# Cambridge

## OL- IGCSE

# Computer science

CODE: (0478)

## Chapter 08

# **Programming**

There are many high-level programming languages to choose from. For IGCSE Computer Science the high-level programming languages recommended are Python, Visual Basic or Java. The use of one of these languages is also required for A Level Computer Science.

Many programming languages need an interactive development environment (IDE) to write and run code. IDEs are free to download and use.

Programs developed in this chapter will be illustrated using the following freely available languages:

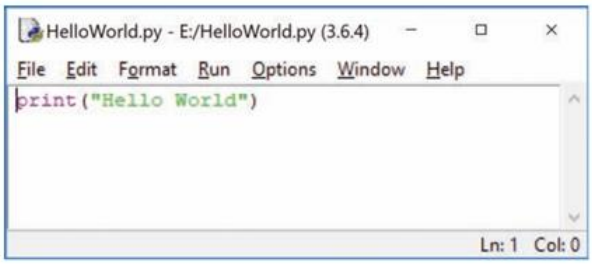
» Python a general purpose, open-source programming language that promotes rapid program development and readable code. The IDE used for screenshots in this chapter is called IDLE.

» Visual Basic, a widely used programming language for Windows. The IDE used for screenshots in this chapter is called Visual Studio.

» Java, a commercial language used by many developers. The IDE used for screenshots in this chapter is called BlueJ.

The traditional introduction to programming in any language is to display the words 'Hello World' on a computer screen. The programs look very different, but the output is the same:

