MAGBOOK

BUILD A GAME IN 30 MINUTES!

Coding for Kids Scratch

Step-by-step friendly advice

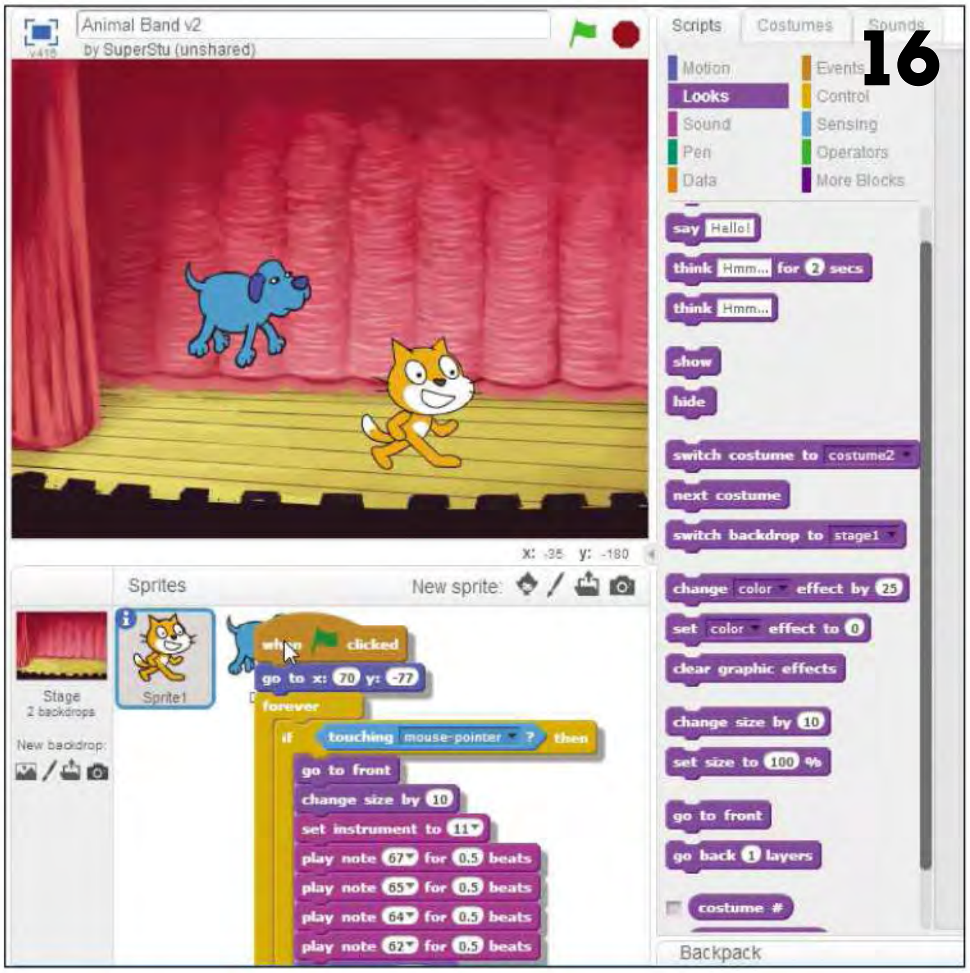
Animate stories

Make your own racing game

Create your own quiz

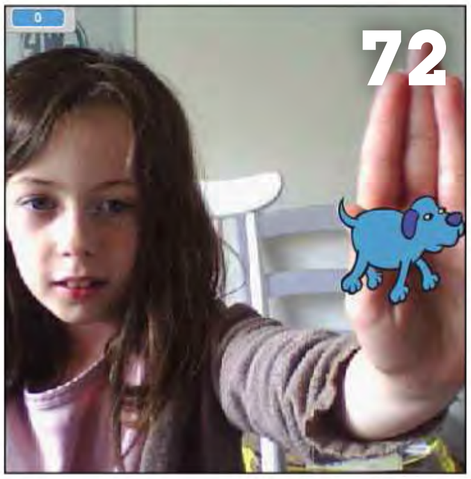
Chapter 3

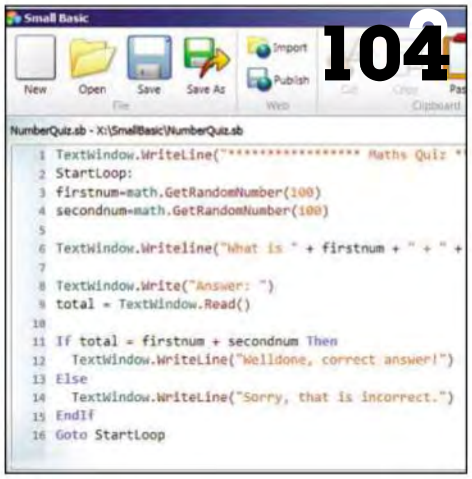
Contents



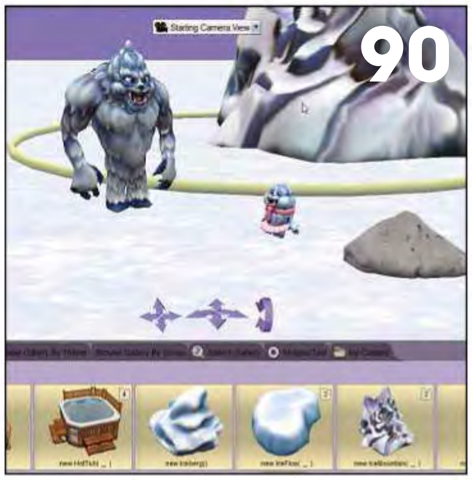
```
when [flag] clicked
go to x: 70 y: -77
forever
  if <touching mouse-pointer ?> then
    go to front
    change size by 10
    set instrument to 11
    play note 67 for 0.5 beats
    play note 65 for 0.5 beats
    play note 64 for 0.5 beats
    play note 62 for 0.5 beats
```

```
say Hello!
think Hmm... for 2 secs
think Hmm...
show
hide
switch costume to costume2
next costume
switch backdrop to stage1
change color effect by 25
set color effect to 0
clear graphic effects
change size by 10
set size to 100 %
go to front
go back 1 layers
costume #
```





```
TextWindow.WriteLine("***************** Maths Quiz *
StartLoop:
firstnum=math.GetRandomNumber(100)
secondnum=math.GetRandomNumber(100)

TextWindow.Writeline("What is " + firstnum + " + " +

TextWindow.Write("Answer: ")
total = TextWindow.Read()

If total = firstnum + secondnum Then
  TextWindow.WriteLine("Welldone, correct answer!")
Else
  TextWindow.WriteLine("Sorry, that is incorrect.")
EndIf
Goto StartLoop
```



Section 3 BASIC basics

Learning Scratch is just the start of your journey into code. For our next steps, we're going to have to dive into BASIC. This is a textual programming language with a long and proud history, and one that many novice programmers have cut their coding teeth on. It's easy to learn and follow, but has enough power to build professional apps. While learning BASIC, you'll see how many of the basic coding elements in Scratch are mirrored in a textual language, and you'll also get to grips with syntax, structure and other programming fundamentals. You might not master it all straight away, but you'll be surprised at how easy it becomes with a little practice. Plus, while you're learning the basics, you'll be working on some fun projects that give you scope to try your own ideas.

IN THIS SECTION

Introducing SmallBASIC

If you're ready to move onto a textual programming language, Microsoft's SmallBASIC has you covered with its simplicity and power

WHAT YOU'LL LEARN

- What is SmallBASIC?
- Where you can it
- How you can use it

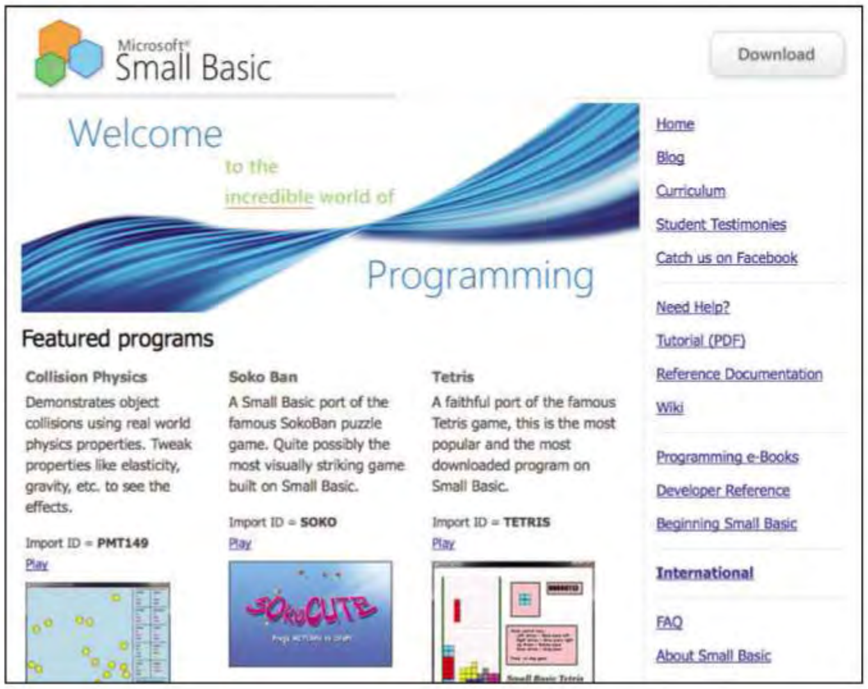
Microsoft SmallBASIC is a simple programming language based on BASIC, which stands for Beginner's All-Purpose Symbolic Instruction Code. BASIC languages have been around for years and come in a range of different varieties, and that largely comes down to how simple yet powerful a language it is. SmallBASIC differs from other forms because it contains just 14 keywords. This makes it easier to learn than other versions, yet you can still make exciting applications and amazing games just by learning a little bit of code.

This killer combination of simplicity and power makes SmallBASIC a good first choice for anyone who's interested in becoming a software developer and wants to go further than a visual programming environment like Scratch. It's ideal for learning the basics of programming, and it's been adopted by many primary and secondary school pupils. It also helps that you can think of SmallBASIC as a cutdown version of Microsoft's Visual Basic, the most popular version of the language, and one still used by many professional programmers today. SmallBASIC and Visual Basic share a similar structure and keywords, so if you master SmallBASIC it's easy to transfer your skills and knowledge to Visual Basic.

Where can you find SmallBASIC?

The easiest answer is to go online and head to smallbasic.com. You need to be careful when searching on the internet, as there's another programming language with exactly the same name. Once you get to the website, you'll find that SmallBASIC has an active online community where you can find help and resources, not to mention applications and games to download. The website also hosts some simple tutorials, as well as more detailed documents explaining every aspect of SmallBASIC coding.

Unlike Scratch, which you access over the internet, SmallBASIC needs to be downloaded and installed. You'll find a button for downloading the package at the top right of the website. After you've installed the program, take some time to have a look around the website, as there are many links to support you in getting started or pages



Featured programs

Collision Physics
Demonstrates object collisions using real world physics properties. Tweak properties like elasticity, gravity, etc. to see the effects.
Import ID = PMT149
Play

Soko Ban
A Small Basic port of the famous SokoBan puzzle game. Quite possibly the most visually striking game built on Small Basic.
Import ID = SOKO
Play

Tetris
A faithful port of the famous Tetris game, this is the most popular and the most downloaded program on Small Basic.
Import ID = TETRIS
Play

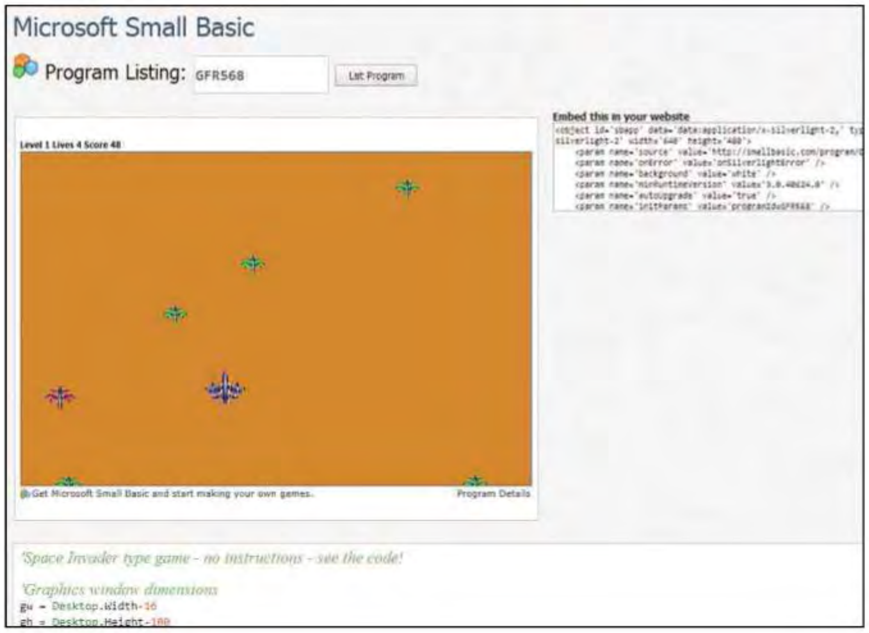
You can download SmallBASIC from the website, then access a huge range of tutorials and sample programs.

