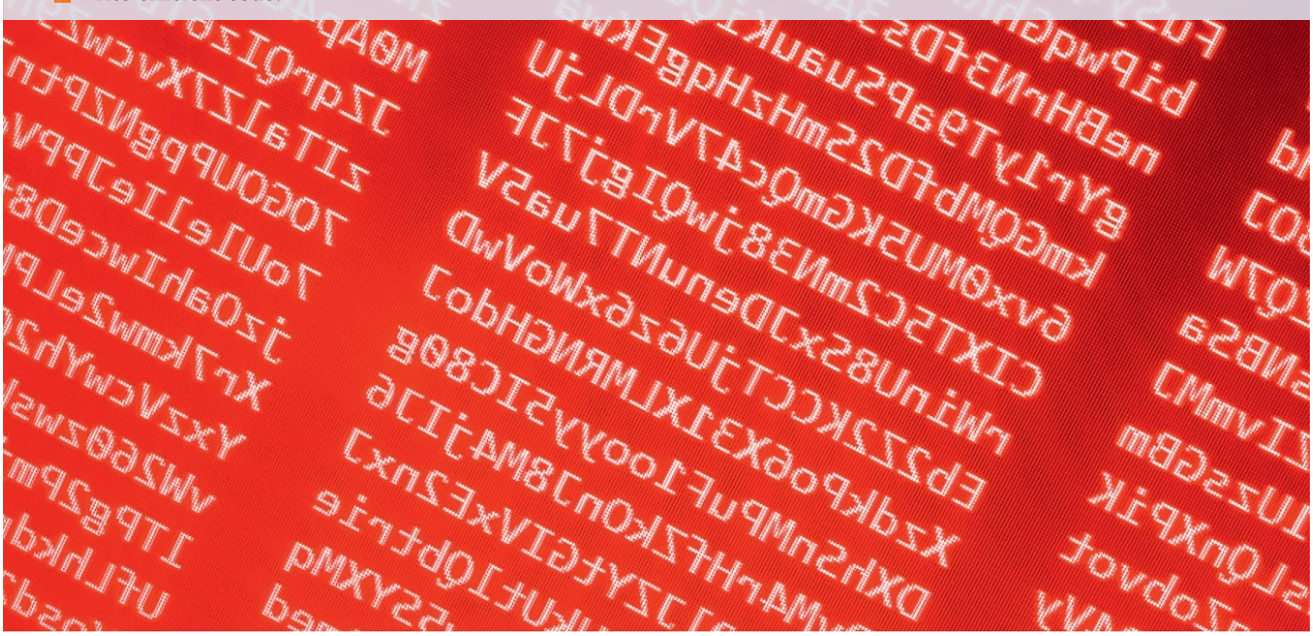
Chapter 14:
Programming and data representation

Learning objectives

By the end of this chapter you should be able to:

- write a program in a high-level language (Python, Visual Basic console mode, Java)
- implement and write pseudocode from a given design presented as either a program flowchart or structured English
- write pseudocode and program statements for:
  - the declaration of variables and constants
  - the assignment of values to variables and constants
  - expressions involving any of the arithmetic or logical operators
  - input from the keyboard and output to the console
- use a subset of the built-in functions and library routines supported by the chosen programming language, including those used for string/character manipulation and random number generator
- use an 'IF' structure including the 'ELSE' clause and nested IF statements
- use a 'CASE' structure
- use a loop ('count controlled', 'post-condition', 'pre-condition')
- justify why one loop structure may be better suited to a problem than the others
- define and use a procedure and explain where in the construction of an algorithm it is appropriate to use a procedure
- use parameters
- show understanding of passing parameters by reference and by value
- define and use a function and explain where in the construction of an algorithm it is appropriate to use a function
- show understanding that a function is used in an expression (the return value replaces the call)
- use the terminology associated with procedures and functions: procedure/function header, procedure/function interface, parameter, argument, return value
- write efficient code.



14.01 Programming languages

Chapters 12 and 13 introduced the concept of solving a problem and representing a solution using a flowchart or pseudocode. We expressed our solutions using the basic constructs: assignment, sequence, selection, iteration, input and output.

To write a computer program, we need to know the syntax (the correct structure of statements) of these basic constructs in our chosen programming language. This chapter introduces syntax for Python, Visual Basic console mode and Java.

Note that for convenience and easy reference, definitive pseudocode syntax is repeated in this chapter at the appropriate points.

You only need learn to program in one of the three languages covered in this book. Programming language is only examined at A Level but it is important to start learning it from AS Level to ensure you are well-prepared. For the AS Level exams you should use pseudocode rather than programming language.

TIP

The only way of knowing whether the algorithm you have designed is a suitable solution to the problem you are trying to solve, is to implement your pseudocode in your chosen programming language and test the program by running it.

Python

Python was conceived by Guido van Rossum in the late 1980s. Python 2.0 was released in 2000 and Python 3.0 in 2008. Python is a multi-paradigm programming language, meaning that it fully supports both object-oriented programming and structured programming. Many other paradigms, including logic programming, are supported using extensions. These paradigms are covered in Chapters 26, 27 and 29.

The Python programs in this book have been prepared using Python 3 (see Python for a free download) and Python's Integrated DeveLopment Environment (IDLE).

Key characteristics of Python include the following.

- Every statement must be on a separate line.
- Indentation is significant. This is known as the 'off-side rule'.
- Keywords are written in lower case.
- Python is case sensitive: the identifier `Number1` is seen as different from `number1` or `NUMBER1`.
- Everything in Python is an object (see Chapter 27).
- Code makes extensive use of a concept called 'slicing' (see Section 14.07).
- Programs are interpreted (see Chapter 8, Section 8.05 for information on interpreted and compiled programs).

You can type a statement into the Python Shell and the Python interpreter will run it immediately (see Figure 14.01).

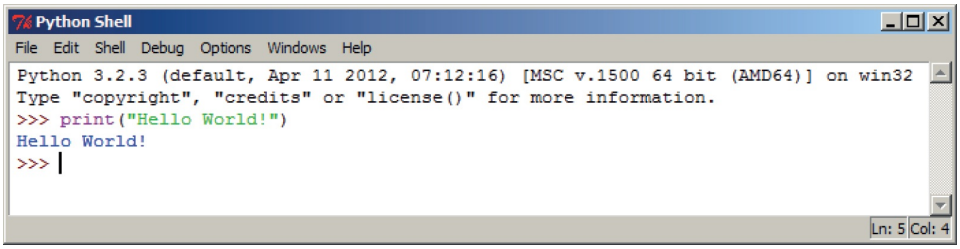
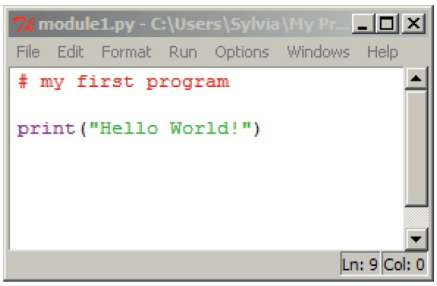


Figure 14.01 Running a statement in the Python shell

You can also type program code into a Python editor (such as IDLE), save it with a .py extension and then run the program code from the Run menu in the editor window (see Figure 14.02).

a



b

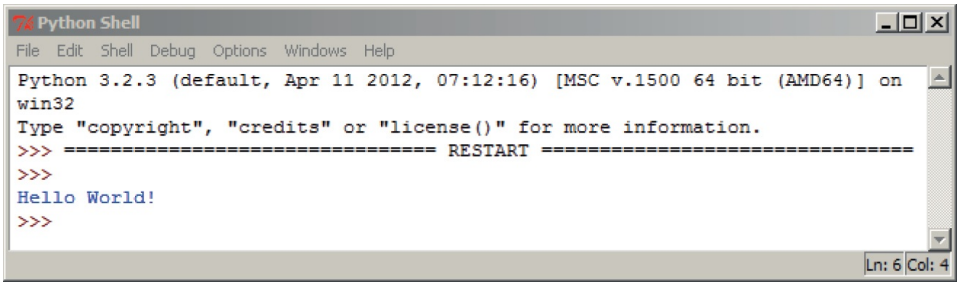


Figure 14.02 (a) A saved program in the Python editor window and (b) running in the Python shell

Visual Basic Console Mode (VB.NET)

VB.NET is a multi-paradigm, high-level programming language, implemented on the .NET Framework. Microsoft launched VB.NET in 2002 as the successor to its original Visual Basic language. Microsoft's integrated development environment (IDE) for developing in VB.NET is Visual Studio. Visual Studio Express and Visual Studio Community are freeware.

The Visual Basic programs in this book have been prepared using Microsoft Visual Basic 2010 Express Console Application. (Free download available from Visual Studio Express)

Key characteristics of VB.NET include the following.

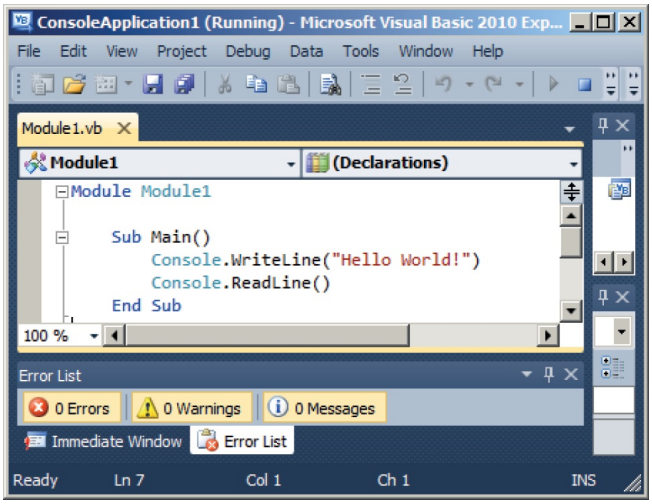
- Every statement should be on a separate line. Statements can be typed on the same line with a colon (:) as a separator. However, this is not recommended.
- Indentation is good practice.
- VB.NET is not case sensitive. Modern VB.NET editors will automatically copy the case from the first definition of an identifier.
- The convention is to use CamelCaps (also known as PascalCaps) for identifiers and keywords.
- Programs need to be compiled (see Chapter 8, Section 8.05 for information on interpreted and compiled programs).

You type your program code into the Integrated Development Environment (IDE) as shown in Figure 14.03 (a), save the program code and then click on the Run button . This starts the compiler. If there are no syntax errors the compiled program will then run. Output will be shown in a separate console window (see Figure 14.03 (b)).

Note that the console window shuts when the program has finished execution. To keep the console window open so you can see the output (see Figure 14.03), the last statement of your program should be

```
Console.ReadLine()
```

a



b

Figure 14.03 (a) A saved program in the VB.NET editor and (b) running in the program execution (console) window

Java

Java was originally developed by James Gosling at Sun Microsystems (now owned by Oracle) and released in 1995. The Java Runtime Environment (JRE) is intended for end users, and the Java Development Kit (JDK) is intended for software developers and includes development tools such as the Java compiler and a debugger.

Java was intended to be platform independent. The Java programs in this book have been prepared using NetBeans 8.2. However, any text editor can be used to write Java source code.

Key characteristics of Java include the following.

- Every statement ends with a semicolon (;). More than one statement can go on a single line, but this is not recommended.
- Indentation is good practice.
- Java is case sensitive.
- The convention is to use camelCaps for identifiers, lower case for keywords and capitalised identifiers for classes.
- A compound statement consists of a sequence of statements enclosed between braces { }.
- Whenever Java syntax requires a statement, a compound statement can be used.
- Programs need to be compiled (see Chapter 8, Section 8.05 for information on interpreted and compiled programs) into bytecode and then run using the Java Virtual Machine.

Java was designed almost exclusively as an object-oriented language. All code is written inside classes. Only the simple data types (such as integer, real) are not objects. Strings are objects.

Source files must be named after the public class they contain, appending the suffix .java, for example, Ex1.java. It must first be compiled into bytecode, using a Java compiler, producing a file named Ex1.class. Only then can it be executed.

The method name “main” is not a keyword in the Java language. It is simply the name of the method the Java launcher calls to pass control to the program.

You type your program statements into the Integrated Development Environment (IDE) as shown in Figure 14.04, save the program code and then click on the Run button ( ). This starts the compiler. If there are no syntax errors the compiled program code will then run. Output will be shown in the Output window (see Figure 14.04).

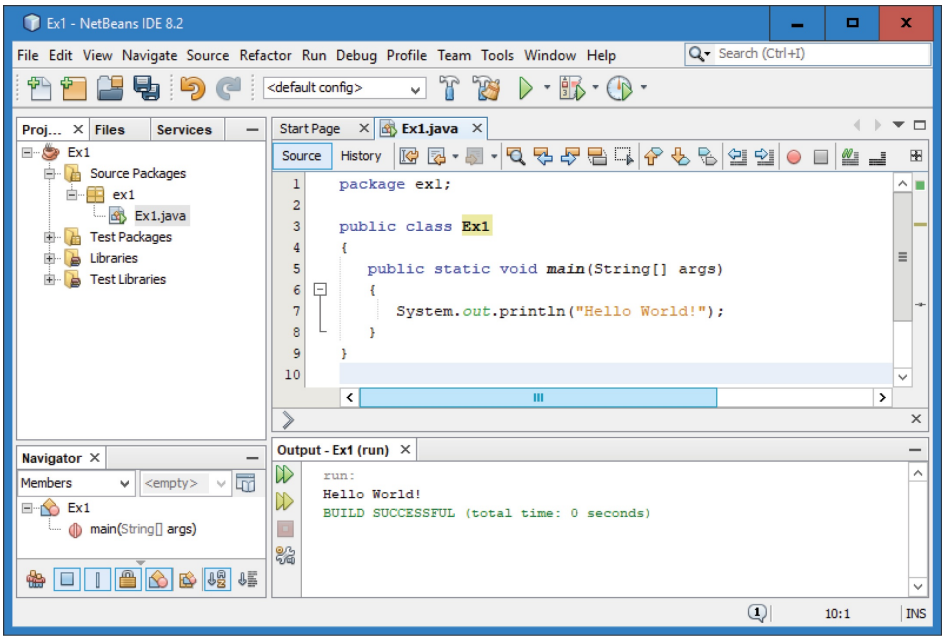


Figure 14.04 A Java program in the NetBeans editor and running in the Output window

14.02 Programming basics

Declaration of variables

Most programming languages require you to declare the type of data to be stored in a variable, so the correct amount of memory space can be reserved by the compiler. A variable declared to store a whole number (integer) cannot then be used to store alphanumeric characters (strings), or the other way around. VB.NET and Java require variables to be declared before they are used.

