

Microsoft Small Basic

Eine Einführung ins Programmieren

Small Basic und Programmieren

Das Programmieren von Computern ist definiert als der Prozess der Erstellung von Computerprogrammen mithilfe von Programmiersprachen. Genau wie wir Englisch, Spanisch oder Französisch sprechen und verstehen, können Computer Programme verstehen, die in bestimmten Sprachen geschrieben sind. Diese werden Programmiersprachen genannt. Am Anfang gab es nur ganz wenige Programmiersprachen und sie waren wirklich leicht zu lernen und zu verstehen. Als aber Computer und Software immer leistungsfähiger wurden, entwickelten sich die Programmiersprachen schnell, wobei sie im Laufe der Zeit immer kompliziertere Konzepte aufnahmen. Als Ergebnis ist das Erfassen der meisten modernen Programmiersprachen und ihrer Konzepte für einen Anfänger eine ganz schöne Herausforderung. Diese Tatsache hat den Leuten den Mut genommen, Programmiersprachen zu lernen oder auszuprobieren.

Die Programmiersprache Small Basic erleichtert Anfängern den Einstieg. Softwareentwicklung mit Small Basic soll Spaß bringen und die wunderbare Welt der Programmierung zugänglicher machen.

Die Entwicklungsumgebung von Small Basic

Lassen Sie uns mit einer kurzen Einführung in die Entwicklungsumgebung von Small Basic anfangen. Wenn das Programm gestartet wird, sieht man folgende Oberfläche:

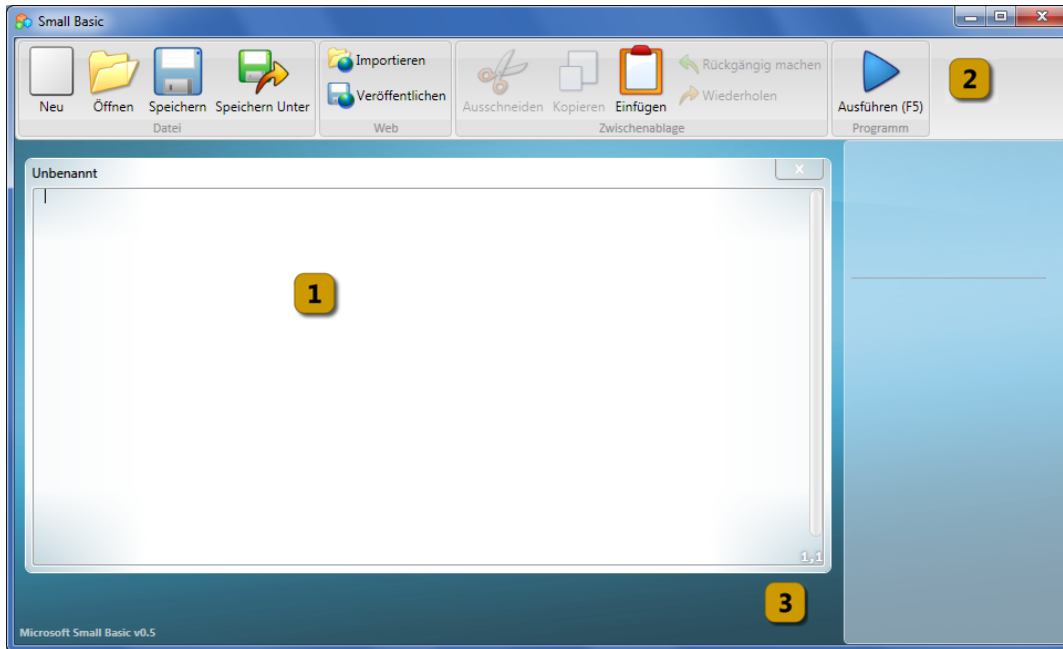


Abbildung 1 - Die Small Basic-Entwicklungsumgebung

Dies ist die Small Basic-Entwicklungsumgebung, in der wir unsere Small Basic-Programme schreiben und ausführen werden. Diese Umgebung hat verschiedene Bereiche, die durch Nummern gekennzeichnet sind.

Mit dem **Editor**, markiert mit [1], werden wir unsere Small Basic Programme schreiben. Wenn Sie ein Beispielprogramm oder ein zuvor gespeichertes Programm öffnen, wird es in diesem Editor angezeigt. Sie können es dann ändern und für späteren Gebrauch speichern. Sie können auch mit mehr als einem Programm zurzeit arbeiten. Jedes Programm, mit dem Sie arbeiten, wird in einem eigenen Editor angezeigt. Der Editor, der das Programm enthält, an dem Sie gerade arbeiten, wird der *aktive Editor* genannt.

Über die **Symbolleiste**, erkennbar an der [2], können Befehle entweder für den aktiven Editor oder die Entwicklungsumgebung gegeben werden. Im Weiteren werden wir die verschiedenen Befehle in der Symbolleiste kennenlernen. Auf der **Oberfläche**, gekennzeichnet durch [3], erscheinen alle Editorfenster.

Unser erstes Programm

Nun, wo Sie ein wenig vertraut mit der Small Basic Entwicklungsumgebung sind, wollen wir weitermachen und anfangen, in ihr zu programmieren. Wie wir gerade gesagt haben, ist der Editor der Bereich, in dem wir unsere Programme schreiben. Also lassen Sie uns vorangehen und tippen Sie die folgende Zeile in den Editor.

```
TextWindow.WriteLine("Hello World")
```

Dies ist unser erstes Small Basic Programm. Und wenn Sie es korrekt eingegeben haben, dann sollte jetzt folgendes im Editor abgebildet sein:

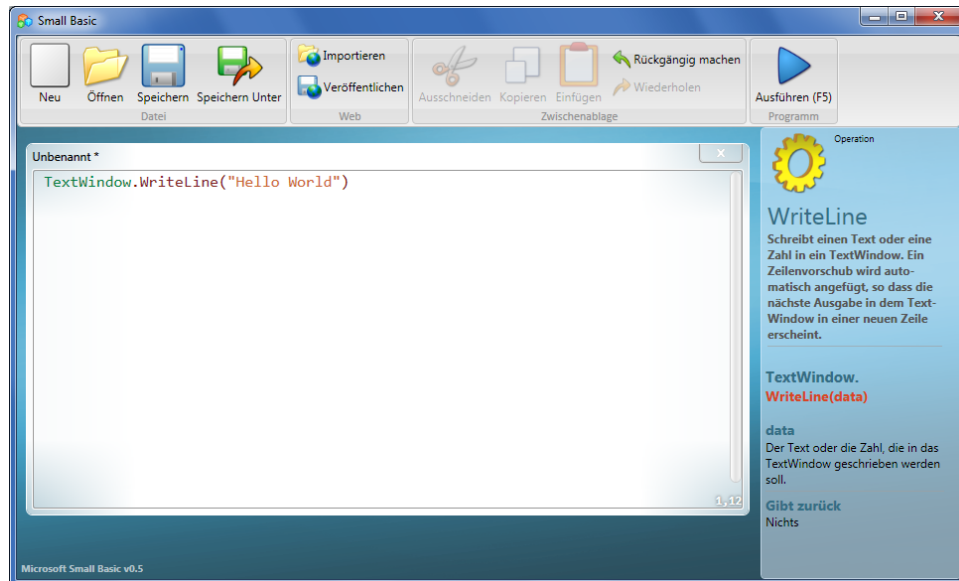


Abbildung 2 - Erstes Programm

Jetzt führen wir das Programm aus, indem wir die Schaltfläche **Ausführen** auf der Symbolleiste klicken oder die F5-Taste auf der Tastatur drücken. Wenn alles seine Richtigkeit hat, wird folgendes Resultat angezeigt:

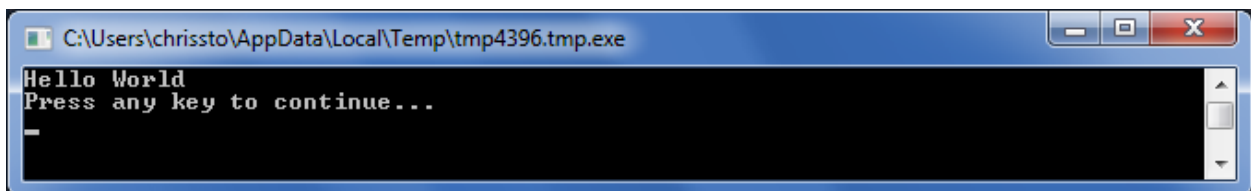


Abbildung 3 – Ausgabe des ersten Programmes

Glückwünsche! Sie haben gerade das erste Small Basic Programm geschrieben und zum Laufen gebracht. Ein sehr kleines und einfaches Programm, aber nichtsdestotrotz ein großer Schritt auf dem Weg, ein richtiger Programmierer zu werden! Nun ist nur noch eine Einzelheit zu behandeln, bevor wir uns daran machen können, größere Programme zu schreiben. Wir müssen verstehen, was gerade passiert ist – was genau haben wir dem Computer gesagt, und warum wusste der Computer, was er tun sollte? Im nächsten Kapitel werden wir das Programm

Beim Schreiben im Editor wird ein kleines Fenster mit Hilfetext angezeigt (Abbildung 4). Diese Funktionalität heißt „IntelliSense“ und bietet viele hilfreiche Tipps. Der verfügbare Hilfetext lässt sich mit den NACH OBEN- und NACH UNTEN-Tasten auf der Tastatur navigieren. Wenn man einen nützlichen Eintrag gefunden hat, lässt sich dieser durch Drücken der EINGABE-Taste ins Programm einfügen.

analysieren, das wir gerade geschrieben haben, so dass wir es verstehen.

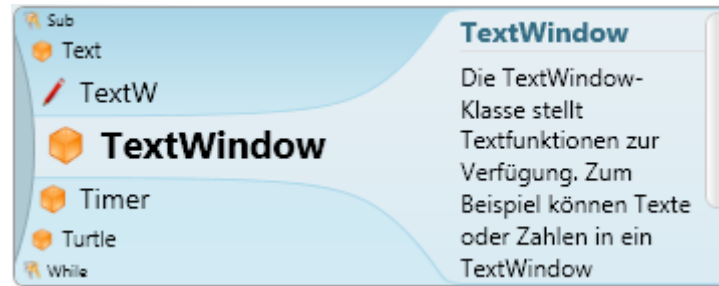


Abbildung 4 - Intellisense

Speichern unseres Programmes

Man sollte sein Programm speichern, wenn man Small Basic schließen und später weiter arbeiten möchte. Es ist sowieso empfehlenswert, während der Arbeit ab und zu einmal zu speichern, damit man keine Informationen verliert, wenn zum Beispiel plötzlich der Strom ausfällt oder der Computer einen ungeplanten Neustart ausführt. Zum Speichern des Programms klickt man lediglich auf die **Speichern**-Schaltfläche auf der Symbolleiste. Die Tastenkombination STRG+S erfüllt den gleichen Zweck. Man hält dabei die STRG-Taste gedrückt, während man die S-Taste drückt.

Unser erstes Programm verstehen

Was ist ein Computerprogramm wirklich?

Ein Programm ist eine Menge von Befehlen für den Computer. Diese Befehle sagen dem Computer ganz genau, was zu tun ist; und der Computer befolgt diese Anweisungen immer. Genau wie bei Menschen können Computer nur Befehle befolgen, die in einer Sprache gegeben werden, die sie verstehen. Solche Sprachen werden Programmiersprachen genannt. Es gibt sehr viele Sprachen, die der Computer verstehen kann und **Small Basic** ist eine davon.

Stellen Sie sich eine Unterhaltung zwischen Ihnen und Ihrem Freund vor. Sie und Ihre Freunde würden Wörter benutzen, die zu Sätzen zusammengefasst werden, um Informationen hin und her zu übermitteln. Entsprechend enthalten Programmiersprachen eine Anzahl von Wörtern, die zu Sätzen zusammengefasst werden können, die dem Computer Informationen übermitteln. Im Grunde sind Programme Mengen von Sätzen (manchmal nur wenige und manchmal viele tausend), die zusammen sowohl für den Programmierer als auch für den Computer Sinn ergeben.

Es gibt viele Sprachen, die der Computer verstehen kann. Java, C++, Python, VB, etc. sind leistungsfähige moderne Computersprachen, die verwendet werden, um einfache und auch komplexe Computerprogramme zu entwickeln.

Small Basic-Programme

Ein typisches Small Basic-Programm besteht aus einem Bündel von *Anweisungen*. Jede Programmzeile repräsentiert eine Anweisung und jede Anweisung ist ein Befehl an den Computer. Wenn wir den Computer auffordern, ein Small Basic Programm auszuführen, nimmt er das Programm und liest die erste Anweisung. Er versteht, was wir ihm zu sagen versuchen und führt dann unsere Anweisung aus. Sobald die erste Anweisung ausgeführt ist, kommt er zum Programm zurück und führt die zweite Zeile

aus. Damit macht er weiter, bis er das Ende des Programmes erreicht. Das ist der Moment, in dem unser Programm endet.

Zurück zu unserem ersten Programm

Hier ist das erste Programm, das wir geschrieben haben:

```
TextWindow.WriteLine("Hello World")
```

Unser Programm besteht aus einer einzigen Anweisung. Diese Anweisung teilt dem Computer mit, die Textzeile **Hello World** in ein Textfenster zu schreiben. Die beiden Elemente der Anweisung **TextWindow** und **WriteLine** sind Bestandteile der Small Basic-Programmiersprache. Im Deutschen wäre der Ausdruck für **TextWindow** zum Beispiel **TextFenster**, und für **WriteLine** könnte man **SchreibZeile** einsetzen. Aber da die Programmiersprache von Small Basic in Englisch verfasst ist, verwenden wir das englische Vokabular, das Small Basic versteht. Die meisten gängigen Programmiersprachen sind heutzutage in Englisch verfasst, also empfiehlt es sich, parallel zum Experimentieren mit den ersten Softwareprogrammierschritten auch sein Englisch etwas aufzufrischen. Ein gutes Wörterbuch online ist dabei unentbehrlich! In unserer Sprache ausgedrückt, versteht der Computer folgendes:

```
Schreib Hello World
```

Möglicherweise haben Sie bereits bemerkt, dass die Anweisung Stück für Stück in kleinere Teile zerlegt werden kann, und zwar so, wie ein Satz in Wörter zerlegt werden kann. In der ersten Anweisung haben wir drei unterschiedliche Teile:

- a) TextWindow
- b) WriteLine
- c) „Hello World“

Der Punkt, die Klammern und die Anführungszeichen sind alles Satzzeichen, die an die richtige Stelle in der Anweisung gesetzt werden müssen, damit der Computer versteht, was wir wollen. Vielleicht erinnern Sie sich an das schwarze Fenster, das erschien, als wir unser erstes Programm laufen ließen. Dieses Fenster wird TextWindow (Textfenster) genannt oder manchmal bezieht man sich darauf als Konsole. Hier erscheinen die Ergebnisse dieses Programmes. Das **TextWindow** in unserem Programm wird *Objekt* genannt. Es ist eine ganze Menge von Objekten verfügbar, die wir in unseren Programmen benutzen können. Wir können verschiedene *Methoden* auf diese Objekte anwenden. Die **WriteLine**-Methode haben wir in unserem Programm bereits verwendet. Eventuell ist Ihnen aufgefallen, dass auf WriteLine

Satzzeichen wie Anführungszeichen, Leerzeichen und Klammern sind in einem Computerprogramm sehr wichtig. In Abhängigkeit von ihrer Position und Anzahl können sie die Bedeutung dessen, was ausgedrückt ist, ändern.

Hello World in Anführungszeichen folgt. Der Text wird als Eingabe an die WriteLine-Methode übergeben, die es dann für den Benutzer ausdrückt. Er wird die **Eingabe** (auf Englisch *input*) der Methode genannt. Einige Methoden erwarten einen oder mehrere Eingabewerte, während andere überhaupt keinen erwarten.