“Make exciting apps and amazing games just by learning a little bit of code”

that tell you how to get things done. The PDF tutorial, step-by-step learning curriculum and reference wiki can all be very useful when you’re trying to learn, although we’re going to give you plenty of help in these pages as well.

How can you use it?

Unlike Scratch, where you drag blocks of code from the Blocks Palette to build your program, the instructions in SmallBASIC – as in most high-end languages – need to be typed manually. This not only means that you’ll have to know the keywords, but also that any spelling mistakes or missed symbols will result in your program not compiling and therefore problems with the end result. If there are any errors, the SmallBASIC Editor will tell you what line and position the errors are on, but it’s down to you to solve them. You’ll also get a short technical explanation of the problem, but the wording can be hard to decipher if you’re new to programming.



Microsoft Small Basic

Program Listing: GFR568 List Program

Level 1 Lives 4 Score 48

Get Microsoft Small Basic and start making your own games. Program Details

Embed this in your website

```
<object id='sbapp' data='data:application/x-silverlight-2,' ty
silverlight-2' width='640' height='480'>
    <param name='source' value='http://smallbasic.com/program/
    <param name='onError' value='onSilverlightError' />
    <param name='background' value='white' />
    <param name='minRuntimeVersion' value='3.0.40624.0' />
    <param name='autoUpgrade' value='true' />
    <param name='initParams' value='programId=GFR568' />
```

```
'Space Invader type game - no instructions - see the code!

'Graphics window dimensions
gw = Desktop.Width-16
gh = Desktop.Height-100
```

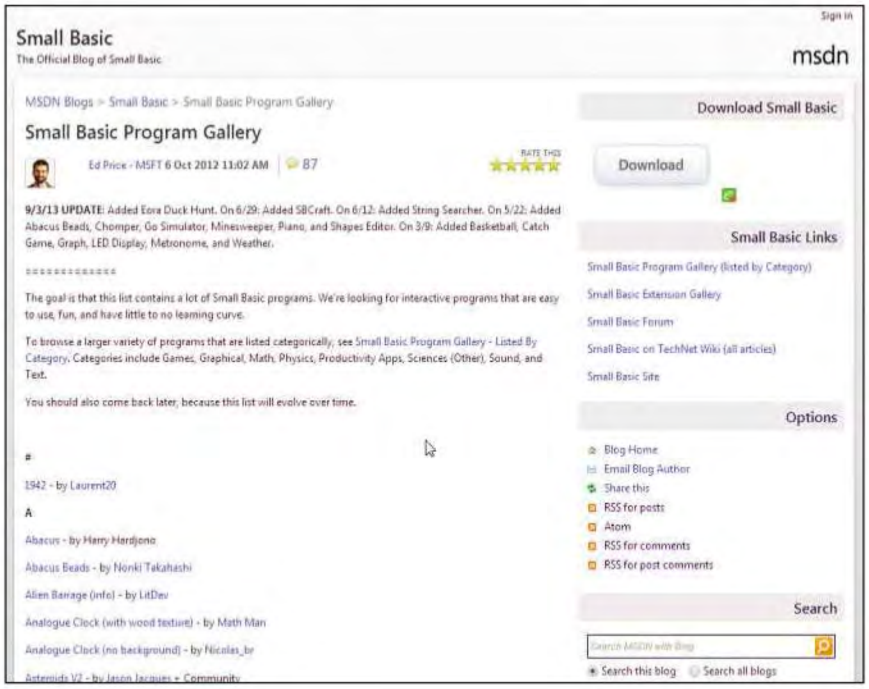
Hundreds of budding programmers are already using SmallBASIC to create and share their own games and apps.

Importance of debugging

Fixing these errors, or ‘debugging’ as programmers like to call it, is a crucial part of coding. You might even find it weirdly satisfying one day. Plus, while this all sounds like a headache, the process really puts you in control. Change your code just a little, and you can make sure the program works the way you want it to, or even make it run more efficiently.

SmallBASIC is classed as a complier language because, when you run the program, the computer will check your code and create a standalone file – a EXE file in Windows – which you can run independently of SmallBASIC on a Windows computer.

Another great feature of SmallBASIC is that, when somebody creates a program or a game, this can be quickly and easily shared with other people online. All you need is an import ID for the program, and you can quickly type in this code. SmallBASIC will download their program for you to test and play, without you having to manually download files and load them yourself.



Small Basic
The Official Blog of Small Basic

Sign in

msdn

MSDN Blogs > Small Basic > Small Basic Program Gallery

Small Basic Program Gallery

Ed Price - MSFT 6 Oct 2012 11:02 AM 87 RATE THIS

9/3/13 UPDATE: Added Eora Duck Hunt. On 6/29: Added SBCraft. On 6/12: Added String Searcher. On 5/22: Added Abacus Beads, Chomper, Go Simulator, Minesweeper, Piano, and Shapes Editor. On 3/9: Added Basketball, Catch Game, Graph, LED Display, Metronome, and Weather.

=============

The goal is that this list contains a lot of Small Basic programs. We're looking for interactive programs that are easy to use, fun, and have little to no learning curve.

To browse a larger variety of programs that are listed categorically, see Small Basic Program Gallery - Listed By Category. Categories include Games, Graphical, Math, Physics, Productivity Apps, Sciences (Other), Sound, and Text.

You should also come back later, because this list will evolve over time.

#

1942 - by Laurent20

A

Abacus - by Harry Hardjono

Abacus Beads - by Nonki Takahashi

Alien Barrage (info) - by LitDev

Analogue Clock (with wood texture) - by Math Man

Analogue Clock (no background) - by Nicolas_br

Asteroids V2 - by Jason Jacques + Community

You can find and download programs from the SmallBASIC Gallery, then look at the source code to see what makes them tick.

SmallBASIC on Facebook

You can also have a look at the SmallBASIC group if you have access to Facebook. It’s a great place to look at what other users around the world have been making. This would also be a great place to talk about and get feedback on all of the programs that you make (www.facebook.com/groups/smallbasic/). ●

Preparing to program

Before you can have fun coding with SmallBASIC, you first need to install the software on your PC, and find your way around its simple interface

WHAT YOU'LL LEARN

- How to install SmallBASIC
- The basics of using SmallBASIC
- What you can do with the SmallBASIC toolbar

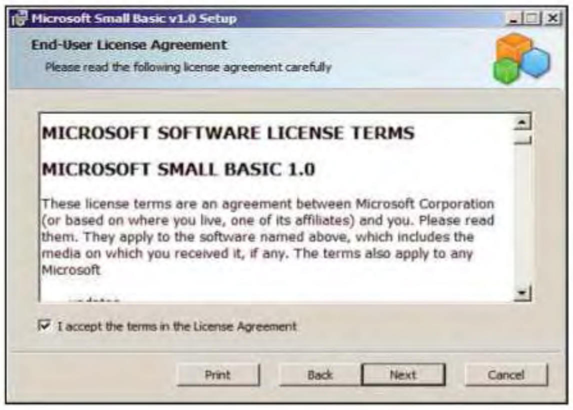
You can download SmallBASIC by clicking on the Download button in the top-right corner of the website's homescreen.

To install it, you'll need a laptop or PC running Windows XP, Vista, 7 or 8, and you'll also need the Microsoft .NET Framework 3.5 installed. If you haven't got this installed already, you can download it for free from www.microsoft.com/en-us/download/details.aspx?id=21.

Once you've downloaded the installer file, you need to run it to install the program on your PC or laptop's local hard drive. The installer file is a small 5.74MB, and even when the core software is installed it will take only 7.40MB. The programs are even tinier, so you don't have to worry about SmallBASIC taking up too much space on your computer.

Install SmallBASIC in three easy steps

STEP 1 Double-click the installer file and the setup wizard will start automatically. Click Next to continue. The end user licence agreement is a standard legal disclaimer for the use of the software. Tick 'I accept the terms in the license agreement', then click Next to continue.


Microsoft Small Basic v1.0 Setup

End-User License Agreement

Please read the following license agreement carefully

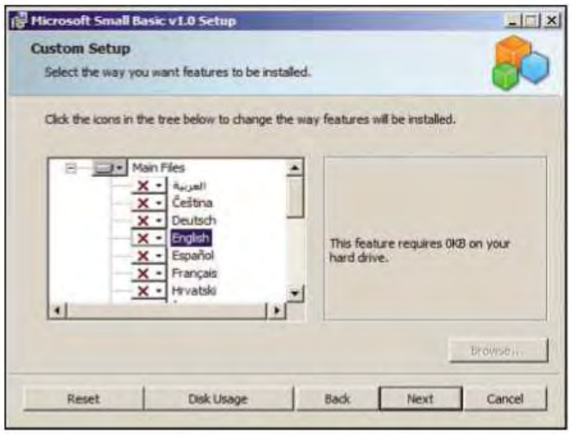
MICROSOFT SOFTWARE LICENSE TERMS

MICROSOFT SMALL BASIC 1.0

These license terms are an agreement between Microsoft Corporation (or based on where you live, one of its affiliates) and you. Please read them. They apply to the software named above, which includes the media on which you received it, if any. The terms also apply to any Microsoft

☑ I accept the terms in the License Agreement

STEP 2 The installer selects all the core files and English language resources needed, but in the unlikely event that you need to install additional language translations, just select them from the tree menu. Click Next to continue.


Microsoft Small Basic v1.0 Setup

Custom Setup

Select the way you want features to be installed.

Click the icons in the tree below to change the way features will be installed.

This feature requires 0KB on your hard drive.

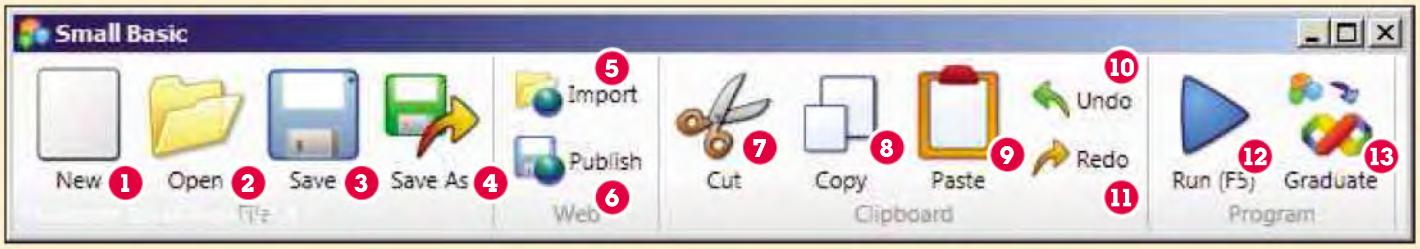
STEP 3 Confirm your installation options by clicking Install, and the program will start the installation. The progress bar will take just a few seconds before it fills, and when it's done

TOP TIP
You don't need to press the Run button to see your SmallBASIC program in action. Just hit the F5 key!

CODING KEYWORDS

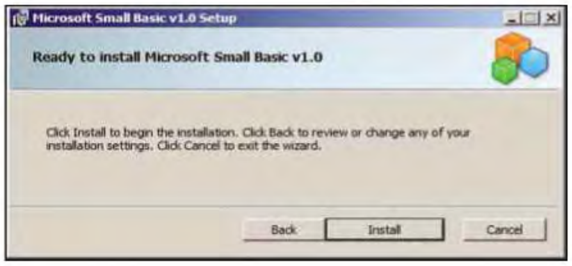
IDE: Integrated Development Environment. A software package and user interface that enable and support coding or programming. One IDE might support several different languages.

