# Unit 2 | 2.2.1 | Programming techniques

## Specification & learning objectives

| A Level | Specification point description |
|---|---|
| 2.2.1a | Programming constructs: sequence, iteration, branching |
| 2.2.1b | Recursion, how it can be used and compares to an iterative approach |
| 2.2.1c | Global and local variables |
| 2.2.1d | Modularity, functions and procedures, parameter passing by value and reference |
| 2.2.1e | Use of an IDE to develop/debug a program |
| 2.2.1f | Use of object-oriented techniques |

## Resources

PG Online textbook page ref: 288-326

Hodder textbook page ref: 37-48

CraignDave videos for SLR 23

Key question: What are the 3 basic programming constructs?

| | |
|---|---|
| Sequence | Code that is executed in order, one line at a time. |
| Iteration | Where a sequence of instructions is repeated multiple times |
| Selection | Choosing which lines of code to run or not to run, based on whether certain conditions have been met. |

## 3 basic programming constructs – Example.

```
'**********************************************
 'Procedure to display the game over message and get high score name
Sub game_over_message()
    Dim key As Char
    Console.SetCursorPosition(17, 12)
    Console.Write("G A M E  O V E R")
    System.Threading.Thread.Sleep(3000)
    Console.SetCursorPosition(14, 14)
    Console.Write("Press a key to continue")
    key = Console.ReadKey.KeyChar
End Sub

 '**********************************************
 'Procedure to compute and display the high score table
Sub high_scores()
    Dim counter1, counter2 As Integer
    Dim key As Char
    Dim isHighScore As Boolean
    Dim name As String

    'check if score is a high score
    isHighScore = False
    counter1 = 0

    'go through scores and check if score is a high score
    Do
        If (score > highScoreValue(counter1)) Then isHighScore = True
        If (isHighScore <> True) Then counter1 = counter1 + 1
    Loop Until (counter1 = 9) Or (isHighScore = True)
```

```
    'if it is a high score, ask for name and add to list
        If (isHighScore = True) Then
            Console.Clear()
            Console.SetCursorPosition(5, 5)
            Console.Write("N E W  H I G H  S C O R E")
            Console.SetCursorPosition(5, 7)
            Console.Write("Please enter your name: ")
            name = Console.ReadLine()
        End If

        ' move other scores down one place
        For counter2 = 9 To counter1 + 1 Step -1
            highScoreName(counter2) = highScoreName(counter2 - 1)
            highScoreValue(counter2) = highScoreValue(counter2 - 1)
        Next

        'add high score to list
        highScoreName(counter1) = name
        highScoreValue(counter1) = score

        'output high scores
        Console.Clear()
        Console.SetCursorPosition(5, 5)
        Console.Write("T O D A Y 'S  H I G H  S C O R E S")
        For counter1 = 0 To 9
            Console.SetCursorPosition(5, 7 + counter1)
            Console.Write(highScoreName(counter1))
            Console.SetCursorPosition(20, 7 + counter1)
            Console.Write(highScoreValue(counter1))
        Next
        Console.SetCursorPosition(5, 23)
        Console.Write("Press a key to play")
        key = Console.ReadKey.KeyChar
    End Sub
```

Sequence

Iteration (count controlled)

