Hoofdstuk 9

Subroutines

Tijdens het schrijven van programma's moeten we heel vaak dezelfde serie stappen telkens opnieuw uitvoeren. In die gevallen is het waarschijnlijk overbodig om deze instructies telkens opnieuw te schrijven. En daarom is het handig om *subroutines* te gebruiken.

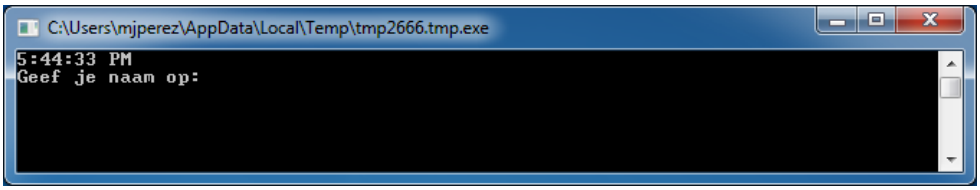
Een subroutine is een stukje code binnen een groter programma dat normaal gesproken iets specifieks doet en dat waar dan ook in het programma kan worden opgeroepen. Subroutines kun je herkennen aan een naam die volgt op het sleutelwoord **Sub** en worden afgesloten door het **EndSub**-sleutelwoord. In het volgende fragment wordt met de subroutine *PrintTime* de huidige tijd naar TextWindow geschreven.

```
Sub PrintTime
  TextWindow.WriteLine(Clock.Time)
EndSub
```

Hieronder volgt een programma met een subroutine die vanaf verschillende plaatsen wordt opgeroepen.

```
PrintTime()
TextWindow.Write("Geef je naam op: ")
naam = TextWindow.Read()
TextWindow.Write(naam + ", Het is nu: ")
PrintTime()

Sub PrintTime
  TextWindow.WriteLine(Clock.Time)
EndSub
```



Afbeelding 44 – Een eenvoudige subroutine oproepen

Je kunt een subroutine uitvoeren door *SubroutineName()* op te roepen. Zoals gewoonlijk heb je de leestekens '()' nodig om de computer te vertellen dat je een subroutine wilt uitvoeren.

Voordelen van subroutines

Zoals we hierboven al hebben gezien, hoef je met subroutines niet zoveel code meer te typen. Als je de *PrintTime*-subroutine eenmaal hebt geschreven, kun je deze waar dan ook in het programma oproepen om de huidige tijd naar het tekstvenster te schrijven.

> *SmallBasic-subroutines kunnen alleen worden opgeroepen binnen hetzelfde programma. Je kunt subroutines niet oproepen vanuit een ander programma.*

Ook kun je met subroutines gecompliceerde problemen opbreken in kleinere, minder moeilijke delen. Als je bijvoorbeeld een complexe vergelijking moet oplossen, kun je meerdere subroutines schrijven die kleinere delen van de complexe vergelijking oplossen. Vervolgens kun je alle resultaten samenvoegen om de oorspronkelijke complexe vergelijking op te lossen.

Met subroutines wordt ook de leesbaarheid van het programma verbeterd. Anders gezegd, als je goed benoemde subroutines gebruikt voor bepaalde gedeeltes van je programma die vaak worden uitgevoerd, is je programma beter te lezen en te begrijpen. Dit is erg belangrijk als je het programma van iemand anders wilt begrijpen of als je wilt dat anderen jouw programma begrijpen. Soms is het zelfs nuttig voor het lezen van je eigen programma als het al een tijdje geleden is dat je het voor het laatst hebt gelezen.

Variabelen gebruiken

Binnen een subroutine kun je elke variabele gebruiken die je in een programma hebt. In het volgende voorbeeld worden er twee getallen geaccepteerd en wordt de grootste van de twee naar het tekstvenster geschreven. Is het je opgevallen dat de variabele *max* binnen en buiten de subroutine wordt gebruikt?

```
TextWindow.Write("Geef het eerste getal op: ")
getal1 = TextWindow.ReadNumber()
TextWindow.Write("Geef het tweede getal op: ")
getal2 = TextWindow.ReadNumber()
```

```
FindMax()
TextWindow.WriteLine("Het hoogste nummer is: " + max)

Sub FindMax
  If (getal1 > getal2) Then
    max = getal1
  Else
    max = getal2
  EndIf
EndSub
```

En de uitvoer van dit programma ziet er als volgt uit.

