Learning to Script with AutoIt 3

Alex Peters
http://yallara.cs.rmit.edu.au/~apeters/

Last updated February 13, 2006

This document is for you if you are interested in learning to script with AutoIt, as the title implies. It tries to assume no prior coding experience and it aims to teach some good general coding habits as well, which will be beneficial should you decide to move on to other languages.

This document is nowhere near complete—I need feedback in order to come close. What do you feel needs more or better explanation? Have I failed to cover something adequately or perhaps even at all?

I encourage any form of feedback whatsoever. You can reach me via my website (listed above) or via the AutoIt forums (I'm LxP):

www.autoitscript.com/forum

Contents

Acknowledgements

`herewasplato` pointed out problems pasting code from the document into a script, prompted better explanation of strings and suggested the addition of exercise solutions.

1 What you must do before proceeding

1.1 Have AutoIt installed

You will find the official AutoIt installer here (as at February 13, 2006):

`www.autoit3.com/downloads.php`

While you also have the option of downloading a ZIPped distribution, I strongly advise using the installer anyway—many advantages are offered including easier access to the help file, the ability to double-click AutoIt scripts to start them and painless use of the included code library within your scripts.

1.2 Know how to access the help file

After installing AutoIt you can access the help file like this:

Start → **Programs** → **AutoIt v3** → **AutoIt Help File**

It's important to get comfortable with the help file because it will be your primary source on how to use any specific function. My document will teach you how to use functions in general (and how to write your own too), but it will not discuss the particulars of each one.

1.3 Complete AutoIt's 'Hello World' tutorial

The *Hello World* tutorial can be found in the help file:

Contents → **AutoIt** → **Tutorials** → **My First Script (Hello World)**

It will teach you how to create an AutoIt script, type code into it and run it; knowledge of these details is assumed by my document. Many references to this tutorial will also be made later so please keep your finished result handy.

1.4 Know what you want to achieve

This is an optional step but if you have an idea in your mind of what you'd like to do with AutoIt before you start, things that will aid you will stand out as you read them and perhaps they'll sink in a little further as well.

Here are some ideas to jog your imagination:

- Start multiple programs and arrange them nicely on the screen
- Start a single program, adjust some values on its window and then hide it
- Periodically perform an action in some open program (without interrupting any work taking place at that time)
- Shut down the computer after some lengthy operation (like a backup or perhaps a download) completes
- Automate some 'housekeeping' chores such as backing up important files and emptying the Recycle Bin

2 Programming—it's all about values

2.1 How values are used

This is only a theory of mine and it may be wrong, but in my opinion the most essential thing to realise when it comes to programming is this: it's not much more than a game of throwing *values* around. I'll try to justify this claim:

- The script made as part of the *Hello World* tutorial (see section 1.3) instructs AutoIt to display a message box. To do this three *values* are needed: one dictating the style of the box (like icon and buttons), one for the title and one for the message.
- Suppose that you wanted to write a simple number guessing game. You would have a *value* for the number to guess and a *value* for what the user has entered as their guess. You would compare those two *values* to determine if the user has made a correct guess, and then you would probably display one of two human-readable messages informing them of the outcome, which are *also values*.
- On the very other end of the spectrum, perhaps you want to write a utility that updates the Windows operating system. You would have *values* to indicate which files are to be manipulated. You would open the files specified by those values and search for certain *values* in each file to determine whether you need to make adjustments. If you do, you would overwrite some part of each file with a new set of *values*.

It can be seen rather clearly from the above scenarios that values can take many forms: numbers, characters, strings of characters, binary data or even a combination. For instance, these are all potential values:

`123`   `abc`   `0.5`   `Z`   `My name is Alex`   MZπ♠_⋆.à⊣

In my opinion, understanding and thinking about things in terms of *values* should make any coding task much easier. Consider what values might be involved for any task that you plan to undertake; by knowing what values will be needed, you'll be better prepared to know exactly how these values must be manipulated to get the job done. (You'll also know exactly what to look for in the help file since AutoIt instructions are named according to what they do.)

