[Micro:bit]

av

[Carlos Aristondo]

INNEHÅLLSFÖRTECKNING

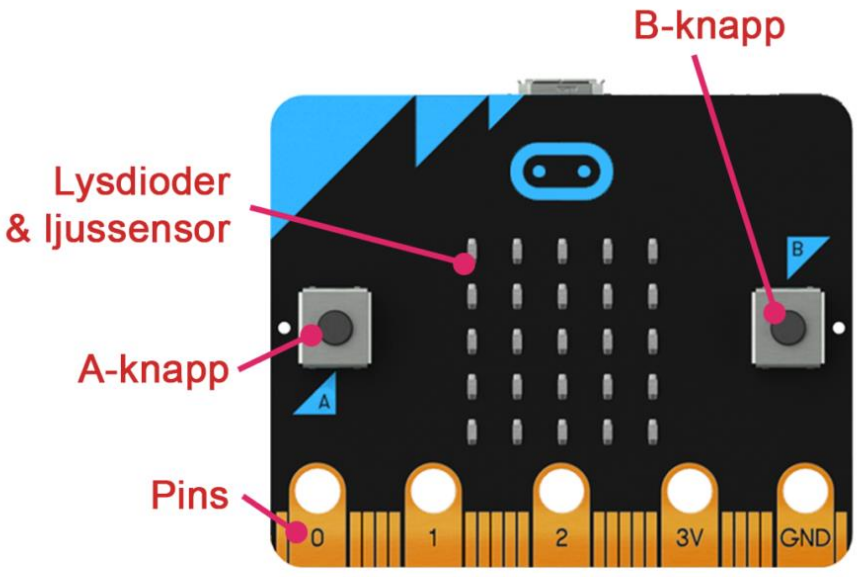
LEKTION 1 SEKVENS

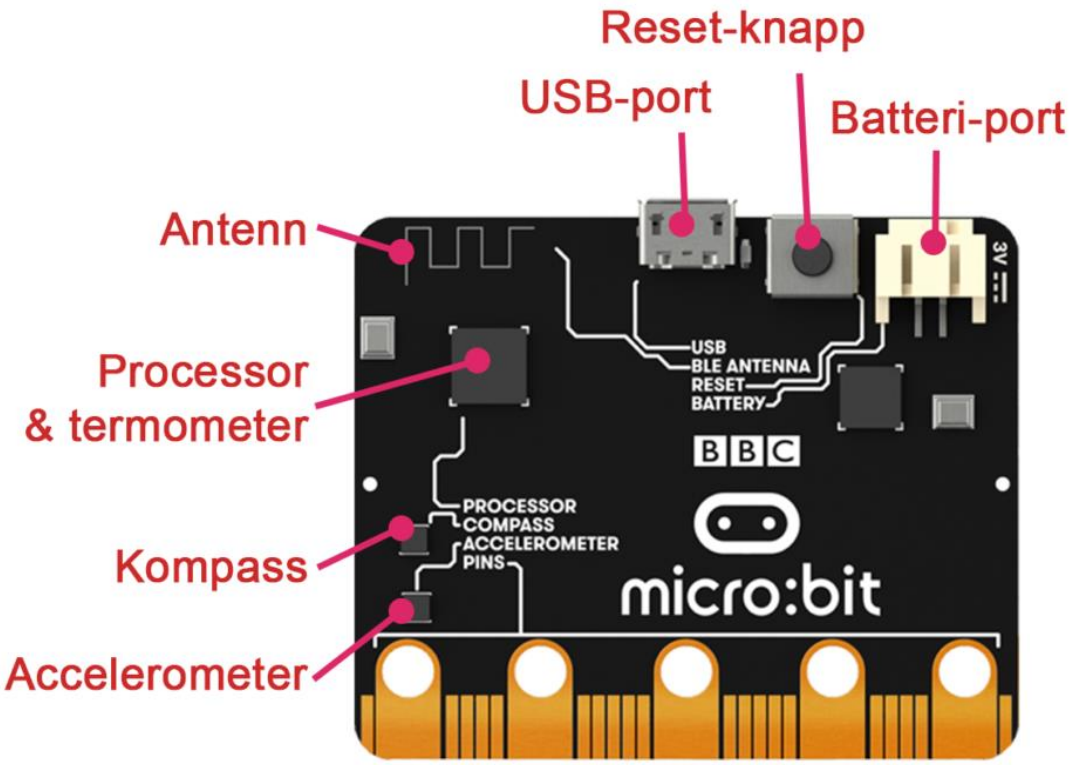
Vad är micro:bit?

Micro:bit är en liten dator som man enkelt kan få saker att göra genom att programmera den.

Vad finns på en micro:bit?

På framsidan hittar vi två **knappar**, markerad med bokstav A och B. Dom har ingen förutbestämd funktion; du får själv hitta på vad ska hända när dom trycks i. Emellan knapparna finns **25 LED lampor** (ljusdioder) som har en röd färg när dom är tända. Varje LED kan styras separat, vilket gör att det kan användas som ett skärm som visar animationer och texter. LED lamporna används också på ett innovativt sätt som en **ljussensor**, som känner av hur mycket ljus som faller på din micro:bit. Längst ner är kanten täkt med metall remsor (markerad med 0, 1, 2, 3V och GND). Detta är **pins**, vilken används för att koppla mer elektronik till micro:bit.





På baksidan finns flera svarta fyrkant, oftast med förklarande text i vit. Den största, uppe i vänster, är **processorn**. Det är där ditt program utförs. I den finns en inbyggd **termometer**. Den är egentligen tillför att stänga ner processorn när den jobbar för hårt och blir för varmt, men det coola är att vi får tillgång till den i vår programmering. Så kan man till exempel programmera att en fläkt går igång när en viss temperatur är nådd. Lite längre ner hittar vi ett mindre fyrkant som är markerad med **kompass**. Det är en sensor som känner av magnetiska krafter och hitta i vilken riktning det magnetiska norden ligger. Direkt under den finns en **accelerometer**. Den mäter, som namnet kanske avslöjar, acceleration i tre riktningar. Med den kan du få saker att hända till exempel när du skakar micro:bitten. En accelerationskraft som alltid finns på jorden är tyngdkraften. Genom att känna av i vilken riktning den är kan ditt program veta hur du håller micro:bitten.

Längst upp till vänster, om man tittar väldigt noggrann, kan du se ett streck som sicksackar som en orm. Det är en **antenn** med hjälp av vilken flera micro:bit trådlöst kan kommunicera med varandra. Via den kan din micro:bit även kommunicera över **bluetooth** till en

smartphone eller surfplatta. Du kan till och med programmera din micro:bit från en smartphone eller surfplatta via bluetooth.

I mitten längst upp finns en **micro USB-port** där micro:bit kopplas till en dator via en USB-sladd, för att föra över kod och ge ström. Så länge micro:bit får ström kör den det sista programmet som laddades ner till den. Till höger sitter en **reset-knapp**. När du trycker på den startar programmet på nytt från början. Längst till höger finns en **batteri-port** där ett batteripaket kan kopplas in, som en alternativ strömkälla till USB-sladden.

JavaScript Block editor

JAVASCRIPT BLOCK EDITOR TIPS

Här hittar du många olika tips kring JavaScript Block editorn som är bra att veta om. Kolla även på allmänna micro:bit tips.

HANTERA BLOCK

När du ska flytta block, kolla så att muspekaren ändras till en hand.

För att dra sig runt på arbetsytan kan du klicka på en tom plats och dra med musen.

Kom ihåg att blocken körs i den ordning som de staplas på varann. Överst först.

Om ett block är utgråat betyder det att det inte är kopplat på rätt sätt, och inte kommer att köras.

Du kan ta bort block på tre sätt: dra åt vänster till soptunnan, högerklick på blocket och välj ta bort, eller selektera med vänsterklick och tryck på delete tangenten.

Några block, som till exempel ”om annars”, har en liten blå kuggikon, för att ändra något på blocken.

Lägg märke till knappen ”Mer…” som visas under vissa kategorier, för att hitta fler block inom denna kategori.

Om det är rörigt på arbetsytan kan du högerklicka och på en tom yta och trycka ”formatera kod”.

Om du är osäker hur ett block fungerar kan du högerklicka på det och trycka på hjälp.

Simulatorn

Om du vill se körningen av ditt program i realtid kan du använda snigelknappen för att följa med i hur koden körs.

Olika

Tänk på att decimaltal inte fungerar i editorn. Och att division avrundas nedåt. (5/3=1)

”Vid start”-blocket finns inte i JavaScript. Det hamnar utanför ”forever”-loopen.

SPARA PROJEKT

Dina projekt blir automatiskt sparade i lokala minnet av din webbläsare. Detta betyder att så länge du använder samma dator och samma webbläsare för att surfa till editorn, du kommer ha kvar din kod. För att behålla ditt program och börja med en ny, tryck på ”projekt”-knappen och ”Nytt projekt”. Här hittar du också alla dina projekt sorterade på när du sist ändrade dom. För att göra det lättare hålla koll kan du ge ett projekt ett namn när den är öppen, i fältet längst ner i mitten. Det räcker med att skriva namnet, du behöver inte trycka på ”spara”-knappen bredvid.

DELA PROJEKT

Om du vill dela ditt program med någon annan, eller ta det med dig till en annan dator finns det två sätt. Det först är att filen (som slutar med .hex) du får när du trycker på ”Ladda ned”-knappen kan du spara och flytta som vanliga filer på en dator. För att öppna dom är det bara att dra ock släppa dom på editorn igen. En detalj här är att det bara funkar med dom .hex filerna dom är skapade med denna editor. Andra sättet att dela med sig är att trycka på ”dela”-knappen uppe till vänster. Efter att du har godkänd att publicera projektet får du en länk som du kan sprida. Alla som går in på länken får en

kopia av ditt projekt. (Alla exempel på denna webbsida har en sådan länk under sig!)

VARIABLER

En variabel är en plats i minnet av din dator som man ger ett namn, och som innehåller ett värde. Värdet i variabeln kan ändra på sig medans ditt program körs, och den kan bli använd på flera platser i koden.

SOM EN LÅDA

Variabler jämförs ibland med lådor. Tänk dig en flyttkartong som man använder för sin julpynt. På lådans utsidan skriver man ”julpynt” så att man vet van den innehåller. I kartongen kanske du lägger 10 stycken julkullor, och ställer undan den på vinden. När det är jul går du upp och hämtar lådan ”julpynt”. Innehållet använder du för att dekorera julgranen, men du råkar tappa en kula i golvet. Så när det blir dags att plocka undan blir innehållet av lådan ändrat till bara 9 stycken. På liknande sätt har en variabel en namn som beskriver vad den är till för, och ett innehåll som kan användas och förändras.

NAMNGIVNING

Det namnet man ger till sin variabel spelar ingen roll för datorns skull. Så det är bra att använda ett namn som beskriver vad variabeln innehåller. Till exempel kan man ha en variabel poäng som innehåller hur mycket poäng man har samlat i ett spel. Namnet får bestå av flera bokstäver och siffror. I JavaScript läge får den inte innehålla mellanslag. Om man ändå vill ha flera ord i namnet kan man vara lite kreativt med till exempel stora bokstäver eller understreck: detSlumpadeTaleteller det_slumpade_talet.

NÄR BEHÖVS DET?

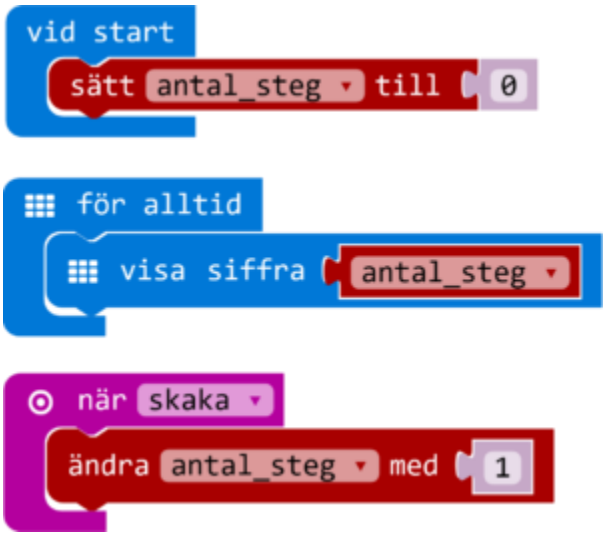
Variabler används när en bit av information ska användas vid ett senare moment i programmet än där den skapas, ofta handlar det om något som ska användas mer än en gång. En viktig egenskap är också att informationen kan ändras medans programmet körs.

STEGRÄKNARE

Du kan också titta på ett Youtube klip av MakerMovies.se *om hur man bygger en stegräknare och använder en variabel.*

För att hålla reda på informationen hur många steg vi har gått behöver vi en variabel. Gå in till kategorin Variabler och tryck högst upp på den gråa knappen ”Skapa en variabel” och fyll i namnet. Eftersom den ska räkna upp antal steg vi har tagit kallar vi den för antal_steg.

Innan vi kan använda variabeln ska vi se till att den innehåller något. Dra ut ”sätt namnlös till 0”-blocket till ”vid start” och byt ”namnlös” till namnet på variabeln du precis har skapat. Nu är vi säker att den alltid börjar på 0 när programmet startar. (Om jag är ärlig: i blockläge görs detta i bakgrunden även om man inte lägger ut detta block. Men det är bra att vänja sig med att göra det innan man byter till textprogrammering.) Nu vill vi att varje gång vi tar ett steg, och micro:bit skakas, antal_steg ökas med 1. Då drar vi ut ”när skaka”-blocket från Input och i den lägger vi ”ändra med 1” från Variabler. Se till att ändra det till antal_steg variabeln. Nu håller vår micro:bit koll på hur många steg vi tar. Men.. vi ser ingenting. Det sista vi ska lägga till är att visa upp värdet som finns i vår variabel. För att få det synligt hela tiden bygger vi det i ”för alltid”-blocket. Dra in ett ”visa siffra”-block. Istället för en 0 ska vi visa upp värdet som finns i antal_steg. Den hittar vi som ett block i Variabler kategorin.




```
let antal_steg = 0
antal_steg = 0
input.onGesture(Gesture.Shake, () => {
 antal_steg += 1
})
basic.forever(() => {
 basic.showNumber(antal_steg)
})
```

OLIKA TYPER

Vår stegräknare har en variabel som innehåller en siffra. Men det finns också andra typer av information som kan sparas i en variabel. Detta kallas för **datatyp**. Strängar (textbitar) är en annan typ. Till exempel kan man ha en variabel som heter användernamn och innehåller texten "andersson007". En annan datatyp möter man när man håller på med om-satser och villkor. Den heter boolesk (boolean) och innehåller antingen sant eller falskt. Listor är ännu en annan typ.

KOMMENTAR

Kommentar är en eller flera rad text i programkod som ignoreras av datorn, och är bara där för att underlätta för människor att förstå vad koden gör.

