STUDIERETNINGSCASE

Matematik og programmering

NEXT Sukkertoppen Gymnasium
SR-Case

Indhold

Introduktion:

Mange unge som lige er blevet 18 år, selv mange voksne, kan ikke finde ud af, hvor vigtigt det er, at man ved, hvad man laver, når man tager et lån eller begynder at investere, for det kan hurtigt gå ud af hånd. Selv er jeg lige blevet 18 år, og der er alverdens ting, som er åbnet op for mig, men som jeg endnu ikke helt forstår. Hvad kan man gøre for at give folk, som er ukendte til investering og lån en basisviden, så de ikke kommer til at dumme sig lige så nemt?

Jeg vil i denne case gerne undersøge, om det er muligt ud fra matematiske modeller og teori, samt programmerings færdigheder, at demonstrere vigtigheden og basisviden om et lån og eller investering. Så evt. unge ikke tager et lån og betaler for lidt om måneden til, at de overhovedet betaler deres lån af, eller at det bare bliver alt for dyrt i længden.

Teori:

Matematik:

Rekursionsligning:

Det kan være, at du har hørt termen "rekusionsligning" eller "rekursive funktioner" før, men ikke helt ved, hvad det præcist er. I de næste par afsnit vil jeg forklare, hvad en rekursiv funktion er, og give nogle eksempler på, hvordan man kan implementere det i vores dagligdag.

En rekursiv funktion er kendetegnet ved, at den kalder sig selv. Det skal forstås på den måde, at hvis vi gerne vil finde en værdi, for eksempel x_n, så skal vi bruge x_n for at finde den, eller rettere sagt, så skal vi bruge en tidligere værdi af x_n, som man kunne kalde x_{n-1}. Hvis man kan finde x_n, kun ved brug af den forrige x_n-værdi, altså x_{n-1}, så kan man sige, at den rekursive ligning er i 1. orden. Hvis man derimod skal bruge både de tidligere og den forrige værdi for at finde x_n, altså x_{n-1} og x_{n-2}, så ville den rekursive ligning være i 2. orden. Det samme med tre, fire, fem osv. Når man kigger på at løse vores rekursionsligning, så kalder vi vores orden, for *k*.

Når vi snakker om at løse en rekursiv ligning, så skal vi lige huske, at der er to forskellige slags rekursionsligninger, nemlig den homogen lineær rekursionsligning og den ikke homogen lineær rekursionsligning. Jeg vil her i denne opgave gå i dybden med den homogen lineær rekursionsligning, da det er noget, som jeg gør brug af i mit program.

Vi kan tage udgangspunkt i en homogen lineær rekursionsligning af første orden: $x_n = r \cdot x_{n-1}$, med $x_0 = 1$ som begyndelsesbetingelse[1], for at lave vores bevis. Her starter man med at lave et induktionsbevis, hvor vi antager at $x_n = r^n$, er en løsning til vores homogene lineære rekursionsligning. Vi kan derefter teste, om den faktisk er sand, ved at tjekke om vores begyndelsesbetingelse passer ind i løsningen:

$$x_0 = r^0$$

⇕ *The equation is solved for x_0 by WordMat*

$$x_0 = 1$$

[1] https://mathtxa.systime.dk/index.php?id=505&L=0&q=p505&redirected=1 (SÆTNING 5.9)

Da vi ved at vores begyndelsesbetingelse er $x_0 = 1$, kan vi konkludere, at løsningen er sand for det første led. Vi antager nu at $x_{n-1} = r^{n-1}$, ud fra den løsning vi lige kom frem til. Vi kan nu tage og erstatte vores x_{n-1} med r^{n-1} i vores rekursionsligning, og forkorte:

$$x_n = r \cdot r^{n-1} = r^1 \cdot r^{n-1} = r^{1+n-1} = r^n$$

Vi kan se, at når den forrige værdi kan løses med denne løsning, så kan den næste også, dermed kan vi konkludere, at denne løsning virker for vores rekursive ligning[2].