2.2 Different types of values

Having completed the *Hello World* tutorial (see section 1.3), you have already used the two most important value types: *numeric* and *'string'* values. These two value types can be used to represent anything that might need to be coded directly into a script.

Strings are sequences of symbols, like a word or a sentence. They can be used to represent names, codes, human-readable instructions or anything else that can be represented in a textual form (like *all* of the 'potential values' shown above). Even numbers can be represented in string form—but only if you don't plan to manipulate them mathematically.

2.3 Representing values in AutoIt code

2.3.1 Numbers

Numeric values are entered into a script in the same fashion as one would enter numbers into spreadsheet software—that is, *digit grouping symbols* (also known as *thousand separators*) are not used, and numbers are made negative by prepending them with a *minus sign* (`-`).

Regardless of the standard notation for your region, a *period* is used to represent a decimal point (e.g. `123.45` and not `123,45`).

2.3.2 Strings

Since the value of a string can be virtually anything, AutoIt needs to be told where strings start and end. This is necessary to prevent symbols belonging to the string from being interpreted by AutoIt in another way, and is done by placing a *single quote* (`'`) or *double quote* (`"`) at both ends of the string. (You may have noticed that the script from the tutorial uses double quotes.)

When a string doesn't contain either type of quote, whichever one you choose to surround the string makes no difference (personally I prefer the tidier-looking single quotes). If a string should contain one type of quote but not the other then it makes sense to use that other type of quote to surround the string.

What happens though if the string should contain both types of quote? In such a case you must nominate which quote you will use to surround the string, and then *double up* on any occurrences of that quote within the string. Here's an example of a line of text containing both types of quote:

```
I'm a string with single and "double" quotes.
```

If we wanted to surround the string with *double quotes* then we would double up on the double quotes within the string, giving us this:

```
"I'm a string with single and ""double"" quotes."
```

If we wanted to instead use *single quotes* then we would double up on the single quote and this would be the result:

```
'I''m a string with single and "double" quotes.'
```

Both of these are complete strings and you are invited to replace one of the strings in the Hello World tutorial with one of these to see the result.

2.4 Exercises

1. Which of the potential values on page 5 can be represented by the *number* value type? Which can be represented as *strings*?

2. Modify the *Hello World* tutorial to instead display the following messages exactly as written below, surrounding the strings in *double quotes* as per the original code:

   (a) `This is a message.`

   (b) `Yoko's in Tokyo.`

   (c) `I said to him, 'Have some pudding.'`

   (d) `He'll have a "royal" time. Ahahahaha.`

   (e) `"This" message begins with a double quote!`

3. Repeat the exercise above, this time surrounding the strings with *single quotes*. (Some other adjustments will also be necessary.)

4. Modify the number passed to `MsgBox()` such that the resulting message box has the following characteristics (refer to the help file for the necessary combinations):

   (a) a *Question* icon and *Yes* and *No* buttons

   (b) a *Stop sign* icon and *OK* and *Cancel* buttons

   (c) an *exclamation point* icon and *Retry* and *Cancel* buttons, with the *Cancel* button selected by default instead of the *Retry* button

5. What characteristics will the numeric value `310` give a message box?
   (Can you determine this without trying it to see the result?)

6. Create a *new script* that, using an *empty string*, invokes a message box with an *empty* title.
7. Write a single script that displays two message boxes, one after the other.
   (This is done by writing two separate `MsgBox()` lines.)

Solutions to these exercises can be found on page 14.

3 Functions

When you get around to writing your own AutoIt scripts—which is soon—you will find that the majority of any work is done by calling *functions*. A function accepts zero or more *parameters* (i.e. input values), does its processing and then gives a *return value* (i.e. an output value) as a result.

Here's an example function call (which should look familiar):

```
MsgBox(64, 'Tutorial', 'Hello world!')
```

The *name* of the function is written to the left of the brackets. The *parameters* are entered within the brackets and are separated by *commas*. Here, we are passing *three* values to the `MsgBox()` function.

