# Quiz 1 solutions and explanations

**IMPORTANT: Even if you do not intend/need to look through the solutions to Quiz 1, you still need to mark this quiz as completed using the blue "Mark as Completed" button in the lower right of this page. By doing so, you will unlock Assignment 1!**

This document is meant to provide clear explanations for the Quiz 1 questions. I do NOT provide feedback during the quiz (like I do for the screencasts) because a learner could just guess, obtain the correct answers, then put them back into the quiz and get 100%!

This document is purely for you to learn more and to correct your misconceptions about the material. If you view this document soon after you take the quiz to see why you missed a certain question, it will serve as a great learning tool!

**PLEASE DO NOT SHARE THIS DOCUMENT WITH ANYONE!** *Using this document to complete Quiz 1 is a violation of Coursera's Honor Code (a.k.a. cheating).*

**Question 1:**

Which of the following statements are true regarding subroutines and functions in VBA? Select all that are true.

A. You wish to use VBA to delete rows of the spreadsheet in which the cells in column B are empty. This is best done using a VBA subroutine as opposed to a function.

Correct! Functions return values or ranges of values in place of their name. For example, we type "**=SomeFunction(x,y)**" into cell **C7** of a worksheet, and the value of that function is returned to cell **C7**. Or, we highlight cells **B1:B5** and type in "**=SomeArrayFunction(A1:A5)**", press **Ctrl-Shift-Enter**, and the results of multiple calculations are returned to cells **B1:B5**. Functions are not used to make changes to the layout of a worksheet, such as deleting entire rows or columns.

B. Functions must always have at least one argument and subroutines never have arguments.

Incorrect. Functions do not have to have arguments. Two examples are the **PI()** and **NOW()** functions in Excel (and you could create user-defined VBA functions similarly). Subroutines *can* have arguments. An example was shown in one of the screencasts in which we delete the entire row of the worksheet, but we pass the row number into the subroutine to specify the row number – maybe something like "**Sub DeleteRow(nr)**". Or, another example would be a subroutine that would create an m x n array of random integers starting in cell C: "**Sub CreateArray(C,m,n)**". This would have to be called in another subroutine, FYI.

C. Subroutines must always have at least one argument and functions never have arguments.

Incorrect. Most subroutines do NOT have arguments and most functions DO have arguments.

D. Functions return a value in place of their name, and this value can be used directly in a procedure.

Correct! If we had a function **SomeFunction(x)** that squared x then added 5, we could directly use that result either in a cell on the worksheet OR in another function or subroutine. As a simple example, in some subroutine we could write "**MsgBox SomeFunction(5)**" and the output of the **SomeFunction(5)** would be outputted in a message box.

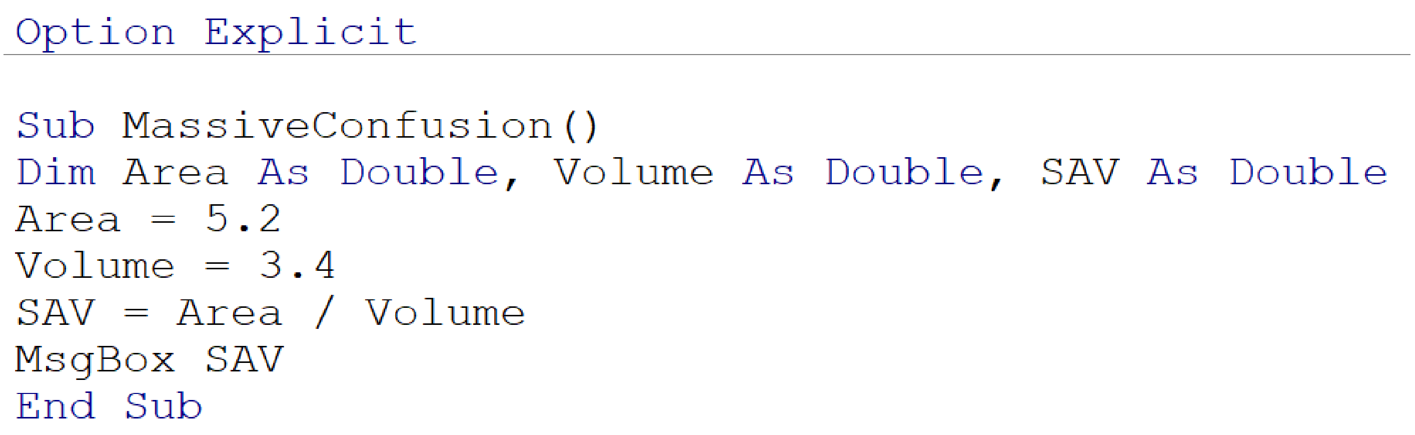
E. Functions can only be used in the cells of the spreadsheet.