I JavaScript blir all text som står till höger av två snedstreck (//) uppfattad som kommentar. Om man vill ha flera rader av kommentar kan man skriva det mellan /* och */.

```
// Detta är en rad av kommentars text.


// Man kan ha flera rader.
// Så länge dom börjar med dubbla snedstreck.


/* Detta är flera rader av kommentar.
   Här kan du förklara vad ditt program gör.
   Det är bra för dig själv och andra som
   ska försöka förstå din kod. */


basic.showNumber(1)
basic.showNumber(2) // Man kan ha kommentar till höger av kod på det här sättet
basic.ShowNumber(3)
```

Operatorer

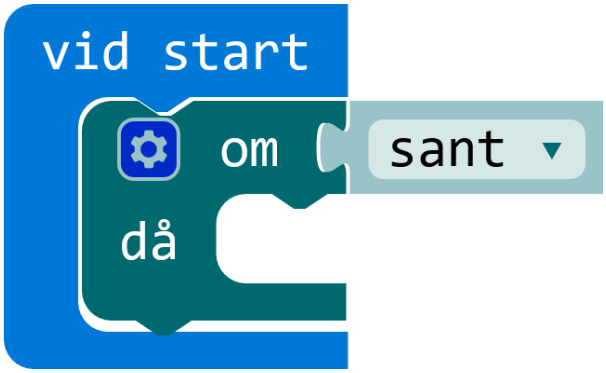
```
Var antalBarn
++   antalbarn++    // antalBarn = antalBarn + 1
+=   antalBarn +=2  // antalBarn = antalBarn + 2
--   antalbarn--    // antalBarn = antalBarn - 1
-=   antalBarn -=2  // antalBarn = antalBarn - 2
*=   antalBarn *=2  // antalBarn = antalBarn * 2
/=   antalBarn /=2  // antalBarn = antalBarn / 2
%=     tal = 5%2      blir 1 resten av division.
```

Uppgift 3: skapa ett blockprogram räkna antal barn som har kommit på studiebesök hos dig!

SELEKTION (IF-SATS)

Selektion används när någonting ska hända beroende på ett villkor.

Ofta vill man ha ett vägval i sitt program, där en del kod ska köras bara i vissa fall. Då kan man använda sig av en if-sats (om-sats på svenska). Sådana satser kan ha lite olika former, beroende på hur många fall du vill välja mellan. Du hittar den i kategorin Logik. I den minsta formen ser den ut så här i block och text:




```
if (true) {


}
```

På platsen där det står platshållare ”sant” (true) ska du bygga villkoret, som man kan se som en fråga som ställs till datorn (”Är detta villkor sant?”). Om svaret på frågan blir att det är sant, då kommer koden som ligger på insidan, till höger av ”då”, köras. Om det inte är sant, händer det inte. Block för att bygga villkoret med (t.ex. lika med, mindre än) hittar du i samma kategori, Logik.

LOGISKA OPERATORER

Villkor

```
== ( lika med)

!= (ej lika med)
```

```
>= större och lika med
<= mindre och lika med
&&  ( och)
||     (eller)
```

Parenteser

```
()
() => {}
```

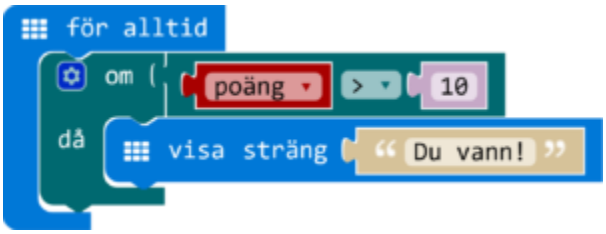
Klammerparenteser/måsvingar

```
{}
```

Hakparenteser

```
[]
```

EXEMPEL

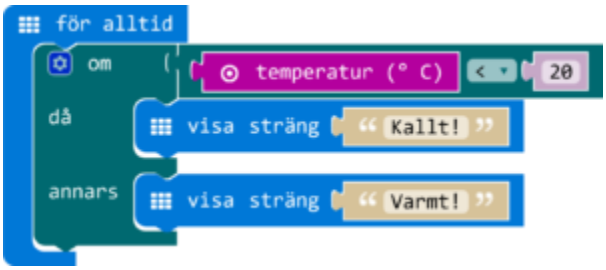



```
if (poäng > 10) {
    basic.showString("Du vann!")
}
```

I det här exempel ställer vi frågan ”Är värdet av variabeln poäng mer än 10?” och om det är sant, då visas texten ”Du vann!” på skärmen.

UTÖKAD FORM

Svaret på frågan (villkoret) blir alltid antingen SANT eller FALSKT. Om man vill att något annat händer när svaret blir falskt, kan man utöka om-då till om-då-annars (if-then-else). Det kan se ut såhär:

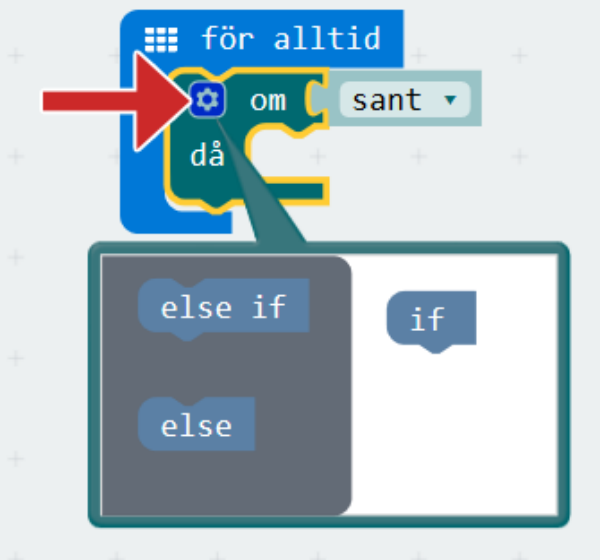


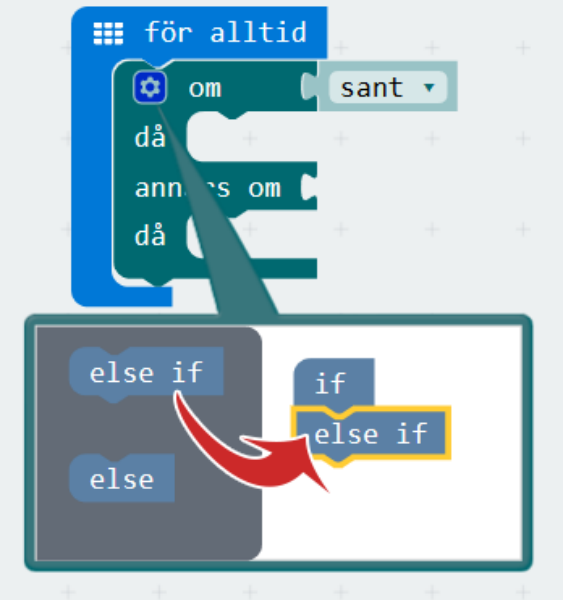

```
if (input.temperature() < 20) {
    basic.showString("Kallt!")
} else {
    basic.showString("Varmt!")
}
```

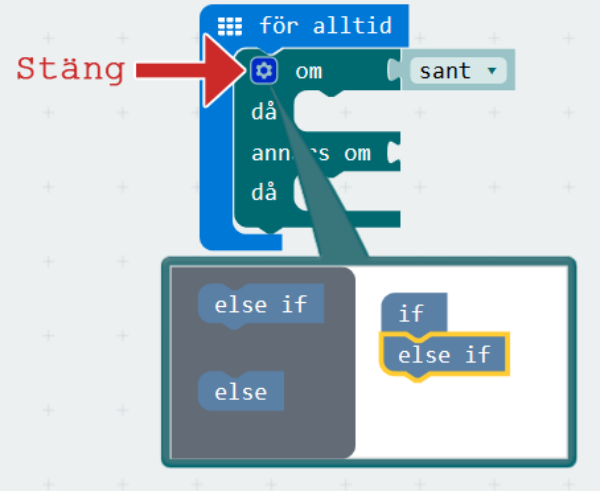
Här visas antingen text ”Kallt!” eller ”Varmt” på skärmen, beroende på om temperaturen är mindre än 20 grader (då / then) eller inte (annars / else).

KOPPLA IHOP FLERA FRÅGOR

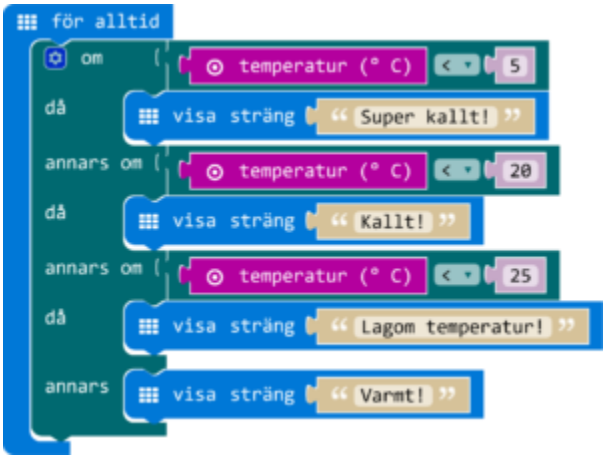
Ibland vill man ha fler än bara två läge, och då kan man skilja mellan dom med hjälp av flera sådana frågor (villkor). Det gör man genom att ha ett eller flera ”annars om” (else if) delar som direkt ställer en ny fråga. För att lägga till det i block läget ska man trycka på det lila, blåa kuggikonen som finns på om-då blocket. Då kommer upp ett litet fönster där du kan pussla till delar till och från sitt om-då block för att ändra den. Efter det stänger man den igen genom att trycka på samma blåa ikon.


för alltid
om
sant
då
else if
else
if


för alltid
om
sant
då
else if
else
if
else if


Stäng
för alltid
om
sant
då
else if
else
if
else if

På det här sättet kan man till exempel bygga följande program, som berättar i mer detalj vilken temperatur det är:

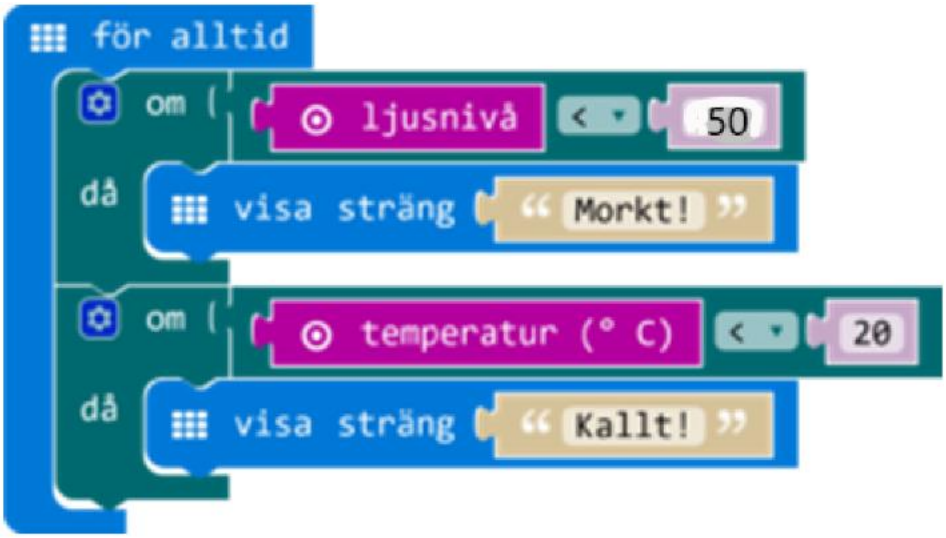



```
if (input.temperature() < 5) {
    basic.showString("Super kallt!")
} else if (input.temperature() < 20)
    basic.showString("Kallt!")
} else if (input.temperature() < 25) {
    basic.showString("Lagom temperatur!")
} else {
    basic.showString("Varmt!")
}
```

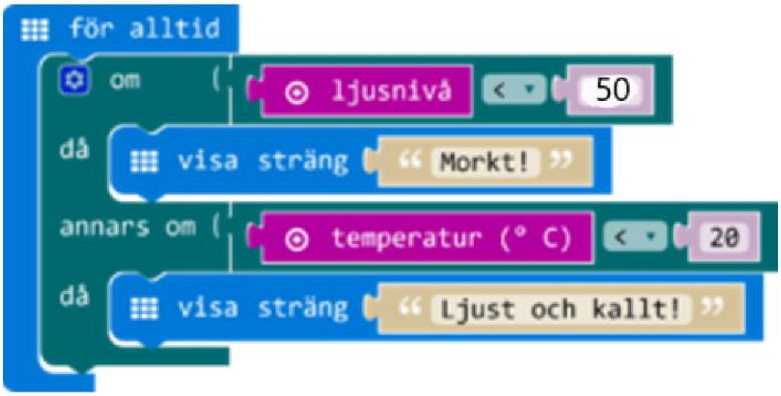
Nu, om det är mindre än 5 grader visas ”Super kallt!”, annars om det är minder än 20 visas ”Kallt!”, annars om det är mindre än 25 visas ”Lagom temperatur!”, annars visas ”Varmt!”

VAD ÄR SKILLNADEN?

Nu tänker du kanske, varför ska jag ha ”annars om” om jag också bara kan duplicera min första ”om då” block, och klistra flera efter varandra? Det finns en skillnad, vilket är viktigt om de olika läge vi skiljer mellan inte uteslutar varandra. Titta på följande exempel, i två varianter. Vad kommer bli resultatet av båda?




```
if (input.lightLevel() < 50) {
    basic.showString("Morkt!")
}
if (input.temperature() < 20) {
    basic.showString("Kallt!")
}
```



```
if (input.lightLevel() < 50) {
    basic.showString("Morkt!")
} else if (input.temperature() < 20) {
    basic.showString("Ljust och kallt!")
}
```

I första varianten är det två frågor som ställs efter varandra. Först: Är det mörkt? Då visa texten ”Morkt!”. Sen: Är det kallt? Då visa texten ”Kallt!”. Resultatet på frågorna hänger inte ihop. Så man kan få båda texter, ifall det är båda mörkt och kallt. Eller så får man en av dom, eller ingen alls.

I andra varianten blir frågan om temperatur bara ställd om svaret på den första var falskt (alltså inte mörkt). Så antingen är det mörkt och får man texten ”Morkt!”, eller, om det inte är mörkt och det är kallt får man texten ”Ljust och kallt!”, eller så får man ingen text.

VILLKOR

Villkor kan man se som frågor som alltid är antingen sant(true) eller falskt(false). Dom används till exempel i om-satser och loopar.

