Microsoft Small Basic

Inleiding in programmeren

Hoofdstuk 1

Een inleiding

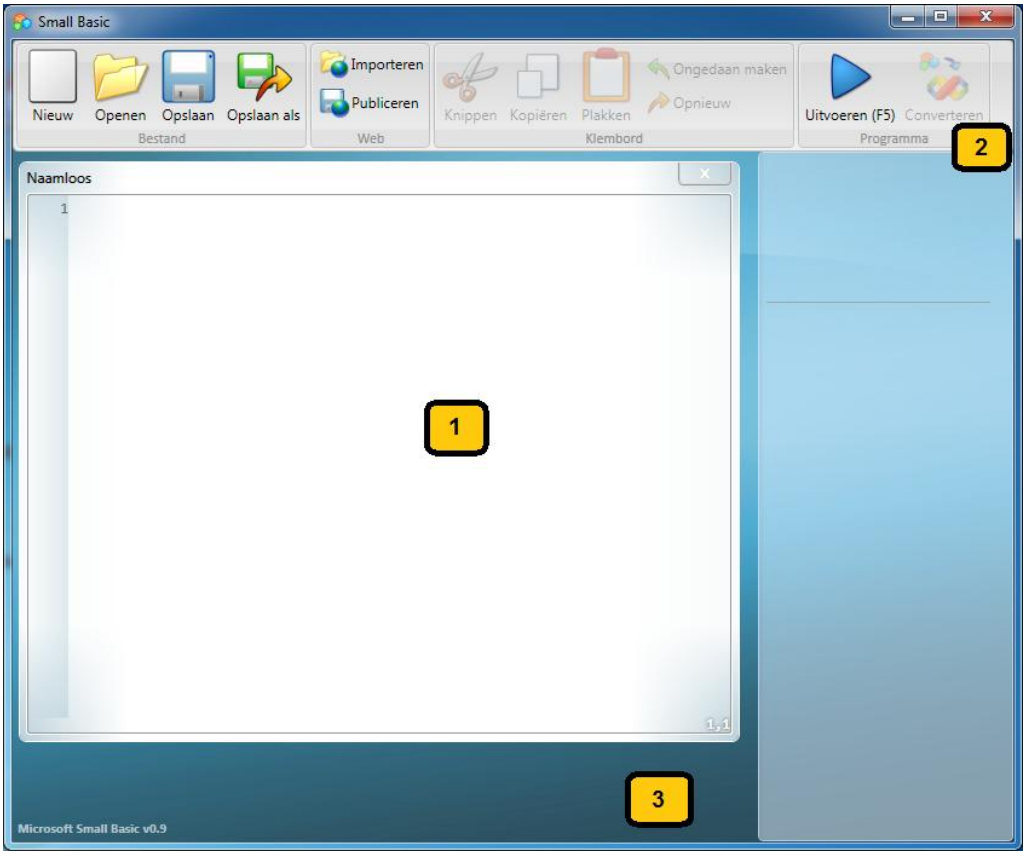
Small Basic en programmeren

Programmeren is het proces waarmee met programmeertalen computersoftware wordt gemaakt. Net zoals wij Engels, Spaans of Frans begrijpen en spreken, kunnen computers programma's begrijpen die in een bepaalde taal zijn geschreven. Dit zijn de zogenaamde programmeertalen. In het begin waren er slechts enkele programmeertalen en deze waren gemakkelijk te leren en te begrijpen. Maar omdat computers en software steeds gecompliceerder werden, moesten er snel meer gecompliceerde concepten in programmeertalen worden opgenomen. Als gevolg hiervan zijn de meeste moderne programmeertalen en de achterliggende concepten voor een beginner steeds moeilijker te begrijpen. Dit weerhoudt mensen van het leren of uitproberen van programmeren.

Small Basic is een programmeertaal die is ontworpen om programmeren voor beginners heel gemakkelijk, toegankelijk en leuk te maken. Het is de bedoeling om met Small Basic de barrières te verwijderen en een springplank te bieden naar de verbazingwekkende wereld van programmeren.

De Small Basic-omgeving

Laten we beginnen met een korte inleiding in de Small Basic-omgeving. Als je Small Basic voor het eerst start, zie je een venster dat er uitziet als in de volgende afbeelding.



Afbeelding 1 – De Small Basic-omgeving

Dit is de Small Basic-omgeving. Hier gaan we onze Small Basic-programma's schrijven en uitvoeren. Deze omgeving heeft enkele verschillende elementen die worden aangegeven met nummers.

In de **Editor**, aangegeven met [1], gaan we onze Small Basic-programma's schrijven. Als je een voorbeeldprogramma of een eerder opgeslagen programma opent, wordt het weergegeven in deze editor. Je kunt het hier wijzigen en opslaan voor later gebruik.

Je kunt ook meerdere programma's tegelijkertijd openen en bewerken. Elk programma waarmee je werkt, wordt in een aparte editor weergegeven. De editor die het programma bevat dat je momenteel bewerkt, wordt de *actieve editor* genoemd.

De **Werkbalk**, aangegeven met [2], wordt gebruikt om opdrachten te geven aan de *actieve editor* of de omgeving. We zullen meer leren over de verschillende opdrachten in de werkbalk als we ermee aan de slag gaan.

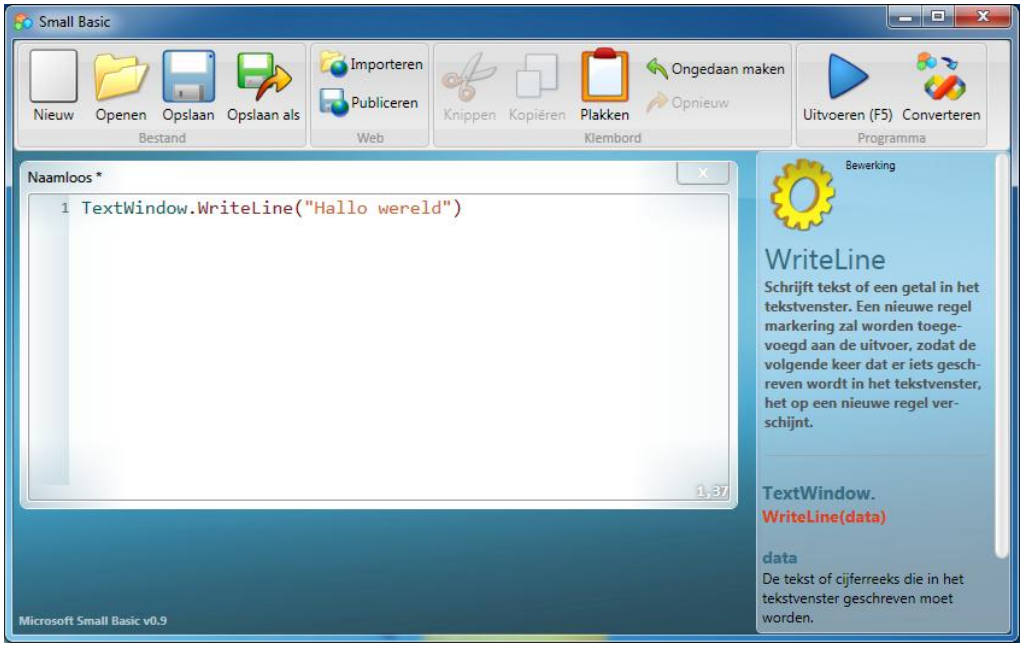
Het **oppervlak**, aangegeven met [3], is de plek waar we alle editorvenster plaatsen.

Ons eerste programma

Nu dat je bekend bent met de omgeving van Small Basic gaan we beginnen met programmeren. Zoals we hierboven al hebben opgemerkt, gaan we onze programma's schrijven in de editor. Laten we daarom de volgende regel in de editor typen.

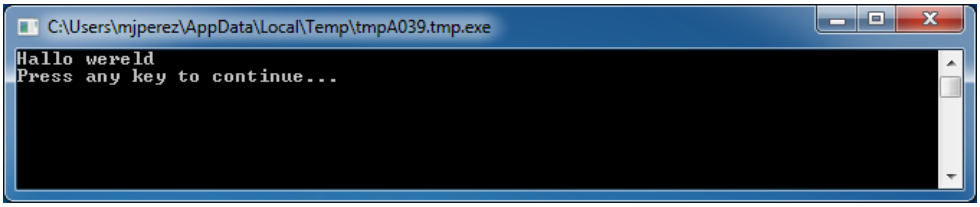
```
TextWindow.WriteLine("Hallo wereld")
```

Dit is ons eerste Small Basic-programma. En als je het goed hebt getypt, zie je iets soortgelijks als in de afbeelding hieronder.



Afbeelding 2 – Eerste programma

Nu dat we ons nieuwe programma hebben getypt, kunnen we het uitvoeren om te zien wat er gaat gebeuren. We kunnen ons programma uitvoeren door te klikken op de knop *Uitvoeren* op de werkbalk of met de sneltoets F5 op het toetsenbord. Als alles goed gaat, wordt ons programma uitgevoerd met het resultaat zoals hieronder wordt weergegeven.



Afbeelding 3 – Uitvoer van eerste programma

Gefeliciteerd! Je hebt zojuist je eerste Small Basic-programma geschreven en uitgevoerd. Een heel klein en eenvoudig programma, maar desalniettemin een grote stap op weg naar het worden van een echte computerprogrammeur! We moeten echter nog een detail bespreken voordat we grotere programma's kunnen maken. We moeten begrijpen wat er zojuist is gebeurd. Wat hebben we de computer precies verteld en hoe wist de computer wat te doen? In het volgende hoofdstuk gaan we het programma analyseren dat we zojuist hebben geschreven zodat we gaan begrijpen hoe het in zijn werk gaat.

Afbeelding 4 – Intellisense

Ons programma opslaan

Als je Small Basic wilt sluiten en later opnieuw wilt werken aan het programma dat je zojuist hebt getypt, kun je het programma opslaan. Het is eigenlijk een goede gewoonte om programma's van tijd tot tijd op te slaan zodat je geen informatie verliest in het geval je per ongeluk afsluit of bij een stroomstoring. Je kunt het huidige programma opslaan door op het pictogram voor opslaan in de werkbalk te klikken of met de sneltoets 'Ctrl+S' (druk op de S-toets terwijl je de Ctrl-toets ingedrukt houdt).

Hoofdstuk 2

Ons eerste programma begrijpen

Wat is nu eigenlijk een computerprogramma?

Een programma is een reeks instructies voor de computer. Deze instructies vertellen de computer precies wat te doen en deze instructies worden altijd opgevolgd. Net zoals mensen, kunnen computers alleen instructies volgen als deze zijn verwoord in een taal die ze kunnen begrijpen. Dit zijn de zogenaamde programmeertalen. Er zijn veel talen die de computer kan begrijpen en **Small Basic** is hier een van.

Stel je eens een gesprek voor tussen jou en je vriend. Jij en je vrienden gebruiken woorden, georganiseerd in zinnen, waarmee je informatie heen en weer overdraagt. Op gelijke wijze kunnen programmeertalen verzamelingen woorden bevatten die in zinnen kunnen worden georganiseerd waarmee informatie aan de computer kan worden overgedragen. En programma's zijn in feite een reeks zinnen (soms maar een paar en soms vele duizenden) die zinvol is voor zowel de programmeur de computer.

De computer kan vele talen begrijpen. Java, C++, Python, VB, enz. zijn allemaal krachtige computertalen die worden gebruikt voor het ontwikkelen van eenvoudige tot complexe softwareprogramma's.

Small Basic-programma's

Een typisch Small Basic-programma bestaat uit een aantal *instructies*. Elke regel van het programma vertegenwoordigt een instructie en elke instructie is een instructie voor de computer. Als we de computer vragen een Small Basic-programma uit te voeren, wordt het programma geladen en de eerste instructie gelezen. De computer begrijpt wat we proberen te zeggen en vervolgens wordt onze instructie uitgevoerd. Als de eerste instructie is uitgevoerd, keert de computer terug naar het programma en wordt de tweede regel gelezen en uitgevoerd. De computer gaat door totdat het einde van het programma wordt bereikt. En dan is ons programma voltooid.

Terug naar ons eerste programma

Hier is het eerste programma dat we hebben geschreven:

```
TextWindow.WriteLine("Hallo wereld")
```

Dit is een eenvoudig programma dat bestaat uit één *instructie*. Deze instructie vertelt de computer een regel tekst, **Hallo wereld**, naar het tekstvenster te schrijven.

In het geheugen van de computer wordt dit letterlijk vertaald naar:

```
Write Hallo wereld
```

Je hebt misschien al opgemerkt dat de instructie kan worden gesplitst in kleinere segmenten, zoals zinnen kunnen wordt opgesplitst in woorden. In de eerste instructie hebben we 3 afzonderlijke segmenten:

a) TextWindow
b) WriteLine
c) “Hallo wereld”

De punt, haakjes en aanhalingsteken zijn allemaal leestekens die op de juiste posities in de instructie moeten worden geplaatst zodat de computer onze bedoeling kan begrijpen.

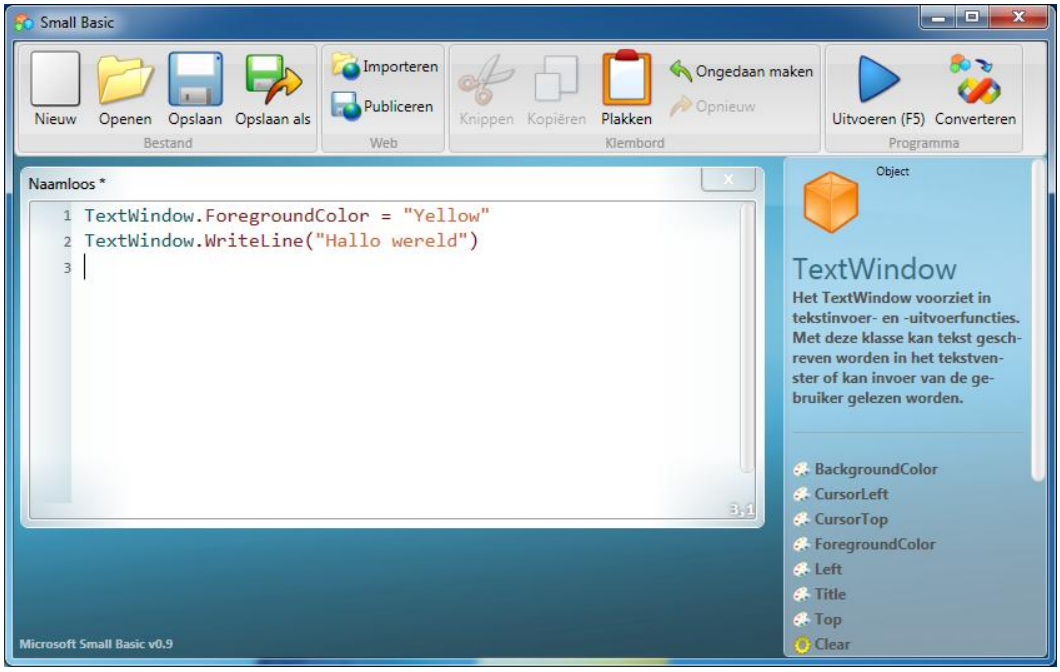
Misschien kun je je nog het zwarte venster herinneren dat verscheen toen we ons eerste programma uitvoerden. Dat zwarte venster wordt het TextWindow genoemd of soms ook console. Dit is de plaatst waar het resultaat van dit programma wordt weergegeven. **TextWindow** wordt in ons programma een *object* genoemd. Een aantal van dergelijke objecten zijn beschikbaar voor gebruik in onze programma’s. We kunnen verschillende *bewerkingen* uitvoeren op deze objecten. We hebben de WriteLine -bewerking al in ons programma gebruikt. Je hebt misschien ook al opgemerkt dat de bewerking WriteLine wordt gevolgd door **Hallo wereld** binnen aanhalingstekens. Deze tekst wordt doorgegeven als invoer bij de WriteLine-bewerking en wordt vervolgens naar het tekstvenster geschreven. Dit wordt een *invoer* bij de bewerking genoemd. Sommige bewerkingen kunnen meer dan een invoer hebben terwijl andere er geen enkele hebben.

Leestekens zoals aanhalingstekens, spaties en haakjes zijn heel belangrijk in een computerprogramma. Gebaseerd op hun positie en aantal, kunnen ze de betekenis wijzigen van wat wordt uitgedrukt.

Ons tweede programma

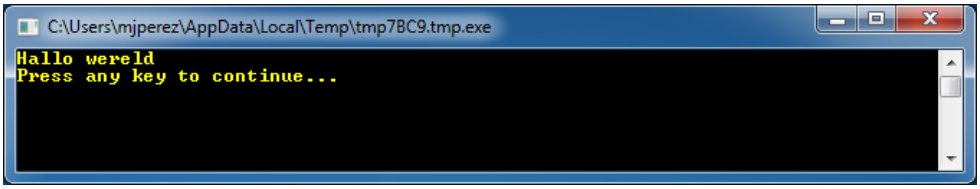
Nu we ons eerste programma begrijpen, kunnen we het ingewikkelder maken door een aantal kleuren toe te voegen.

```
TextWindow.ForegroundColor = "Yellow"
TextWindow.WriteLine("Hallo wereld")
```



Afbeelding 5 – Kleuren toevoegen

Wanneer je het bovenstaande programma uitvoert, zie je dat dezelfde “Hallo wereld”-zin naar het TextWindow wordt geschreven, maar deze zin verschijnt nu in geel in plaats van in grijs zoals in het vorige voorbeeld.



