Begin Microsoft Small Basic: Hoofdstuk 4: Ontwerp van klein basisprogramma, invoermethoden

Bekijken en bekijken

U zou nu redelijk vertrouwd moeten zijn met het maken en uitvoeren van eenvoudige Small Basic-programma's. In deze les blijven we nieuwe small basic-onderwerpen leren om onze programmeerkennis uit te breiden. We zullen kijken naar enkele ideeën voor programmaontwerp, enkele wiskundige functies en naar manieren om input te krijgen van gebruikers van uw programma's. En we bouwen een spaarcalculatorprogramma.

Programma Ontwerp

Je staat op het punt om te beginnen met het ontwikkelen van vrij gedetailleerde programma's met behulp van Small Basic. We geven je programma's om te bouwen en misschien heb je ideeën voor je eigen programma's. Hoe dan ook, het is leuk en opwindend om te zien dat ideeën eindigen als computerprogramma's. Maar voordat u aan een programma begint, is het een goed idee om een beetje tijd te besteden aan het nadenken over wat u probeert te doen. Dit idee van een goed **programma-ontwerp** bespaart u veel tijd en resulteert in een veel beter programma.

Een goed programma-ontwerp is niet echt moeilijk. Het belangrijkste idee is om een programma te maken dat gemakkelijk te gebruiken, gemakkelijk te begrijpen en vrij van fouten is. Dat is logisch, nietwaar? Besteed wat tijd aan het nadenken over alles wat u wilt dat uw programma doet. Welke informatie heeft het programma nodig? Welke informatie bepaalt de computer? Bepaal welke programmeerstappen u moet volgen om de gewenste taken uit te voeren.

Dit hoofdstuk is een bewerking van het boek *BEGINNING Microsoft Small Basic* van Philip Conrod en Lou Tylee.

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

Maak de Small Basic-code in uw methoden leesbaar en gemakkelijk te begrijpen. Dit maakt het werk om latere wijzigingen aan te brengen (en u zult wijzigingen aanbrengen) veel gemakkelijker. Volg geaccepteerde programmeerregels - u leert deze regels naarmate u meer leert over Small Basic. Zorg ervoor dat er geen fouten in uw programma staan. Dit lijkt misschien een voor de hand liggende verklaring, maar veel programma's zijn niet foutloos.

Het belang van deze paar uitspraken over programma-ontwerp is op dit moment misschien niet erg logisch, maar ze zullen dat wel doen. Het eenvoudige idee is om een nuttig, duidelijk geschreven, foutloos programma te maken dat gemakkelijk te gebruiken en gemakkelijk te wijzigen is. Zorgvuldig plannen en vooruit plannen helpt je dit doel te bereiken. Voor elk programma dat in deze cursus is gebouwd, zullen we proberen u enig inzicht te geven in het ontwerpproces van het programma. We zullen altijd proberen uit te leggen waarom we doen wat we doen bij het bouwen van een programma. En we zullen altijd proberen alle overwegingen die we maken op te sommen.

Een andere overweging bij het ontwerpen van programma's is om uw programma altijd in fasen op te bouwen. Probeer niet je hele Small Basic-programma te bouwen en het allemaal in één keer te testen. Dit vergroot alleen maar de kans op fouten. We raden u aan om uw programma altijd in fasen op te bouwen. Schrijf een beetje code. Test dat kleine beetje code om er zeker van te zijn dat het correct werkt. Voeg langzaam steeds meer code toe. Voer elke codetoevoeging uit en test deze. Ga door met deze aanpak totdat uw programma is voltooid. U zult merken dat deze "go slow" -benadering voor het maken van een Small Basic-programma uw programmeertaak veel eenvoudiger zal maken. Probeer het eens in programma's die we bouwen.

Small Basic - De tweede les

We hebben in de laatste les veel Small Basic behandeld. Dit was nodig om je kennis te laten maken met veel basisbegrippen, zodat je je eerste programma kon schrijven. In deze kortere tweede les kijken we naar enkele wiskundige functies.