Man kan så se, at den generelle løsning for vores homogene lineære rekursionsligning af første orden kan skrives som[3]:

$$x_n = b \cdot r^n$$

Hvor $b \in \mathbb{R}$

Nu skal vi have vores karakterligning[4], som vi kan finde, da vi antager at $x_n = r^n$, også er en løsning til den homogene lineære rekursionsligning af k'te orden:

$$x_n = a_1 x_{n-1} + a_2 x_{n-2} + \cdots + a_k x_{n-k}$$

Ud fra dette har vi altså to ligninger, som giver x_n. Vi kan derfor fjerne x_n og sætte de to ligninger lig med hinanden, så vi får:

$$r^n = a_1 x_{n-1} + a_2 x_{n-2} + \cdots + a_k x_{n-k}$$

Vi ved også fra før at $x_{n-1} = r^{n-1}$. Derfor kan sætte r^{n-1} ind på alle x_{n-1}'s pladser:

$$r^n = a_1 r^{n-1} + a_2 r^{n-2} + \cdots + a_k r^{n-k}$$

Vi kan så forkorte denne ligning, ved at dividere med r^{n-k} i alle led:

$$\frac{r^n}{r^{n-k}} = \frac{a_1 r^{n-1}}{r^{n-k}} + \frac{a_2 r^{n-2}}{r^{n-k}} + \cdots + \frac{a_k r^{n-k}}{r^{n-k}}$$

De går ud med hinanden, så vi ender med:

$$r^k = a_1 r^{k-1} + a_2 r^{k-2} + \cdots + a_k$$

Her tager vi så og flytter hele over på venstre side og ændre selvfølgelig alle fortegnene på det vi flytter:

$$r^k - a_1 r^{k-1} - a_2 r^{k-2} - \cdots - a_k = 0$$

Dette er, hvad vi kalder for vores karakterligning, som man bruger til, at finde ens rødder *r*, som vi skal bruge i vores generelle løsning. Der er også noget, som hedder det karakteristiske polynomium, som er, når man kun kigger på venstresiden af karakterligningen.

[2] https://mathtxa.systime.dk/index.php?id=505&L=0&q=p505&redirected=1 (BEVIS)
[3] https://mathtxa.systime.dk/index.php?id=505&L=0&q=p505&redirected=1 (SÆTNING 5.10)
[4] https://mathtxa.systime.dk/index.php?id=505&L=0&q=p505&redirected=1 (Karakterligning og det karakteristiske polynomium)

Ud fra hvad vi har set på, kan vi opstille vores generelle løsning for en homogen lineær rekursionsligning af k'te orden, hvis der er *k forskellige reelle* rødder i det karakteristiske polynomium[5]:

$$x_n = b_1 r_1^n + b_2 r_2^n + \cdots + b_k r_k^n$$

Hvor $b_i \in \mathbb{R}$, $b_k \neq 0$ og $n = 1, 2, 3, \ldots$

Hermed har vi bevist, hvordan man udregner den generelle løsning for en homogen lineær rekursionsligning.

Rentes rente rekursionsligning:

Når man kigger på en investering, kræver det, at vi har en begyndelsesværdi og en renteprocent, som lægges til. Hvis vi antager, at man sætter sin begyndelsesværdi ind og ikke sætter mere ind, end den profit man får, og siger, at der kommer renter på hver år, så kan man opstille denne rekursive ligning:

$$x_n = x_{n-1} \cdot (1 + y)$$

Hvor *y* svarer til vores rente.

Vi kan så aflæse vores *a* og *k* værdi fra ligningen:

$$y = rente$$
$$a_1 = (1 + y)$$
$$k = 1$$

Herefter kan vi tage og sætte vores værdier ind i vores karakterligning, og udregne vores rødder, eller i vores tilfælde rod, da vores ligning er i første orden.

$$r^1 - (1 + y) \cdot r^{1-1} = 0 \xrightarrow{\text{solve for r}} [[r = 1 + y]]$$

Vi kan nu sætte det ind i vores generelle løsning:

$$x_n = b \cdot (1 + y)^n$$

Vi har hermed fundet en ligning til at finde, hvor mange penge man har tjent efter n antal år, hvor man bare skal kende ens begyndelsesværdi *b* og ens rente procent *y*.

Eksponentiel funktion:

Når vi taler om renter, er eksponentielle funktioner rigtig relevante at tale om. Grundlagt er eksponentielle funktioner nemlig funktioner, der enten kan forøges eller formindskes over tid. Hvis man kigger på vores fundne generelle løsning, vil man også kunne se, at den minder rigtig

[5] https://mathtxa.systime.dk/index.php?id=505&L=0&q=p505&redirected=1 (SÆTNING 5.12)

meget om den eksponentielle funktionsforskrift: $f(x) = b \cdot a^x$.[6] Ud fra dette kan vi altså konkludere, at det er en eksponentiel funktion, vi har med at gøre. Det med at vi lægger vores procent til hver år, gør også, at vores graf gerne skulle stige mere og mere, jo længere til der går, som også passer meget godt med den eksponentielle funktion.