Afbeelding 6 – Hallo wereld in geel

Let op de nieuwe instructies die we aan ons oorspronkelijke programma hebben toegevoegd.
Er wordt een nieuw woord gebruikt: ForegroundColor. Dit woord hebben we gelijkgesteld aan de waarde *“Yellow”.* Dit betekent *dat we* “Yellow” hebben toegewezen aan ForegroundColor. Het verschil nu tussen ForegroundColor en de bewerking WriteLine is dat ForegroundColor geen invoer of haakjes nodig heeft. In plaats hiervan wordt het gevolgd door een *gelijk aan*-symbool en een woord. We definiëren ForegroundColor als een *eigenschap* van TextWindow. Hieronder volgt een lijst met waarden die geldig zijn voor de eigenschap ForegroundColor. Probeer “Yellow” met een van deze waarden te vervangen en bekijk de resultaten. Vergeet de aanhalingstekens niet, dit zijn verplichte leestekens.

```
Black
Blue
Cyan
Gray
Green
Magenta
Red
White
Yellow
DarkBlue
DarkCyan
DarkGray
DarkGreen
DarkMagenta
DarkRed
DarkYellow
```

Hoofdstuk 3

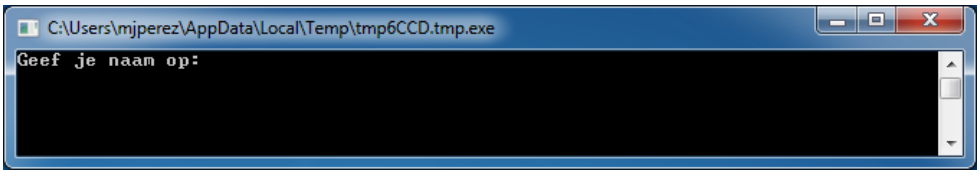
Variabelen introduceren

Variabelen gebruiken in ons programma

Het zou leuk zijn als ons programma “Hallo” kan zeggen met de naam van de gebruiker in plaats van het algemene “Hallo wereld?”. Als we dit willen doen, moeten we eerst de gebruiker om zijn of haar naam vragen, dan deze naam opslaan en vervolgens “Hallo” met de naam van de gebruiker naar het tekstvenster schrijven. Laten we eens kijken hoe we dit kunnen doen:

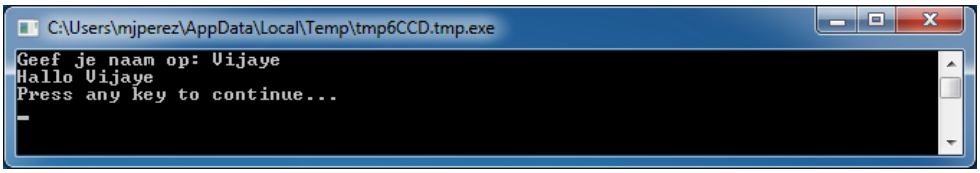
```
TextWindow.Write("Geef je naam op: ")
naam = TextWindow.Read()
TextWindow.WriteLine("Hallo " + naam)
```

Als je dit programma typt en uitvoert, zie je de volgende uitvoer:



Afbeelding 7 – Vraag om de naam van de gebruiker

En wanneer je je naam typt en op ENTER klikt, zie je de volgende uitvoer:



Afbeelding 8 – Een warme groet

Als je het programma opnieuw uitvoert, wordt er opnieuw dezelfde vraag gesteld. Je kunt een andere naam typen en de computer zal Hallo zeggen met die naam.

Analyse van het programma

De volgende regel in het programma dat je zojuist hebt uitgevoerd, heeft mogelijk je aandacht getrokken:

```
naam = TextWindow.Read()
```

Read() ziet er precies zo uit als *WriteLine()*, maar zonder invoer. Het is een bewerking waarmee de computer wordt verteld te wachten totdat de gebruiker iets typt en op de ENTER-toets drukt. Nadat de gebruiker op de ENTER-toets heeft gedrukt, slaat de computer op wat de gebruiker heeft getypt en geeft dit terug aan het programma. Het interessante hier is dat wat de gebruiker heeft getypt, is nu opgeslagen in een *variabele* met de naam **naam**. Een *variabele* wordt gedefinieerd als een plaats waar je waarden tijdelijk kunt opslaan om later te gebruiken. In de regel hierboven is **naam** gebruikt om de naam van de gebruiker op te slaan.

De volgende regel is ook interessant:

```
TextWindow.WriteLine("Hallo " +
naam)
```

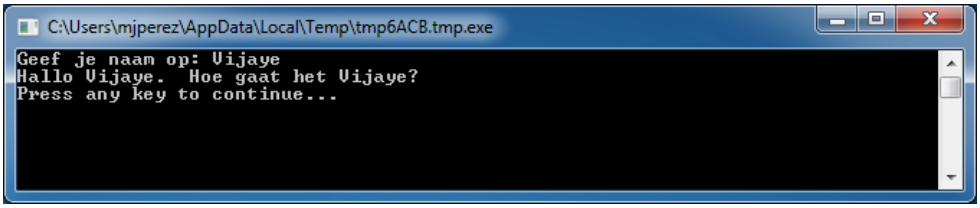
> Write *is net als* WriteLine *een andere bewerking in het tekstvenster. Met Write kun je iets naar het tekstvenster schrijven, maar met deze bewerking kun je ook de tekst die volgt op dezelfde regel als de huidige tekst plaatsen.*

Dit is de waar we de waarden gebruiken die zijn opgeslagen in onze variabele **naam**. We nemen de waarde in **naam** en hechten dit aan “Hallo” en schrijven dit naar het TextWindow.

Als een variabele eenmaal is ingesteld, kun je deze zo vaak gebruiken als je dat wilt. Je kunt bijvoorbeeld het volgende doen:

```
TextWindow.Write("Geef je naam op: ")
naam = TextWindow.Read()
TextWindow.Write("Hallo " + naam + ".  ")
TextWindow.WriteLine("Hoe gaat het " + naam + "?")
```

En je zult de volgende uitvoer zien:



Afbeelding 9 – Een variabele opnieuw gebruiken

Regels voor het benoemen van variabelen

Variabelen hebben namen waaraan je ze kunt herkennen. Er zijn bepaalde eenvoudige regels en enkele goede richtlijnen voor het benoemen van deze variabelen. Dit zijn:

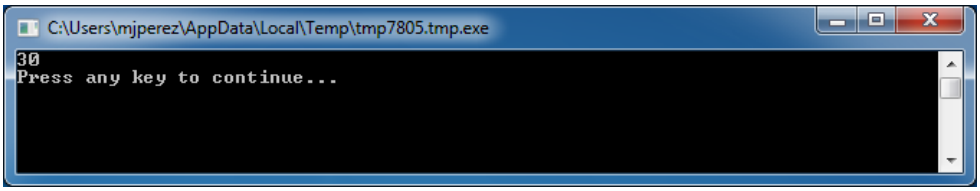
1. De naam moet beginnen met een letter en mag niet conflicteren met een van de sleutelwoorden zoals **if**, **for**, **then** enz.
2. Een naam kan elke combinatie van letters, cijfers en onderstrepingen bevatten.
3. Het is handig om de variabele een zinvolle naam te geven. En aangezien je de variabele zolang kunt maken als je wilt, kun je de bedoeling beschrijven.

Spelen met getallen

We hebben zojuist gezien hoe je variabelen kunt gebruiken om de naam van de gebruiker op te slaan. In de volgende programma's zullen we zien hoe we getallen in variabelen kunnen opslaan en manipuleren. Laten we beginnen met een heel eenvoudig programma:

```
getal1 = 10
getal2 = 20
getal3 = getal1 + getal2
TextWindow.WriteLine(getal3)
```

Als je dit programma uitvoert, zie je de volgende uitvoer:



Afbeelding 10 – Twee getallen optellen

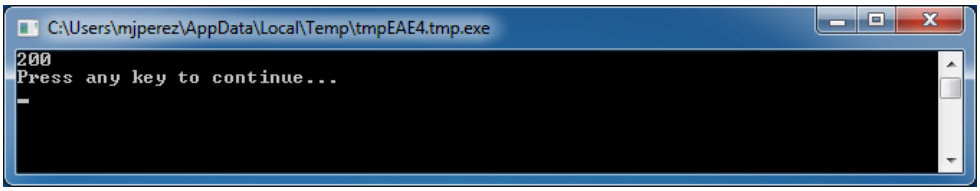
Op de eerste regel van het programma wijs je de variabele **getal1** toe met een waarde van 10. En op de tweede regel wijs je de variabele **getal2** toe met een waarde van 20. Op de derde regel voeg je **getal1** en **getal2** toe en wijs je het resultaat hiervan aan **getal3** toe. In dit geval heeft **getal3** daarom een waarde van 30. En dit is wat we naar het TextWindow hebben geschreven.

Misschien heb je opgemerkt dat de getallen niet worden omringd door aanhalingstekens. Aanhalingstekens zijn niet nodig voor getallen. Je hoeft alleen aanhalingstekens te gebruiken voor tekst.

Laten we nu het programma enigszins aanpassen en de resultaten bekijken:

```
getal1 = 10
getal2 = 20
getal3 = getal1 * getal2
TextWindow.WriteLine(getal3)
```

In het programma hierboven wordt **getal1** vermenigvuldigd met **getal2** en worden de resultaten opgeslagen in **getal3**. De resultaten van dat programma worden hieronder weergeven:



Afbeelding 11 – Twee getallen vermenigvuldigen

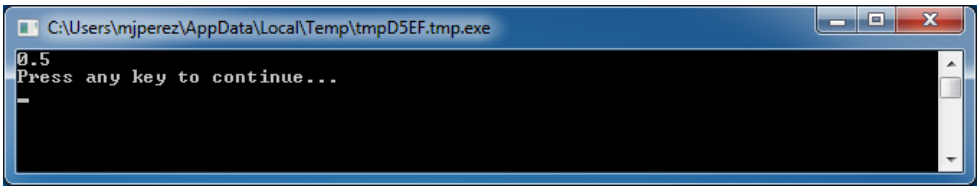
Op gelijke wijze kun je getallen aftrekken of delen. Hier volgt de aftrekking:

```
getal3 = getal1 - getal2
```

En het symbool voor deling is '/'. Het programma ziet er als volgt uit:

```
getal3 = getal1 / getal2
```

En het resultaat van deze deling is:



Afbeelding 12 – Twee getallen delen

Een eenvoudige temperatuuromzetter

Voor het volgende programma gebruiken we de formule $℃ = \frac{5(°F-32)}{9}$ voor het omrekenen van temperaturen in Fahrenheit naar temperaturen in Celsius.

Eerst moeten we de temperatuur in Fahrenheit, die we van de gebruiker hebben gekregen, opslaan in een variabele. Er is een speciale bewerking waarmee we getallen van de gebruiker kunnen lezen: **TextWindow.ReadNumber**.

```
TextWindow.Write("Geef de temperatuur op in Fahrenheit: ")
fahr = TextWindow.ReadNumber()
```

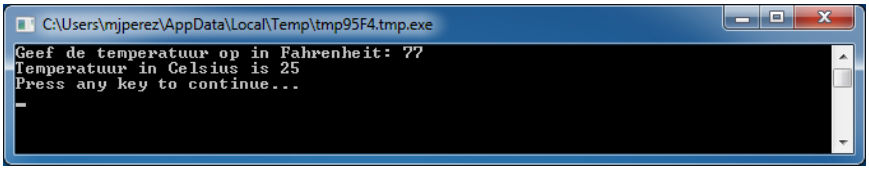
Nadat we de temperatuur in Fahrenheit hebben opgeslagen in een variabele, kunnen we deze als volgt naar Celsius omrekenen:

```
celsius = 5 * (fahr - 32) / 9
```

Met de haakjes wordt de computer de opdracht gegeven het gedeelte **fahr – 32** eerst te berekenen en vervolgens de rest te verwerken. Het enige wat we nu nog moeten doen, is het schrijven van de resultaten naar de gebruiker. Met dit alles krijgen we het volgende programma:

```
TextWindow.Write("Geef de temperatuur op in Fahrenheit: ")
fahr = TextWindow.ReadNumber()
celsius = 5 * (fahr - 32) / 9
TextWindow.WriteLine("Temperatuur in Celsius is " + celsius)
```

En het resultaat van dit programma is:



Afbeelding 13 – Temperatuurconversie

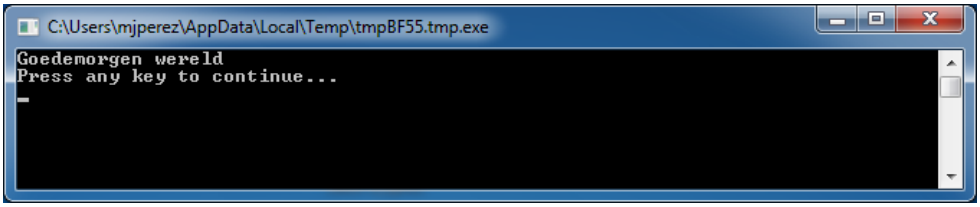
Hoofdstuk 4

Voorwaarden en vertakkingen

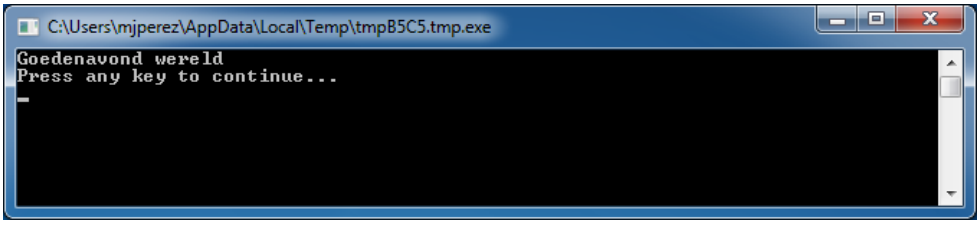
Laten we eens teruggaan naar ons eerste programma. Zou het niet leuk zijn als we in plaats van het algemene *Hallo wereld,* afhankelijk van de tijd van de dag, *Goedemorgen wereld* of *Goedenavond wereld* konden zeggen? In het volgende programma laten we de computer *Goedemorgen wereld* zeggen als het eerder is dan 12:00 uur en *Goedenavond* als het later is dan 12:00 uur.

```
If (Clock.Hour < 12) Then
  TextWindow.WriteLine("Goedemorgen wereld")
EndIf
If (Clock.Hour >= 12) Then
  TextWindow.WriteLine("Goedenavond wereld")
EndIf
```

Afhankelijk van wanneer je het programma uitvoert, zie je een van de volgende resultaten:



Afbeelding 14 – Goedemorgen wereld



Afbeelding 15 – Goedenavond wereld

Laten we de eerste drie regels van het programma eens analyseren. Je hebt al ontdekt dat je met deze regels de computer verteld dat als Clock.Hour minder is dan 12, de woorden "Goedemorgen wereld" naar het tekstvenster worden geschreven. De woorden **If**, **Then** en **EndIf** zijn speciale woorden die de worden geïnterpreteerd wanneer het programma wordt uitgevoerd. Het woord If wordt altijd gevolgd door een voorwaarde. In dit geval is dit **(Clock.Hour < 12)**. Vergeet niet dat de haakjes noodzakelijk zijn. De computer zal anders je bedoelingen niet begrijpen. De voorwaarde wordt gevolgd door **then** en de daadwerkelijke bewerking die moet worden uitgevoerd. En na de bewerking komt **EndIf**. Hiermee wordt aangegeven dat de conditionele uitvoering is afgerond.

In Small Basic kun je het klok*-object gebruiken voor toegang tot de huidige datum en tijd. Je beschikt ook over een aantal eigenschappen waarmee je de dag, de maand, het jaar, het uur, de minuut en de seconde van dit moment onafhankelijk van elkaar kunt verkrijgen.*

Tussen **then** en **EndIf** kunnen meer dan een bewerking worden geplaats en de computer zal ze allemaal uitvoeren als de voorwaarde geldig is. Je kunt bijvoorbeeld het volgende schrijven:

```
If (Clock.Hour < 12) Then
  TextWindow.Write("Goedemorgen. ")
  TextWindow.WriteLine("Hoe was het ontbijt?")
EndIf
```

Else

In het eerste programma in dit hoofdstuk, is het je misschien opgevallen dat de tweede voorwaarde min of meer overbodig is. De waarde **Clock.Hour** kan ofwel minder dan 12 zijn of niet. De tweede controle is niet echt nodig. Voor zulke gevallen kunnen we de twee instructies **if..then..endif** samenvoegen tot één met een nieuw woord, **else**.

Als we het programma herschrijven met **else**, ziet het er als volgt uit:

```
If (Clock.Hour < 12) Then
  TextWindow.WriteLine("Goedemorgen wereld")
Else
  TextWindow.WriteLine("Goedenavond wereld")
EndIf
```

Dit programma doet precies hetzelfde als het andere programma en dit brengt ons tot een belangrijke les in programmeren:

> *Binnen programmeren zijn er meestal vele manieren om hetzelfde te doen. Soms is een bepaalde manier logischer dan de andere. Het is aan de programmeur om de keuze te maken. Naargelang je meer programma's schrijft en meer ervaring opdoet, zul je deze verschillende technieken en hun voor- en nadelen beginnen te ontdekken.*

Inspringen

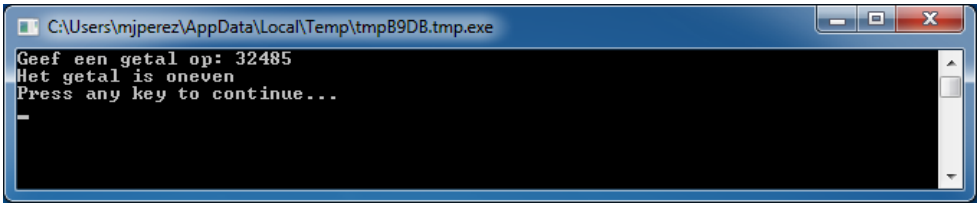
In alle voorbeelden kun je zien hoe de instructies tussen *If Else* en *EndIf* worden ingesprongen.
Dit inspringen is niet noodzakelijk. Het programma wordt ook zonder deze inspringingen begrepen. De structuur van het programma is echter wel gemakkelijker te zien en te begrijpen. Vandaar dat het normaal gesproken een goede gewoonte is om instructies tussen zulke blokken in te springen.

