MAGBOOK

BUILD A GAME IN 30 MINUTES!

# Coding for Kids

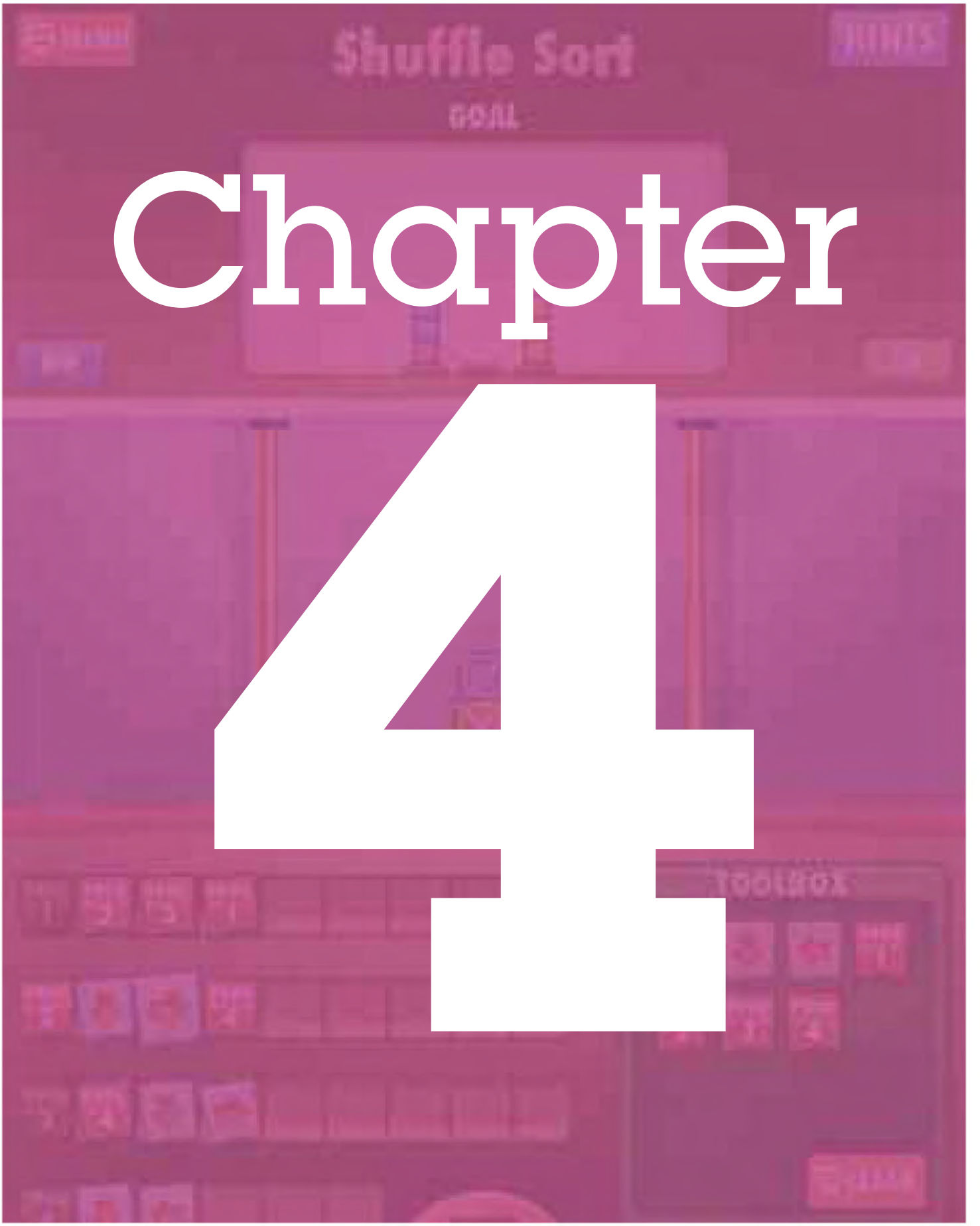
# Scratch

Step-by-step friendly advice

Animate stories

Make your own racing game

Create your own quiz



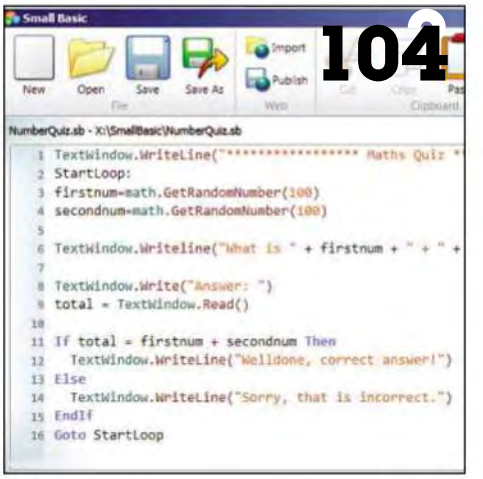

# Chapter 4

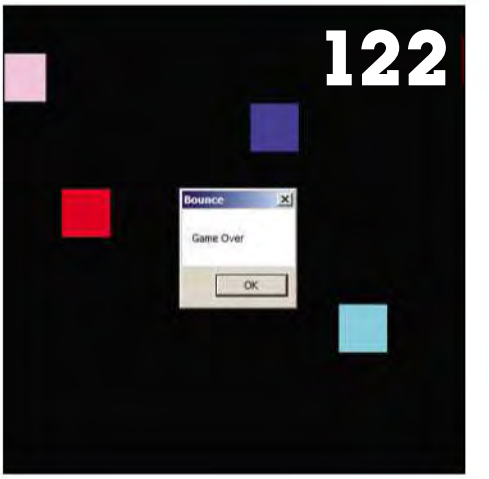
# Contents


16

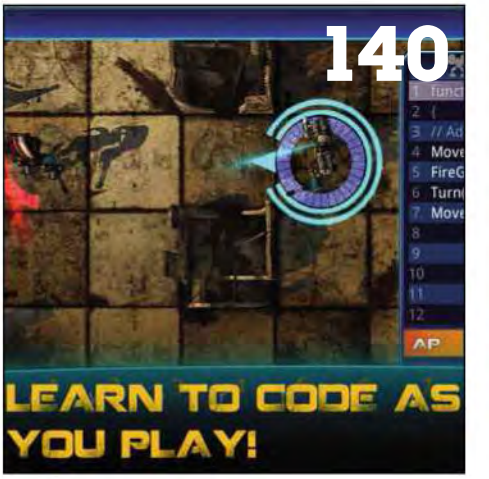
15

32



# Section 4

# The next level

With the help of SmallBASIC, we've learnt to put together simple BASIC programs. Yet while SmallBASIC is a surprisingly powerful and easy-to-learn implementation of the language, there will come a time when you'll need something more flexible, and with more built-in features. That's where Visual Basic comes in.

For the next few projects, we'll be using a free version of a proper programming environment – the same tools used by millions of professional software developers every day. That means you'll have to cope with a little more complexity, but you'll soon find that it isn't as hard as it first looks.

By the end of this chapter, you'll have some serious coding skills. The only question is, where will you take them next?

**IN THIS SECTION**

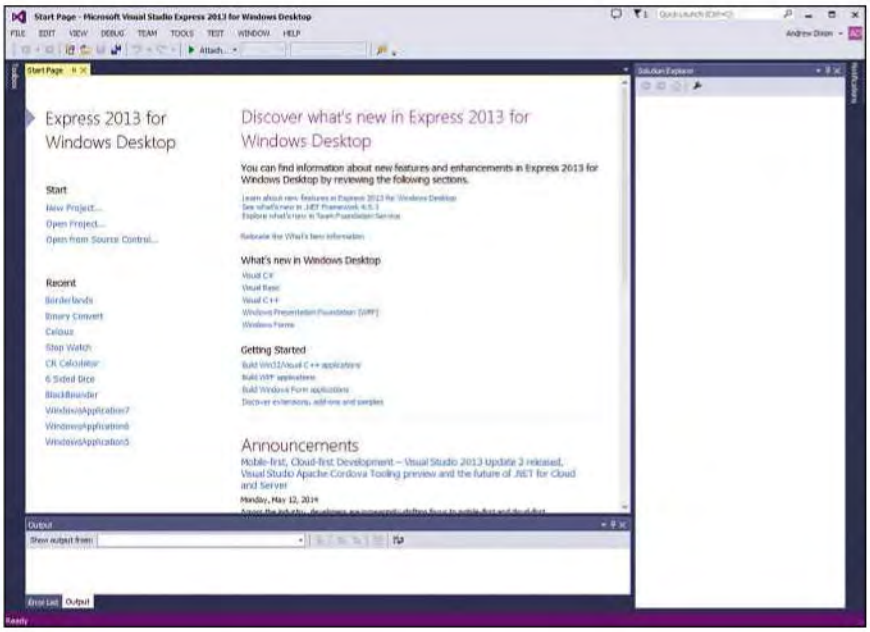
# Introducing Visual Basic

SmallBASIC can take you a long way on your coding journey, but eventually you'll want a little more power. That's where Visual Basic comes in

**WHAT YOU'LL LEARN**

- About VB, and how to get it for free
- How to work with buttons, forms and code
- How to code a simple control panel

Visual Basic is part of Microsoft's Visual Studio, an IDE (Integrated Development Environment) suite of programming tools that lets you work with a variety of coding languages, including C++, C# and Visual Basic. The full professional package doesn't come cheap, but there's a free edition, Visual Studio Express 2013 for Desktop, which gives you access to a still very powerful set of tools and built-in functions, with which you can build almost any kind of program, game or app. The latest 2013 version requires you to sign up for a Microsoft account, which you'll need to do in order to use the free version after the 30-day trial period is up.




## Where can you find it?

There are several versions of Visual Studio, so make sure you search for the free version – Visual Studio Express 2013 for Desktop – rather than the high-end professional versions. You can find it at www.visualstudio.com/downloads/download-visual-studio-vs#d-express-windows-desktop

The Visual Studio website also has a variety of guides and documentation, covering everything from setting up the software to developing applications. You can find these at www.visualstudio.com/get-started/overview-of-get-started-tasks-vs

## How can I use it?

Unlike Scratch or SmallBASIC, Visual Studio is a professional piece of software that you'd use as a professional developer for the Windows platform. The free version that you'll install will look nearly identical to the full version, so when you first start using it the menus and buttons may seem daunting. Don't worry. We're going to guide you through all the sections that you'll need to use at first.

As Visual Studio itself covers a variety of languages and can be used for many different Windows applications, when you first create a new application it will ask you what type of program you're going to create. For all the examples in the book, we'll choose the most common application type, which is a Windows Forms Application.

The code for Visual Basic isn't a million miles away from the code you might use for SmallBASIC, but now we have full control over our interface. We can design and draw buttons, forms, textboxes, and label boxes with our mouse before we type any

◀ There are several versions of Visual Studio, so make sure you search for the free version.

> “The code for Visual Basic isn’t a million miles away from the code for SmallBASIC”

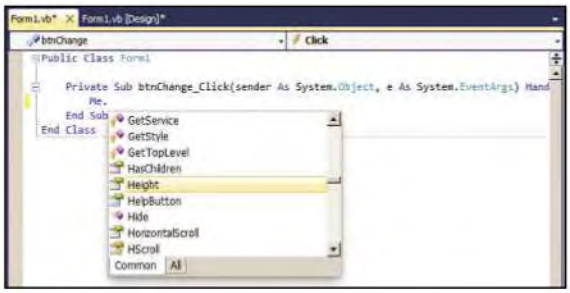
code. All of these visual components are located in a toolbox on the left of the screen. You can either scroll through these or search for a pre-made object and drag it onto the form.

Visual Basic is an event-driven language, so when we’re ready to add code we double-click on an object, such as a button, then add the code that works behind that button. Your Visual Studio environment will try to help you code by guessing what you’re about to type. We call this ‘Intellisense’. It works a bit like predictive texting on mobile phones, but the computer knows the language keywords and every property, so you can use Enter or space on the keyboard to automatically complete the words you’re typing.

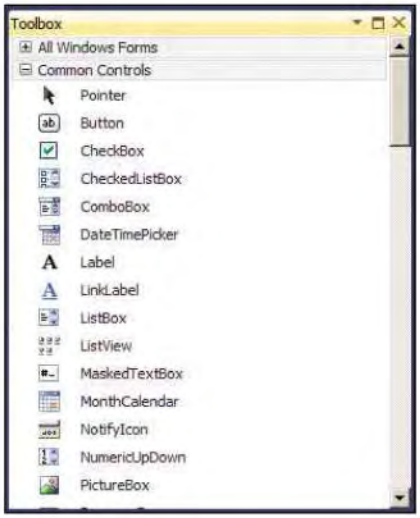
## What is Solution Explorer?

Many of the options we’re going to change when designing the GUI in Visual Basic will be through the Properties window on the left of the screen. This menu will change depending on what item is selected. Also, above the Properties window you’ll see the Solution Explorer: this will tell you the names of your documents and resources, as well as what forms you’ve created inside the current project.

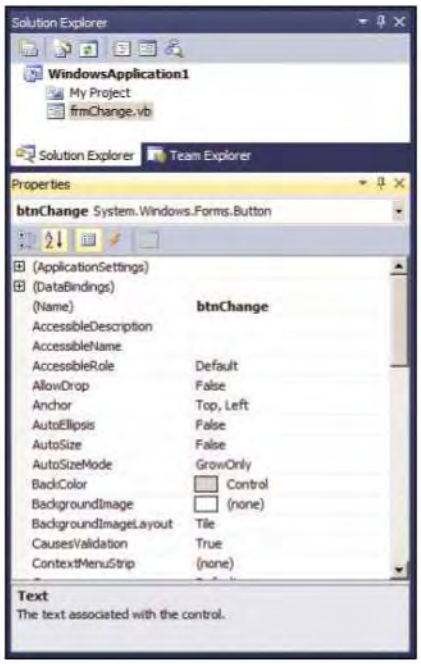
Just like SmallBASIC, programs are executed with the Run button at the top of the page. This gets the computer to compile your code and creates an EXE file. If there are any problems or bugs, you’ll get a report at the bottom of the screen. As Visual Basic is a more complex language, you’re more likely to, and the help you get on any problems can use some quite technical wording. Luckily, the internet can help. If you get stuck with an error, try typing it into Google or Bing, and you’ll often find some other programmers talking about the issue.




▲ Intellisense works a bit like predictive texting on mobile phones, automatically completing words you type.




◀ All of the visual components are located in the toolbox.

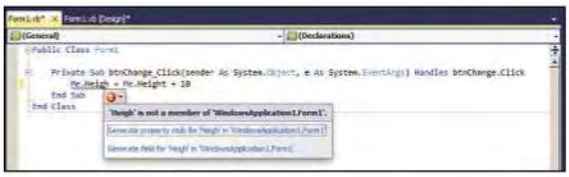



▲ Many of the options we’re going to change when designing the GUI in Visual Basic will be through the Properties window.

## Our first Visual Basic app

To get familiar with the IDE and the programming language, we’re going to create a simple app that allows the user to change the colour and dimensions of a form and its buttons, as well as add a counter and hide the buttons.

Start Visual Studio and create a new Windows Forms Application. Change the name of the project to ControlPanel and click OK. You’ll get the default form (300 x 300 pixels), which can be resized. Drag the lower-left corner out so we have more room to add controls. With the form selected, have a look through the Property window on the right of the screen. This menu will show all the properties (name, colours, appearance, etc) of our form, and we can change and customise all of the



▲ If there are any problems or bugs, you’ll get a report at the bottom of the screen.

