App Lab

Vigtige kommandoer

Malene Cramer Engebjerg

Version 3

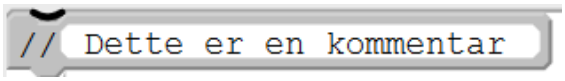
Indhold

I App Lab kan man lave det, der hedder "blok-programmering". Det vil sige, at man kan programmere en app, uden at skulle *skrive* en masse tekst, hvor man er nødt til at huske på en masse kommandoer og sørge for at skrive dem meget præcist. I stedet kan man trække "blokke" over i et arbejdsområde i App Lab og arbejde derfra.

Denne note er ikke tænkt som en egentlig introduktion til App Lab og kan således ikke stå alene. Den er mere tænkt som et supplement og et opslagsværk med uddybende forklaringer til nogle af de vigtigste kommandoer, man får brug for i App Lab.

1. Kommentarer

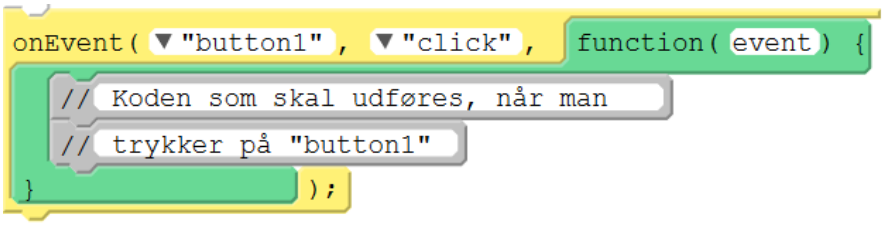
En kommentar i koden (som du bør bruge ofte!) ser sådan ud:



Den findes i værktøjskassen under "Functions" (grøn).

2. Der skal ske noget, når man trykker på en knap

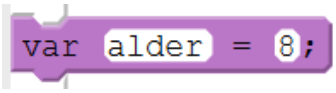
Hvis man ønsker, at der skal ske noget, når man f.eks. trykker på en knap, så skal man bruge følgende kommando:



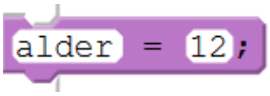
Her er `"button1"` id'et på knappen og "click" betyder, at der sker noget, når man trykker/klikker på knappen.

3. Variable og datatyper

Man kan tænke på en variabel som en lille skuffe i "maven" af computeren, som har et navn. Inde i denne skuffe kan man så opbevare en værdi. En variabel i App Lab laves sådan her:



Her er der lavet (erklæret) en variabel, som har navnet `alder` og denne variabel har fået tildelt værdien 8. Hvis man får brug for denne værdi, kan man bare skriver `alder` i stedet for 8. Hvis man gerne vil ændre værdien af variablen - lad os sige at den nu skal have værdien 12 - så skriver man bare:



Læg mærke til at der ikke længere står `var` foran. Det er fordi, at variablen allerede er "lavet", den har bare fået en ny værdi (man har taget værdien 8 op af skuffen og puttet 12 ned i stedet for).

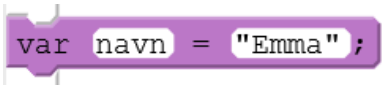
Hvis man i stedet gerne vil lægge 1 til den nuværende værdi af variablen, gør man sådan her:

Nu vil variablen `alder` have værdien 13 i stedet for 12.

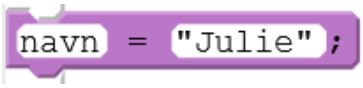
Den slags værdi, som en variabel kan indeholde, kaldes for *datatypen*. Variablen `alder` ovenfor har datatypen *tal* (på engelsk *number*).

Variable behøver dog ikke at indeholde tal, de kan også indeholde tekst (på engelsk siger man, at datatypen er en "string" og på dansk nogle gange "streng" eller "tekststreng").

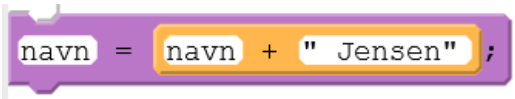
Her er et eksempel:



Denne variabel har navnet `navn` og værdien `"Emma"`. Bemærk at tekst skrives i dobbelt anførselstegn ("plinger"), mens variabelnavne og tal ikke gør det. Lad os sige at vi ikke længere vil gemme navnet `"Emma"`, men i stedet `"Julie"` i variablen `navn`, så gøres det sådan her:



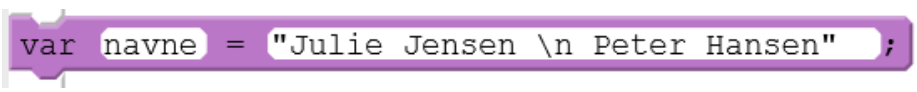
Hvis vi i stedet for `"Julie"` også gerne vil gemme Julies efternavn, som er `"Jensen"`, så kan man tilføje noget til en tekst-variabel på følgende måde:



Nu indeholder variablen `navn` værdien `"Julie Jensen"`. Man kan tænke på det på den måde, at man lægger noget ekstra ned i "skuffen". Læg mærke til at der står et mellemrum lige inden `"Jensen"`. Hvis det ikke havde stået der, ville variablen `navn` i stedet have fået værdien

`"JulieJensen"`

og det var ikke helt det, vi havde tænkt. Ønsker man at indsætte en ny linje bruges `"\n"` (`n` står for newline). F.eks. vil

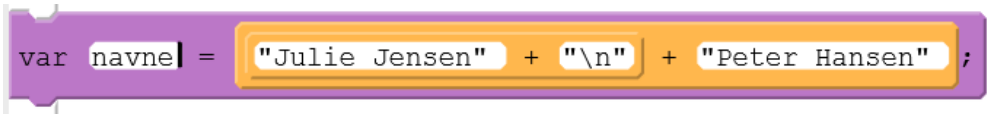


give resultatet (mere korrekt: værdien af variablen `"navne"` vil være):

```
"Julie Jensen

Peter Hansen"
```

Alternativt kan det også skrives:

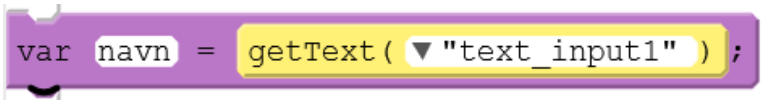


4. At få fat i tekst fra et tekstfelt

Nedenfor ses en enkel app med et tekst inputfelt, som har id `"text_input1"` (vælg selv bedre og mere sigende navne!).