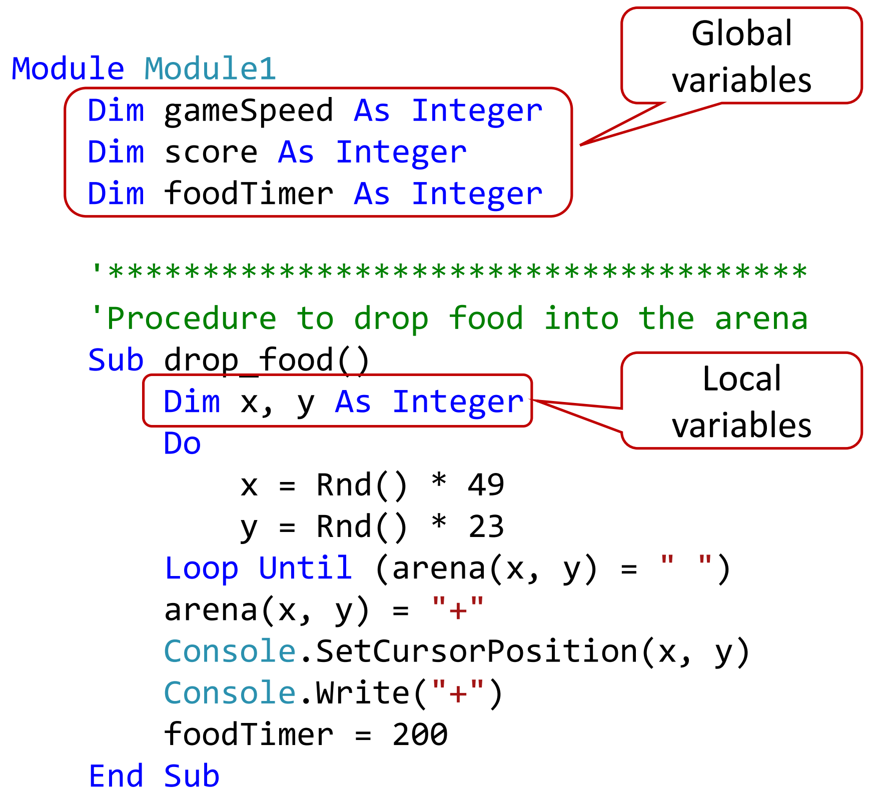
Iteration (condition controlled)

Branching

Key question: What is the difference between local and global variables and when should they be used?

1. What is a local variable?
*A variable which is defined and can only be used within one part of the program (normally a single function or procedure). It's scope is limited to the block of code in which it is declared.*

2. What is a global variable?
*A variable which is defined outside of any single procedure / function and can be used anywhere in the program. It's scope spans the entire program.*



```
Module Module1
    Dim gameSpeed As Integer
    Dim score As Integer
    Dim foodTimer As Integer

    '*************************************
    'Procedure to drop food into the arena
    Sub drop_food()
        Dim x, y As Integer
        Do
            x = Rnd() * 49
            y = Rnd() * 23
        Loop Until (arena(x, y) = " ")
        arena(x, y) = "+"
        Console.SetCursorPosition(x, y)
        Console.Write("+")
        foodTimer = 200
    End Sub
```

Key question: What is the difference between procedures and functions?

When writing programs, we should avoid long, repetitive code. Procedures and functions help to keep our programs simple and short, in a more modular manner.

## Procedure

**A procedure is a small section of a program that performs a specific task. Procedures can be used repeatedly throughout a program.**

## Function

**A function is also a small section of a program that performs a specific task that can be used repeatedly throughout a program, but the task is usually a calculation. Functions perform the task**

## Advantages to writing programs in a modular way.

- *Program easier to read, and easier to debug.*
- *Enables different programmers to work on different parts of the code.*
- *Enables routines to be reused in other programs.*
- *Reduces need for duplicated code.*
- *Functions can be stored in libraries for other programmers to use in their programs.*
- *Library functions are already tested.*
- *Library functions make use of another programmer's skill.*
- *Library functions can be written in other languages because they are already compiled.*

Key question: What is the difference between passing parameters by value and by reference?

*In a computer solution each procedure / function needs to have some data to work with.*

*This data is called the* ***parameters*** *if they are called at the same time as the procedure.*

*There are two ways of telling the system, what these* ***values*** *are.*

1. *One is to give the values as part of the statement, for example* ***RECTANGLE(3,4)****.*
    - *This is called passing the parameters* ***by value****.*

2. *The alternative is to give the locations where the values can be found, for example* ***RECTANGLE(x,y).***
    - *If the values x and y are defined as local variables then the parameters are still being passed by value because any changes made to them will not be allowed to affect their values outside the procedure.*
    - *However, if the values of x and y are global variables then any changes made during execution of the procedures will be carried back to the calling program when the procedure is exited and the parameters are said to have been passed* ***by reference****.*