Wiskundige functies

In klasse 3 zagen we de kleine basis rekenkundige operatoren waarmee we de basisprincipes van optellen, aftrekken, vermenigvuldigen en delen kunnen uitvoeren. Net als andere computerprogrammeertalen heeft Small Basic ook de mogelijkheid om zeer krachtige wiskundige berekeningen uit te voeren. De ingebouwde wiskundige **functies** van Small Basic (ook wel **methoden** genoemd) worden vaak gebruikt in deze berekeningen. We hebben een aantal van deze functies gebruikt in het Sub Sandwich-programma (**Math.Floor** en **Math.Remainder**)

We verwachten niet dat je een wiskundig genie bent om door deze noten te werken, dus we zullen alleen naar drie wiskundige functies kijken. Ten eerste, wat is een **functie** eigenlijk? Een functie is een routine die een bepaalde waarde voor u berekent, gegeven bepaalde informatie. De indeling voor het gebruik van een functie is:

```
FunctionValue = FunctionName(ArgumentList)
```

FunctionName is de naam van de functie en **ArgumentList** is een lijst met waarden (**invoer**, gescheiden door komma's) die aan de functie worden verstrekt zodat deze zijn werk kan doen. In deze toewijzingsinstructie gebruikt FunctionName de waarden in ArgumentList om een resultaat te berekenen en dat resultaat toe te wijzen aan de variabele die we **FunctionValue hebben genoemd**.

Hoe weet je welke kleine wiskundige basisfuncties er bestaan, wat voor soort informatie ze bieden en wat de argumenten zijn? Controleer verschillende Small Basic referenties en de Microsoft Small Basic website. Of typ gewoon het woord **Wiskunde** in het venster Kleine basiscode en het helpgebied geeft alle functies weer:

Kies vervolgens een functienaam en het helpvenster vertelt u hoe u deze moet gebruiken. Hier is informatie over de eerste functie waar we naar kijken (**Abs**):

Zoals gezegd zullen we hier drie wiskundige functies bekijken. De methoden die wiskundige functies ondersteunen, worden geïmplementeerd in de klasse Small Basic met de naam **Math**. Om naar een bepaalde functie te verwijzen, schrijf je **dus Wiskunde**, dan een punt, dan de functienaam.

De eerste functie die we onderzoeken is de **absolute waardefunctie**. In de wiskunde is de absolute waarde het positieve deel van een getal. De small basic functie is:

Math.Abs(Argument)

waar **argument** getal is waar we de absolute waarde van willen hebben. Het argument kan een **int**- of **dubbeltype** zijn en de geretourneerde waarde is hetzelfde type als het argument. Enkele voorbeelden:

Example	**Result**
Math.Abs(7)	7
Math.Abs(-11)	11
Math.Abs(-3.14)	3.14
Math.Abs(72.1)	72.1

Have you ever needed the **square root** of a number? A square root is a number that when multiplied by itself gives you the original number. For example, the square root of 4 is 2, since 2 times 2 is four. There’s a button on your calculator (**√**) that will do this for you. In Small Basic, the square root function is:

Math.SquareRoot(Argument)

where **Argument** is number we want the square root of. The argument must be a non-negative number. Some examples:

Example	Result
Math.SquareRoot(4)	2
Math.SquareRoot(36)	6
Math.SquareRoot(72.1)	8.491

The last function we will use in this class is the **exponentiation** method. In exponentiation, a number is multiplied times itself a certain number of times. If we multiply a number by itself 4 times, we say we raise that number to the 4^{th} power. The Small Basic function used for exponentiation is:

Math.Power(Argument1, Argument2)

Notice the **Power** function has two arguments. **Argument1** is the number we are multiplying times itself **Argument2** times. In other words, this function raises Argument1 to the Argument2 power. Some examples:

Example	Result
Math.Power(4, 2)	16
Math.Power(-3, 3)	-27
Math.Power(10, 2)	10000

In each example here, the arguments have no decimal parts. We have done this to make the examples clear. You are not limited to such values. It is possible to use this function to compute what happens if you multiply 7.654 times itself 3.16 times!! (The answer is 620.99, by the way.)

