MINECRAFT

Eksploderende kyllinger!

(I gang med progammering)

Erik Mejer Hansen

Installation

Inden vi kan komme i gang skal vi have gjort din computer klar.
Det kan godt være at du skal have hjælp af en voksen til det. Men bare rolig, resten af bogen er meget nemmere.

Vi skal have:

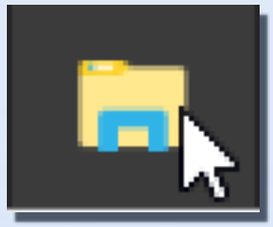
- Installeret Java
- Installeret en Minecraft-server
- Lavet en ny “Start indstilling” i Minecraft

Kopier filerne fra USB'en

Inden vi kan komme i gang skal vi have kopieret nogle filer til din computer.

Sæt USB'en i din computer.

Vi skal bruge "`Stifinder`" til at kopiere filerne.

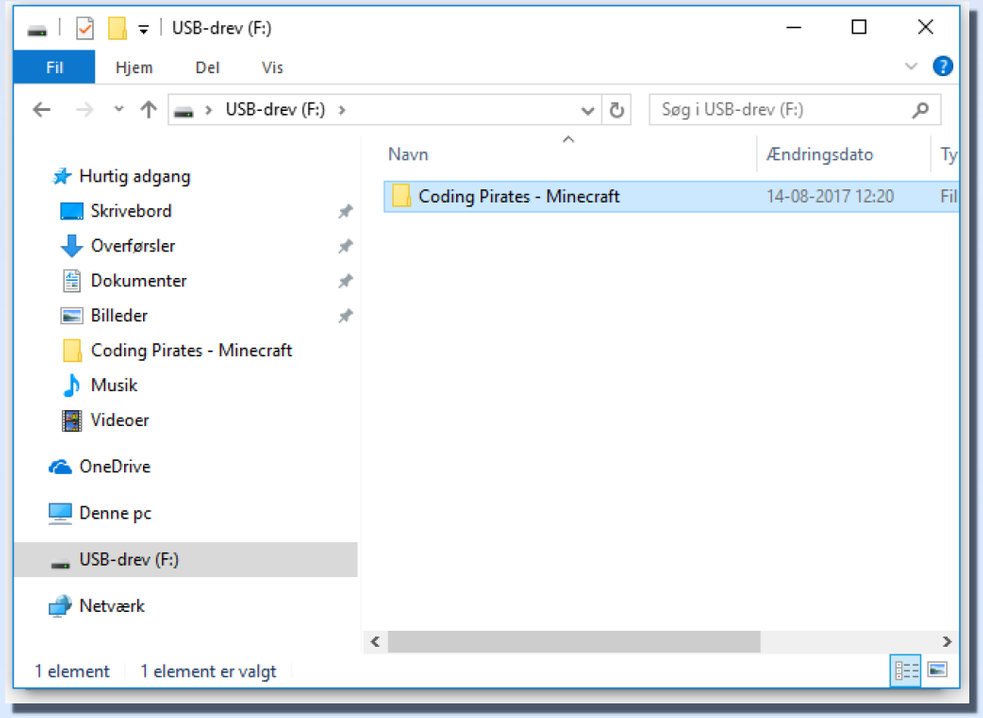


Du starter "`Stifinder`" ved at trykke på billedet af den gule mappe.

Du finder det nederst på din skærm.

Når du har åbnet "`Stifinder`" skal du finde USB'en og klikke på den.

Til sidst skal du trykke på mappen "`Coding Pirates - Minecraft`" og trække den ud på "`Skrivebordet`".

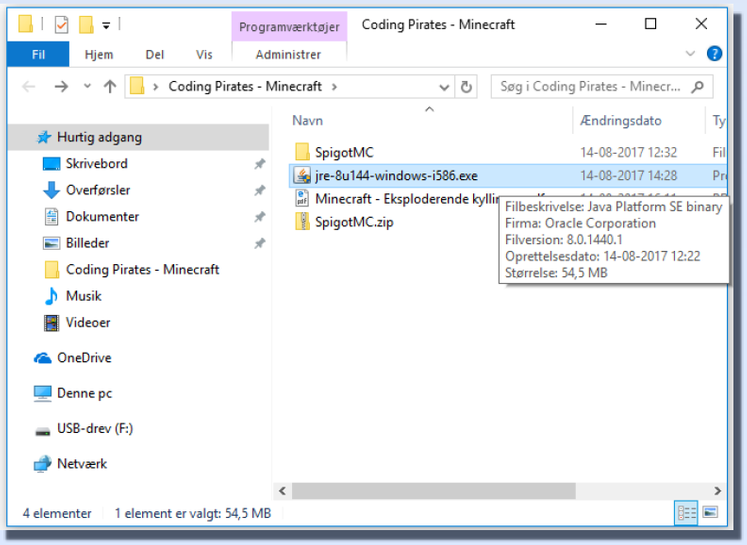


Installer Java

Vi skal have installeret et program der hedder "`Java`".

Det ligger i den mappe du lige har kopieret ud på "`Skrivebordet`".

Du skal dobbelt-klikke på filen "`jre-8u144-windows-i586.exe`".



Nu åbner der et nyt vindue.

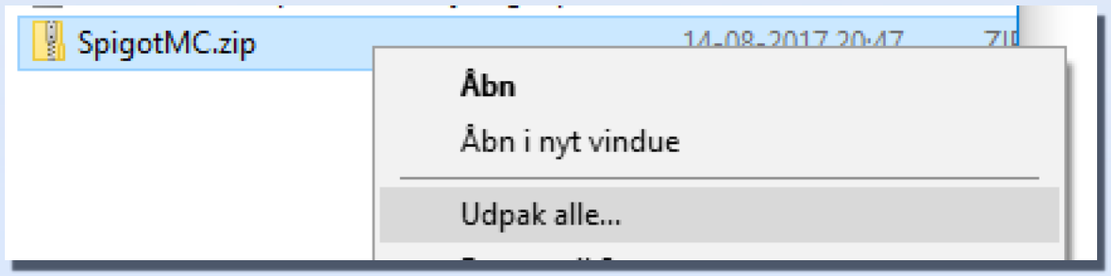
Tryk på knappen "`Install >`" og vent lidt.

Installer Minecraft server

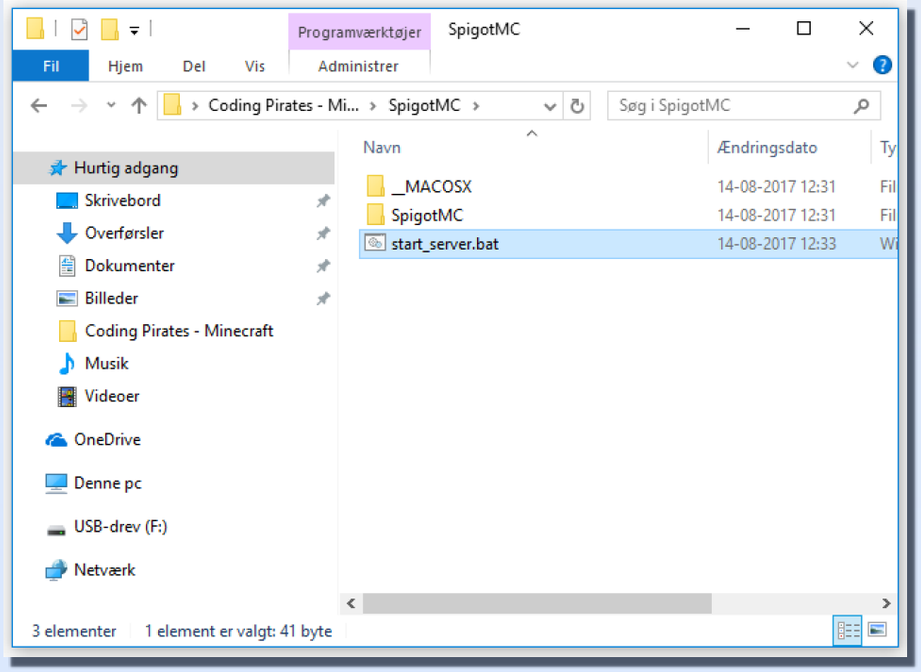
Vi skal bruge vores egen Minecraft server.

Find den mappe du har kopieret ud på skrivebordet.

Højre-klik på filen "`SpigotMC.zip`" og klik så på "`Udpak alle...`".



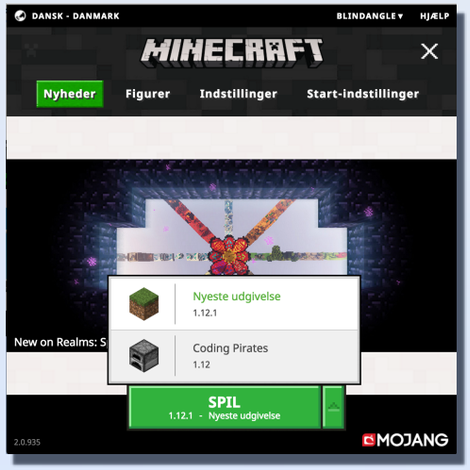
Du kan nu starte din helt egen Minecraft server ved at dobbelt-klikke på "`start_server.bat`".



Start den bare nu. Vi skal bruge den om et øjeblik.

Start Minecraft

Nu er vi næsten klar til at starte Minecraft.



Klik først på "`Start-indstillinger`".

Klik på "`Tilføj ny`".

Skriv "`Coding Pirates`" under navn og vælg "`release 1.12`" under version. Tryk "`Gem`".

Tryk på Minecraft-logoet for oven.

Tryk på den lille pil ved "Spil"-knappen og tryk så på "Coding Pirates"

Nu kan du endelig starte Minecraft!

Multiplayer

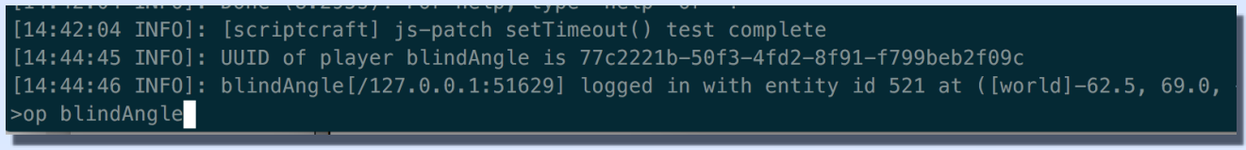
Nu skal vi have logget dig på din Minecraft server.

Tryk på "`Multiplayer`".

Tryk på "`Forbind direkte`".

Skriv "`localhost`" under "`Serveradresse`" og tryk på "`Deltag i serveren`".

Skift over til vinduet der åbnede da du startede din Minecraft server . Der skal du nu skrive "`op `" og så dit Minecraft-navn. Du kan se dit navn i serverens vindue hvis du ikke kan huske det.



Her står der "`blindAngle`" så der skrives "`op blindAngle`"

Kommandoer

Du plejer sikkert at bygge, grave, kæmpe og slås i Minecraft.

Men har du nogensinde tryllet en gris eller en regnbue frem? Har du forvandlet dag til nat og fremkaldt tordenvejr?

Det kan man alt sammen i Minecraft med kommandoer.

Og det er dem du skal lære at bruge nu.

Minecraft-konsollen

Kigger man i ordbogen står der at "kommando" betyder "en kort ordre til at gøre noget bestemt". Og det er faktisk lige præcis det vi skal. Vi skal sige til Minecraft at den skal gøre noget bestemt. Som for eksempel at trylle en gris frem.

I Minecraft er kommandoer noget man skriver og man skriver dem i Minecraft-konsollen.

Åbn Minecraft-konsollen

Inden du kan bruge Minecraft-konsollen skal du først have åbnet den. Tryk på "`T`". Nu dukker der et lille felt op hvor du kan skrive tekst.

Hvis du skriver en tekst og trykker på "`Enter`"-tasten så sender du en chat-besked.

Prøv at åbne Minecraft-konsollen igen, men denne gang skal du starte med at skrive "`/summon pig`" og tryk på "`Enter`"-tasten.

Og så dukkede der en gris frem (det kan være du skal dreje dig for at se den)! Du har lige givet din første kommando. Prøv et par gange mere så du har en hel flok grise.

Du skriver "`/`" ved at holde "`Shift`"-tasten inde og trykke på "`7`".

På næste side skal vi til de første eksperimenter. Et eksperiment er lidt ligesom en opgave du skal løse. Men det er ikke sikkert at eksperimentet har et rigtigt svar. Prøv først at gøre som teksten siger. Prøv så at lave noget om og læg mærke til hvad der ændrer sig. Det kalder vi at "eksperimentere med tingene".

At prøve sig frem og se hvad der sker en rigtig god måde at finde ud af hvordan noget virker.

Hvis der er noget du syntes er svært eller ikke kan få til at virke - så kald på en voksen!

Eksperimenter

Eksperiment #1

Åben Minecraft-konsollen og skriv "`/time set 0`".

- Hvad skete der?
- Hvad sker der hvis du skriver 1000 i stedet for 0?
- Kan du få det til at blive nat?
- Og dag?

Eksperiment #2

Åben Minecraft-konsollen og skriv "`/weather thunder`"

- Hvad skete der?
- Hvad sker der hvis du skriver "`/weather clear`"?

Eksperiment #3

Hvad gør kommandoen: "`/js rainbow()`"?

Eksperiment #4

Prøv at skrive: "`/js box(blocks.wood, 5, 5, 5)`"

Hvad tror du tallene du skrev betyder? Prøv at lave om på dem.

Hvad tror du "`blocks.wood`" gør?

Hint: Kig bagerst i bogen og prøv at skrive en af de andre ting i stedet for "`blocks.wood`".

Dit første program

Ved du hvad et program er? Et program er en måde at fortælle computeren hvad den skal gøre.

Kan du huske hvad en kommando er? Det var "en ordre om at gøre noget".

Kan du se at de to ting ligner hinanden?

Et program er faktisk bare en række af kommandoer som vi beder computeren om at lave. Og nu skal du til at skrive dine første programmer.

Bygge-robotten

De første programmer vi skriver skal bygge ting og til det skal vi bruge vores usynlige ven Bygge-robotten. Vores programmer vil fortælle Bygge-robotten hvor den skal bygge blokke og hvilken slags blokke det skal være.

Vi kan ikke bare fortælle Bygge-robotten at den skal: "Bygge et slot med voldgrav og vindebro". Bygge-robotten forstår kun nogle bestemte kommandoer.

Bygge-robottens kommandoer

Bygge-Robotten har to grupper af kommandoer. Kommandoer der flytter robotten og kommandoer der bygger blokke.

Flyt Bygge-robotten rundt

`fwd()` : Flytter robotten ét felt frem.

`back()` : Flytter robotten ét felt tilbage.

`left()` : Flytter robotten ét felt til venstre.

`right()` : Flytter robotten ét felt til højre.

Byg med Bygge-robotten

`box(blocks.stone)` : Bygger en sten blok

`box(blocks.diamond)` : Bygger en diamant blok.

Hvis du gerne vil bygge med noget andet end sten eller diamant så kig på mulighederne bagerst i bogen.

Brug Minecraft-konsollen

Vi skriver vores første programmer i Minecraft-konsollen. Den er et godt sted at afprøve ting i fordi vi kan se resultatet med det samme. Når vi vil skrive et program i Minecraft-konsollen skal vi *altid* starte med "`/js `" og så skrive vores program bagefter. Lad os prøve os lidt frem.