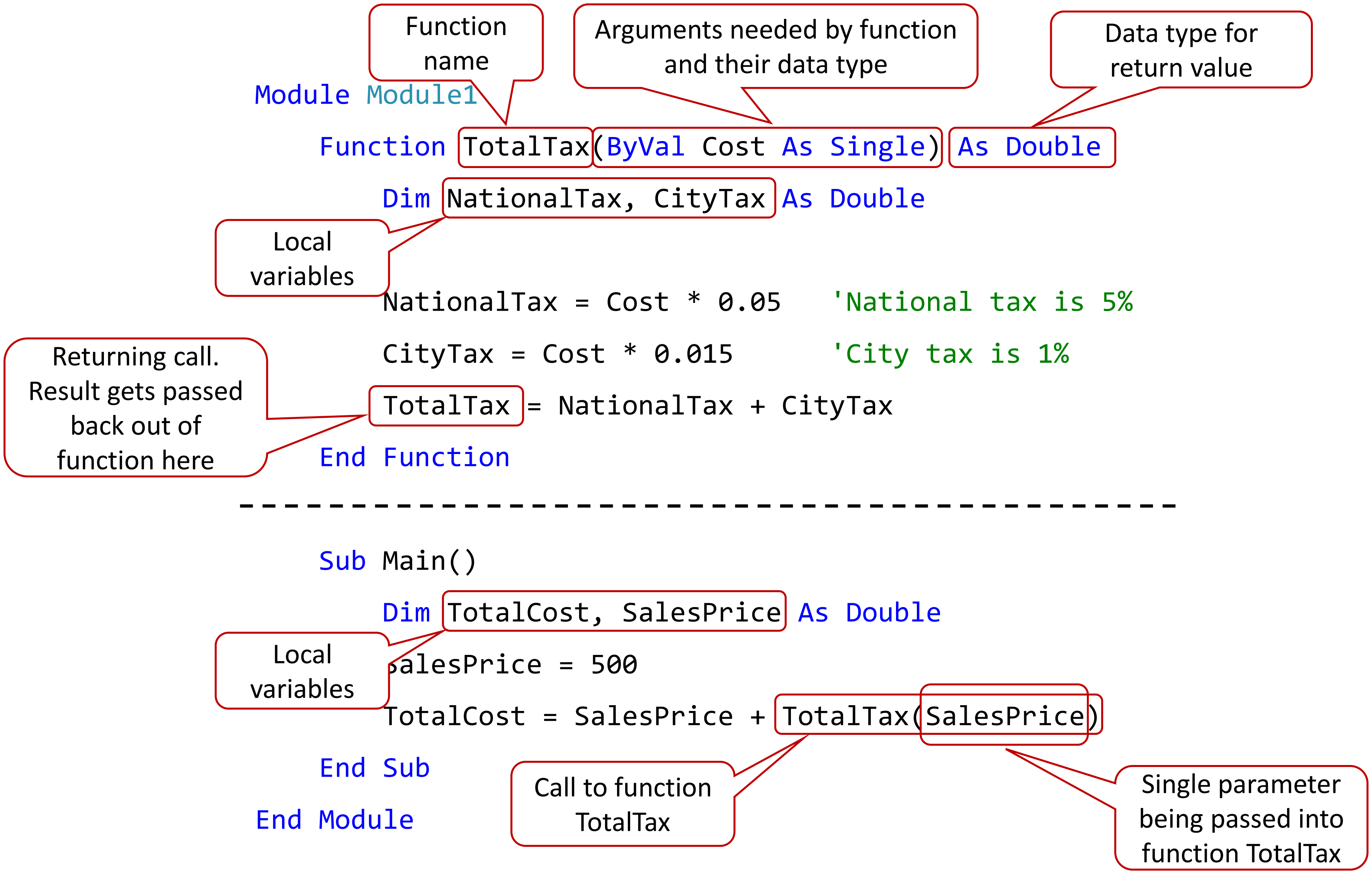
## Modularity, functions and procedures, parameter passing by value and reference – An example.

```
Module Module1
    Dim gameSpeed As Integer
    Dim score As Integer
    Dim foodTimer As Integer

    '**************************************
    'Procedure to drop food into the arena
    Sub drop_food()
        Dim x, y As Integer
        Do
            x = Rnd() * 49
            y = Rnd() * 23
        Loop Until (arena(x, y) = " ")
        arena(x, y) = "+"
        Console.SetCursorPosition(x, y)
        Console.Write("+")
        foodTimer = 200
    End Sub
```

By reference

By value

## Modularity, functions and procedures, parameter passing by value and reference – An example.


Function name
Arguments needed by function and their data type
Data type for return value
Local variables
Returning call. Result gets passed back out of function here
Local variables
Call to function TotalTax
Single parameter being passed into function TotalTax

