Kapitel 4: Loopar

Tillämpning: Bankmeddelanden

Denna tillämpning använder loopar för att mata in så många värden som behövs, och som en utvidgning, kontrollerar för giltiga värden som matats in och använder If-satser för att visa ett lämpligt meddelande.

Lärarkommentar: Det här projektet visar tendensen mot större komplexitet vid utveckling av en mjukvara. Med "nästling" menas här att man att man sätter en kontrollstruktur inuti en annan. Såsom förklaras nedan ”vet” programvaran TI-Nspire vilket End som hör ihop med vilken sats.

Syfte:

- Använda räkneverk och ackumulatorsatser.
- Använda en loop i ett program för att få en obestämd mängd av data
- Använda ett ”flagg”-värde för att avsluta en loop.

Nästlade Strukturer

- *Nästling* är en programmeringsteknik för att placera en kontrollsats inne i en annan.
- Termen härrör från idén att placera en kartong inne i en annan för att spara plats.
- En programmerare placerar loopar inom loopar, If-strukturer inuti loopar och loopar inuti If-strukturer för att utföra mer komplicerade uppgifter.
- Det är viktigt att placera en komplett struktur helt och hållet inom ett block med en annan för att undvika fel.
- Programlistningen till höger visar en **While** -loop och en **If** struktur innanför en annan **While** loop.
- Observera att en **EndWhile**-sats förekommer två gånger. Datorn ”vet” vilken **EndWhile** som hör till vilken **While**.
- Indenteringen (indragen) i programmet är bara för att göra koden lättläst och logiken tydligare.

```
perfekt_kvadrat                    12/15
Define perfekt_kvadrat()=
Prgm
a:=1
While a>0
 Request "Mata in ett tal",a
 While a<0
  Disp "Måste vara icke negativt!"
  Request "Mata in ett positivt tal",a
 EndWhile
 s:=√(a· 1.)
 If s=int(s) Then
  Disp "Det är en perfekt kvadrat"
 Else
  Disp "det är inte en perfekt kvadrat"
 EndIf
 Disp "Dess kvadratrot är",s
EndWhile
EndPrgm
```

Programmet sätter först in en loop som ska fortsätta tills det angivna värdet är noll. Sedan testar det för att se om **a < 0**. Om det är så visar programmet "Måste vara icke-negativt!" och frågar sedan efter ett annat värde för **a**. Men när **a** inte är negativt så beräknas kvadratroten, och en If-sats, och två Disp-satser utförs.

Sammanfattning av de tre looparna:

For(*var*, *start*, *slut***)** **While** <*villkor*> **Loop**

……. If <*villkor*> **: Exit**

EndFor **EndWhile** **EndLoop**

For(används när man ”räknar upp” eller processar en aritmetisk sekvens av värden (iteration).

While används när du har möjlighet att helt hoppa över loopkroppen.

Loop används när du är säker på att du vill att loopkroppen ska ”köras” åtminstone en gång och den måste innehålla en Exit-sats, vanligtvis som en del av en If-sats.

Lärarkommentar: Den enda nödvändiga loopen är While-loopen. Den kan utföra samma uppgifter som de andra två looparna. Använd inte GoTo här för att avsluta en Loop-struktur och inte heller i något annat program. Det är dålig programmering! Det finns andra kontrollstrukturer i Kontroll-menyn i program-editorn som vi inte tar upp i denna serie av aktiviteter. Du kan läsa mer om dessa i *Referensguiden* för TI-Nspire.
Variabeln ***α*** multipliceras med **1.** i kvadratrotsberäkningen för att säkerställa att du får samma resultat i både den numeriska plattformen och CAS-plattformen. Om du använder en decimalpunkt i ett uttryck så ”tvingar” du fram ett resultat i approximativ form.

Kapitel 4: Program "Bankmeddelanden"

En bankkund kan ha flera bankkonton på en viss bank. Banken kräver ett minsta genomsnittligt saldo på 1000 kr på alla konton för att undvika att betala serviceavgifter. Om genomsnittet är mellan 1000 kr och 1250 kr skickar banken kunden ett meddelande med en varning om serviceavgift.
Om genomsnittet är mer än 1250 kr skickar banken kunden ett meddelande om att saldot ligger klart över gränsen. Vi ska nu skriva ett program som ger användarinformation om konton. Användaren matar in saldon för sina konton. Programmet kommer att beräkna genomsnittligt saldo och visa antalet konton, genomsnittliga saldovärdet och ett meddelande för användaren. Meddelandena kan t.ex. vara: "Serviceavgift tas ut", "Risk för serviceavgift" och "Tack för att du använder våra tjänster”

Vi kan använda två metoder för att mata in ett okänt antal värden:

- Metod 1: fråga efter det totala antalet konton först och använd en **For**-loop för att mata in värdena.
- Metod 2: fråga efter belopp men använd ett s.k. flaggvärde, t.ex. -999, för att indikera att det inte finns fler konton. Denna metod använder en **While**-loop eller en **Repeat** -loop.

I båda metoderna kommer vi att ha ett *löpande* totalvärde för de värden som matats in.

I metod 2 måste vi också räkna *antalet* konton så att vi sedan kan dividera totalvärdet med antalet. Följande information ska då programmet visa:

1) antalet konton, 2) genomsnittligt saldo 3) meddelande till kunden utifrån genomsnittsvärdet på saldona. Se nedan.