Bemærk, når man holder musen henover tekst inputfeltet, så vises feltets placering og feltets id.

Brugeren kan skrive noget i dette feltet, og hvis vi gerne vil *læse*, det brugeren har skrevet, og gemme det i en variabel, gøres det vha. kommandoen `getText()`:



Bemærk, at i parentesen skriver man tekst inputfeltets id.

5. At skrive tekst til et tekstfelt

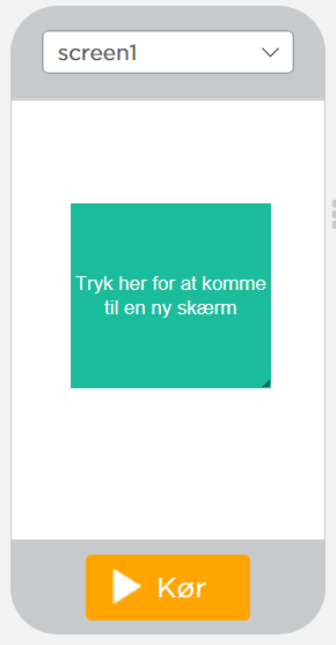
Ønsker man omvendt at *skrive* til et tekstfelt gøres det vha. `setText()` kommandoen:

```
setText( "text_input1" , "Her er min tekst" );
```

Det første man skriver i parentesen er id'et på tekstfeltet, mens det andet er den tekst, man gerne vil have skrevet.

6. At skifte skærm

Hvis man gerne vil skifte til en ny skærm, når man f.eks. trykker på en knap gøres det vha. `setScreen()` kommandoen.



Knappen på skærmen til venstre har id'et `"button1"`. Nedenstående kode sørger for at skifte til en ny skærm med id'et `"screen2"`, når man trykker på knappen (`"button1"`).

```
onEvent( "button1" , "click" , function( event ) {
  setScreen( "screen2" );
} );
```

Se også videoen "Tilføje skærmbillede og knap".

7. Forgrening med én gren (if)

Hvis man kun skal udføre et stykke kode (B), hvis en given betingelse er opfyldt, skal man bruge en if-sætning:

```
if ( A==sand ) {
  // udfør B
}
```

Her udføres koden B kun, hvis betingelsen A er sand. Betingelsen kunne f.eks. være en af følgende:

```
alder > 18
```

```
farve == "blå"
```

```
BMI <= 25
```

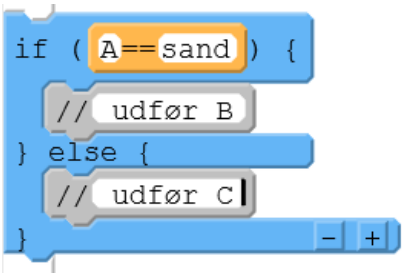
```
antal != 6
```

Her vil betydningerne henholdsvis være:

1) Hvis variablen `alder` har en værdi, der er større end `18`, udføres koden i B.
2) Hvis variablen `farve` har værdien `"blå"`, udføres koden i B.
3) Hvis variablen `BMI` har en værdi, som er mindre end eller lig med `25`, udføres koden i B.
4) Hvis variablen `antal` *ikke* er `6`, udføres koden i B.

8. Forgrening med to grene (if-else)

Hvis man ønsker at få udført et stykke kode (B), hvis betingelsen A er sand, og et andet stykke kode (C), hvis betingelsen A ikke er sand (dvs. den er falsk) skal man bruge en if-else sætning:



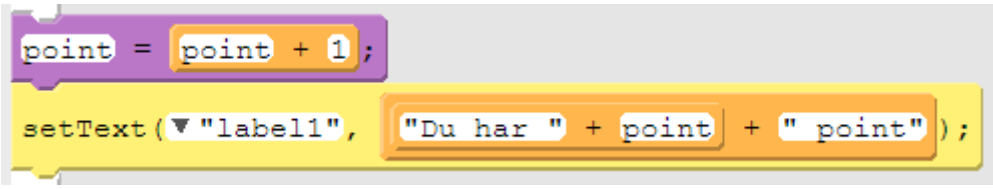
Betingelsen, som skal vurderes, kan igen være én af de foregående.

Se også videoen "Sådan laves en if-else sætning".

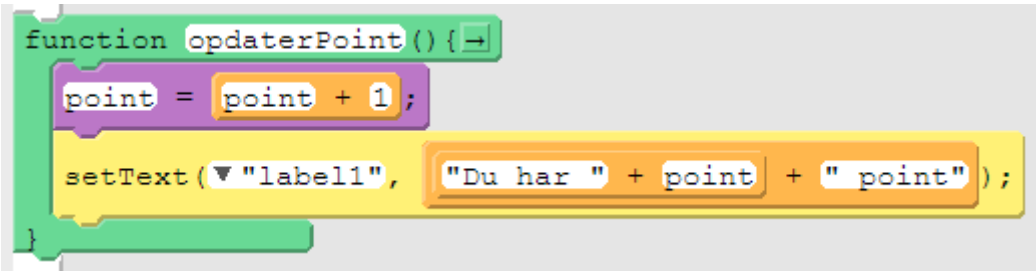
9. Funktioner

Når man programmerer, finder man nogle gange ud af, at man skriver den samme stump kode flere gange. Hvis man gør det, kan man med fordel indkapsle denne stump kode i en funktion.

