Access VBA Made Easy

Data Types, Variables, Constants, Operators

03

www.accessallinone.com

This guide was prepared for AccessAllInOne.com by:
Robert Austin

This is one of a series of guides pertaining to the use of Microsoft Access.

Contents

Data Types, Variables, Constants and Operators

Variables

When writing code in V.B.A. we often need to do calculations based on values that can change. An example would be working out the area of a circle. Take a look at the code below to see how we have used variables to for values that we do not know when the code starts running.

```
Public Sub AreaOfCircle()

Dim d As Double
Dim radius As Double
Dim measure As String
   'This is where we declare our variables

   'They are variables because when the sub-procedure runs,
   'we do not know their values

Const PI = 3.14159265359
   'We have declared pi as a constant and not a variable
   'because its value never changes...

radius = InputBox("Please enter a radius")
   'We obtain the value of radius using an input box

measure = InputBox("Please enter cms or inches")
   'We obtain the value of measure using an input box

d = PI * (radius * radius)
   'We work out the area of the circle and assign it to d

MsgBox "The area of your circle is " & Round(d, 4) & " " & measure


End Sub
```

Figure 3.1

Declaring variables

Variable declaration is the act of telling VBA the name of your variables before you actually use them. You should always declare the variables you will use as soon as logically possible, which usually means at the top of your function or sub procedure. You should also state the data type of your variables. In the above code we are telling V.B.A. that we would like to declare a variable called Radius which has a data type double.

It is a good idea and standard practice to declare variables and data types

Dim

To declare a variable in VBA we use the keyword **Dim** (which stands for dimension).

```
Dim Age As Integer       ‘ VBA makes space for Age, of type Integer
Dim Name as string       ‘ VBA makes space for Name, of type String
```

Figure 3.2

The name of the variable must follow Dim and the type of the variable *should follow* with the keywords "As" and then the type.

***Note:** If, at the top of the module, you include the "Option Explicit" statement, you must let V.B.A. know the data type that you will be assigning the variable (e.g. as Integer, as Double, as String). If, however, you omit "Option Explicit" at the top of the module, you don't have to let V.B.A. know what type of data you are going to use. V.B.A. will assume that you are using the data type "Variant" and proceed accordingly. Always use "Option Explicit"!!!*

Restrictions on naming variables

The names we can use for variables must conform to a small set of rules:

1. They must begin with a letter or an underscore (_).
2. They must end with a number or letter.
3. They may contain any sequence of numbers or letters or underscores (_).
4. They may contain upper or lower case letters.
5. They must not be one of VBA's keywords.

The compiler will automatically tell you if a variable is illegally named and will not execute unless variables are valid.

```
Dim a as String          ‘ is a valid variable name
Dim b_ as String         ‘ is a valid variable name
Dim _b as String         ‘ variable names must start with a letter
Dim 2b as String         ‘ variable names must start with a letter
Dim c1 as String         ‘ is a valid variable name
Dim d12 as String        ‘ is a valid variable name
Dim e_e1 as String       ‘ is a valid variable name
Dim f! as String         ‘ punctuation not allowed in variable names
Dim g as String          ‘ is a valid variable name

Dim dim as String        ‘ is not valid – "Dim" is a keyword
Dim string as String     ‘ is not valid – "String" is a keyword
Dim number as String     ‘ number is not a keyword so this is valid
```

Figure 3.3

Naming Conventions

A naming convention is a way of naming variables which enables us to easily understand both the data type of the variable and the reason for its existence. There are a couple of rules to follow when naming variables.

- Use meaningful variable names – make your variables mean something. Zzxd isn’t meaningful, but fileNotFound means something to a human, even though it doesn’t affect the computer or VBA in any way.
- Use camelCase for variables – that is, for every word in your variable name make the first letter of the first word lower-case, and the remaining letters upper-case. thisIsCamelCase.
- Use UPPER_CASE for constants (see below)– when you declare a constant, the name of that constant is usually capitalised. This means nothing to the compiler but means everything to you (we look at constants later on in this unit).

