Notes and guidance: VB.NET

The VB.NET code is described below to help students prepare for their AQA GCSE Computer Science exam (8525/1).

We will use a consistent style of VB.NET code in all assessment material. This will ensure that, with enough preparation, students will understand the syntax of the code used in assessments. Students do not have to use this style of code in their own work or written assessments, although they are free to do so. The only direction to students when answering questions or describing algorithms written in code is that their code is clear, consistent and unambiguous.

This resource may be updated as required and the latest version will always be available on our website. It is not confidential and can be freely shared with students.

General Syntax

- Code is shown in `this font`
- `DataType` means a datatype such as `Integer`, `Single`, `Double`, `Boolean`, `Char` or `String`.
- `Exp` means any expression.
- `IntExp`, `RealExp`, `BoolExp`, `StringExp`, `CharExp` and `ListExp` mean any expression which can be evaluated to an integer, real, Boolean (`False` or `True`), string, character or list respectively.

Indentation

VB.NET code will use indentation to indicate the range of statements controlled by iteration and selection statements (as well as declarations for subroutines). Indentation will be shown with three spaces per indentation level, although if doing so makes lines too long for the page, this may be reduced to two spaces. Questions will show indentation guides (vertical lines) within the answer space. Students should be encouraged to use these to explicitly show their indentation.

Comments

Single line comments	`' A comment`	
Multi-line comments	`' A comment` `' Another comment`	

String and character literals

String and character literals will be delimited using the " (double quote) character.

Variables and constants

Variable names will be written in camel case eg `numberOfItemsSold`. Camel case is the practice of writing phrases without spaces or punctuation, indicating the separation of words with a single capitalised letter and the first word starting with either case.

Constant names will be written in upper case, using an underscore to indicate a break between words, eg `ACCELERATION_DUE_TO_GRAVITY`.

Questions will use meaningful variable names wherever possible, eg `quantityInStock`, `quantity` or `qty` rather than just `n` to hold the quantity of an item in stock. For layout and/or lack of context reasons, this resource may not always follow this advice. This rule may not be followed for common idioms, eg using `i` as a loop index or for exam-related reasons.

Before a variable is used, it will be declared in a `Dim` statement. This will give the name of the variable, its data type and, optionally, an initial value. Note that multiple variables may be defined in the same `Dim` statement and given the same data type. If an initial value is given and its data type is unambiguous then the data type may be omitted; "S", 0 are examples of ambiguous values since "S" could be a string or a character, and 0 could be integer or single.

Variable declaration	`Dim Identifier As DataType`
	`Dim Identifier As DataType = Value`
		`Dim aNumber As Integer`
		`Dim anotherNumber As Integer = 0`
		`Dim theString, Message As String`
		`' Both theString and Message are`
		`' declared as strings`

Variable assignment	`Identifier = Exp`	`aNumber = 3`
		`anotherNumber = aNumber + 1`
		`theString = "Hello"`
		`message = "Invalid number"`
		`totalNumberOfItems = 0`

Constant declaration	`Const IDENTIFIER = Exp`
	
	`Const IDENTIFIER As DataType = Value`
		`Const CLASS_SIZE = 23`
		`Const PI = 3.141`
		
		`Const TOTAL As Single = 0.0`

Arithmetic operations

Standard arithmetic operations	`+`
	`-`
	`*`
	`/`
		Used in the normal way with brackets to indicate precedence where needed. So, `a + b * c` would multiply `b` and `c` together and then add the result to `a`, whereas `(a + b) * c` would add `a` and `b` together and then multiply the result by `c`.
		
