10 PRINT

> **Jeppe Veirum Larsen**, *opdateret 30/4 2020*
> ***Fag:*** *Programmering C*
> ***Sprog:*** *JavaScript*
> ***Tid:*** *5 timer*
>
> **Emner**
>
> - Løkker
> - Forgreninger
> - Funktioner
> - JavaScript
> - p5.js

I dette forløb kommer vi til at benytte og sammensætte både løkker og forgreninger. Eksemplet vi tager udgangspunkt i, generere et specielt møsnter og er bedere kendt som *10 PRINT*. Eksemplet stammer fra et gammelt og meget simpelt stykke kode, som fylder skærmen med et simpelt, men stadig visuelt imponerende randomiseret mønster. Den originale kode blev skrevet i sproget *BASIC* der blev benyttet på Commadore 64 computeren (verdens mest solgte computer). Den originale kode i BASIC kan ses neden for.

```
10 print chr$(205.5+rnd(1)); : goto 10
```

Her er et videoeksempel af hvordan det ser ud, når koden eksekveres.

Gennemgang af 10 PRINT eksemplet

For bedre at forstå hvad der sker i 10 print eksemplet, bliver basic koden gennemgået et element ad gangen.

- `10` er linie numret i BASIC. Man skal selv skrive linie nummer som er en meget vigtig komponent i sproget. I BASIC kan man ikke bare copy, past, cut og rykke rundt på kode som i dag. Derfor blev det en konvention, at man starter på linie 10 og lave nye linier i spring af 10 efterfølgende. Dette er for at skabe og sikre plads til, at man kan tilføje kode før og efter linierne, hvis behovet skulle opstå.
- `print` betyder at der skal outputtes tekst.
- `chr$()` er en funktion som tager en *integer* som input og slå op i en database og returnere det tilsvarende symbol fra en tabel i dette tilfælde `\` for værdien 205 og `/` for værdien 206.
- `205.5` en *float* værdi. Grunden til at der er valgt værdien 205.5 er at værdien bliver til en integer så alt efter kommaet bliver smidt væk. Så hvis rnd() returnere mellem 0 - 49 bliver resultatet 205 hvis den returnere 50 - 99 bliver det 206. Så det er en måde at lave en form for forgrening på.
- `+` er addition
- `rnd()` er en *funktion* der tager et tal som on input og returnere en random float mellem 0 og tallet(i dette tilfælde 0 til 0.99).
- `;` separere items i print statements.
- `:` er Basic's separator mellem instruktioner, hvor vi bruger `;` de fleste stedder i dag.
- `goto` springer til det linienummer angivet efter.

Eksemplet er det som kaldes en uendelig løkke. Den starter på linie 10, kører koden og får at vide at den skal starte forfra på på linie 10. Den får aldrig ordre om at stoppe og kører der for uendeligt eller til at computeren slukkes. Dette er sjældent ønskeligt, så derfor vil man ofte lave en konstruktion, hvor den kun kører et vist antal gange.

Mere information

For at se alle symbolerne i BASIC på Commodore 64 er her et link til Commodore 64 PETSCII codes.

Der er udgivet en hel bog om lige præcis dette stykke kode den findes på bogen tilhørende hjemmeside . Så hvis man ønsker at bladre lidt i den, kan hele 10 print bogen downloades her (50 Mb).

Opgave

> Opgave 🟥
>
> Implementer 10 PRINT eksemplet i p5.js uden at følge teksten videre.
>
> Ekstra opgave 1: Lav fire muligheder i stedet for to: tilføj mellemrum eller kryds som mulighed.
>
> Ekstra opgave 2: Gør implementeringen opløsning uafhængig. Lige megen hvilken opløsning du vælger skalerer den der til.

> Opgave 🟨
>
> Følg dispositionen herunder og implementer 10 PRINT eksemplet i p5.js
>
> *Evt. tilføj to ekstramuligheder (kryds, firkant, cirkel eller mellemrum)*

Når den røde eller gule opgave er løst kan man vælge at se eksemplet udført på video. Det er tit interessant at se andre skrive kode, da man løser forskellige problemer på forskellige måder. Det giver en god indsigt i hvordan man også kan løse bestemte ting, og man kan også se at andre også laver fejl.