For the more mathematically inclined reader, you should know that there are many more Small Basic functions available for your use. You might want to look into using them. There are trigonometric functions and inverse trig functions, functions to convert from radians to degrees and vice versa, functions to find extreme values, functions for rounding, logarithm and inverse logarithm functions and a built-in value for pi. (If none of this means anything to you, don't worry – we won't be using them in this class).

Program Input Methods

In the example (Sub Sandwich Program) we built in the last class, we established variable values in code and ran the program to see the results. The results were printed by the Small Basic output method **WriteLine**. If we want to use different values, we need to change the code and rerun. This is a pain! It would be nice, in such a program, to allow a user to type in values while the program is running and have the computer do the computations based on the inputs. This way no code changes or recompiling would be needed to get a new answer. We need such capabilities in our programs.

The Small Basic language has two general methods that supports typed input. The methods are part of the **TextWindow** object we have been using. The first input method (**ReadNumber**) allows reading numeric

(integer and floating) inputs. Its usage is:

ReturnedValue = TextWindow.ReadNumber()

where **ReturnedValue** is the number input by the user.

The other method (**Read**) returns a line of text (string information) input by the user. Its usage is:

ReturnedValue = TextWindow.Read()

where **ReturnedValue** is the text input by the user.

With either of these methods, the user types the requested input and presses the **<Enter>** key to have the computer accept the value.

Input Methods Example

Start **Small Basic**. Click the **New Program** button in the toolbar. A blank editor will appear. Immediately save the program as **InputExample** in a folder of your choice. Type these lines in the editor

```
1. TextWindow.WriteLine("What is your age?")
2. UserAge = TextWindow.ReadNumber()
3. TextWindow.WriteLine("You input " + UserAge)
```

In this code, we ask for a user to input their age, then write it in the text window.

Run the program (click the **Run** button or press <**F5**>). You should see:

Notice how the prompt appears. Type in a value and press <**Enter**>. Once the number is input, it is assigned to the variable **UserAge** and the **WriteLine** method displays the entered value:

Notice the input value appears on a separate line after the prompting question. Most times, you would like this value to be on the same line as the prompt. This can be done by using a different **TextWindow** method. The **WriteLine** method appends a new line character to the output text, hence subsequent information goes to that new line. The Write method does not begin a new line. Modify the first line of code with the shaded changes (change the WriteLine method to Write and add a space after the question mark):

TextWindow.Write("What is your age? ")

Rerun the program. Now when you type your age, it appears next to the prompting question:

Run the program again and try to enter non-numeric characters – you won't be able to. The **ReadNumber** method only accepts numeric data (the digits 0 through 9, a leading minus sign, or a single decimal point).

Now, let's test the **Read** method (to input a string of text). Add these three lines of code that ask for a user's name in a manner similar to requesting the age:

```
1. TextWindow.Write("What is your name? ")
2. UserName = TextWindow.Read()
3. TextWindow.WriteLine("You input " + UserName)
```

Run the program again. Type your age, press <Enter> and you will see the prompt asking for your name:

Enter a string (any characters at all can be typed) and press <Enter> to see:

It seems the input methods are working just fine. Did you notice how building a program in stages (adding a few lines of code at a time) is good? Always follow such a procedure. Before leaving this example and building another program, let's take a quick look at one other useful Small Basic concept. In the text window above, it would be nice if there was a blank line between each input request. This just makes your output appear a little cleaner, a quality of a well designed Small Basic program. To insert a blank line in the output, just use a **WriteLine** method with a blank argument:

TextWindow.WriteLine("")

Add the shaded line to the current code:

```
1. TextWindow.Write("What is your age? ")
2. UserAge = TextWindow.ReadNumber()
3. TextWindow.WriteLine("You input " + UserAge)
4. TextWindow.WriteLine("")
5. TextWindow.Write("What is your name? ")
6. UserName = TextWindow.Read()
7. TextWindow.WriteLine("You input " + UserName)
```

Run the program again and answer the prompts. Notice the new blank line.

