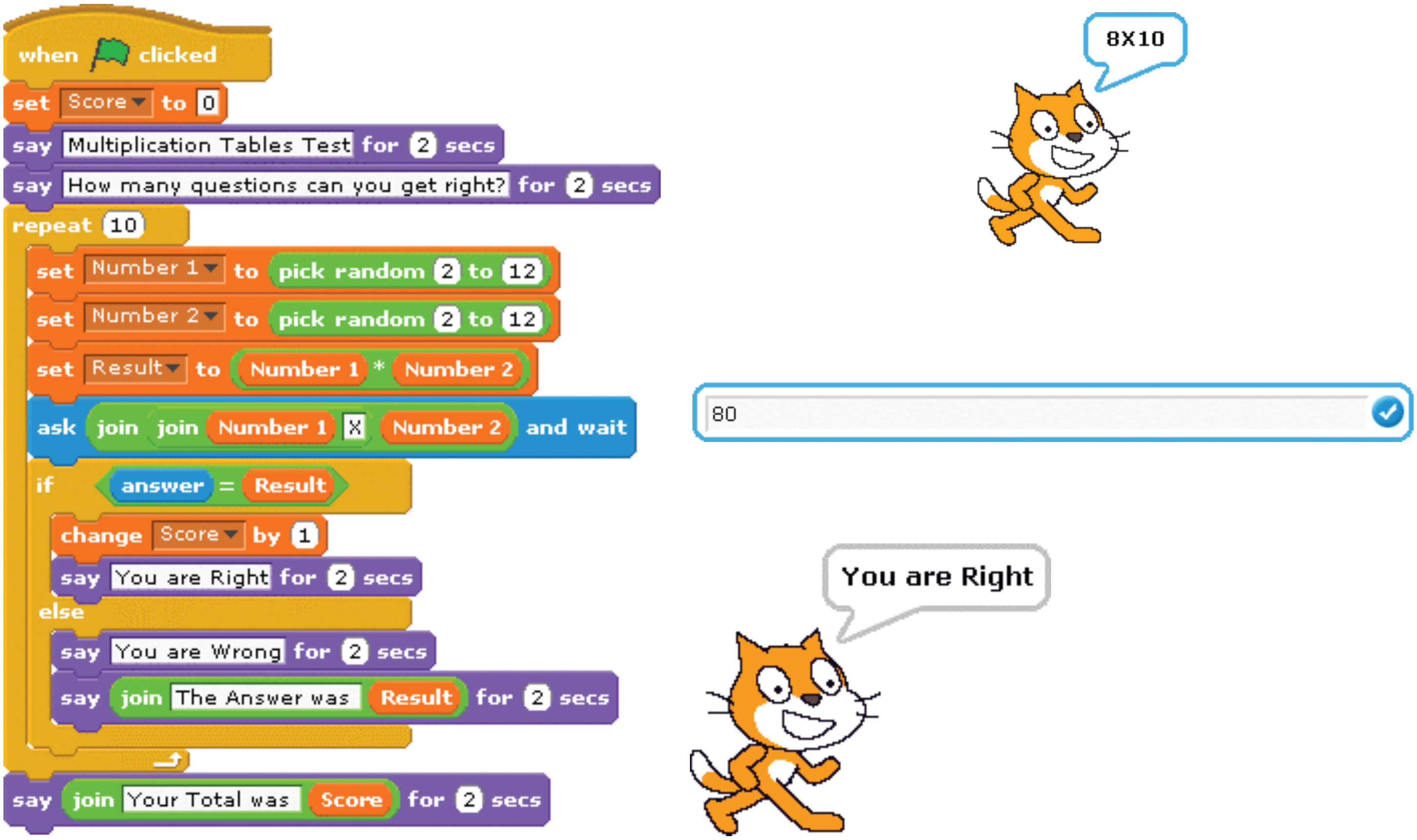
4.2 Types of programming language, translators and integrated development environments (IDEs)

People use many different languages to communicate with each other. In order for two people to understand each other they need to speak the same language or another person, an interpreter, is needed to translate from one language to the other language.

