Gissa mitt hemliga tal med programmering

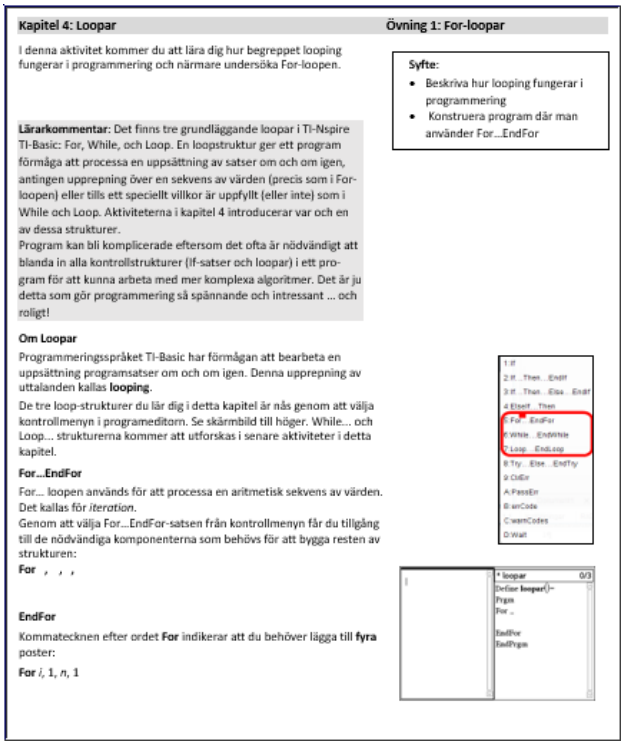
I denna aktivitet ska vi jämföra 2 metoder för att söka rätt på ett hemligt tal. Talet slumpas fram av TI-Nspire´s slumptalsgenerator med kommandot **talet:=randInt(1,*n*)** där *n* är talstorleken. randInt(1,100) betyder ett slumptal mellan 1 och 100. Slumptalet lagras i variabeln *talet*.

Problem 1

```
Define gissa_ett_tal()=
Prgm
Request "hur många tal",n
talet:=randInt(1,n)
For i,1,n
Disp "Min gissning är",i
If i=talet Then
Disp "rätt"
Exit
Else
 Disp "Gissa igen"
EndIf
Wait 0.5
EndFor
EndPrgm
```

Här har vi ett ganska enkelt program som söker fram det hemliga talet genom att söka i ordning 1, 2, ..osv fram till det eftersökta talet. Det kallas för *linjär* sökning.

Här finns en For-sats, **For i,1,n** som gör att programmet söker igenom alla tal från 1 fram till det rätta talet. Om talet är rätt avslutas sökningen med kommandot **Exit**. Pröva att ta bort Exit-raden i programmet och se vad som händer. I **10 minutes of Code, Kap 4 övning 1,** som behandlar Loopar, kan man läsa mer om For EndFor-satser.



Kapitel 4: Loopar — **Övning 1: For-loopar**

I denna aktivitet kommer du att lära dig hur begreppet looping fungerar i programmering och närmare undersöka For-loopen.

Syfte:
- Beskriva hur looping fungerar i programmering
- Konstruera program där man använder For...EndFor

Lärarkommentar: Det finns tre grundläggande loopar i TI-Nspire TI-Basic: For, While, och Loop. En loopstruktur ger ett program förmåga att processa en uppsättning av satser om och om igen, antingen upprepning över en sekvens av värden (precis som i For-loopen) eller tills ett speciellt villkor är uppfyllt (eller inte) som i While och Loop. Aktiviteterna i kapitel 4 introducerar var och en av dessa strukturer.
Program kan bli komplicerade eftersom det ofta är nödvändigt att blanda in alla kontrollstrukturer (If-satser och loopar) i ett program för att kunna arbeta med mer komplexa algoritmer. Det är ju detta som gör programmering så spännande och intressant ... och roligt!

Om Loopar

Programmeringsspråket TI-Basic har förmågan att bearbeta en uppsättning programsatser om och om igen. Denna upprepning av uttalanden kallas **looping**.

De tre loop-strukturer du lär dig i detta kapitel är nås genom att välja kontrollmenyn i programeditorn. Se skärmbild till höger. While... och Loop... strukturerna kommer att utforskas i senare aktiviteter i detta kapitel.