This program has been saved in the **InputExample** folder in the course programs folder (**\BeginSB\BSB Code**).

Program – Savings Calculator

In this program, we will build a savings account calculator. We will input how much money we can put into an account each month and the number of months we put money in the account. The program will then compute how much we saved. This program is saved in the **Savings** folder in the course programs folder (**\BeginSB \BSB Code**).

Program Design

The steps needed to do this calculation are relatively simple:

1. Obtain an amount for each month's deposit.
2. Obtain a number of months.
3. Multiply the two input numbers together.
4. Output the product, the total savings.

We will use the **ReadNumber** method to get user input. The **WriteLine** and **Write** methods will be used to output the savings amount. We'll throw in an additional step to ask for the user's name (an example of using the **Read** method).

Program Development

Start **Small Basic**. Click the **New Program** button in the toolbar. A blank editor will appear. Immediately save the program as **Savings** in a folder of your choice.

First, type the following header information and code that adds a window title:

```
'
' Savings Program
' Beginning Small Basic
'
TextWindow.Title = "Savings Calculator"
```

We will use four variables in this program: one for the user's name (**YourName**), one for the deposit amount (**Deposit**), one for the number of months (**Months**) and one for the total amount (**Total**). Type these lines to initialize the variables:

```
YourName = ""
Deposit = 0.0
```

```
3. Months = 0
4. Total = 0.0
```

Now, we start the code, using the steps outlined under Program Design. At any time, after typing some code, you might like to stop and run just to see if things are going okay. That is always a good approach to take. First, ask the user his/her name using this code:

```
1. ' ask user name
2. TextWindow.Write("Hello, what is your name? ")
3. YourName = TextWindow.Read()
```

Next, determine how much will be deposited in the savings account each month:

```
1. TextWindow.WriteLine("")
2. ' get deposit amount
3. TextWindow.Write("How much will you deposit each month? ")
4. Deposit = TextWindow.ReadNumber()
```

Notice the insertion of a blank line before printing the prompt. Finally, obtain the number of months:

```
1. TextWindow.WriteLine("")
2. ' get number of months
3. TextWindow.Write("For how many months? ")
4. Months = TextWindow.ReadNumber()
```

With this information, the total deposit can be computed and displayed using a WriteLine method:

```
1. TextWindow.WriteLine("")
2. ' compute and display total
3. Total = Deposit * Months
4. TextWindow.WriteLine(yourName + ", after " + months + " months, you will have $" + t
5. TextWindow.WriteLine("")
```

Save your program by clicking the **Save** button.

The finished code in the Small Basic editor should appear as:

```
1. '
2. ' Savings Program
3. ' Beginning Small Basic
```

```
'
TextWindow.Title = "Savings Calculator"
' initialize variables
YourName = ""
Deposit = 0.0
Months = 0
Total = 0.0

' ask user name
TextWindow.Write("Hello, what is your name? ")
YourName = TextWindow.Read()
TextWindow.WriteLine("")
' get deposit amount
TextWindow.Write("How much will you deposit each month? ")
Deposit = TextWindow.ReadNumber()
TextWindow.WriteLine("")
' get number of months
TextWindow.Write("For how many months? ")
Months = TextWindow.ReadNumber()
TextWindow.WriteLine("")
' compute and display total
Total = Deposit * Months
TextWindow.WriteLine(yourName + ", after " + months + " months,
you will have $" + total + " in your savings.")
TextWindow.WriteLine("")
```

Run the Program

Run your program. If the program does not run successfully, try to find out where your errors are using any error messages that may appear. We will cover some possible errors in the next class.