		Brackets may be used to indicate precedence even where not strictly necessary, eg testing for divisibility by 3 could be written as `(n Mod 3) = 0` rather than `n Mod 3 = 0`

Integer division	`IntExp \ IntExp`	`9 \ 5` evaluates to `1`
		`5 \ 2` evaluates to `2`
		`8 \ 4` evaluates to `2`

Modulus operator	`IntExp Mod IntExp`	`9 Mod 5` evaluates to `4`
		`5 Mod 2` evaluates to `1`
		`8 Mod 4` evaluates to `0`

Relational operators for types that can be clearly ordered (numbers, strings, characters)

Less than	`Exp < Exp`	`4 < 6`
		`"A" < "B"`
		`"adam" < "adele"`

Greater than	`Exp > Exp`	`4.1 > 4.0`
Equal to	`Exp = Exp`	`3 = 3`
Not equal to	`Exp <> Exp`	`qty <> 7`
Less than or equal to	`Exp <= Exp`	`3 <= 4` `4 <= 4`
Greater than or equal to	`Exp >= Exp`	`4 >= 3` `4.5 >= 4.5`

Boolean operations

Logical AND	`BoolExp And BoolExp`	`(3 = 3) And (3 <= 4)`
Logical OR	`BoolExp Or BoolExp`	`(x < 1) Or (x > 9)`
Logical NOT	`Not BoolExp`	`Not (a < b)`

Indefinite (condition controlled) iteration

WHILE (while the Boolean expression is `True`, repeat the statements). If the Boolean expression is `False` the first time the `While` statement is reached then the indented statements are never executed.	`While BoolExp`
	`  ' indented statements here`
	`End While`
		`Dim a As Integer = 1`
		`While a < 4`
		`  Console.WriteLine(a)`
		`  a = a + 1`
		`End While`
		`' outputs 1, 2, 3`
		
		whereas
		
		`Dim a As Integer = 5`
		`While a < 4`
		`  Console.WriteLine(a)`
		`  a = a + 1`
		`End While`
		`' does not output anything since`
		`' a < 4 is false the first time the`
		`' While is encountered`

Indefinite (condition controlled) iteration (continued)

<table>
<tr>
<td>Another form of the WHILE works in exactly the same way as above (notice the similarity between this form and that in REPEAT-UNTIL).</td>
<td>

```
Do While BoolExp
   ' indented statements here
Loop
```

</td>
<td>

```
Dim a As Integer = 1
Do While a < 4
   Console.WriteLine(a)
   a = a + 1
Loop
' outputs 1, 2, 3
```

whereas

```
Dim a As Integer = 5
Do While a < 4
   Console.WriteLine(a)
   a = a + 1
Loop
' does not output anything since
' a < 4 is false the first time the
' While is encountered
```

</td>
</tr>
<tr>
<td>REPEAT-UNTIL (repeat the statements until the Boolean expression is <code>True</code>). The indented statements are always executed at least once.</td>
<td>

```
Do
   ' indented statements here
Loop Until BoolExp
```

</td>
<td>

```
Dim a As Integer = 1
Dim carryOn As String
Do
   Console.WriteLine(a)
   a = a + 1
   Console.Write("Continue? ")
   carryOn = Console.ReadLine()
Loop Until carryOn = "N"
' outputs 1, … until "N" is entered
```

</td>
</tr>
</table>

Definite (count controlled) iteration

FOR
(repeat the statements the number of times indicated, each time giving the loop variable (`Identifier`) the value of the next value/number in the range). `As DataType` may be omitted if the type is unambiguous (see the fourth example).
	`For Identifier As DataType = IntExp1 To IntExp2`
	`  ' indented statements here`
	`Next`
	`' Identifier will first have the value IntExp1,`
	`' then IntExp1 + 1, IntExp1 + 2, all the way`
	`' up (still in steps of 1) to the value of`
	`' IntExp2`
	
	`For Identifier As DataType = IntExp1 To IntExp2 Step IntExp3`
	`  ' indented statements here`
	`Next`
	`' Identifier will first have the value IntExp1,`
	`' then IntExp1 + IntExp3, IntExp1 + 2 * IntExp3,`
	`' all the way up (or down if IntExp3 is negative)`
	`' to the value of IntExp2`
		`For i As Integer = 0 To 6`
		`  Console.WriteLine(i)`
		`Next`
		`' outputs 0, 1, 2, 3, 4, 5, 6`
		