Unser zweites Programm

Nachdem Sie jetzt unser erstes Programm verstanden haben, wollen wir weitermachen und es ansprechender gestalten, indem wir einige Farben hinzufügen.

```
TextWindow.ForegroundColor = "Yellow"  
TextWindow.WriteLine("Hello World")
```

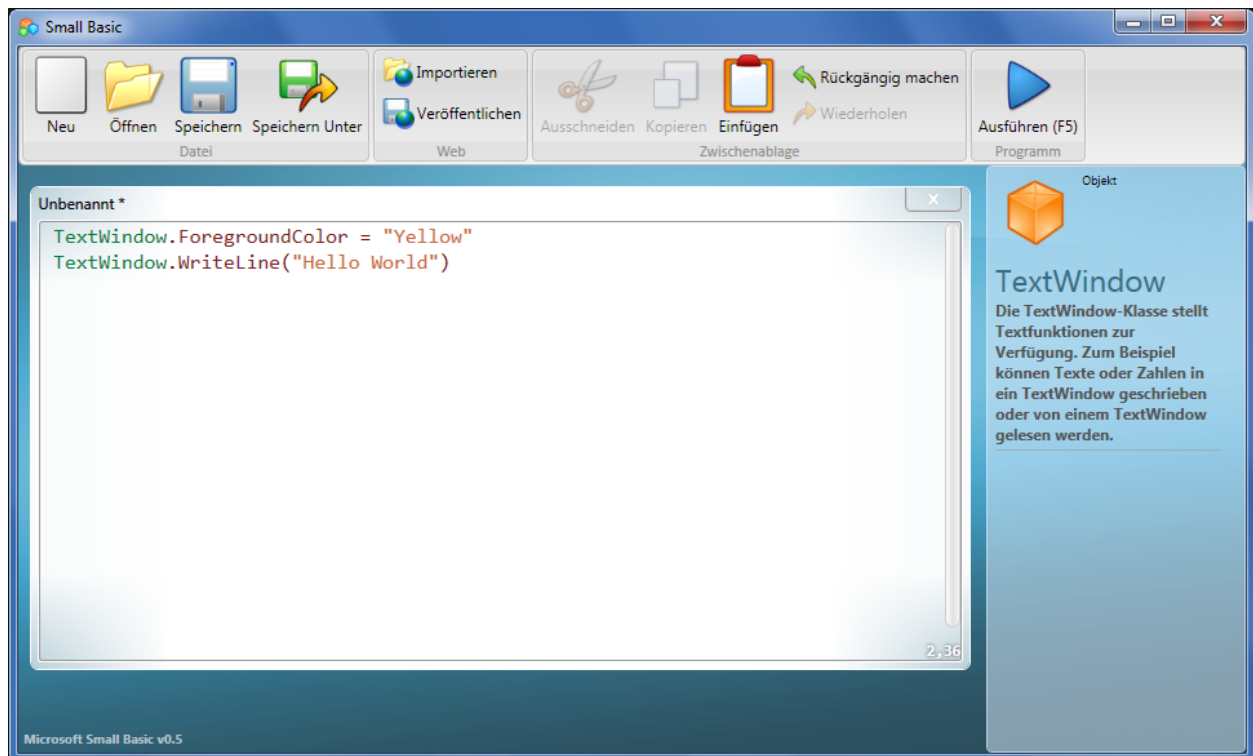


Abbildung 5 – Farben hinzufügen

Wenn Sie das obenstehende Programm laufen lassen, werden Sie feststellen, dass es den gleichen Ausdruck „Hello World“ ausgibt, aber dieses Mal schreibt es ihn im Textfenster in Gelb und nicht in grau wie zuvor.

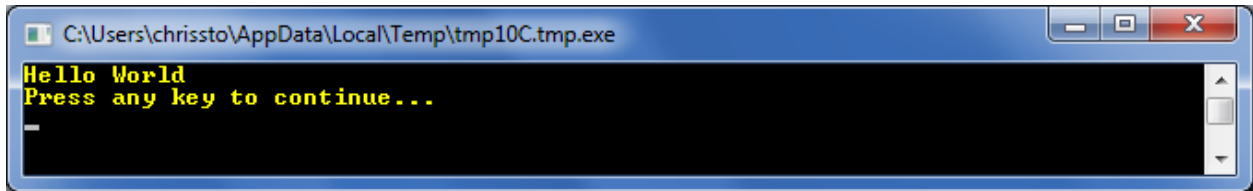


Abbildung 6 - Hello World in gelb

Schauen Sie sich die neue Anweisung an, die wir unserem Programm hinzugefügt haben. Sie benutzt ein neues Wort **ForegroundColor** (Vordergrundfarbe), dem wir den Wert „Yellow“ (Gelb) gegeben haben. Das bedeutet, wir haben **ForegroundColor** den Wert „Yellow“ zugewiesen. Der Unterschied zwischen **ForegroundColor** und der Methode **WriteLine** ist, dass **ForegroundColor** keine Eingabe erwartet und auch keine Klammern benötigt, stattdessen folgen ihm ein Gleichheitszeichen und ein Wort. Wir bezeichnen **ForegroundColor** als Eigenschaft von **TextWindow**. Hier ist eine Liste von zulässigen Werten für die **ForegroundColor**-Eigenschaft. Versuchen Sie, „Yellow“ durch einen Wert aus dieser Liste zu ersetzen und sehen Sie sich das Ergebnis an. Vergessen Sie die Anführungszeichen nicht, es sind erforderliche Satzzeichen.

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

Einführung von Variablen

Die Verwendung von Variablen in unserem Programm

Wäre es nicht schön, wenn unser Programm „Hello“ gefolgt vom Namen des Programmbenutzers anzeigen könnte, statt einfach nur „Hello World“? Um das zu erreichen, müssen wir zuerst den Benutzer nach seinem Namen fragen, diesen irgendwo speichern und dann „Hello“ mit dem Namen des Benutzers ausgeben. Gehen wir die Schritte einmal durch:

```
TextWindow.Write("Enter your Name: ")  
name = TextWindow.Read()  
TextWindow.WriteLine("Hello " + name)
```

Wenn Sie das Programm eingeben und ausführen, werden Sie eine Ausgabe wie die folgende sehen:

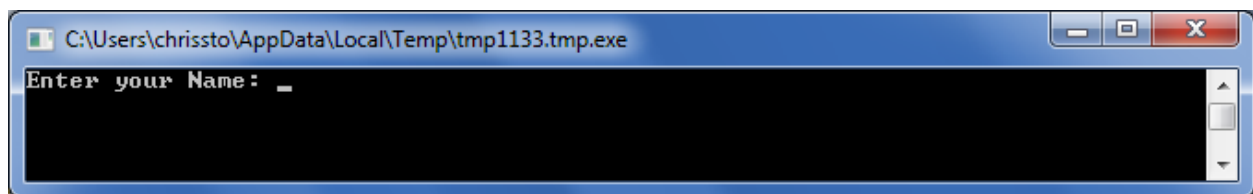


Abbildung 7 – Frage nach dem Namen des Benutzers

Und wenn Sie Ihren Namen eingeben und die EINGABETASTE drücken, werden Sie eine ähnliche Ausgabe wie diese sehen:

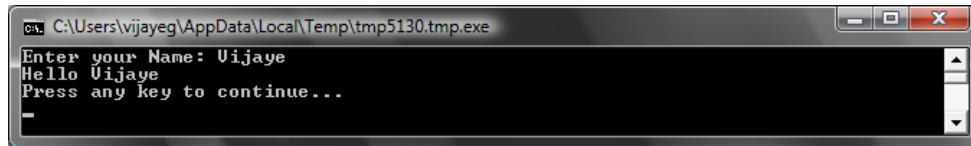


Abbildung 8 – Ein freundlicher Gruß

Wenn Sie das Programm nochmals laufen lassen, wird Ihnen die gleiche Frage wieder gestellt. Sie können einen anderen Namen eingeben und der Computer wird Hello mit diesem Namen sagen.

Analyse des Programmes

Im gerade abgelaufenen Programm mag diese Zeile Ihre Aufmerksamkeit erregt haben:

```
name = TextWindow.Read()
```

Read() sieht beinahe wie **WriteLine()** aus, aber ohne Eingaben. Es ist eine Methode teilt dem Computer lediglich mit, darauf zu warten, dass der Benutzer etwas eingibt und die EINGABETASTE drückt. Wenn der Benutzer die EINGABETASTE drückt, liefert diese Methode die Benutzereingabe an das Programm zurück. Das interessante hier ist, dass die Benutzereingabe in einer Variablen mit dem Namen **name** gespeichert wird. Eine *Variable* ist definiert als ein Ort, an dem Sie Werte zeitweise speichern und später nutzen können. In der obenstehenden Zeile ist **name** benutzt worden, um den Namen des Benutzers zu speichern.

Variablen werden vom Programmierer definiert, daher sind wir nicht an die englische Terminologie gebunden, die Small Basic kennt. Für Variablen können also auch beliebige deutsche Namen verwendet werden.

Write ist genauso wie WriteLine eine Methode von ConsoleWindow. Write erlaubt es Ihnen, etwas ins Konsolenfenster zu schreiben, wobei nachfolgender Text in der gleichen Zeile wie der ausgegebene erscheint.

Die nächste Zeile ist ebenfalls interessant:

```
TextWindow.WriteLine("Hello " + name)
```

Hier benutzen wir den in unserer Variablen **name** gespeicherten Wert. Wir nehmen den Wert in **name**, hängen ihn an "Hello " an und schreiben ihn ins Textfenster. Sobald eine Variable einen Wert erhalten hat, können Sie sie beliebig oft wieder benutzen. Zum Beispiel können Sie folgendes machen:

```
TextWindow.Write("Enter your Name: ")  
name = TextWindow.Read()  
TextWindow.Write("Hello " + name + ". ")
```

```
TextWindow.WriteLine("How are you doing " + name + "?")
```

Und Sie werden folgende Ausgabe sehen:

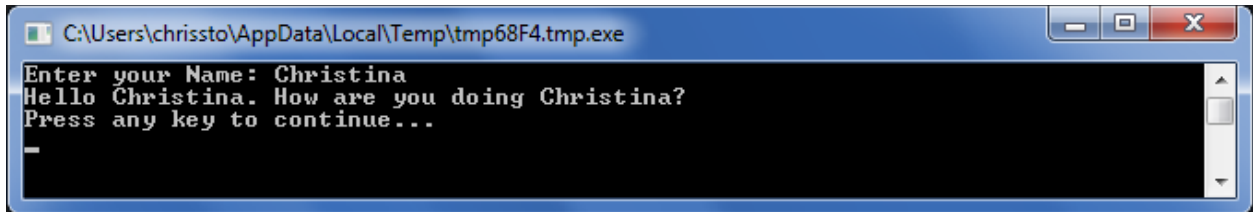


Abbildung 9 – Wiederbenutzung einer Variablen

Spielen mit Zahlen

Wir haben gerade gesehen, wie Sie eine Variable benutzen können, um den Namen des Benutzers zu speichern. In den nächsten paar Programmen werden wir sehen, wie Sie Variablen nutzen können, um Zahlen darin zu speichern und zu verändern. Lassen Sie uns mit einem wirklich einfachen Programm anfangen:

```
number1 = 10  
number2 = 20  
number3 = number1 + number2  
TextWindow.WriteLine(number3)
```

Wenn Sie das Programm laufen lassen, werden Sie folgende Ausgabe erhalten:

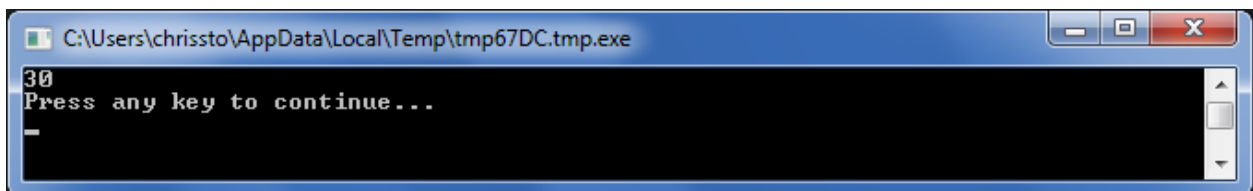


Abbildung 10 – Zwei Zahlen addieren

In der ersten Zeile des Programmes legen Sie für die Variable **number1** den Wert 10 fest. Und in der zweiten Zeile weisen Sie der Variablen **number2** den Wert 20 zu. In der dritten Zeile addieren Sie **number1** und **number2** und weisen das Ergebnis **number3** zu. Somit wird in diesem Fall **number3** einen Wert von 30 haben. Und das ist das, was wir im Textfenster ausgegeben haben.

Beachten Sie, dass die Zahlen keine Anführungszeichen um sich herum haben. Für Zahlen sind keine Anführungszeichen erforderlich. Sie brauchen Anführungszeichen nur dann, wenn Sie Text benutzen.

Nun lassen Sie uns das Programm ein wenig abändern und die Resultate ansehen.

```
number1 = 10  
number2 = 20  
number3 = number1 * number2  
TextWindow.WriteLine(number3)
```

Das obenstehende Programm wird number1 mit number2 multiplizieren und das Ergebnis in number3 speichern. Das Ergebnis des Programmes können Sie hier sehen:

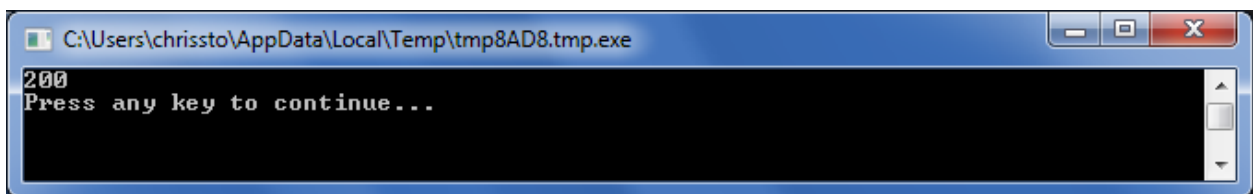


Abbildung 11 – Zwei Zahlen multiplizieren

Entsprechend können Sie Zahlen subtrahieren oder dividieren. Hier die Subtraktion:

```
number3 = number1 - number2
```

Und das Zeichen für Division ist '/'. Das Programm sieht dann so aus:

```
number3 = number1 / number2
```

Und das Ergebnis dieser Division wäre:

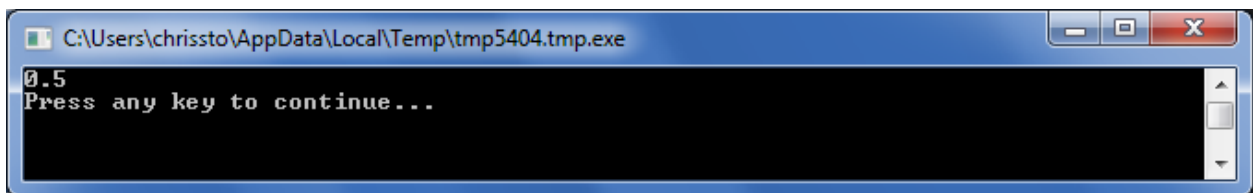


Abbildung 12 – Zwei Zahlen dividieren

Ein einfacher Temperaturumrechner

Für das nächste Programm benutzen wir die Formel $^{\circ}\text{C} = \frac{5(^{\circ}\text{F}-32)}{9}$ um Temperaturen von Fahrenheit in Celsius umzurechnen.

Zuerst holen wir uns die Temperatur in Fahrenheit und speichern sie in einer Variablen. Es gibt eine spezielle Methode, mit der wir Zahlen von einem Benutzer einlesen können und das ist **TextWindow.ReadNumber**.

```
TextWindow.Write("Enter temperature in Fahrenheit: ")  
fahr = TextWindow.ReadNumber()
```

Wenn wir erst einmal die Temperatur in Fahrenheit in einer Variablen gespeichert haben, können wir sie so in Celsius umrechnen:

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

Die Klammern sagen dem Computer, den **fahr - 32** Teil zuerst zu berechnen und dann den Rest auszuwerten. Nun müssen wir dem Benutzer nur noch das Ergebnis anzeigen:

```
TextWindow.Write("Enter temperature in Fahrenheit: ")  
fahr = TextWindow.ReadNumber()  
celsius = 5 * (fahr - 32) / 9  
TextWindow.WriteLine("Temperature in Celsius is " + celsius)
```

Und das Ergebnis des Programmes wäre:

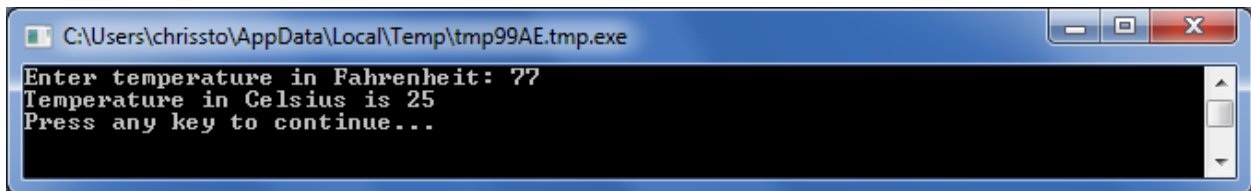


Abbildung 13 - Temperaturumrechnung

Bedingungen und Verzweigungen

Kommen wir auf unser erstes Programm zurück. Wäre es nicht cool, wenn wir anstelle ganz generell *Hello World* zu sagen, je nach Tageszeit *Good Morning World* oder *Good Evening World* anzeigen könnten? Mit unserem nächsten Programm lassen wir den Computer *Good Morning World* sagen, wenn es früher als 12 Uhr mittags und *Good Evening World*, wenn es später als 12 Uhr mittags ist.

```
If (Clock.Hour < 12) Then
    TextWindow.WriteLine("Good Morning World")
EndIf
If (Clock.Hour >= 12) Then
    TextWindow.WriteLine("Good Evening World")
EndIf
```

Je nachdem, wann Sie das Programm laufen lassen, werden Sie eine der folgenden Ausgaben sehen.

