeren med at holde lidt styr på brugerens økonomi. I denne synopsis vil jeg dokumentere min arbejdsgang med udviklingen af programmet. Her vil jeg komme ind på de forskellige metoder fra programmeringsfaget som jeg har brugt til planlægningen og udførelsen af programmet.

# Planlægning af programmet

## Krav

For at kunne lave et program, så kræver det at man sætter sig nogle krav for hvad der skal være med i programmet. Kravene skal også være med til at sætte nogle rammer for hvordan programmet skal fungere når man bruger det som bruger. Dette gør det nemmere for programmøren at programmerer programmet. Jeg har derfor lavet 3 krav til programmet som skal sikre at programmet fungere som planlagt, samtidigt med at programmet har de funktionaliteter som der vil være passende.

### Krav 1 - GUI

Programmet skal have en GUI. På programmets GUI skal brugeren blandt andet kunne indtaste hvor ofte brugeren får løn, hvor meget brugeren får i løn, hvor mange penge brugeren har brugt/fået siden sit sidste besøg. Brugeren skal til det sidste have mulighed for at kunne lave sine egne kategorier såsom ”Brændstof”, som kan opdele brugerens økonomi i kategorier som brugeren har lavet. De forskellige informationer skal være vist på GUIen og brugerens kontosaldo skal vises med en graf.

Programmets GUI skal starte med at bede brugeren om at logge ind, eller lave en konto. Derefter vil brugeren kunne komme ind og ændre sin økonomi.

### Krav 2 - Administrering af økonomi

Programmet skal kunne administrere brugernes økonomi igennem udregninger. Programmet skal kunne administrere økonomien, ved at udregne hvor mange penge der er tilbage på kontoen, hvis brugeren har brugt penge osv. Programmet skal også kunne holde styr på løn fra brugerens arbejde, hvis brugeren har tilføjet sit arbejde.

### Krav 3 - Visning af økonomi

Programmet skal via programmet GUI vise brugerens nuværende kontosaldo. Programmet skal også via programmet GUI kunne vise forløbet for brugerens kontosaldo for de datoer der er sket ændringer i brugerens kontosaldo gennem en graf. Visningen af disse økonomiske genstande skal kunne give brugeren et hurtigt overblik over sin kontosaldo igennem dagene.

## brugerhistorier

Jeg har til mit program valgt at lave 3 brugerhistorier. Brugerhistorierne vil jeg bruge til at beskrive nogle mere detaljerede krav og en beskrivelse af hvordan brugeren skal interagere med programmet. Jeg har her valgt at beskrive de 2 brugerhistorier som står for hoveddelen af mit program. I Bilag 1 har jeg den sidste brugerhistorie, som beskriver hvordan brugeren tilføjer og fjerner sit job. I afsnittet ”Brugerhistorie x - Something” vil jeg beskrive hvordan jeg har udført de forskellige brugerhistorier igennem forskellige iterationer.

### Brugerhistorie 1 - Login system

Denne brugerhistorie kan kun startes når programmet bliver åbnet af brugeren.

* Brugeren åbner programmet
* Programmet registrerer at brugeren ikke er logget ind og viser skærmen for login
* Hvis brugeren har en konto
  + Brugeren indtaster sit brugernavn og adgangskode i de tilhørende felter
  + Brugeren trykker på ”login”
  + Programmet logger brugeren ind og viser hovedsiden for programmet
* Hvis brugeren ikke har en konto
  + Brugeren trykker på knappen ”Don’t have an account? Sign up here”
  + Programmet viser ”Sign up” skærmen
  + Brugeren indtaster sit brugernavn, fornavn, efternavn, adgangskode, bekræftelse af adgangskode og e-mail ind i de tilhørende felter
  + Brugeren trykker på ”Create account”
  + Programmet tjekker om brugernavnet er ledigt
  + Hvis brugernavnet ikke er ledigt
    - Programmet laver en fejl, hvor der bliver beskrevet hvad fejlen er. Teksten ”username” bliver vist med rødt. Brugeren kan ændre brugernavn og prøve igen
  + Hvis brugernavnet er ledigt
    - Programmet tjekker om felterne ”password” og ”confirm password” er ens
    - Hvis felterne er ens
      * Programmet opretter brugerens konto og viser login siden
    - Hvis felterne ikke er ens
      * Programmet laver en fejl, hvor der bliver beskrevet hvad fejlen er, og felterne der er forkerte, bliver vist med rødt. Brugeren kan rette felterne og prøve igen

### Brugerhistorie 2 - Hvor mange penge har brugeren brugt eller fået

Denne brugerhistorie kan kun startes når brugeren er logget ind på programmet, og befinder sig på programmets hovedside.