Programmers use many different programming languages to communicate with computers. Computers only 'understand' their own language, called **machine code**. A program needs to be translated into machine code before it can be 'understood' by a computer.

Programs are our way of telling a computer what to do, how to do it and when to do it. This enables a single computer to perform many different types of task. A computer can be used to stream videos, write reports, provide weather forecasts and many, many other jobs.

Here is an example of a simple task that can be performed by a computer:



```
when [flag] clicked
set Score to 0
say Multiplication Tables Test for 2 secs
say How many questions can you get right? for 2 secs
repeat 10
    set Number 1 to pick random 2 to 12
    set Number 2 to pick random 2 to 12
    set Result to Number 1 * Number 2
    ask join join Number 1 X Number 2 and wait
    if answer = Result
        change Score by 1
        say You are Right for 2 secs
    else
        say You are Wrong for 2 secs
        say join The Answer was Result for 2 secs
say join Your Total was Score for 2 secs
```

8X10

80

You are Right

Figure 4.14 A Scratch multiplication table test program

Find out more

Find at least ten different tasks that computer programs perform in your school.

A **computer program** is a list of instructions that enable a computer to perform a specific task. Computer programs can be written in **high-level languages** and **low-level languages** depending on the task to be performed and the computer to be used. Most programmers write programs in high-level languages.

4.2.1 High-level languages and low-level languages

High-level languages

High-level languages enable a programmer to focus on the problem to be solved and require no knowledge of the hardware and instruction set of the computer that will use the program. Many high-level programming languages are portable and can be used on different types of computer.

Link

For more on instruction sets, see Section 3.1.4.

High-level languages are designed with programmers in mind; programming statements are easier to understand than those written in a low-level language. This means that programs written in a high-level language are easier to:

- read and understand as the language used is closer to English
- write in a shorter time
- debug at the development stage
- maintain once in use.

The following snippet of program to add two numbers together is a single program statement written in a typical high-level language. It shows how easy it is to understand what is happening in a high-level language program:

```
Sum := FirstNumber + SecondNumber
```

There are many different high-level programming languages in use today including C++, Delphi, Java, Pascal, Python, Visual Basic and many more. Once a programmer has learned the techniques of programming in any high-level language, these can be transferred to writing programs in other high-level languages.

Find out more

High-level programming languages are said to be 'problem oriented'. What type of problems are the languages named above used for? Find out about five more high-level languages. Name each programming language and find out what it is used for.

Low-level languages

Low-level languages relate to the specific architecture and hardware of a particular type of computer. Low-level languages can refer to **machine code**, the binary instructions that a computer understands, or **assembly language** that needs to be translated into machine code.

Machine code

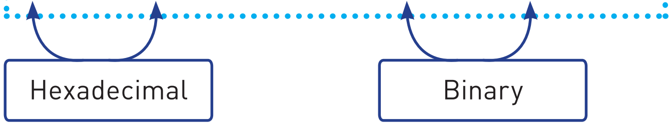
Programmers do not usually write in machine code as it is difficult to understand, and it can be complicated to manage data manipulation and storage.

Link

For more on hexadecimal see Section 1.1.2.

The following snippet of program to add two numbers together is written in typical machine code, shown in both hexadecimal and binary, and consists of three statements:

```
1        12                 0001         00010010
4        13                 0100         00010011
0        1A                 0000         00011010
```



As you can see, this is not easy to understand in binary! Machine code is usually shown in hexadecimal.

Find out more

Find out about two different types of machine code. Name each chip set the machine code is used for and find the codes for load, add and store.

▼ **Table 4.2** Differences between high-level and low-level languages

Language	Advantages	Disadvantages
High-level	independent of the type of computer being used easier to read, write and understand programs quicker to write programs programs are easier and quicker to debug easier to maintain programs in use	programs can be larger programs can take longer to execute programs may not be able make use of special hardware
Low-level	can make use of special hardware includes special machine-dependent instructions can write code that doesn't take up much space in primary memory can write code that performs a task very quickly	it takes a longer time to write and debug programs programs are more difficult to understand

4.2.2 Assembly languages

Fewer programmers write programs in an assembly language. Those programmers who do, do so for the following reasons:

- to make use of special hardware
- to make use of special machine-dependent instructions
- to write code that doesn't take up much space in primary memory
- to write code that performs a task very quickly.

