

VBA Made Easy

Functions, Sub-procedures and Arguments

05

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.

© AXLSolutions 2012

All rights reserved. No part of this work may be reproduced in any form, or by any means, without permission in writing.

Contents

Functions, Sub Procedures And Arguments.....	3
VBA Language in Context	3
What is a Sub-Procedure?	3
What is a Function?	4
Calling Sub Procedures And Functions From The Immediate Window.....	4
Calling Sub Procedures from other Sub Procedures.....	6
Calling Functions	8
Built-in Functions	8
Using the Query Expression Builder to locate functions	8
Commonly Used Built-In Functions	11
String Functions	11
Conversion.....	11
Date and Time Functions	12
Is Functions	13
DFunctions – Database Functions	15
Custom Functions and Sub Procedures	16
Anatomy of a Sub Procedure	18
Anatomy of a Function	19
Declaring Functions and Procedures	20
Scope.....	20
Declarations in a Module and Global Scope (and a little private-cy).....	20
Declarations in a Form or Report Modules.....	23
Questions.....	24
Answers	28

Functions, Sub Procedures And Arguments

In this unit you will learn about Functions, Sub Procedures and Arguments.

VBA Language in Context

The core of the English language is its sentences and paragraphs. The sentence describes some action (verb) that is performed on or by an object (noun), and a paragraph is a set of sentences communicating some overall desired goal or aim. VBA is not unlike English in this sense.

VBA's paragraphs are called Procedures and Functions. Sentences are then the variables, operations, object methods and assignment statements in the Code Block. All recent programming languages share this same structure. To continue the analogy, functions and procedures (the paragraphs) are contained within books called VBA Modules. There are three types of book, or module:

- Forms and Reports Module (Microsoft Office Access Class Object Modules);
- Standard Modules.
- Class Modules;

When you write your code, it will always be written within a Function...End Function or a Sub...End Sub statement. VBA is what is known as a functional programming language. That is, we cannot just write code within a standard module and expect it to run; Access won't recognise this and will complain terribly, we must put Sub or Function around it.

What is a Sub-Procedure?

A sub-procedure is a code block that executes a series of actions. In other words, it is code that does something.

```
1 Sub getPriceIncVATSub()  
2  
3 Dim ItemPrice As Double  
4 Dim SalesTax As Double  
5 Dim PriceIncVAT As Double  
6  
7 ItemPrice = InputBox("What is the price of the item?")  
8 SalesTax = InputBox("What is the tax? (20%=0.2)")  
9 PriceIncVAT = ItemPrice + (ItemPrice * SalesTax)  
10 MsgBox ("The price of the item including VAT is: $" & PriceIncVAT)  
11  
12 End Sub
```

Figure 5.1

In this sub procedure you may notice that all the code is held within the *Sub getPriceIncVAT()* and the *End Sub* statements. These are the outer limits of the sub procedures and any code that comes before *Sub getPriceIncVAT()* and after *End Sub* do not form part of the sub procedure

What is a Function?

Functions are not dissimilar to sub procedures in that they do something but where they differ is that they also return a value.

```
1  Function getPriceIncVATFunction(ItemPrice As Double)
2
3  Dim SalesTax As Double
4  Dim PriceIncVAT As Double
5  SalesTax = 0.2
6  getPriceIncVATFunction = ItemPrice + (ItemPrice * SalesTax)
7
8  End Function
```

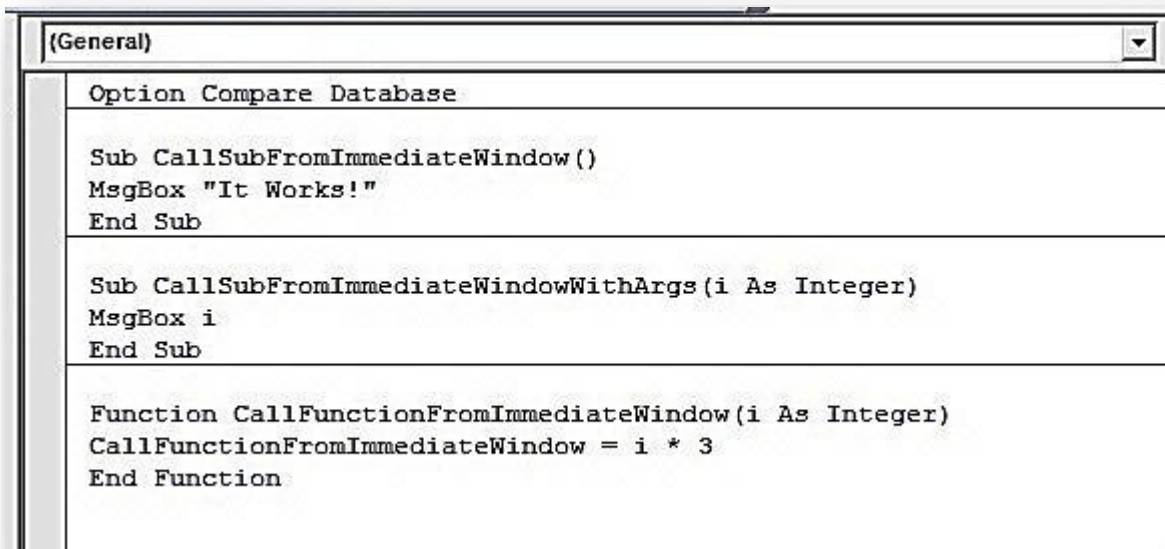
Figure 5.2

In Figure 5.2 we have changed the sub procedure into a function and it is now returning a value.

Note: Please only take into consideration the structure of the function as we will be covering the syntax in greater detail later on in this unit.

Calling Sub Procedures And Functions From The Immediate Window

One of the benefits of the immediate window is that we can use it to test sub procedures and functions.



```
(General)
Option Compare Database

Sub CallSubFromImmediateWindow()
MsgBox "It Works!"
End Sub

Sub CallSubFromImmediateWindowWithArgs(i As Integer)
MsgBox i
End Sub

Function CallFunctionFromImmediateWindow(i As Integer)
CallFunctionFromImmediateWindow = i * 3
End Function
```

Figure 5.3

Take a look at Figure 5.3 where we have 2 very simple sub procedures and 1 very simple function.

In order to call the procedure *CallSubFromImmediateWindow* using the immediate window, we merely need to write its name (without the parentheses).

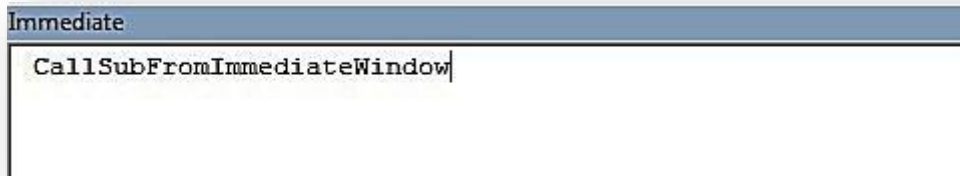


Figure 5.4

This will cause a message box to pop it that states “It works!”