* Hvis brugeren vil tilføje/fjerne penge
  + Brugeren vælger en kategori i dropdown-menuen ved ”Select category for obtained/used money”
  + Brugeren finder inputfeltet ved enten ”Money obtained” / ”Money used”, og indtaster hvor mange penge brugeren har fået / brugt
  + Brugeren trykker på knappen ”Add money obtained” eller ”Add money used”
  + Programmet tjekker om inputfeltet kun består af tal
  + Hvis feltet ikke kun består af tal
    - Programmet laver en fejl og skriver på skærmen ”Please make sure you only used numbers!”, og teksten ved inputfeltet bliver vist med rødt. Brugeren kan rette og prøve igen.
  + Hvis feltet kun består af tal
    - Programmet tjekker om brugeren har valgt en category
    - Hvis brugeren ikke har valgt en category
      * Programmet laver en fejl og skriver på skærmen ”Please select a category!”, og teksten ved dropdown-menuen bliver vist med rødt. Brugeren kan rette og prøve igen.
    - Hvis brugeren har valgt en category
      * Programmet modtager tallet fra inputfeltet og kategorien og indsætter i databasen for brugeren.
      * Programmet opdaterer programmets GUI og grafen for brugerens saldo ved den nuværende dato er opdateret og viser den nye kontosaldo.

## iterationer

Til programmet har jeg valgt at lave 3 iterationer. Iterationerne skal hjælpe mig med at lave programmet i en rækkefølge som giver mening. Iterationerne hjælper også med at tjekke programmets kvalitet, da hver iteration skal testes og godkendes af programmøren, før man starter på den næste iteration. Hver iteration udføres i en bestemt rækkefølge. Der startes først med at planlægge iterationen, derefter udfører man iterationen som så leder videre til testen af iterationen. Består iterationen testen, så er den godkendt og man kan fortsætte med den næste iteration. Består iterationen ikke testen, så starter man forfra.

### Iteration 1 - Login system

Den første iteration kommer til at omhandle ”Brugerhistorie 1 - Login system”, hvor selve login systemet laves. Der er altså kun én brugerhistorie med i denne iteration.

### Iteration 2 - Hovedmenu

Den anden iteration kommer til at omhandle ”Brugerhistorie 2 - Hvor mange penge har brugeren brugt eller fået” og ”Brugerhistorie 3 - Add job / Remove job”. Disse to brugerhistorier er med til at danne hovedmenuen for mit program, så de er helt essentielle.

### Iteration 3 - Graf

Den tredje og sidste iteration kommer til at omhandle ”Krav 3 - Visning af økonomi”. Dette er det sidste krav, og er et krav som ikke er inddækket ordentligt af brugerhistorierne, da det er noget programmet skal gøre automatisk.

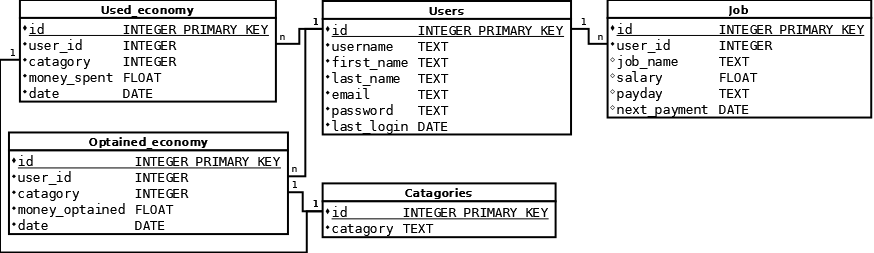
# Udførelse af programmet

I dette afsnit kommer jeg ind på hvordan jeg har fået udført mit program. Dette kommer til at omhandle både, hvordan jeg har opsat min database, men også hvordan de forskellige iterationer er blevet udført.

