Kapitel 4: Listor, grafik och dynamiska program

Tillämpning: Sierpinskitriangeln

I denna applikation ska vi utveckla ett spel, *Kaosspelet*, som genererar den s.k. Sierpinskitriangeln, en berömd fraktal.

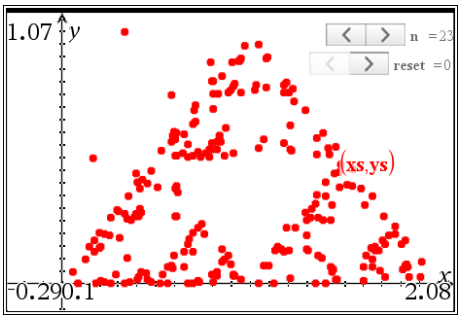
Syfte:

- Skriva ett program som genererar en intressant bild (Sierpinski-triangeln)
- Styra tillåtna åtgärder hos ett skjutreglage

Sierpinskitriangeln

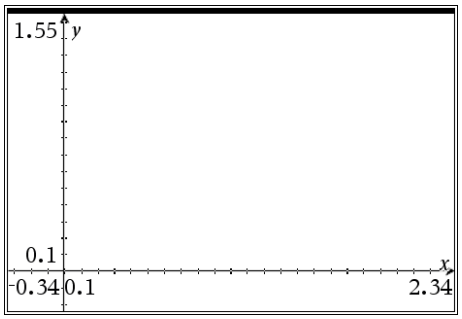
Sierpinskitriangeln är en *fraktal* som bildas genom att starta med en liksidig triangel och succesivt ta bort ”mitten-trianglar” som i bilden ovan. Denna procedur kan göras i all oändlighet så att triangeln till slut blir oändligt “hålad” och varje del av bilden är likformig med bilden av sig själv (*självlikformighet*).

Ett annat sätt att skapa triangeln är att köra ett spel, **Kaosspelet**. Här är reglerna för detta spel:



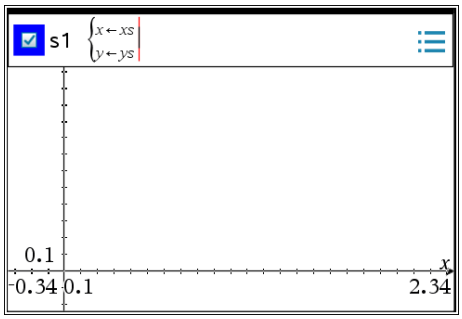
1. Välj tre (3) punkter i planet som bildar de tre hörnen (0, 0), (2, 0), and (1, 1) i en triangel. Vi ser att detta *inte* är en liksidig triangel.
2. Börja med att slumpmässigt välja någon punkt (helst inuti triangeln men det spelar egentligen ingen roll) och betrakta denna punkt som din nuvarande position.
3. Välj slumpmässigt något av de tre hörnen.
4. Flytta halva avståndet från din nuvarande position till det valda hörnet (det betyder att du beräknar mittpunkten mellan din nuvarande position och hörnet).
5. Plotta denna nya position.
6. Börja om från 3.

Vi ska nu använda ett växande spridningsdiagram för att visuellt representera detta spel. Längs vägen i detta projekt ska vi lära oss hur man **återställer (reset)** ett spridningsdiagram och hur man förhindrar ett skjutreglage från att gå bakåt (inaktivera vänsterpil eller nedåtpil hos ett minimerat skjutreglage).



1. Starta ett nytt dokument och infoga appen **Grafer**.
2. Flytta origo till det nedre vänstra hörnet och sträck ut axlarna så att x-axeln spänner från0 till drygt 2 och y-axeln från 0 till en bit ovanför 1. Vår triangels hörn är ju (0, 0), (2, 0) and (1, 1) och de ligger nu inne i fönstret. Se bild till höger.

3. Pröva att ta tag i hela koordinatsystemet för att flytta origo och ta också tag i axlarna för att sträcka ut/in skalan på *x*-axeln.

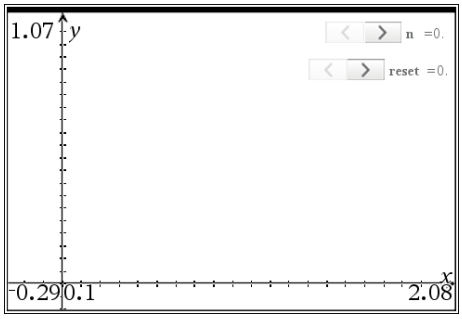


Det är en bra idé att spara ditt dokument nu och sedan spara det då och då (använd Crtl-s) eftersom något kan gå snett under arbetet med detta ganska stora dokument.