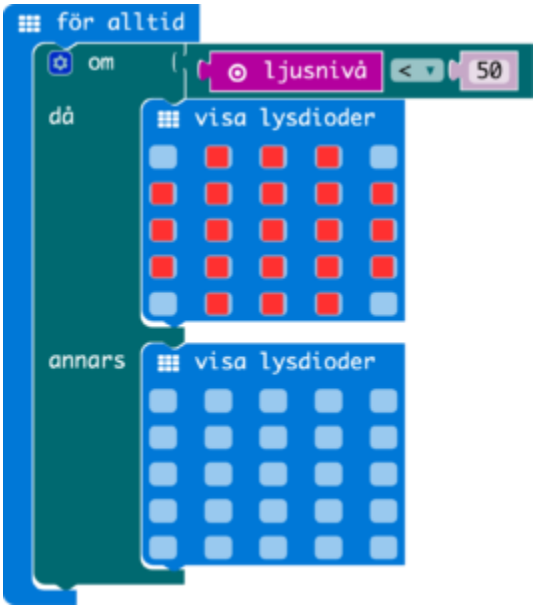
Det finns flera block som man kan kombinera för att bygga stora villkor med. Dom hittar du i kategorin Logik. I JavaScript blir dom översatta till olika tecken.

JÄMFÖRA

För att jämföra finns det följande block/tecken i kategorin Logik. Alla dessa jämförelser ger som resultat antingen sant eller falskt.

Block	JavaScript	Beskrivning
	==	Lika med
	!=	Inte lika med
	<	Mindre än
	<=	Mindre än eller lika med
	>	Större än
	>=	Större än eller lika med

Med sådana jämförelser kan vi till exempel bygga en lampa som tänds automatiskt om det ljusnivå som mäts blir **mindre än** en viss gräns.

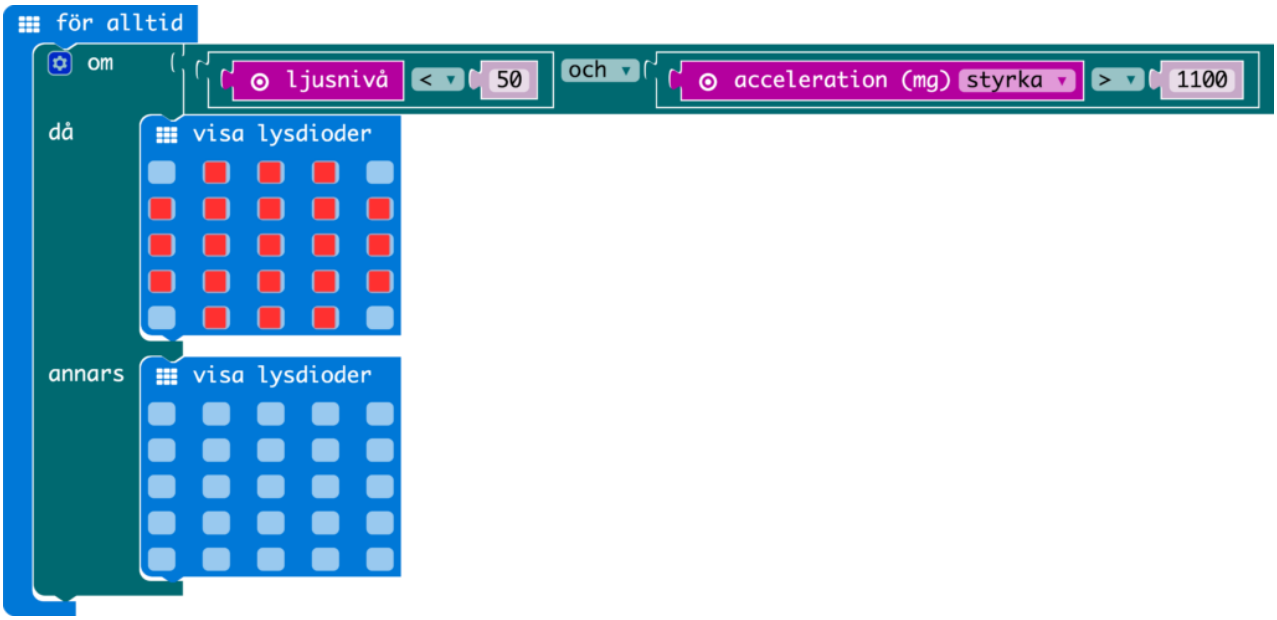



```
basic.forever(() => {
    if (input.lightLevel() < 50) {
        basic.showLeds(`
            . # # # .
            # # # # #
            # # # # #
            # # # # #
            . # # # .
            `)
    } else {
        basic.showLeds(`
            . . . . .
            . . . . .
            . . . . .
            . . . . .
            . . . . .
            `)
```

```
  }
})
```

Kombinera

För att kombinera flera av dom finns två följande block: ”och” och ”eller”. I JavaScript skrivs dom som && (och) och || (eller). Då kan man ändra föregående exempel till en lampa som bara tänds om det är mörkt **och** micro:bitten rörs samtidigt.

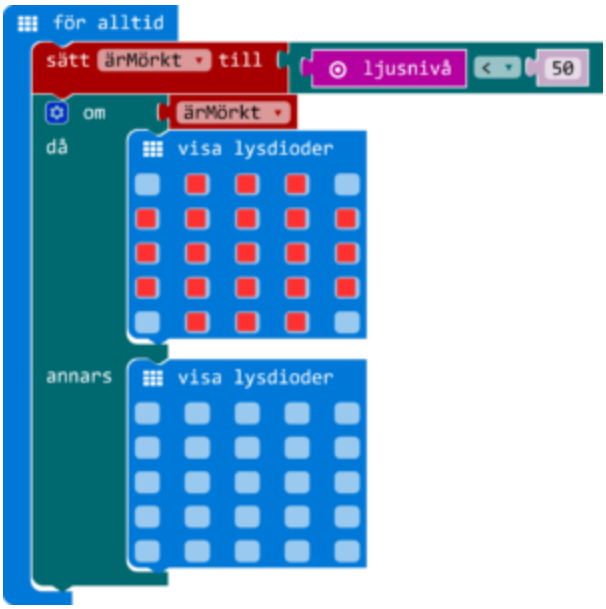


```
basic.forever(() => {
    if (input.lightLevel() < 50 && input.acceleration(Dimension.Strength) > 1100) {
        basic.showLeds(`
            . # # # .
            # # # # #
            # # # # #
            # # # # #
            . # # # .
            `)
    } else {
        basic.showLeds(`
            . . . . .
            . . . . .
            . . . . .
            . . . . .
            . . . . .
            `)
    }
})
```

BOOLESK

Det svaret vi får från block som ”<”, vilket är antingen sant(true) eller falskt(false), kallas för boolesk (boolean på engelska). Vi kan spara ett sådant värde i en variabel. Den kallas då för en boolesk variabel.

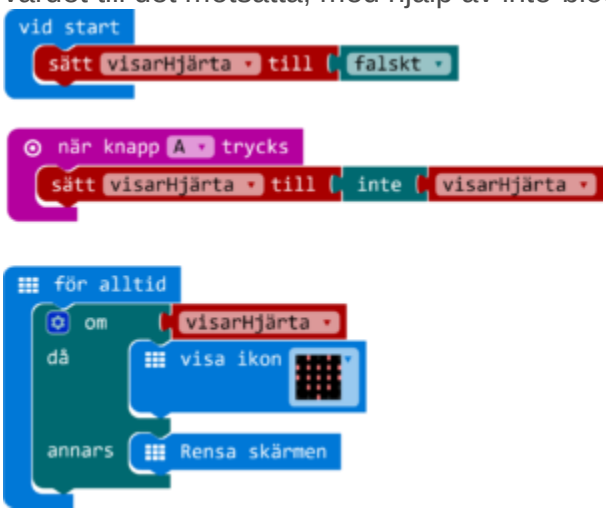
Om vi bygger om vår första exempel kan vi använda en variabel som heter ”ärMörkt”. Den innehåller sant(true) om ljusnivån är under 50, eller falskt(false) om det är 50 eller mer. Därefter, om det ärMörkt, tänd lamporna. I den om-sats kollar vi vilket värde finns i variabeln, och utför olika kod beroende på den.




```
let ärMörkt = false
basic.forever(() => {
    ärMörkt = input.lightLevel() < 50
    if (ärMörkt) {
        basic.showLeds(`
            . # # # .
            # # # # #
            # # # # #
            # # # # #
            . # # # .
            `)
    } else {
        basic.showLeds(`
            . . . . .
            . . . . .
            . . . . .
            . . . . .
            . . . . .
```

```
        `)
    }
})
```

Det sista tre block som finns under Logik (inte, sant, falskt) kan användas ihop med den booleska logiken som vi har lärt oss. I det följande exemplet har vi en variabel ”visarHjärta” som börjar med värdet falskt. Sen, varje gång man trycker på A knappen byter den värdet till det motsatta, med hjälp av inte-blocket.

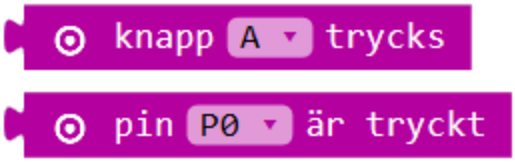



```
let visarHjärta = false
input.onButtonPressed(Button.A, () => {
    visarHjärta = !(visarHjärta)
})
basic.forever(() => {
    if (visarHjärta) {
        basic.showIcon(IconNames.Heart)
    } else {
        basic.clearScreen()
    }
})
visarHjärta = false
```

ANDRA BLOCK SOM GER BOOLESKT SVAR

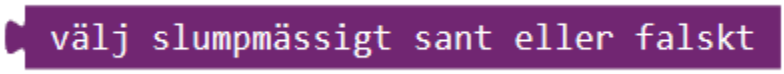
Förutom Logik kategorin och variabler finns det några andra block som också ger ett boolesk värde.

Under Input finns två block som ger sant om en viss knapp eller pin är tryckt just nu.



Under Leds finns ett block som svarar sant om ljusdioden med denna koordinater är tänd.

Under Matematik hittar du ett block för att få slumpmässigt sant eller falskt.



Uppgift 4:

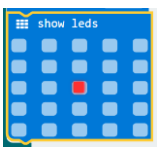
Skriv ett blockprogram som visar en fyrkant om ljudnivån är större än 50 (>)

Annars visar en hjärta.

Uppgift 5:

Skriv ett blockprogram som kastar en tärning m h a skaka micro:biten. Tärningsvärdet ska visas på ljusdioder, en i taget, Du kan rita själv dina tärningsögon med hjälp av ljusdioderna.

T ex en etta



Uppgift 6:

Skriv ett blockprogram som simulera spel sten, sax och påse. Rita sten, sax och påse med hjälp av ljusdioderna.

Tips:

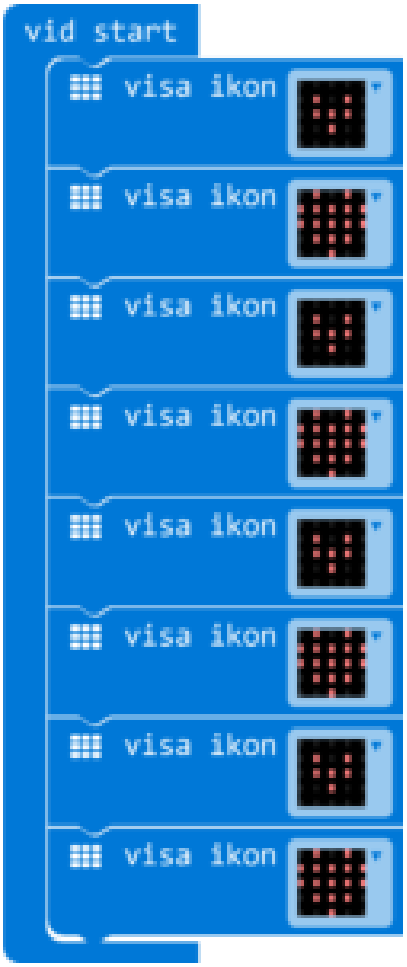
Se länken.

https://makecode.microbit.org/#lang=en

ITERATION (LOOPAR)

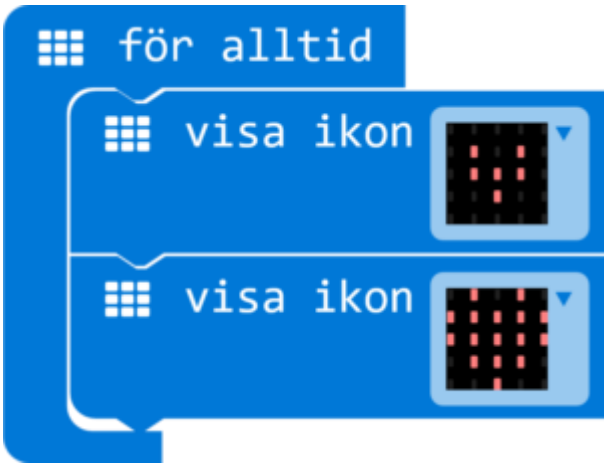
Iteration är ett annat ord för upprepning. Detta använder man när någonting ska utföras flera gånger. Det finns olika sätt att bestämma hur många gånger koden ska upprepas.

Datorer är väldigt bra på att upprepa samma sak många gånger. Dom tröttnar inte på det som vi människor ofta gör. Till exempel om jag vill visa en animation av ett hjärta som slår går det bra med en upprepning av två bilder.



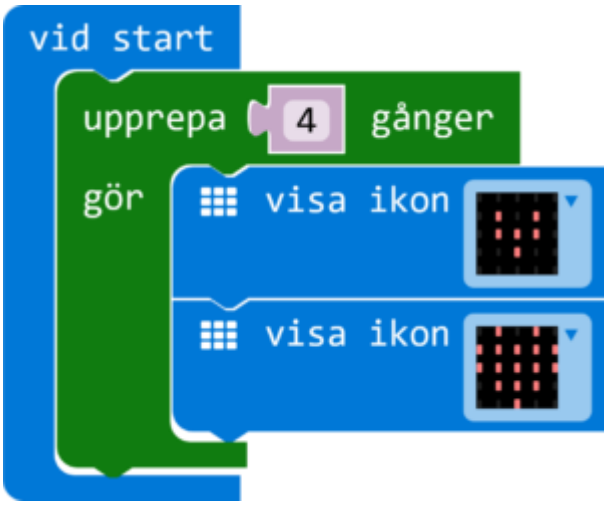
Men oj vad många block det blev. För ett hjärta som slår fyra gånger blev det nio block att pyssla ihop. Kan vi inte göra det enklare? Javisst!

Ett enkelt sätt att få det att hända många gånger är att lägga det i ”för alltid”-blocket (kategori Grundläggande). Då behöver vi bara två ikon-block som upprepas för alltid.



ETT BESTÄMD ANTAL GÅNGER

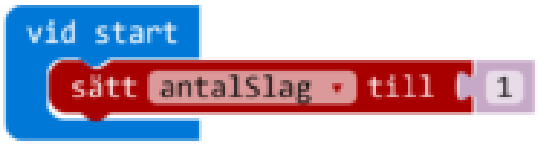
Men om vi nu inte vill att det upprepas för alltid, men ett bestämd antal gånger? Då finns det också ett block för det, och den hittar vi kategori Loopar. ”upprepa 4 gånger, gör” är ett block som utför allt i den, så många gånger som står i det gråa fältet. Så denna kod gör precis samma som det första exemplet, men använder fem färre block. Och vi kan super enkelt ändra hur många gånger det ska upprepas.

