De developer's reference guide to Small Basic: 2. Overzicht van Small Basic Programming

Voorbeeld

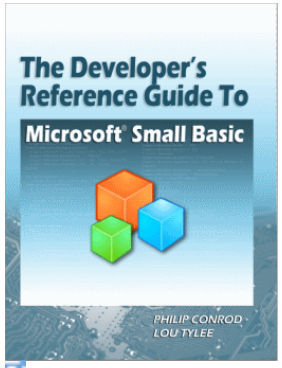
In dit hoofdstuk houden we ons bezig met het schrijven van code voor onze programma's. We zullen een overzicht geven van veel van de elementen van de taal die in Small Basic wordt gebruikt. Dit hoofdstuk is in wezen een op zichzelf staande gids voor de kleine basistaal. Dit geeft ons de basis die nodig is om te beginnen met het leren over kleine basisobjecten en het schrijven van meer gedetailleerde programma's.

Een korte geschiedenis van BASIC

De BASIC-taal werd ontwikkeld in de vroege jaren 1960 aan het Dartmouth College als een apparaat voor het onderwijzen van programmeren aan "gewone" mensen. Het heet niet voor niets BASIC:

B (Beginner's)
A (All-Purpose)
S (Symbolisch)
I (Instructie)
C (Code)

Toen timesharing-systemen in de jaren 1960 werden geïntroduceerd, was BASIC de taal bij uitstek. Veel van de eerste computersimulatiespellen (Star Trek, bijvoorbeeld) werden geschreven in timeshare BASIC. In het midden van de jaren 1970 besloten twee studenten dat de nieuwe Altair-microcomputer een BASIC-taaltolk nodig had. Ze verkochten hun product op cassettebandje voor een bedrag van $ 350. Je hebt misschien wel eens gehoord van deze ondernemers: Bill Gates en Paul Allen!



Dit hoofdstuk is een bewerking van het boek *The Developer's Reference Guide To Microsoft Small Basic* van Philip Conrod en Lou Tylee.

Om dit boek in zijn geheel te kopen, zie de Computer Science For Kids website.

Elke BASIC die sindsdien is geschreven, is gebaseerd op die vroege versie. Voorbeelden hiervan zijn: GW-Basic, QBasic, QuickBasic, Visual Basic. Alle speelgoedcomputers van de vroege jaren 80 (herinnert iemand zich TI99/4A, Commodore 64, Timex, Atari 400?) gebruikten BASIC voor het programmeren. Small Basic zet de traditie van BASIC programmeren voort. Het maakt gebruik van de eenvoudige concepten van de vroege BASIC-taal met een moderne ontwikkelomgeving.

Dit hoofdstuk geeft een overzicht van de BASIC-taal die wordt gebruikt in de Small Basic-omgeving. Als je ooit een andere programmeertaal (of een versie van BASIC) hebt gebruikt, zie je gelijkwaardige structuren in de taal van Small Basic.

Variabelen

Variabelen worden door Small Basic gebruikt om informatie vast te houden die nodig is voor een toepassing. Variabelen moeten de juiste naam krijgen. Regels die worden gebruikt bij het benoemen van variabelen:

- Niet meer dan 40 tekens
- Ze kunnen letters, cijfers en onderstrepingstekens (_) bevatten.
- Het eerste teken moet een letter zijn
- U kunt geen gereserveerd woord gebruiken (trefwoorden die worden gebruikt door Small Basic)

Gebruik betekenisvolle variabelenamen die u (of andere programmeurs) helpen het doel van de informatie die door de variabele is opgeslagen, te begrijpen.

Voorbeelden van acceptabele variabelenamen:

Starttijd	Interest_Value	Brief05
Johnsage	Number_of_Days	TimeOfDay

Kleine basisgegevenstypen

Elke variabele wordt gebruikt om informatie van een bepaald **type** op te slaan. Small Basic gebruikt drie soorten gegevens: **numerieke**, **tekenreeksen** (of tekst) en **Booleaanse** variabelen. U moet altijd weten welk type informatie is opgeslagen in een bepaalde variabele.

Numerieke variabelen kunnen gehele getallen of decimale getallen opslaan. Ze kunnen positief of negatief zijn.