Python handles variables differently to most programming languages. It tags values. This is why Python does not have variable declarations.

TIP

It is good programming practice to include a comment about the variables you are planning to use and the type of data you will store in them.

In pseudocode, variable declarations are written as:

```
DECLARE <identifier> : <dataType>
```

For example, you may declare the following variables:

```
DECLARE Number1 : INTEGER // this declares Number1 to store a whole number
DECLARE YourName : STRING // this declares YourName to store a
                          // sequence of characters
DECLARE N1, N2, N3 : INTEGER  // declares 3 integer variables
DECLARE Name1, Name2 : STRING // declares 2 string variables
```

Syntax definitions

The syntax of variable declarations in language code is as follows:

Python	Python does not have variable declarations
VB.NET	`Dim` <identifier>[, <identifier>] `As <dataType>` Each line of declarations must start with the keyword `Dim`.
Java	`<datatype> <identifier>[, <identifier>];`

Code examples

Python	`# Number1 of type Integer`
	`# YourName of type String`
	`# N1, N2, N3 of type integer;`
	`# Name1, Name2 of type string;`
		There are no declarations, but comments should be made at the beginning of a module (see the section about comments at the end of Section 14.02).

VB.NET	`Dim Number1 As Integer`
	`Dim YourName As Integer`
	`Dim N1, N2, N3 As Integer`
	`Dim Name1, Name2 As String`
		You can group more than one variable of the same type on the same line.

Java	`int number1;`
	`String yourName;`
	`int n1, n2, n3;`
	`String name1, name2;`
		You can group more than one variable of the same type on the same line.

Declaration and assignment of constants

Sometimes we use a value in a solution that never changes, for example, the value of the mathematical constant pi (π). Instead of using the actual value in program statements, it is good practice and helps

readability, if we give a constant value a name and declare it at the beginning of the program.

In pseudocode, constant declarations are written as:

```
CONSTANT <identifier> = <value>
```

For example:

```
CONSTANT Pi = 3.14
```

Syntax definitions

Python	**<identifier> = <value>**
VB.NET	`Const` **<identifier> = <value>** Each line of declarations must start with the keyword `Const`.

Java	`static final <datatype> <identifier> = <value>;`
	Each line of constant declarations must start with the keywords
	`static final`

Code examples

Python	`PI = 3.14`	Python convention is to write constant identifiers using uppercase only. The values can be changed, although you should treat constants as not changeable.
VB.NET	`Const Pi = 3.14`	The value of a constant in VB.NET cannot be altered within the program.
Java	`static final double PI = 3.14;`	The value of a constant in Java cannot be altered within the program.

Assignment of variables

Once we have declared a variable, we can assign a value to it (See Chapter 12, Section 12.05).

In pseudocode, assignment statements are written as:

```
<identifier> ← <expression>
```

For example:

```
A ← 34
B ← B + 1
```

Syntax definitions and code examples

Python	`<identifier> = <expression>`	`A = 34` `B = B + 1`	The assignment operator is =
VB.NET	`<identifier> = <expression>`	`A = 34` `B = B + 1`	The assignment operator is =
Java	`<identifier> = <expression>;`	`A = 34;` `B = B + 1;`	The assignment operator is =

VB.NET allows you to initialise a variable as part of the declaration statement, for example:

```
Dim Number1 As Integer = 0
```

Java allows you to initialise a variable as part of the declaration statement, for example:

```
int number1 = 0;
```

VB.NET and Python allow increment statements such as B = B + 1 to be written as B += 1.

Java allows increment statements such as `b = b + 1` to be written as `b++;`

Arithmetic operators

Assignments don't just give initial values to variables. We also use an assignment when we need to store the result of a calculation. The arithmetic operators used for calculations are shown in Table 14.01.

Operation	Pseudocode	Python	VB.NET	Java
Addition	+	+	+	+
Subtraction	-	-	-	-
Multiplication	*	*	*	*
Division	/	/	/	/ (when dividing float or double types)
Exponent	^	**	^	No operator available, only method: `Math.pow(n,e)`
Integer division	`DIV`	`//`	`\`	/ (when dividing integer types)
Modulus	`MOD`	`%`	`Mod`	`%`

Table 14.01 Arithmetic operators

TIP

The result of integer division is the whole number part of the division. For example, 7 DIV 2 gives 3.

TIP

The result of the modulus operation is the remainder of a division. For example, 7 MOD 2 gives 1.

When more than one operator appears in an expression, the order of evaluation depends on the mathematical **rules of precedence**: parentheses, exponentiation, multiplication, division, addition, subtraction.

Question 14.01

Evaluate each of the following expressions:

4 * 3 – 3 ^ 2

(4 * 3 – 3) ^ 2

4 * (3 – 3) ^ 2

4 * (3 – 3 ^ 2)

Outputting information to the screen

In pseudocode, output statements are written as:

```
OUTPUT <string>
OUTPUT <identifier(s)>
```

When outputting text and data to the console screen, we can list a mixture of output strings and variable values in the print list.

Syntax definitions

Python	`print(<printlist>)` `print(<printlist>, end ='')`	Print list items are separated by commas (,). To avoid moving onto the next line after the output, use `end =''`)
VB.NET	`Console.WriteLine(<printlist>)` `Console.Write(<printlist>)`	Print list items are joined using &. `Console.WriteLine` will move onto the next line after the output; `Console.Write` will remain on the same line.

Java	`System.out.print(<printlist>);` `System.out.println(<printlist>);`	Print list items are joined using +. `System.out.println` will move onto the next line after the output; `System.out.print` will remain on the same line.

In the examples below, the print list consists of four separate items:

“Hello ” and “. Your number is ” are strings and

YourName and Number1 are variables, for which we print the value.

In pseudocode, we can indicate whether a new line should be output at the end by a comment at the end of the statement.

```
OUTPUT "Hello ", YourName, ". Your number is ", Number1 // newline
OUTPUT "Hello " // no new line
```

Code examples

Python

```
print("Hello ", YourName,
   ". Your number is ", Number1)
print("Hello ", end= '')
```

VB.NET

```
Console.WriteLine("Hello " & YourName &
   ". Your number is " & Number1)
Console.Write("Hello")
```

Java

```
System.out.println("Hello " + yourName +
   ". Your number is " + number1);
System.out.print("Hello");
```

In the code examples above you can see how output statements can be spread over more than one line when they are very long. You must break the line between two print list items. You cannot break in the middle of a string, unless you make the string into two separate strings.

In Python and VB.NET you can also use the placeholder method for output: the variables to be printed are represented by sequential numbers in { } in the message string and the variables are listed in the correct order after the string, separated by commas:

Python

```
print ("Hello {0}. Your number is {1}".format(YourName, Number1))
```

VB.NET

```
Console.WriteLine("Hello {0}. Your number is {1}", YourName,
Number1)
```

Getting input from the user

When coding an input statement, it is good practice to prompt the user as to what they are meant to enter. For example, consider the pseudocode statement:

```
INPUT "Enter a number: " A
```

Note the space between the colon and the closing quote. This is significant. It gives a space before the user types their input.

Code examples

Python

```
A = input("Enter a number: ")
```

The prompt is provided as a parameter to the input function. Single quotes are also accepted. All input is taken to be a string; if you want to use the input as a number the input string has to be converted using a function (see Section 14.07).

VB.NET

```
Console.Write("Enter a number: ")
A = Console.ReadLine()
```

The prompt has to be supplied as an output statement separately.

Java	`import java.util.Scanner;`
	`Scanner console = new Scanner(System.in);`
	`System.out.print("Enter a number: ");`
	`a = console.next();`
		The Scanner class has to be imported from the Java library first and a scanner object has to be created before it can be used to read an input string.
		The prompt has to be supplied as an output statement separately.

Comments

It is good programming practice to add comments to explain code where necessary.

Code examples

Python	`# this is a comment` `# this is another comment`
VB.NET	`' this is a comment` `' this is another comment`

Java	`// this is a comment`
	`// this is another comment`
	`/* this is a multi-line`
	`comment`
	`*/`

TASK 14.01

Use the IDE of your chosen programming language (in future just referred to as 'your language'). Type the program statements equivalent to the following pseudocode (you may need to declare the variable `YourName` first):

```
INPUT "What is your name? " YourName
OUTPUT "Have a nice day ", YourName
```

Save your program as Example1 and then run it. Is the output as you expected?

14.03 Data types

Every programming language has built-in data types. Table 14.02 gives a subset of those available. The number of bytes of memory allocated to a variable of the given type is given in brackets for VB.NET and Java.

Description of data	Pseudocode	Python	VB.NET	Java
Whole signed numbers	`INTEGER`	`int`	`Integer` (4 bytes)	`int` (4 bytes)
Signed numbers with a decimal point	`REAL`	`float`	`Single` (4 bytes) `Double` (8 bytes)	`float` (4 bytes) `double` (8 bytes)
A single character	`CHAR` Use single (`'`) quotation marks to delimit a character	Not available	`Char` (2 bytes – Unicode)	`char` (2 bytes – Unicode)
A sequence of characters (a string)	`STRING` Use double (`"`) quotation marks to delimit a string.	`str` (stored as ASCII but Unicode strings are also available) Use single (`'`), double (`"`) or triple (`'''` or `"""`) quotation marks to delimit a string.	`String` (2 bytes per character) Use double (`"`) quotation marks to delimit a string.	`String` (2 bytes per character) Use double (`"`) quotation marks to delimit a string.

Logical values:
True (represented as 1) and
False (represented as 0)
	`BOOLEAN`	`bool`
		possible values:
		`True`
		`False`
			`Boolean` (2 bytes)
			possible values:
			`True`
			`False`
				`Boolean`
				possible values:
				`true`
				`false`

Table 14.02 Simple data types

In Python, a single character is represented as a string of length 1.

In VB.NET, each character in a string requires two bytes of memory and each character is represented in memory as Unicode (in which, the values from 1 to 127 correspond to ASCII).

Date has various internal representations but is output in conventional format.

Description of data	Pseudocode	Python	VB.NET	Java
Date value	`DATE`	Use the datetime class	`Date` (8 bytes)	Date is a class in Java. To make use of it use: `import java.util.Date;`

Table 14.03 The Date data types

In Python and Java, date is not available as a built-in data type. Date is provided as a class (see Table 14.03).

VB.NET stores dates and times from 1.1.0001 (0 hours) to 31.12.9999 (23:59:59 hours) with a resolution of 100 nanoseconds (this unit is called a ‘tick’). Floating-point (decimal) numbers are stored in binary-coded decimal format (see Section 1.02).

There are many more data types. Programmers can also design and declare their own data types (see Chapter 16 (Section 16.01) and Chapter 26 (Section 26.01).

TASK 14.02

1. Look at the identifier tables in Chapter 12 (Tables 12.02 and 12.04 to 12.12). Decide which data type from your language is appropriate for each variable listed.
2. Write program code to implement the pseudocode from Worked Example 12.01 in Chapter 12.

14.04 Boolean expressions

In Chapter 12 (Section 12.06), we covered logic statements. These were statements that included a condition. Conditions are also known as Boolean expressions and evaluate to either True or False. True and False are known as Boolean values.

Simple Boolean expressions involve comparison operators (see Table 14.04). Complex Boolean expressions also involve Boolean operators (see Table 14.05).

Operation	Pseudocode	Python	VB.NET	Java
equal	=	==	=	==
not equal	<>	!=	<>	!=
greater than	>	>	>	>
less than	<	<	<	<
greater than or equal to	>=	>=	>=	>=
less than or equal to	<=	<=	<=	<=

Table 14.04 Comparison operators

Operation	Pseudocode	Python	VB.NET	Java
AND (logical conjunction)	`AND`	`and`	`And`	`&&`
OR (logical inclusion)	`OR`	`or`	`Or`	`||`
NOT (logical negation)	`NOT`	`not`	`Not`	`!`

Table 14.05 Boolean operators

14.05 Selection

IF...THEN statements

In pseudocode the IF...THEN construct is written as:

```
IF <Boolean expression>
  THEN
    <statement(s)>
ENDIF
```

Syntax definitions

Python	`if <Boolean expression>:` `  <statement(s)>`	Note that the `THEN` keyword is replaced by a colon (:). Indentation is used to show which statements form part of the conditional statement.

VB.NET	`If <Boolean expression> Then`
	`  <statement(s)>`
	`End If`
		Note the position of `Then` on the same line as the Boolean expression. The `End If` keywords should line up with the `If` keyword.

Java	`if (<Boolean expression>)` `  <statement>;`	Note that the Boolean expression is enclosed in brackets. If more than one statement is required as part of the conditional statement, the statements must be enclosed in braces { }.