Functions may or may not have a set number of parameters—for instance, there is actually a *fourth* argument to `MsgBox()` which assigns a *timeout* to the display of the message box. As it is an *optional* parameter, we have not used it. The help file states which parameters are optional for each function and what will be assumed if you do not specify them (in the case of `MsgBox()`, if a timeout isn't specified then the box will show indefinitely).

If a function doesn't want any parameters at all then the brackets remain there, but are left empty:

```
FunctionWithoutInputs()
```

The return value of the `MsgBox()` function is a *number* representing which button was pressed, but in the above examples we don't actually catch any return values to use them; they are lost as a result. This is for two reasons:

1. It demonstrates that return values *can* be ignored.
   (We don't need to know what button the user pressed since we only offered one.)
2. We have no other choice as we have not yet discussed *variables*.

However, it is still possible to write useful scripts even if the output of every function call is ignored. At this point you have been given the knowledge to be able to write scripts that:

- open, close, resize and focus windows;
- adjust controls within them;
- manipulate folders, files and their locations;
- send mouse clicks and keyboard input;
- shut down or restart the computer;
- start programs;
- play sounds through the computer's speakers; or
- take any linear sequence of the above actions.

All you need to do is refer to the AutoIt help file, find the functions you'd like to experiment with and provide them with the right values.

3.1 Example function calls

3.1.1 `Run()` and `RunWait()`

These functions are used to start programs. Using `Run()` instructs AutoIt to continue with the next instruction in the script immediately after the specified program *starts*, while `RunWait()` forces AutoIt to wait until the specified program *ends* before continuing.

Try creating a new script (the tutorial taught you how to do this), pasting the following lines into it and running it:

```
Run('Calc')
Run('Notepad')
MsgBox(64, 'Functions', 'I started Calculator and Notepad.')
```

Notice how both Calculator and Notepad open on your screen, and the message box appears around the same time (possibly beneath the other windows). After closing them all, try changing the `Run()` calls to `RunWait()` and running the script again—you will find this time that Notepad only appears after Calculator is closed, and the message box only appears after Notepad is closed.

3.1.2 `WinWait()`

This function instructs AutoIt to wait until a window with a specific *title* exists. You pass this function a *string* indicating what the title is (or how it starts).

Try this script:

```
Run('Calc')
MsgBox(64, 'Functions', 'Your Calculator is showing.')
```

In its current form it can't be guaranteed that the message box will appear after Calculator appears (especially on slower computers). This can be remedied by adding a `WinWait()` call like so:

```
Run('Calc')
WinWait('Calculator')
MsgBox(64, 'Functions', 'Your Calculator is showing.')
```

Try changing the window title in the script above to `Claculator` and running it. You will find that the message box never shows—why is this?[*]

3.1.3 `Send()`

This function will simulate keyboard input to the window that currently has focus. Try this script (you may need to adjust the value passed to `WinWait()` if Notepad starts with a different title on your system):

```
Run('Notepad')
WinWait('Untitled - Notepad')
Send('AutoIt is typing text{ENTER}into this window.')
```

Notice when you run this that the `{ENTER}` token is replaced with an actual press of the *Enter* key. The appropriate help page will discuss other tokens that give this behaviour, as well as how to send *Ctrl+* and *Alt+* combinations.

3.2 Exercise

Write a script that starts Calculator and sends keys to it to perform the following calculations, leaving the window open to display the answer:

(a) 861747 − 738291

(b) 704277 + 295723

(c) 2 − 1 + 4 − 3 + 6 − 5 + 9 − 8

Hints:

- Sending the '–' key will press the subtraction button.
- Sending the *Enter* key will perform the calculation.
- Sending '+' will press the addition button, but this must be done in a special way or else AutoIt will read this as a request to hold down the *Shift* key. Refer to the help file for how to accomplish this.

Solutions can be found on page 14.

[*] *Answer:* The script will wait indefinitely for a window that will never exist.

4 Variables

Variables are areas in computer memory, identified by name, that hold values. The value contained within a variable could come from the script itself, from a function (through its *return value*), from some other calculation (potentially involving other variables) or from any combination of these during the running lifetime of a script. A variable's value can be read any number of times and it can be overwritten with some new value as often as desired.