Een **tekenreeksvariabele** is precies dat - een die een tekenreeks (lijst) van verschillende tekens opslaat. Een tekenreeks kan een naam zijn, een reeks cijfers, een zin, een alinea, alle tekens. En vaak bevat een tekenreeks helemaal geen tekens (een lege tekenreeks). We zullen veel strings gebruiken in Small Basic, dus het is iets waar je vertrouwd mee moet raken. Strings staan altijd tussen aanhalingstekens (""). Voorbeelden van strings:

```
"Ik ben een Small Basic programmeur"    "012345"    "Titel Auteur"
```

Booleaanse variabelen kunnen een van de twee verschillende tekenreekswaarden hebben: **"true"** of **"false"**. De aanhalingstekens moeten aangeven dat dit tekenreekswaarden zijn. Booleaanse variabelen zijn nuttig bij het nemen van beslissingen.

Met alle verschillende variabele typen moeten we oppassen dat we typen niet onjuist mengen. We kunnen alleen wiskundige bewerkingen uitvoeren op getallen (integer- en decimale typen). Tekenreekstypen mogen alleen met andere tekenreekstypen werken. Booleaanse typen worden gebruikt voor beslissingen.

Small Basic heeft geen vereisten (of mogelijkheden) voor het declareren van variabelen voordat ze worden gebruikt. Ze worden in wezen aangegeven de eerste keer dat ze worden gebruikt.

Arrays

Small Basic heeft faciliteiten voor het verwerken van arrays, die een manier bieden om een groot aantal variabelen onder dezelfde naam op te slaan. Elke variabele, een element genoemd, in een array moet hetzelfde gegevenstype hebben en ze worden van elkaar onderscheiden door een array-index die tussen haakjes [**]** staat.

Arrays worden gebruikt op een manier die identiek is aan die van reguliere variabelen. Het negende element van een array met de naam **Item** is bijvoorbeeld:

```
Artikel[9]
```

De index op een arrayvariabele begint bij 0 en eindigt bij de hoogste gebruikte waarde. Daarom heeft de array **Item** in het bovenstaande voorbeeld eigenlijk **tien** elementen, variërend van Item[0] tot Item[9]. Dit is anders dan in andere talen. U gebruikt arrayvariabelen net als elke andere variabele - vergeet niet om de naam en de index op te nemen. Vaak wordt de 0-index genegeerd en beginnen we gewoon met punt 1. Maar soms kan het^{0e} element niet worden genegeerd. U ziet voorbeelden van zowel 0-gebaseerde als 1-gebaseerde arrays in de cursusvoorbeelden.

U kunt multidimensionale arrays hebben. Een tweedimensionaal array-element wordt geschreven als:

```
AnotherArray[2][7]
```

Dit verwijst naar het element in de^{2e} rij en^{7e} kolom van de array **AnotherArray**.

Intellisense-functie

Werken in het venster van de code-editor is eenvoudig. U zult zien dat het typen van code net als het gebruik van een tekstverwerker is. De gebruikelijke navigatie- en bewerkingsfuncties zijn er allemaal.

Een functie waar u vertrouwd mee zult raken en versteld zult staan, wordt **Intellisense** genoemd. Terwijl u code typt, biedt de Intellisense-functie soms hulp bij het invullen van coderegels. Zodra u bijvoorbeeld een objectnaam en een punt (.) typt, wordt een vervolgkeuzelijst met mogelijke eigenschappen en methoden weergegeven.

Intellisense is een zeer nuttig onderdeel van Small Basic. U moet bekend raken met het gebruik ervan en hoe u voorgestelde waarden kunt selecteren. We vertellen je over nu, zodat je niet verbaasd zult zijn als er kleine vakjes verschijnen terwijl je code typt.

Kleine basisverklaringen en uitdrukkingen

De eenvoudigste (en meest voorkomende) instructie in Small Basic is de **opdrachtverklaring**. Het bestaat uit een variabelenaam, gevolgd door de toewijzingsoperator (**=**), gevolgd door een soort **expressie**. De expressie aan de rechterkant wordt geëvalueerd, waarna de variabele (of eigenschap) aan de linkerkant van de toewijzingsoperator wordt **vervangen** door die waarde van de expressie.

Voorbeelden:

```
StartTime = Now ExplorerName = "Captain Spaulding" TextWindow.Title = "My Program" BitCou
```

In het toewijzingsoverzicht wordt informatie opgeslagen.

Opmerkingsinstructies beginnen met één citaat ('). Bijvoorbeeld:

```
' This is a comment x = 2 * y ' another way to write a comment
```

U, als programmeur, moet beslissen hoeveel u uw code wilt becommentariëren. Houd rekening met factoren als hergebruik, uw doelgroep en de erfenis van uw code. In onze notities en voorbeelden proberen we commentaarverklaringen in te voegen wanneer dat nodig is om wat details uit te leggen.