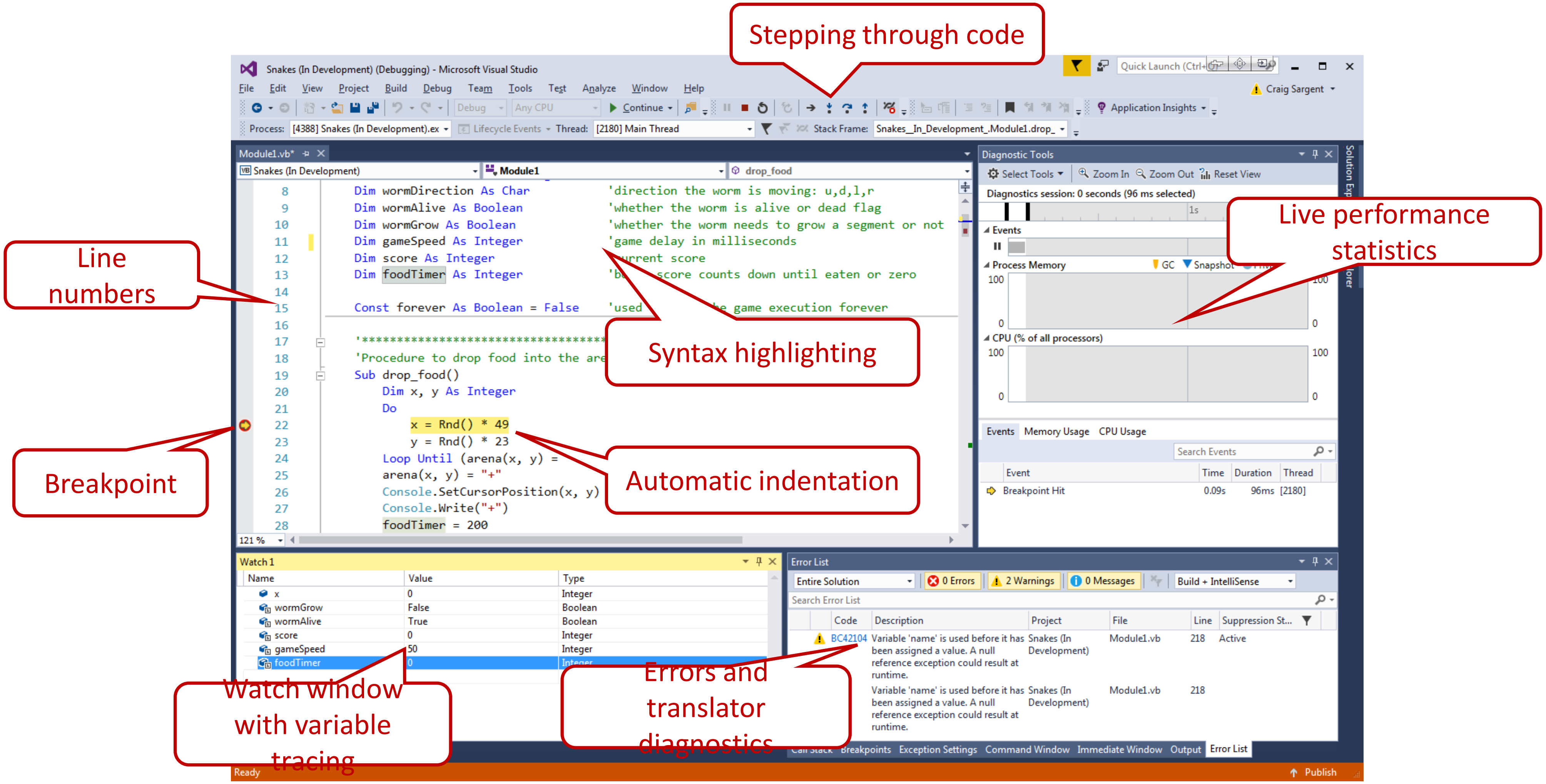
```
Module Module1
    Function TotalTax(ByVal Cost As Single) As Double
        Dim NationalTax, CityTax As Double

        NationalTax = Cost * 0.05   'National tax is 5%
        CityTax = Cost * 0.015      'City tax is 1%
        TotalTax = NationalTax + CityTax
    End Function
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
    Sub Main()
        Dim TotalCost, SalesPrice As Double
        SalesPrice = 500
        TotalCost = SalesPrice + TotalTax(SalesPrice)
    End Sub
End Module
```

Key question: What are the features of an IDE?

