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.

Til login systemet, for at encrypte og decrypte adgangskoderne, har jeg brugt to funktioner som jeg har fundet og brugt direkte.[[1]](#footnote-1)

Koden til grafen er en kode jeg har fundet. Koden virkede dog ikke til at starte med, så jeg har ændret den, så den virker.[[2]](#footnote-2)

# 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 programmet som bruger. Jeg har derfor lavet 3 krav til mit program.

### 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 ”Udførelse af iterationer” 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. Iteration 1, som er login system findes i Bilag 2.

### 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.

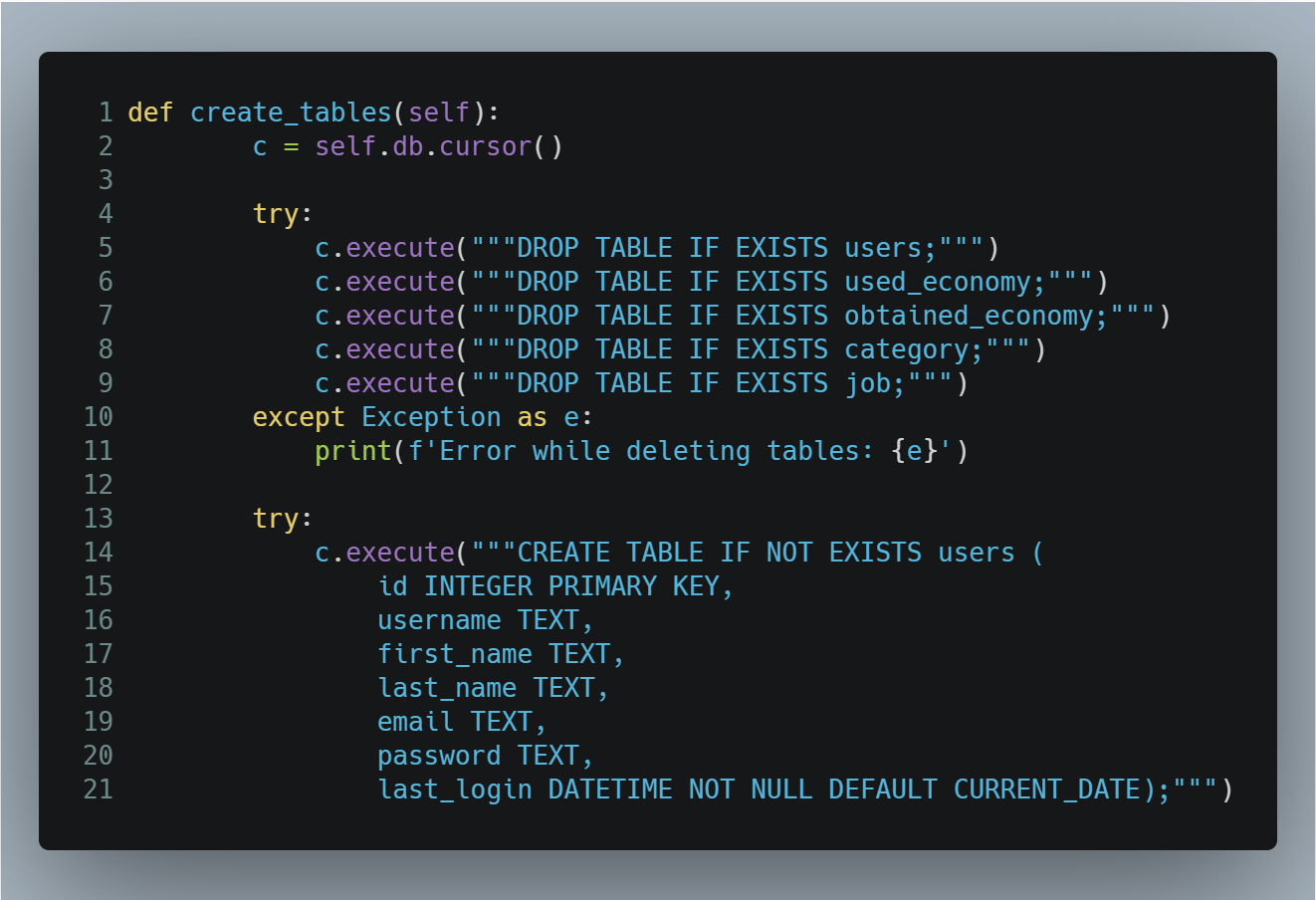
### Iteration 3 - Graf

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.

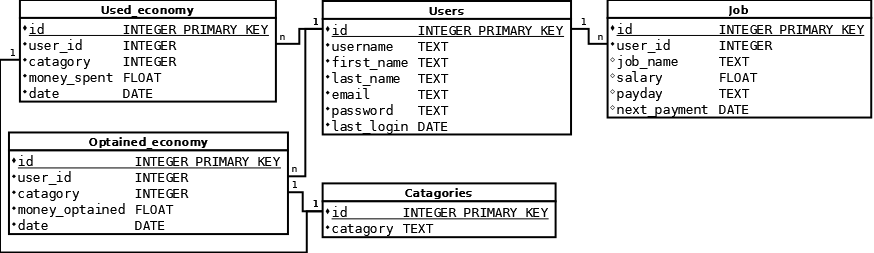
# Udførelse af programmet

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 , Eksempel på oprettelse af tabeller

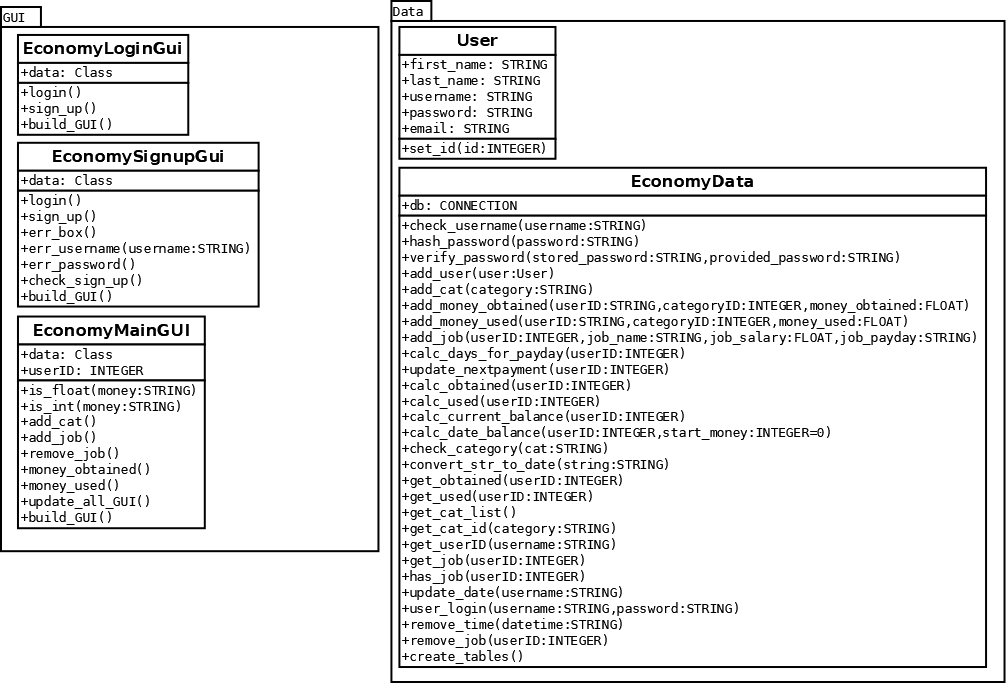


Figur , ER-diagram, figurern er vedhæftet under ”Database\_diagram.png” i ”Documents” mappen