Kleine elementaire rekenkundige operatoren

Operatoren wijzigen waarden van variabelen. De eenvoudigste **operatoren** voeren **rekenkundige** bewerkingen uit. Er zijn vier **rekenkundige operatoren** in Small Basic.

Optellen gebeurt met het plusteken (**+**) en **aftrekken** gebeurt met het minteken (**-**). Eenvoudige voorbeelden zijn:

Operatie	**Voorbeeld**	**Resultaat**
Optelling	7 + 2	9
Optelling	3.4 + 8.1	11.5
Aftrekking	6 - 4	2

Aftrekking	11.1 – 7.6	3.5

Vermenigvuldigen gebeurt met het sterretje (*****) en **deling** gebeurt met de schuine streep (**/**). Eenvoudige voorbeelden zijn:

Operatie	**Voorbeeld**	**Resultaat**
Vermenigvuldiging	8 * 4	32
Vermenigvuldiging	2.3 * 12.2	28.06
Divisie	12 / 2	6
Divisie	45.26 / 6.2	7.3

De wiskundige operatoren hebben de volgende **prioriteit** die de volgorde aangeeft waarin ze worden geëvalueerd zonder specifieke groeperingen:

1. Vermenigvuldiging (*) en deling (/)
2. Optellen (+) en aftrekken (-)

Als vermenigvuldigingen en delingen of optellingen en aftrekkingen in dezelfde expressie zijn, worden ze uitgevoerd in de volgorde van links naar rechts. **Haakjes** rond expressies worden gebruikt om de gewenste prioriteit af te dwingen.

Vergelijking en logische operatoren

Er zijn zes **vergelijkingsoperatoren** in Small Basic die worden gebruikt om de waarde van twee expressies te vergelijken (de expressies moeten van hetzelfde gegevenstype zijn). Dit zijn de basis voor het nemen van beslissingen:

Bediener	**Vergelijking**
>	Groter dan
<	Minder dan
>=	Groter dan of gelijk aan
<=	Kleiner dan of gelijk aan
=	Gelijk aan
<>	Niet gelijk aan

Het moet duidelijk zijn dat het resultaat van een vergelijkingsbewerking een Booleaanse waarde is (**"waar"** of **"onwaar"). Voorbeelden**:

A = 9,6, B = 8,1, A **>**B retourneert "true"
A = 14, B = 14, A **<** B retourneert "false"
A = 14, B = 14, A **>=** B retourneert "true"
A = 7, B = 11, A **<=** B returns "true"
A = "Small", B="Small", A **=** B returns "true"
A = "Basic", B = "Basic", A **<>** B returns "false"

Logische operatoren werken op Booleaanse gegevenstypen en leveren een Booleaans resultaat. Ze worden ook gebruikt bij de besluitvorming. We gebruiken twee **logische** operatoren

Bediener	**Operatie**
En	Logisch en
Of	Logisch of

De operator **And** controleert of twee verschillende Booleaanse gegevenstypen beide "waar" zijn. Als beide "waar" zijn, retourneert de operator een "waar". Anders retourneert het een "valse" waarde. Voorbeelden:

A = "waar", B = "waar", dan A **En** B = "waar"
A = "waar", B = "onwaar", dan A **En** B = "onwaar"
A = "onwaar", B = "waar", dan A **En** B = "onwaar"
A = "onwaar", B = "onwaar", dan A **En** B = "onwaar"

De operator **Or** controleert of een van de twee Booleaanse gegevenstypen "waar" is. Als een van beide "waar" is, retourneert de operator een "true". Anders retourneert het een "valse" waarde. Voorbeelden:

A = "waar", B = "waar", dan A **of** B = "waar"
A = "waar", B = "onwaar", dan A **of** B = "waar"
A = "onwaar", B = "waar", dan A **of** B = "waar"
A = "onwaar", B = "onwaar", dan A **of** B = "onwaar"

Logische operatoren volgen rekenkundige operatoren in voorrang. Het gebruik van deze operators zal duidelijk worden naarmate we ons verder verdiepen in codering.

