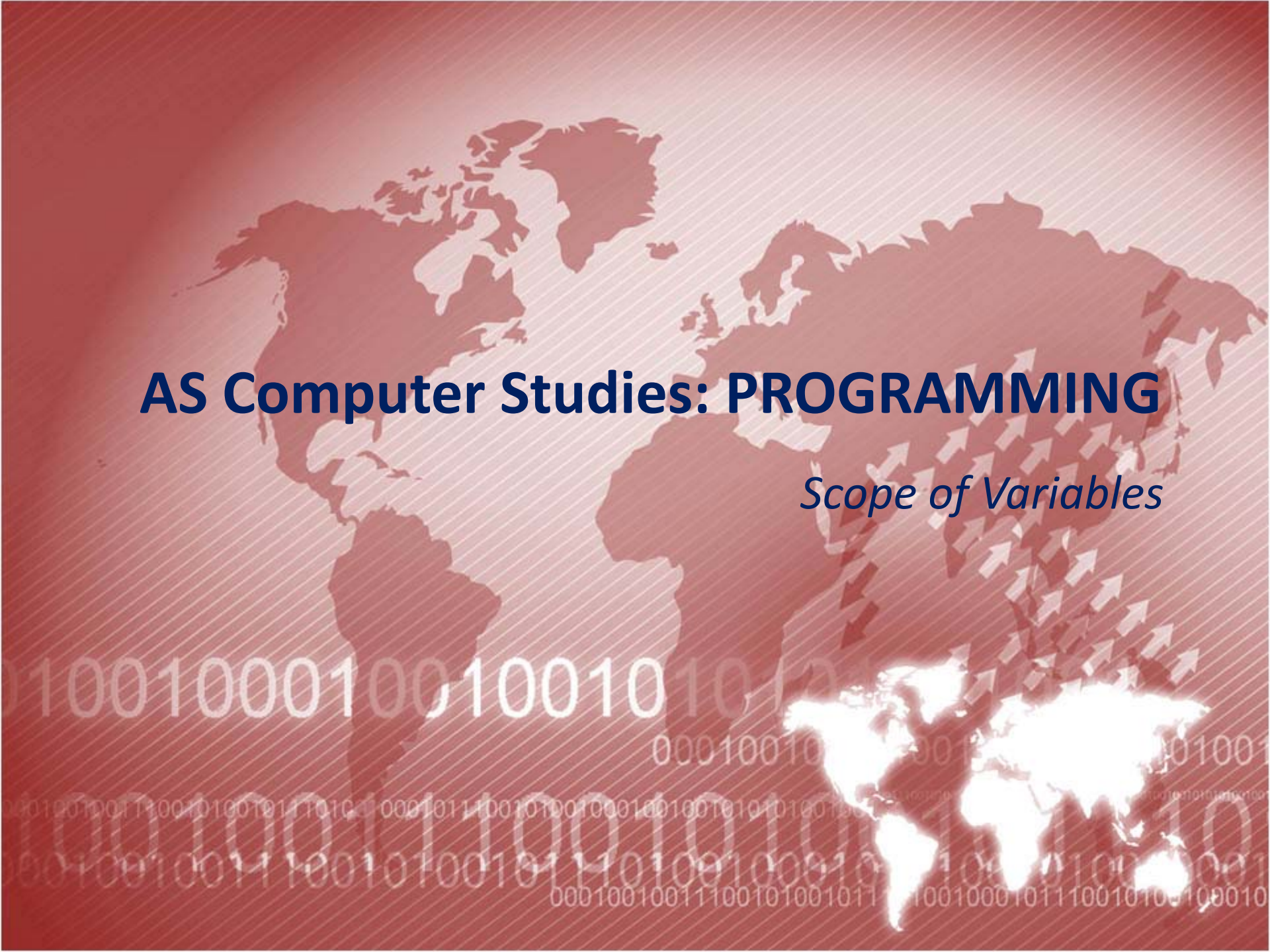

AS Computer Studies: PROGRAMMING

Scope of Variables

Starter

- What is wrong with the following piece of code:

```
Option Explicit Off

Dim num1 as string
Dim num2 as integer

num1 = 12
num2 = 19
num3 = num1+num2

console.writeline(num3)
```

Objectives

- Develop our knowledge of programming.
- Understand the concept of **variables** and **constants**
- Demonstrate the **use of variables and variable types**.
- Understand **scope of a variable**.

