Programmering och algebra

Introduktion

Under de här lektionerna kommer eleverna att lära sig ett programmeringsspråk som heter BSL via en webbsida som heter WeScheme.
Alla program är redan gjorda och du har länkar till dem i denna handledning.

Språket är till sin uppbyggnad likt matematikens uttrycksformer. T.ex används variabler på samma sätt och funktionerna spelar nästa samma roll.
I matematiken ska de ges möjligheter att utveckla kunskaper i att använda digitala verktyg och programmering för att undersöka problemställningar, matematiska begrepp och göra beräkningar.

Målet med lektionerna är att skriva funktioner i ett program som styr hastigheten på två bilar som startar på varsin ända av en lång väg med en given sträcka.
De ska även förändra i programmet så att bilarna inte krockar. Du kan även visa kollisionen i en graf eftersom din version av programmet plottar bilarnas färd. Tidsåtgång är ca 120 minuter.

Eleverna behöver ha tillgång till dator eller iPad. Eftersom programmet är webbaserat fungerar det på alla digitala plattformar. Man kan programmera utan inloggning.

För att eleverna ska kunna förstå och skriva funktioner i programmet behöver de ha kunskap om hur man skriver beräkningar och definitioner. Det finns arbetsmaterial till detta.

Elevernas arbetshäfte börjar med prioriteringsregler. De förutsätter att du har haft första delen av lektionen med dem.

WeScheme

1. Prioriteringsregler

Delmål är att kunna identifiera koordinaterna på figurerna i en bild och kunna omvandla flera aritmetiska uttryck mellan flera representationer samt känna till och förstå begrepp som är knutna till programmeringen
Matematik är ett språk, precis som engelska, spanska eller svenska. Vi använder oss av substantiv som ”bröd”, ”tomat”, ”ost” och ”senap” för att beskriva fysiska objekt. Matematik har värden, tex siffror för att beskriva kvantiteter.

Vi använder oss också av verb som ”kasta”, ”springa”, ”hoppa” som beskriver användningen av substantiven. Matematik har funktioner som addition och subtraktion, vilka styr vad som ska utföras med siffrorna.

Ett matematiskt uttryck är som en mening, det är en instruktion för att göra något. Uttrycket 6 + 4 talar om för oss att vi ska addera 6 och 4. För att utvärdera ett uttryck följer vi instruktionerna i uttrycket. Uttrycket 6 + 4 utvärderar vi till 10.

Ibland behöver vi använda oss av flera uttryck för att kunna utföra en uppgift. I programmering är de stegvisa instruktionerna viktiga. Precis som funktionsordningen är viktig i matematiken. Om någon säger ” fem plus tre minus en ”, kan det innebära flera saker:
- Addera fem och tre och subtrahera sedan med en (5 + 3) - 1
- eller addera fem till resultatet av tre subtraherat med ett 5 + (3 - 1)

Låt eleverna träna på att skriva uttryck som kan betyda olika saker, men har samma svar.

Om man har ett matematiskt uttryck som innehåller flera olika räknesätt eller parenteser, då kan resultatet påverkas av i vilken ordning man gör de olika räkneoperationerna. Vilket skulle kunna ge oss bekymmer, då vi ofta delar uträkningar mellan oss människor. Om till exempel en ingenjör räknar på ett sätt och en annan ingenjör på annat sätt, då skulle detta kunna leda till att byggnader eller broar blev felkonstruerade och rasade samman och människor kom till skada.

Matematiker har inte alltid varit överens om arbetsordningen, men nu har vi en gemensam uppsättning regler för i vilken ordning de olika räkneoperationerna ska utföras.

1. Parenteser
2. Multiplikation och division
3. Addition och subtraktion

Låt eleverna träna på att använda prioriteringsreglerna för uttryck i deras arbetshäfte.