Incorrect. See the explanation to option D above. We can use functions in other functions or subroutines.

**Question 2:**

Which of the following blocks of code would result in a compiler error if executed?

A.

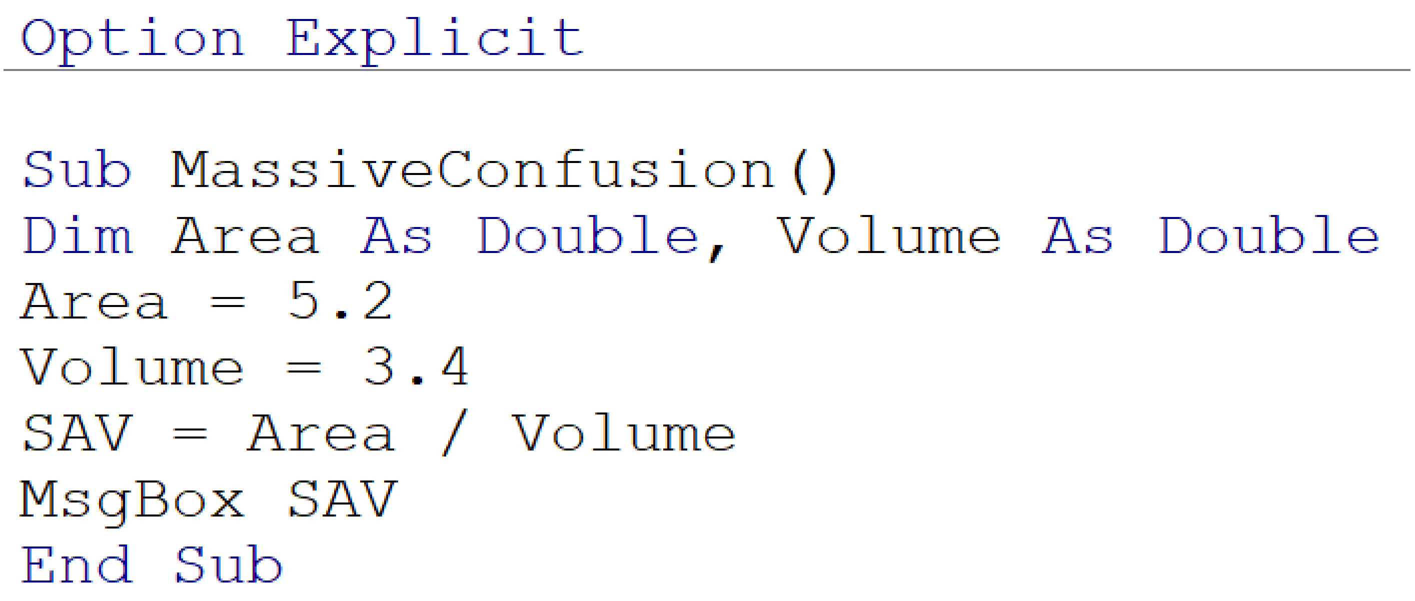


```
Option Explicit

Sub MassiveConfusion()
Dim Area As Double, Volume As Double, SAV As Double
Area = 5.2
Volume = 3.4
SAV = Area / Volume
MsgBox SAV
End Sub
```

Incorrect. There are no errors here. Since the**Option Explicit** statement appears at the top of the module, all variables must be **Dim**'med. All variables are, in fact, **Dim**'med so no errors. Furthermore, there are no misspelled statements or variables.

B.