### NAMING BASICS

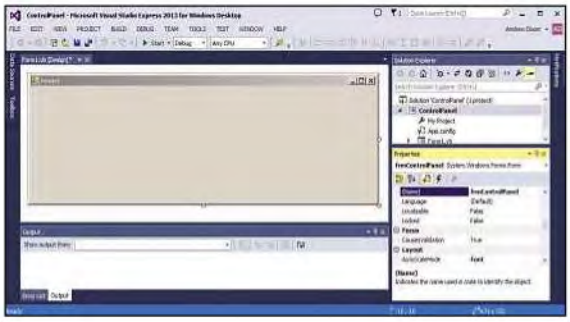
Naming your forms and controls is important, both because you want to make your code understandable and because you’ll have to reference them in the code by their name. In order to remember what Button1, Button2 and Button 3 are, we’ll give them descriptive names. A method developers use to start a name is to shorten the type of the control or form down to three letters first, followed by a descriptive name without any spaces:

Buttons – btnStart
Textbox – txtPlayerName
Labels – lblPlayerScore
Forms – frmMainGame

values. Scroll through your list and find the Name property, and change this to frmControlPanel.

To add buttons to our form, we need to look through the Visual Basic toolbox located on the left of the screen, which has a list of pre-made controls that we can drag to our form; by default, it will be called Button1. Find the control Button

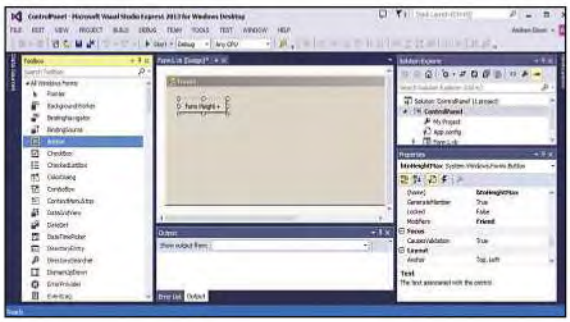
▶ Start Visual Studio and create a new Windows Forms Application.



▶ Find the Name property, and change this to frmControlPanel.

in the list and either double-click to automatically draw the button to the form, or single-click and manually drag the size of button you want. Your first action after adding a new control is to rename it to btnHeightMax. Before then, change the Text property (the wording on the button) to Form Height +. Next, create another button next to your first one and rename the button btnHeightMin and change the Text property to Form Height -.

We're now ready to start adding our first piece of code. Double-click your first button btnHeightMax and you'll be taken to Code view,



▲ After adding a new control, rename it to btnHeightMax.

where Visual Studio has already put in all the required code to make a subroutine. Don't change or delete any of this code, as this will probably result in an error that you may not be able to fix.

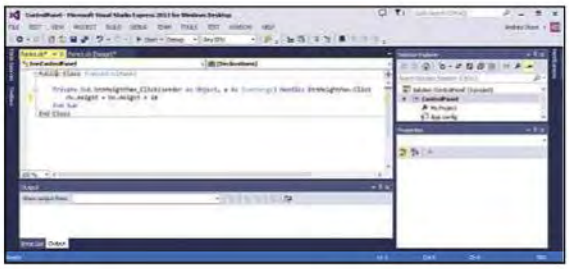
We're going place our code in the middle of the Button subroutine, which you can find after the Private Sub line and before the End Sub line. We want our button to increase the height of our form by 10 pixels with the following line of code:

```
Me.Height = Me.Height + 10
```

At the top of the page, you'll now have two tabs with your Form name, so you can switch between the form design and our Code view. Try switching between the two now, so you know how to get from one to the other.

Double-click the btnHeightMin button on the form, and Visual Studio will again add the initial subroutine code. Just add the following code in between the Private Sub line and the End Sub line:

```
Me.Height = Me.Height - 10
```



▲ Place your code in the middle of the Button subroutine.

To test that our program works so far, we can compile and execute it using the green Play button at the top of the page, or by pressing F5. Try your btnHeightMax button and see if every click increases the height by 10 pixels, then test your btnHeightMin button to see if it does the opposite. You won't be able to make any changes to your form or code while the program is running, so close it with the Stop button, next to the Play button.

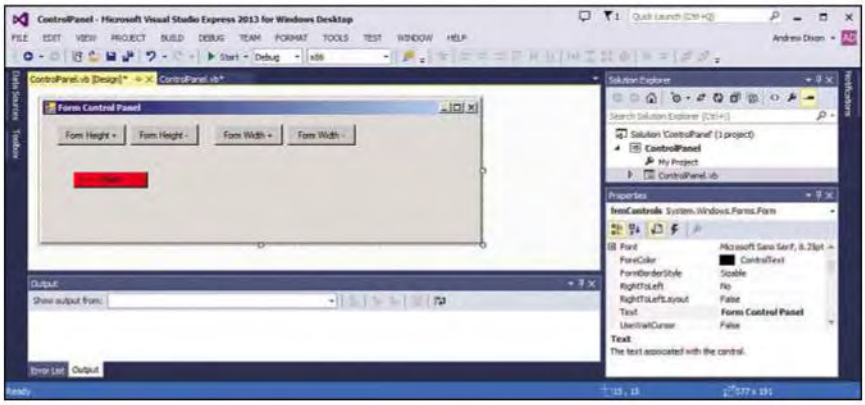
## TOP TIP

**You can add your own written notes to your code so you can remind yourself or others what each line does: we call this commenting. In Visual Basic, the symbol to start a comment is an apostrophe ('). All text after that point will turn green and is ignored by the compiler.**

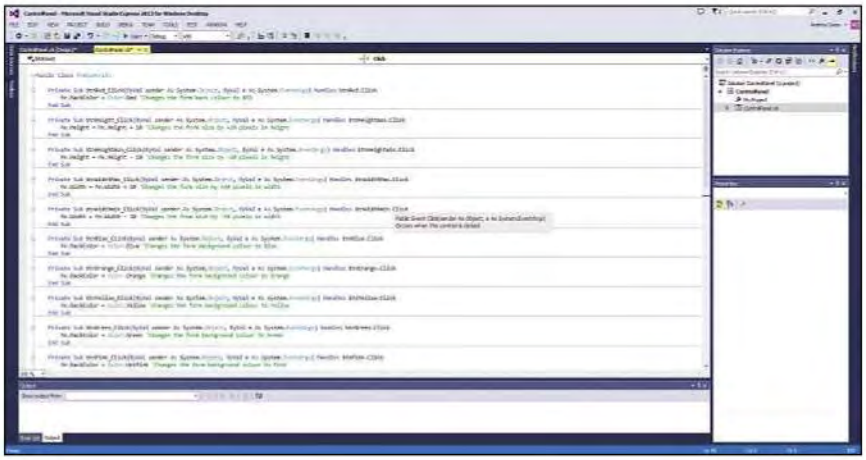
## BASIC ANALYSIS

```
Me.height = Me.height + 10
```

As in SmallBASIC, we refer to the object first, then use a full stop and refer to its property. In Visual Basic, we can't directly refer to the name of the form that we're currently using, so we use the word: Me. Our code is instructing the computer to get the box's current height in pixels and add 10 to that number.



To emphasise that this button is going to change something to red, we can make the actual button red.



Create a selection of other colours, which the user can decide to change to.

You have all the skills now to add two more buttons that will increase and decrease the width of the form. The code will nearly be identical; just remember to change the width not the height.

## Changing colours

Now we're going to spice things up by giving our form colour themes. Create a new button called btnRed and change the display text to say Red. To emphasise that this button is going to change something to red, we can make the actual button red by changing BackColor open to red in the Property window. When you're ready to add the code, double-click the red button and type the following statement:

```
Me.BackColor = Color.Red
```

Run your program again and see what effect your red button has!

Repeat the task we've just completed with the red button to create a selection of other colours, which the user can decide to change to. Each time, rename the button, change the displayed text and manually change the button colour to match the colour it will be in the code. Arrange your buttons in a logical order and add a groupbox from the Toolbox menu, which can be drawn around your buttons to give them a separated border. Then, change the text to Colour Palette.

## Adding a Reset button

We're going to add a Reset button to our form, which will revert all of our dimensions back to the state when you first started the program, and we'll set the colour back to the starting grey, so it will look like we've just restarted the entire program. Add a button onto your form and call it btnReset and change the display text to Reset. You can change the style and size if you wish:

```
Me.Height = 191
```

```
Me.Width = 577
```

```
Me.BackColor = Color.LightGray
```

The size of our form will differ from yours, so click on your form in Design view and have a look down the Property menu for Size. You'll see two numbers separated by a comma. This is the width and height in pixels; use those numbers in the code not ours for Me.Width and Me.Height.

Most visible toolbox controls such as buttons, textboxes and labels have a visibility property that we can change in the code as a Boolean value. Create two new buttons called btnVisible and btnInvisible, give their button text an appropriate name, then position them by your Reset button, as this will hide or show this button. The code for the invisible button will be:

```
btnReset.Visible = False
```

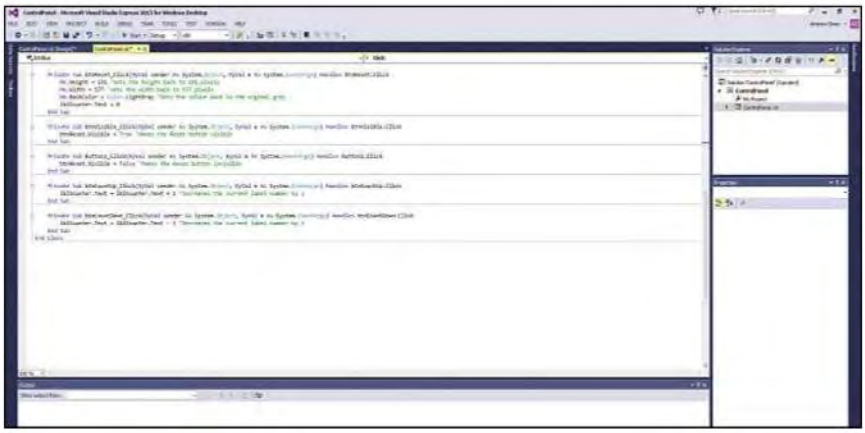
The code of the visible button will be the same; just reverse the Boolean value:

```
btnReset.Visible = True
```

### BASIC ANALYSIS

```
Me.BackColor = Color.Red
```

Again, we can't reference our form by name while we're working inside it, so we use Me.Backcolor to identify what's going to be changed, then we choose a colour using Color.Red. If you just type the word color, Intellisense will give you a list of valid names to choose from.



▲ Our counter, when clicked, will increase a number in a label box, and another button to decrease the number.

## Adding a counter

The last feature to add to our control panel is a counter, which when clicked will increase a number in a label box, and another button to decrease the number. For this, we're going to need three controls. Place two buttons on the form called btnCountUp and btnCountDown, and for the display text insert a + for one and a – for the other. Don't forget fonts and sizes can be found in the Property menu under Font. In between the two numbers, have a label box called lblCounter and set the display text to 0.

The code for our btnCountUp button is:

```
lblCounter.Text = lblCounter.Text + 1
```

The code for our btnCountDown button is:

```
lblCounter.Text = lblCounter.Text - 1
```

Give your program a test to see if all buttons on the page work as you want them to, either by pressing F5 or clicking the Run button at the top of the page. Save your program so you can see how you've written the code, in case you want to revisit it in the future to expand some new ideas.

## Final code

```
Public Class frmControls

    Private Sub btnRed_Click(ByVal sender
As System.Object, ByVal e As System.
EventArgs) Handles btnRed.Click
        Me.BackColor = Color.Red 'Changes
the form back colour to RED
    End Sub

    Private Sub btnHeight_Click(ByVal
sender As System.Object, ByVal e As
System.EventArgs) Handles btnHeightmax.
Click
        Me.Height += +10 'Changes the
form size by +10 pixels in height
    End Sub

    Private Sub btnHeightmin_Click(ByVal
sender As System.Object, ByVal e As
System.EventArgs) Handles btnHeightmin.
Click
        Me.Height = Me.Height - 10
'Changes the form size by -10 pixels in
height
    End Sub

    Private Sub btnWidthMax_Click(ByVal
sender As System.Object, ByVal e As
System.EventArgs) Handles btnWidthMax.
Click
        Me.Width = Me.Width + 10 'Changes
the form size by +10 pixels in width
    End Sub

    Private Sub btnWidthmin_Click(ByVal
sender As System.Object, ByVal e As
System.EventArgs) Handles btnWidthmin.
Click
        Me.Width = Me.Width - 10 'Changes
the form size by -10 pixels in width
    End Sub

    Private Sub btnBlue_Click(ByVal
sender As System.Object, ByVal e As
System.EventArgs) Handles btnBlue.Click
        Me.BackColor = Color.Blue
'Changes the form background colour to
blue
    End Sub

    Private Sub btnOrange_Click(ByVal
sender As System.Object, ByVal e As
System.EventArgs) Handles btnOrange.Click
        Me.BackColor = Color.Orange
'Changes the form background colour to
Orange
    End Sub

    Private Sub btnYellow_Click(ByVal
sender As System.Object, ByVal e As
System.EventArgs) Handles btnYellow.Click
        Me.BackColor = Color.Yellow
'Changes the form background colour to
Yellow
```

**TOP TIP**

**Instead of using Me.Height = Me.Height – 10 we can use Me.Height -= 10 and it would have the same effect but with less code. We've done this the long way because it's easier to read for your first program.**

**CODING KEYWORDS**

**Boolean:** A type of data that can only exist in one state – True – or another – False.


```
    End Sub

    Private Sub btnGreen_Click(ByVal
sender As System.Object, ByVal e As
System.EventArgs) Handles btnGreen.Click
        Me.BackColor = Color.Green
'Changes the form background colour to
Green
    End Sub

    Private Sub btnPink_Click(ByVal
sender As System.Object, ByVal e As
System.EventArgs) Handles btnPink.Click
        Me.BackColor = Color.HotPink
'Changes the form background colour to
Pink
    End Sub

    Private Sub btnReset_Click(ByVal
sender As System.Object, ByVal e As
System.EventArgs) Handles btnReset.Click
        Me.Height = 191 'Sets the height
back to 191 pixels
        Me.Width = 577 'sets the width
back to 577 pixels
        Me.BackColor = Color.LightGray
'Sets the colour back to the original
grey
        lblCounter.Text = 0 'Resets the
counter back to 0
    End Sub

    Private Sub btnVisible_Click(ByVal
sender As System.Object, ByVal e As
System.EventArgs) Handles btnVisible.
Click
        btnReset.Visible = True 'Makes
the Reset button visible
    End Sub

    Private Sub Button1_Click(ByVal
sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        btnReset.Visible = False 'Makes
the Reset button invisible
    End Sub

    Private Sub btnCountUp_Click(ByVal
sender As System.Object, ByVal e As
System.EventArgs) Handles btnCountUp.
Click
        lblCounter.Text = lblCounter.Text
+ 1 'Increases the lblCounter number by 1
    End Sub

    Private Sub btnCountDown_Click(ByVal
sender As System.Object, ByVal e As
System.
EventArgs) Handles btnCountDown.Click
        lblCounter.Text = lblCounter.Text
- 1 'Decreases the lblCounter number by 1
    End Sub
End Class
```

▲ Give your program a test to see if all buttons on the page work as you want them to.

**TOP TIP**

**When you've completed a working program, high-five the nearest human being!**

## EXPERIMENT

Can you add a line of code to btnReset so that the text of lblCounter is set back to 0?

# Build your first Basic game

Now you've got a feel for Visual Basic, it's time to have some fun building our first simple game – a version of the original blockbuster Breakout

**WHAT YOU'LL LEARN**

- How to use PictureBoxes to create elements for a game
- How to move objects around the screen
- How to detect when objects collide
- How to use private variables
- How to use subroutines to make your programs more efficient

The first game we're going to create in Visual Basic is a version of the pioneering arcade favourite, Breakout. The player controls a bat that has to deflect a ball around the arena, hitting all the blocks without letting the ball slip behind them. Points are scored for each block you hit, and you need to hit all the blocks to clear the level; if you hit the floor too many times, the game will end. It's one of the oldest and simplest arcade games, and one developers still come back to for inspiration today. Putting it together is reasonably straightforward.