THE SMALLBASIC TOOLBAR



The SmallBASIC toolbar at the top of the screen gives you instant, easy access to a range of helpful and important features:

1. **New:** Creates a new SmallBASIC program
2. **Open:** Opens an existing SmallBASIC program
3. **Save:** Saves the current program to its current location
4. **Save as:** Saves the current program to a specific location
5. **Import:** Imports a SmallBASIC program using the Program ID from the online community
6. **Publish:** Publishes your program to the internet for others to download
7. **Cut:** Cuts any highlighted text and saves it to the clipboard, ready to use later
8. **Copy:** Copies any highlighted text to the clipboard, ready to use later
9. **Paste:** Pastes any text you saved to the clipboard earlier into the Editor
10. **Undo:** Takes back your last action
11. **Redo:** Brings back whatever it was you last undid
12. **Run:** Executes your code so you can see and test your program
13. **Graduate:** Converts your SmallBASIC program to a full Visual Basic program, so that you can easily carry on working on it when you're ready to get more advanced


Microsoft Small Basic v1.0 Setup

Ready to install Microsoft Small Basic v1.0

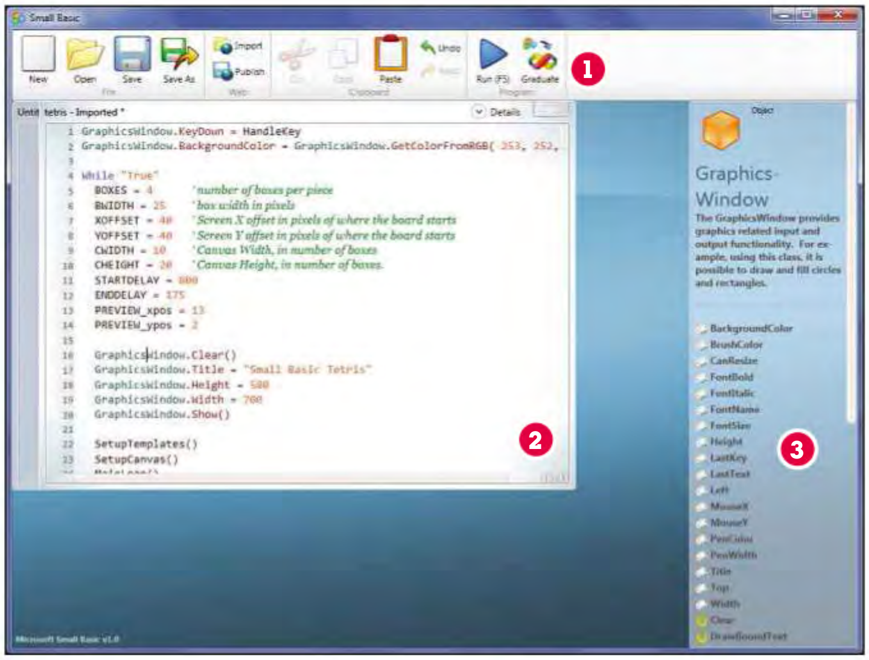
Click Install to begin the installation. Click Back to review or change any of your installation settings. Click Cancel to exit the wizard.

you can complete the installation by clicking the Finish button. Now that you've installed SmallBASIC, you can launch the program by clicking on the Start button, then All Programs, then SmallBASIC in Windows Vista to Windows 7, or by using the Search charm or clicking on the icon in the Apps screen on Windows 8.

Using SmallBASIC

When you first load SmallBASIC, you'll be greeted with a simple interface that's split into three sections. You have a toolbar 1 across the top of the page with the kind of basic features you'll find in any software package, including buttons for Open, Save, Copy and Paste – not to mention the all-important Run to run your program.

The Editor window 2 is the main white area where you'll type your code. This is automatically numbered when you go onto a new line, so if you have an error the debug program will tell you the line followed by the number of characters across.



```
GraphicsWindow.KeyDown = HandleKey
GraphicsWindow.BackgroundColor = GraphicsWindow.GetColorFromRGB( 253, 252,

While "True"
  BOXES = 4          'number of boxes per piece
  BWIDTH = 25        'box width in pixels
  XOFFSET = 40       'Screen X offset in pixels of where the board starts
  YOFFSET = 40       'Screen Y offset in pixels of where the board starts
  CWIDTH = 10        'Canvas Width, in number of boxes
  CHEIGHT = 20       'Canvas Height, in number of boxes.
  STARTDELAY = 800
  ENDDELAY = 175
  PREVIEW_xpos = 13
  PREVIEW_ypos = 2

  GraphicsWindow.Clear()
  GraphicsWindow.Title = "Small Basic Tetris"
  GraphicsWindow.Height = 580
  GraphicsWindow.Width = 700
  GraphicsWindow.Show()

  SetupTemplates()
  SetupCanvas()
```

Graphics-Window

The GraphicsWindow provides graphics related input and output functionality. For example, using this class, it is possible to draw and fill circles and rectangles.

BackgroundColor
BrushColor
CanResize
FontBold
FontItalic
FontName
FontSize
Height
LastKey
LastText
Left
MouseX
MouseY
PenColor
PenWidth
Title
Top
Width
Clear
DrawBoundText

▲ **It shouldn't take you long to get used to SmallBASIC's simple interface.**

The side panel 3 is a help feature that will give you information about the built-in features in SmallBASIC and their properties. This will change depending on what you click or type in the Editor window, so that you get the most helpful information, where available. ●

My first Small-BASIC program

It's time for our first steps into textual programming with what's probably the simplest program you can possibly write

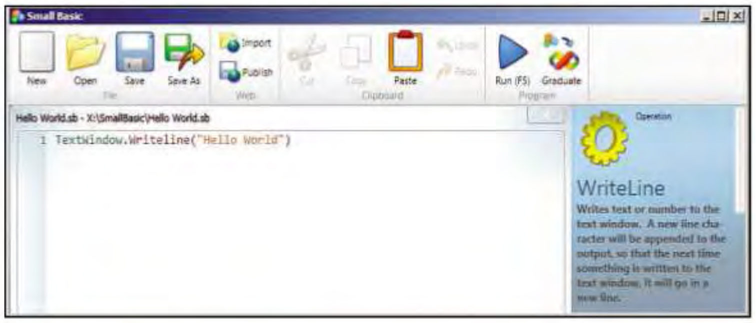
WHAT YOU'LL LEARN

- How to type or paste code into the SmallBASIC Editor
- How to use the TextWindow instruction
- How to change the properties of an object, and so change its looks and behaviour

Let's start by taking SmallBASIC for a test drive. Our first program is a classic exercise in the software development world. Just about everyone does the 'Hello World' program, regardless of what language they're learning. It's the simplest program you can write, as it just displays 'Hello World' back to you in a text window. Still, it's a good way to familiarise yourself with a new environment, and it ensure you can enter text, compile the program and see the results.

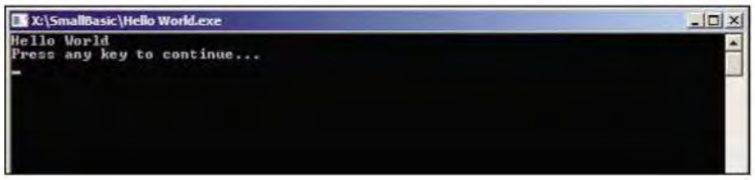
Type the following line of code in the editor window:

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



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

WriteLine
Writes text or number to the text window. A new line character will be appended to the output, so that the next time something is written to the text window, it will go in a new line.



```
Hello World
Press any key to continue...
_
```

▲ 'Hello World' is the simplest program you can write, and a great way to get used to a new coding environment.

To execute your program, you can click the Run button on the toolbar or press F5 on the keyboard.

Congratulations! You've just written your first program, and you should see the text window with your 'Hello world' statement. SmallBASIC will automatically add a 'press any key to continue' message, which will close the text window and end your first program.

Changing properties

```
TextWindow.Foregroundcolor = "Green"
```

When we call an object in SmallBASIC, we can change the properties of it – the bits that alter its appearance or behaviour. In our example, TextWindow is the object and Foregroundcolor is one of the properties. Assigned colours need to be in quotation marks: e.g. "Magenta". As an experiment, try replacing "Green" with another colour.

SmallBASIC, like many other programming languages, goes through and processes your code

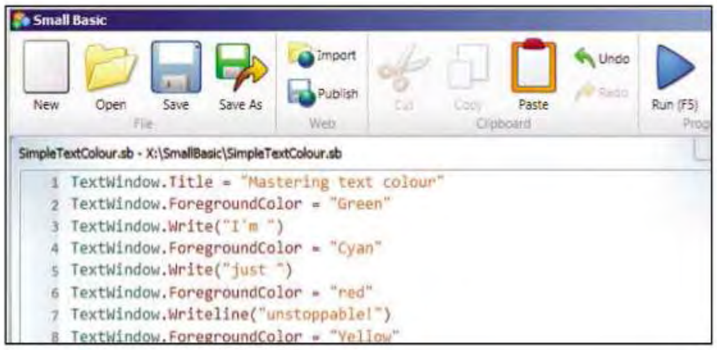
TOP TIP
As SmallBASIC is from Microsoft, some words such as colour are spelt the American way. Don't try changing Color to the standard British Colour or you'll get an error.

EXPERIMENT

Try adding the following code to the start of your program:

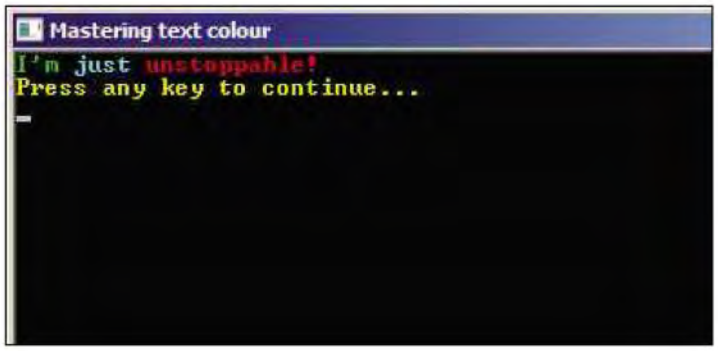
```
TextWindow.Backgroundcolor = "DarkBlue"
```

What effect do you think this will have?



SimpleTextColour.sb - X:\SmallBasic\SimpleTextColour.sb

```
TextWindow.Title = "Mastering text colour"
TextWindow.ForegroundColor = "Green"
TextWindow.Write("I'm ")
TextWindow.ForegroundColor = "Cyan"
TextWindow.Write("just ")
TextWindow.ForegroundColor = "red"
TextWindow.Writeline("unstoppable!")
TextWindow.ForegroundColor = "Yellow"
```



Mastering text colour

```
I'm just unstoppable!
Press any key to continue...
```

▲ **Have a go at adding different colours to your text.**

one line at a time in the order that you typed it, so when we assign a colour to the text window it will stick with that colour until the end of the program or until you change it again within the code.

Now let's try to add different colours to our text by entering the code below:

```
TextWindow.Title = "Mastering text
colour"

TextWindow.ForegroundColor = "Green"

TextWindow.Write("I'm ")

TextWindow.ForegroundColor = "Cyan"

TextWindow.Write("just ")

TextWindow.ForegroundColor = "red"

TextWindow.Writeline("unstoppable!")

TextWindow.ForegroundColor = "Yellow"
```

Again, execute your program with the Run button or F5. As you can see, the order in which you type your code is crucial to getting the desired colour effect. The colour change at the end to yellow is for the 'press any key to continue' text. Without that change, it would still be red from the previous word. In this example, we've used the code TextWindow.Write and TextWindow.Writeline.

Both are used to display text in the window, but TextWindow.Write lets you keep writing on the same line as the previous text, whereas TextWindow.Writeline goes to a new line at the end.

Remember

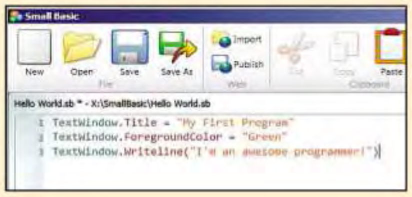
- TextWindow displays just text
- TextWindow.Write adds text to the current line
- TextWindow.Writeline adds text to the current line, but the next output will be on the line below ●

CODING KEYWORDS

Compile: The process where all the raw code for a program and everything it needs to run is put together as a single file that anyone with the right operating system should be able to run. In Windows, this would be a EXE file.