For...EndFor

For... loopen används för att processa en aritmetisk sekvens av värden. Det kallas för *iteration*.
Genom att välja For...EndFor-satsen från kontrollmenyn får du tillgång till de nödvändiga komponenterna som behövs för att bygga resten av strukturen:

For , , ,

EndFor

Kommatecknen efter ordet **For** indikerar att du behöver lägga till **fyra** poster:

For *i*, 1, *n*, 1

Kommandot **Wait** gör att du hinner se hur programmet arbetar vid programkörningen. Pröva gärna olika värden.

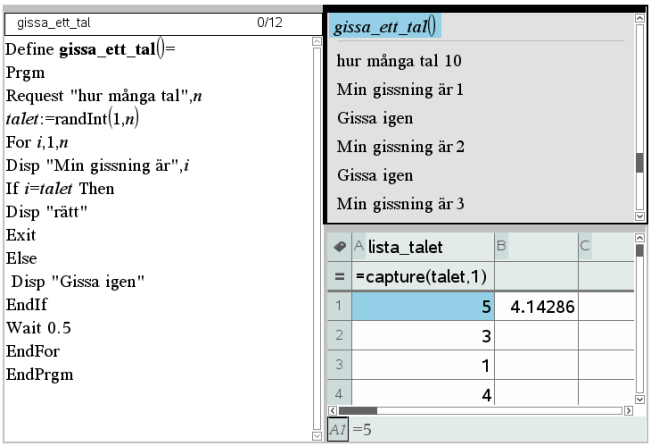
Detta är ett ganska enkelt program. **Request**-satsen gör att man i en dialogruta får skriva in hur många tal man som ska ingå i sökningen.

Gissa mitt hemliga tal

Denna aktivitet handlar om att gissa ett tal som slumpas fram av TI−Nspire´s slumptalsgenerator. Först har vi ett enklare program som gissar talet genom direkt uppräkning. *Ingen* strategi alltså. Vi kan kalla detta för en *linjärsökning*.

In computer science, linear search or sequential search is a method for finding a target value within a list. It sequentially checks each element of the list for the target value until a match is found or until all the elements have been searched (Wikipedia)

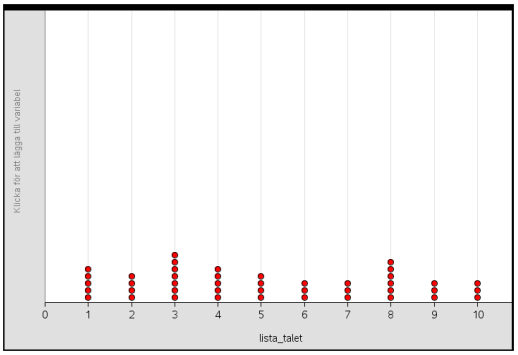
I kalkylarket så sparas alla resultat från gissningarna i en lista genom en speciell funktion, *capture*. Genomför ett större antal gissningar utifrån samma talstorlek och titta på resultatet på sid 3. Hur tror du fördelningen blir? Hur många sökningar blir i genomsnitt om du har tio tal, 1−10.



Så här ser sidan 2 ut med programmet till vänster, programkörningsfönstret uppe till höger och sedan ett kalkylark där vi ”samlar upp” resultaten från alla exekveringar. I cell b1 redovisar vi medelvärdet. Vad blir medelvärdet efter många körningar?

Man kan naturligtvis dela upp sidan på annat sätt. Om man går till Redigera/Sidlayout/Dela grupp får man tre sidor.

Sid 3 visar i diagramform hur fördelningen ser ut. Den sidan blir bara intressant efter många programkörningar.



Problem 2

Här gör vi nu en annan typ av sökning som brukar kallas binärsökning.

Binärsökning är en algoritm för att avgöra om en mängd innehåller ett givet element. Sökningen utförs i flera steg och i varje steg skall man utesluta att halva den kvarvarande mängden innehåller elementet och därmed kunna koncentrera sig på den andra halvan. (Wikipedia)

Gissa mitt hemliga tal med binärsökning

I problem 2 ska vi också gissa ett tal men nu har vi lägger vi en kontroll "är talet för högt" eller "är talet för lågt". Pröva med samma antal tal som i förra försöket och gör sedan en jämförelse. Gör ett antal gissningar och titta på hur programmet fungerar. Jämför sedan med koden. Titta till exempel på raden

$$\text{gissning:=ceiling}\left(\frac{\text{högsta+lägsta}}{2}\right)$$

Ceiling ger resultatet "det närmaste heltal som är större än eller lika med argumentet".