Another convention is to use up to 3 small letters before the variable name to indicate the data type.

- iMyNumber would be of type Integer
- dblMyOtherNumber would be of type Double
- strText would be of type String

Constants

Constants differ from variables in that their value does not change after they have been declared. This is how we code with constants:

```
Dim a as String         ‘ is a regular variable declaration
Const B = 1             ‘ declare the constant B with a value of 1
Const DATABASE_NAME = “accdb_firsttime”
                        ‘ new constant called DATABASE_NAME
```

Figure 3.4

You may have noticed that constants are not given a data type; this is because VBA makes some intuitive assumptions about the data. For example, any text surrounded by double quotation marks is a String; any number without a decimal point will fit into a Long; any decimal number will fit into a Double, and any True or False values fit into a Boolean value (True or False). This is the same logic VBA will take if you were not to define your data type on a variable using Dim.

Variable Scope

When you declare a variable in your program you also implicitly determine which parts of your code can access it. In VBA there are three types of declaration that affect the scope of a variable; Procedure, Module and Public.

```
    'Global Declaration
Public SalesTax As Double

    'Module Level Declaration
Private ItemPrice As Double

Sub getPriceIncVAT()
    Dim PriceIncVAT As Double
    Call getSalesTax
    Call getItemPrice
    PriceIncVAT = ItemPrice + (ItemPrice * SalesTax)
    MsgBox ("The price of the item including VAT is: $" & PriceIncVAT)
End Sub

Sub getSalesTax()
    SalesTax = InputBox("What is the tax? (20%=0.2)")
End Sub

Sub getItemPrice()
    ItemPrice = InputBox("What is the price of the item?")
End Sub
```

Figure 3.5

Procedure Level Scope

Procedure level scope means that a variable is recognised only within that procedure. In the above code, the variable **PriceIncVAT** has a procedure level scope and is only recognised within the sub-procedure **getPriceIncVAT**. To achieve this we use the dim or static keywords and declare the variable *inside* the sub-procedure we wish to recognise it.

Module Level Scope

Module level scope means that a variable can be recognised by any sub procedures within the module. **ItemPrice** has a module level scope and this is reflected in the fact that the variable is recognised in **getItemPric**e and **getPriceIncVAT**. To give a variable a module scope we declare it at the top of the module and use the private keyword (private means it is only available to sub-procedures within the module it is declared in).

Public Level Scope (also known as Global Scope)

Public level scope means that a variable is recognised by every sub-procedure and function within the active application. (In the above code **SalesTax** has a public level scope.) This can be useful for variables that should be consistent throughout the application (e.g. **SalesTax** shouldn't be 20% in one sub procedure and 15% in another). It is convention to create a module with a name like "**Globals**" where it is possible to keep all of the public variables in one place where they can easily be maintained and modified as required.

Arithmetic Operators

Like all languages VBA has a set of operators for working on Integer (whole) and floating-point (decimal) numbers. The table below demonstrates all 9 of them. VBA also offers many other operations built in as commands in the language.

```

Sub ArithmeticOperators()

Dim a1 As Integer
Dim b1 As Integer
Dim c1 As Integer

Dim a2 As Double
Dim b2 As Double
Dim c2 As Double

' + addition
a1 = 10
b1 = 20
c1 = a1 + b2   ' c1 = 30

' - subtract
a2 = 9.8
b2 = 5.3
c2 = a2 - b2   ' c2 = 4.5

' * multiplication
a1 = 9
b1 = 8
c1 = a1 * b1 ' c1 = 72

' / division floating-point
a2 = 120.5
b2 = 8.12
c2 = a2 / b2 ' c2 = 14.8399014778325

' \ division integers
a1 = 256
b1 = 8
c1 = a1 \ b1 ' c1 = 32

' mod - returns the remainder of a division
a1 = 100 Mod 3    ' a1 = 1
a2 = 100 Mod 3.1  ' a2 = 1, mod only returns whole numbers


' ^ powers
a1 = 2 ^ 2       ' a1 = 4
b1 = 3 ^ 3 ^ 3   ' b1 = 19683
End Sub
```

Figure 3.6

Common Errors

Not using the Option Explicit Statement

The option explicit statement is useful because it ensures that we *must* declare our variables. (As mentioned, VBA assumes the data type variant for all non-declared variables when option explicit isn't used).