Pseudocode example:

```
IF x < 0
  THEN
    OUTPUT "Negative"
ENDIF
```

Code examples

Python	`if x < 0:` `  print("Negative")`

VB.NET	`If x < 0 Then`
	`  Console.WriteLine("Negative")`
	`End If`

Java	`if (x < 0)` `  System.out.println("Negative");`

TASK 14.03

Write program code to implement the pseudocode from Worked Example 12.03 in Chapter 12.

IF...THEN...ELSE statements

In pseudocode, the IF...THEN...ELSE construct is written as:

```
IF <Boolean expression>
  THEN
    <statement(s)>
  ELSE
    <statement(s)>
ENDIF
```

Syntax definitions

Python	`if <Boolean expression>:`
	`  <statement(s)>`
	`else:`
	`  <statement(s)>`
		Indentation is used to show which statements form part of the conditional statement; the `else` keyword must line up with the corresponding `if` keyword.

VB.NET	`If <Boolean expression> Then`
	`  <statement(s)>`
	`Else`
	`  <statement(s)>`
	`End If`
		The `Else` keyword is on its own on a separate line. It is good programming practice to line it up with the corresponding `If` keyword and indent the statements within the conditional statement.

Java	`if (<Boolean expression>)`
	`  <statement>;`
	`else`
	`  <statement>;`
		If more than one statement is required in the `else` part of the statement, the statements must be enclosed in braces `{ }`.

Pseudocode example:

```
IF x < 0
  THEN
    OUTPUT "Negative"
  ELSE
    OUTPUT "Positive"
ENDIF
```

Code examples

Python	`if x < 0:`
	`  print("Negative")`
	`else:`
	`  print("Positive")`

VB.NET	`If x < 0 Then`
	`  Console.WriteLine("Negative")`
	`Else`
	`  Console.WriteLine("Positive")`
	`End If`

Java	`if (x < 0)`
	`  System.out.println("Negative");`
	`else`
	`  System.out.println("Positive");`

Nested IF statements

In pseudocode, the nested IF statement is written as:

```
IF <Boolean expression>
  THEN
    <statement(s)>
  ELSE
    IF <Boolean expression>
      THEN
        <statement(s)>
      ELSE
        <statement(s)>
    ENDIF
ENDIF
```

Syntax definitions

Python	`if <Boolean expression>:`
	`  <statement(s)>`
	`elif <Boolean expression>:`
		Note the keyword `elif` (an abbreviation of `else if`). This keyword must line up

	<statement(s)> else: <statement(s)>	with the corresponding `if`. There can be as many `elif` parts to this construct as required.

VB.NET	If <Boolean expression> Then
	<statement(s)>
	ElseIf
	<statement(s)>
	Else
	<statement(s)>
	End If
		If `ElseIf` is used as one word, only one `End If` is required at the end of this construct.
		There can be as many `ElseIf` parts as required.

Java	if (<Boolean expression>)
	<statement>;
	else if (<Boolean expression>)
	<statement>;
	else
	<statement>;	

Pseudocode example:

```
IF x < 0
  THEN
    OUTPUT "Negative"
  ELSE
    IF x = 0
      THEN
        OUTPUT "Zero"
      ELSE
        OUTPUT "Positive"
    ENDIF
ENDIF
```

Code examples

Python	`if x < 0:`
	`  print("Negative")`
	`elif x == 0:`
	`  print("Zero")`
	`else:`
	`  print("Positive")`

VB.NET	`If x < 0 Then`
	`  Console.WriteLine("Negative")`
	`ElseIf x = 0 Then`
	`  Console.WriteLine("Zero")`
	`Else`
	`  Console.WriteLine("Positive")`
	`End If`

Java	`if (x < 0)`
	`{`
	`  System.out.println('Negative');`
	`}`
	`else if (x == 0)`
	`{`
	`  System.out.println('Zero');`
	`}`
	`else`
	`{`
	`  System.out.println('Positive');`
	`}`

TASK 14.04

Write program code to implement the pseudocode from Worked Example 12.02 in Chapter 12.

CASE statements

An alternative selection construct is the CASE statement. Each considered CASE condition can be:

- a single value
- single values separated by commas
- a range.

In pseudocode, the CASE statement is written as:

```
CASE OF <expression>
  <value1>             : <statement(s)>
  <value2>,<value3>    : <statement(s)>
  <value4> TO <value5> : <statement(s)>
  .
  .
  OTHERWISE <statement(s)>
ENDCASE
```

The value of `<expression>` determines which statements are executed. There can be as many separate cases as required. The `OTHERWISE` clause is optional and useful for error trapping.

Syntax definitions

Python	Python does not have a CASE statement. You need to use nested If statements instead.

VB.NET	`Select Case <expression>`
	`  Case value1`
	`  <statement(s)>`
	`  Case value2,value3`
	`  <statement(s)>`
	`  Case value4 To value5`
	`  <statement(s)>`
	`  .`
	`  .`
	`  .`
	`  Case Else`
	`  <statement(s)>`
	`End Select`

Java	`switch (<expression>)`
	`{`
	`  case value1:`
	`  <statement(s)>;`
	`  break;`
	`  case value2: case value3:`
	`  <statement(s)>;`
	`  break;`
	`  .`
	`  .`
	`  .`
	`  default: <statement(s)>;`
	`}`

In pseudocode, an example CASE statement is:

```
CASE OF Grade
  "A"      : OUTPUT "Top grade"
  "F", "U" : OUTPUT "Fail"
  "B".."E" : OUTPUT "Pass"
OTHERWISE OUTPUT "Invalid grade"
ENDCASE
```

Code examples

Python	`if Grade == "A":`
	`  print("Top grade")`
	`elif Grade == "F" or Grade == "U":`

	print("Fail")
	elif Grade in ("B", "C", "D", "E"):
	print("Pass")
	else:
	print("Invalid grade")

VB.NET	Select Case Grade Case "A" Console.WriteLine("Top grade") Case "F","U" Console.WriteLine("Fail") Case "B" To "E" Console.WriteLine("Pass") Case Else Console.WriteLine("Invalid grade") End Select

Java	switch (grade)
	{
	case 'A':
	System.out.println("Top Grade");
	break;
	case 'F': case 'U':
	System.out.println("Fail");
	break;
	case 'B': case 'C': case 'D': case 'E':
	System.out.println("Pass");
	break;
	default:
	System.out.println("Invalid grade");
	}

TASK 14.05

The problem to be solved: the user enters the number of the month and year. The output is the number of days in that month. The program has to check if the year is a leap year for February.

The pseudocode solution is:

```
INPUT MonthNumber
INPUT Year
Days ← 0
CASE OF MonthNumber
  CASE 1,3,5,7,8,10,12 : Days ← 31
  CASE 4,6,9,11 : Days ← 30
  CASE 2 : Days ← 28
    If Year MOD 400 = 0
      THEN // it is a leap year
        Days ← 29
    ENDIF
    IF (Year MOD 4 = 0) AND (Year MOD 100 > 0)
      THEN // it is a leap year
        Days ← 29
    ENDIF
  OTHERWISE OUTPUT "Invalid month number"
ENDCASE
OUTPUT Days
```

Write program code to implement the pseudocode above.

14.06 Iteration

Count-controlled (FOR) loops

In pseudocode, a count-controlled loop is written as:

```
FOR <control variable> ← s TO e STEP i // STEP is optional
    <statement(s)>
NEXT <control variable>
```

The control variable starts with value `s`, increments by value `i` each time round the loop and finishes when the control variable reaches the value `e`.

Syntax definitions

Python	`for <control variable> in range(s, e, i):` `    <statement(s)>`	The values `s`, `e` and `i` must be of type integer. The loop finishes when the control variable is just below `e`. The values for `s` and `i` can be omitted and they default to 0 and 1, respectively.

VB.NET	`For <control variable> = s To e Step i`
	`    <statement(s)>`
	`Next`
		The values `s`, `e` and `i` can be of type integer or float.

Java	`for (int i = s; i < e; i++)` `    <statement>;`	Where `i` is the control variable.

In pseudcode, examples are:

```
FOR x ← 1 TO 5
    OUTPUT x
NEXT x
FOR x = 2 TO 14 STEP 3
    OUTPUT x
NEXT x
FOR x = 5 TO 1 STEP -1
    OUTPUT x
NEXT x
```

Code examples

Python	`for x in range(5):` `    print(x, end=' ')`	The start value of x is 0 and it increases by 1 on each iteration. Output: `0 1 2 3 4`
	`for x in range(2, 14, 3):` `    print(x, end=' ')`	Output: `2 5 8 11`
	`for x in range(5, 1, -1):` `    print(x, end=' ')`	The start value of x is 5 and it decreases by 1 on each iteration. Output: `5 4 3 2`
	`for x in ["a", "b", "c"]:` `    print(x, end='')`	The control variable takes the value of each of the group elements in turn. Output: `abc`

VB.NET	`For x = 1 To 5`
	`    Console.Write(x)`
	`Next`
		Output: `1 2 3 4 5`

	`For x = 2 To 14 Step 3`	Output: `2 5 8 11 14`

	`    Console.Write(x)` `Next`	

	`For x = 5 To 1 Step -1`
	`    Console.Write(x)`
	`Next`
		Output: `5 4 3 2 1`

	`For x = 1 To 2.5 Step 0.5`
	`    Console.WriteLine(x)`
	`Next`
		`Output:`
		`1`
		`1.5`
		`2`
		`2.5`

	`For Each x In {"a", "b", "c"}`
	`    Console.Write(x)`
	`Next`
		The control variable takes the value of each of the group elements in turn.
		Output: `abc`

Java	`for (int x = 1; x < 6; x++)`
	`{`
	`    System.out.print(x);`
	`}`
		Output: `12345`

	`for (int x = 2; x < 15; x = x + 3)`
	`{`
	`    System.out.print(x + "  ");`
	`}`
		Output: `2  5  8  11  14`

	`for (int x = 5; x > 0; x--)`
	`{`
	`    System.out.print(x + "  ");`
	`}`
		Output: `5  4  3  2  1`

	`for (double x = 1; x < 3; x = x + 0.5)`
	`{`
	`    System.out.print(x + "  ");`
	`}`
		Output: `1.0  1.5  2.0  2.5`

	`char[] letter = {'a', 'b', 'c'};`
	`for (char x : letter )`
	`{`
	`   System.out.print(x);`
	`}`
		The control variable takes the value of each of the group elements in turn.
		Output: `abc`

TASK 14.06

1. Write program code to implement the pseudocode from Worked Example 12.05 in Chapter 12.
2. Write program code to implement the pseudocode from Worked Example 12.08 in Chapter 12.
3. Write program code to implement the pseudocode from Worked Example 12.09 in Chapter 12.

Post-condition loops

A post-condition loop, as the name suggests, executes the statements within the loop at least once. When the condition is encountered, it is evaluated. As long as the condition evaluates to False, the statements within the loop are executed again. When the condition evaluates to True, execution will go to the next statement after the loop.

When coding a post-condition loop, you must ensure that there is a statement within the loop that will at some point change the end condition to True. Otherwise the loop will execute forever.

In pseudocode, the post-condition loop is written as:

```
REPEAT
    <statement(s)>
UNTIL <condition>
```

Syntax definitions

Python	Post-condition loops are not available in Python. Use a pre-condition loop instead.

VB.NET	`Do`
	`    <statement(s)>`
	`Loop Until <condition>`

Java	`do`
	`{`
	`    <statement(s)>`
	`} while <condition>;`

Pseudocode example:

```
REPEAT
    INPUT "Enter Y or N: " Answer
UNTIL Answer = "Y"
```

Code examples

VB.NET	`Do`
	`   Console.Write("Enter Y or N: ")`
	`   Answer = Console.ReadLine()`
	`Loop Until Answer = "Y"`

Java	`do`
	`{`
	`   System.out.print("Enter Y or N: ");`
	`   answer = console.next();`
	`} while  (!(answer.equals("Y")));`

TASK 14.07

1. Write program code to implement the pseudocode from Worked Example 12.04 in Chapter 12.
2. Write program code to implement the first algorithm from Worked Example 12.06 in Chapter 12.

Pre-condition loops

Pre-condition loops, as the name suggests, evaluate the condition before the statements within the loop are executed. Pre-condition loops will execute the statements within the loop as long as the condition evaluates to True. When the condition evaluates to False, execution will go to the next statement after the loop. Note that any variable used in the condition must not be undefined when the loop structure is first encountered.

When coding a pre-condition loop, you must ensure that there is a statement within the loop that will at some point change the value of the controlling condition. Otherwise the loop will execute forever.

In pseudocode the pre-condition loop is written as:

```
WHILE <condition> DO
    <statement(s)>
ENDWHILE
```

Syntax definitions

Python	`while <condition>:` `    <statement(s)>`	Note that statements within the loop must be indented by a set number of spaces. The first statement after the

		loop must be indented less.

VB.NET	`Do While <condition>`
	`    <statement(s)>`
	`Loop`
	`Do Until <condition>`
	`    <statement(s)>`
	`Loop`
		Note the keyword `Loop` indicates the end of the loop.
		VB.NET also has a pre-condition `Until` loop. This will execute the statements within the loop as long as the condition evaluates to False. If the condition evaluates to True when the loop is first encountered, the statements within the loop are not executed at all.