Aaneenschakelingsoperator

Als u twee gegevenstypen tekenreeksen wilt **concatten** (aan elkaar koppelen), gebruikt u het symbool **+**, de aaneenschakelingsoperators voor tekenreeksen:

```
CurrentTime = "The current time is " + TimeNow SampleText = "Hook this " + "to this"
```

Wiskundige functies

Small Basic biedt een reeks methoden (of functies) die taken uitvoeren zoals vierkantswortels, goniometrische relaties en exponentiële functies. Ja, sommige programma's hebben betrekking op wiskunde!

Elk van de kleine wiskundefuncties is afkomstig uit de wiskundeles**.** Dit alles betekent dat elke functienaam moet worden voorafgegaan door **Math.** (zeg Math-dot) om goed te werken. De functies en de geretourneerde waarden worden hieronder weergegeven.

Wiskundige functies:

`Math.Abs(x) Returns the absolute value of x. Math.ArcSin(x) Returns the angle in radians,`

Voorbeelden:

`Math.Abs(-5.4) returns the absolute value of –5.4 (returns 5.4) Math.Cos(2.3) returns the`

Willekeurige getallen

We kiezen één wiskundige methode uit vanwege het belang ervan. Bij het schrijven van games en leersoftware gebruiken we een random number generator om onvoorspelbaarheid te introduceren. De methode **Math.GetRandomNumber** wordt in Small Basic gebruikt voor willekeurige getallen.

Wanneer u een willekeurige geheel getalwaarde (geheel getal) nodig hebt, gebruikt u deze methode:

```
Math.GetRandomNumber(Limiet)
```

Deze instructie genereert een willekeurig geheel getal dat tussen 1 en **Limiet ligt**. Bijvoorbeeld de methode:

```
Math.GetRandomNumber(5)
```

genereert willekeurige getallen van 1 tot 5. De mogelijke waarden zijn 1, 2, 3, 4 en 5.

Een rol van een matrijs kan een getal van 1 tot 6 produceren. Om **GetRandomNumber** te gebruiken om een dobbelsteen te rollen, zouden we schrijven:

```
DieNumber = Math.GetRandomNumber(6)
```

Voor een kaartspel zouden de willekeurige gehele getallen variëren van 1 tot 52, omdat er 52 kaarten in een standaard speelspel zijn. Code om dit te doen:

```
CardNumber = Math.GetRandomNumber(52)
```

Als we een getal tussen -100 en 100 willen, gebruiken we:

```
YourNumber = 101 - Math.GetRandomNumber(201)
```

Bekijk de bovenstaande voorbeelden om er zeker van te zijn dat u ziet hoe de willekeurige getalgenerator het gewenste bereik van gehele getallen produceert.

Kleine basisbeslissingen - Als-verklaringen

Het concept van een **If-verklaring** voor het nemen van een beslissing is heel eenvoudig. We controleren of een bepaalde aandoening "waar" is. Dan ondernemen we een bepaalde actie. Zo niet, dan doen we iets anders. **Als** instructies ook **wel vertakkingsinstructies** worden genoemd. **Vertakkingsinstructies** worden gebruikt om bepaalde acties binnen een programma te veroorzaken als aan een bepaalde voorwaarde wordt voldaan.

De eenvoudigste vertakkingsverklaring is:

```
If (Condition) Then [process this code] EndIf
```

Hier, als **Voorwaarde "**true" is, wordt de code die wordt begrensd door de **If/EndIf** uitgevoerd. Als **voorwaarde** "false" is, gebeurt er niets en wordt de code uitgevoerd na de **instructie EndIf**.

Voorbeeld:

```
If (Balance - Check < 0) Then Trouble = "true" CheckingStatus = "Red" EndIf
```

In dit geval, als **Saldo - Controle** minder dan nul is, worden twee regels informatie verwerkt: **Probleem** is ingesteld op "waar" en de **CheckingStatus** heeft de waarde "Rood". Let op de inspringing van de code tussen de regels **If** en **EndIf**. De Small Basic Intellisense-functie doet deze inspringing automatisch. Het maakt het begrijpen (en debuggen) van uw code veel gemakkelijker.

Wat als je één ding wilt doen als een voorwaarde "waar" is en een andere als het "onwaar" is? Gebruik een **If/Then/Else/EndIf-blok**:

```
If (Condition) Then [process this code] Else [process this code] EndIf
```

Als **voorwaarde in** dit blok "true" is, wordt de code tussen de **regels Als** en **Else** uitgevoerd. Als **Voorwaarde** "false" is, wordt de code tussen de **instructies Else** en **EndIf** verwerkt.