The Option Explicit statement should go at the top of the application before any code has been written.

Data Types

When working with data in V.B.A. it is important that we don't try to add 15 to the word "Hello" or try to divide 07/02/12 by 53 as V.B.A. will not be able to make sense of these calculations (and, frankly, neither can we). In order to ensure the integrity of the data that we make calculations upon we are required to use data types.

Data types are essentially restrictions that are placed on data that are manipulated in the V.B.A. environment. These restrictions allow us to tell V.B.A. that we are creating a variable and that variable will only accept a specific type of data. An example of this would be the integer data type. Integer is just a fancy way of saying whole number and if we declare a data type as an integer it will only accept a whole number.

```
Dim HouseNumber as integer ' This variable will only accept integers
```

Figure 3.7

Here we have created a variable called HouseNumber and informed V.B.A. that we wish this variable to be of type integer. This means that we will only be able to assign a whole number to it. Ergo…

```
Dim HouseNumber as integer

HouseNumber = 5
```

Figure 3.8

…would be a perfectly acceptable assignment statement whilst…

```
Dim HouseNumber as integer

HouseNumber="Car"
```

Figure 3.9

…would not.

There is also another reason for the existence of data types. Different data types take up different amounts of memory depending on how complex they are. An example of this would be the integer data type vs the double data type.

We can use the double data type to store non-integer numbers. So, for example, whereas we couldn't accurately store the number 2.531 as an integer (it would round it up to 3) we could use the double data type for this value. The double data type, though, uses twice as much memory as the integer data type (8 bytes vs. 4 bytes) and, although not a big drain on the memory, with large applications that use many variables, it can and will affect performance if the correct data are not assigned the correct data type. So, if you need to store integers, use the integer data type; if you need to store text strings, use the string data type. And so on.

Data types and definition

Firstly a word on VBA variable names; a variable may be named anything you wish as long as it conforms to VBA's naming rules. Variable names must start with a letter, may contain number characters or underscores (_) but that's it! Punctuation marks are not allowed. Also unlike other languages VBA is **case-insensitive**! This is important to understand and is demonstrated below.

Finally, there are some keywords that cannot be used as variable names.

```
Dim a as String            ‘ a valid variable name
Dim b_ as String           ‘ a valid variable name
Dim _b as String           ‘ variable names must start with a letter
Dim 2b as String           ‘ variable names must start with a letter
Dim c1 as String           ‘ a valid variable name
Dim d12 as String          ‘ a valid variable name
Dim e_e1 as String         ‘ a valid variable name
Dim f! as String           ‘ punctuation not allowed in variable names
Dim g as String            ‘ a valid variable name
Dim G as String            ‘ an invalid variable name.  VBA is case-
                           ‘   insensitive, variables also cannot be
                           ‘   declared more than once in a code block
Dim aVariableName as String   ‘ a valid variable name
Dim a_Variable_Name as String ‘ a valid variable name
Dim HELLOWORLD as String      ‘ a valid variable name

Dim dim as String          ‘ variable name is invalid as Dim is a keyword
```

Figure 3.10

Boolean - (Yes/No)

A variable of type Boolean is the simplest possible data type available in VBA. It can only be set to 0 or -1. These are often thought of as *states* and correspond to Access's Yes/No fields. In VBA you can assign a Boolean variable to True (-1) or False (0) or the numbers indicated in the brackets.

Notice we used capitalised words for True and False, which is because they are VBA keywords and you cannot name a variable a Keyword.