The following snippet of program to add two numbers together is written in a typical assembly language and consists of three statements:

```
LDA          First
ADD          Second
STO          Sum
```

In order to understand this program, the programmer needs to know that:

- `LDA` means load value of the variable (in this case, **First**) into the accumulator
- `ADD` means add value of variable (in this case, **Second**) to the value stored in the accumulator
- `STO` replace the value of the variable (in this case, **Sum**) by the value stored in the accumulator

Assembly language needs to be translated into machine code using an assembler in order to run. See the next section for more details.

Find out more

Find out about two assembly languages. Name each assembly language and find out what type of computer it is used for.

4.2.3 Translators

Computer programs can exist in several forms.

Programs are written by humans in a form that people who are trained as computer programmers can understand. In order to be used by a computer, programs need to be translated into the binary instructions, machine code, that the computer understands. Humans find it very difficult to read binary, but computers can only perform operations written in binary.

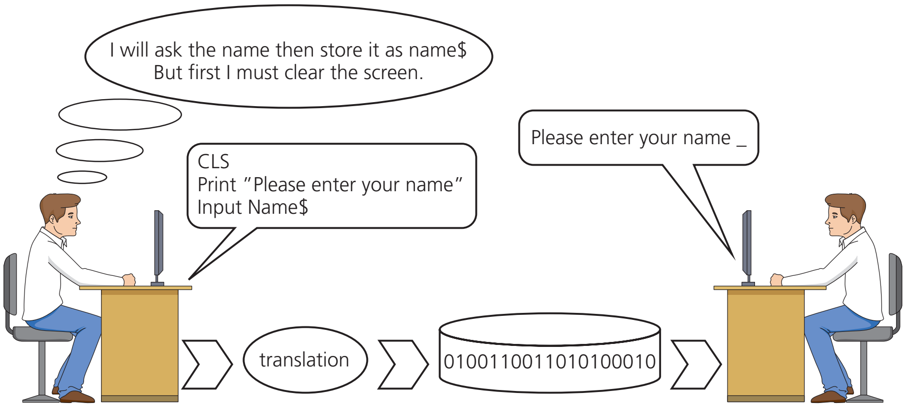

I will ask the name then store it as name$
But first I must clear the screen.
CLS
Print "Please enter your name"
Input Name$
Please enter your name _
translation
0100110011010100010

Figure 4.15 Translation

A program must be translated into binary before a computer can use it; this is done by a utility program called a **translator**. There are several types of translator program in use; each one performs a different task.

Compilers

A **compiler** is a computer program that translates an entire program written in a high-level language (HLL) into machine code all in one go so that it can be directly used by a computer to perform a required task. Once a program is compiled the machine code can be used again and again to perform the same task without re-compilation. If errors are detected, then an error report is produced instead of a compiled program.

The high-level program statement:

```
Sum := FirstNumber + SecondNumber
```

becomes the following machine code instructions when translated:

```
0001        00010010
0100        00010011
0000        00011010
```

Interpreters

An **interpreter** is a computer program that reads a statement from a program written in a high-level language, translates it, performs the action specified and then does the same with the next statement and so on. If there is an error in the statement then execution ceases and an error message is output, sometimes with a suggested correction.

A program needs to be interpreted again each time it is run.

Assemblers

An **assembler** is a computer program that translates a program written in an assembly language into machine code so that it can be directly used by a computer to perform a required task. Once a program is assembled the machine code can be used again and again to perform the same task without re-assembly.

The assembly language program statements:

```
LDA         First
ADD         Second
STO         Sum
```

become the following machine code instructions when translated:

```
0001        00010010
0100        00010011
0000        00011010
```

▼ **Table 4.3** Translation programs summary

Compiler	Interpreter	Assembler
Translates a high-level language program into machine code.	Executes a high-level language program one statement at a time.	Translates a low level assembly language program into machine code.
An executable file of machine code is produced.	No executable file of machine code is produced.	An executable file of machine code is produced.
One high-level language statement can be translated into several machine code instructions.	One high-level language program statement may require several machine code instructions to be executed.	One low-level language statement is usually translated into one machine code instruction.
Compiled programs are run without the compiler.	Interpreted programs cannot be run without the interpreter.	Assembled programs are used without the assembler.
A compiled program is usually distributed for general use.	An interpreter is often used when a program is being developed.	An assembled program is usually distributed for general use.