Afbeelding 45 – Met een subroutine het hoogste getal uit twee getallen verkrijgen

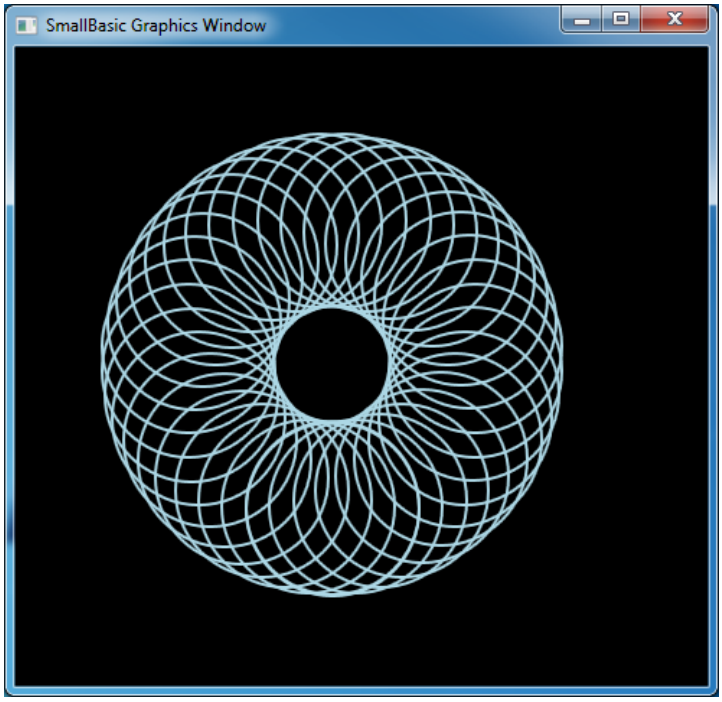
Laten we eens kijken naar een ander voorbeeld waarin het gebruik van subroutines wordt toegelicht. Deze keer gaan we een grafisch programma gebruiken waarmee verschillende punten worden berekend die worden opgeslagen in de variabelen *x* en *y*. Vervolgens wordt een subroutine met de naam **DrawCircleUsingCenter** opgeroepen waarmee een cirkel wordt getekend met *x* en *y* als het middelpunt.

```
GraphicsWindow.BackgroundColor = "Black"
GraphicsWindow.PenColor = "LightBlue"
GraphicsWindow.Width = 480
For i = 0 To 6.4 Step 0.17
  x = Math.Sin(i) * 100 + 200
  y = Math.Cos(i) * 100 + 200

  DrawCircleUsingCenter()
EndFor

Sub DrawCircleUsingCenter
  startX = x - 40
  startY = y - 40

  GraphicsWindow.DrawEllipse(startX, startY, 120, 120)
EndSub
```



Afbeelding 46 – Grafisch voorbeeld voor subroutines

Subroutines oproepen binnen lussen

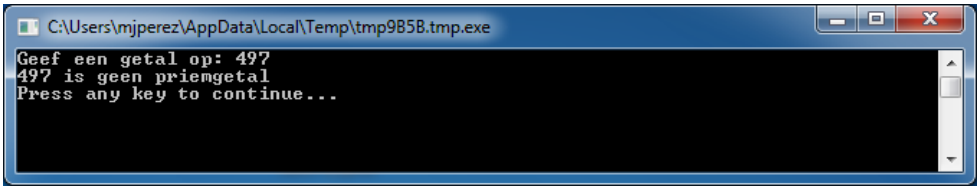
Soms worden subroutines opgeroepen binnen een lus. Als dit wordt gedaan, worden dezelfde serie instructies uitgevoerd, maar met andere waarden in een of meer variabelen. Je hebt bijvoorbeeld een subroutine met de naam *PrimeCheck* en met deze subroutine kun je bepalen of een getal een priemgetal is of niet. Je kunt een programma schrijven dat de gebruiker een waarde laat opgeven waarvan jij dan met deze subroutine kunt zeggen of het een priemgetal is of niet. In het programma hieronder wordt dit duidelijk gemaakt.

```
TextWindow.Write("Geef een getal op: ")
i = TextWindow.ReadNumber()
isPriemgetal = "Waar"
PrimeCheck()
If (isPriemgetal = "Waar") Then
 TextWindow.WriteLine (i + " is een priemgetal")
Else
 TextWindow.WriteLine (i + " is geen priemgetal")
EndIf

Sub PrimeCheck
 For j = 2 To Math.SquareRoot(i)
 If (Math.Remainder(i, j) = 0) Then
  isPriemgetal = "Onwaar"
```

```
  Goto EndLoop
 EndIf
 Endfor
EndLoop:
EndSub
```