Recap: Variables

- **What is a variable?**
  - A variable is the name for a **storage location** allocated to a **piece of data**.
- **Why is it called a variable?**
  - The value of the data can change. This is why it is called a **variable**.
- **A variable is given a name so you refer to it in your code. This is called an .....**
  - identifier
- **Variables should be ..... with a starting value and then can be .....data.**
  - Initialised
  - assigned

Recap: Variable Types

Built-in	Description	Memory Allocated	Range	Example
Integer	Signed 32-bit number	4 bytes	-2,147,483,648 to 2,147,483,648	12000
Byte	Unsigned 8-bit number	1 byte	0 to 255	25
Single	Single precision floating point number	4 bytes	n/a (limited to 7 digits)	23.52
Double	Double Precision floating point number	8 bytes	n/a (limited to 15 digits)	25.6744322
Decimal	Used when minimising rounding errors, ideal for working with currency.	16 bytes	-	12.34
Char	A single Unicode character (e.g. a symbol)	1 byte	-	& $ £
String	A set of characters together.	As required – depends on length of string.	-	“Hello World”

When using variables, you need to think about:

-What data is being stored.

-What you need to do with the data.

-How much memory is taken up – this is IMPORTANT!

Recap: Working with Variables

- **Option Explicit On** – THIS IS VITAL and good practice.
- A variable is created by "**dimensioning**" a part of memory, giving it a name, and stating its type.
- Refer back to **HOUSE STYLE** when using variable names.
- Once declared, you can **ASSIGN** values (This is called **Assignment**). E.g. strStudentName = "Hello"

```
'force all variables to be declared
Option Explicit On

Module Module1

    Sub Main()
        Dim strStudentName As String
```

Recap: Declaring Multiple Variables

- You can declare **multiple variables** on the same line.
- It is considered good practice to only **declare variables of the same type on any one line**.
- Once declared, you can **ASSIGN values** (This is called Assignment). E.g. strStudentName = “Hello”

```
Sub Main()
    Dim strStudentName As String, strStudentClass As String, strTeacherName As String

    Dim intTestScore As Integer, intEssayScore As Integer

    Dim sngStudentAverage As Single, sngScorePercentage As Single
```

Variables vs. Constants

- We know that a variable is something which **changes**.
- What if we want to use a value which does not change within our code? For example VAT.
- Yes, we could put it in a variable, but what if we accidentally assign a value to it and overwrite the VAT rate?
- Instead we could use a **CONSTANT**.

Constant Declaration

- This is done in a similar way to variable declarations.
- Instead of the word "**Dim**," you use "**Const**"
- The biggest difference is that you assign a value at the same time as declaration.

E.g. Const VAT as Single = 17.5

- You can refer to it in the normal way.

Scope of a Variable

- Refers to the **part of the program code** that can use a variable.
- There are two main levels of scope:
  - **Local** - VB divides this level into Block and Procedure levels.
  - **Global** – VB divides this level into Module and Namespace levels.

- For now, at least, you only need to know about VB's **Procedure** and **Module** levels.

Scope of a Variable

Level	Description
(Local) Procedure scope	Available to / Visible by all code **within the procedure in which it is declared**
(Global) Module scope	Available to / Visible by all code within the module, class, or structure in which it is declared i.e. **defined outside procedure**.

How to: Break Long Statements in Code

- When writing your code, you might at times create lengthy statements that necessitate horizontal scrolling in the Code Editor.
- While this does not affect the way your code runs, it makes it **difficult for you** or anyone else to **read** the code as it appears on the monitor.
- In such cases, you should consider breaking the single long statement into several lines.
  - Use the line-continuation character, which is an underscore (_), at the point at which you want the line to break.
  - The underscore must be immediately preceded by a space and immediately followed by a line terminator (carriage return).
  - The next line must begin with & (followed by a space).