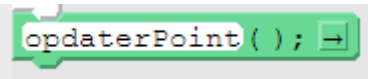
Lad os forestille os at vi har et spil, hvor brugeren i forskellige situationer får et ekstra point samtidig med, at den nye pointscore skal vises på skærmen. Så kunne man forestille sig, at man flere i gange i koden ville få brug for at skrive:



I stedet for at gentage disse to linjer kode mange gange, kan man i stedet for erklære en funktion. Vi vælger her, at kalde funktionen for `opdaterPoint()`. Det ser sådan her ud:



Man kan tænke på det på den måde, at vi har bygget en lille "maskine", som kan opdatere antallet af point og skrive antallet af point ud i en label på skærmen. Selve definitionen af funktionen ovenfor gør ikke noget i sig selv. Man er nødt til at "tænde for maskinen", førend der kommer til at ske noget. Man siger, at man *kalder funktionen*. Det gør man simpelthen bare ved at skrive funktionens navn efterfulgt af parenteser:



Hver gang denne linje optræder i ens kode, så bliver den faktisk erstattet af de to linjer, som står inde i funktionsdefinitionen ovenfor. Det er altså på denne måde, at man "tænder" for den maskine, som vi har bygget. På skærmen kunne resultatet f.eks. være:

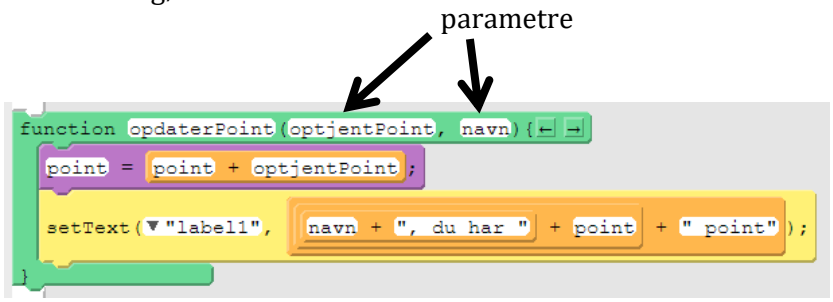
Du har 7 point

Det kan måske virke som spild af tid og kræfter at definere funktioner, men der er flere grunde til, at det er smart:

1) Koden kommer til at fylde mindre. Når en funktion defineres, kan selve definitionen være lang, men hver gang funktionen kaldes, kan man nøjes med en enkelt linje.
2) Koden bliver nemmere at læse. Vi behøver bare én gang at sætte os ind i, hvad funktionen gør, i stedet for at man skal forholde sig til det, hver gang man har brug for funktionen.
3) Hvis man får brug for at ændre lidt i pointgivningen - det kunne f.eks. være, at man beslutter sig for at lægge to point til i stedet for ét, så skal man kun ændre i koden ét sted. Alternativet vil være, at man skal ændre koden alle de steder, hvor man har haft brug for at opdatere pointscoren. At ændre noget ét sted i stedet for mange steder reducerer risikoen for at lave fejl, hvis man f.eks. glemmer at lave ændringen et eller flere steder. Man siger, at koden er nemmere at *vedligeholde*.

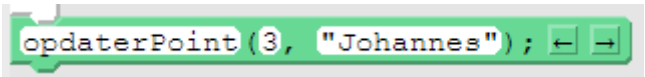
Det er ikke altid sådan, at man fra start tænker "Hov, det her skal jeg bruge flere gange - jeg laver lige en funktion!". Ofte finder man først ud af det efter noget tid, og så kan man vælge at ændre i sin kode ved at indkapsle de linjer kode, som man har brug for flere gange, i en funktion. Når man gør det, siger man, at man *restrukturerer* sin kode.

Faktisk er funktioner endnu smartere. Forestil dig at du kender spillerens navn, og at vi sammen med pointscoren også vil skrive navnet på spilleren. Samtidig forestiller vi os også, at man kan optjene et forskelligt antal point, som skal lægges til den samlede pointscore. Det kan man klare ved at *parametrisere funktionen*. Det gøres sådan her:

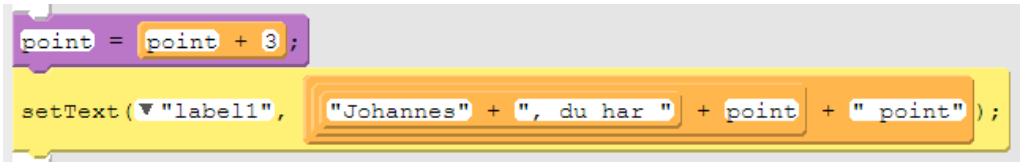


Læg mærke til at der nu står to ting i parentesen efter funktionsnavnet `opdaterPoint`: Nemlig `optjentPoint` og `navn`. Disse to kaldes for parametre. Idéen er nu, at man kan sende et bestemt antal optjente point og et bestemt navn med, når man kalder funktionen, og så vil disse konkrete værdier blive indsat de steder i funktionen, hvor der står `optjentPoint` og `navn`.

Lad os sige, at navnet på spilleren er Johannes, og at den samlede pointscore skal opdateres med 3 point. Så kaldes funktionen på denne måde:



De faktisk værdier (her `3` og `"Johannes"`), som man sender med funktionen, kaldes for argumenter. Når man kalder funktionen med disse to argumenter, så bliver `optjentPoint` i definitionen af funktionen erstattet med `3`, og `navn` bliver erstattet med `"Johannes"`, så der reelt set kommer til at stå:



Men hele pointen er, at man ikke behøver at lave disse ændringer direkte i koden, når blot man kalder funktionen med argumenter `3` og `"Johannes"`. På skærmen kunne der så f.eks. stå:

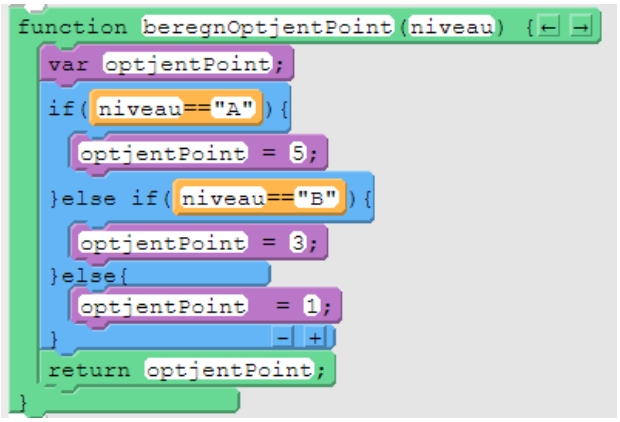
Johannes, du har 9 point

Der er ingen begrænsninger på antallet af parametre, som en funktion kan have og idéen med at parametrisere funktioner gør, at man med forholdsvis få linjer kode, kan skrive noget, som er utrolig generelt.