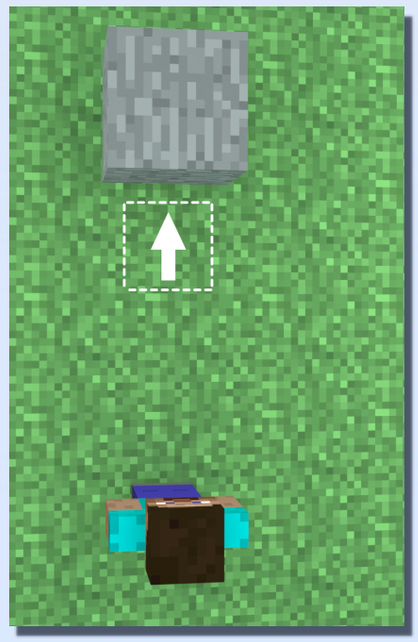
Byg en sten-blok

Lad os starte med bare at placere en sten-blok. Hvis vi kigger på kommandoerne for at bygge med Bygge-robotten kan vi se at "`box(blocks.stone)`" bygger en sten-blok. Lad os prøve at skrive "`/js box(block.stone)`" i Minecraft-konsollen og se hvad der sker.

Flyt et felt frem og byg en blok

Nu ved vi hvordan vi bygger en blok. Kommandoen "`fwd()`" flytter Bygge-Robotten et felt frem (men husk, bygge robotten er usynlig så vi kan ikke se det).

"`fwd()`" flytter robotten frem og "`box()`" bygger en kasse, men hvordan fortæller vi Bygge-robotten at den først skal gå frem og *bagefter* bygge en blok?



Vi gør det ved at sætte kommandoerne sammen! Så vi skriver "`/js fwd().box(blocks.stone)`".

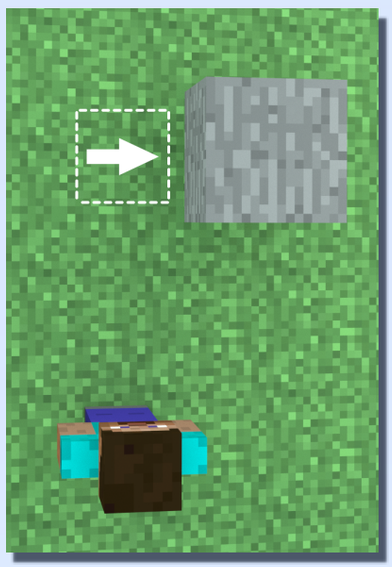
Læg mærke til at der er et "." mellem de to kommandoer. Det er vigtigt ellers forstår Bygge-robotten ikke hvad vi siger.

Flyt et felt til højre og byg en blok

Vi kan flytte Bygge-robotten et felt til højre med "`right()`". Så hvad skal vi skrive hvis vi vil flytte Bygge-robotten et felt til højre og bygge en sten blok?

Det er rigtigt! Vi skal skrive "`/js right().box(blocks.stone)`".

Husk at der skal være et "." mellem hver kommando.

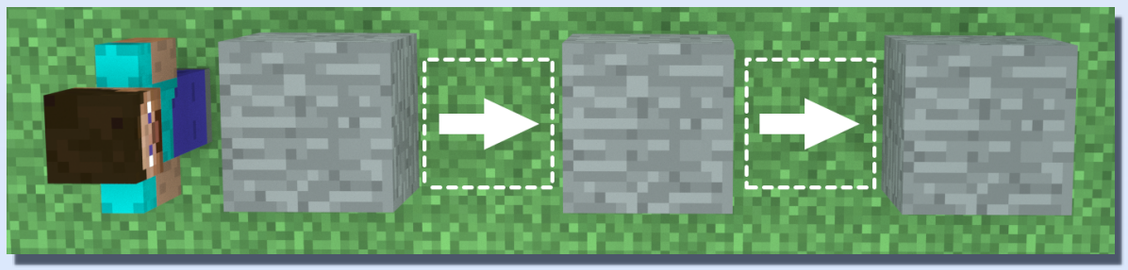


Program: En række af sten

Nu ved vi lidt om hvordan vi styrer vores robothjælper, så lad os prøve at lave vores første program.

Programmet skal placere 3 sten med mellemrum i mellem.

Vi bygger en blok med "`box`" og bagerst i bogen kan



vi se at sten er "`blocks.stone`". Så vi kan altså bygge en sten-blok med "`box(blocks.stone)`".

Åben Minecraft og prøv det. Husk at du skal skrive "`/js`" inden dit program i Minecraft-konsollen. Du skal altså skrive "`/js box(blocks.stone)`" for at bygge en sten-blok.

Vi kan flytte robothjælperen et felt frem ved at skrive "`fwd()`".

Lad os prøve at sætte de to dele sammen til: "`/js box(blocks.stone).fwd().box(blocks.stone)`". Prøv det i Minecraft og se hvad der sker.

Vi er der næsten! Der mangler bare én enkelt sten mere.

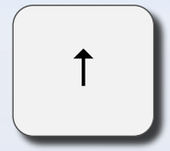
Kan du selv udvide programmet så der bliver bygget tre sten-blokke?

Minecraft-konsollen kan huske

Prøv at skrive et lille program i Minecraft-konsollen og tryk på "Enter"-tasten så det bliver kørt (Du kan for eksempel skrive "`/js fwd().box(blocks.stone)`").

Åbn nu Minecraft-konsollen igen og tryk på pil op ("↑"). Nu kan du se det program du skrev lige før. Du kan ændre lidt i teksten og trykke på "Enter" igen.

Prøv for eksempel at ændre "`blocks.stone`" til "`blocks.diamond`" og tryk på "Enter"-tasten.



Hvis du nu åbner Minecraft-konsollen igen og trykker på "↑" et par gange kan du se at du kan bladre igennem de ting du har skrevet før. Du kan også bruge "↓" til at bladre med.

Det gør det nemt og hurtigt at prøve sig frem.

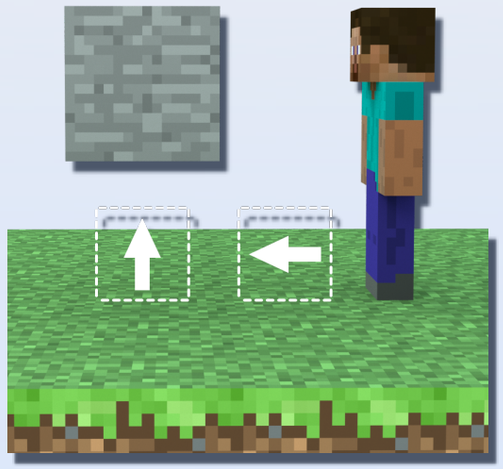
Byg opad

Indtil nu har vi bygget på jorden, men Bygge-robotten kan også bevæge sig op og ned.

`up()` : Flytter robotten ét felt op.

`down()` : Flytter robotten ét felt ned.

Så vi kan skrive "`/js fwd().up().box(blocks.stone)`" for at bygge opad.

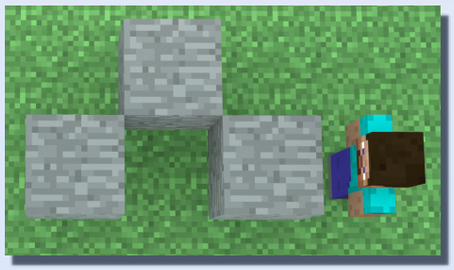


Nu er det tid til at lave nogle eksperimenter med Bygge-Robotten.

Eksperimenter

Eksperiment #1

Kan du lave et program der bygger det her mønster?



Du skal bruge kommandoerne:

`fwd()` : Som flytter Bygge-robotten ét felt frem.

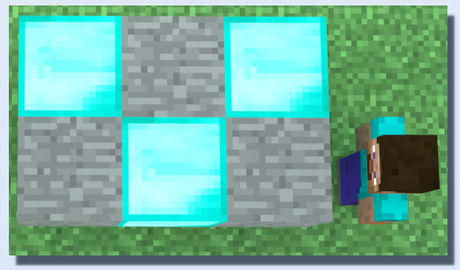
`left()` : Som flytter Bygge-robotten ét felt til venstre.

`right()` : Som flytter Bygge-robotten ét felt til højre.

`box(blocks.stone)` : Som bygger en sten-blok.

Eksperiment #2

Kan du lave et program der bygger det her mønster?



Du skal bruge kommandoerne:

`fwd()` : Som flytter Bygge-robotten ét felt frem.

`left()` : Som flytter Bygge-robotten ét felt til venstre.

`right()` : Som flytter Bygge-robotten ét felt til højre.

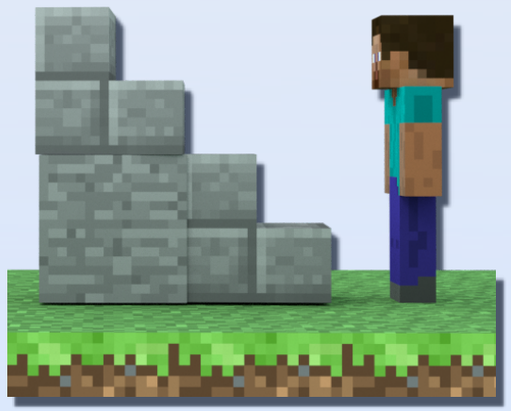
`box(blocks.stone)` : Som bygger en sten-blok.

`box(blocks.diamond)` : Som bygger en sten-blok.

Eksperimenter

Eksperiment #3

Kan du lave et program der bygger en trappe?



Du skal bruge kommandoerne:

`fwd()` : Som flytter Bygge-robotten ét felt frem.

`up()` : Som flytter Bygge-robotten ét felt op.

`box(blocks.stone)` : Som bygger en sten-blok.

`box(blocks.stairs.stone)` : Som bygger en sten-trappe.

Eksperiment #4

Kan du gætte hvad dette program bygger?

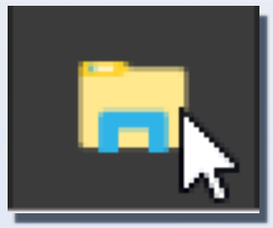
```
box(blocks.stone).fwd().box(blocks.stone).up().box(bl
ocks.stone)
```

Kode-editor

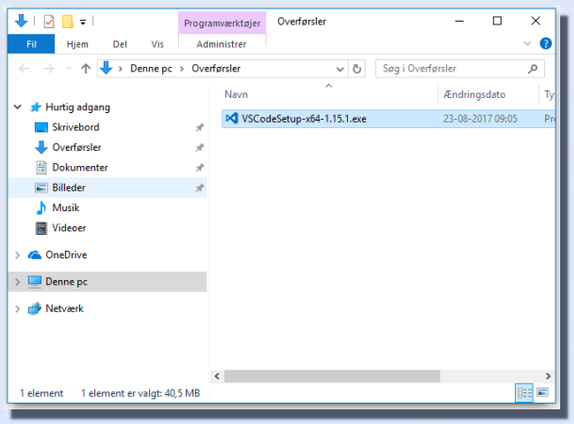
Indtil nu har du skrevet dine programmer direkte i Minecraft-konsollen. Minecraft-konsollen er rigtig god til at afprøve ting, men når man skal skrive lidt længere programmer så er det rart at gøre det i en fil.

Når man skriver programmer i en fil skal man helst bruge et specielt program. Vi installerer programmet nu og bruger det i næste kapitel.

Først skal du sætte USB'en i din computer og åbne "`Stifinder`".

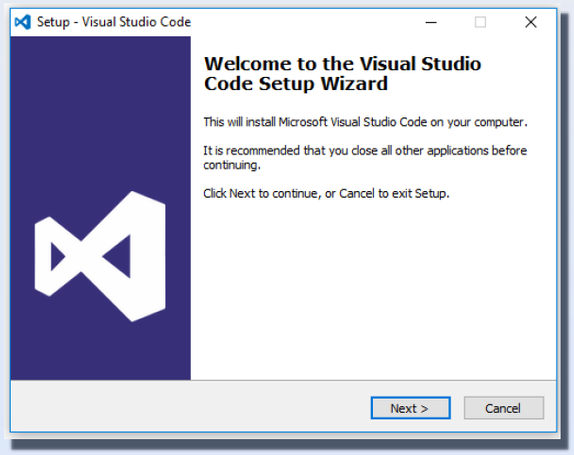


Dobbelt-klik på filen "`VSCodeSetup-x64-1.15.1.exe`".

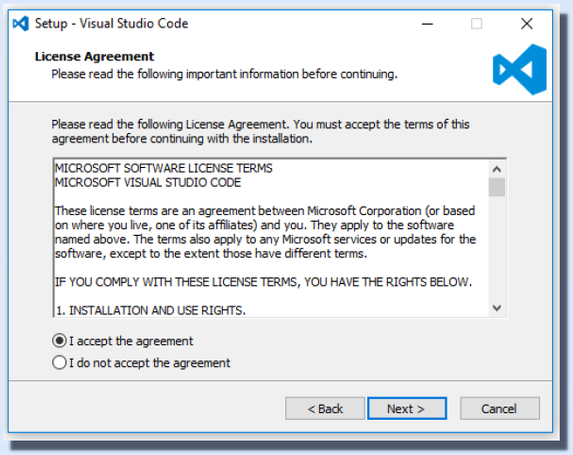


Tryk på "`Ja`"

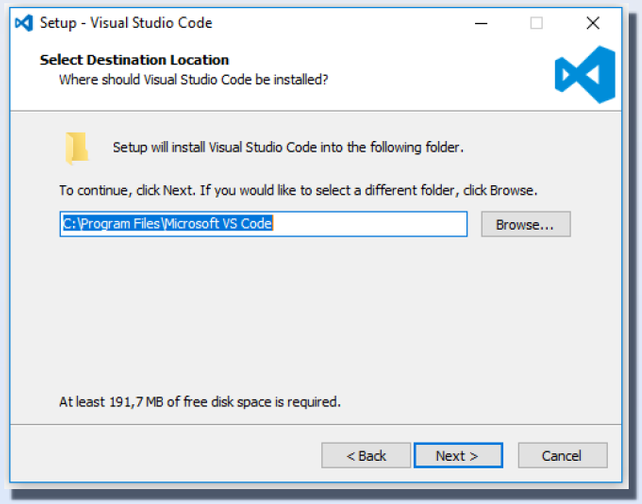
Tryk på "`Next >`"



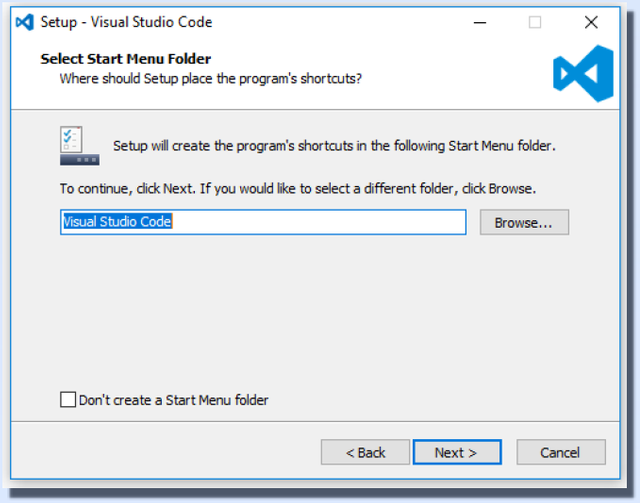
Vælg "`I accept the agreement`" og tryk på "`Next >`"



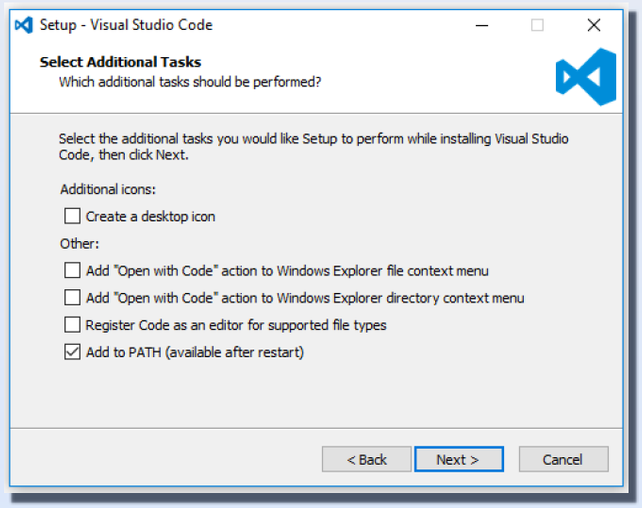
Tryk på "`Next >`"