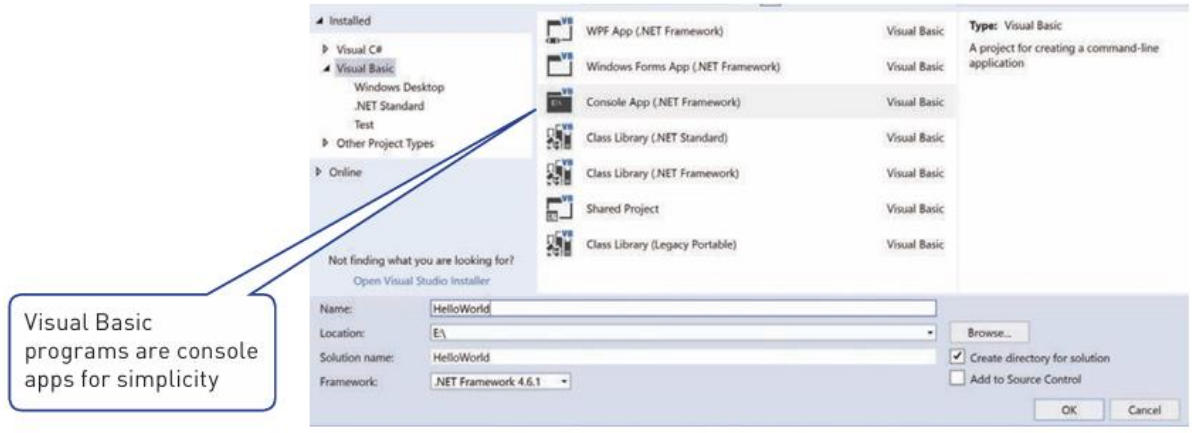
**Python**



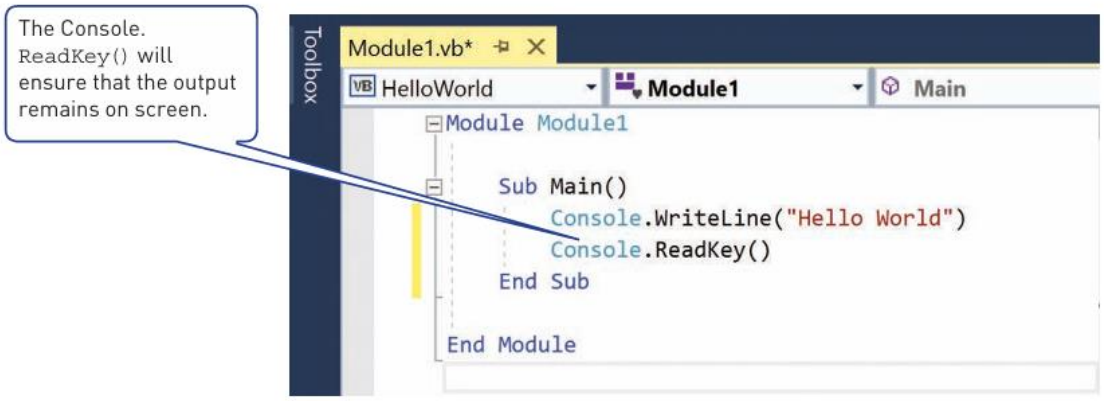

▲ **Figure 8.1** The editing window for Python


▲ **Figure 8.2** The runtime window for Python

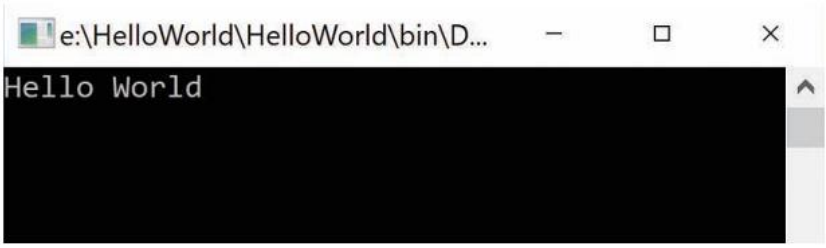
**Visual Basic**




▲ **Figure 8.3** The Visual Basic program will run in a command line window

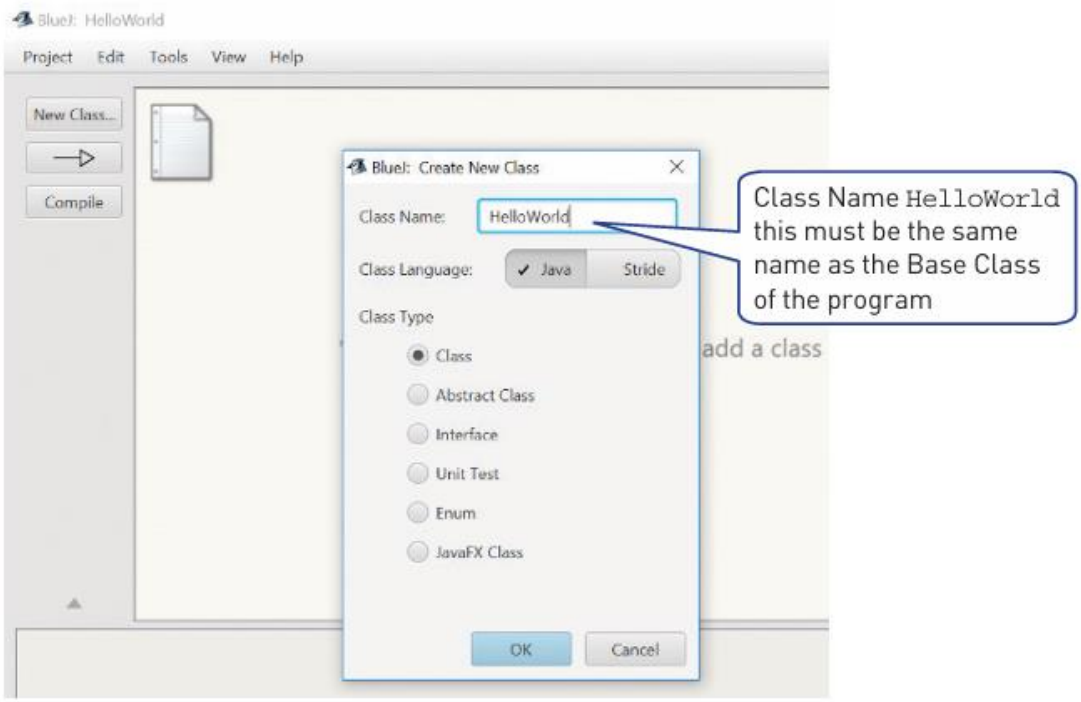



▲ **Figure 8.4** The editing window for Visual Basic




▲ **Figure 8.5** The runtime window for Visual Basic

## Java




▲ **Figure 8.6** Setting up the class for Java


▲ **Figure 8.7** The project window for Java


▲ **Figure 8.8** The editing window for Java


▲ **Figure 8.9** The terminal window to show the output from the program Java

# 8.1 Programming concepts

There are five basic constructs to use and understand when developing a program: » data use – variables, constants and arrays

» Sequence – order of steps in a task
» Selection – choosing a path through a program
» Iteration – repetition of a sequence of steps in a program
» Operators use – arithmetic for calculations, logical and Boolean for decisions.

## 8.1.1 Variables and constants

In a program, any data used can have a fixed value that does not change while the program is running.

A **variable** in a computer program is a named data store than contains a value that may change during the execution of a program. In order to make programs understandable to others, variables should be given meaningful names.

A **constant** in a computer program is a named data store than contains a value that does not change during the execution of a program.

Not all programming languages explicitly differentiate between constants and variables, but programmers should be clear which data stores can be changed and which cannot be changed. There are several ways of highlighting a constant, for example:

| | |
|---|---|
| Use of capital letters | `PI = 3.142` |
| Meaningful names that begin with Const | `ConstPi = 3.142` |

It is considered good practice to **declare** the constants and variables to be used in that program. Some languages require explicit declarations, which specifically state what type of data the variable or constant will hold.

Here are some sample declaration statements in pseudocode and programming code – just look at the sections for pseudocode and the programming language you are using.

▼ **Table 8.1** How to declare variables and constants

| Declarations of constants and variables | Language |
|---|---|
| `DECLARE FirstVar : INTEGER`<br>`DECLARE SecondVar : INTEGER`<br>`CONSTANT FirstConst ← 500`<br>`CONSTANT SecondConst ← 100` | Pseudocode declarations, variables are explicit but constants are implicit. |
| `FirstVar = 20`<br>`SecondVar = 30`<br>`FIRSTCONST = 500`<br>`SECONDCONST = 100`<br>or<br>`FirstVar, SecondVar = 20, 30`<br>`FirstConst, SecondConst = 500,100` | Python does not require any separate declarations and does not differentiate between constants and variables.<br>Programmers need to keep track of and manage these differences instead. |
| `Dim FirstVar As Integer`<br>`Dim SecondVar As Integer`<br>`Const FirstConst As Integer = 500`<br>`Const SecondConst As Integer = 500`<br>or<br>`Dim FirstVar, SecondVar As Integer`<br>`Const First, Second As Integer = 500, 100` | In Visual Basic variables are explicitly declared as particular data types before use. Declarations can be single statements or multiple declarations in a single statement. Constants can be explicitly typed as shown or implicitly typed, for example:<br>`Const First = 500`<br>... which implicitly defines the constant as an integer. |
| `int FirstVar;`<br>`int SecondVar;`<br>`final int FIRSTCONST = 500;`<br>`final int SECONDCONST = 100;` | In Java constant values are declared as variables with a final value so no changes can be made. These final variable names are usually capitalised to show they cannot be changed.<br>Variables are often declared as they are used rather than at the start of the code. |

### 8.1.2 Basic data types

In order for a computer system to process and store data effectively, different kinds of data are formally given different types. This enables:

» Data to be stored in an appropriate way, for example, as numbers or as characters

» Data to be manipulated effectively,

» Automatic validation in some cases. The basic data types you will need to use for IGCSE Computer Science are:

» Integer – a positive or negative whole number that can be used with mathematical operators

» Real – a positive or negative number with a fractional part. Real numbers can be used with mathematical operators

» Char – a variable or constant that is a single character

» String – a variable or constant that is several characters in length. Strings vary in length and may even have no characters (known as an empty string); the characters can be letters and/or digits and/or any other printable symbol (If a number is stored as a string, then it cannot be used in calculations.)

» Boolean – a variable or constant that can have only two values TRUE or FALSE.

**Table 8.2** Examples of data types

| Pseudocode | Python | Visual Basic | Java |
|---|---|---|---|
| INTEGER | FirstInteger = 25 | Dim FirstInt As Integer | int FirstInt; or<br>byte FirstInt; |
| REAL | FirstReal = 25.0 | Dim FirstReal As Decimal | float FirstReal; or<br>double FirstReal; |
| CHAR | Female = 'F' or<br>Female = "F" | Dim Female As Char | char Female; |
| STRING | FirstName = 'Emma' or<br>FirstName = "Emma" | Dim FirstName As String | String FirstName; |
| BOOLEAN | Flag = True | Dim Flag As Boolean | boolean Flag; |

## 8.1.3 Input and output

For a program to be useful, the user needs to know what they are expected to input, so each input needs to be accompanied by a prompt stating the input required.

In a programming language the data type of the input must match the required data type of the variable where the input data is to be stored. All inputs default as strings, so if the input should be an integer or real number, commands are also used to change the data type of the input (for instance, in Python these are int() or float()).