Om genomsnittet är under 1000 kr: “Serviceavgift tas ut”

Om genomsnittet är 1000 till 1250 kr: “Risk för serviceavgift”

Om genomsnittet är över 1250 kr: “Tack för att du använder våra tjänster”

Lärarkommentar: I nästa avsnitt diskuteras två programmeringsidéer som hör ihop: ett räkneverk och en ackumulatorvariabel. Understryk att variabeln på den vänstra sidan av tilldelningsoperatorn (:=) är densamma som den på högra sidan men att de representerar olika värden beroende på involverade processer.

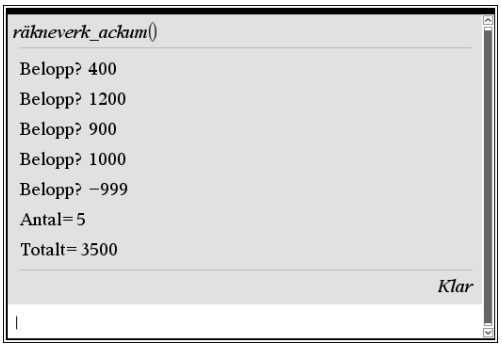
Räkneverk och ackumulatorer

En sats som **c:=c+1** kallas ett räkneverk eftersom den ilägger till 1 till variabeln **c** varje gång som satsen exekveras.

En sats som **t:=t+n** kallas en *ackumulator* eftersom den löpande håller reda på totalen för variabeln **n**. Värdet hos **n** adderas till variabeln **t** och sedan lagras den summan tillbaka till variabeln **t**. I slutet av en loop kommer **t** att innehålla totalen av **n**-värdena.

Här är ett exempel som använder ett räkneverk, en ackumulator och ett ”flagg-värde” (-999) för att hålla reda på antalet belopp som matats in i programmet.

Prgm	Kommentarer
Local *räkneverk*,*belopp*,*totalt*	initiera variabler
totalt:=0	
räkneverk:=1	
Request "Belopp?",*belopp*	första beloppet
While *belopp*≠−999	så länge det inte är -999
räkneverk:=*räkneverk*+1	addera 1 till räkneverket
totalt:=*totalt*+*belopp*	lägg till belopp till totalt
Request "Belopp?",*belopp*	fråga efter nytt belopp
EndWhile	
Disp "Antal=",*räkneverk*	
Disp "Totalt=",*totalt*	
EndPrgm	



While loopen ovan fortsätter räkna och ackumulera belopp så länge-999 inte matas in. När-999 matas in visas stoppar Loopen och resultaten visas.

Lärarkommentar: Titta närmare på de två **Request**-satserna i koden ovan. Den första ger det första beloppet och den sista ger resten av beloppen. Den sista finns i *slutet* av loopen för att förbereda för att gå tillbaka till **While**-satsen och testa värdet på *belopp* igen.

Utvidgning

Som en del av inmatningsrutinen ska du se till att värden som inmatas är giltiga (större än 0). Om inmatat värde ligger utanför detta intervall ska lämplig åtgärd vidtas.

Lärarkommentar: Utvidgningen kräver en annan loop runt var och en av inmatningssatserna för att kontrollera att det angivna värdet är giltigt. Exempelkoden på nästa sida stoppar helt enkelt programmet när ett ogiltigt värde har angetts. Många elever kan säkert hitta bättre lösningar.

Här är ett förslag på lösning:

```
Prgm
local belopp, räkneverk, totalt, genomsnitt
0→ belopp
1→räkneverk
0→totalt
Request "MATA IN  ett belopp: ", belopp
While belopp ≠ -999
 If belopp <0 Then
  Disp "UTANFÖR GILIGT INTERVALL!"
  Stop
 ElseIf
  räkneverk+1→räkneverk
  totalt+ belopp →totalt
 End
 Request "ETT NYTT belopp (ELLER -999): ", belopp
EndWhile
totalt/räkneverk→genomsnitt
Disp "ANTAL konton:", räkneverk
Disp "TOTALT saldo:", totalt
Disp "GENOMSNITT saldo:",genomsnitt
If genomsnitt<1000 Then
 Disp “Serviceavgift tas ut”
ElseIf genomsnitt<1250 Then
 Disp “Risk för serviceavgift”
Else
 Disp “Tack för att du använder våra tjänster”
EndIf
EndPrgm
```

När du testar ett program så börja med att prova med extrema fall, t.ex. som att mata in -999 som första värde och sedan negativa värden varsomhelst.

Om programmet ger ett felmeddelande så har programmeraren inte tagit hänsyn till alla scenarion. Programmet fungerar inte om du matar in -999 som första värde eftersom det kommer att försöka dividera 0 med 0 som inte är definierat.

Beräkningen av genomsnittsvärdet ska infogas i ett test för att säkerställa att åtminstone ett belopp har matats in. En möjlig lösning är:

```
If räkneverk >0
Then
 genomsnitt:=totalt/räkneverk
ElseIf
 Disp “INGET BELOPP ÄR INMATAT!”
 Stop
End
```

Stop-satsen används i detta program för att avsluta ett program omedelbart. Man kan lägga till en sådan sats var som helst i ett program så att exekveringen avbryts och meddelandet ”Klar” visas.