4.2.4 Advantages and disadvantages of compilers and interpreters

The advantages and disadvantages of compilers and interpreters are compared in Table 4.4.

▼ **Table 4.4** Comparing translators

Translators	Advantages	Disadvantages
Interpreter	easier and quicker to debug and test programs during development easier to edit programs during development	programs cannot be run without the interpreter programs can take longer to execute
Compiler	a compiled program can be stored ready for use a compiled program can be executed without the compiler a compiled program takes up less space in memory when it is executed a compiled program is executed in a shorter time	it takes a longer time to write, test and debug programs during development

4.2.5 Integrated Development Environment (IDE)

An Integrated Development Environment (IDE) is used by programmers to aid the writing and development of programs. There are many different IDEs available; some just support one programming language, others can be used for several different programming languages. You may be using PyCharm (for Python), Visual Studio (for Visual Basic) or BlueJ (for Java) as your IDE.

Find out more

Find out which programming language and IDE you are using in school and see if there are any other IDEs available for your programming language.

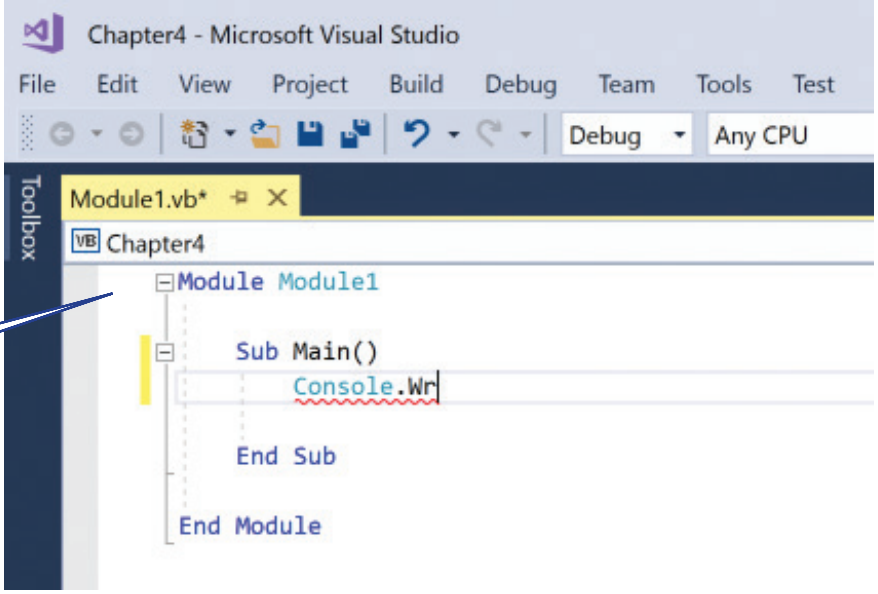
IDEs usually have these features:

- code editors
- a translator
- a runtime environment with a debugger
- error diagnostics
- auto-completion
- auto-correction
- an auto-documenter and prettyprinting.

Let's look at each of these features in turn and see how they help the development process.

Code editor

A code editor allows a program to be written and edited without the need to use a separate text editor. This speeds up the program development process, as editing can be done without changing to a different piece of software each time the program needs correcting or adding to.