Notice I used capitalised words for True and False, which is because they are VBA keywords and you cannot call a variable a Keyword.

```
Sub trueOrFalse()
   Dim foo As Boolean
   Dim bar As Boolean
   foo = True        ' foo holds the value True
   bar = False       ' bar holds the value False
End Sub
```

Figure 3.11

Integer

At the beginning of the post we said that we have to tell the computer what type of data to expect before we can work on it. An Integer is another number data type, but its value must be between -32,768 and 32,767, and it must be a whole number, that is to say, it mustn't contain decimal places. If you or your users try to save a decimal value (eg 2.5) to an integer variable, VBA will round the decimal value up or down to fit into an Integer data-type.

```
Sub IntegerDataType()
   Dim foo As Integer
   Dim bar As Integer
   Dim oof As Integer
   foo = 12345      ' foo is assigned the value 12,345
   bar = 2.5        ' bar is assigned the value 3 as VBA rounds it up
   bar = 2.4        ' bar is assigned the value 3 as VBA rounds it down
   foo = 32768      ' causes an overflow error as 32,768 is too big
End Sub
```

Figure 3.12

Long

Long is another number type and works just like Integer except it can hold a much larger range; Any number between -2,147,483,648 and +2,147,483,647.

```
Sub LongDataType()
   Dim foo As Long
   foo = 74345       ' foo is a variable assigned the value 74,345
End Sub
```

Figure 3.13

Single

Single is the smaller of the two "floating point" data types. Singles can represent any decimal number between -3.4028235E+38 through 1.401298E-45 for negative numbers and 1.401298E-45 through 3.4028235E+38 for positive numbers. Put more simply, the single data type has a decimal point in it.

```
Sub SingleDataType()
   Dim foo As Single
   Dim bar As Single
   foo = 1.1         ' foo keeps the .1 decimal part
   bar = -20.2       ' bar also keep the decimal part
   foo = foo * bar   ' foo equals -22.2200008392334
End Sub
```

Figure 3.14

Double

This is a “floating point” number as well and range in value from -1.79769313486231570E+308 through -4.94065645841246544E-324 for negative values and from 4.94065645841246544E-324 through 1.79769313486231570E+308 for positive values.

```
Sub DoubleDataType()
   Dim foo As Double
   Dim bar As Double
   foo = 1.1       ' foo keeps the .1 decimal part
   bar = -20.2     ' bar also keep the decimal part
   foo = foo * bar ' foo equals -22.2200008392334
End Sub
```

Figure 3.15

Currency

This data-type is a third “floating-point data” type in disguise. It’s a Single which has been engineered to represent behaviours typical of currencies. In particular it rounds off numbers to four decimal places. See the Figure below:

```
Sub CurrencyDataType()
   Dim bar As Single
   Dim foo As Currency
   bar = 1.1234567     ' this is the Single
   foo = bar           ' add the Single to the Currency
   MsgBox bar     ' bar contains 1.1234567
   MsgBox foo     ' foo contains 1.1235. Notice that the 4th digit
                   '   has been rounded up to 5
End Sub
```

Figure 3.16

Date

The Date data type is used to perform operations that involve dates AND times. In VBA there are several functions that operate on date variables which perform date and time calculations. It is important to know that date and time operations can be quite complicated and to help ease your burden you can use VBA’s DateTime object which encapsulates a lot of the difficulty of working with dates and time and can make them *a little less of a headache* to deal with. Date data types are the most complicated of all the data types to work with.

Here are a few operations we can do with date data types.

```
Sub DateDataTypes()
   Dim bar As Date
   Dim foo As Date
   bar = #11/15/1978#           ' bar set to this date but has no time
   foo = #12/10/2012 11:37:00 PM#  ' foo is set to this date and time
   bar = #1:00:09 AM#              ' bar is 1 hour and 9 seconds
   foo = #9:00:00 PM#              ' foo is 9PM
   foo = foo + bar                 ' foo is now 22:00:09
   MsgBox foo
   foo = foo - bar                 ' foo is back to 9PM
   MsgBox foo
End Sub
```

Figure 3.17

String

A String is any set of characters that are surrounded by double-quotation marks. For example "dog" is a String that contains three characters. Strings are very important to us as they can contain human language, and in fact contain almost any data we want, even numbers and punctuation marks. Strings are very versatile and you will use them extensively in your code. Often when you ask your users for information you will first store their input in a String before actually using the data provided; in this way Strings are often thought of as a safe data type.