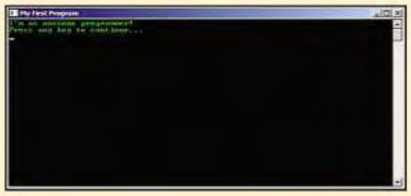
Execute: To launch your program. This is why many Windows' programs have a EXE suffix jammed onto the end.

BASIC ANALYSIS



Hello World.sb * - X:\SmallBasic\Hello World.sb

```
TextWindow.Title = "My First Program"
TextWindow.ForegroundColor = "Green"
TextWindow.Writeline("I'm an awesome programmer!")
```



My First Program

```
I'm an awesome programmer!
Press any key to continue...
```

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

You can see that your code can be split into three distinct sections: TextWindow calls the non-graphical text window object; WriteLine instructs the computer to display text; ("Hello World") gives the computer the words to display.

As software developers, we always have control over what the program does and how it looks, so let's have some fun with our text window. Modify your first program with the following code:

```
TextWindow.Title = "My First Program"

TextWindow.ForegroundColor = "Green"

TextWindow.WriteLine("I'm an awesome programmer!")
```

Execute your program with the Run button on the toolbar or F5 on the keyboard. Notice how your text window has changed from your first program because you've changed the properties of the text window object in the code. You now have a title at the top of the window and we have green text.

Sentence generator

Putting something on the screen is one thing, but programs also need to take some input from the user. We show you how

WHAT YOU'LL LEARN

- How to use TextWindow.Read() to get information from the user
- How to use variables
- How to join words and variables to form sentences
- How to call on SmallBASIC's built-in functions
- To generate random numbers

TOP TIP
It's very important that you use the correct symbols in the Editor. If you miss out a single equals sign or quotation mark, your program will have errors.

In our first basic project we learnt how to display a message to the user, but what if we need to ask the user for some information? Where do we store it and how do we use it? Handling input is a crucial part of programming, whether you're building a word processor or a paint package, or just a website with a form that needs to be filled in. We've already covered how you do this kind of thing in Scratch, but how do you do it in SmallBASIC?

It's easy. Type the following line of code in the editor window:

```
TextWindow.Write("Please enter your name:
")
```

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

```
TextWindow.Writeline("I'm very pleased to
meet you " + name)
```

Now run the program. Notice how it waits until the user has entered a name before it continues with the rest of the code. This is because TextWindow.Read() pauses the program until the Enter key is pressed.

Storing information while our program is running is very important when we want to make more complex programs or have any interaction with the user. We call this storing variables. You might remember working with variables in Scratch. As in Scratch, a variable is a piece of data stored in memory, ready to be recalled by our code. Imagine we had a program that asked for the user's name, age and favourite sports team. If everyone gave the same answers, we could hardwire these answers into the code – we'd call these answers 'constants'. In real-life, however, everyone would give a different set of answers. They'd vary, which is why we call them variables.

Working with numbers

Now that we've mastered text input, let's work with numbers. SmallBASIC has a built-in library of functions that we can call on to use in our own code. These save us precious time, as we don't have to fully code each function and define its purpose.

SmallBASIC has a math function that can perform a variety of maths tasks, such as dealing with degrees and radians, or even finding a number's square root. We're going to use one such function to get a random number.

Type the following code in the editor window:

```
RandomNumber=math.GetRandomNumber(49)
```

Now follow that up with another line:

```
TextWindow.Writeline("Your Random number
is " + RandomNumber)
```

As you can see, we've concatenated our variable into a sentence, just as we did with the name earlier. Execute your program with Run or F5 and you can watch a random number get generated.

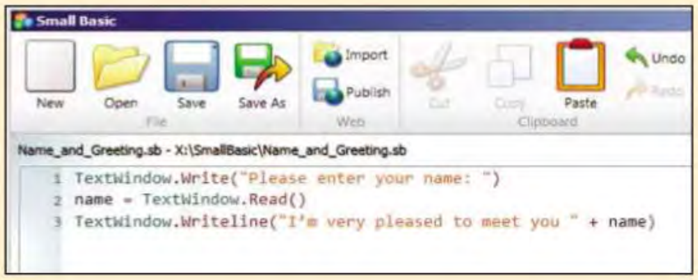
Making a sentence generator

We're going to combine both of our previous two examples to create a program that asks for the

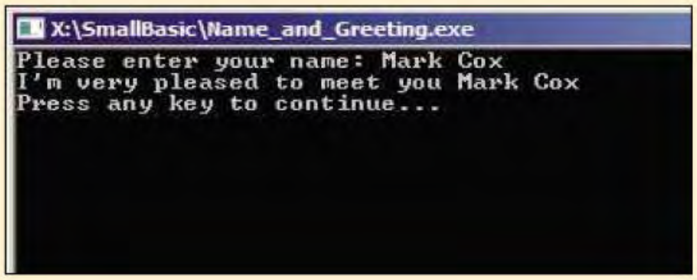
CODING KEYWORDS

Concatenation: joining sentence fragments with variable names to make full sentences.

BASIC ANALYSIS



```
TextWindow.Write("Please enter your name: ")
name = TextWindow.Read()
TextWindow.WriteLine("I'm very pleased to meet you " + name)
```



```
Please enter your name: Mark Cox
I'm very pleased to meet you Mark Cox
Press any key to continue...
```

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

Take a close look at this instruction. Here, the TextWindow.Read() asks the computer to pause and wait for the user to type in something, then store it in memory as a variable called 'name'.

Now take a look at the final line of code: TextWindow.Writeline("I'm very pleased to meet you " + name). Everyone gets the same greeting message. We're displaying our greeting message as normal, but then we recall the stored variable (name) and add it to the end, using the code '+ name'. This is called concatenation. Well done! You can now display text information on the screen as well as get information from a user.

user's name, then generates silly sentences by using random verbs and nouns. The trick to building a more complex program is to break down our problem into sections, and to make sure we get our events in the right order.

For our sentence denerator to work, we need the following events, in order.

1. Display a message asking for a name in a text window
2. Store that name in a variable
3. Generate and store a random verb from a list
4. Generate and store a random noun from a list
5. Display the user's name followed by the stored verb, then the stored noun in a text window.

OK. We're ready to start coding. Create a new page for the Code Editor and enter the following code:

```
TextWindow.Write("Please enter a name: ")
name = TextWindow.Read()
```

```
v=math.GetRandomNumber(4)
verb[1] = "walked "
verb[2] = "answered "
verb[3] = "danced "
verb[4] = "ate "
n=math.GetRandomNumber(4)
noun[1] = "the cake"
noun[2] = "the cat"
noun[3] = "the phone"
noun[4] = "the tango"
TextWindow.Writeline(name + " just " +
verb[v] + noun[n])
```

Run your code by pressing run on the toolbar or tapping F5.

The way we generate the variables for the verb and noun in this example is through a

BASIC TRIVIA

Computers have to be told exactly what to do all the time, so asking them for a truly random number is actually more difficult than you might think. Some languages use the current clock speed of the CPU to determine the number, as it's constantly changing.

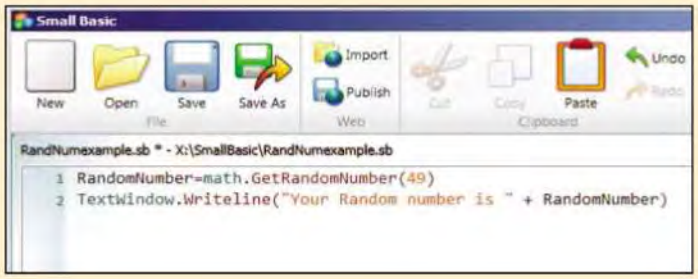
TOP TIP

Planning out code and dividing it into a series of logical steps is an essential part of software development. We call this type of planning Pseudocode.

DIMENSIONS AND ARRAYS

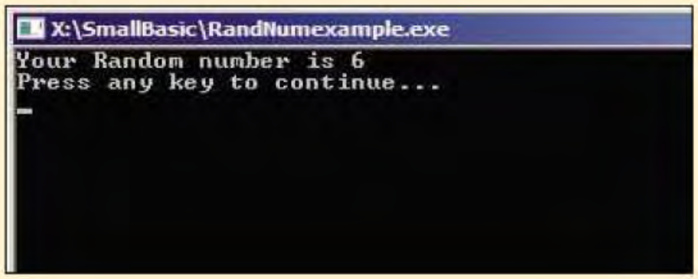
An array that contains just one value is known as a single dimension array. There are more complex arrays that can have multiple values. These have a cool sci-fi name: a multi-dimensional array!

BASIC ANALYSIS



RandNumexample.sb * - X:\SmallBasic\RandNumexample.sb

```
RandomNumber=math.GetRandomNumber(49)
TextWindow.Writeline("Your Random number is " + RandomNumber)
```



X:\SmallBasic\RandNumexample.exe

```
Your Random number is 6
Press any key to continue...
```

`RandomNumber=math.GetRandomNumber(49)`

This code will generate a random number from 1 to 49 and assign it to a variable called RandomNumber. The math.GetRandomNumber function works much like the Pick random operator block in Scratch. You can change the number in the brackets to any positive number, and it will then generate a random number from 1 to that maximum amount.

programming technique called an array. Using this method, you can have the same variable name followed by an index number to store multiple values. Remember the lists in Scratch? Those lists are really simple one-dimensional arrays. In this example, we generate a random number and match it to the index in the array. The syntax of an array is very straightforward in SmallBASIC:

```
VariableName[number] = "Option"
```

Don't forget to use square brackets with an array, not the usual rounded brackets that we use to display text.

We don't have to stop here. We can keep adding more words to the program's vocabulary by adding to the two verb and noun arrays. For example, we could add the following under our list of verbs:

SBSillySentence Code ext

```
verb[5] = "jumped over "
```

```
verb[6] = "high-fived the "
```

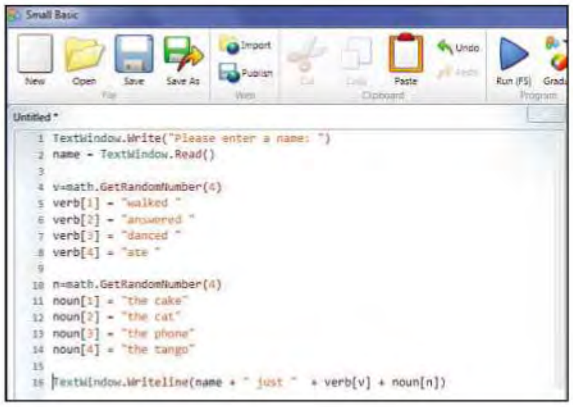
then the following to our list of nouns:

```
noun[5] = "the moon"
noun[6] = "the pandas"
```

All we'd have to do is change the figures in brackets after the two =math.GetRandomNumber instructions to the new total number of nouns or verbs, to read:

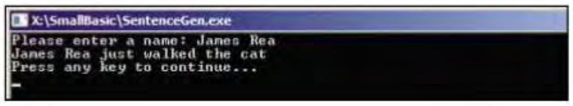
```
n=math.GetRandomNumber(6)
```

```
v=math.GetRandomNumber(6)
```



Untitled *

```
TextWindow.Write("Please enter a name: ")
name = TextWindow.Read()

v=math.GetRandomNumber(4)
verb[1] = "walked "
verb[2] = "answered "
verb[3] = "danced "
verb[4] = "ate "

n=math.GetRandomNumber(4)
noun[1] = "the cake"
noun[2] = "the cat"
noun[3] = "the phone"
noun[4] = "the tango"

TextWindow.Writeline(name + " just "  + verb[v] + noun[n])
```



X:\SmallBasic\SentenceGen.exe

```
Please enter a name: James Rea
James Rea just walked the cat
Press any key to continue...
```

▲ **Our program generates silly sentences by using random verbs and nouns.**

Try changing the verbs and nouns in the list, or start adding a greater selection. Just don't forget to increase the maximum random number in brackets, and do so for the nouns and the verbs! ●

TOP TIP

Take care when giving your variables an assigned name. Don't start them with numbers or symbols like #Name or 99Name. It's often helpful to use several descriptive words joined together like FirstName or UserAge. This can make it much easier to understand and debug your program later on.

BASIC ANALYSIS

```
TextWindow.Writeline(name + " just " +
verb[v] + noun[n])
```

Take a closer look at this instruction. Here, we're displaying the user's name stored at the start, followed by the word "just", then we're calling our verb with a random index number, and the same again with our noun. When we put all of the pieces in the correct order, we get a complete sentence, albeit a very silly one.