Create a new Window Forms Applications project, and give it a cool-sounding name for the game's title like 'Oblong Offensive'. Our first task is to resize the form so that we have a bigger arena to bounce the ball. So with the form selected, change the size value in the Property window to 1000, 760, as this is the length and width in pixels.

While we have the Property window open, change the Name value to the title of the form name to 'Oblong Offensive' in the Name field.

| | |
|---|---|
| ⊟ **Design** | |
| (Name) | **OblongOffensive** |
| Language | (Default) |
| Localizable | False |
| Locked | False |
| ⊟ **Focus** | |
| CausesValidation | True |
| ⊟ **Layout** | |
| AutoScaleMode | **Font** |
| AutoScroll | False |
| ⊞ AutoScrollMargin | 0, 0 |
| ⊞ AutoScrollMinSize | 0, 0 |
| AutoSize | False |
| AutoSizeMode | GrowOnly |
| ⊞ Location | 0, 0 |
| ⊞ MaximumSize | 0, 0 |
| ⊞ MinimumSize | 0, 0 |
| ⊞ Padding | 0, 0, 0, 0 |
| ⊞ Size | **1000, 760** |
| StartPosition | WindowsDefaultLocation |
| WindowState | Normal |

▶ Change the size value in the Property window to 1000, 760.

## Adding walls

We now need to add walls to each side of the arena so the ball can bounce off each one. For each wall, we're going to use the PictureBox control from the toolbox, which is great for inserting images or filling them with colours. Drag a PictureBox onto the form and resize it to the length of the form, then place it at the top of the screen. Change the name of the PictureBox to pctWallTop, then the colour using both the Name and Backcolor values in the Property window.

Now create another PictureBox, rename it pctWallBottom, give it a colour, then position it at the bottom of the page. Next, create a third PictureBox, rename it pctWallLeft, give it a colour and position it on the left of the page. Finally, create a fourth PictureBox, rename it pctWallRight, give it a colour and position it at the right of the page.

We now need to add the ball. Again, this is going to be a PictureBox called pctBall, but this time change the size to equal width and height (18, 18) by changing the Size property.

## Getting the ball moving

It's time to get our ball moving around the screen and bouncing off each of the four walls, so we'll need a Timer control from the toolbox. Drag one from the toolbox onto our form (this will appear underneath) and rename it tmrMovement. We're also going to set the interval value in the property window to 25. Double-clicking the tmrMovement

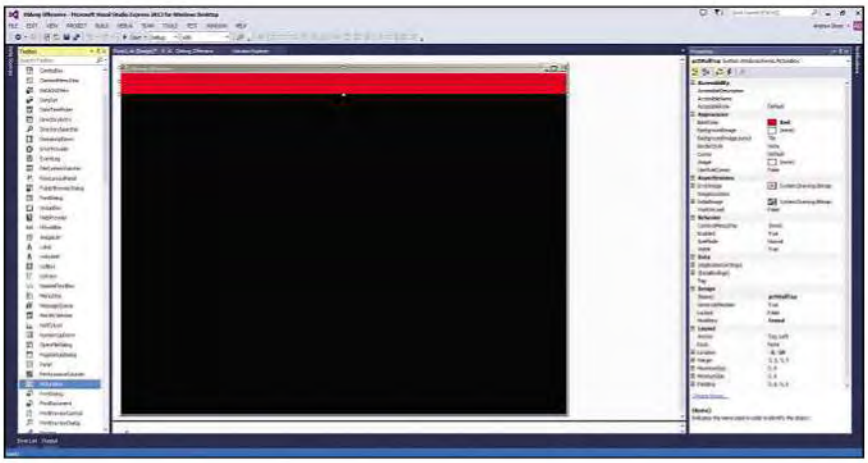
## FIND THE RIGHT DIRECTION

This handy table shows how we need to set the variables if we want to move the ball's vertical and horizontal pixel location, so that it goes where we want it to on screen. If you want to test if the direction is working, you can manually change isBallRight and isBallUp to either true or false in the code below. Just watch out: it won't bounce yet!

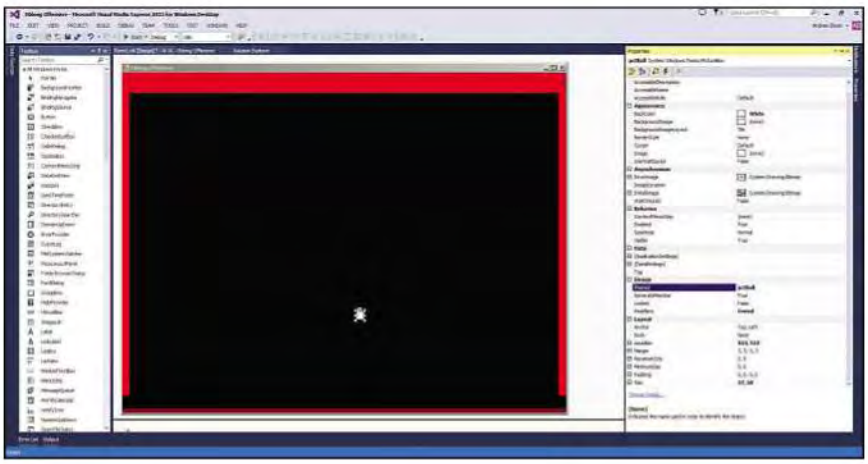
| | | |
|---|---|---|
| **Top left**<br>**+ Vertical & - Horizontal** | | **Top Right**<br>**+ Vertical & + Horizontal** |
| | **Ball**<br>**(pctBall)** | |
| **Bottom Left**<br>**- Vertical & - Horizontal** | | **Bottom right**<br>**- Vertical & + Horizontal** |

icon under the form will take us through to the Code view.

We're going to need some variables to set both the horizontal and vertical speed of our ball



▲ For each wall, we're going to use the PictureBox control from the toolbox.



▲ To add the ball, create a PictureBox called pctBall.

> "For each wall, we're going to use the PictureBox control from the toolbox"

(pctBall), as well as variables for which direction it will be moving in. Just above the tmrMovement subroutine that we've created, and underneath the name of our Class, we can insert our global variables that all subroutines can access. Enter the following variables:

```
Dim isBallRight As Boolean = True
```

```
Dim isBallUp As Boolean = False
```

```
Dim ballSpeedVertical As Integer = 3
```

```
Dim ballSpeedHorizontal As Integer = 3
```

To check what direction our ball is moving in, we're going to have two Boolean variables called isBallRight and isBallUp. As you might remember, a Boolean can have only two states: either true or false. The variable isBallRight = True will flag that the ball should be going right. If it's false, we can assume the opposite, to indicate the ball should be travelling left. The same is true of the isBallUp variable, because if it equals true it will signify an upward movement, and a false value will be the opposite downwards movement. The last two variables are going to regulate the speed of our ball (pctBall) on both the horizontal and vertical plane.

Visual Basic can move objects around in the window by increasing or decreasing their position by pixels. Because the ball is in a Timer control, it will keep moving. If you look at a flight of stairs in a house from a side angle, the shape of it will go slightly across then up, then across and up, then across and up, etc. This is exactly how our ball shape will move, changing the horizontal pixels followed by the vertical pixels. Since this happens so fast, it will appear that the ball is moving smoothly in all four diagonal directions.

We now need to get our ball moving by assigning each possible combination of our Boolean value variables to the corresponding horizontal and vertical speed in the tmrMovement subroutine:

```
If isBallRight = True Then pctBall.Left
```

```
Form1.vb   Form1.vb [Design]
OblongOffensive                                   (Declarations)
Public Class OblongOffensive

    Dim isBallRight As Boolean = True
    Dim isBallUp As Boolean = False
    Dim ballSpeedVertical As Integer = 3
    Dim ballSpeedHorizontal As Integer = 3

    Private Sub tmrMovement_Tick(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles tmrMovement.Tick

        If isBallRight = True Then pctBall.Left += ballSpeedHorizontal Else pctBall.Left -= ballSpeedHorizontal
        If isBallUp = True Then pctBall.Top -= ballSpeedVertical Else pctBall.Top += ballSpeedVertical

        If pctBall.Bounds.IntersectsWith(pctWallBottom.Bounds) Then
            isBallUp = True
        ElseIf pctBall.Bounds.IntersectsWith(pctWallTop.Bounds) Then
            isBallUp = False
        End If

        If pctBall.Bounds.IntersectsWith(pctWallLeft.Bounds) Then
            isBallRight = True
        ElseIf pctBall.Bounds.IntersectsWith(pctWallRight.Bounds) Then
            isBallRight = False
        End If

    End Sub
```

▲ We're using variables to set both the horizontal and vertical speed of our ball.

```
+= ballSpeedHorizontal Else pctBall.Left
-= ballSpeedHorizontal
```

```
If isBallUp = True Then pctBall.Top -=
ballSpeedVertical Else pctBall.Top +=
ballSpeedVertical
```

We use the .Left and .Top property of the PictureBox to give us the horizontal and vertical plane.

## Making the ball bounce

To make our ball bounce around the arena, we're going to create an If statement for left and right, and one for top and bottom. These can go under the last piece of code in the tmrMovement subroutine:

```
If pctBall.Bounds.
IntersectsWith(pctWallBottom.Bounds) Then
            isBallUp = True
   ElseIf pctBall.Bounds.
IntersectsWith(pctWallTop.Bounds) Then
            isBallUp = False
End If
```

```
If pctBall.Bounds.
IntersectsWith(pctWallLeft.Bounds) Then
            isBallRight = True
   ElseIf pctBall.Bounds.
IntersectsWith(pctWallRight.Bounds) Then
            isBallRight = False
End If
```

Every Picture object has a boundary area, which we use to check collision with another boundary using bounds.IntersectsWith(). When we use this in an If statement, as above, we can check if any collision has occurred. Now test your code and see if the ball bounces off each wall.

## Controlling the paddle

Now we have the ball bouncing, we need to create the paddle for the player to control, then assign it to the computer mouse for side-to-side movement. The paddle sprite is going to be another PictureBox control. Drag one from the toolbox onto the form, resize it to 100, 25 in the Size property and

### BASIC ANALYSIS

```
pctPaddle.Left  = e.x - (pctPaddle.
width / 2)
```

The MouseMove subroutine comes with a variable called 'e' that you can use. All you need to do is map it to the horizontal x plane (left and right). Since we're using the left side of the paddle to match it to, we need to divide the width of the paddle by two, so that it matches to the centre of the paddle, not the far left.

### TOP TIP

**Most developers will refer to subroutines as 'subs'.**


▲ The paddle sprite is another PictureBox control.

give it the name pctPaddle. Feel free to pick the colour by changing the BackColor value. Place the paddle at the bottom of the screen, just above pctWallBottom.

We now need to add the code that maps the mouse's horizontal movement (x) to the paddle. To get back to the Code view, click View on the top bar and select Code, or double-click the tmrMovement timer. There's a preset subroutine we can call for mouse movement events called MouseMove. To get to this, click on the dropdown box at the top of the Code view and select OblongOffensive Events, then in the dropdown box to the left of that select MouseMove. This will add all the necessary subroutines, start and end code. Add the following line of code with the MouseMove subroutine:

```
pctPaddle.Left = e.X - (pctPaddle.Width /
2)
```

Test the program and see if the mouse moves the pctPaddle left and right.

We now need to code the paddle so that the ball bounces off it when they collide together. To give the game an element of skill, we're also going to detect which side of the paddle has been hit and send the ball back in the opposite direction. For this, we're going to create our own subroutines, which are easy to code with the following syntax:

```
Private Sub NameofSub()

End Sub
```

**TOP TIP**

**Variables declared inside a private subroutine aren't accessible anywhere else. This is why we set the global variables at the top of the code outside of any subroutine.**

Start with the keywords Private Sub, then give it a name of your choice. Always make it descriptive; the bracket after the name of your subroutine is an opportunity to pass data and variables into it if needed. To tell the complier we're finishing the subroutine, we use the keywords End Sub.

The subroutine we're going to create will check if the ball has touched the paddle and, depending on which half of the paddle it has touched, it will bounce off in the opposite direction. We're going to call the subroutine checkPaddleBounce. Add the following code underneath the End Sub of tmrMovement, but before End Class:

```
Private Sub checkPaddleBounce()

        Dim leftOffset As Integer
```

## BASIC ANALYSIS

```
Private Sub checkBounce(ByVal collider
As PictureBox)
```

The big difference between this subroutine and the one for the paddle is that this subroutine has one of the PictureBox blocks pasted into it, which we rename 'collider'. We're being very efficient with our code here, as we could have created the code for each of the eight individual PictureBox blocks. Instead, we've created the collision code once and passed each block into it. This is why subroutines are awesome!

```
oblongoffensive.vb   oblongoffensive.vb [Design]
(OblongOffensive Events)                    MouseMove

    Private Sub oblongoffensive_MouseMove(sender As Object, e As MouseEventArgs) Handles Me.MouseMove
        pctPaddle.Left = e.X - (pctPaddle.Width / 2)
    End Sub
End Class
```

▲ Add the code that maps the mouse's horizontal movement (x) to the paddle.

```
        If pctPaddle.Bounds.
IntersectsWith(pctBall.Bounds) Then
            isBallUp = True

            leftOffset = pctPaddle.Left -
pctBall.Left

If leftOffset < -(pctPaddle.Width / 2)
Then
                isBallRight = True
            Else
                isBallRight = False
            End If
            ballSpeedHorizontal = (Rnd()
* 6) + 1
        End If
    End Sub
```

As we saw earlier, collision detection in Visual Basic uses the bounds.Intersectswith() function, which basically means the boundary of the shape touched another shape. So, pctPaddle.Bounds. IntersectsWith(pctBall.Bounds) is our conditional check if the paddle has touched the ball. If it has, we set isBallUp to true, which moves the ball back up.

To give the game increased user interaction, we need to check where on the paddle the ball has hit, so we can deflect it left or right. As there's no preset function in Visual Basic, to do this we're going to use a variable called leftOffset, which is the left value of the paddle minus the left value of the ball. We then use an If statement to check if that number is less than or greater than half the width of the paddle. We use our Boolean variable isBallRight to change the horizontal direction of the ball, by changing it to either true or false.

Our last piece of code for checkPaddleBounce() will be to randomise the speed of the deflection off the bat by giving our ballSpeedHorizontal variable a number between 1 and 7:

```
ballSpeedHorizontal = (Rnd() * 6) + 1
```

Now that we've created our subroutine, we'll need to call it from within the tmrMovement code by typing the name of it with brackets. Add the following code at the end of tmrMovement:

```
checkPaddleBounce()
```

Test your game now to see whether the paddle deflects the ball left or right.