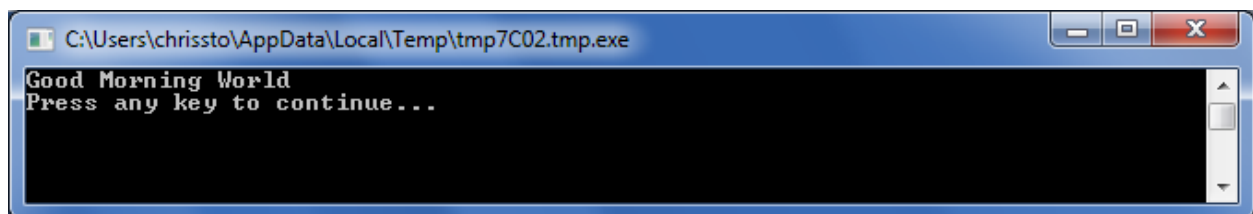


Abbildung 14 - Good Morning World

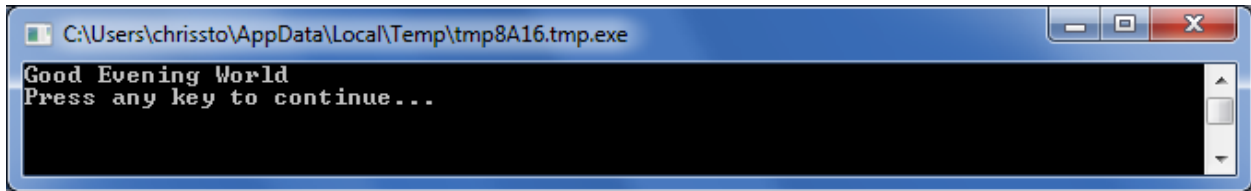


Abbildung 15 - Good Evening World

Lassen Sie uns die ersten drei Zeilen des Programmes analysieren. Sie sollten schon herausbekommen haben, dass diese Zeilen dem Computer sagen, dass er „Good Morning World“ ausgeben soll, wenn Clock.Hour (Uhr.Stunde) kleiner als 12 ist. Die Wörter **If**, **Then** und **EndIf** sind Spezialwörter, die beim Programmablauf vom Computer verstanden werden. Auf das Wort **If** folgt immer eine Bedingung, die in diesem Fall (**Clock.Hour < 12**) **lautet**.
Erinnern Sie sich daran, dass die Klammern erforderlich sind, damit der Computer versteht, was Sie wollen. Auf die Bedingung folgt **then** und die aktuelle Anweisung, die ausgeführt werden soll. Und nach der Anweisung kommt **EndIf**. Dies sagt dem Computer, dass der Bereich der bedingten Programmausführung vorüber ist.

In Small Basic können Sie das Clock (Uhr) Objekt nutzen, um auf das gegenwärtige Datum und die Zeit zuzugreifen. Es sieht ebenfalls ein ganzes Bündel von Eigenschaften vor, mit denen Sie getrennt auf Tag, Monat, Jahr, Stunde, Minuten und Sekunden zugreifen können

Zwischen dem **then** und dem **EndIf** könnte mehr als eine Anweisung stehen und der Computer wird sie alle ausführen, wenn die Bedingung wahr ist. Beispielsweise könnten Sie so etwas schreiben:

```
If (Clock.Hour < 12) Then
    TextWindow.Write("Good Morning. ")
    TextWindow.WriteLine("How was breakfast?")
EndIf
```

Else

Haben Sie am Anfang des Kapitels bemerkt, dass die zweite Bedingung überflüssig ist? Der Wert von **Clock.Hour** kann entweder kleiner als 12 sein oder nicht. Wirklich durchführen müssen wir die zweite Prüfung nicht. In solchen Situationen können wir die zwei **if..then..endif**-Anweisungen auf eine einzige Anweisung verkürzen, indem wir einfach ein neues Wort verwenden, nämlich **else**.

Wenn wir das Programm neu schreiben und **else** verwenden, sieht es so aus:

```
If (Clock.Hour < 12) Then
    TextWindow.WriteLine("Good Morning World")
Else
    TextWindow.WriteLine("Good Evening World")
```


EndIf

Und dieses Programm wird genau das gleiche tun wie das andere, was uns zu einer sehr wichtigen Aussage in der Computerprogrammierung bringt:

“ Bei der Programmierung gibt es viele Wege, um dasselbe Resultat zu erzielen. Manchmal ist ein Weg sinnvoller als ein anderer. Die Wahl ist dem Programmierer überlassen. Wenn Sie mehr Programme schreiben und erfahrener werden, werden Sie diese unterschiedlichen Techniken mit ihren Vor- und Nachteilen kennenlernen.

Texteinzüge

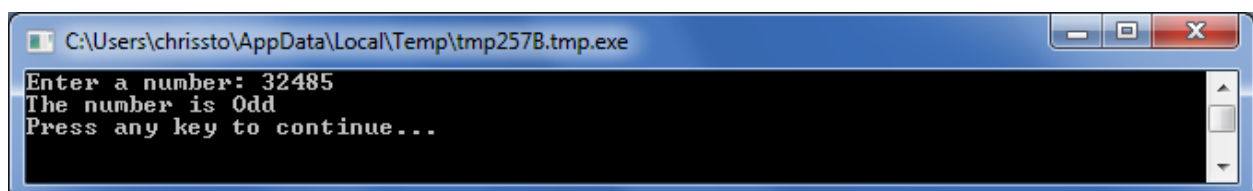
In allen Beispielen können Sie sehen, wie die Anweisungen zwischen *If*, *Else* und *EndIf* eingerückt sind. Der Texteinzug ist nicht erforderlich. Der Computer versteht das Programm genauso gut ohne sie. Uns jedoch hilft er, die Struktur des Programmes leichter zu erkennen und zu verstehen. Deshalb empfiehlt es sich, Anweisungen zwischen solchen Blöcken einzurücken.

Gerade oder ungerade

Nun, wo wir die **If..Then..Else..EndIf**-Anweisung in unserer Trickkiste haben, lassen Sie uns ein Programm schreiben, das sagt, ob eine eingegebene Zahl gerade oder ungerade ist.

```
TextWindow.Write("Enter a number: ")
num = TextWindow.ReadNumber()
remainder = Math.Remainder(num, 2)
If (remainder = 0) Then
    TextWindow.WriteLine("The number is Even")
Else
    TextWindow.WriteLine("The number is Odd")
EndIf
```

Wenn Sie dieses Programm laufen lassen, sehen Sie folgende Ausgabe:



```
C:\Users\chrissto\AppData\Local\Temp\tmp257B.tmp.exe
Enter a number: 32485
The number is Odd
Press any key to continue...
```

Abbildung 16 – Gerade oder ungerade

Mit diesem Programm haben wir eine weitere neue und nützliche Anweisung eingeführt: **Math.Remainder**. Und ja, wie Sie wohl schon herausgefunden haben, **Math.Remainder** liefert den Rest der Division der ersten Zahl durch die zweite Zahl.

Verzweigungen

Erinnern Sie sich daran, dass Sie im zweiten Kapitel gelernt haben, dass der Computer ein Programm so verarbeitet, dass jeweils eine Anweisung zurzeit ausgeführt wird, und zwar von oben nach unten. Es gibt jedoch eine spezielle Anweisung, die den Computer veranlasst, außerhalb der Reihe zu einer anderen Anweisung zu springen. Das schauen wir uns jetzt an.

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

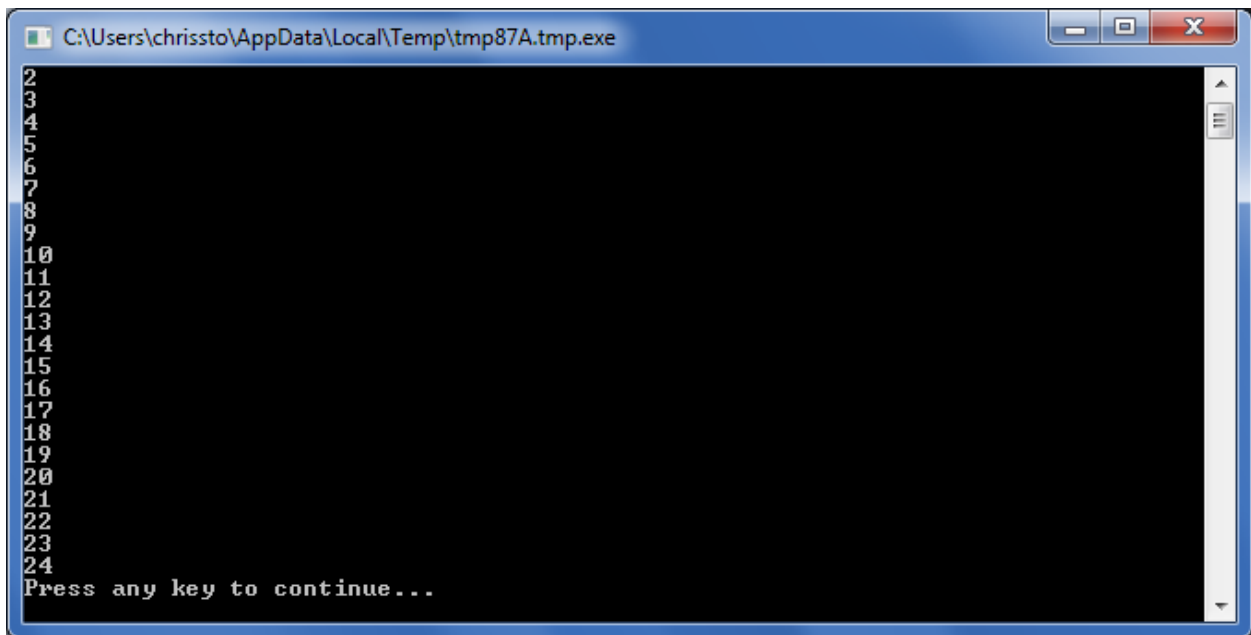


Abbildung 17 – Die Verwendung von Goto

Im obenstehenden Programm haben wir der Variablen *i* einen Wert von 1 zugewiesen. Und dann haben wir eine neue Anweisung hinzugefügt, die mit einem Doppelpunkt (:) endet.

```
start:
```

Das wird *Sprungmarke* genannt. Sprungmarken sind wie Lesezeichen, die der Computer versteht. Sie können der Sprungmarke einen beliebigen Namen geben und Sie können Ihrem Programm so viele Sprungmarken hinzufügen wie Sie wollen, solange sie alle unterschiedlich benannt sind.

Eine andere interessante Anweisung ist diese:

```
i = i + 1
```

Sie sagt dem Computer einfach, 1 zum Wert der Variablen *i* zu addieren und das Ergebnis wieder *i* zuzuweisen. Wenn der Wert von *i* vorher 1 war, wird er nachdem die Anweisung ausgeführt worden ist, 2 sein.

Und schließlich:

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

Das ist der Teil, der dem Computer sagt, das er Anweisungen ab der Sprungmarke **start** ausführen soll, wenn der Wert von *i* kleiner als 25 ist.

Endlose Ausführung

Mit der Verwendung der **Goto**-Anweisung können Sie den Computer etwas beliebig oft ausführen lassen. Zum Beispiel können Sie das Programm „Gerade oder ungerade“ nehmen und wie unten verändern; das Programm wird endlos laufen. Sie können das Programm beenden, indem Sie auf die Schließen-Schaltfläche (X) in der oberen rechten Ecke des Fensters klicken.

```
begin:  
TextWindow.Write("Enter a number: ")  
num = TextWindow.ReadNumber()  
remainder = Math.Remainder(num, 2)  
If (remainder = 0) Then  
    TextWindow.WriteLine("The number is Even")  
Else  
    TextWindow.WriteLine("The number is Odd")  
EndIf  
Goto begin
```

```
C:\Users\chrissto\AppData\Local\Temp\tmpD873.tmp.exe
Enter a number: 456
The number is Even
Enter a number: 2222
The number is Even
Enter a number: -34
The number is Even
Enter a number: -859
The number is Odd
Enter a number: 3302090
The number is Even
Enter a number: _
```

Abbildung 18 – Gerade oder ungerade mit endloser Ausführung

Die For-Schleife

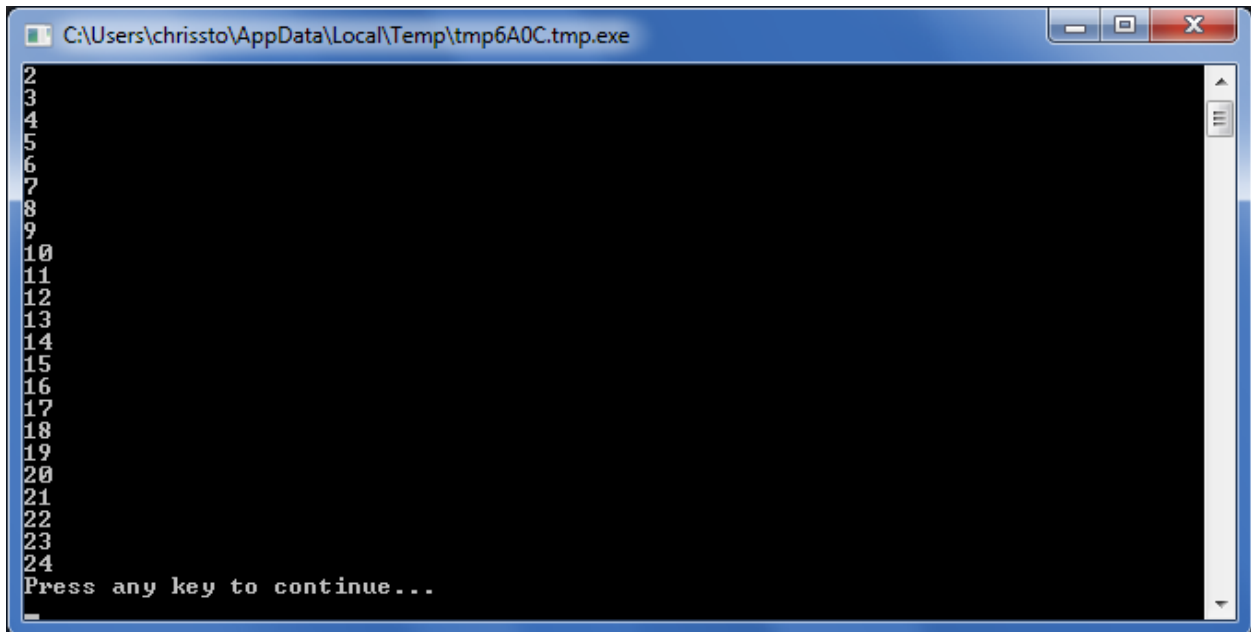
Lassen Sie uns ein Programm nehmen, das wir im vorherigen Kapitel geschrieben haben.

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

Dieses Programm gibt nacheinander die Zahlen von 1 bis 24 aus. Das Erhöhen des Wertes einer Variablen ist in der Programmierung so gebräuchlich, dass Programmiersprachen üblicherweise eine einfachere Methode vorsehen, um dies zu bewerkstelligen. Das Programm oben erzielt dasselbe Ergebnis wie dieses.

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

Und die Ausgabe ist:



```
C:\Users\chrissto\AppData\Local\Temp\tmp6A0C.tmp.exe
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
Press any key to continue...
```

Abbildung 19 – Verwendung der For-Schleife

Beachten Sie, dass wir das 8-Zeilen-Programm auf ein 4-Zeilen-Programm reduziert haben, das noch genau das gleiche macht wie das 8-Zeilen-Programm! Denken Sie daran, dass wir früher gesagt haben, dass es oft verschiedene Wege gibt, um ein Ergebnis zu erzielen. Dies hier ist ein gutes Beispiel.

For..EndFor wird in der üblichen Programmierausdrucksweise eine **Schleife** genannt. Sie erlaubt es Ihnen, einer Variablen einen Start- und einen Endwert zu geben, und es dem Computer zu überlassen, die Variable für Sie zu erhöhen. Jedes Mal, wenn der Computer die Variable erhöht, führt er die Anweisungen zwischen **For** und **EndFor** aus.

Wenn Sie die Variable um 2 anstatt um 1 erhöhen oder die ungeraden Zahlen zwischen 1 und 24 ausgeben wollen, können Sie die Schleife auch hierfür verwenden.

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

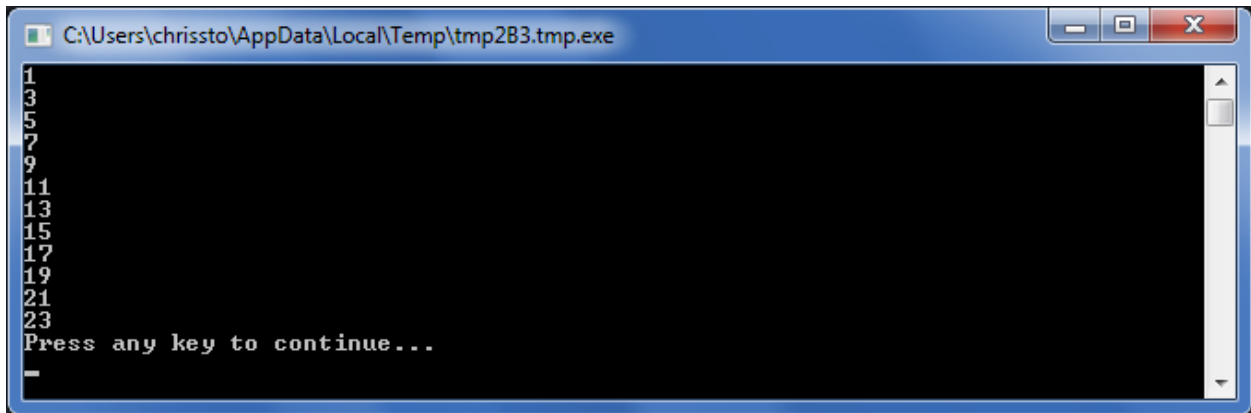


Abbildung 20 – Nur die ungeraden Zahlen

Der Teil **Step 2** der **For**-Anweisung sagt dem Computer, den Wert von **i** um zwei anstelle wie üblich um 1 zu erhöhen. Durch die Verwendung von **Step** können Sie jede beliebige Erhöhung angeben. Sie können sogar einen negativen Wert für die Schrittweite angeben und den Computer so veranlassen, rückwärts zu zählen, wie im Beispiel unten:

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

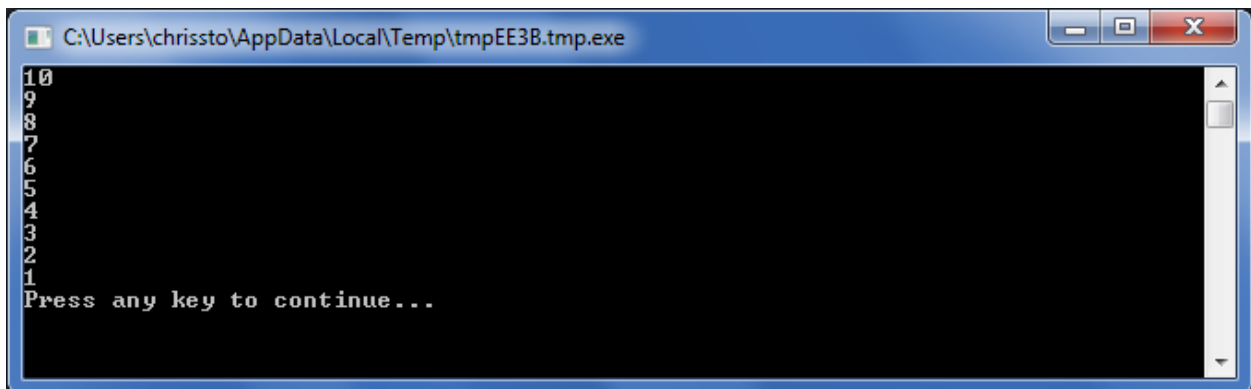


Abbildung 21 – Rückwärts zählen

Die While-Schleife

Die While-Schleife ist noch eine weitere Schleifenart, die insbesondere dann nützlich ist, wenn die Anzahl der Schleifendurchläufe nicht vorher bekannt ist. Während eine For-Schleife eine vordefinierte Anzahl von Malen läuft, läuft die While-Schleife, solange eine vorgegebene Bedingung erfüllt ist. Im Beispiel unten halbieren wir eine Zahl solange sie größer als 1 ist.

```

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

```

```

100
50
25
12.5
6.25
3.125
1.5625
Press any key to continue...