**Table 8.3** Examples of input statements with prompts

| Input statements | Language |
|---|---|
| radius = float(input("Please enter the radius of the cylinder ")) | Python combines the prompt with the input statement. |
| Console.Write("Please enter the radius of the cylinder ")<br>radius = Decimal.Parse(Console.ReadLine()) | Visual Basic uses a separate prompt and input. The input specifies the type of data expected. |
| import java.util.Scanner;<br>Scanner myObj = new Scanner(System.in);<br>System.out.println("Please enter the radius of the cylinder ");<br>double radius = myObj.nextDouble(); | In Java the input library has to be imported at the start of the program and an input object is set up. Java uses a separate prompt and input. The input specifies the type and declares the variable of the same type. |

For a program to be useful, the user needs to know what results are being output, so each output needs to be accompanied by a **message** explaining the result. If there are several parts to an output statement, then each part is separated by a separator character.

**Table 8.4** Examples of output statements with messages

| Output the results | Language |
|---|---|
| `print("Volume of the cylinder is ", volume)` | Python uses a comma |
| `Console.WriteLine("Volume of the cylinder is " & volume)` | VB uses & |
| `System.out.println("Volume of the cylinder is " + volume);` | Java uses + |

Examples of the complete programs are shown below:

**Python**

```
CONSTPI = 3.142
Radius = float(input("Please enter the radius of the cylinder "))
Length = float(input("Please enter the length of the cylinder "))
Volume = Radius * Radius * Length * CONSTPI
print("Volume of the cylinder is ", Volume)
```

**Visual Basic**

Every console program in VB must contain a Main module. These statements are shown in red.

```
Module Module1
    Public Sub Main()
        Dim Radius As Decimal
        Dim Length As Decimal
        Dim Volume As Decimal
        Const PI As Decimal = 3.142
        Console.Write("Please enter the radius of the cylinder ")
        Radius = Decimal.Parse(Console.ReadLine())
        Console.Write("Please enter the length of the cylinder ")
        Length = Decimal.Parse(Console.ReadLine())
        Volume = Radius * Radius * Length * PI
        Console.WriteLine("Volume of cylinder is " & volume)
        Console.ReadKey()
    End Sub
End Module
```

**Java**

Every console program in Java must contain a class with the file name and a main procedure. These statements are shown in red.

Every statement in Java must have a semi-colon ; at the end.

```
import java.util.Scanner;
class Cylinder
{
   public static void main(String args[])
    {
          Scanner myObj = new Scanner(System.in);
          final double PI = 3.142;
          double Radius;
          double Length;
          System.out.println("Please enter the radius of the
          cylinder ");
          Radius = myObj.nextDouble();
          System.out.println("Please enter the length of the
          sphere ");
          Length = myObj.nextDouble();
          double Volume = Radius * Radius * Length * PI;
      System.out.println("Volume of cylinder is " + volume);
      }
}
```

8.1.4 Basic concepts When writing the steps required to solve a problem, the following concepts need to be used and understood:

» Sequence
» Selection
» Iteration
» Counting and totalling
» String handling
» Use of operators.

## 8.1.4(a) Sequence

The ordering of the steps in an algorithm is very important. An incorrect order can lead to incorrect results and/or extra steps that are not required by the task.

▼ **Table 8.5** Trace table of dry run for algorithm with an incorrect sequence

| Total | Average | Counter | Mark | OUTPUT |
|---|---|---|---|---|
| 0 | 0 | 0 | | Enter marks, 999 to finish |
| 25 | | 1 | 25 | |
| 52 | | 2 | 27 | |
| 75 | | 3 | 23 | |
| 1074 | 358 | 4 | 999 | |
| | | | | The total mark is 1074 |
| | | | | The average mark is 358 |
| | | | | The number of marks is 4 |

As you can see all the outputs are incorrect.

However if the order of the steps is changed, and the unnecessary test removed, the algorithm now works.

```
DECLARE, Total, Average : REAL
DECLARE Mark, Counter : INTEGER
Total ← 0
Mark ← 0
Counter ← -1
OUTPUT "Enter marks, 999 to finish "
REPEAT
  Total ← Total + Mark
  INPUT Mark
  Counter ← Counter + 1
UNTIL Mark = 999
OUTPUT "The total mark is ", Total
Average ← Total / Counter
OUTPUT "The average mark is ", Average
OUTPUT "The number of marks is ", Counter
```

Counter starts at –1 to allow the extra mark of 999 not to be counted.

Average now calculated outside the loop. No need to test if the last input.

A trace table is completed using the same test data:

25, 27, 23, 999

▼ **Table 8.6** Trace table of dry run for algorithm with a correct sequence

| Total | Average | Counter | Mark | OUTPUT |
|---|---|---|---|---|
| 0 | 0 | –1 | | Enter marks, 999 to finish |
| 25 | | 0 | 25 | |
| 52 | | 1 | 27 | |
| 75 | | 2 | 23 | |
| | | 3 | 999 | |
| | | | | The total mark is 75 |
| | 25 | | | The average mark is 25 |
| | | | | The number of marks is 3 |

As you can see all the outputs are now correct.

## 8.1.4(b) Selection

Selection is a very useful technique, allowing different routes through the steps of a program. For example, data items can be picked out according to given criteria, such as: selecting the largest value or smallest value, selecting items over a certain price, selecting everyone who is male

### IF statements

Look at some of the different types of IF statements available in your programming language.

**Table 8.7** IF statements single choice

| IF statement single choice example | Language |
|---|---|
| `IF Age > 17`<br>`  THEN`<br>`  OUTPUT "You are an adult"`<br>`ENDIF` | Pseudocode |
| `if Age > 17:`<br>`  print ("You are an adult")` | Python does not use **THEN** or **ENDIF** just a colon : and indentation |
| `If Age > 17 Then`<br>`  Console.WriteLine("You are an adult")`<br>`End If` | Visual Basic uses **Then** and **End If** |
| `If (Age > 17) {`<br>`  System.out.println ("You are an adult");`<br>`}` | Java uses curly brackets, {, instead of **THEN** and uses } instead of **ENDIF**. |

**Table 8.8** IF statements single choice with alternative

| IF statement single choice with alternative example | Language |
|---|---|
| `IF Age > 17`<br>`  THEN`<br>`  OUTPUT "You are an adult"`<br>`  ELSE`<br>`  OUTPUT "You are a child"`<br>`ENDIF` | Pseudocode |

▼ **Table 8.8** (*Continued*)

| | |
|---|---|
| `if Age > 17:`<br>`  print ("You are an adult")`<br>`else:`<br>`  print ("You are a child")` | Python uses else with a colon, : and indentation. |
| `If Age > 17 Then`<br>`  Console.WriteLine("You are an adult")`<br>`Else`<br>`  Console.WriteLine("You are a child")`<br>`End If` | Visual Basic uses **Else** and **End If** |
| `If (Age > 17) {`<br>`  System.out.println ("You are an adult");`<br>`} else {`<br>`  System.out.println ("You are a child");`<br>`}` | Java uses **else** and curly brackets, {, and uses } instead of **ENDIF**. |

## Case statements

Case statements are used when there are multiple choices to be made. Different programming languages provide different types of statement to do this. Have a look at the method your programming language uses.