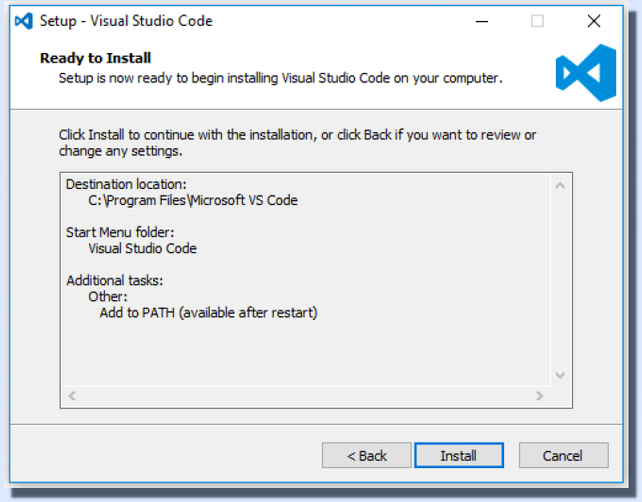
Tryk på "`Next >`"



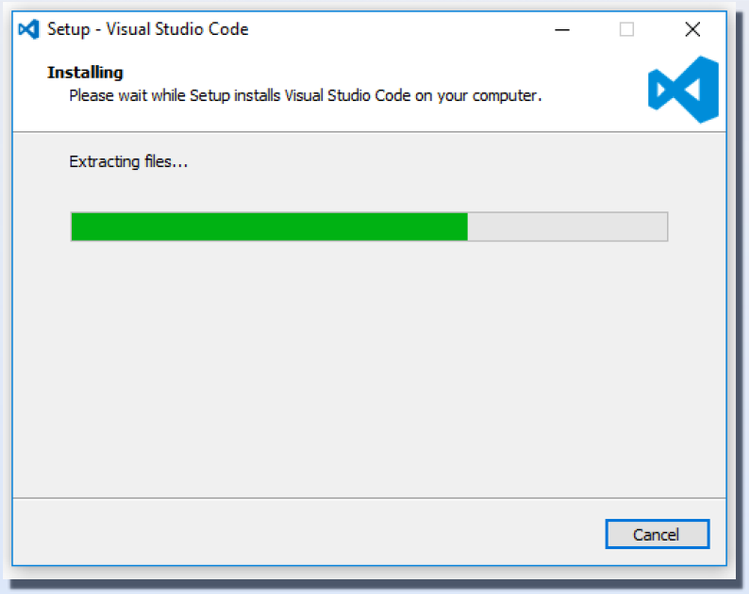
Vælg på "`Create a desktop icon`" og tryk på "`Next >`"



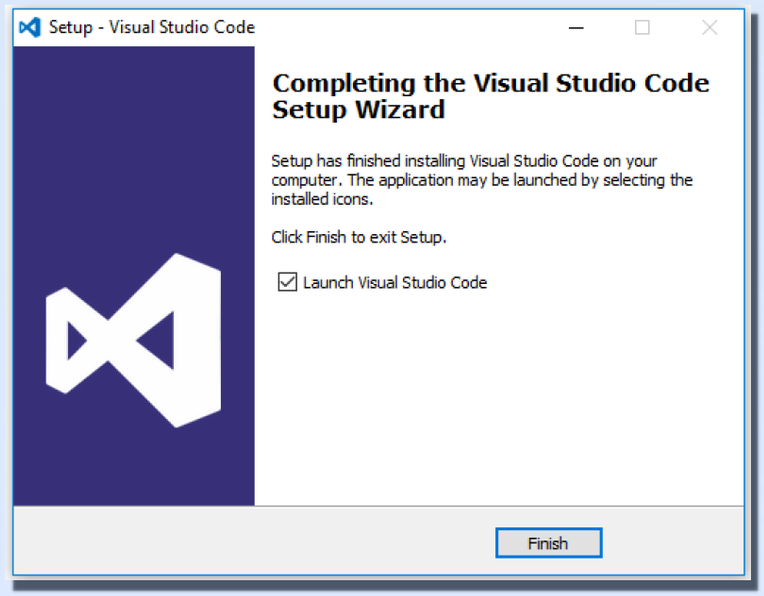
Tryk på "`Next >`"



Vent lidt...



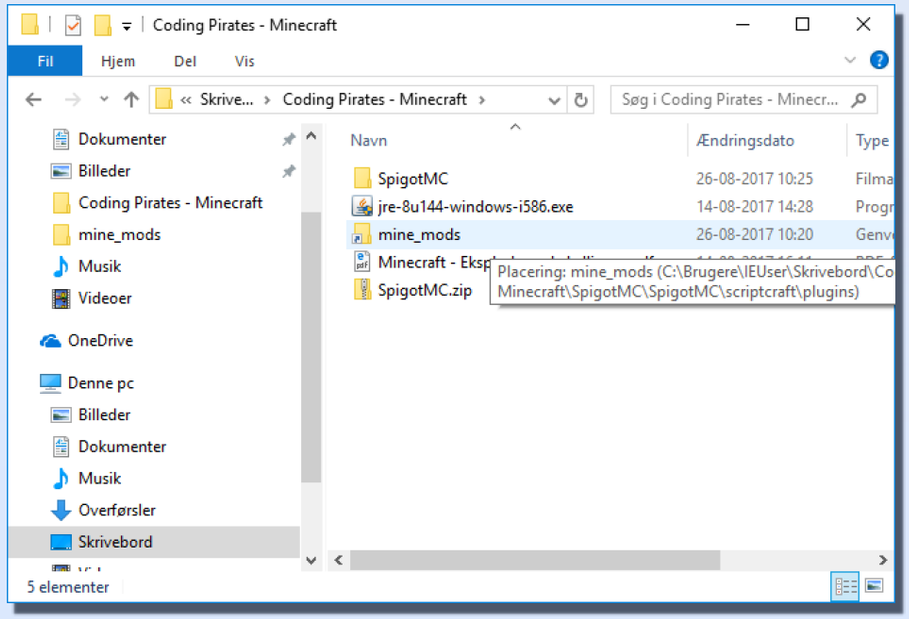
Tryk "`Finish`"



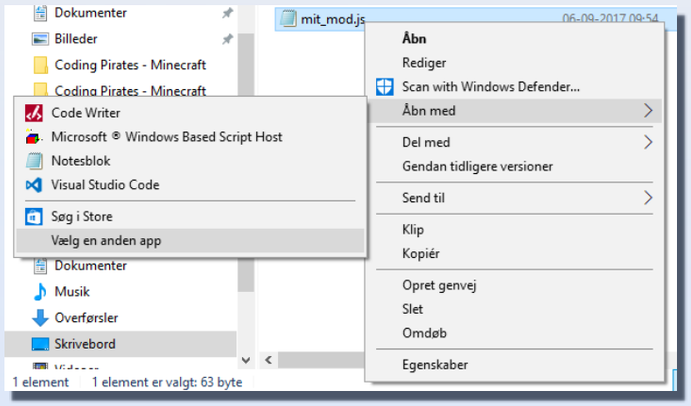
Åbn en kode-fil

Nu mangler vi bare én ting: Vi skal have vores kode-editor til at åbne vores kode filer.

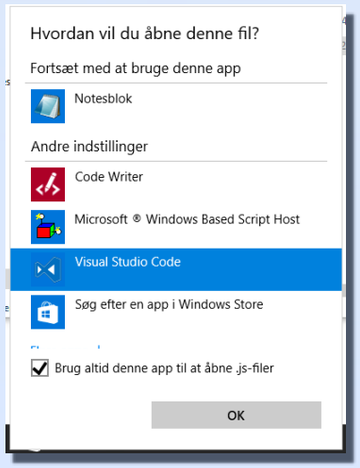
Åbn "`Coding Pirates`"-mappen som ligger på dit skrivebord. Dobbelt-klik på mappen"`mine_mods`".



Nu skal du højre-klikke på "`mit_mod.js`" og vælg "`Åbn med > Vælg en anden app`".

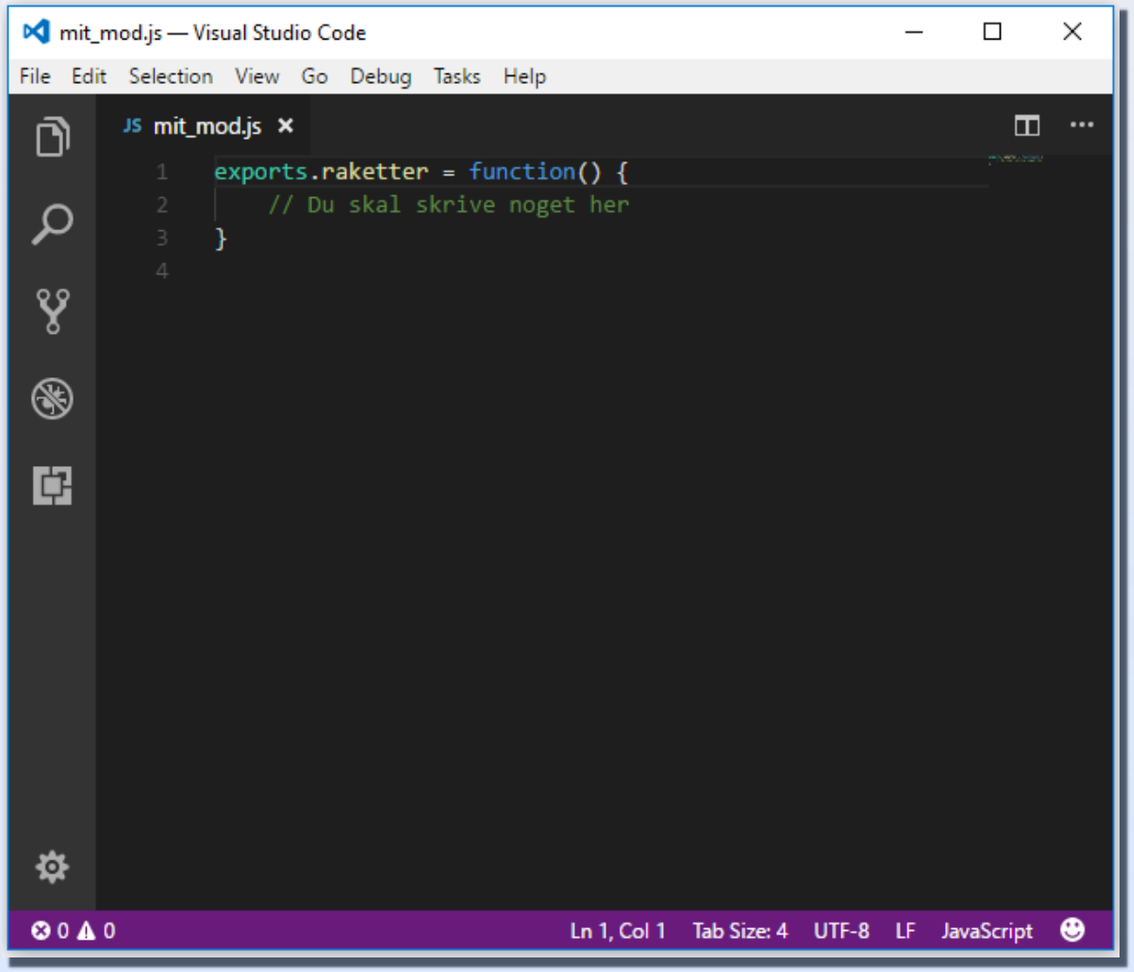


Nu skal du finde "Visual Studio Code" og sætte flue-ben i "Brug altid denne app til at åbne .js-filer".



Tryk på "OK"

Nu åbner filen i vores kode editor.



Næste gang du skal åbne en fil i "`mine_mods`" mappen skal du bare dobbelt-klikke på den.

Dit første mod

Nu har du fået en kode-editor der er rigtig god til at skrive programmer i.

I dette kapitel skal vi skrive vores første lille mod.

Skriv - Afprøv - Ret - Udvid

Ind til nu har vi skrevet vores progammer i Minecraft-konsollen. Nu har har vi en kode-editor vi kan skrive programmerne i. Og nu skal vi til at skrive vores første mod.

Når vi skriver programmer i en kode-editor skal vi også til at arbejde på en ny måde.

Skriv

Først skriver vi en lille del af vores program i kode-editoren. Vi skriver kun en lille del ad gangen. Bare en enkelt kommando eller to.

Afprøv

Når vi har skrevet lidt af vores program så afprøver vi det i Minecraft. Vi ser efter om programmet gør det vi gerne vil have det til.

Ret

Du vil tit opleve at dit program ikke helt gør det du havde håbet på det ville. Det kan være at Bygge-robotten bygger en blok et forkert sted eller at den bruger en forkert slags blok.

Så går vi tilbage til kode-editoren og prøver at rette det. Bagefter afprøver vi vores program igen.

Udvid

Når vi syntes at vores program gør som det skal kan vi udvide det. Vi tilføjer igen kun en enkelt eller to kommandoer og så afprøver vi det igen.

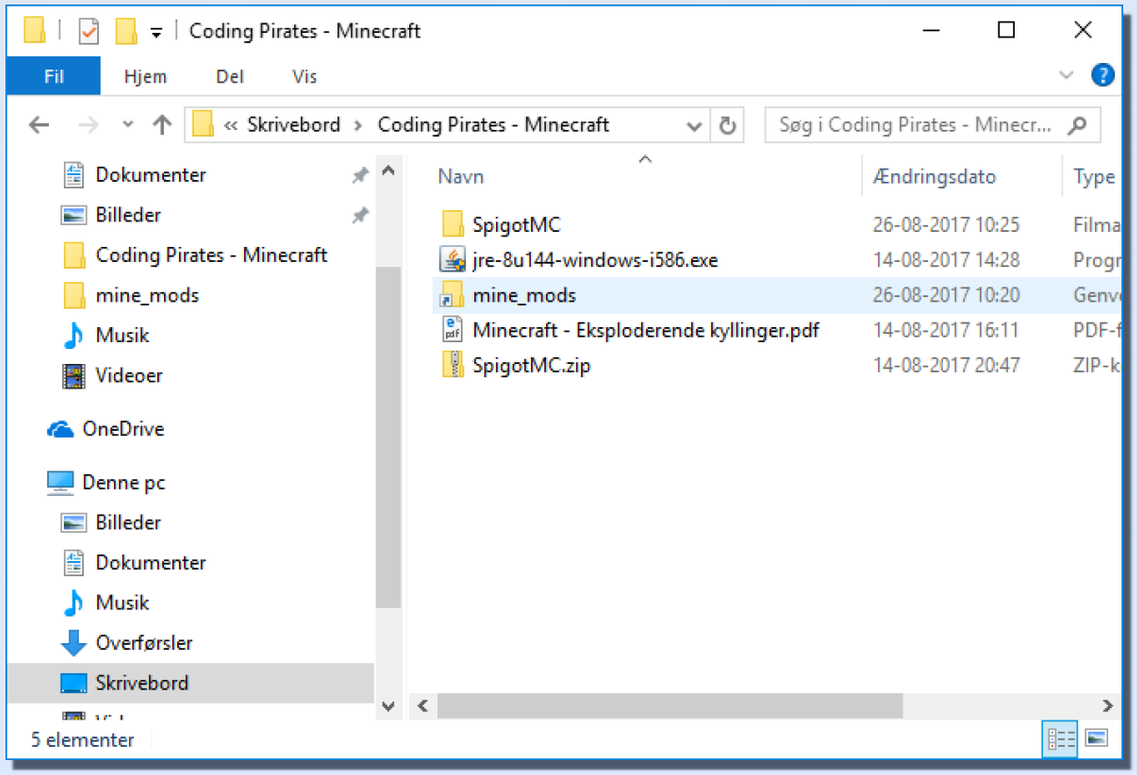
Det er meget nemmere at få dit program til at virke hvis du husker at afprøve det hver gang du har tilføjet en lidt til det.