```

Abbildung 22 – Halbieren in einer Schleife

Im Programm oben weisen wir ihr den Wert 100 zu und durchlaufen die While-Schleife solange *number* größer als 1 ist. Innerhalb der Schleife geben wir *number* aus und teilen den Wert dann durch 2. Und wie erwartet sind Zahlen, die nacheinander halbiert werden, die Ausgabe des Programmes.

Es wäre wirklich schwierig, dieses Programm mit einer For-Schleife zu schreiben, weil wir nicht wissen, wie oft die Schleife durchlaufen wird. Bei einer While-Schleife kann eine Bedingung leicht geprüft werden und in Abhängigkeit davon dem Computer gesagt werden, den Schleifendurchlauf fortzusetzen oder zu beenden.

In der Tat schreibt der Computer jede While-Schleife intern in eine Anweisung um, die If..Then-Anweisungen, kombiniert mit einer oder mehreren Goto-Anweisung(en) enthält.

Es ist interessant, darauf hinzuweisen, dass jede While-Schleife in eine If..Then-Anweisung umgeschrieben werden kann. Zum Beispiel kann das Programm oben wie folgt umgeschrieben werden, ohne das Endergebnis zu ändern.

```

number = 100
startLabel:
TextWindow.WriteLine(number)
number = number / 2

If (number > 1) Then
    Goto startLabel
EndIf

```


Arbeiten mit Grafik

Bisher haben wir in all unseren Beispielen das Textfenster benutzt, um die Grundlagen der Sprache Small Basic darzustellen. Small Basic bietet auch verschiedene Grafikfähigkeiten, deren Erkundung wir in diesem Kapitel beginnen wollen.

Einführung von GraphicsWindow

TextWindow erlaubte es uns, mit Text und Zahlen zu arbeiten. Mit dem **GraphicsWindow** (Grafikfenster) können wir Formen zeichnen.

```
GraphicsWindow.Show()
```

Wenn Sie dieses Programm laufen lassen, werden Sie feststellen, dass Sie anstelle des üblichen schwarzen Textfensters ein weißes Fenster wie das unten gezeigte erhalten werden. Viel können Sie mit diesem Fenster bisher nicht machen. Es wird aber das grundlegende Fenster sein, mit dem wir in diesem Kapitel arbeiten werden. Sie können dieses Fenster schließen, indem Sie auf die ‚X‘-Schaltfläche in der oberen rechten Ecke klicken.

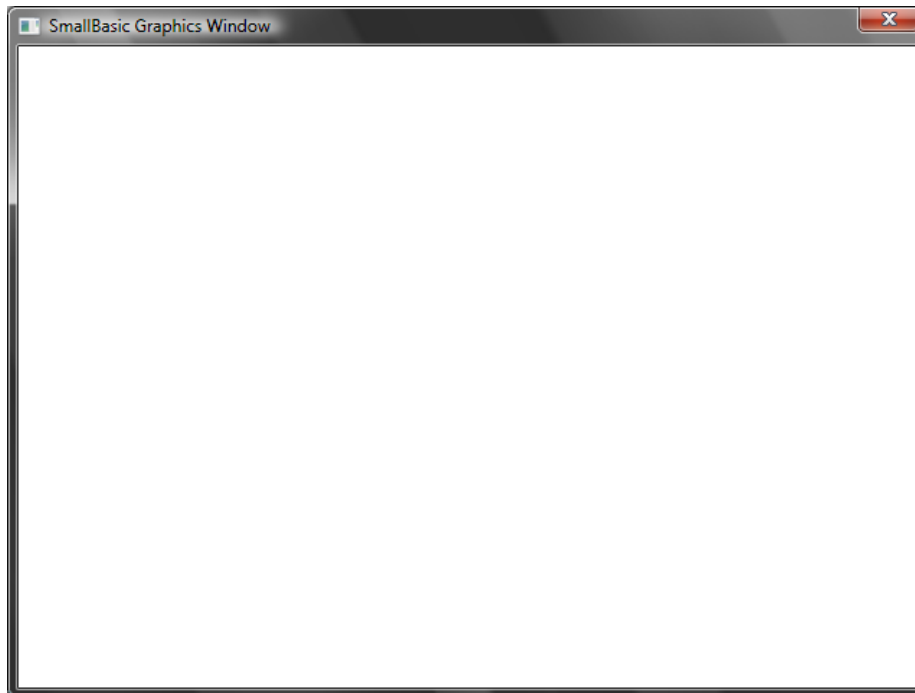


Abbildung 23 – Ein leeres Grafikfenster (Graphics Window)

Grundeinstellungen des Grafikfensters

Das Grafikfenster erlaubt es Ihnen, seine Erscheinung an Ihre Wünsche anzupassen. Lassen Sie es uns ein wenig verändern, einfach nur, um uns mit dem Fenster vertraut zu werden.

```
GraphicsWindow.BackgroundColor = "SteelBlue"  
GraphicsWindow.Title = "My Graphics Window"  
GraphicsWindow.Width = 320  
GraphicsWindow.Height = 200  
GraphicsWindow.Show()
```

Hier ist zu sehen, wie das angepasste Fenster aussieht. Sie können die Hintergrundfarbe in eine der vielen in Anhang B aufgelisteten ändern. Spielen Sie ein wenig mit diesen Eigenschaften, um zu sehen, wie Sie das Erscheinungsbild des Fensters verändern können.

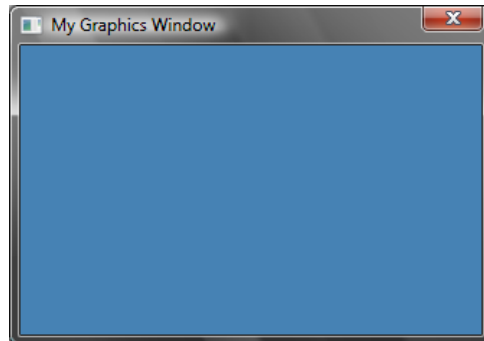


Abbildung 24 – Ein angepasstes Grafikfenster

Linien zeichnen

Sobald das Grafikfenster erst einmal da ist, können wir Formen, Text und sogar Bilder darauf zeichnen. Lassen Sie uns zuerst einfache Formen zeichnen. Hier ein Programm, das ein Paar von Linien in das Grafikfenster zeichnet.

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

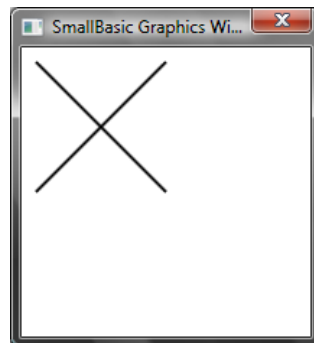


Abbildung 25 – Kreuz

Die ersten beiden Programmzeilen erstellen das Fenster, die nächsten zwei Zeilen zeichnen die Linien des Kreuzes. Die ersten zwei Zahlen nach **DrawLine** bestimmen die x- und y-Koordinaten des

Anstatt Namen für Farben zu benutzen, können Sie die im Web übliche Angabe für Farben benutzen (#RRGGBB). Zum Beispiel bezeichnet #FF0000 Rot, #FFFF00 Gelb, usw.

Anfanges, die anderen beiden die x- und y-Koordinaten des Endes.

Interessant bei der Computergrafik ist, dass die Koordinaten (0,0) in der oberen linken Ecke des Fensters liegen.

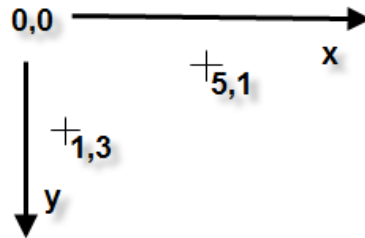


Abbildung 26 – Das Koordinatensystem

Wenn wir zu unserem Linien-Programm zurückkehren, werden wir interessiert bemerken, dass Small Basic es Ihnen ermöglicht, die Eigenschaften der Linie - wie Farbe oder Linienstärke - zu verändern. Lassen Sie uns zuerst die Farbe der Linien, wie im Programm unten gezeigt, verändern.

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

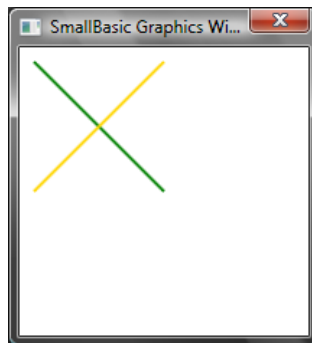


Abbildung 27 – Verändern der Linienfarbe

Nun wollen wir auch die Größe verändern. Im Programm unten ändern wir die Linienstärke auf 10 anstelle des Standardwertes 1.

```
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)
```

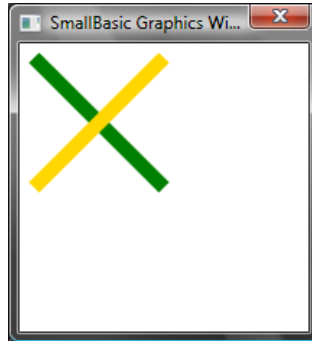


Abbildung 28 – Breite farbige Linien

PenWidth (Stiftbreite) und **PenColor** (Stiftfarbe) verändern den Stift, mit dem die Linien gezeichnet sind. Sie beeinflussen nicht nur Linien, sondern jede Form, die gezeichnet wird, nachdem die Eigenschaften geändert wurden. Mit den Schleifenanweisungen, die wir in den vorigen Kapiteln gelernt haben, können wir leicht ein Programm schreiben, das verschiedene Linien mit größer werdender Stiftbreite zeichnet.

```
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
```

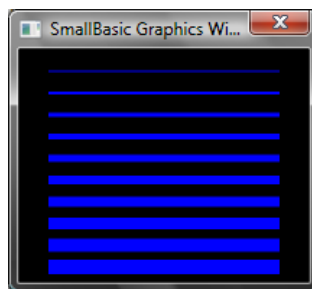


Abbildung 29 – Viele Stiftbreiten

Der interessante Teil des Programmes ist die Schleife, in der wir bei jedem Schleifendurchlauf **PenWidth** erhöhen und dann eine neue Linie unter der alten zeichnen.

Formen zeichnen und füllen

Beim Zeichnen von Formen gibt es üblicherweise zwei verschiedene Arbeiten für jede Form. Dies sind **Draw**- und **Fill**-Operationen. Zeichenoperationen zeichnen den Umriss einer Form mit einem Stift und Fülloperationen bemalen die Form mit einem Pinsel (Brush). Im untenstehenden Programm gibt es zum Beispiel zwei Rechtecke, von denen eines mit einem roten Stift gezeichnet und das andere mit einem grünen Pinsel bemalt ist.

```
GraphicsWindow.Width = 400  
GraphicsWindow.Height = 300  
  
GraphicsWindow.PenColor = "Red"  
GraphicsWindow.DrawRectangle(20, 20, 300, 60)  
  
GraphicsWindow.BrushColor = "Green"  
GraphicsWindow.FillRectangle(60, 100, 300, 60)
```

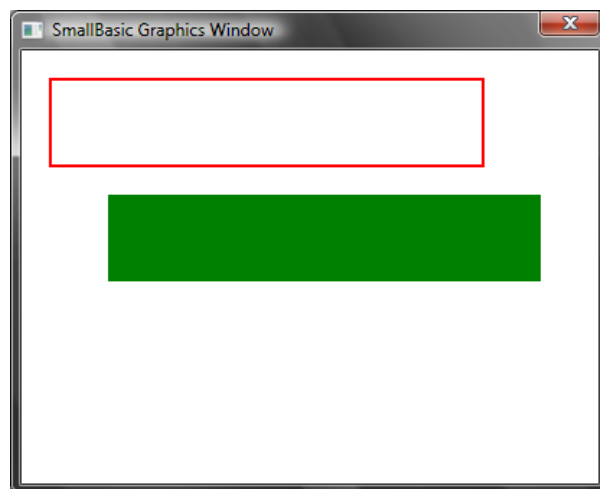


Abbildung 30 Zeichnen und Füllen

Um ein Rechteck zu zeichnen oder zu füllen, brauchen Sie vier Zahlen. Die ersten beiden Zahlen repräsentieren die x- und y-Koordinaten der linken oberen Ecke des Rechteckes. Die dritte Zahl gibt die Breite des Rechteckes an, während die vierte die Höhe bestimmt. Gleiches gilt tatsächlich auch für das Zeichnen und Füllen von Ellipsen, wie im folgenden Programm beschrieben.

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

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

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

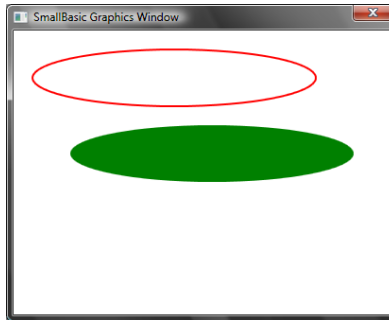


Abbildung 31 – Zeichnen und Füllen von Ellipsen

Ellipsen sind Kreisen sehr ähnlich. Wenn Sie Kreise zeichnen wollen, müssen Sie lediglich die gleiche Breite wie Höhe angeben.

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

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

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

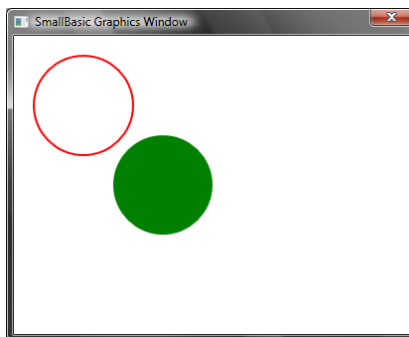


Abbildung 32 – Kreise

Spaß mit Formen

In diesem Kapitel wollen wir ein wenig Spaß haben mit all dem, was wir bisher gelernt haben. Es enthält Beispiele, die einige interessante Wege aufzeigen, das bisher Gelernte anzuwenden, um ein paar coole kleine Programme zu erstellen.