▼ **Table 8.9** CASE statements multiple choice

| Case statement examples | Language |
|---|---|
| `CASE OF OpValue`<br>`  "+" : Answer ← Number1 + Number2`<br>`  "-" : Answer ← Number1 - Number2`<br>`  "*" : Answer ← Number1 * Number2`<br>`  "/" : Answer ← Number1 / Number2`<br>`  OTHERWISE OUTPUT "Please enter a valid choice"`<br>`ENDCASE` | Pseudocode |
| `if OpValue == "+":`<br>`  Answer = Number1 + Number2`<br>`elif OpValue == "-":`<br>`  Answer = Number1 - Number2`<br>`elif OpValue == "*":`<br>`  Answer = Number1 * Number2`<br>`elif OpValue == "/":`<br>`  Answer = Number1 - Number2`<br>`else: print("invalid operator")` | Python uses **elif** for multiple tests |

▼ **Table 8.9** (*Continued*)

| | |
|---|---|
| `Select Case OpValue`<br>`  Case "+"`<br>`  Answer = Number1 + Number2`<br>`  Case "-"`<br>`  Answer = Number1 - Number2`<br>`  Case "*"`<br>`  Answer = Number1 * Number2`<br>`  Case "/"`<br>`  Answer = Number1 / Number2`<br>`  Case Else`<br>`  Console.WriteLine("invalid operator")`<br>`  End Select` | Visual Basic uses **Select Case** and **Case Else** instead of **CASE** and **OTHERWISE** |
| `switch (OpValue) {`<br>`  case "+":`<br>`  Answer = Number1 + Number2;`<br>`  break;`<br>`  case "-":`<br>`  Answer = Number1 - Number2;`<br>`  break;`<br>`  case "*":`<br>`  Answer = Number1 * Number2;`<br>`  break;`<br>`  case "/":`<br>`  Answer = Number1 / Number2;`<br>`  break;`<br>`  default:`<br>`  System.out.println("invalid operator");`<br>`}` | Java uses **default** instead of **OTHERWISE** and uses **break** to pass control to the end of the code block when a section is finished. |

### 8.1.4(c) Iteration

As stated in Chapter 7, there are three types of loop structures available to perform iterations so that a section of programming code can be repeated under certain conditions. These are:

» Count-controlled loops (for a set number of iterations)
» Pre-condition loops – may have no iterations
» Post-condition loops – always has at least one iteration.

## Count-controlled loops

FOR loops are used when a set number of iterations are required. Look at some of the different types of FOR statements with a counter starting at one, finishing at ten and increments by two for every iteration.

**Table 8.10** FOR loops

| For statement examples | Language |
|---|---|
| `for Counter in range (1,11,2):`<br>`  print(Counter)` | Python uses the **range** function, a colon to show the start of the **for** loop and indentation of all statements in the **for** loop. |
| `For Counter = 1 To 10 Step 2`<br>`  Console.WriteLine(Counter)`<br>`Next` | Visual Basic uses **Step** and **Next** |
| `for (int Counter = 1; Counter <= 10; Counter = Counter + 2)`<br>`{`<br>`  System.out.println(Counter);`<br>`}` | Java uses {} to show the start and end of the **for** loop. |

## Condition-controlled loops

When the number of iterations is not known, there are two options:
» Pre-condition loops which may have no iterations
» Post-condition loops which always have at least one iteration. Look at some of the different pre- and post-condition loops used in your programming language.

**Table 8.11** Pre-condition loops

| Pre-condition loops | Language |
|---|---|
| `while TotalWeight < 100:`<br>`  TotalWeight = TotalWeight + Weight` | Python uses a colon to show the start of the **while** loop and indentation to show which statements are in the **while** loop. |
| `While TotalWeight < 100`<br>`  TotalWeight = TotalWeight + Weight`<br>`End While` | Visual Basic uses **While** and **End While** |
| `while (TotalWeight < 100)`<br>`{`<br>`  TotalWeight = TotalWeight + Weight;`<br>`}` | Java uses {} to show the start and end of the **while** loop. |

**Table 8.12** Post-condition loops

| Post-condition loops | Language |
|---|---|
| | Python only uses pre-condition loops. |
| `Do`<br>`  NumberOfItems = NumberOfItems + 1`<br>`Loop Until NumberOfItems > 19` | Visual Basic uses `Do` and `Loop Until` |
| `Do`<br>`{`<br>`  NumberOfItems ++;`<br>`}`<br>`while (NumberOfItems <= 20);` | Java uses `do` and `while` so the condition is the opposite of the `until` condition used in Visual Basic. Look at the code carefully to see the difference. Java uses {} to show the start and end of the loop. |

### 8.1.4(d) Totalling and counting

**Totalling** and **counting** were introduced in Chapter 7 as standard methods. Take a look at how these methods are implemented in your programming language. Totalling is very similar in all three languages.

**Table 8.13** Totalling

| Totalling | Language |
|---|---|
| `TotalWeight = TotalWeight + Weight` | Python |
| `TotalWeight = TotalWeight + Weight` | Visual Basic |
| `TotalWeight = TotalWeight + Weight;` | Java |

Counting is also similar in Python and Visual Basic, but Java uses a different type of statement.

**Table 8.14** Counting

| Counting | Language |
|---|---|
| `NumberOfItems = NumberOfItems + 1`<br>`NumberOfItems +=1` | Python |
| `NumberOfItems = NumberOfItems + 1`<br>`NumberOfItems +=1` | Visual Basic |
| `NumberOfItems ++;`<br>`NumberOfItems = NumberOfItems + 1;` | Java |

### 8.1.4(e) String Handling

String handling is an important part of programming. As an IGCSE Computer Science student, you will need to write algorithms and programs for these methods:

» Length – finding the number of characters in the string. For example, the length of the string "Computer Science" is 16 characters as spaces are counted as a character.

» Substring – extracting part of a string. For example, the substring "Science" could be extracted from "Computer Science".

» Upper – converting all the letters in a string to uppercase. For example, the string "Computer Science" would become "COMPUTER SCIENCE".

» Lower – converting all the letters in a string to lowercase. For example, the string "Computer Science" would become "computer science".