We can also add arguments to the immediate window. In the second sub called *CallSubFromImmediateWindowWithArgs* we need to pass a value *i*. We do this by writing the name of the procedure and then adding the necessary argument to the right.

Note: *If there are more than one arguments, separate them with a comma.*

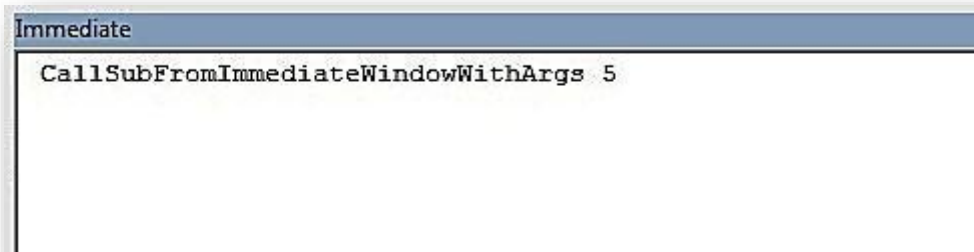


Figure 5.5

In Figure 5.5, we call *CallSubFromImmediateWindowWithArgs* and provide the argument *i*. In this case, we pass the value 5 and a message box will pop up with the value 5 in it. Whatever we change the value of the argument to will be reflected in the value that the message box displays.

We can also test functions. Remember that functions are essentially the same as sub procedures with the difference that they return a value.

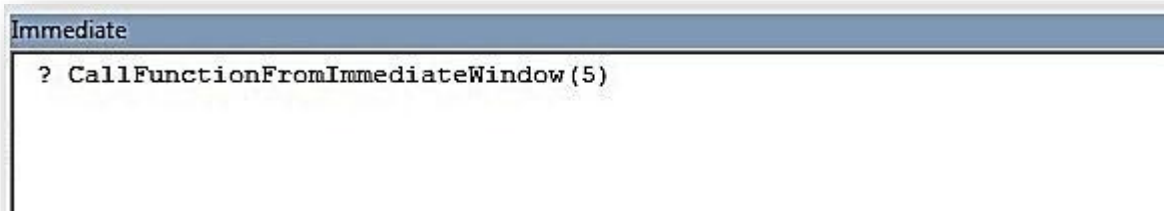


Figure 5.6

To test a function from the immediate window we use a question mark and then we write the name of the function. We follow the function with parentheses and any relevant arguments are placed inside the parentheses. We have done this in Figure 5.6.

```
? CallFunctionFromImmediateWindow(5)
15
```

Figure 5.7

Figure 5.7 shows that if we provide 5 as an argument for this particular function we get a value returned of 15. Try adding different values as the argument to see what return value you get.

Calling Sub Procedures from other Sub Procedures

One of the most important features of VBA is the ability to call sub procedures from other sub procedures. What do we mean? Take a look at this code to find out:

```
(General) Main
Option Compare Database
Private strName As String
Private strAge As Integer

Sub Main()
Call getName
Call getAge
Call printDetails
End Sub

Sub getName()
strName = InputBox("What is your name?")
End Sub

Sub getAge()
strAge = InputBox("How old are you?")
End Sub

Sub printDetails()
Debug.Print "Your name is " & strName & " and you are " & strAge & " years old."
End Sub
```

Figure 5.8

In Figure 5.8 we have 4 sub procedures *Main*, *getName*, *getAge* and *printDetails*. The main sub procedure we have cleverly called *Main* and this sub procedure calls all the other sub-procedures within the module. It first calls *getName* which has the objective of asking the user's name. This value is then assigned to *strName* which is a module level variable. Next, *getAge* is called which involves another input box asking you for your age and again the value is stored in a module level variable called *strAge*. Finally *printDetails* is called which takes the 2 module level variables and concatenates them in a string which is printed in the immediate window.

In Figure 5.9 below we call the sub Main from the immediate window by writing Main and pressing the return key and then provide Steve and 25 as the values for the variables.

Note: Breaking code down into manageable chunks and having a main procedure that calls other procedures (and functions) is an excellent way to code.