Matematik i min kode:

I min kode har jeg gjort brug af mange af de forskellige matematiske formler og funktioner, som jeg lige har gennemgået. Jeg har taget mig den lejlighed, til både at lave et program til at udregne lån og et til at udregne investeringer. Man kan vælge mellem de to ved at sætte den, som man ikke vil have som kommentar:

```
//NegativRente();
PositivRente();
```

Lånet går ud på, at den tager og sætter, hvad man mangler at betale af ens lån til at være lig med, hvad man skulle betale sidste år, ganget med renten, som ryger oven i og til sidst trukket fra med, hvad man betaler det år, for at få betalt ens lån af:

```
debt = debt*renteProcent-(12*monthlyPayment);
```

Det er den rekursive ligning til et lån, som gør at mit program, tager alle værdierne, indtil du har betalt dit lån helt af, og danner en graf over det[7], og siger efter, hvor lang tid du har betalt dit lån af.

```
Dit lån er blevet betalt af efter 23 år og 8.852116468475264 måneder
```

(Alle beregninger gør brug af variabler, som man selv kan ændre på oppe i toppen af programmet)

Ud over lånet, har jeg også lavet kode til en investering. Tidligere i rapporten tog jeg udgangspunkt i en investering til at beregne den generelle ligning, og jeg har selvfølgelig gjort brug af den generelle ligning til at udregne, hvor mange penge, man har tjent efter n antal år:

```
//xn = startAmount * rente^n
profit = startAmount * pow(renteProcent,year);
```

Det var rigtig fint og det hele, men jeg endte med også at lave den rekursive ligning, som vi startede med at finde, da det gjorde, at det også ville være muligt at lave en flot graf over det[8].

```
profit = profit * renteProcent;
```

Her fungerer det ligesom med lånet, at man tager det antal penge, man har i et år, og sætter lig med antal penge, man havde sidste år, og gange med den procent, som er sat i toppen.

[6] https://matbhtx.systime.dk/index.php?id=1331#c13414
[7] Bilag 2 (Eksempel på grafer)
[8] Bilag 2 (Eksempel på grafer)

Teori + min kode:

Programmering:

Variabler:

En computer består af mange ting. Det kan være alt fra et motherboard, som kører det hele, og på en måde er computeren hjerne, til en fan, som bruges til at køle ens computer ned, eller til at skubbe den varme, som dannes inden i computeren, ud. Ikke nok med det, så har computeren også en (eller flere) RAM (Random Access Memory), som man kan kalde for computerens korttidshukommelse. I ens RAM gemmes der data midlertidig, til man slutter programmet eller computeren, hvorefter alt denne data vil enten blive slettet eller overført til en harddisk, som man kan kalde for computerens hukommelse, hvor den kan gemme ting, selv efter at computeren er slukket.

Når man programmerer, er det for det meste i ens RAM, man gør det. En variabel er noget, som jeg vil påstå, at næsten alle computerprogrammer gør brug af. Dette er, fordi det er så smart og essentielt og kan bruge i adskillelige tilfælde. En variabel er en værdi, som kan ændres igennem hele programmet. Variablerne bliver gemt i RAM, indtil man slutter programmet, eller man ikke skal bruge variablen mere, hvorefter den bliver frigivet fra RAM. For at man kan bruge en variabel i et program, kræver det, at man starter med at lave ens variabel. Her er et eksempel på, hvordan man kan lave en variabel i JavaScript:

var x = 25;

"var" står for, at det er en variabel, vi gerne vil oprette, "x" er et navn, som vi tilgiver denne variabel, og "25" er så den værdi, som vi sætter denne variabel "x" til at være. Med denne variabel kan vi altså nu tage i resten af vores program og benytte og ændre "x" lige som vi har lyst til. Man kan også undlade at give ens variabel en værdi og bare skrive:

var x;

Her ville man så give ens variabel en værdi senere hen.