When developing programs we utilise an *Integrated Development Environment (IDE)*. This is a program that allows a programmer to write, develop and test code much more quickly than if the program were written, compiled and then run as an independent program.

A typical IDE consists of a user interface to allow the selection of objects, commands or help, a *debugger* which will highlight errors in the syntax of the code, and a *compiler* which will convert the source code and allow the programmer to test the program.

## Key question: What are the features of an IDE?


Stepping through code
Live performance statistics
Line numbers
Syntax highlighting
Breakpoint
Automatic indentation
Watch window with variable tracing
Errors and translator diagnostics
Snakes (In Development) (Debugging) - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test Analyze Window Help
Continue
Application Insights
Craig Sargent
Quick Launch (Ctrl+Q)
Diagnostic Tools
Select Tools Zoom In Zoom Out Reset View
Events Memory Usage CPU Usage
Call Stack Breakpoints Exception Settings Command Window Immediate Window Output Error List
Ready
Publish

Process: [4388] Snakes (In Development).ex Lifecycle Events Thread: [2180] Main Thread Stack Frame: Snakes_In_Development_.Module1.drop_

Module1.vb* — Snakes (In Development) — Module1 — drop_food

```
    Dim wormDirection As Char            'direction the worm is moving: u,d,l,r
    Dim wormAlive As Boolean             'whether the worm is alive or dead flag
    Dim wormGrow As Boolean              'whether the worm needs to grow a segment or not
    Dim gameSpeed As Integer             'game delay in milliseconds
    Dim score As Integer                 'urrent score
    Dim foodTimer As Integer             'b     score counts down until eaten or zero

    Const forever As Boolean = False     'used          he game execution forever

    '**********************************
    'Procedure to drop food into the are
    Sub drop_food()
        Dim x, y As Integer
        Do
            x = Rnd() * 49
            y = Rnd() * 23
        Loop Until (arena(x, y) =
        arena(x, y) = "+"
        Console.SetCursorPosition(x, y)
        Console.Write("+")
        foodTimer = 200
```

121 %

Diagnostics session: 0 seconds (96 ms selected)

Process Memory (GC, Snapshot): 100 / 0

CPU (% of all processors): 100 / 0

| Event | Time | Duration | Thread |
|---|---|---|---|
| Breakpoint Hit | 0.09s | 96ms | [2180] |

Watch 1

| Name | Value | Type |
|---|---|---|
| x | 0 | Integer |
| wormGrow | False | Boolean |
| wormAlive | True | Boolean |
| score | 0 | Integer |
| gameSpeed | 50 | Integer |
| foodTimer | 0 | Integer |

Error List — Entire Solution — 0 Errors — 2 Warnings — 0 Messages — Build + IntelliSense

| Code | Description | Project | File | Line | Suppression St... |
|---|---|---|---|---|---|
| BC42104 | Variable 'name' is used before it has been assigned a value. A null reference exception could result at runtime. | Snakes (In Development) | Module1.vb | 218 | Active |
| | Variable 'name' is used before it has been assigned a value. A null reference exception could result at runtime. | Snakes (In Development) | Module1.vb | 218 | |

Key question: What is recursion and how does it compare to using an iterative approach?

Recursion is the ability that a subroutine (see later sections) has to call on itself to complete its task until a condition is met. Recursive solutions can be harder to produce but can often lead to very elegant solutions to a problem.

A recursive solution has two parts: the *recursive* and the *limiter*. The recursive is the code that calls itself for another iteration and passes new variable values, whereas the limiter is what stops the code from creating an infinite loop.

Recursive methods act as a loop that calls on itself and runs every line of code using the result of the previous call in the current, and will pass the current result into the next call until the exit condition is met.

One of the best examples of recursion is factorials. Factorials are given by the mathematical formula:

```
n! = n × (n-1)!
```

This means a factorial is the product of a number times the factorial of the previous number. For example, 4!

```
4! = 4 × 3! = 3 × 2! = 2 × 1!
4! = 4 × 3 × 2 × 1 = 24
```

## Key question: How are objects constructed?

As procedural programs became more widespread, people started to notice that types of data and the procedures/functions associated with them tended to be grouped together; this was the start of what is known as *object-oriented programming* (OOP).