Java	`while (<condition>)`
	`{`
	`    <statement(s)>;`
	`}`	

Pseudocode example,

```
Answer ← ""
WHILE Answer <> "Y" DO
    INPUT "Enter Y or N: " Answer
ENDWHILE
```

Code examples

Python	`Answer = ''`
	`while Answer != 'Y':`
	`    Answer = input("Enter Y or N: ")`

VB.NET	`Dim Answer As String = ""`
	`Do While Answer <> "Y"`
	`    Console.Write("Enter Y or N: ")`
	`    Answer = Console.ReadLine()`
	`Loop`
	`Answer = ""`
	`Do Until Answer = "Y"`
	`    Console.Write("Enter Y or N: ")`
	`    Answer = Console.ReadLine()`
	`Loop`

Java	`String answer = "";`
	`while(answer.equals("Y") == false)`
	`{`
	`    System.out.print("Enter Y or N: ");`
	`    answer = console.next();`
	`}`

TASK 14.08

Write program code to implement the second algorithm from Worked Example 12.06 in Chapter 12.

Which loop structure to use?

If you know how many times around the loop you need to go when the program execution gets to the loop statements, use a count-controlled loop. If the termination of the loop depends on some condition determined by what happens within the loop, then use a conditional loop. A pre-condition loop has the added benefit that the loop may not be entered at all, if the condition does not require it.

TIP

Computer Scientists like efficient code. Choosing the most suitable types of selection statements and loop structures is an important step along the way to design efficient

code.

14.07 Built-in functions

Programming environments provide many built-in functions. Some of them are always available to use; some need to be imported from specialist module libraries.

Discussion Point:

Investigate your own programming environment and research other library routines.

String manipulation functions

Table 14.06 contains some useful functions for manipulating strings.

Description	Pseudocode	Python	VB.NET	Java
Access a single character using its position P in a string `ThisString`	`ThisString[P]` Counts from 1	`ThisString[P]` Counts from 0	`ThisString(P)` Counts from 0	`ThisString.charAt(P)` Counts from 0
Returns the character whose ASCII value is i	`CHR(i : INTEGER) RETURNS CHAR`	`chr(i)`	`Chr(i)`	`(char) i;`
Returns the ASCII value of character ch	`ASC(ch) RETURNS INTEGER`	`ord(ch)`	`Asc(ch)`	`(int) ch;`
Returns the integer value representing the length of string S	`LENGTH(S : STRING) RETURNS INTEGER`	`len(S)`	`len(S)`	`S.length();`
Returns leftmost L characters from S	`LEFT(S : STRING, L : INTEGER) RETURNS STRING`	`S[0:L]` See the next section, on slicing	`Left(S, L)`	`S.subString (0, L)`
Returns rightmost L characters from S	`RIGHT(S: STRING, L : INTEGER) RETURNS STRING`	`S[-L:]` See the next section, on slicing	`Right(S, L)`	`S.subString (S.length() – L)`
Returns a string of length L starting at position P from S	`MID(S : STRING, P : INTEGER, L : INTEGER) RETURNS STRING`	`S[P : P + L]` See the next section, on slicing	`mid(S, P, L)`	`S.subString(P, P + L)`
Returns the character value representing the lower case equivalent of Ch	`LCASE(Ch : CHAR) RETURNS CHAR`	`Ch.lower()`	`LCase(Ch)`	`Character.toLowerCase(ch)`
Returns the character value representing the upper case equivalent of Ch	`UCASE(Ch : CHAR) RETURNS CHAR`	`Ch.upper()`	`UCase(Ch)`	`Character.toUpperCase(ch)`
Returns a string formed by converting all alphabetic characters of S to upper case	`TO_UPPER(S : STRING) RETURNS STRING`	`S.upper()`	`S.ToUpper`	`S.toUpperCase()`
Returns a string formed by converting all alphabetic characters of S to lower	`TO_LOWER(S : STRING) RETURNS STRING`	`S.lower()`	`S.ToLower`	`S.toLowerCase()`

case				
Concatenate (join) two strings	`S1 & S2`	`s = S1 + S2`	`s = S1 + S2` `s = S1 & S2`	`s = S1 + S2;`

Table 14.06 Some useful string manipulation functions

Slicing in Python

In Python a subsequence of any sequence type (e.g. lists and strings) can be created using 'slicing'.

For example, to get a substring of length L from position P in string S we write S[P : P + L].

Figure 13.05 shows a representation of `ThisString`. If we want to return a slice of length 3 starting at position 3, we use `ThisString[3 : 6]` to give 'DEF'. Position is counted from 0 and the position at the upper bound of the slice is not included in the substring.

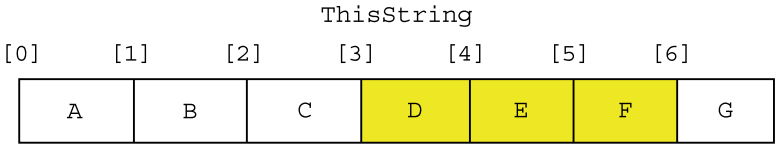


Figure 14.05 A representation of `ThisString`

If you imagine the numbering of each element to start at the left-hand end (as shown in Figure 14.05), then it is easier to see how the left element (the lower bound) is included, but the right element (the upper bound) is excluded. Table 14.07 shows some other useful slices in Python.

Expression	Result	Explanation
`ThisString[2:]`	`CDEFG`	If you do not state the upper bound, the slice includes all characters to the end of the string.
`ThisString[:2]`	`AB`	If you do not state the lower bound, the slice includes all characters from the beginning of the string.
`ThisString[-2:]`	`FG`	A negative lower bound means that it takes the slice starting from the end of the string.
`ThisString[:-2]`	`ABCDE`	A negative upper bound means that it terminates the string at that position.

Table 14.07 Some useful slices in Python

Truncating numbers

Instead of rounding, sometimes we just want the whole number part of a real number.

This is known as 'truncation'.

Pseudocode	`INT(x : REAL) RETURNS INTEGER`	Returns the integer part of `x`.
Python	`int(x)`	If `x` is a floating-point number, the conversion truncates towards zero.
VB.NET	`Math.Truncate(x)`	The whole number part of the real number `x` is returned.
Java	`(int) x;`	Casts the number as an integer.

Converting a string to a number

Sometimes a whole number may be held as a string. To use such a number in a calculation, we first

need to convert it to an integer. For example, these functions return the integer value 5 from the string `"5"`:

Python	`int(S)`
VB.NET	`CInt(S)`
Java	`Integer.valueOf(S)`

Sometimes a number with a decimal point may be held as a string. To use such a number in a calculation, we first need to convert it to a real (float). For example, these functions return the real number 75.43 from the string `"75.43"`:

Pseudocode	`STRING_TO_NUM(x : STRING) RETURNS REAL`	Returns a numeric representation of a string.
Python	`float(x)`	The returned value is a floating-point number.
VB.NET	`CDbl(x)`	The returned value is of type double.
Java	`Float.valueOf(x)`	The returned value is a floating-point number.

Random number generator

Random numbers are often required for simulations. Most programming languages have various random number generators available. As the random numbers are generated through a program, they are referred to as 'pseudo-random' numbers. A selection of the most useful random number generators are shown in the following code examples.

Code examples

Python	`# in the random library:` `randint(1, 6)`	This code produces a random number between 1 and 6 inclusive.

VB.NET	`Dim RandomNumber As New Random`
	`Dim x As Integer`
	`x = RandomNumber.Next(1, 6)`
		You have to set up a RandomNumber object (see Chapter 27). This code generates an integer between 1 (inclusive) and 6 (exclusive).

Java	`import java.util.Random;`
	`Random randomNumber = new Random();`
	`int x = randomNumber.nextInt(6) + 1;`
		You have to set up a RandomNumber object (see Chapter 27). This code generates an integer between 1 (inclusive) and 6 (inclusive).

TASK 14.09

1. Write program code to generate 20 random numbers in the range 1 to 10 inclusive.
2. Write program code to implement the pseudocode using a pre-condition loop from Worked Example 12.07 in Chapter 12.

Discussion Point:

What other useful functions can you find? Which module libraries have you searched?

14.08 Procedures

In Chapter 12 (Section 12.09), we used procedures as a means of giving a group of statements a name. When we want to program a procedure we need to define it before the main program. We call it in the main program when we want the statements in the procedure body to be executed.

In pseudocode, a procedure definition is written as:

```
PROCEDURE <procedureIdentifier> // this is the procedure header
    <statement(s)>   // these statements are the procedure body
ENDPROCEDURE
```

This procedure is called using the pseudocode statement:

```
CALL <procedureIdentifier>
```

Syntax definitions

Python	`def <identifier>():` `    <statement(s)>`

VB.NET	`Sub <identifier>()`
	`    <statement(s)>`
	`End Sub`

Java	`void <identifier>()`
	`{`
	`    <statement(s)>;`
	`}`

When programming a procedure, note where the definition is written and how the procedure is called from the main program.

Here is an example pseudocode procedure definition:

```
PROCEDURE InputOddNumber
    REPEAT
        INPUT "Enter an odd number: " Number
    UNTIL Number MOD 2 = 1
    OUTPUT "Valid number entered"
ENDPROCEDURE
```

This procedure is called using the CALL statement:

```
CALL InputOddNumber
```

Code examples

Python

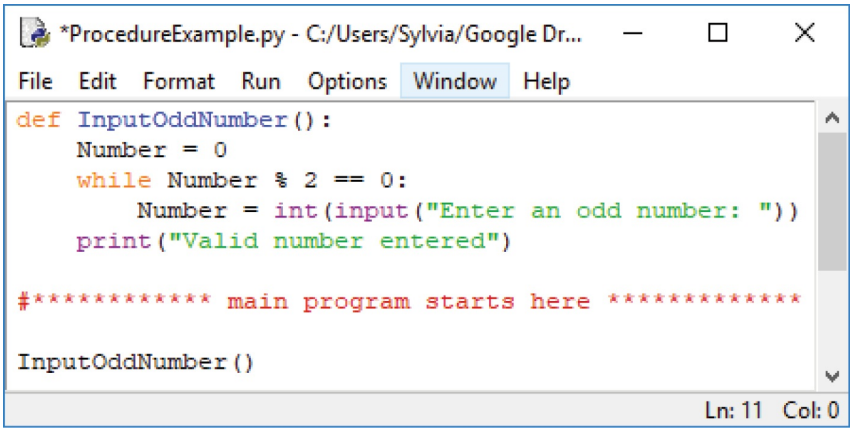


Figure 14.06 The Python editor with a procedure

The Python editor colour-codes the different parts of a statement. This helps when you are typing your own code. The indentation shows which statements are part of the loop.
The built-in function `input` returns a string, which must be converted to an integer

before it can be used as a number.

VB.NET

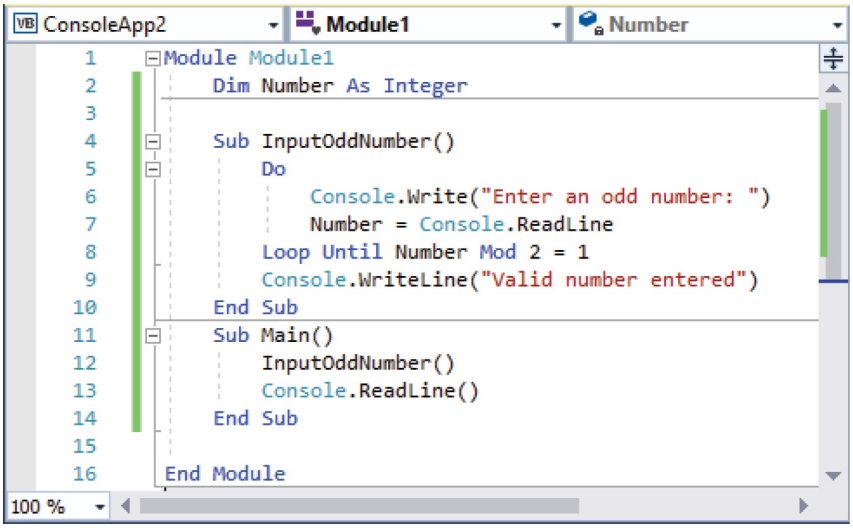


Figure 14.07 The Visual Basic Express editor with a procedure

The Visual Basic Express editor colour-codes different parts of the statement, so it is easy to see if syntax errors are made. The editor also auto-indents and capitalises keywords.
Variables need to be declared before they are used. The editor will follow the capitalisation of the variable declaration when you type an identifier without following your original capitalisation.
The editor is predictive: pop-up lists will show when you type the first part of a statement.
When you execute the Main program, `Console.ReadLine()` keeps the run-time window open.