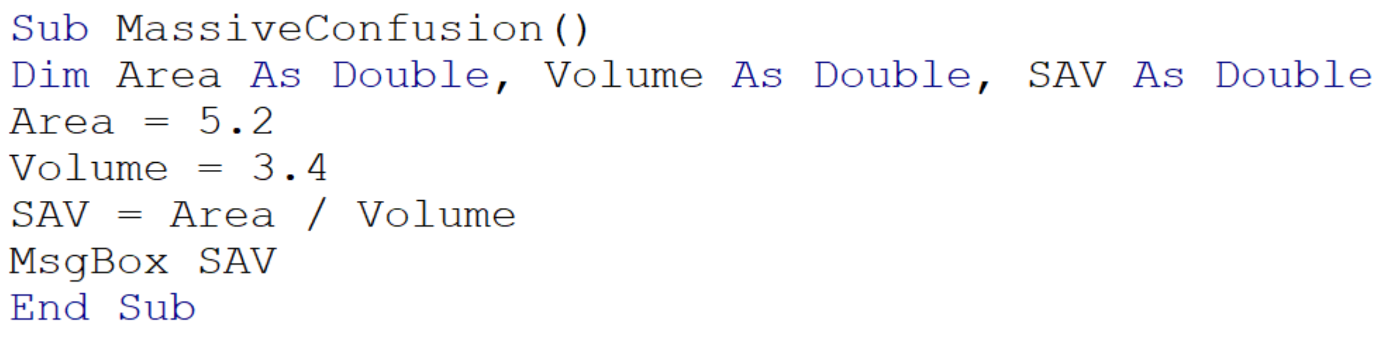


```
Option Explicit

Sub MassiveConfusion()
Dim Area As Double, Volume As Double
Area = 5.2
Volume = 3.4
SAV = Area / Volume
MsgBox SAV
End Sub
```

Correct! Since the **Option Explicit** statement appears at the top of the module and **SAV** is not **Dim**'med, there would be a compiler error here.

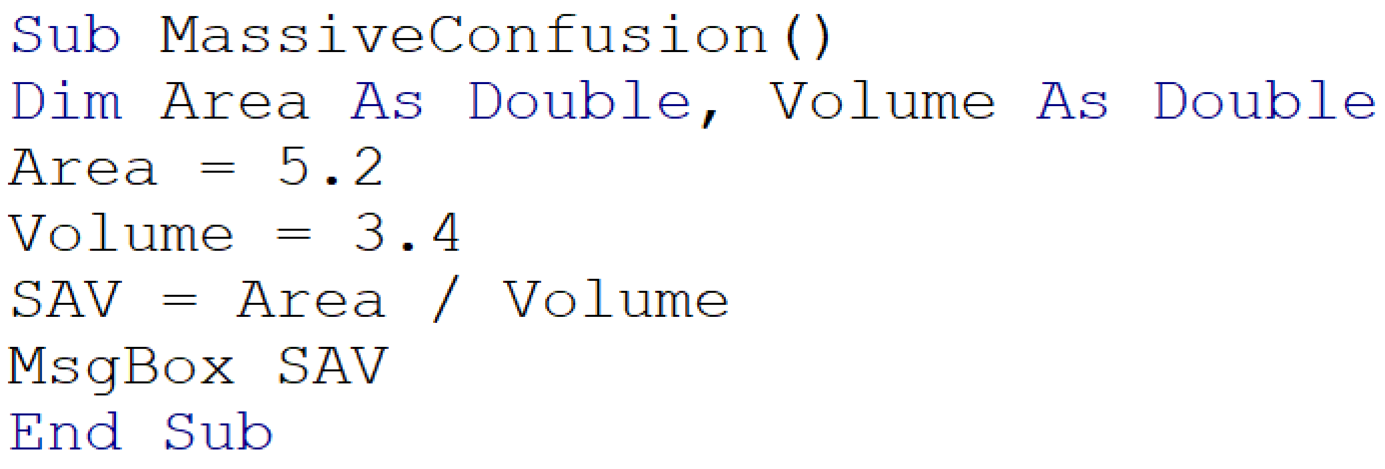
C.



```
Sub MassiveConfusion()
Dim Area As Double, Volume As Double, SAV As Double
Area = 5.2
Volume = 3.4
SAV = Area / Volume
MsgBox SAV
End Sub
```

Incorrect. There are no errors in the code, but even if there were, because we don't have **Option Explicit** at the top of the module, there wouldn't be any compiler errors.

D.



```
Sub MassiveConfusion()
Dim Area As Double, Volume As Double
Area = 5.2
Volume = 3.4
SAV = Area / Volume
MsgBox SAV
End Sub
```

Incorrect. Although **SAV** has not been **Dim**'med, we do not have **Option Explicit**, which requires variable declaration, so there will not be a compiler error here.