		`For i As Integer = 1 To 7`
		`  Console.WriteLine(i)`
		`Next`
		`' outputs 1, 2, 3, 4, 5, 6, 7`
		
		`For i As Integer = 1 To 7 Step 2`
		`  Console.WriteLine(i)`
		`Next`
		`' outputs 1, 3, 5, 7`
		
		`For i = 7 To 1 Step -2`
		`  Console.WriteLine(i)`
		`Next`
		`' outputs 7, 5, 3, 1`

Definite (count controlled) iteration (continued)

FOR
(repeat the statements the number of times that there are characters in a string, each time giving `Identifier` the value of the next character in the string).
	`For Each Identifier In StringExp`
	`  ' indented statements here`
	`Next`
		`Dim length As Integer = 0`
		`For Each ch In message`
		`  length = length + 1`
		`Next`
		`Console.WriteLine(length)`
		`' calculate the number of`
		`' characters in message`
		`' and output it`
		
		`Dim reversed As String = ""`
		`For Each ch In message`
		`  reversed = ch + reversed`
		`Next`
		`Console.WriteLine(reversed)`
		
		`' reversed is set to the`
		`' reverse of message and`
		`' output`
		`' eg if message = "Hello"`
		`' then reversed will become`
		`' "olleH"`

Selection

IF-THEN-ENDIF (execute the statements only if the Boolean expression is `True`: see *VB.NET Boolean expressions* above).	`If BoolExp Then`
	`  ' indented statements here`
	`End If`
		`Dim a As Integer = 1`
		`If (a Mod 2) = 0 Then`
		`  Console.WriteLine("even")`
		`End If`

IF-THEN-ELSE-ENDIF (execute the statements following the THEN if the Boolean expression is `True`, otherwise execute the statements following the ELSE).	`If BoolExp Then`
	`  ' indented statements here`
	`Else`
	`  ' indented statements here`
	`End If`
		`Dim a As Integer = 1`
		`If (a Mod 2) = 0 Then`
		`  Console.WriteLine("even")`
		`Else`
		`  Console.WriteLine("odd")`
		`End If`

NESTED IF-THEN-ELSE ENDIF (use nested versions of the above to create more complex conditions).
 	
Note that IF statements can be nested inside the THEN part, the ELSE part or both.
	`If BoolExp Then`
	`  ' indented statements here`
	`Else`
	`  If BoolExp Then`
	`  ' indented statements here`
	`  Else`
	`  ' indented statements here`
	`  End If`
	`End If`
		`a = 1`
		`If (a Mod 4) = 0 Then`
		`  Console.WriteLine("multiple of 4")`
		`Else`
		`  If (a Mod 4) = 1 Then`
		`  Console.WriteLine("remainder 1")`
		`  Else`
		`  If (a Mod 4) = 2 Then`
		`  Console.WriteLine("remainder 2")`
		`  Else`
		`  Console.WriteLine("remainder 3")`
		`  End If`
		`  End If`
		`End If`

Selection (continued)

IF-THEN-ELSE IF ENDIF (removes the need for multiple indentation levels).	`If BoolExp Then`
	`  ' indented statements here`
	`ElseIf BoolExp Then`
	`  ' indented statements here`
	`  ' possibly more ElseIfs`
	`Else`
	`  ' indented statements here`
	`End If`
		`Dim a As Integer = 1`
		`If (a Mod 4) = 0 Then`
		`  Console.WriteLine("multiple of 4")`
		`ElseIf (a Mod 4) = 1 Then`
		`  Console.WriteLine("remainder of 1")`
		`ElseIf (a Mod 4) = 2 Then`
		`  Console.WriteLine("remainder of 2")`
		`Else`
		`  Console.WriteLine("remainder of 3")`
		`End If`

Arrays

Declaration	`Dim Identifier(IntExp) As DataType`
	`' IntExp is the highest value element`
	`' that can be accessed. Since the first`
	`' element is element 0 there are`
	`' IntExp + 1 elements in the array`
	