Java

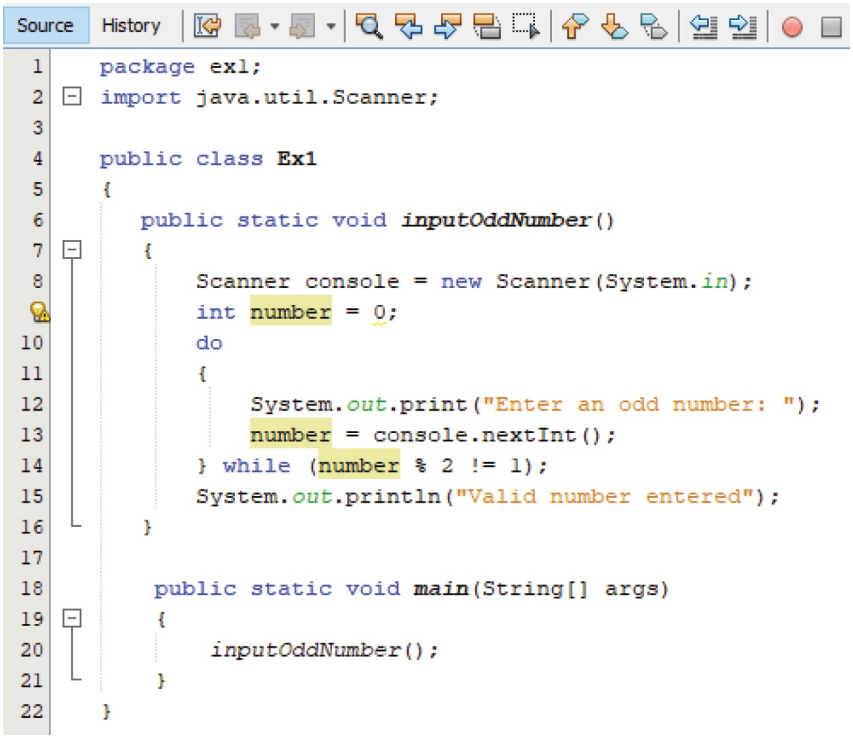


Figure 14.08 The NetBeans editor with a void method

The editor automatically colour codes keyword and strings.
The procedure body is enclosed within braces { and }.
The editor is predictive: pop-up lists will show when you type the first part of a statement.

Variables need to be declared before they are used.

TASK 14.10

Write program code to implement the pseudocode from Worked Example 12.11 in Chapter 12.

14.09 Functions

In Section 14.07 we used built-in functions. These are useful subroutines written by other programmers and made available in module libraries. The most-used ones are usually in the system library, so are available without you having to import them.

You can write your own functions. Any function you have written can be used in another program if you build up your own module library.

A function is used as part of an expression. When program execution gets to the statement that includes a function call as part of the expression, the function is executed. The **return value** from this function call is then used in the expression.

When writing your own function, ensure you always return a value as part of the statements that make up the function (the function body). You can have more than one RETURN statement if there are different paths through the function body.

In pseudocode, a function definition is written as:

```
FUNCTION <functionIdentifier> RETURNS <dataType> // function header
    <statement(s)> // function body
    RETURN <value>
ENDFUNCTION
```

Syntax definitions

Python	`def <functionIdentifier>():`
	`    <statement(s)>`
	`    return <value>`

VB.NET	`Function <functionIdentifier>() As <dataType>`
	`    <statement(s)>`
	`    <functionIdentifier> = <value> 'Return <value>`
	`End Function`

Java	`<data type> <functionIdentifier>()`
	`{`
	`    <statement(s)>;`
	`    return <value>;`
	`}`

When programming a function, the definition is written in the same place as a procedure. The function is called from within an expression in the main program, or in a procedure.

Different programming languages use different terminology for their subroutines, as listed in Table 14.08.

Pseudocode	PROCEDURE	FUNCTION
Python	void function	fruitful function
VB	Subroutine	Function
Java	void method	method

Table 14.08 Programming language terminology for subroutines

Void means ‘nothing’. Both Python and Java use this term to show that their procedure-type subroutine does not return a value. Python refers to both types of subroutines as functions. The fruitful function returns one or more values.

We can write the example procedure from Section 14.09 as a function. In pseudocode, this is:

```
FUNCTION InputOddNumber RETURNS INTEGER
    REPEAT
        INPUT "Enter an odd number: " Number
    UNTIL Number MOD 2 = 1
    OUTPUT "Valid number entered"
    RETURN Number
```

```
ENDFUNCTION
```

Code examples

Python

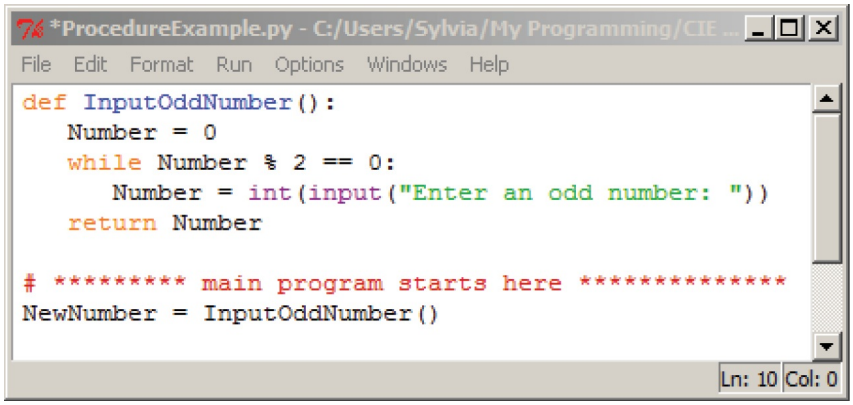
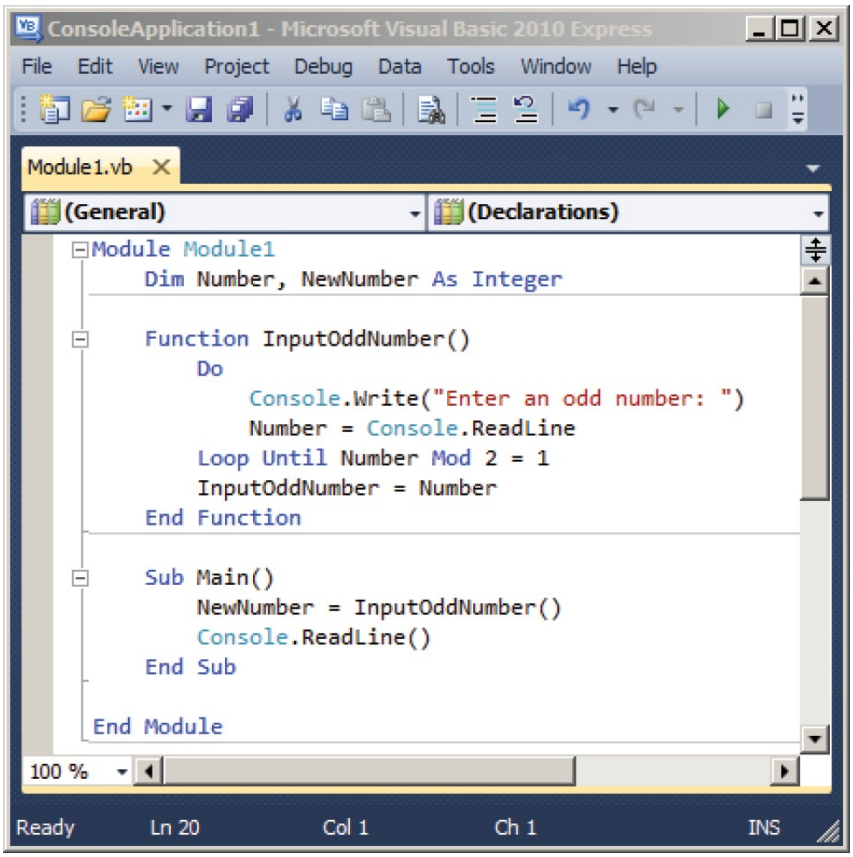


Figure 14.09 The Python editor with a function and local variable

The variable `Number` in Figure 14.09 is not accessible in the main program. Python's variables are local unless declared to be global.

VB.NET

(a)



(b)

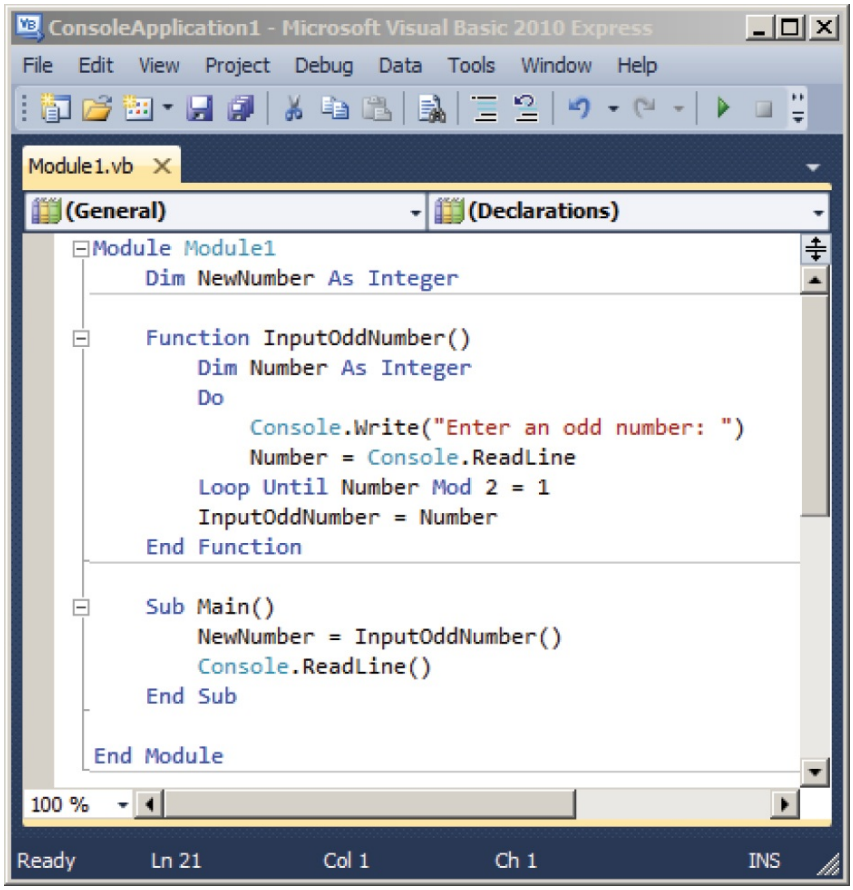


Figure 14.10 The VB.NET editor with (a) global variables and (b) a local variable

The variable `Number` in Figure 14.10(a) is declared as a global variable at the start of the module. This is not good programming practice.

In Figure 14.10(b), the variable `Number` is declared as a local variable within the function.

Java

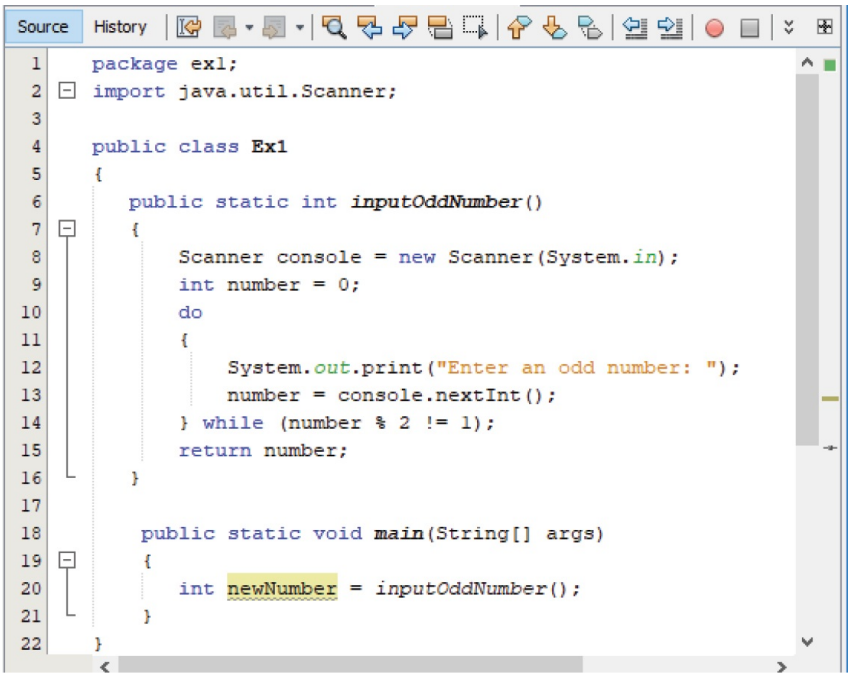


Figure 14.11 The NetBeans editor with a function and local variable

The variable `number` in Figure 14.11 is not accessible in the main program.

A global variable is available in any part of the program code. It is good programming practice to declare a variable that is only used within a subroutine as a local variable.

In Python, every variable is local, unless it is overridden with a global declaration. In VB.NET you need

to write the declaration statement for a local variable within the subroutine. Java does not support global variables. However, static variables declared in a class are accessible throughout the class.

TASK 14.11

Write program code to implement the pseudocode from Worked Example 13.05 in Chapter 13. Which variables are global and which are local?

14.10 Passing parameters to subroutines

When a subroutine requires one or more values from the main program, we supply these as **arguments** to the subroutine at call time. This is how we use built-in functions. We don’t need to know the identifiers used within the function when we call a built-in function.

When we define a subroutine that requires values to be passed to the subroutine body, we use a parameter list in the subroutine header. When the subroutine is called, we supply the arguments in brackets. The arguments supplied are assigned to the corresponding **parameter** of the subroutine (note the order of the parameters in the parameter list must be the same as the order in the list of arguments). This is known as the **subroutine interface**.

14.11 Passing parameters to functions

The **function header** is written in pseudocode as:

```
FUNCTION <functionIdentifier> (<parameterList>) RETURNS <dataType>
```

where `<parameterList>` is a list of identifiers and their data types, separated by commas.