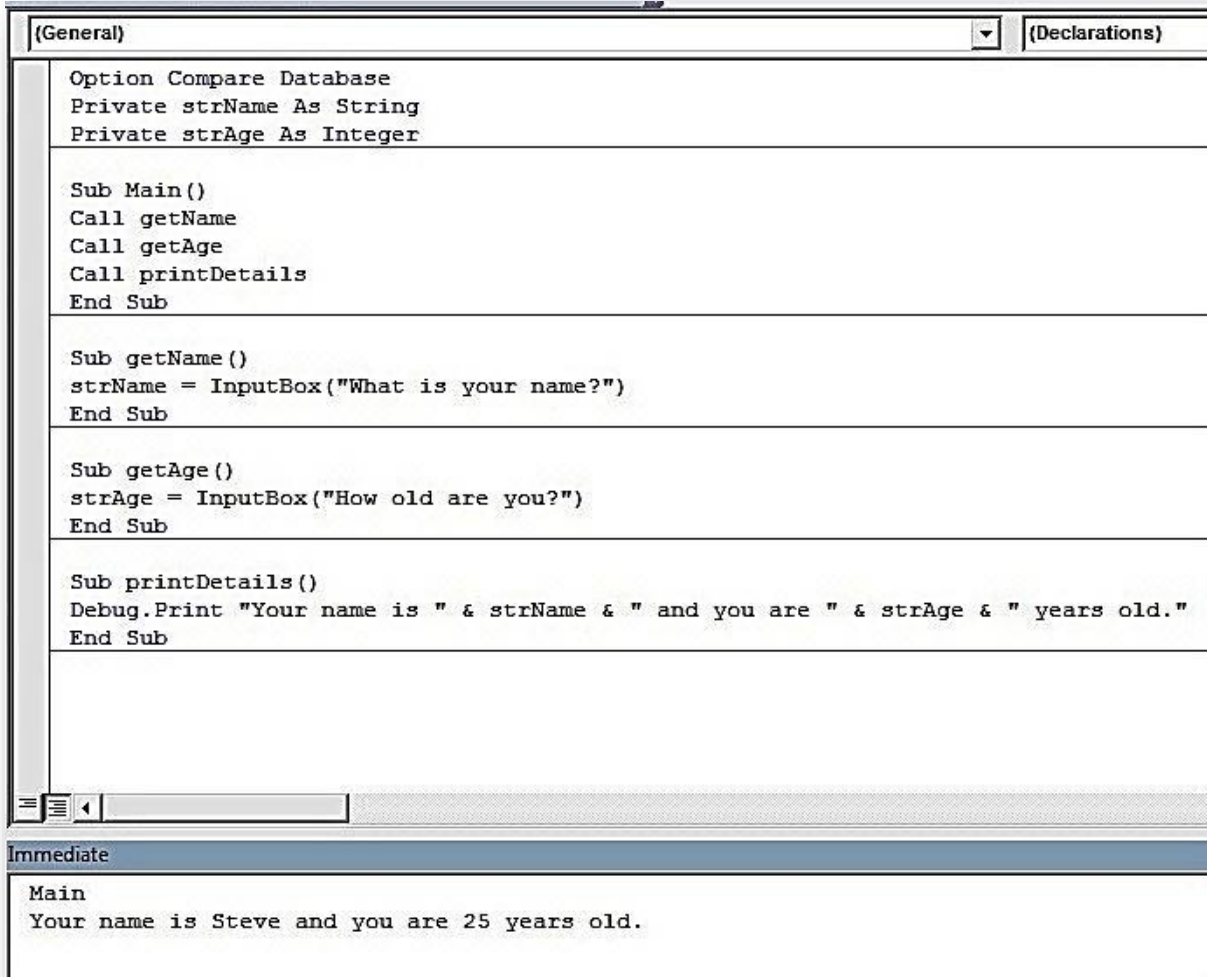
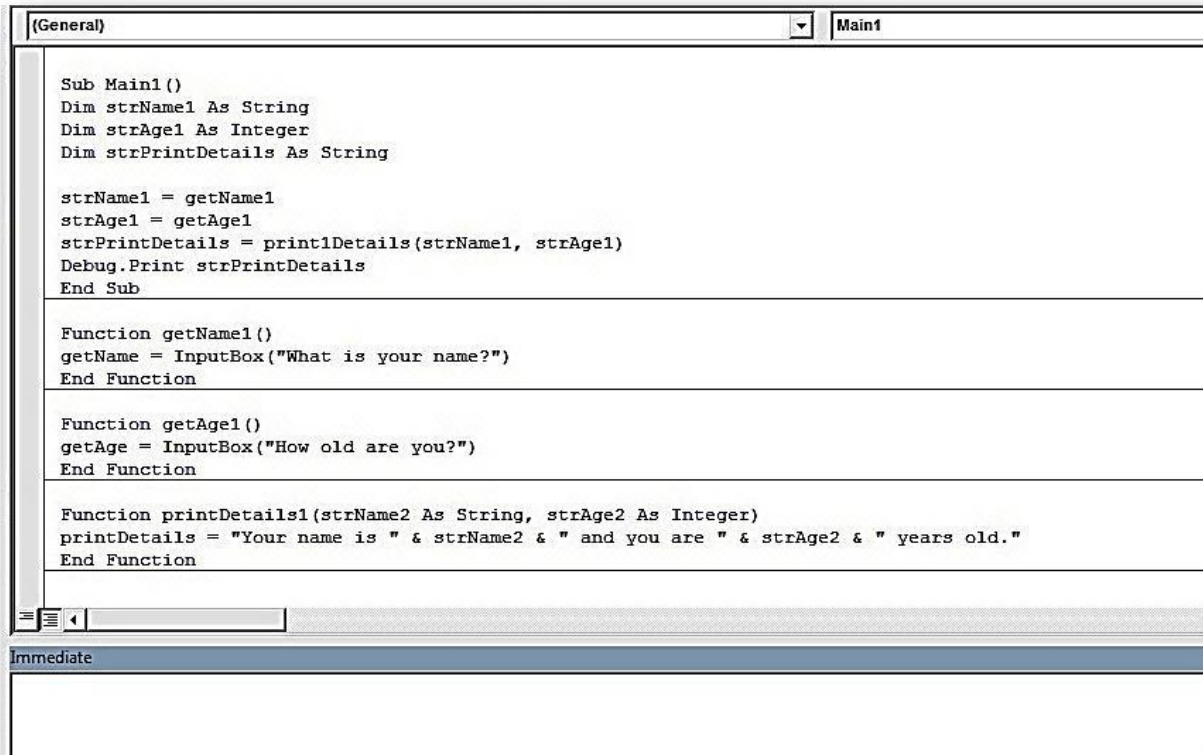


Figure 5.9

Calling Functions

Figure 5.10 has the same concept (you are asked for your name and age which are printed in the immediate window) but this time we are using 1 sub procedure (Main) which is calling functions. As functions return values, it is those that are used as the basis for the concatenated string at the end.



```
Sub Main1()  
Dim strName1 As String  
Dim strAge1 As Integer  
Dim strPrintDetails As String  
  
strName1 = getName1  
strAge1 = getAge1  
strPrintDetails = print1Details(strName1, strAge1)  
Debug.Print strPrintDetails  
End Sub  
  
Function getName1()  
getName = InputBox("What is your name?")  
End Function  
  
Function getAge1()  
getAge = InputBox("How old are you?")  
End Function  
  
Function printDetails1(strName2 As String, strAge2 As Integer)  
printDetails = "Your name is " & strName2 & " and you are " & strAge2 & " years old."  
End Function
```

Figure 5.10

Using functions is another great way to break down your code into manageable chunks. In the previous example, we wrote custom functions but VBA has plenty of built-in functions all of its own.

Built-in Functions

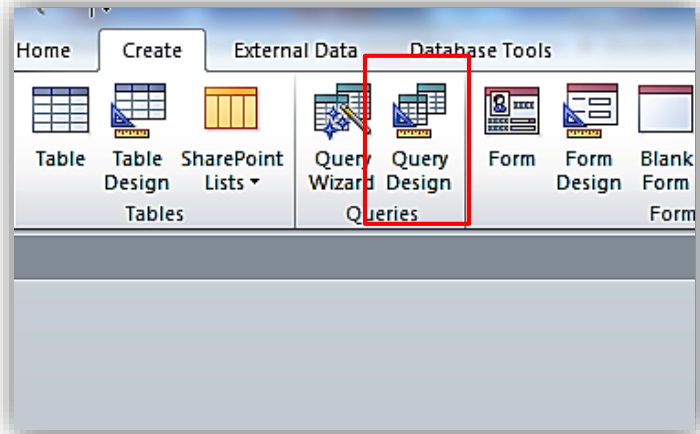
VBA has a wide library of built-in functions. Please look through them and experiment with them. Most coding issues you try to overcome and actions to be fulfilled can be performed by using these functions, so try not to reinvent the wheel.

Using the Query Expression Builder to locate functions

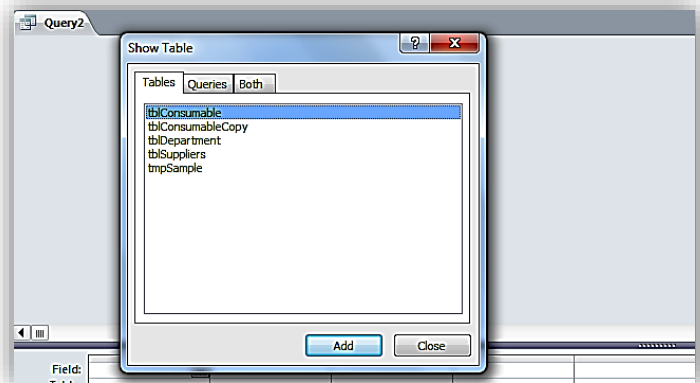
As there are scores of built-in functions in Access/VBA, wouldn't it be great if we had an easily accessible list that listed not only the functions but also their uses? Well, rest assured, we do (kind of). We can use the expression builder in a query to perform this particular function (do you like what we did there?)