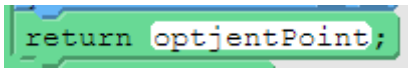
De funktioner, vi har set på indtil nu, har opdateret teksten i en label på skærmen. Det vil sige, at effekten af funktionen har været en opdatering i brugergrænsefladen. Nogle gange har man imidlertid brug for en funktion, som beregner en konkret værdi. Lad os forestille os at man i spillet kan være på tre forskellige niveauer: A, B og C. Man starter på niveau C, og så arbejder man sig på en eller anden måde op på niveau B, og til sidst ender man på niveau A. Niveauet får betydning for, hvor mange point man optjener, når man har udført en eller anden handling. Vi siger, at det hænger sammen på denne måde:

Niveau	**Optjener…**
A	5 point
B	3 point
C	1 point

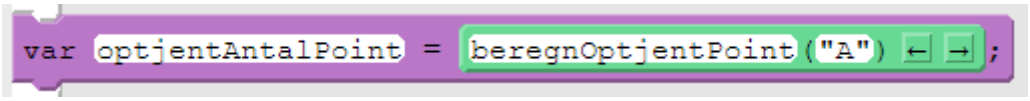
Så kunne man lave en funktion, der har niveauet som parameter og som returnerer værdien af det optjente antal point. Det kunne se sådan her ud:



Læg her mærke til at vi først erklærer en variabel `optjentPoint`. Denne variabel tildeles en værdi ved hjælp af en forgrening svarende til den regel, vi opstillede i tabellen ovenfor. Det nye er linjen:

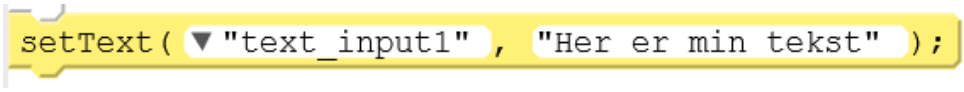


Det betyder, at funktionen ender med at "spytte" værdien af `optjentPoint` ud (man siger, at værdien af `optjentPoint` returneres af funktionen). Det kan f.eks. bruges til at tildele en variabel en værdi. Her er et eksempel:



Variablen `optjentAntalPoint` får her værdien `5`, fordi funktionen `beregnOptjentPoint()` kaldes med argumentet `"A"`. Havde vi f.eks. i stedet kaldt funktionen med argumentet `"B"`, ville værdien af `optjentAntalPoint` i stedet være `3`.

Faktisk er du stødt på funktioner i App Lab mange gange før helt uden at tænke over det! F.eks. har vi tidligere set på eksemplet:



Her kaldes funktionen `setText()` (som er en funktion, som App Lab stiller til rådighed for os) med argumenter `"text_input1"` og `"Her er min tekst"`. Det nye er, at vi nu også har lært, hvordan vi programmerer vores egne funktioner.

10. Liste/array

Hvis man har brug for en variabel, som kan indeholde mere end én værdi ad gangen, skal man bruge et array (kaldes også for en liste). Det gøres sådan her:

```
var minListe = ["Ida","Viktor","Frederik"];
```

Dette array har navnet `minListe` og indeholder værdierne `"Ida"`, `"Viktor"` og `"Frederik"` (som også kaldes for elementer).

Ønsker man at få fat i det første element i arrayet skriver man:

```
minListe[0];
```

Bemærk, at man starter ved 0. Derfor fås det tredje element i arrayet ved at skrive:

```
minListe[2];
```

Ønsker man længden af arrayet, skriver man:

```
minListe.length;
```

I dette eksempel er længden af listen 3, fordi der er 3 elementer i listen.

11. For-løkke

Når man ønsker at gentage noget kode et bestemt antal gange, kan man bruge en for-løkke. Hvis man f.eks. vil udskrive alle navnene i arrayet i det foregående eksempel til konsollen (som vises nederst på siden, når man har App Lab åben), så skriver man:

```
for (var i = 0; i < minListe.length; i++) {
  console.log(minListe[i]);
}
```

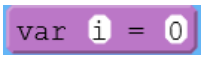
Det kræver lidt forklaring. For det første har vi set, at `minListe.length` har værdien `3`, fordi der er 3 elementer i listen. Dvs. at i dette eksempel svarer ovenstående til:

```
for (var i = 0; i < 3; i++) {
  console.log(minListe[i]);
}
```

Overvej hvorfor det er smartere at skrive `minListe.length` i stedet for 3!

Dernæst ser vi, at der i parentesen lige efter `for` står tre sætninger:

Den første sætning:

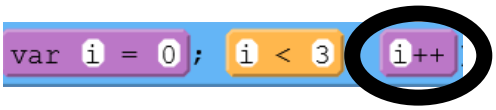


definerer midlertidigt en variabel `i`, som starter med at have værdien `0`. Dernæst undersøges det, om betingelsen i den anden sætning er opfyldt. Dvs. vi spørger om

Da 0 er mindre en 3, er betingelsen opfyldt, og koden der står inde i løkken udføres. I dette eksempel svarer dette til, at følgende kode udføres:

Bemærk, at `i` er skiftet ud med `0`, fordi variablen `i` har værdien `0`. Det, der står i `minListe` på plads nr. 0 (man starter som tidligere nævnt altid med at tælle fra 0), er `"Ida"`, og derfor skrives der nu ”Ida” i konsollen.

Nu sker der det at den tredje sætning i:



udføres. Det ser igen lidt mærkeligt ud, men det betyder: gør værdien af variabel `i` én større. Da værdien af variablen `i` er `0`, så skal værdien af `i` altså ændres til `1`. Og så kører det hele en gang til. Vi spørger igen - er:

Og ja 1 er mindre end 3 og derfor udføres følgende kode:

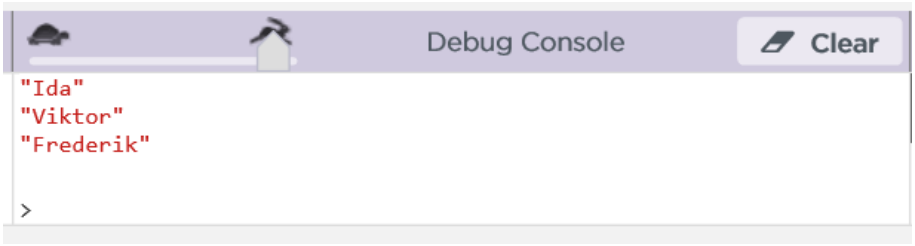
Elementet på plads nr. 1 i arrayet er `"Viktor"`, og derfor skrives der nu ”Viktor” i konsollen. Så hopper vi igen tilbage til den tredje sætning:

og tæller værdien af variablen `i` én op. Den har altså nu værdien `2`. Igen ser vi at 2 er mindre end 3, og vi udfører derfor koden:

Der skrives altså nu "Frederik" i konsollen. Så tæller vi igen værdien af variable `i` én op, så den nu har værdien `3`. Og vi spørger igen - er:

Men da 3 ikke er mindre end 3, er betingelsen altså ikke længere sand, og vi forlader derfor `for`-løkken.