	`Dim Identifier() As DataType = {value, …}`
	`' In this form there are the number of`
	`' elements in the array that there are in`
	`' {value, …}`
		`Dim primes(5) As Integer`
		`' Has 6 elements, primes(0) to primes(5)`
		
		`Dim evens() As Integer = {2, 4, 6, 8, 10}`
		`' Has 5 elements, evens(0) to evens(4)`
		
		`Dim names() = {"John", "Paul", "George"}`
		`' If the type of the elements is clear`
		`' then As DataType can be omitted`

Assignment	`Identifier = {Exp, … ,Exp}`	`primes = {2, 3, 5, 7, 11, 13, 17, 19}`
		`' Note that primes will now have 8 elements`
		`' primes(0) to primes(7)`

Accessing an element (indexing)	`Identifier(IntExp)`	`Console.WriteLine($"Only even prime is {primes(0)}")`
		
		`' prints "Only even prime is 2"`

Updating an element	`Identifier(IntExp) = Exp`	`primes(5) = 17`
		
		`' position 5 within the array now`
		`' contains the value 17`

Arrays (continued)

Declaring a two-dimensional array	`Dim Indentifier(IntExp1, IntExp2) As DataType`
	`Dim Identifier(,) As DataType = {{value, …}, …}`
		`Dim board(7, 7) As String`
		`' board has 64 elements each of which is`
		`' a string. board(0, 0) to board(7, 7)`
		
		`Dim game(,) As String = {{"O", " ", "X"},`
		`{"X", "O", " "},`
		`{"O", "X", "O"}}`
		`' game has 9 elements, game(0, 0) to`
		`' game(2, 2)`
		
		`Dim prices(,) = {{2.56, 4.34},`
		`{11.96, 12.41}}`
		`' As DataType omitted`

Accessing an element in a two-dimensional array	`Identifier(IntExp, IntExp)`	`Dim game(,) As String = {{"O", " ", "X"},`
		`{"X", "O", "?"},`
		`{"O", "!", "X"}}`
		`Console.WriteLine($"Row 1 column 2:`
		`{game(1, 2)}")`
		
		`' prints "Row 2 Column 3: ?" as the`
		`' third column (with index 1) of the`
		`' second row (with index 2) in array is "?"`
		
		`' Note that game(2, 1) would be "!" and`
		`' that game(3, 1) would give an error`
		`' since there is no fourth row`

Arrays (continued)

Updating an element in a two- dimensional array	`Identifier(IntExp, IntExp) = Exp`	`game(1, 2) = "#"`
		
		`' game is now`
		`' {{"O", " ", "X"},`
		`'  {"X", "O", "#"},`
		`'  {"O", "!", "X"}}`

Array length	`Identifier.Length`	`Dim evens() As Integer = {2, 4, 6, 8, 10}`
		`evens.Length`
		`' evaluates to 5 using example above`
		
		`Dim board(7, 7) As String`
		`board.Length`
		`' evaluates to 64 using example above`
		
		`Dim costs(4, 3) As Single`
		`costs.GetLength(0)`
		`' evaluates to 5 and`
		`costs.GetLength(1)`
		`'evaluates to 4`

Further examples of array lengths	`Dim t(,) As Integer = {{1, 2}, {2, 4}, {3, 6}}`
	
	`Console.WriteLine($"The entire array t contains {t.Length} items")`
	`Console.WriteLine($"There are {t.GetLength(0)} sub arrays")`
	`Console.WriteLine($"Each sub array contains {t.GetLength(1)} items")`
	
	`' outputs:`
	`' The entire array t contains 6 items`
	`' There are 3 sub arrays`
	`' Each sub array contains 2 items`	

Arrays (continued)