4.1 Naming variables

A variable's name consists of a *dollar sign* (`$`) followed by any combination of one or more letters, numbers and the *underscore* character (`_`).[†] Here are some examples of valid (although not necessarily advisable) variable names:

```
$MyVar    $Msg    $WAIT_TIME    $Answer    $_    $MsgBox    $19283
```

Variable names are *case-insensitive*, which means that the names below would all point to the same variable in memory:

```
$ABC    $ABc    $AbC    $Abc    $aBC    $aBc    $abC    $abc
```

4.2 Accessing variable values

To use the value contained by a variable in place of some hard-coded value, simply give the variable's name wherever a value is expected. For instance, when this code is run:

```
MsgBox(0, $MyTitle, $MyMsg)
```

the value contained by `$MyTitle` will be used for the title of the resultant message box (since the second parameter to `MsgBox()` sets the title) and the value contained by `$MyMsg` will be displayed as the message.

4.3 Assigning variable values

A variable should have a value assigned to it before its value is requested. To do this the = symbol is used. Here are some examples of assigning values to variables:

Assign to `$ABC`	**Code**
Some number	`$ABC = 123`
Some string	`$ABC = "I'm a string!"`
The return value of some function	`$ABC = SomeFunction()`

Since a variable's name can be given wherever a value is expected, it is possible to assign the value of one variable to another. After running the following code:

```
$Var1 = 'West Richmond'
$Var2 = $Var1
```

both `$Var1` and `$Var2` will contain the value `West Richmond`. If the value of one of these variables is subsequently changed, the other variable will still contain the value `West Richmond`.

[†] Technically the `$` is not part of the variable name itself; it is used to indicate to AutoIt that a variable name follows. This detail is only important in very few cases however, such as when using the special-purpose `Assign()` and `Eval()` functions (which aren't covered in this tutorial).

4.4 Constants

One clever use of variables is to save unnecessary repetition. Consider the following code:

```
MsgBox(0, 'Example Script from Section 4', 'This is the first message box.')
MsgBox(0, 'Example Script from Section 4', 'This is the second message box.')
MsgBox(0, 'Example Script from Section 4', 'This is the third message box.')
```

Here the string value `Example Script from Section 4` is used in multiple places. Imagine this string value being used hundreds of times and then imagine the work that would be required if the section number needed to be changed.

By preparing a *variable* to hold this data we can define it once (so that if it ever needs changing, it will only need changing in one location) and improve the overall size (and readability) of the script. Here's a functionally identical script embracing this idea:

```
$Title = 'Example Script from Section 4'

MsgBox(0, $Title, 'This is the first message box.')
MsgBox(0, $Title, 'This is the second message box.')
MsgBox(0, $Title, 'This is the third message box.')
```

Notice how the value of `$Title` is set once and not changed again. Variables of this nature are called *constants* as they are used for reading only (their values are constant). It is good programming practice to explicitly declare variables as constants when that is their intended use, because AutoIt can then protect the value of such a variable from being accidentally changed later. This is done by adding the word `Const` before the variable name when its value is initially declared, e.g.—

```
Const $Title = 'Example Script from Section 4'
```

4.5 Macros

Macros are like predefined constants in that they can be used wherever a value is expected. Refer to the help file for a complete list:

Contents → Macro Reference

4.6 Exercises

1. What will the variable `$ABC` contain after the following code runs? Why won't it contain a different value?

   ```
   $ABC = 123
   $abc = 456
   ```

2. Write a script that performs the following:

   - Prompt the user to press *Yes* or *No*, saving the return value of `MsgBox()` to a variable.
   - Display another appropriately titled `MsgBox()` that contains the value returned by the previous `MsgBox()` call.

3. Write a script that uses `InputBox()` to prompt the user for their name and then calls `MsgBox()` to display what was entered by the user.

4. Determine the values of `$A` and `$B` after this code is executed:

   ```
   $A = 10
   $B = 20
   $A = $A
   $B = $B
   $A = $B
   $B = $A
   ```

- Which lines of the above code have no effect? Why?