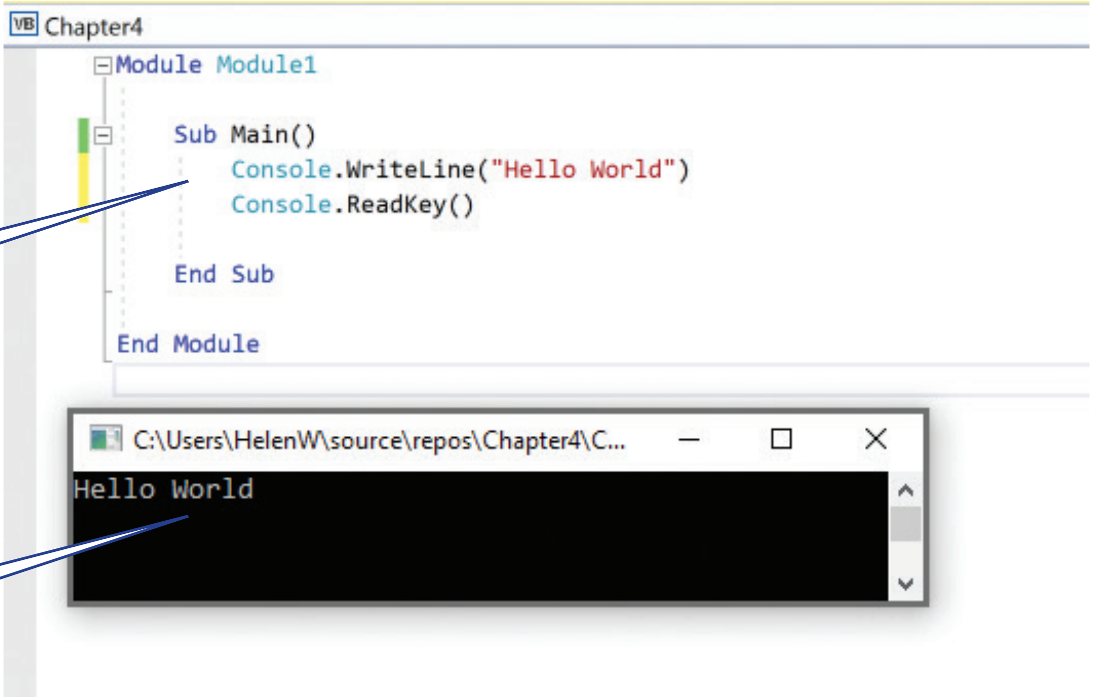
```
Module Module1

    Sub Main()
        Console.Wr

    End Sub

End Module
```

Code editing window

▲ **Figure 4.16** Visual Studio code editor

Translator

Most IDEs usually provide a translator, this can be a compiler and/or an interpreter, to enable the program to be executed. The interpreter is often used for developing the program and the compiler to produce the final version of the program to be used.



```
Module Module1

    Sub Main()
        Console.WriteLine("Hello World")
        Console.ReadKey()

    End Sub

End Module
```

```
Hello World
```