Even of oneven

Nu dat we de instructies **If..Then..Else..EndIf** onder de knie hebben, gaan we een programma schrijven dat van een bepaald getal kan aangeven of het even of oneven is.

```
TextWindow.Write("Geef een getal op: ")
getal = TextWindow.ReadNumber()
rest = Math.Remainder(getal, 2)
If (rest = 0) Then
  TextWindow.WriteLine("Het getal is even")
Else
  TextWindow.WriteLine("Het getal is oneven")
EndIf
```

En als je dit programma uitvoert, ziet de uitvoer er als volgt uit:



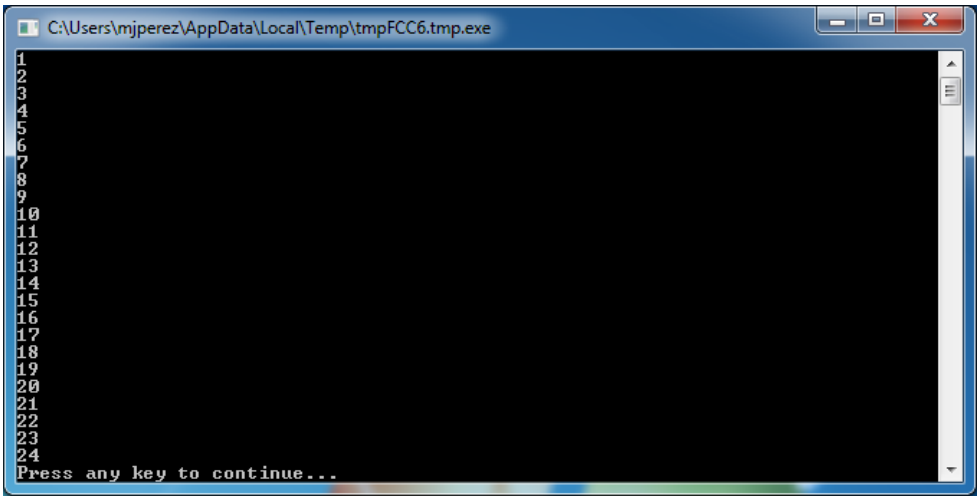
Afbeelding 16 – Even of oneven

In dit programma hebben we een andere nuttige bewerking geïntroduceerd, **Math.Remainder**. En zoals je misschien al hebt begrepen, met **Math.Remainder** wordt het eerste getal door het tweede getal gedeeld en wordt de rest vervolgens teruggegeven.

Vertakking

Misschien kun je je nog herinneren dat we in het tweede hoofdstuk hebben geleerd dat de computer tijdens de uitvoering van een programma slechts één instructie tegelijkertijd, op volgorde en van boven naar beneden, kan verwerken. Er is echter een speciale instructie waarmee de computer niet in volgorde naar een andere instructie kan springen. Laten we eens kijken naar het volgende programma.

```
i = 1
start:
TextWindow.WriteLine(i)
i = i + 1
If (i < 25) Then
  Goto start
EndIf
```



Afbeelding 17 – Goto gebruiken

In het programma hierboven hebben we de waarde 1 toegewezen aan de variabele **i**. En vervolgens hebben we een nieuwe instructie toegevoegd die eindigt op een dubbele punt (:)

```
start:
```

Dit wordt een *label* genoemd. Labels lijken op bladwijzers die de computer kan begrijpen. Je kunt de bladwijzer elke willekeurige naam geven en je kunt zoveel labels aan een programma toevoegen als je wilt. Alle namen moeten echter wel uniek zijn.

Een andere interessante instructie hier is:

```
i = i + 1
```

Hiermee vertel je de computer om de waarde 1 bij de variabele **i** op te tellen en deze waarde weer terug te geven aan **i**. Dus als de waarde van **i** 1 was voor deze instructie, dan wordt de waarde 2 nadat deze instructie wordt uitgevoerd.

En ten slotte,

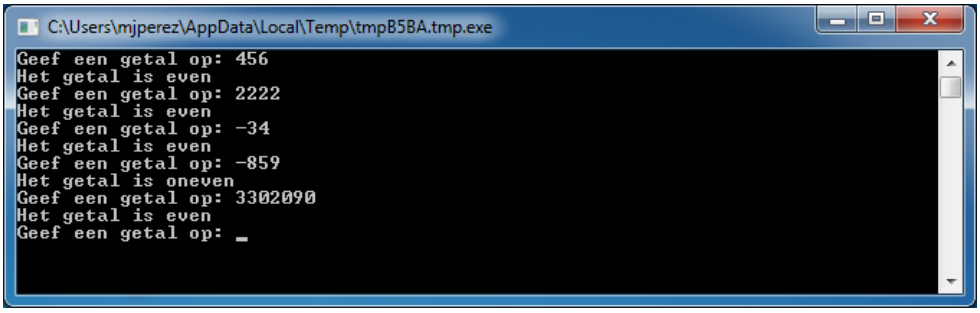
```
If (i < 25) Then
  Goto start
EndIf
```

Met dit onderdeel vertel je de computer dat er moet worden begonnen met het uitvoeren van instructies van de bladwijzer **start** als de waarde van **i** minder is dan 25.

Eindeloze uitvoering

Met de instructie **Goto** kun je de computer iets een willekeurig aantal keren laten herhalen. Je kunt bijvoorbeeld het programma Even en oneven aanpassen zoals hieronder wordt aangegeven en het programma eindeloos uitvoeren. Je kunt het programma stoppen door op de knop voor sluiten (X) te klikken in de rechterbovenhoek van het venster.

```
begin:
TextWindow.Write("Geef een getal op: ")
getal = TextWindow.ReadNumber()
rest = Math.Remainder(getal, 2)
If (rest = 0) Then
  TextWindow.WriteLine("Het getal is even")
Else
  TextWindow.WriteLine("Het getal is oneven")
EndIf
Goto begin
```



Afbeelding 18 – Even of oneven eindeloos uitvoeren

Hoofdstuk 5

Lussen

For-lus

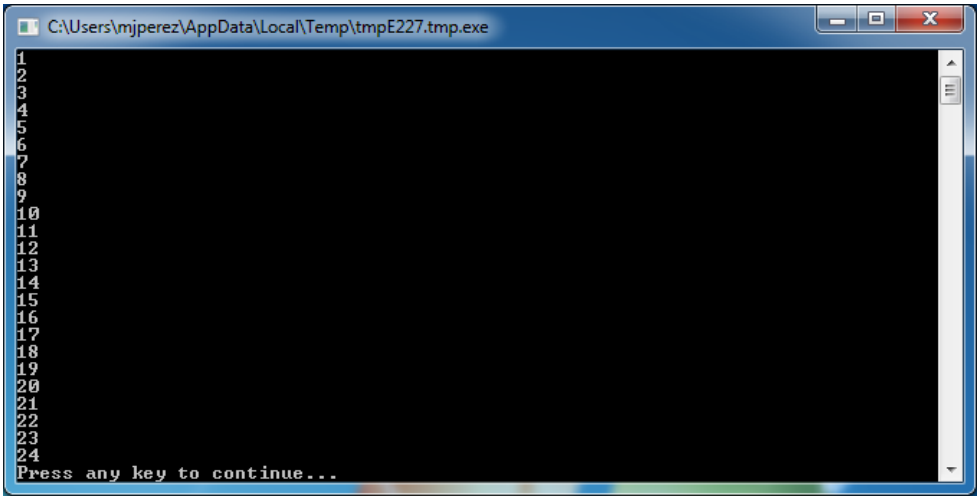
Laten we nog eens kijken naar het programma dat we in het vorige hoofdstuk hebben geschreven.

```
i = 1
start:
TextWindow.WriteLine(i)
i = i + 1
If (i < 25) Then
  Goto start
EndIf
```

Met dit programma worden de getallen van 1 tot 24 in volgorde naar het tekstvenster geschreven. Dit proces waarmee variabelen toenemen wordt veel gebruikt tijdens het programmeren en daarom vind je in programmeertalen normaal gesproken een gemakkelijkere manier om dit te doen. Het bovenstaande programma is gelijk aan het programma hieronder:

```
For i = 1 To 24
  TextWindow.WriteLine(i)
EndFor
```

En de uitvoer is:



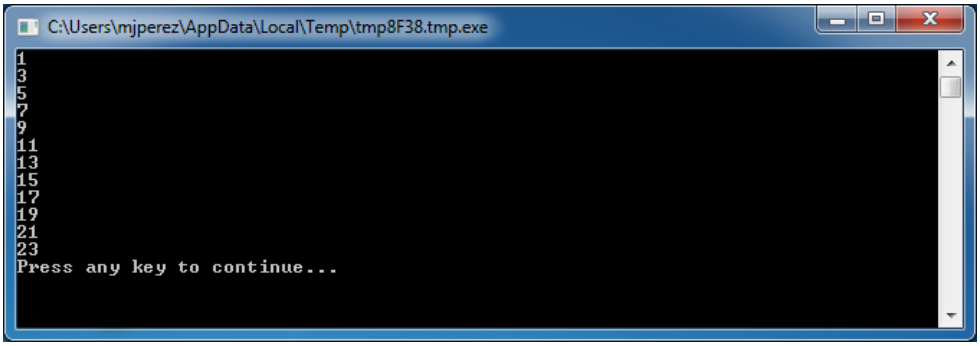
Afbeelding 19 – De For-lus gebruiken

Valt het je op dat we het achtregelige programma hebben gereduceerd tot een vierregelig programma en dat dit programma nog steeds precies hetzelfde doet als het achtregelige programma? Misschien kun je je nog herinneren dat we eerder hebben gezegd dat er vaak meerdere manieren zijn om hetzelfde te doen. Dit is een goed voorbeeld.

For.EndFor wordt, in programmeertermen, een *lus* genoemd. Hiermee kun je een variabele een begin- en eindwaarde geven en de computer deze variabele laten toenemen. Telkens wanneer de computer de variabele laat toenemen, worden de instructies tussen **For** en **EndFor** uitgevoerd.

Je kunt de lus ook gebruiken als je de variabele met 2 wilt laten toenemen in plaats van met 1, als je bijvoorbeeld alle oneven getallen tussen 1 en 24 naar het tekstvenster wilt schrijven.

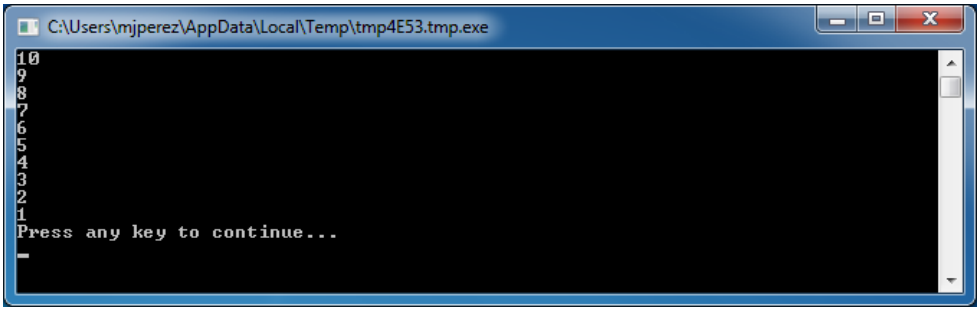
```
For i = 1 To 24 Step 2
  TextWindow.WriteLine(i)
EndFor
```



Afbeelding 20 – Alleen de oneven getallen

Met het gedeelte **Step 2** van de instructie **For** neemt de waarde van **i** met 2 toe in plaats van met de gebruikelijke waarde 1. Met **Step** kun je elke toename specificeren die je wilt. Je kunt ook een negatieve waarde voor de stap specificeren en hiermee de computer terug laten tellen, zoals in het voorbeeld hieronder:

```
For i = 10 To 1 Step -1
  TextWindow.WriteLine(i)
EndFor
```

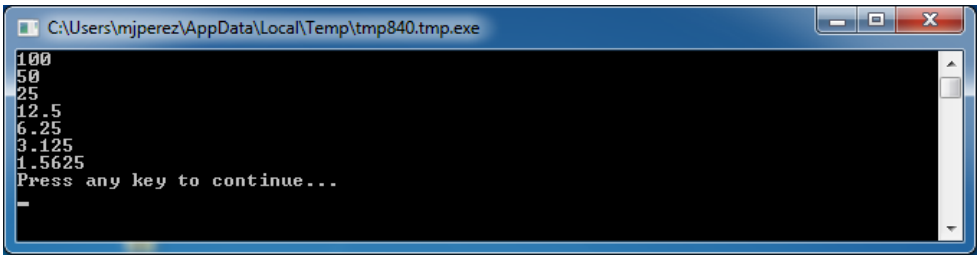


Afbeelding 21 – Terugtellen

While-lus

De While-lus is een lusmethode die handig is wanneer je het aantal lussen niet van tevoren weet. Daar waar de For-lus een van tevoren vastgesteld aantal keren wordt uitgevoerd, wordt de While-lus uitgevoerd totdat een gegeven voorwaarde waar is. In het voorbeeld hieronder halveren we het getal totdat het resultaat groter is dan 1.

```
getal = 100
While (getal > 1)
  TextWindow.WriteLine(getal)
  getal = getal / 2
EndWhile
```



Afbeelding 22 – Halverende lus

In het programma hierboven hebben we de waarde 100 aan *getal* toegewezen en voeren we de While-lus net zolang uit totdat getal groter is dan 1. We schrijven het getal binnen de lus naar het tekstvenster en delen het vervolgens door twee, waarmee we het halveren. En zoals verwacht, de uitvoer van het programma zijn getallen die toenemend na elkaar worden gehalveerd.

Het is moeilijk om dit programma te schrijven met een For-lus, omdat we niet weten hoe vaak de lus zal worden uitgevoerd. Met een While-lus kun je een voorwaarde gemakkelijk controleren en de lus voortzetten of beëindigen.

Het is ook interessant om te vermelden dat elke While-lus kan worden uitgepakt in een If..Then-instructie. Het programma hierboven kan bijvoorbeeld als volgt worden herschreven zonder het eindresultaat te beïnvloeden.

```
getal = 100
startLabel:
TextWindow.WriteLine(getal)
getal = getal / 2

If (getal > 1) Then
  Goto startLabel
EndIf
```

Elke While-lus wordt in feite intern herschreven naar instructies die If..Then samen met een of meer Goto-instructies gebruiken.

Hoofdstuk 6

Beginnen met grafische afbeeldingen

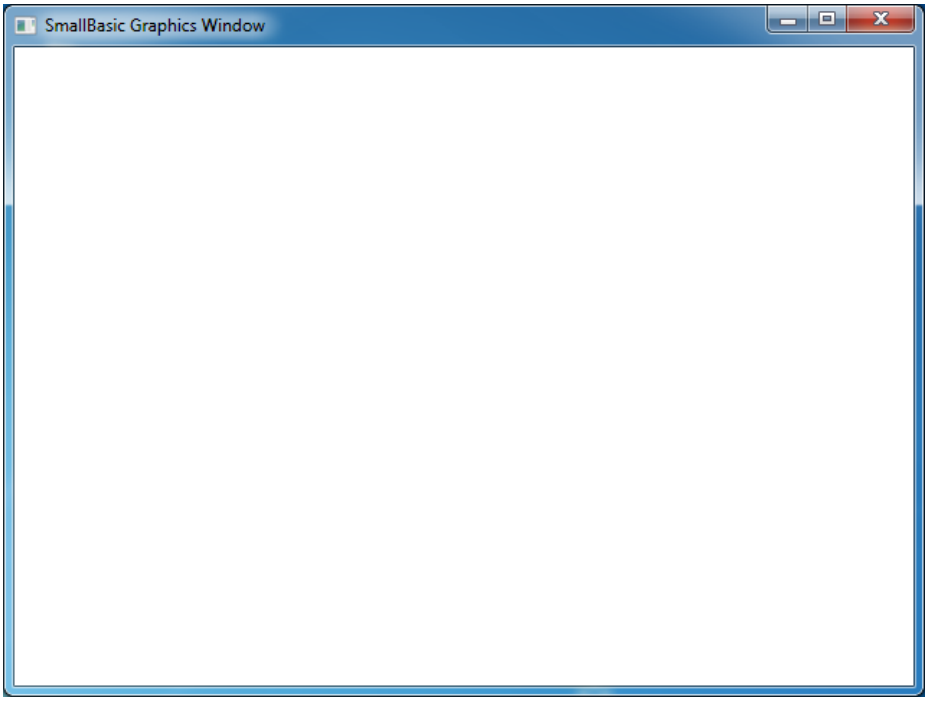
Tot dusver hebben we in al onze voorbeelden TextWindow gebruikt om de basisbeginselen van de Small Basic-taal uit te leggen. Small Basic heeft echter ook een krachtige set met grafische mogelijkheden en deze gaan we ontdekken in dit hoofdstuk.

Inleiding in GraphicsWindow

Net zoals we een TextWindow hebben waarin we met tekst en getallen kunnen werken, heeft Small Basic ook een **GraphicsWindow** dat we kunnen gebruiken voor tekenen. Laten we beginnen met het weergeven van GraphicsWindow.

```
GraphicsWindow.Show()
```

Als je dit programma uitvoert, wordt er in plaats van het gebruikelijke zwarte venster een wit venster zoals hieronder weergeven. Er valt nog niet zoveel te doen in dit venster, maar dit is het basisvenster waarmee we in dit hoofdstuk gaan werken. Je kunt dit venster sluiten door op 'X' te klikken in de rechterbovenhoek van het venster.



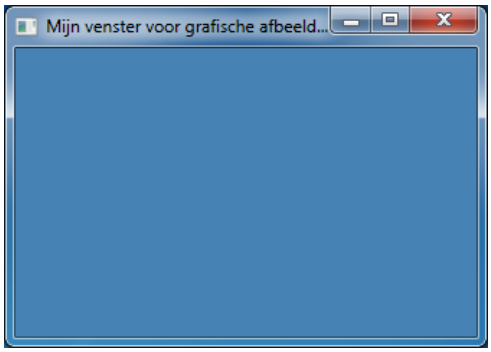
Afbeelding 23 – Een leeg venster voor grafische afbeeldingen

Het venster voor grafische afbeeldingen instellen

Je kunt de weergave van het venster voor grafische afbeeldingen aanpassen als je dat wilt. Je kunt de titel, achtergrond en de afmeting wijzigen. Laten we beginnen met een aantal kleine wijzigen zodat je vertrouwd raakt met het venster.

```
GraphicsWindow.BackgroundColor = "SteelBlue"
GraphicsWindow.Title = "Mijn venster voor grafische afbeeldingen"
GraphicsWindow.Width = 320
GraphicsWindow.Height = 200
GraphicsWindow.Show()
```

Zo ziet het aangepaste venster voor grafische afbeeldingen eruit. Je kunt de achtergrondkleur wijzigen naar een van de vele waarden die je in Bijlage B kunt vinden. Probeer deze eigenschappen uit om erachter te komen hoe je de weergave van het venster kunt wijzigen.