> "Testing your game is very important to make sure everything works as intended"

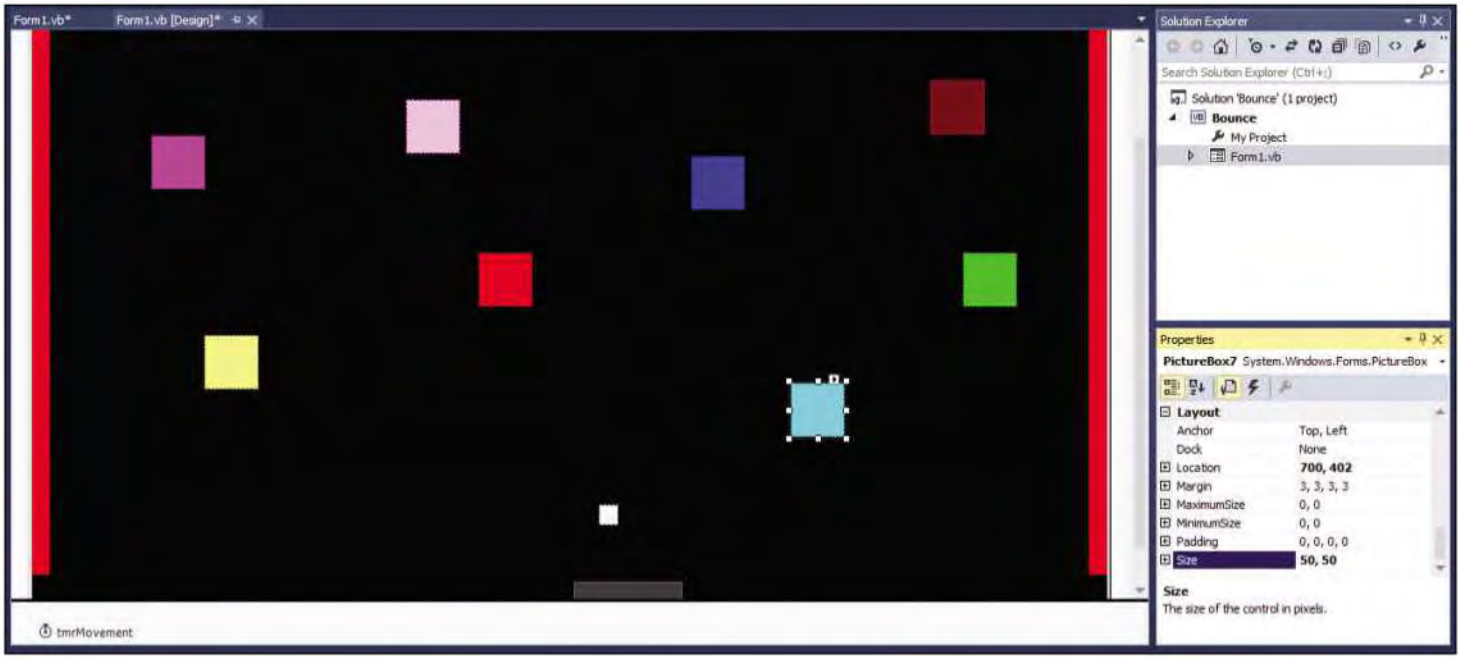
## Making blocks

We need some blocks to hit in our arena, and we're going to use the PictureBox control from the toolbox to make each block. Create eight 50 x 50 PictureBoxes and colour each one. We don't have to rename the PictureBoxes, as Visual Basic will name them sequentially for us, and we'll refer to them only once in the code.

Just like the paddle, we're going to create a subroutine called checkBounce, to check if the block has been hit. Depending where the ball hits, we need to deflect the ball in the correct direction from all four sides. Create the following subroutine:

```
    Private Sub checkBounce(ByVal
collider As PictureBox)

        Dim topOffset As Integer
        Dim leftOffset As Integer

        If collider.Bounds.
IntersectsWith(pctBall.Bounds) Then

            topOffset = collider.Top -
pctBall.Top
            leftOffset = collider.Left -
pctBall.Left

            If topOffset > 0 And
topOffset
 > leftOffset Then
                isBallUp = True
            ElseIf topOffset < 0 And
topOffset < leftOffset Then
                isBallUp = False
            End If
```

**TOP TIP**
**The order of variables doesn't matter, as long as they're declared before they're used.**




▲ Use the PictureBox control to make each block.

```
            If leftOffset < 0 And
leftOffset < topOffset Then
                isBallRight = True
            Else
                isBallRight = False
            End If
            collider.Left = -100
        End If
    End Sub
```

As we have to detect collision on the top, sides and bottom of each block, we're going to have two variables – leftOffset and topOffset – and use them in the same way as we did with the paddle, to check whether it needs to travel up or down, and left or right.

The only issue we have is that if the block has been hit, we need to make it disappear completely from the arena, so we set the x position of the PictureBox block (collider) to -100, which sends it far left out of the arena.

We now need to call our subroutine in tmrMovement next to where we called the checkPaddleBounce(), but this time we pass in each PictureBox. Add the following code to tmrMovement just above checkPaddleBounce():

```
checkBounce(PictureBox1)
        checkBounce(PictureBox2)
        checkBounce(PictureBox3)
        checkBounce(PictureBox4)
        checkBounce(PictureBox5)
        checkBounce(PictureBox6)
        checkBounce(PictureBox7)
        checkBounce(PictureBox8)
```

Test your game to see if the blocks detect the ball, bounce in the correct direction and vanish when hit.

## Creating a scoring system

To make this a proper game, we need a score that increases for every block hit and a set of lives, which will decrease every time the ball hits the bottom of the screen under the paddle. Also, we could give our game a cool title at the top, so players know the name of your amazing software!

We're going to need three label boxes from the toolbox, which will be our title, lives and score. Drag the first label onto the form from the toolbox and rename it lblLives, then change the text of the label to read 'Lives: 3'. Create a variable for our lives in our global variables section of the code (at the top) and give it the value of 3:

```
Dim lives As Integer = 3
```

To get the lives to count down, we need to add the following to the current If statement that checks if the ball has hit the bottom, in tmrMovement:

```
If pctBall.Bounds.
IntersectsWith(pctWallBottom.Bounds) Then
            isBallUp = True
            lives -= 1
            lblLives.Text = "Lives: " &
lives
```

**TOP TIP**

**Testing your program is very important to make sure everything works as intended. Check to see what happens if you win the level or lose all of your lives before you let others play the game.**


▲ We need three label boxes from the toolbox for our title, lives and score.

```
            If lives = 0 Then
                tmrMovement.Enabled =
False
                MsgBox("Game Over")
                Me.Close()
            End If
```

We've had to put in another If statement to check when the lives equal 0. This then needs to stop the timer, display a 'Game Over' message box, and close the program.

To create a scoring system, we're going to use a new label from the toolbox and call it lblScore. Replace the default label text with 'Score:0'. Next, create a new variable at the top with our other global variables:

```
Dim score As Integer = 0
```

Add the following code to the checkBounce subroutine at the end, so that each block hit will increase the score variable by 10 and display the new score in lblScore.Text:

```
     collider.Left = -100
            score += 10
            lblScore.Text = "Score: " &
score
        End If
```

The last task with the score is to create an If statement that checks if the maximum score of 80 has been achieved, effectively clearing all blocks, stopping the timer, displaying a congratulations message, then quitting the program. Add the following code at the end of tmrMovement:

```
  If score = 80 Then
            tmrMovement.Enabled = False
            MsgBox("Congratulations!  You
have cleared the level")
            Me.Close()
        End If
```

The last label box to add will just be the name of the game in the top-centre position. With all three boxes, feel free to change the fonts, background and size to match the style of your game.

Now test and play your game to see if everything works!

> "Give your game a cool title so players know the name of your amazing software!"

## Adding keyboard controls

The user's reflex with the mouse might make the game too easy, so if you want to add an extra layer of difficulty, we could change the control method of the paddle from the mouse to the keyboard. The first job is to disable the mouse control in the code, using the comment key ('') in front of that line of code, so the complier ignores it:

```
Private Sub Form1_MouseMove(sender As
Object, e As MouseEventArgs) Handles
Me.MouseMove
        'pctPaddle.Left = e.X -
(pctPaddle.Width / 2)
    End Sub
```

If done correctly, the line should turn green, and we know that this line is temporarily removed from the game without having to delete what we've done. This also means we can switch back, should you want to re-enable the mouse, by taking out the comment symbol ('') at the start of the line. This can be a useful trick when you're experimenting with an app or game.

We now need to code the paddle movement to use the keyboard. We do this using a method

### EXPERIMENT

Depending on how difficult you want to make the game, you could try changing the numbers from 20 to a smaller value, which means it will take longer to move the paddle to the right position to deflect the ball. If you really want to throw the player, you could have different values for the left and right movement, so it takes time to adapt to the game!

```
(OblongOffensive Events)
    Private Sub checkPaddleBounce()

        'Dim topOffset As Integer
        Dim leftOffset As Integer

        If pctPaddle.Bounds.IntersectsWith(p
            isBallUp = True
            leftOffset = pctPaddle.Left - pc
            If leftOffset < -(pctPaddle.Widt
                isBallRight = True
            Else
                isBallRight = False
            End If
            ballSpeedHorizontal = (Rnd() * 6
        End If
    End Sub
```

```
MouseMove
ImeModeChanged
InputLanguageChanged
InputLanguageChanging
Invalidated
KeyDown
KeyPress
KeyUp
Layout
Leave
Load
LocationChanged
LostFocus
MaximizedBoundsChanged
```

▲ Coding the paddle movement to use the keyboard.

> “You can map almost any key from the keyboard to your controls”

very similar to the one we used for the mouse movement subroutine. Go into your code and select the Form events from the top dropdown menu, and this time select KeyDown from the right dropdown box.

This creates the subroutine code needed for Keyboard events. Add the following code to the subroutine, which will map the left and right arrow to the movement of the paddle:

```
If e.KeyValue = Keys.Left Then
            pctPaddle.Left -= 20
        End If

        If e.KeyValue = Keys.Right Then
            pctPaddle.Left += 20
        End If
```

With each If statement, we're binding the left and right arrow keys from the keyboard to increase the left-side position of the paddle by 20 or reduce the left-side position by 20. You can map almost any key from the keyboard to your controls by changing Keys.Left to something like Keys.A. Alternatively, you can just type Keys. and Intellisense will give you a full list of keys to choose from.

If you want to move back to mouse control, you can comment out the code we've just entered – with the comment key (') on each line of both If statements – then uncomment the mouse control code. This method is really effective for testing parts of your code with new ideas, while not deleting anything you've previously coded.

## Expanding the game

There are several ways you can expand the game to make it more challenging for the player:

- Change the speed of the ball (pctBall) for each block you've hit
- Shorten the width of the paddle (pctPaddle) every time your score increases
- Create more blocks of various sizes
- Change the form size and the position of the blocks
- Create 'power up' blocks that when collected could change the speed of the ball, alter the length of the paddle, increase the score or randomly choose one of the previous options.
- Randomise the Y coordinate position of the ball in form_load; this is code that runs when the form is loaded. You can access this by double-

```
(OblongOffensive Events)                                   KeyDown
Private Sub OblongOffensive_KeyDown(sender As Object, e As KeyEventArgs) Handles Me.KeyDown
    If e.KeyValue = Keys.Left Then
        pctPaddle.Left -= 20
    End If

    If e.KeyValue = Keys.Right Then
        pctPaddle.Left += 20
    End If
End Sub
```

▲ Map the left and right arrow to the movement of the paddle.

clicking the background of your form: pctBall.Top = (200 * Rnd()) + 30

- Randomise the Y coordinate position of the blocks in form_load: pctBall.Top = (200 * Rnd()) + 30

## Final code

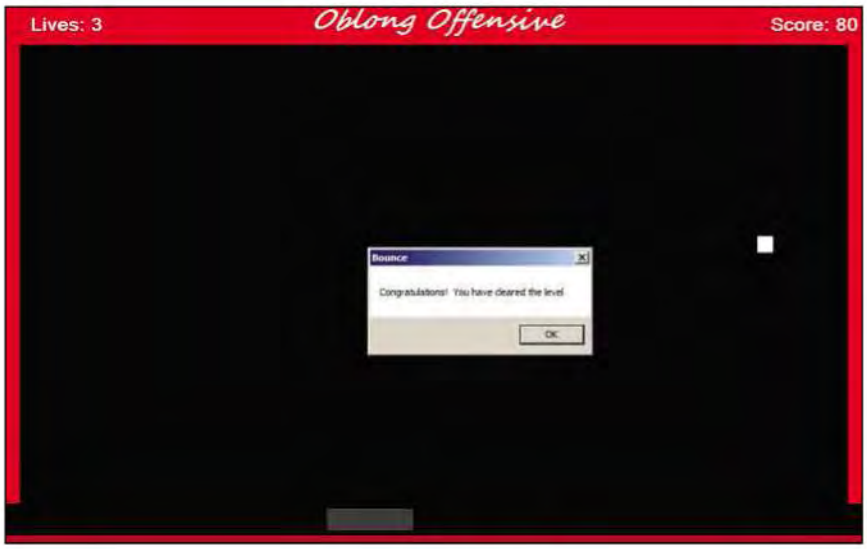
```
Public Class Form1

    Dim isBallRight As Boolean = False
    Dim isBallUp As Boolean = True
    Dim ballSpeedVertical As Integer = 3
    Dim ballSpeedHorizontal As Integer =
3
    Dim lives As Integer = 3
    Dim score As Integer = 0

    Private Sub tmrMovement_Tick(ByVal
sender As System.Object, ByVal e As
System.EventArgs) Handles tmrMovement.
Tick

        If isBallRight = True Then
pctBall.Left += ballSpeedHorizontal Else
pctBall.Left -= ballSpeedHorizontal
        If isBallUp = True Then pctBall.
Top -= ballSpeedVertical Else pctBall.Top
+= ballSpeedVertical

        If pctBall.Bounds.
IntersectsWith(pctWallBottom.Bounds) Then
            isBallUp = True
            lives -= 1
            lblLives.Text = "Lives: " &
lives

            If lives = 0 Then
                tmrMovement.Enabled =
False
                MsgBox("Game Over")
                Me.Close()
            End If
        ElseIf pctBall.Bounds.
IntersectsWith(pctWallTop.Bounds) Then
            isBallUp = False
        End If

        If pctBall.Bounds.
IntersectsWith(pctWallLeft.Bounds) Then
            isBallRight = True
        ElseIf pctBall.Bounds.
IntersectsWith(pctWallRight.Bounds) Then
            isBallRight = False
        End If

        checkBounce(PictureBox1)
        checkBounce(PictureBox2)
        checkBounce(PictureBox3)
        checkBounce(PictureBox4)
        checkBounce(PictureBox5)
        checkBounce(PictureBox6)
        checkBounce(PictureBox7)
        checkBounce(PictureBox8)

        checkPaddleBounce()

        If score = 80 Then
            tmrMovement.Enabled = False
            MsgBox("Congratulations!  You
have cleared the level")
            Me.Close()
        End If

    End Sub

    Private Sub checkBounce(ByVal
collider As PictureBox)

        Dim topOffset As Integer
        Dim leftOffset As Integer

        If collider.Bounds.
IntersectsWith(pctBall.Bounds) Then
```

### DID YOU KNOW?

The original 1976 Breakout was inspired by one of the very first arcade hits, Atari's 1972 game, Pong. Breakout was designed by Nolan Bushnell and Steve Bristow, but the code was written by Steve Wozniak with help from Steve Jobs, who would later go onto found Apple.