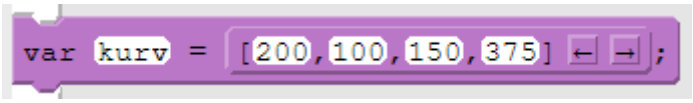
Resultatet af denne lille stump kode bliver, at der i konsollen til sidst står:



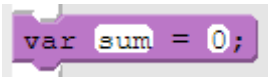
Se også videoen "Sådan laves en løkke".

12. At beregne en sum ved hjælp af en for-løkke

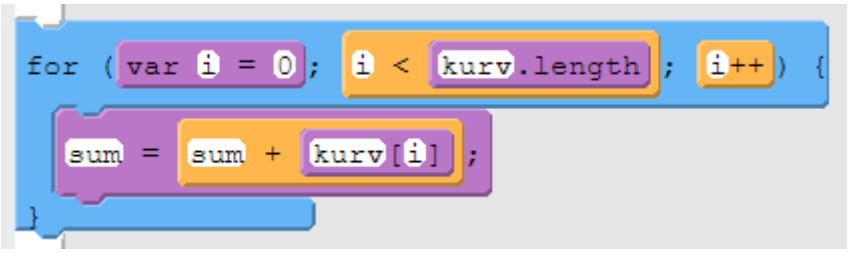
Man får ofte brug for at beregne summen af en række tal, som står i et array. Det kunne f.eks. være, hvis man har et array, som angiver prisen på en række varer i en kurv. Lad os forestille os at vi har følgende array:



som repræsenterer prisen på de varer, vi har i en kurv i en webshop. Vi vil nu gerne udregne summen af disse tal, som jo svarer til, hvor meget vi har købt for. Det kan gøres ved, at vi først erklærer en variabel `sum`, som til start tildeles værdien `0`:



Vi vil så ved hjælp af en `for`-løkke gennemløbe alle tallene i arrayet og løbende lægge de enkelte værdier til variablen `sum`. Det gøres sådan her:



I første gennemløb af `for`-løkken er `i=0`, og der kommer derfor inde i `for`-løkken til at stå:

```
sum = sum + kurv[0]
```

På dette tidspunkt har variablen `sum` værdien `0`, og `kurv[0]` har værdien `200`. Det vil sige, at den nye værdi af `sum` bliver `200`.

I næste gennemløb er `i=1`, og der kommer til at stå:

```
sum = sum + kurv[1]
```

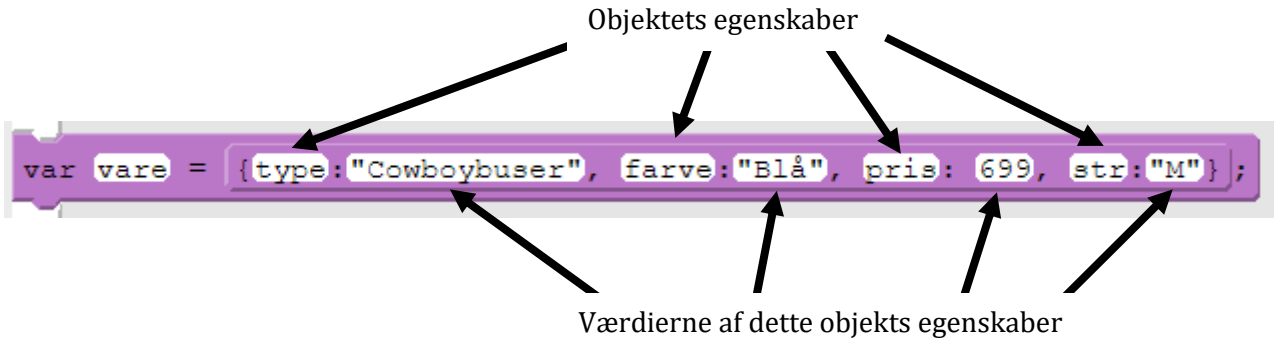
Nu har variablen `sum` værdien `200` og `kurv[1]` har værdien `100`. Det vil sige, at den nye værdi af `sum` bliver `300`.

Sådan fortsættes indtil hele arrayet er løbet igennem, og variablen `sum` vil til sidst have værdien `825`.

13. Objekter

Nogle gange har man brug for at gemme oplysninger om en størrelse, som er lidt mere kompleks, end hvad der kan beskrives med en tekststreng eller et tal. Det kunne f.eks. være, at man gerne vil gemme oplysninger om en vare i en webbutik. Vi forestiller os, at vi for alle varer i butikken gerne vil gemme oplysninger om varens type (f.eks. om det er cowboybukser, en bluse eller en nederdel), farven, prisen og størrelsen. I stedet for at erklære fire forskellige variable, hvori man kan gemme de fire oplysninger, så kan man gemme al informationen i én variabel, som er af den komplekse datatype "objekt".

Et sådant objekt erklæres og tildeles en værdi på denne måde:



Her siger man, at `type`, `farve`, `pris` og `str` er objektets egenskaber, som i dette eksempel har værdierne `"Cowboybukser"`, `"Blå"`, `699` og `"M"`.