Met de PrimeCheck-subroutine wordt de waarde van *i* gedeeld door kleiner getallen. Als een getal deelbaar is door *i* en er is geen rest, dan is *i* geen priemgetal. De waarde van *isPriemgetal* wordt ingesteld op 'Onwaar' en de subroutine wordt afgesloten. Als het getal niet deelbaar is door kleinere getallen, dan blijft de waarde van *isPriemgetal* 'Waar'.



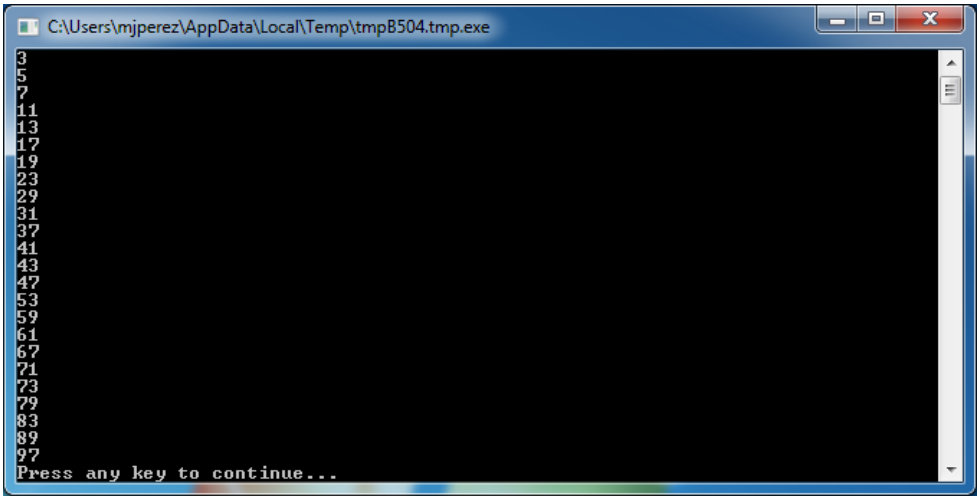
Afbeelding 47 – Priemgetallen testen

Nu je een subroutine hebt waarmee je priemgetallen kunnen testen, kun je deze routine gebruiken om een lijst te maken van alle priemgetallen onder bijvoorbeeld 100. Je kunt het bovenstaande programma heel eenvoudig aanpassen om *PrimeCheck* vanuit een lust op te roepen. Hiermee wordt met de subroutine telkens een andere waarde berekend wanneer de lus wordt uitgevoerd. In het voorbeeld hieronder wordt uitgelegd hoe je dit moet doen.

```
For i = 3 To 100
  isPriemgetal = "Waar"
  PrimeCheck()
  If (isPriemgetal = "Waar") Then
    TextWindow.WriteLine(i)
  EndIf
EndFor

Sub PrimeCheck
  For j = 2 To Math.SquareRoot(i)
    If (Math.Remainder(i, j) = 0) Then
      isPriemgetal = "Onwaar"
      Goto EndLoop
    EndIf
  Endfor
EndLoop:
EndSub
```

In het bovenstaande programma wordt de waarde van *i* telkens bijgewerkt wanneer de lus wordt uitgevoerd. Binnen de lus wordt de subroutine *PrimeCheck* opgeroepen. De subroutine *PrimeCheck* neemt de waarde van *i* en berekent of *i* een priemgetal is of niet. Het resultaat wordt opgeslagen in de variabele *isPriemgetal*. Met de lus buiten de subroutine wordt vervolgens toegang gezocht tot deze variabele. De waarde van *i* wordt naar het tekstvenster geschreven als het een priemgetal is. En aangezien de lus begint bij 3 en tot 100 doorgaat, krijgen we een lijst met alle priemgetallen tussen 3 en 100. Het resultaat van het programma vind je hieronder.



Afbeelding 48 – Priemgetallen