E.

```
Sub MassiveConfusion()
Dim Area As Double, Volume As Double
Area = 5.2
Volume = 3.4
SAV = Arae / Volume
MsgBox SAV
End Sub
```

Incorrect. Area has been misspelled on line 5 and **SAV** is not **Dim**'med. However, because we don't have **Option Explicit** written on the top of the module, there won't be any errors. This is why we use **Option Explicit** – to catch mistakes like this!

**Question 3:**

If the subroutine shown below is run consecutive times, what will be displayed in the message box just before the program enters into debug mode?

```
Option Explicit

Sub IDontLikeQuizzes()
Static i As Integer
MsgBox i
i = 2 * i + 1
Debug.Assert i < 30
End Sub
```

Let's start with the **Debug.Assert** command. When that becomes FALSE, the subroutine enters into debug mode. i will start at 0. i is a Static variable, which means that even after the sub is ended the previous value of i will be stored (until the workbook is closed). The first message box will display 0. Then, i will be incremented to 1 (2*i+1). i is still less than 30 so it will not go into debug mode. The 2nd time the sub is run, i will be set to 3 (2*i+1). Still less than 30. 3rd time the sub is run, i will be set to 7 (still less than 30). 4th time, i will be set to 15 (still less than 30). on the 5th time, i of 15 will be displayed in the message box, and this occurs *before* i is set to 31. After this last message box, i will be set to 31 then the sub will enter debug mode. So, **15** is the last value displayed in the message box before debug mode is entered.

## Question 4:

Which of the following statements are true regarding the topics that you learned in this module? Select all that are true.

A. One way to make a variable available to multiple procedures (subroutines and functions) is to declare it as **Public**.

Correct! This statement is true.

B. Arguments to functions should never be **Dim**'med inside the function using **Dim** statements.

Correct! For example, if we had "**SomeFunction(x As Double) As Double**", if on the first line inside the function we wrote "**Dim x as Double**", then this would automatically reset x to 0 and NOT use the value of x that was input into the call to the function. The compiler unfortunately does not give an error, so you have to be aware of this!

C. If cell **B4** is selected and formatted bold during macro recording in relative referencing mode, cell **B4** will always be formatted when the macro is executed.

Incorrect. In relative referencing mode, what matters is the relative position of cells to one another during recording. For example, if we started in cell **A3** during macro recording (relative referencing mode) and selected **B4** and formatted cell **B4**, what really matters is the relative position of cell **B4** to **A3**(one row down, one column over). So, if we start in cell **D6** when we run the macro, the macro will format the contents of cell **E7** (one row down, one column over).

D. "**Debug.Assert n > 20**" will cause debugging mode to commence when **n** is greater than 20.

Incorrect. Debugging begins when the conditional statement that follows "Debug.Assert" becomes FALSE. So, "Debug.Assert n > 20" will cause debugging to commence when n is less than or equal to 20.

E. Most subroutines do not have arguments.

Correct! The vast majority of subroutines do not use arguments.

## Question 5:

Cell **B3** was set as the active cell and the following steps were taken:

A macro recording was initiated (referencing mode was set to absolute referencing - the default).