5. Write some code that swaps the values of variables `$A` and `$B`. For instance, if `$A` initially contains the value `Catch` and `$B` initially contains the value `22`, `$B` should afterwards contain the value `22` and `$B` should afterwards contain the value `Catch`.

   Hint: Adopt a temporary variable to hold the value of one of the variables before attempting to swap them.

Solutions to these exercises can be found on page 15.

5 Still to come

I have released this document in its current incomplete state because I'm a slow worker with little free time and of the impression that what is already written can be of use to others in the meantime. Here is a list of what I'd like to get done in the future:

Variables This is the most important thing that needs doing—anyone who uses them knows how limited life would be without them. I needed to discuss the concept of values first. Also discuss macros here.

Expressions Numeric calculations and string concatenation form expressions. These can be used anywhere that a value is expected because expressions are calculated down to a single value.

Branching Learn to create scripts that conditionally carry out tasks depending on some circumstance. Program scripts to 'think'. Briefly discuss checking the `@Error` macro for writing fail-safe scripts.

Looping Why copy and paste code 20 times (with minimal or no adjustments to each copy) when you could write the code once and add a couple extra lines that essentially say 'repeat x times or until y'? Code written using loops is far more manageable than e.g. a 700-line script that does a simple task 20 times. Requires knowledge of variables.

Commenting How to comment your code effectively, i.e. *why* the code is there (or what it achieves) rather than *what* it does literally.

Arrays The ability to store many different values under a 'single variable' and access them by number. Most handy in conjunction with looping.

Here's the good news though: there's nothing stopping you from researching the above items via the help file and the forums if you'd like to learn about them now.

A Exercise solutions

A.1 Solutions to Values exercises (section 2.4)

1. Values `123` and `0.5` can be represented by the *numeric* data type. All of the values can be represented by *strings* including the numbers.

2. (a) `MsgBox(0, "Tutorial", "This is a message.")`
   (b) `MsgBox(..., "Yoko's in Tokyo.")`
   (c) `MsgBox(..., "I said to him, 'Have some pudding.'")`
   (d) `MsgBox(..., "He'll have a ""royal"" time. Ahahahaha.")`
   (e) `MsgBox(..., """This"" message begins with a double quote!")`

3. (a) `MsgBox(0, 'Tutorial', 'This is a message.')`
   (b) `MsgBox(..., 'Yoko''s in Tokyo.')`
   (c) `MsgBox(..., 'I said to him, ''Have some pudding.''')`
   (d) `MsgBox(..., 'He''ll have a "royal" time. Ahahahaha.')`
   (e) `MsgBox(..., '"This" message begins with a double quote!')`

4. (a) `MsgBox(32 + 4, 'Title', 'Text')`
   (b) `MsgBox(16 + 1, ...)`
   (c) `MsgBox(48 + 5 + 256, ...)`

5. `310` is `256` + `48` + `6` which would give a message box with an *exclamation point* icon and *Cancel, Try Again* and *Continue* buttons. The *Try Again* button would be selected by default.

6. `MsgBox(0, '', 'This message box has an empty title.')`
   (Note that there is nothing at all between the quotes of the title string.)

7.
```
MsgBox(0, 'Hello', 'This is the first message box.')
MsgBox(0, 'Goodbye', 'This is the second message box.')
```

A.2 Solutions to Functions exercise (section 3.2)

(a)
```
Run('Calc')
WinWait('Calculator')
Send('861747-738291{ENTER}')
```
The answer should read `123 456`.

(b)
```
Run('Calc')
WinWait('Calculator')
Send('704277{+}295723{ENTER}')
```
The answer should read `1 000 000`.

(c)
```
Run('Calc')
WinWait('Calculator')
Send('2-1{+}4-3{+}6-5{+}9-8{ENTER}')
```
The answer should read `4`.

A.3 Solutions to Variables exercises (section 4.6)

1. `$ABC` will contain the value `456` after the code runs. It won't contain the value `123` because `$ABC` and `$abc` are equivalent, and as a result the value `123` is overwritten by the value `456`.