These string manipulation methods are usually provided in programming languages by library routines, see later in this chapter for further use of **library routines.**

**Table 8.15** Find the length of a string

| Length | Language | Notes |
|---|---|---|
| `LENGTH("Computer Science")`<br>`LENGTH(MyString)` | Pseudocode | Text in quotes can be used or a variable with data type string. |
| `len("Computer Science")`<br>`len(MyString)` | Python | Text in quotes can be used or a variable with data type string. |
| `"Computer Science".Length()`<br>`MyString.Length()` | Visual Basic | Text in quotes can be used or a variable with data type string. |
| `"Computer Science".length();`<br>`MyString.length();` | Java | Text in quotes can be used or a variable with data type string. |

**Table 8.16** Extracting a substring from a string

| Substring – to extract the word 'Science' | Language | Notes |
|---|---|---|
| `SUBSTRING("Computer Science", 10, 7)`<br>`SUBSTRING(MyString, 10, 7)` | Pseudocode | Text in quotes can be used or a variable with data type string. First parameter is the string, second parameter is the position of the start character, third parameter is the length of the required substring.<br>Pseudocode strings start at position one. |
| `"Computer Science"[9:16]`<br>`MyString[9:16]` | Python | Text in quotes can be used or a variable with data type string. Strings are treated as lists of characters in Python. First index is the start position of the substring. Second index is the end position of the substring.<br>Python strings start at position zero. |
| `"Computer Science".Substring(9, 7)`<br>`MyString.Substring(9, 7)` | Visual Basic | Text in quotes can be used or a variable with data type string. First parameter is the start position of the substring, second parameter is the length of the substring.<br>Visual Basic strings start at position zero. |
| `"Computer Science".substring(9,17);`<br>`MyString.substring(9,17);` | Java | Text in quotes can be used or a variable with data type string. First parameter is the start position of the substring, second parameter is the exclusive end position of the substring. 'Exclusive' means the position after the last character, i.e. in this example the substring is made from characters 9 to 16.<br>Java strings start at position zero. |

**Table 8.17** Converting a string to upper case

| Upper | Language | Notes |
|---|---|---|
| `UCASE("Computer Science")`<br>`UCASE(MyString)` | Pseudocode | Text in quotes can be used or a variable with data type string. |
| `"Computer Science".upper()`<br>`MyString.upper()` | Python | Text in quotes can be used or a variable with data type string. |
| `UCase("Computer Science")`<br>`UCase(MyString)` | Visual Basic | Text in quotes can be used or a variable with data type string. |
| `"Computer Science".toUpperCase();`<br>`MyString.toUpperCase();` | Java | Text in quotes can be used or a variable with data type string. |

▼ **Table 8.18** Converting a string to lower case

| Lower | Language | Notes |
|---|---|---|
| `LCASE("Computer Science")`<br>`LCASE(MyString)` | Pseudocode | Text in quotes can be used or a variable with data type string. |
| `"Computer Science".lower()`<br>`MyString.lower()` | Python | Text in quotes can be used or a variable with data type string. |
| `LCase("Computer Science")`<br>`LCase(MyString)` | Visual Basic | Text in quotes can be used or a variable with data type string. |
| `"Computer Science".toLowerCase();`<br>`MyString.toLowerCase();` | Java | Text in quotes can be used or a variable with data type string. |

## 8.1.4(f) Arithmetic, logical and Boolean operators

**Arithmetic operators**

All programming languages make use of arithmetic operators to perform calculations. Here are the ones you must be able to use for IGCSE Computer Science.

▼ **Table 8.19** Mathematical operators

| Operator | Action | Python | Visual Basic | Java |
|---|---|---|---|---|
| + | Add | + | + | + |
| - | Subtract | - | - | - |
| * | Multiply | * | * | * |
| / | Divide | / | / | / |
| ^ | Raise to the power of | ** | ^ | import java.lang.Math;<br>Math.pow(x, y) |
| MOD | Remainder division | For more on these see Section 8.1.7 Library routines. | | |
| DIV | Integer division | | | |

**Logical operators**

All programming languages make use of logical operators to decide which path to take through a program. Here are the ones you must be able to use for IGCSE Computer Science.

▼ **Table 8.20** Logical operators

| Operator | Comparison | Python | Visual Basic | Java |
|---|---|---|---|---|
| > | Greater than | > | > | > |
| < | Less than | < | < | < |
| = | Equal | == | = | == |
| >= | Greater than or equal | >= | >= | >= |
| <= | Less than or equal | <= | <= | <= |
| <> | Not equal | != | <> | != |

**Boolean operators**

All programming languages make use of Boolean operators to decide whether an expression is true or false. Here are the ones you must be able to use for IGCSE Computer Science.

▼ **Table 8.21** Boolean operators

| Operator | Description | Python | Visual Basic | Java |
|---|---|---|---|---|
| AND | Both True | and | And | && |
| OR | Either True | or | Or | \|\| |
| NOT | Not True | not | Not | ! |

### 8.1.5 Use of nested statements

Selection and iteration statements can be nested one inside the other. This powerful method reduces the amount of code that needs to be written and makes it simpler for a programmer to test their programs.

### 8.1.6 Procedures and functions

When writing an algorithm, there are often similar tasks to perform that make use of the same groups of statements. Instead of repeating these statements and writing new code every time they are required, many programming languages make use of subroutines, also known as named procedures or functions. These are defined once and can be called many times within a program.

#### Procedures, functions and parameters

A **procedure** is a set of programming statements grouped together under a single name that can be called to perform a task at any point in a program.

A **function** is a set of programming statements grouped together under a single name that can be called to perform a task at any point in a program. In contrast to a procedure, a function will return a value back to the main program.

**Parameters** are the variables that store the values of the arguments passed to a procedure or function. Some but not all procedures and functions will have parameters.

#### Definition and use of procedures and functions, with or without parameters

**Procedures without parameters**

Here is an example of a procedure without parameters in pseudocode:

```
PROCEDURE Stars
    OUTPUT"************"
ENDPROCEDURE
```

The procedure can then be called in the main part of the program as many times as is required in the following way:

```
CALL Stars
```

Instead of calling them procedures, different terminology is used by each programming language. Procedures are known as:

- void functions in Python
- subroutines in VB
- methods in Java.

**Table 8.22** Procedure calls

| Procedure Stars – definition | Call | Language |
|---|---|---|
| `def Stars():`<br>`  print("************")` | `Stars()` | Python |
| `Sub Stars()`<br>`  Console.WriteLine("************")`<br>`End Sub` | `Stars()` | Visual Basic |
| `static void Stars()`<br>`{`<br>`  System.out.println("************");`<br>`}` | `Stars();` | Java |

## Procedures with parameters

Here is an example of how a procedure with parameters can be defined in pseudocode.