> Opgave 🟩

> Læs hele dokumentet og følg videoen for at implementere 10 PRINT eksemplet i p5.js
>
> Her er en video der viser hvordan man laver 10 PRINT i p5.js

Fra BASIC til JavaScript

For at kunne genskabe eksemplet fra BASIC skal vi kigge lidt nærmere på koden og se hvordan den kan oversættes til JavaScript og p5.js.

P5.js - setup() og draw()

P5. fungere ved at have to funktioner `function setup()` og `function draw()`. Disse to functioner tjene to forskellige formål. Når koden eksekveres køres `function setup()` én gang og kun én gang modsat `function draw()` der kører ca. 60 gange i sekundet ind til programmet lukkes.

I dette forløb vil vi kun skrive kode i setup da vi ikke ønsker at eksekvere koden til 10 print 60 gange i sekundet, men kun en enkelt gang for at fylde vores canvas.

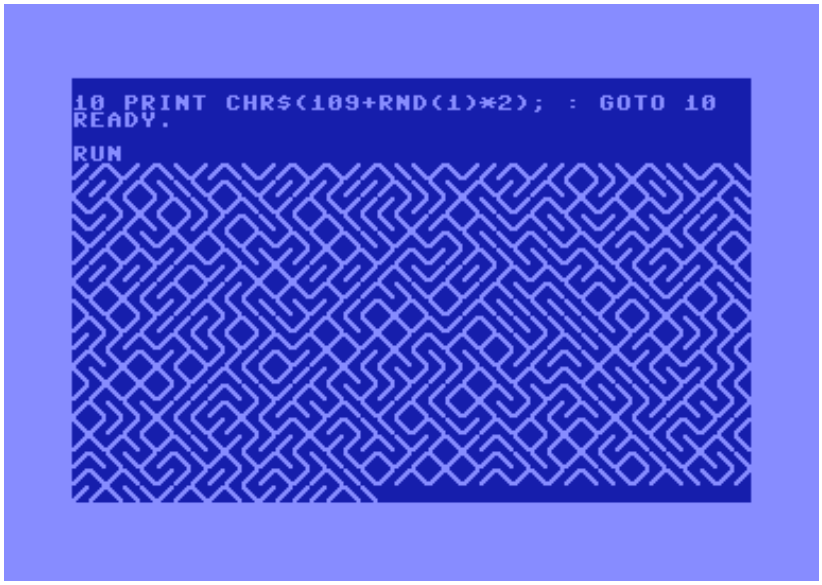
```
// globalde variabler skrives her
let jegErEnGlobalVariabel = 10;
let detErJegOgså = "Svend";

function setup(){
// her skriver vi vores kode til 10 print eksemplet
  createCanvas(800, 800);
}

function draw(){
// vi benytter ikke denne funktion i dette eksempel
}
```

Det visuelle

Den visulle del af det originale eksempel er gennem en form for tekst-konsol hvor symboler/tekst output printers en efter en.



Det kan vi ikke på samme måde i JavaScript og p5.js da konsollen kun er for debugging og ikke synlig for den almindelige bruger. Så her er man nødt til at finde finde de tilsvarende elementer i p5.js eller noget som kan erstatte det.

Opløsning

Modsat eksemplet i BASIC, der har predefinerede størrelser på symboler, hvor mange der kan være per linie og lineafstand, har vi brug for selv at definere en overordnet opløsning som vores eget eksempel skal foregå i. det gør vi med `createCanvas(bredde, højde)`. Derud over skal vi inddele vores overordnede opløsninger i felter/gitter. Det kunne f.eks. være en canvas opløsning på 800 x 800 og et gitter på 20 felter x 20 felter. Hvis canvas f.eks. er 800 pixel x 800 pixel, så er et gitterfelt 40 x 40 pixels stort.