2.
   ```
   $MsgBoxRetVal = MsgBox(4, 'Exercise', 'Press Yes or No.')
   MsgBox(0, 'MsgBox() Return Value', $MsgBoxRetVal)
   ```

3.
   ```
   $Input = InputBox('Exercise', 'What is your name?')
   MsgBox(0, 'Input Entered', $Input)
   ```

4. `$A` and `$B` both have the value `20` after the code runs. The lines `$A = $A` and `$B = $B` have no overall effect because when they are run, AutoIt is instructed to set a variable to the current value of that same variable. This is a useless action.

5.
   ```
   $Temp = $A
   $A = $B
   $B = $Temp
   ```

Everything beyond this point should be ignored!

It’s basically my very first draft of things, here for your perusal if you’re brave. Items have been chopped out and placed into the above sections, so some text is likely less coherent than the rest. You’ve been warned!

Branching

<Brief spiel about embracing Boolean logic to go here> . . . This gives the benefit of the code appearing more English. Compare the two translations (where the word order is determined by the code):

- `If WinExists('Notepad') Then`
  ‘If a window exists by the title of “Notepad” then’
- `If WinExists('Notepad') = 1 Then`
  ‘If the function determining if a window exists by the title of “Notepad” returns 1 *then’*

```
; Useless comment -- don’t state -what- the code does
; Set $A to 5
  $A = 5

; Acceptable but obvious -- still states -what- the code does
; Prompt the user for their name
  $Name = InputBox(’Question’, ’What is your name?’)

; Very good comment -- states -why- the code is there; its -result-
; Discard the first line of data (does nothing if single-lined)
  $Data = StringTrimLeft($Data, StringInStr($Data, @LF))
```

Variables

(Had not talked about values at this point, which made things confusing. Rewrite.)
(Discuss `Opt('MustDeclareVars')` and why it’s useful.)

What are they?

Variables are areas in computer memory to hold information that you specify. This information can be in the form of numbers, textual characters, binary data or a combination of all three, e.g.—

How are they useful?

To save repetition What if you have some value that needs to be used many times? You *could* just copy and paste the value as many times as you need it in your script, *or* you could place the value into a variable and then refer to that variable as needed. Think about how much easier it will be if you ever have to change the value to something else.

To receive information What if you don’t have some information when you write your script? What if your script needs to determine that information? You can use a variable to keep that data once you determine it.

For calculations Description needed!

How to use them

In traditional programming languages, variables must be *declared* before first being used and you must specify what *type* of information will be held within that variable. Most scripting languages (including AutoIt) do not require that. There are advantages and disadvantages to both choices.

Before using a variable in AutoIt it's a good idea to declare it. This is a good habit to get into for later and it can help to prevent spelling errors.

Let's declare three variables, called `$A`, `$B` and `$C`:

```
Local $A
Local $B
Local $C
```

We can also do it this way:

```
Local $A, $B, $C
```

Notice in particular that every variable begins with a `$`. We now have three places to store or *assign* values. We can place direct values into them like this:

```
$A = 123
$B = 456
$C = "If it’s not a number, it goes in double quotes"
```

We can also *copy* the value of one variable to another, such that both variables carry *copies* of the same value:

```
$A = $B
```

`$A` will now contain the value that `$B` already had (and still has). Note however that this does not cause the two variables to become 'linked' in any way—if you change the value of one of the variables, the other variable won't be updated to carry that same new value. Their relationship begins and ends on that very line.

When the above line is executed, AutoIt checks what value is in `$B`, substitutes that into the line and then assigns `$A` when the right-hand side of the equation contains only concrete values. In other words, the right-hand side is computed until there's only one value left; that final value is *then* assigned to the variable on the left-hand side.

Functions (old draft)

What are they?

Knowledge of variables is necessary for almost any application of a programming or scripting language, but they are almost useless if they only originate from hard-coded values (as per the last lesson). This is where *functions* come in.

A *function* takes zero or more *inputs* and (usually) generates an *output*. We assign the output to a variable in the following manner:

```
$OutputVariable = FunctionName($InputVariable1, $Etc)
```

Some useful built-in AutoIt functions

The `MsgBox()` function is used to display a message box on the screen. You specify the *title* and *text* of the message box, along with the *icon* to be displayed and the *choice of buttons*. A number is returned which indicates the user's choice.