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 har jeg lavet et ER-diagram som kan ses på Figur 1. 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 så køres funktionen ”create\_tables()”, se linje 1. Denne funktion starter med at slette alle nuværende tabeller i databasen i linje 4-9. Herefter i 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.

## Oprettelse af klasserne

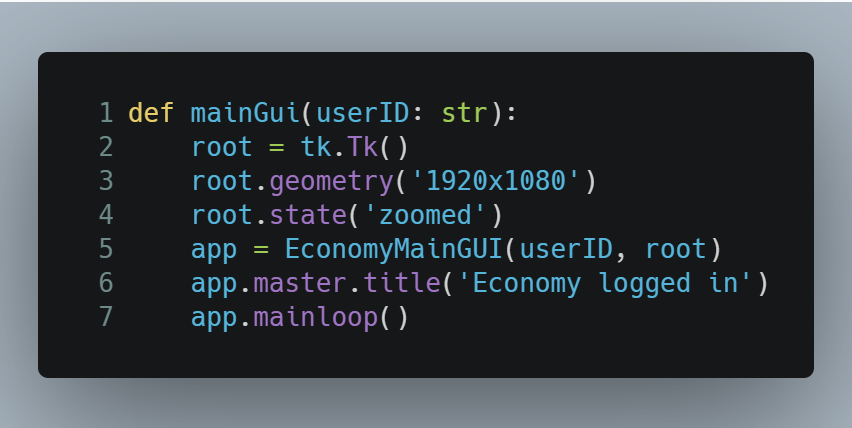


Figur 3, Klasse-diagram, figurern er vedhæftet under ”Klasse\_diagram.png” i ”Documents” mappen

Klasser i et program er en måde at opdele programmet i flere enkeltstående dele. For nemt at kunne arbejde med dataen fra databasen har jeg lavet klasser som kun arbejder med dataen. Disse klasser kan ses på Figur 3 under ”Data”. Her er klassen ”User”, som skal lave brugeren af programmet til et objekt. Den anden klasse er ”EconomyData” som arbejder sammen med databasen. 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 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.

## Udførelse af iterationer

### Iteration 2 - Hovedmenu

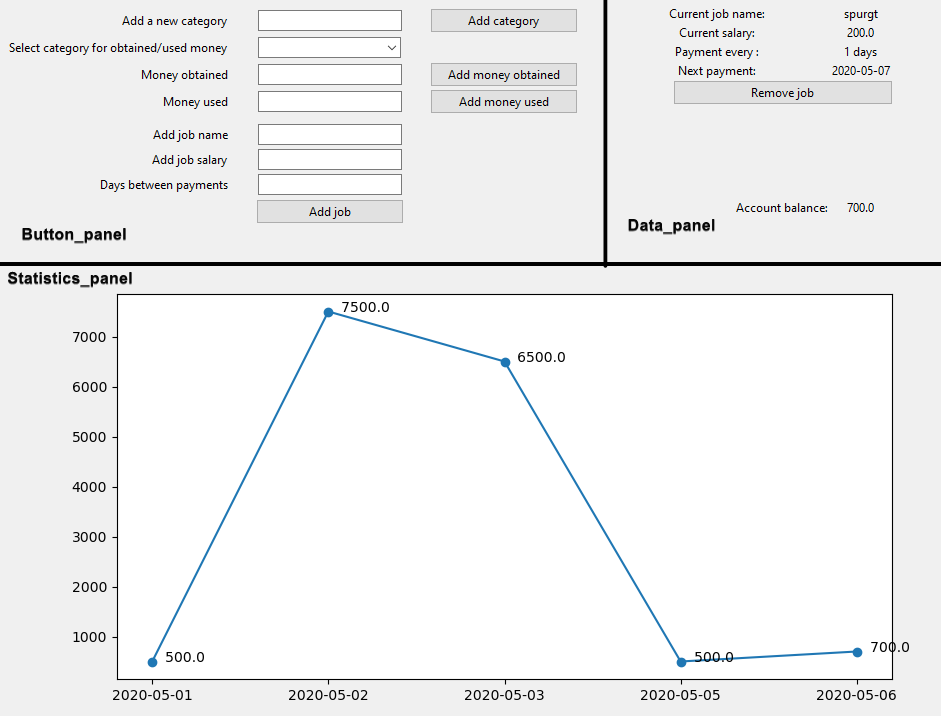


Figur 4, funktionen mainGui(). Fil: EconomyGui.py

Hovedmenuen i mit program er lavet af biblioteket Tkinter. Jeg starter med funktionen mainGui(), som kan ses på Figur 4. Denne funktion definerer hele vinduet som alt grafikken laves i. Jeg sætter størelsen af vinduet til maksimal størrelse af min skærm, og så gør jeg den maksimeret så vinduet fylder hele min skærm, dette sker i linje 2-4. Herefter defineres hvilken klasse som vinduet skal indlæse og vise på skærmen, som her er ”EconomyMainGui()”.