Fundera över varför vi har denna sats i programmet.

Lägg nu till ett kalkylark och en Data&Statistiksida som i problem 1 och gör en analys av skillnaderna mellan de två programmen. Hur många gissningar behöver du i genomsnitt göra om du gissar på tal mellan 1 och 10.

Om vi har 10 tal och om vi läser koden från början kommer vi till satsen för den första gissningen. Här utnyttjas en funktion som heter *ceiling* (kallas ibland takvärde). Den ger som vi skriver i Nspire-dokumentet det närmaste heltalet som är större än eller lika med argumentet.

$$\text{ceiling}\left(\frac{5+10}{2}\right) \blacktriangleright 8$$

Medelvärdet av 5 och 10 ovan blir ju 7,5 men vi kan i detta program inte jobba med heltal som funktionen ger ett svar som är nästa heltal.

Låt oss nu säga att vi att det hemliga talet är något av heltalen 1, … 10. Enligt binärsökningsmetoden ska vi då börja med en första gissning som är

$$\text{ceiling}\left(\frac{0+10}{2}\right)=5$$

Vi säger nu att vårt hemliga tal är större än 5. Då blir vår lägsta gissning 5 och högsta 10. Detta ger nu ett nytt värde på variabeln gissning:

$$\text{ceiling}\left(\frac{5+10}{2}\right)=8$$

Om det också är för lågt blir nästa gissningsberäkning:

$$\text{ceiling}\left(\frac{8+10}{2}\right)=9 \quad \text{osv.}$$

Så här kan det då se ut i programkörningen.

```
gissa_talet()
Hur många tal? 10
Min gissning är 5
för lågt!
Min gissning är 8
för lågt!
Min gissning är 9
Rätt! med 3 gissningar
                    Klar
```

Så här ser hela programkoden ut. Den ser lång ut men detta är också ett program som håller reda på alla gissningar och sedan räknar ut hur många gissningar som behövs för att komma fram till rätt svar.

```
Define gissa_talet()=
Prgm
Local talet,n,i,lägsta,högsta,gissning
Request "Hur många tal?",n
talet:=randInt(1,n)
lägsta:=0
högsta:=n
gissning:=ceiling((lägsta+högsta)/2)
antal_gissningar:=1
Disp "Min gissning är",gissning
While gissning≠talet
If gissning<talet Then
Disp "för lågt!"
lägsta:=gissning
Else
Disp "för högt!"
högsta:=gissning
EndIf
gissning:=ceiling((högsta+lägsta)/2)
antal_gissningar:=antal_gissningar+1
Disp "Min gissning är",gissning
Wait 0.1
EndWhile
Disp "Rätt! med ",antal_gissningar,"gissningar"
EndPrgm
```

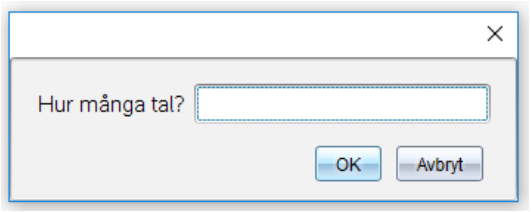
På nästa sida tittar vi på några viktiga delar av koden. Vi har gulmarkerat detta i koden ovan.

Request ”Hur många tal?,*n*

Hos TI-Nspire™ CX finns det två liknande satser som tillåter dig att mata in värden till programmet *medan det körs*. De kallas för inmatningskommandon. Ett av dessa kommandon är

Request “meddelande”, variabel (*för numerisk inmatning*)

Du hittar dessa satser i I/O-menyn i program-editorn. Se nedan.



While *gissning* ≠ *talet*

.
.
.

EndWhile

En While...EndWhile-*slinga* upprepar ett block av kommandon så länge ett specificerat villkor är sant. I detta fall så länge villkoret

gissning ≠ talet är sant.