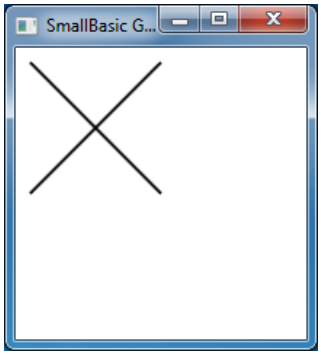


Afbeelding 24 – Een aangepast venster voor grafische afbeeldingen

Lijnen tekenen

Als we GraphicsWindow hebben ingesteld, kunnen we vormen, tekst en zelfs afbeeldingen tekenen. Laten we beginnen met het tekenen van enkele eenvoudige vormen. Hier volgt een programma dat een paar lijnen tekent in het venster voor grafische afbeeldingen.

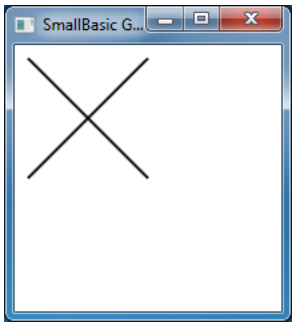
```
GraphicsWindow.Width = 200
GraphicsWindow.Height = 200
GraphicsWindow.DrawLine(10, 10, 100, 100)
GraphicsWindow.DrawLine(10, 100, 100, 10)
```



Afbeelding 25 – Kriskras

De eerste twee regels van het programma stellen het venster in en de volgende twee regels tekenen de kruiselingse lijnen. De eerste twee getallen die volgen op *DrawLine* geven het startpunt van de x- en y-coördinaten aan en de ander twee bepalen het eindpunt van de x- en y-coördinaten. Het interessante met grafische afbeelding op de computer is dat de coördinaten (0, 0) in de linkerbovenhoek van het venster beginnen. Als gevolg hiervan bevindt het venster zich in de coördinatenruimte op de 2e kwadrant.

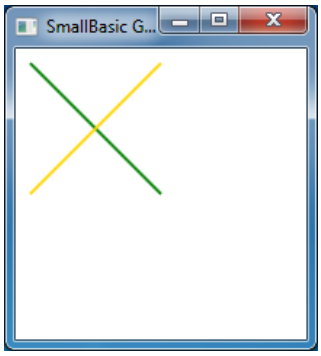
In plaats van namen te gebruiken voor kleuren, kun je ook de webnotatie voor kleuren gebruiken (#RRGGBB). Bijvoorbeeld: #FF0000 betekent rood, #FFFF00 is geel, enzovoort. We komen meer te weten over kleuren in [TODO kleurenhoofdstuk]



Afbeelding 26 – De coördinatenkaart

Als we teruggaan naar het lijnenprogramma, is het interessant op te merken dat je met Small Basic de eigenschappen van de lijnen, zoals kleur en dikte kunt wijzigen. Eerst gaan we de kleur van de lijnen wijzigen zoals je in het programma hieronder kunt zien.

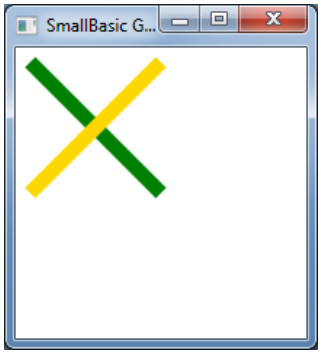
```
GraphicsWindow.Width = 200
GraphicsWindow.Height = 200
GraphicsWindow.PenColor = "Green"
GraphicsWindow.DrawLine(10, 10, 100, 100)
GraphicsWindow.PenColor = "Gold"
GraphicsWindow.DrawLine(10, 100, 100, 10)
```



Afbeelding 27 – De lijnkleur wijzigen

Nu gaan we ook de breedte wijzigen. Laten we in het programma hieronder nu de lijnbreedte van de standaardwaarde 1 naar 10 wijzigen.

```
GraphicsWindow.Width = 200
GraphicsWindow.Height = 200
GraphicsWindow.PenWidth = 10
GraphicsWindow.PenColor = "Green"
GraphicsWindow.DrawLine(10, 10, 100, 100)
GraphicsWindow.PenColor = "Gold"
GraphicsWindow.DrawLine(10, 100, 100, 10)
```

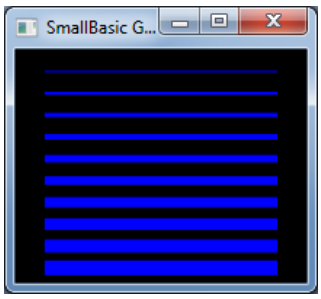


Afbeelding 28 – Dikke kleurrijke lijnen

Met *PenWidth* en *PenColor* kun je de pen wijzigen waarmee deze lijnen worden getekend. Deze wijzigingen zijn niet alleen van toepassing op de lijnen die we al hebben getekend, maar worden ook toegepast op elke ander vorm die je tekent nadat de eigenschappen zijn bijgewerkt.

Met lusinstructies kun je, zoals we hebben geleerd in eerdere hoofdstukken, gemakkelijk een programma schrijven waarmee je meerdere lijnen met toenemende pendiktes kunnen tekenen.

```
GraphicsWindow.BackgroundColor = "Black"
GraphicsWindow.Width = 200
GraphicsWindow.Height = 160
GraphicsWindow.PenColor = "Blue"

For i = 1 To 10
  GraphicsWindow.PenWidth = i
  GraphicsWindow.DrawLine(20, i * 15, 180, i * 15)
endfor
```



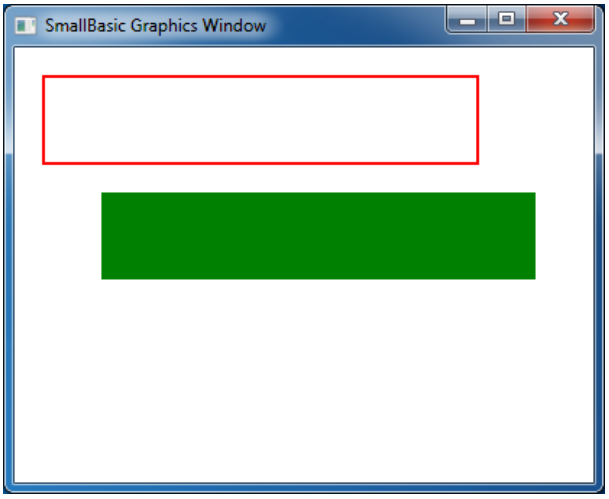
Afbeelding 29 – Meerdere pendiktes

Het interessante aan dit programma is de lus waarmee je de *PenWidth* telkens als de lus wordt uitgevoerd laten toenemen en dan een nieuwe lijn onder de oude laat tekenen.

Vormen tekenen en vullen

Voor het tekenen van vormen zijn er normaal gesproken twee typen bewerkingen voor elke vorm. Dit zijn *tekenbewerkingen* en *vulbewerkingen*. Met tekenbewerkingen kunnen we de omtrek van de vorm met een pen tekenen en met vulbewerkingen kunnen deze met een kwast vullen. In het voorbeeld hieronder worden twee rechthoeken getekend, een wordt getekend met een rode pen en een wordt gevuld met een groene kwast.

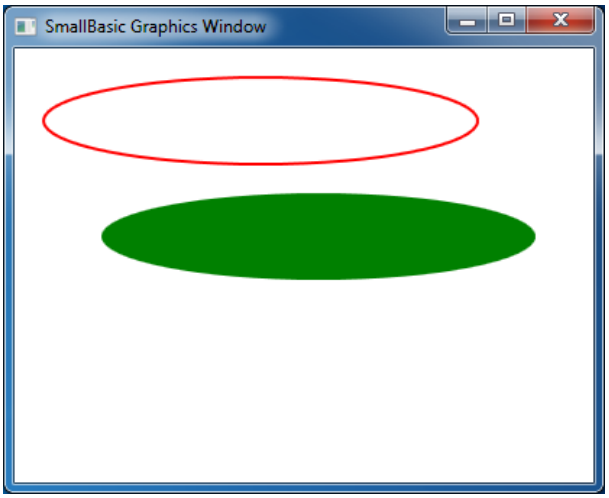
```
GraphicsWindow.Width = 400
GraphicsWindow.Height = 300

GraphicsWindow.PenColor = "Red"
GraphicsWindow.DrawRectangle(20, 20, 300, 60)

GraphicsWindow.BrushColor = "Green"
GraphicsWindow.FillRectangle(60, 100, 300, 60)
```



Afbeelding 30 – Tekenen en vullen

Je hebt vier getallen nodig voor het tekenen of vullen van een rechthoek. De eerste twee getallen vertegenwoordigen de X- en Y-coördinaten voor de linkerbovenhoek van de rechthoek. Het derde getal geeft de breedte van de rechthoek aan en het vierde getal specificeert de hoogte. Dit is in feite ook van toepassing op het tekenen en vullen van ovalen, zoals wordt weergeven in het programma hieronder.

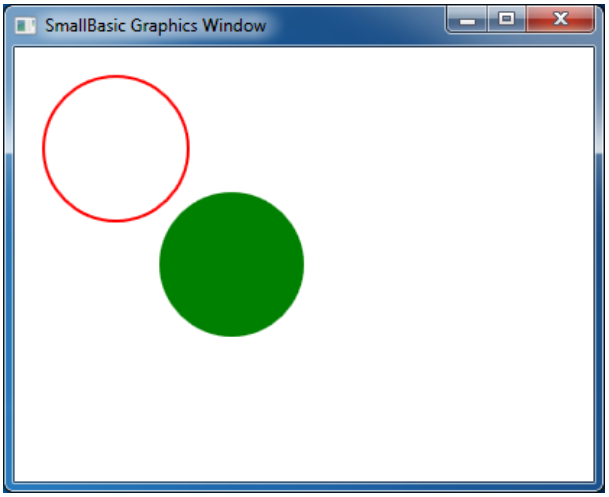
```
GraphicsWindow.Width = 400
GraphicsWindow.Height = 300

GraphicsWindow.PenColor = "Red"
GraphicsWindow.DrawEllipse(20, 20, 300, 60)

GraphicsWindow.BrushColor = "Green"
GraphicsWindow.FillEllipse(60, 100, 300, 60)
```



Afbeelding 31 – Ovalen tekenen en vullen

Een ovaal is een bepaald type cirkel. Als je cirkels wilt tekenen, moet je dezelfde breedte en hoogte opgeven.

```
GraphicsWindow.Width = 400
GraphicsWindow.Height = 300

GraphicsWindow.PenColor = "Red"
GraphicsWindow.DrawEllipse(20, 20, 100, 100)

GraphicsWindow.BrushColor = "Green"
GraphicsWindow.FillEllipse(100, 100, 100, 100)
```



Afbeelding 32 – Cirkels

Hoofdstuk 7

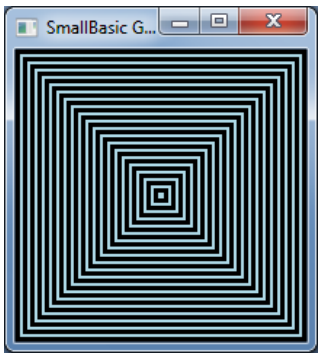
Plezier met vormen

In dit hoofdstuk gaan we plezier beleven aan alles wat we tot nu toe hebben geleerd. Dit hoofdstuk bevat voorbeelden waarin alles wat we tot nu toe hebben geleerd wordt gecombineerd voor het maken van enkele leuke programma's.

Rechthoeken in overvloed

Hier tekenen we in een lus meerdere rechthoeken met toenemende afmetingen.

```
GraphicsWindow.BackgroundColor = "Black"
GraphicsWindow.PenColor = "LightBlue"
GraphicsWindow.Width = 200
GraphicsWindow.Height = 200

For i = 1 To 100 Step 5
  GraphicsWindow.DrawRectangle(100 - i, 100 - i, i * 2, i * 2)
EndFor
```

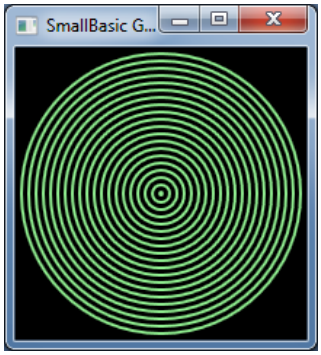


Afbeelding 33 – Rechthoeken in overvloed

Spectaculaire cirkels

Met een variant op het vorige programma teken we cirkels in plaats van vierkanten.

```
GraphicsWindow.BackgroundColor = "Black"
GraphicsWindow.PenColor = "LightGreen"
GraphicsWindow.Width = 200
GraphicsWindow.Height = 200

For i = 1 To 100 Step 5
  GraphicsWindow.DrawEllipse(100 - i, 100 - i, i * 2, i * 2)
EndFor
```

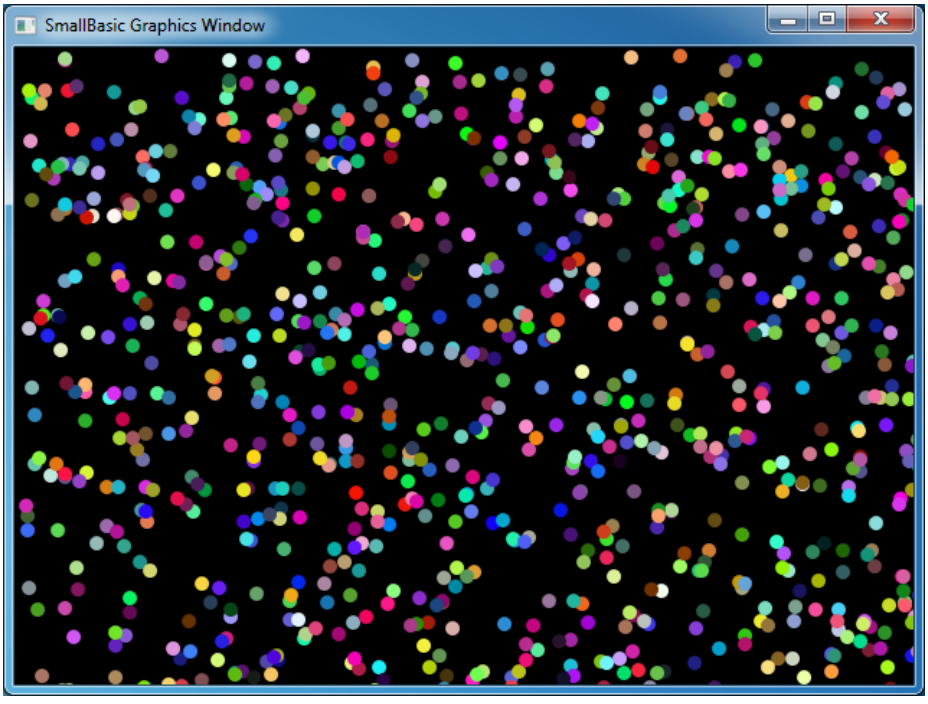


Afbeelding 34 – Spectaculaire cirkels

Willekeurig maken

Met dit programma gebruiken we de bewerking GraphicsWindow.*GetRandomColor* voor het instellen van willekeurige kleuren voor de kwast en vervolgens stellen we de x- en y-coördinaten voor de cirkels in met Math.*GetRandomNumber*. Deze twee bewerkingen kunnen op interessante manieren worden gecombineerd om interessante programma's te maken met verschillende resultaten telkens als ze worden uitgevoerd.

```
GraphicsWindow.BackgroundColor = "Black"
For i = 1 To 1000
  GraphicsWindow.BrushColor = GraphicsWindow.GetRandomColor()
  x = Math.GetRandomNumber(640)
  y = Math.GetRandomNumber(480)
  GraphicsWindow.FillEllipse(x, y, 10, 10)
EndFor
```