Verschachteltes Rechteck

Hier zeichnen wir in einer Schleife mehrere Rechtecke mit ansteigender Größe:

```
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
```

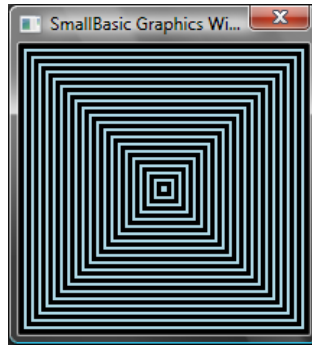



Abbildung 33 – Rechteck als Muster

Verschachtelte Kreise

Eine Variante des vorigen Programmes zeichnet Kreise anstelle von Quadraten.

```
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
```

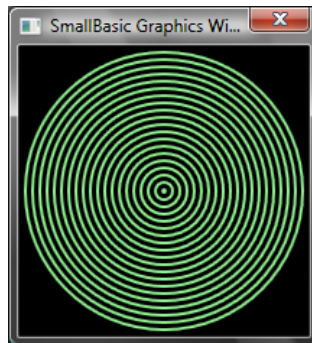


Abbildung 34 – Tolle Kreise

Zufallsprinzip

Dieses Programm nutzt die Methode **GraphicsWindow.GetRandomColor**, um zufällige Farben für den Pinsel zu erzeugen und dann **Math.GetRandomNumber**, um die x- und y-Koordinaten für die Kreise festzulegen. Diese beiden Vorgänge können auf interessante Arten kombiniert werden, um interessante Programme zu schreiben, die bei jedem Programmlauf andere Resultate erzeugen.

```

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

```

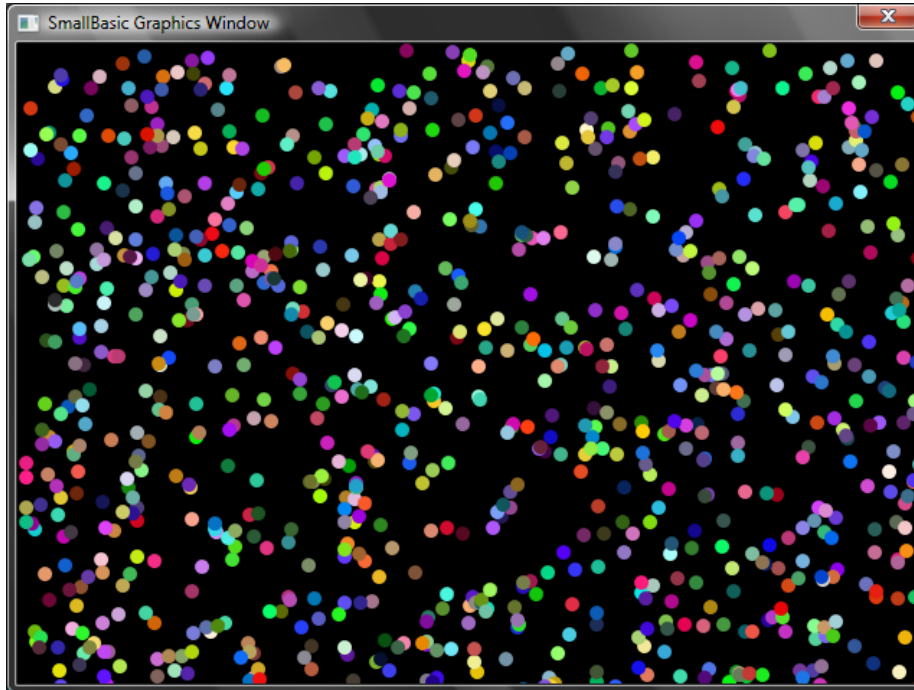


Abbildung 35 – Zufallsprinzip

Fraktale

Das folgende Programm verwendet Zufallszahlen, um ein einfaches dreieckiges Fraktal zu zeichnen. Ein Fraktal ist eine geometrische Figur, die in Teile unterteilt werden kann, von denen jedes der Elternfigur ähnelt. In diesem Fall zeichnet das Programm Hunderte von Dreiecken, von denen jedes seinem Elterndreieck gleicht. Und während das Programm einige Sekunden läuft, können Sie tatsächlich sehen, wie die Dreiecke langsam aus immer mehr Punkten gezeichnet werden. Die Programmlogik ist schwer zu beschreiben und deshalb überlasse ich es Ihnen als Übung, sie selbst zu erforschen.

```

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

```

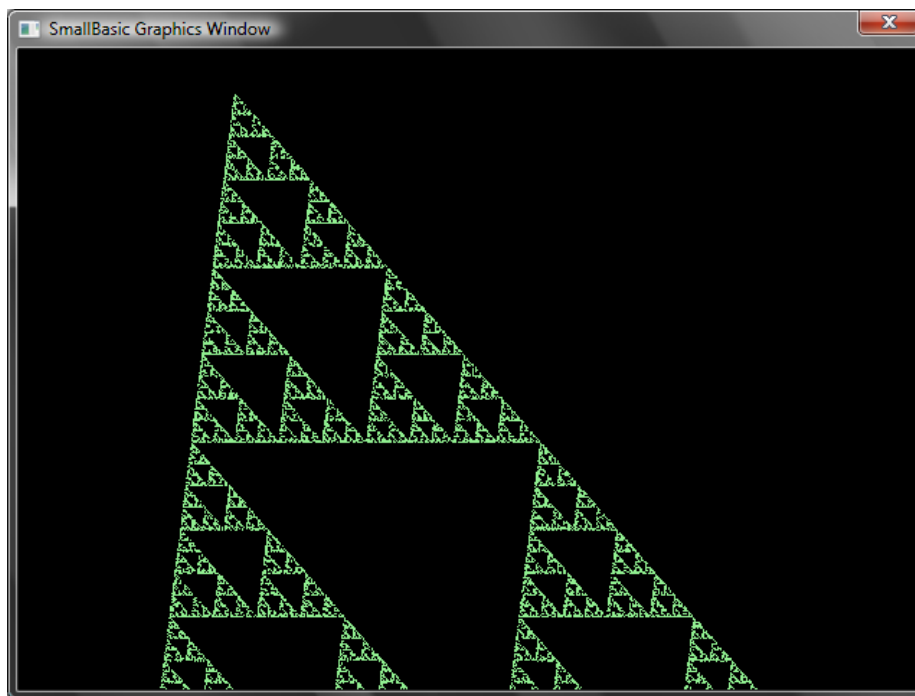


Abbildung 36 – Sierpinski-Dreieck/Dreiecks-Fraktal

Wenn Sie wirklich sehen wollen, wie die Punkte langsam das Fraktal aufbauen, können Sie in der Schleife mit der **Program.Delay**-Methode eine Verzögerung einbauen. Die Methode erwartet eine Zahl, die in Millisekunden angibt, wie lang die Verzögerung ist. Hier das modifizierte Programm mit der geänderten Zeile in Fettdruck (**Program.Delay(2)**).

```

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

```

Eine Erhöhung der Verzögerung macht das Programm langsamer. Probieren Sie verschiedene Zahlen aus um zu sehen, was am meisten nach Ihrem Geschmack ist. Als weitere Veränderung können Sie diese Zeile

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

durch folgende ersetzen:

```

color = GraphicsWindow.GetRandomColor()
GraphicsWindow.SetPixel(x, y, color)

```

Diese Änderung lässt das Programm die Punkte der Dreiecke in zufälligen Farben zeichnen.

Schildkrötengrafik

Logo

In den Siebzigerjahren gab es eine sehr einfache aber leistungsfähige Programmiersprache mit dem Namen Logo, die nur von einigen wenigen Wissenschaftlern benutzt wurde. Das änderte sich schlagartig, als die „Schildkrötengrafik“ hinzugefügt wurde, womit eine auf dem Bildschirm sichtbare „Schildkröte“ (Turtle) verfügbar gemacht wurde, die auf Kommandos wie **Move Forward** (Vorwärts bewegen), **Turn Right** (Rechts drehen), **Turn Left** (Links drehen) usw. hörte. Mit der Schildkröte konnte man schon damals interessante Formen zeichnen. Das machte diese Sprache für Programmierer sehr attraktiv und war weitgehend verantwortlich für die enorme Popularität in den Achtzigerjahren. Small Basic hat ein **Turtle**-Objekt mit vielen Befehlen, die aus Small Basic-Programmen heraus aufgerufen werden können. In diesem Kapitel werden wir mit der Schildkröte Grafiken zeichnen.

Die Schildkröte (Turtle)

Machen wir die Schildkröte zunächst einmal auf dem Bildschirm sichtbar. Dies kann mit einem einfachen Einzeiler erreicht werden.

```
Turtle.Show()
```

Wenn Sie dieses Programm ablaufen lassen, werden Sie ein weißes Fenster sehen - genauso wie wir es im vorigen Kapitel gesehen haben - aber mit der Ausnahme, dass in der Mitte eine Schildkröte zu sehen ist. Diese Schildkröte wird unseren Anweisungen folgen und alles zeichnen, worum wir sie bitten.

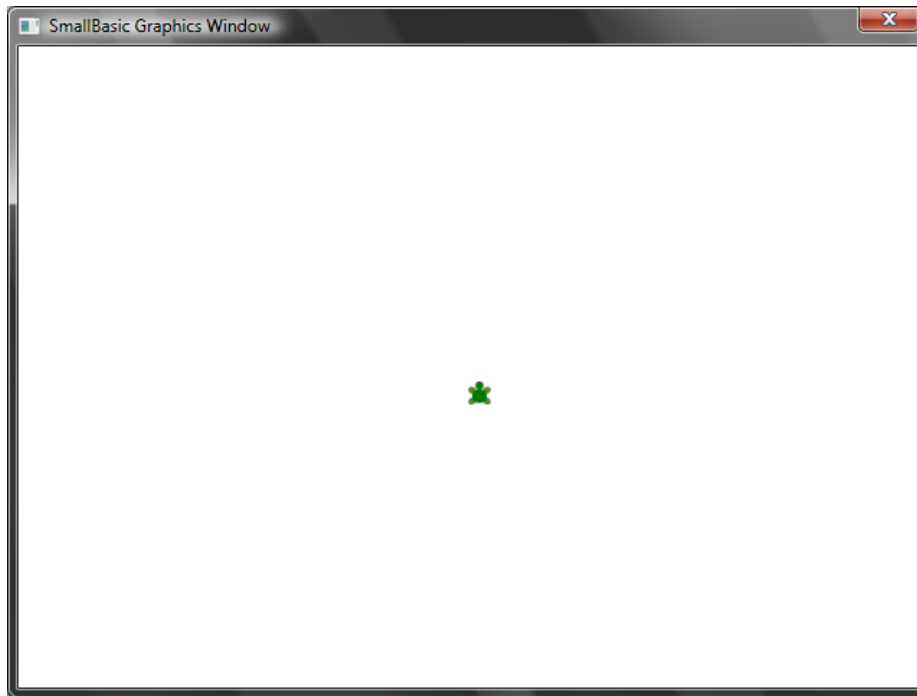


Abbildung 37 – Die Schildkröte

Bewegen und Zeichnen

Einer der Befehle, den die Schildkröte versteht, ist **Move** (Bewegen). Der Befehl erwartet die Eingabe einer Zahl. Die Zahl gibt an, wie weit sich die Schildkröte bewegen soll. Im Beispiel unten bitten wir die Schildkröte, sich 100 Pixel weit zu bewegen.

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

Wenn Sie das Programm laufen lassen, sehen Sie tatsächlich, wie die Schildkröte sich langsam 100 Pixel nach oben bewegt. Sie bemerken, dass sie bei der Bewegung eine Linie hinter sich zeichnet. Wenn die Schildkröte die Bewegung beendet hat, wird das Ergebnis aussehen wie in der Abbildung unten.

Wenn Sie der Schildkröte Befehle erteilen, muss Show() nicht aufgerufen werden. Die Schildkröte wird automatisch sichtbar, wenn eine Schildkrötenanweisung ausgeführt wird.

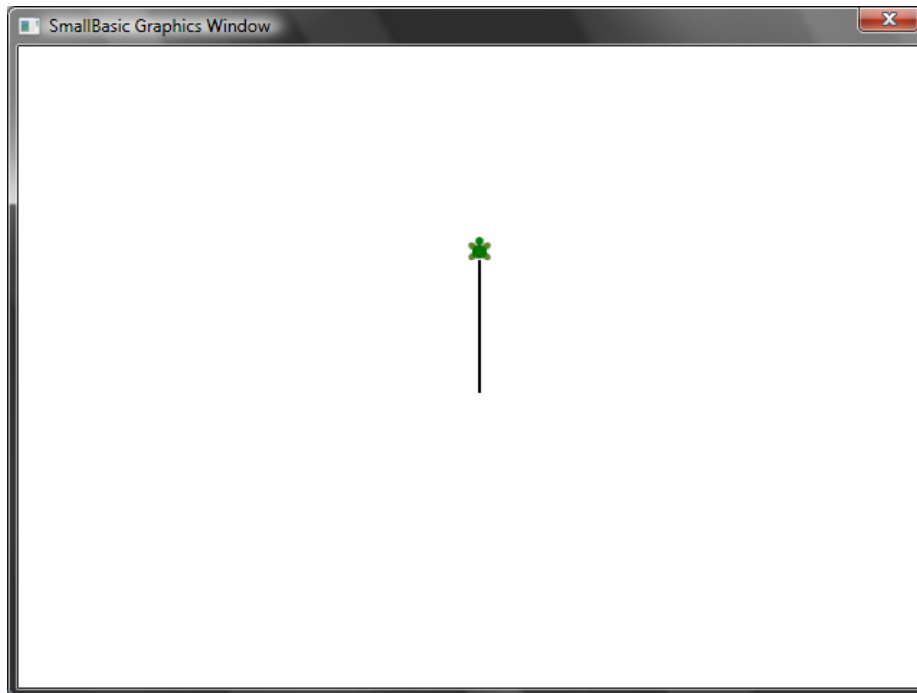


Abbildung 38 – Hundert Pixel nach oben

Zeichnen eines Quadrates

Ein Quadrat hat vier Seiten, zwei vertikale und zwei horizontale. Um ein Quadrat zu zeichnen, müssen wir der Schildkröte mitteilen, eine Linie zu zeichnen, sich dann nach rechts zu drehen, eine weitere Linie zu zeichnen und dies fortzusetzen, bis alle vier Seiten fertig sind. Das Ganze sieht so aus:

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

Wenn Sie das Programm starten, können Sie sehen, wie die Schildkröte Linie für Linie ein Quadrat zeichnet. Das Ergebnis sieht dann wie folgt aus:

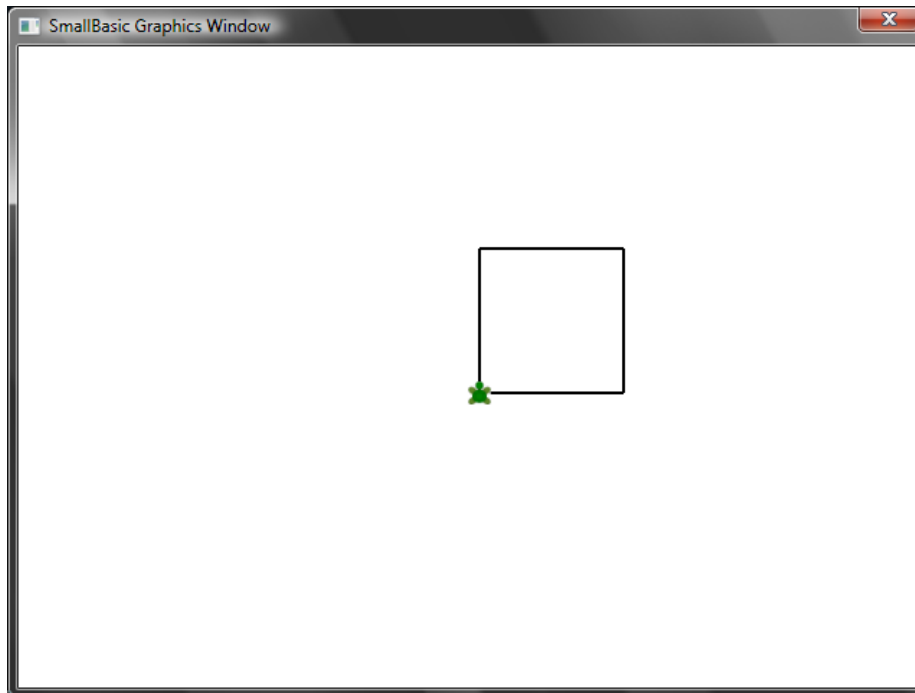


Abbildung 39 – Die Schildkröte hat ein Quadrat gezeichnet

Wir haben hierbei viermal die gleiche Anweisung ausgeführt. Und wir haben bereits gelernt, dass solche sich wiederholenden Befehle unter Benutzung von Schleifen ausgeführt werden können. Wenn wir das Programm so verändern, dass es eine **For..EndFor**-Schleife verwendet, können wir es auf die folgenden Zeilen reduzieren:

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

Farben verändern

Die Schildkröte zeichnet in das gleiche Grafikfenster, das wir im vorherigen Kapitel kennengelernt haben. Dies bedeutet, dass all die Operationen vom vorherigen Kapitel auch hier gültig sind. Zum Beispiel wird dieses Programm das Quadrat mit jeder Seite in einer anderen Farbe zeichnen.

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

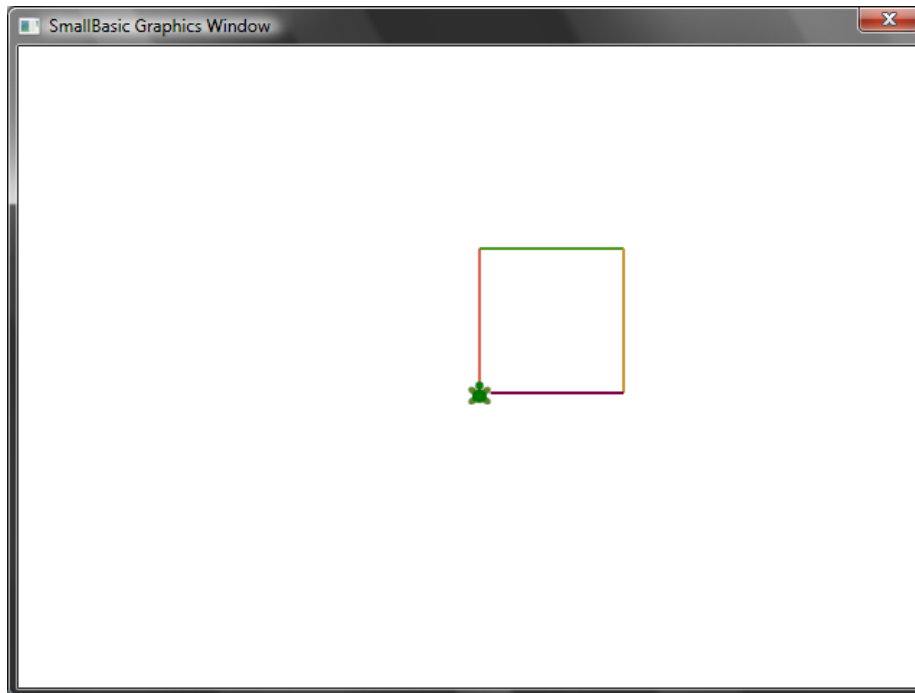



Abbildung 40 – Unterschiedliche Farben

Komplexe Formen zeichnen

Zusätzlich zu den **TurnRight**- und **TurnLeft**-Methoden hat die Schildkröte auch eine **Turn**-Methode. Diese Methode erwartet eine Eingabe, die den Winkel der Drehung angibt. Mit dieser Methode können beliebigseitige Vielecke gezeichnet werden. Das folgende Programm zeichnet ein Hexagon (ein sechsseitiges Polygon/Vieleck).