4 Förbered plottning av ett spridningsdiagram med variablerna **xs** och **ys**. (Meny > Grafinmatning/Redigera > Spridningsdiagram). Dessa variabler är ännu ej definierade så du kommer inte att se något diagram när du trycker på [enter].

Infoga *två* skjutreglage (**Meny > Åtgärder > Infoga skjutreglage**):

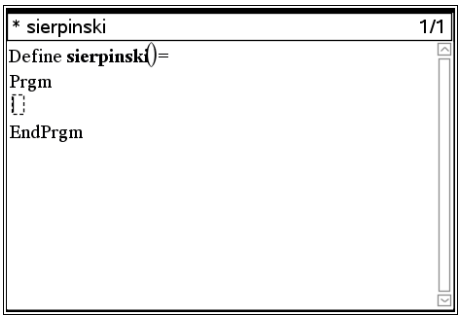
- **n** – värde 0, Minimum 0, Maximum 2000, Stegstorlek 1, horisontell and minimerad.
- **reset** – värde 0, Minimum 0, Maximum 1, Stegstorlek 1, horisontell and minimerad.



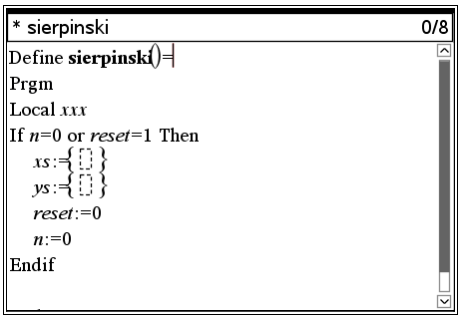
5 Placera skjutreglagen i skärmens övre högra hörn så att de inte skymmer triangeln.

Grafappen är nu klar.

Programmering



1. Infoga en sida med Programeditorn. Välj Nytt.
2. Döp programmet till **sierpinski**.
3. Lägg till en sats för lokala variabler (**Local**) i fall det behövs några hjälpvariabler senare i programmet.
   - Använd tillfälligt variabeln **xxx** som en platshållare.
4. Börja koden med the 'reset'-konceptet.
   - När skjutreglaget för **reset** används ska alla data i listorna *xs* och *ys* raderas och variabeln **reset** och variabeln **n** ska ställas tillbaka till 0.
   - Första gången som programmet försöker köras, initiera det (när **n** är 0) (och reset inte är 1).



Att sätta **reset:=0** i denna del av programmet kan tyckas vara ett konstigt val men resultatet blir att värdet på **reset** på skärmen alltid verkar vara 0.
Att klicka på **reset**-knappen orsakar att **reset** blir 1, ”tvingar” programmet att köras och omedelbart därefter sätts värdet hos **reset** tillbaka till 0 (tillsammans med andra saker i denna del av programmet.)

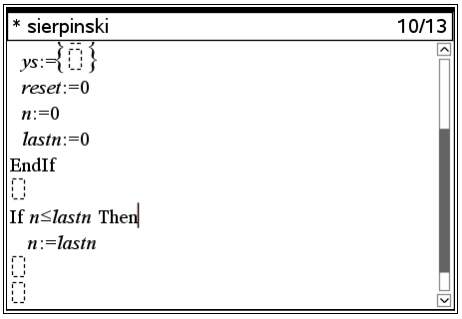
Hindra n från att minska

Vi vill att värdet hos **n** (antalet punkter som ritas) att öka och inte minska. Gör detta genom att hålla koll på det senaste (*föregående*) värdet hos **n** och jämför det med nya (*nuvarande*) värdet på **n**. Om det nya värdet är mindre än det sista värdet, kommer det att ignoreras genom att ange **n** att vara det sista värdet.

1. Använd variabeln **lastn** att hålla det föregående värdet på **n**. Detta händer precis i slutet av programmet:

```
lastn := n
EndPrgm
```

Nästa del av koden kontrollerar om **n** är mindre än det föregående **n** (när man klickar på vänsterpilen hos skjutreglaget).