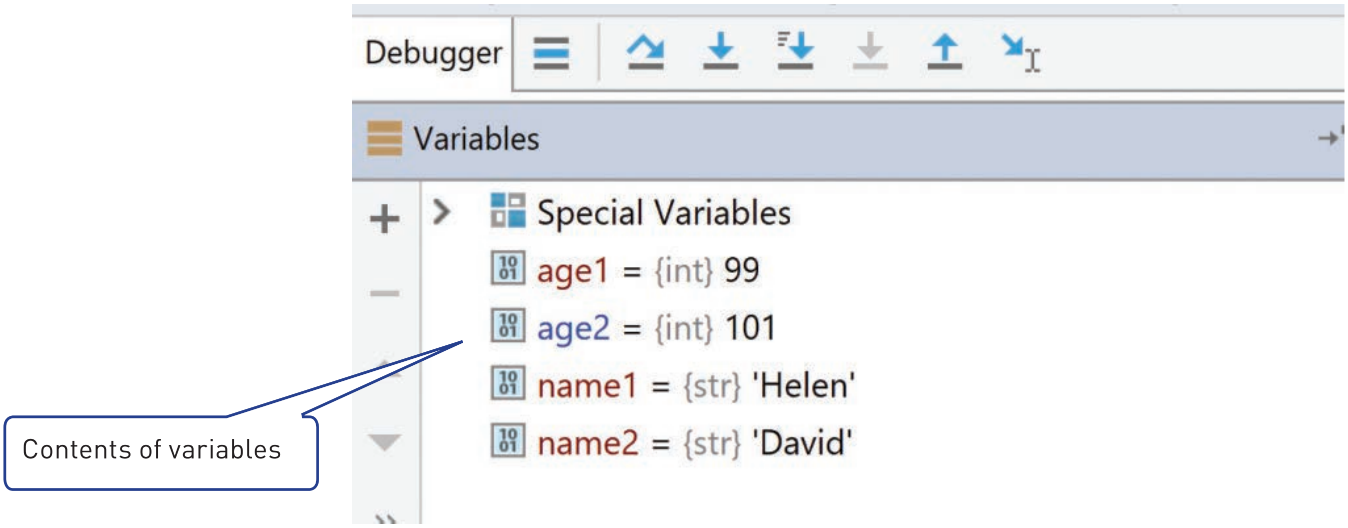
Code editing window

Code executing window

▲ **Figure 4.17** Visual Studio code editor and program running

A runtime environment with a debugger

A debugger is a program that runs the program under development and allows the programmer to step through the program a line at a time (single stepping) or to set a breakpoint to stop the execution of the program at a certain point in the source code. A report window then shows the contents of the variables and expressions evaluated at that point in the program. This allows the programmer to see if there are any logic errors in the program and check that the program works as intended.



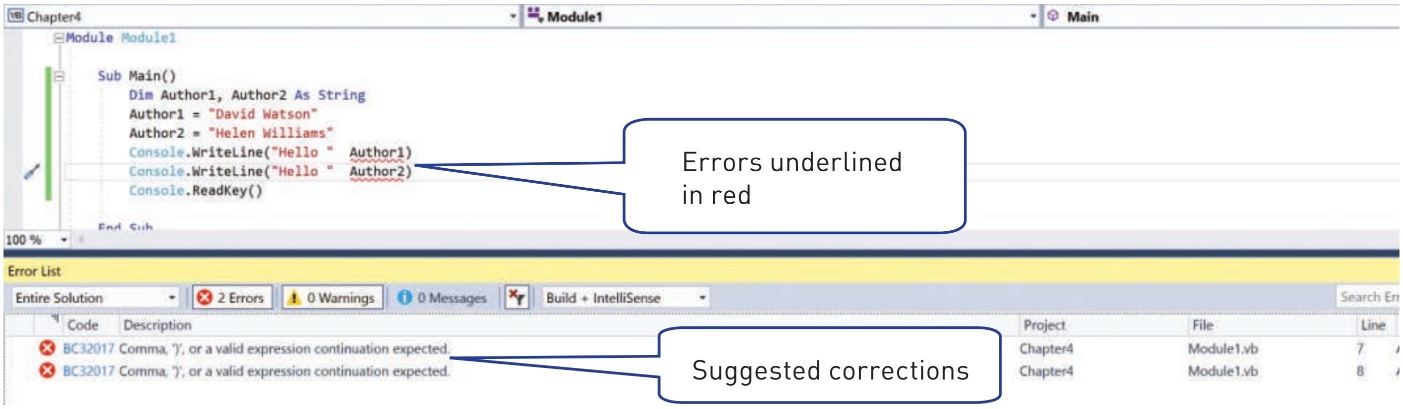
```
> Special Variables
age1 = {int} 99
age2 = {int} 101
name1 = {str} 'Helen'
name2 = {str} 'David'
```

Contents of variables

▲ **Figure 4.18** PyCharm debugger

Error diagnostics and auto-correction

Dynamic error checking finds possible errors as the program code is being typed, alerts the programmer at the time and provides a suggested correction. Many errors can therefore be found and corrected during program writing and editing before the program is run.



```
Module Module1

    Sub Main()
        Dim Author1, Author2 As String
        Author1 = "David Watson"
        Author2 = "Helen Williams"
        Console.WriteLine("Hello "  Author1)
        Console.WriteLine("Hello "  Author2)
        Console.ReadKey()

    End Sub
```

Code	Description	Project	File	Line
BC32017	Comma, ')', or a valid expression continuation expected.	Chapter4	Module1.vb	7
BC32017	Comma, ')', or a valid expression continuation expected.	Chapter4	Module1.vb	8

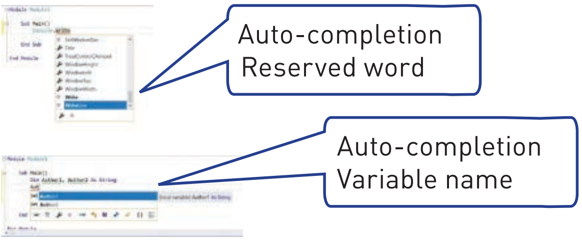
Errors underlined in red

Suggested corrections

▲ **Figure 4.19** Visual Studio error list with suggested corrections

Auto-completion

Code editors can offer context-sensitive prompts with text completion for variable names and reserved words.