Create your own quiz

With a few simple instructions under our belt, we can now start to build something more exciting. Let's kick off with a maths quiz

WHAT YOU'LL LEARN

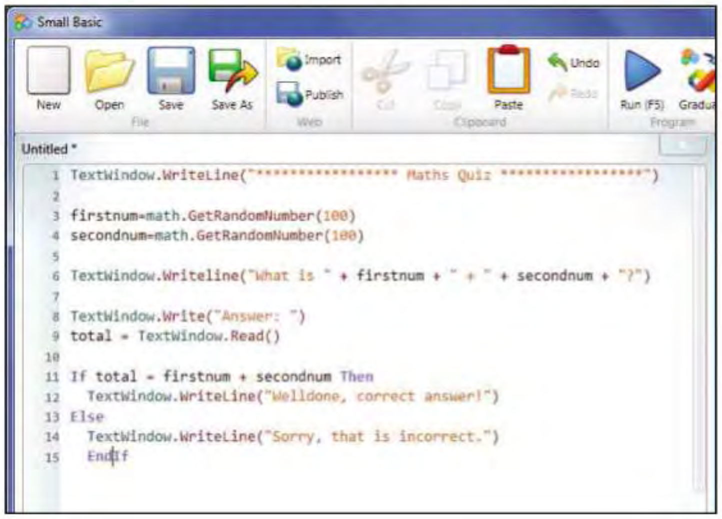
- How to use If statements to check conditions and make selections
- How to use Elseif when there are more than two conditions
- How to use loops to rerun blocks of code
- How to make effective use of subroutines

TOP TIP
Notice how SmallBASIC automatically colours specific words or phrases in different colours. Words such as If, Else, Then and EndIf will be automatically coloured blue, as they're SmallBASIC keywords.

After the last few projects, you should have enough skills to start building more complex programs, but we need to experiment with working with numbers, especially addition, subtraction, multiplication and division. With that in mind, we're going to make a game that manipulates numbers. As always, before we start coding, we need to get a clear idea of what we want to do and the order in which it will happen.

Our first maths program is going to follow this process:

1. Display game title in the text window
2. Generate the first random number between 1 and 100 and save that to a variable
3. Generate the second random number between 1 and 100, and save that to a variable
4. Display the addition equation to the user in the text window with both variables
5. Ask for an answer in the text window
6. Save their response into a variable
7. Check if the answer typed in matches the total of the two numbers
8. Tell the user if they are correct or incorrect



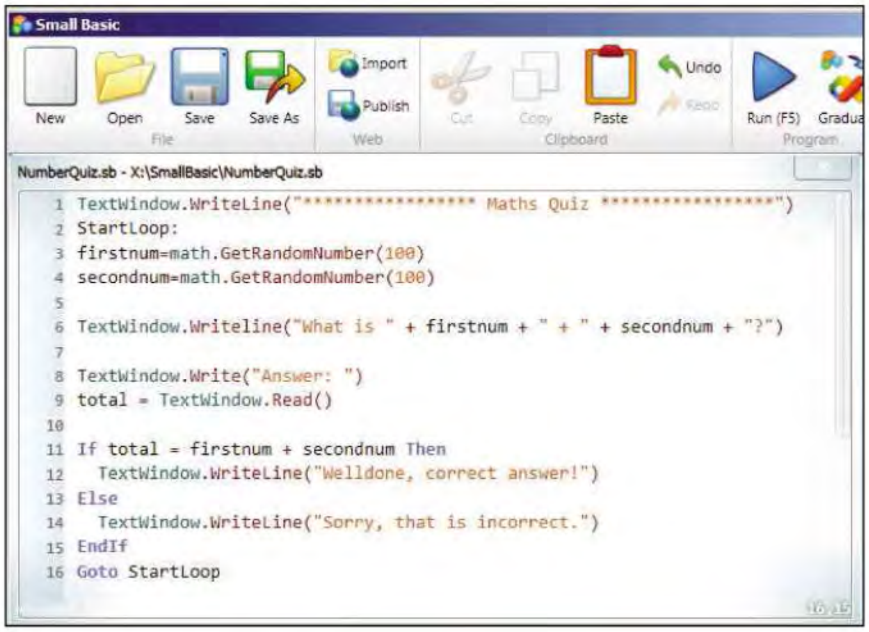
```
TextWindow.WriteLine("***************** Maths Quiz *****************")

firstnum=math.GetRandomNumber(100)
secondnum=math.GetRandomNumber(100)

TextWindow.Writeline("What is " + firstnum + " + " + secondnum + "?")

TextWindow.Write("Answer: ")
total = TextWindow.Read()

If total = firstnum + secondnum Then
  TextWindow.WriteLine("Welldone, correct answer!")
Else
  TextWindow.WriteLine("Sorry, that is incorrect.")
  EndIf
```

▲ Our maths quiz program uses a If statement to check whether the answer is correct or not.

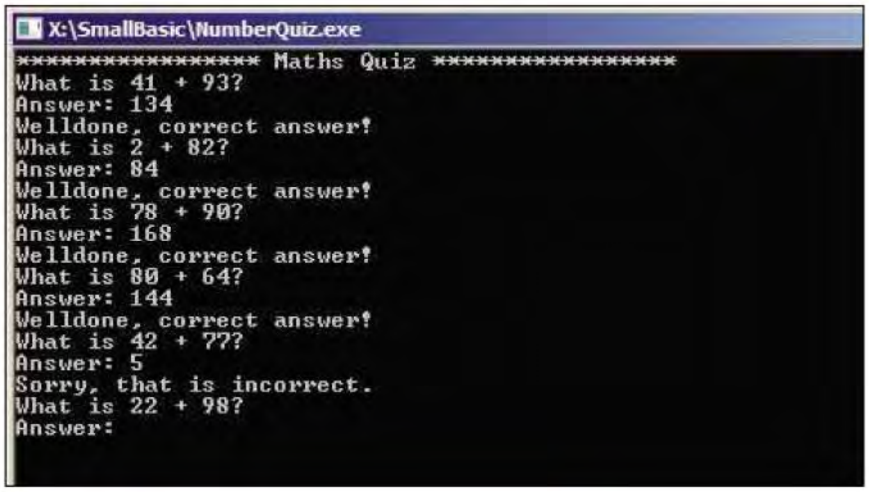
Begin the quiz

Start a new program and type in the following code. When you're done, run the program with F5:

```
TextWindow.WriteLine("*****************
Maths Quiz *****************")

firstnum=math.GetRandomNumber(100)
secondnum=math.GetRandomNumber(100)

TextWindow.Writeline("What is " +
firstnum + " + " + secondnum + "?")

TextWindow.Write("Answer: ")
total = TextWindow.Read()

If total = firstnum + secondnum Then
```



```
 1 TextWindow.WriteLine("***************** Maths Quiz *****************")
 2 StartLoop:
 3 firstnum=math.GetRandomNumber(100)
 4 secondnum=math.GetRandomNumber(100)
 5
 6 TextWindow.Writeline("What is " + firstnum + " + " + secondnum + "?")
 7
 8 TextWindow.Write("Answer: ")
 9 total = TextWindow.Read()
10
11 If total = firstnum + secondnum Then
12   TextWindow.WriteLine("Welldone, correct answer!")
13 Else
14   TextWindow.WriteLine("Sorry, that is incorrect.")
15 EndIf
16 Goto StartLoop
```



```
****************** Maths Quiz ******************
What is 41 + 93?
Answer: 134
Welldone, correct answer!
What is 2 + 82?
Answer: 84
Welldone, correct answer!
What is 78 + 90?
Answer: 168
Welldone, correct answer!
What is 80 + 64?
Answer: 144
Welldone, correct answer!
What is 42 + 77?
Answer: 5
Sorry, that is incorrect.
What is 22 + 98?
Answer:
```

▲ Loops in SmallBASIC work just like 'repeat' and 'forever' blocks in Scratch.

```
  TextWindow.WriteLine("Welldone, correct
answer!")
Else
  TextWindow.WriteLine("Sorry, that is
incorrect.")
EndIf
```

This program uses a conditional statement to check whether the answer is correct. If statements are used frequently in many programming languages to make choices and to interact with the user. In SmallBASIC, the structure of an If statement follows:

```
If condition Then
```

Code if the condition has been met

```
Else
```

Code if the condition has not been met

```
EndIf
```

Look at the If statement we've just typed in:

```
If total = firstnum + secondnum Then
  TextWindow.WriteLine("Welldone, correct
answer!")
Else
  TextWindow.WriteLine("Sorry, that is
incorrect.")
EndIf
```

Our condition in the statement was to check if the total variable is equal to the sum of the first and second variables, which gives us a true or false scenario that works well with mathematical equations, as they've either entered a correct or incorrect answer.

Using loops

One annoying problem we have with our program is that we only get one maths question before the program ends and we have to restart it. Wouldn't it be great if we could tell the computer to go back to the beginning of the code and ask another question automatically? Well, luckily SmallBASIC can do that by using a simple loop. Remember the 'repeat' and 'forever' blocks in Scratch? Loops work in exactly the same way.

Add the following code to line 2 within your existing program, just above generating the first random number and below the title:

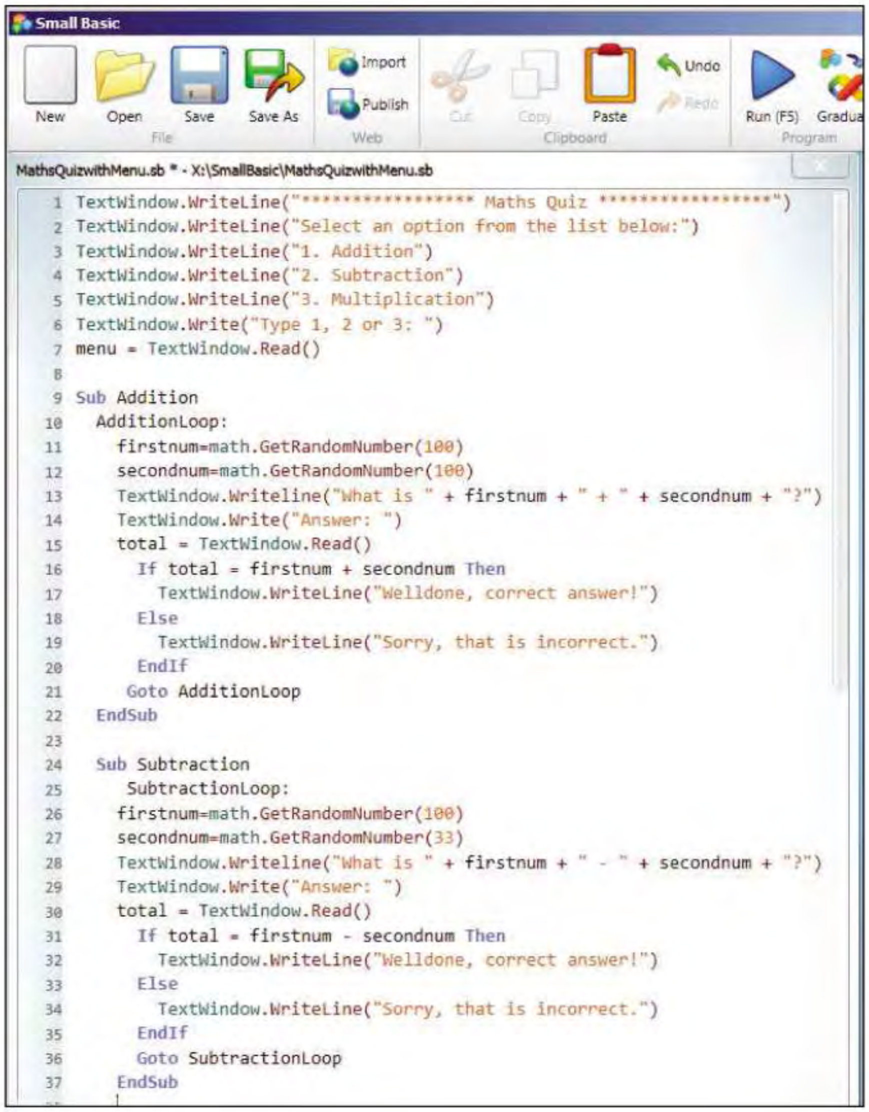
```
StartLoop:
```

Add the following code to the very end of the code after the If statement block:

```
Goto StartLoop
```

What we've done is to create a marker at the top of the program called StartLoop. We can use any words we like for our marker, as long as they're joined together with no spaces and there's a colon at the end. When the code reaches Goto StartLoop at the end, it will head back to the top and rerun the code after our marker to repeat our code. Run your program again and you should get an infinite amount of questions, each one randomly generated.