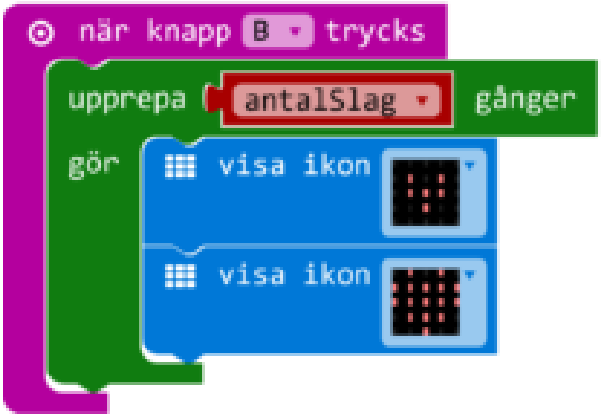


BEROENDE PÅ EN VARIABEL

Att enkelt ändra på hur många gånger hjärtan slår är redan smidigt, men det blir ännu bättre när man kan ändra på det medans

programmet körs. Följande program använder sig av en variabelsom heter ”antalSlag”. Den börjar med värdet 1, och ökar varje gång man trycker på A-knappen. När man sen trycker på B-knappen slår hjärtan så många gånger som just då ”antalSlag” innehåller för värde. I ”upprepa”-blocket ser vi att siffran fyra är ersätt med variabel blocket, som ger det värdet som finns i variabeln.

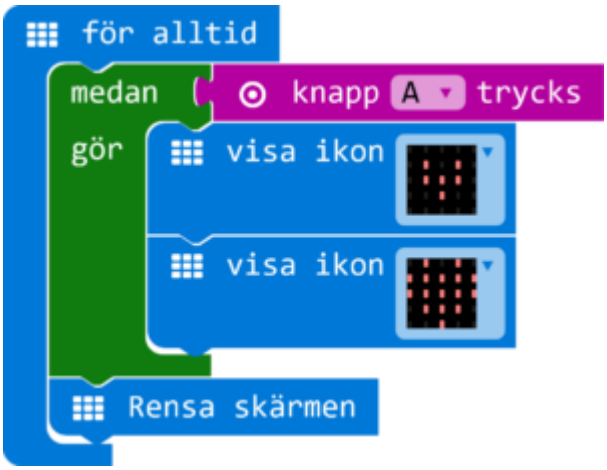





```
let antalSlag = 1
input.onButtonPressed(Button.A, () => {
    antalSlag += 1
    basic.showNumber(antalSlag)
})
input.onButtonPressed(Button.B, () => {
    for (let i = 0; i < antalSlag; i++) {
        basic.showIcon(IconNames.SmallHeart)
        basic.showIcon(IconNames.Heart)
    }
})
```

BEROENDE PÅ ETT VILLKOR

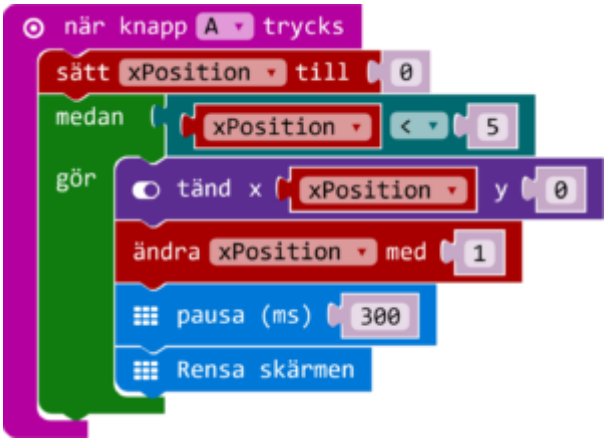
Ett annat sätt att få kod att upprepas ett visst antal gånger är med hjälp av villkor. Med ett ”medan sant”-blocket (while true) kommer allt inuti hända så länge villkoret (som kommer istället för platshållaren ”sant”) resulterar i svaret sant. I följande exempel slår hjärtat bara så länge som A-knappen är nertryckt (villkoret ”knapp A trycks” är sant).




```
basic.forever(() => {
    while (input.buttonIsPressed(Button.A)) {
        basic.showIcon(IconNames.SmallHeart)
        basic.showIcon(IconNames.Heart)
    }
    basic.clearScreen()
})
```

FLYTTA EN PIXEL

En annan kul grej vi kan göra med upprepning baserad på villkor är att använda en variabel i det villkoret. Värdet av variabeln kan ändra på sig när loopen körs, så att någon gång villkoret blir falskt och loopen slutar. I nästa koden kommer vi, när Knapp A trycks, tända en LED lampa på skärmen, med en viss x- och y-koordinat. X-koordinaten styrs av en variabel (”xPosition”), som varje gång börjar på `0` och ökar med `1` varje genomgång av loopen. Kan du lista ut vad x kommer innehålla som högsta värde?




```
let xPosition = 0
input.onButtonPressed(Button.A, () => {
    xPosition = 0
    while (xPosition < 5) {
        led.plot(xPosition, 0)
        xPosition += 1
        basic.pause(300)
        basic.clearScreen()
    }
})
```

Vi ser att `xPosition` börjar med `0`, det är för att koordinaterna på micro:bit börjar räkna med 0. I varje iteration visas pixeln, och höjs `xPosition` med `1`. Sen upprepas det igen, om `xPosition`är mindre än `5`. Så den sista pixel som blir tänd har x-koordinat `4`. Efter det höjs `xPosition` till `5`, men när då villkoret blir kontrollerat en gång till är det `falskt` (5 är inte mindre än 5), och är koden färdigt.

KAN DET BLI KORTARE?

Den här typen av loop, där en variabel skapas, används inom loopen, ändras på varje steg, och är del av villkoret, används ganska ofta inom programmering. Det är därför det finns en variant av loopar som gör precis detta. Den kallas ”för loop” (for loop) och ser ut så här:


```
for (let plats = 0; plats <= 4; plats++) {


}
```

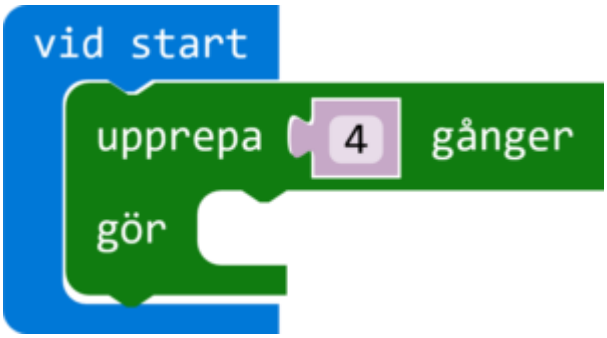
Standard kommer den med en variabel som heter ”plats”, vilken kommer börja med värdet 0, och ökar varje loop med 1, till och med värdet 4 (den innehåller 4 i sista genomgång).

I JavaScript heter loopen ”for” och har den tre delar inom sina parenteser, som är separerad med `;`. Första delen är där variabeln är skapad och får sitt första värde (`let plats = 0`). Andra delen är villkoret (`plats <= 4`).

Här ser vi att det är ”mindre än eller lika med”, vilket gör att 4 är inkluderad som sista genomgång av loopen. Sista delen är vad som ska hända efter varje genomgång, vilket i det här fall är öka plats med 1 (`plats++`). ”`plats++`” är bara en förkortad sätt att skriva `plats = plats + 1`.)

Genom att ändra på dom tre delar kan man få loopen att genomföras på många olika sätt. Detta är ett tydligt fall där JavaScript ger mer möjligheter än blockprogrammering.

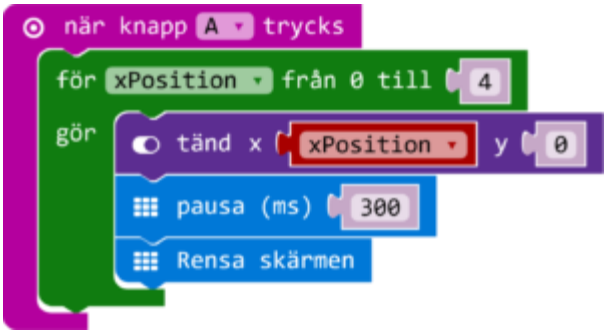
En rolig detalj är att blocket som vi använde i början, ”upprepa 4 gånger” faktiskt översätts till samma typ av for-loop. Det är bara att man inte ser variabeln i block-läget. Villkoret är också skriven i ”mindre än” format (istället för ”mindre än eller lika med”), vilket är också hur det används väldigt ofta inom programmering.




```
for (let i = 0; i < 4; i++) {
```

```
}
```

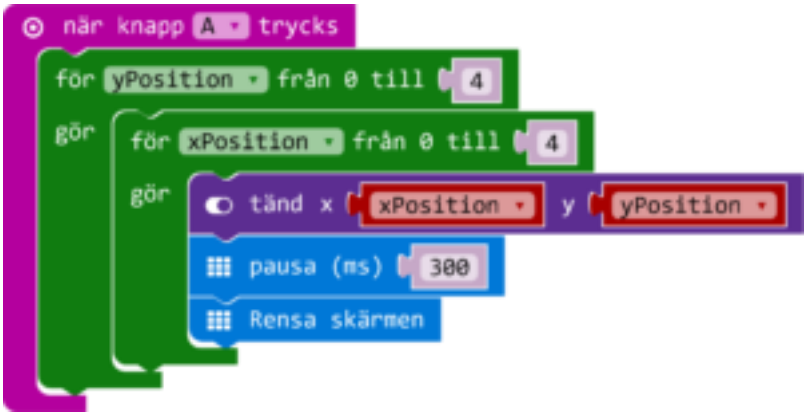
Men! Tillbaka till våra pixlar. Hur skulle den koden se ut när vi använder en för-loop (for loop)? Det blir mycket kortare. Man behöver inte använda variabelnamn ”plats”, så jag bytte den till ”xPosition”. Den här koden gör precis samma som våra förre pixel koden, men med mindre block eller rader kod.



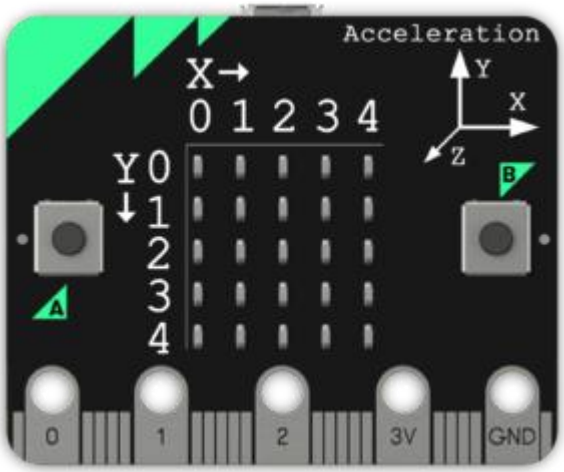

```
input.onButtonPressed(Button.A, () => {
    for (let xPosition = 0; xPosition <= 4; xPosition++) {
        led.plot(xPosition, 0)
        basic.pause(300)
        basic.clearScreen()
    }
})
```

DUBBELT SÅ KUL

Om man trycker att det var kul att flytta en pixel vänster till höger, kommer det nu bli dubbelt så kul. För att vi kan också styra y-koordinaten. Ska vi flytta en pixel över hela skärmen? Det kan man göra med hjälp av två loopar, där EN LOOP LIGGER PÅ INSIDAN AV DEN ANDRA. Hur funkar det? Se om du kan följa vad som händer i denna kod:




```
input.onButtonPressed(Button.A, () => {
    for (let yPosition = 0; yPosition <= 4; yPosition++) {
        for (let xPosition = 0; xPosition <= 4; xPosition++) {
            led.plot(xPosition, yPosition)
            basic.pause(300)
            basic.clearScreen()
        }
    }
})
```



För att förstå vad som händer kan vi läsa oss igenom koden, i samma ordning som micro:bit läser, och se vad som händer steg för steg. Koden börjar när A-knappen är tryckt. Vi hoppar in i yttersta loopen. Där skapas variabeln yPosition med värdet 0.

Nu hoppar vi in i innersta loopen och skapa variabeln xPosition med värdet 0. Näst tänds lampan med koordinaterna som vi har i variablerna, så (0,0). Efter en liten paus rensas skärmen, vilket gör att lampan släcks igen. Då hoppar vi tillbaka till början av innersta loopen, och xPosition höjs med 1. Nu tänds en lampa igen, med koordinaterna som dom är nu: punkt (1,0). Skärmen rensas igen. Innersta loopen upperpas en gång till, med xPosition lika med 2… Detta fortsätter tills loopen är genomförd med xPosition lika med 4. Då är innersta loopen färdigt. Nu hoppar vi tillbaka till början av yttersta loopen, och ökar yPosition med 1. Då kommer vi fram till innersta loopen igen. xPosition skapas på nytt, med värdet 0 igen. In i loopen tänds en lampa igen med koordinaterna som finns i variablerna, vilket nu blir (0,1). Innersta loopen körs en gång till och ger punkt (1,1).

Detta fortsätter till punkt (4,1). Då är den färdigt igen och hoppar vi till början av yttersta loopen igen med yPosition höjd till 2. Man blir nästan snurrigt av alla upprepningar. Vi kan skriva ner i en tabell hur variablerna, och med dom koordinatpunkten, förändras under programmets körning.

xPosition	yPosition	Punkt
0	0	(0,0)

xPosition	yPosition	Punkt
1	0	(1,0)
2	0	(2,0)
3	0	(3,0)
4	0	(4,0)
0	1	(0,1)
1	1	(1,1)
2	1	(2,1)
3	1	(3,1)
4	1	(4,1)
0	2	(0,2)
1	2	(1,2)
2	2	(2,2)

xPosition	yPosition	Punkt
3	2	(3,2)
4	2	(4,2)
0	3	(0,3)
1	3	(1,3)
2	3	(2,3)
3	3	(3,3)
4	3	(4,3)
0	4	(0,4)
1	4	(1,4)
2	4	(2,4)
3	4	(3,4)
4	4	(4,4)

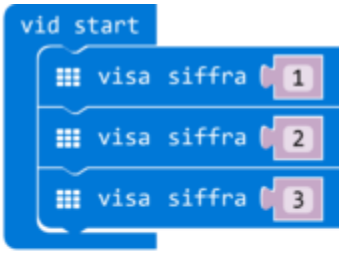
Uppgift 8: skapa ett blockprogram som visar ditt namn på microbit-skärmen 10 gånger med 100 millisekunder paus.

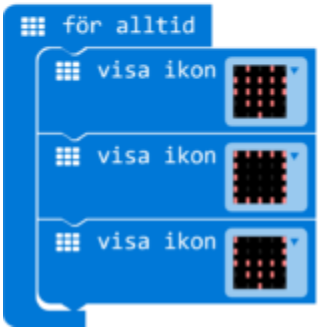
Uppgift 9: skapa ett blockprogram som tänder alla dioder diagonal som en X.

