Data type	Bytes used	Range of Values
Boolean	2 bytes	**`True`** or **`False`**
Integer	2 bytes	-32,768 to 32,767
Long	4 bytes	–2,147,483,648 to 2,147,483,647
Double	8 bytes	About 15-16 decimal digits of precision
String (fixed length)	Length of string	1 to approximately 65,400 characters
String (variable length)	10 bytes + string length	0 to approximately 2 billion characters
Variant (with numbers)	16 bytes	Any numeric value up to the range of a double data type. It can also hold special values, such as **`Empty`**, **`Error`**, **`Nothing`**, and **`Null`**.
Variant (with characters)	22 bytes + string length	0 to approximately 2 billion
Date	8 bytes	January 1, 0100 to December 31, 9999
Object (ex: Range)	4 bytes	Any object reference
User-defined	Varies	Varies by element

Operators		
+, -, *, /	Addition, subtraction, multiplication, division	`x = 2 + 3`
****	Integer division	`x = 7 \ 2`
=	Assignment, or comparison if inside an If statement	`x = 3` `If x = 3 Then`
<>	Not equal	`If x <> 3 Then`
<, >, <=, >=	Less than, greater than, less than or equal to, greater than or equal to	`If x >= 3 Then`

&	String concatenation	`Dim firstName As String`
		`firstName = "Alan"`
		`MsgBox "Hello, " & firstName`

And, Or, Not	Logical and, logical or, logical negation	`If x > 0 And x < 10 Then` `If x Is Not True Then`
Mod	Modulus	`If x Mod 2 = 0 Then`
()	Function parameters and array access	`Dim scores(4) As Integer` `scores(0) = 90`
'	Comment	`x = 2 + 2 'should equal 4`

Built-in functions, properties, and constants		
`MsgBox()`	Shows a message box.	`MsgBox "Hello"`
`InputBox()`	Prompt for user input.	`Dim name As String` `name = InputBox("Enter name")`
`Len()`	Get the length of a string.	`Dim strLen As Integer` `strLen = Len("Hello")`
`IsNumeric()`	Checks if a value is a number.	`Dim isNum As Boolean` `isNum = IsNumeric("123")`
`Now(), Date(), Time(), Timer()`	Gets the current date/time.	`MsgBox Now`
`Trim()`	Removes leading and trailing spaces.	`Trim(myStr)`
`Unload()`	Removes a Form or UserForm from memory.	`Unload Me`
`ActiveCell`	The active cell.	`ActiveCell.Value = "I'm here!"`
`ActiveSheet`	Active chart sheet or chart contained in ChartObject or a worksheet.	`ActiveSheet.Name = "Summary"`
`ActiveWindow`	The active window.	`ActiveWindow.Zoom = 120`
`ActiveWorkbook`	The active workbook.	`ActiveWorkbook.Save`
`Application`	Excel as a whole.	`Application.ScreenUpdating = False` `Application.UserName`
`Selection`	Could be a `Range` object (cell or cells), `Shape`, `ChartObject`. Can	`Selection.Font.Bold = True`

	set **Selection** with **Range().Select**.	
ThisWorkbook	Workbook containing the VBA procedure in execution.	**MsgBox ThisWorkbook.Name**
Worksheets()	An object of the worksheet specified by the argument.	**Worksheets("Sheet1").Range("B1").Value = "Hello"**
Range()	Refers to one or more specific cells by their address.	**Range("A1").Value = "Hello"** **Range("A1:B2").Font.Bold = True**
Cells()	Refers to a cell by row and column number, instead of the letter-number format. First argument is row number, second is column number.	**Cells(1, 1).Value = "Top left"**
Offset()	Returns a **cell relative to another cell**, by shifting **rows and columns**. First argument is row offset, second is column offset.	**Range("A1").Offset(1, 0).Value = "Below A1"**
xlUp	For finding last used row. Goes up from the current cell until it hits a non-empty cell or row 1. Like ctrl + up arrow key.	**Dim lastRow As Long** **lastRow = Cells(Rows.Count, 1).End(xlUp).Row**
xlDown	Goes down until it hits a non-empty cell or the last row. Like ctrl + down arrow key.	**Dim bottomCell As Range** **Set bottomCell = Range("A2").End(xlDown)**
xlLandscape	Landscape page orientation.	**Sub SetToLandscape()** **ActiveSheet.PageSetup.Orientation = xlLandscape** **End Sub**
xlPortrait	Portrait page orientation.	**Sub SetToPortrait()** **ActiveSheet.PageSetup.Orientation = xlPortrait** **End Sub**
vbYes, vbNo, vbYesNo, vbOKOnly, vbInformation	MsgBox styles.	**Dim ans As Boolean** **ans = MsgBox("Continue?", vbYesNo)**
vbCrLf	Newline character (note you can't use "\n").	**MsgBox "Hello," & vbCrLf & "How are you?"**

<table>
<tr><th colspan="3">User Forms</th></tr>
<tr><td>Checkbox

ComboBox

ListBox

CommandButton

Frame

Image

Label

Multipage

OptionButton

TextBox</td><td>Lets the user select or deselect an option.