Below are some Figures of Strings in action.

```
Sub StringDataTypes()
   Dim bar As String
   Dim foo As String
   Dim foobar As String

   bar = "Hello"                  ' bar now contains "Hello"
   foo = "world!"                 ' foo contains "world!"
   foobar = bar & " " & foo       ' foobar now contains "Hello world!"
 ' notice that foobar has a +" "+ this means a SPACE character has been
   ' inserted into the String, without it foobar would contain
"Helloworld!"

   foobar = bar + " " + foo    ' This Figure also shows that you can add
   ' Strings together (but you cannot subtract!)

   foo = "H" & "E" & "L" & "P"    ' foo now contains "HELP"
   bar = foo & foo                ' bar now contains "HELPHELP"
End Sub
```

Figure 3.18

As stated above, when you collect input from a user you will usually collect it into a String. But be careful not to confuse String with Number data types. For example:

```
Sub Confusion()
   Dim bar, foo As String
   Dim foobar As String

   foo = "12.5"          ' user inputs "12.5"
   bar = "6.3"           ' user inputs "6.3"
   foobar = foo * bar    ' we multiple 12.5 and 6.3
   Debug.Print foobar    ' print the result - 0
                        ' It's ZERO!
  ' Remember foo and bar are STRING data types,
  'so multiplying foo and bar as above is like
  'saying "aaa" * "bbb" = 11  ? It doesn't make sense.
  'But we collect data in a String because a String
  'can accept all user input, even if they 12  put a
  'punctuation mark in there.

  foo = "12.5.2"        ' user made a mistake
  bar = "ifvgj212m"     ' cat walks across the keyboard

  ' When collecting user input the data held in a String
  'can be tested for accuracy and correctness before we
  'load it into an Integer. If the user has not entered
```

```
    'data correctly we ignore or display a useful message
    'like "Error"...
End Sub
```

Figure 3.19

Variant

A variant is a special type which can contain any of the data types mentioned above (along with some others).

When a value is assigned to the variant data type the variable *mutates* into the type of the data assigned to it, and in some cases VBA can "detect" the type of data being passed and automatically assign a "correct" data type. This is useful for collecting data from users and also for writing procedures and functions for which you want to be able to call with a variable of any type.

```
Sub VariantDataType()
   Dim bar As Variant
   Dim foo As Variant
   Dim foobar As Variant
   bar = 1              ' bar is now an Integer
   foo = "oi!"          ' foo is now a String
   foobar = bar + 1.1   ' foobar is now a Double with the value of 2.1
   MsgBox TypeName(bar)     ' Integer
   MsgBox TypeName(foo)     ' String
  MsgBox TypeName(foobar) ' Double
End Sub
```

Figure 3.20

Questions

With Option explicit set in all your modules answer the following questions.

Write each answer in a function called myAnswer_ and end if it your question number. For example:

```
Option Compare Database
Option Explicit          ' make sure this line is in your code

Function myAnswer_1()

       ' your code goes here

End Function
```

Figure 3.21

1. Write code that will declare the following types and set them all to a value
    a. boolean
    b. integer
    c. long
    d. date
    e. single
    f. double
    g. currency
    h. string
    i. date
    j. a Variant String

2. write code that performs the following:
    a. declares a constant with the value "Hello".
    b. declares another constant called YEAR with the value 2012.
    c. declare a variable called myName and assign it your name.
    d. declare a second variable called myMesage and join the constant in (a) with the variable in (c).
    e. now add to myMessage the text ". The year is".
    f. now add to myMessage the constant YEAR (hint: function cstr()).
    g. finally add the following:

        debug.print myMessage

3. What is the output of the following sub?

```
Option Compare Database
Option Explicit          ' make sure this line is in your code

Sub myAnswer_3()

      iVar1 = 10
      iVar2 = "value of iVar1=" + iVar1
      msgbox iVar2
```

```
End Sub
```

Figure 3.22

4. What will the next code sequence do and why?

```
Option Compare Database
Option Explicit          ' make sure this line is in your code

Sub myAnswer_3()
      Dim iAnswer as Integer
      iAnswer = "42"

End Sub
```

Figure 3.23

5. What is the difference between the following?
   a. A1 = "42.2"
   b. A2 = 42.2
   c. And what would be the result of A1 * A2?