Uppgift 10: Gör ditt eget blockprogram.

VID START

Alla block som ligger i ”vid start”-blocket kommer köras direkt i början när micro:bitten startar, i tur och ordning, en i taget, en gång. Efter det fortsätter ”för alltid”-blocket.



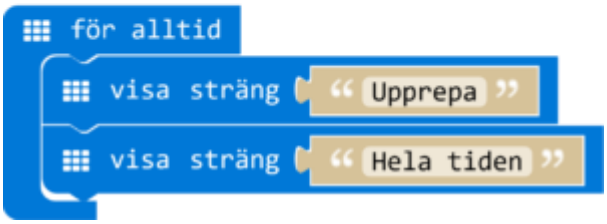



```
basic.forever(() => {
    basic.showIcon(IconNames.Heart)
    basic.showIcon(IconNames.Square)
    basic.showIcon(IconNames.Cow)
})
basic.showNumber(1)
basic.showNumber(2)
basic.showNumber(3)
```

Här ser vi ett exempel som först visar siffran ett, sen två, sen tre. Då har den gått genom allt i ”vid start”. Då börjar ”för alltid”-blocket, som visar ett hjärta, sen en fyrkant och till sist en (abstrakt) ko. Efter det börjar ”för alltid”-blocket om igen med att visa ett hjärta, och så vidare i all evighet tills micro:bit inte får ström längre.

HUR SER DET UT I JAVASCRIPT

När man byter från block till JavaScriptläge märker man att det inte finns ”vid start” i textkoden. All kod som finns i ”vid start”-blocket hamnar istället längst ner, efter allt annat. Detta är kanske inte det mest logiska, men när man vet om det är det mindre förvirrande. (Möjligen kommer detta ändras i framtiden.



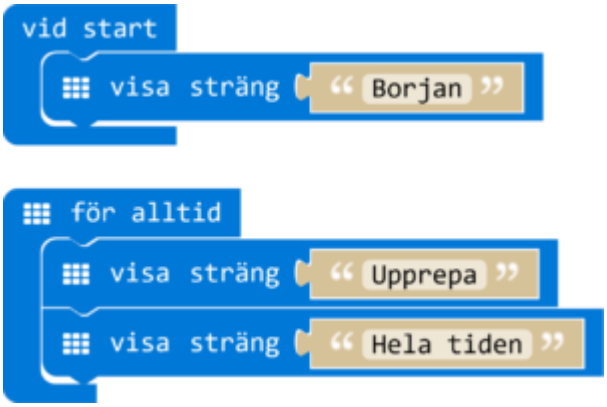

```
basic.forever(() => {
    basic.showString("Upprepa")
    basic.showString("Hela tiden")
})
basic.showString("Borjan")
```

OM DU SAKNAR BLOCKET

Ifall man har tagit bort sitt ”vid start”-block kan man hämta en ny i Grundläggande kategorin. Men se till att inte ha fler en en, för det funkar inte.

FÖR ALLTID

Alla block som ligger i ”för alltid”-blocket kommer köras efter att ”vid start”-blocket är färdigt, i tur och ordning, en i taget, och upprepas i all evighet.




```
basic.forever(() => {
    basic.showString("Upprepa")
    basic.showString("Hela tiden")
})
basic.showString("Borjan")
```

HUR SER DET UT I JAVASCRIPT

När vi byter från block till JavaScript läge ”för alltid”-blocket översatt till raden:

```
basic.forever( () => { } )
```

Alla block som fanns inom ”för alltid”-blocket blir översatt till kod som hamnar mellan måsvingarna ({}). Detta kan blir hur många rader som helst. Alla dessa rader kod har fått fyra mellanslag i början. Det gör att det är lättare att se för oss människor vad som finns inne i ”för alltid”-loopen(forever).

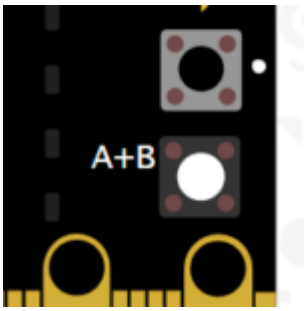
OM DU SAKNAR BLOCKET

Ifall man har tagit bort sitt ”för alltid”-block kan man hämta en ny i Grundläggande kategorin. Det går att ha flera ”för alltid”-block, men det kan leda till en förvirrande kod eftersom båda loopar försöker köra samtidigt.

KNAPPAR

Det finns två knappar på micro:bits framsidan, markerad med A och B. Vad som händer när man trycker på dom (eller båda samtidigt) kan vi bestämma genom att programmera micro:bit.

I kategorin Input finns ett stort block ”När knapp A trycks”. I den kan man lägga andra block, vilka då kommer utföras, i tur och ordning, när A knappen är tryckt och släppt. För att byta till B knappen, eller till ”båda knappar samtidigt”(A+B), klickar man på det ljusare fältet där det står ”A▼”.



Detta exempel visar ett hjärta när A knappen trycks, en glad gubbe när B knappen trycks, och ett spöke när båda A och B trycks samtidigt. I simulatorn ser vi nu en extra knapp som är markerad med ”A+B”. Den finns bara där eftersom man inte kan klicka på A och B samtidigt med din muspekare. Så när man klickar där simulerar man att man trycker på båda knapparna.


```
input.onButtonPressed(Button.A, () => {
```

```
    basic.showIcon(IconNames.Heart)
    basic.clearScreen()
})
input.onButtonPressed(Button.B, () => {
    basic.showIcon(IconNames.Happy)
    basic.clearScreen()
})
input.onButtonPressed(Button.AB, () => {
    basic.showIcon(IconNames.Ghost)
    basic.clearScreen()
})
```

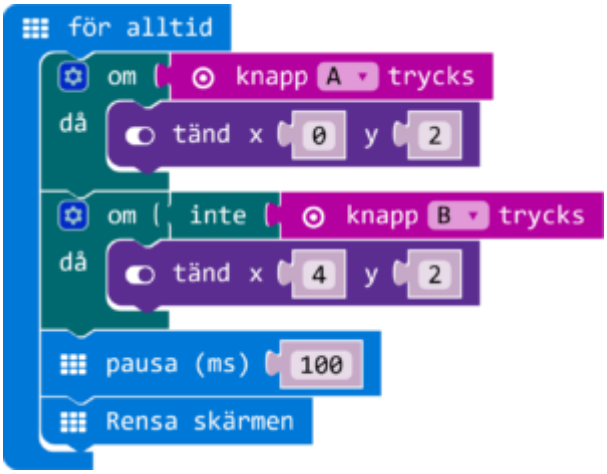
DET FINNS EN KNAPP TILL

På baksidan av micro:bit finns en knapp till, men den har en förutbestämd funktion. Det är reset knappen. När man trycker på den startar micro:bit om och ditt program körs igång från början. (Programmet raderas inte.)

ETT ANNAT SÄTT ATT ANVÄNDA KNAPPARNA

Om vi tittar i Input kategorin hittar vi ett till block som handlar om knapparna. Där står det ”knapp A trycks” på ett block som ser ut som en pusselbit på vänster sidan. Detta block ger ett resultat som är antingen sant(true) eller falskt(false) (Läs mer här), beroende på om knappen är i nertryckt läge just nu(sant) eller inte(falskt). Det kan man till exempel använda när man vill få någonting att hända så länge som en knapp hålls nertryckt.

Följande kod tänder en lampa på koordinaterna (0, 2) så länge som A knappen är nertryckt. På andra om-satsen, som kollar på B knappen, har vi lagt till ett ”inte”-block som gör att sant blir falskt, och falskt blir sant. Så där blir lampan på koordinaterna (4, 2) tänd bara så länge B knappen inte trycks ner.




```
basic.forever(() => {
    if (input.buttonIsPressed(Button.A)) {
        led.plot(0, 2)
    }
    if (!(input.buttonIsPressed(Button.B))) {
        led.plot(4, 2)
    }
    basic.pause(100)
    basic.clearScreen()
})
```

LJUSSENSOR

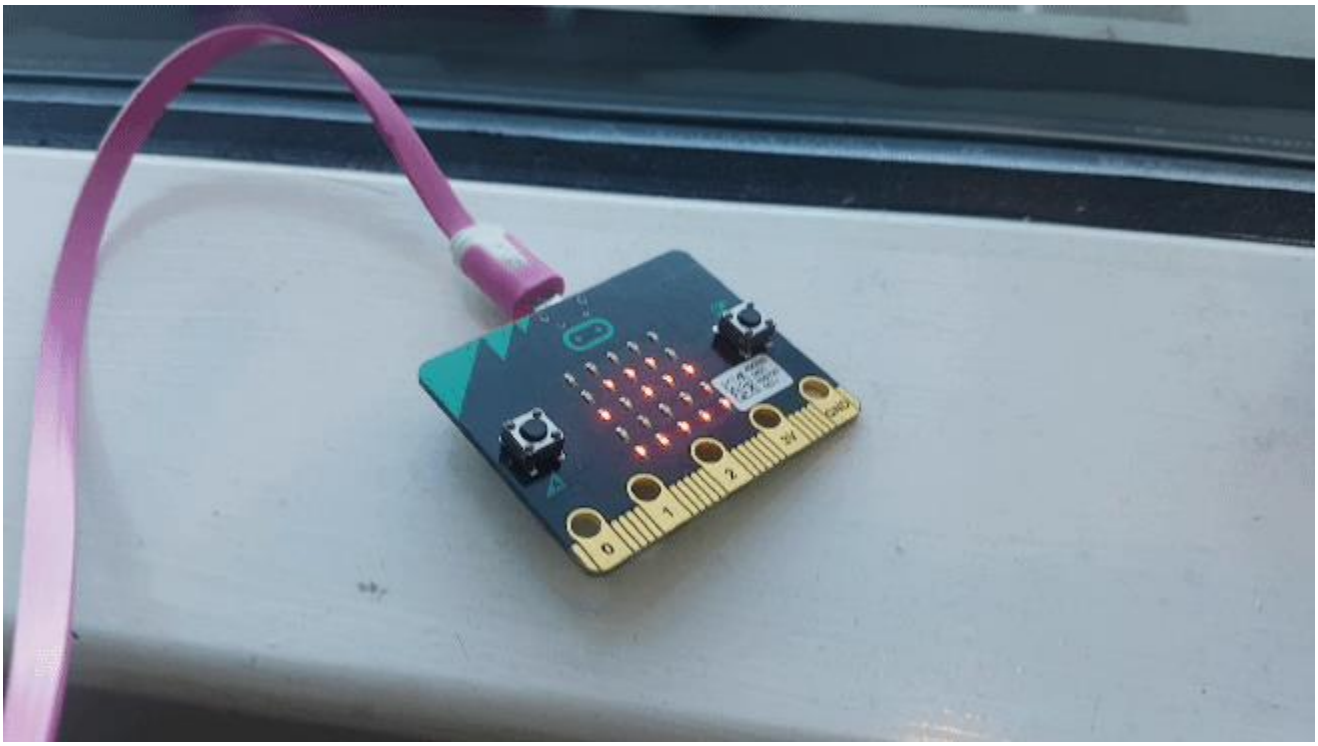
Micro:bit kan känna av hur mycket ljus som faller på sin skärm. Det ger är värde från 0 (mörkt) till 255 (ljust).

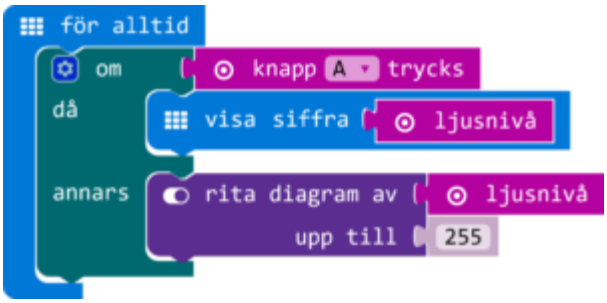
I kategorin Input finns ljusnivå-blocket. Med sin pusselbitsformade vänstersida kan den kopplats till alla block som förväntar sig ett siffra, eftersom detta block ger ett siffra mellan 0 (mörkt) och 255 (ljust).

EXPERIMENTERA!

För att testa hur mycket ljus din micro:bit upplever där du är nu, ladda ner följande exempel. Den kommer att visa en graf som tänder fler LED lampor på skärmen desto högre siffra vi får från ljusnivå-blocket, med som högsta värde 255. När du trycker på A knappen visas vilket värde det är just då.

I simulatorn visas nu en halv-gul cirkel där man kan dra upp och ner för att simulera olika mycket ljus. Men se till att ladda ner och testa med en riktig micro:bit också!






```
basic.forever(() => {
    if (input.buttonIsPressed(Button.A)) {
        basic.showNumber(input.lightLevel())
    } else {
        led.plotBarGraph(
            input.lightLevel(),
            255
        )
    }
})
```

STOPPA TJUVEN!

Nu vi har lite koll på vad blocket gör kan vi hitta på vad vi vill använda det till. Jag gillar godis, och jag är inte ensam om det. Kanske vi kan använda micro:bit för att vakta mitt godis?

Om jag lägger mitt godis tillsammans med micro:bit i ett stängd skåp är det mörkt. Så snabbt som någon öppnar skåpet kommer det bli ljust, och ska micro:bit ge ett larm. För att göra det har vi en om-sats, som kollar hela tiden om ljusstyrkan är mer än det vi sätter som gräns (t.ex. 50). För att slå larm använder jag två block från Musik kategorin, och jag kopplar min micro:bit till en högtalare (Se här hur Musik block fungerar).

```
basic.forever(() => {
    if (input.lightLevel() > 50) {
        music.playTone(698, music.beat(BeatFraction.Half))
        music.rest(music.beat(BeatFraction.Eighth))
    }
})
```

TERMOMETER

Inbyggd i processorn finns en termometer. Temperaturen den mäter kan vi få från temperatur-blocket i grader Celcius.

VARFÖR DÄR?

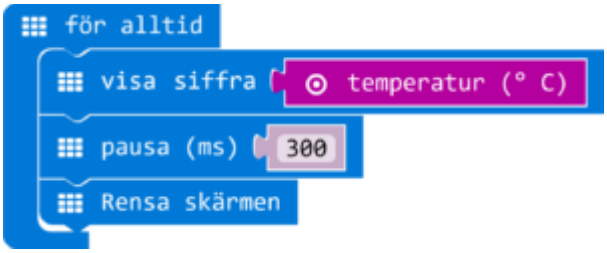
Anledningen till varför termometern finns i processorn är att den ska känna av när processorn jobbar för hårt och blir för varmt. Då kan den stängas ner innan någonting går sönder.

HUR NOGGRANN?

Eftersom termometern finns i processorn, vilken blir lite varm när den jobbar, kommer den mätade temperaturen ofta vara lite högre än temperaturen av luften runt omkring micro:bit. Men skillnaden på hur mycket temperaturen ändras upp och ner är den väldigt bra på. Om du håller micro:bit med processorn mot någonting varmt (dina fingrar?) eller kallt (en bit av metal?) kommer den kunna känna av det.