Voorbeeld:

```
If (Balance - Check < 0) Then Trouble = "true" CheckingStatus = "Red" Else Trouble = "fal
```

Hier worden dezelfde twee regels uitgevoerd als u overtekend bent (**Saldo - Controleer < 0**), maar als u niet overtekend bent (**Else**), is de **probleemvlag** uitgeschakeld en is uw **CheckingStatus** "Zwart".

Ten slotte kunnen we meerdere voorwaarden testen door de **ElseIf-instructie toe te** voegen:

```
If (Condition1) Then [process this code] ElseIf (Condition2) Then [process this code] Els
```

Als **voorwaarde1** in dit blok "true" is, wordt de code tussen de regel **If** en first **ElseIf** uitgevoerd. Als **Voorwaarde1** "onwaar" is, wordt **Voorwaarde2** aangevinkt. Als **Voorwaarde2** "true" is, wordt de aangegeven code uitgevoerd. Als **Voorwaarde2** niet waar is, wordt **Voorwaarde3** gecontroleerd. Elke volgende voorwaarde in de structuur wordt gecontroleerd totdat een "echte" voorwaarde is gevonden, een **Else-instructie** is bereikt of de **EndIf** is bereikt.

Voorbeeld:

```
If (Balance - Check < 0) Then Trouble = "true" CheckingStatus = "Red" ElseIf (Balance – C
```

Nu is er nog een voorwaarde toegevoegd. Als uw **saldo** gelijk is aan het **chequebedrag** (**ElseIfBalance - Check = 0**), bent u nog steeds niet in de problemen en is de **CheckingStatus** "Geel".

Zorg er bij het gebruik van vertakkingsverklaringen voor dat u rekening houdt met alle haalbare mogelijkheden in de **If/Else/EndIf-structuur**. Houd er ook rekening mee dat elke **If** en **ElseIf** in een blok sequentieel wordt getest. De eerste keer dat aan een **If-test** wordt voldaan, wordt de code die aan die voorwaarde is gekoppeld, uitgevoerd en wordt het **if-blok** afgesloten. Als een latere aandoening ook "waar" is, zal deze nooit worden overwogen.

Kleine Basic Looping

Veel toepassingen vereisen herhaling van bepaalde codesegmenten. U kunt bijvoorbeeld een dobbelsteen (gesimuleerde dobbelsteen natuurlijk) rollen totdat deze een zes toont. Of u kunt financiële resultaten genereren totdat een bepaalde som van het rendement is bereikt. Dit idee van het herhalen van code wordt iteratie of **looping** genoemd.

In Small Basic is een manier van looping met de **While-lus**:

```
While (Condition) ' Small Basic code block to repeat while Condition is true EndWhile
```

In deze structuur wordt alle code tussen **While** en **EndWhile** herhaald **terwijl** de gegeven logische **voorwaarde** "true" **is**.

Let op a **Terwijl** de lusstructuur zelfs niet één keer wordt uitgevoerd als **voorwaarde** de eerste keer "false" is. Als we de lus betreden (**Voorwaarde** is "waar"), wordt aangenomen dat op een gegeven moment **Voorwaarde** onwaar zal worden om exit mogelijk te maken. Zodra dit gebeurt, gaat de uitvoering van de code door bij de instructie na de instructie **EndWhile**. Dit brengt een heel belangrijk punt naar voren over lussen - als je erin

komt, zorg er dan voor dat je er op een gegeven moment uitstapt. In de **While-lus**, als **Condition** altijd "waar" is, zul je voor altijd een lus maken - iets dat een oneindige lus wordt genoemd.

Voorbeeld:

```
Counter = 1 While (Counter <= 1000) Counter = Counter + 1 EndWhile
```

Deze lus herhaalt zich zolang (**While**) de variabele **Teller** kleiner is dan of gelijk is aan 1000.

Een ander voorbeeld:

```
Rolls = 0 Counter = 0 While (Counter < 10) ' Roll a simulated die Rolls = Rolls + 1 If (M
```

Deze lus herhaalt zich terwijl de variabele **Counter** minder dan 10 blijft. De variabele **Counter** wordt verhoogd (verhoogd met één) telkens wanneer een gesimuleerde matrijs een 6 rolt (de hier gebruikte functie **GetRandomNumber** retourneert een willekeurige waarde van 1 tot 6). De **rolls** variabele vertelt je hoeveel rollen van de matrijs nodig waren om 10 zessen te rollen. Theoretisch zou het 60 rollen moeten duren, omdat er een kans van 1 op 6 is om een zes te rollen.