We can add parameters to a procedure:

```
PROCEDURE Stars (Number : INTEGER)
    DECLARE Counter : INTEGER
    FOR Counter ← 1 TO NUMBER
        OUTPUT "*"
    NEXT Counter
ENDPROCEDURE
```

Parameter with data type

Procedure with parameters are called like this – in this case to print seven stars:

```
CALL Stars (7)
```

Or:

```
MyNumber ← 7
CALL stars (MyNumber)
```

A procedure call must match the procedure definition. This means that when a procedure is defined with parameters, the arguments in the procedure call should match the parameters in the procedure definition. For IGCSE Computer Science the number of parameters used is limited to two.

▼ **Table 8.23** Procedures with parameters

| Procedure stars with parameter – definition | Call | Language |
|---|---|---|
| `def Stars(Number):`<br>`  for counter in range (Number):`<br>`  print("*", end = "")` | `Stars(7)` | Python<br>Note: `end = ""` ensures that the stars are printed on one line without spaces between them. |
| `Sub Stars(Number As Integer)`<br>`  Dim Counter As Integer`<br>`  For Counter = 1 To Number`<br>`  Console.Write("*")`<br>`  Next`<br>`End Sub` | `Stars(7)` | VB |
| `static void Stars(int Number)`<br>`{`<br>`  for (int Counter = 1; Counter <= Number; Counter ++)`<br>`  {`<br>`  System.out.print("*");`<br>`  }`<br>`}` | `Stars(7);` | Java |

## Functions

A function is just like a procedure except it **always** returns a value. Just like a procedure it is defined once and can be called many times within a program. Just like a procedure it can be defined with or without parameters.

Instead of naming them functions, different terminology is used by some programming languages. Functions are known as:
» Fruitful functions in Python
» Functions in VB
» Methods with returns in Java.

The keyword **RETURN** is used as one of the statements in a function to specify the value to be returned. This is usually the last statement in the function definition.

For example, here is a function written in pseudocode to convert a temperature from Fahrenheit to Celsius:

Parameter and function return both have defined data types.

```
FUNCTION Celsius (Temperature : REAL) RETURNS REAL
    RETURN (Temperature - 32) / 1.8
ENDFUNCTION
```

Because a function returns a value, it can be called by assigning the return value directly into a variable as follows:

```
MyTemp ← Celsius(MyTemp)
```

▼ **Table 8.24** Function definitions

| Function temperature conversion example | Language |
|---|---|
| `def Celsius(Temperature):`<br>`  return (Temperature - 32) / 1.8` | Python – data type of function does not need to be defined |
| `Function Celsius(ByVal Temperature As Decimal) As Decimal`<br>`  Return (Temperature - 32) / 1.8`<br>`End Function` | Visual Basic |
| `static double Celsius(double Temperature)`<br>`{`<br>`  return (Temperature - 32) / 1.8;`<br>`}` | Java |

Just like with a procedure, a function call must match the function definition. When a function is defined with parameters, the **arguments** in the function call should match the parameters in the procedure definition. For IGCSE Computer Science the number of parameters used is limited to two.

When procedures and functions are defined, the first statement in the definition is a header, which contains:
» The name of the procedure or function
» Any parameters passed to the procedure or function, and their data type
» The data type of the return value for a function.

## Local and global variables

A **global variable** can be used by any part of a program – its **scope** covers the whole program.

A **local variable** can only be used by the part of the program it has been declared in – its **scope** is restricted to that part of the program.

### 8.1.7 Library routines

Many programming language development systems include library routines that are ready to incorporate into a program. These routines are fully tested and ready for use.

You will need to use these **library routines** in your programs for IGCSE Computer Science:
» MOD – returns remainder of a division
» DIV – returns the quotient (i.e. the whole number part) of a division
» ROUND – returns a value rounded to a given number of decimal places
» RANDOM – returns a random number.

Here are some examples of these library routines in pseudocode:

`Value1` ← `MOD(10,3)` returns the remainder of `10` divided by `3`

`Value2` ← `DIV(10,3)` returns the quotient of `10` divided by `3`

`Value3` ← `ROUND(6.97354, 2)` returns the value rounded to `2` decimal places

`Value4` ← `RANDOM()` returns a random number between `0` and `1` inclusive

Here are the same examples, written in each programming language:

▼ **Table 8.25** MOD, DIV, ROUND and RANDOM programming examples

| Programming examples for MOD, DIV, ROUND and RANDOM | Language |
|---|---|
| `Value1 = 10%3`<br>`Value2 = 10//3`<br>`Value = divmod(10,3)`<br>`Value3 = round(6.97354, 2)`<br>`from random import random`<br>`Value4 = random()` | Python – MOD uses the `%` operator and DIV the `//` operator<br>The function `divmod(x,y)` provides both answers where the first answer is DIV and the second answer is MOD<br>RANDOM needs to import the library routine `random`, and then the function: `random()` can be used afterwards |
| `Value1 = 10 Mod 3`<br>`Value2 = 10 \ 3`<br>`Value3 = Math.Round(6.97354, 2)`<br>`Value4 = Rnd()` | Visual Basic – DIV uses the `\` operator |
| `import java.lang.Math;`<br>`Value1 = 10%3;`<br>`Value2 = 10/3;`<br>`Value3 = Math.round(6.97354 * 100)/100.0;`<br>`import java.util.Random;`<br>`Random rand = new Random();`<br>`double Value4 = rand.nextDouble();` | Java – MOD uses the `%` operator<br>DIV uses the normal division operator; if the numbers being divided are both integers then integer division is performed, as shown.<br>Java imports the library routine `Math`<br>`Math.round` only rounds to whole numbers<br>RANDOM needs to import the library routine `Random` |

### 8.1.8 Creating a maintainable program

Once a program is written, it may need to be maintained or updated by another programmer later.

A maintainable program should:
» Always use meaningful identifier names for:
– variables – constants – arrays – procedures – functions
» Be divided into modules for each task using: – procedures– functions

» Be fully commented using your programming language's commenting feature.

# 8.2 Arrays

An array is a data structure containing several elements of the same data type; these elements can be accessed using the same identifier name. The position of each element in an array is identified using the array's index.

▼ **Table 8.26** Programming languages commenting features

| Example comments | Language |
|---|---|
| `#Python uses hash to start a comment for every line` | Python |
| `'Visual Basic uses a single quote to start a comment for every line` | Visual Basic |
| `// Java uses a double slash to start a single line comment and /* to start multiple line comments and to end them */` | Java |

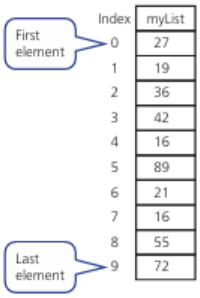
## 8.2.1 One- and Two-dimensional arrays

Arrays are used to store multiple data items in a uniformly accessible manner; all the data items use the same identifier, and each data item can be accessed separately by the use of an index.