Opening the Expression Builder

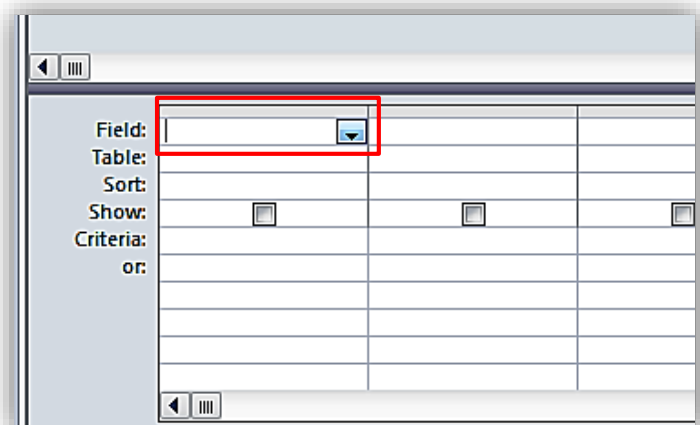
In the main Access window click on the Query Design button which can be found in the Queries group of the Create tab of the Ribbon.



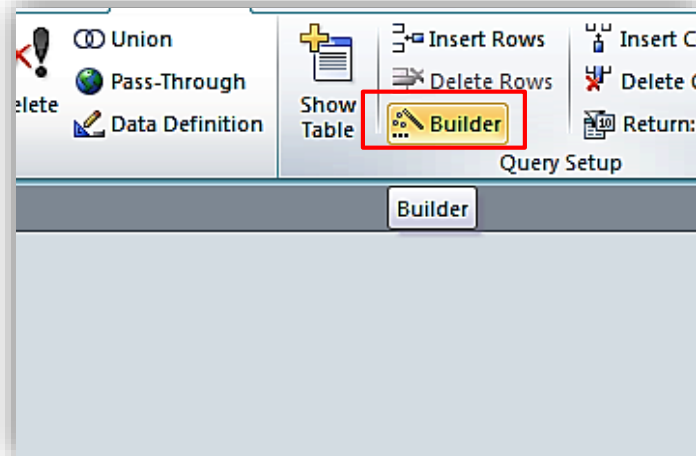
Dismiss the Show Table Dialog Box.



Click in the Field row in any column in the field designer window.



Click on the Builder button which is located in the Query Setup group of the Design tab of the Ribbon.



The Expression Builder dialog box will pop up.

Open the Functions Node (1) in the Expression Elements window and a list of all functions will be displayed.

If you click on one of the functions in the Expression Values window (2) you will get a brief explanation of what it does (3).

Clicking on the hyperlink text of the function syntax (4) will open up a more detailed explanation of the function in a browser window.

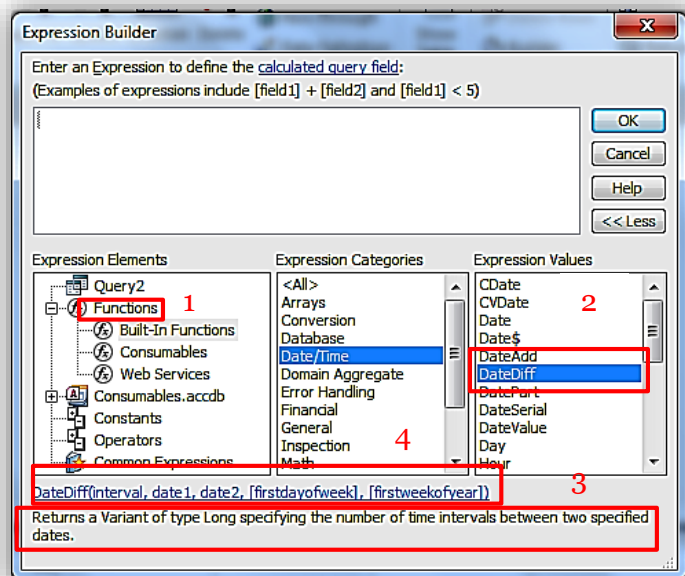


Figure 5.11

Commonly Used Built-In Functions

This section will provide you with examples of commonly used built-in functions

String Functions

- `Len(s)` – returns the length of String `s`.
- `Left(s, n)` – returns a substring of `s` that is `n` chars long from the left of the string `s`.
- `Right(s, n)` – returns a substring of `s` that is `n` chars long from the right of the string `s`.
- `Mid(s,nb,ne)` – returns a substring of `s` from characters `nb` to `ne`, inclusive.

```
1 Sub testStrings()  
2   Debug.Print Len("Hello World")  
3   Debug.Print Left("Hello World", 10)  
4   Debug.Print Right("Hello World", 7)  
5   Debug.Print Mid("Hello World", 7, 10)  
6 End Sub
```

Output in immediate window:

```
11  
Hello Worl  
o World  
World
```

Figure 5.12

Conversion

- `CInt(anything)` – converts anything into an Integer type (if possible).
- `Cdbl(anything)` – converts anything into an Double type (if possible).
- `CInt(anything)` – converts anything into an Long type (if possible).
- `CStr(anything)` – converts anything into a String.
- `CDate(string)` – converts a string to a Date type (if possible).

If any of the conversion functions are passed a variable that cannot be parsed – e.g. `CInt("oioi!")` – a Type Mismatch error occurs.

```
1 Sub testConversions()  
2   Dim i As Integer, d As Double, l As Long, s As String  
3   i = 19  
4   d = 12.6  
5   l = 32768  
6   s = "42.001"  
7  
8   ' to display the answers provided by the conversion functions we have to  
9   ' CStr() all the number variables or VBA will throw a Type Mismatch error  
10  ' so just to prove that CStr works we'll do it first  
11  Debug.Print "First test CStr on all types"  
12  Debug.Print "CStr(i) = '" + CStr(i) + "'" ' 42'  
13  Debug.Print "CStr(d) = '" + CStr(d) + "'" ' 42.001'  
14  Debug.Print "CStr(l) = '" + CStr(l) + "'" ' 42'  
15  Debug.Print "CStr(s) = '" + CStr(s) + "'" ' 42.001'  
16  Debug.Print ""  
17  Debug.Print "Second, CInt"  
18  Debug.Print "CInt(i) = " + CStr(CInt(i)) ' 19  
19  Debug.Print "CInt(d) = " + CStr(CInt(d)) ' 13  
20  Debug.Print "CInt(l) = Overflow Error. Integers are valued <32768"  
21  Debug.Print "CInt(s) = " + CStr(CInt(s)) ' 42  
22  Debug.Print ""  
23  Debug.Print "Third, Cdbl"  
24  Debug.Print "Cdbl(i) = " + CStr(Cdbl(i))  
25  Debug.Print "Cdbl(d) = " + CStr(Cdbl(d))  
26  Debug.Print "Cdbl(l) = " + CStr(Cdbl(l))
```

```

25  Debug.Print "CDBl(s) = " + CStr(CDBl(s))
26  Debug.Print ""
27  Debug.Print "Fourth, CLng"
28  Debug.Print "CLng(i) = " + CStr(CLng(i)) ' 19
29  Debug.Print "CLng(d) = " + CStr(CLng(d)) ' 13
30  Debug.Print "CLng(l) = " + CStr(CLng(l)) ' 32768
31  Debug.Print "CLng(s) = " + CStr(CLng(s)) ' 42
32  End Sub