2. Räknecirklar

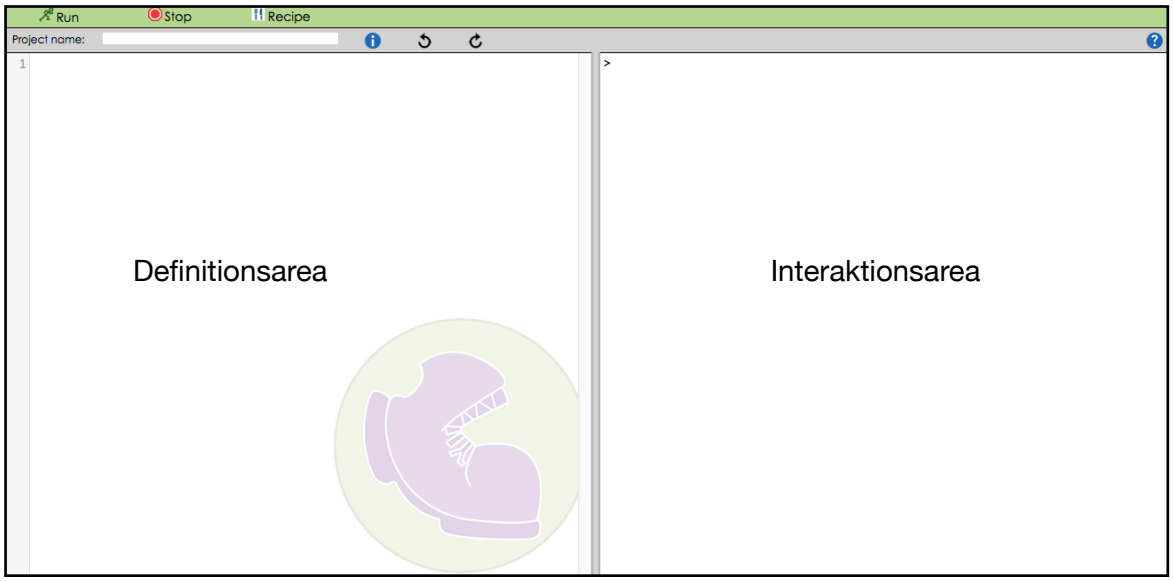
Ett sätt att ange ordningsföljden i ett uttryck är att först rita uttrycket som ett diagram. Det diagrammet vi ska använda oss av kallas räknecirkel. Dessa cirklar visar strukturen som händer inom ett uttryck, vilket är viktigt att veta om man ska skriva en funktion som utför beräkningar i WeScheme. Alla cirklar har två regler:

1. Varje cirkel måste ha en funktion som ligger högst upp i cirkeln.
2. Numren är skrivna nedan, i ordning från vänster till höger.

Låt eleverna arbeta med räknecirklarna i häftet.

3. Dags att börja programmera i WeScheme

Vi kommer att använda ett program som heter wescheme och är webbaserat. Du behöver inget konto för att kunna programmera.
När du kommer in så kommer det att se ut så här. Det som man möts av kallas editor och visas nedan. Här finns två stora rutor: definitions area och interactions area.
Definitionsområdet är där programmerare definierar värden och funktioner i sitt program, medan området interaktioner gör att de kan experimentera med dessa värden och funktioner.



När ett program körs producerar det ett värde. Precis som när man skriver meningar så finns det vissa regler som bestämmer om det man skriver verkar vettigt. Program har också regler.

Regel nummer 1. Alla värden är riktiga uttryck.
Följande finns i elevernas arbetshäfte. Välj om ni ska göra det tillsammans eller ej.
Låt eleverna prova att skriva siffran 7 eller något annat tal i interaktions area för att se vad som händer. Vad händer om de skriver sitt namn?