Syntaxen för While-kommandot är:

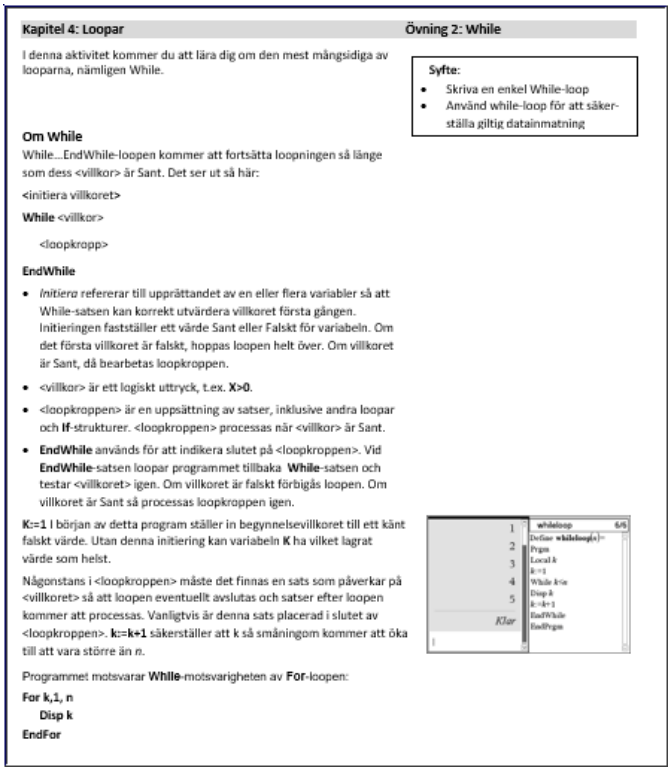
While *villkor*

När While exekveras utvärderas villkor. Om villkor är sant exekveras slingan. I annat fall hoppar kontrollen till det kommando som kommer efter EndWhile.

Obs: While-kommandot ändrar inte villkoret automatiskt. Du måste inkludera kommandon som låter funktionen eller programmet gå ur slingan.

I slutet av slingan (EndWhile) hoppar kontrollen tillbaka till While-kommandot där villkoret utvärderas på nytt.

I kapitel 4 övning 2 bland **10 minutes of Code**-aktiviteterna finns en ordentlig genomgång av hur man kan använda While. Det finns tre grundläggande loopar i TI-Nspire TI-Basic: For, While, och Loop. En loopstruktur ger ett program förmåga att processa en uppsättning av satser om och om igen, antingen upprepning över en sekvens av värden (precis som i For-loopen) eller tills ett speciellt villkor är uppfyllt (eller inte) som i While och Loop. Aktiviteterna i kapitel 4 introducerar var och en av dessa strukturer.


Kapitel 4: Loopar **Övning 2: While**

I denna aktivitet kommer du att lära dig om den mest mångsidiga av looparna, nämligen While.

Syfte:
- Skriva en enkel While-loop
- Använd while-loop för att säkerställa giltig datainmatning

Om While

While...EndWhile-loopen kommer att fortsätta loopningen så länge som dess <villkor> är Sant. Det ser ut så här:

<initiera villkoret>
While <villkor>
  <loopkropp>
EndWhile

- *Initiera* refererar till upprättandet av en eller flera variabler så att While-satsen kan korrekt utvärdera villkoret första gången. Initieringen fastställer ett värde Sant eller Falskt för variabeln. Om det första villkoret är falskt, hoppas loopen helt över. Om villkoret är Sant, då bearbetas loopkroppen.
- <villkor> är ett logiskt uttryck, t.ex. **X>0**.
- <loopkroppen> är en uppsättning av satser, inklusive andra loopar och **If**-strukturer. <loopkroppen> processas när <villkor> är Sant.
- **EndWhile** används för att indikera slutet på <loopkroppen>. Vid **EndWhile**-satsen loopar programmet tillbaka **While**-satsen och testar <villkoret> igen. Om villkoret är falskt förbigås loopen. Om villkoret är Sant så processas loopkroppen igen.

K:=1 I början av detta program ställer in begynnelsevillkoret till ett känt falskt värde. Utan denna initiering kan variabeln **K** ha vilket lagrat värde som helst.