Men "var" er ikke den eneste type variabel, som man kan lave. Der findes forskellige variabeltyper angående, hvad din variabel skal være. Man laver normalt en variabeltype, som hedder "int", hvis man har med heltal at gøre. Hvis man derimod har med decimale tal at gøre, bruger man variabeltypen "float". Der findes også typer som "string", som har med tekst at gøre og en "boolean", som man kan sætte til enten at være sand eller falsk. Man skriver altså en variabel sådan her:

(variabeltype) (navn) = (værdi, tekst, sand/falsk);

Syntaksen kan selvfølgelig være forskellig fra programmeringssprog til programmeringssprog. I Python skriver man for eksempel ikke variabeltypen eller bruger semikolon.

I mit program gør jeg brug af et par variabler:

```
1
2  const startAmount = 10000;
3  const renteProcent = 1.03;
4  const monthlyPayment = 100;
5  const year = 50;
6  let x = 0;
7  let uendelig = 100000;
8  let debt;
9  let months;
10 let profit;
11
```

Før i tiden var der kun "var" til at danne en variabel, men efter ECMAScript 2015, blev der lavet to nye, som hed "const" og "let". "const" er en variabel, som ikke kan ændres på, eller lavet om til noget andet. Så når jeg, som man kan se, har sat "year" til at være 50, så er "year" 50 resten af min kode, jeg kan altså ikke lige pludselig lægge år til eller trække fra, som man kan med de andre variabler. "let" minder mere om "var", den eneste forskel er, at "let" er mere kontroleret i, hvor den fungerer. Et godt eksempel er det her:

```
var i = 5;
for (var i = 0; i < 10; i++) {
  // some statements
}
// Here i is 10
```

```
let i = 5;
for (let i = 0; i < 10; i++) {
  // some statements
}
// Here i is 5
```

Til venstre kan vi se at vores "var", som vi laver inde i vores forloop, overskriver den forrige "var", vi lavede, og virker også efter forloopet. Til højre kan vi derimod se, at vores "let", som vi danner inde i forloopet, kun virker inde i det forloop, og efterfølgende er det vores første "let", som fungere.

I mit tilfælde har jeg altså lavet fire "const", som er værdier, som man kan ændre på i starten og dermed se, hvordan det påvirker resultatet og de værdier, som man skriver ind forbliver, som de er, og bliver ikke lavet om på. Jeg har så også fem "let" variabler (nogle med, og nogle uden værdier), som jeg, igennem mit program, ændre for, at det hele virker, så dem skal man ikke ændre på.

Funktioner:

Når man starter en ny sketch i JavaScript, starter man normalt med at have to funktioner: setup og draw. Ens "function setup" køres én gang, lige når man starter programmet, hvor ens "function draw" kører i et loop en gang hver frame. Ud over ens start funktioner, så kan man også danne sine egne funktioner, som man så senere kan kalde på. Dette er noget, som stort set alle såkaldte 'rigtige' programmører gør brug af. En funktion kan laves i JavaScript sådan her:

```
function test(){

}
```

Når man så vil kalde denne funktion i f.eks. ens setup, vil man gøre det sådan her:

```
function setup() {
  test();
}
```

Men funktioner er ikke så simple. Man kan også lave funktioner, som kan have et eller flere input, som kaldes parametre. Man kan så inde i ens funktion danne et output via disse parametre. Et eksempel på det kunne være:

```
function setup() {
  test(5,7);
}

function test(x,y){
  console.log(x + y);
}
```

Vi har altså en funktion her, som skal bruge en x- og en y-værdi, som den så lægger sammen. Vi angiver disse to værdier, når vi kalder på funktionen i fx. vores setup. I dette tilfælde ville den så printe 12 i vores consol.

I min kode har jeg også gjort brug af nogle funktioner. Til at starte med kan man vælge, om man vil køre min negative eller positive funktion, som indikerer, om man kigger på et lån eller en investering. Pga. mangel på tid skifter man bare mellem de to, ved at gøre den, man ikke vil have til en kommentar:

```
function setup() {
  createCanvas(windowWidth,windowHeight);
  background(220);

  //NegativRente();
  PositivRente();

}
```