The InputBox() function displays an input box allowing the user to enter information that cannot be conveyed via a set of buttons.

You are strongly encouraged to look at the help file included with AutoIt, which contains a complete function reference. You will definitely see functions whose usefulness will appeal to you—remember them for the near future.

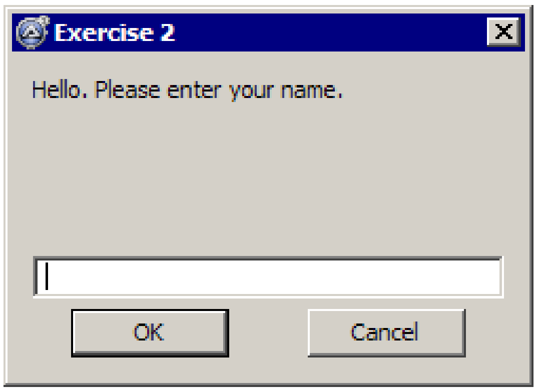
Example script

We'll use the two functions described above to do something useful. Try this script out and notice how it works—what does the 64 + 0 mean? (*Hint:* Refer to the AutoIt help file for the answer.)

```
; Any line beginning with a semicolon is ignored by AutoIt
; Always use this to your advantage (comment your code)

; This will hold the user's response to a MsgBox()
Local $Button

; This will hold the user's name
Local $Name

; Message boxes have an icon and an OK button
Local $MsgBoxOptions = 64 + 0

; Message boxes will have this title
Local $Title = "Script from Lesson 2"

$Button = MsgBox($MsgBoxOptions, $Title, "I will ask for your name.")

; The value of $Button does not matter
; because there was only one choice anyway

; Ask the user for their name
$Name = InputBox($Title, "What's your name?")

; Display what we were given
; The & means 'join the left and right together'
$Button = MsgBox($MsgBoxOptions, $Title, "Your name is " & $Name & "!")
```

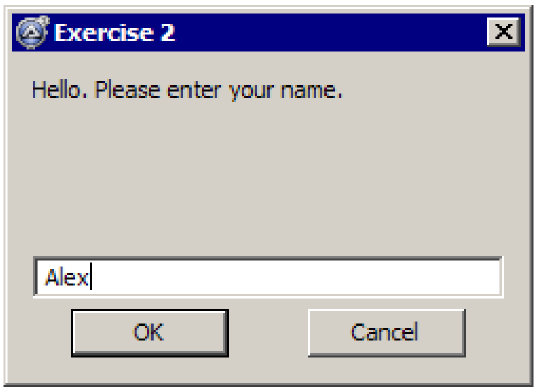
Exercise

Using both the script above and the AutoIt help file as a guide, write a script that displays the following dialogues in the following order (you will need three calls to InputBox() and one call to MsgBox()):


Exercise 2

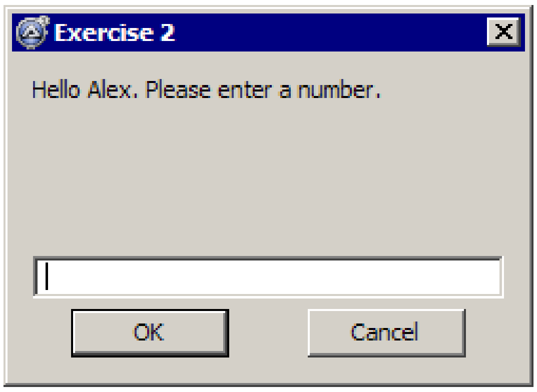
Hello. Please enter your name.