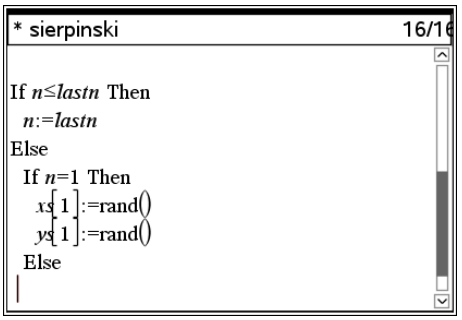
2. Efter initieringsavsnittet så lägger du till:

```
If n < lastn  Then
n := lastn
Else
```

3. Initiera också **lastn** i den första **If** strukturen:

```
lastn:=0
```

4. Därefter ska vi skapa listor för spridningsdiagrammet.
   - Den första datapunkten är speciell eftersom det kan vara vilken punkt som helst. Om n är lika med 1 så tilldelar vi ett slumptal till xs[1] och ys[1]:



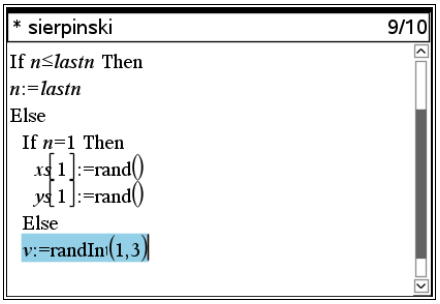

```
Else
 If n = 1  Then
   xs[1] := rand()
   ys[1] := rand()
 Else
```

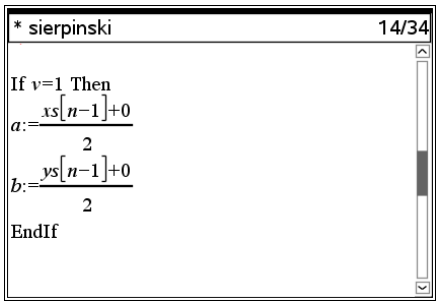
Slumptalsfunktionen rand( ) alstrar ett slumpmässigt decimaltal mellan 0 och 1 så denna punkt är verkligen en slumpmässig punkt i *kvadraten* mellan (0, 0) och (1, 1).

Vi är nu redo att ge oss på själva kärnan i algoritmen som vi nämnde alldeles i början. Här en kort repetition för att friska upp minnet:

3. Välj slumpmässigt något av de tre hörnen.
4. Flytta halva avståndet från din nuvarande position till det valda hörnet (det betyder att du beräknar mittpunkten mellan din nuvarande position och hörnet).
5. Plotta denna nya position. *Börja sedan om från steg 3*.

5. Använde **v := randint(1,3)** för att välja ett slumpmässigt heltal som hjälper till att skilja mellan de tre hörnen.
6. Konstruera nu en uppsättning av If-satser, baserade på v, för att beräkna nästa mittpunkt.
   - Kom ihåg att de tre hörn vi valt att använda är (0, 0), (2, 0), och (1, 1).
7. Om v = 1, använd godtyckligt punkten (0, 0) och den sista punkten i listorna för att beräkna en mittpunkt.
   - Från geometrin vet vi att koordinaterna hos mittpunkten mellan två punkter (x1, x2) och (y1, y2) är (x1+x2)/2 och (y1+y2)/2.





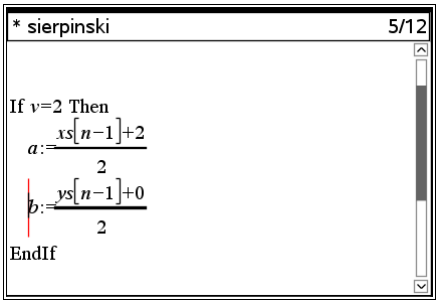
På detta ställe i programmet så är de sista värdena av **xs** och **ys xs[n-1]** och **ys[n-1]** eftersom vi hamnade här genom att öka **n** men ännu inte lagt till ytterligare en punkt till listorna.

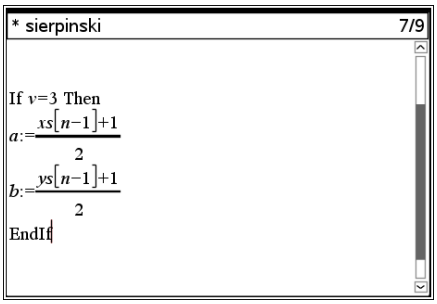
Denna analys utmynnar i koden du ser i skärmbilden till höger. **v**, **a** och **b** är temporära lokala variabler.