FOR
(repeat the statements the number of times that there are elements in a array, each time giving Identifier the value of the next element in the array).
	`For Each Identifier In Array`
	`  ' indented statements here`
	`Next`
		`Dim a() As Integer = {15, 27, 19, 18}`
		`Dim sum As Integer = 0`
		`For Each age In a`
		`  sum = sum + age`
		`Next`
		`Dim mean As Single = sum / a.Length`
		`Console.WriteLine(mean)`
		`' calculates the total of the ages`
		`' held in the array and the mean`
		`' age and then outputs the mean (19.75)`

Records

Record declaration (exam questions will use this method to declare and use records)	`Structure RecordName`
	`  Dim field1 As DataType`
	`  Dim field2 As DataType`
	`  …`
	`  Sub New(param1 As DataType,`
	`  param2 As DataType,`
	`  …)`
	`  Me.field1 = param1`
	`  Me.field2 = param2`
	`  …`
	`  End Sub`
	`End Structure`
	
	`' The name of the structure`
	`' will start with a capital letter.`
		`Structure Car`
		`  Dim make As String`
		`  Dim model As String`
		`  Dim reg As String`
		
		`  Sub New(mk As String,`
		`  md As String,`
		`  rg As String)`
		`  Me.make = mk`
		`  Me.model = md`
		`  Me.reg = rg`
		`  End Sub`
		`End Structure`

Variable instantiation	`Dim varName As New`
	`StructureType(value, …)`
		`Dim myCar As New Car("Ford",`
		`  "Focus",`
		`  "EF56 ZFG")`

Assigning a value to a field in a record	`varName.field = Exp`	`myCar.model = "Fiesta"`
		
		`' The model field of the myCar`
		`' record is assigned the value "Fiesta".`

Accessing values of fields within records	`varName.field`	`Console.WriteLine(myCar.model)`
		
		`' Outputs the value stored in the`
		`' model field of the myCar record`

Subroutines

Note: subroutines that are defined using the keyword `Sub` are procedures, while those that are defined using the keyword `Function` are functions and use the `Return` statement to return a value.

Subroutine definition	`Sub Identifier(parameter As Type, …)`
	`  ' indented statements here`
	`End Sub`
	
	`Function Identifier(parameter As Type) As ReturnValueType`
	`  ' indented statements here`
	`  ' including at least one return`
	`  ' statement`
	`End Function`
		`Sub showAdd(a As Integer, b As Integer)`
		`  Dim result As Integer = a + b`
		`  Console.WriteLine(result)`
		`End Sub`
		
		`Sub sayHi()`
		`  Console.WriteLine("Hi")`
		`End Sub`
		
		`' Both of these subroutines are procedures`

Subroutine return value	`Return Exp`	`Function add(a As Single, b As Single) As Single`
		`  Dim result As Single = a + b`
		`  Return result`
		`End Function`
		
		`' This subroutine is a function`

Calling subroutines	`' Subroutines without a return value`
	
	`Identifier(parameters)`
	
	`' Subroutines with a return value`
	
	`Identifier = Identifier(parameters)`
		`' Subroutine without a return value`
		`showAdd(2, 3)`
		
		`' Subroutine with a return value`
		`Dim n1, n2 As Single`
		`Console.Write("First number? ")`
		`n1 = Console.ReadLine()`
		`Console.Write("Second number? ")`
		`n2 = Console.ReadLine()`
		`Dim answer As Single = add(n1, n2) * 6`

String handling

String length	`StringExp.Length`	`"computer science".Length` `' evaluates to 16 (includes space)`

Position of a character	`StringExp.IndexOf(CharExp)`	`"computer science".IndexOf("m")`
		`' evaluates to 2`
		
		`Dim t As String = "Algorithms"`
		`Dim s As Integer = t.IndexOf(" ")`
		`' s will have the value -1 since`
		`' t does not contain a space`

Substring (the substring runs from the character **at** the first `IntExp`, starting at 0, for the number of characters given by the second `IntExp`).	`StringExp.Substring(IntExp, IntExp)`	`Dim t As String = "Computer Programs"`
		`Dim p As String = t.Substring(9, 8)`
		