Zoals vermeld, als de logische voorwaarde die door een **While-lus** wordt gebruikt ,,false' is wanneer de lus voor het eerst wordt aangetroffen, wordt het codeblok **in de** while-lus niet uitgevoerd. Dit kan acceptabel gedrag zijn - misschien is het dat niet.

We kunnen een lus bouwen die altijd minstens één keer wordt uitgevoerd. Om dit te doen, moeten we de Small Basic **Goto-verklaring** introduceren. Met **een Goto** kunt u de uitvoering van code overbrengen naar elke plaats in uw code. Voor een **Goto** is een **label** vereist. Een label is als een bladwijzer - het kan alles worden genoemd wat u maar wilt. Een labelnaam wordt altijd gevolgd door een dubbele punt. Een voorbeeld is:

MyLabel:

Telkens wanneer we de uitvoering van het programma naar deze labelverklaring willen overbrengen, gebruiken we een **Goto**:

Goto MijnLabel

U schrijft de dubbele punt niet in de **Goto-verklaring**.

Met behulp van deze nieuwe concepten in een lus, hebben we wat we een **Goto-lus** zullen noemen:

```
MyLabel: ' Small Basic code block to process If (Condition) Then Goto MyLabel EndIf
```

Het codeblok herhaalt zich zolang **voorwaarde** "waar" is. In tegenstelling tot de While-lus wordt deze lus altijd minstens één keer uitgevoerd. Ergens in de lus moet **voorwaarde** worden gewijzigd in "onwaar" om het afsluiten mogelijk te maken.

Laten we eens kijken naar voorbeelden van de **Goto-lus**. Wat als we drie willen blijven toevoegen aan een **som** totdat de waarde hoger is dan 50. Deze lus doet het:

```
Sum = 0 SumLoop: Sum = Sum + 3 If (Sum <= 50) Then Goto SumLoop EndIf
```

Nog een dobbelsteen voorbeeld:

```
Sum = 0 Rolls = 0 SumLoop: ' Roll a simulated die Die = Math.GetRandomNumber(6) Sum = Sum
```

Deze lus rolt een gesimuleerde dobbelsteen (**Die**) terwijl de **Som** van de rollen niet hoger is dan 30. Het houdt ook het aantal rollen (**rollen**) bij dat nodig is om dit bedrag te bereiken.

U moet beslissen welke van de lusstructuren (**While**, **Goto**) bij uw programma past. Bedenk dat het grote verschil is dat een **Goto-lus** altijd minstens één keer wordt uitgevoerd; een **While-lus** mag nooit worden uitgevoerd.

En zorg ervoor dat je altijd uit een lus kunt komen. In beide loopingstructuren betekent dit dat op een gegeven moment de controlerende logische voorwaarde "onwaar" moet worden om het verlaten van de lus mogelijk te maken. Wanneer u een **While-lus** afsluit, wordt de verwerking voortgezet bij de volgende Small Basic-instructie na **de EndWhile**. In een **Goto-lus** gaat de verwerking verder bij de instructie Small Basic nadat de **if-structuur** heeft gecontroleerd of de lus moet worden herhaald.

Als u op een bepaald moment in het codeblok van een lus besluit dat u de lus onmiddellijk moet verlaten of naar een ander punt in de code moet gaan, kan een **Goto-instructie** dit ook doen. U hebt alleen een labelverklaring op de juiste plaats nodig. Wanneer de **Goto-instructie** wordt aangetroffen, wordt de verwerking onmiddellijk overgebracht naar de gelabelde instructie.

Kleine basistellingen

Met **While** loop structuren wisten we meestal niet van tevoren hoe vaak we een lus uitvoeren of herhalen. Als u weet hoe vaak u een bepaalde code moet herhalen, wilt u Small **Basic-telling** gebruiken. Tellen is handig voor het toevoegen van items aan een lijst of misschien het optellen van een bekend aantal waarden om een gemiddelde te vinden.

Small Basic counting wordt uitgevoerd met behulp van de For-lus:

```
For Variable = Start To End Step Increment ' Small Basic code to execute goes here EndFor
```

In deze lus is **Variabele** de teller (hoeft niet per se een geheel getal te zijn). De eerste keer via de lus wordt **Variabele** geïnitialiseerd bij **Start**. Telkens wanneer de bijbehorende **EndFor-instructie** wordt bereikt, wordt

variabele verhoogd met een bedrag **increment**. Als de **stapwaarde** wordt weggelaten, wordt een standaardtoenamewaarde van één gebruikt. Negatieve verhogingen zijn ook mogelijk. De telling herhaalt zich totdat **Variabele** gelijk is aan of hoger is dan de eindwaarde **Einde**.