8. Alldeles i början av programmet så ska du redigera satsen om lokala variabler till

   Local v,a,b

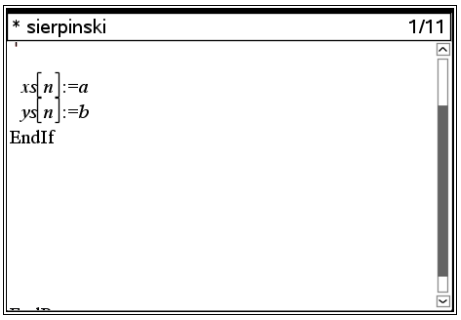
Vi kan nu enkelt kopiera denna **If**-struktur för att hantera de andra två hörnen, (2, 0) and (1, 1).



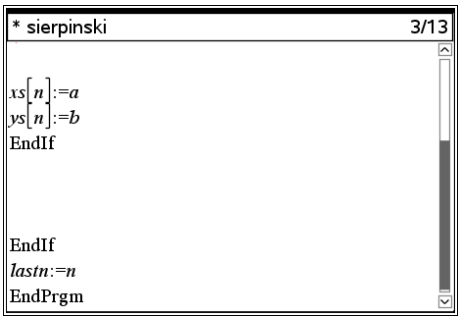


Obs: En mer effektiv struktur skulle här vara If…Then…ElseIf…Else…EndIf-strukturen. Undersök nu om du kan genomföra denna del av programmet med strukturen ovan istället.

9. Lägg till värdena hos **a** och **b** i slutet av våra två listor.
   - Den n:te positionen är faktiskt ett steg efter slutet i listorna och det tillåts alltid.



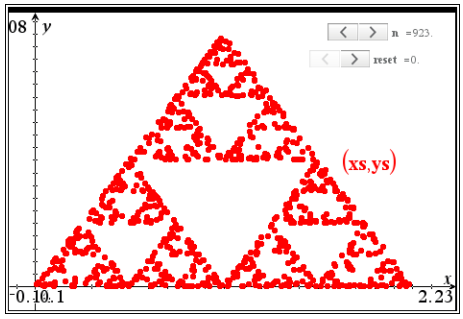
Om du har valt **If**-strukturerna från Kontrollmenyn och infogat blocken på rätt plats så får du ett till EndIf i slutet av programmet just före **lastn:=n**.



10. Tryck **ctrl-B** för att 'Kontrollera syntax & lagra.
    - Om det finns några fel måste du kontrollera din kod noggrant. En komplett programlistning finns nedan.

11. Infoga appen **Anteckningar** och infoga sedan en **Matematikruta** (ctrl-M). Skriv namnet på programmet, en vänsterparentes och tryck [enter].

12. Gå nu till **Graf**-appen, och pröva de två skjutreglagen.
    - **n** lägger till punkter i spridningsdiagrammet och **reset** ska ta bort alla punkter och sätta tillbaka värdet på **n** till 0.
    - Vänsterpilen på reglaget för **n** ska inte fungera och vänsterpilen på reglaget för **reset** ska alltid vara avaktiverad.
    - **reset** ska alltid visas som 0 eftersom programmet upptäcker dess ändring till 1 och då omedelbart ändrar tillbaka till 0.
    - Värdet hos **n** är begränsat till maximala värdet i inställningarna för skjutreglaget.
    - Ju fler punkter du plottar ju mer liknar den bild som du får på skärmen Sierpinskis triangel.



```
Define sierpinski()=
Prgm
Local v, a, b
If n=0 or reset=1 Then
  xs:={ }
  ys:={ }
  reset:=0
  n:=0
  lastn:=0
EndIf
If n≤lastn Then
   n:=lastn
Else
   If n=1 Then
     xs[1]:=rand()
     ys[1]:=rand()
   Else
      v:=randInt(1,3)
     If v=1 Then
        a:=((xs[n-1]+0)/2)
        b:=((ys[n-1]+0)/2)
     EndIf
     If v=2 Then
        a:=((xs[n-1]+2)/2)
        b:=((ys[n-1]+0)/2)
     EndIf
     If v=3 Then
         a:=((xs[n-1]+1)/2)
         b:=((ys[n-1]+1)/2)
     EndIf
     xs[n]:=a
     ys[n]:=b
   EndIf
EndIf
lastn:=n
EndPrgm
```