6. You start your code with the following instruction. Why does it not compile?

```
Option Compare Database
Option Explicit          ' make sure this line is in your code

Sub myAnswer_3()
      Dim function as String
      function = "Hello World!"
      msgbox function

End Sub
```

Figure 3.24

7. Rewrite the following in camelCase
   a. Calculate the age of a tree
   b. Tape reader file position
   c. User input
   d. File pointer
   e. Input
   f. sMyMessage

8. rewrite the following as constants
   a. semaphore stop
   b. semaphore start
   c. semaphore paused
   d. file open
   e. end of file
   f. carriage return and line feed
   g. new line

9. In a new function write code to do the following:
    a. Define a global constant called database name and give it the value “mysqldb_website1”
    b. Define another global variable with a meaningful name to hold an IP address, eg 127.0.0.1
    c. In local scope, declare a variable named sDBDetails adding the value of the constants from (a) and (b) making sure to add a space between them
    d. Add the following code and execute your code from the immediate window
        i. Msgbox sDBDetails

10. In a new function perform the following mathematical expressions by first assigning the numbers to letters and then saving the result into z, for example:
    a. 12 + 16, would be
        i. Dim a, b, z as integer
        ii. a=12
        iii. b=16
        iv. z=a+b
        v. debug.print z
    b. 100+1+20+2
    c. 76 * 89
    d. (-50 * 3 * -1) / 10
    e. 10 mod 3
    f. 2 to the power of 2 to the power of 2
    g. 2.5 * 7.6, make sure to preserve the decimal number
    h. Assign to an integer the value 2.7 . What is the integer’s value?
    i. Concatenate the following Strings with addition spaces between
        i. “Winston, you are drunk! ”
        ii. “Yes madam, and you are ugly!”
        iii. “But in the morning, I shall be sober”
    j. What is the square of 27 to the nearest whole number

11. What is displayed in the pop-up message box?

```
Option Compare Database
Option Explicit          ‘ make sure this line is in your code

Sub myAnswer_11()
      Dim s as String
      s = “I” + “ like “ + “Chinese food!”
      s = s + “ The wai-ters never are rude.”
      msgbox s

End Sub
```

Figure 3.25

12. What must you do to make the following code work?

```
Option Compare Database
'Option Explicit        ' make sure this line is in your code

Sub myAnswer_3()
      Dim d as Date
      d = 12 Dec 2012
      msgbox d

End Sub
```

Figure 3.26

13. In a new function assign the following dates to variables
    a. 11 November 1918
    b. 3 December 1992
    c. 18 October 1871
    d. 10 30 PM
    e. 12 – 12 – 2012 00:21
    f. 1969, July, 20th

14. Declare three variant variables, set their values to a person's name, any integer value and any floating-point number, respectively.

15. Write code to answer the following expressions:
    a. 20-True
    b. True+ True+ True-False
    c. (7656 mod 7) / 3
    d. 12 + 66 / 11
    e. #12-dec-2012# + #01/01/01#

16. Explain the differences between a Long number and a floating point number.

17. Explain why "10:26 PM" and #10:26 PM# are not the same?

18. If you are asking the user for their birth date, which data type would you / could you temporarily store their answer for further checking?

19. True or false:

    a. 20-20 = true?
    b. True and true = false?
    c. False or true = true?