Åbn din første kode fil

Allerførst skal vi have åbnet den fil vi gerne vil skrive vores program i.

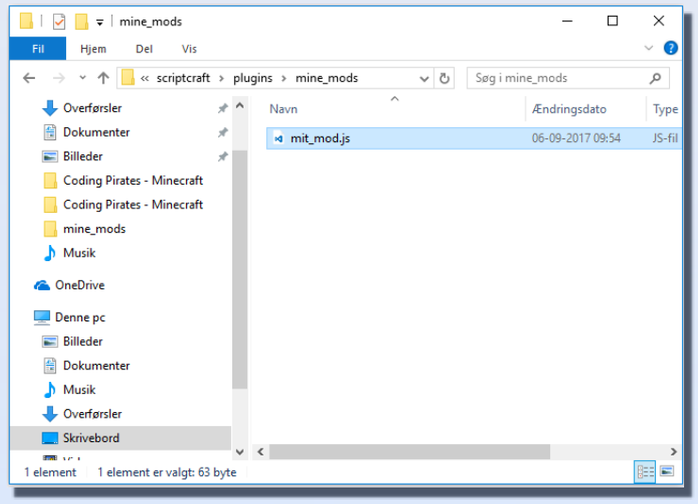
Dobbelt-klik på mappen "`Coding Pirates`" som ligger på dit skrivebord.

Dobbelt klik på "`mine_mods`".

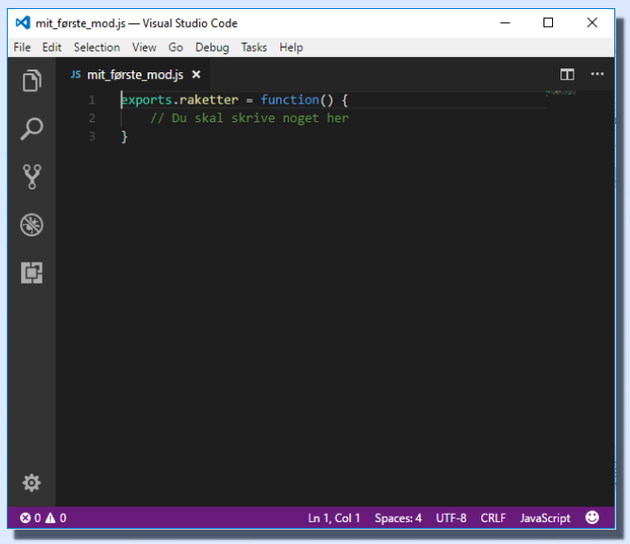


Alle de mods du skriver kommer til at ligge i mappen "`mine_mods`".

Dobbelt-klik nu på "`mit_mod.js`"



Nu er vi klar til at skrive vores første mod!

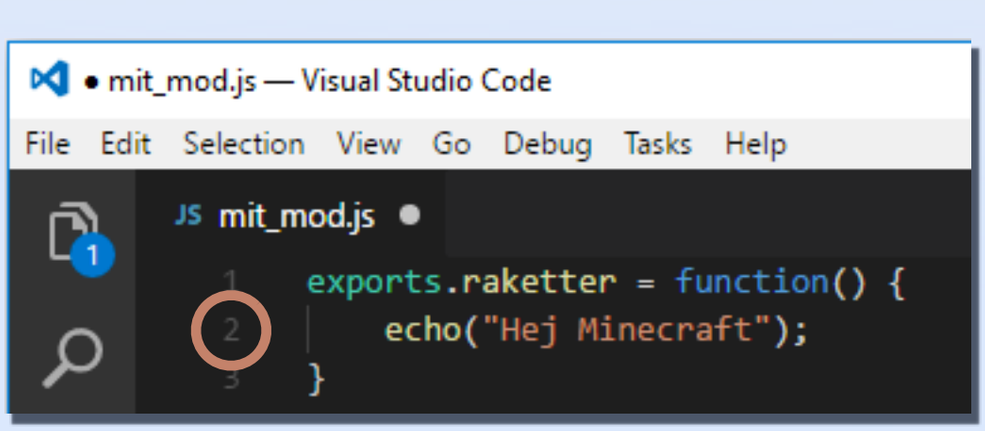


Lær at bruge din kode-editor

Det kan tage lidt tid at lære din kode-editor rigtigt at kende, men hæng i så skal du nok få det lært.

Start med at lave en lille ændring i filen. Prøv at skrive "`echo("Hej Minecraft");`" på linje 2.

Du kan se linje-numrene ude i venstre side af din kode-editor (der er sat en ring omkring to-tallet i billedet nedenunder):

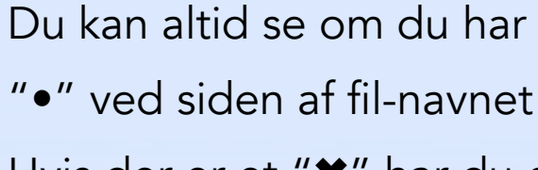


Gem din fil

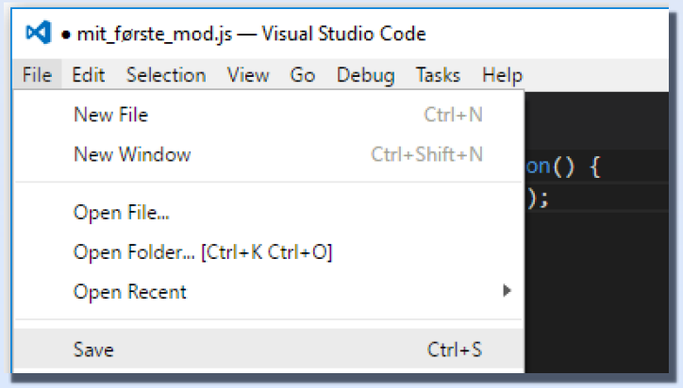
Så er vi igang med at skrive vores første mod. Men vi skal huske at gemme filen.

Du kan gemme på to måder:

Du kan holde "Ctrl" knappen nede og trykke på "S".



Eller du kan vælge "File" og bagefter "Save" fra menu-linjen.



Du kan altid se om du har gemt din fil. Hvis der er en "•" ved siden af fil-navnet er filen ikke blevet gemt. Hvis der er et "✖" har du gemt filen.

Afprøv dit program

Nu er det tid til at få afprøvet vores program. Hvis du ikke allerede har gjort det så start din Minecraft-server og Minecraft.

Nu kan du åbne Minecraft-konsollen og skrive "`/js raketter()`". Du skulle nu se beskeden "Hej Minecraft".

Genopfrisk Minecraft

Hvis du allerede havde Minecraft åben da du skrev "`echo("Hej Minecraft");`" i kode-editoren så skal du åbne Minecraft-konsollen og skrive "`/js refresh()`".

Minecraft opdager nemlig ikke selv at du har lavet ændringer i dit program. Men skriver du kommandoen "`/js refresh()`" i Minecraft-konsollen opdaterer den dit nye progam.

Husk at køre "`/js refresh()`" i Minecraft-konsollen hver gang du har gemt din fil i kode-editoren og du gerne vil afprøve dit nye program.

Editoren hjælper dig

Vi bruger en kode-editor fordi den gør det nemmere at skrive kode. Prøv at holde øje med om der dukker røde streger op. Det er kode-editorens måde at fortælle dig at der er en fejl.

```
exports.raketter = function() {
    echo(Hej Minecraft!");
}
```

```
exports.raketter = function() {
    echo("Hej Minecraft!");
}
```

Her er fejlen at der mangler et "-tegn. Skriver vi det igen forsvinder den røde streg.

Skriv et mod

Nu har du lært lidt om din kode-editor. Nu er skal vi igang med at skrive vores første mod.
Vores første lille mod skal være en kommando der:

- Skriver en besked på skærmen
- Bygger nogle blokke
- Skyder fyrværkeri af

Når vi skriver programmer i en kode-editor har vi mulighed for at skrive lidt noter til os selv. Hvis vi starter en linje med "//" så tæller den linje ikke med i programmet.

Det kan være en god ide at starte med at skrive nogle noter om hvad vores program skal kunne.

Vi starter med:

```
exports.raketter = function() {
    //Skriv en besked på skærmen

    //Byg nogle blokke

    //Fyrværkeri!
 }
```

Så glemmer vi ikke hvad det var vores program skulle kunne.

Skriv en besked på skærmen

Først skal vi have skrevet en besked på skærmen. Det kan vi gøre med "`echo`". Vi kan starte med at skrive "`echo("Hej Minecraft verden");`" så vores program kommer til at se sådan her ud:

```
exports.raketter = function() {
      //Skriv en besked på skærmen
      echo("Hej Minecraft verden");
      //Byg nogle blokke

      //Fyrværkeri!
 }
```

Vi har ikke set "`echo()`" før. "`echo()`" får Minecraft til at skrive en besked på skærmen.

Afprøv

Nu har vi skrevet den første kommando i vores program. Så nu er det tid til at afprøve vores program.

Prøv at starte Minecraft og afprøv din nye kommando (hvis du allerede havde startet Minecraft, så husk at du skal skrive "`/js refresh()`" først).

Du kan afprøve din kommando ved at skrive "`/js raketter()`".

Når du har fået dit program til at skrive en besked på skærmen kan vi gå videre til næste del af programmet.

Husk: *Skriv - Afprøv - Ret - Udvid*.

Udvid: Byg nogle blokke

Vi har fået skrevet noget tekst og nu er første del af vores program færdigt. Nu er det tid til at bygge nogle blokke.

Start med at udvide dit program så det sådan her ud:

```
exports.raketter = function() {
      //Skriv en besked på skærmen
      echo("Hej Minecraft verden");

      //Byg nogle blokke
      var drone = new Drone(self);

      //Fyrværkeri!
 }
```

Bygge-robotten igen

Lad os kigge lidt nærmere på den kode vi lige har skrevet. Kan du huske Bygge-robotten vi byggede med tidligere? Det er faktisk bare den vi bruger igen.

Når vi skal bruge Bygge-robotten i et program ser det lidt anderledes ud end når vi bruger den i Minecraft-konsollen.

Først skal vi have fat i en Bygge-robot.
Når vi skriver "`var drone = new Drone(self);`" siger vi "Vi vil have en ny Bygge-Robot og den skal hedde `drone`".

Hvis den skulle have heddet "robot" skulle vi have skrevet "`var robot = new Drone(self);`".

Nu har vi os en Bygge-robot (og vi har kaldt den `drone`) og nu kan vi begynde at give den kommandoer!

Vi starter med at stor kasse af sten blokke:

```
exports.raketter = function() {
      //Skriv en besked på skærmen
      echo("Hej Minecraft verden");

      //Byg nogle blokke
      var drone = new Drone(self);
      drone.box(blocks.stone, 5, 1, 5);

      //Fyrværkeri!
 }
```

Hvis du vil give vores nye Bygge-robot en kommando skal du starte med at skrive "`drone`", så et punktum og til sidst din kommando. Altså "`drone.box(blocks.stone);`".

Det svarer lidt til at sige robottens navn og så give den en besked bagefter. Vi afslutter altid vores kommandoer med at skrive et semi-kolon "`;`".

Afprøv: Byg nogle blokke

Vi har nu udvidet vores program med to kommandoer. Så det er tid til at afprøve programmet i Minecraft igen.
Virkede det som du gerne ville have? Og har du husket "`/js refresh()`".

Argumenter

Du har leget lidt med "`box`" kommandoen før - vi bruger den når vi skal bygge en kasse. Men en kasse er

ikke bare en kasse. Den kan være lavet af forskellige blokke og kan have forskellige størrelser.

Derfor kan vi også sige til vores Bygge-robot at vi gerne vil have en sten kasse og at den skal være 5 blokke bred, 1 blok høj og 5 blokke lang. Det er præcist hvad vi gør når vi skriver "`box(blocks.stone, 5, 1, 5)`".

Det vi skriver mellem parenteserne "`(`" og "`)`" kalder vi *argumenter*.

Når vi skriver "`box(blocks.stone, 5, 1, 5)`" er der fire argumenter:

```
                1       2  3  4
drone.box(blocks.stone, 5, 1, 5);
```

- Første argument ("`blocks.stone`") bestemmer hvilken type blokke der skal bygges med.
- Andet argument ("`5`") bestemmer hvor bred kassen skal være.
- Trejde argument ("1") bestemmer hvor høj kassen skal være.
- Fjerde argument ("5") bestemmer hvor lang kassen skal være.

Det er vigtigt at huske at skrive et komma "," mellem hvert af argumenterne.

Udvid: Et lag sten mere

Det næste vi vil udvide vores program med er at bygge et lag sten mere. Det nye lag sten skal lægges oven på det gamle lag.

Bevæg robotten

Hvis vi skal bygge et nyt lag sten så skal vi først bevæge Bygge-robotten.

Vi ved allerede hvordan vi får robotten til at bevæge sig. Vi bruger kommandoerne "`up()`", "`down()`", "`fwd()`", "`back()`", "`left()`" og "`right()`" som vi allerede har prøvet tidligere.

Kommandoerne bruger vi på samme måde som vi brugte "`box`"-kommandoen. Vi skriver "`drone.up();`" hvis vi vil flytte Bygge-robotten op. "`drone.fwd();`" flytter Bygge-robotten frem. Og på samme måde med de andre kommandoer.

Her er den kode der skal til for at bygge næste lag af sten.

```
exports.raketter = function() {
     //Skriv en besked på skærmen
     echo("Hej Minecraft verden");

     //Byg nogle blokke
     var drone = new Drone(self);

     drone.box(blocks.stone, 5, 1, 5);

     drone.up();
     drone.fwd();
     drone.right();
     drone.box(blocks.stone, 3, 1, 3);

     //Fyrværkeri!
}
```

Husk på at du skal skrive "`/js refresh()`" i Minecraft-konsollen hver gang du har lavet om i dit program.

Fyrværkeri

Det sidste vores program skal gøre er at skyde fyrværkeri af. Bygge-robotten har en kommando til fyrværkeri: "`firework()`". Den bruges præcist på samme måde som de kommandoer vi allerede har brugt. Start med at skrive Bygge-robottens navn "`drone`", så et punktum "`.`" og til sidst kommandoen "`firework()`" - afslut med et semi-kolon"`;`". Altså "`drone.firework();`"

```
exports.raketter = function() {
     //Skriv en besked på skærmen
     echo(“Hej Minecraft verden”);

     //Byg nogle blokke
     var drone = new Drone(self);

     drone.box(blocks.stone, 5, 1, 5);

     drone.up();
     drone.fwd();
     drone.right();
     drone.box(blocks.stone, 3, 1, 3);

     //Fyrværkeri!
     drone.firework();
 }
```

Vigtige ting vi har lært:

Inden vi går i gang med eksperimenterne så lad os lige kigge tilbage på hvad vi har lært af vigtige ting i dette kapitel.

Lav en Bygge-robot

Vi kan lave en Bygge-robot og give den et navn med: “`var drone = new Drone(self);`”. Hvis vi syntes Bygge-robotten skal hedde “`robot`” kan vi skrive “`var robot = new Drone(self);`”.

Giv Bygge-robotten en kommando

Vi ved hvordan vi giver en kommando til Bygge-robotten. Vi skriver dens navn, skriver et punktum, så kommandoen og til sidst et semi-kolon. For eksempel “`drone.box(blocks.stone)`”.

Argumenter

Vi har lært at vi kan bruge argumenter når vi bruger en kommando.
Til “`box()`” kommandoen kan vi skrive fire argumenter: Hvilken type blokke der skal bruges, hvor bred kassen skal være, hvor høj kassen skal være og hvor lang kassen skal være.