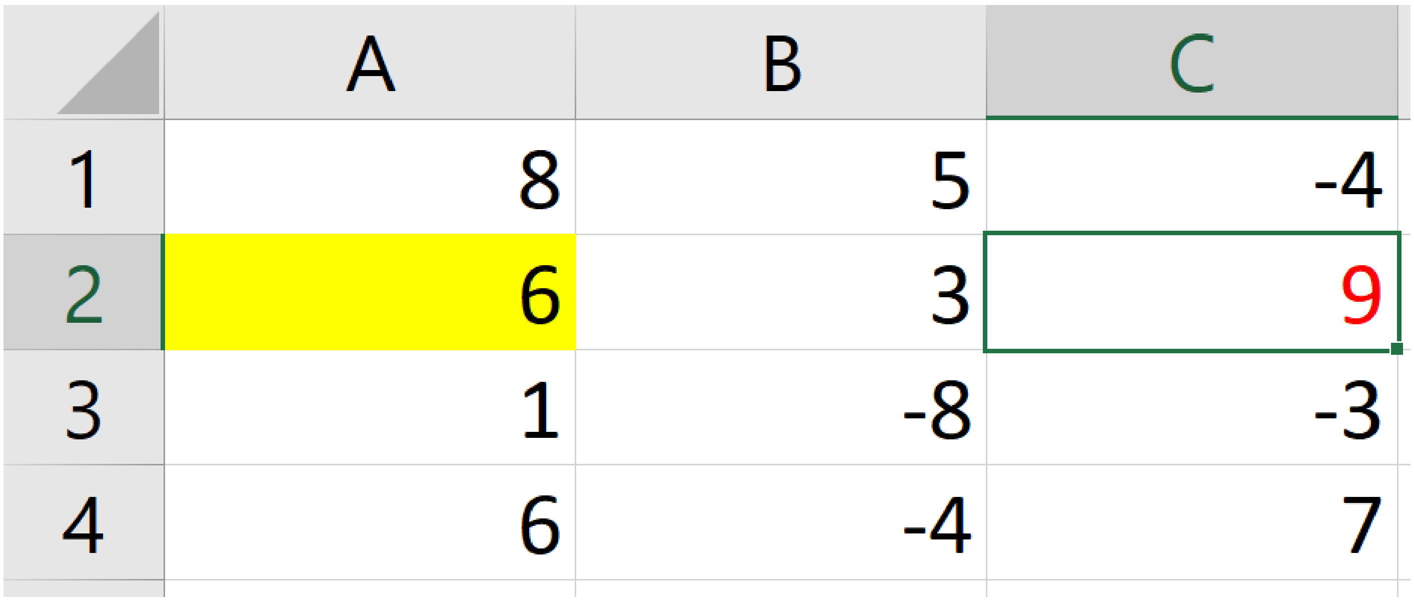
1. Cell **A2** was highlighted yellow.

2. Referencing mode changed to relative referencing.

3. The font color of cell **C2** was changed to red.

4. The recording was stopped.

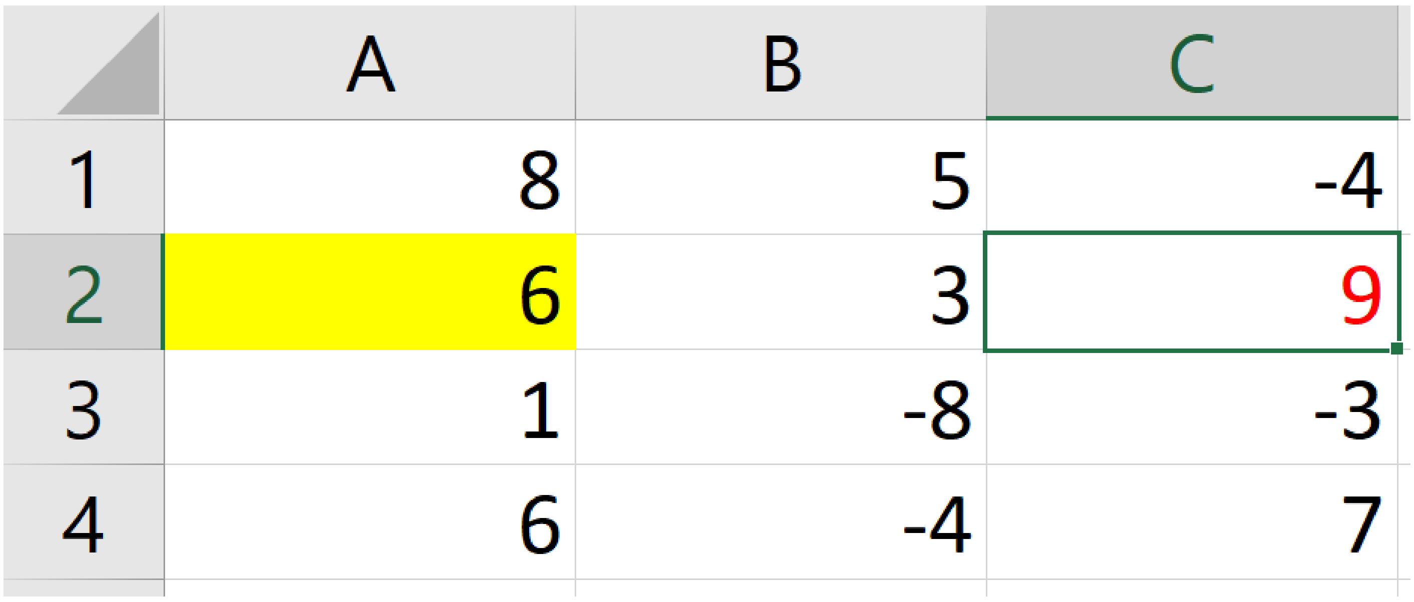
This resulted in the following appearance:



| | A | B | C |
|---|---|---|---|
| 1 | 8 | 5 | -4 |
| 2 | 6 | 3 | 9 |
| 3 | 1 | -8 | -3 |
| 4 | 6 | -4 | 7 |

The formatting in the cells is then reset to the initial formatting (nothing formatted yellow nor red). Which of the following represents the appearance of the spreadsheet when cell **B4** is selected and the macro is executed? **NOTE: Many learners get this problem incorrect (and email me to indicate that the correct answer is not available - but it definitely is!), so make sure you are reasoning through the answer!!**

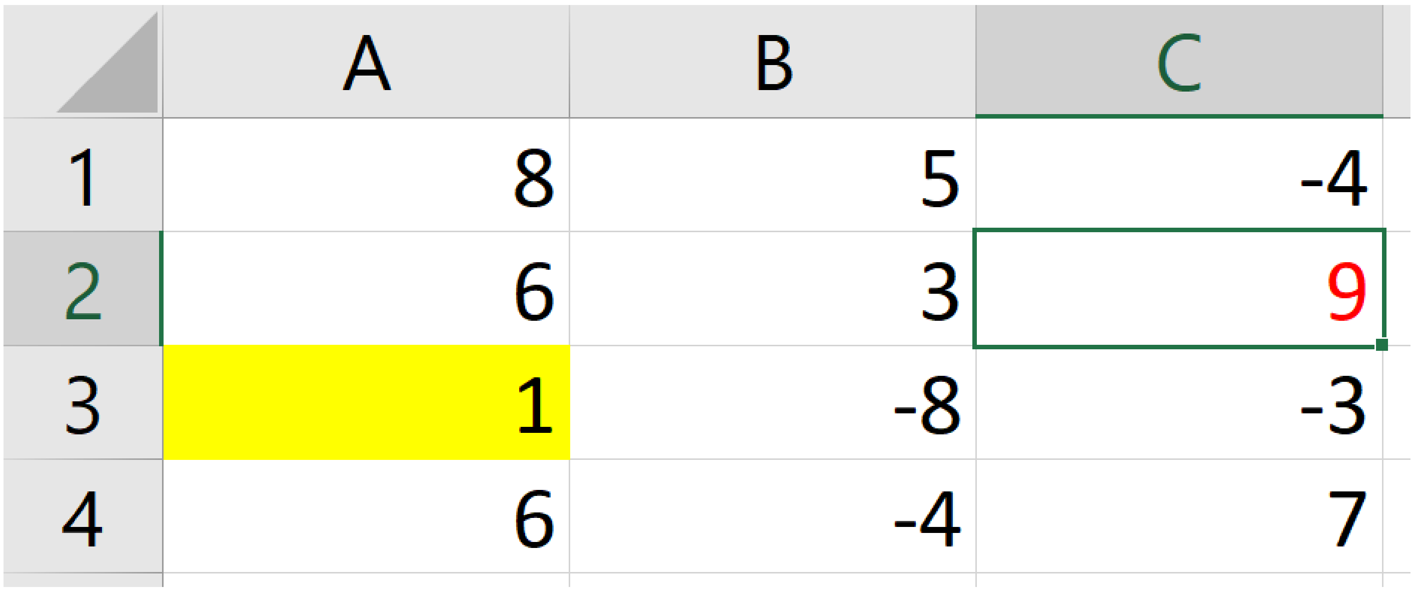
A.



| | A | B | C |
|---|---|---|---|
| 1 | 8 | 5 | -4 |
| 2 | 6 | 3 | 9 |
| 3 | 1 | -8 | -3 |
| 4 | 6 | -4 | 7 |

Correct! Cell **B3** was the active cell when the recording was initiated, and the recording mode was absolute referencing. The first step was to select cell **A2**. Therefore, cell **A2** will always be selected first, since referencing mode was absolute (it doesn't matter which cell we started macro recording with). Next, the referencing mode was changed to relative referencing, and cell **C2** was formatted. **C2** is on the same row but 2 columns right of cell **A2**, but since **A2** will always be selected in the previous step (regardless of the active cell when the macro is run), cell **C2** will always be selected in this step (it's still a relative reference, but always starts on cell **A2** – tricky!).