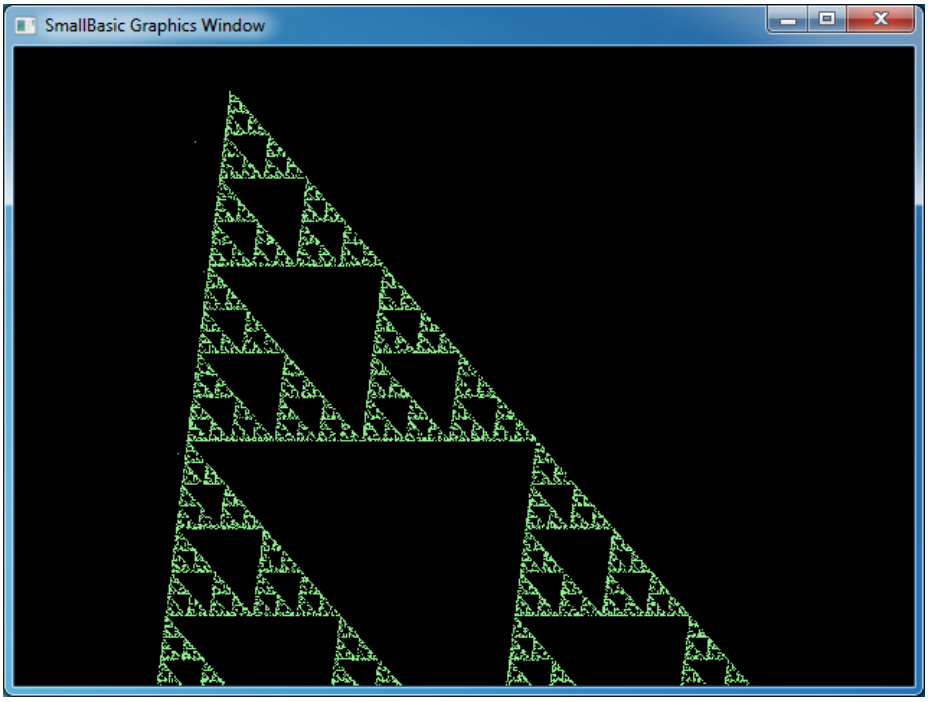


Afbeelding 35 – Willekeurig maken

Fractalen

In het volgende programma tekenen we een eenvoudige driehoekfractal met willekeurige getallen. Een fractal is een geometrische vorm die kan worden opgedeeld in delen, waarbij elk deel precies lijkt op de oorspronkelijke vorm. In dit geval worden er honderden driehoeken getekend die lijken op de oorspronkelijke driehoek. En aangezien dit programma een aantal seconden duurt, kun je zien hoe de driehoeken langzaam worden gevormd van puntjes. De logica hier achter is enigszins moeilijk te beschrijven en daarom laat ik je dit ontdekken in een oefening.

```
GraphicsWindow.BackgroundColor = "Black"
x = 100
y = 100

For i = 1 To 100000
 r = Math.GetRandomNumber(3)
  ux = 150
  uy = 30
  If (r = 1) then
    ux = 30
    uy = 1000
  EndIf

  If (r = 2) Then
    ux = 1000
    uy = 1000
  EndIf

  x = (x + ux) / 2
  y = (y + uy) / 2

  GraphicsWindow.SetPixel(x, y, "LightGreen")
EndFor
```



Afbeelding 36 – Een driehoekfractal

Als je goed wilt zien hoe de fractal langzaam uit puntjes wordt gevormd, kun je de lus vertragen met de bewerking **Program.***Delay*. Voor deze bewerking moet je een getal opgeven dat in milliseconden specificeert hoelang de vertraging is. Hier volgt het aangepaste programma, met de gewijzigde regel vetgedrukt.

```
GraphicsWindow.BackgroundColor = "Black"
x = 100
y = 100

For i = 1 To 100000
  r = Math.GetRandomNumber(3)
  ux = 150
  uy = 30
  If (r = 1) then
    ux = 30
    uy = 1000
  EndIf

  If (r = 2) Then
    ux = 1000
    uy = 1000
  EndIf
```

```
  x = (x + ux) / 2
  y = (y + uy) / 2

  GraphicsWindow.SetPixel(x, y, "LightGreen")
  Program.Delay(2)
EndFor
```

Het programma wordt langzamer uitgevoerd naarmate je de vertraging vergroot. Experimenteer met de getallen om erachter te komen wat de beste vertraging voor jou is.

Een andere wijziging aan dit programma is de vervanging van de volgende regel:

```
GraphicsWindow.SetPixel(x, y, "LightGreen")
```

met

```
kleur = GraphicsWindow.GetRandomColor()
GraphicsWindow.SetPixel(x, y, kleur)
```

Met deze wijziging worden de pixels van de driehoek met willekeurige kleuren getekend.

Hoofdstuk 8

Grafische afbeeldingen met een schildpad

Logo

In de jaren 70 was er een eenvoudige, maar krachtige programmeertaal met de naam Logo die werd gebruikt door een klein aantal onderzoekers. Totdat iemand 'Turtle Graphics' aan de taal toevoegde en een 'schildpad' op het scherm zichtbaar maakte die reageerde op opdrachten als *Ga vooruit, Ga rechtsaf, Ga linksaf.* enz. Met de schildpad konden mensen interessante vormen op het schermen tekenen. Dit maakte de taal onmiddellijk toegankelijk en aantrekkelijk voor mensen van alle leeftijden en de taal werd daarom razend populair in de jaren 80.

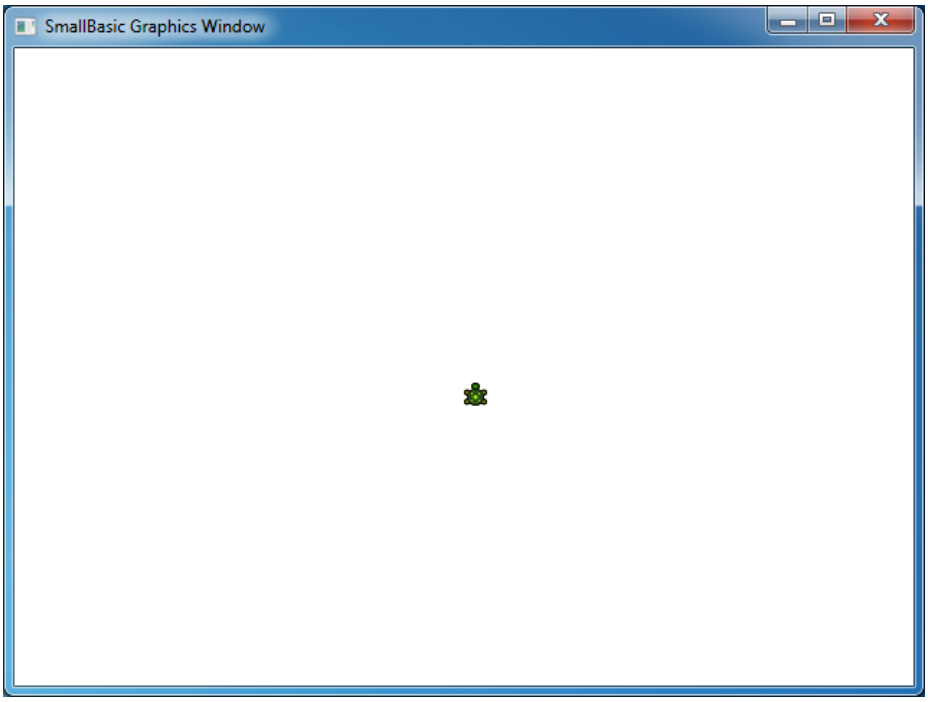
Small Basic heeft een **Turtle**-object met vele opdrachten die kunnen worden opgeroepen binnen Small Basic-programma's. In dit hoofdstuk zullen we de schildpad gebruiken voor het tekenen van grafische afbeeldingen op het scherm.

De schildpad

Voordat we kunnen beginnen, moeten we de schildpad zichtbaar maken op het scherm. Dit kunnen we doen met een eenvoudig eenregelig programma.

```
Turtle.Show()
```

Als je dit programma uitvoert, zul je een wit venster zien, net als het venster dat je in het vorige hoofdstuk zag, behalve dat er nu een schildpad in het midden van het venster is geplaatst. Deze schildpad gaat onze instructies volgen en tekenen wat we willen dat er wordt getekend.



Afbeelding 37 – Schildpad is zichtbaar

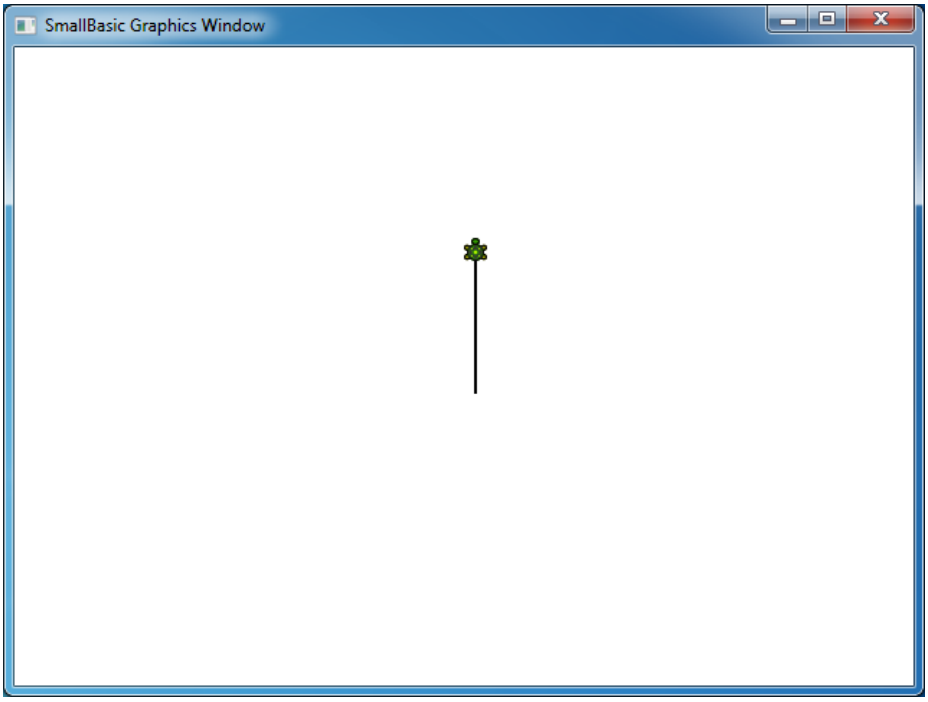
Verplaatsen en tekenen

Een van de instructies die je met de schildpad kunt uitvoeren is **Move**. Je moet een getal opgeven als de invoer voor deze bewerking. Dit getal geeft aan hoever de schildpad moet worden verplaatst. In het voorbeeld hieronder gaan we de schildpad 100 pixels verplaatsen.

```
Turtle.Move(100)
```

Als je dit programma uitvoert, kun je zien dat de schildpad langzaam 100 pixels naar boven wordt verplaatst. Tijdens deze verplaatsing zie je ook dat er een lijn wordt getekend achter de schildpad. Als je klaar bent met het verplaatsen van de schildpad, ziet het resultaat eruit als in de afbeelding hieronder.

Wanneer je bewerkingen van de schildpad gebruikt, hoef je Show() niet op te roepen. De schildpad wordt automatisch zichtbaar gemaakt wanneer er een Turtle-bewerking wordt uitgevoerd.



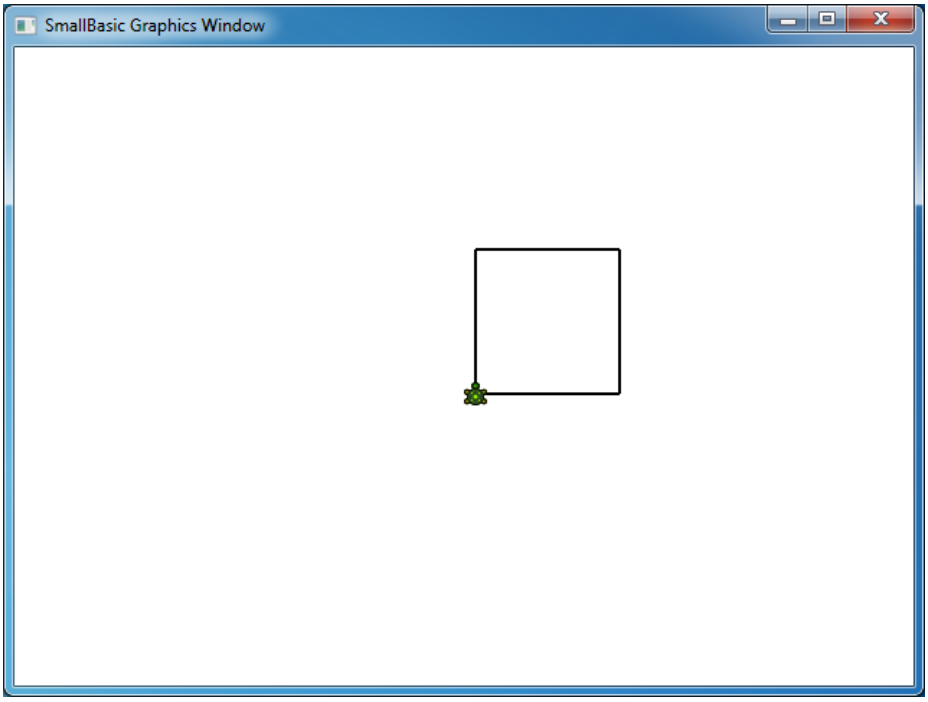
Afbeelding 38 – Honderd pixels verplaatsen

Een vierkant tekenen

Een vierkant heeft vier zijkanten, twee verticaal en twee horizontaal. Als we een vierkant willen tekenen, moeten we de schildpad een lijn laten tekenen, rechtsaf laten gaan, een andere lijn laten teken en rechtsaf laten gaan totdat alle vier zijkanten zijn getekend. In een programma ziet dit er als volgt uit:

```
Turtle.Move(100)
Turtle.TurnRight()
Turtle.Move(100)
Turtle.TurnRight()
Turtle.Move(100)
Turtle.TurnRight()
Turtle.Move(100)
Turtle.TurnRight()
```

Als je dit programma uitvoert, kun je zien hoe met de schildpad het vierkant één lijn tegelijkertijd wordt getekend. Het resultaat ziet er uit als in de afbeelding hieronder.



Afbeelding 39 – Schildpad tekent een vierkant

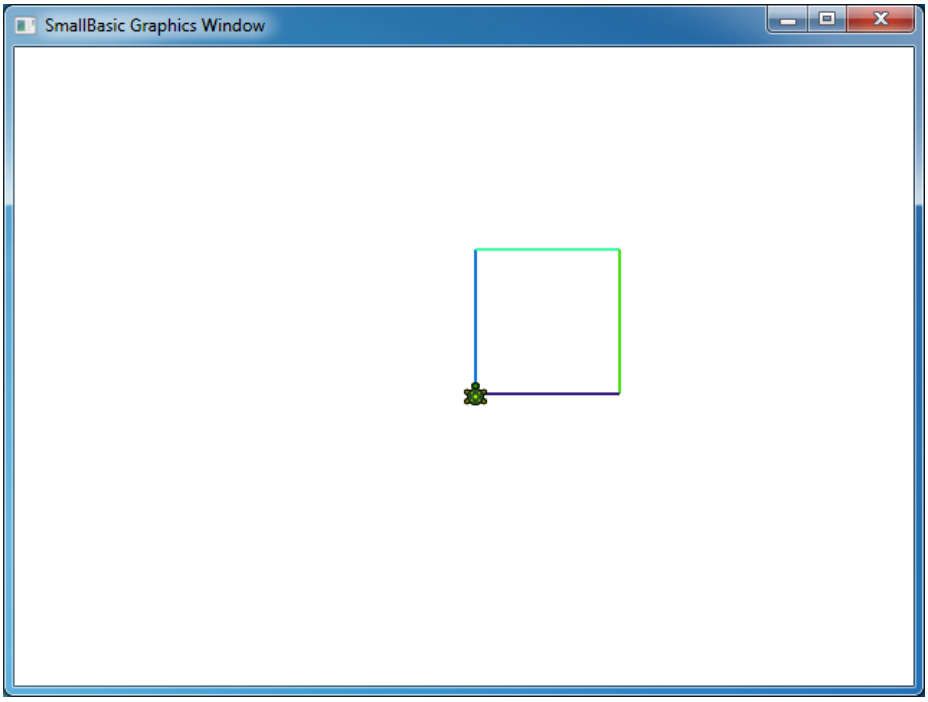
Het is interessant om op te merken dat we dezelfde twee instructies herhaaldelijk uitvoeren, vier keer om precies te zijn. En we weten al dat zulke herhalende opmerkingen kunnen worden uitgevoerd met lussen. Als we daarom het programma hierboven wijzigen om het de **For..EndFor**-lus te laten gebruiken, wordt het programma veel eenvoudiger.

```
 For i = 1 To 4
  Turtle.Move(100)
  Turtle.TurnRight()
EndFor
```

Kleuren wijzigen

De schildpad tekent op hetzelfde GraphicsWindow dat we in het vorige hoofdstuk hebben gezien. Dit betekent dat alle bewerkingen die we hebben geleerd in het vorige hoofdstuk hier nog steeds van toepassing zijn. Het volgende programma tekent bijvoorbeeld een vierkant met elke zijkant in een andere kleur.

```
For i = 1 To 4
  GraphicsWindow.PenColor = GraphicsWindow.GetRandomColor()
  Turtle.Move(100)
  Turtle.TurnRight()
EndFor
```



Afbeelding 40 – Kleuren wijzigen

Meer gecompliceerde vormen tekenen

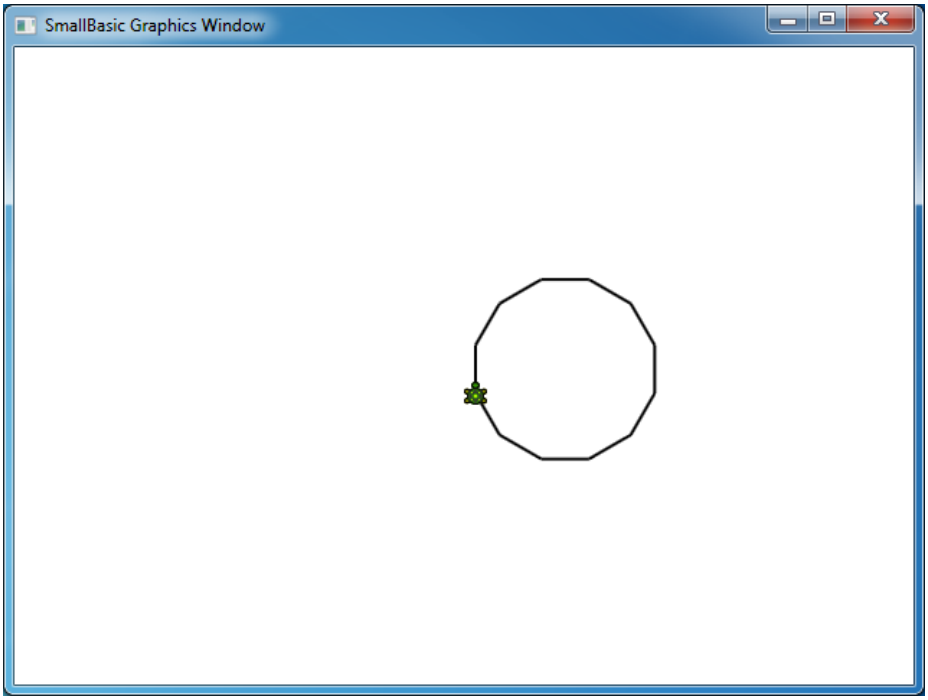
De Turtle, heeft in aanvulling op de bewerkingen **TurnRight** en **TurnLeft** ook een **Turn**-bewerking. De hoek van de rotatie wordt met één invoer bepaald. Met deze bewerking kun je veelhoeken met een willekeurig aantal zijkanten tekenen. Het volgende programma tekent een hexagoon (een zeszijdige veelhoek).

```
For i = 1 To 6
  Turtle.Move(100)
  Turtle.Turn(60)
EndFor
```

Probeer dit programma zelf om te zien dat er echt een hexagoon wordt getekend. De hoek tussen de kanten is 60 graden en daarom gebruiken we **Turn(60)**. Voor een veelhoek waarvan de zijkanten gelijk zijn, kunnen we de hoek tussen de zijkanten gemakkelijk verkrijgen door 360 te delen door het aantal zijkanten. Met deze informatie en met variabelen kunnen we een vrij algemeen programma schrijven waarmee we een veelhoek met een willekeurig aantal zijkanten kunnen tekenen.

```
zijkanten = 12

lengte = 400 / zijkanten
hoek = 360 / zijkanten

For i = 1 To zijkanten
  Turtle.Move(lengte)
  Turtle.Turn(hoek)
EndFor
```

Met dit programma kun je een veelhoek tekenen door de variabele **zijkanten** te wijzigen. Als je hier 4 opgeeft, krijg je het vierkant waar we mee begonnen. Als je de waarde groot genoeg maakt, zeg 50, is het resultaat niet meer te onderscheiden van een cirkel.