So far, we've been working only with addition (+), but we can just as easily change this to subtraction, multiplication or division by altering the equation symbol in the first line of the If statement.



MathsQuizwithMenu.sb * - X:\SmallBasic\MathsQuizwithMenu.sb

```
TextWindow.WriteLine("***************** Maths Quiz *****************")
TextWindow.WriteLine("Select an option from the list below:")
TextWindow.WriteLine("1. Addition")
TextWindow.WriteLine("2. Subtraction")
TextWindow.WriteLine("3. Multiplication")
TextWindow.Write("Type 1, 2 or 3: ")
menu = TextWindow.Read()

Sub Addition
  AdditionLoop:
    firstnum=math.GetRandomNumber(100)
    secondnum=math.GetRandomNumber(100)
    TextWindow.Writeline("What is " + firstnum + " + " + secondnum + "?")
    TextWindow.Write("Answer: ")
    total = TextWindow.Read()
      If total = firstnum + secondnum Then
        TextWindow.WriteLine("Welldone, correct answer!")
      Else
        TextWindow.WriteLine("Sorry, that is incorrect.")
      EndIf
     Goto AdditionLoop
  EndSub

  Sub Subtraction
     SubtractionLoop:
    firstnum=math.GetRandomNumber(100)
    secondnum=math.GetRandomNumber(33)
    TextWindow.Writeline("What is " + firstnum + " - " + secondnum + "?")
    TextWindow.Write("Answer: ")
    total = TextWindow.Read()
      If total = firstnum - secondnum Then
        TextWindow.WriteLine("Welldone, correct answer!")
      Else
        TextWindow.WriteLine("Sorry, that is incorrect.")
      EndIf
      Goto SubtractionLoop
    EndSub
```

▲ Here, we've further developed our quiz using blocks of reusable code called subroutines.

To change your program to test subtraction:

```
If total = firstnum – secondnum Then
```

To change your program to test multiplication:

```
If total = firstnum * secondnum Then
```

To change your program to test division:

```
If total = firstnum / secondnum Then
```

EXPERIMENT

Try changing the maximum random number in the brackets to make your game harder or easier.

The only cosmetic change you'll need to make is where you display the equation to the user (line 6), as it will still have the addition symbol between the two numbers.

Just as if we were using spreadsheet software, the symbols for multiplication (*) and division (/) will differ to what you might be used to from your maths lessons at school. Multiplication is normally just an x symbol from the alphabet, but the computer needs the x to form different words, so we can't use that. There's also no standard division symbol on a keyboard, so we use the * and / symbols to represent them.

Developing the quiz

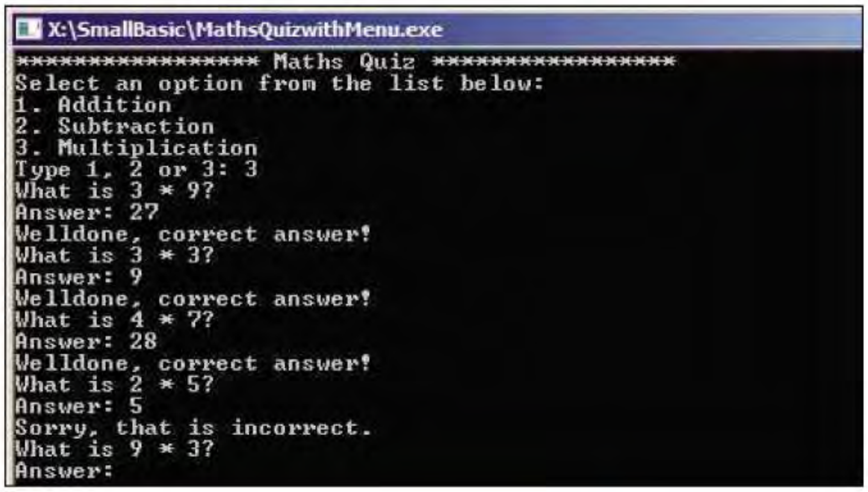
The great thing about programming is that once you have the basic code sorted out, you have a solid foundation on which to add extra features. Here, we need to give our program greater functionality by letting the user decide what area of arithmetic to practise, instead of having to change the source code manually. So, we're going to code a menu system, and the user can select which area of maths to improve.

Our new and improved maths quiz program is going to follow this process:

1. Display game title and menu options in the text window
2. Ask the user for their input
3. If the user selects Addition, call the subroutine code
4. If the user selects Multiplication, call the subroutine code
5. If the user selects Subtraction, call the subroutine code

The trick is to create reusable blocks of code called subroutines. These subroutine blocks will perform one of the arithmetic operations, such as a block for subtraction and one for multiplication, depending on what the user selects from the menu. If you remember the 'make' and 'define' blocks in Scratch, these effectively created subroutines that could be reused again and again. Creating a subroutine in SmallBASIC is very easy. You just need the Keyword sub followed by the name of your subroutine. Any code after this point will be part of that subroutine until you close it with EndSub. To call your subroutine, just use the name of it followed by two brackets: e.g. Addition ().

Modify your existing program or create a new one with the following code:



```
X:\SmallBasic\MathsQuizwithMenu.exe
****************** Maths Quiz ******************
Select an option from the list below:
1. Addition
2. Subtraction
3. Multiplication
Type 1, 2 or 3: 3
What is 3 * 9?
Answer: 27
Welldone, correct answer!
What is 3 * 3?
Answer: 9
Welldone, correct answer!
What is 4 * 7?
Answer: 28
Welldone, correct answer!
What is 2 * 5?
Answer: 5
Sorry, that is incorrect.
What is 9 * 3?
Answer:
```

▲ Our new and improved maths quiz program!

```
TextWindow.WriteLine("*****************
Maths Quiz *****************")

TextWindow.WriteLine("Select an option
from the list below:")

TextWindow.WriteLine("1. Addition")

TextWindow.WriteLine("2. Subtraction")

TextWindow.WriteLine("3. Multiplication")

TextWindow.Write("Type 1, 2 or 3: ")
menu = TextWindow.Read()

Sub Addition
  AdditionLoop:
    firstnum=math.GetRandomNumber(100)
    secondnum=math.GetRandomNumber(100)
    TextWindow.Writeline("What is " +
firstnum + " + " + secondnum + "?")
    TextWindow.Write("Answer: ")
    total = TextWindow.Read()
     If total = firstnum + secondnum Then
       TextWindow.WriteLine("Welldone,
correct answer!")
      Else
       TextWindow.WriteLine("Sorry, that
is incorrect.")
      EndIf
     Goto AdditionLoop
  EndSub

  Sub Subtraction
     SubtractionLoop:
    firstnum=math.GetRandomNumber(100)
    secondnum=math.GetRandomNumber(33)
    TextWindow.Writeline("What is " +
firstnum + " - " + secondnum + "?")
    TextWindow.Write("Answer: ")
    total = TextWindow.Read()
     If total = firstnum - secondnum Then
       TextWindow.WriteLine("Welldone,
correct answer!")
      Else
       TextWindow.WriteLine("Sorry, that
is incorrect.")
      EndIf
      Goto SubtractionLoop
    EndSub

Sub Multiplication
   MultiplicationLoop:
    firstnum=math.GetRandomNumber(12)
    secondnum=math.GetRandomNumber(12)
    TextWindow.Writeline("What is " +
firstnum + " * " + secondnum + "?")
    TextWindow.Write("Answer: ")
    total = TextWindow.Read()
     If total = firstnum * secondnum Then
       TextWindow.WriteLine("Welldone,
correct answer!")
      Else
       TextWindow.WriteLine("Sorry, that
is incorrect.")
      EndIf
      Goto MultiplicationLoop
  EndSub

If menu = 1 Then
    Addition()
  Elseif menu = 2 Then
    Subtraction()
  Elseif menu = 3 Then
    Multiplication()
EndIf
```

TOP TIP

With more advanced subroutines, you can feed data, like variables, in and send data back out to the main program. The same subroutine can even be used by entirely different parts of a program, which is a lot more efficient than writing a new one every time you need it.

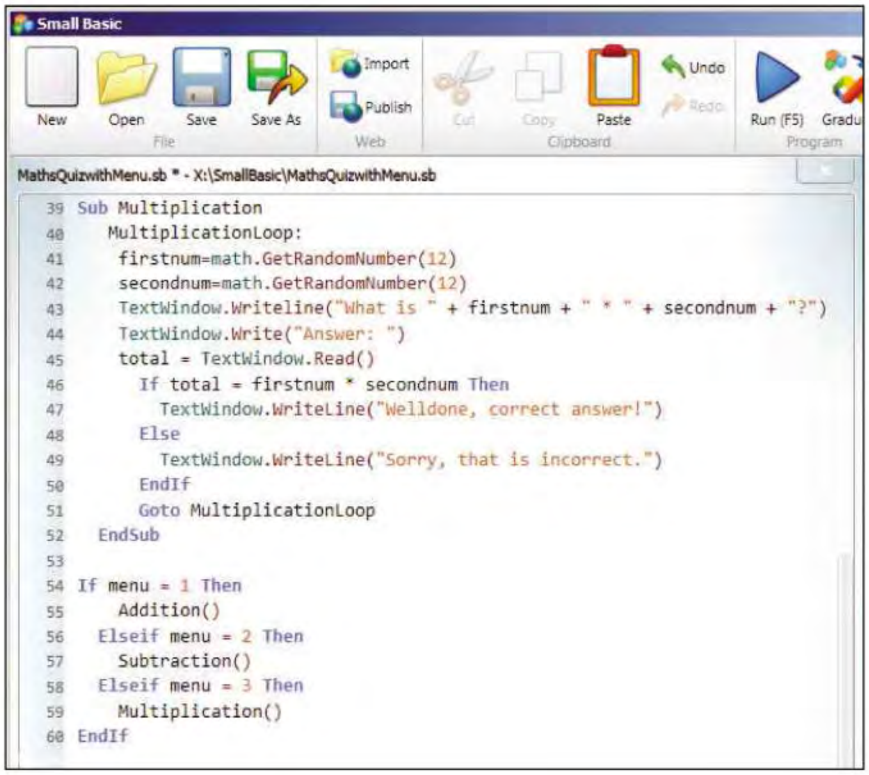
Using Elseif

In SmallBASIC, we have to code all of the subroutines before we can call them – this is why the menu selection code is at the end of the program. You'll have also noticed that the If statement we're using now differs from the one we used in our first maths program, and we have a new keyword 'Elseif'. Because our menu has more than two possible options, we need to use Elseif to give us more conditional options. The syntax for using Elseif is very simple:

```
If condition1 then
```

CODING KEYWORDS

Subroutine: An independent block of code that can be recalled from within the main program.



```
Sub Multiplication
   MultiplicationLoop:
    firstnum=math.GetRandomNumber(12)
    secondnum=math.GetRandomNumber(12)
    TextWindow.Writeline("What is " + firstnum + " * " + secondnum + "?")
    TextWindow.Write("Answer: ")
    total = TextWindow.Read()
      If total = firstnum * secondnum Then
        TextWindow.WriteLine("Welldone, correct answer!")
      Else
        TextWindow.WriteLine("Sorry, that is incorrect.")
      EndIf
      Goto MultiplicationLoop
  EndSub

If menu = 1 Then
    Addition()
  Elseif menu = 2 Then
    Subtraction()
  Elseif menu = 3 Then
    Multiplication()
EndIf
```

▲ **Add as many Elseif statements as you like, depending on how many conditions you have to deal with.**

Run this code

```
Elseif condition2 then
```

Run this code

```
Elseif condition3 then
```

Run this code

```
Else
```

Run this code

```
End if
```

TOP TIP

As subroutines are independent blocks of the code, the order in which you code them doesn't matter.

Now, we left the division element of the maths game out of our program so that you can have your first go at editing the program without having the solution. Think about all the aspects of the game so far, and you'll see that you need to add:

1. The new division option in the menu at the start

2. A subroutine for division. Don't forget to use the / symbol

3. A new Elseif option for the If statement at the end

You can add as many Elseif statements as you like, depending on how many conditions you have to deal with. The final statement is just an Else, as this is basically saying that if it's none of the above (condition1, condition2 or condition3) do the last piece of code in the list.

