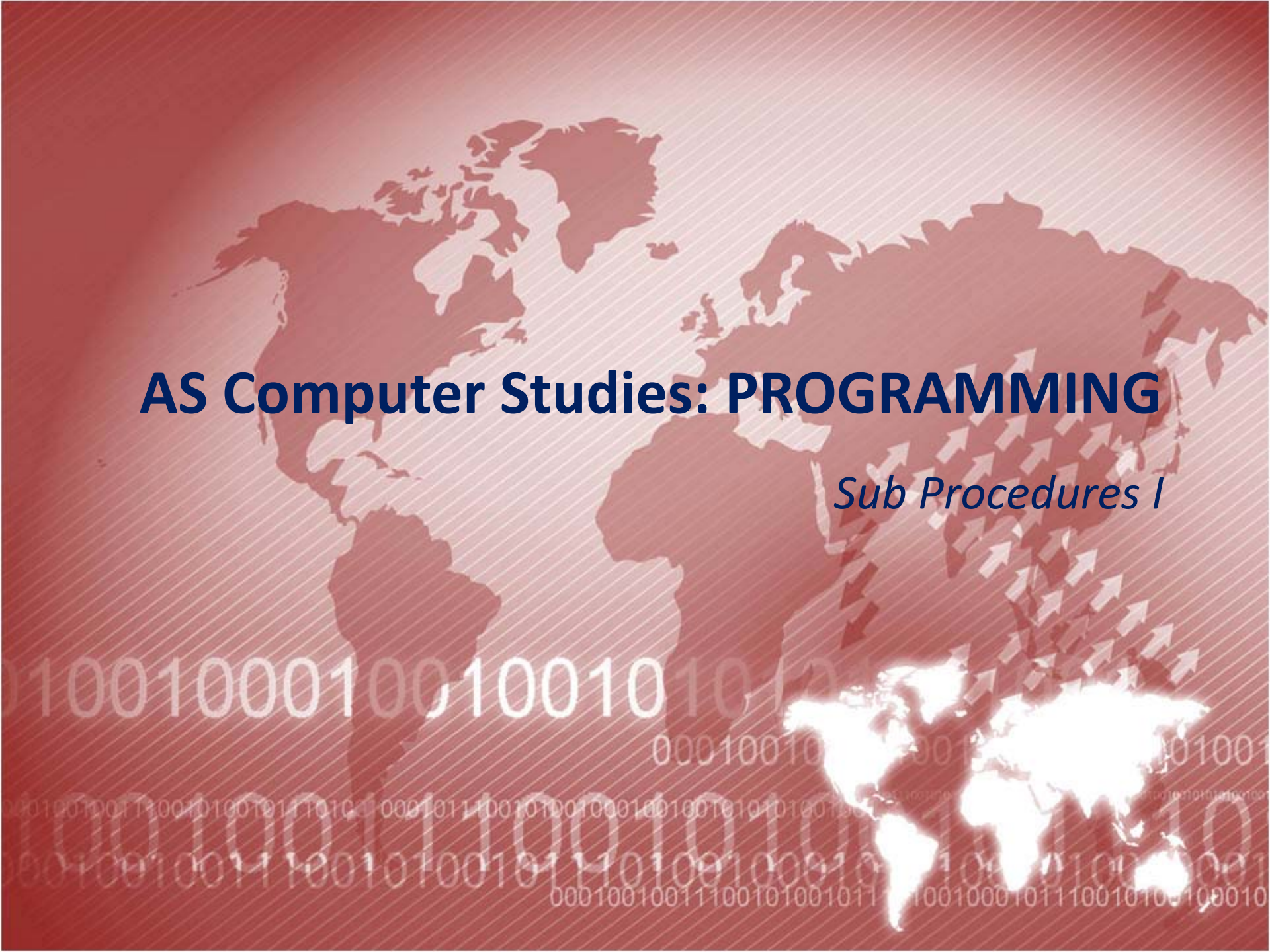

AS Computer Studies: PROGRAMMING

Sub Procedures I

Starter

- Identify the **separate tasks** to be performed in the problem below (break it down into numbered sections).