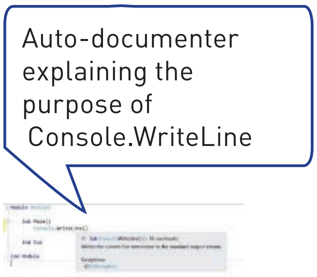
Auto-completion
Reserved word

Auto-completion
Variable name

▲ **Figure 4.20** Visual Studio showing auto-completion

Auto-documenter and prettyprinting

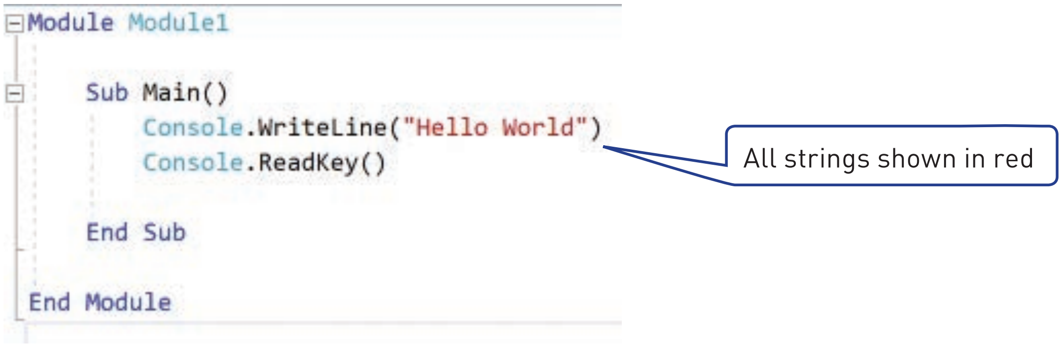
IDEs can provide an auto-documenter to explain the function and purpose of programming code.



Auto-documenter explaining the purpose of Console.WriteLine

▲ **Figure 4.21** Visual Studio auto-documenter

Most code editors colour code the words in the program and lay out the program in a meaningful way – this is called **prettyprinting**.



```
Module Module1

    Sub Main()
        Console.WriteLine("Hello World")
        Console.ReadKey()

    End Sub

End Module
```

All strings shown in red

▲ **Figure 4.22** Visual Studio code editor showing prettyprinting

Find out more

Find out which of these features are available in the IDE you are using in school.

Activity 4.2

1 Tick (✓) the appropriate column in the following table to indicate whether the statement about the translator is True or False.

	True	False
An assembler translates a high-level language program.		
It is more difficult to write a program in a low-level language.		
Java is an assembly language.		
It is quicker to develop a program using a high-level language.		
You always need a compiler to run a compiled program.		
A program that is interpreted takes a longer time to run than a compiled program.		
Low-level languages are machine dependent.		

2 a Suki is writing a program in a high-level language. Describe three features of an IDE that she would find helpful.

b Describe the difference between a compiler and an interpreter.

c Explain why a programmer would choose to write a program in assembly language.

Extension

For those students considering the study of this subject at A Level, the following shows how interrupts are used in the Fetch–(Decode)–Execute cycle.