Adding a progress meter

The great thing about coding is that you can always add another feature. To make our program feel even more useful, we can add a progress tracker, so the user can see how many questions they've attempted, how many they've answered correctly and a percentage of their success. To do this, we're going to need three new variables: one to keep a count of how many questions have been asked (questions), one to keep track of correct responses (score) and one to calculate the percentage (percentage). To start with, add the following code above the first addition subroutine:

```
score = 0

questions = 0

percentage= 0
```

We need to amend each subroutine to include our score tracker, so let's start with the Addition subroutine. First, let's calculate how many questions we have correct: find our If statement that congratulates the user and just under that add the following code:

```
score = score + 1
```

This now means that after every correct question we're getting the current value of our variable score and moving it up by 1.

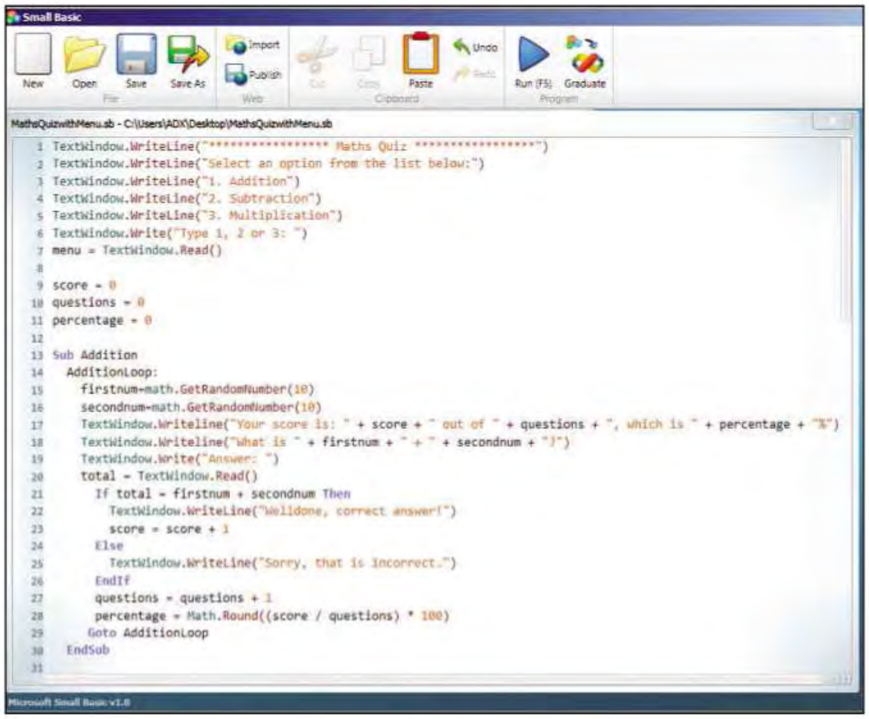
Now we need to keep track of how many questions have been asked so far by using our Questions variable. Add the following code underneath the If statement in our Addition subroutine:

```
questions = questions + 1
```

Again, just like our correct score counter, we're getting the value in the Questions variable and moving it up by 1.

If we have the number of questions correct and the total number of questions, we can calculate the percentage and keep that in a variable. Add the following code underneath our questions counter:

```
percentage = Math.Round((score /
questions) * 100)
```



```
TextWindow.WriteLine("***************** Maths Quiz *****************")
TextWindow.WriteLine("Select an option from the list below:")
TextWindow.WriteLine("1. Addition")
TextWindow.WriteLine("2. Subtraction")
TextWindow.WriteLine("3. Multiplication")
TextWindow.Write("Type 1, 2 or 3: ")
menu = TextWindow.Read()

score = 0
questions = 0
percentage = 0

Sub Addition
  AdditionLoop:
    firstnum=math.GetRandomNumber(10)
    secondnum=math.GetRandomNumber(10)
    TextWindow.Writeline("Your score is: " + score + " out of " + questions + ", which is " + percentage + "%")
    TextWindow.Writeline("What is " + firstnum + " + " + secondnum + "?")
    TextWindow.Write("Answer: ")
    total = TextWindow.Read()
      If total = firstnum + secondnum Then
        TextWindow.WriteLine("Welldone, correct answer!")
        score = score + 1
      Else
        TextWindow.WriteLine("Sorry, that is incorrect.")
      EndIf
      questions = questions + 1
      percentage = Math.Round((score / questions) * 100)
     Goto AdditionLoop
  EndSub
```

In this line of code, we're first taking score and dividing it by questions, then multiplying by 100 to find the percentage number. We then take the result and store it in the variable called Percentage. We're also taking advantage of another built-in function in SmallBASIC: Math.Round. This gets our final number and rounds it either up or down. Lastly, display the score tracker with the percentage at the start of every question. Add this code underneath where we generated our random numbers:

```
TextWindow.Writeline("Your score is: " +
score + " out of " + questions + ", which
is " + percentage + "%")
```

▲ Test your program to make sure the score tracker is working for every subroutine.

BASIC ANALYSIS

```
percentage = Math.Round((score /
questions) * 100)
```

In this line of code, we've had to use a number of brackets. Math.Round needs a set of brackets that surround the entire equation, so that the number produced from the division gets rounded to a whole number. The brackets inside calculate the score divided by questions variable before it's multiplied by 100. In your maths lessons at school, you use BIDMAS (Brackets – Indices – Multiplication – Addition – Subtraction) to order the maths operation so you get the correct number. We follow the exact same rules when we're programming.

This code is basically a long line of variable concatenation that forms a complete sentence. It works in the same way as earlier in the program, when we displayed the questions with our random numbers in for the user. Don't forget normal text is just displayed within quotations marks (" "). We add in our variables by using the plus sign, variable name, then another plus sign, so add in the next piece (+ variablename +). The only time we don't need the final plus sign is when we're at the end of our sentence.

Our final addition subroutine should now look like the following, with the new code highlighted in red:

```
Sub Addition
  AdditionLoop:
    firstnum=math.GetRandomNumber(10)
    secondnum=math.GetRandomNumber(10)

    TextWindow.Writeline("Your score is:
" + score + " out of " + questions + ",
which is " + percentage + "%")

    TextWindow.Writeline("What is " +
firstnum + " + " + secondnum + "?")
    TextWindow.Write("Answer: ")
    total = TextWindow.Read()
     If total = firstnum + secondnum Then
        TextWindow.WriteLine("Welldone,
correct answer!")
        score = score + 1
      Else
        TextWindow.WriteLine("Sorry, that
is incorrect.")
      EndIf
      questions = questions + 1
      percentage = Math.Round((score /
questions) * 100)
     Goto AdditionLoop
  EndSub
```

Of course, to complete our full quiz, we're going to need to add those highlighted bits of code to the subtraction, multiplication and division subroutines in exactly the same place for each one. Remember to test your program to make sure the score tracker is working for every subroutine. ●

TAKE IT FURTHER

Try changing the program so that the user can select the difficulty of questions, which would be the highest values of your generated numbers. You can put variables in the random number brackets instead of numbers – math.GetRandomNumber(variablename)

SmallBASIC graphics

We've tangled with text and messed around with mathematics. Now it's time we got to grips with coding graphics

WHAT YOU'LL LEARN

- How to use the Graphics Window function
- How to change the properties of the shapes
- How to position shapes by x and y coordinates
- How to use Shapes. Animate to move shapes
- How to use For Loops
- How to use Program.Delay to control the speed of your program
- How to use SetOpacity to control transparency

So far, we've been working exclusively with SmallBASIC's TextWindow object. However, when dealing with graphics, animation and drawing, we could call the GraphicsWindow.

Just like the TextWindow object, GraphicsWindow can be called into a program, and it has a number of properties we can modify. We can call shape outlines or filled shapes into the graphics window, including lines, rectangles, triangles and ellipses.

Try out the code below to see some of these shapes in action:

```
GraphicsWindow.Title = "Graphics"
```

```
GraphicsWindow.BackgroundColor =
"DarkBlue"
```

```
GraphicsWindow.Width = 200
```

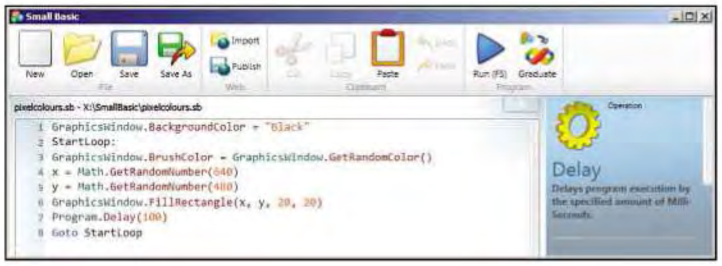
```
GraphicsWindow.Height = 200
```

```
GraphicsWindow.PenWidth = 8
GraphicsWindow.PenColor = "LightBlue"
```

```
GraphicsWindow.DrawLine(10, 10, 10, 190)
```

CODING KEYWORDS

Arguments: Values that can be changed in properties of an object such as its position, height, width and colour. Programmers will usually refer to arguments as 'args'.


Small Basic
New
Open
Save
Save As
Import
Publish
Paste
Run (F5)
Graduate
pixelcolours.sb - X:\SmallBasic\pixelcolours.sb
Operation
Delay
Delays program execution by the specified amount of MilliSeconds.

```
GraphicsWindow.BackgroundColor = "Black"
StartLoop:
GraphicsWindow.BrushColor = GraphicsWindow.GetRandomColor()
x = Math.GetRandomNumber(640)
y = Math.GetRandomNumber(480)
GraphicsWindow.FillRectangle(x, y, 20, 20)
Program.Delay(100)
Goto StartLoop
```

▲ This program shows you how to generate a random colour and create a rectangle with random coordinates, and more.

```
GraphicsWindow.PenWidth = 5
```

```
GraphicsWindow.DrawLine(22, 10, 22, 190)
```

```
GraphicsWindow.PenWidth = 2
```

```
GraphicsWindow.DrawLine(30, 10, 30, 190)
```

```
GraphicsWindow.PenColor = "Blue"
```

```
GraphicsWindow.FillEllipse(110, 110, 80,
80)
```

```
GraphicsWindow.DrawEllipse(40, 110, 80,
80)
```

```
GraphicsWindow.FillRectangle(40, 10, 70,
72)
```

```
GraphicsWindow.DrawRectangle(120, 10, 70,
70)
```

Drawing shapes in random locations

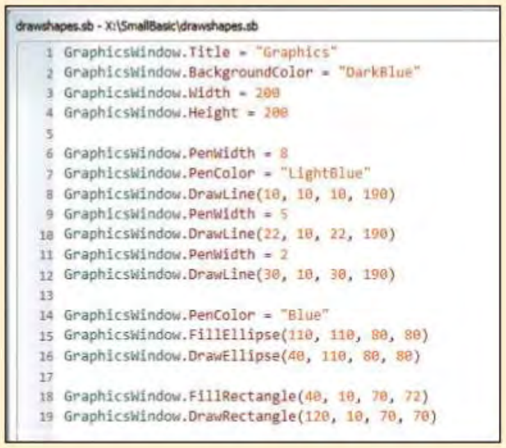
We can now add static shapes to our graphics window, but that's not very exciting, is it? Let's try to use the SmallBASIC code to make something

BASIC ANALYSIS – DRAWING GRAPHICS

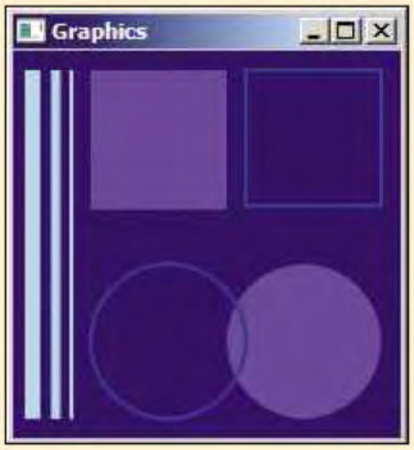
Just like with TextWindow object, we call the graphics object with GraphicsWindow, then define the type of shape, and whether it has a solid fill colour. The first two numbers in the brackets are the x and y pixel coordinates of the graphics window. To give you a point of reference: in a SmallBASIC graphics window that is 200 x 200 pixels, the top-left corner of the graphics window is coordinates 0,0 and the bottom right has the coordinates 200,200. The next two numbers after the coordinates in the brackets are the width and height of the shape in pixels.