> Husk at den positive retning på y-aksen er ned.
>
> x
>
> ------>
>
> |
>
> y v

```
let setup(){
  createCanvas(800, 800);
}
```

Line()

Da vi ikke kan benytte os af prefabrikerede symboler, er vi nødt til at lave vores egen. Her kommer p5.js på banen, der har en masse indbyggede klasser og funktioner rettet mod hurtig grafisk fremstilling.

For at tegne en streg benyttes p5 funktionen `line(x1, y1, x2, y2)` . Den tager to koordinatpunkter og tegner en streg mellem dem.

```
// tegner en linie i første gitterfelt (backslash) fra (0,0) til (20,20)
line(0, 0, 20, 20)
```

Random()

For at skabe et interessant mønster, som i eksemplet, har vi brug for noget tilfældighed. Her kan vi gøre brug af `random(fra, til)` funktionen som er indbygget i p5. Den tager to parametre som input, der er fra og til.

```
random(0,1); // returnere en værdi mellem 0 (inclusiv) og 1 (eksklusiv)
```

If/else

Vi operere med to muligheder, slash og backslash. Derfor passer en *if/else* fint til dette.

Ligesom i BASIC eksemplet skal vi bruge vores random værdi til at vælge om vi skal tegne den ene eller den anden streg.

```
if(random > værdi){
  //tegn den ene type
} else {
  //tegn den anden
}
```

Løkker

I det oprindelig eksempel klarede konsollen den visuelle del. Både den horisontale og vertikale. Vi er dog nødt til selv, at styre hvor vi tegner vores streger, da vi ikke har den mulighed.

Her kommer et koncept der hedder nested loops. Det vil sige en løkke inde i en anden løkke. For at gøre det meget simplere, deler vi det op i to trin.

1 dimension

Start med at løse problemet i 1 dimension, dvs. for den første linie fyldt af tilældige streger. Hvis dette først er løst, er det 'bare', at rykke en linie ned og gør det igen og igen og igen...

Styrken ved en løkke er at vi kan udnytte tællevariablen i dette tilfælde `i`. I eksemplet under tæller jeg i spring af 20 og ligger det til min streg.

```
for(let i = 0; i < breden; i = i+20){
  line(0 + i, 0, 20 + i, 20)
}
```

2 dimensioner

I det 1 dimensionelle eksempel går det fint med en enkelt linie, men vi mangler at kunne bevæge os ned og tegne den næste på samme måde. Det er her *nested loops* kommer ind.

```
for(let j = 0; j < højde; j = j + 20){
  for(let i = 0; i < breden; i = i+20){
    line(0 + i, 0 + j, 20 + i, 20 + j)
  }
}
```

Med eller uden funktioner

Det er muligt at inddrage funktioner i løsningen. Her er et eksempel på hvordan man kan lave en model af backslash i en funktion og pass parametre ind i fuktionen og derved tgne den hvor man ønsker.

```
function backSlash(x1, y1, x2, y2){
  line(0 + x1, 0 + y1, 20 + x2, 20 + y2);
}
```

Disposition over fremgangsmåde

- Vælg en opløsning `creteCanvas(bredde, højde)` og en gitter størrelse på f.eks. 20 x 20.
- Brug `line(x1, y1, x2, y2)` til at tegne de to skråstreger inden for grid størrelsen.
- Lav en random generator `random(min, max)`
- Lav en `if()` der kan vælge mellem den en eller anden type streg baseret på din random værdi. Og tegn den valgte streg i det gørste gitter/ på første plads.

```
if(random > værdi){
  //tegn den ene type
} else {
  //tegn den anden
}
```

- Brug to løkker til at tegne linierne i gitteret, hvor den ene løkke incrementerer x og den anden incrementerer y.

```
for(fra; til; størrelse skridt){
  //gør noget x antal gange
  for(fra; til; størrelse skridt){
  //gør noget x antal gange
  }
}
```