**BASICS OF AUTO IT:**

AutoIt provides a wide range of built-in functions to perform various tasks, including GUI creation, file manipulation, string handling, window management, and more. Here are some examples of commonly used built-in functions in AutoIt:

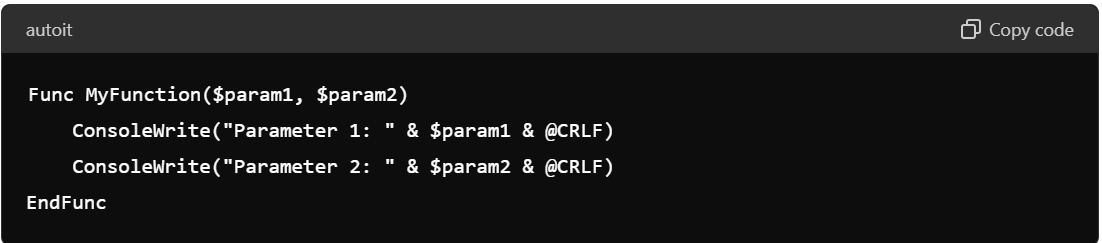
TO START WITH THE BASIC CONVENTIONS:

* Variables: **$myVariable**, **$count**, **$strName**
* Functions: **FuncName()**, **StringSplit()**, **WinGetTitle()**
* Macros: **@DesktopWidth**, **@OSVersion**, **@ScriptName**
* Keywords: **If**, **Else**, **For**, **While**

**FUNCTIONS IN AUTOIT:**

1. **File functions**:
   * **FileExists($filePath)**: Checks if a file exists.
   * **FileRead($filePath)**: Reads the contents of a file.
   * **FileWrite($filePath, $data)**: Writes data to a file.
2. **String functions**:
   * **StringLen($string)**: Returns the length of a string.
   * **StringInStr($string, $substring)**: Searches for a substring within a string.
   * **StringSplit($string, $delimiter)**: Splits a string into an array based on a delimiter.
3. **GUI functions**:
   * **GUICreate($title, $width, $height)**: Creates a new GUI window.
   * **GUICtrlCreateButton($text, $x, $y, $width, $height)**: Creates a button control.
   * **GUICtrlSetData($controlID, $data)**: Sets the data of a control, such as text for a label or input for an edit control.
4. **Window functions**:
   * **WinExists($title)**: Checks if a window exists.
   * **WinGetTitle($title)**: Retrieves the title of a window.
   * **WinSetState($title, $text, $state)**: Sets the state of a window (e.g., minimize, maximize, restore).
5. **Mouse and keyboard functions**:
   * **MouseClick($button [, $x, $y [, $clicks [, $speed]]])**: Simulates a mouse click at the specified coordinates.
   * **Send($string)**: Sends keystrokes to the active window.
6. **Process functions**:
   * **Run($program [, $workingDir [, $showFlag [, $options]]])**: Runs an external program.
   * **ProcessExists($processName)**: Checks if a process is running.
7. **Array functions**:
   * **UBound($array [, $dimension])**: Returns the number of elements in an array.
   * **ArraySearch($array, $value [, $start [, $end [, $step]]])**: Searches for a value in an array.
8. **Miscellaneous functions**:
   * **Sleep($milliseconds)**: Pauses script execution for a specified amount of time.
   * **ConsoleWrite($text)**: Writes output to the console.

In AutoIt, functions are typically written using the **Func** keyword followed by the function name, parameters (if any), and the function body enclosed in curly braces **{}**. Here's an example:



In AutoIt, functions are typically written using the **Func** keyword followed by the function name, parameters (if any), and the function body enclosed in curly braces **{}**. Here's an example:



In this example:

* **Func** is the keyword used to define a function.
* **MyFunction** is the name of the function.
* **$param1** and **$param2** are parameters passed to the function.
* The function body contains the code to be executed when the function is called.