Drawing lines and triangles is a slightly different process. The first two numbers for DrawLine are the x and y coordinates, which define where the start of the line should go. The last numbers are the x and y coordinates for the end of the line. The program then draws a straight line between those two points. When you call the triangle shape DrawTriangle you have six numbers in the brackets, as you have to provide x and y coordinates for each of the three points of the triangle.

Just as with our text colour, we can change the colour properties of our shapes, and will keep


drawshapes.sb - X:\SmallBasic\drawshapes.sb

```
GraphicsWindow.Title = "Graphics"
GraphicsWindow.BackgroundColor = "DarkBlue"
GraphicsWindow.Width = 200
GraphicsWindow.Height = 200

GraphicsWindow.PenWidth = 8
GraphicsWindow.PenColor = "LightBlue"
GraphicsWindow.DrawLine(10, 10, 10, 190)
GraphicsWindow.PenWidth = 5
GraphicsWindow.DrawLine(22, 10, 22, 190)
GraphicsWindow.PenWidth = 2
GraphicsWindow.DrawLine(30, 10, 30, 190)

GraphicsWindow.PenColor = "Blue"
GraphicsWindow.FillEllipse(110, 110, 80, 80)
GraphicsWindow.DrawEllipse(40, 110, 80, 80)

GraphicsWindow.FillRectangle(40, 10, 70, 72)
GraphicsWindow.DrawRectangle(120, 10, 70, 70)
```



that colour until it's changed later in the code:

```
GraphicsWindow.PenColor =
"Magenta"
```

When drawing lines, you can specify how thick the line can be using the following line of code before the drawing the line, and again it will keep that thickness until it's given another value:

```
GraphicsWindow.PenWidth = 2
```

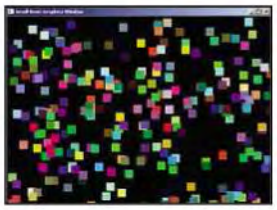
We can now start to tell our program the width and height of our graphics window in pixels, using the following code:

```
GraphicsWindow.Width = 200
```

```
GraphicsWindow.Height = 200
```

less predictable. Our next program is going to follow
this sequence:

1. Set the background colour to black
2. Start a loop marker
3. Generate a random colour
4. Generate a random number (max 640) and assign it to a variable called x
5. Generate a random number (max 480) and assign it to a variable called y
6. Create a rectangle (20 x 20) with the random x and y coordinates


▲ Our program in action!

7. Loop back around to the start of the loop marker

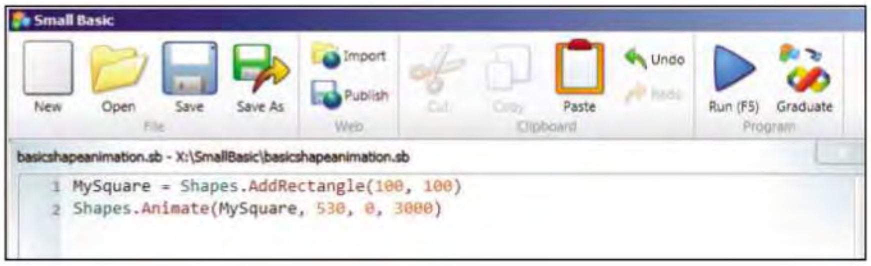
Create a new code window and add the following code.

```
GraphicsWindow.BackgroundColor = "Black"
StartLoop:

GraphicsWindow.BrushColor =

GraphicsWindow.GetRandomColor()

x = Math.GetRandomNumber(640)

y = Math.GetRandomNumber(480)

GraphicsWindow.FillRectangle(x, y, 20,
20)

Program.Delay(100)

Goto StartLoop
```

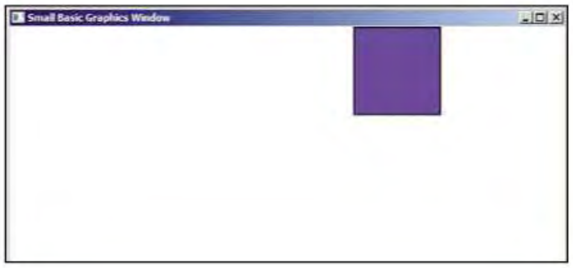
Just before the end of the code, we've added a new feature called Program.Delay(100), which will pause the program before it loops around, giving us the ability to change the speed in which the program runs. The length of that pause can be changed with the number in brackets, which

EXPERIMENT

Try adding your own shapes to get used to how the coordinate system works. If you get stuck, change just one number in the brackets and see what effect that has on the shape or its position in the GraphicsWindow.



```
MySquare = Shapes.AddRectangle(100, 100)
Shapes.Animate(MySquare, 530, 0, 3000)
```



▲ Have a go at animating a square across the screen.

represents time in milliseconds: try making that number higher and lower from its value of 100 and see what effect that has on the program.

Another good experiment with this program is to change the size of the rectangle from 20, 20 to values higher or lower, but you'll need to keep them the same if you want a perfect square.

Animating shapes

Now we can try to animate a square across the screen by using some new code keywords, Shape. AddRectangle and Shape.Animate.

Quickly test the following code in a new window:

```
MySquare = Shapes.AddRectangle(100, 100)
Shapes.Animate(MySquare, 530, 0, 3000)
```

We're defining the shape called MySquare, just like we would a variable, and giving it a rectangle that is 100 x 100 in width and height. Now that we have a shape name defined, we can use Shapes.Animate to move MySquare from its starting 0,0 coordinate to 530,0 coordinate over a time span of 3,000 milliseconds.

Make a snow globe simulator

For our final graphics program, we're going to create a snow globe simulator, in which we can change the variables to switch between a gentle snowstorm or a raging blizzard. The mechanics of the program are simple: we're creating an array of ellipses off screen at the top, and randomising the start and end position. We can then use variables to control how our simulator works, changing the speed, size and density of the snowflakes.

Create a new code window and add the following code:

```
GraphicsWindow.BackgroundColor = "Black"

GraphicsWindow.BrushColor = "White"

GraphicsWindow.fillEllipse(300, 400, 75,
75)

GraphicsWindow.fillEllipse(315, 370, 45,
45)

GraphicsWindow.BrushColor = "black"

GraphicsWindow.fillEllipse(322, 380, 10,
10)

GraphicsWindow.fillEllipse(340, 380, 10,
10)

flakesize = 7
flakespeed = 3000
flakedensity = 50
rows = 20
columns = 20

For r = 1 To rows
  For c = 1 To columns
    startpos = Math.GetRandomNumber(700)
    opacity = Math.GetRandomNumber(120)
    GraphicsWindow.BrushColor = "White"
    snow[r][c] = Shapes.
AddEllipse(flakesize, flakesize)
     Shapes.SetOpacity(snow[r][c],
opacity)
Shapes.Move(snow[r][c], startpos, -30)
 EndFor
EndFor
```

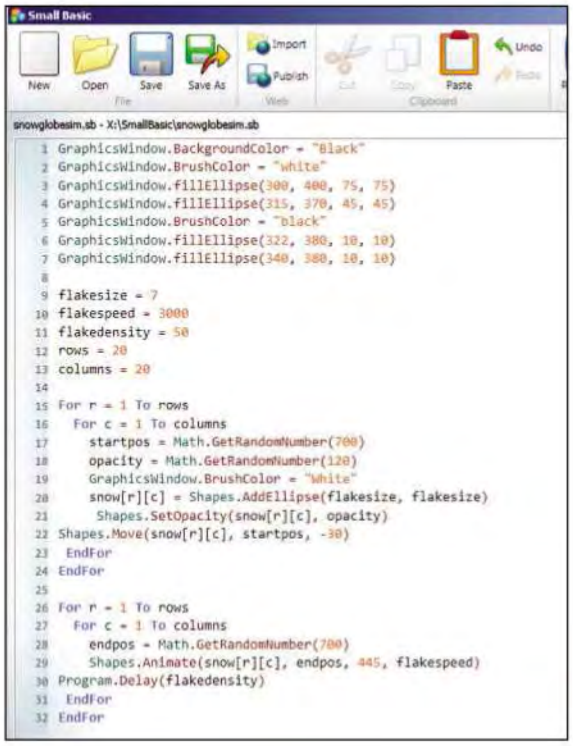
BASIC ANALYSIS

```
Shapes.Animate(MySquare, 530, 0 , 3000)
```

We call our animation function with Shapes.Animate. Our first argument in brackets is the name of the shape – MySquare. Next, we have to give coordinate locations of where the shape will finish (530,0) followed by how long the animation will take in milliseconds (3000).

GO FURTHER

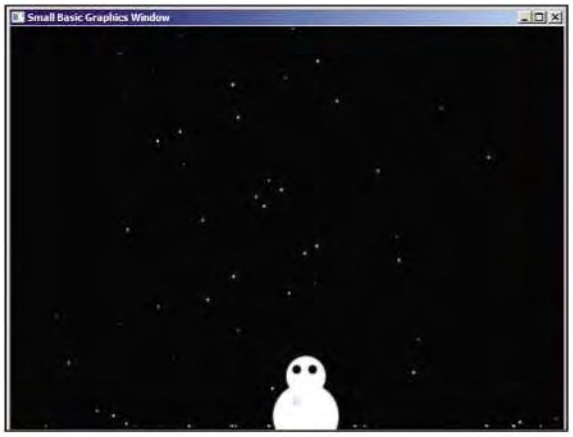
Instead of squares, try changing the code so that it produces ellipses or triangles. While you're at it, see if you can modify the program so the rectangles that appear each time are random sizes. Have a look at how we've used our x and y variables for the shape position to help you.



```
GraphicsWindow.BackgroundColor = "Black"
GraphicsWindow.BrushColor = "white"
GraphicsWindow.fillEllipse(300, 400, 75, 75)
GraphicsWindow.fillEllipse(315, 370, 45, 45)
GraphicsWindow.BrushColor = "black"
GraphicsWindow.fillEllipse(322, 380, 10, 10)
GraphicsWindow.fillEllipse(340, 380, 10, 10)

flakesize = 7
flakespeed = 3000
flakedensity = 50
rows = 20
columns = 20

For r = 1 To rows
  For c = 1 To columns
    startpos = Math.GetRandomNumber(700)
    opacity = Math.GetRandomNumber(120)
    GraphicsWindow.BrushColor = "White"
    snow[r][c] = Shapes.AddEllipse(flakesize, flakesize)
     Shapes.SetOpacity(snow[r][c], opacity)
Shapes.Move(snow[r][c], startpos, -30)
 EndFor
EndFor

For r = 1 To rows
  For c = 1 To columns
    endpos = Math.GetRandomNumber(700)
    Shapes.Animate(snow[r][c], endpos, 445, flakespeed)
Program.Delay(flakedensity)
 EndFor
EndFor
```

Here's our snowman enjoying a gentle snowstorm!

The mechanics of the snow globe simulator are actually pretty simple.



```
For r = 1 To rows
  For c = 1 To columns
    endpos = Math.GetRandomNumber(700)
    Shapes.Animate(snow[r][c], endpos,
445, flakespeed)
Program.Delay(flakedensity)
 EndFor
EndFor
```

This program uses the For loop, which repeats a block of code a certain number of times using a counter. When the counter reaches the predetermined value – which can be a variable value – the loop will stop and return to running the main program. The syntax for the For loop is simple:

```
For condition
code to loop
EndFor
```

As this is a simulator, you can change the variable numbers and see what happens within the globe. Change the following variable for the size of each virtual snowflake (ellipse):

```
flakesize = 7
```

The following variable is put into the Delay function at the end; the lower the number (milliseconds), the more snowflakes will be released simultaneously:

```
flakedensity = 80
```

The next variable is the speed at which each snowflake will fall from the start to the end of its run. Remember, the number is in milliseconds so the lower you go, the faster the snow will fall:

```
flakespeed = 3000
```

To make our snow more realistic, we're using the code SetOpacity, which will work from random numbers, 0 being invisible and 100 being completely visible. This means that each square or virtual snowflake that's created will have a random transparency value, so some snowflakes are more visible than others, giving the illusion of depth and size. You could set this number higher than 100 as we've done, because then there's a greater probability of generating higher numbers, although anything generated over 100 will still be treated as 100. ●

CODING KEYWORDS

Syntax: The structure and order of the code in a program. Syntax is a vital part of any textual programming language. Ignore syntax, and your program won't run.

EXPERIMENT

Try putting Program.Delay(3000) after the last Shapes.Animate code, then try moving it to another coordinate location with another line of animation. See if you can make the square move around the perimeter of the graphics window with a delay separating each line of new animate code.