## Oprettelse af databasen



Figur 1, Eksempel på oprettelse af tabeller

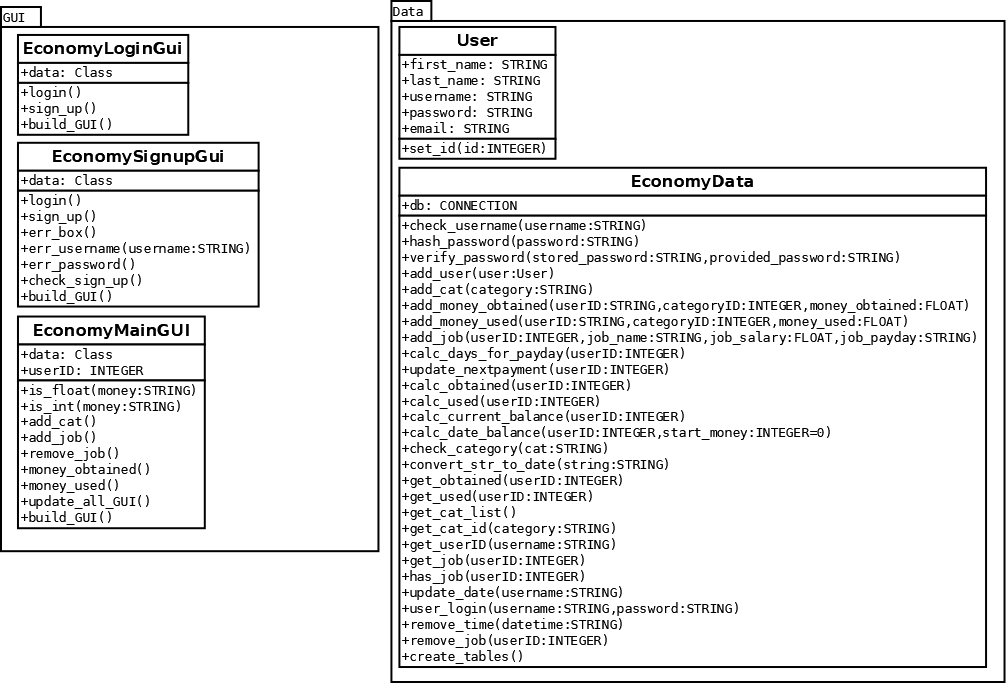


Figur 2, ER-diagram, figurern er vedhæftet under ”Database\_diagram.png” i ”Documents” mappen

Databasen er en helt essentiel del i mit program. Uden databasen ville hele mit program ikke komme til at virke.

For at lave min database, så startede jeg med at lave et ER-diagram. ER-diagrammet er med til at give mig et overblik over hvordan databasen skal sættes op, hvilke tabeller der skal være i databasen og hvad de tabeller skal indeholde. På figur 1, kan ER-diagrammet for min database i programmet ses. I mit program er der altså 5 tabeller der hver især opbevarer forskellige data. Til at lave databasen bruger jeg biblioteket ”SQLite3”, hvilket er et bibliotek som gør at man kan skrive SQL, hvilket er det programmeringssprog databaser arbejder med. SQLite3 gør det muligt at lave en lokal database til ens program, så informationer kan gemmes. For at lave tabellerne i mit program så kører jeg funktionen ”create\_tables()”, se Figur 2 linje 1. Denne funktion starter med at slette alle nuværende tabeller i databasen, hvis de eksisterer, dette foregår med et på Figur 2 linje 4 - 9. Efter funktionen har fjernet de nuværende tabeller, så bliver tabellerne så oprettet igen. På Figur 2 linje 14 - 21, vil programmet prøve at lave tabellen ”Users”, som skal indeholde brugeroplysninger til loginsystemet. Det samme gør sig gældende for alle de andre tabeller i databasen. Det gode ved at fjerne alle tabellerne er at programmet så er frisk når det er færdig udviklet. Man skal dog huske ikke at køre funktionen når programmet er færdigt, da ALT data forsvinder.