When the program runs successfully, you will see:

Type in your name, a deposit amount and a number of months. Your total will be given to you in a nicely formatted string output. Notice how the name, deposit, months and total are all put together (concatenated) in a single sentence, along with a dollar sign ($). Make sure the answer is correct. Remember, a big step in program design is making sure your program works correctly! If you say you want to save 200 dollars a month for 10 months and your computer program says you will have a million dollars by that time, you should know something is wrong somewhere!

When I tried the program, I got:

Notice if I deposit 403.52 (you don't, and can't, enter the dollar sign) for 11 months, the program tells me I will have $4438.72 in my savings account.

Dit programma lijkt misschien niet zo ingewikkeld. En dat is het niet. We hebben immers maar twee getallen bij elkaar vermenigvuldigd. Maar het programma demonstreert stappen die in elk Small Basic-programma worden gebruikt. Waardevolle ervaring is opgedaan met het herkennen van het lezen van invoerwaarden, het uitvoeren van de wiskunde om de gewenste resultaten te verkrijgen en die resultaten aan de gebruiker uit te voeren.

Andere dingen om te proberen

De meeste spaarrekeningen leveren rente op, dat wil zeggen dat de bank je daadwerkelijk betaalt om ze je geld te laten gebruiken. Dit spaarrekeningprogramma heeft de rente genegeerd. Maar het is vrij eenvoudig om de nodige wijzigingen aan te brengen om rekening te houden met interesse - de wiskunde is net iets ingewikkelder. We zullen u de stappen geven, maar niet laten zien hoe, om uw programma te wijzigen. Probeer het eens als je wilt:

- Definieer een variabele **rente** om de jaarlijkse spaarrente op te slaan. Rentetarieven zijn zwevende decimale getallen.
- Voeg extra overzichten toe zodat de gebruiker een rentevoet kan invoeren.
- Wijzig de code om Interest in computing **Total** te gebruiken. De code voor die berekening is (maak je klaar - het ziet er rommelig uit):

  Totaal = 1200 * (Aanbetaling * (Math.Power((1 + Rente / 1200), Maanden) - 1) / Rente)

  Zorg ervoor dat u dit allemaal op één regel typt - zoals vaak gebeurt, heeft de tekstverwerker het laten lijken alsof het op twee staat. Zoals we al zeiden, is dit een behoorlijk rommelige uitdrukking, maar het is een goede gewoonte om haakjes en een wiskundige functie (**Macht) te** gebruiken. Het getal '1200' wordt hier gebruikt om de rente om te rekenen van een jaarwaarde naar een maandwaarde.

Voer nu het gewijzigde programma uit. Typ waarden in voor storting, maanden en rente. Zorg ervoor dat je redelijke antwoorden krijgt. (Als u een stortingswaarde van 300, een maandwaarde van 14 en een rentewaarde van 6,5 gebruikt, moet het totale antwoord $ 4351,13 zijn. Merk op dat je $ 4200 zonder rente zou hebben, dus dit is logisch). Sla uw programma op.

Ik vertelde een leugentje, ik kreeg geen $4351,13 in het bovenstaande voorbeeld met rente. Ik kreeg eigenlijk $ 4351.127205217292307692307692 3!!!:

Ik rondde het antwoord af. In dergelijke gevallen moet het nummer alleen worden weergegeven met twee cijfers achter de komma. Het is mogelijk om dit te doen met behulp van Small Basic, maar buiten het bereik van onze discussie op dit moment.

Samenvatting

Merk op dat de programma's een beetje gedetailleerder worden naarmate je meer Leert Small Basic. In deze les leerde je over het juiste programmaontwerp, wiskundige functies en hoe je invoermogelijkheden kunt toevoegen aan je Small Basic-programma's. Je hebt een klein spaarrekeningprogramma gebouwd. En een belangrijk concept om te onthouden terwijl u doorgaat met deze cursus, is om altijd te proberen uw programma's een paar regels code tegelijk te bouwen. Een goede mantra is 'code een beetje, test een beetje'. U zult met deze aanpak minder fouten in uw programma's introduceren.