```

Output in immediate window:

```

testConversions
First test CStr on all types
CStr(i) = '19'
CStr(d) = '12.6'
CStr(l) = '32768'
CStr(s) = '42.001'

Second, CInt
CInt(i) = 19
CInt(d) = 13
CInt(l) = Overflow Error. Integers are valued <32768
CInt(s) = 42

Third, CDBl
CDBl(i) = 19
CDBl(d) = 12.6
CDBl(l) = 32768
CDBl(s) = 42.001

Fourth, CLng
CLng(i) = 19
CLng(d) = 13
CLng(l) = 32768
CLng(s) = 42

```

Figure 5.13

Date and Time Functions

Date and time functions are quite complex due to the nature of dates. VBA has a special way of handling dates by putting # around them; for example `dMyDate = #18-Dec-2012#`. Here are some of the functions to help with dates.

- `Date ()` – returns the current date.
- `Now()` – returns the current date and time.
- `DateSerial(year, month, day)` – returns a Date object if parameters are valid.
- `Year(date)` – returns the *year* of *date* as an integer.
- `Month(month)` – returns the *month* of *date* as an integer, 1-12.
- `Day(Day)` – returns the *day* of *date* as an integer, 1-31.
- `DateDiff(interval, date, date)` – *date* are dates, *interval* is day, month, year, etc.
- `DateAdd(interval, number, date)` – add to *date* intervals multiplied by *number*

Date Intervals

In the above *interval* refers to one of the following:

Interval	Description
yyyy	Year
q	Quarter
m	Month
y	Day of year
d	Day
w	Weekday
ww	Week
h	Hour
n	Minute
s	Second

Figure 5.14

Note: The *Date* function returns the current date (as defined by your operating system) so the results you get from the following example will be different from the results we obtained.

```
1 Sub testDateTime()  
2   Debug.Print Date  
3   Debug.Print Now()  
4   Debug.Print DateSerial(2012, 12, 18)  
5   Debug.Print Year(Date)  
6   Debug.Print Month(Date)  
7   Debug.Print Day(Date)  
8   Debug.Print DateAdd("d", 421, Date)  
9   Debug.Print DateDiff("d", Date, #1/1/2020#)  
10 End Sub
```

Output in immediate window:

```
27/12/2012  
27/12/2012 22:50:08  
18/12/2012  
2012  
12  
27  
21/02/2014  
2561
```

Figure 5.15

Is Functions

When inspecting whether a variable has a value we usually use the equals = operator, but equals does not work if a variable is null, empty or is nothing. Nor can equals be used to interrogate the variable for its type. There are special 'Is' operators which provide for that functionality.

- *IsDate(anything)* – returns true if variable is a date.
- *isArray(anything)* – return true if variable is an array.
- *IsNull(anything)* – returns true if variable is Null.
- *IsEmpty(anything)* – returns true when type variable is uninitialized.
- *isObject(anything)* – returns true when variable is an Object.
- *TypeName(anything)* – returns a string.

IsDate and IsEmpty

```
1 Sub dateAndEmptyFunctions()  
2   Dim myDate  
3  
4   Debug.Print IsDate(myDate)  
5   Debug.Print IsEmpty(myDate)  
6  
7   myDate = #12/20/2012#  
8   Debug.Print IsDate(myDate)  
9   Debug.Print IsEmpty(myDate)  
10 End Sub
```

Output in immediate window:

```
False  
True  
True  
False
```

Figure 5.16

Note: We will be covering arrays in a future unit.

IsArray and IsNull

```
1 Sub arrayAndNullFunctions()  
2   Dim myArray As Variant  
3   myArray = Array("first_name", "surname", "dob", "town", Null)  
4  
5   Debug.Print IsArray(myArray)  
6   Debug.Print IsNull(myArray(0))  
7   Debug.Print IsNull(myArray(1))  
8   Debug.Print IsNull(myArray(2))  
9   Debug.Print IsNull(myArray(3))  
10  Debug.Print IsNull(myArray(4))  
11  
12 End Sub
```

Output in immediate window:

```
True  
False  
False  
False  
False  
True
```

Figure 5.17

IsObject and TypeName

```
1
2 Sub objectAndTypeNameFunctions()
3     Dim varA, varB As Object, varC As Date, varD As DAO.Recordset
4
5     Debug.Print
6     Debug.Print "isObject(varA) = "; CStr(IsObject(varA)); Tab; "TypeName(varA) =
7 "; TypeName(varA)
8     Debug.Print "isObject(varB) = "; CStr(IsObject(varB)); Tab; "TypeName(varB) =
9 "; TypeName(varB)
10    Debug.Print "isObject(varC) = "; CStr(IsObject(varC)); Tab; "TypeName(varC) =
11 "; TypeName(varC)
12    Debug.Print "isObject(varD) = "; CStr(IsObject(varD)); Tab; "TypeName(varD) =
13 "; TypeName(varD)
14 End Sub
```

Output in immediate window:

```
isObject(varA) = False      TypeName(varA) = Empty
isObject(varB) = True       TypeName(varB) = Nothing
isObject(varC) = False     TypeName(varC) = Date
isObject(varD) = True       TypeName(varD) = Nothing
```

Figure 5.18

DFunctions - Database Functions

Sometimes it is necessary to retrieve certain data from the database - e.g. a manufacturer's name – or perform a quick count on records. Rather than having to create objects and write SQL statements VBA offers a couple of smart and concise routines to obtain what you need without all the object/SQL hassle.

All DFunctions have the same signature *expression, table[, criteria]* which is similar in structure to SQL itself.