20. Which of the following are valid variable names
    a. aVariable
    b. aFunction

c. end
d. while
e. STATE_HOLD
f. STATE OVER
g. File1
h. outputFile_10
i. input-file2
j. $helloWorld
k. 9LivesACatHas
l. todayisyesterday
m. Tomorrow NeverComes

Answers

1. If written as a statement in a function (1 mark)
   If all given different names (1 mark)
   Otherwise 1 mark for each of the following
   a. Dim a As Boolean /cr/lf/ a = true or false or -1 or 0
   b. Dim a As Integer
   c. Dim a As Long
   d. Dim a As Date
   e. Dim a As Single
   f. Dim a As Double
   g. Dim a As Currency
   h. Dim a As String
   i. Dim a As Date
   j. Dim a As Variant
2. 1 mark for each line
   a. Dim CONSTANT_NAME = “Hello “ ‘ there’s a space at the end
   b. Dim YEAR = 2012
   c. Dim myName as String
      i. myName = “pupil’s name”
   d. Dim myMessage as String
      i. myMessage = CONSTANT_NAME + myName
   e. myMessage = myMessage + “. The year is “
   f. myMessage = myMessage + CStr(YEAR)
   g. debug.print myMessage
3. No output as this subroutine does not compile
4. 1 mark for stating 42, +1 mark VBA automatically converts “42” into Integer type
5. 1 mark for each
   a. The String “42.2” is added to A1
   b. The Double 42.2 is added to A2
   c. Type mismatch error
   d. Cannot multiply string by integer
6. Function is a keyword
7. 1 mark for each
   a. calculateTheAgeOfATree
   b. TapeReaderFilePosition
   c. UserInput
   d. FilePointer
   e. Input
   f. sMyMessage
8. 1 mark for each
   1. SEMAPHORE_STOP
   2. SEMAPHORE_START
   3. SEMAPHORE_PAUSED
   4. FILE_OPEN
   5. END_OF_FILE
   6. CARRIAGE_RETURN_AND_LINE_FEED
   7. NEW_LINE
9. 1 mark for each

   a. Const DATABASE_NAME = “mysqldb_website1”
   b. Dim IP – IP may be anything as long as its meaningful and camelCase
      i. In function – IP=”127.0.0.1”
   c. In function
      i. sDBDetails = DATABASE_NAME + “ “ + (b variable)
   d. Msgbox sDBDetails
   e. Everything in a function

10. 1 mark for each. They should all follow the same basic format given in (a)
    8. Value is 3, rounding
    9. “Winston, you are drunk! ” + “Yes madam, and you are ugly!” + “But in the morning, I shall be sober”
    10. Trick question, “nearest whole number”
        10.1.1. Dim a, z as Integer
        10.1.2. a = 27
        10.1.3. z = a*a
11. `1 mark`
    a. `“I like Chinese food! The wai-ters never are rude.”`
12. 1 mark,
    a. Line 6 needs #’s: `d = #12 Dec 2012#`
13. 1 mark for each
    a. All should have #’s around them EXCEPT f, which needs to be rewritten without the “th”
14. 1 mark for each
    a. Dim [variable name] as Variant / or Dim name
    b. Followed by respective values
15. 1 mark for putting all into a single function
    1 mark for each expression
    a. 21
    b. -2
    c. 1.66666666666667
    d. 18
    e. 15/12/2113 – yes, that’s the answer; VBA doesn’t make sense here
16. 1 mark for each
    a. An integer holds whole numbers
    b. An floating-point number holds decimals / numbers and fractions
    c. An integer holds less data than a floating-point number / or vice-versa
    d. Integers are calculated in the CPU
    e. Floating-point numbers are calculated in the FPU / ALU
17. 1 mark – the first is a string, second a date
18. 1 mark – String
19. 1 mark for each
    a. False
    b. False
    c. true
20. 1 mark for each
    a. aVariable - valid
    b. aFunction - valid
    c. end – invalid keyword

d. while – invalid, keyword
e. STATE_HOLD - valid
f. STATE OVER – invalid, no spaces allowed
g. File1 - valid
h. outputFile_10 - valid
i. input-file2 – invalid, means input subtract file2, input is also a keyword
j. $helloWorld – invalid, variables must start with a letter
k. 9LivesACatHas - valid, variables must start with a letter
l. todayisyesterday – valid, but should be camelCase
m. Tomorrow NeverComes – invalid, nospaced in strings