Afbeelding 41 – Een veelhoek met 12 zijkanten tekenen

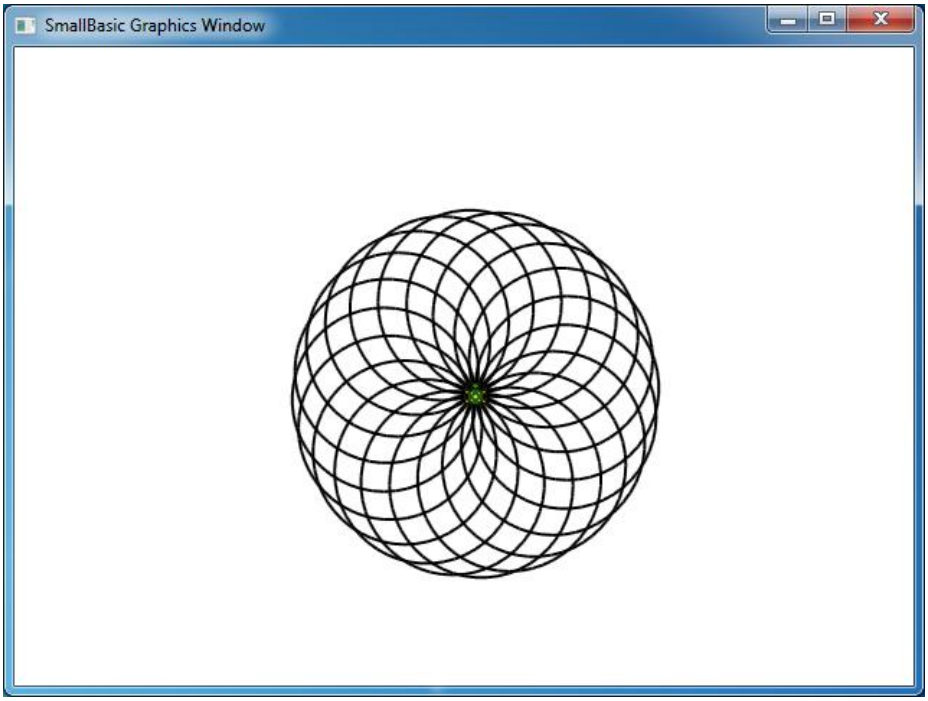
Met de techniek die we zojuist hebben geleerd, kunnen we de schildpad meerdere cirkels laten tekenen met telkens een kleine verschuiving en dit resulteert in een interessante uitvoer.

```
zijkanten = 50
lengte = 400 / zijkanten
hoek = 360 / zijkanten

Turtle.Speed = 9
```

```
For j = 1 To 20
  For i = 1 To zijkanten
    Turtle.Move(lengte)
    Turtle.Turn(hoek)
  EndFor
  Turtle.Turn(18)
EndFor
```

Het programma hierboven heeft twee **For..EndFor**-lussen, de een in de andere. De binnenste lus (*i = 1 to zijkanten*) is gelijk aan het veelhoekprogramma en hiermee wordt de cirkel getekend. Met de buitenste lus (*j = 1 to 20*) wordt de schildpad telkens een klein beetje gedraaid wanneer een er een nieuwe cirkel wordt getekend. Dit vertelt de schildpad 20 cirkels te tekenen. Tezamen resulteert dit programma in een erg interessant patroon en dat kun je hieronder zien.

In het programma hierboven verplaatsen we de schildpad door de snelheid in te stellen op 9. Je kunt deze eigenschap instellen op elke willekeurige waarde tussen 1 en 10 om de schildpad zo snel te verplaatsen als dat jij dat wilt.



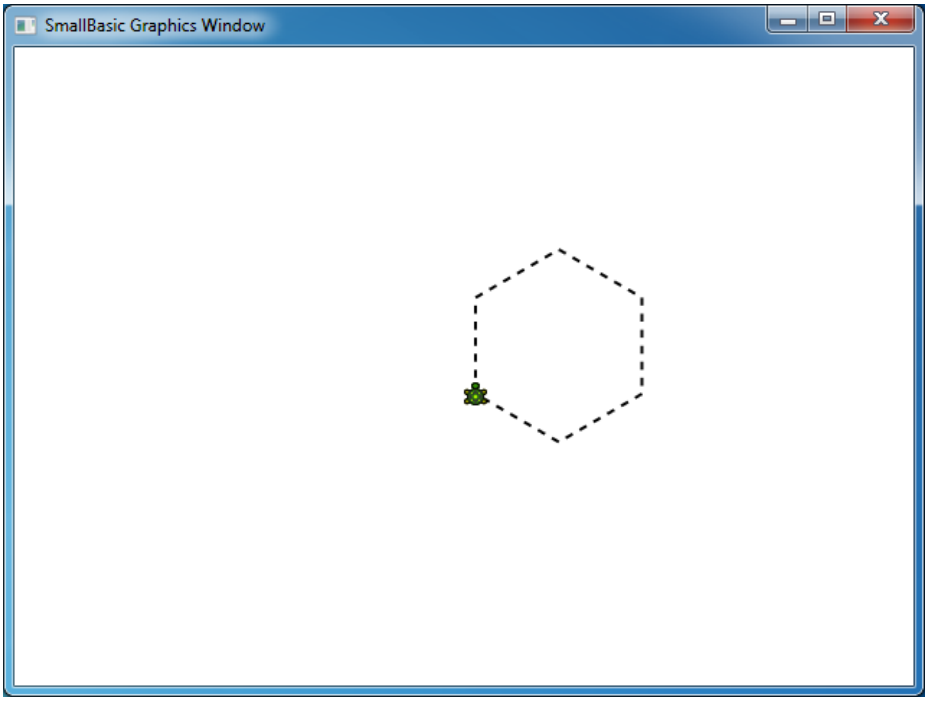
Afbeelding 42 – In rondjes lopen

In alle richtingen verplaatsen

Je kunt de schildpad laten stoppen met tekenen met de bewerking **PenUp**. Hiermee kan de schildpad overal op het scherm worden verplaatst zonder dat er een lijn wordt getekend. Met het oproepen van **PenDown** laat je de schildpad weer tekenen. Dit kun je gebruiken om interessante effecten, zoals stippellijnen, te verkrijgen. Hier volgt een programma waarin dit wordt gebruikt voor het tekenen van een veelhoek met stippellijnen.

```
zijkanten = 6

lengte = 400 / zijkanten
hoek = 360 / zijkanten

For i = 1 To zijkanten
  For j = 1 To 6
    Turtle.Move(lengte / 12)
    Turtle.PenUp()
    Turtle.Move(lengte / 12)
    Turtle.PenDown()
  EndFor
  Turtle.Turn(hoek)
EndFor
```

Ook dit programma heeft twee lussen. Met de binnenste lus tekenen we een enkele stippellijn en met de buitenste lus specificeren we hoeveel lijnen er moeten worden getekend. In ons voorbeeld hebben we 6 gebruikt voor de variabele **zijkanten** en daarom wordt de hexagoon met stippellijn als hieronder getekend.



Afbeelding 43 – PenUp en PenDown gebruiken

Hoofdstuk 9

Subroutines

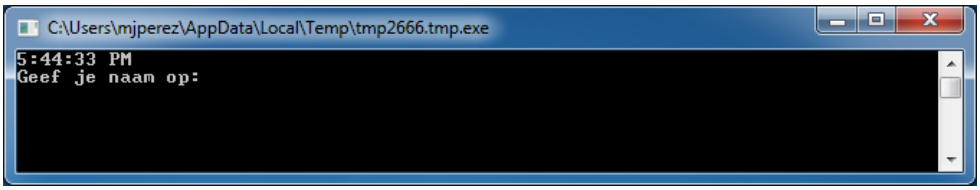
Tijdens het schrijven van programma's moeten we heel vaak dezelfde serie stappen telkens opnieuw uitvoeren. In die gevallen is het waarschijnlijk overbodig om deze instructies telkens opnieuw te schrijven. En daarom is het handig om *subroutines* te gebruiken.

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()
```

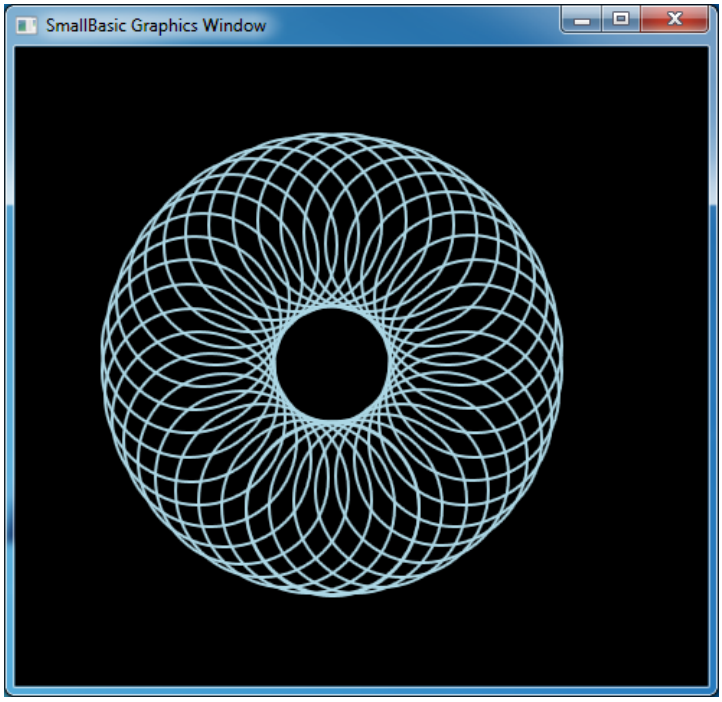
```
FindMax()
TextWindow.WriteLine("Het hoogste nummer is: " + max)

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

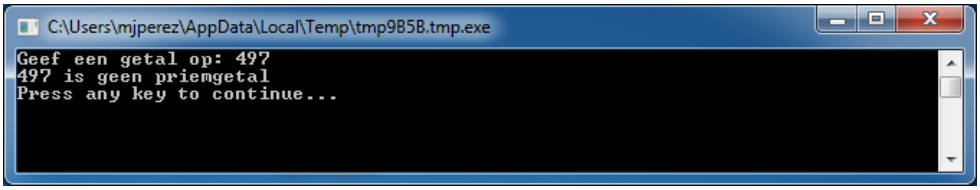
Subroutines oproepen binnen lussen

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'.

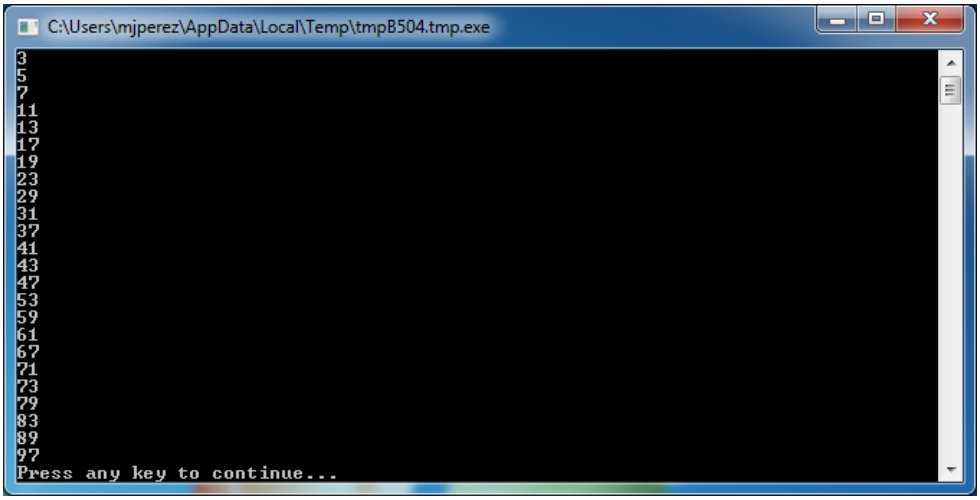


Afbeelding 47 – Priemgetallen testen

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

Hoofdstuk 10

Matrices

Inmiddels weet je al goed hoe je variabelen kunt gebruiken. Je hebt al veel geleerd en hopelijk vind je het nog steeds leuk.

Laten we het eerste programma met variabelen dat we hebben geschreven nog eens opnieuw bekijken:

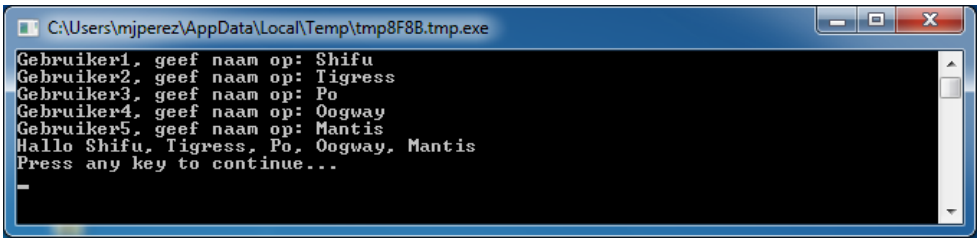
```
TextWindow.Write("Geef je naam op: ")
naam = TextWindow.Read()
TextWindow.WriteLine("Hallo " + naam)
```

In dit programma hebben we de naam van de gebruiker ontvangen en opgeslagen in de variabele **naam**. Later hebben we de gebruiker begroet met “Hallo”. Laten we in dit voorbeeld aannemen dat er meerdere gebruikers zijn, zeg er zijn 5. Hoe kunnen we al hun namen opslaan? Hier volgt één manier hoe we dit kunnen doen:

```
TextWindow.Write("Gebruiker1, geef naam op: ")
naam1 = TextWindow.Read()
TextWindow.Write("Gebruiker2, geef naam op: ")
naam2 = TextWindow.Read()
TextWindow.Write("Gebruiker3, geef naam op: ")
naam3 = TextWindow.Read()
TextWindow.Write("Gebruiker4, geef naam op: ")
naam4 = TextWindow.Read()
TextWindow.Write("Gebruiker5, geef naam op: ")
naam5 = TextWindow.Read()
```

```
TextWindow.Write("Hallo ")
TextWindow.Write(naam1 + ", ")
TextWindow.Write(naam2 + ", ")
TextWindow.Write(naam3 + ", ")
TextWindow.Write(naam4 + ", ")
TextWindow.WriteLine(naam5)
```

Als je dit programma uitvoert, zie je het volgende resultaat:



Afbeelding 49 – Zonder het gebruik van matrices

Het is duidelijk dat er een betere manier moet zijn om een dergelijk eenvoudig programma te schrijven. Vooral omdat de computer heel goed is in het uitvoeren van herhalende taken. Het is niet nodig om dezelfde code voor elke nieuwe gebruiker te schrijven. De truc hier is om met dezelfde variabele de naam van meerdere gebruikers op te slaan en op te halen. Als we dit kunnen doen, kunnen we een **For**-lus gebruiken die we in vorige lessen hebben geleerd. En hier kunnen we gebruikmaken van matrices.

Wat is een matrix?

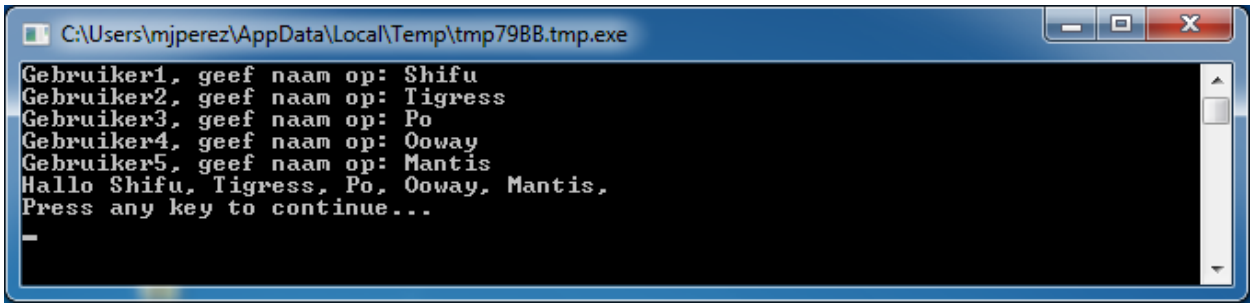
Een matrix is een speciale variabele die meer dan een waarde tegelijkertijd kan opslaan. Dit betekent dat we **naam1, naam2, naam3, naam4** en **naam5** niet hoeven te maken om deze vijf gebruikersnamen op te slaan, maar dat we met **naam** alle vijf gebruikersnamen kunnen opslaan. We kunnen meerdere waarden opslaan met gebruik van een 'index'. Bijvoorbeeld, **naam[1], naam[2], naam[3], naam[4]** en **naam[5]** kunnen elk een waarde opslaan. De getallen 1, 2, 3, 4 en 5 worden *indexen* bij de matrix genoemd.