Variable Scope

- Specification:
  - This program will hopefully illustrate the concept of scope by declaring the same variable both (globally) module scope and (locally) procedure scope.
  - The (local) procedure contents of this variable will only be available locally and will not disturb the (global) module contents which is available throughout the form.

Variable Scope

- Create a new Console application named

  '**VariableScope**'.

- Declare two variables as below:

```
Module Module1
    Dim globalNumber As Integer = 200
    Sub Main()
        Dim localNumber As Integer = 15
```

- Notice BOTH variables have been initialised. It is GOOD PRACTICE to do that; in the exam you MAY lose marks if you don't.

Variable Scope

- To test that both variables are viewed correctly in the Main() sub:

```
Module Module1
    Dim globalNumber As Integer = 200
    Sub Main()
        Dim localNumber As Integer = 15

        Console.WriteLine("Local is: " & localNumber)
        Console.WriteLine("Global is: " & globalNumber)
        Console.ReadLine()

    End Sub
```

Variable Scope

- Now, we're going to create a new variable, but with the same name and different scope:

```
Module Module1
    Dim globalNumber As Integer = 200
    Sub Main()
        Dim localNumber As Integer = 15
        Dim globalNumber As Integer = 30

        Console.WriteLine("Local is: " & localNumber)
        Console.WriteLine("Global is: " & globalNumber)
        Console.ReadLine()

    End Sub
```

- Which of the two globalNumber values will be displayed?

Variable Scope

Conclusion:

- If you use THE SAME NAME for two variables with different scopes, the local one overrides the global

Variable Scope

- A **global variable** is declared outside any procedure but within the Module's Declarations section.
  - it is available to all procedures within this Module
- A **local variable** is declared inside a procedure, locally.
- If the local and global variables have the same name, the value stored in the local variable takes priority during the execution of that procedure; once that is finished the variable will revert to its global contents.

Variable Scope

```
Module Module1
    Dim globalNumber As Integer = 200
    Sub Main()
        Dim localNumber As Integer = 15
        Dim globalNumber As Integer = 30

        Console.WriteLine("Local is: " & localNumber)
        Console.WriteLine("Global is: " & globalNumber)
        Console.ReadLine()

        newProcedure()
        Console.ReadLine()

    End Sub

    Sub newProcedure()
        'Console.WriteLine("Local is: " & localNumber)
        Console.WriteLine("Global is: " & globalNumber)
        Console.ReadLine()
    End Sub

End Module
```

This is a new procedure to check what is the contents of the global variable OUTSIDE the Sub Main() procedure.

And this is where the new procedure is 'called', executed.

Narrow Scope

- It is good programming practice, and efficient, to **declare variables with as narrow a scope as possible**.
- Certain errors/problems can occur **if you don't**:
  - ***Name Conflict*** – several different procedures can use the same variable name/identifier. As long as each instance is declared as a (local) procedure variable, each procedure recognises only its own version of the variable.
  - ***Memory Consumption*** – (Local) Procedure variables consume memory only while their procedure is running. Their memory is released when the procedure finishes. (Global) Module variables obviously use memory until the module finishes i.e. longer.

Tasks

FIRST:

Programming Guide (blue booklet) - Read & take notes p10-19

Programming Tasks booklet – work through the following programming tasks.

- Task 4 – Understanding Code Activities – p9-10
- Task 5 – Extended String and Number activities – p11-14
- Task 6 – String and Number Functions recap – p14-16
- Task 7 – Further Practice of Inputs and Outputs – p17

In all tasks **observe the scope of the variables used**!

Plenary

- How do you declare a variable?
- How do you declare a constant?
- What is a constant and why would you use it?
- How many variable TYPES can you name?
- What is the difference between a GLOBAL variable and a LOCAL variable?

Objectives

- Develop our knowledge of programming.
- Understand the concept of **variables** and **constants**
- Demonstrate the **use of variables and variable types**.
- Understand **scope of a variable**.