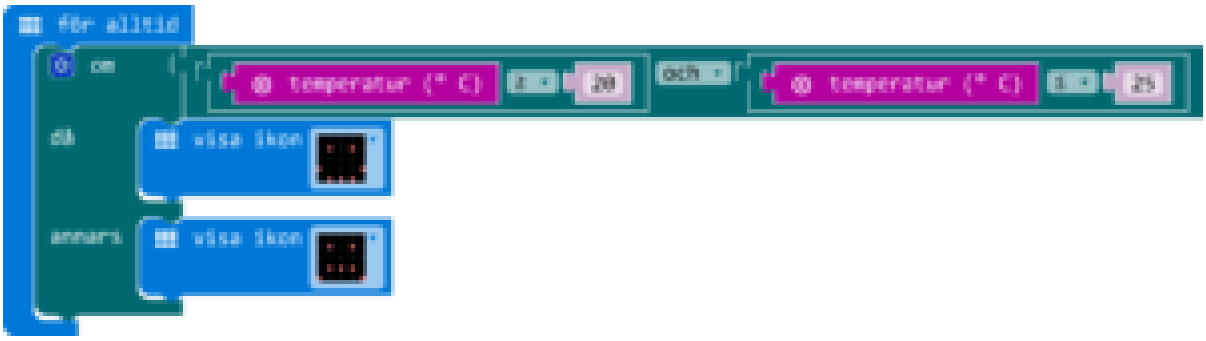
ANVÄNDA

På liknande sätt som ljusnivå-blocket har temperatur-blocket en pusselbit form på vänstersidan. Med den kan vi pyssla ihop den med andra block som vill få ett siffra. Till exempel kan vi få våra micro:bit att visa temperaturen på skärmen med följande kod. Pausen och rensningen är till för att se en tydlig skillnad varje gång temperaturen scrollas förbi på nytt.



LAGOM VARM

Vi kan också visa temperaturen på ett annat sätt. Den här koden visar en glad gubbe när det är lagom varm, annars blir det en sur gubbe. Gör din egen variant där du bestämmer vad är definitionen av lagom!



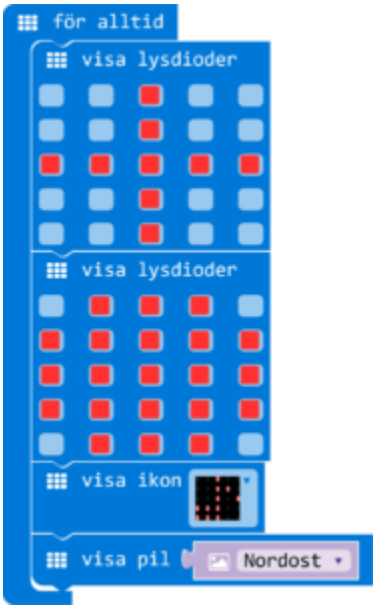

```
basic.forever(() => {
    if (input.temperature() >= 20 && input.temperature() <= 25) {
        basic.showIcon(IconNames.Happy)
    } else {
        basic.showIcon(IconNames.Sad)
    }
})
```

LED LAMPOR

Micro:bit har 25 röda lysdioder (LED Lampor) som man kan tända och släcka med sin kod. Använd dom för att visa text, siffror, animationer och mer.

VISA BILDER

Det finns många olika block för att styra lamporna. När man vill visa en bild, eller göra animationer med hjälp av flera bilder efter varandra, är ”visa lysdioder”-blocket väldigt smidigt. Tryck på en ruta för att byta mellan grå (släckt) och röd (tänd). I JavaScript läget blir det en punkt (släckt) eller ett nummertecken (tänd). Det finns även ett block med några förhandsgjorda bilder, som heter ”visa ikon”. När man vill visa en riktning, till exempel när man bygger en kompass, finns det även ett block specifikt för att visa pilar. Den heter ”visa pil”, och finns under MER när man har öppnat Grundläggande kategorin. Följande exempel byter mellan att visa ett plustecken, en cirkel, en åttonde musiknot, och en pil Nordost.

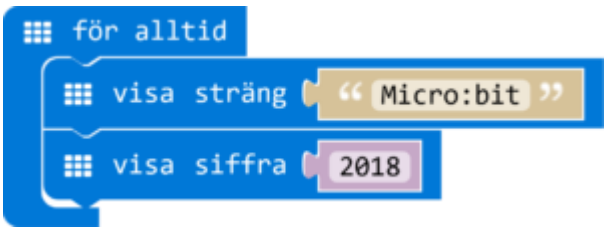



```
basic.forever(() => {
    basic.showLeds(`
        . . # . .
        . . # . .
        # # # # #
```

```
        . . # . .
        . . # . .
    `)
    basic.showLeds(`
        . # # # .
        # # # # #
        # # # # #
        # # # # #
        . # # # .
    `)
    basic.showIcon(IconNames.EigthNote)
    basic.showArrow(ArrowNames.NorthEast)
})
```

VISA TEXT OCH SIFFROR

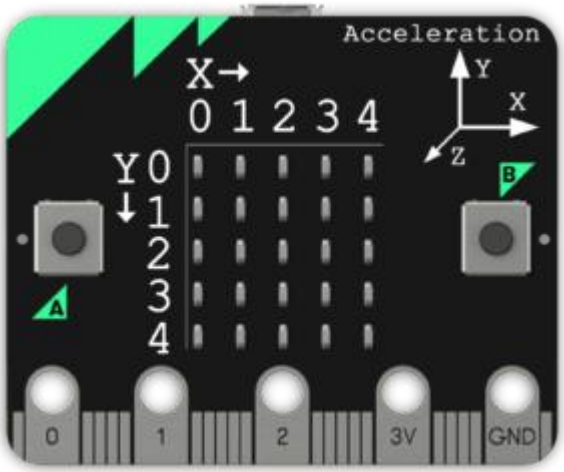
När man vill skriva något men inte rita ut varje bokstav själv, finns det även ett block för att visa text och en ett för siffror. Båda hittar man i Grundläggande kategorin. Om det handlar om mer än ett tecken, vilket inte ryms, scrollar texten/siffran automatiskt förbi på skärmen. Det finns en begränsning med ”visa text”-blocket, och det är att den inte klarar av ä, å och ö bokstäver. Istället visas då ett mellanslag.



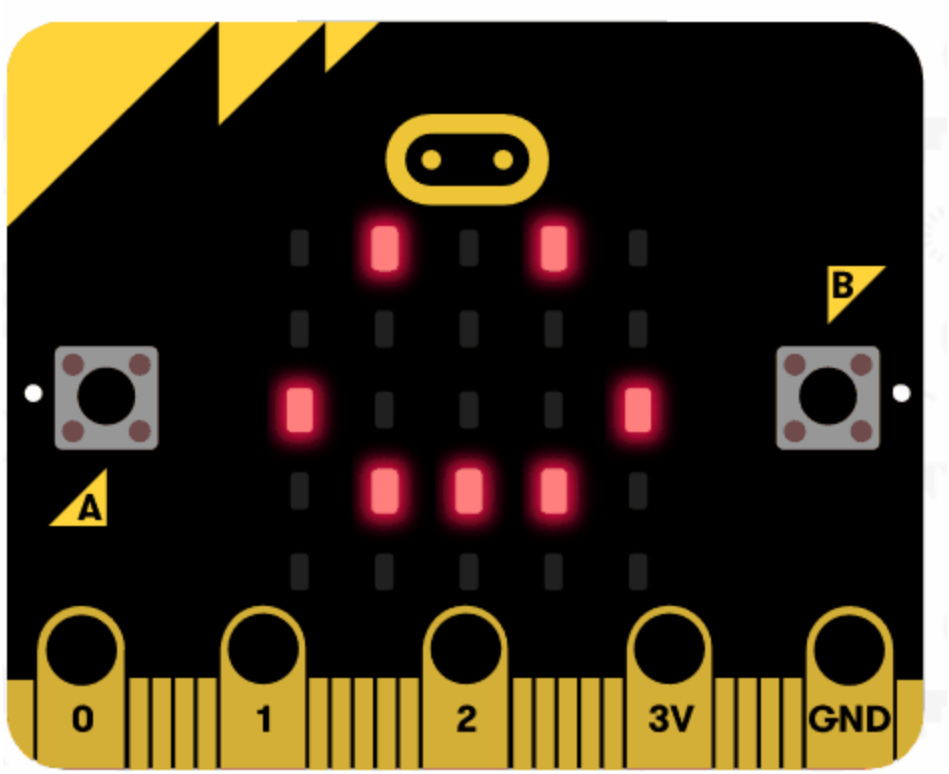

```
basic.forever(() => {
    basic.showString("Micro:bit")
    basic.showNumber(2018)
})
```

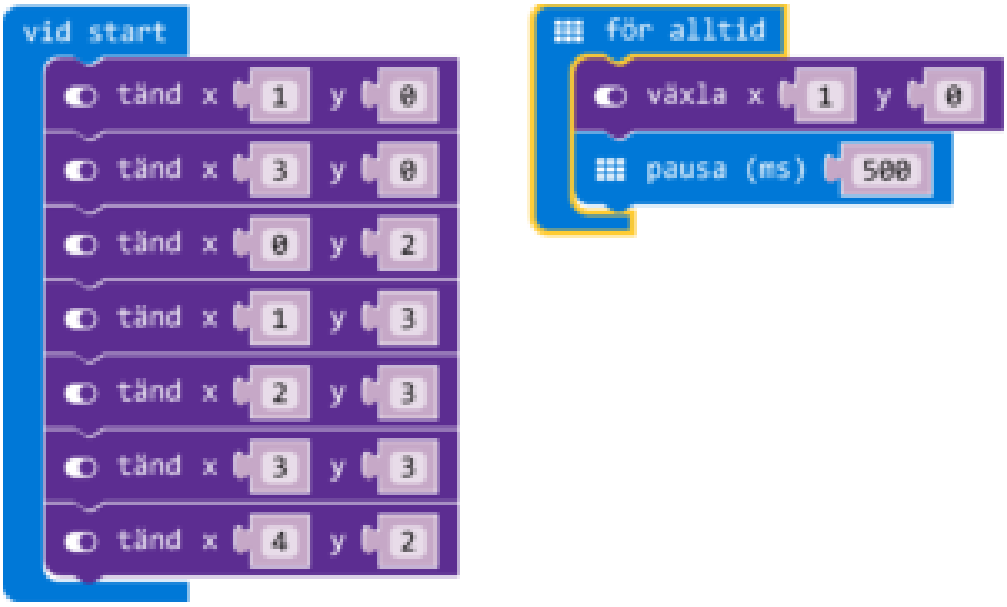
PIXEL PRECIS

I kategorin Led finns olika block som gör det möjligt att tända och släcka varenda lampa. För att bestämma vilken lampa det handlar om används ett x och y koordinatsystem. Till skillnad från det koordinatsystemet man är van vid från matematik börjar det med (0, 0) längst upp till vänster. X ökar för varje steg åt höger, och Y ökar för varje steg neråt. Detta är det standard koordinatsystemet som används inom programmering för att rita med pixlar på digitala skärmar.



Följande exempel ritar först en glad gubbe, och sen får den att blinka för alltid genom att tända ock släcka ett öga med hjälp av ett växla-block. Kan du lista ut vilka koordinater motsvarar vilka punkter?






```
led.plot(1, 0)
led.plot(3, 0)
led.plot(0, 2)
led.plot(1, 3)
led.plot(2, 3)
```

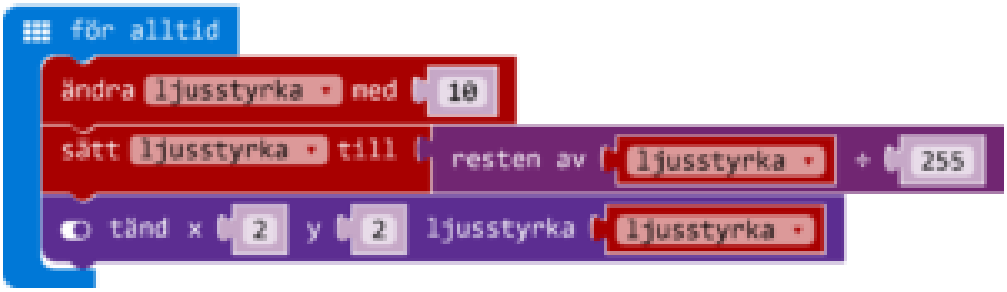
```
led.plot(3, 3)
led.plot(4, 2)
basic.forever(() => {
    led.toggle(1, 0)
    basic.pause(500)
})
```

På samma plats finns även ett ”punkt”-block. Den har ett pusselbitskant på vänster sidan, vilket betyder att den ger tillbaka ett värde. Detta block ger nämligen svar om en pixel är tänd eller inte, med som svar värdet sant eller falskt. Detta kan till exempel användas kopplat till en om-sats. Följande kodexempel tänder ett slumpmässigt antal pixlar när man trycker på A knappen. Sen när man trycker på B knappen går den genom alla punkter och räknas hur många är tänd.


```
let mängdPixlar = 0
input.onButtonPressed(Button.A, () => {
```

```
    for (let i = 0; i < Math.random(11); i++) {
        led.plot(Math.random(5), Math.random(5))
        basic.pause(100)
    }
})
input.onButtonPressed(Button.B, () => {
    mängdPixlar = 0
    for (let x = 0; x <= 4; x++) {
        for (let y = 0; y <= 4; y++) {
            if (led.point(x, y)) {
                mängdPixlar += 1
            }
        }
    }
    basic.clearScreen()
    basic.showNumber(mängdPixlar)
    basic.pause(100)
    basic.clearScreen()
})
```

En annan kul block som finns under Mer av Led kategorin är ”tänd x y ljusstyrka”-blocket. Med den kan man inte bara tända en lampa, men även bestämma hur ljust den ska lysa. Värdet ska vara något från 0 (släckt) till 255 (fullt ljus). Här ett exempel som snabbt går genom olika styrkor.

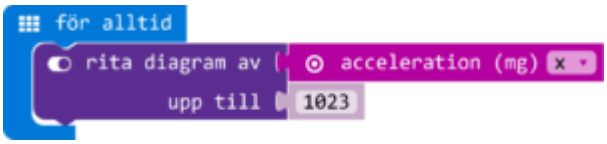


```
let ljusstyrka = 0
ljusstyrka = 0
basic.forever(() => {
    ljusstyrka += 10
    ljusstyrka = ljusstyrka % 255
    led.plotBrightness(2, 2, ljusstyrka)
})
```

RITA EN DIAGRAM

Ett annat kul block är ”rita diagram”. Den visar ett värde du väljer, genom att fylla upp skärmen mer när värdet blir högre. Du bestämmer också vad det maximala värdet är, när skärmen ska vara helt fyllt. Detta är ett smidigt sätt att snabbt få en känsla över hur ett visst värde förändrats.