		`' prints the string "Programs"`
		`' If there is only one number then the`
		`' string from the position given to`
		`' the end of the string is returned`
		`Dim t As String = "Computers"`
		`Console.WriteLine(t.Substring(3))`
		`' prints "puters"`

Accessing a single character in a string (this treats a string as if it were an array)	`StringExp(IntExp)`	`Dim t As String = "Computers"`
		`Console.WriteLine(t(2))`
		`' prints "m"`

Concatenation	`StringExp + StringExp`	`Console.WriteLine("C" + "S")`
		`' prints the string "CS"`
		
		`' Note that no space is automatically`
		`' added between each string`

String and character conversion

Converting string to integer	`Convert.ToInt32(StringExp)`	`Convert.ToInt32("16")`
		
		`' evaluates to the integer 16`

Converting string to real	`Convert.ToSingle(StringExp)`	`Convert.ToSingle("16.3")`
		
		`' evaluates to the single (real) 16.3`

Converting integer to string	`Convert.ToString(IntExp)`	`Convert.ToString(16)`
		
		`' evaluates to the string "16"`

Converting real to string	`Convert.ToString(RealExp)`	`Convert.ToString(16.3)`
		
		`' evaluates to the string "16.3"`

Converting character to character code	`Asc(CharExp)`	`Asc("a")`
		
		`' evaluates to 97 using ASCII/Unicode`

Converting character code to character	`Chr(IntExp)`	`Chr(97)`
		
		`' evaluates to "a" using ASCII/Unicode`

Input/output

User input	`Console.ReadLine()`
	
	`' If you need a prompt then use`
	`' Console.Write(prompt) before`
	`' using Console.ReadLine`
		`Dim name As String = Console.ReadLine()`
		`' Console.ReadLine() returns a string,`
		`' so one of the conversion functions`
		`' above must be used to convert the`
		`' string to an integer (Convert.ToInt32)`
		`' or a single (Convert.ToSingle) unless`
		`' assigning the result to a variable with`
		`' one of those types.`
		
		`Console.Write("What is your name? ")`
		`Dim name As String = Console.ReadLine()`
		
		`Console.Write("How many cans? ")`
		`Dim q As Integer = Console.ReadLine()`
		`' Note` **`no`** `Convert.ToInt32`
		
		`Console.Write("How much? ")`
		`Dim price As Single = Console.ReadLine()`
		`' Note` **`no`** `Convert.ToSingle`

Input/output (continued)

Output	`Console.WriteLine(Exp)`
	
	`Console.Write(Exp)`
	`' Doesn't move to new line after`
	`' outputting Exp`
		`Dim a As Integer = 45;`
		`Console.Write(a);`
		`Dim g As Single = 23.45;`
		`Console.Write(g);`
		
		`' To output more than one thing, or to`
		`' include text at the same time`
		`' use $ strings, eg`
		
		`Dim qty As Integer = 15;`
		`Console.WriteLine($"Quantity {qty}");`
		
		`' outputs the string "Quantity 15"`
		
		`Console.WriteLine("Mary had ")`
		`Console.WriteLine("a little lamb")`
		`' will print the text over two lines, but`
		
		`Console.Write("Mary had ")`
		`Console.WriteLine("a little lamb")`
		`' will print the text on the same line`

Formatted outputs will be shown using interpolated strings (`$""` strings).	`$"…{identifier}…{identifier}…"`	`Dim name = "BT"`
		`Dim staff = 125000`
		
		`Console.WriteLine($"{name} has {staff} staff")`
		
		`' outputs "BT has 125000 staff"`

Random number generation

Random integer generation (The first `IntExp` is inclusive, the second `IntExp` is exclusive.)	`Dim Identifier As New Random()`
	
	`Dim varName As Integer = Identifier.Next(IntExp, IntExp)`
		`Dim rGen As New Random()`
		`Dim num1 As Integer = rGen.Next(3, 7)`
		
		`' generates an integer between 3 and 6`