There are a few conventions commonly followed when writing functions in AutoIt:

1. **Use descriptive function names**: Choose function names that accurately describe what the function does. This makes your code more readable and understandable.
2. **Use camelCase or underscores for function names**: While AutoIt doesn't enforce a specific naming convention for functions, it's common to use camelCase or underscores to separate words in function names. For example, **MyFunction**, **DoSomething**, **calculateTotal**, etc.
3. **Declare function parameters**: If your function takes parameters, declare them in the function definition within parentheses **()**. This makes it clear what input the function expects.
4. **Use Return statement if needed**: If your function produces a result, use the **Return** statement to return that result. This makes the function more versatile and reusable.

Here's an example that incorporates these conventions:



**MACROS & THER WRITING CONVENTIONS:**

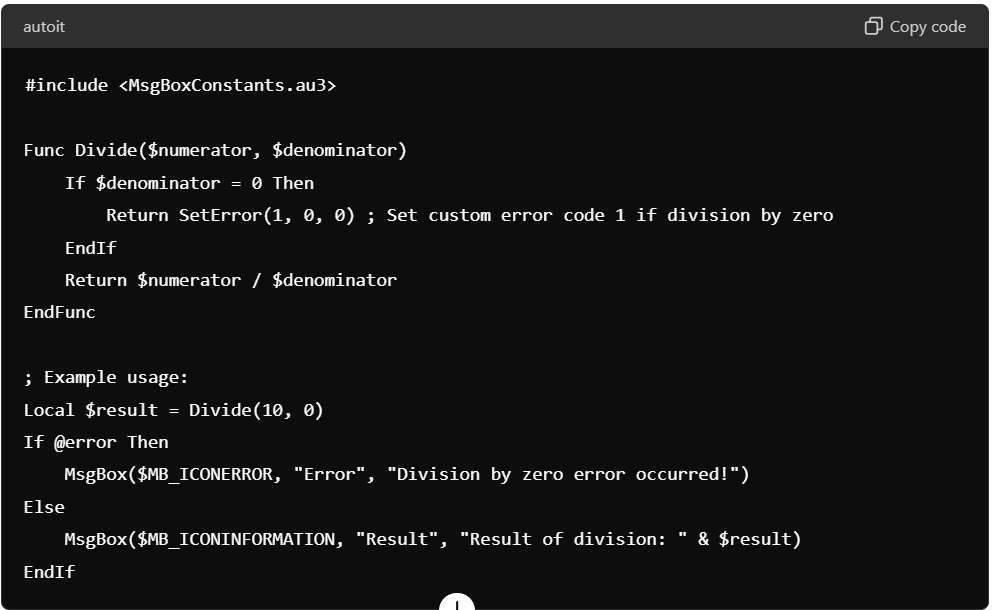
In AutoIt, macros are predefined constants that provide information about the system, script, or environment. They are typically written in uppercase letters with underscores **\_** separating words. Here are some common conventions for writing macros in AutoIt:

1. **Use UPPERCASE letters**: Macros are conventionally written in uppercase to distinguish them from variables, functions, and other identifiers in the script.
2. **Use underscores to separate words**: If a macro name consists of multiple words, use underscores to separate them. This improves readability and makes the macro name more descriptive.
3. **Use meaningful names**: Choose macro names that clearly indicate their purpose or the information they represent. This helps make your code more self-explanatory.
4. **Prefixes**: Some macros have prefixes to indicate their category or usage. For example:
   * **@** prefix for built-in AutoIt macros (e.g., **@DesktopWidth**, **@ScriptName**).
   * **SW\_** prefix for window state macros (e.g., **@SW\_MINIMIZE**, **@SW\_RESTORE**).
5. **Avoid using reserved words**: Avoid using macro names that conflict with AutoIt's reserved words or built-in functions to prevent unexpected behavior.