- DLookup (*expression, table, [criteria]*) – Looks up a value in a table or query.
- DCount (*expression, table, [criteria]*) – Counts the records in a table or query.
- DSum(*expression, table, [criteria]*) – Returns the sum of a set of records in a range.
- DMax (*expression, table, [criteria]*) – Retrieves the largest value from a range.
- Dmin(*expression, table, [criteria]*) – Retrieves the smallest value from a range.
- DAvg(*expression, table, [criteria]*) – Returns the average set of numeric values from a range.
- DFirst (*expression, table, [criteria]*) – Returns the first value from a range.
- DLast (*expression, table, [criteria]*) - Returns the last value from a range.

```
1
2 Sub DFunctions()
3
4     'These D-Functions will be using data from the teachers table
5
6     Debug.Print DLookup("[LastName]", "tblTeachers", "[FirstName]='Anna'")
7     'We are looking up a value in the [LastName] field of tblTeachers.
8
9     Debug.Print DCount("*", "tblTeachers")
10    'The asterisk (*) means that we are counting
11    ' all the records in the         table
12
13    Debug.Print DSum("[TotalPaid]", "tblTeachers")
14    'Adds up all of the values from [TotalPaid]
```



```

15
16     Debug.Print DMax("[RatePerHour]", "tblTeachers")
17     'Returns the largest value from [RatePerHour]
18
19     Debug.Print DMin("[RatePerHour]", "tblTeachers")
20     'Returns the smallest value from [RatePerHour]
21
22     Debug.Print DFirst("[LastName]", "tblTeachers",
23 "[ZIPPostal]='98052'")
24     'Returns the [LastName] of the first record where
25 [ZIPPostal]='98052'
26
27     Debug.Print DLast("[LastName]", "tblTeachers",
28 "[ZIPPostal]='98052'")
29     'Returns the [LastName] of the last record where
30 [ZIPPostal]='98052'
31
32 End Sub

```

Output in immediate window:

```

Gratacos Solsona
9
2980.4
13.2
11.5
Axen
Wacker

```

Figure 5.19

Custom Functions and Sub Procedures

Having looked at built-in functions we are now going to create our own custom function.

Let's write a function that calculates the age of a student given the date of birth. The details we know are as follows:

- A returned value is needed, so we must use a function.
- The value returned will be somebody's age, so we should return an Integer.
- The function needs to know the student's DOB, so a Date parameter is needed.
- We also need a relevant function name; let's call it *calculateAge*.

The signature of the function then is:

```

Function calculateAge(DOB As Date) As Integer
End Function

```

We need a variable to store the age and to store today's date:

```

Dim iAge as Integer
Dim dToday as Date

```

Figure 5.20

Now we need to know the difference between DOB and today's date in years. VBA has a function for that, *DateDiff*. Let's set *dToday* to today's date and use *DateDiff* to give us the age in years.

```
dToday = Date()  
  
iAge = DateDiff("yyyy", DOB, dToday) ' yyyy interval date
```

Figure 5.21

Finally, we also need to return iAge to the calling method by doing the following:

```
calculateAge = iAge
```

Figure 5.22

The whole function now looks like this:

```
1 Function calculateAge(DOB As Date) As Integer  
2     Dim iAge As Integer  
3     Dim dToday As Date  
4  
5     dToday = Date  
6     iAge = DateDiff("yyyy", DOB, dToday) ' yyyy interval date  
7     calculateAge = iAge  
8 End Function
```

Figure 5.23

In the immediate window we call the function with a known anniversary date, e.g. today's date minus 1 year:

```
Output in immediate window:  
  
Print calculateAge (#19/12/2011#)  
1
```

Figure 5.24

Let's try with another known date, your own age:

```
Output in immediate window:  
  
? calculateAge (#15/11/1978#)  
34
```

Figure 5.25

Note: The Date function returns the current date (as defined by your operating system) so the results you get from the following example will be different from the results we obtained.

So, we know how to use sub procedures and functions. Let's take a closer look at the syntax of each one.

Anatomy of a Sub Procedure

In VBA the *Sub* keyword denotes a *procedure*. Procedures are designed to perform some action.

The syntax of a procedure is:

```
Sub nameOfSub (arguments | optional arguments As Datatype[=defaultValue] )  
    [Code Block]  
End Sub
```

nameOfSub – name of the sub procedure.

Arguments – are a list of values and types that are collected and used within the sub procedure.

Optional arguments As Datatype[=defaultValue] – an argument may be optional and if it is then you may provide a default value.

```
1  \ Declarations of Procedures-syntax highlighting to aid understanding  
2  \ put this section in the module window  
3  
4  Sub DoNothing()  \ basic procedure  
5      MsgBox "Do Nothing ☹"  
6  End Sub  
7  
8  Sub DoNothing2(name as String)  \ one argument provided  
9      MsgBox "the name is " + name  
10 End Sub  
11  
12 Sub DoNothin3(optional name as String)  \ one optional argument  
13     MsgBox "The name is " + name  
14 End Sub  
15  
16 \ one optional argument which defaults to Julia  
17 Sub DoNothing4(optional name as String = "Julia")  
18     MsgBox "The name is " + name  
19 End Sub  
20  
21  
22 Sub DoNothing5(name as String, age as Integer)  \ two arguments provided  
23     MsgBox "the name is " + name + " with age " + CStr(age)  
24 End Sub  
25  
26 \ put this section into the immediate window  
27 DoNothing  \ Simple call  
28 DoNothing2 "Julia"  \ Julia displayed  
29 DoNothing3  \ optional name left out, blank appears  
30 DoNothing4  \ optional name left out but will default to Julia  
31 DoNothing5 "Julia", 32  \ two arguments
```

Figure 5.26

Anatomy of a Function

In VBA a Function is a Procedure that returns a value. Functions accept data through arguments, they perform operations internally just like a procedure, but finish with a value which may be returned by the function.

```
Function nameOfFunction (arguments | optional arguments As  
Datatype[=defaultValue] ) _  
    As returnDataType  
    [Code Block]  
    [nameOfFunction = expression]  
End Function
```

nameOfFunction – is the name of the function.

Arguments – are a list of values and types that are collected and used within the function.

optional [arguments] [=defaultValue]] – an argument may be optional and if it is then you may provide a default value.

returnDataType – If stated, this is the value returned by the function, its data type.

```
1  ' Declarations of functions -syntax highlighting to aid understanding
2  ' put this section in the module window
3
4  Function returnName1() ' basic procedure
5      returnName1 = "returnName1 Called"
6  End Function
7
8  Function returnName2(name as String) as String ' return name
9      returnName2 = name
10 End Function
11
12 Function returnName3(optional name as String) ' return name or Shaun
13     If name="" Then returnName3="Julia" else returnName3=name
14 End Function
15
16 ' one optional argument which defaults to Julia
17 Function returnName4(optional name as String = "Julia")
18     returnName4 = name
19 End Function
20
21
22 Function returnName5(name as String, age as Integer) ' two arguments
23     returnName5 = "the name is " + name + " with age " + CStr(age)
24 End Function
25
26 ' put this section into the immediate window
27 Debug.Print returnName1()
28 Debug.Print returnName2("Robert")
29 Debug.Print returnName3()
30 Debug.Print returnName3("Robert")
31 Debug.Print returnName4()
32 Debug.Print returnName5("Robert", 34)
```

Figure 5.27

Declaring Functions and Procedures

Above we've read about what the differences are between functions and procedures

Scope

As we have seen, it is possible to call functions and sub procedures from other functions and sub procedures. But you can restrict which sub procedures and functions can be called. This is known as the scope of a function or sub procedure and is dependent on the location in which it is written and also the modifiers you put before the function or sub procedure name.