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

Probieren Sie, ob es wirklich ein Hexagon zeichnet. Achten Sie darauf, dass wir **Turn(60)** angeben, weil der Winkel zwischen den Seiten 60 Grad beträgt. Für ein solches Polygon, dessen Seiten alle gleich sind, kann der Winkel zwischen den Seiten ganz einfach errechnet werden, indem 360 durch die Anzahl der Seiten geteilt wird. Ausgerüstet mit dieser Information und unter Verwendung von Variablen können wir ein allgemeingültiges Programm schreiben, das beliebigseitige Polygone zeichnet.

```
sides = 12

length = 400 / sides
angle = 360 / sides
```

```
For i = 1 To sides
  Turtle.Move(length)
  Turtle.Turn(angle)
EndFor
```

Mit diesem Programm können Sie jedes Polygon zeichnen, indem Sie einfach die Variable **sides** verändern. Die Eingabe von 4 ergäbe das Quadrat, mit dem wir angefangen haben. Ein ausreichend großer Wert, sagen wir 50, ergibt ein Ergebnis, das von einem Kreis nicht unterschieden werden kann.

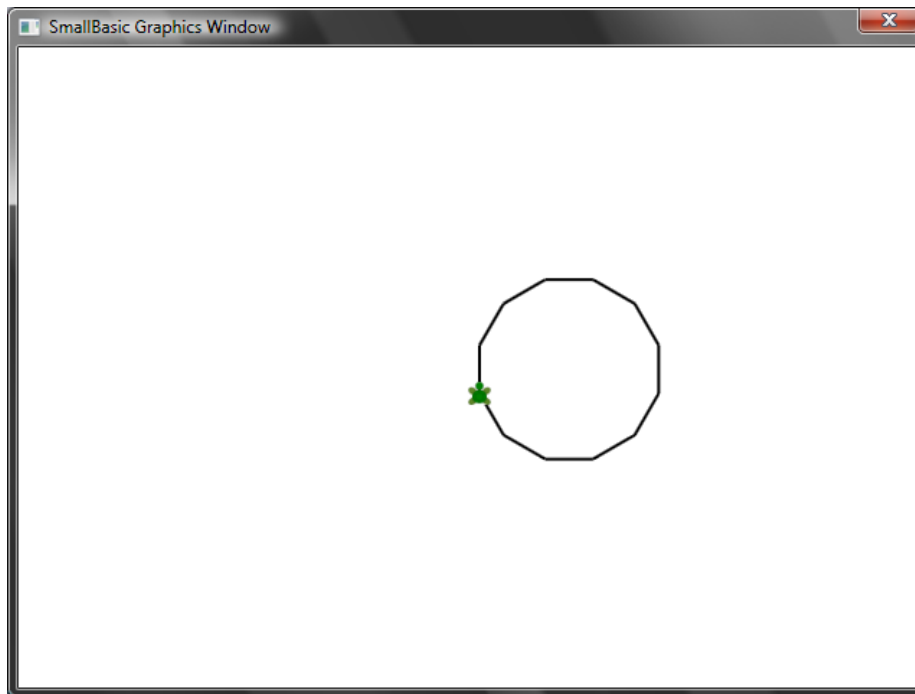


Abbildung 41 – Zeichnen eines 12-seitigen Vielecks

Mit dieser neuen Technik können wir die Schildkröte zahlreiche Kreise zeichnen lassen, und zwar jedes Mal mit einer kleinen Verschiebung, was zu einem interessanten Ergebnis führt.

```

sides = 50
length = 400 / sides
angle = 360 / sides

Turtle.Speed = 9

For j = 1 To 20
  For i = 1 To sides
    Turtle.Move(length)
    Turtle.Turn(angle)
  EndFor
  Turtle.Turn(18)
EndFor

```

Das Programm oben hat zwei **For..EndFor**-Schleifen, eine in der anderen. Die innere Schleife ($i = 1$ to $sides$) ähnelt dem Polygon-Programm und ist dafür verantwortlich, einen Kreis zu zeichnen. Die äußere Schleife ($j = 1$ to 20) ist dafür verantwortlich, die Schildkröte nach jedem gezeichneten Kreis ein wenig zu drehen. Damit wird der Schildkröte gesagt, sie solle zwanzig Kreise zeichnen. Das ergibt ein sehr interessantes Muster:

Im obenstehenden Programm haben wir die Schildkröte schneller gemacht, indem wir Speed (Geschwindigkeit) auf 9 gesetzt haben. Sie können diese Eigenschaft auf jeden Wert von 1 bis 10 setzen, um die Schildkröte so schnell gehen zu lassen wie Sie wollen.

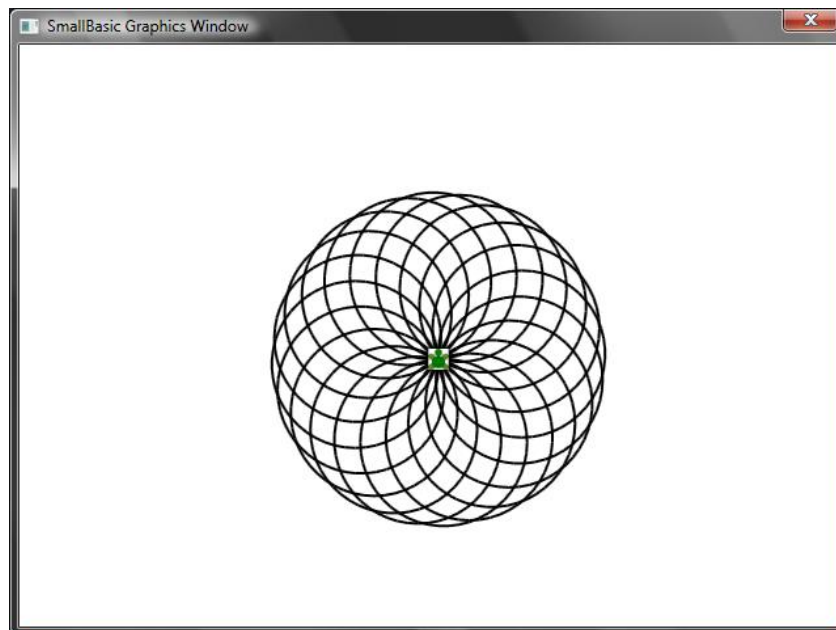


Abbildung 42 – In Kreisen gehen

Umhergehen

Sie können die Schildkröte mit der **PenUp**-Methode veranlassen, nicht zu zeichnen. Damit können Sie die Schildkröte an eine beliebige Stelle auf dem Bildschirm bewegen, ohne eine Linie zu zeichnen.

Der Aufruf von **PenDown** (Stift hinunter) lässt die Schildkröte wieder zeichnen. Damit können interessante Effekte erzielt werden, zum Beispiel gepunktete Linien. Hier ist ein Programm, das ein Polygon aus gepunkteten Linien zeichnet.

```
sides = 6

length = 400 / sides
angle = 360 / sides

For i = 1 To sides
  For j = 1 To 6
    Turtle.Move(length / 12)
    Turtle.PenUp()
    Turtle.Move(length / 12)
    Turtle.PenDown()
  EndFor
  Turtle.Turn(angle)
EndFor
```

Auch dieses Programm hat zwei Schleifen. Die innere Schleife zeichnet eine einzelne gepunktete Linie, während die äußere Schleife angibt, wie viele Linien gezeichnet werden. In unserem Beispiel verwendeten wir 6 für **sides** und bekamen aus diesem Grunde ein Sechseck aus gepunkteten Linien:

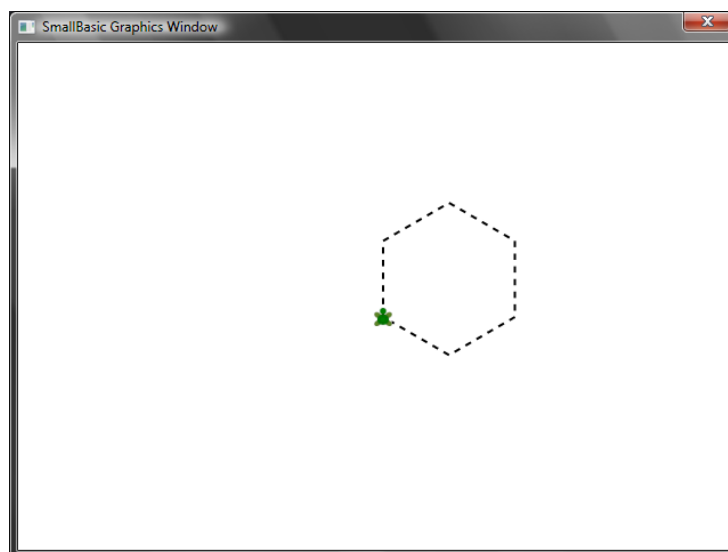


Abbildung 43 - PenUp und PenDown

Unterprogramme

Sehr oft beim Schreiben von Programmen werden Fälle auftreten, in denen wir die gleichen Schritte immer und immer wieder ausführen müssen. In diesen Fällen würde es wahrscheinlich keinen Sinn machen, die gleichen Anweisungen mehrfach zu schreiben. Hierfür sind *Unterprogramme* sehr nützlich.

Ein Unterprogramm ist ein Teil eines größeren Programmes, das üblicherweise etwas ganz Spezielles macht und von überall im Programm aus aufgerufen werden kann. Unterprogramme werden durch ihren Namen gekennzeichnet, auf den das Schlüsselwort **Sub** folgt. Das Schlüsselwort **EndSub** beendet ein Unterprogramm. Zum Beispiel repräsentiert der folgende kleine Programmteil ein Unterprogramm mit dem Namen **PrintTime**, der die Aufgabe erledigt, die momentane Uhrzeit im Textfenster auszugeben.

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

Hier ein Programm, welches das Unterprogramm enthält und es von verschiedenen Orten aus aufruft.

```
PrintTime()  
TextWindow.Write("Enter your name: ")  
name = TextWindow.Read()  
TextWindow.Write(name + ", the time now is: ")  
PrintTime()  
  
Sub PrintTime  
    TextWindow.WriteLine(Clock.Time)  
EndSub
```

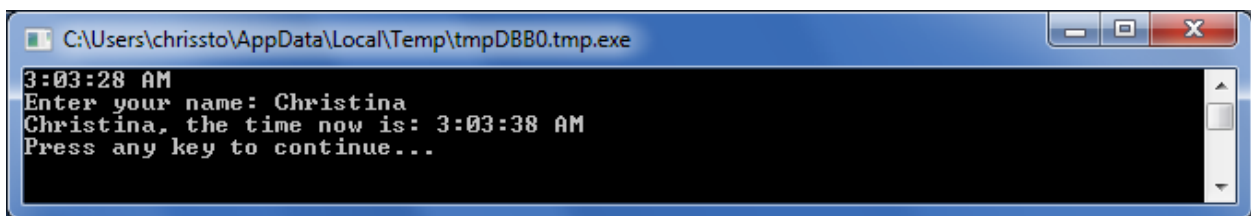


Abbildung 44 – Aufruf eines einfachen Unterprogrammes

Ein Unterprogramm wird ausgeführt beim Aufruf von `UnterprogrammName()`. Die Klammern `()` sind erforderlich, um dem Computer zu sagen, dass Sie ein Unterprogramm ausführen wollen.

Denken Sie daran, dass Sie ein Small Basic-Unterprogramm nur aus demselben Programm aus aufrufen können. Sie können ein Unterprogramm nicht von einem anderen Programm aus aufrufen.

Vorteile durch den Gebrauch von Unterprogrammen

Wie wir gerade gesehen haben, können Unterprogramme helfen, die Menge von Programmcode zu vermindern, den Sie eingeben müssen. Sobald Sie das **PrintTime**-Unterprogramm einmal geschrieben haben, können Sie es von überall im Programm aus aufrufen und es wird die aktuelle Zeit ausgeben.

Zudem können Unterprogramme helfen, komplizierte Probleme in einfachere Teile zu zerlegen. Sagen wir mal, Sie hätten eine komplizierte Gleichung zu lösen, dann können Sie verschiedene Unterprogramme schreiben, die kleinere Teile der komplizierten Gleichung lösen. Danach können Sie die Ergebnisse zusammenfügen, um die Lösung der komplizierten Gleichung zu erhalten.

Unterprogramme können zudem dazu führen, dass die Lesbarkeit eines Programmes verbessert wird. Mit anderen Worten, wenn Sie eingängig benannte Unterprogramme für gewöhnlich ablaufende Programmteile verwenden, wird ihr Programm leicht lesbar. Dies ist sehr wichtig, wenn Sie das Programm eines anderen Programmiers verstehen wollen oder wenn Sie Ihr eigenes Programm lesbarer machen wollen. Manchmal ist es auch hilfreich, wenn Sie Ihr eigenes Programm lesen wollen, wenn Sie zum Beispiel eine längere Zeit nicht daran gearbeitet haben.

Die Verwendung von Variablen

Auf jede Variable, die Sie in einem Programm verwenden, können Sie aus einem Unterprogramm zugreifen. Zum Beispiel nimmt das folgende Programm zwei Zahlen entgegen und gibt die größere von beiden aus. Beachten Sie, dass die Variable *max* sowohl außerhalb als auch innerhalb des Unterprogramms benutzt wird.

```
TextWindow.Write("Enter first number: ")
num1 = TextWindow.ReadNumber()
TextWindow.Write("Enter second number: ")
num2 = TextWindow.ReadNumber()

FindMax()
TextWindow.WriteLine("Maximum number is: " + max)

Sub FindMax
  If (num1 > num2) Then
    max = num1
  Else
    max = num2
  EndIf
EndSub
```

Und die Ausgabe des Programmes sieht so aus.

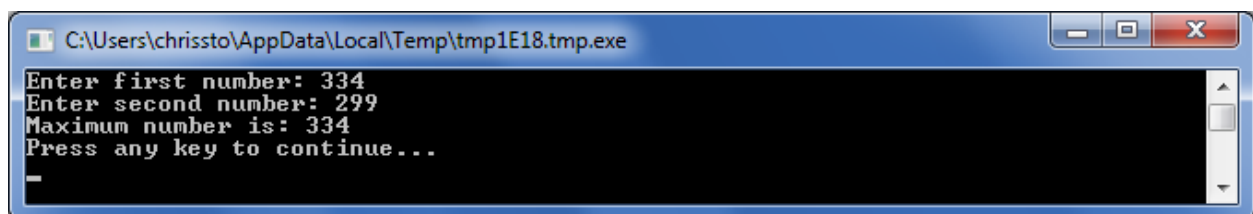


Abbildung 45 – Die größte zweier Zahlen unter Verwendung eines Unterprogrammes

Lassen Sie uns ein anderes Beispiel ansehen, welches den Gebrauch von Unterprogrammen illustriert. Dieses Mal werden wir ein Grafikprogramm verwenden, das verschiedene Punkte berechnet, die es in den Variablen *x* und *y* speichert. Dann ruft es ein Unterprogramm **DrawCircleUsingCenter**

(ZeichneKreisBenutzeMittelpunkt), welches dafür verantwortlich ist, einen Kreis zu zeichnen und dabei x und y als Mittelpunkt zu verwenden.

```
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
```