## 8.2.2 Declaring and populating arrays with iteration

**One-dimensional arrays**

A one-dimensional array can be referred to as a list. Here is an example of a list with 10 elements in it where the first element has an index of zero.




▲ **Figure 8.10** A one-dimensional array

When a one-dimensional array is declared in pseudocode:

- the name of the array
- the first index value
- the last index value
- and the data type

are included.

For example, to declare a new array called **MyList**:

```
DECLARE MyList : ARRAY[0:9] OF INTEGER
```

Each position in the array can be populated in an array by defining the value at each index position. For instance, we can add the number 27 to the fourth position in the array **MyList** as follows:

```
MyList[4] ← 27
```

To populate the entire array instead we can use a loop:

```
OUTPUT "Enter these 10 values in order 27, 19, 36, 42, 16, 89, 21, 16, 55, 72"
FOR Counter ← 0 TO 9
    OUTPUT "Enter next value "
    INPUT MyList[Counter]
NEXT Counter
```

Notice that in this code we have used the variable **Counter** as the array index.

We can display the data that lies in a particular location in an array as follows:

```
OUTPUT MyList[1]
```

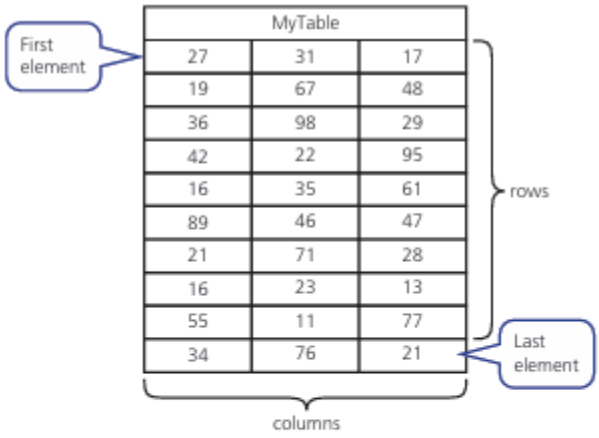
This would display the value 19.

▼ **Table 8.27** Array population

| Array | Language |
|---|---|
| `myList = [27, 19, 36, 42, 16, 89, 21, 16, 55, 72]` | Python |
| `Dim myList = New Integer() {27, 19, 36, 42, 16, 89, 21, 16, 55, 72}` | Visual Basic |
| `int[] myList = {27, 19, 36, 42, 16, 89, 21, 16, 55, 72};` | Java |

### Two-dimensional arrays

A two-dimensional array can be referred to as a table, with rows and columns. Here is an example of a table with 10 rows and 3 columns, which contains 30 elements. The first element is located at position 0,0.




▲ **Figure 8.11** A two-dimensional array

When a two-dimensional array is declared in pseudocode:

- the first index value for rows
- the last index value for rows
- the first index value for columns
- the last index value for columns
- and the data type

Two-dimensional arrays are populated in each language as follows:

| Two-dimensional arrays | Language |
|---|---|
| `MyTable = [[27, 31, 17], [19, 67, 48],[36, 98, 29],[42, 22, 95], [16, 35, 61], [89, 46, 47], [21, 71, 28], [16, 23, 13], [55, 11, 77] [34, 76, 21]]` | Python |
| `Dim MyTable = New Integer(8, 2) {{27, 31, 17}, {19, 67, 48}, {36, 98, 29}, {42, 22, 95}, {16, 35, 61}, {89, 46, 47}, {21, 71, 28}, {16, 23, 13}, {55, 11, 77}, (34, 76, 21)}` | Visual Basic |
| `int[][] MyTable = {{27, 31, 17}, {19, 67, 48}, {36, 98, 29}, {42, 22, 95}, {16, 35, 61}, {89, 46, 47}, {21, 71, 28}, {16, 23, 13}, {55, 11, 77}, {34, 76, 21}};` | Java |

# 8.3 File handling

## 8.3.1 Purpose of storing data in a file

Computer programs store data that will be required again in a file. While any data stored in RAM will be lost when the computer is switched off, when data is saved to a file it is stored permanently.

### 8.3.2 Using files

Every file is identified by its filename. In this section, we are going to look at how to read and write a line of text or a single item of data to a file.

Here are examples of writing a line of text to a file and reading the line of text back from the file. The pseudocode algorithm has comments to explain each stage of the task.

**pseudocode**

```
DECLARE TextLine : STRING  // variables are declared as normal
DECLARE MyFile : STRING
MyFile ← "MyText.txt"
// writing the line of text to the file
OPEN MyFile FOR WRITE  // opens file for writing
    OUTPUT "Please enter a line of text"
    INPUT TextLine
    WRITEFILE, TextLine  // writes a line of text to the file
CLOSEFILE(MyFile)  // closes the file
// reading the line of text from the file
OUTPUT "The file contains this line of text:"
OPEN MyFile FOR READ  // opens file for reading
    READFILE, TextLine  // reads a line of text from the file
    OUTPUT TextLine
CLOSEFILE(MyFile)  // closes the file
```

**Python**

```
# writing to and reading a line of text from a file
MyFile = open ("MyText.txt","w")
TextLine = input("Please enter a line of text ")
MyFile.write(TextLine)
Myfile.close()
print("The file contains this line of text")
MyFile = open ("MyText.txt","r")
TextLine = MyFile.read()
print(TextLine)
Myfile.close()
```

**Visual Basic**

```
'writing to and reading from a text file
Imports System.IO
Module Module1
    Sub Main()
        Dim textLn As String
        Dim objMyFileWrite As StreamWriter
        Dim objMyFileRead As StreamReader
        objMyFileWrite = New StreamWriter("textFile.txt")
        Console.Write("Please enter a line of text  ")
        textLn = Console.ReadLine()
        objMyFileWrite.WriteLine(textLn)
        objMyFileWrite.Close()
        Console.WriteLine("The line of text is  ")
        objMyFileRead = New StreamReader("textFile.txt")
        textLn = objMyFileRead.ReadLine
        Console.WriteLine(textLn)
        objMyFileRead.Close()
        Console.ReadLine()
    End Sub
End Module
```