Någonstans i <loopkroppen> måste det finnas en sats som påverkar på <villkoret> så att loopen eventuellt avslutas och satser efter loopen kommer att processas. Vanligtvis är denna sats placerad i slutet av <loopkroppen>. **k:=k+1** säkerställer att k så småningom kommer att öka till att vara större än *n*.

Programmet motsvarar **While**-motsvarigheten av **For**-loopen:

For k,1, n
  Disp k
EndFor

Villkorssatser

För en struktur som ser ut så här

If *villkor* **Then**
Block av kod
Else
Block av kod
EndIf

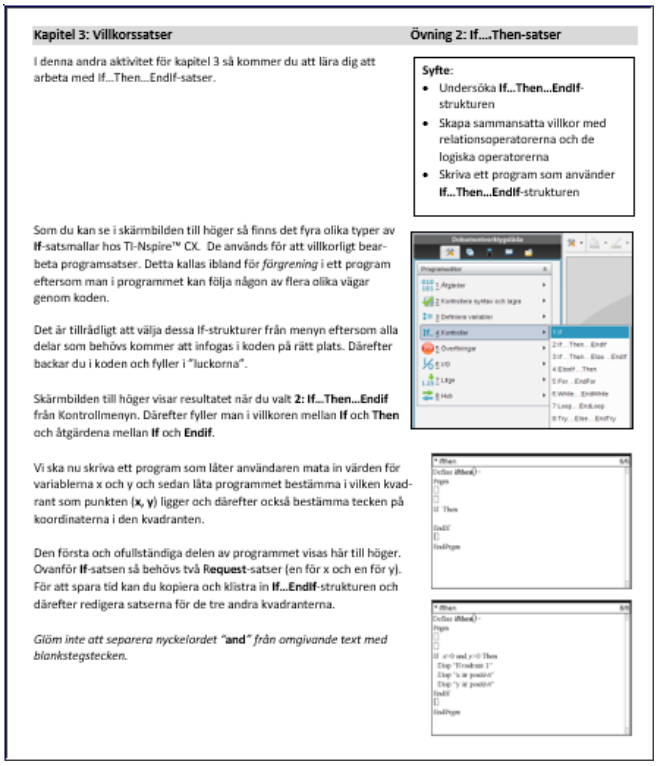
så exekveras det första blocket med kod om villkoret är sant och det andra blocket med kod om villkoret inte är sant.

I programmet förekommer följande villkorsstruktur:

```
If gissning<talet Then
Disp "för lågt!"
lägsta:=gissning
Else
Disp "för högt!"
högsta:=gissning
EndIf
```

Ibland är det alltså nödvändigt att ha en åtgärd som inträffar när ett villkor är sant och en annan åtgärd när samma villkor är falskt. Det är detta man kan åstadkomma med en *Else*-sats

Det mesta som rör villkorssatser kan man studera närmare i kapitel 3 övning 2 bland **10 minutes of Code**-aktiviteterna.



Kapitel 3: Villkorssatser — **Övning 2: If....Then-satser**

I denna andra aktivitet för kapitel 3 så kommer du att lära dig att arbeta med If...Then...EndIf-satser.

Syfte:
- Undersöka **If...Then...EndIf**-strukturen
- Skapa sammansatta villkor med relationsoperatorerna och de logiska operatorerna
- Skriva ett program som använder **If...Then...EndIf**-strukturen

Som du kan se i skärmbilden till höger så finns det fyra olika typer av **If**-satsmallar hos TI-Nspire™ CX. De används för att villkorligt bearbeta programsatser. Detta kallas ibland för *förgrening* i ett program eftersom man i programmet kan följa någon av flera olika vägar genom koden.

Det är tillrådligt att välja dessa If-strukturer från menyn eftersom alla delar som behövs kommer att infogas i koden på rätt plats. Därefter backar du i koden och fyller i "luckorna".

Skärmbilden till höger visar resultatet när du valt **2: If...Then...EndIf** från Kontrollmenyn. Därefter fyller man i villkoren mellan **If** och **Then** och åtgärdena mellan **If** och **EndIf**.

Vi ska nu skriva ett program som låter användaren mata in värden för variablerna x och y och sedan låta programmet bestämma i vilken kvadrant som punkten **(x, y)** ligger och därefter också bestämma tecken på koordinaterna i den kvadranten.