At their most basic level, object-oriented programming languages are concerned with the data for the *objects* you are trying to manipulate rather than the logic required to manipulate them.

An *object* is an *instantiation* (or an *instance*) of a class. Each object will have their *states* and *behaviours* that are local to that object. For example, a computer can have states (on, off) and behaviours (compute, load, shut down…). Objects are created using a *constructor* and a *reference* that has been assigned to a variable of the class type.

## Typical exam questions

1. What is meant by the term sequence? **[1]**

2. What is meant by the term iteration? **[1]**

3. What is meant by the term program branch? [**1**]

4. What is the difference between global and local variables? **[4]**

5. State two similarities and one difference between a function and a procedure. **[3]**

## Typical exam questions

6. What features does an integrated development environment (IDE) offer a programmer, to assist them in writing and debugging code? **[6]**

# Unit 2 | 2.2.1 | Programming techniques

Target: ☐ Overall grade: ☐

## Minimum expectations & learning outcomes

| | |
|---|---|
| ☐ | Terms 201-224 from your A Level Key Terminology should be included and formatted. |
| ☐ | You must include annotated code that shows sequence, iteration & branching. |
| ☐ | You must include annotated code that explains the difference between global and local variables. |
| ☐ | You must include some annotated code that shows the difference between functions & procedures. |
| ☐ | You must include some annotated code that shows the difference between passing by value and by reference. |
| ☐ | You must include an annotated screen shot of a development IDE, explaining some of the features. |
| ☐ | You must include an example of how recursion compares to iteration. |
| ☐ | You must include some annotated code that shows your understanding of object-oriented techniques. |
| ☐ | Answer the exam questions. |

## Feedback

| Breadth | Depth | Presentation | Understanding |
|---|---|---|---|
| ☐ All | ☐ Analysed | ☐ Excellent | ☐ Excellent |
| ☐ Most | ☐ Explained | ☐ Good | ☐ Good |
| ☐ Some | ☐ Described | ☐ Fair | ☐ Fair |
| ☐ Few | ☐ Identified | ☐ Poor | ☐ Poor |

## Comment & action required

# Unit 2 | 2.2.1 | Programming techniques

## Reflection & Revision checklist

| Confidence | Clarification |
|---|---|
| ☹ 😐 ☺ | Candidates need to be able to understand the constructs of sequence, iteration and branching. |
| ☹ 😐 ☺ | Candidates must be able to use these constructs independently of each other and combine them to produce a solution. These include the selection statements of if (include elseif and else) and select case statements. These include both condition-based iteration (e.g. while, repeat until) and count controlled iteration (e.g. for) – as well as how condition based can be used as count controlled iteration. |
| ☹ 😐 ☺ | Candidates need to be able to read code using these constructs, create code using these constructs and trace code (for example using a trace table). |
| ☹ 😐 ☺ | Candidates need to understand the use and need for variables in a program, and must understand the difference, benefits and drawbacks of both global and local variables. |
| ☹ 😐 ☺ | Candidates must be able to recognise where local and global variables are used, and the impact that these have on the program, for example the amount of memory used by the program. |
| ☹ 😐 ☺ | Candidates need to understand how a program using global variables can be changed to use local variables – and vice-versa. |
| ☹ 😐 ☺ | Candidates need to understand what is meant by modular code, and how this can be produced using functions and procedures. |
| ☹ 😐 ☺ | Candidates need to understand the differences between functions and procedures and how each is used within a program. |
| ☹ 😐 ☺ | Candidates need to be able to read, trace and write code using functions and procedures. |
| ☹ 😐 ☺ | Candidates need to understand the purpose and use of parameters within a program, and how they are used in functions and procedures. |
| ☹ 😐 ☺ | Candidates will need to be able to read, trace and write code that makes use of parameters. |
| ☹ 😐 ☺ | Candidates need to understand the difference between passing a parameter by value and by reference, they need to understand the benefits and drawbacks of each, recommending which should be used for a given situation. |
| ☹ 😐 ☺ | Candidates need to be able to read, trace and write code that makes use of parameters passed both by value and by reference. |
| ☹ 😐 ☺ | Candidates should have had experience of using an IDE to produce code. |
| ☹ 😐 ☺ | Candidates need to understand how an IDE can be used to produce code and understand the range of features and tools that are within an IDE that can be used to help produce and debug a program. |