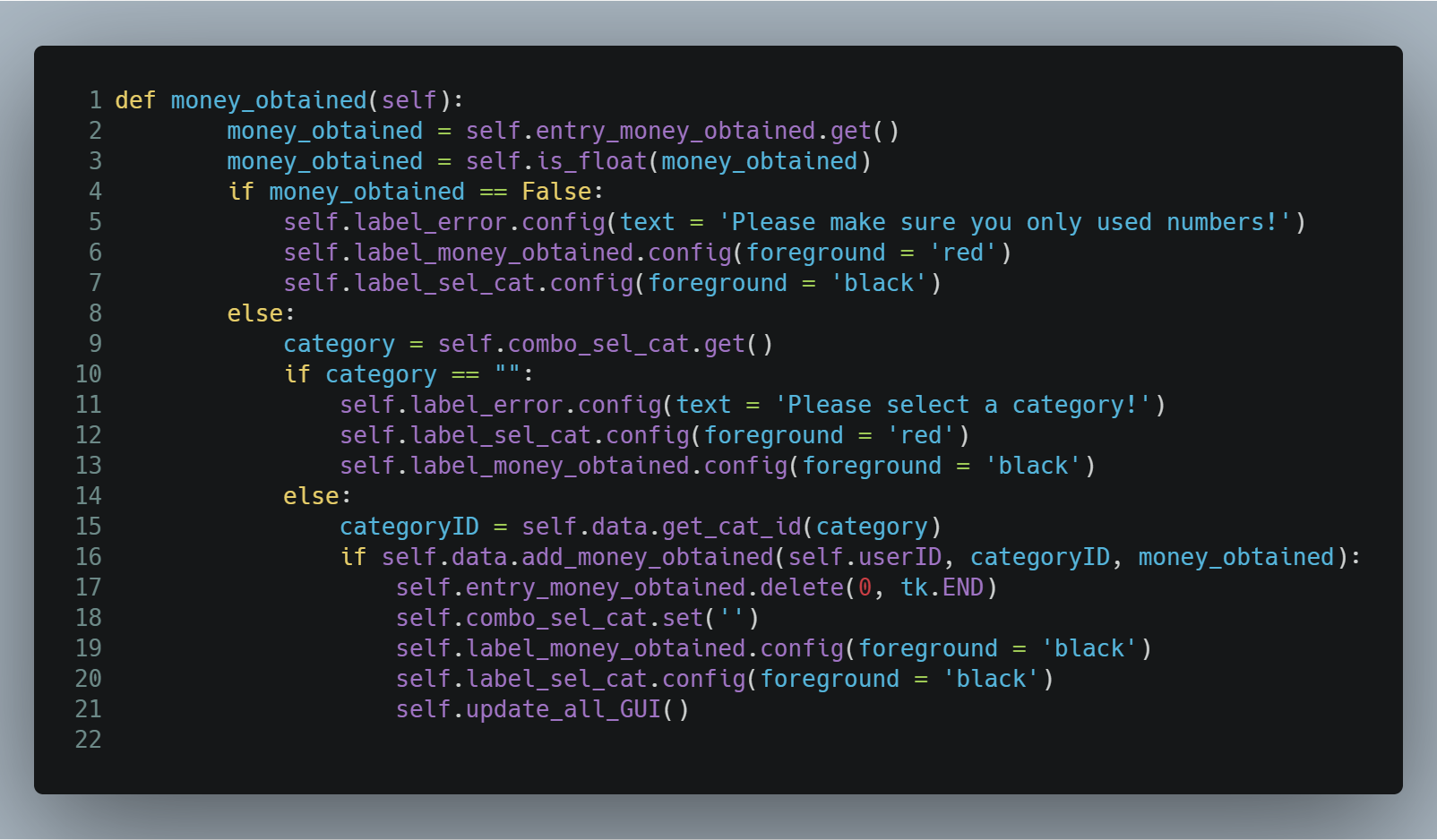
Figur 5, funktionen build\_GUI(). Fil: EconomyGui.py



Figur , billede af hovedmenuen

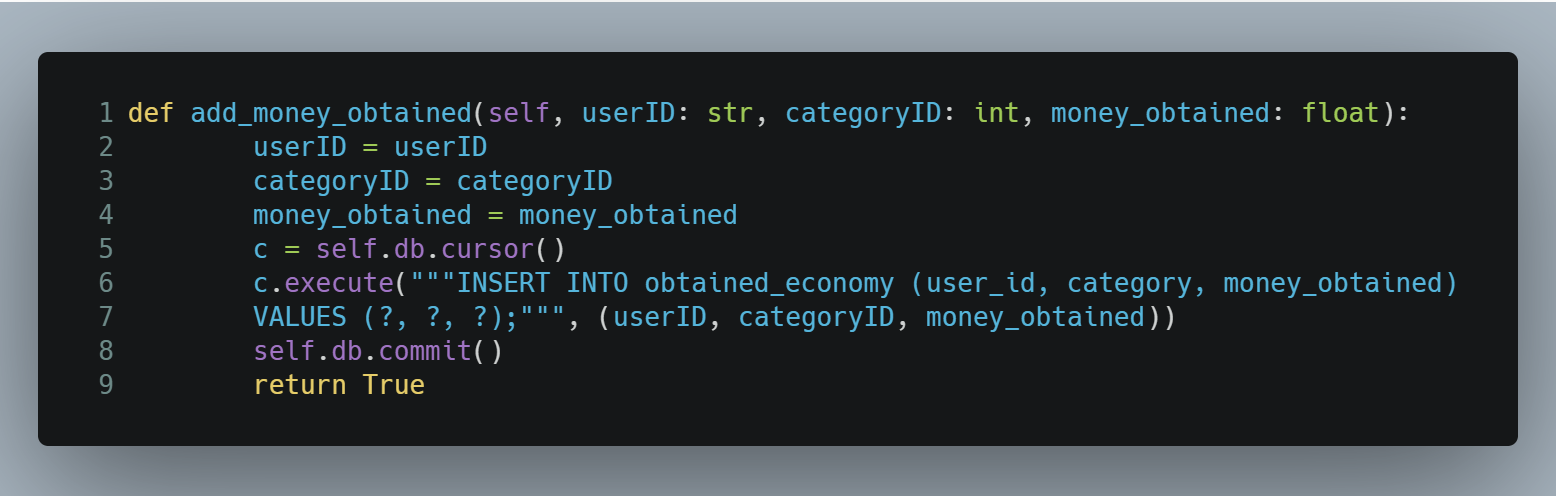
I klassen findes funktionen ”build\_GUI()”, det er denne funktion som tegner alt hvad brugeren kan se på skærmen. På Figur 5 er der et lille eksempel på funktionen, som her laver en tekst i toppen af skærmen. Hovedmenuen af mit program er delt op i tre paneler. Der er ”button\_panel”, ”data\_panel” og ”statistics\_panel”. Disse tre paneler indeholder de forskellige elementer der skal vises på skærmen. Button\_panel, er det panel der indeholder alle de inputboxe som brugeren har, samt tilhørende knapper så brugeren kan tilføje, job, kategorier, indtjente penge og brugte penge. Data\_panel er det panel der viser forskellige informationer. I programmet viser data\_panel informationerne om brugerens job, samt brugerens kontosaldo. Statistics\_panel viser grafen over brugerens kontosaldo, om bliver gennemgået i den næste iteration ”Iteration 3 - graf”. På Figur 6 ses et billede af hovedmenuen opdelt i de tre forskellige paneler.