> There are several ways you can expand the game to make it more challenging

```
            topOffset = collider.Top -
pctBall.Top
            leftOffset = collider.Left -
pctBall.Left

           If topOffset > 0 And topOffset
> leftOffset Then
                isBallUp = True
            ElseIf topOffset < 0 And
topOffset < leftOffset Then
                isBallUp = False
            End If
            If leftOffset < 0 And
leftOffset < topOffset Then
                isBallRight = True
            Else
                isBallRight = False
            End If

            collider.Left = -100
            score += 10
            lblScore.Text = "Score: " &
score
        End If
    End Sub

    Private Sub checkPaddleBounce()

        Dim topOffset As Integer
        Dim leftOffset As Integer

        If pctPaddle.Bounds.
IntersectsWith(pctBall.Bounds) Then

            topOffset = pctPaddle.Top -
pctBall.Top
            leftOffset = pctPaddle.Left -
pctBall.Left

           If topOffset > 0 And topOffset
> leftOffset Then
                isBallUp = True
            ElseIf topOffset < 0 Then
                isBallUp = False
            End If
            If leftOffset < -(pctPaddle.
```





▲ Test and play your game to see if everything works!

```
Width / 2) And leftOffset < topOffset  Then
                isBallRight = True
            Else
                isBallRight = False
            End If
            ballSpeedHorizontal = (Rnd()
* 6) + 1
        End If

    End Sub

    Private Sub OblongOffensive_
MouseMove(sender As Object, e As
MouseEventArgs) Handles Me.MouseMove
        pctPaddle.Left = e.X -
(pctPaddle.Width / 2)
    End Sub
End Class
```

# Build a Visual Basic app

Now you're getting to grips with Visual Basic, we can start to code more complex and exciting apps, such as this image viewer with built-in slideshow

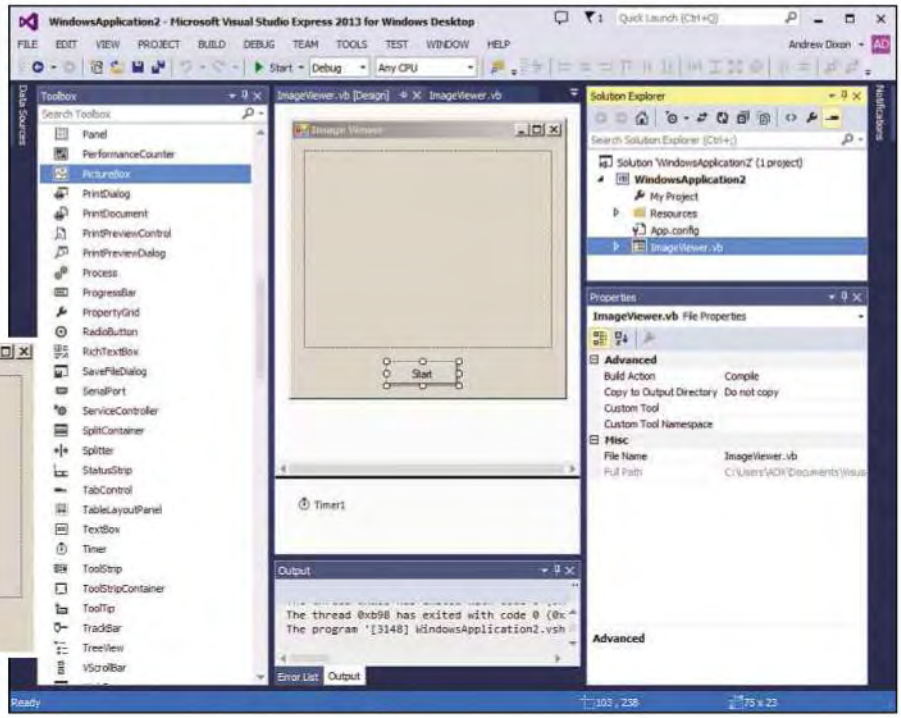
**WHAT YOU'LL LEARN**

- How to use variables
- How to display images
- How to use the timer
- How to restyle the user interface
- How to add password protection to an app

Hopefully you're now ready to try something a little more complicated. To start with, we're going to create an image slideshow application that will use the PictureBox control, Timer control, and the Project Resource folder. This little app will import images from a computer and display them as a slideshow in a random order on a continuous loop.

Create a new project and select Windows Forms Application. Our form needs three basic components from the toolbox – PictureBox, Button and Timer. All of these controls can be found in the Toolbox menu on the left-hand side of the screen. When you've selected one, you can draw the control on the form so you have full control over the size, layout and position of each component.

You can also change the name of the form by selecting it with a left mouse click, then choosing Text from the Properties window on the lower right-hand of the IDE. By default, it's named Form1: change it to something like frmPictureViewer. The only toolbox control we're going to rename is our button, which we can give the name btnStart, then change the button text to Start.




▶ Our form needs three basic components from the toolbox – PictureBox, Button and Timer.


▲ Change the button text to Start.

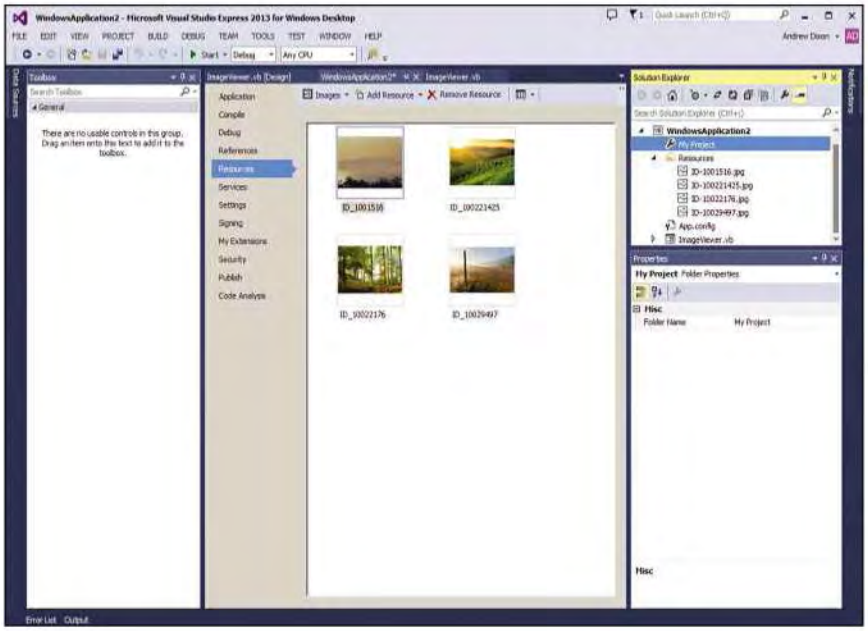
**TOP TIP**

**The VB.net Property window is dynamic and will change depending on what you click on in the form. Make sure you click on the correct object first to change its properties.**

## DATA TYPES

Unlike SmallBASIC, VB needs to have a data type declared with the variable so the computer knows what type to expect, and can allocate the correct amount of memory. There are many different data types, but here are some of the most frequently used:

| DATA TYPE | VALUE | EXAMPLE |
|---|---|---|
| **Integer** | Whole numbers | -32, 45, 1024 |
| **Single** | Decimal numbers | 3.145 |
| **String** | Words (Text) | Kevin, Quirky |
| **Boolean** | True or False | True, False |



▲ We need to add some images to our Resources folder, to be added into the random sequence.

You can drag the Timer icon on the form, but you won't see anything get added. Only the name Timer1 will appear underneath the form, with a Stopwatch icon, to show a timer has been added. The Timer control can repeat code at set time intervals, which will be useful when we want to display images for a certain amount of time. The interval value for the Timer control is measured in milliseconds, so every 1,000 ticks are equal to one second.

We also need to add images to our Resources folder, so that we have a bank of them ready to be added into the random sequence. Above the Properties window on the right of the screen, you'll see the Solution Explorer. This shows all of the forms you've created so far, and you should also find an option called My Project, which when double-clicked will give access to all of the options for the current application you're making. Find the Resources option, click 'Add resource' and finally select 'Existing resource'. This gives you a window to select the image files you'd like in your application. To keep things simple, just choose four images to import.

> "My Project will give access to all of the options for the current app you're making"

We're also going to declare a single variable for our application, as it will store a random number in memory that we can use to select which image will be displayed. All variables in Visual Basic start with Dim, then the name of the variable, and finally what data type. In this case, it will be an Integer (a whole number, without any decimal point):

```
Dim iCounter As Integer
```

Randomising a number is easy; you can call the Randomize() function to assign our variable a random number. The first number, 3, is the maximum we want to randomise, then we multiply that by Rnd(), which generates a random decimal number between 0 and 1. We add the +1 to the end, because if it generates 0 it will move to 1 instead:

```
Randomize()
```

```
iCounter = (3 * Rnd()) + 1
```

We then want to resize any given image to the width and height of our PictureBox control, so that all images appear with the same dimensions:

```
PictureBox1.SizeMode =
PictureBoxSizeMode.StretchImage
```

The timer now needs to be set, so it knows

## BASIC ANALYSIS

```
PictureBox1.Image = My.Resources.ID_1001516
```

The line of code used to add images to the Picture window is simple: we first refer to the PictureBox by its name and property, picturebox1.image, then give it the location of the image file that we've uploaded to the Resources folder. In this example, our image was called ID_1001516. We never use the image file extension .jpg or .png; this would confuse Visual Basic, as most code uses a full stop to access further functions and properties.

how long to display each image until the next random picture. The interval number value is in milliseconds. Our timer has 3000, so it will display for three seconds:

```
Timer1.Interval = 3000
```

Everything is now ready, so we can add the code that will use our random number, and assign it to a particular picture with a basic If statement:

```
If iCounter = 1 Then
            PictureBox1.Image =
My.Resources.ID_1001516
        ElseIf iCounter = 2 Then
            PictureBox1.Image =
My.Resources.ID_100221425
        ElseIf iCounter = 3 Then
            PictureBox1.Image =
My.Resources.ID_10022176
        Else
            PictureBox1.Image =
My.Resources.ID_10029497
End If
```

This is all the code we'll need inside the Timer1 control. Our final step will be to add code to the Start button (btnStart), which will activate the timer. Double-click the Start button and add the following code inside it. This will set the Timer Interval to a quicker value; otherwise, we'd have to wait three seconds for the first image to appear. To activate the timer and effectively start the program, we set the Enabled property to true:

```
Timer1.Interval = 10
```

```
Timer1.Enabled = True
```

Run the program with the Start play button on the top bar, or press F5, when you have all the code in the correct place:

```
Public Class Form1
```

```
    Private Sub Timer1_Tick(sender As
Object, e As EventArgs) Handles Timer1.
Tick
        Dim iCounter As Integer
        Randomize()
        iCounter = (3 * Rnd()) + 1
        PictureBox1.SizeMode =
PictureBoxSizeMode.StretchImage
        Timer1.Interval = 3000
```

```
        If iCounter = 1 Then
            PictureBox1.Image =
My.Resources.ID_1001516
        ElseIf iCounter = 2 Then
            PictureBox1.Image =
My.Resources.ID_100221425
        ElseIf iCounter = 3 Then
            PictureBox1.Image =
My.Resources.ID_10022176
        Else
            PictureBox1.Image =
My.Resources.ID_10029497
        End If
    End Sub
```

```
    Private Sub BtnStart_Click(sender As
Object, e As EventArgs) Handles BtnStart.
Click
        Timer1.Interval = 10
```

## CODING KEYWORDS

**GUI:** Graphical User Interface. The layout, size and appearance of your application, and all the buttons, icons, menus and sliders you use to control it.

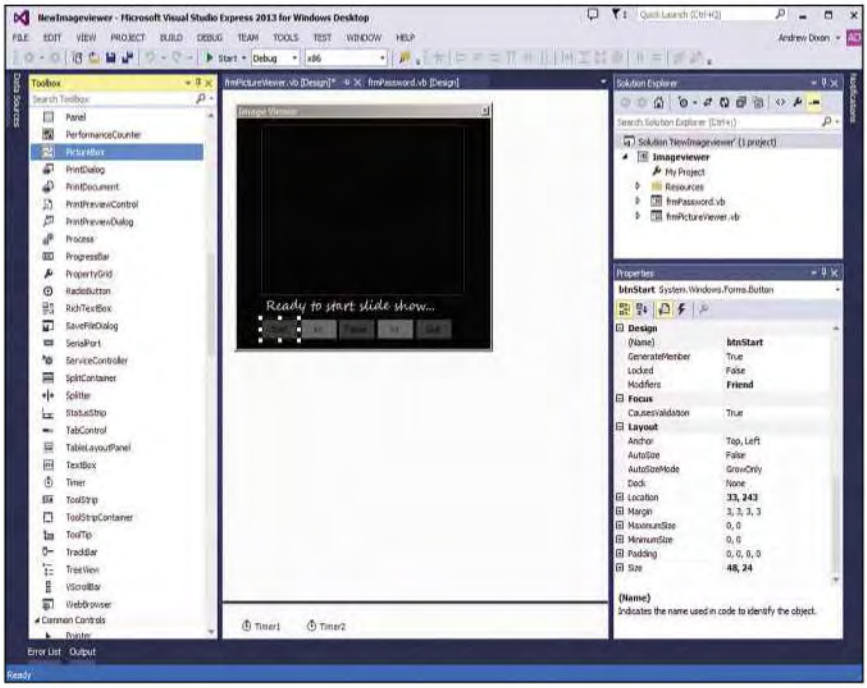
## EXPERIMENT

Try adding more pictures to the Resource folder, then change the code to accommodate the new images. For each new image, you'll have to increase the random number generated, currently (3 * Rnd()) + 1, then add to the PictureBox If statement using Elseif. The last image added will just be Else not Elseif.


▲ Our program randomly displays one of four images.

## TOP TIP

**The two main ways to display text in Visual Basic are through a textbox or a label box. The textbox allows the user to enter information, while the label box is used just to display text.**




▲ Add some style and functionality to your app.

```
        Timer1.Enabled = True
    End Sub
End Class
```

Our finished program will now display one of the four images in a random order every three seconds. We can change the speed of the rotation by altering the value of Timer1.Interval; the higher the number, the longer it will stay on each image before moving to the next.

Just remember that our pictures used filenames like ID_1001516. Yours will be different, so make sure you use the same names when you're coding the PictureBox1.image section as the names inside your Resources folder. Intellisense is useful here, and can help you get the job done faster.

## Changing styles

We now have a good working prototype, but we can give it some style and functionality. Let's first work on the GUI of our application.

Visual Basic always defaults to a drab grey colour, so to change our background we need to select the form with a left mouse click and find BackColor from the Properties menu (on the bottom right) and change this to black. Next, we're going to change the window appearance to give it a sleeker look with no icon or minimise buttons. Find FormBorderStyle in the Properties menu and select FixedToolWindowi. This means we can move the image viewer around the screen, but the user can't resize the window with the mouse.

Our Start button now looks out of place, with the cool new look we're giving our image viewer program, so let's change its appearance. Make the following changes from the Properties window: change FlatStyle to 'popup', and BackColor to a dark grey. To change the font colour of the button, you need to choose a colour from ForeColor.

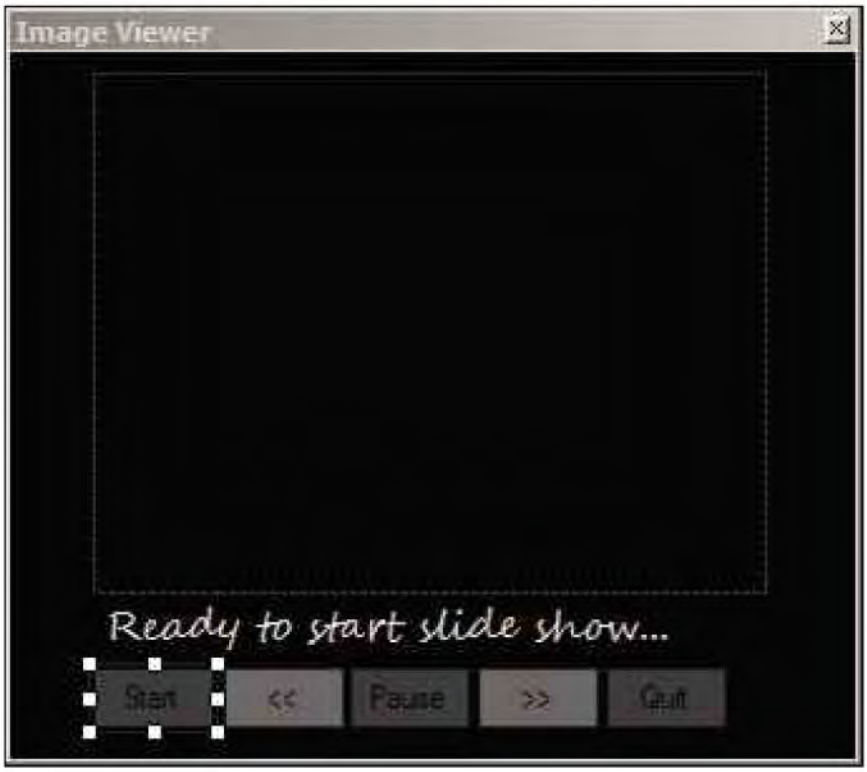
The last change to our form is a caption underneath each picture. To do this, we'll need a label box from the toolbox. Position this underneath the photo, and change the colour and font to suit your theme. By default, we can add the phrase "Ready to start slide show..." by editing the Text property of the label. To set the caption when each image is loaded, we're going to edit the If statement in Timer1 with the highlighted code, but the message can be changed to suit your images:

> "Change the speed of the rotation by altering the value of the Timer1.Interval"

```
If iCounter = 1 Then
            PictureBox1.Image =
My.Resources.ID_1001516
            Caption.Text = "A beautiful
image"
        ElseIf iCounter = 2 Then
PictureBox1.Image = My.Resources.
ID_100221425
            Caption.Text = " I love the
colours here "
        ElseIf iCounter = 3 Then
            PictureBox1.Image =
My.Resources.ID_10022176
            Caption.Text = "I wish I was
back there now!"
        Else
PictureBox1.Image = My.Resources.
ID_10029497
            Caption.Text = "Simply
stunning"
        End If
```

### EXPERIMENT

If you want to change the speed of the fade-in effect, try changing the value in Me.Opacity += 0.01 to more decimal places e.g. 0.001.