Hvis man specifikt gerne vil have fat i egenskaben `type`, så skriver man

```
vare.type
```

Og tilsvarende hvis det f.eks. er prisen man er ude efter:

```
vare.pris
```

Fordelen ved objekter er, at vi nu logisk kan samle al information om den samme vare i én variabel i stedet for fire. Vi kan derfor også nemt definere flere varer på tilsvarende vis:

```
var vare0 = {type:"Cowboybuser", farve:"Blå", pris: 699, str:"M"};
var vare1 = {type:"Nederdel", farve:"Gul", pris: 250, str:"S"};
var vare2 = {type:"Bluse", farve:"Grøn", pris: 499, str:"L"};
```

Og vi kan nu gemme informationen om alle varerne i webbutikken i et array:

```
var varer = [vare0, vare1, vare2];
```

Her vil det selvfølgelig være realistisk at forestille sig, at vi har mange flere varer end kun tre! Tilsvarende kan man også gemme de varer, som en kunde lægger i sin kurv i et array. Lad os sige, at vi har købt to par cowboybukser og en bluse. Så vil kurven se sådan her ud:

```
var kurv = [vare0, vare0, vare2];
```

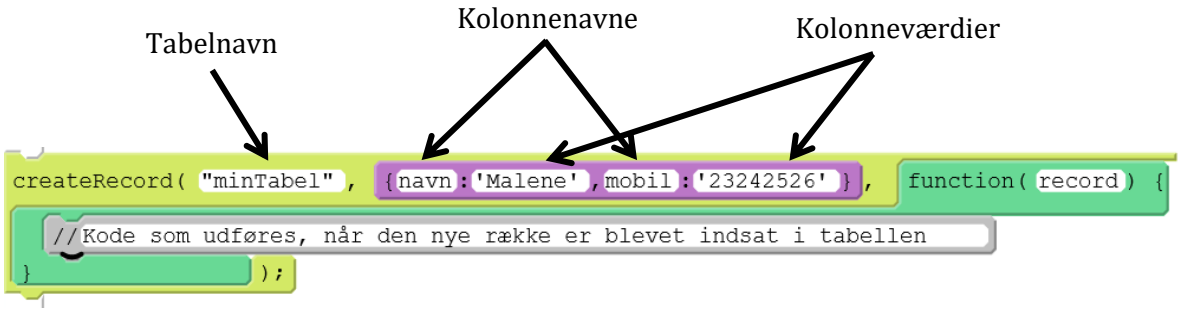
Hvis vi nu er interesseret i at finde den samlede pris på varerne i kurven, så kræver det kun en lille ændring i forhold til den måde, som vi beregnede den samlede pris på i foregående afsnit:

```
for (var i = 0; i < kurv.length; i++) {
  sum = sum + kurv[i].pris;
}
```

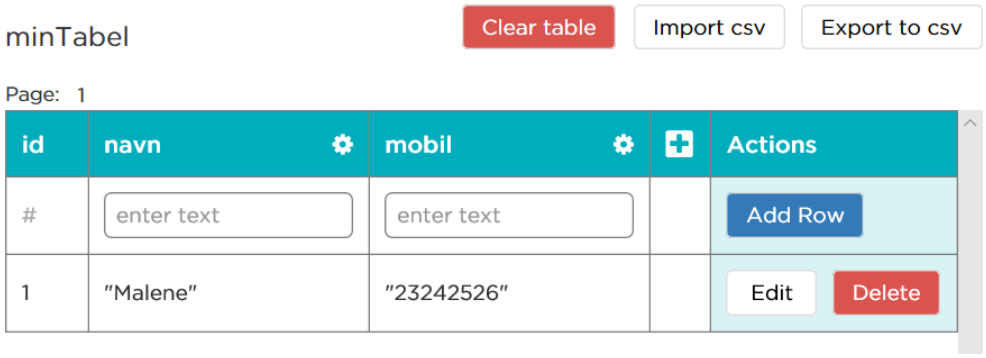
Bemærk her at vi nu er nødt til at skrive `kurv[i].pris`, fordi `kurv[i]` nu er et objekt, som repræsenterer en vare, og hvis man skal bruge egenskaben `pris`, så må man skrive `kurv[i].pris`.

14. Gemme data i datalaget

Ønsker man at indsætte en række i en tabel i datalaget i App Lab skal `createRecord()`-kommandoen benyttes. Nedenfor er vist et eksempel på en kommando, hvor man indsætter en række i en tabel, som hedder `"minTabel"`. Tabellen har to kolonner (dvs. søjler), som hedder `navn` og `mobil`. Kommandoen herunder indsætter værdien `"Malene"` i kolonnen `navn` og værdien `"23242526"` i kolonnen `mobil`.

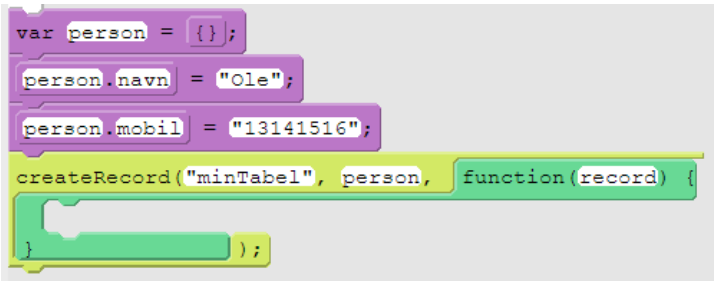


Det giver følgende resultat i datalaget:



Læg mærke til at det faktisk er et objekt, vi giver funktionen `createRecord()` med som argument (objektet står i det lilla felt som `{navn: 'Malene', mobil: '23242526'}`).

En alternativ måde at indsætte en række i datalaget er derfor direkte, at give `createRecord()` et objekt med som argument:



Her erklæres først en variabel `person`, som tildeles værdien `{}`, som svarer til det tomme objekt. Herefter gives egenskaben `navn` værdien `"Ole"`, og egenskaben `mobil` gives værdien `"13141516"`. Funktionen `createRecord()` kaldes nu med dette objekt som argument. Resultatet i datalaget er nu:

id	navn	mobil		Actions
#	enter text	enter text		Add Row
1	"Malene"	"23242526"		Edit Delete
2	"Ole"	"13141516"		Edit Delete

Bemærk, at objektets egenskaber skal have præcis de samme navne som de tilsvarende kolonner i datalaget (vær også opmærksom på små og store bogstaver).