#### Button\_panel

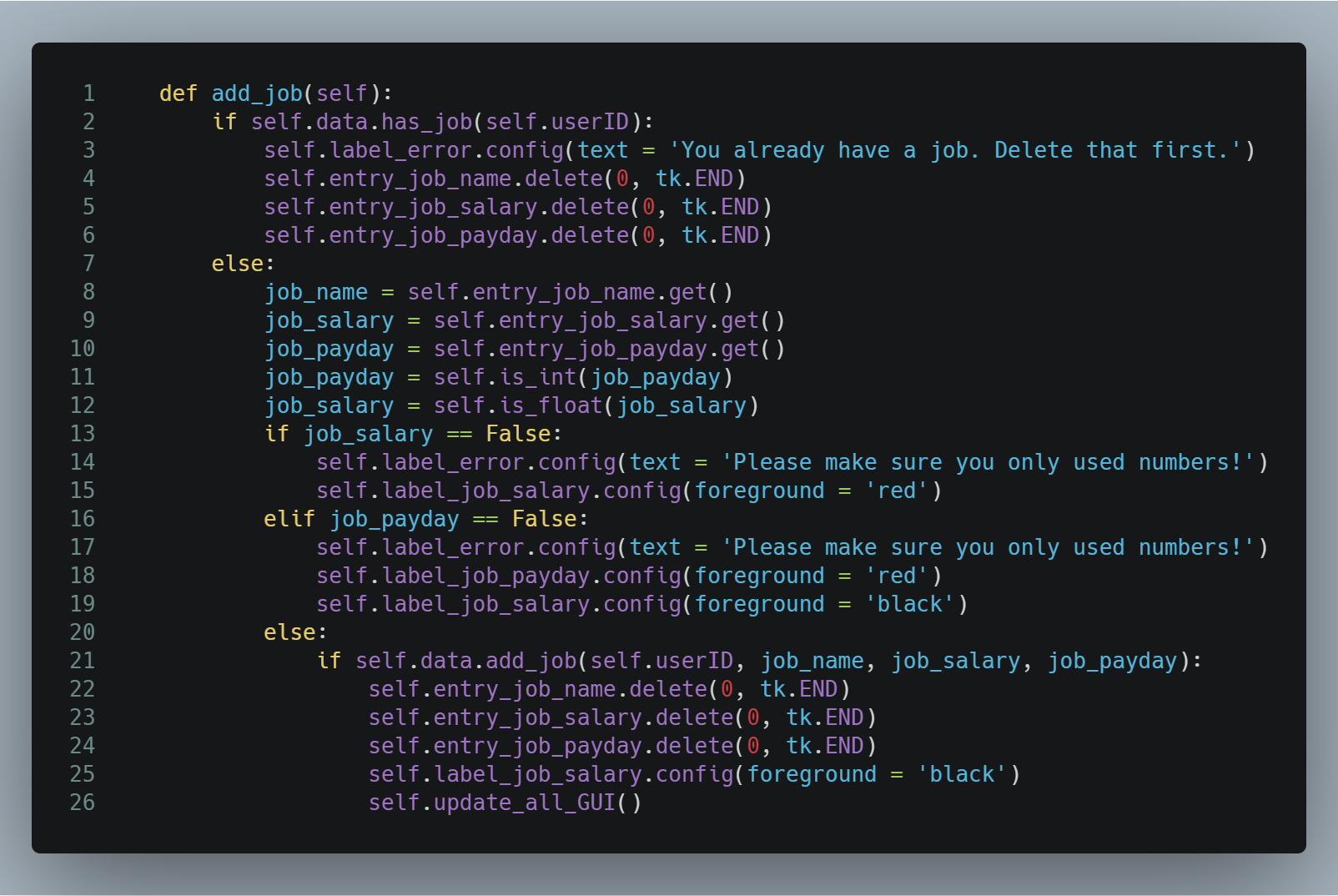


Figur 7, funktionen money\_obtained(). Fil: EconomyGui.py

Button\_panel er det panel som brugeren mest kommer til at interagerer med. Det er her brugeren skal tilføje og fjerne alle de forskellige informationer som programmet skal modtage. Brugeren kan på dette panel tilføje forskellige kategorier som brugeren vil bruge når der skal tilføjes eller fjernes penge fra kontoen. Det er et krav et brugeren vælger sådan en kategori, hvilket er grundlaget for at de funktioner er øverst på skærmen. Herefter kan brugeren tilføje hvor mange penge brugeren har fået eller brugt. Dette sker igennem funktionen ”money\_obtained()” og funktionen ”money\_used()”. Begge funktioner er 100% ens, de indsætter bare forskellige steder i databasen. På Figur 7 ses funktionen for money\_obtained(). Denne funktion starter med at modtage det beløb brugeren har skrevet ind. For at være sikker på at brugeren kun har skrevet tal ind i inputboxen så køres funktionen ”is\_float()”, i linje 3. Denne funktion tjekker bare om tallet kan laves til et komma tal. Hvis der er bogstaver, vil programmet lave en fejl på skærmen. Ellers så vil programmet modtage den kategori som brugeren har valgt, hvis ingen er valgt melder programmet en fejl. Se linje 9-13. Hvis det hele er som det skal være, så køres funktionen ”data.add\_money\_obtained()”, som sætter pengene ind i databasen og herefter genindlæser programmets GUI, for at vise ændringerne. Se linje 15-21. Funktionen ”add\_money\_obtained()” fra linje 16, kan ses på Figur 8. Funktionen modtager brugerens id, kategorien og pengene og så bliver de indsat i databasen i linje 6-7.