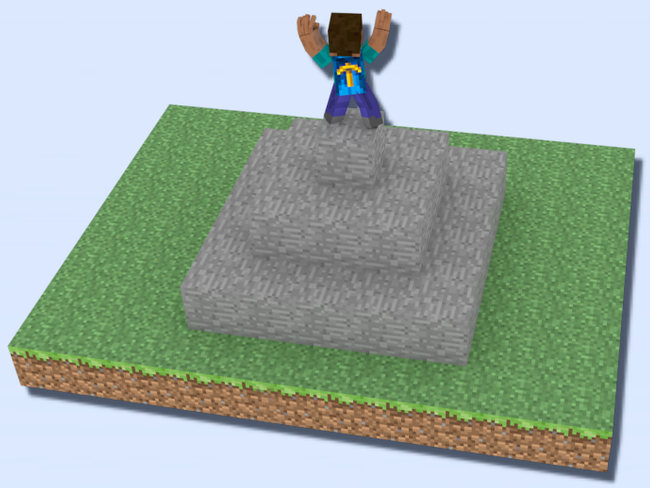
Eksperimenter

Eksperiment #1

Kan du lave et program der bygger en pyramide? Det nederste lag skal være 5 blokke bred, 5 blokke langt og 1 blok høj.

Næste lag skal være 3 blokke bred, 3 blokke lang og 1 blok høj.

Sidste lag skal bare være en enkelt blok.



Eksperiment #2

Kan du også sætte trapper på pyramiden?

Byg et hus

Vi er nu kommet godt i gang med at bruge vores kode-editor og vi har lært Bygge-robotten lidt bedre at kende.

Nu er det tid til at vi kaster os ud vores første store program.

Vi vil lave en kommando der bygger et hus.

Og undervejs skal vi lære lidt nye kommandoer til Bygge-robotten.

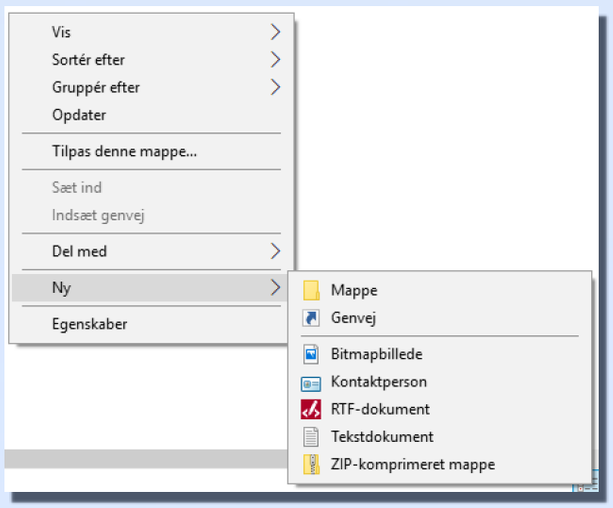
Byg et hus

Vi skal igang med at bygge et hus nu, men først skal vi have lavet en ny fil vi kan skrive vores kode i.

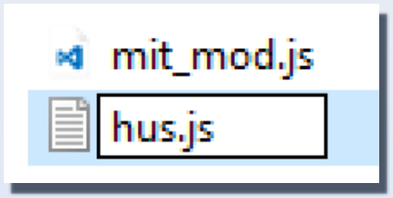
Lav en ny fil

Åben "`Coding pirates`" mappen som ligger på dit skrivebord og åben så "`mine_mods`" mappen. Det er her vi vil lave en ny fil vi kan skrive vores kode i.

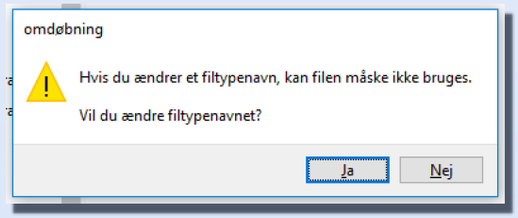
Nu skal du højre-klikke i vinduet og vælge "`Ny`" og bagefter "`Tekstdokument`".



Nu får du mulighed for at skrive filens navn. Du skal skrive "`hus.js`"

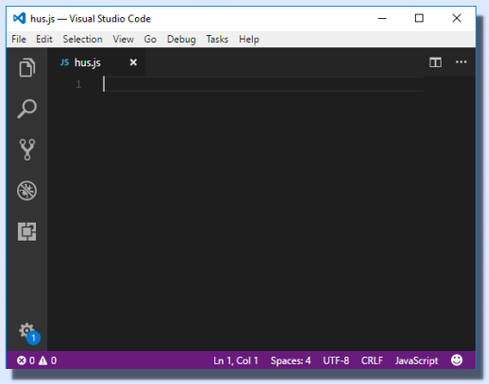


Du vil blive spurgt om du vil ændre fil-navnet.



Tryk på "`Ja`".

Hvis du dobbelt-klikker på "`hus.js`" åbner filen i din kode-editor.



En ny kommando: Hus

Vi starter med at lave en ny kommando, som vi kalder "hus", vi starter med den her kode:

```
exports.hus = function() {
      //Byg et hus
      echo("Nu skal vi bygge et hus");
 }
```

Nye tricks med Bygge-robotten

Inden vi går i gang med at bygge vores hus skal vi lige lære et par nye tricks og kommandoer til Bygge-robotten.

Flyt Bygge-robotten (langt)

Vi har allerede øvet os meget i at flytte rundt med Bygge-robotten med: `fwd()`, `back()`, `left()` og `right()`.

Hvis vi vil flytte Bygge-Robotten tre felter frem har vi skrevet:

```
drone.fwd();
drone.fwd();
drone.fwd();
```

Men der er faktisk en nemmere måde. Vi kan bruge et argument når vi bruger en `fwd()` kommandoen:

```
drone.fwd(3);
```

Sådan kan vi også flytte Bygge-robotten. Det virker også med `back()`, `left()`, `right()`.

Nye kommanoder til Bygge-robotten

Vi har ikke lært alle Bygge-robottens kommandoer at kende endnu. Du kan sagtens bygge et hus med de kommandoer du allerede har lært. Men de næste kommandoer gør det meget nemmere at bygge et hus.

`box0()`

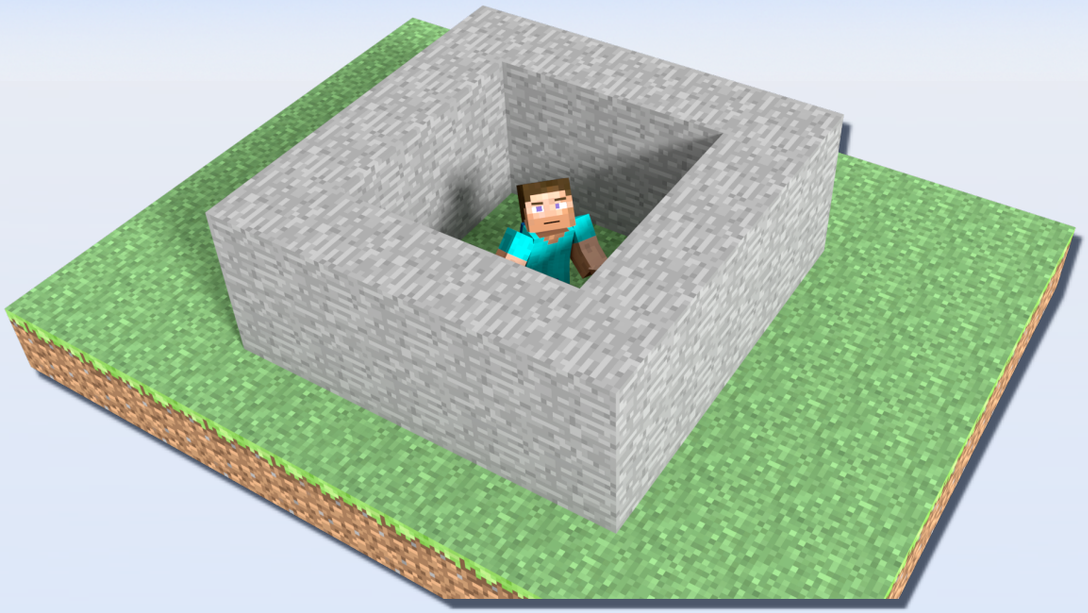
Du kender allerede Bygge-robottens `box()` kommando. Det er den du bruger når du skal bygge en blok. I sidste kapitel lærte du også hvordan du kunne få den til at bygge en stor kasse - ved at bruge argumenter.

Vi afprøver det lige igen, så prøv at starte Minecraft og prøv så at skrive `/js box(bloks.stone,5, 5, 5)` Minecraft-konsollen.

Slå nogle af blokkene i stykker - den kasse du lige har bygget er massiv. Det er ikke så smart hvis man gerne vil bygge et hus.

Prøv nu med `/js box0(bloks.stone,5, 5, 5)` (der står `box` og så tallet `0`, ikke bogstavet `0`).

Kan du se forskellen? Ny byggede du fire vægge i stedet for en massiv kasse! Perfekt når man skal bygge et hus.



`door()`

De fleste huse har døre. Døre kan vi bygge med Bygge-robottens `door()` kommando.

Afprøv selv `/js door()` i Minecraft-konsollen. Du kan bruge `/js door2()` til at bygge en dobbelt dør.

Og hvis du vil have en jern-dør kan du bruge `/js door(blocks.door_iron)`. Det virker også med `door2()`

`prism()`

Du vil sikkert gerne have et tag på dit hus. Du kan enten vælge at bygge et hus med et flat tag, så kan du bare bruge `box()` til at bygge det med.

Men hvis du gerne vil have et skråt tag så kan du bruge `prism()` som bygger en trekant.

Du skal skrive tre argumenter til `prism()`.

```
                    1          2  3
drone.prism(blocks.stairs.oak, 12, 13);
```

- Første argument er den type blok du gerne vil bruge. Du får det pæneste tag hvis du vælger en blok der starter med `blocks.stairs` for eksempel:`blocks.stairs.oak`, `blocks.stairs.birch`, `blocks.stairs.dark_oak` eller `blocks.stairs.stone`.
- Andet argument er hvor højt dit tag skal være.
- Tredje argument er hvor bredt dit tag skal være.

Der findes også en kommando der hedder `prism0()`, prøv at bruge Minecraft-konsollen til at finde ud af

hvad den gør.

`chkpt()` og `move()`

Vi skal flytte rigtigt meget rundt på Bygge-robotten når vi skal have den til at bygge et hus og så kan det godt blive lidt svære at holde styr på hvor den er.

Derfor kan det være rart at kunne flytte den tilbage til et bestemt sted.

Det kan vi med kommandoerne `chkpt()` og `move()`. `chkpt()` fungere som en slags bogmærke. Og `move()` flytter Bygge-robotten tilbage til dit bogmærke.

Du skal skrive et argument til både `chkpt()` og `move()` - nemlig navnet på dit bogmærke.

Du kan lave lige så mange bogmærker som du har lyst til, bare du giver dem forskellige navne.

Her er et eksempel på hvordan man kan bruge `chkpt()` og `move()`:

```
drone.door();
drone.chkpt("dør");
// Flyt rundt på Bygge-robotten og byg en masse
drone.move("dør");
```

Først bygger vi en dør og lige bagefter laver vi et bogmærke vi kalder "`dør`". Vi kan så bygge en masse (det kunne for eksempel være en væg med vinduer i). Og til sidst flytter vi Bygge-robotten tilbage til vores bogmærke ved døren ved at skrive `drone.move("dør")`.

Vinduer

Hvis du vil lave vinduer i dine vægge kan du bruge box kommandoen sammen med `blocks.glass_pane`:

```
drone.box(blocks.glass_pane, 2, 1, 1);
```

Trapper

Hvis du skal bygge en høj trappe er det nemmest at bruge `stairs()` kommandoen.

Du skal skrive tre argumenter til `stairs()` kommandoen:

```
                  1       2  3
drone.stairs(blocks.stairs.oak, 2, 10);
```

- Første argument er typen af trappe du vil bygge. For eksempel:`blocks.stairs.oak`, `blocks.stairs.birch`, `blocks.stairs.dark_oak` eller `blocks.stairs.stone`.
- Andet argument er hvor bred trappen skal være.
- Tredje argument er hvor høj trappen skal være.

Slet en blok

Du kan fjerne en blok du har bygget ved at bruge `box()` kommandoen du skal bare bygge med en luft blok.

Eksempel:

```
drone.box(blocks.air);
```

Fakler

Du kan bygge fakler på to måder. Du kan enten bruge `box(blocks.torch)`. Det virker rigtigt godt hvis du vil placere en fakkel på jorden.
Hvis du gerne vil placere en fakkel på en væg skal du

bruge kommandoen `hangtorch()` som hænger en fakkel på væggen.

Eksempel:

```
drone.box(blocks.stone, 10, 5, 1);
drone.right();
drone.up();
drone.back();
drone.hangtorch();
```

Senge

Du kan hurtigt bygge en seng ved at bruge `bed()` kommandoen.

Eksempel:

```
drone.bed();
```

Eksperimenter

Eksperiment #1

Byg et hus! Husk at arbejde efter: *Skriv - Afprøv - Ret - Udvid*.

Det kommer nok til at tage nogle forsøg inden det bliver helt perfekt, men det gør ikke noget.

I næste kapitel skal vi lave nogle eksploderende pile vi kan bruge til at "rydde op" med ;)

Sandt/falskt

Vi har brug for at kunne foretage valg i vores programmer.

Det kan være at vores program skal fremtrylle en mine-vogn når vi er på land, men en båd når vi befinder os på havet. Vores program skal altså kunne gøre noget forskelligt alt efter situationen.

I det her kapitel skal vi lære hvordan man får programmet til det.

Men vi starter lige med at springe noget i luften.

Spring noget i luften

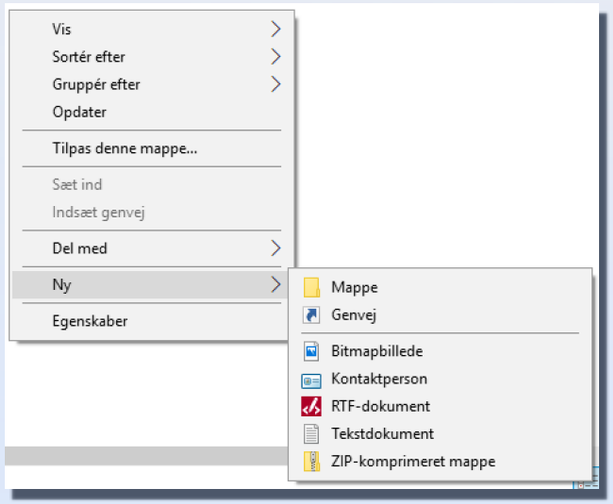
Vi starter det her kapitel med at springe ting i luften. Helt præcist vil vi lave en bue der skyder med pile der springer i luften når de rammer noget.

Men først skal vi have lavet en ny fil vi kan skrive vores kode i.

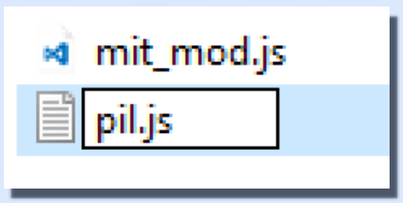
En ny fil

Åben "`Coding pirates`" mappen som ligger på dit skrivebord og åben så "`mine_mods`" mappen. Det er her vi vil lave en ny fil vi kan skrive vores kode i.

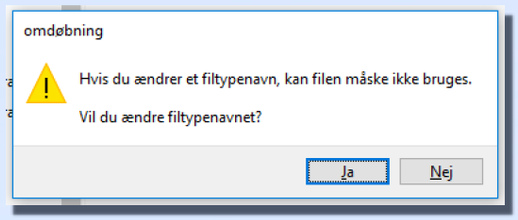
Nu skal du højre-klikke i vinduet og vælge "`Ny`" og bagefter "`Tekstdokument`".