Bemærk også, at det faktisk ikke er nødvendigt at oprette tabellen i datalaget først. Det sker automatisk, første gang App Lab bliver bedt om at indsætte data i en tabel, som ikke allerede er oprettet. Det betyder også, at hvis du får stavet tabelnavnet forkert i et kald til `createRecord()`, så får du bare indsat i en ny tabel i stedet for den tabel, som du havde tænkt. Det kan godt give anledning til lidt hovedbrud 😉.

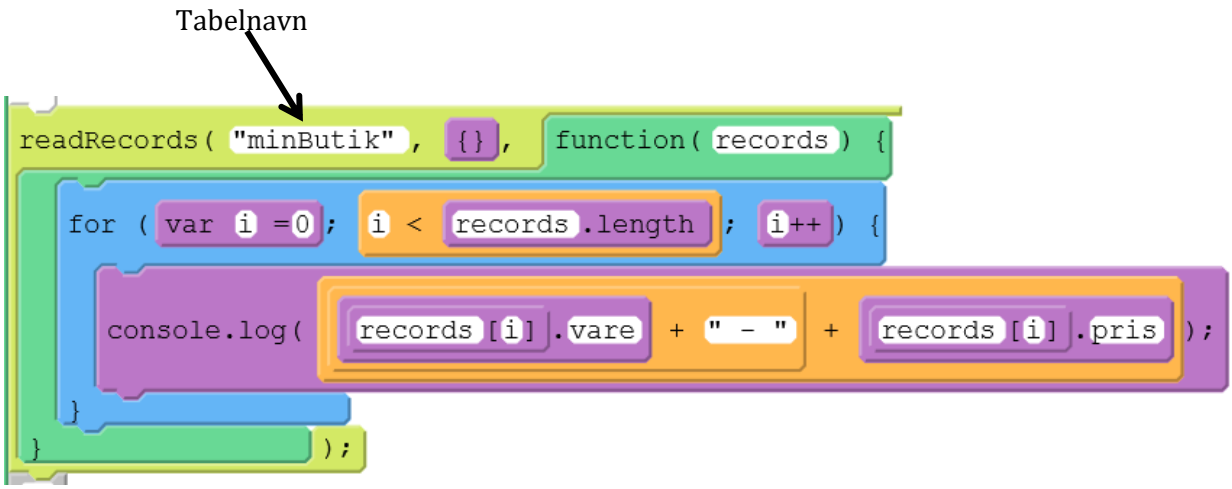
Se også videoen "Sådan gemmes i datalaget".

15. Læse alle data fra datalaget

Lad os sige at vi har nedenstående tabel (kaldet for "`minButik`") gemt i datalaget:

id	vare	pris	farve
#	enter text	enter text	enter text
1	"nederdel"	300	"sort"
2	"busker"	450	"blå"
3	"bluse"	200	"hvid"
4	"t-shirt"	100	"sort"

Hvis vi gerne vil læse alle data fra tabellen gøres det vha. kommandoen `readRecords()`:



Det ser en anelse kompliceret ud, så vi tager det lidt ad gangen. Det første, vi bemærker, er, at lige efter `readRecords()` skrives navnet på tabellen (her "`minButik`"). Når man vha. `readRecords()` læser fra en tabel, bliver alle tabellens rækker lagt over i et array, som kaldes for `records`. Den første række kan man således få fat i ved at skrive `records[0]` (husk vi altid starter ved 0 og ikke 1), den anden række fås ved at skrive `records[1]` osv. Se nedenstående figur:

id	vare	pris	farve	
#	enter text	enter text	enter text	
1	"nederdel"	300	"sort"	← records[0]
2	"busker"	450	"blå"	← records[1]
3	"bluse"	200	"hvid"	← records[2]
4	"t-shirt"	100	"sort"	← records[3]

Her er elementerne i `records` arrayet faktisk objekter, som repræsenterer de forskellige rækker. Det vil sige, at hvis vi nu f.eks. gerne vil have fat i det, der står i kolonnen `pris` i den tredje række, så skriver vi blot

```
records[2].pris
```

Se flere eksempler i figuren herunder:

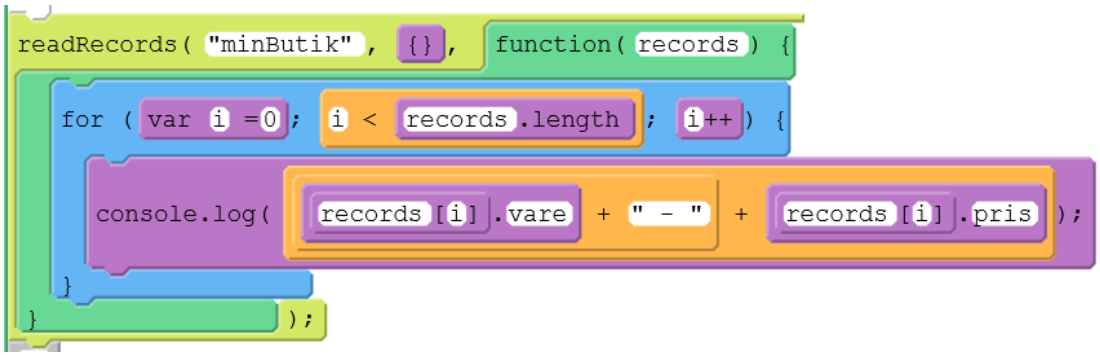
id	vare	pris	farve
#	enter text	enter text	enter text
1	"nederdel"	300	"sort"
2	"busker"	450	"blå"
3	"bluse"	200	"hvid"
4	"t-shirt"	100	"sort"

`records[0].vare`

`records[1].farve`

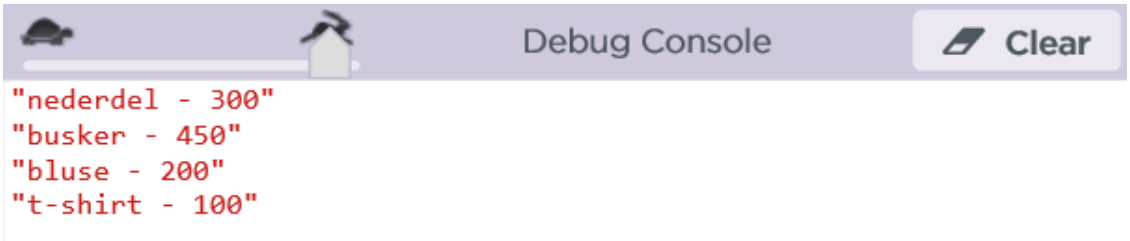
`records[3].pris`

Går vi tilbage til den oprindelige `readRecords()`-kommando:



Så kan vi nu se, at `for`-løkken gennemløber alle rækkerne i tabellen og for hver række udskrives det, der står i kolonnen `vare` og `pris` i konsollen (adskilt af en bindestreg).