▲ Our toolbar's buttons can manually change the speed of the image transitions.

Visual Basic doesn't give us much in the way of applying effects to our pictures, but we do have an Opacity property with our form that can give a fade-in effect. To use this, we need to add another Timer control and use a Do While loop to check the opacity value, before gradually increasing it until it gets to 1, which is fully visible. Add a new timer (Timer2) from the toolbox and add the following code:

```
Timer2.Interval = 30
    Do While Me.Opacity < 1
      Me.Opacity += 0.01
    Loop
```

Every time we load a new random picture, we'll need to reset the opacity down to a nearly invisible value, then our Do While loop in Timer2 will kick in and slowly return the value back to completely visible (1). Add the following code to the Timer1 PictureBox If statement:

```
        If iCounter = 1 Then
Me.Opacity = 0.001
              PictureBox1.Image =
My.Resources.ID_1001516
Caption.Text = "A beautiful image"
        ElseIf iCounter = 2 Then
Me.Opacity = 0.001
              PictureBox1.Image =
My.Resources.ID_100221425
Caption.Text = "I love the colours here"
        ElseIf iCounter = 3 Then
Me.Opacity = 0.001
              PictureBox1.Image =
My.Resources.ID_10022176
Caption.Text = "Wish I was back there
now!"
        Else
              Me.Opacity = 0.001
              PictureBox1.Image =
My.Resources.ID_10029497
Caption.Text = "Simply stunning"
        End If
    End Sub
```

We'll also need to edit our Start button (btnStart), so that we now start to enable both timers. Add the following to the existing code:

```
Timer2.Enabled = True
```

## Making a toolbar

For our next change, we're going to create a custom toolbar with buttons that can manually change the speed of the image transitions to be faster or slower than our default value. We'll also add Pause and Quit buttons. All of our buttons will have the same style as our Start button, but we can alternate our grey colours to make things more interesting.

Our Pause button will effectively stop both timers, so any picture currently onscreen will stay there until the Start button is pressed again. Add the following code by double-clicking on the Pause button on the form:

```
Timer1.Enabled = False
```

```
Timer2.Enabled = False
```

The Quit button will close our app. Double-click the Quit button and add the following line of code:

```
Close()
```

The Speedup button (>>) will shorten the Timer1.interval value by 500 milliseconds (half a second), but as we've mentioned before we can't go below 0 or we'll get a critical error and our app will crash. To prevent this, we're going to put an If statement to check if the interval value has dropped below 1000 and, if it has, we'll hide the button, so the user can't click it anymore. Double-click the Slowdown button (<<) and add the following code:

```
Timer1.Interval = Timer1.Interval - 500
```

**TOP TIP**
**Secure passwords should contain a combination of upper- and lowercase characters, as well as numbers and symbols. The most guessed passwords are 1233456, password, qwerty, iloveyou, letmein and monkey!**

> Our Slowdown button will increase the interval value of the Timer1 by 500

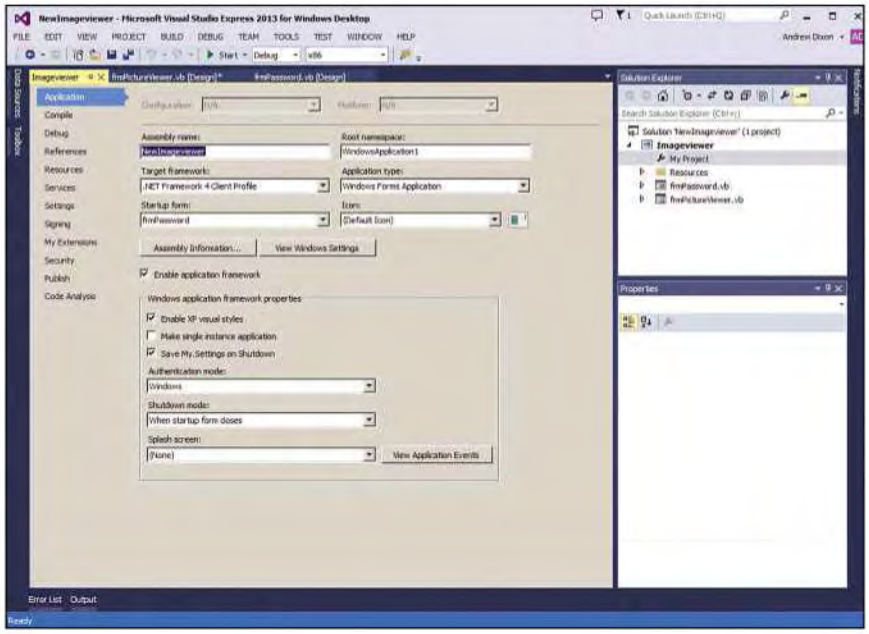
```
    If Timer1.Interval < 1000 Then
       btnSpeedUp.Visible = False
    End If
```

Our Slowdown button will increase the interval value of the Timer1 by 500 milliseconds. We'll also need a short If statement that brings back the visibility of the Speedup button if it's been hidden by the other speed control button. Double-click the Slowdown button and add this code:

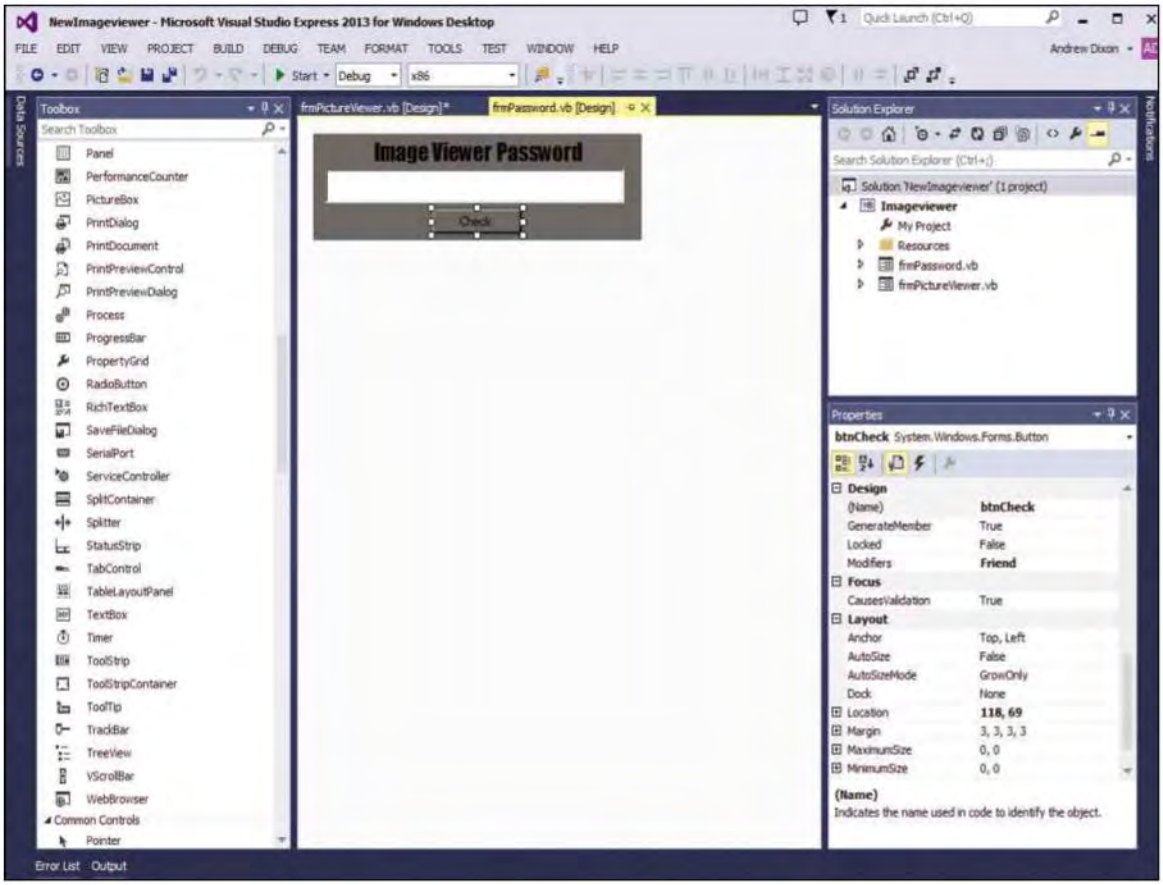
```
Timer1.Interval = Timer1.Interval + 500
    If Timer1.Interval > 1000 Then
       btnSpeedUp.Visible = True
    End If
```

## Final code

```
Public Class frmPictureViewer

    Private Sub Button1_Click(ByVal
sender As System.Object, ByVal e As
System.EventArgs) Handles btnStart.Click
        Timer1.Interval = 3000
        Timer1.Enabled = True
        Timer2.Enabled = True
        Caption.Text = "Getting ready to
start...."
    End Sub

    Private Sub Timer1_Tick(ByVal sender
As System.Object, ByVal e As System.
EventArgs) Handles Timer1.Tick
        Dim iCounter As Integer
        Randomize()
        iCounter = (4 * Rnd()) + 1
        PictureBox1.SizeMode =
PictureBoxSizeMode.StretchImage

        If iCounter = 1 Then
Me.Opacity = 0.001
             PictureBox1.Image =
My.Resources.ID_1001516
Caption.Text = "A beautiful image"
        ElseIf iCounter = 2 Then
Me.Opacity = 0.001
             PictureBox1.Image =
My.Resources.ID_100221425
Caption.Text = "I love the colours here"
        ElseIf iCounter = 3 Then
Me.Opacity = 0.001
             PictureBox1.Image =
My.Resources.ID_10022176
Caption.Text = "Wish I was back there
now!"
        Else
             Me.Opacity = 0.001
             PictureBox1.Image =
My.Resources.ID_10029497
Caption.Text = "Simply stunning"
        End If
    End Sub

    End Sub

    Private Sub Timer2_Tick(ByVal sender
As System.Object, ByVal e As System.
EventArgs) Handles Timer2.Tick
        Timer2.Interval = 30
        Do While Me.Opacity < 1
            Me.Opacity += 0.01
        Loop
    End Sub

    Private Sub btnQuit_Click(ByVal
sender As System.Object, ByVal e As
System.EventArgs) Handles btnQuit.Click
        Close()
    End Sub
```



▲ Under the Application menu, you'll find a Startup form option, where you can select any form you've made so far.




▶ To complete our image viewer, we've added a password screen.

```
    Private Sub btnPause_Click(ByVal
sender As System.Object, ByVal e As
System.EventArgs) Handles btnPause.Click
        Timer1.Enabled = False
        Timer2.Enabled = False
    End Sub

    Private Sub btnSlowDown_Click(ByVal
sender As System.Object, ByVal e As
System.EventArgs) Handles btnSlowDown.
Click
        Timer1.Interval = Timer1.Interval
+ 500
        If Timer1.Interval > 1000 Then
            btnSpeedUp.Visible = True
        End If
    End Sub

    Private Sub btnSpeedUp_Click(ByVal
sender As System.Object, ByVal e As
System.EventArgs) Handles btnSpeedUp.
Click
        Timer1.Interval = Timer1.Interval
- 500
        If Timer1.Interval < 1000 Then
            btnSpeedUp.Visible = False
        End If
    End Sub

End Class
```

## Adding security

To complete our image viewer, we're going to add a security feature that will stop any unwanted users from accessing our slideshow. We'll need to add another form, which you can access under the Project menu, selecting Add Windows Form. We'll call this form frmPassword.

On our password form, we're going to add a label for a title, which can say Image Viewer Password, by changing the Text property. We'll also need a textbox, call it txtPassword, so the user can type in the password and a button (btnCheck) to check if the password is correct. You can apply all the style changes we've done for frmImageViewer, such as colour, fonts and form style, to give it the feel that it all belongs in one application.

We'll set the password in the code using an If statement. If the user enters the correct password, we're going to hide the visibility of our password form, then enable the visibility of our frmPictureViewer. If the user enters anything but

> “Why not make your own logo that represents you or your software studio?”

the password, we'll use a neat pop-up box function called MsgBox that can display a message in a pop-up screen with a single OK button, to cancel the message and return back to the password screen.

When you've created and edited the button, double-click it and add the following code:

```
If txtPassword.Text = "letmein" Then
            Me.Visible = False
            frmPictureViewer.Visible =
True
        Else
            MsgBox("Unauthorised!")
        End If
```

In this example, we've used the password 'letmein', but feel free to change it to something more personal and certainly more secure!

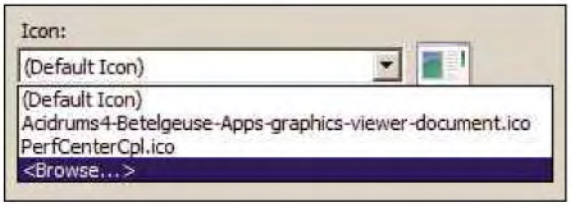
It's no good having a password box if anyone can see what you're typing onscreen, so we're going to stealth our text behind a symbol. Simply select our textbox and find the property value PasswordChar on the right-hand side of the screen. Enter an asterix symbol (*) in the field, and this will instruct the computer to display that symbol whenever the user types on the keyboard.

As we've made this form last, the program won't be the first form we see, as Visual Basic will always default to the first form created – in this case, the form with our slideshow. To change the order, we need to go back to My Project in the Solutions Window. Under the Application menu, you'll find a Startup form option, where you can select any form you've made so far. Change this to frmPassword.

Now test your program by clicking Run on the toolbar or pressing F5.

## Making your app ready to run

We have a good working program, but don't really want to load Visual Studio, then run the program each time we want to see our slideshow. Whenever you run your code, Visual Studio compiles it in a single executable file (EXE). Basically, it takes the Basic code we write and converts it into a series of instructions that your PC or laptop can understand.


▲ Give your application or game its own icon for a more professional finish.

To find the executable file, go to the Project file on your computer, and you'll see a folder called Bin. Inside, there's another folder called Debug. This will contain the single EXE file of your complied code. You can drag this out of the folder and onto your Windows desktop.

To run an EXE file, just double-click on it. Your program will then appear and work its magic. The great thing is that you now have a version of your game or app that other people can run, whether they have Visual Basic installed or not. This is a great way of distributing your app or game.

## Changing an icon file

The icon for your executable file will be the default Windows icon, but we can change that to an exciting graphic that better represents your application by using the Project Settings window, where we changed the load order of the forms. Next to this, there's a dropdown box under the heading Icon, where we can browse for a new image or select one we've already used.

Icon files aren't like normal image files such as GIF, JPEG or PNG. They have their own file type, ICO, which indicates they're icons used on files. You either need to download some free ICO files for your application from the internet, or you can convert an image you've already made using an image editor that supports exporting to the ICO format. If you need something to do the job, we'd recommend the popular free image editor, IrfanView. You can get this from www.irfanview.com. Click on browse and find a new icon image.

When you've changed the icon for the application, run the program again with F5 or the Run button. Check the Debug folder, and the new EXE file will be there, with your new icon image ready for you to use or distribute to others.