Nu får du mulighed for at skrive filens navn. Du skal skrive "`pil.js`"

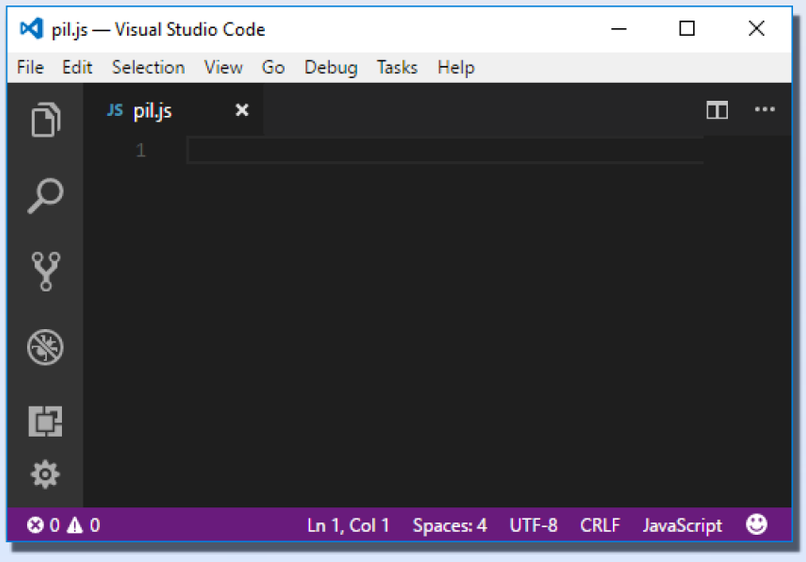


Du vil blive spurgt om du vil ændre fil-navnet.



Tryk på "`Ja`".

Hvis du dobbelt-klikker på "`pil.js`" åbner filen i din kode-editor.



Eksploderende pile

Nu skal vi have skrevet koden der kan få vores pile til at eksplodere.

Det kan vi gøre sådan her:

```
events.projectileHit(function ( event ) {
    var pil = event.entity;
    var world = pil.world;

    world.createExplosion( pil.location, 5);
});
```

Du skal ikke tænke for meget over koden den skal nok blive forklaret senere.

Gem din fil, start Minecraft og afprøv din kode (Husk: *Skriv - Afprøv - Ret - Udvid*)

Spring ikke grisene i luften

Det er ikke pænt af os at springe dyr i luften, så vi vil gerne have lavet vores program om så pilene kun springer i luften hvis vi rammer en blok.

Vi vil gerne fortælle Minecraft: "**Hvis** pilen rammer en blok **så** lav en eksplosion". Når vi gerne vil have Minecraft til kun at gøre noget i bestemte situationer så skal vi bruge en "`if`"-sætning, "if" er det engelske ord for "hvis".

if-sætning

En "if"-sætning ser sådan her ud:

```
if( Betingelse ) {
// Det her sker hvis betingelsen er sand
}
```

"if"-sætningen har to dele: En betingelse (1) og det som skal ske hvis betingelsen er sand (2).

Hvad er en betingelse

Lad os starte med at finde ud af hvad en betingelse er. Vi kigger i ordbogen igen. Ordbogen siger at en betingelse: "Krav der stilles før noget kan ske". Lidt knudret så vi tager et par eksempler:

- Vi vores program vil vi kun have en eksplosion hvis vi rammer en blok.Så vores betingelse for at lave en eksplosion er: *At vi har ramt en blok*.
- Vi gerne vil fremtrylle en båd hvis vores pil rammer noget vand. Så vores betingelse for at fremtrylle en båd er: *At vores pil rammer noget vand*.

Sandt og falskt

Nu har vi en lidt bedre idé om hvad en betingelse er, men computere forstår os ikke hvis vi bare skriver "At min pil har ramt en blok" som betingelse.

Faktisk forstår computeren kun to ting som betingelser: `"true"` og `"false"`. `"true"` og `"false"` er de engelske ord for "Sandt" og "Falsk".

Lad os prøve os lidt frem.

Skriv det her nederst i "`pil.js`" og afprøv det i Minecraft.

```
exports.gætEtTal = function(){
    if(true) {
        echo("Det er sandt");
    }
}
```

Husk at bruge "`/js refresh()`" først. Du kan afprøve det med "`/js gætEtTal()`". Dukkede der en besked op på skrærmen?

Prøv nu at ændre det til:

```
exports.gætEtTal = function(){
    if(false) {
        echo("Det er sandt");
    }
}
```

Og afprøv det i Minecraft igen. Dukkede der nu en besked op på skærmen?

Så hvis din betingelse er "`true`" så får bliver beskeden "Det er sandt" vist på skærmen.

Og hvis din betingelse er "`false`" bliver det beskeden ikke vist.

Lad os lige kigge på "`if`"-sætningen igen:

```
        1
if( Betingelse ) {
// Det her sker hvis betingelsen er sand 2
}
```

Så nu ved vi at hvis 1 er sand så sker det vi skriver i 2.

Spring kun blokke i luften

Nu vender vi tilbage til vores eksploderende pile lidt. Vi skal have fundet ud af hvordan vi får skrevet betingelsen "**Hvis** pilen rammer en blok **så** lav en eksplosion" på en måde så computeren kan forstå det.

Vi prøver os lidt frem først, ved at ændre i vores program ved at tilføje en "`if`"-sætning til. Så der nu står:

```
events.projectileHit(function ( event ) {
    var pil = event.entity;
    var world = pil.world;

    if(false) {
        world.createExplosion( pil.location, 5);
    }
});
```

Afprøv det nu i Minecraft ved at skyde et par pile af.

Prøv nu at ændre din kode til:

```
events.projectileHit(function ( event ) {
    var pil = event.entity;
    var world = pil.world;

    if(true) {
        world.createExplosion( pil.location, 5);
    }
});
```

Vi kan nu bestemme om vi pilen eksplodere. Nu skal vi bare have fundet den rigtige betingelse.

Inden vi skal det skal vi have kigget lidt mere på den kode vi har skrevet.

Vi startede med den her kode:

```
1 events.projectileHit(function ( event ) {
2     var pil = event.entity;
3     var world = pil.world;

4     world.createExplosion( pil.location, 5);
  });
```

Linje 1 fortæller Minecraft at vi gerne vil have kørt vores kode hvergang en pil rammer noget.

Linje 4 laver en eksplosion.

For at lave en eksplosion vi bruge to ting: ~~Dynamit og en tændstik.~~ Noget der kan lave en eksplosion og stedet hvor eksplosionen skal ske.

Når Minecraft kører kører vores kode giver den os lidt at vide om hvad der er sket.
Kan du huske at vi kan bruge argumenter til Bygge-robottens kommandoer? Minecraft kan også bruge argumenter når den kører vores kode.

Kigger du i linje 1 kan du se at der står "function(event)". Det betyder at Minecraft sender ét argument til os nemlig: event.

"event" kan fortælle os en masse forskelligt om hvad der lige er sket i Minecraft.

Lige nu er vi interesserede i hvor pilen ramte. For at få det at vide skal vi først have fat i pilen. Vi får fat i pilen i linje 2.

I linje 3 får vi fat i den Minecraft-verden vi er i. Vi skal bruge Minecraft-verdenen til at lave en eksplosion.

Vi er vant til at kunne fortælle Bygge-robotten at den skal bygge en blok. Det gør vi ved at give den en kommando.

Byggerobotten kan ikke lave eksplosioner, men det kan den Minecraft-verden vi er i.

Ramte vi en blok?

"event" kan også fortælle os om vi ramte en blok. Vi kan nemlig spørge "event.hitBlock".

Det betyder at vi kan ændre vores kode til:

```
events.projectileHit(function ( event ) {
    var pil = event.entity;
    var world = pil.world;

    if(event.hitBlock) {
        world.createExplosion( pil.location, 5);
    }
});
```

Vi har ændret vores betingelse til: "har ramt blok".

Afprøv koden i Minecraft. Husk at bruge "/js refresh" først og skyd så en pil af!

Og ellers?

Nu har vi lavet en pil der eksplodere hvis den rammer en blok og virker som en normal pil hvis den rammer et dyr.

Nu vil gerne lave vores program om sådan at: **Hvis** vi rammer en blok **så** lav en eksplosion **ellers** lav en vand-blok.

Vi kan allerede finde ud af at lave en eksplosion hvis vi rammer en blok, så vi skal bare have fundet ud af hvordan vi klarer "**ellers** lav en vand-blok".

Det engelske ord for "ellers" er "else" og det er præcist det ord vi er på jagt efter. Vi kan nemlig skrive en "if-else"-sætning.

```
if( Betingelse ) {
// Det her sker hvis betingelsen er sand
} else {
// Ellers sker det her
}
```

Du kender allerede den første del: Betingelsen 1.

Hvis betingelsen i 1 er "sand" sker det vi har skrevet i 2.

Men hvis betingelsen i 1 er "falsk" så sker det vi skriver i 3.

Lad os udvide vores program:

```
events.projectileHit(function ( event ) {
    var pil = event.entity;
    var world = pil.world;

    if(event.hitBlock) {
        world.createExplosion( pil.location, 5);
    } else {
        var drone = new Drone(pil.location);
    }
});
```

Nu har du en Bygge-robot du kan bygge noget med.

Bygge-robotten starter der hvor pilen ramte.

Prøv at bygge noget med den.

Lav et gætte program

Snak om hvordan man sammenligner ting

Hændelser

Kyllinge kanonen

Vi skal nu lave os en kyllinge-kanon!

Vi skal bygge videre på sandt/falsk og hændelser som vi lært i de sidste to kapitler.

Kyllinge-kanonen

Som det første skal vi have lavet os en kyllinge-kanon. Egentligt er det bare en pind vi har skrevet "Kyllinge-kanon" på, men behøver vi jo ikke sige til nogen.

Du skal starte med at lave en ny fil dit mod kan være i, kald den "kyllingekanon.js" (kig tilbage i de forrige kapitler hvis du ikke kan huske hvordan du laver en ny fil).

Vi starter med at lave en kommando der giver os den mægtige kyllinge-kanon:

```
exports.kyllingeKanon = function() {
   echo("Du skænkes hermed den mægtige kyllinge-kanon");
}
```

Farvet tekst

Hvis du afprøver dit mod i Minecraft nu med "/js kyllingeKanon()" (husk "/js refresh()") vil du få vist beskeden: "Du skænkes hermed den mægtige kyllinge-kanon".

Det er jo sådan set fint nok. Men det virker ikke særligt mægtigt. Lad os lige se om vi ikke kan gøre noget ved den besked.

Prøv at ændre dit program til så der nu står:

```
exports.kyllingeKanon = function() {
   echo("Du skænkes hermed den mægtige kyllinge-kanon".gold());
}
```

Afprøv dit program igen. Kan du se forskellen? Teksten blev nu skrevet med guld skrift. Du kan også bruge `.red()`, `.green()` og `.yellow()` for at skrive med andre farver.

Giv dig selv ting

Vores besked er nu tilpas mægtig, nu må vi videre! Det næste vi skal gøre er at lave noget vi kan bruge som kanonen.

Planen er at vi bruger en pind, men vi giver pinden "Kyllinge-kanonen". Vi starter sådan her:

```
var inventory = require('inventory');
var items = require('items');
exports.kyllingeKanon = function() {
   echo("Du skænkes hermed den mægtige kyllinge-kanon".gold());
}
```

De to første linjer gør at vi nu kan bruge lidt flere af de ting som Minecraft kan. Vi kan nu lave genstande ("`items`" er det engelske ord for genstande) og vi kan give spillere genstande til deres inventar ("`inventory`" er det engelske ord for inventar).

Lav en pind

Som det næste skal vi have lavet en pind og givet den navnet "Kyllinge-kanonen".

Vi starter med at lave en pind sådan her:

```
var inventory = require('inventory');
var items = require('items');
exports.kyllingeKanon = function() {
   echo("Du skænkes hermed den mægtige kyllinge-kanon".gold());

   var stick = items.stick(1);

}
```

Vi skal gøre lidt pinden senere så vi gemmer den som "`stick`". Husk at når vi skriver "`var stick`" betyder det at vi senere kan bruge pinden ved at skrive "`stick`" (på samme måde som når vi kaldte vores bygge-robot "`robot`" eller "`drone`").

"`items.stick(1)`" giver os én pind. Ville vi have haft et æg kunne vi have skrevet "`items.egg(1)`".

Ville vi have haft 10 pinde så skulle vi have skrevet "`items.stick(10)`".

Lige nu nøjes vi med en enkelt pind.

Nu skal vi have givet pinden et navn:

```
var inventory = require('inventory');
var items = require('items');
exports.kyllingeKanon = function() {
   echo("Du skænkes hermed den mægtige kyllinge-kanon".gold());

   var stick = items.stick(1);
   var meta = stick.itemMeta;
   meta.displayName = "Kyllinge-kanonen";
   stick.itemMeta = meta;

}
```

Det var lidt en smøre, men du behøver ikke forstå det endnu. Bare husk at hvis du vil give et navn til et element så er det sådan her du kan gøre det.

Giv pinden til dig selv

Nu mangler vi bare at give ~~pinden~~ kyllinge-kanonen til os selv. Det kan vi gøre med:

```
var inventory = require('inventory');
var items = require('items');
exports.kyllingeKanon = function() {
   echo("Du skænkes hermed den mægtige kyllinge-kanon".gold());

   var stick = items.stick(1);
   var meta = stick.itemMeta;
   meta.displayName = "Kyllinge-kanonen";
   stick.itemMeta = meta;

   inventory(self).add(stick);
}
```

"`inventory(self)`" giver os adgang til en spillers inventar. Nu kan vi give inventaret en kommando om at tilføje pinden vi lige har lavet med: "`add(stick)`"

Afprøv det nu i Minecraft. Bruger du kommandoen "`/js kyllingeKanon()`" skulle du nu gerne få en pind der hedder "`Kyllinge-kanonen`".

Hårdt kastede kyllinger

Nu har vi anskaffet os vores mægtige kyllinge-kanon. Nu skal vi have den til at skyde med kyllinger.

Kyllingen skal skydes når du slår med kyllinge-kanon pinden. Du har tidligere lavet eksploderende pile, og det er lidt det samme vi skal igen.

Vi starter med:

```
events.playerInteract(function(event) {
    echo("Skyd en kylling!");
    var spiller = event.player;
}
```

Afprøv det i Minecraft. Tag en tilfældig ting i hånden og prøv at slå med dem. Blev teksten skrevet? Perfekt!

Skab en kylling

Nu ved vi at vi har fat i den rigtige hændelse. Nu er det på tide at få skabt en kylling.

```
var entities = require('entities');
events.playerInteract(function(event) {
    echo("Skyd en kylling!");
    var spiller = event.player;
    var verden = spiller.world;
    var kylling =
       verden.spawnEntity(spiller.location, entities.chicken())
}
```

Først får vi fat i den verden som du er i, det gør vi med "`spiller.world`". Vi gemmer den som "`verden`".

Nu kan vi give verdenen en kommando om at lave et væsen. Det kan vi bruge "`spawnEntity`" kommandoen til. Du skal give "`spawnEntity`" to argumenter:

1) Hvor i verdenen væsnet skal skabes

2) Hvilket slags væsen der skal skabes

Denne gang vil vi gerne lave en kylling (`entities.chicken()`) og den skal laves der hvor spilleren er (`spiller.location`).

Kun kyllinge-kanonen virker

```
var entities = require('entities');
events.playerInteract(function(event) {
    echo("Skyd en kylling!");
    var spiller = event.player;
    var verden = spiller.world;
    var holder = spiller.itemInHand;
    var navn = holder.itemMeta.displayName;

    if(navn === "Kyllinge-kanonen") {
       var kylling =
         verden.spawnEntity(spiller.location, entities.chicken())
    }

}
```

```
var retning = spiller.location.direction
        retning.x *= 6;
        retning.y *= 6;
        retning.z *= 6;

kylling.velocity = retning
```

Tryllestave

Det første vi laver til vores mod er en masse tryllestave og andre våben. For hver tryllestav skal vi beslutte tre ting:

- Hvordan skal den se ud?
- Hvad skal den skyde med?
- Hvad skal der ske når den rammer?

Ud fra de fire ting kan vi lave næsten uendeligt mange forskellige våben. Se med på næste side, så går vi igang!

De tre ingredienser til en tryllestav

1: Vi skal vælge hvordan vores tryllestav skal se ud. Skal de være en pind? En fisk? En kage?