⇒


Exercise 2

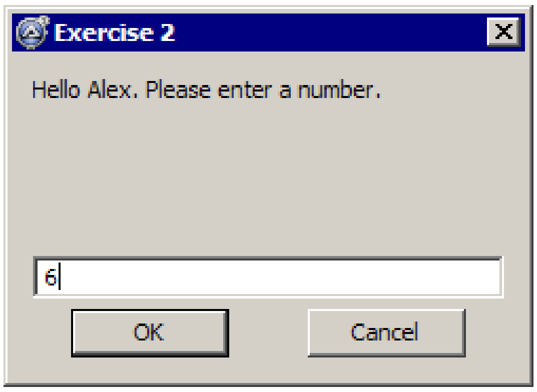
Hello. Please enter your name.

Alex


Exercise 2

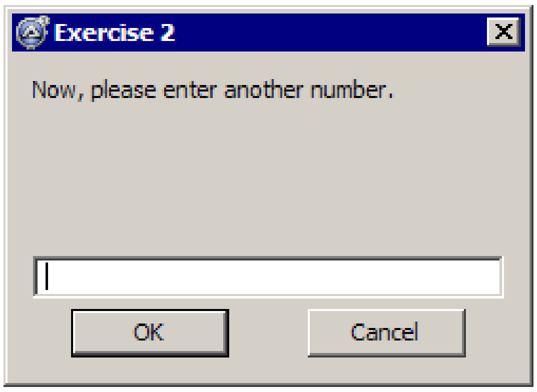
Hello Alex. Please enter a number.

⇒


Exercise 2

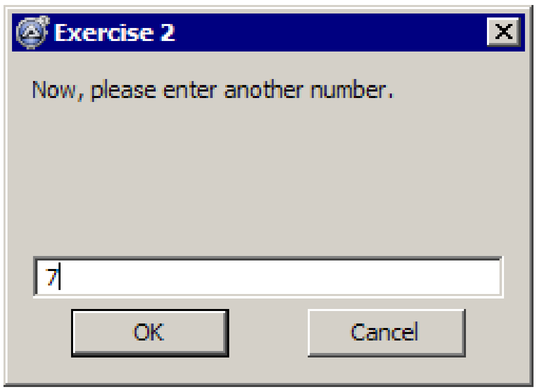
Hello Alex. Please enter a number.

6


Exercise 2

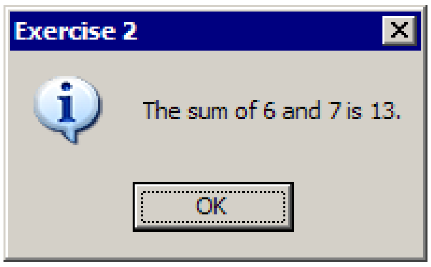
Now, please enter another number.

⇒


Exercise 2

Now, please enter another number.

7


Exercise 2

The sum of 6 and 7 is 13.

To get started you might want to adopt the following format, but you are of course free to write the script in any way that you wish:

```
; Variables to hold responses and calculations
Local $Name, $Num1, $Num2, $Sum

; Since we will use the title many times
; let’s put it into a variable --
; then we can change it easily if we ever want to
Local $Title = "Exercise 2"

; Get the information from the user
; (Fill this part in)

; Perform any necessary calculations
; (Fill this part in)

; Display the result/s to the user
; (Fill this part in)
```

Conditional execution

Some code generally can be written to perform some tasks in a definite order with no need to ever stray from that specific order. There are other times, however, when it would be quite convenient to do different sets of actions epending on certain conditions.

The conditions can vary greatly—you might wish to determine your plan of action depending on user input, or maybe a certain file doesn't exist, or a certain amount of time has elapsed.

Let's take a look at user input for an example. We (should) already know how to display a message box and we should know how to read the value returned by one, which will tell us which button was pressed. Something like this:

```
; Declare our variables of course
Local $Response

; Display the message box and capture the result
$Response = MsgBox(0x21, 'My Script', 'Click OK to perform some action.')
```

(The `0x21` *is* `0x20` + `0x01` *which means* 'question mark icon' *and* 'OK/Cancel buttons'*. Refer to the AutoIt help file for other possibilities.)*

Epilogue

If your time reading this document was well-spent in your opinion, please consider donating to the AutoIt team:

`www.autoitscript.com/donate.php`

or me if you feel inclined:

`http://yallara.cs.rmit.edu.au/~apeters/`