A dropdown list the user can pick from or type into. Very useful!

Presents a list of items, and the user can select an item (or multiple items).

A clickable button that runs code (such as one that submits the form and one that cancels).

Groups related controls (like checkboxes or option buttons).

Displays a picture

Displays static text or a description.

A control with multiple pages/tabs, like tabs in a settings dialog.

A radio button — users can select only one from a group.

Allows the user to type text input.</td><td><pre>
Private Sub UserForm_Initialize()
    ' Populate ComboBox
    ComboBox1.AddItem "Red"
    ComboBox1.AddItem "Green"
    ComboBox1.AddItem "Blue"

    ' Default selections
    OptionButton1.Value = True
    CheckBox1.Value = False

    MultiPage1.Pages(0).Caption = "Page 1"
    MultiPage1.Pages(1).Caption = "Page 2"
End Sub

Private Sub CommandButtonOK_Click()
    Dim message As String
    message = "Name: " & TextBox1.Text & vbCrLf
    message = message & "Favorite Color: " &
ComboBox1.Value & vbCrLf
    message = message & "Subscribed: " &
IIf(CheckBox1.Value, "Yes", "No") & vbCrLf

    If OptionButton1.Value Then
        message = message & "Selected: Option A"
    ElseIf OptionButton2.Value Then
        message = message & "Selected: Option B"
    End If

    MsgBox message, vbInformation, "Form Results"
    Unload Me
End Sub

Private Sub CommandButtonCancel_Click()
    Unload Me
End Sub
</pre></td></tr>
</table>

Declaring and using variables and functions

A local variable can be declared in VBA using the following syntax:

```
Dim <variable name> As <type>
```

Dim stands for Dimension, a reference to when it was mostly used for arrays. Local variables can only be used in the procedure in which they're declared. In general, naming the type is not required, and if it's not specified, the type will by default be a Variant. But, if you want to enforce that all variables are declared (recommended), include this at the top of the module:

```
Option Explicit
```

You can also declare multiple variables in a single line, as shown:

```
Dim x As Integer, y As String
```

Subsequent assignments to previously declared variables do not need to be preceded by **Dim**.

You can declare global variables that are available to all procedures like so:

```
Public CurrentRate as Long
```

Const
Variables declared with const can't be changed after being declared. Example:

```
Const Pi As Double = 3.14159
```

Arrays
Example of declaring a fixed-size one-dimensional array of 5 integers:

```
Dim scores(4) As Integer
```

Example of declaring a multi-dimensional 3x3 array of integers:

```
Dim grid(1 To 3, 1 To 3) As Integer
```

Example of declaring a dynamic one-dimensional array:

```
Dim nums() As Integer
```

It can be re-sized later, as shown:

```
ReDim nums(1 To 10)
```

It can be re-sized while keeping the existing values, as shown:

```
ReDim Preserve nums(1 To 20)
```

Procedures
The syntax for a function procedure is the following:

```
Function <function name>(<parameter name 1> As <parameter type 1>,
<parameter name 2> As <parameter type 2>)
        <statements>
        <function name> = <return value>
End Function
```

Somewhere inside the function procedure, you need to assign a value to **<function name>**. After the last statement of the function procedure executes, the value of **<function name>** is returned to the caller.

If you want, you can declare a procedure as **Private** (for example, **Private function add(num1 As Double, num2 As Double)**) so that it can only be used in the same module that it was defined. If it's not specified, procedures are by default **Public**, meaning they can be used in other modules outside of the one they were defined in. Also note that you don't use Dim when declaring procedure parameters.

A sub procedure (sub is short for subroutine) is like a function procedure except it doesn't return a value. The syntax is the following:

```
Sub <subroutine name>(<parameter name 1> As <parameter type 1>,
<parameter name 2> As <parameter type 2>)
        <Statements>
End Sub
```

Unlike functions, subroutines cannot be used directly in worksheet cell formulas.