2: Hvad skal vi skyde med? Snebold? Ildkugle? Eller måske Lamaspyt?

3: Sidst med ikke mindst hvad skal der ske når vi rammer? Skal modstanderen kastes op i luften? Hegnes inde? Skal jorden under dem laves til lava?

Tryllestavens tre dele

Før vi kan lave vores første tryllestav skal vi have en ny fil (kig tilbage i bogen hvis du ikke kan huske hvordan du gør). Hver gang vi laver en ny tryllestav gør vi det i en ny fil - det gør det nemmere for os at finde rund i vores kode senere. Vælg et godt navn til din fil: `lavaTryllestav.js` er bedre end `tryllestav47.js` hvis du senere skal finde rund i din utrolige samling af diabolske tryllestave.

Første del: Hvordan skal den se ud

Vi starter ud med med det her:

```
var items = require("items");
var inventory = require("inventory");

exports.trylleStav = function() {
  echo("Et styk tryllestav!");
};
```

Her har vi bare startet med at lave en ny kommando der skriver en tekst på skærmen. Det er et godt sted at starte fordi det gør at vi i Minecraft kan teste om vi har lavet filen det rigtige sted.

Start Minecraft og afprøv din kommando (husk at du skal skrive `/js refresh()` først).

Nu skal vi vælge hvordan vores tryllestav skal se ud og hvad den skal hedde. Vi starter med en pind og vi kalder den "`Stav#1`" (det er et ret dårligt navn, find bare på noget bedre!)

```
var navn = "Stav#1";
exports.trylleStav = function() {
  echo("Et styk tryllestav!");
  // Hvad skal tryllestaven være lavet af?
  var stav = items.stick(1);

  // Hvad skal den hedde?
  var meta = stav.itemMeta;
  meta.displayName = navn
  stav.itemMeta = meta;

  // Giv den til dig selv
  inventory(self).add(stav);
};
```

Variabler

Der er meget ny kode her, så lad os kigge på det en linje af gangen.

Første linje siger: `var navn = "Stav#1";`

Her laver vi en *variabel*. Du kan tænke på variable som en kasse du putter ting i. Her laver vi en kasse der hedder "`navn`" og putter teksten "`Stav#1`" i kassen.

Når du så senere i din kode gerne vil bruge teksten "`Stav#1`" skriver du så i stedet "`navn`". Så kigger computeren i kassen der hedder "`navn`" og finder teksten "`Stav#1`". Du har allerede brugt variabler da du brugte Bygge-robotten.

Nu tænker du måske "Hvorfor er variabler smarte"? Vi kommer til at skulle bruge navnet på vores tryllestav *mange* steder i vores kode. Vi kunne godt bare skrive "`Stav#1`" alle de steder, men hvis vi skifter mening og gerne vil skifte navnet til "`Lava-inator`" så skal vi huske at rette *alle* de steder vi har skrevet "`Stav#1`" ellers går vores kode i stykker. Hvis vi istedet har brugt vores variabel, så skal vi kun rette navnet ét sted, nemlig i linje 1.

Materialet

I linje 2 laver vi vores kommando, det har vi set mange gange før. I linje 3 skriver vi en besked ud.

Det næste spændene sker i linje 5: "`var stav = items.stick(1);`". Her laver vi vores tryllestav (og gemmer den i en kasse der heder stav).

Afprøv hvad der sker hvis du skriver et andet tal en "`1`" i parenteserne.

Her er vores tryllestav lavet af en pind. Hvis du gerne vil have en fisk i stedet skal du skrive "`var stav = items.rawFish(1);`".

Der er rigtig mange forskellige ting man kan bruge. Her er lidt eksempler:

Materiale	Kode
Fisk	items.rawFish(1)
Kage	items.cake(1)
Stav	items.stick(1)

Du kan finde endnu flere materialer her: https://hub.spigotmc.org/javadocs/spigot/org/bukkit/Material.html

Navnet

I linjerne 7 - 9 giver vi vores tryllestav et navn. Og til sidst giver vi os selv tryllestaven med: "`inventory(self).add(stick);`". Hvis vi oversætter står der: tilføj tryllestaven (`add (stick)`) til min rygsæk (`inventory(self)`).

Så er det tid til at test din kommando igen. Får du en tryllestav (eller tryllefisk) med det rigtige navn?

Anden del: Hvad skal vi skyde med?

Nu er det på tide at vælge hvad vi skal skyde med. Vi starter med en snebold. Men hvordan ved vi hvornår spilleren bruger tryllestaven?

Hændelser

Vi ved at en spiller bruger tryllestaven fordi Minecraft fortæller os det og det gør den via hændelser.

Der er mange forskellige typer hændelser i Minecraft: Det er en hændelse når en blok bliver ødelagt. Det er en hændelse når nogen tager skade. Og så er der "`playerInteract`" hændelsen. Det er lige præcis den vi skal bruge.

Tilføj det her til din kode:

```
events.playerInteract(function(event) {
   echo("Et slag i luften");
});
```

Start Minecraft (husk "`/js refresh()`"). Tag din tryllestav i hånden og prøv at bruge den.

Bliver der skrevet "Et slag i luften" på skærmen? Perfekt.

Prøv nu at tage noget andet i hånden og prøv igen. Så blev der også skrevet "Et slag i luften". Øv!

Valg

Vi skal have vores kode til kun at gøre noget når vi står med lige præcis vores tryllestav i hånden. Kan du huske kapitlet om "Sandt/Falsk"? Der lærte vi hvordan vi kan få koden til kun at gøre noget når en betingelse var sand.

Nu er vores betingelse: Gør kun noget når vi står med vores tryllestav i hånden.

Hvad står vi med i hånden?

Først skal vi vide hvad vi står med i hånden. Det klare vi med:

```
events.playerInteract(function(event) {
   var spiller = event.player;
   var ting = spiller.itemInHand;
});
```

I linje 1 siger vil til Minecraft: Giv mig besked når en spiller bruger noget de har i hånden.

Når Minecraft giver os besked om en hændelse giver den os lidt information med om hvad der er sket. Det kommer med som en *variabel* der hedder `event`.

I linje 2 får vi fat i den spiller der har gjort noget og gemmer det i en variabel vi kalder spiller.

Nu skal vi så vide hvad spilleren har i hånden. Det kan vi få at vide med `spiller.itemInHand` (oversat: "ting i hånden").

Funktioner

Hvad er det der sker når vi skriver `spiller.itemInHand`?

"`spiller`" er en variabel vi har lavet, men denne gang indeholder kassen ikke bare en tekst. Denne gang ligger der en spiller nede i kassen. Og en spiller kan vi stille spørgsmål som f.eks. "Hvad for en ting holder du i hånden". Vi kan også *gøre* noget ved spilleren, som for eksempel at kaste dem op i luften.

Vi gør det ved hjælp af noget man kalder funktioner. En funktion er faktisk bare nogle linjer kode som er pakket ind så de kan blive kørt når vi beder om det.

Du har faktisk allerede skrevet en masse funktioner. Faktisk hver gang du laver en kommando. Det engelske ord for funktion er "`function`". Kan du få øje på det ord i din kode?

Spiller har en funktion der kan give os hvad de holder i hånden. Og det har vi nu gemt i en ny variabel vi har kaldt "`ting`". Puha! Det var en ordentlig omgang, men nu er vi der også næsten.

Faktisk mangler vi bare at spørge "`ting`" om den er vores tryllestav, gad vide om den har en *funktion* der kan fortælle os det?

Det har den heldigvis:

```
events.playerInteract(function(event) {
   var spiller = event.player;
   var ting = spiller.itemInHand;
   var tingensNavn = ting.itemMeta.displayName;

   if (tingensNavn === navn) {
      echo("Tryl!");
   }
});
```

Kan du se at vi har tilføjet tre linjer? En variabel der hedder `tingensNavn` og i den har vi gemt navnet på det spilleren holder i hånden.

Og så har vi lavet en if-sætning: Den sammenligner det navn vi har givet vores tryllestav med tingens navn. Og kun hvis det er sandt skriver den teksten "Tryl!" på skærmen.

Afprøv det!

Skyd!

Nu har vi styr på at vores kode kun kører når spilleren bruger vores tryllestav.

Nu er det på tide at skyde med noget! Og det er heldigvis ret nemt :)

```
events.playerInteract(function(event) {
   var spiller = event.player;
   var ting = spiller.itemInHand;
   var tingensNavn = ting.itemMeta.displayName;

   if (tingensNavn === navn) {
    // Projektil typer
    var fireball = org.bukkit.entity.SmallFireball.class;
    var largeFireball = org.bukkit.entity.LargeFireball.class;
    var lamaSpyt = org.bukkit.entity.DragonFireball.class;
    var snebold = org.bukkit.entity.Snowball.class;

    spiller.launchProjectile(snebold);
   }
});
```

Det er `spiller.launchProjectile(snebold)` som gør alt arbejdet. `launchProjectile` er en funktion der skyder et projektil afsted. Argumentet "`snebold`" betyder at vi skyder en snebold afsted.

Afprøv det i Minecraft.

Prøv så at skifte snebold ud med fireball.

Tredje del: Hvad skal der ske når vi rammer?

Nu har vi skudt en snebold afsted. Nu skal vi bestemme os for hvad der skal ske når den rammer.

Hvordan ved vi at den overhovedet har ramt? Det har Minecraft en hændelse der kan fortælle os, nemlig "`projectileHit`" (oversat: projektil har ramt).

Tilføj det her til din fil:

```
events.projectileHit(function(event) {
    var denDerSkød = event.entity.shooter;
    echo(denDerSkød, "Av! Du fik mig!");
});
```

Afprøv det med din tryllestav. Prøv så at skyde med en pil. Øv, nu sker det igen for alle tingene.

Men vi ved heldigvis allerede hvordan vi løser det:

```
events.projectileHit(function(event) {
  var denDerSkød = event.entity.shooter;
  var navnPåTryllestav =
     denDerSkød.itemInHand.itemMeta.displayName;

  if(navnPåTryllestav === navn) {
    echo(denDerSkød, "Av! Du fik mig!")
  }
});
```

Entity eller Block

Nu er vi *endeligt* nået så langt at vi ved at vi har ramt og nu skal vi beslutte os for hvad der nu skal ske.

Vi kan have ramt to forskellige ting: Levende væsner eller blokke. I Minecraft termer: `Entity` (et levende væsen) eller `Block` (en blok).

Det kan være at vi kun vil gøre noget hvis man rammer et levende væsen (f.eks en modspiller eller en gris) eller kun hvis man rammer en blok.

Vi er så heldige at Minecraft fortæller os hvilken Block eller Entity vi ramte, så vi kan fortsætte sådan her:

```
events.projectileHit(function(event) {
  var denDerSkød = event.entity.shooter;
  if(navnPåTryllestav === navn) {
    var block = event.hitBlock;
    var entity = event.hitEntity;

    if(block) {
      echo(denDerSkød, "Du ramte en Block")
    } else {
      echo(denDerSkød, "Du ramte en Entity")
    }
  }
}
});
```

Afprøv det i Minecraft. Prøv at skyde på både blokke og levende væsner.

Effekter

Vores tryllestav gør stadig ikke noget, så her er et par ideer til hvad den kan gøre

Lav en block om til lava

```
if(block) {
   echo(denDerSkød, "Du ramte en Block")
   block.type = org.bukkit.Material.LAVA;
} else {
  echo(denDerSkød, "Du ramte en Entity")
}
```

Kast dem op i luften

```
if(block) {
   echo(denDerSkød, "Du ramte en Block")
} else {
  echo(denDerSkød, "Du ramte en Entity")
  entity.setVelocity(new org.bukkit.util.Vector(0,2,0))
}
```

Sæt ild til dem

```
if(block) {
   echo(denDerSkød, "Du ramte en Block")
} else {
  echo(denDerSkød, "Du ramte en Entity")
  entity.setFireTicks(20);
}
```

Lav vand under dem

```
if(block) {
   echo(denDerSkød, "Du ramte en Block")
} else {
  echo(denDerSkød, "Du ramte en Entity")
   var denBlockDeStårPå =
       event.hitEntity.location.subtract(0, 1, 0).block;

    denBlockDeStårPå.type = org.bukkit.Material.WATER;
}
```

Man kan få lov at gøre næsten alt.

En skabelon

Nu har vi faktisk en skabelon vi kan bruge til alle vores tryllestave! Når du vil lave en tryllestav starter du bare med skabelonen og ændre den til den passer med din ide.

```
var items = require("items");
var inventory = require("inventory");
// Navn oå tryllestav
var navn = "Stav#1";
exports.trylleStav = function() {
  //Hvad skal tryllestaven være lavet af?
  var stav = items.stick(1);

  //Hvad skal den hedde?
  var meta = stav.itemMeta;
  meta.displayName = navn
  stav.itemMeta = meta;

  //Giv den til dig selv
  inventory(self).add(stav);
};

events.playerInteract(function(event) {
  var spiller = event.player;
  var ting = spiller.itemInHand;
  var tingsNavn = ting.itemMeta.displayName;

  if (tingsNavn === navn) {
    //Projektil typer
    var fireball = org.bukkit.entity.SmallFireball.class;
    var largeFireball = org.bukkit.entity.LargeFireball.class;
    var lamaSpyt = org.bukkit.entity.DragonFireball.class;
    var snebold = org.bukkit.entity.Snowball.class;

    spiller.launchProjectile(snebold);
  }
});
```

```
events.projectileHit(function(event) {
  var denDerSkød = event.entity.shooter;
  var navnPåTryllestav = denDerSkød.itemInHand.itemMeta.displayName;

  if(navnPåTryllestav === navn) {
    var block = event.hitBlock;
    var entity = event.hitEntity;

    if(block) {
      //Hvad skal der ske når du rammer en blok?
      echo(denDerSkød, "Du ramte en Block")
    } else {
      //Hvad skal der ske når du rammer et levende væsen?
      echo(denDerSkød, "Du ramte en Entity")
    }
  }
```

Gentagelser

I sidste kapitel så vi blandt andet hvordan man kunne lave en bestemt blok om til vand. Men hvad nu hvis vi gerne vil lave ti blokke om? Vi kan enten skrive den samme kode ti gange eller vi kan fortælle computeren at den skal gentage noget 10 gange. I programmering kalder vi det for "løkker" og det er det vi skal se på i det her kapitel.

Den første løkke

Nu skal vi lave vores første løkke: Vi får computeren til at tælle til ti.

Vi start som vi plejer med at lave en ny fil i vores kode-editor. Control-N for at få en ny fane, Control-S for at gemme filen. Husk at filen skal ligge under "mine_mods" og skal slutte med .js

Så laver vi os en ny kommando som vi kalder op.

```
exports.op = function() {
  for (i = 0; i < 10; i++) {
    echo(i);
  }
};
```

Start Minecraft og kør din nye kommando. Bliver der talt til ti?

For-løkke

Super! Du har lige skrevet din første for-løkke. Lad os tage et nærmere kig på den:

```
for (i = 0; i < 10; i++) {
 echo(i);
}
```

Der er fire elementer i en for-løkke:

1 Vi skal bruge noget der holder styr på hvor vi skal starte og langt vi er kommet. Det sørger "i=0" for. Vi laver en variabel "i" og starter med at sige den skal være 0. Vi kalder tit "i" for tælle-variablen, fordi vi bruger den som en tæller. Du behøver ikke kalde den "i", men det er tit det vi kalder den.

2 Vi skal vide hvornår vi skal stoppe. Det bruger vi "i < 10" til. "i < 10" er en betingelse (kan du huske kapitlet om betingelser? Det er dem som altid enten er "sande" eller "falske"). Du kender nok "<" fra dine matematik timer. Den er betingelse siger: "Så længe i er mindre end ti er jeg sand".