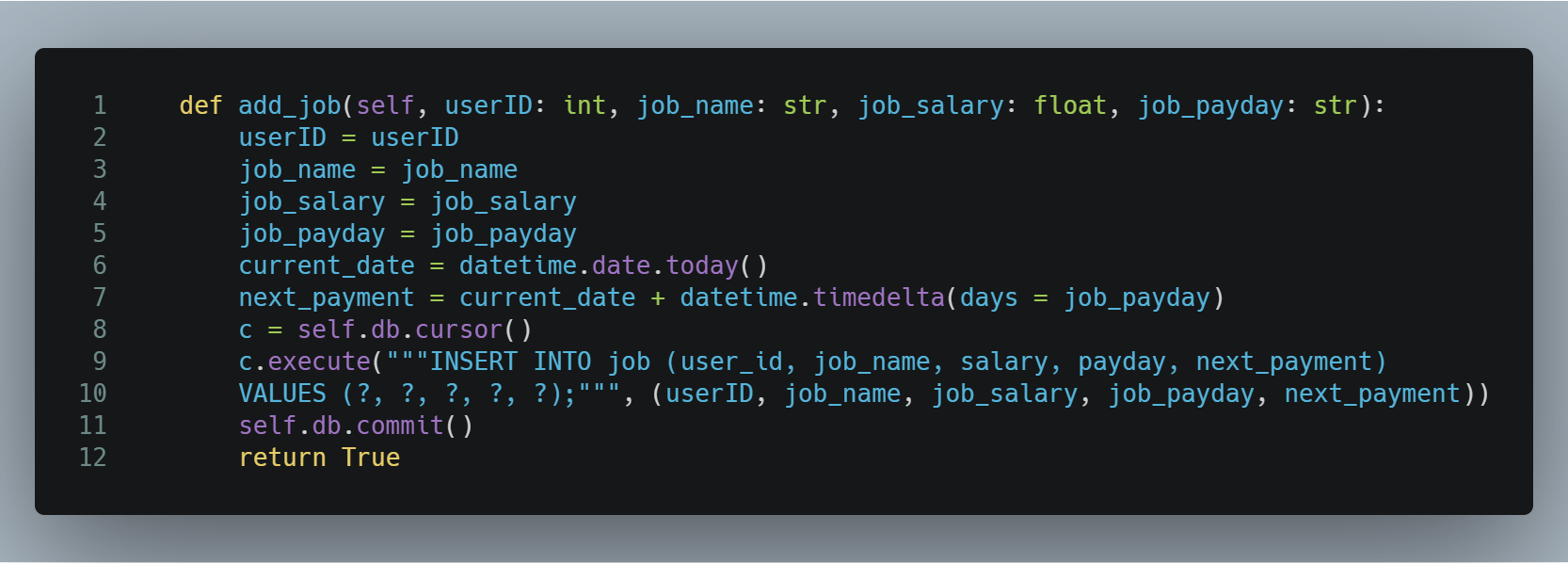


Figur 8, funktionen add\_money\_obtained(). Fil: econodata.py



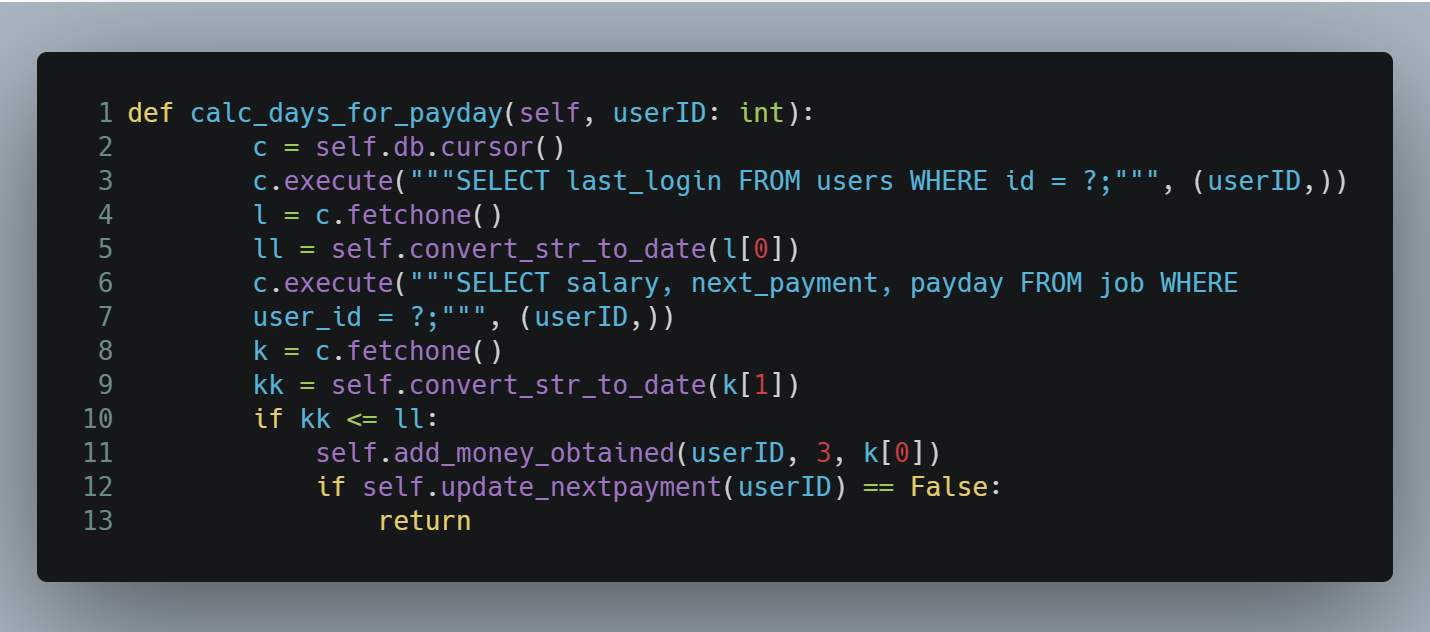
Figur 9, funktionen add\_job(). Fil: EconomyGui.py

Brugeren kan på hovedmenuen også tilføje sit job. Jobbet bliver tilføjet med funktionen ”add\_job()”, som kan ses på Figur 9. Brugeren kan her tilføje navnet på sit arbejde. Hvor meget brugeren får i løn og hvor mange dage der er mellem hver løn. Alle de her informationer modtager programmet, med mindre brugeren allerede har et job tilføjet, da programmet er lavet til at man kun kan have ét job. Se linje 2 hvor funktionen ”data.has\_job()”, kaldes for at tjekke brugeren har et job eller ej. Hvis intet job for brugeren findes, så modtager funktionen alle informationerne om jobbet på linje 8-12. Herefter kommer nogle checks for at dagene mellem lønnen og selve lønnen kun består af tal. Hvis alle de tjeks er opfyldte, så vil jobbet blive tilføjet i databasen gennem funktionen ”data.add\_job()”, som ses på linje 21. Herefter genindlæser programmets GUI, og vil nu vise informationerne omkring brugerens job. På Figur 10 ses funktionen ”add\_job()” som er i filen ”econodata.py”. Funktionen modtager alle de forskellige informationer om jobbet i linje 2-5. Herefter modtager funktionen den nuværende dato som computeren kører med. Dette sker i linje 6 med biblioteket datetime. Denne dato bruges så i linje 7, hvor programmet udregner, hvornår brugeren skal have løn næste gang. Alle disse informationer bliver så tilføjet til databasen i linje 9-10.