Till exempel kan vi visa upp accelationskraften. Vilket värde tar vi som max? Från dokumentationssida om accelerationsblocket hittar vi 1023 som ett värde när kraften är 1 g, så vi kan ta det och se vad som händer. Experimentera med att sätta in olika max värden, och med andra värden att rita en diagram av.



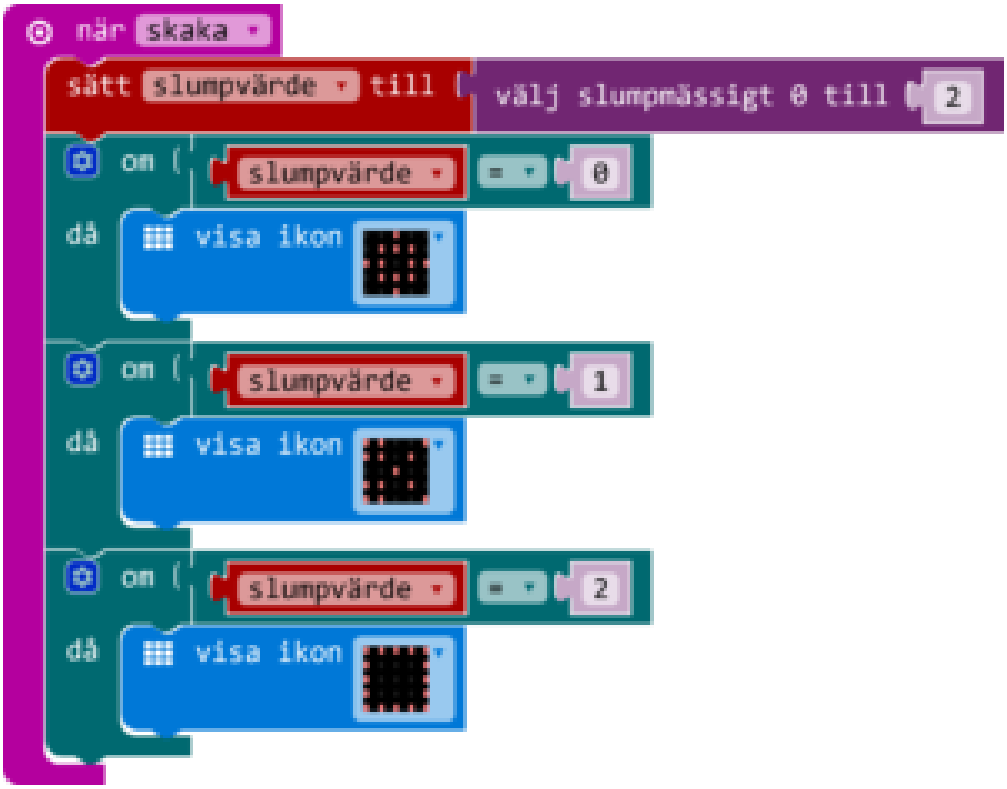

```
basic.forever(() => {
    led.plotBarGraph(
        input.acceleration(Dimension.X),
        1023
    )
})
```

ACCELEROMETERN

Accelerometer är en sensor på baksidan av din micro:bit som kan känna av acceleration. Med hjälp av den kan din micro:bit veta till exempel om den skakas, tappas, eller lutas i olika vinklar.

OMSKAKAS!

Vi börjar med skakning. I kategorin Input finns ett block som körs igång när en micro:bit skakas, och alla block som ligger i den utförs då. Till exempel kan man bygga ett sten-sax-påse spel som slumpmässigt visar en av tre möjligheter när micro:bit skakas.




```
let slumpvärde = 0
input.onGesture(Gesture.Shake, () => {
 slumpvärde = Math.random(3)
 if (slumpvärde == 0) {
 basic.showIcon(IconNames.Target)
 }
 if (slumpvärde == 1) {
 basic.showIcon(IconNames.Scissors)
 }
 if (slumpvärde == 2) {
 basic.showIcon(IconNames.Square)
 }
```

```
})
```

ANDRA GESTER, SAMMA BLOCK

Samma block kan även användas till andra händelser än skakning. Genom att trycka på ”skaka” kan det bytas ut till andra händelser som accelerometern kan känna av. Det finns sex olika riktningar du kan hålla micro:bit i, och fyra olika styrka av accelerationskrafter som kan väljas.

Logotyp upp
När du håller micro:bit så att USB-porten pekar uppåt.

Logotyp ner
När du håller micro:bit så att USB-porten pekar neråt.

Skärm upp
När du håller micro:bit så att skärmen pekar uppåt.

Skärm ner
När du håller micro:bit så att skärmen pekar neråt.

Luta vänster
När du håller micro:bit så att sidan där A-knappen sitter pekar neråt.

Luta höger
När du håller micro:bit så att sidan där B-knappen sitter pekar neråt.

Fritt fall
När micro:bit upplever fritt fall. Då känner den ingen tyngdkraft, och ingen acceleration alls.

3g
När micro:bit upplever en kraft tre gånger så stark som tyngdkraften.

6g
När micro:bit upplever en kraft sex gånger så stark som tyngdkraften.

8g
När micro:bit upplever en kraft åtta gånger så stark som tyngdkraften.

Blocket körs igång när händelsen händer. Det vill säga att om vi till exempel väljer ”skärm upp” så körs koden när vi lutar på micro:bit så att skärmen pekar uppåt. Men så länge vi håller den positionen körs koden inte igen. Bara om vi lutar den på något annat sätt, och sen går tillbaka till att ha skärmen uppåt, körs vår kod en gång till.

Följande exempel lyssnar på dom sex olika positioner och visar en pil eller ikon beroende på hur du håller micro:bit.


```
input.onGesture(Gesture.ScreenUp, () => {
 basic.showIcon(IconNames.Happy)
})
input.onGesture(Gesture.ScreenDown, () => {
 basic.showIcon(IconNames.Sad)
})
input.onGesture(Gesture.LogoDown, () => {
```

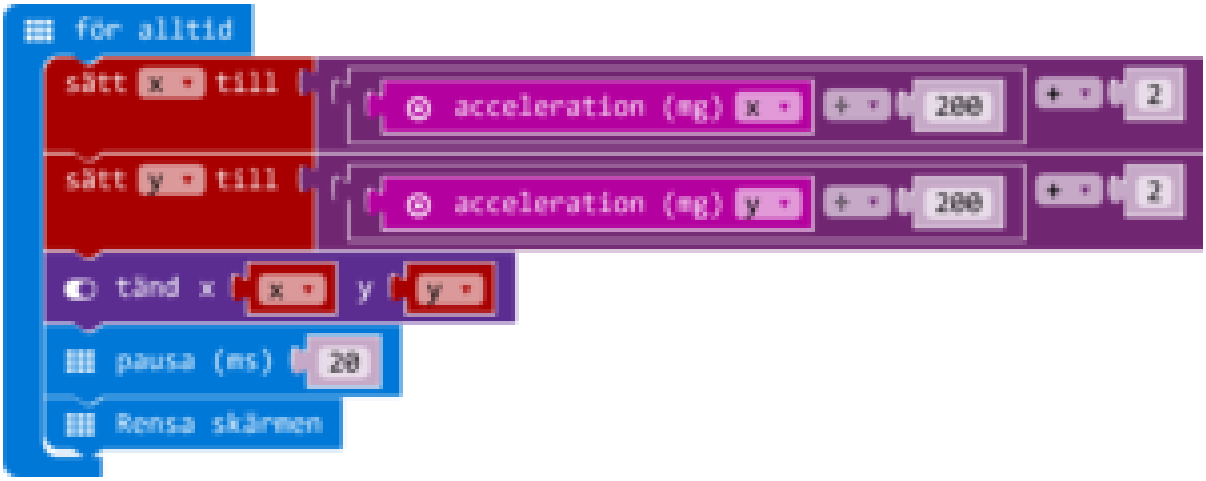
```
 basic.showArrow(ArrowNames.North)
})
input.onGesture(Gesture.LogoUp, () => {
 basic.showArrow(ArrowNames.South)
})
input.onGesture(Gesture.TiltLeft, () => {
 basic.showArrow(ArrowNames.West)
})
input.onGesture(Gesture.TiltRight, () => {
 basic.showArrow(ArrowNames.East)
})
```

ACCELERATION

Ibland vill man ha mer fin justerbart kontroll över vad som ska hända beroende på accelerationskrafter. För dom tillfällen finns ”acceleration (mg)”-blocket. Den ger oss ett siffra som berättar hur stor accelerationskraft micro:bit känner precis nu, i en viss riktning. Enheten mg i parenteser är en förkortning av milligravitation. Detta betyder att ett värde av 1000mg är lika med 1g, vilket är styrkan av tyngdkraften på jorden. Nu finns det en liten skillnad med micro:bit, den ger oss 1023 som värde när den upplever 1g.

I blocket kan vi välja i vilken riktning vi vill mäta accelerationen. X-axeln går från vänster (där A-knappen sitter) till höger(där B-knappen sitter), y-axeln går från ner (där pins sitter) till upp (där USB-porten sitter), och z-axeln går från baksidan till framsidan. Så om du lägger micro:bit perfekt platt på ett bord känner den 0 i båda x- och y-riktningen, och 1023 i z-riktningen (i verkligheten kan det skilja sig lite). Om du lägger den perfekt platt med skärmen neråt får du -1023 i z-riktningen, eftersom tyngdkrafterna nu drar i den på precis motsatt håll.

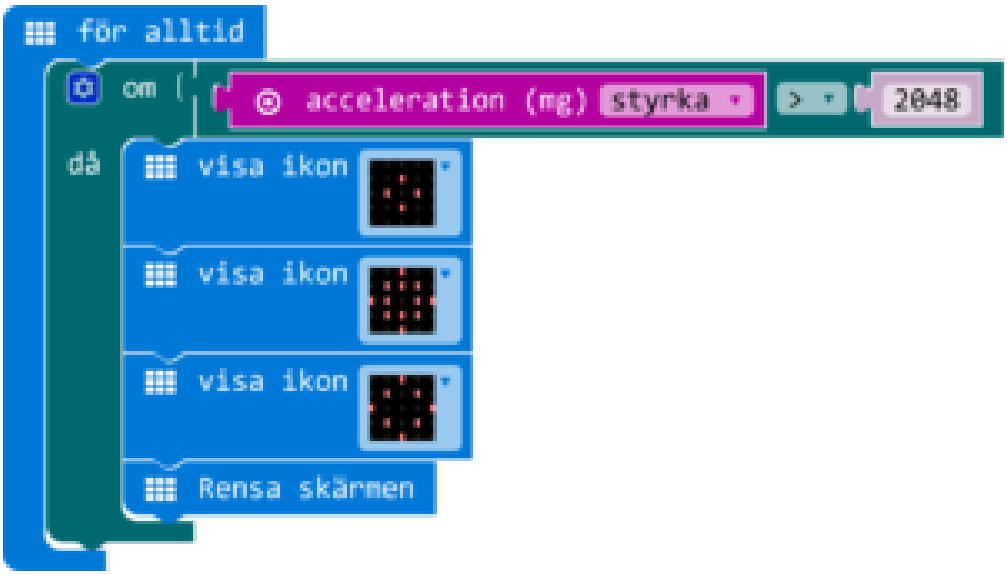
Följande exempel använder sig av accelerationsvärdet i x- och y-axeln, för att flytta en pixel över skärmen. Värdet delas först med `200`, för att justera känsligheten (Testa att ändra den!). Sen ökas den med `2` för att få pixeln att hamna i mitten, på koordinat `(2,2)`, när accelerationsvärdet är `0`. ”Tänd”-blocket tänder pixeln, och efter en liten paus (20 millisekunder) släcks alla pixlar igen.




```
let y = 0
let x = 0
basic.forever(() => {
 x = input.acceleration(Dimension.X) / 200 + 2
 y = input.acceleration(Dimension.Y) / 200 + 2
 led.plot(x, y)
 basic.pause(20)
 basic.clearScreen()
})
```

VILKEN RIKTNING SOM HELST

Fjärde valmöjligheten istället för x, y och z är ”styrka”. Om vi väljer den får vi ett värde som är en samling av alla riktningar tillsammans, som alltid är positivt. Om du håller micro:bit stilla, ska du få 1023, oberoende i vilken position du håller den. Styrka kan vi använda till exempel för att bygga vår egen variant på ”när skaka”-blocket där vi själv kan bestämma hur hårt man behöver skaka micro:bit.




```
basic.forever(() => {
 if (input.acceleration(Dimension.Strength) > 2048) {
   basic.showIcon(IconNames.SmallDiamond)
   basic.showIcon(IconNames.Target)
   basic.showIcon(IconNames.Diamond)
   basic.clearScreen()
 }
})
```

ROTATION

Under Mer i Input-kategorin finns även ett ”rotation”-block. Den använder sig av samma värdet vi får från ”acceleration”-blocket, men räknar ut för oss i vilken vinkel (i grader) vi lutar micro:bit. Här kan vi välja ”lutningsgrad” för upp och ner riktningen, och ”roll” för höger och vänster riktningen.

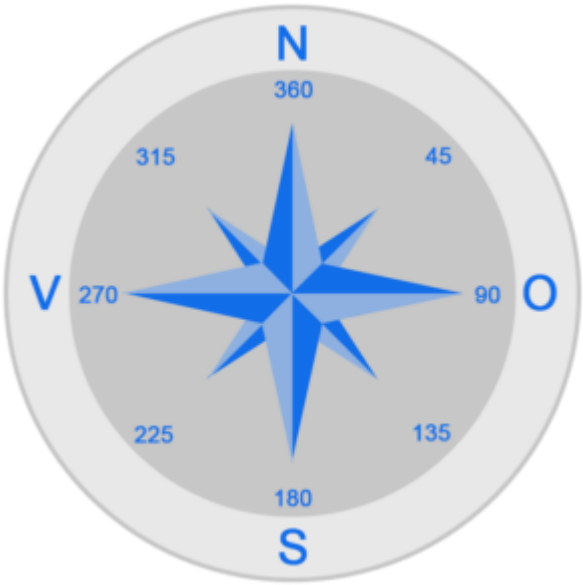
RÄCKVIDD

På samma plats hittar vi också ett block för att ändra räckvidden för accelerationsmetern. Med den ställer man in vad det högsta värdet är sensorn kommer ge oss.

Kompass och magneter

Med det digitala kompass (magnetometer) som finns på micro:bit kan den känna av magnetiska krafter som var norden ligger eller när en vanlig magnet kommer nära.

I kategorin Input finns kompassriktning-blocket. Den ger oss ett värde från 0 till 359 vilket står för hur många grader micro:bit är roterade jämfört med var magnetiska norden ligger. (Läs mer på wikipedia om skillnaden mellan nordpolen och magnetiska fältet.)



KALIBRERING

När micro:bit startar med ett program som använder sig av kompassen behövs det kalibreras. Först visas texten ”Draw a circle” på skärmen. Efter det ska du rotera runt micro:bit så att skärmen har pekat en hel cirkel runt åt alla väderstreck. För varje riktning som är kalibrerad tänds en pixel, som alla tillsammans ska forma en cirkel på skärmen. Då visas en glad gubbe och är kalibreringen färdigt. Nu körs ditt program igång. Nästa gång du kör programmet behöver du kanske inte göra detigen, om micro:bit kommer ihåg kalibreringen.