Giving your application or game its own icon won't change the way it works, but it will add a more professional finish to the final product. You could even make your own logo that represents you or your software studio, which you can use within all of your programs. It will give them some identify and tell the world who's made this awesome software! ●

**TOP TIP**
**If you want, you can rename the EXE file with a right-click, then select Rename.**

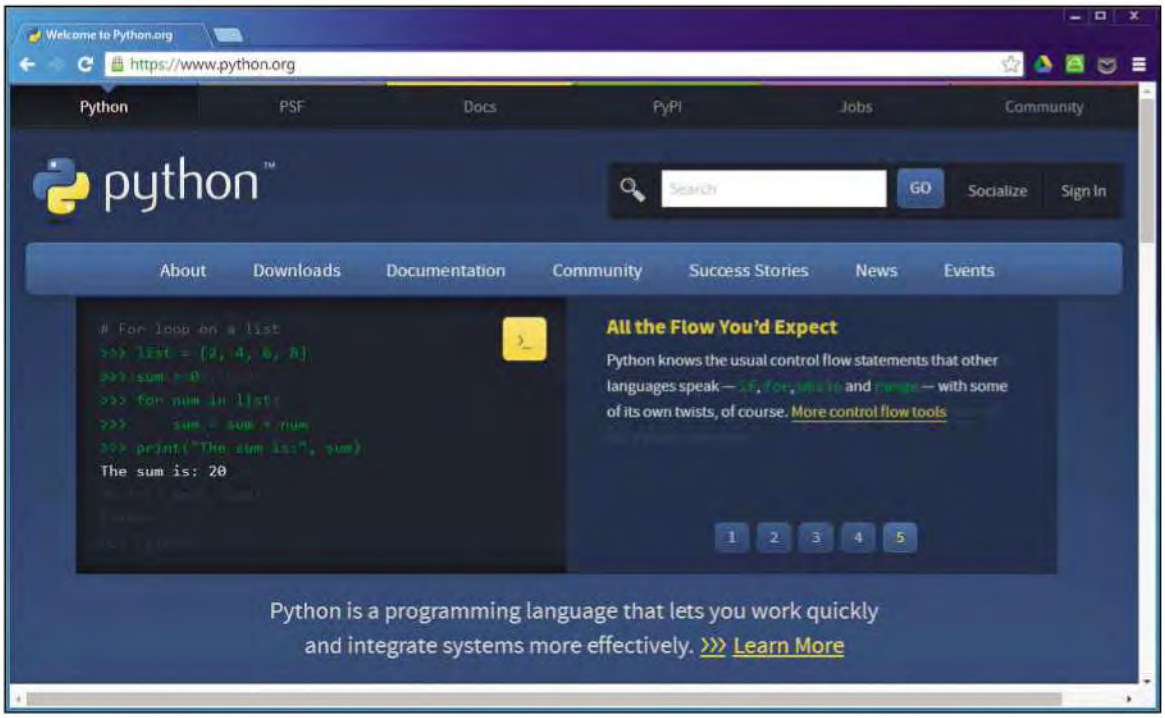
# Where do you go next?

You might have completed the last project, but your journey into code has only just begun. What you do next is up to you

Congratulations! If you've made it this far, you're already developing a good working knowledge of how programs work, and you should be able to start programming your own apps and games. The best way to learn about programming isn't to learn a lot of theory or read a lot of books, although neither ever hurts. No, the best way to learn about programming is to keep coding. Sometimes you'll make something without any effort, and at other times you'll struggle and make mistakes. By learning how to fix problems and debug your own programs, you'll learn even more about programming and discover ways to make your programs more efficient. The time and effort you put in is never wasted.

## Mastering new languages

Now that you've had a taste of Scratch, Small Basic and Visual Basic, you might want to expand your talents to different platforms and devices. For instance, you may want to develop apps for mobile phones or tablets, or start working on desktop programs for MacOS X or Linux. Choosing what device and operating system you

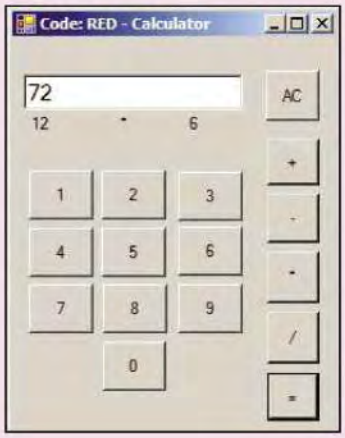



▶ Python is now used in many schools and also professionally across a whole variety of industries.

**TOP TIP**
**Learning different languages will make you a very skilled and employable developer!**

## PROJECT IDEAS

To get better at programming, you need to practise, first writing simple applications, then working your way up to bigger programs and games. Here are some simple ideas for projects in Visual Basic using the skills that you've learnt from this book:

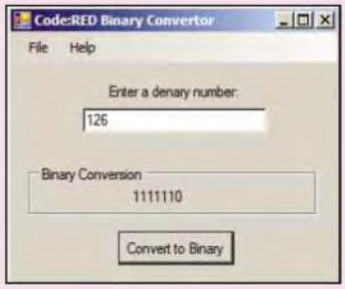



### VISUAL CALCULATOR

Use the number buttons to enter values into the textbox, then include buttons for addition, subtraction, multiplication and division. See if you can mimic some of the popular functions you'd find on a calculator. The AC button should clear all boxes and variables. If you display the variables underneath the textbox, it makes testing a whole lot easier.

### BINARY CONVERTOR

This project takes a standard (or denary) number and converts it into




computer-friendly binary. See if you can work out how to do it.

### VISUAL DICE

This bite-sized project creates a random number from 1 to 6, then displays the correct image for the dice roll. We covered producing random numbers earlier on. See if


you can remember where, then put that knowledge to good use. You can expand this to form different types of dice, which is great for virtual board and role-playing games.


### STOPWATCH

It might sound complex, but all this project needs to do is use a timer to count up or down on label boxes. Change the speed of the timer or set the initial time by using + and – buttons on hours, minutes and seconds. The Pause button should stop the Timer, and the Reset button should set all numbers back to 0 and the speed back to real time. You could even use a MsgBox to make alarms. Can you work out how to pull this off?

### TEMPERATURE CONVERTOR

This project takes a number from the user and pastes it into one of two functions to convert between Celsius and Fahrenheit. The calculations for converting are simpler than you might think. The selection buttons


are called radio buttons and they have a Boolean value for on and off, which makes them easy to use with If statements.

### WEAPON GENERATOR

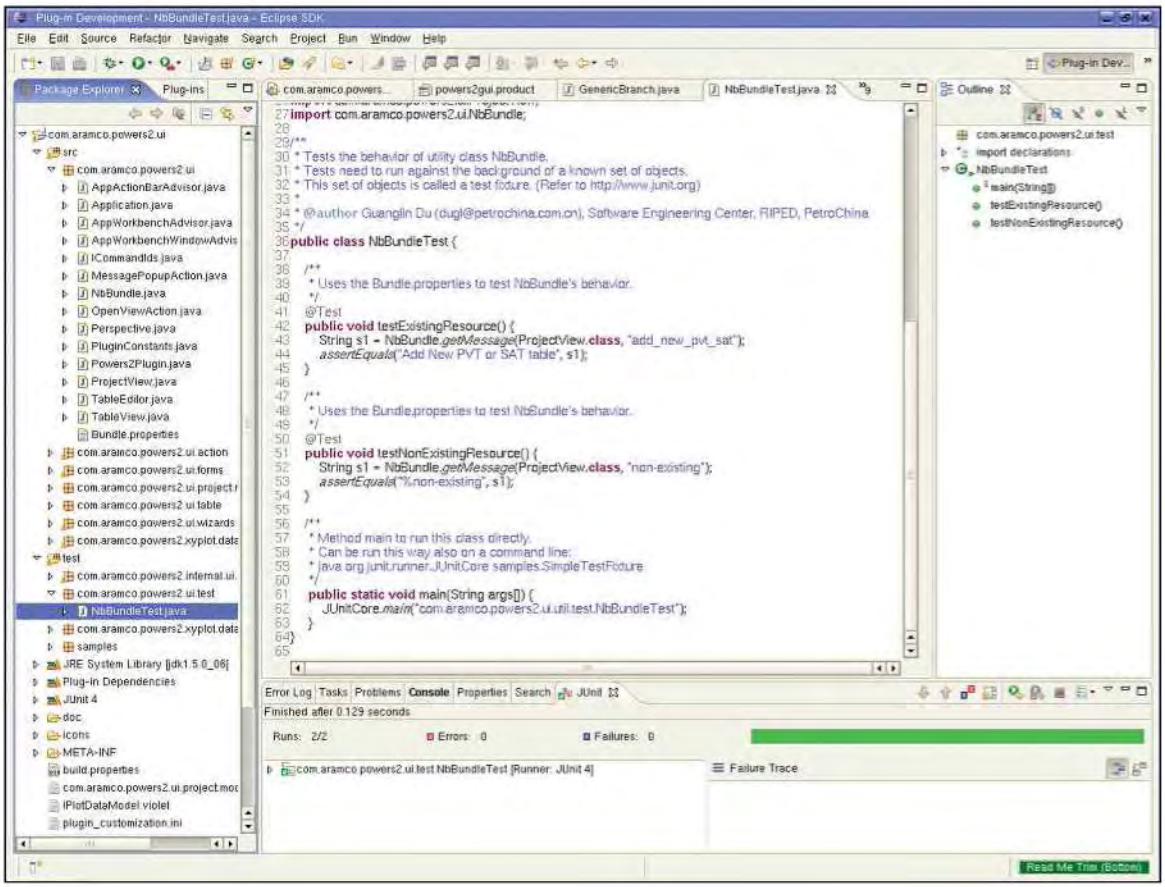
This fun project creates random statistics for weapons in a video game. Working within certain boundaries, it can simulate the 'loot drops' you find in many action games and RPGs. Have fun with the name by having a list of adjectives and a list of nouns, such as animals, and randomly putting them together. High rolls could generate extra elemental damage.


want to develop on will determine which language to learn.

To develop for any Apple device, you'll need to learn a language called ObjectiveC. This language is based on the C style of programming, with some new features added by NextStep and Apple. To start coding, you'll need either a MacBook or an iMac and the Xcode software, which can be downloaded for free from the Apple store. Using Xcode, you can develop for the Apple desktop, mobile iOS phones and tablets. The only issues with ObjectiveC and Xcode are that they both have a steep learning curve, and to publish your applications on the Apple store you'll need to be a registered Apple developer, which costs £60 a year.

The other major mobile platform is Android, and you can program applications using the Android SDK (Software Development Kit). You'll also need an Integrated Development Environment, or IDE, which provides you with all the tools and software you need to write and debug code. We'd recommend Eclipse, which is a powerful IDE that can be used with many different types of programming languages. The Android SDK and Eclipse will run on most major OS platforms, and more importantly, it can be downloaded for free at developer.android.com/sdk/index.html.

▶ Download the Java SDK and you can use Eclipse as an IDE.




## Coding for the web

HTML (HyperText Markup Language) is the code used to create web pages. It's not, strictly speaking, a programming language, but it's useful to know when developing internet-dependent applications and games. To view the HTML code of any website, you can find the View Source option in your browser, usually under the View menu. This will pull up a window showing all the code at work in your favourite websites. There are many types of software on the market to develop HTML, but as long as you have a text editor, such as Notepad, you can code using that and your internet browser will display the result.

Fancy making web games or interactive websites? JavaScript is a client-side language – it runs in your browser rather than on the server that powers the website – that's used primarily for internet browsers and webpages. It fits nicely into HTML to give your webpages interactivity and, just like HTML, you don't have to install software to get coding. If you want to work with a more fully featured IDE, several good ones are available for free. The syntax for JavaScript is simple to understand and, while the name suggests otherwise, it has nothing to do with the Java programming language.

## Java and Python

Not that there's anything wrong with Java, one of the most popular cross-platform programming languages. The code can look complex, and it does have a steep learning curve. It takes much of its syntax style from C and C++, but if you can handle it you can make plenty of great games and apps for free. Simply download the Java SDK and you can use Eclipse as an IDE. To find out more about Java, visit docs.oracle.com/javase/tutorial/

If you want a more approachable option, try Python. It's now used in many schools and also professionally across a whole variety of industries. What seems like simple syntax quickly turns into a powerful cross-platform language that can create games and apps. The Python software is free to download and will work on most operating systems. The only confusing part is that there are two versions; the older version 2, which people still love and view as perfection, and the newer version 3, which attempts to modernise the language, but by doing that changes the syntax and commands.

There are plenty of other great languages that we don't have space to look at here such as PHP, C#, Ruby, PEARL, SQL and C++. Try googling them and do some research; you might find the perfect language for you!

## PRACTISE CODING ON YOUR TABLET

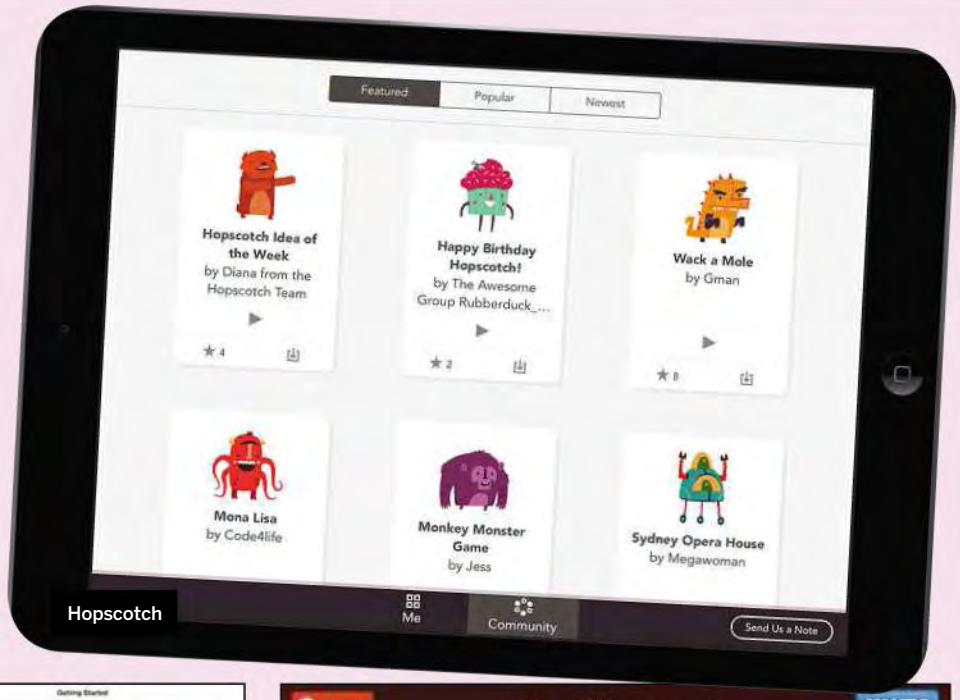
If you have an iPad or Android tablet, there are some great free apps to practise coding techniques using your device:

**HOPSCOTCH** (iOS) is a free app that looks and behaves very similar to Scratch, where you have cartoon characters and drag your code in blocks onto the page. It's very simplistic, but really great fun for younger coders.
● **www.gethopscotch.com**

**HAKITZU ELITE: ROBOT HACKERS** (iOS and Android) is a robot fighting game that uses JavaScript syntax to move the robots and make them fight. It's a free app, but there are IAPs (in-app purchases) to disable ads and buy new parts for your robots.
● **www.kuatostudios.com**