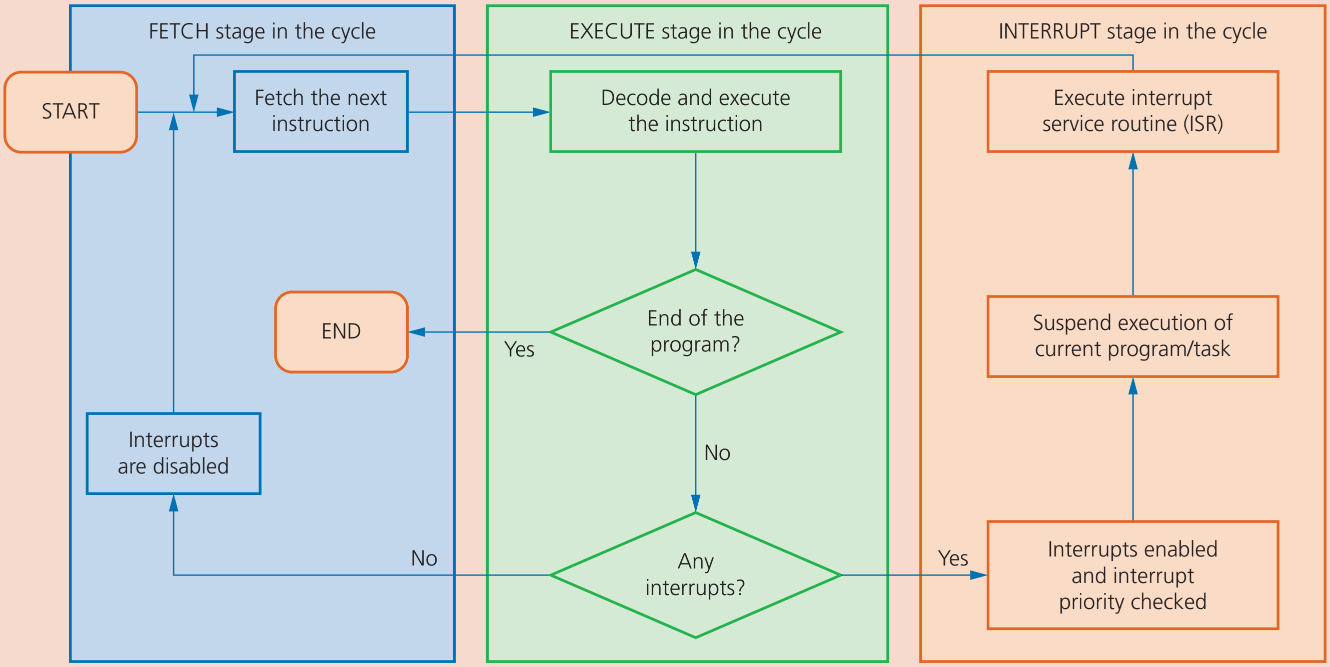
Use of interrupts in the Fetch–Execute cycle

The following figure shows a general overview of how a computer uses interrupts to allow it to operate efficiently and to allow it, for example, to carry out multi-tasking functions. Just before we discuss interrupts in this general fashion, the following notes explain how interrupts are specifically used in the Fetch–Execute cycle.

A special register called the interrupt register is used in the Fetch–Execute cycle. While the CPU is in the middle of carrying out this cycle, an interrupt could occur that will cause one of the bits in the interrupt register to change its status. For example, the initial status might be 0000 0000 and a fault might occur while writing data to the hard drive; this would cause the register to change to 0000 1000. The following sequence now takes place:

- at the next Fetch–Execute cycle, the interrupt register is checked bit by bit
- the contents 0000 1000 would indicate an interrupt occurred during a previous cycle and it still needs servicing; the CPU would now service this interrupt or 'ignore' it for now depending on its priority
- once the interrupt is serviced by the CPU, it stops its current task and stores the contents of its registers
- control is now transferred to the interrupt handler (or interrupt service routine, ISR)
- once the interrupt is fully serviced, the register is reset and the contents of registers are restored.

The following flow diagram summarises the interrupt process during the Fetch–Execute cycle:


FETCH stage in the cycle
EXECUTE stage in the cycle
INTERRUPT stage in the cycle
START
Fetch the next instruction
Decode and execute the instruction
Execute interrupt service routine (ISR)
END
Yes
End of the program?
Suspend execution of current program/task
No
Interrupts are disabled
No
Any interrupts?
Yes
Interrupts enabled and interrupt priority checked

▲ **Figure 4.23** The interrupt process in the Fetch–Execute cycle