Het lijkt alsof **naam[1]**, **naam[2]**, **naam[3]**, **naam[4]** en **naam[5]** verschillende variabelen zijn, maar in werkelijkheid vormen ze samen één variabele. En je kunt je afvragen wat het voordeel hier van is. Het grootste voordeel van het opslaan van waarden in een matrix is dat je de index kunt specificeren met een andere variabele, waarmee je gemakkelijk toegang krijgt tot matrices binnen lussen.

Laten we nu eens kijken hoe we met deze nieuwe kennis ons vorig programma met matrices kunnen herschrijven.

```
For i = 1 To 5
  TextWindow.Write("Gebruiker" + i + ", geef naam op: ")
  naam[i] = TextWindow.Read()
EndFor

TextWindow.Write("Hallo ")
For i = 1 To 5
  TextWindow.Write(naam[i] + ", ")
EndFor
TextWindow.WriteLine("")
```

Dit is veel gemakkelijker te lezen of niet dan? Let op de twee vetgedrukte regels. Met de eerste regel wordt er een waarde in de matrix opgeslagen en met de tweede wordt de waarde van de matrix gelezen. De waarde die je opslaat in **naam[1]** zal de waarde die je hebt opgeslagen in **naam[2]** niet veranderen. Voor de meeste doeleinden kunnen je daarom **naam[1]** en **naam[2]** beschouwen als twee verschillende variabelen met dezelfde identiteit.



Afbeelding 50 – Matrices gebruiken

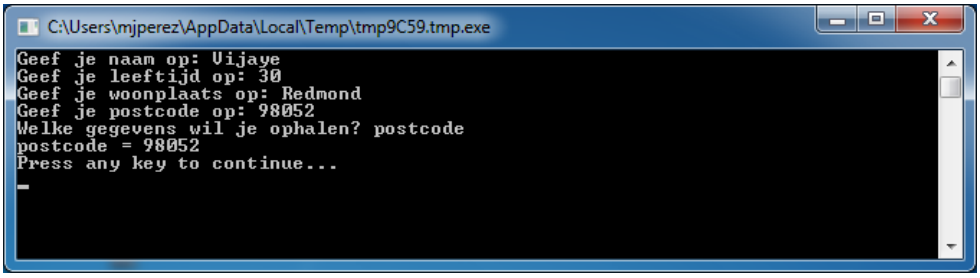
In het bovenstaande programma is het resultaat precies hetzelfde als in het programma zonder matrices. Het enige verschil is de komma aan het einde van *Mantis*. We kunnen dit oplossen door de printloop als volgt te herschrijven:

```
TextWindow.Write("Hallo ")
For i = 1 To 5
  TextWindow.Write(naam[i])
  If i < 5 Then
    TextWindow.Write(", ")
  EndIf
EndFor
TextWindow.WriteLine("")
```

Een matrix indexeren

In ons vorig programma heb je gezien dat we getallen hebben gebruikt als indexen om waarden van de matrix op te slaan en op te halen. Indexen worden echter niet alleen gebruikt voor getallen. In de praktijk is het erg nuttig om ook tekstuele indexen te gebruiken. In het volgende programma gaan we bijvoorbeeld informatie over een gebruiker opslaan en vervolgens laten we de computer alleen de gegevens naar het tekstvenster te schrijven waar de gebruiker om vraagt.

```
TextWindow.Write("Geef je naam op: ")
gebruiker["naam"] = TextWindow.Read()
TextWindow.Write("Geef je leeftijd op: ")
gebruiker["leeftijd"] = TextWindow.Read()
TextWindow.Write("Geef je woonplaats op: ")
gebruiker["woonplaats"] = TextWindow.Read()
TextWindow.Write("Geef je postcode op: ")
gebruiker["postcode"] = TextWindow.Read()

TextWindow.Write("Welke gegevens wil je ophalen? ")
index = TextWindow.Read()
TextWindow.WriteLine(index + " = " + gebruiker[index])
```



Afbeelding 51 – Niet-numerieke indexen gebruiken

Meer dan een dimensie

Zeg dat je de naam en het telefoonnummer van al je vrienden wilt opslaan om hun telefoonnummers op te kunnen zoeken wanneer je die nodig hebt. Net als in een telefoonboek. Hoe gaan we een dergelijk programma schrijven?

Matrixindexen zijn niet hoofdlettergevoelig. Net zoals bij gewone variabelen hoeven de hoofdletters van de overeenkomsten van matrixindexen niet precies overeen te komen.

In dit geval zijn er twee series indexen (ook bekend als de dimensie van de matrix). Laten we aannemen dat we elke vriend kunnen herkennen aan zijn of haar bijnaam. Dit wordt onze eerste index in de matrix. Nadat we onze eerste index hebben gebruikt voor het ophalen van onze vriendvariabele, gebruiken we de tweede index, **naam** en **telefoonnummer** om de daadwerkelijke naam en telefoonnummer van die vriend op te halen.

Deze gegevens worden op de volgende manier opgeslagen:

```
vrienden["Rob"]["Naam"] = "Robert"
vrienden["Rob"]["Telefoon"] = "06-34345658"

vrienden["VJ"]["Naam"] = "Vijaye"
vrienden["VJ"]["Telefoon"] = "06-41732679"

vrienden["Ash"]["Naam"] = "Ashley"
vrienden["Ash"]["Telefoon"] = "06-21345678"
```

Aangezien we twee indexen hebben in dezelfde matrix, **vrienden**, wordt dit een tweedimensionale matrix genoemd.

Als we dit programma eenmaal hebben ingesteld, kunnen we de bijnaam van onze vrienden als invoer gebruiken en vervolgens de informatie die we over ze hebben opgeslagen door de computer naar het tekstvenster laten schrijven. Hier is het volledige programma dat dit doet:

```
vrienden["Rob"]["Naam"] = "Robert"
vrienden["Rob"]["Telefoon"] = "06-34345658"

vrienden["VJ"]["Naam"] = "Vijaye"
vrienden["VJ"]["Telefoon"] = "06-41732679"

vrienden["Ash"]["Naam"] = "Ashley"
vrienden["Ash"]["Telefoon"] = "06-21345678"

TextWindow.Write("Geef de bijnaam op: ")
bijnaam = TextWindow.Read()

TextWindow.WriteLine("Naam: " + vrienden[bijnaam]["Naam"])
TextWindow.WriteLine("Telefoon: " + vrienden[bijnaam]["Telefoon"])
```

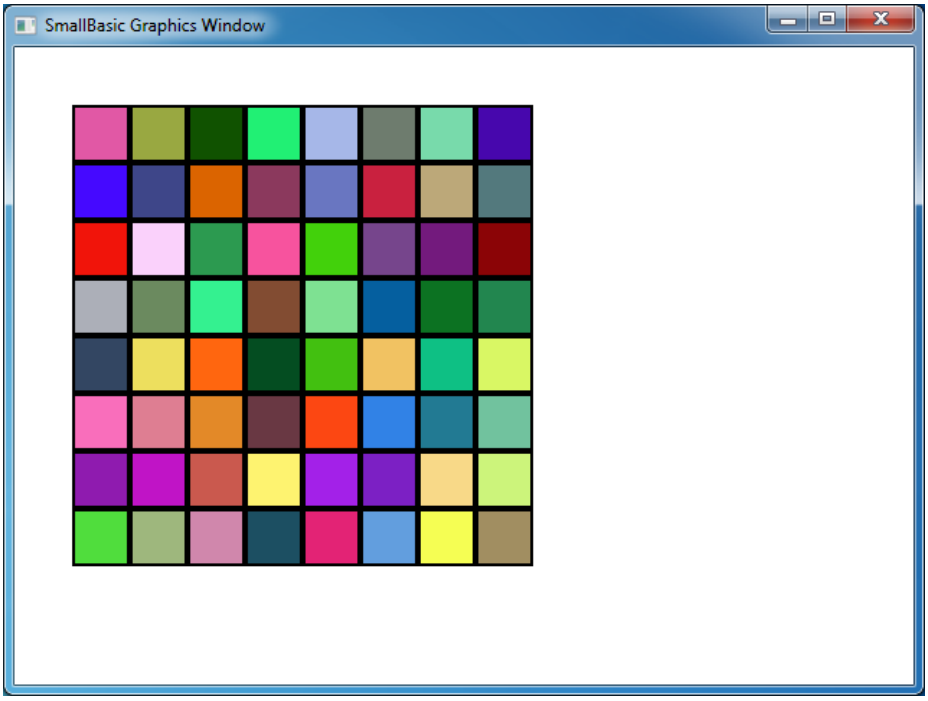
Afbeelding 52 – Een eenvoudig telefoonboek

Matrices gebruiken voor rasters

Multidimensionale matrices worden veel gebruikt voor het maken van rasters/tabellen. Rasters hebben rijen en kolommen die goed passen in een tweedimensionale matrix. Hieronder wordt een eenvoudig programma weergeven waarin vakken in een raster worden gerangschikt:

```
rijen = 8
kolommen = 8
afmeting = 40

For r = 1 To rijen
  For k = 1 To kolommen
    GraphicsWindow.BrushColor = GraphicsWindow.GetRandomColor()
    vakken[r][k] = Shapes.AddRectangle(afmeting, afmeting)
    Shapes.Move(vakken[r][k], k * afmeting, r * afmeting)
  EndFor
EndFor
```

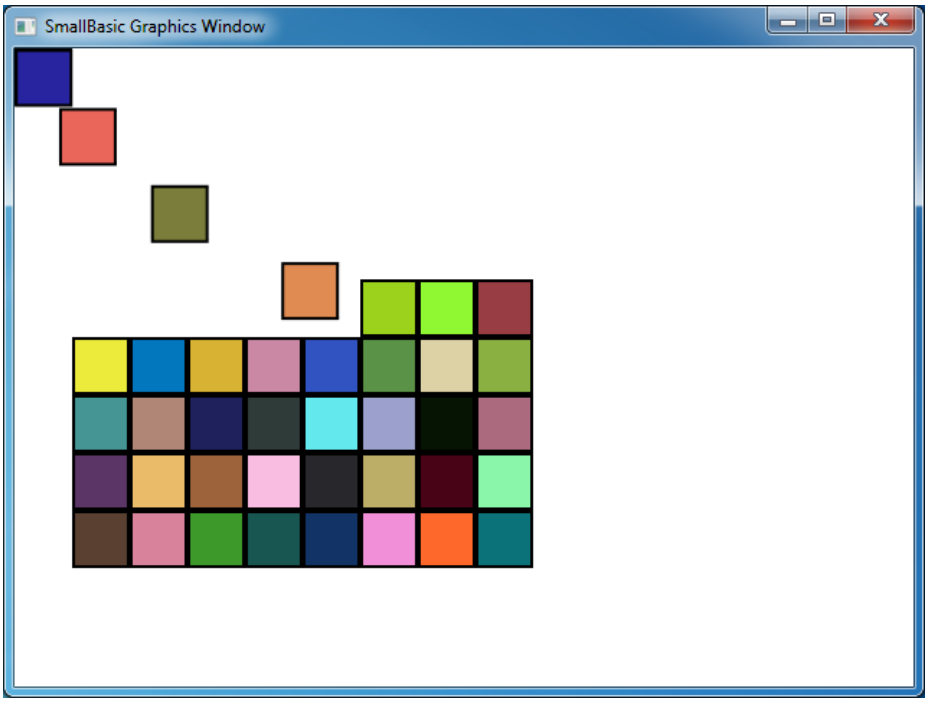
Met dit programma worden rechthoeken in een raster van 8x8 geplaatst. Naast dat deze vakken worden gerangschikt, worden ze ook opgeslagen in een matrix. Hiermee kunnen we de vakken gemakkelijk opnieuw ophalen als we de ze later opnieuw nodig hebben.



Afbeelding 53 – Vakken in een raster rangschikken

Als je bijvoorbeeld de volgende code toevoegt aan het einde van het vorige programma, worden de vakken naar de linkerbovenhoek verplaatst.

```
For r = 1 To rijen
  For k = 1 To kolommen
    Shapes.Animate(vakken[r][k], 0, 0, 1000)
    Program.Delay(300)
  EndFor
EndFor
```



Afbeelding 54 – De vakken in het raster bijhouden

Hoofdstuk 11

Gebeurtenissen en interactiviteit

In de eerste twee hoofdstukken hebben we objecten met *eigenschappen* en *bewerkingen* geïntroduceerd. Sommige objecten hebben naast eigenschappen en bewerkingen ook ***gebeurtenissen***. Gebeurtenissen kunnen je vergelijken met seintjes die bijvoorbeeld worden gegeven als een reactie op gebruikersacties zoals het verplaatsen van of klikken op de muis. Gebeurtenissen zijn in zekere zin het tegenovergestelde van bewerkingen. In het geval van een bewerking vraag jij als programmeur de computer iets te doen, terwijl in het geval van gebeurtenissen de computer jou laten weten wanneer er iets interessants is gebeurd.

Waarom zijn gebeurtenissen nuttig?

Gebeurtenissen zijn belangrijke als je interactiviteit aan een programma wilt toevoegen. Je moet gebeurtenissen gebruiken als je een gebruiker wilt laten deelnemen aan je programma. Als je bijvoorbeeld een boter-kaas-en-eieren-spelletje aan het schrijven bent, wil je natuurlijk dat de gebruiker een keuze kan maken als het zijn of haar beurt is. En hier gaan we gebeurtenissen gebruiken. Met gebeurtenissen ontvang je gebruikersinvoer binnen je programma. Dit lijkt nogal ingewikkeld, maar maak je geen zorgen, we zullen in een eenvoudig voorbeeld uitleggen wat gebeurtenissen zijn en hoe ze worden gebruikt.

Hieronder volgt een zeer eenvoudig programma met slechts één instructie en één subroutine. De subroutine gebruikt de bewerking *ShowMessage* op het object GraphicsWindow om zo een berichtvenster aan de gebruiker te tonen.

```
GraphicsWindow.MouseDown = OnMouseDown

Sub OnMouseDown
  GraphicsWindow.ShowMessage("Je hebt geklikt.", "Hallo")
EndSub
```

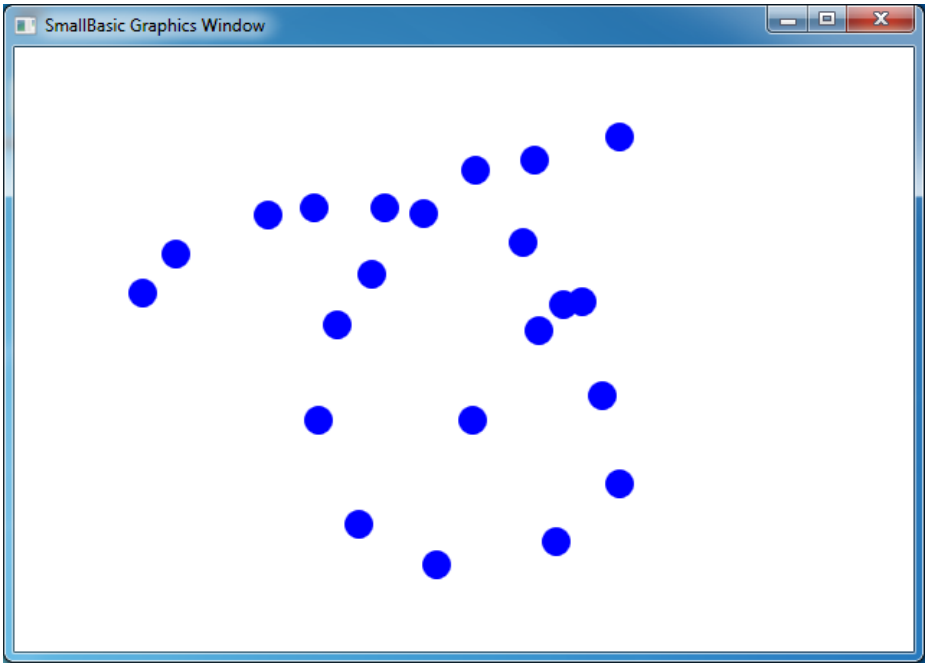
Het interessante gedeelte in het programma hierboven is de regel waar we de naam van de subroutine toewijzen aan de **MouseDown** -gebeurtenis van het GraphicsWindow-object. Je zult zien dat MouseDown erg veel lijkt op een eigenschap, maar in plaats van dat we waarden toewijzen, wijzen we de subroutine *OnMouseDown* toe. Dit maakt gebeurtenissen zo speciaal. Wanneer de gebeurtenis plaatsvindt, wordt de subroutine automatisch opgeroepen. In dit geval wordt de subroutine *OnMouseDown* elke keer opgeroepen als de gebruiker met de muis op het GraphicsWindow klikt. Voer het programma uit en probeer het zelf. Telkens wanneer je met de muis op het GraphicsWindow klikt, zie je een berichtvenster net zoals hieronder wordt weergegeven.

Afbeelding 55 – Reageren op een gebeurtenis

Deze manier van gebeurtenisverwerking is krachtig en stelt je in staat creatieve en interessante programma's te maken. Programma's die op deze manier worden geschreven, worden vaak gebeurtenisafhankelijke programma's genoemd.