Voorbeeld:

```
For Degrees = 0 To 360 Step 10 'convert to radians R = Degrees * Math.PI / 180 A = Math.S
```

In dit voorbeeld berekenen we goniometrische functies voor hoeken van 0 tot 360 graden in stappen (stappen) van 10 graden.

Een ander voorbeeld:

```
For Countdown = 10 To 0 Step -1 TextWindow.WriteLine(Countdown + " Seconds") EndFor
```

NASA belde en vroeg ons om af te tellen van 10 naar 0. De bovenstaande lus volbrengt de taak.

En, nog een voorbeeld:

```
Sum = 0 For I = 1 to 100 Sum = Sum + MyValues[I] EndFor Average = Sum / 100
```

Deze code vindt de gemiddelde waarde van 100 getallen die zijn opgeslagen in de array **MyValues**. Het somt eerst elk van de waarden op in een Voor-lus. Die som wordt vervolgens gedeeld door het aantal termen (100) om het gemiddelde op te leveren.

U kunt een **For-lus** vroegtijdig afsluiten met een **Goto-instructie**. Hiermee wordt het programmabeheer overgebracht naar de bijbehorende gelabelde instructie, meestal de regel na de **EndFor**.

Kleine Basic Subroutines

In de loopingdiscussie zagen we hoe code in een bepaald blok kon worden herhaald totdat aan een gewenste voorwaarde was voldaan. Vaak hebben we in Small Basic-programma's de behoefte om een bepaald codeblok op verschillende punten in het programma te herhalen. Waarom zou je dit willen doen?

Stel dat we een spel hadden waarbij we 5 dobbelstenen moesten gooien en hun individuele waarden moesten optellen om een som op te leveren. Wat als we dit op 10 verschillende plaatsen in ons programma zouden moeten doen? We konden de code schrijven en vervolgens kopiëren en plakken op de 10 verschillende plaatsen. Ik denk dat je al problemen ziet. Wat als u de code moet wijzigen? Je zou het op 10 verschillende plaatsen moeten veranderen. Wat als u de code op een andere plaats in uw programma moet plaatsen? U zou nog een 'kopieer- en plakbewerking' moeten uitvoeren. Er is een betere manier. En die manier is om een Small Basic **subroutine** te gebruiken.

Met een subroutine kunt u de code schrijven om bepaalde taken slechts één keer uit te voeren. Wanneer u vervolgens toegang moet krijgen tot de code in uw programma, kunt u deze "aanroepen", met alle informatie die het nodig heeft om zijn taken uit te voeren. Subroutines zijn de bouwstenen van een Small Basic programma. Het gebruik van subroutines in uw Small Basic-programma's kan helpen bij het opdelen van een complexe toepassing in beter beheersbare code-eenheden. Denk maar aan een subroutine als een codeblok dat je overal in een Small Basic-programma kunt openen. Wanneer u de subroutine aanroept, gaat het programmabesturingselement naar die subroutine, voert het de toegewezen taken uit en keert het terug naar het aanroepprogramma. Zo eenvoudig is het.

Laten we eens kijken hoe we een subroutine kunnen maken en aanroepen. Subroutines gaan aan het einde van uw '**hoofd**' programmacode. Een subroutine met de naam **MySubroutine** zou het formulier hebben (begint met een **subzoekwoord** en eindigt met **EndSub**):

```
Sub MySubroutine ' Code to be executed in the subroutine EndSub
```

U voert deze subroutine uit of roept deze aan met behulp van:

```
MySubroutine()
```

De haakjes zijn nodig om de computer te vertellen dat u een subroutine uitvoert. Wanneer de subroutine wordt aangeroepen, wordt de bijbehorende code uitgevoerd totdat de **EndSub-regel** is bereikt. Op dit punt keert de uitvoering van het programma terug naar de coderegel direct na de regel die de subroutine heeft aangeroepen.

Een subroutine kan toegang krijgen tot en gebruik maken van elke variabele die u in uw programma gebruikt. Op dezelfde manier kan uw programma alle variabelen gebruiken die in uw subroutines zijn gedefinieerd. In computerpraat zeggen we dat alle variabelen in een Small Basic-programma een **wereldwijde reikwijdte** hebben.