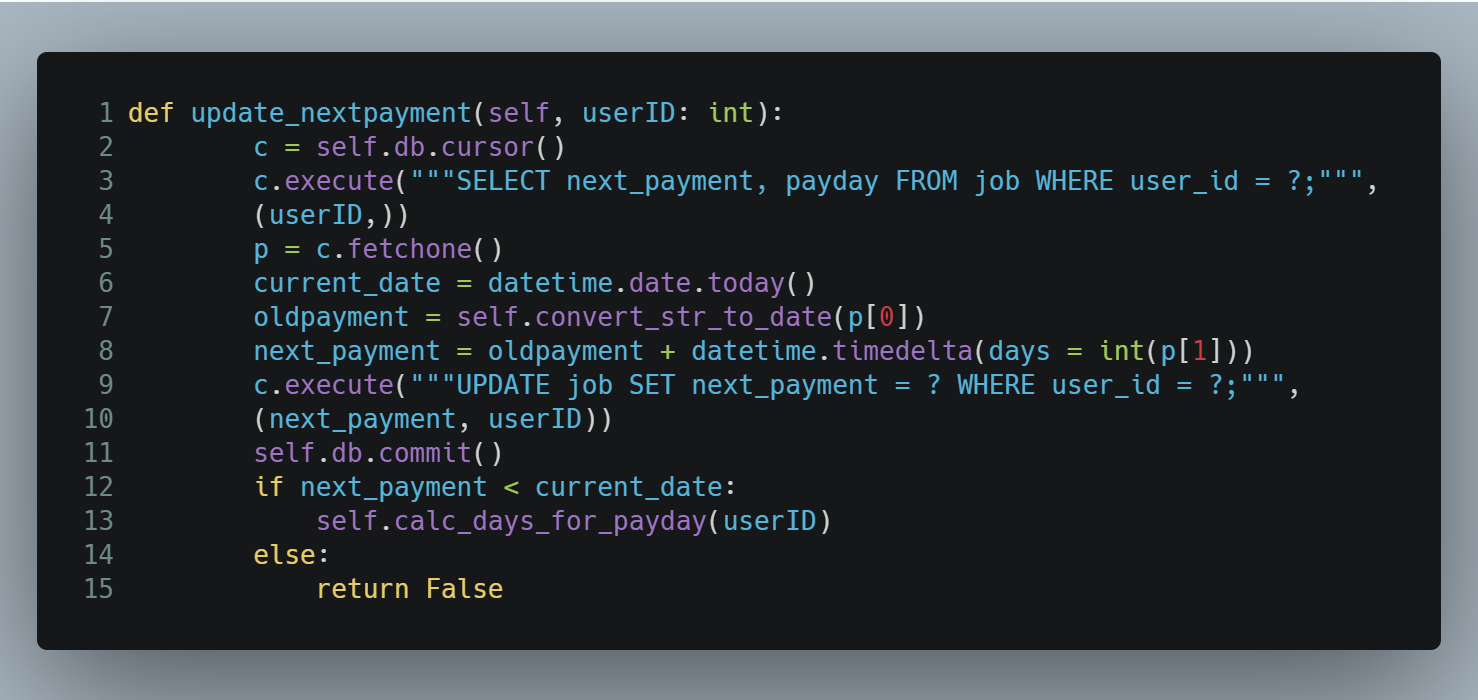
Figur 10, funktionen add\_job(). Fil: econodata.py

Programmet kan ud fra brugerens job automatisk tilføje lønnen som brugeren får af jobbet, hvis den dato brugeren logger ind med er den samme eller efter den dato brugeren skal have løn næste gang. Dette sker i funktionen ”calc\_days\_for\_payday()”, som kan ses på Figur 11. Funktionen tjekker her hvornår brugeren sidst loggede ind, se linje 3. Eftersom datoen i databasen er en STRING, så skal den laves om til et date objekt. Dette sker i funktionen ”convert\_str\_to\_date()”, i linje 5. Herefter modtager programmet informationerne om brugerens job. Her bliver datoen next\_payment også lavet om til et date objekt, programmet tjekker så om login datoen er mindre end eller lig med datoen fra next\_payment. Hvis dette er sandt, så bliver brugerens løn tilføjet til databasen, hvor programmet så opdatere, hvornår brugeren skal have løn næste gang. Dette sker i funktionen ”update\_nextpayment()”, se linje 12.



Figur 11, funktionen calc\_days\_for\_payday(). Fil: econodata.py

På Figur 12, ses funktionen update\_nextpayment(). Funktionen modtager her alle informationerne om brugerens job, og udregner så hvornår brugeren skal have løn igen. Dette sker i linje 3-8. Herefter bliver den nye dato indsat i databasen. Programmet tjekker så om den nye dato er mindre end login datoen for brugeren. Hvis den er det, så skal brugeren have løn igen, hvilket er grunden til at funktionen calc\_days\_for\_payday() køres igen i linje 13. Hvis datoen er over login datoen, så modtager brugeren ikke løn længere.



Figur 12, funktionen update\_nextpayment(). Fil: econodata.py

### Iteration 3 - graf

# 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

Molina, A. (20. september 2018). *Hashing Passwords in Python*. Hentet 06. maj 2020 fra vitoshacademy: https://www.vitoshacademy.com/hashing-passwords-in-python/

pythonprogrammring. (u.d.). *How to embed a Matplotlib graph to your Tkinter GUI*. Hentet fra pythonprogrammring: https://pythonprogramming.net/how-to-embed-matplotlib-graph-tkinter-gui/