Possible modifiers are:

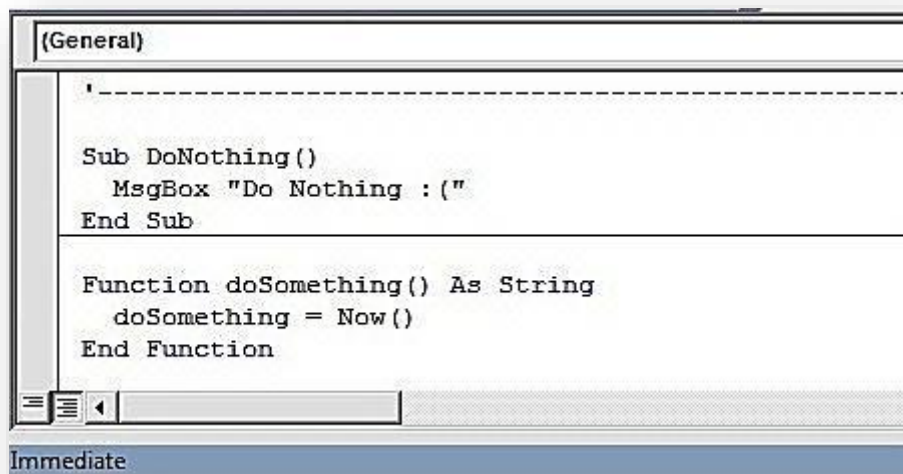
- **Private**- eg. `Private Sub txtName_Click()`
- **Public** - eg. `Public Function getCustomerName() As string`
- **Nothing** - eg. `Function isLeapYear() As Boolean`

For all modules, Private stops anything seeing the private function or sub procedure except for other functions or sub procedures in the same module.

Putting Public before a method in a Standard Module, or putting nothing at all means that the method is available anywhere in the application, its GLOBAL! The reason for this is that Standard Modules are in global context.

Declarations in a Module and Global Scope (and a little private-cy)

In the example below we have a sub procedure and a function.



```
(General)
-----
Sub DoNothing()
    MsgBox "Do Nothing :("
End Sub

Function doSomething() As String
    doSomething = Now()
End Function
```

Immediate

Figure 5.28

You can execute this function and sub procedure by entering their names directly into the immediate window one after the other:

```
DoNothing
doSomething
debug.print doSomething()
```

-You will notice that DoNothing displays a dialog box

-At line 2 doSomething() appears to do nothing

-At line 3 printing the output of doSomething() reveals the current time

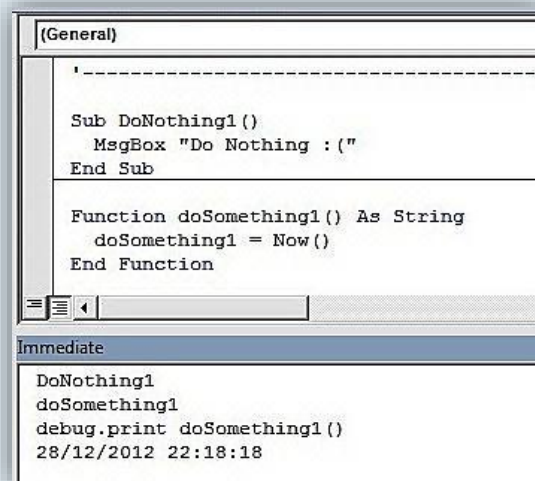


Figure 5.29

In fact, you can execute this function and sub procedure from anywhere in your application. For example, navigate to the Module *FromAnywhere* and call *CallFromHere* from the immediate window.

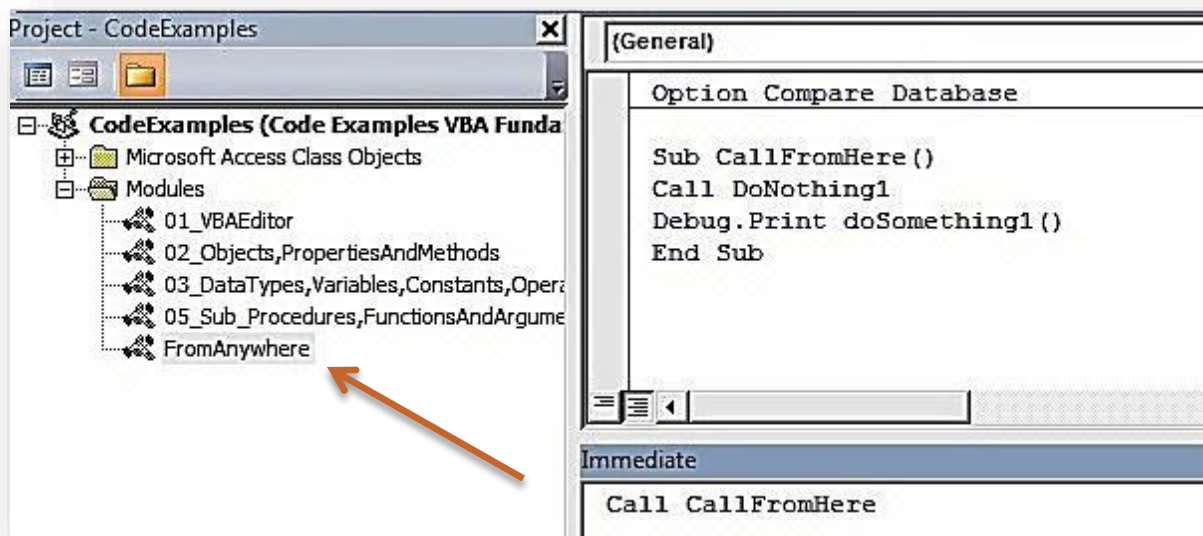


Figure 5.30

```
DoNothing
doSomething
debug.print doSomething()
```

-You will notice that DoNothing displays a dialog box

-At line 2 doSomething() appears to do nothing

-At line 3 printing the output of doSomething() reveals the current time

Figure 5.31

To demonstrate scoping with the Private modifier, add Private to the sub procedure *DoNothing1* and the function *doSomething1* and rerun the immediate window tests.

The image shows a screenshot of the Visual Basic Editor's 'General' tab. The code is as follows:

```
Private Sub DoNothing1()  
    MsgBox "Do Nothing : ("  
End Sub  
  
Private Function doSomething1() As String  
    doSomething1 = Now()  
End Function
```

Figure 5.32

.Now *DoNothing* does nothing, except give you the error below! Private in a module means no VBA code outside the Module can see this sub procedure or function.