Note that if you're calling a procedure but ignoring its return values, you can do

```
<procedure name> <parameter 1>, <parameter 2>
```

Or

```
Call <procedure name>(<parameter 1>, <parameter 2>)
```

If you're using the return value, do:

```
<variable name> = <procedure name>(<parameter 1>, <parameter 2>)
```

Objects
An Object variable can represent things like a range or an entire worksheet. They can be declared as normal using **Dim** or **Public** as shown:

```
Dim InputArea As Range
```

One difference is that when assigning a value to an Object variable, you need to use the Set keyword as shown:

```
Set InputArea = Range("C16:E16")
```

User-defined Types
Users can define their own Type, like a struct in C, as shown:

```
Type Student
      Name As String
      Age As Integer
      GPA As Double
End Type
```

The user-defined Type can then be declared and its fields accessed as shown:

```
Dim s As Student
s.Name = "Alan"
```

You can also modify it like this, which executes faster (also works on built-in VBA objects with multiple properties):

```
With s
    .Name = "Alan"
    .Age = 22
    .GPA = 3.73
End With
```

Conditionals
If Then

```
If <test expression> Then
      <statements>
ElseIf <test expression> Then
      <statements>
Else
      <statements>
End If
```

The **`ElseIf`** and **`Else`** blocks are optional.

IIf

```
IIf(<test expression>, <return value if true>, <return value if false>)
```

Select Case

```
Select Case <expression>
Case <test expression>
      <statements>
Case Else
      <statements>
End Select
```

Some examples of **<test expression>** are **Is Null**, **0 To 24**, and **Is >= 75**.

Loops

For Each-Next

```
For Each <element> In <collection>
      <statements>
Next <element>
```

Note that the loop can be exited early by using the **Exit For** statement.

For-Next

```
Dim <variable name> As <number type>
For <variable name> = <start> To <end>
      <statements>
Next <variable name>
```

Note that the **<end>** value is inclusive.
Optionally, you can include **Step <count>** after **<start> To <end>** to define a step.

Do While

```
Do While <test expression>
      <statements>
Loop
```

You can also put **While <test expression>** after **Loop** instead of **Do**.
Note that **While <test expression>** can be omitted to make it an infinite loop.
Also note that the loop can be exited early by using the **Exit Do** statement.

Do Until

```
Do Until <test expression>
      <statements>
Loop
```

Other notes

- To make the Developer tab show up in Excel, File > Options > Customize Ribbon, then check the box next to Developer under Main Tabs and hit OK. Now you can click Developer > Visual Basic to open the VBA code editor. Create a new module with Insert > Module or a new UserForm with Insert > UserForm.
- Workbooks containing VBA code should be saved as .xlsm files.
- If running macros is blocked due to security, enable running macros by right clicking the workbook in your files, clicking Properties, and at the bottom of the General tab select Unblock, then Apply.
- If you want to write code that can be run anywhere in the workbook (such as any sheet within the workbook), write the code in ThisWorkbook. If you want to write some code that can only be run in a specific sheet, write the code in the corresponding sheet code file. But these are generally used for writing code that runs when certain events occur such as opening the sheet/workbook; in general, you'll want to code in a module.
- To insert a button into an Excel sheet, go to Developer > Insert > Button under Form Controls. After dragging the area the button will occupy, you will be prompted to assign a macro, which is where you can choose what public procedure you want to run when the button is clicked. You can change the button's text or re-assign the macro by right-clicking the button.

- Each CommandButton in a user form has an event handler that runs when it is clicked with this syntax:
    - ```
      Private Sub <subroutine name>_Click()
            <statements>
      End Sub
      ```
- You can run **`<UserForm name>.Show`** to open a UserForm, **`Me.Hide`** to hide a UserForm, and **`Unload Me`** to delete it from memory so that the next time that form is opened up, it won't have the data that was filled in last time.

- The Record Macro button in the Developer tab will automatically save the actions you took during the recording as VBA code that you can inspect.

- During intensive writing to a spreadsheet, it's a good idea to precede it with **`Application.ScreenUpdating = False`** and afterwards put **`Application.ScreenUpdating = True`** to reduce lag and make it faster.
- This:
    - ```
      For i = 1 to NumElements
            Cells(i, 1) = MyArray(i)
      Next i
      ```

  Is much slower than this (copying a variant/array to a range):
    - ```
      TheRange.Value = MyArray
      ```
      ```