# 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.

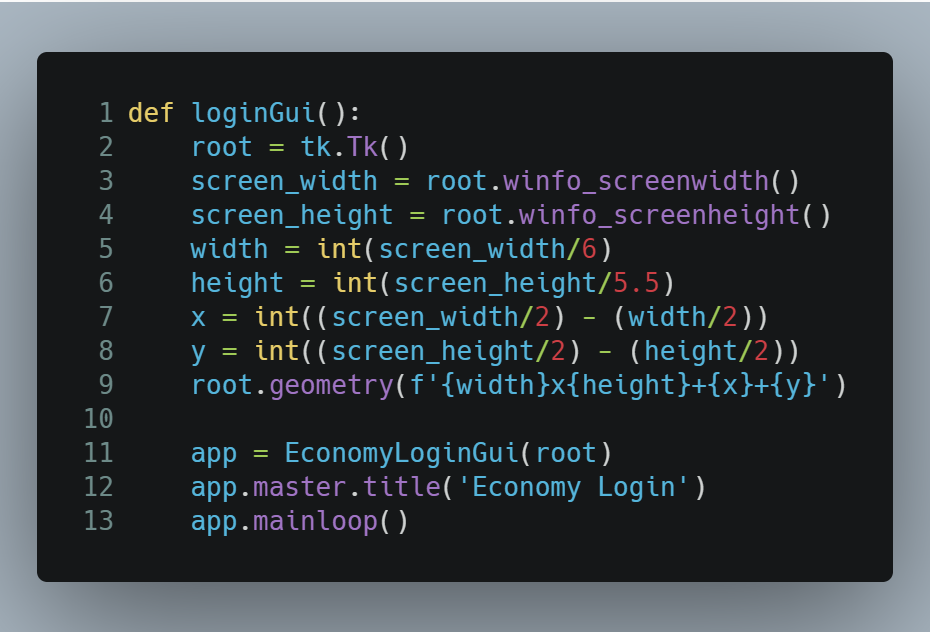
## Bilag 2

### 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.

## Bilag 3

### Iteration 1 - Login system



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

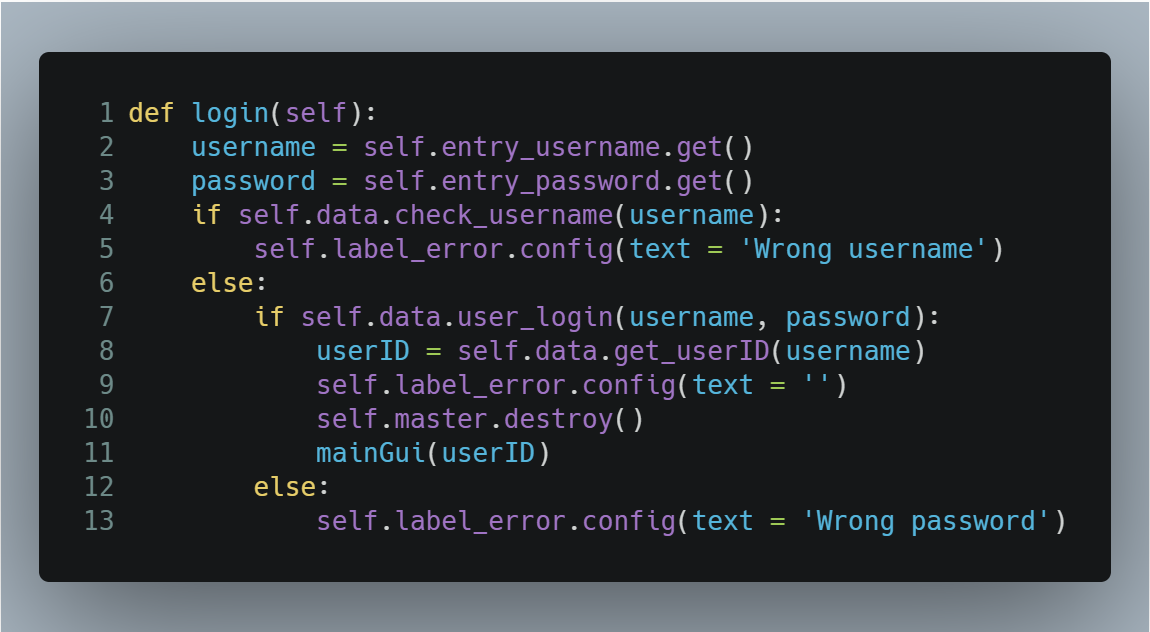
Udførelsen af den første iteration, hvilket var ”Brugerhistorie 1 - Login system” har krævet at jeg havde lavet 3 ud af de 5 klasser som var på Figur 3. Klasserne som loginsystemet krævede, er EconomyData, EconomyLoginGui og User.

Jeg 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 første, hver gang programmet åbner. Funktionen starter med at lave en root, 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. I linje 3-9 definerer jeg så størrelsen og placeringen af selve loginvinduet. Dette bliver udregnet ud fra brugerens nuværende skærmstørrelse.



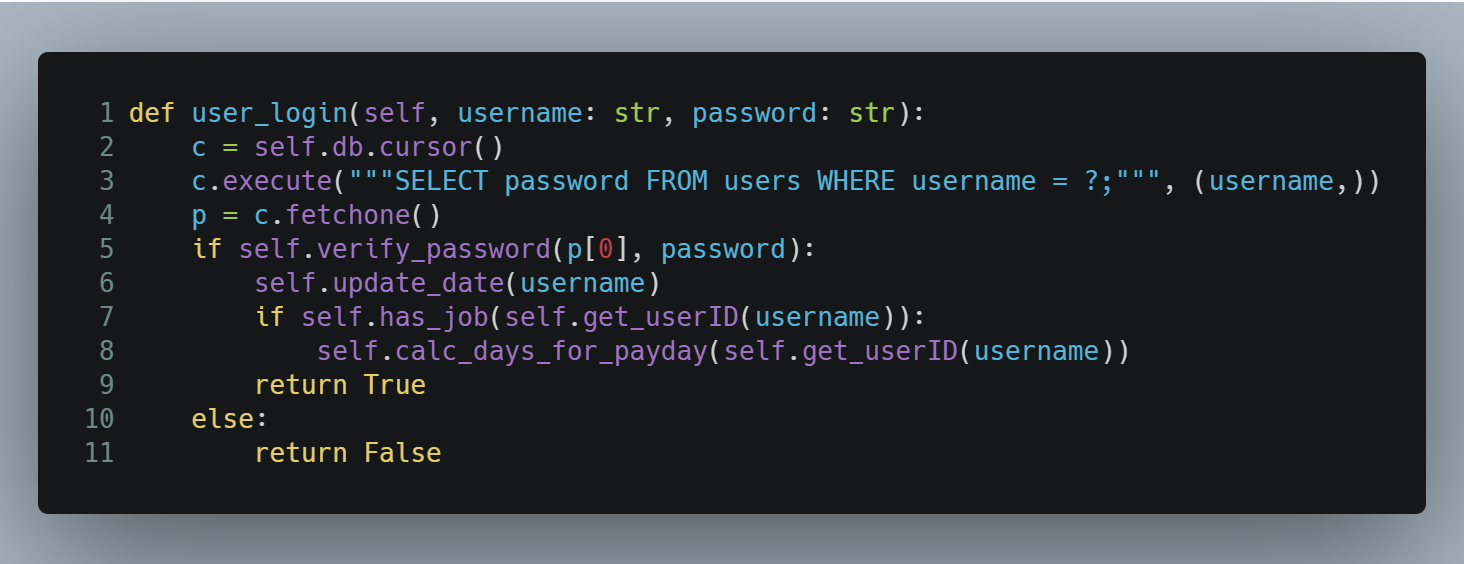
Figur , eksempel af funktionen build\_GUI(). Fil: EconomyGui.py

Til sidst definerer jeg, hvilken klasse som vinduet skal vise, i dette tilfælde er det EconomyLoginGui, se 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 funktionen.



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

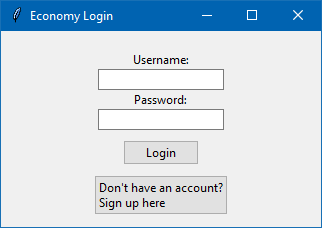
På Figur 6 ses funktionen ”login()”. Funktionen starter med at modtage brugerens brugernavn og adgangskode i linje 2 og 3. Herefter tjekkes om brugernavnet findes i databasen, med funktionen ”data.check\_username()” som er en funktion der er i min data klasse. Eksistere brugernavnet, så tjekker programmet om den indtastede kode er det samme, som den kode som brugeren brugte når brugeren opretter sin konto. Dette sker i funktionen ”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 i programmet. Vinduet bliver fjernet og så åbnes hovedvinduet 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.



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

På Figur 7 ses funktionen ”user\_login()”. Det er denne funktion som bliver kaldt når brugeren logger ind.

Funktionen modtager brugerens brugernavn og brugeren kode, se linje 1. Herefter tjekker programmet om koden er korrekt. Dette sker i line 2 til 5. Hvis koden er korrekt, så bliver brugerens login dato opdateret til den nuværende dato af funktionen ”update\_date()” i linje 6, herefter tjekker programmet om brugeren har tilføjet et job. Hvis brugeren har et job tilføjet, så udregner programmet datoen for hvornår brugeren skal have løn igen i funktionen ”calc\_days\_for\_payday()” i linje 7 til 8. Loginsystemet bortset fra sign up er nu lavet se Figur 8.