I konsollen kommer der derfor til at stå:



Se også videoen "Sådan hentes fra datalaget".

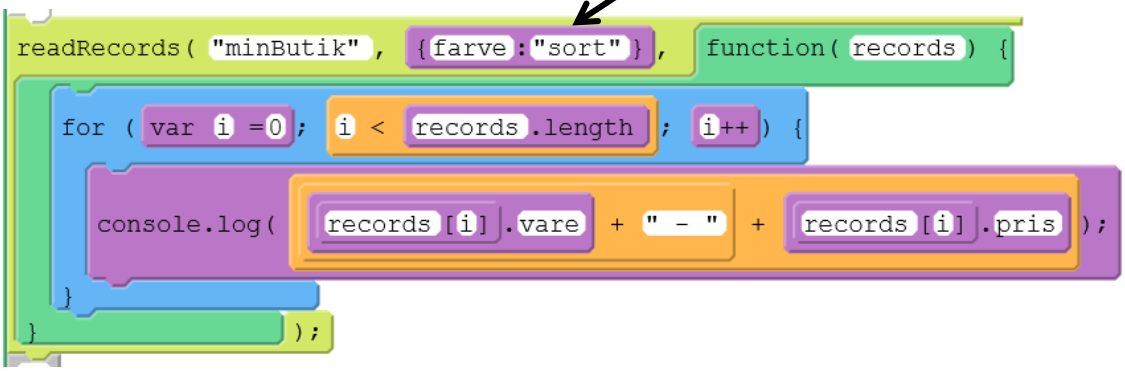
16. Læse udvalgte data fra datalaget

Vi går videre med eksemplet fra forrige afsnit og tænker nu, at vi gerne vil have fat i alle de stykker tøj, hvor farven er sort:

id	vare	pris	farve
#	enter text	enter text	enter text
1	"nederdel"	300	"sort"
2	"busker"	450	"blå"
3	"bluse"	200	"hvid"
4	"t-shirt"	100	"sort"

Det gøres næsten som før:

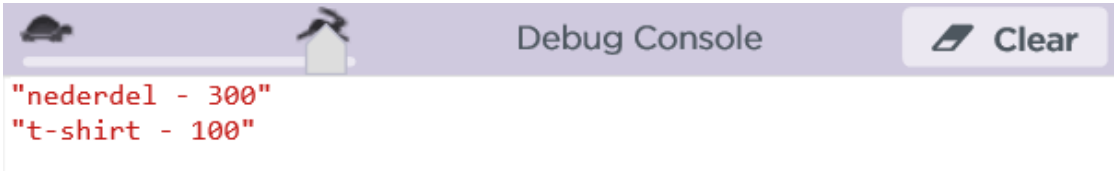
Betingelse som angiver, at vi kun ønsker at læse de rækker, hvor kolonnen `farve` har værdien `"sort"`



Da der kun er to rækker, hvor kolonnen `farve` har værdien `"sort"`, bliver `records` nu et array med to elementer:

id	vare	pris	farve	
#	enter text	enter text	enter text	
1	"nederdel"	300	"sort"	← `records[0]`
2				
3				
4	"t-shirt"	100	"sort"	← `records[1]`

Igen er elementerne i `records` arrayet objekter, som repræsenterer de rækker, der opfylder den angivne betingelse. Derfor bliver resultatet af den ovenstående `readRecord()`-kommando følgende:



Se også videoen "Sådan hentes fra datalaget".

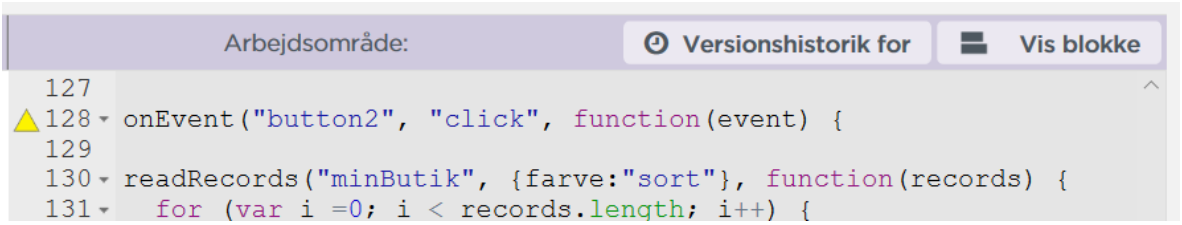
17. Yderligere hjælp at hente

Når man har valgt en kommando i værktøjskassen i App Lab, kan man holde musen hen over kommandoen, og der dukker en blå boks op, hvor det er muligt at trykke på "See examples". Her står lidt generelt om, hvordan kommandoen fungerer, og der er vist nogle eksempler. Det er ofte en stor hjælp.

Desuden er navnene på kommandoerne ofte ret sigende, så det er tit en hjælp bare at sidde og bladre lidt i de forskellige værktøjskasser.

Når man programmerer i App Lab, programmerer man i virkeligheden i et programmeringssprog, som hedder Javascript. I stedet for at se kommandoerne som blokke, kan man i stedet se selve teksten ved at vælge "</> Vis tekst" i øverste højre hjørne i arbejdsområdet:

Trykker man her får man:



Nogle gange er det nemmere at redigere her, men man skal også være varsom, da man kan komme til at lave (syntaks-)fejl, som gør, at man ikke kan komme tilbage til blokkene.

En anden fordel er, at man her kan indsætte Javascript kode, der ikke på forhånd er lavet blokke til.

Mere generel hjælp til Javascript kan findes på w3schools:

https://www.w3schools.com/js/default.asp

Hvis man f.eks. har brug for lidt mere matematik (udregning af potenser eller kvadratrødder), så kan man få hjælp her:

https://www.w3schools.com/js/js_math.asp