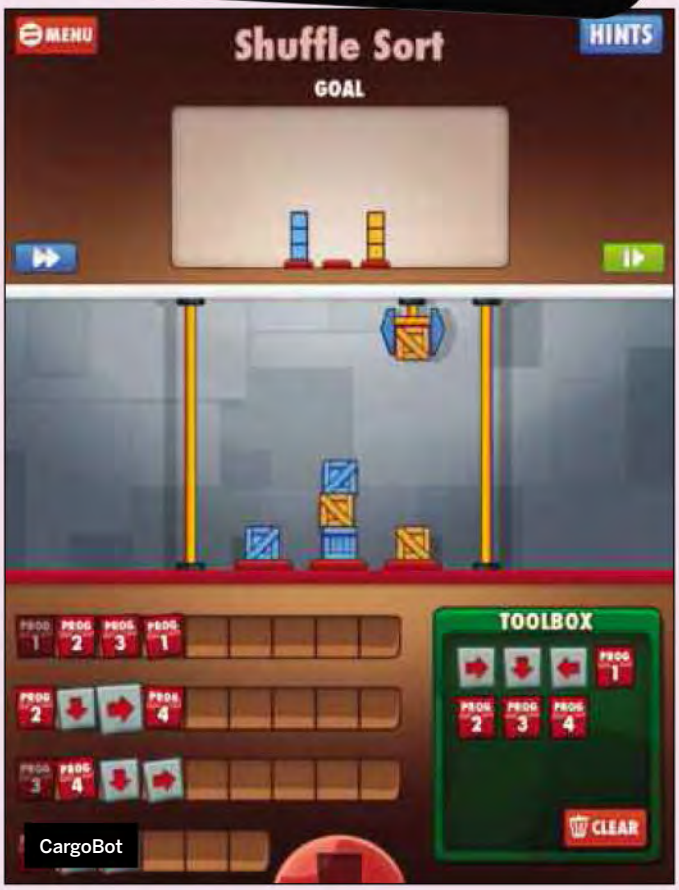
**CODE ACADEMY: CODE HOUR** (iOS) is a free app that guides you through the basics of programming techniques such as data types, variables and If statements. Don't worry Android users, it will be coming to your platform soon.
● **www.codecademy.com**

**CARGO-BOT** (iOS) is a free puzzle game that really forces you to think logically and rewards the user for efficient looping with minimal code. The game was actually developed on the iPad with another app called Codea, which is a programming environment that uses another language, Lua.
● **twolivesleft.com/CargoBot/**

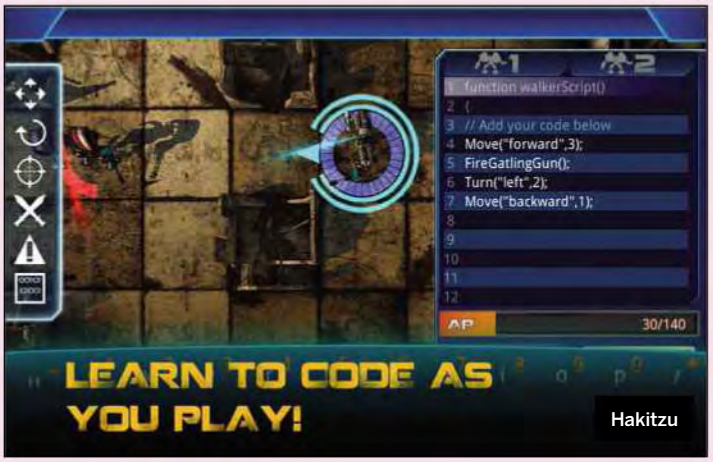


Hopscotch

Code Academy



CargoBot



Hakitzu

# Glossary

## All those essential coding terms defined in plain English

**Application:** A complete program or a group of linked programs designed to perform a certain task or a set of tasks. While they're very different in what they do and how they're used, Adobe Photoshop, Google Chrome and Minecraft are all applications. Applications for smartphones or tablets are often described as apps.

**Argument:** A value or a reference to a value that's passed to a function, so that the function can do some work with it. If "sum" was the function in the instruction answer = sum (value1, value2), then value1 and value2 would be the arguments.

**Array:** A collection of values, strings or variables that can be accessed through an index number. When the program needs to access the information stored in the array, it just needs to call on the item by its index number.

**BASIC:** Beginner's All-purpose Symbolic Instruction Code. A high-level programming language designed with new programmers in mind, BASIC emphasises ease of use over performance, logical structure or sophistication.

**Boolean:** A value that has only two possible states: true or false.

**C:** A general-purpose, high-level programming language used widely in every area of software development. C is the basis for a whole family of popular languages, including C++, Objective-C and C#.

**Class:** The initial version of an object to be used in a program, which can then be used to create further instances. For example, you might define one circle as a class, then use that class to draw further circles.

**Code:** The lines of text or numbers that tell the computer what to do in a program. Code has to be written to conform to the specific style – or protocol – of a programming language.

**Compiler:** A program that takes the code written in a programming language and turns it into an application that other users can run and use.

**Conditional:** An instruction or statement in a program that's only run when a certain condition is met. For example, if the traffic light is green when your car reaches the lights, the car can go.

**Control flow:** The order in which instructions, function calls and statements are checked or executed as a program runs. Programmers use loops, subroutines and conditionals to affect how the control flow works.

**Costume:** In Scratch, a graphic that defines the visual appearance of a sprite. A sprite can have many costumes, and switch between them to express different states or simulate animation.

**CPU:** Central Processing Unit. The main processor in a computer, which runs the lion's share of the code in any application.

**Debug:** The process of checking through code, looking for a mistake that might stop a program running, or prevent it from running properly.

**Function:** A self-contained bit of code that performs a specific task, usually taking in some data, working with it and sending back a result.

**HTML:** HyperText Markup Language. The standard markup language used to create web pages. HTML code is processed by the browser, which then draws out and operates the page.

**Instance:** A single version or realisation of an object. The basic form of the object is defined by its class, but the instance might vary from this in any number of ways.

**Instruction:** An order given to the CPU by a computer program.

**Integer:** A whole number.

**Interpretor:** A program that takes the code written in a programming language and runs it line by line as an application without compiling it first. A program run in an interpreter won't be as fast as a compiled version, but has the advantage that it can be debugged or changed and run again without recompiling.

**Java:** An object-oriented, high-level programming language designed to run programs across as many computers as possible. Java is a hugely popular programming language, and used for many applications that run on the web.

**JavaScript:** A scripting programming language that shares some things in common with Java and C, and which is most commonly used in web pages and web-based applications.

**Language:** A language specifically designed to communicate instructions to a computer. Those instructions come in the shape of a program, written according to the syntax of that specific language.

**Loop:** A structure in a program that tells the processor to keep repeating one or more instructions, either forever, a specified number of times, or until certain conditions are met.

**Object-oriented:** A type of programming that focuses on objects (like a circle, a sprite or a menu), and on the data and behaviour attached to those objects. Object-oriented programs are theoretically more efficient, and easier to understand, maintain and adapt.

**Operator:** An object that manipulates values or variables. For example, a + or – symbol would be the operator in a sum.

**Optimisation:** The process of making a program work more efficiently and often at a higher speed.

**Program:** A series of instructions designed to perform a task on a computer, and written in a programming language. Programs have to be compiled or interpreted to be run.

**Python:** A general-purpose, high-level programming language that's designed to be highly efficient and easy to read. Python is a very popular language, and reasonably approachable to beginners.

**Random:** Something that's made or that happens without any pattern and can't be predicted.

**Routine:** A sequence of instructions that performs a specific task as part of a larger program.

**Ruby:** A general-purpose, high-level, object-oriented programming language. Ruby is designed to be efficient, easy to use and fun, based more on the way programmers think than the way computers operate.

**Scratch:** A simplified programming environment aimed at new programmers and especially young programmers. Scratch teaches the basics of programming without the user having to learn any actual code.

**Script:** A kind of program that tells an application, a web browser or an operating system what to do, line by line. Some programming languages specialise in scripting, and are designed to be easier to understand and use than other languages.

**SDK:** Software Development Kit. A group of programs that enable a programmer to develop applications for a specific operating system or device, such as a tablet, games console, computer or smartphone.

**Source code:** The code for a program before it's compiled or interpreted.

**Sprite:** In Scratch, an object that appears on the Stage and performs actions according to the blocks of script attached to it.

**Stage:** In Scratch, the area of the screen in which sprites move and operate according to their scripts.

**Statement:** The smallest standalone element of a program. Statements describe an action to be carried out.

**String:** A sequence of letters, words or numbers, stored by and used in a program. A string might be anything from a series of numbers to a word, a sentence or a larger chunk of text.

**Subroutine:** A routine within a routine. Subroutines are often used to handle tasks that might be needed again and again by a program.

**Syntax:** The structure of a programming language, and the rules that govern how the different instructions need to be written and laid out.

**Toolbar:** A horizontal or vertical bar containing icons that launch different tools in an application.

**Value:** A number, letter or symbol stored and used in a computer program. Values can either be constant, where they stay the same no matter what happens in the program, or variables.

**Variable:** A value that changes as a program runs its course. Variables aren't so much the value itself as the location that stores the value. The program can refer to the location by pointing to the variable, then use or change whatever value is held in it.

# Teach Your Kids to Code


## EDITORIAL

**Editor:** Stuart Andrews
**Managing Editor:** Priti Patel
**Art Editor:** Billbagnalldesign.com
**Production:** Rachel Storry
**Contributors:** Andrew Dixon

## ADVERTISING & MARKETING

**MagBook Advertising Manager:** Simone Daws +44 20 7907 6617
**Production Manager:** Nicky Baker +44 20 7907 6056
**MagBook Manager:** Dharmesh Mistry +44 20 7907 6100

## MANAGEMENT

**Managing Director:** John Garewal
**Deputy Managing Director:** Tim Danton
**MD of Advertising:** Julian Lloyd-Evans
**Newstrade Director:** David Barker
**MD of Enterprise:** Martin Belson
**Group Managing Director:** Ian Westwood
**Chief Operating Officer:** Brett Reynolds
**Group Finance Director:** Ian Leggett
**Chief Executive:** James Tye
**Chairman:** Felix Dennis

MAGBOOK

The 'MagBook' brand is a trademark of Dennis Publishing Ltd, 30 Cleveland Street, London W1T 4JD. Company registered in England.

All material © Dennis Publishing Ltd, licensed by Felden 2014, and may not be reproduced in whole or part without the consent of the publishers.

**Coding for Kids: Scratch**
ISBN: 1-78106-450-4

**LICENSING & SYNDICATION**
To license this product please contact Carlotta Serantoni on +44 20 7907 6550 or email carlotta_serantoni@dennis.co.uk
To syndicate content from this product please contact Anj Dosaj-Halai on +44 20 7907 6132 or email anj_dosaj-halai@dennis.co.uk


**LIABILITY**
While every care was taken during the production of this MagBook, the publishers cannot be held responsible for the accuracy of the information or any consequence arising from it. Dennis Publishing takes no responsibility for the companies advertising in this MagBook.


The paper used within this MagBook is produced from sustainable fibre, manufactured by mills with a valid chain of custody.

Dennis

Printed at Wyndeham Southernprint

# Resources

There are hundreds of websites where you can find tutorials, advice and support while you learn more about coding. Here are our favourites

### www.w3schools.com
This site provides tutorials and reference for many of the big programming languages and tools used on the web. A good resource for information and education on HTML and JavaScript.

### smallbasic.com/
Microsoft's SmallBASIC website isn't just the place where you can download the language, but a useful source of sample projects, tutorials and eBooks. Here, you can also find the SmallBASIC blog, which is full of useful hints and tips on the language.

### www.khanacademy.org
Khan Academy teaches just about everything, but it also runs free courses in programming and computer science, with a JavaScript course that can take you all the way from simple drawing and animations to advanced natural simulations.

### www.codeacademy.com
Code Academy is the best known of the specialist code schools, running courses in Ruby, Python, JavaScript and HTML. CodeAcademy was partly responsible for the computer science curriculum being used in many UK schools, and offers simple projects where you can learn a few coding skills in under half an hour.

### www.kidsruby.com
KidsRuby is a simplified coding environment aimed at aspiring young programmers. It all works with real Ruby code, and you just write your code in one window and run it in another, so you can see exactly what any code you add or change does immediately. Colour-coded text makes it easy to use, and the team has started creating lessons and examples to help kids on their way.

### www.kodugamelab.com
Part game, part visual programming environment, Kodu does a great job of introducing programming concepts while helping kids create their own simple games. It might not teach you how to code, but it does help teach computational thinking, and it's easy to use and fun. Versions for Windows 8, the Xbox 360 and earlier versions of Windows are available.

### python4kids.wordpress.com/
A blog full of Python tutorials written by a programmer for his son. The tutorials work in Python 2.7 rather than the latest version, but they're easy to follow and provide a good background.

### www.pygame.org
Pygame is a set of Python modules designed specifically for writing games, and one that makes the job of writing games in Python a whole lot easier. It's not designed specifically for kids, and you'll need python tuition elsewhere to make much out of it; a rough working knowledge is essential before you even start coding games. ●



# Fun projects that will see you get to grips with programming fast

## Start coding

Take a tour around Scratch, say "Hello World" with a magic cat, and build your first Scratch game

## BASIC basics

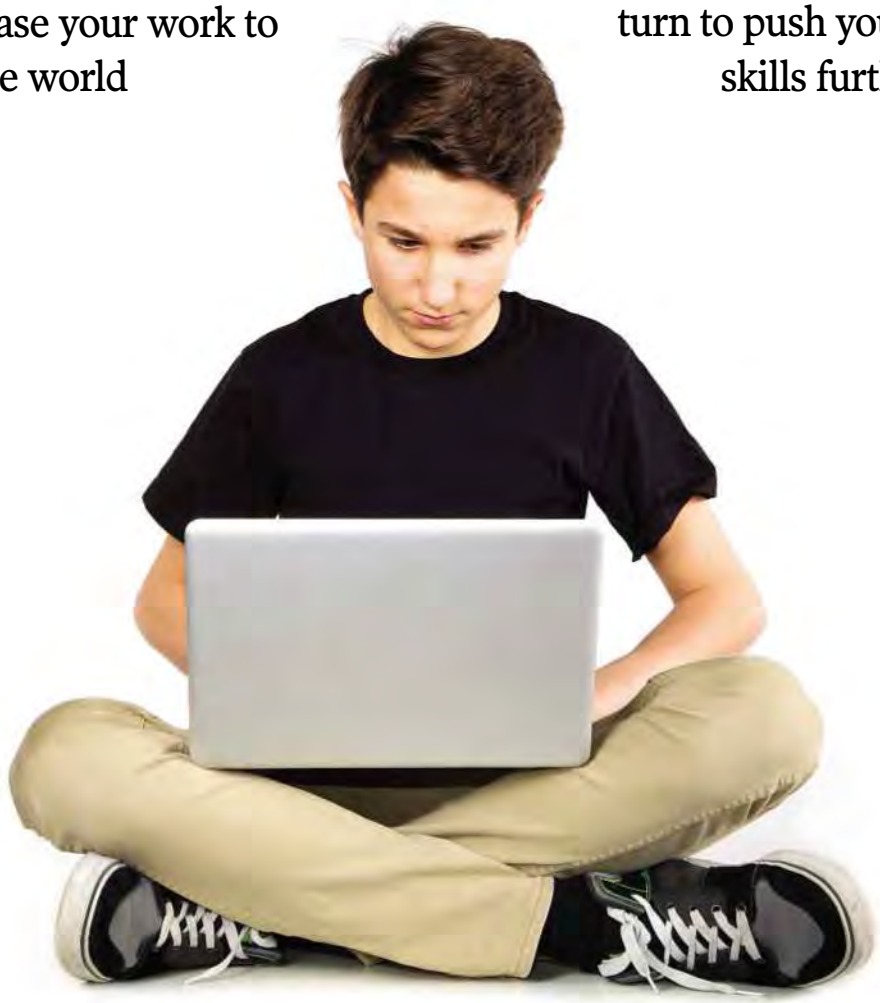
Get to grips with SmallBASIC, master its graphics functions, and create your own quiz

## Build your skills

Learn how to paint with Scratch, use webcam graphics and motion controls, and showcase your work to the world

## The next level

We introduce Visual Basic, build a slideshow app, and show where to turn to push your coding skills further



MAGBOOK
CODING FOR KIDS: SCRATCH
£9.99
ISBN:1-78106-450-4
9 781781 064504