## Oprettelse af klasserne

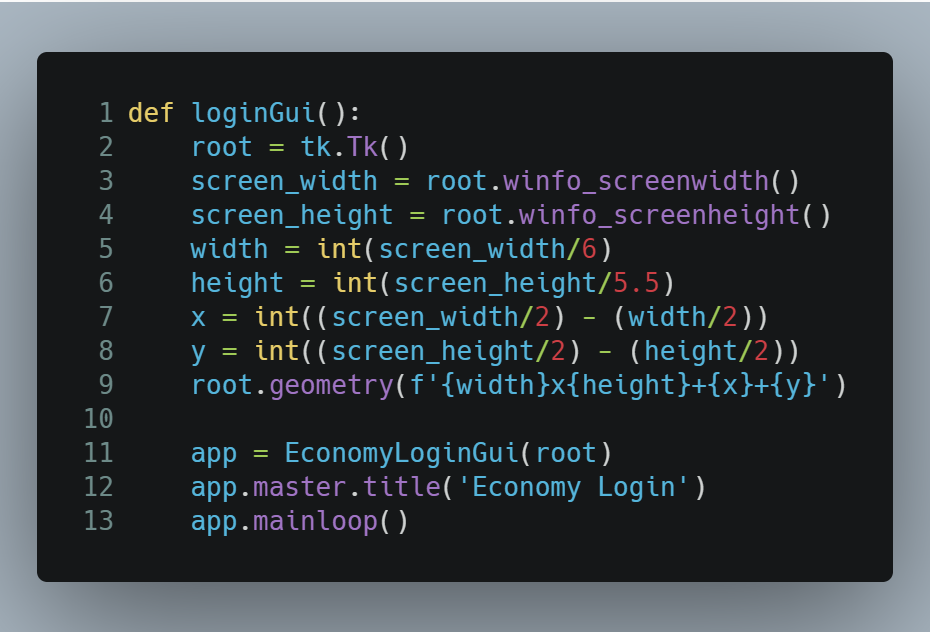


Figur 3, Klasse-diagram, figurern er vedhæftet under ”Klasse\_diagram.png” i ”Documents” mappen

Klasser i et program er en måde at opdele programmet i flere enkeltstående dele. For at mit program nemt kan arbejde med alt dataen fra databasen, så har jeg lavet en klasse, hvor funktionerne på en eller anden måde hører sammen med databasen. Denne klasse kan ses på Figur 3 under ”Data”. Her er der to forskellige klasser. Der er klassen ”User”, som skal lave brugeren af programmet til et objekt, når brugeren skal oprettes i databasen. Den anden klasse hedder ”EconomyData” og er den klasse som tidligere beskrevet, arbejder sammen med databasen. Klassen EconomyData er hoved-klassen i mit program. Det er blandt andet i denne klasse at alle funktionerne til de forskellige brugerhistorier befinder sig. På Figur 3 under ”GUI” er der 3 forskellige klasser. Hver klasse er her et ”vindue” i programmet. Det vil sige at vinduet, hvor brugeren kan logge ind er fra klassen ”EconomyLoginGui”, hvor selve hovedprogrammet bliver lavet af klassen ”EconomyMainGui”. Hver klasse har her også nogle specifikke funktioner som har med deres område at gøre. Klassediagrammet som er på Figur 3 bruges til at give et indblik over, hvordan programmet skal sættes op så programmet har den ønskede virkning. Derudover giver det et godt overblik over, hvilken klasse der står for, hvilken del af programmet. Klassen EconomyLoginGui består af 3 funktioner, Figur 3. Funktionen ”build\_GUI” er den funktion som laver alt det som brugeren kan se. Dette er lige fra tekstfelter til knapper osv.