Here is an example pseudocode function definition that uses parameters:

```
FUNCTION SumRange(FirstValue : INTEGER, LastValue : INTEGER) RETURNS INTEGER
    DECLARE Sum, ThisValue : INTEGER
    Sum ← 0
    FOR ThisValue ← FirstValue TO LastValue
        Sum ← Sum + ThisValue
    NEXT ThisValue
    RETURN Sum
ENDFUNCTION
```

Code examples

Python

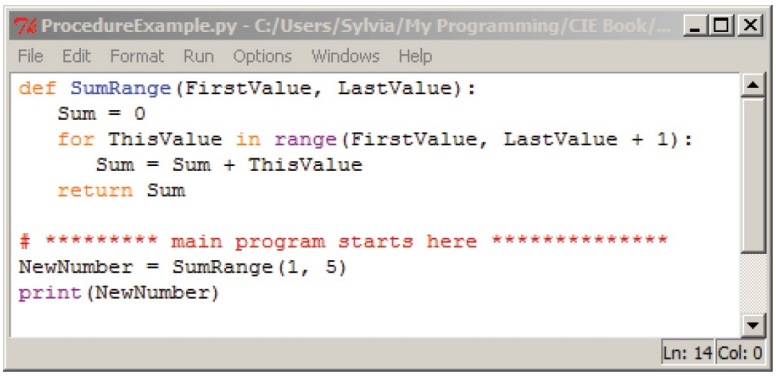


Figure 14.12 The `SumRange()` function in Python

VB.NET

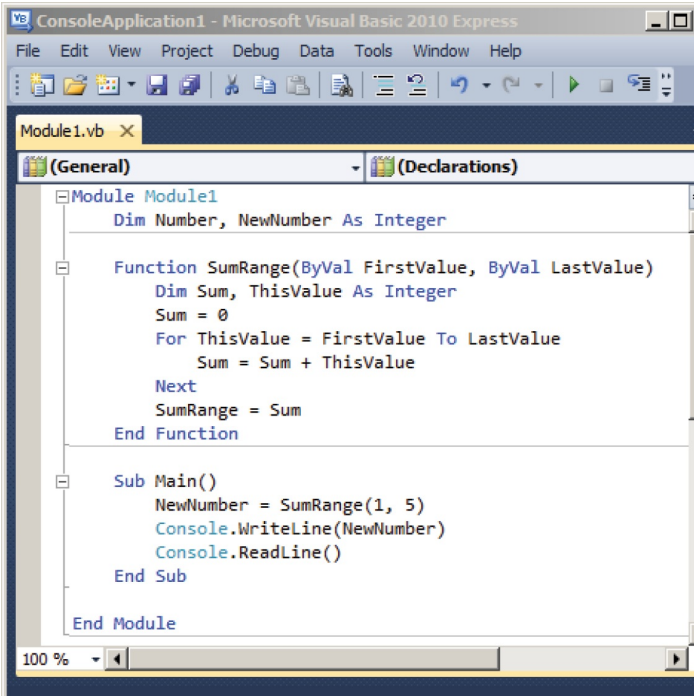
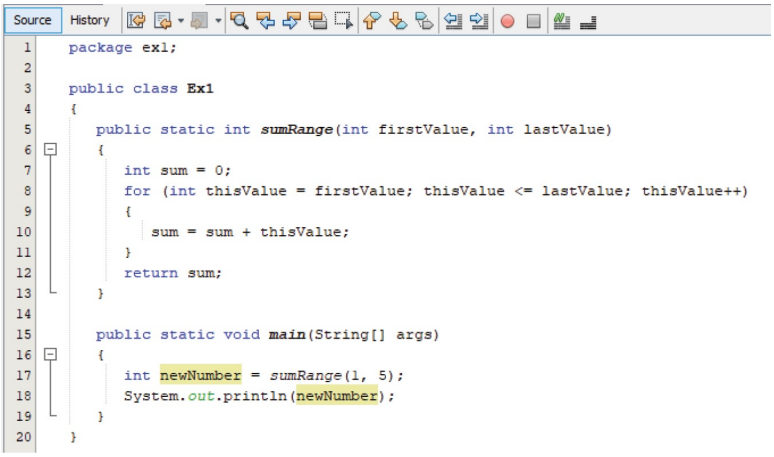


Figure 14.13 The `SumRange()` function in VB.NET

Java



```java
package ex1;

public class Ex1
{
   public static int sumRange(int firstValue, int lastValue)
   {
      int sum = 0;
      for (int thisValue = firstValue; thisValue <= lastValue; thisValue++)
      {
         sum = sum + thisValue;
      }
      return sum;
   }

   public static void main(String[] args)
   {
      int newNumber = sumRange(1, 5);
      System.out.println(newNumber);
   }
}
```

Figure 14.14 The `SumRange()` function in Java

TASK 14.12

Write a function to implement the following pseudocode:

```
FUNCTION Factorial (Number : INTEGER) RETURNS INTEGER
    DECLARE Product : INTEGER
    Product ← 1
    FOR n ← 2 TO Number
        Product ← Product * n
    NEXT n
    RETURN Product
ENDFUNCTION
```

14.12 Passing parameters to procedures

If a parameter is passed **by value**, at call time the argument can be an actual value (as we showed in the code examples in Section 14.11). If the argument is a variable, then a copy of the current value of the variable is passed into the subroutine. The value of the variable in the calling program is not affected by what happens in the subroutine.

For procedures, a parameter can be passed **by reference**. At call time, the argument must be a variable. A pointer to the memory location (the address) of that variable is passed into the procedure. Any changes that are applied to the variable's contents will be effective outside the procedure in the calling program/module.

Note that neither of these methods of parameter passing applies to Python. In Python or Java, the method is called pass by object reference. This is basically an object-oriented way of passing parameters and is beyond the scope of this chapter (objects are dealt with in Chapter 27). The important point is to understand how to program in Python and Java to get the desired effect.

The full **procedure header** is written in pseudocode, in a very similar fashion to that for function headers, as:

```
PROCEDURE <ProcedureIdentifier> (<parameterList>)
```

The parameter list needs more information for a procedure definition. In pseudocode, a parameter in the list is represented in one of the following formats:

```
BYREF <identifier1> : <dataType>
BYVALUE <identifier2> : <dataType>
```

Passing parameters by value

The pseudocode for the pyramid example in Chapter 12 (Section 12.09) includes a procedure definition that uses two parameters passed by value. We can now make that explicit:

```
PROCEDURE OutputSymbols(BYVALUE NumberOfSymbols : INTEGER, Symbol : CHAR)
    DECLARE Count : INTEGER
    FOR Count ← 1 TO NumberOfSymbols
        OUTPUT Symbol // without moving to next line
    NEXT Count
    OUTPUT NewLine
ENDPROCEDURE
```

In Python (see Figure 14.15), all parameters behave like local variables and their effect is as though they are passed by value.

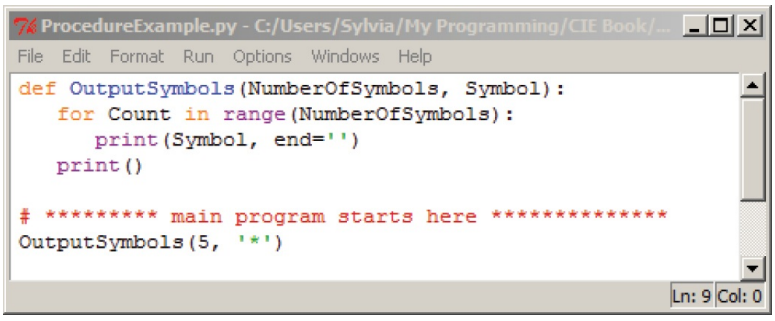


Figure 14.15 Parameters passed to a Python subroutine

In VB.NET (see Figure 14.16), parameters default to passing by value. The keyword ByVal is automatically inserted by the editor.

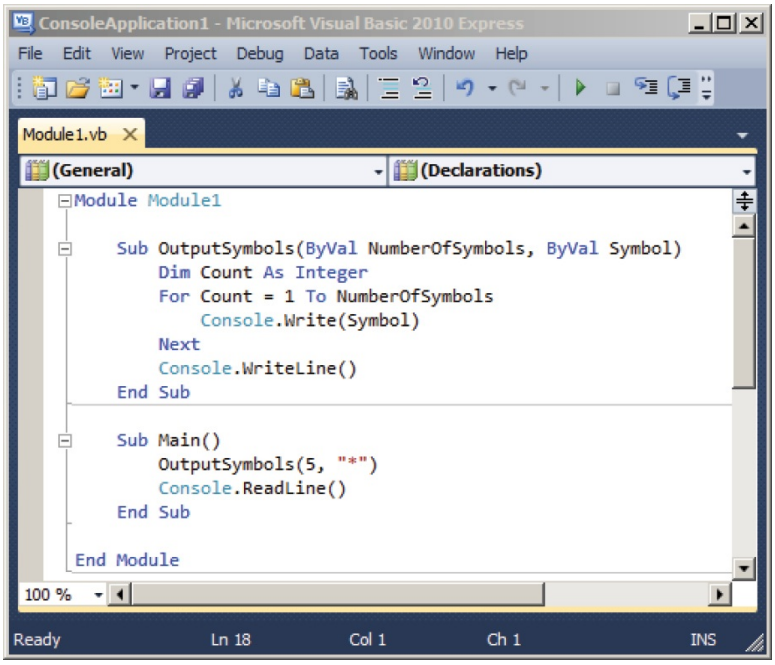


Figure 14.16 Parameters passed by value to a VB.NET procedure

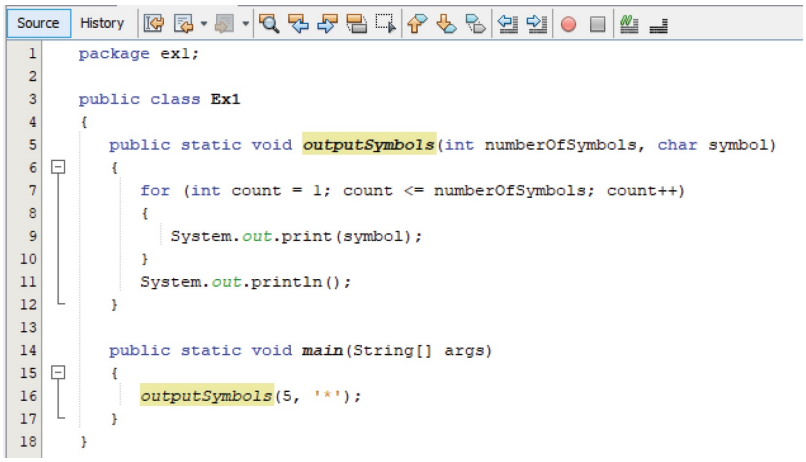


Figure 14.17 Parameters passed by value to a Java procedure

In Java (see Figure 14.17), all parameters behave like local variables and their effect is as though they are passed by value.

Passing parameters by reference

When parameters are passed by reference, when the values inside the subroutine change, this affects the values of the variables in the calling program.

Consider the pseudocode procedure AdjustValuesForNextRow below.

The pseudocode for the pyramid example generated in Chapter 12 (Section 12.09) includes a procedure definition that uses two parameters passed by reference. We can now make that explicit:

```
PROCEDURE AdjustValuesForNextRow(BYREF Spaces : INTEGER, Symbols : INTEGER)
    Spaces ← Spaces - 1
    Symbols ← Symbols + 2
ENDPROCEDURE
```

The pseudocode statement to call the procedure is:

```
CALL AdjustValuesForNextRow(NumberOfSpaces, NumberOfSymbols)
```

The values of the parameters `Spaces` and `Symbols` are changed within the procedure when this is called. The variables NumberOfSpaces and `NumberOfSymbols` in the program code after the call will store the updated values that were passed back from the procedure.

Python does not have a facility to pass parameters by reference. Instead the subroutine behaves as a

function and returns multiple values (see Figure 14.18). Note the order of the variables as they receive these values in the main part of the program.

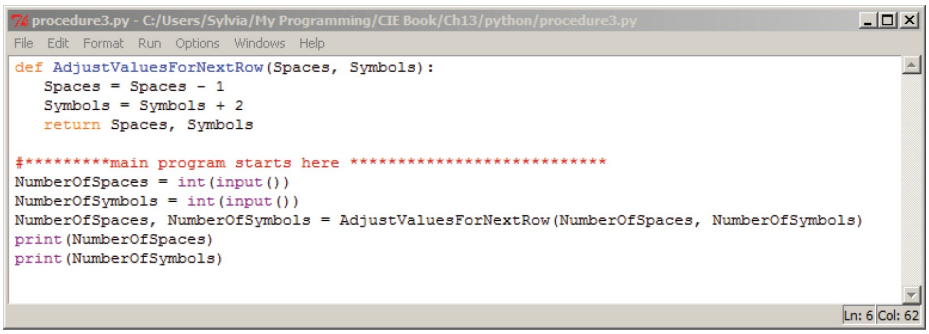


Figure 14.18 Multiple values returned from a Python subroutine

This way of treating a multiple of values as a unit is called a 'tuple' (see Chapter 11, Section 11.02). You can find out more by reading the Python help files.

In VB.NET (see Figure 14.19), the ByRef keyword is placed in front of each parameter to be passed by reference.