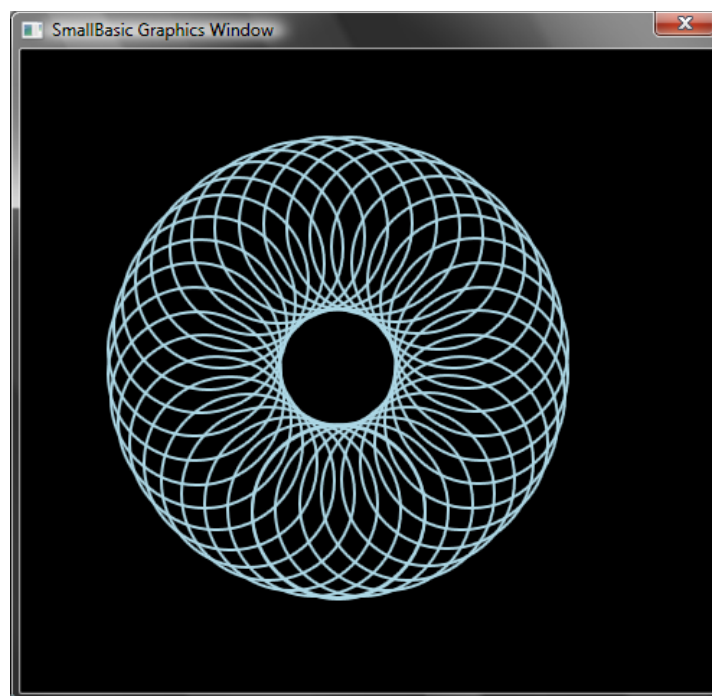


Abbildung 46 – Grafikbeispiel mit Unterprogramm

Unterprogramme innerhalb von Schleifen aufrufen

Manchmal werden Unterprogramme aus einer Schleife heraus aufgerufen, wobei jeweils die gleichen Anweisungen, aber mit verschiedenen Werten für eine oder mehrere Variablen ausgeführt werden. Sagen wir zum Beispiel einmal, Sie haben ein Unterprogramm mit dem Namen **PrimeCheck** (Primzahlprüfung), und dieses Unterprogramm bestimmt, ob eine Zahl eine Primzahl ist oder nicht. Sie können ein Programm schreiben, das den Benutzer eine Zahl eingeben lässt. Sie können dann unter Benutzung des Unterprogramms sagen, ob diese Zahl eine Primzahl ist oder nicht, wie hier dargestellt:

```
TextWindow.Write("Enter a number: ")
i = TextWindow.ReadNumber()
isPrime = "True"
PrimeCheck()
If (isPrime = "True") Then
    TextWindow.WriteLine(i + " is a prime number")
Else
    TextWindow.WriteLine(i + " is not a prime number")
EndIf

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

Das PrimeCheck-Unterprogramm nimmt den Wert von *i* und versucht, ihn durch kleinere Zahlen zu teilen. Wenn *i* durch eine Zahl geteilt wird und kein Rest bleibt, dann ist *i* keine Primzahl. An dieser Stelle gibt das Unterprogramm **isPrime** (istPrimzahl) den Wert **False** (Falsch) und endet. Wenn die Zahl nicht ohne Rest durch kleinere Zahlen teilbar war, bleibt der Wert von **isPrime True** (Wahr).

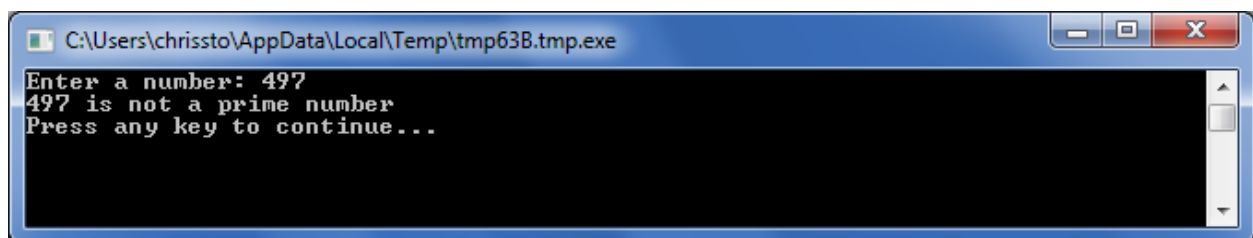


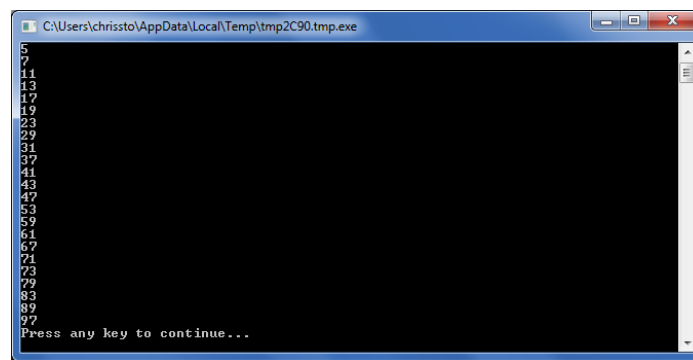
Abbildung 47 - Primzahltest

Nun, wo wir ein Unterprogramm haben, das für uns den Primzahlentest durchführen kann, könnten Sie den Wunsch haben, alle Primzahlen unter, sagen wir mal 100, aufzulisten. Es ist wirklich einfach, das obige Programm zu verändern und den Aufruf von **PrimeCheck** aus einer Schleife heraus auszuführen. Damit wird dem Unterprogramm bei jedem Schleifendurchlauf ein anderer Wert zur Berechnung übergeben. Anhand dieses Beispiels wollen wir uns ansehen, wie das gemacht wird:

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

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

Im Programm erhält *i* bei jedem Schleifendurchlauf einen neuen Wert. Innerhalb der Schleife wird das Unterprogramm **PrimeCheck** aufgerufen. Das Unterprogramm **PrimeCheck** nimmt dann den Wert von *i* und berechnet, ob *i* eine Primzahl ist oder nicht. Das Ergebnis wird in der Variablen *isPrime* gespeichert, auf die dann durch die Schleife außerhalb des Unterprogrammes zugegriffen wird. Der Wert von *i* wird ausgegeben, wenn sich herausgestellt hat, dass es eine Primzahl ist. Und weil die Schleife bei drei startet und bis 100 läuft, erhalten wir eine Liste aller Primzahlen von 3 bis 100. Hier das Ergebnis des Programms:



```
C:\Users\chrissto\AppData\Local\Temp\tmp2C90.tmp.exe
3
5
7
11
13
17
19
23
29
31
37
41
43
47
53
59
61
67
71
73
79
83
89
97
Press any key to continue...
```

Abbildung 48 – Primzahlen

Inzwischen sind uns Variablen schon vertraut, und das Programmieren macht immer noch Spaß.

Hier noch einmal zur Veranschaulichung unser erstes Programm, in dem wir Variablen verwendeten:

```
TextWindow.Write("Enter your Name: ")
name = TextWindow.Read()
TextWindow.WriteLine("Hello " + name)
```

Wir speichern den Benutzernamen in der Variablen **name**. Später grüßen wir den Benutzer unseres Programms mit „Hello“. Nehmen wir einmal an, dass es mehr als einen Benutzer gibt, und zwar fünf. Wie können wir alle fünf Namen speichern? Hier ist eine Möglichkeit:

```
TextWindow.Write("User1, enter name: ")
name1 = TextWindow.Read()
TextWindow.Write("User2, enter name: ")
name2 = TextWindow.Read()
TextWindow.Write("User3, enter name: ")
name3 = TextWindow.Read()
TextWindow.Write("User4, enter name: ")
name4 = TextWindow.Read()
TextWindow.Write("User5, enter name: ")
name5 = TextWindow.Read()

TextWindow.Write("Hello ")
```

```
TextWindow.Write(name1 + ", ")
TextWindow.Write(name2 + ", ")
TextWindow.Write(name3 + ", ")
TextWindow.Write(name4 + ", ")
TextWindow.WriteLine(name5)
```

So sieht das Programm bei der Ausführung aus:

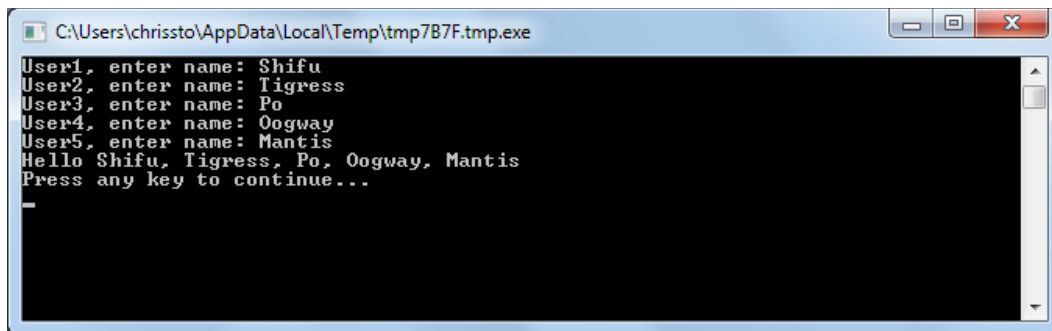


Abbildung 49 – Hier wird noch kein Array verwendet

Das könnte man jedoch auch einfacher haben. Der Computer ist schließlich dafür da, sich wiederholende Aufgaben zu erledigen, also warum sollten wir denselben Programmcode für alle zusätzlichen Benutzer diverse Male wiederholen? Wir möchten den Namen aller Benutzer einfach mit derselben Variablen speichern und abrufen. Das ginge zum Beispiel mit der **For**-Schleife, die wir bereits durchgenommen haben. Hier kommt uns das Array zur Hilfe.

Was ist ein Array?

Ein Array ist eine besondere Sorte von Variablen, die mehr als einen Wert zur gleichen Zeit speichern kann. Anstelle die fünf Benutzernamen **name1**, **name2**, **name3**, **name4** und **name5** separat erstellen zu müssen, um fünf verschiedene Namen speichern zu können, genügt lediglich **name** in Kombination mit einem Index. Zum Beispiel können **name[1]**, **name[2]**, **name[3]**, **name[4]** und **name[5]** jeweils einen Wert speichern. Die Zahlen 1, 2, 3, 4 und 5 sind in diesem Fall Arrayindizes.

Obwohl **name[1]**, **name[2]**, **name[3]**, **name[4]** und **name[5]** so aussehen, als ob es verschiedene Variablen wären, handelt es sich lediglich um eine Variable. Der Vorteil davon ist, dass man den Index mithilfe einer weiteren Variablen festlegen kann, was uns dann den Zugriff auf Arrays innerhalb von Schleifen ermöglicht.

Schreiben wir also unser Programm diesmal mit Arrays.

```
For i = 1 To 5
    TextWindow.Write("User" + i + ", enter name: ")
    name[i] = TextWindow.Read()
```

```

EndFor

TextWindow.Write("Hello ")
For i = 1 To 5
    TextWindow.Write(name[i] + ", ")
EndFor
TextWindow.WriteLine("")

```

Das ist doch schon viel lesbarer, oder? Die erste fettgedruckte Zeile speichert einen Wert im Array und die zweite liest dann diesen Wert. Der Wert in **name[1]** ist unabhängig vom Wert in **name[2]**. Also könnte man **name[1]** und **name[2]** auch wie zwei verschiedene Variablen betrachten.

```

C:\Users\christo\AppData\Local\Temp\tmpA6E4.tmp.exe
User1, enter name: Shifu
User2, enter name: Tigress
User3, enter name: Po
User4, enter name: Oogway
User5, enter name: Mantis
Hello Shifu, Tigress, Po, Oogway, Mantis.
Press any key to continue...

```

Abbildung 50 – Mit Arrays

Das oben abgebildete Programm liefert fast dasselbe Resultat wie das Programm ohne Arrays, mit Ausnahme des Kommas am Ende von *Mantis*. Wir können das korrigieren, indem wir die **Print**-Schleife wie folgt ändern:

```

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

```

Indizieren eines Arrays

Im vorherigen Programm haben wir gelernt, dass wir Zahlen als Indizes einsetzen können, um Werte in einem Array zu speichern und aus dem Array abzurufen. Diese Indizes sind nicht nur auf Zahlen

beschränkt, also kann auch sehr gut mit Text verwendet werden. Im folgenden Programm rufen wir Benutzerinformationen ab und zeigen die gewünschten Daten an.

```
TextWindow.Write("Enter name: ")
user["name"] = TextWindow.Read()
TextWindow.Write("Enter age: ")
user["age"] = TextWindow.Read()
TextWindow.Write("Enter city: ")
user["city"] = TextWindow.Read()
TextWindow.Write("Enter zip: ")
user["zip"] = TextWindow.Read()

TextWindow.Write("What info do you want? ")
index = TextWindow.Read()
TextWindow.WriteLine(index + " = " + user[index])
```

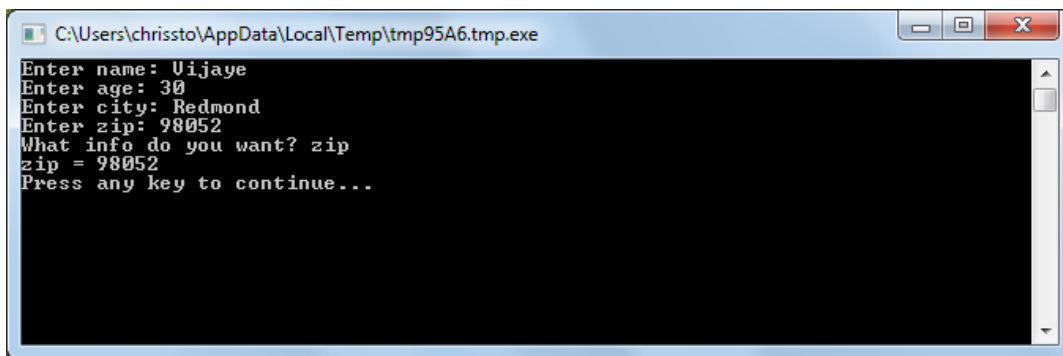


Abbildung 51 – Verwenden von nicht-numerischen Indizes

Mehrere Dimensionen

Gehen wir einmal davon aus, dass wir den Namen und die Telefonnummer aller unserer Freunde in einer Art elektronischem Telefonbuch speichern wollen. Wie sähe ein solches Programm aus?

In diesem Fall haben wir es mit zwei Indizes zu tun, einer sogenannten Arraydimension.

Sagen wir einmal, wir wollen jeden unserer Freunde mit einem Spitznamen identifizieren. Das wäre dann der erste Index in unserem Array. Der zweite Index enthält **name** und **phone number**.

Das sähe so aus:

```
friends["Rob"]["Name"] = "Robert"
```

Wie bei Variablen wird auch bei Arrayindizes die Groß- und Kleinschreibung nicht berücksichtigt.

```
friends["Rob"]["Phone"] = "555-6789"

friends["VJ"]["Name"] = "Vijaye"
friends["VJ"]["Phone"] = "555-4567"

friends["Ash"]["Name"] = "Ashley"
friends["Ash"]["Phone"] = "555-2345"
```

Da wir zwei Indizes in unserem Array **friends** verwenden, handelt es sich um ein zweidimensionales Array.

Wenn das Programm erstellt ist, kann der Spitzname eines Freundes oder einer Freundin eingegeben werden, worauf der komplette Name und die Telefonnummer angezeigt werden. Hier ist das Codebeispiel:

```
friends["Rob"]["Name"] = "Robert"
friends["Rob"]["Phone"] = "555-6789"

friends["VJ"]["Name"] = "Vijaye"
friends["VJ"]["Phone"] = "555-4567"

friends["Ash"]["Name"] = "Ashley"
friends["Ash"]["Phone"] = "555-2345"

TextWindow.Write("Enter the nickname: ")
nickname = TextWindow.Read()

TextWindow.WriteLine("Name: " + friends[nickname]["Name"])
TextWindow.WriteLine("Phone: " + friends[nickname]["Phone"])
```

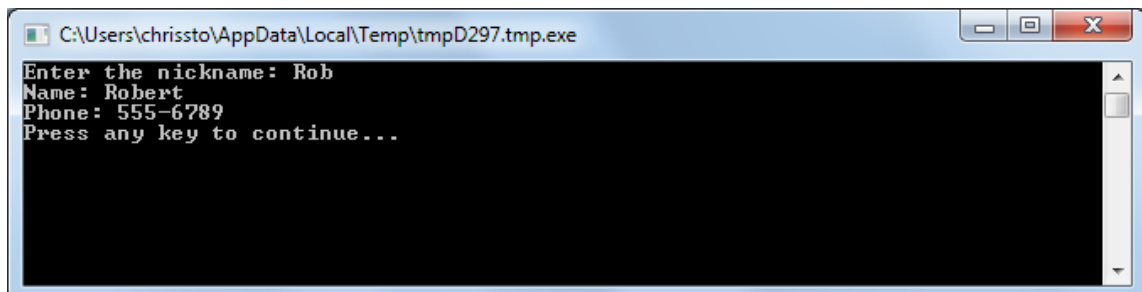


Abbildung 52 – Ein einfaches Telefonbuch

Arrays und Raster

Mehrdimensionale Arrays eignen sich bestens für das Darstellen von Rastern oder Tabellen. Raster haben Zeilen und Spalten, die gut in ein zweidimensionales Array passen. Hier ist ein einfaches Programm, das Kästchen in einem Raster zeichnet:

```
rows = 8
columns = 8
size = 40

For r = 1 To rows
  For c = 1 To columns
    GraphicsWindow.BrushColor = GraphicsWindow.GetRandomColor()
    boxes[r][c] = Shapes.AddRectangle(size, size)
    Shapes.Move(boxes[r][c], c * size, r * size)
  EndFor
EndFor
```

Dieses Programm fügt Kästchen hinzu und positioniert sie in Form eines Rasters der Ausmaße 8x8. Die Kästchen werden außerdem in einem Array gespeichert. Mit diesem Vorgang kann man schnell auf die Kästchen zugreifen, sollte man sie für ein anderes Programm benötigen.

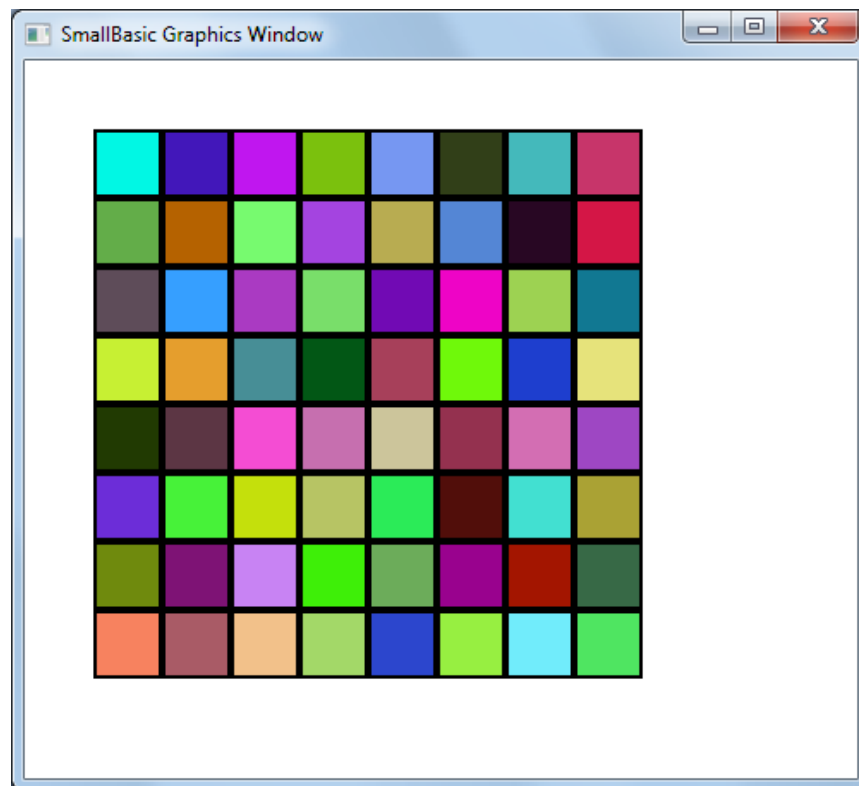


Abbildung 53 – Kästchen im Raster

Zum Beispiel kann man mit dem folgenden Codebeispiel erreichen, dass die Kästchen in die linke obere Ecke des Rasters wandern. Das Beispiel kann einfach an den vorherigen Programmcode drangehängt werden.