Jeg har så en funktion, som hedder "NegativRente", som så kører beregningerne som om, at det var et lån, vi kiggede på, som man skulle betale tilbage, hvis man vælger at se det. Og så har jeg også en funktion, som hedder "PositivRente", som danner to knapper, som går til hver sin egen funktion, som skal indikere, om man ser investeringen med den rekursive ligning, så man også kan se en graf over det, eller om man bare ser den generelle løsning, så man bare ser antal penge på kontoen efter, hvad ens "year" er sat til:

```
function PositivRente (){

  button1 = createButton('Rekursiv løsning');
  text('(plus graf)', windowWidth / 3 - 15, windowHeight / 2 + 35);
  button1.position(windowWidth / 3 - 45, windowHeight / 2);
  button1.mousePressed(rekursive);

  button2 = createButton('Generelle løsning');
  button2.position(windowWidth / 3 * 2 - 50,windowHeight / 2);
  button2.mousePressed(generelle);

}
```

Betinget udførelse:

Det computerprogram som du laver, opererer ud fra et sæt given programmerede instrukser, som du har givet. Det kan være simple instrukser, som at sætte en variabel til at være lig med 25, som vi så på før. Men man kan også lave nogle lidt mere komplicerede instrukser, som åbner op for alverdens ting inden for programmering. Hvis nu du har en variabel, som du kalder "kortSaldo" og sætter dens værdi til 100, for at indikere, at der er 100 tilbage på dit kort. Hvis vi så gerne vil lave noget, som siger at vi skal huske at tanke op, når vores kort er ved at løbe for penge, så kan vi gøre brug af betinget udførelse. Et eksempel på en betinget udførelse kunne være en "if statements", som det hedder på engelsk, og som man normalt kalder det. Et if statement fungerer på den måde, at du kan forstille dig at du opstiller et spørgsmål; hvis (det her er sandt) så gør (det her). Det kunne se sådan ud i JavaScript:

```
var kortSaldo = 100;

function setup() {

  if(kortSaldo < 5){
    console.log("Husk at tanke dit kort op!");
  }

}
```

Nu har vi altså vores kortSaldo, som er på 100kr, og hvis den ryger under 5kr, så siger den, at vi skal huske at tanke op. Lige nu er der ikke noget, som gør så kortSaldo bliver mindre, men det var bare for at vise et eksempel.

Jeg har primært gjort brug af forloops, som jeg vil komme ind på lidt senere, men jeg har også gjort brug af et if statement i min kode:

```
    if(debt <= 0){

      console.log("Dit lån er blevet betalt af efter " + (i - 1) + " år og " + months + "
måneder");

      i = year;
    }
```

”<=” betyder her; mindre eller lig med

Det her er inde i mit forloop for den negative rente. Den siger altså, at når vores restgæld er under eller i lig med 0, altså så vi ikke mangler at betale flere penge, så skal den skrive i consolen, hvor lang tid det tog at betale lånet af, og den skal stoppe mit forloop.

Løkker:

Man kan på en måde sige, at løkker er det samme som if statement, bare med den ene forskel at løkkerne bliver ved med at køre så længe, at betingelsen er sand, hvorimod if statement bare stopper, efter den har gjort det en gang. Der findes mange forskellige løkker, men de to mest kendte er ”while”- og ”for” løkken, eller “whileloop” og “forloop” på engelsk, som man normalt kalder dem, hvor whileloopet er den nemmeste at forstå.

For at forstå forloopet er det nemmere lige at lære lidt om whileloopet først. Man opsætter sit whileloop på samme måde som vores if statement, ved at stille spørgsmålet; hvis (det her er sandt) så gør (det her). Hvor if statementet stopper efter, at den har gjort (det her), så tager whileloopet og stiller det samme spørgsmål igen og igen, indtil betingelsen ikke er sand mere.

```
var WheelOnTheBuss = 0;

function setup() {

  while(WheelOnTheBuss < 5){
    console.log("Go round and round!");

    WheelOnTheBuss ++;
  }
}
```

”++” betyder her; at vi lægger 1 til

I dette eksempel kan man se, at vi har en variabel, som hedder “WheelOnTheBuss”, som er sat til 0. Vores whileloop siger så, at hvis 0 er under 5, så skal den gøre det inden i. Den skriver så i consolen, og lægger 1 til WheelOnTheBuss, som så nu er 1 i stedet for 0. Så spørger whileloopet, om 1 nu er mindre end 5, og det er det, så den kører igen, og 1 lægges til WheelOnTheBuss igen. Den bliver så ved med at køre, indtil WheelOnTheBuss er blevet over 5, hvorefter betingelsen ikke er opfyldt, så den stopper. Det vil så ende med, at der ville stå “Go round and round!” 5 gange i consolen.