Java

```
//writing to and reading from a text file, Note: the try and catch commands implement
// something called exception handling - this concept goes beyond IGCSE level
import java.util.Scanner;
import java.io.BufferedReader;
import java.io.PrintWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
class TextFile {
    public static void main(String[] args) {
        Scanner myObj = new Scanner(System.in);
        String textLn;
       try {
            FileWriter myFileWriter = new FileWriter("textFile.txt", false);
            PrintWriter myPrintWriter = new PrintWriter(myFileWriter);
            System.out.println("Please enter a line of text  ");
            textLn = myObj.next();
            myPrintWriter.printf("%s" + "%n", textLn);
            myPrintWriter.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
        try {
            FileReader myFileReader  = new FileReader("textFile.txt");
            BufferedReader myBufferReader  = new BufferedReader(myFileReader);
            textLn = myBufferReader.readLine();
            System.out.println(textLn);
            myFileReader.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

## Key terms used throughout this chapter

**variable** – a named data store that contains a value that may change during the execution of a program

**constant** – a named data store that contains a value that does not change during the execution of a program

**declare** – define the value and data type of a variable or constant

**integer** – a positive or negative whole number that can be used with mathematical operators

**real number** – a positive or negative number with a fractional part; Real numbers can be used with mathematical operators

**char** – a variable or constant that is a single character

**string** – a variable or constant that is several characters in length. Strings vary in length and may even have no characters (an empty string); the characters can be letters and/or digits and/or any other printable symbol

**sequence** – the order in which the steps in a program are executed

**selection** – allowing the selection of different paths through the steps of a program

**iteration** – a section of programming code that can be repeated under certain conditions

**counting** – keeping track of the number of times an action is performed

**totalling** – keeping a total that values are added to

**operator** – a special character or word in a programming language that identifies an action to be performed

**arithmetic operator** – an operator that is used to perform calculations

**logical operator** – an operator that is used to decide the path to take through a program if the expression formed is true or false

**Boolean operator** – an operator that is used with logical operators to form more complex expressions

**nesting** – the inclusion of one type of code construct inside another

**procedure** – a set of programming statements grouped together under a single name that can be called to perform a task in a program, rather than including a copy of the code every time the task is performed

**function** – a set of programming statements grouped together under a single name which can be called to perform a task in a program, rather than including a copy of the code every time; just like a procedure except a function will return a value back to the main program

**parameters** – the variables in a procedure or function declaration that store the values of the arguments passed from the main program to a procedure or function

**MOD** – an arithmetic operator that returns the remainder of a division; different languages use different symbols for this operation

**DIV** – an arithmetic operator that returns the quotient (whole number part) of a division; different languages use different symbols for this operation

**ROUND** – a library routine that rounds a value to a given number of decimal places

**RANDOM** – a library routine that generates a random number

**array** – a data structure containing several elements of the same data type; these elements can be accessed using the same identifier name

**index** – identifies the position of an element in an array

**file** – a collection of data stored by a computer program to be used again

# Revision questions

1. Four statement types and four examples are shown below. Draw a line to connect each statement type to the correct example

| Statement type | Example |
|---|---|
| Assignment | `FOR X ← 1 TO 10` |
| Iteration | `READ X` |
| Input | `PRINT X` |
| Output | `X ← Y + Z` |

2. A programmer writes a program to store a patient's temperature every hour for a day. State the data structure that would be most suitable to use and give the reason for your choice.

Data structure .....

Reason .....

3. Identify two different selection statements that you can use when writing pseudocode.

4. Most programming languages include basic data types. Ahmad is describing the basic data types he has used. State the data type that Ahmad is describing in each sentence. Choose the data type from this list of programming terms.

**Array** **Boolean** **Char** **Constant** **Function** **Integer**

**Iteration** **Procedure** **Real** **String** **Variable**

A number with a fractional part that can be positive or negative and used in calculations

Data type .....

A whole number that can be positive, negative or zero and used in calculations

Data type .....

A single number, symbol or letter

Data type .....

A sequence of characters

Data type .....

A data type with two values, True or False

Data type .....

5. Four validation checks and four descriptions are shown. Draw a line to connect each validation check to the correct description

| Validation Check | Description |
|---|---|
| Range check | Checks that some data is entered. |
| Presence check | Checks for a maximum number of characters in the data entered. |
| Length check | Checks that the characters entered are all numbers. |
| Type check | Checks that the value entered is between an upper value and a lower value. |

6. A programmer writes a program to weigh baskets of fruit in grams, keeping a total of the weight and counting the number of baskets. The total weight is stored in a variable Total and the number of baskets is stored in a variable BasketCount.

Explain, including examples of programming statements, how totalling and counting could be used in this program.

7. An algorithm has been written in pseudocode to calculate a check digit for a four-digit number. The algorithm then outputs the five-digit number including the check digit. The algorithm stops when –1 is input as the fourth digit.

```
01 Flag FALSE
02 REPEAT
03 Total 0
04 FOR Counter 1 TO 4
05 OUTPUT "Enter a digit ", Counter
06 INPUT Number[Counter]
07 Total Total + Number * Counter
08 IF Number[Counter] = 0
09 THEN
10 Flag TRUE
11 ENDIF
12 NEXT Counter
13 IF NOT Flag
14 THEN
15 Number[5] MOD(Total, 10)
16 FOR Counter 0 TO 5
17 OUTPUT Number[Counter]
18 NEXT
19 ENDIF
20 UNTIL Flag
```

**Give the line number(s)** for the statements showing:

Totalling .....

Count-controlled loop .....

Post-condition loop .....

8.

**Explain the difference** between a WHILE ... DO ... ENDWHILE and a REPEAT ... UNTIL loop

```
B ← FALSE
INPUT Num
FOR Counter ← 1 TO 12
IF A[Counter] = Num
THEN
B ← TRUE
ENDIF
NEXT Counter
```

9. An algorithm has been written in pseudocode to calculate a check digit for a four-digit number. The algorithm then outputs the five-digit number including the check digit. The algorithm stops when –1 is input as the fourth digit.

```
01 Flag FALSE
02 REPEAT
03 Total 0
04 FOR Counter 1 TO 4
05 OUTPUT "Enter a digit ", Counter
06 INPUT Number[Counter]
07 Total Total + Number * Counter
08 IF Number[Counter] = 0
09 THEN
10 Flag ← TRUE
11 ENDIF
12 NEXT Counter
13 IF NOT Flag
14 THEN
15 Number[5] MOD(Total, 10)
16 FOR Counter 0 TO 5
17 OUTPUT Number[Counter]
18 NEXT
19 ENDIF
20 UNTIL Flag
```

The algorithm does not check that each input is a single digit.
Identify the place in the algorithm where this check should occur.

**Write pseudocode for this check**. Your pseudocode must make sure that the input is a single digit and checks for –1