  - A program is needed to prompt a user for their name and monthly wage. From this it should then calculate their yearly salary and yearly pension deductions (8% of their yearly salary). This summary should be written to file and displayed on screen.

- **TASKS**:
  1. Obtain Name and Monthly Wage
  2. Calculate Salary and Deductions.
  3. Write name, monthly wage, salary and deductions to file
  4. Display name, monthly wage, salary and deductions on screen

Objectives

- Understand the importance of **modular programming**.
- Know the **role of procedures** within programming.
- Understand the concept of **parameters**; **ByRef** and **ByVal**
- Use procedures within your programming.

Modular Programming

- **Big problem**

  -> break it down into **smaller chunks**, or **sub-tasks**

  -> makes things easier to follow / makes the problem easier to understand.

- As you add more code to a program, it becomes longer and longer.
- If you wanted to **re-do** things over and over again.
  - E.g. calculate an average, display it on screen, every time a different set of numbers was entered (at different times – so a loop is no good).
  - Normally, you would need the same code every time you wanted to do this
  - This causes **excessive** amounts of the same code to be added to your program, repeatedly.

Why Go Modular

- Big problem → bunch of smaller problems. Why is this GOOD?
  - smaller problems are **easier to code** than the larger problem of which they are a part;
  - a **team of people can work on the smaller problems**, each member concentrating on one, or just a few, of them.
- A **procedure** is the code equivalent of a solution to one of the smaller problems
- **Code can be re-used** over and over again
- Allows you to focus **first** on **what** needs to be done, rather than **how** to do it.
- Procedures make a program **easy to understand** and **easier to maintain**.

What is a procedure?

- Every program you have written so far is one!
- A procedure, then, is a **small *part* of a program**
  - which has its own ***name***
  - performs a ***specific job*** (usually a **single task**, or a small group of related tasks)
  - and is executed when it is "***called***".
- **Sub Main** – is the one you have written everything in so far. In VB it is still a 'subprocedure'.
- You will have noticed your code has gradually become longer and longer – this is not good! **Everything in one sub is BAD**.

Declaring a Procedure

```
Sub Name()
    statement(s)
End Sub
```

It doesn't matter where you place new Procedures (above or below the existing Sub Main).

- The word **Sub** is essential, and *must* appear exactly like that.
- ***Name*** is any legal VB identifier and should be chosen carefully to indicate what the procedure does.
- The ***parentheses*** () are also obligatory - you cannot omit them.
- ***Statement(s)*** - the actual code that you write to perform the required task.
- **End Sub** (two separate words) is also essential.

Writing a Procedure

```
Sub AverageCalc()
    'sub to calculate average of 2 numbers
    Dim intNum1 As Integer, intNum2 As Integer
    Dim average As Single


    'get numbers
    Console.WriteLine("Enter number 1:")
    intNum1 = Console.ReadLine()

    Console.WriteLine("Enter number 2:")
    intNum2 = Console.ReadLine()


    'calculate average
    average = (intNum1 + intNum2) / 2


    'output average
    Console.WriteLine("Average equals: {0}", average)

    Console.ReadLine()

End Sub
```

- You should always ensure that the **first thing** in the code for any procedure you write is a ***comment*** which explains what the procedure does
- In the example here, we have declared a procedure called **AverageCalc**.
- It calculates the average and displays it on screen.
- Notice how it is just doing **ONE** job – that is calculating the average and displaying it.

Using a Procedure

Call *Name*

will result in the code for the procedure *Name* being executed.

- In Visual Studio 2008 “call” is not needed but it does make good sense to use it.
- We can call the procedure from anywhere (within our Main procedure or within other procedures). For example:

```
Sub Main()
    'calculate average
    Call AverageCalc()
End Sub
```

- You try it!

Tasks

Complete the questions in **Task 17** (page 36).

We will go through them together.

THEN:

Have a go at **Task 18**:

Questions 1 & 2

Objectives Review

- Understand the importance of **modular programming**.
- Know the **role of procedures** within programming.
- Understand the concept of **parameters**; **ByRef** and **ByVal**
- Use procedures within your programming.

Plenary

- What is meant by modular programming?
- Why is it important to break down a program into smaller parts – what are the benefits?
- What is a procedure?
- How do you declare them?
- How do you use them?