Et forloop gør præcis det, men er lidt sværere at forstå. Forskellen på while- og forloopet er, at forloopet bare gør det hele lidt mere kompakt. Forloopet tager både, når vi laver vores variabel, vores betingelse og der hvor vi lægger en eller mere til for at stoppe loopet, og sætter det hele sammen et sted. Et eksempel kunne være det her:

```
for(var i = 0; i < 5; i++){
  console.log("Go round and round!");
}
```

Som man allerede nu kan se, er det en del mere kompakt, og denne kode gør det præcis samme som vores whileloop før. Man starter med at lave en variabel inde i forloopet og tildele den en værdi. Normalt skriver man bare ting som "i", i stedet for WheelOnTheBuss, da det fylder en del mere. Man skriver så et ";", hvorefter man skriver ens betingelse, ligesom man ville i ens whileloop eller if statement. Efter endnu et ";" skal man så sige, hvor meget man vil lægge til sin variabel, efter den er kørt igennem forloopet en gang, i dette tilfælde er det sat til 1 for, at vi får den samme kode som i vores whileloop. Så forloopet sætter "i" til at være 0, siger så i betingelsen, at "i" er lavere end 5, da den er sat til 0, derefter kører den koden og lægger til sidst 1 til "i", og tjekker om betingelsen stadig er sand.

I min kode har jeg gjort brug af forloops et par steder, f.eks. når jeg laver mine grafer til mine værdier, og når jeg udregner selve mine værdier igen og igen, men jeg har ikke brugt nogle whileloops. I det hele taget gør jeg mig ikke så meget brug af whileloops, da jeg personligt er mere til forloop, men det er meget subjektivt. For mig hjælper forloopet mig med at holde styr på det hele, så det er samlet et sted og ikke flyder over det hele i min kode.

Konklusion

Jeg har igennem mig SRC fundet ud af at man, ved hjælp af matematikken og programmeringens verden, som ligger meget tæt op om hinanden, som jeg også beskriver under *Matematik i min kode*, kan kreere noget, som kan hjælpe til, i ens hverdag. Ikke nok med at finde sammenhængen mellem matematik og programmering, er jeg også kommet frem til, at man kan bruge matematiske beregninger, som rekursionsligninger, til noget brugbart i virkeligheden, såsom udregning af renter. For at hoppe tilbage til min introduktion og problemstilling, så har jeg dannet et simpelt, men brugbart, program, som kan hjælp folk, som har svært ved at sætte sig ind i ting som lån og investeringer, med at forstå det. Det kan også hjælpe dem med at udregne, hvor meget de skal betale månedligt for at komme af med deres lån efter x antal år.

Bibliografi

1. Copyright forfatterne og Systime A/S 2021 (26. februar 2021) *Systime*. https://mathtxa.systime.dk/index.php?id=505&L=0&q=p505&redirected=1
2. Copyright forfatterne og Systime A/S 2021 (26. februar 2021) *Systime*. https://matbhtx.systime.dk/index.php?id=1331#c13414
3. w3schools.com (26. februar 2012) *w3schools* https://www.w3schools.com/js/js_let.asp
4. w3schools.com (26. februar 2012) *w3schools* https://www.w3schools.com/js/js_const.asp

Bilag 1 (Kode):

Alt kode er skrevet og fundet på af mig:

startAmount = 10000
renteProcent = 1.03
year = 50
Er flot for den positive graf

startAmount = 10000
renteProcent = 1.11
monthlyPayment = 100
year = 25
Er en flot graf for negativ graf

```
1
2  //vores fire variabler til at ændre vores udkomst
3  //monthlyPayment er kun til den NegativeRente, da vi ikke skal betale noget ved den positive
4  const startAmount = 10000;
5  const renteProcent = 1.03;
6  const monthlyPayment = 100;
7  const year = 50;
8
9  //vores faste variabler
10 let x = 0;
11 let uendelig = 100000;
12 let debt;
13 let months;
14 let profit;
15
16
17 function setup() {
18   createCanvas(windowWidth,windowHeight);
19   background(220);
20
21   //her vælger vi om vi vil have den positive eller negative rente, ved at lave den, som vi ikke
   vil have til en kommentar
22
23   //NegativRente();
24   PositivRente();
25
26 }
27
28 function NegativRente (){
29
30   //her laves den første rect af vores graf
31   rect(x, windowHeight - startAmount / 35, windowWidth / year, uendelig);
32
33   //der lægges til så den næste rect bliver lige til højre for
34   x = x + windowWidth / year;
35
36   //debt bliver sat til hvor meget man mangler at betale efter et år, og det skrives i cosolen
37   //xn = xn-1 * rente - yearlypayment
38   debt = startAmount * renteProcent - (12 * monthlyPayment);
39   console.log("Din restgæld efter 1 år er: " + debt);
40
```