Je kunt de subroutine *OnMouseDown* wijzigen om andere dingen te doen dan berichtvensters weer te geven. In het programma hieronder kun je bijvoorbeeld grote blauwe stippen tekenen daar waar de gebruiker met de muis klikt.

```
GraphicsWindow.BrushColor = "Blue"
GraphicsWindow.MouseDown = OnMouseDown

Sub OnMouseDown
  x = GraphicsWindow.MouseX - 10
  y = GraphicsWindow.MouseY - 10
  GraphicsWindow.FillEllipse(x, y, 20, 20)
EndSub
```



Afbeelding 56 – MouseDown-gebeurtenis verwerken

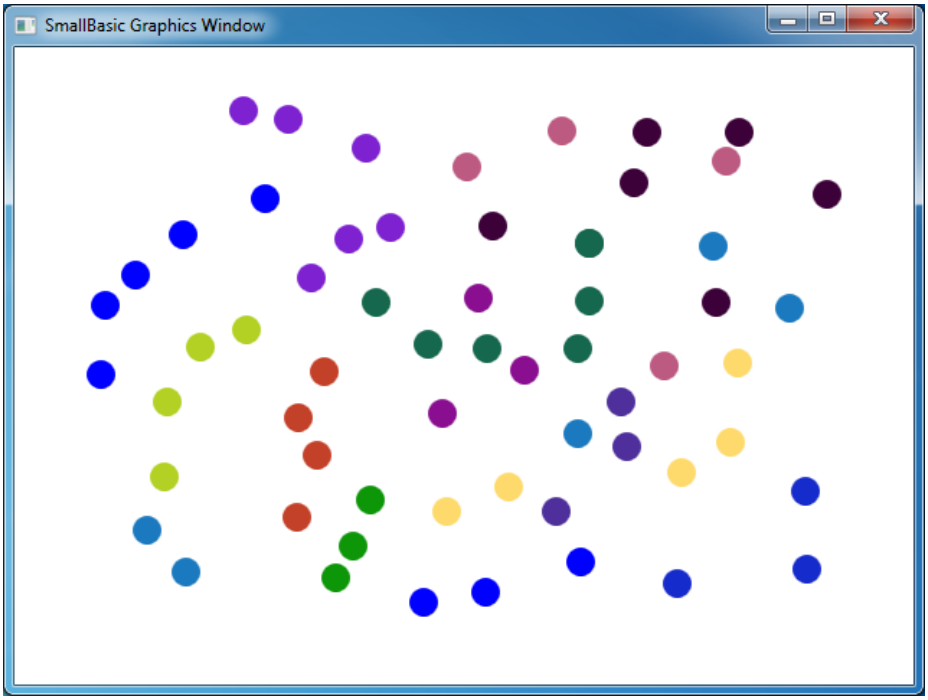
In het programma hierboven hebben we *MouseX* en *MouseY* gebruikt om de muiscoördinaten te verkrijgen. We gebruiken dit vervolgens om een cirkel te tekenen met de muiscoördinaten als het middelpunt van de cirkel.

Meerdere gebeurtenissen verwerken

Het gebruik van gebeurtenissen is ongelimiteerd. Je kunt zelfs één subroutine meerdere gebeurtenissen laten verwerken. Je kunt een gebeurtenis echter slechts één keer verwerken. Als je twee subroutines aan dezelfde gebeurtenis probeert toe te wijzen, wordt de tweede verwerkt.

In het volgende voorbeeld wordt dit duidelijk gemaakt met een subroutine die toetsaanslagen verwerkt. Ook laten we deze nieuwe subroutine de kleur van de kwast wijzigen zodat de stip een nieuwe kleur krijgt wanneer je met de muis klikt.

```
GraphicsWindow.BrushColor = "Blue"
GraphicsWindow.MouseDown = OnMouseDown
GraphicsWindow.KeyDown = OnKeyDown

Sub OnKeyDown
  GraphicsWindow.BrushColor = GraphicsWindow.GetRandomColor()
EndSub

Sub OnMouseDown
  x = GraphicsWindow.MouseX - 10
  y = GraphicsWindow.MouseY - 10
  GraphicsWindow.FillEllipse(x, y, 20, 20)
EndSub
```



Afbeelding 57 – Meerdere gebeurtenissen verwerken

Als je dit programma hebt uitgevoerd en op het venster hebt geklikt, wordt er een blauwe stip weergegeven. Als je nu één keer op een willekeurige toets klikt, wordt er een stip in een andere kleur weergegeven. Dit gebeurt omdat je op een toets drukt waardoor de subroutine *OnKeyDown* wordt uitgevoerd en hiermee wordt de kleur van de kwast naar een willekeurige kleur gewijzigd. Als je hierna op de muis klikt, wordt er een cirkel getekend met de nieuwe ingestelde kleur en dit geeft de stippen de willekeurige kleuren.

Een tekenprogramma

Met gebeurtenissen en subroutines kunnen we nu een programma schrijven waarmee we gebruikers op het venster kunnen laten tekenen. Het is verassend eenvoudig een dergelijk programma te schrijven als we het programma in kleinere stukjes opsplitsen. Als eerste stap gaan we een programma schrijven waarmee de gebruiker de muis overal in het venster voor grafische afbeeldingen kan verplaatsen en een spoor kan achterlaten.

```
GraphicsWindow.MouseMove = OnMouseMove

Sub OnMouseMove
  x = GraphicsWindow.MouseX
  y = GraphicsWindow.MouseY
  GraphicsWindow.DrawLine(prevX, prevY, x, y)
  prevX = x
  prevY = y
EndSub
```

Als je dit programma uitvoert, begint de eerste lijn echter altijd aan de linkerbovenkant van het venster (0, 0). We kunnen dit probleem oplossen met het verwerken van de *MouseDown*-gebeurtenis en het vastleggen van *prevX*- en *prevY*-waarden wanneer die gebeurtenis wordt verwerkt.

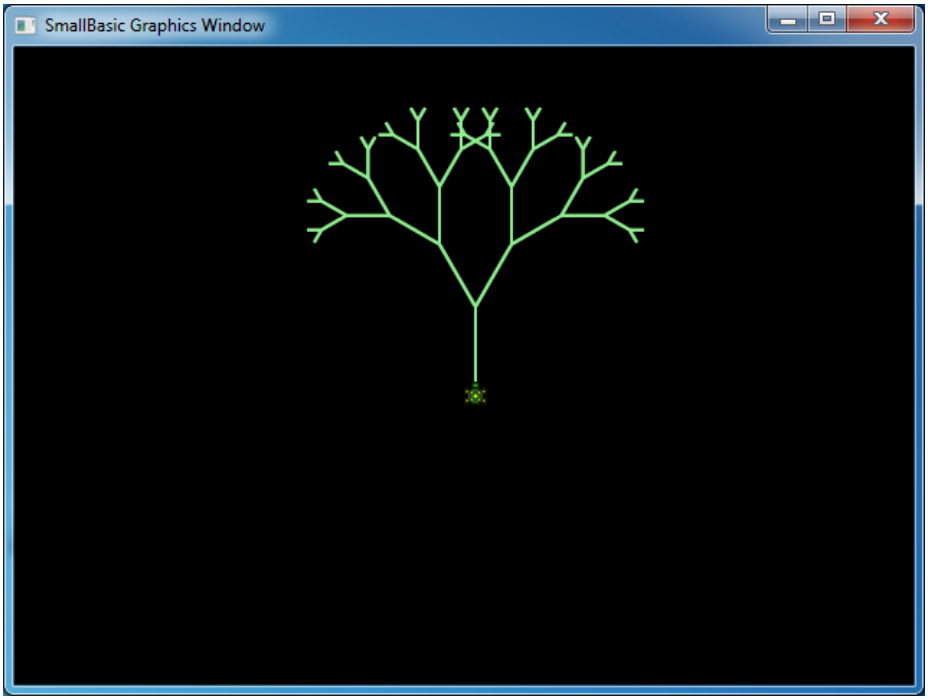
Bovendien hoeven we eigenlijk alleen maar een spoor achter te laten wanneer de gebruiker de muisknop ingedrukt houdt. Als de gebruiker dit niet doet, moet de lijn niet worden getekend. Als we dit gedrag willen verkrijgen, gebruiken we de *IsLeftButtonDown*-eigenschap van het **Mouse**-object. Deze eigenschap vertelt ons wanneer de linkerknop wordt ingedrukt of niet. Als deze waarde waar is, wordt de lijn getekend en anders wordt de lijn niet getekend.

```
GraphicsWindow.MouseMove = OnMouseMove
GraphicsWindow.MouseDown = OnMouseDown

Sub OnMouseDown
 prevX = GraphicsWindow.MouseX
  prevY = GraphicsWindow.MouseY
EndSub

Sub OnMouseMove
  x = GraphicsWindow.MouseX
  y = GraphicsWindow.MouseY
  If (Mouse.IsLeftButtonDown) Then
    GraphicsWindow.DrawLine(prevX, prevY, x, y)
  EndIf
  prevX = x
  prevY = y
EndSub
```

Bijlage A:

Leuke voorbeelden

Schildpadfractal



Afbeelding 58 – Schildpad tekent boomfractal

```
hoek = 30
delta = 10
afstand = 60
Turtle.Speed = 9
GraphicsWindow.BackgroundColor = "Black"
GraphicsWindow.PenColor = "LightGreen"
DrawTree()

Sub DrawTree
  If (afstand > 0) Then
    Turtle.Move(afstand)
    Turtle.Turn(hoek)

    Stack.PushValue("afstand", afstand)
    afstand = afstand - delta
    DrawTree()
    Turtle.Turn(-hoek * 2)
    DrawTree()
    Turtle.Turn(hoek)
    afstand = Stack.PopValue("afstand")

    Turtle.Move(-afstand)
  EndIf
EndSub
```

Foto's van Flickr

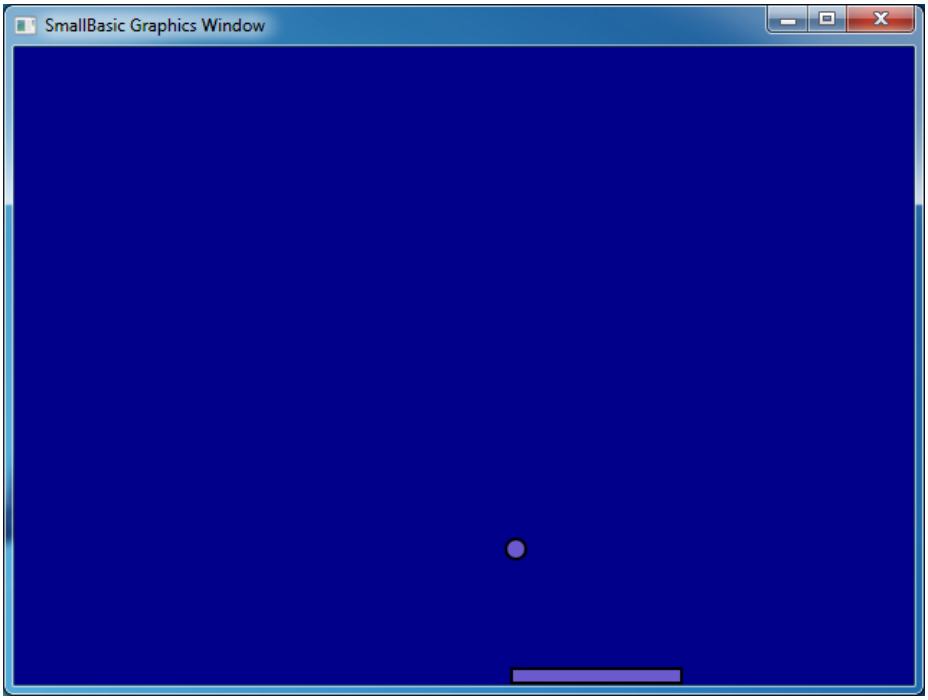
Afbeelding 59 – Foto's ophalen van Flickr

```
GraphicsWindow.BackgroundColor = "Black"
GraphicsWindow.MouseDown = OnMouseDown

Sub OnMouseDown
    foto = Flickr.GetRandomPicture("bergen, rivieren")
    GraphicsWindow.DrawResizedImage(foto, 0, 0, 640, 480)
EndSub
```

Een dynamische bureaubladachtergrond

```
For i = 1 To 10
  foto = Flickr.GetRandomPicture("bergen")
  Desktop.SetWallPaper(foto)
  Program.Delay(10000)
EndFor
```

Peddelspel



Afbeelding 60 – Peddelspel

```
GraphicsWindow.BackgroundColor = "DarkBlue"
peddel = Shapes.AddRectangle(120, 12)
bal = Shapes.AddEllipse(16, 16)
GraphicsWindow.MouseMove = OnMouseMove

x = 0
y = 0
deltaX = 1
deltaY = 1

RunLoop:
  x = x + deltaX
  y = y + deltaY

  gw = GraphicsWindow.Width
  gh = GraphicsWindow.Height
  If (x >= gw - 16 or x <= 0) Then
    deltaX = -deltaX
  EndIf
  If (y <= 0) Then
    deltaY = -deltaY
  EndIf
```

```
  pedX = Shapes.GetLeft (peddel)
  If (y = gh - 28 and x >= pedX and x <= pedX + 120) Then
    deltaY = -deltaY
  EndIf

  Shapes.Move(bal, x, y)
  Program.Delay(5)

  If (y < gh) Then
    Goto RunLoop
  EndIf

GraphicsWindow.ShowMessage("Jij verliest", "Peddel")

Sub OnMouseMove
  peddelX = GraphicsWindow.MouseX
  Shapes.Move(peddel, peddelX - 60, GraphicsWindow.Height - 12)
EndSub
```

Bijlage B:

Kleuren

Hieronder volgt een lijst met benoemde kleuren, gecategoriseerd op hun basistint, die binnen Small Basic worden ondersteund.

Rode kleuren

IndianRed	#CD5C5C
LightCoral	#F08080
Salmon	#FA8072
DarkSalmon	#E9967A
LightSalmon	#FFA07A
Crimson	#DC143C
Red	#FF0000
FireBrick	#B22222
DarkRed	#8B0000

Roze kleuren

Pink	#FFC0CB
LightPink	#FFB6C1
HotPink	#FF69B4
DeepPink	#FF1493
MediumVioletRed	#C71585
PaleVioletRed	#DB7093

Oranje kleuren

LightSalmon	#FFA07A
Coral	#FF7F50
Tomato	#FF6347
OrangeRed	#FF4500
DarkOrange	#FF8C00
Orange	#FFA500

Gele kleuren

Gold	#FFD700
Yellow	#FFFF00
LightYellow	#FFFFE0
LemonChiffon	#FFFACD
LightGoldenrodYellow	#FAFAD2
PapayaWhip	#FFEFD5
Moccasin	#FFE4B5

PeachPuff	#FFDAB9
PaleGoldenrod	#EEE8AA
Khaki	#F0E68C
DarkKhaki	#BDB76B

Paarse kleuren

Lavender	#E6E6FA
Thistle	#D8BFD8
Plum	#DDA0DD
Violet	#EE82EE
Orchid	#DA70D6
Fuchsia	#FF00FF
Magenta	#FF00FF
MediumOrchid	#BA55D3
MediumPurple	#9370DB
BlueViolet	#8A2BE2
DarkViolet	#9400D3
DarkOrchid	#9932CC
DarkMagenta	#8B008B
Purple	#800080
Indigo	#4B0082
SlateBlue	#6A5ACD
DarkSlateBlue	#483D8B
MediumSlateBlue	#7B68EE

Groene kleuren

GreenYellow	#ADFF2F
Chartreuse	#7FFF00
LawnGreen	#7CFC00
Lime	#00FF00
LimeGreen	#32CD32
PaleGreen	#98FB98
LightGreen	#90EE90
MediumSpringGreen	#00FA9A
SpringGreen	#00FF7F
MediumSeaGreen	#3CB371
SeaGreen	#2E8B57
ForestGreen	#228B22
Green	#008000
DarkGreen	#006400
YellowGreen	#9ACD32
OliveDrab	#6B8E23
Olive	#808000
DarkOliveGreen	#556B2F
MediumAquamarine	#66CDAA
DarkSeaGreen	#8FBC8F
LightSeaGreen	#20B2AA
DarkCyan	#008B8B
Teal	#008080

Blauwe kleuren

Aqua	#00FFFF
Cyaan	#00FFFF
LightCyan	#E0FFFF
PaleTurquoise	#AFEEEE
Aquamarine	#7FFFD4
Turquoise	#40E0D0
MediumTurquoise	#48D1CC
DarkTurquoise	#00CED1
CadetBlue	#5F9EA0
SteelBlue	#4682B4
LightSteelBlue	#B0C4DE
PowderBlue	#B0E0E6
LightBlue	#ADD8E6

SkyBlue	#87CEEB
LightSkyBlue	#87CEFA
DeepSkyBlue	#00BFFF
DodgerBlue	#1E90FF
CornflowerBlue	#6495ED
MediumSlateBlue	#7B68EE
RoyalBlue	#4169E1
Blue	#0000FF
MediumBlue	#0000CD
DarkBlue	#00008B
Navy	#000080
MidnightBlue	#191970

Bruine kleuren

Cornsilk	#FFF8DC
BlanchedAlmond	#FFEBCD
Bisque	#FFE4C4
NavajoWhite	#FFDEAD
Wheat	#F5DEB3
BurlyWood	#DEB887
Tan	#D2B48C
RosyBrown	#BC8F8F
SandyBrown	#F4A460
Goldenrod	#DAA520
DarkGoldenrod	#B8860B
Peru	#CD853F
Chocolate	#D2691E
SaddleBrown	#8B4513
Sienna	#A0522D
Brown	#A52A2A
Maroon	#800000

Witte kleuren

White	#FFFFFF
Snow	#FFFAFA
Honeydew	#F0FFF0
MintCream	#F5FFFA
Azure	#F0FFFF
AliceBlue	#F0F8FF
GhostWhite	#F8F8FF
WhiteSmoke	#F5F5F5
Seashell	#FFF5EE
Beige	#F5F5DC
OldLace	#FDF5E6
FloralWhite	#FFFAF0
Ivory	#FFFFF0
AntiqueWhite	#FAEBD7
Linen	#FAF0E6
LavenderBlush	#FFF0F5
MistyRose	#FFE4E1

Grijze kleuren

Gainsboro	#DCDCDC
LightGray	#D3D3D3
Silver	#C0C0C0
DarkGray	#A9A9A9
Gray	#808080
DimGray	#696969
LightSlateGray	#778899
SlateGray	#708090
DarkSlateGray	#2F4F4F
Black	#000000