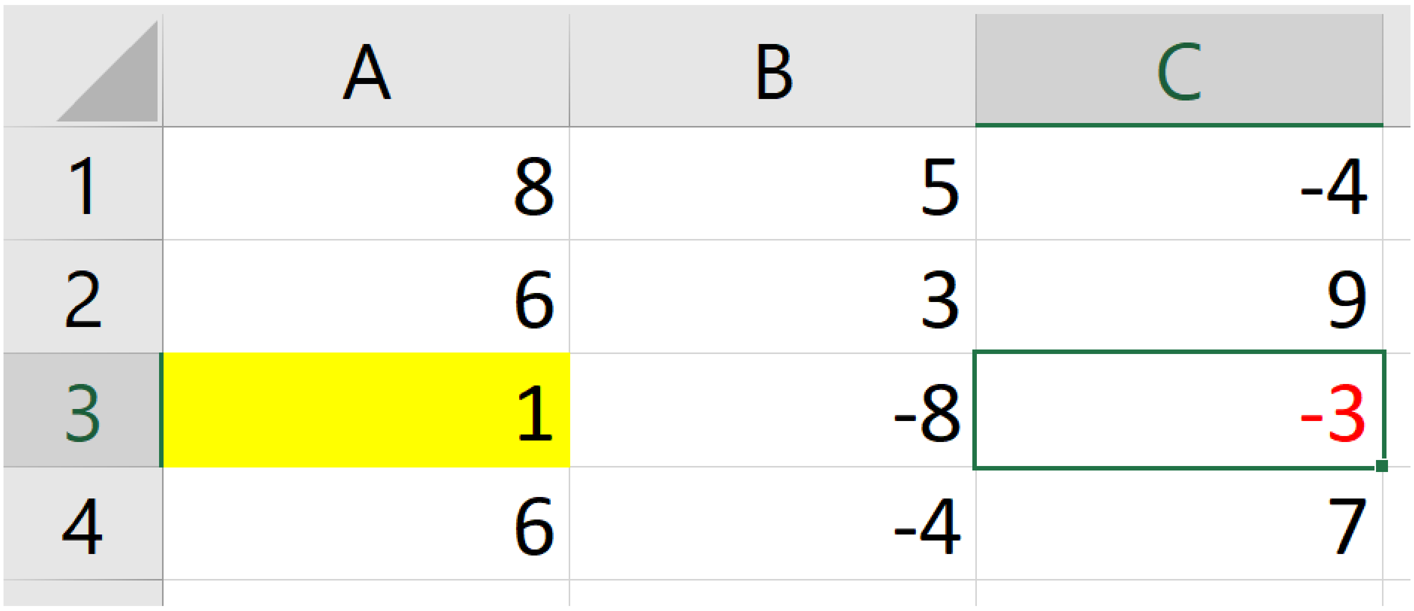
B.



| | A | B | C |
|---|---|---|---|
| 1 | 8 | 5 | -4 |
| 2 | 6 | 3 | 9 |
| 3 | 1 | -8 | -3 |
| 4 | 6 | -4 | 7 |

Incorrect. Cell **B3** was the active cell when the recording was initiated, and the recording mode was absolute referencing. The first step was to select cell **A2**. Therefore, cell **A2** will always be selected first, since referencing mode was absolute (it doesn't matter which cell we started macro recording with). Next, the referencing mode was changed to relative referencing, and cell **C2** was formatted. **C2** is on the same row but 2 columns right of cell **A2**, but since **A2** will always be selected in the previous step (regardless of the active cell when the macro is run), cell **C2** will always be selected in this step (it's still a relative reference, but always starts on cell **A2** – tricky!).

C.



| | A | B | C |
|---|---|---|---|
| 1 | 8 | 5 | -4 |
| 2 | 6 | 3 | 9 |
| 3 | 1 | -8 | -3 |
| 4 | 6 | -4 | 7 |

Incorrect. Cell **B3** was the active cell when the recording was initiated, and the recording mode was absolute referencing. The first step was to select cell **A2**. Therefore, cell **A2** will always be selected first, since referencing mode was absolute (it doesn’t matter which cell we started macro recording with). Next, the referencing mode was changed to relative referencing, and cell **C2** was formatted. **C2** is on the same row but 2 columns right of cell **A2**, but since **A2** will always be selected in the previous step (regardless of the active cell when the macro is run), cell **C2** will always be selected in this step (it’s still a relative reference, but always starts on cell **A2** – tricky!).