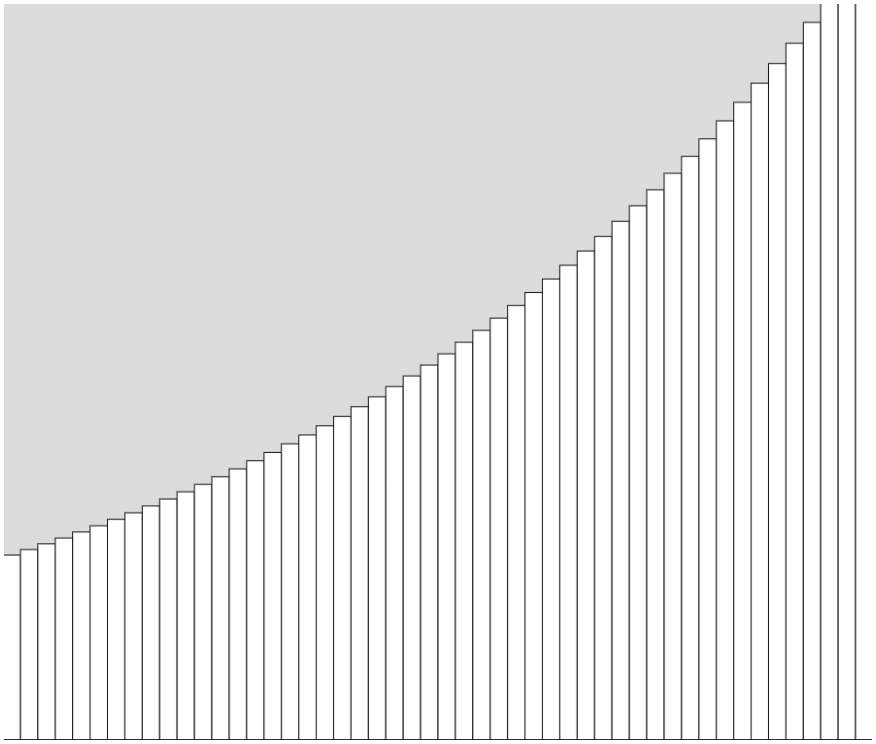
```
  //forloop bliver ved til man når det antal år man sætter ind i toppen
  for(i = 2; i <= year; i++){

    //tegner den nye rect
    rect(x, windowHeight - debt / 35,windowWidth / year, uendelig);

    //der lægges til så den næste rect bliver lige til højre for
    x = x + windowWidth / year;

    //udregner hvor mange måneder man mangler til når man når under 0
    months = debt/monthlyPayment;

    //debt bliver sat til hvor meget man mangler at betale efter forrige år, og det skrives i cosolen
    debt = debt*renteProcent-(12*monthlyPayment);
    console.log("Din restgæld efter " + i + " år er: " + debt);

    //når debt når under 0, skrives antal år før man har udbetalt sit lån i consolen og stopper forloopet
    if(debt <= 0){

      console.log("Dit lån er blevet betalt af efter " + (i - 1) + " år og " + months + " måneder");

      i = year;
    }
  }
}


function PositivRente (){

  //her laves en knap til den Rekursive løsning
  button1 = createButton('Rekursiv løsning');
  text('(plus graf)', windowWidth / 3 - 15, windowHeight / 2 + 35);
  button1.position(windowWidth / 3 - 45, windowHeight / 2);
  button1.mousePressed(rekursive);

  //her laves en knap til den Generelle løsning
  button2 = createButton('Generelle løsning');
  button2.position(windowWidth / 3 * 2 - 50,windowHeight / 2);
  button2.mousePressed(generelle);

}

function rekursive(){

  //baggrund sættes for at fjerne text fra før
  background(220);

  //fjerner de to knapper før
  button1.hide();
  button2.hide();

  //her laves den første rect af vores graf
  rect(x, windowHeight - startAmount / 50, windowWidth / year, uendelig);

  //der lægges til så den næste rect bliver lige til højre for
  x = x + windowWidth / year;

  //profit bliver sat til hvor mange penge man har efter et år, og det skrives i cosolen
  //xn = xn-1 * rente
  profit = startAmount*renteProcent;
  console.log("Antal penge efter 1 år: " + profit);
```

```
102
103   //forloop bliver ved til man når det antal år man sætter ind i toppen
104   for(i = 2; i <= year; i++){
105
106     //tegner den nye rect
107     rect(x, windowHeight - profit / 50,windowWidth / year, uendelig);
108
109     //der lægges til så den næste rect bliver lige til højre for
110     x = x + windowWidth / year;
111
112     //profit bliver sat til hvor mange penge man har efter forrige år, og det skrives i cosolen
113     profit = profit * renteProcent;
114     console.log("Antal penge efter " + i + " år: " + profit);
115
116   }
117 }
```