## Udførelse af iterationer

### Iteration 1 - Login system



Figur , funktionen loginGui(). Fil: EconomyGui.py

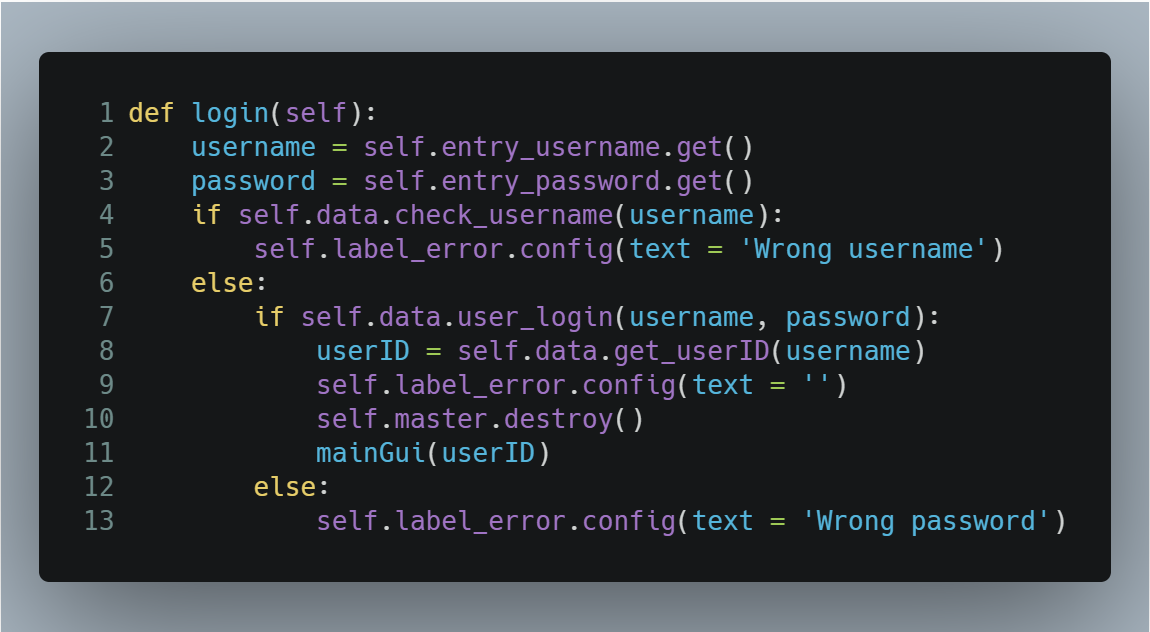
Udførelsen af den første iteration, hvilket var ”Brugerhistorie 1 - Login system” har krævet at jeg havde lavet 3 ud af de 5 klasser som var på Figur 3. Klasserne som loginsystemet krævede, er EconomyData, EconomyLoginGui og User. Jeg skulle her få klasserne til at snakke sammen, og på den måde ende med et fungerende login system.

Jeg startede med at få lavet vinduet til min loginskærm. Det er denne skærm som programmet altid skal starte med at åbne. Jeg lavede derfor funktionen ”loginGui()” som kan ses på Figur 4. Funktionen her bliver kørt som det først, hver gang programmet åbner. Funktionen starter med at lave en root (Figur 4 linje 2), hvilket er det der modtager alt info og giver alt informationen om størrelsen af vinduet osv. videre til klassen som tegner selve GUIen. På Figur 4 linje 3 - 9 definerer jeg så størrelsen og placeringen af selve loginvinduet. Dette bliver udregnet ud fra brugerens nuværende skærmstørrelse