Laten we proberen dit duidelijker te maken door naar een subroutine voorbeeld te kijken. We zullen het dobbelsteenvoorbeeld doen van het gooien van vijf dobbelstenen en het berekenen van hun som. De subroutine die deze taak uitvoert, is:

```
Sub RollDice Die1 = Math.GetRandomNumber(6) Die2 = Math.GetRandomNumber(6) Die3 = Math.Ge
```

Deze subroutine heet **RollDice** en de variabele **SumDice** heeft de resulterende som.

Met behulp van deze subroutine, elke keer dat je de som van vijf dobbelstenen in je programma nodig hebt, zou je gebruiken:

```
RollDice() A = SumDice
```

Nadat deze code is uitgevoerd, heeft de variabele **A** een som van vijf dobbelstenen.

Naarmate je vordert in je Small Basic programmeeropleiding, zul je je meer op je gemak voelen met het gebruik van subroutines en zien hoe nuttig ze zijn. In de rest van deze cursus zullen we subroutines gebruiken voor een groot deel van de code. Bestudeer elk voorbeeld om te leren hoe u subroutines kunt bouwen en gebruiken.

Kleine basisobjecten

Objecten worden door de taal Small Basic gebruikt om programma's te bouwen. Een object kan **eigenschappen**, **methoden** en/of **gebeurtenissen** hebben.

Een **eigenschap** beschrijft iets over het object. In hoofdstuk 1 hadden we deze eenvoudige coderegel:

```
TextWindow.Title = "Hallo Programma"
```

Hier is **TextWindow** het object, **Title** is de eigenschap (de informatie die in de titelbalk van het venster verschijnt) en "**Hello Program**" is de waarde van de eigenschap. Dus, in het algemeen, om de **eigenschap** van een **object** in te stellen, gebruikt u deze "puntnotatie":

```
Object.Property = Waarde
```

Een **methode** doet iets met een object. Nogmaals, in hoofdstuk 1 hadden we deze regel:

```
TextWindow.WriteLine("Dit is de eerste regel van het programma.")
```

Hier is **WriteLine** de methode (het schrijft informatie in het tekstvenster) en het item tussen haakjes wordt het methodeargument genoemd. Het argument (of soms argumenten) geeft informatie die de methode nodig heeft om zijn werk te doen. Als u een **methode** wilt toepassen op een **object**, gebruikt u:

```
Object.Methode(Argumenten)
```

Soms kan de methode een waarde berekenen en retourneren. In dit geval wordt die **waarde** gevonden met behulp van:

```
Waarde = Object.Methode(Argumenten)
```

Ten slotte is een **gebeurtenis** iets dat met een object gebeurt. Voorbeelden van gebeurtenissen zijn het klikken op een knopbesturingselement, het indrukken van een muisknop of het indrukken van een toets op het toetsenbord. Als u op een gebeurtenis wilt reageren, moet aan de gebeurtenis een **subroutine** worden

toegewezen die wordt aangeroepen als de gebeurtenis plaatsvindt. Als u de subroutine **EventSub** wilt toewijzen aan de **gebeurtenis** voor **object**, gebruikt u:

```
Object.Event = EventSub
```

In de volgende hoofdstukken kijken we naar objecten en hoe ze te gebruiken. Voor elk object bekijken we de eigenschappen, methoden en gebeurtenissen (indien aanwezig).

Hoofdstuk Review

Na het voltooien van dit hoofdstuk moet u begrijpen:

- Hoe variabelen op de juiste manier te gebruiken.
- Kleine basisverklaringen.
- De toewijzingsoperator, wiskundige operatoren, vergelijkings- en logicaoperators en aaneenschakelingsoperatoren.
- De grote verscheidenheid aan ingebouwde Small Basic-methoden, met name stringmethoden en wiskundige methoden.
- Grafische methoden gebruiken om in het grafische venster te tekenen.
- De **if/then/elseif/else/endif** structuur die wordt gebruikt voor vertakkingen en beslissingen.
- Hoe de **while-lus** en **goto-lus** werken.
- Hoe de **for-lus** wordt gebruikt voor het tellen.
- Het belang van subroutines in Small Basic-programma's.
- Eigenschappen, methoden en gebeurtenissen, zoals gerelateerd aan kleine basisobjecten.

We hebben nu de basis die nodig is om te beginnen met het schrijven van een aantal programma's. Om dit te doen, zullen we veel van de kleine basisobjecten behandelen, te beginnen met het **object Program**.

Volgende hoofdstuk > >