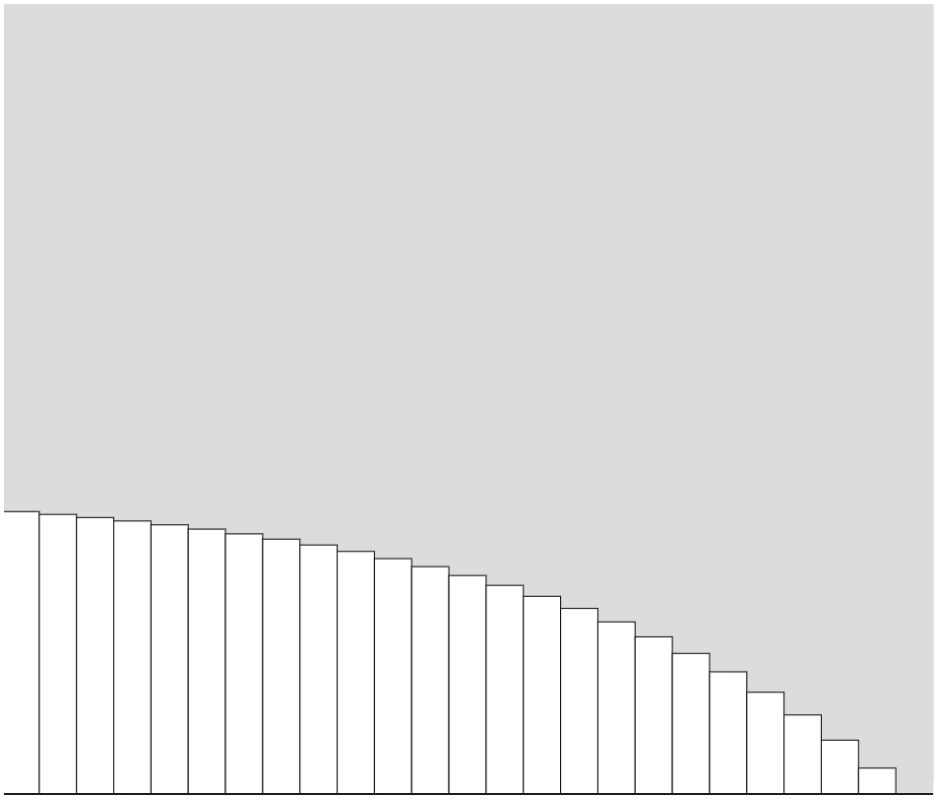
```
118
119 function generelle(){
120
121   //baggrund sættes for at fjerne text fra før
122   background(220);
123
124   //fjerner de to knapper før
125   button1.hide();
126   button2.hide();
127
128   //profit bliver sat til hvor mange penge man har det år man skriver i toppen, og det skrives
    midt på skærmen
129   //xn = startAmount * rente^n
130   profit = startAmount * pow(renteProcent,year);
131   textSize(20);
132   textAlign(CENTER);
133   text('Antal penge efter ' + year + ' år: ' + profit + ' kr', windowWidth / 2, windowHeight /
    2);
134
135 }
136
137 function draw() {
138
139 }
```

Bilag 2 (Eksempel på grafer)

Positiv rente (investering):



Negetiv rente (lån):



19.200 tegn = 8 sider