3 Vi vil gerne have vores løkke til at stoppe på et tidspunkt. For-løkken stopper når betingelsen er falsk. Hvornår er den det? Prøv at åbne din konsol i Minecraft og afprøv: "/js 1 < 10". Kan du se at vi har sat "1" ind på i's plads? Prøv at sætte 2 ind i stedet. Hvor langt skal du tælle op før der bliver skrevet "false"? Når vi i 3 skriver "i++" fortæller vi computeren: Når du har kørt 4 så tæl "i" op med én.

4 Her imellem { og } står den kode vi gerne vil have computeren til at gentage.

Hvordan ser det ud for computeren?

I sidste kapitel snakkede vi om "variable". Variable var kasser vi kunne give et navn og putte ting i. Ved senere at skrive navnet vi havde givet kassen kunne vi få fat det der lå i kassen.

Lad os lege computer og køre vores for-løkke i hovedet. Vi bruger en orange pil til at holde styr på hvor vi er.

```
for (i = 0; i < 10; i++) {
 echo(i);
}
```

"i = 0" siger at nu skal vi lave en kasse der hedder "i" og putte "0" ned i.

```
for (i = 0; i < 10; i++) {
 echo(i);
}
```

Nu skal vi se efter om vi skal kører videre. Vi laver sammenligningen i < 10. Først kigger vi i kassen "i" og ser hvad der ligger i? Ok, sammenligningen bliver "0 < 10". Er nu mindre end 10? Ja, vi kører videre.

```
for (i = 0; i < 10; i++) {
 echo(i);
}
```

Nu springer vi ned til det der står mellem { og } og kører det. Vi siger højt "0".

```
for (i = 0; i < 10; i++) {
 echo(i);
}
```

Så kom vil til "i++". Det betyder tag indholdet af kasse "i" og tæl det op med én.

```
for (i = 0; i < 10; i++) {
 echo(i);
}
```

Så kigger vi på sammenligningen igen. i = 1, så sammenligningen bliver "1 < 10". Det er stadig sandt så vi fortsætter.

```
for (i = 0; i < 10; i++) {
 echo(i);
}
```

Vi siger nu højt "1" (hvorfor siger vi 1 her?)

```
for (i = 0; i < 10; i++) {
 echo(i);
}
```

Så tæller vi det der ligger i kassen "i" op med én.

```
for (i = 0; i < 10; i++) {
 echo(i);
}
```

Vi kigger på sammenligningen igen som nu er? 2 < 10, stadig sandt.

```
for (i = 0; i < 10; i++) {
 echo(i);
}
```

Vi fortsætter og siger højt "2".

Lad os spole lidt frem til vi har kørt så mange gange at vi lige har sagt "9" højt.

```
for (i = 0; i < 10; i++) {
 echo(i);
}
```

Vi tæller indholdet af i op med én så nu ligger der "10" i kassen "i".

```
for (i = 0; i < 10; i++) {
 echo(i);
}
```

Og vi kigger på sammenligningen igen. “i” er nu 10, så sammenligningen bliver “10 < 10”. Er det sandt? Nej, og så stopper vores løkke.

Eksperimenter

Eksperiment #1

Lav en for-løkke der tæller til 20.

Eksperiment #2

Lav en for-løkke der tæller til 20, men springer hver andet tal over.

Eksperiment #3

Lav en nedtælling fra 10 til 0. Du kan bruge “i--” til at tælle ned med, men du skal også lave om betingelsen.

Du kan sætte den her kode ind efter din nedtælling:

```
var sted = self.location.subtract(i, 0, 0);
var verden = self.world;
var ko = verden.spawn(sted, org.bukkit.entity.Cow.class);
ko.setVelocity(new org.bukkit.util.Vector(0, 2, 0));
```

Forsinkelser

Indtil nu har computeren altid gjort hvad vi har bedt om med det samme.

I det her kapitel skal vi se på hvordan vi beder computeren skal gøre noget *senere*.

Vi skal lave to forskellige ting:

- En kylling der eksplodere efter 3 sekunder
- En ildmur der langsomt bevæger sig

Eksploderende kyllingebombe

Vi starter med at springe nogle kyllinger i luften.

Start med at lave en ny fil og og kommando:

```
exports.kyllingebombe = function() {
  echo("kyllinge bombe");
}
```

Start Minecraft og afprøv din nye kommando.

Nu skal vi først have skaffet os en kylling.

```
exports.kyllingebombe = function() {
  echo("kyllinge bombe");
1 var sted = self.location.subtract(0, 0, 0);
2 var verden = self.world;
3 var kylling = verden.spawn(sted, org.bukkit.entity.Chicken.class);
}
```

Vi har set kode der ligner det her før, men lad os lige gå det igennem.

1. Vi finder ud af hvor vi Minecraft verdenen vi står
2. Vi får fat i den verden vi står i. Vi skal bruge den til at lave en eksplosion lidt senere.
3. Vi laver en kylling

Sprængt kylling

Nu skal den kylling springes i luften.

```
exports.kyllingebombe = function() {
  echo("kyllinge bombe");
  var sted = self.location.subtract(0, 0, 0);
  var verden = self.world;
  var kylling = verden.spawn(sted, org.bukkit.entity.Chicken.class);

1 var kyllingPlacering = kylling.location;
2 verden.createExplosion(kyllingPlacering, 2);
3 kylling.remove();

}
```

1. Vi finder ud af hvor kyllingen står
2. Vi laver en eksplosion der hvor kyllingen er
3. Vi fjerner kyllingen (hvis den nu skulle have overlevet)

Afprøv din kommando i Minecraft.

Forsinket sprængning

Hov! Kyllingen springer i luften med det samme, det må vi lave om på.

Vi skal først pakke vores "spring kylling i luften" kode ind i en funktion. En funktion er bare en samling af kode-linjer som vi så kan bruge senere. Vi gør sådan her:

```
exports.kyllingebombe = function() {
  echo("kyllinge bombe");
  var sted = self.location.subtract(0, 0, 0);
  var verden = self.world;
  var kylling = verden.spawn(sted, org.bukkit.entity.Chicken.class);

  var boom = function() {
    var kyllingPlacering = kylling.location;
    verden.createExplosion(kyllingPlacering, 2);
    kylling.remove();
  };
  boom();
}
```

1 Her laver vi vores funktion. Vi skriver "function()" og de linjer der kommer mellem { og } er de kode linjer der bliver kørt når vi bruger vores funktion.

2 Vi laver også en variabel til vores funktion. Ligesom vi kan gemme tal i variable kan vi også gemme funktioner.

3 Vi kører vores funktion. Det gør vi ved at skrive navnet på den kasse (variabel) som vi har gemt funktionen i og så skriver vi () bagefter.

Afprøv det i Minecraft.

setTimeout

Ok, kyllingen springer stadig med det samme. Lad os fikse det:

```
exports.kyllingebombe = function() {
  echo("kyllinge bombe");
  var sted = self.location.subtract(0, 0, 0);
  var verden = self.world;
  var kylling = verden.spawn(sted, org.bukkit.entity.Chicken.class);

  var boom = function() {
    var kyllingPlacering = kylling.location;
    verden.createExplosion(kyllingPlacering, 2);
    kylling.remove();
  };
  setTimeout(boom, 5000);
}
```

1 Vi kan bruge "setTimeout" til at sætte et æggeur.

2 Når æggeuret "ringer" sker det vi har skrevet ind som 2. Bemærk at vi *ikke har skrevet () efter boom.* Vi har fortalt computeren: "Når æggeuret ringer så skal du køre det som ligger i kassen der hedder boom".

3 Det er her vi indstiller tiden på æggeuret. Vi indstiller tiden i millisekunder. Der går 1000 millisekunder på 1 sekund.

Afprøv det i Minecraft

Ildmur

Nu er det tid til ildmuren!

Vi starter ud med at lave en ny fil med en ny kommando

```
exports.ildmur = function() {
1 var standpunkt = self.location.subtract(0, 0, 0);
2 var block = standpunkt.block;
3 for (i = 0; i < 10; i++) {
4   var naesteBlock = block.getRelative(i, 0, 0);
5   naesteBlock.type = org.bukkit.Material.FIRE;
  }
};
```

1 Vi starter igen ud med at finde ud af hvor vi står

2 Så tager vi fat i den blok vi står på

3 Nu vil vi gerne lave en ild-mur der er 10 felter lang. Vi kunne bare gentage lidt kode ti gange, men nu ved vi jo hvordan man laver en for-løkke! For-løkken skal køre ti gange. En gang for hver blok vi vil lave om til ild.

4 Hver gang for-løkken kører skal vi lave den næste blok om til ild. Vi kan bruge "block.getRelative(i,0,0)" til at få fat i den blok der er "i" felter væk fra den første. Så først får vi den der er 1 felt væk, næste gange den der er 2 felter væk, så 3 felter...

5 Så laver vi feltet om til ild.

Afprøv det i Minecraft. Kan du lave en ild-mur der er 10 felter lang? Hvad med en der er 20 lang?

Flere ild-mure

Nu skal vi have ild-muren til at bevæge sig.
Vi starter med at pakke vores "lav ild-mur kode" ind i en funktion.

```
exports.ildmur = function() {
  var standpunkt = self.location.subtract(0, 0, 0);

1 var ildmur = function() {
    var block = standpunkt.block;
    for (i = 0; i < 10; i++) {
      var naesteBlock = block.getRelative(i, 0, 0);
      naesteBlock.type = org.bukkit.Material.FIRE;
2   }
    standpunkt.add(0, 0, 1);
  };
3 ildmur();
  ildmur()
```

1 Her pakker vi vores ild-mur kode ind i en funktion og gemmer det i variablen "ildmur".

2 Den er linje kode er ny. Vi flytter det sted hvor ildmuren starter. Det betyder at ild-muren flytter sig et felt hver gang vi kalder vores "ildmur" funktion.

3 Vi laver to ildmure.

Afprøv i Minecraft. Kom der to rækker ild?

Flyt ild-muren

Lagde du mærke til hvor hurtigt den anden række ild kom? Vi vil gerne have muren til at flytte sig *langsomt*. Det kan vi med "setInterval". "setInterval" virker lidt ligesom "setTimeout" men der er en vigtig forskel.

"setTimeout" kørte vores funktion én gang når "æggeuret ringende".

"setInterval" kører vores kode flere gange med en pause i mellem. Skriver vi "setInterval(ildmur, 2000)" vil den kører funktionen "ildmur" hvert andet sekund. Eller sagt på en lidt anden måde: "ildmur" bliver kørt med et interval på 2 sekunder.

Prøv om du kan få “ildmuren” til at flytte sig med “setInterval”.

Stop, stop, stop

Hele verden står i flammer. Hvordan stopper jeg min ildmur?

Du kan køre “/js refresh()” for at stoppe din ildmur i at flytte sig (og du kan bruge regnvejr til at slukke den med).

Eksperimenter

Eksperiment #1

Lav en kommando der laver en ild-mur hele vejen omkring dig.

Kan du lave din kommando så ild-muren stadig er omkring dig, men nu er fire felter væk hele vejen rundt?

Har du brugt for-løkker til at lave muren?

Eksperiment #2 (svær)

Kan du lave en ild-mur der langsom bevæger sig ind mod dig?

Eksploderende kyllinger

Multiplayer

Noget med at lave mulitplayer mods

Appendiks: Blok typer

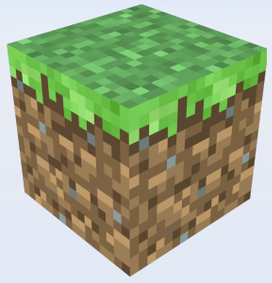
Hvis du vil bruge forskellige blok-typer i dit Minecraft program, så er her en liste med nogle af dem du kan bruge.

Ud for hver blok står der den tekst du skal skrive for at bruge blokken.

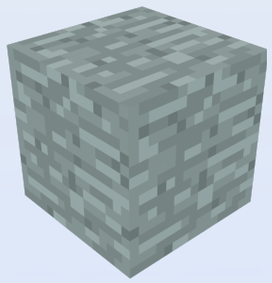
Skal du bruge diamant skal du skrive "`blocks.diamond`".

F.eks:

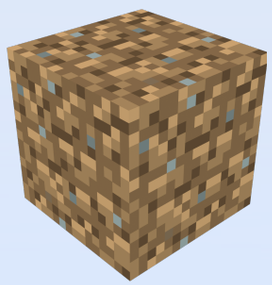
```
/js box(blocks.diamond, 5, 5, 5)
/js box(blocks.oak, 5, 5, 5)
```



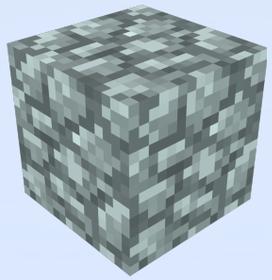
blocks.grass



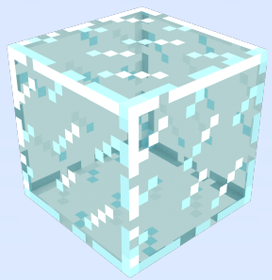
blocks.stone



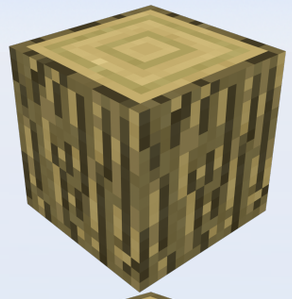
blocks.dirt



blocks.cobblestone

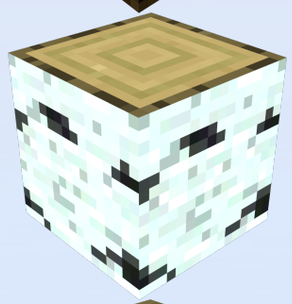


blocks.glass

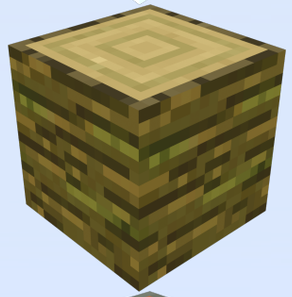


blocks.oak

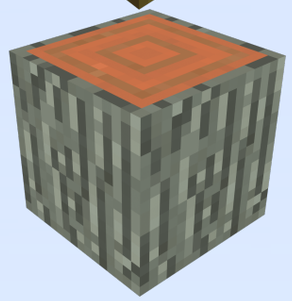
blocks.spruce



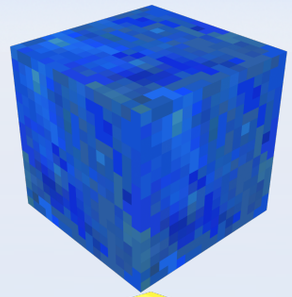
blocks.birch



blocks.jungle



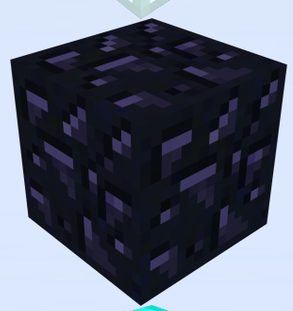
blocks.acacia



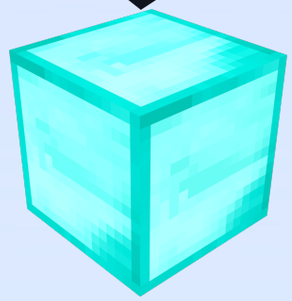
blocks.lapis_lazuli_block

blocks.gold

blocks.iron



blocks.obsidian



blocks.diamond

Appendiks:Bygge-robotten

Vi bruger Bygge-robotten rigtigt meget i denne bog og på de følgende sider kan du se en liste over de kommandoer som Bygge-robotten forstår.

Husk at du altid kan afprøve kommandoerne direkte i Minecraft-konsollen. Du skal blot huske at skrive "`/js `" foran kommandoen.

Appendiks: Nyttinge kommandoer

Diverse noter

Tip: Kopier tekst

Når du ser et kode eksempel her i bogen kan du kopiere teksten over i din kode-editor. Marker teksten, hold Ctrl-tasten nede og tryk på “C”. Gå så over i din kode-editor, hold Ctrl-tasten nede og tryk på “V”.