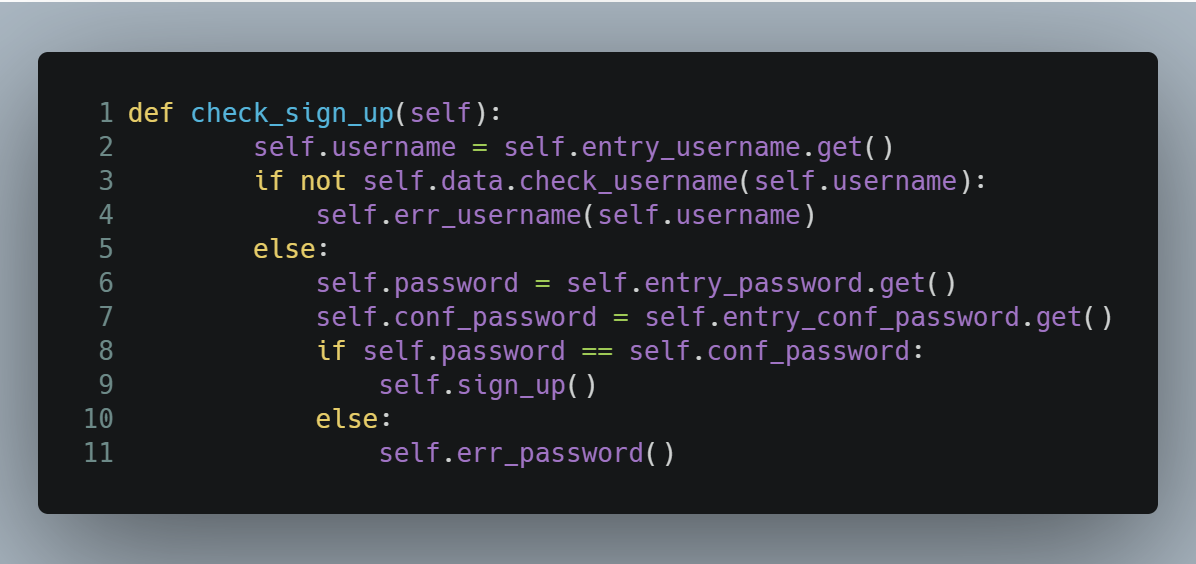


Figur , billede af loginskærmen.

Oprettelsen af en ny konto hører også under login systemet, så det skal også under den første iteration. Her køres funktionen ”signUpGui()” når brugeren trykker på knappen ”Don’t have an account? Sign up here” på loginskærmen se Figur 8. Den eneste forskel mellem funktionen signUpGui() og loginGui() er beregnelsen af vinduestørrelsen og placeringen vinduet , se Figur 4 og Figur 9.

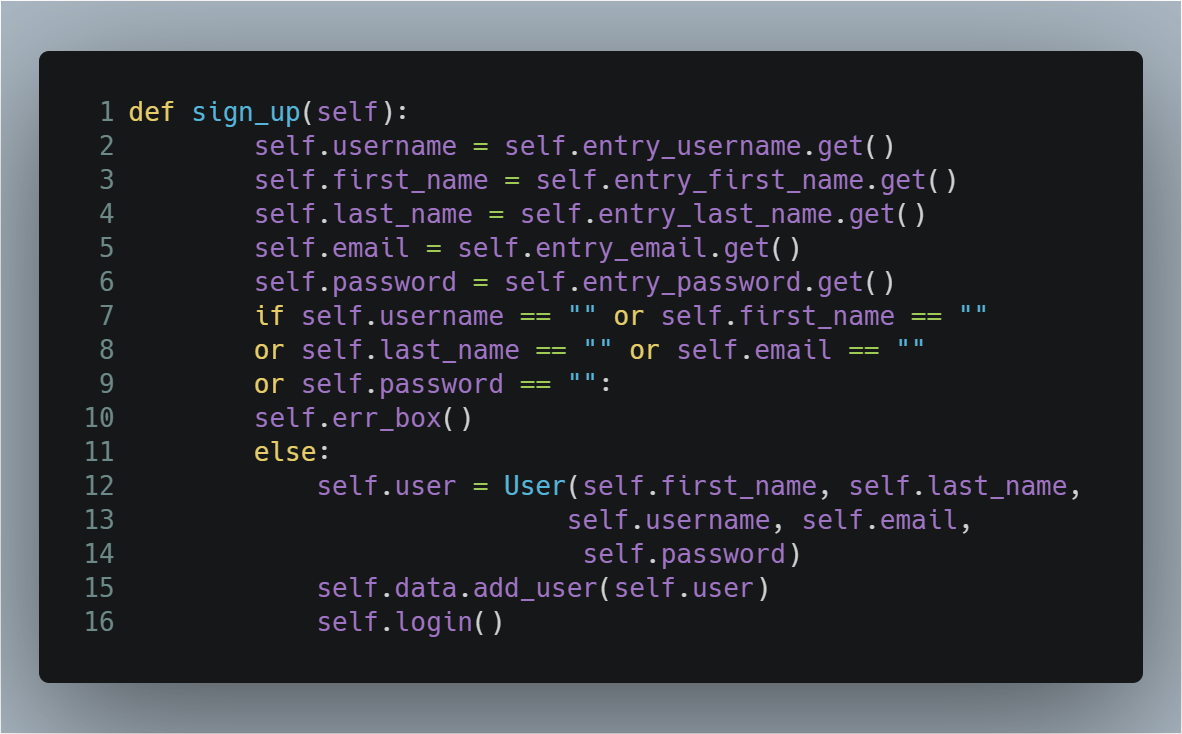


Figur , funktinen signUpGui(). Fil: EconomyGui.py



Figur 19, funktionen check\_sign\_up(). Fil: EconomyGui.py

For at oprette en konto, så køres funktionen ”check\_sign\_up()” i klassen ”EconomySignupGui”, se Figur 10. Funktionen modtager brugerens brugernavn. Herefter tjekkes om brugernavnet allerede eksisterer i databasen i linje 3. Funktionen er præcis den samme som i funktionen user\_login() på Figur 7.Er brugernavnet ledigt, så modtager funktionen det kodeord og bekræftelses kodeord som brugeren har indskrevet. Ellers kommer der en fejlbesked. Se linje 3-5. Funktionen tjekker koderne er ens, hvis de er ens, så køres funktionen ”sign\_up()”, se Figur 10 linje 5-9. Hvis koderne ikke er ens kommer der en fejlbesked, som ses på linje 11.



Figur 20, funktionen sign\_up(). Fil: EconomyGui.py

Funktionen sign\_up() ses på figur 11.

1. Funktionerne: hash\_password() og verify\_password i filen: econodata.py: (Molina, 2018) [↑](#footnote-ref-1)
2. Linje 327 - 336: (pythonprogrammring, u.d.) [↑](#footnote-ref-2)