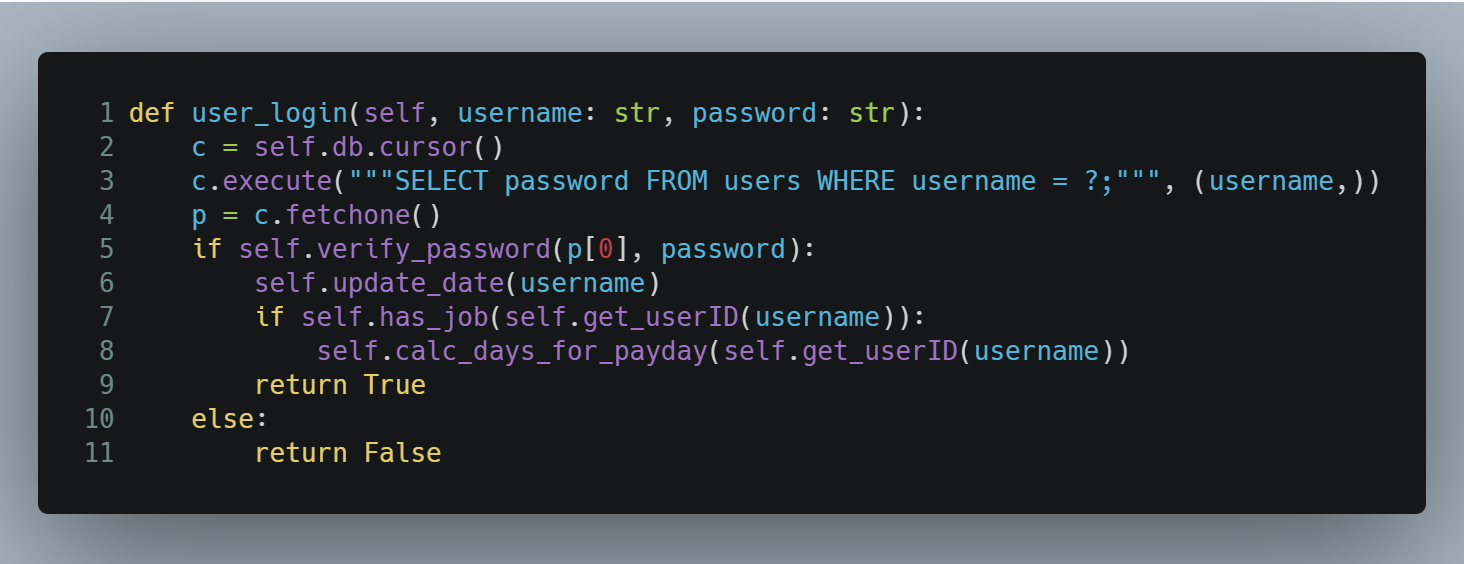
Figur , eksempel af funktionen build\_GUI. Fil: EconomyGui.py

(Figur 4 linje 3 & 4) som jeg så kan bruge til at udregne en størrelse og placering af vinduet som er ens for alle skærme. Til sidst definerer jeg, hvilken klasse som vinduet skal vise, i dette tilfælde er det EconomyLoginGui, se Figur 4 linje 11 - 13. I klassen findes funktionen ”build\_GUI”, det er denne funktion som tegner alt hvad brugeren kan se på skærmen. På Figur 5 er der et lille eksempel på et af de ”labels” som login skærmen består af. I linje 2, definerer jeg variablen ”self.label\_username” til at være et label fra biblioteket ”Tkinter”, hvor der står ”Username:”. I linje 3 definerer jeg, hvor det label skal placeres henne på skærmen. Dette sker igennem rækker og kolonner ligesom i en tabel i et excel dokument. Til sidst bliver det grafik jeg har lavet pakket ind til vinduet så det sidder som det skal og bliver vist på skærmen, se linje 4.



Figur , funktionen login(). Fil: EconomyGui.py

På Figur 6 ses funktionen som logger brugeren ind. Funktionen starter med at modtage brugerens brugernavn og adgangskode i linje 2 og 3. Herefter tjekker programmet om brugernavnet findes i databasen. Dette sker med funktionen ”self.data.check\_username()” som er en funktion der er i min data klasse. Hvis brugerens brugernavn så kommer der en fejlbesked på brugeren skærm, linje 5. Eksistere brugernavnet dog, så tjekker programmet om brugerens indtastede kode er det samme som den kode som brugeren brugte når brugeren opretter sin konto. Dette sker i funktionen ”self.data.user\_login()” som igen er en funktion i min data klasse. Er Koden korrekt, så modtager programmet brugerens id, som skal bruges til alle de andre ting, og derefter bliver vinduet fjernet også så åbnes hovedvinduet, hvilket er mainGui, dette sker fra linje 8 til linje 11. Hvis koden var forkert, ville der komme en fejlbesked med at koden er forkert, se linje 12 og 13. På Figur 7 ses funktionen ”user\_login()”. Det er denne funktion som bliver kaldt når brugeren logger ind. Funktionen modtager 2 parametre som er brugerens brugernavn og brugeren kode, se linje 1. Herefter finder funktionen brugerens kode som er gemt i databasen, og tjekker om koden er korrekt. Dette sker i line 2 til 5. Hvis koden her er korrekt, så bliver brugerens sidste login dato opdateret til den nuværende dato af funktionen ”update\_date()”, herefter tjekker programmet om brugeren har tilføjet et job. Hvis brugeren har et job tilføjet, så udregner programmet datoen, hvornår brugeren skal have løn igen i funktionen ”calc\_days\_for\_payday()” i linje 7 til 8. Funktionen ”calc\_days\_for\_payday()”, tilføjer også automatisk lønnen ind på brugerens konto, hvis datoen for at have løn er oversteget, eller er den samme som den nuværende dato. Mere om dette senere. Loginsystemet bortset fra sign up er nu lavet.