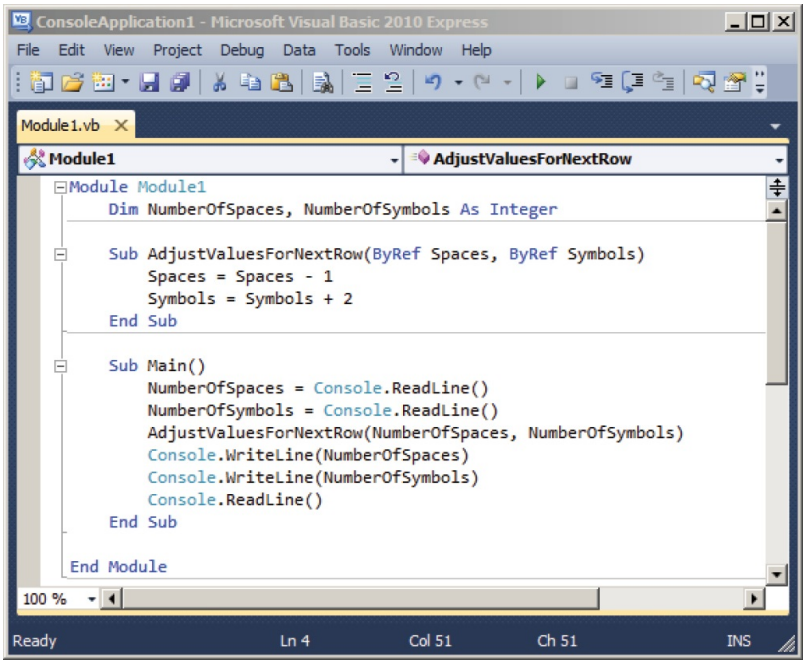


Figure 14.19 Parameters passed by reference to a VB.NET procedure

Java does not have a facility to pass simple variable parameters by reference. Only objects can be passed by reference. Arrays are objects in Java, so these are passed by reference.

TIP

If only one value needs to be returned, the subroutine can be written as a function. If more than one value needs to be returned, a work-around is to declare a class and return it as an object (Figure 14.20). If the values to be returned are of the same type, they can be grouped into an array and the array is passed as a reference parameter.

TIP

A preferable solution is to amend the algorithm and write several functions (Figure 14.21).

```java
package ex1;
import java.util.Scanner;

public class Ex1
{
   static class RowData
   {
      public int spaces = 0;
      public int symbols = 0;
   }

   public static void adjustValuesForNextRow(RowData thisRow)
   {
      thisRow.spaces--;
      thisRow.symbols = thisRow.symbols + 2;
   }

   public static void main(String[] args)
   {
      Scanner console = new Scanner(System.in);
      RowData thisRow = new RowData();
      System.out.print("Enter number of spaces: ");
      thisRow.spaces = console.nextInt();
      System.out.print("Enter number of symbols: ");
      thisRow.symbols = console.nextInt();
      adjustValuesForNextRow(thisRow);
      System.out.println(thisRow.spaces);
      System.out.println(thisRow.symbols);
   }
}
```

Figure 14.20 Multiple values returned as an object from a Java subroutine

```java
package ex1;
import java.util.Scanner;

public class Ex1
{
   public static int adjustSpacesForNextRow(int spaces)
   {
      spaces--;
      return spaces;
   }

   public static int adjustSymbolsForNextRow(int symbols)
   {
      symbols = symbols + 2;
      return symbols;
   }

   public static void main(String[] args)
   {
      Scanner console = new Scanner(System.in);
      System.out.print("Enter number of spaces: ");
      int spaces = console.nextInt();
      System.out.print("Enter number of symbols: ");
      int symbols = console.nextInt();
      spaces = adjustSpacesForNextRow(spaces);
      symbols = adjustSymbolsForNextRow(symbols);
      System.out.println(spaces);
      System.out.println(symbols);
   }
}
```

Figure 14.21 The algorithm changed into two subroutines for Java

14.13 Putting it all together

The programs in this section are full solutions to the pyramid-drawing program developed in Section 14.12.

The parameters of the subroutines have different identifiers from the variables in the main program. This is done deliberately, so that it is quite clear that the parameters and local variables within a subroutine are separate from those in the calling program or module. If a parameter is passed by reference to a procedure, the parameter identifier within the procedure references the same memory location as the variable identifier passed to the procedure as argument.

The pyramid-drawing program in Python, VB.NET and Java

Python

```
SPACE = ' ' # constant to give a space a name
def InputMaxNumberOfSymbols():
   Number = 0
   while Number % 2 == 0:
      print("How many symbols make the base? ")
      Number = int(input("Input an odd number: "))
   return Number

def SetValues():
   Symbol = input("What symbol do you want to use? ")
   MaxSymbols = InputMaxNumberOfSymbols()
   Spaces = (MaxSymbols + 1) // 2
   Symbols = 1
   return Symbol, MaxSymbols, Spaces, Symbols

def OutputChars(Number, Symbol):
   for Count in range (Number):
      print(Symbol, end='')

def AdjustValuesForNextRow(Spaces, Symbols):
   Spaces = Spaces - 1
   Symbols = Symbols + 2
   return Spaces, Symbols

def main():
   ThisSymbol, MaxNumberOfSymbols, NumberOfSpaces, NumberOfSymbols = SetValues()
   while NumberOfSymbols <= MaxNumberOfSymbols:
      OutputChars(NumberOfSpaces, SPACE)
      OutputChars(NumberOfSymbols, ThisSymbol)
      print()  # move to new line
      NumberOfSpaces, NumberOfSymbols = AdjustValuesForNextRow(NumberOfSpaces, NumberOfSymbols)
main()
```

VB.NET

```
Module Module1
   Const Space = " " 'constant to give a space a name
   Dim NumberOfSpaces, NumberOfSymbols As Integer
   Dim MaxNumberOfSymbols As Integer
   Dim ThisSymbol As Char

Sub InputMaxNumberOfSymbols(ByRef Number As Integer)
   Do
      Console.WriteLine("How many symbols make the base? ")
      Console.Write("Input an odd number: ")
      Number = Console.ReadLine()
   Loop Until (Number Mod 2 = 1)
End Sub

Sub SetValues(ByRef Symbol, ByRef MaxSymbols, ByRef Spaces, ByRef Symbols)
   Console.Write("What symbol do you want to use? ")
   Symbol = Console.ReadLine()
   InputMaxNumberOfSymbols(MaxSymbols)
   Spaces = (MaxSymbols + 1) \ 2
   Symbols = 1
End Sub
```

```
Sub OutputChars(ByVal Number, ByVal Symbol)
   Dim Count As Integer
   For Count = 1 To Number
      Console.Write(Symbol)
   Next
End Sub

Sub AdjustValuesForNextRow(ByRef Spaces, ByRef Symbols)
   Spaces = Spaces - 1
   Symbols = Symbols + 2
End Sub

Sub Main()
   SetValues(ThisSymbol, MaxNumberOfSymbols, NumberOfSpaces, NumberOfSymbols)
   Do
      OutputChars(NumberOfSpaces, Space)
      OutputChars(NumberOfSymbols, ThisSymbol)
      Console.WriteLine()'move to new line
      AdjustValuesForNextRow(NumberOfSpaces, NumberOfSymbols)
   Loop Until NumberOfSymbols > MaxNumberOfSymbols
   Console.ReadLine()
End Sub

End Module
```

Java

```
package ex1;
import java.util.Scanner;

public class Ex1
{
   static final char SPACE = ' ';

   public static char getSymbol()
   {
      Scanner console = new Scanner(System.in);
      System.out.print("What symbol do you want to use? ");
      String response = console.next();
      return response.charAt(0);
   }
   public static int inputMaxNumberOfSymbols()
   {
      Scanner console = new Scanner(System.in);
      int number = 0;
      while ((number % 2) == 0)
      {
         System.out.print("How many symbols make the base? ");
         number = console.nextInt();
      }
      return number;
   }

   public static void outputChars(int number, char symbol)
   {
      for (int count = 0; count < number; count++)
      {
         System.out.print(symbol);
      }
   }
   public static int adjustSpacesForNextRow(int spaces)
   {
      spaces--;
      return spaces;
   }
   public static int adjustSymbolsForNextRow(int symbols)
   {
      symbols = symbols + 2;
      return symbols;
   }
   public static void main(String[] args)
   {
      char thisSymbol = getSymbol();
```

```
        int maxNumberOfSymbols = inputMaxNumberOfSymbols();
        int numberOfSpaces = (maxNumberOfSymbols + 1)/ 2;
        int numberOfSymbols = 1;

        while (numberOfSymbols <= maxNumberOfSymbols)
        {
           outputChars(numberOfSpaces, SPACE);
           outputChars(numberOfSymbols, thisSymbol);
           System.out.println();
           numberOfSpaces = adjustSpacesForNextRow(numberOfSpaces);
           numberOfSymbols = adjustSymbolsForNextRow(numberOfSymbols);
        }
     }
}
```

Discussion Point:

Can you see how the two procedures OutputSpaces and OutputSymbols have been replaced by a single procedure OutputChars without changing the effect of the program?

14.14 Arrays

Creating 1D arrays

VB.NET, Python and Java number array elements from 0 (the lower bound).

In pseudocode, a 1D array declaration is written as:

```
DECLARE <arrayIdentifier> : ARRAY[<lowerBound>:<upperBound>] OF <dataType>
```

Syntax definitions

Python	In Python, there are no arrays. The equivalent data structure is called a list. A list is an ordered sequence of items that do not have to be of the same data type.
VB.NET	`Dim <arrayIdentifier>(<upperBound>) As <dataType>`
Java	`<datatype>[] <arrayIdentifier>;` `<arrayIdentifier> = new int[<upperbound>+1];`

Pseudocode example:

```
DECLARE List1 : ARRAY[1:3]   OF STRING  // 3 elements in this list
DECLARE List2 : ARRAY[0:5]   OF INTEGER // 6 elements in this list
DECLARE List3 : ARRAY[1:100] OF INTEGER // 100 elements in this list
DECLARE List4 : ARRAY[0:25] OF STRING  // 26 elements in this list
```

Code examples

Python	`List1 = []`
	`List1.append("Fred")`
	`List1.append("Jack")`
	`List1.append("Ali")`
		As there are no declarations, the only way to generate a list is to initialise one. You can append elements to an existing list.

	`List2 = [0, 0, 0, 0, 0, 0]`	You can enclose the elements in `[ ]`.
	`List3 = [0 for i in range(100)]`	You can use a loop.
	`AList = [""] * 26`	You can provide an initial value, multiplied by number of elements required.

VB.NET	`Dim List1 As String () = {"","",""}`
	`Dim List2(5) As Integer`
	`Dim List3(100) As Integer`
	`Dim AList(0 To 25) As String`
		You can initialise an array at declaration time (as with `List1`). Note that `List3` has 101 elements. You can use a range as an array dimension (as with `AList`) however the lower bound must be 0.

Java	`String[] list1 = {"","",""};`
	`int[] list2;`
	`list2 = new int[5];`
	`int[] list3;`
	`list3 = new int[100];`
	`String[] aList;`
	`aList = new String[25];`
		You can initialise an array at declaration time (as with `list1`).

Accessing 1D arrays

A specific element in an array is accessed using an index value. In pseudocode, this is written as:

```
<arrayIdentifier>[x]
```

Pseudocode example:

```
NList[25] = 0  // set 25th element to zero
AList[3] = "D" // set 3rd element to letter D
```

Code examples

Python	`NList[24] = 0` `AList[3] = "D"`
VB.NET	`NList(25) = 0` `AList(3) = "D"`
Java	`nList[25] = 0;` `aList[3] = "D";`

In Python, you can print the whole contents of a list using `print(List)`. In VB.NET and Java, you need to use a loop to print one element of an array at a time.

TIP

When writing a solution using pseudocode, always use a loop to print the contents of an array.

TASK 14.13

1. Write program code to implement the pseudocode from Worked Example 13.01 in Chapter 13.
2. Write program code to implement the pseudocode from Worked Example 13.02 in Chapter 13.
3. Write program code to implement the improved algorithm from Worked Example 13.03 in Chapter 13.

Creating 2D arrays

In pseudocode, a 2D array declaration is written as:

```
DECLARE <identifier> : ARRAY[<lBound1>:<uBound1>,
<lBound2>:<uBound2>] OF <dataType>
```

Syntax definitions

Python	In Python, there are no arrays. The equivalent data structure is a list of lists.
VB.NET	`Dim <arrayIdentifier>(<uBound1, uBound2>) As <dataType>`
Java	`<dataType> <arrayIdentifier>;` `<arrayIdentifier> = new <datatype>[<uBound1>][<uBound2>];`

To declare a 2D array to represent a game board of six rows and seven columns, the pseudocode statement is:

```
DECLARE Board : ARRAY[1:6, 1:7] OF INTEGER
```

Code examples

Python	`Board = [[0, 0, 0, 0, 0, 0, 0],`
	`[0, 0, 0, 0, 0, 0, 0],`
	`[0, 0, 0, 0, 0, 0, 0],`
	`[0, 0, 0, 0, 0, 0, 0],`
	`[0, 0, 0, 0, 0, 0, 0],`
	`[0, 0, 0, 0, 0, 0, 0]]`
	`Board = [[0 for i in range(7)]`
	`for j in range(6)]`
	`Board = [[0] * 7] * 6`
		2D lists can be initialised in a similar way to 1D lists. Remember that elements are numbered from 0.
		These are alternative ways of initialising a 6 × 7 list. The rows are numbered 0 to 5 and the columns 0 to 6.
		The upper value of the range is not included.

VB.NET	`Dim Board(6, 7) As Integer`	Elements are numbered from 0 to the given number. This declaration has one row and one column too many. However, the algorithm may be such that it is easier to convert to program code if row 0 and column 0 are ignored.
Java	`int[][] board = {` `{0, 0, 0, 0, 0, 0, 0},`	2D arrays can be initialised in a similar way to 1D arrays. Remember that elements are numbered from 0.

	{0, 0, 0, 0, 0, 0, 0},
	{0, 0, 0, 0, 0, 0, 0},
	{0, 0, 0, 0, 0, 0, 0},
	{0, 0, 0, 0, 0, 0, 0},
	{0, 0, 0, 0, 0, 0, 0}
	}
	int[][] board;
	board = new int[6][7];	

Accessing 2D arrays

A specific element in a table is accessed using an index pair. In pseudocode this is written as:

```
<arrayIdentifier>[x, y]
```

Pseudocode example:

```
Board[3,4] ← 0 // sets the element in row 3 and column 4 to zero
```

The following code examples demonstrate how to access elements in each of the three languages.