Den första och ofullständiga delen av programmet visas här till höger. Ovanför **If**-satsen så behövs två **Request**-satser (en för x och en för y). För att spara tid kan du kopiera och klistra in **If...EndIf**-strukturen och därefter redigera satserna för de tre andra kvadranterna.

*Glöm inte att separera nyckelordet "**and**" från omgivande text med blankstegstecken.*

I slutet av programmet finns även ett räkneverk:

antal_gissningar:=antal_gissningar+1

som ser till att beräkna hur många gånger vi behöver gissa för att få rätt svar:

	A lista_gissning	B	C
=	=capture(antal_gissningar,1)		
1	4	3.	
2	3		
3	4		
4	1		
5	2		
6	4		
7	3		
8	4		
9	2		

B1 =approx(mean(lista_gissning))

Om du kör programmet med 10 tal och sedan med funktionen *capture* samlar upp alla resultat i kalkylarket, kan det se ut så som i skärmbilden nedan. I cell b1 visar vi medelvärdet.

Fråga: Diskutera hur många gissningar man måste göra som mest för att vara säker på att hitta det gömda talet. Jämför med linjär sökning som i problem 1.

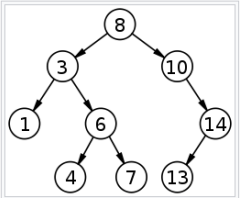
Pröva också med att göra sökningar med större tal än 10.

På nätet finns det mycket att läsa om binärsökning.

Binärsökning [redigera | redigera wikitext]

Binärsökning är en algoritm för att avgöra om en mängd innehåller ett givet element. Sökningen utförs i flera steg och i varje steg skall man utesluta att halva den kvarvarande mängden innehåller elementet och därmed kunna koncentrera sig på den andra halvan.

Intervallhalveringsmetoden är en term som används om både binärsökning och den matematiska problemlösningsmetoden i Bolzanos sats.



Binärsökningsträd med 9 noder och höjden 4.

Ett exempel på binärsökning är uppslagning av ett ord eller namn i en alfabetiskt ordnad lista, till exempel en tryckt telefonkatalog eller en ordbok. Man börjar med att titta i mitten av listan. Genom att jämföra det sökta ordet med det som står i mitten av listan, vet man vilken halva av listan man ska fortsätta med. Efter andra uppslagningen återstår bara en fjärdedel av listan.

Om hela listan har N uppslagsord, krävs högst $\lceil \log_2 N \rceil$ uppslagsningar eller halveringar för att hitta rätt ställe, eller 2-logaritmen avrundad uppåt.

Antal noder N	2-logaritmen $\log_2 N$	avrundad uppåt $\lceil \log_2 N \rceil$	Kommentar
9	3,17	4	Grafen i bilden här intill har 9 noder och höjden 4.
900	9,81	10	
1 953 033	20.8	21	Alla artiklar i svenskspråkiga Wikipedia (januari 2015).
9 miljoner	23,1	24	En katalog över hela Sveriges befolkning.
6 miljarder	32,5	33	En katalog över hela jordens befolkning.

Här har vi en sökning på talen 1–10 000 000! (Storleksordningen är antal invånare i Sverige)

Rätt svar efter 22 gissningar. Se nedan. Diskutera varför det aldrig behövs mer än 24 gissningar.

Hur många tal? 10000000
Min gissning är 5000000
för lågt!
Min gissning är 7500000
för lågt!
Min gissning är 8750000
för lågt!
Min gissning är 9375000
för högt!
Min gissning är 9062500
för lågt!
Min gissning är 9218750
för lågt!
Min gissning är 9296875
för lågt!
Min gissning är 9335938
för lågt!
Min gissning är 9355469
för högt!
Min gissning är 9345704
för lågt!
Min gissning är 9350587
för lågt!
Min gissning är 9353028
för högt!
Min gissning är 9351808
för högt!
Min gissning är 9351198
för lågt!
Min gissning är 9351503
för lågt!

Min gissning är 9351656
för högt!
Min gissning är 9351580
för lågt!
Min gissning är 9351618
för högt!
Min gissning är 9351599
för högt!
Min gissning är 9351590
för högt!
Min gissning är 9351585
för lågt!
Min gissning är 9351588

Rätt! med 22 gissningar