Here are some examples of well-formatted macros:

* **@DesktopWidth**
* **@DesktopHeight**
* **SW\_HIDE**
* **SW\_SHOWNORMAL**

**EXCEPTION HANDLING IN AUTOIT:**



In this example, we define a **Divide** function that takes two parameters: **$numerator** and **$denominator**. If the denominator is zero, division by zero would result in an error. Instead of letting the script crash, we use the **SetError()** function to set a custom error code (in this case, 1). We return this error code from the function.

When calling the **Divide** function, we check the value of **@error**. If it is non-zero, it indicates that an error occurred during the function execution. In this case, we display an error message using **MsgBox**. Otherwise, if **@error** is zero, it means the function executed successfully, and we display the result of the division.

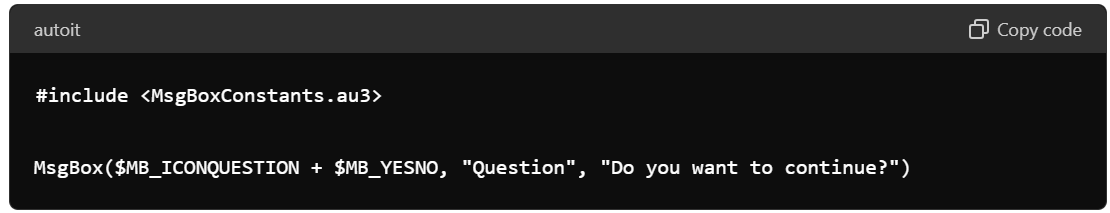
**THE MESSAGE BOX**

#inlcude<MsgBoxConstants.au3>

In AutoIt, the **MsgBox()** function is used to display a message box to the user. It can also include buttons, icons, and other attributes to customize its appearance and behavior. The **MsgBox()** function returns an integer indicating which button the user clicked.

The **$MB** constants are used to specify various attributes of the message box. Here are some commonly used **$MB** constants and their meanings:

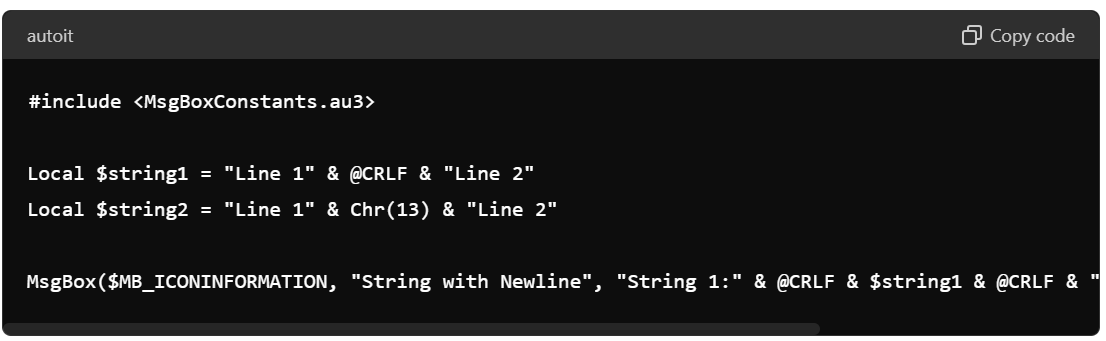
1. **Icon Constants**:
   * **$MB\_ICONERROR**: Displays a stop sign icon.
   * **$MB\_ICONINFORMATION**: Displays an "i" icon.
   * **$MB\_ICONQUESTION**: Displays a question mark icon.
   * **$MB\_ICONWARNING**: Displays an exclamation mark icon.
2. **Button Constants**:
   * **$MB\_OK**: Displays an OK button.
   * **$MB\_OKCANCEL**: Displays OK and Cancel buttons.
   * **$MB\_ABORTRETRYIGNORE**: Displays Abort, Retry, and Ignore buttons.
   * **$MB\_RETRYCANCEL**: Displays Retry and Cancel buttons.
   * **$MB\_YESNO**: Displays Yes and No buttons.
   * **$MB\_YESNOCANCEL**: Displays Yes, No, and Cancel buttons.
3. **Default Button Constants**:
   * **$MB\_DEFBUTTON1**: Sets the first button as the default button.
   * **$MB\_DEFBUTTON2**: Sets the second button as the default button.
   * **$MB\_DEFBUTTON3**: Sets the third button as the default button.
   * **$MB\_DEFBUTTON4**: Sets the fourth button as the default button.
4. **Other Constants**:
   * **$MB\_TOPMOST**: Makes the message box topmost.
   * **$MB\_SYSTEMMODAL**: Makes the message box system modal.
   * **$MB\_RIGHT**: Aligns text to the right.
   * **$MB\_RTLREADING**: Supports right-to-left reading order.