Figur , funktionen user\_login(). Fil: econodata.py

# Test af programmet

## Test 1

## Test 2

## Test 3

# Beskrivelse af arbejdsprocessen

## Trello

Jeg har i mit projekt haft brug for at have et overblik over de delopgaver som jeg skal have arbejdet med. Jeg har derfor brugt hjemmesiden trello.com til at hjælpe mig med at holde et overblik over mit projekt, hvilket har sikret mig en let tilgang til at holde styr på hvor meget der mangler osv. Man kan på de forskellige elementer angive hvornår de skal være færdige, så man ender med at bruge tiden fornuftigt.

## Tidsplan

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Moduler  (12 moduler  i alt) | 2 | 4 | 2 | 1 | 3 |
| Arbejdsopgaver | Krav specifikation til programmet | Program opsætning | Database opsætning | Test af programmet | Synopsis skrivning |

# Litteraturliste

# Bilag

## Bilag 1

### Brugerhistorie 3 - Add job / Remove job

Denne brugerhistorie kan kun startes når brugeren er logget ind på programmet, og befinder sig på programmets hovedside.

* Hvis brugeren vil tilføje et job
  + Programmet tjekker om brugeren allerede har et job
  + Hvis brugeren ikke har et job
    - Brugeren skriver navnet på brugerens job i inputfeltet ved ”Add job name”
    - Brugeren skriver lønnen som brugeren får i inputfeltet ved ”Add job salary”
    - Brugeren skriver dagene der er mellem hver gang brugeren får løn i inputfeltet ved ”Days between payments”
    - Brugeren trykker på knappen ”Add job”
    - Programmet tjekker om alle 3 felter er udfyldte
      * Hvis der er én eller flere felter der ikke er udfyldte
        + Programmet laver en fejl og skriver en passende fejltekst alt efter hvilke felter der mangler. Teksten ved inputfeltet bliver markeret med rødt, og brugeren kan rette og prøve igen.
      * Hvis alle felter er udfyldte
        + Programmet tjekker om der kun er tal i inputfelterne ved ”Add job salary” og ”Days between payments” (”Days between payments må kun være heltal”).
        + Hvis felterne ikke kun består af tal

Programmet laver en fejl og skriver ”Please make sure you only used numbers!”, og teksten ved inputfeltet bliver markeret med rødt. Brugeren kan rette oplysningerne og prøve igen.

* + - * + Hvis felterne kun består af tal

Programmet modtager de forskellige informationer fra inputfelterne og tilføjer dem til databasen.

Programmet opdaterer programmets GUI og viser nu oplysningerne for brugerens job.

Programmet udregner hvornår brugeren får løn næste gang og viser datoen for næste løn på skærmen.

* + Hvis brugeren har et job
    - Programmet laver en fejl og skriver ”You already have a job. Remove your job before adding a new job.”
    - Brugeren kan fjerne sit job og prøve igen.
* Hvis brugeren vil fjerne sit job
  + Brugeren trykker på knappen ”Remove job”.
  + Programmet fjerner brugerens job oplysninger fra databasen op opdaterer programmets GUI så informationer om brugerens job ikke vises længere.