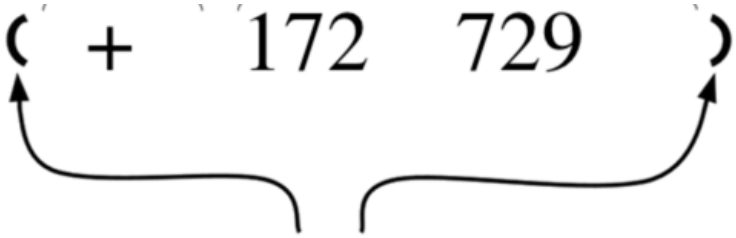
Felmeddelanden som kommer upp är användbara för programmerare. Snarare än att säga "det här programmet fungerar inte", gör programmet sitt bästa för att berätta vad som gick fel och att ge dig så mycket information som möjligt för att hjälpa dig att åtgärda problemet. Påminn eleleverna om att läsa och använda felmeddelanden! För de kan verkligen hjälpa när man ska att gå vidare.

Regel nummer 2. Varje öppen parentes följs av en funktion, därefter med ett eller flera riktiga uttryck, och slutligen av en slutlig parentes.

Alla uttryck som kommer efter namnet på funktionen kallas argument.

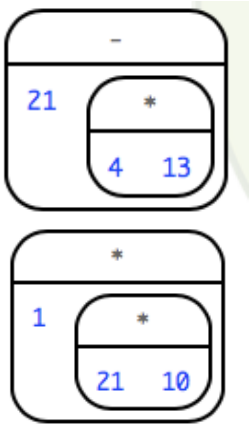
Namn, kommer alltid direkt efter den första öppna parentesen.

Argument, vad funktionen ska göra.



Varje funktion har en parantes som öppnar och stänger uttrycket.

Låt eleverna skriva följande i kod på pappret och i WeScheme.



_____

_____

4. Att skriva kontrakt för program

Målet är att utveckla elevernas förståelse för hur en funktion byggs upp i WeScheme och andra program är det bra att börja att tänka på vad funktionen ska göra, vad den ska heta, vilken typ av data den förväntar sig samt vilken typ av data den producerar.

Då är det bra att arbeta med kontrakt. Det tydliggör vilken typ av data funktionen förväntar sig och vad den producerar.

Tex: Kontraktet för / (division)

Programmet förväntar sig två siffror och producerar en siffra.

namn: divide domaine: number number range: number

på engelska Domain. Men ni måste välja! namn - name? domän - domain? bild -range?

Programmet kan också förvänta sig en radie, information om cirkeln är fylld eller ej, samt färgen och sedan producera en blå, fylld cirkel med radien 30. Tänk på att programmet använder engelska ord, så att cirkel skrivs circle.

Namn: circle **Domaine**: nummer string string -> **Range:** image

Programmet kan se ut så här:

(circle 30) (circle 30 ”solid” ”blue”)

Låt eleverna arbeta med snabba funktioner som det kallas (dvs man skriver exempel som det ovan och även träna på att skriva kontrakt på arbetsblad ….)

5. Programmera bilar

Målet är att eleverna ska beräkna efter hur lång tid det tar för två bilar att kollidera och sedan skriva koden så att programmet förhindrar en kollision.

Eleverna har detta i sitt arbetshäfte också så att de kan arbeta själva eller i grupp.
Du kan även ha samtal om bilen med klassen. Gör i så fall så hör:

Starta programmet i Wescheme via länk nedan. Välj edit för att kunna se koden. För att starta programmet tryck på run i menyn. :

En bil

Observera lite stav fel i kommentarer iprogrammet. Också, skriv en kommentar för vad 13 står för i main!

Samtala med eleverna, gärna enligt någon kooperativ modell, som tex först par sedan alla (tidsbegränsat, gärna någon minut bara) om vad som ändras hos bilen, samt mer specifikt vad det är.

Bilen kör från ena sidan till den andra.
I programmet rör sig bilen med en hastighet av tre pixlar per klocktick i programmets klocka. Det går 28 klocktick på en sekund.