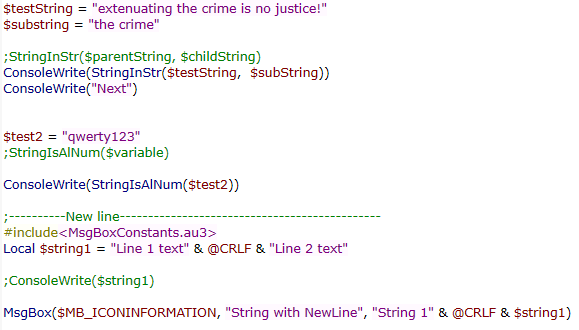
**NEWLINE:**

In AutoIt, you can represent a newline character using the **@CRLF** macro or the **Chr()** function with ASCII code 13 (**Chr(13)**). Both methods are commonly used to insert a newline or carriage return in strings.

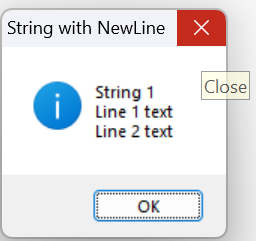
Here's an example of how to use both methods to insert a newline character in a string:



**AN EXAMPLE USING STRING FUNCTIONS AND NEWLINE:**



**OUTPUT:**



**THE WINDOW FUNCTIONS:**

Window functions in AutoIt are used to interact with windows on the desktop. These functions allow you to manipulate windows in various ways, such as finding windows, activating them, moving them, resizing them, and retrieving information about them. Here are some commonly used window functions in AutoIt:

1. **WinWaitActive($title [, $text [, $timeout]])**: Waits for a window to become active (focused) and ready for user input. You can specify the window title and optionally the window text. It returns **True** if the window becomes active within the specified timeout, otherwise **False**.
2. **WinActivate($title [, $text])**: Activates (brings to the foreground) a window with the specified title and optional text. If multiple windows match the criteria, the most recently active window is activated.
3. **WinClose($title [, $text])**: Closes a window with the specified title and optional text. It sends a close message to the window, asking it to close gracefully.
4. **WinExists($title [, $text])**: Checks if a window with the specified title and optional text exists. It returns **True** if the window exists, otherwise **False**.
5. **WinGetHandle($title [, $text])**: Retrieves the handle (HWND) of a window with the specified title and optional text. The handle can be used with other window functions that require a handle parameter.
6. **WinGetTitle($handle)**: Retrieves the title of a window given its handle (HWND).
7. **WinGetPos($title [, $text])**: Retrieves the position and size of a window with the specified title and optional text. It returns an array containing the x-coordinate, y-coordinate, width, and height of the window.
8. **WinMove($title, $text, $x, $y [, $width [, $height]])**: Moves and/or resizes a window to the specified position and size.
9. **WinSetState($title, $text, $state)**: Sets the state (minimized, maximized, or restored) of a window with the specified title and optional text.
10. **WinWait($title [, $text [, $timeout]])**: Waits for a window to exist. It's similar to **WinExists()** but waits for the window to be created if it doesn't already exist.

These functions can be very useful for automating tasks involving interacting with other applications' windows, such as GUI automation, testing, and scripting. They provide a way to manipulate windows programmatically, enabling you to control and automate various aspects of the user interface.