```
For r = 1 To rows
  For c = 1 To columns
    Shapes.Animate(boxes[r][c], 0, 0, 1000)
    Program.Delay(300)
  EndFor
EndFor
```

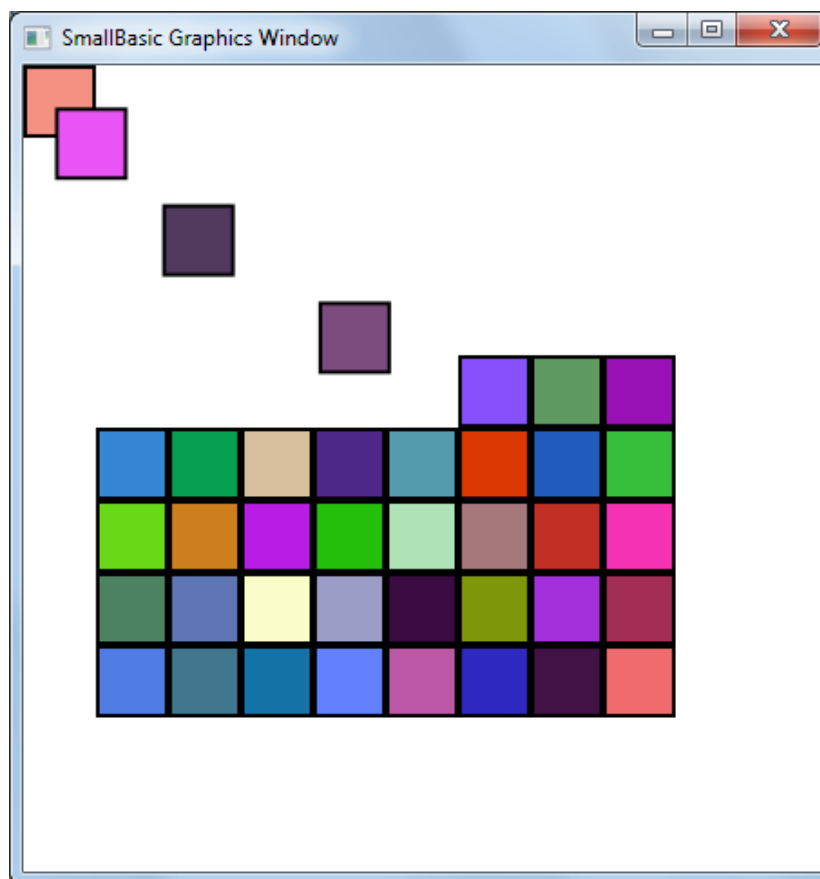


Abbildung 54 – Anordnen von Kästchen im Raster

Ereignisse und Interaktivität

In den ersten beiden Kapiteln haben wir Objekte eingeführt, die Eigenschaften und Methoden haben. Zusätzlich zu Eigenschaften und Methoden können auch **Ereignisse** mit Objekten verwendet werden. Ereignisse sind wie Signale, die zum Beispiel als Folge von Handlungen des Benutzers gegeben werden, wie Mausbewegungen oder Mausklicks. Man könnte sagen, dass Ereignisse das Gegenteil von Anweisungen sind. Eine Anweisung rufen Sie als Programmierer auf, um den Computer etwas ausführen zu lassen. Bei einem Ereignis teilt Ihnen der Computer mit, dass eine Handlung eingetreten ist.

Wofür sind Ereignisse nützlich?

Ereignisse helfen dabei, ein Programm interaktiv zu gestalten. Wenn Sie einem Benutzer erlauben wollen, mit Ihrem Programm zu interagieren, setzen Sie Ereignisse ein. Sagen wir, Sie schreiben ein Tic-Tac-Toe-Spiel. Sie wollen doch dem Benutzer erlauben, das Spiel allein zu spielen? Sie erhalten die Eingaben des Benutzers in Ihrem Programm durch Ereignisse. Nicht verzagen, falls sich das kompliziert anhört. Wir werden uns ein sehr einfaches Beispiel anschauen, das das Arbeiten mit Ereignissen veranschaulicht.

Hier ein sehr einfaches Programm, das lediglich aus einer Anweisung und einem Unterprogramm besteht. Das Unterprogramm benutzt die **ShowMessage**-Methode des Grafikfensters, um dem Benutzer ein Nachrichtenfenster anzuzeigen.

```
GraphicsWindow.MouseDown = OnMouseDown

Sub OnMouseDown
    GraphicsWindow.ShowMessage("You Clicked.", "Hello")
EndSub
```

Der interessante und bemerkenswerte Teil im Programm ist die Zeile, in der wir das **MouseDown**-Ereignis des GraphicsWindow-Objektes zuweisen. Sie bemerken, dass MouseDown einer Eigenschaft ähnlich sieht, mit der Ausnahme, dass wir ihm nicht einen Wert, sondern den Namen des Unterprogrammes **OnMouseDown** zuweisen. Das ist das Besondere an Ereignissen – wenn das Ereignis geschieht, wird automatisch das Unterprogramm aufgerufen. In diesem Fall wird das Unterprogramm **OnMouseDown** jedes Mal aufgerufen, wenn der Benutzer mit der Maus auf das Grafikfenster klickt. Starten Sie das Programm, und probieren Sie es aus. Jedes Mal, wenn Sie mit der Maus ins Grafikfenster klicken, sehen Sie ein Meldungsfenster, wie es im untenstehenden Bild gezeigt wird:



Abbildung 55 – Antwort auf ein Ereignis

Diese Art von Ereignisbehandlung ist sehr leistungsfähig und erlaubt das Erstellen von kreativen Programmen, die oft *ereignisgesteuerte* Programme genannt werden.

Sie können das **OnMouseDown**-Unterprogramm ändern, so dass es etwas anderes tut, als ein Meldungsfenster zu öffnen. Zum Beispiel können Sie dort große blaue Punkte zeichnen, wo der Benutzer klickt. Hier ist ein Beispiel:

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

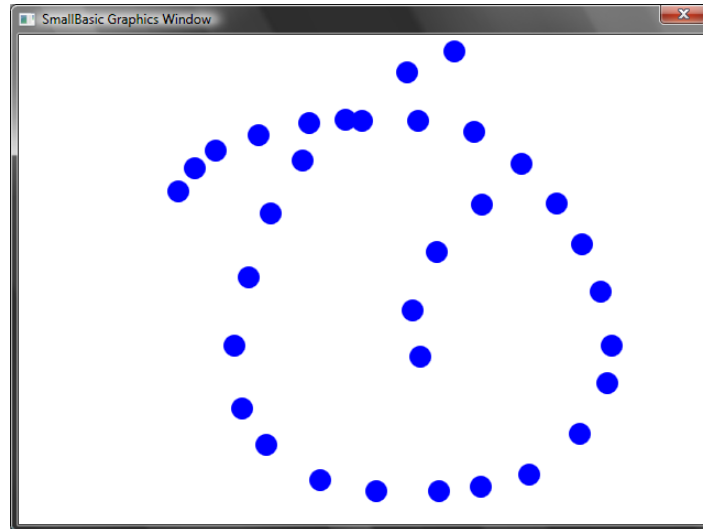


Abbildung 56 – Behandlung des Maus-Unten-Ereignisses

Beachten Sie bitte im obenstehenden Programm, dass wir **MouseX** und **MouseY** benutzt haben, um die Koordinaten der Maus zu erhalten. Mit den Mauskoordinaten als Mittelpunkt zeichnen wir dann einen Kreis.

Behandlung mehrerer Ereignisse

Sie können in ihrem Programm beliebig viele Ereignisse erlauben. Sogar Unterprogramme können mehrere Ereignisse handhaben. Sie können jedoch jedes Ereignis nur einmal verwenden. Wenn Sie zwei Unterprogramme einem Ereignis zuweisen, erhält das zweite Priorität. Um das zu illustrieren, nehmen wir das letzte Beispiel und fügen ein Unterprogramm hinzu, das Tastendrucke behandelt. Das neue Unterprogramm soll die Farbe des Pinsels ändern, so dass Sie bei jedem Mausklick einen andersfarbigen Punkt erhalten.

```
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
```

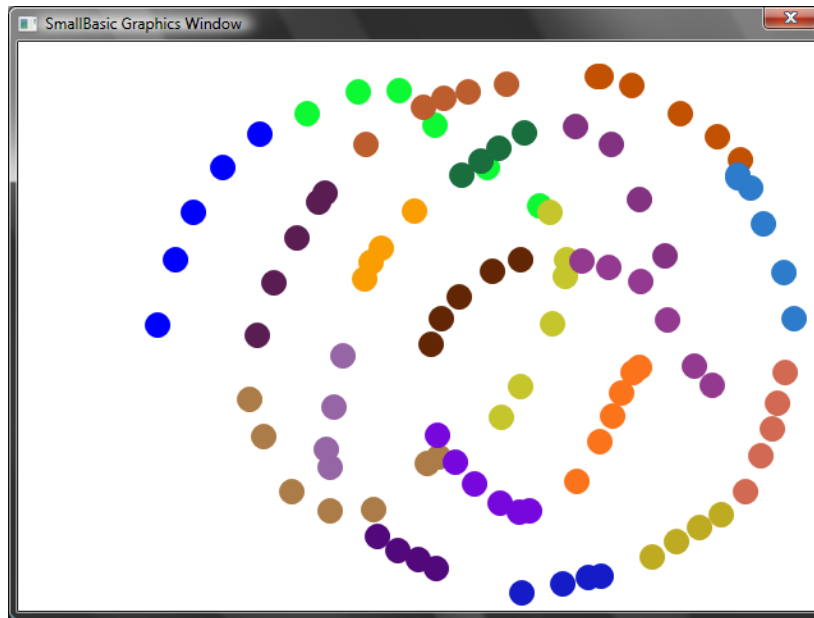


Abbildung 57 – Mehrere Ereignisse

Wenn Sie dieses Programm laufen lassen und aufs Fenster klicken, bekommen Sie einen blauen Punkt. Wenn Sie nun eine Taste drücken und wieder klicken, bekommen Sie einen andersfarbigen Punkt. Wenn Sie eine Taste drücken, wird das Unterprogramm **OnKeyDown** ausgeführt, welches die Pinselfarbe in eine zufällig ausgewählte Farbe ändert. Wenn Sie danach mit der Maus klicken, wird ein Kreis in der neu festgesetzten Farbe gezeichnet, was viele bunte Punkte ergibt.

Ein Zeichenprogramm

Mit Ereignissen und Unterprogrammen bewaffnet können wir nun ein Programm schreiben, das den Benutzer im Fenster zeichnen lässt. Es ist überraschend einfach, ein solches Programm zu schreiben, besonders, wenn wir das Problem in kleine Teile zerlegen. Lassen Sie uns als ersten Schritt ein Programm schreiben, das es dem Benutzer erlaubt, die Maus beliebig im Grafikfenster zu bewegen, wobei eine Mausspur hinterlassen wird.

```
GraphicsWindow.MouseMove = OnMouseMove

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

Wenn Sie das Programm starten, beginnt die erste Zeile an der oberen linken Ecke des Fensters (0,0). Wir können das Problem lösen, indem wir das **MouseDown**-Ereignis behandeln und die Werte für **prevX** und **prevY** erfassen, wenn das Ereignis eintritt .

Zudem brauchen wir die Spur nur, solange der Benutzer die Maustaste gedrückt hat, sonst sollten wir die Linie nicht zeichnen. Dafür benutzen wir die **IsLeftButtonDown**-Eigenschaft des **Mouse**-Objektes. Diese Eigenschaft gibt an, ob die linke Maustaste gedrückt ist oder nicht. Wenn Sie den Wert **True** hat, zeichnen wir die Linie.

```
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
```

Beispiele , die Spaß machen

Baumfraktal

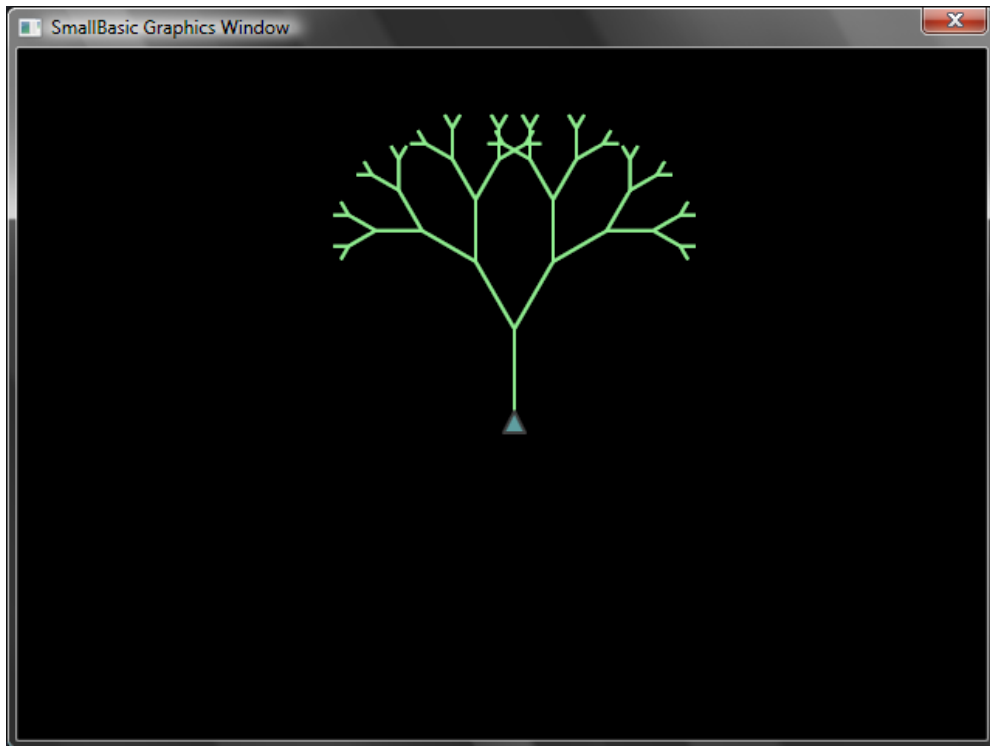


Abbildung 58 – Die Schildkröte zeichnet ein Baumfraktal

```

angle = 30
delta = 10
distance = 60
Turtle.Speed = 9
GraphicsWindow.BackgroundColor = "Black"
GraphicsWindow.PenColor = "LightGreen"
DrawTree()

Sub DrawTree
  If (distance > 0) Then
    Turtle.Move(distance)
    Turtle.Turn(angle)

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

    Turtle.Move(-distance)
  EndIf
EndSub

```


Fotos von Flickr



Abbildung 59 – Bilder von Flickr herunterladen

```
GraphicsWindow.BackgroundColor = "Black"  
GraphicsWindow.MouseDown = OnMouseDown  
  
Sub OnMouseDown  
    pic = Flickr.GetRandomPicture("mountains, river")  
    GraphicsWindow.DrawResizedImage(pic, 0, 0, 640, 480)  
EndSub
```

Dynamischer Bildschirmhintergrund

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

Konsolenspiel

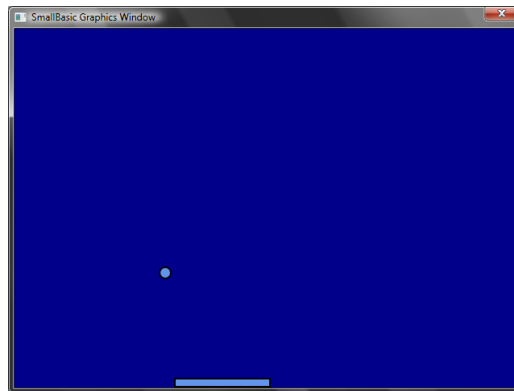


Abbildung 60 – Konsolenspiel

```
GraphicsWindow.BackgroundColor = "DarkBlue"
paddle = Shapes.AddRectangle(120, 12)
ball = 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

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

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

If (y < gh) Then
    Goto RunLoop
EndIf

GraphicsWindow.ShowMessage("You Lose", "Paddle")

Sub OnMouseMove
    paddleX = GraphicsWindow.MouseX
    Shapes.Move(paddle, paddleX - 60, GraphicsWindow.Height - 12)
EndSub

```

Hier ist eine Liste der von Small Basic unterstützten namentlich bezeichneten Farben, sortiert nach der Basisfarbe.

Rote Farben

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

Pinke Farben

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

PaleVioletRed	#DB7093
---------------	---------

Orange Farben

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

Gelbe Farben

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

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

Violette Farben

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

Grüne Farben

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

Blaue Farben

Aqua	#00FFFF
Cyan	#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

Braune Farben

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

Helle Farben

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

Graue Farben

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