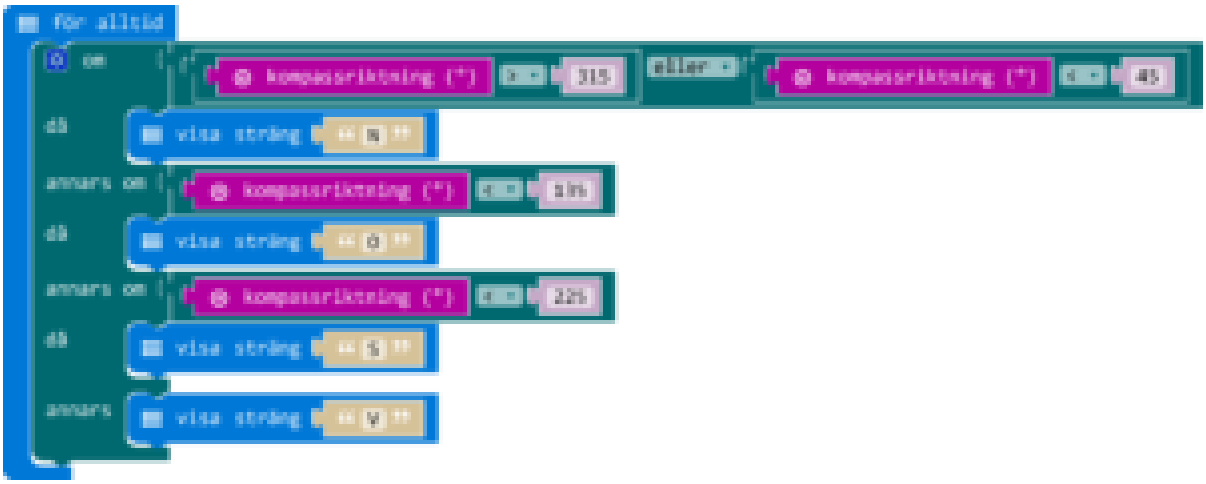
TESTA VÄRDEN

Följande program är ett enkelt sätt att testa vilka värden vi får. Tänk på att det tar lite tid att visa siffran, så det uppdateras inte så snabbt.


```
basic.forever(() => {
  basic.showNumber(input.compassHeading())
})
```

DIGITAL KOMPASS

Vi kan bygga ett digitalt kompass som visar kompassrosens bokstäver baserad på sensorn. Då delar vi upp 360 grader (hela cirkeln runt) i fyra delar, och beroende på i vilken del vi hamnar visar vi olika bokstäver. Norra delen är olika från dom andra eftersom det ligger vid dom högsta och lägsta värden. Så där blir villkoret en kombination med hjälp av ”eller”.




```
basic.forever(() => {
  if(input.compassHeading()> 315 || input.compassHeading()< 45){
    basic.showString("N")
  }elseif(input.compassHeading()< 135){
    basic.showString("O")
  }elseif(input.compassHeading()< 225){
    basic.showString("S")
  }else{
    basic.showString("V")
```

```
  }
})
```

KÄNNA KRAFTEN

Under Mer i Input kategorin finns ett block som ger oss ett mätvärde på magnetiska kraften i olika riktningar. Enheten för att mäta magnetiska krafter är Tesla (T). Eftersom 1 Tesla är ett riktigt starkt magnetfält ger micro:bit värden i mikrotesla (µT).

Några exempel:

- 31 µT Jordens magnetfält på ekvatorn
- 5 mT Vanlig kylskåpsmagnet
- 1,5 till 3 T MRI kamera i sjukhuset (Se experiment på youtube)

Vi kan göra ett litet program för att testa nära till olika magneter. Du kan experimentera med olika riktningar (x, y, z, och ”Styrka” som kombinerar alla). Micro:bit kommer visa en graf baserad på hur stor kraft den mäter, med som högst 5000 µT.


```
basic.forever(() => {
 led.plotBarGraph(
   input.magneticForce(Dimension.X),
   5000
 )
})
```

RADIO

Med block från kategorin Radio kan flera micro:bit kommunicera med varandra trådlöst.

Det första som är viktig med Radio är att se till att alla micro:bit pratar på samma frekvens, vilket kallas grupp. Grupp anges med ett nummer från 0 till 255. Det spelar ingen roll vilket nummer du väljer, bara att det inte finns någon nära dig som skickar i samma grupp. För då kan det bli förvirrande. Det behövs bara göras en gång, så det passar bra i ”vid start”-blocket.

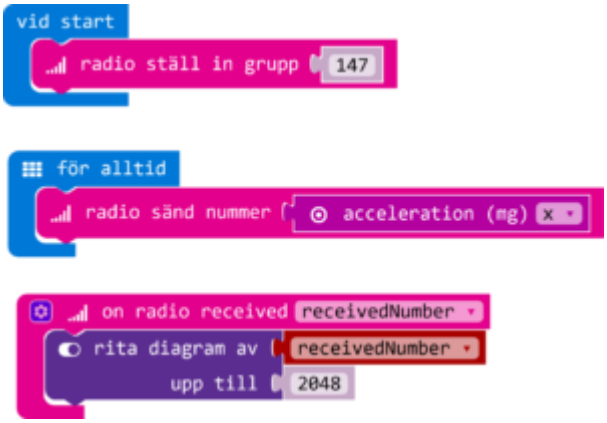
Sen finns det olika block för att skicka information. Man kan skicka en sträng (textbit), nummer, eller en kombination av en sträng och ett nummer. Detta tas emot i ett ”on radio received”-block. Värdet som detta block tar emot hamnar i en variabel vars namn (`receivedString`) visas i blocket. Efter att vi har dragit ur radioblocket kan vi hitta denna variabel i Variabler kategorin. Då kan vi enkelt visa upp strängen.


```
input.onButtonPressed(Button.A, () => {
    radio.sendString("Ja")
})
input.onButtonPressed(Button.B, () => {
    radio.sendString("Nej")
})
radio.onDataPacketReceived( ({ receivedString }) => {
```

```
    basic.showString(receivedString)
})
radio.setGroup(97)
```

STRÖMMA DATA

Ett annat roligt exempel är att använda radio för att hålla koll på en sensor på distans. Så här kan du skicka data hela tiden, och enkelt visa upp det på en annan micro:bit.

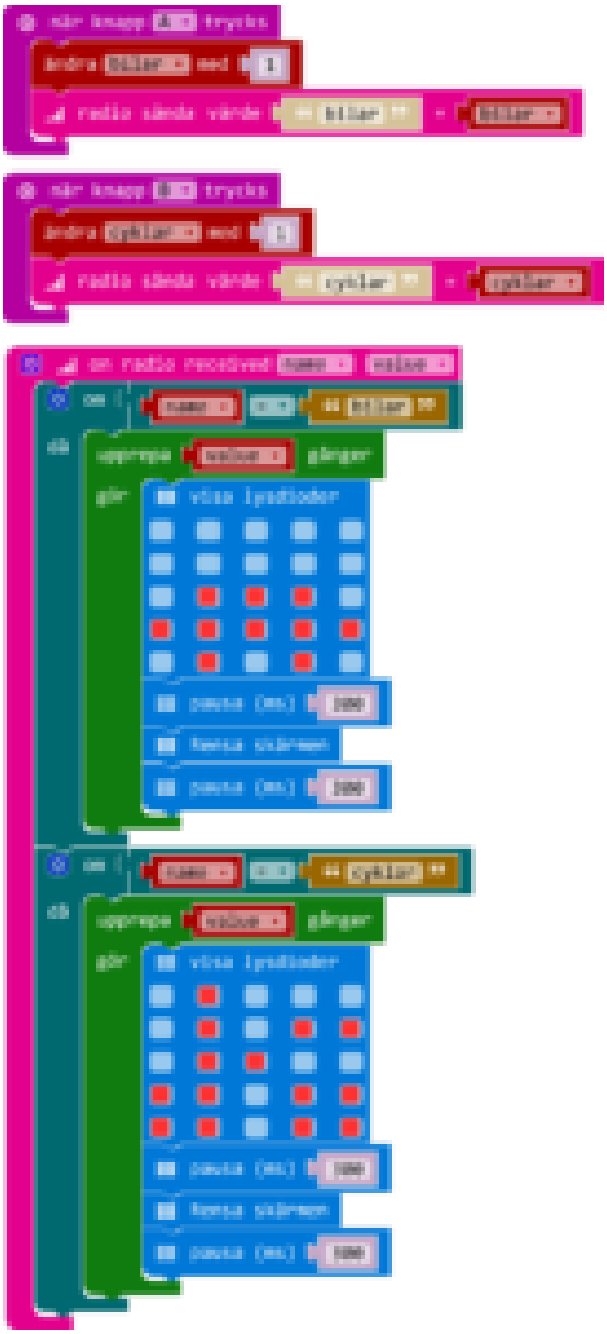



```
radio.onDataPacketReceived( ({ receivedNumber }) => {
    led.plotBarGraph(
        receivedNumber,
        2048
    )
})
basic.forever(() => {
    radio.sendNumber(input.acceleration(Dimension.X))
})
radio.setGroup(147)
```

RÄKNA FORDON

Om man vill kunna skicka olika information, och veta vilka information som skickas, kan man göra det med ”radio sända värde”-blocket. I blocket som tar emot har man tillgång till båda strängen och numret

som skickas ihop. Följande exemplet använder det för att räkna två olika fordon och skicka det vidare trådlöst. Där visas det på skärmen med en (ganska abstrakt) ikon beroende på fordonstyp, som blinkar lika många gånger som den var räknat.



```
let cyklar = 0
let bilar = 0
```

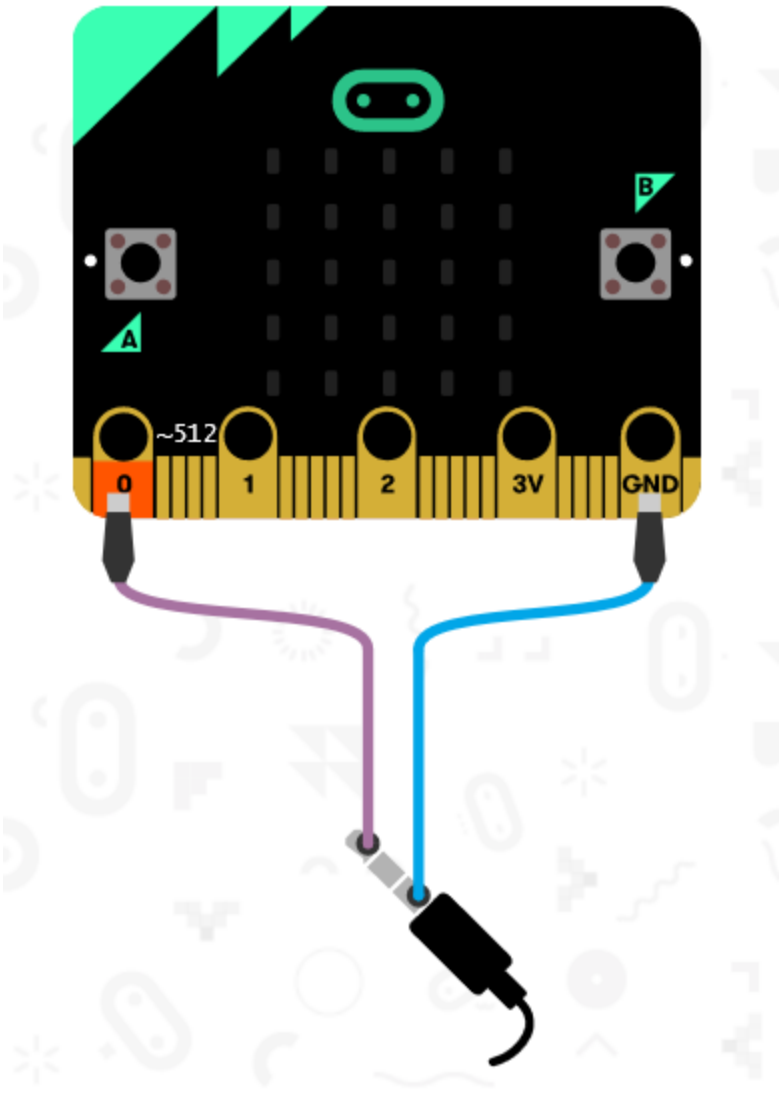
```
input.onButtonPressed(Button.A, () => {
    bilar += 1
    radio.sendValue("bilar", bilar)
})
input.onButtonPressed(Button.B, () => {
    cyklar += 1
    radio.sendValue("cyklar", cyklar)
})
radio.onDataPacketReceived( ({ receivedString: name, receivedNumb
er: value }) => {
    if (name == "bilar") {
        for (let i = 0; i < value; i++) {
            basic.showLeds(`
                . . . . .
                . . . . .
                . # # # .
                # # # # #
                . # . # .
            `)
            basic.pause(300)
            basic.clearScreen()
            basic.pause(300)
        }
    }
    if (name == "cyklar") {
        for (let i = 0; i < value; i++) {
            basic.showLeds(`
                . # . . .
                . # . # #
                . # # . .
                # # . # #
                # # . # #
            `)
```

```
            basic.pause(300)
            basic.clearScreen()
            basic.pause(300)
        }
    }
})
```

LJUD

För att få micro:bit att skapa ljud behövs någon form av högtalare.

En enkel lösning är att koppla en hörlurar sladd till micro:bit med två krokodilklämmor. När du drar ut ett block från Musik-kategorin visas i simulatorn hur du kopplar sladdarna.



TILLBEHÖR

Andra sätt är med hjälp av olika tillbehör:

Audio-sladd med krokodilklämmor – Länk

Mi:power board (har en inbyggd buzzer) – länk

BATTERI

Om du vill köra din micro:bit utan USB-sladd kan ett batteri vara en alternativ strömkälla.

Det finns olika sätt att koppla en batteri till micro:bit, varav flera är olika tillbehör. Det går också att koppla en 3 volt strömkälla till `3v`(+) och `gnd`(-) pins.

LÄNKAR

http://mermicrobit.se/lar-dig-mer/

http://mermicrobit.se/lar-dig-mer/

Microbit

http://mermicrobit.se/komma-igang-med-microbit/vad-ar-microbit/

http://mermicrobit.se/

filmer

https://www.youtube.com/playlist?list=PLD0HD_3AJljWGkt3q4cfzHrrRvQBW9nOy

http://www.microbit-i-skolen.dk/438968204

http://www.microbit-i-skolen.dk/

Andra länkar:

https://www.hackster.io/microbit

https://www.youtube.com/results?search_query=Micro%3Abit+Basics+for+Teachers+-+Part+2%3A+Programming+with+Javascript+Blocks

https://www.microbit.co.uk/blocks/contents

https://www.youtube.com/watch?v=RkWDYTx_mg4