Låt eleverna lösa hur många pixlar det blir per sekund.
(Det går 28 x3 pixlar på en sekund. Hastigheten blir alltså 84 pixlar/s)

Nästa uppdrag är att låta eleverna förändra villkoret i funktionen ”crash t” så att den förhindra att bilen kör över trädet. Bakgrunden i programmet är en rektangel. Koden för den ser ut så här: (rectangle (50 600 ”solid” ”green”)).
De ändrar i koden i programmet och skriver sin förklaring till vad programmet gör i den rosamarkrerade texten innan.

Två bilar

Berätta att du har ett program med två bilar som rör sig. De ha olika hastigheter och har olika startpunkter. En börjar längst ut till vänster och en längst ut till höger och de åker rakt mot varandra.

Visa hur bilarna kör genom att gå in på länken ” Två bilar krock lärare”. Välj läge ”run” så att eleverna bara ser vad som händer och inte koden. Du har fullständig kod i din länk. Eleverna behöver komplettera i sin kod för att programmet ska fungera.

Två bilar krock lärare
Två bilar krock elev

Samtala med eleverna. Vad är det som händer?
Hur tror de att programmet vet var bilarna ska starta?
Har bilarna samma hastighet? Om inte, vilken är snabbast?
Kan man räkna ut var en bil befinner sig vid en viss tidpunkt?
Vad måste man veta för att kunna räkna ut det?

Låt eleverna arbeta i sitt arbetsmaterial med att definiera så att programmet kan räkna ut var bilarna befinner sig i x-led vi tiden t, dvs samma sak som ni gjorde tillsammans tidigare.

```
(define (car1-pos t) ( + CAR1-X0 ( * _ _) )
  )
(define (car2-pos t) ( + CAR2-X0 ( * _ _ ))
  )
```

Säg att en bil rör sig med en hastighet av 7 pixlar/sekund och att den har färdats i 4 sekunder.
Den utgår från tid 0 och läge 0. Det innebär att bilen vid tiden 4 sekunder har åkt:

0 + 4s * 7pixlar långt .

Så här ser definitionen av startläget för varje bil ut i programmet.

```
(define CAR1-X0 0)
(define CAR2-X0 500)
```

Vad händer om man förändrar startvärdet ? (den bil som får ändrat värde kommer att starta vid en annan punkt). Därför är det viktigt att definiera var bilen startar i programmet så att man kan beräkna var de befinner sig.

Låt eleverna arbeta med sitt arbetsmaterial. De ska testa med både negativa och positiva tal och sedan bestämma vilken hastighet de ska ha genom att de ska gå "lagom" fort, dvs de ska kunna följa deras färd, samt ha olika hastigheter och börja från var sin kant.

Kollision
Till sist ska eleverna försöka att ändra i stop-villkoret (crash t) för att förhindra en krock. De behöver använda kunskaperna från övningen med stop-villkor i ”en bil”-övningen för att lösa problemet.

I din version av programmet finns ytterligare ett sätt att visualisera när en kollision inträffar. Varje bils kan beskrivas med en graf med sträckan som en funktion av tiden. Om du raderar de semikolon som finns vid varje rosa textrad så visar programmet en graf. Du har nu tillfälle att samtala om grafer, räta linjer och skärningspunkter mellan grafer!

Begreppslista

- *contract*: a statement of the name, domain, and range of a function
- *define*: associate a descriptive name with a value
- *definitions window*: the text box at the top of the Editor (DrRacket or WeScheme), where definitions for values and functions are written
- *design recipe*: a sequence of steps that help people document, test, and write functions
- *domain*: the type of data that a function expects
- *example*: shows the use of a function on specific inputs and the computation the function should perform on those inputs
- *function definition*: code that names a function, lists its variables, and states the expression to compute when the function is used
- *name*: how we refer to a function or value defined in a language (examples: +, *, star, circle)
- *range*: the type of data that a function produces
- *type*: refers to a general kind of data, like Number, String, Image, or Boolean
- *value*: a specific piece of data, like 5 or "hello"
- *variable name*: name of the information that can be different each time a function is used