Code examples

Python	`Board[2][3] = 0`	Elements are numbered from 0 in Python, so [3] gives access to the fourth element.
VB.NET	`Board(3, 4) = 0`	We are ignoring row 0 and column 0.
Java	`board[2][3] = 0;`	Elements are numbered from 0 in Java, so [3] gives access to the fourth element.

TASK 14.14

1. Write program code to implement the pseudocode from Worked Example 13.04 in Chapter 13; first initialise the table and then output its contents.
2. Write program code to implement the pseudocode from Worked Example 13.05 in Chapter 13.

14.15 Text files

Writing to a text file

The following pseudocode statements provide facilities for writing to a file:

```
OPENFILE <filename> FOR WRITE       // open the file for writing
WRITEFILE <filename>, <stringValue> // write a line of text to the file
CLOSEFILE <filename>                // close file
```

The following code examples demonstrate how to open, write to and close a file called `SampleFile.TXT` in each of the three languages. If the file already exists, it is overwritten as soon as the file handle is assigned by the 'open file' command.

Code examples

<table>
<tr><td>Python</td><td><pre>FileHandle = open("SampleFile.TXT", "w")
FileHandle.write(LineOfText)
FileHandle.close()</pre></td><td>You specify the filename and mode ('w' for write) when you call the <code>open</code> function. The line of text to be written to the file must contain the newline character "\n" to move to the next line of the text file.</td></tr>
<tr><td rowspan="2">VB.NET</td><td><pre>Dim FileHandle As IO.StreamWriter
Dim LineOfText As String
FileHandle = New
IO.StreamWriter("SampleFile.TXT")
FileHandle.WriteLine(LineOfText)
FileHandle.Close()</pre></td><td>The file is accessed through an object (see Chapter 27) called a <code>StreamWriter</code>.</td></tr>
<tr><td colspan="2">Alternative method:
<pre>Dim LineOfText As String
Dim Channel As Integer = 1
FileSystem.FileOpen(Channel, "SampleFile.TXT", OpenMode.Output, OpenAccess.Write)
FileSystem.PrintLine(Channel, LineOfText)
FileSystem.FileClose(Channel)</pre></td></tr>
<tr><td>Java</td><td><pre>import java.io.FileWriter;
import java.io.PrintWriter;
import java.io.IOException;
FileWriter fileHandle = new
FileWriter("SampleFile.TXT", false);
PrintWriter printLine = new PrintWriter(fileHandle);
String lineOfText;
printLine.printf("%s"+"%n", lineOfText);
printLine.close();</pre></td><td>Input output operations throw exceptions. The easiest way to manage these is to change your main heading to:
<pre>public static void main(String[] args)
throws IOException</pre></td></tr>
</table>

Reading from a text file

An existing file can be read by a program. The following pseudocode statements provide facilities for reading from a file:

```
OPENFILE <filename> FOR READ          // open file for reading
READFILE <filename>, <stringVariable>  // read a line of text from the file
CLOSEFILE <filename>                  // close file
```

The following code examples demonstrate how to open, read from and close a file called `SampleFile.TXT` in each of the three languages.

Code examples

<table>
<tr><td>Python</td><td><pre>FileHandle = open("SampleFile.TXT", "r")
LineOfText = FileHandle.readline()
FileHandle.close ()</pre></td><td>You specify the filename and mode ('r' for read) when you call the <code>open</code> function.</td></tr>
<tr><td>VB.NET</td><td><pre>Dim LineOfText As String
Dim FileHandle As IO.StreamReader
FileHandle = New IO.StreamReader("SampleFile.TXT")</pre></td><td>The file is accessed through an object (see Chapter 27) called a <code>StreamReader</code>.</td></tr>
</table>

	`LineOfText = FileHandle.ReadLine()` `FileHandle.Close()`	

	Alternative method:
	`Dim LineOfText As String`
	`Dim Channel As Integer = 1`
	`FileSystem.FileOpen(Channel, "SampleFile.TXT", OpenMode.Input, OpenAccess.Read)`
	`FileSystem.Input(Channel, LineOfText)`
	`FileSystem.FileClose(Channel)`	

Java	`import java.io.IOException;`
	`import java.io.FileReader;`
	`import java.io.BufferedReader;`
	`FileReader fileHandle = new`
	`FileReader("SampleFile.TXT");`
	`BufferedReader textReader = new`
	`BufferedReader(fileHandle);`
	`String lineOfText = textReader.readLine();`
	`textReader.close();`
		There are other library classes that can be used for input/output, such as Scanner.

Appending to a text file

The following pseudocode statements provide facilities for appending to a file:

```
OPENFILE <filename> FOR APPEND      // open file for append
WRITEFILE <filename>, <stringValue> // write a line of text to the file
CLOSEFILE <filename>                // close file
```

The following code examples demonstrate how to open, append to and close a file called `SampleFile.TXT` in each of the three languages.

Code examples

Python	`FileHandle = open("SampleFile.TXT", "a")`
	`FileHandle.write(LineOfText)`
	`FileHandle.close()`
		You specify the filename and mode ('a' for append) when you call the `open` function.

VB.NET	`Dim FileHandle As IO.StreamWriter`
	`FileHandle = New`
	`IO.StreamWriter("SampleFile.TXT", True)`
	`FileHandle.WriteLine(LineOfText)`
	`FileHandle.Close()`
		The file is accessed through a `StreamWriter`. The extra parameter, `True`, tells the system to append to the object.

	Alternative method:
	`Dim LineOfText As String`
	`Dim Channel As Integer = 1`
	`FileSystem.FileOpen(Channel, "SampleFile.TXT", OpenMode.Append, OpenAccess.ReadWrite)`
	`FileSystem.Print(Channel, LineOfText)`
	`FileSystem.FileClose(Channel)`	

Java	`import java.io.FileWriter;`
	`import java.io.PrintWriter;`
	`import java.io.IOException;`
	`FileWriter fileHandle = new`
	`FileWriter("SampleFile.TXT"), true);`
	`PrintWriter printLine = new PrintWriter(fileHandle);`
	`String lineOfText;`
	`printLine.printf("%s"+"%n", lineOfText);`
	`printLine.close();`
		Input output throws exceptions. The easiest way is to change your main heading to:
		`public static void main(String[] args)`
		`throws IOException`

The end-of-file (EOF) marker

The following pseudocode statements read a text file and output its contents:

```
OPENFILE "Test.txt" FOR READ
WHILE NOT EOF("Test.txt") DO
    READFILE "Test.txt", TextString
    OUTPUT TextString
ENDWHILE
CLOSEFILE "Test.txt"
```

The following code examples demonstrate how to read and then output the contents of a file in each of the three languages.

Code examples

Python	`FileHandle = open("Test.txt", "r")`
	`LineOfText = FileHandle.readline()`
	`while len(LineOfText) > 0:`
	`  LineOfText = FileHandle.readline()`
	`  print(LineOfText)`
	`FileHandle.close()`
		There is no explicit EOF function. However, when a line of text has been read that only consists of the end-of-file marker, the line of text is of length 0.

VB.NET	`Dim LineOfText As String`
	`Dim FileHandle As System.IO.StreamReader`
	`FileHandle = New`
	`System.IO.StreamReader("Test.txt")`
	`Do Until FileHandle.EndOfStream`
	`  LineOfText = FileHandle.ReadLine()`
	`  Console.WriteLine(LineOfText)`
	`Loop`
	`FileHandle.Close()`
		When the end-of-file marker is detected, the `EndOfStream` method returns the value True and so the loop will end.

VB.NET	Alternative method:
	`Dim LineOfText As String`
	`Dim Channel As Integer = 1`
	`FileSystem.FileOpen(Channel, "Test.txt",`
	`OpenMode.Input, OpenAccess.Read)`
	`Do While Not FileSystem.EOF(Channel)`
	`  FileSystem.Input(Channel, LineOfText)`
	`  Console.WriteLine(LineOfText)`
	`Loop`
	`FileSystem.FileClose(Channel)`	

Java	`import java.io.IOException;`
	`import java.io.FileReader;`
	`import java.io.BufferedReader;`
	`FileReader fileHandle = new FileReader("Test.txt");`
	`BufferedReader textReader = new`
	`BufferedReader(fileHandle);`
	`String lineOfText = textReader.readLine();`
	`while (lineOfText != null)`
	`{`
	`  System.out.println(lineOfText);`
	`  lineOfText = textReader.readLine();`
	`}`
	`textReader.close();`
		There is no explicit EOF function. However, when a line of text has been read that only consists of the end-of-file marker, the line of text is effectively null.

TASK 14.15

Fred surveys the students at his college to find out their favourite hobby. He wants to present the data as a tally chart.

Fred plans to enter the data into the computer as he surveys the students. After data entry is complete, he wants to output the total for each hobby.

1	Reading books	\\\
2	Play computer games	\\\\\\\
3	Sport	\\\\\
4	Programming	\\
5	Watching TV	\\\\\\\\\\

He starts by writing an algorithm:

```
Initialise Tally array
REPEAT
```

```
    INPUT Choice // 1 for Reading, 2 for computer games,
                 // 3 for Sport, 4 for Programming, 5 for TV
                 // 0 to end input
    Increment Tally[Choice]
UNTIL Choice = 0
FOR Index = 1 TO 5
    OUTPUT Tally[Index]
NEXT Index
```

1. Write program code to declare and initialise the array `Tally : ARRAY[1:5] OF INTEGER`.
2. Write program code to implement the algorithm above.
3. Write program code to declare an array to store the hobby titles and rewrite the `FOR` loop of your program in part 2 so that the hobby title is output before each tally.
4. Write program code to save the array data in a text file.
5. Write program code to read the data from the text file back into the initialised array.

Reflection Point:

How much practice have you had writing programs? Did you get them to work?

How difficult did you find the different constructs?

Put the following in order of difficulty:

- Using a FOR loop
- Using a WHILE loop
- Using an IF ELSE statement
- Declaring a variable of a standard data type
- Declaring and using a constant
- Using a 1D array
- Using a nested loop to access each element in a 2D array
- Reading from and writing to a text file
- Using a built-in function
- Writing a procedure and calling it from the main program
- Writing a function and using its return value in an expression in the main program
- Using parameters with procedures and functions

Summary

- Programming constructs in Python, VB.NET and Java include:
  - declaration and assignment of constants and variables
  - the basic constructs of assignment, selection, repetition, input and output
  - built-in data types and functions.
- Code should be commented where it helps understanding.
- Boolean expressions are needed for conditions.
- Declaration of subroutines (functions and procedures) is done before the main program body.
- Calling a procedure is a program statement.

- Calling a function is done within an expression, for example an assignment.
- VB.NET and Java functions return exactly one value.
- Parameters can be passed to a subroutine. This is known as the interface.
- VB.NET passes parameters by value, as a default, but can return one or more values via parameters if they are declared as reference parameters.
- In Python, parameters can only pass values into a subroutine. The only way to update a value of a variable in the calling program is to return one or more values from a function.
- In Java, parameters can only pass values into a subroutine. The only way to update a value of a variable in the calling program is to return one value from a function. Note that object parameters are always passed by reference.
- When a subroutine is defined, parameters are the ‘placeholders’ for values passed into a subroutine.
- Arguments are the values passed to the subroutine when it is called.

Exam-style Questions

1 Preeti wants a program to output a conversion table for ounces to grams (1 ounce is 28.35 grams). She writes an algorithm using Structured English:

```
OUTPUT "Ounces Grams"
FOR Ounces FROM 1 TO 30
    SET Grams TO Rounded(Ounces * 28.35) // whole number of grams only
    OUTPUT Ounces and Grams
```

Write pseudocode to implement the algorithm. Include formatting, so that the output is tabulated. [7]

2 Write pseudocode to accept an input string `UserID`. The pseudocode is to test the `UserID` format. A valid format `UserID` consists of three upper case letters and four digits. The program is to output a message whether `UserID` is valid or not. [5]

3 Write pseudocode for a procedure `OutputTimesTable` that takes one integer parameter, `n`, and outputs the times table for `n`. For example the procedure call `OutputTimesTable(5)` should produce:

1 × 5 = 5
2 × 5 = 10
3 × 5 = 15
4 × 5 = 20
5 × 5 = 25
6 × 5 = 30
7 × 5 = 35
8 × 5 = 40
9 × 5 = 45
10 × 5 = 50 [6]

4 Write pseudocode for a function `isDivisible()` that takes two integer parameters, `x` and `y`. The function is to return the value True or False to indicate whether `x` is exactly divisible by `y`. For example, `isDivisible(24, 6)` should return True and `isDivisible(24, 7)` should return False. [6]

5 A poultry farm packs eggs into egg boxes. Each box takes six eggs. Boxes must not contain fewer than six eggs.

Write pseudocode for a procedure `EggsIntoBoxes` that takes an integer parameter, `NumberOfEggs`. The procedure is to calculate how many egg boxes can be filled with the given number of eggs and how many eggs will be left over. The procedure is to return two values as parameters, `NumberOfBoxes` and `EggsLeftOver`. [9]

6 In a certain country, car registrations consist 7 alphanumerical characters. The format of a car registration is either

LLNNLLL

or

LLLNNLL

where L is any capital letter and N is any numeral 0 to 9.

Use pseudocode to write a function that takes a string as parameter and returns TRUE if the format is valid and FALSE otherwise.

The string-handling functions available are those listed in Table 14.06. [7]