Figure 5.33

Declarations in a Form or Report Modules

In the Events unit you may have seen that all event subs created by the IDE are declared with the Private modifier. Private ensures that it is not possible for code outside the Form to call its own code. This is particularly important for Forms as executing any of the event procedures could cause a modification of data! That is why all Event Procedures are Private.

In Form and Report modules, only put that code which is unique and specific to that form or report. You may include Public sub procedures if you need to give access to some functionality unavailable by conventional mechanisms.

Forms and reports do not need to be open for public sub procedures to be called and variables set or actions performed.

Questions

1. Why would you want to use a function instead of a sub procedure?
2. Which one of the following signatures is valid for a function called **appointmentDate**?
 - a. `Function appointmentDate(customerID As Integer) As Date`
 - b. `Function Date appointmentDate(Integer customerID)`
 - c. `Sub appointmentDate(customerID As Integer) As Date`
 - d. `Date appointmentDate(Integer customerID)`
3. The signatures below have been extracted from a Standard Module. Which are available in Global scope?
 - a. `Private Function getNewID() As Integer`
 - b. `Public sub updateCustomerName(id as Integer, name as String)`
 - c. `Function IsClass(text As String) As Boolean`
 - d. `Sub updateModificationDate(recorded As Long)`
 - e. `Private Sub GetNextRecord()`
4. Match each DFunction on the left with its description on the right
 - a. DSum
 - b. DCount
 - c. DLookup
 - d. DMin
 - a. Returns the value of a field in a table for which ID=20.
 - b. Ordered by invoice number the function will return the smallest numerical value.
 - c. Returns a value equal to the number of records in a table.
 - d. For a table of invoices this function will return the total value of all invoices
5. Using the expression builder find the mathematical functions which do the following:
 - a. Calculates the square of a number.
 - b. Returns today's date.
 - c. Returns the time now.
 - d. Returns the difference between two dates.
 - e. Converts a Boolean value to a string.
 - f. Returns true when an object reference is empty.
 - g. Returns false when a recordset field doesn't have the value of null.
 - h. Gives back the aggregate sum value of a table's tax field.
 - i. Converts a string into a date.
6. Which function returns the string value of a variable type?

7. Function giveMeTime(name As String) As Date
 - a. What is the return data type?
 - b. Is this a procedure or a function?
 - c. With time As Date can time=giveMeTime("Mike")?
 - d. Which of the following will give a compiler error
 - i. A = giveMeTime "Mike"
 - ii. giveMeTime "Mike"

8. Match the following String functions on the left with their description on the right

<ol style="list-style-type: none"> a. Mid(s, a, b) b. Len(s) c. Left(s, a) d. Right(s, a) e. InStr(1, s, c) 	<ol style="list-style-type: none"> a. Gives the ending of a string from character position A to the end b. Returns a substring of a string c. From the beginning returns a smaller string from position no with length a d. Searches for one string inside another e. Give a count of the characters in a string
--	---

9. What does Now() provide you with that Date() does not?
10. What is the return value of Month(#29-February-2012#)
11. Write the following function called textAddNumber:
 - a. Parameters of myText and myNumber.
 - b. Returns a string equal to the text of myText with myNumber appended to the end.
 - c. Such that "Your score is" and 13 returns "Your score is 13".
12. Write the following procedure called calculate:
 - a. Parameters of a(integer), b(string), c(string)
 - b. Allocate a to houseNo, b to teleNum, c to Surname
 - c. Concatenate c+b+a to d
 - d. Write debug,print d
13. Using DLookup, write an expression that retrieves the [surname] of a [pupil] with [id] of 1192.

14. Using DCount write an expression that counts the number of [students] with a [telephone] number beginning with “555”.

15. Match the following date intervals with the description

Interval	Description
d	Weekday
h	Year
m	Month
n	Day
s	Second
w	Minute
yyyy	Hour

16. True or False (; a semi colon denotes a new line)?

- IsDate(#05/11/2012#)
- IsDate(#01:36:01#)
- Dim var As Application; IsObject(var)
- Dim foobar; IsEmpty(foobar)
- Dim foo as String; TypeName(foo) = "String"
- Dim bar as Object; TypeName(bar) = "Empty"

17. Write a function that, given an array (myArray) and an integer (i), returns the value of the myArray element i

18. In which module would you place the following code? Answer a) Standard Module, b) Form Module or c) Class Module.

- A globally available function?
- A procedure that can only be used by a form?
- A procedure that operates on a form but is available outside the form?
- A function that is specific to a class?
- A class function that can only be used by the same class?
- A procedure available to the whole project that minimises all windows and opens the form MainMenu?

19. On a new form you place three buttons named btnButton1, btnButton2, btnButton3.
When btnButton1 is clicked a message is displayed.
When btnButton2 is double-clicked the form closes.

When btnButton3 is clicked nothing happens.
What has buttons 1 and 2 that button 3 doesn't?

20. Read the following code

```
Sub DoNothing4(optional name as String = "Julia")  
  
    MsgBox "Morning Dave. My name is " + name  
  
End Sub
```

- a. What does optional mean?
- b. What is the default value of name?
- c. What is the name of the method?
- d. When the method is execute with the following values, what is the result?
DoNothing4 ("Hal 9000")

Answers

1. If you want a returned value
2. a
3. b, c, d
4. a-d, b-c, c-a, d-b
5. a
 - a. sqr
 - b. date()
 - c. now()
 - d. datediff
 - e. CStr
 - f. IsEmpty
 - g. IsNull
 - h. DSum
 - i. CDate
6. TypeName
7.
 - a. Date
 - b. Function
 - c. Yes
 - d. i
8. a-b, b-e, c-c, d-a, e-d
9. Now() has a time element, Date() has only date
10. 2
11. Function
 - a. Function textAddNumber (myText As String, myNumber as Long) As String
 - b. textAddNumber = myTest + " " + CStr(myNumber)
 - a. End Function
 - b.
 - c. Function textAddNumber (myText As String, myNumber as Long) As String
 - d. textAddNumber = myTest; " "; CStr(myNumber)
 - c. End Function
12. Sub
 - e. Sub calculate(a As Integer, b String, c String)
 - f. Dim houseNo As Integer
 - g. Dim teleNum As String
 - h. Dim Surname As String
 - i. Dim d As String
 - j. D = CStr(houseNo) + telNum + Surname
 - k. Debug.print d
 - l. End Sub
13. eg. DLookup("[surname],[pupils]","id=1192")
14. eg. DCount("*",[students], "left([telephone],3)="555")
15. see page on dates for answers
16. All are true :)
17. Function
 - a. Function getElement(myArray as Variant, i as Integer)
 - b. getElement = myArray(i)

c. End Function

18. Multi choice

a. A

b. B

c. B

d. C

e. C

f. A

19. Button 3 doesn't have an event procedure, specifically no onClick or onDbClick

20. Multiple answers

a. Optional means name doesn't have to be passed

b. Julia

c. Donothing

d. "Morning Dave. My name is Hal 9000"