Functions

Overview

A function is a block of organized code that is used to perform a single task. Functions let you more efficiently run the same code multiple times without the need to copy and paste large blocks of code over and over again.

Functions make your payloads more modular and reusable, as each function may organize code that performs a single task.

FUNCTION

The `FUNCTION` command defines the name of a function, and includes the function body — the block of code that will execute when the function is called. Defining a function with the `FUNCTION` command in of itself does not execute the code within. To execute the code block within a function, it is called using the name of the function. All functions are named ending in open and close parenthesis ("`()`").

Syntax

- The `FUNCTION` command consists of these parts
- The `FUNCTION` keyword.
- The function name ending in open and close parenthesis ("`()`").
- One or more newlines containing the block of code to execute.
- One or more optional `RETURN` commands.
- The `END_FUNCTION` keyword.

Example

```
REM Example Function

FUNCTION COUNTDOWN()
    WHILE ($TIMER > 0)
        STRING .
```

```
        $TIMER = ($TIMER - 1)
        DELAY 500
    END_WHILE
END_FUNCTION

STRING And then it happened
VAR $TIMER = 3
COUNTDOWN()

SPACE
STRING a door opened to a world
$TIMER = 5
COUNTDOWN()
```

Passing Arguments

Within DuckyScript 3.0, the scope of a variable are global to the payload. This means that any function may access any defined variable. Unlike programming languages with strict scoping, functions within DuckyScript do not require variables to be passed as arguments within the open and close parenthesis ("`()`").

In the example above, the `$TIMER` variable may be considered an argument as it is referenced within the function, however keep in mind that this variable may be used by any other function within the payload.

Return Values

A function may include a `RETURN` value, like a variable containing an integer or boolean. This allows the function as a whole to be evaluated similar to an expression.

```
REM Example FUNCTION with RETURN

ATTACKMODE HID
DELAY 2000

BUTTON_DEF
    STRING !
```

```
END_BUTTON

FUNCTION TEST_BUTTON()
    STRING Press the button within the next 5 seconds.
    VAR $TIMER = 5
    WHILE ($TIMER > 0)
        STRING .
        DELAY 1000
        $TIMER = ($TIMER - 1)
    END_WHILE
    ENTER
    IF ($_BUTTON_PUSH_RECEIVED == TRUE) THEN
        RETURN TRUE
    ELSE IF ($_BUTTON_PUSH_RECEIVED == FALSE) THEN
        RETURN FALSE
    END_IF
    $_BUTTON_PUSH_RECEIVED = FALSE
END_FUNCTION

IF (TEST_BUTTON() == TRUE) THEN
    STRINGLN The button was pressed!
ELSE
    STRINGLN The button was not pressed!
END_IF
```

Result

- When the `IF` statement on line 27 checks the condition of the function `TEST_BUTTON`, the function is called and executed.
- Based on whether or not the button is pressed, the `RETURN` value (lines 20 and 22) will be set to `TRUE` or `FALSE`.
- The IF statement on line 27 evaluates the `RETURN` of the function `TEST_BUTTON` and types the result accordingly.

Avoiding Errors

- Function names should only contain letters, numbers and underscore ("`_`").
- Function names must end with open and close parenthesis ("`()`").
- They may begin with a letter or an underscore, but not a number.
- Spaces cannot be used in naming a function — however underscore makes for a suitable replacement. For example: `FUNCTION BLINK_LED()`.
- Be careful when using the uppercase letter `O` or lowercase letter `l` as they may be confused with the numbers `0` and `1`.
- Avoid using the names of commands or internal variables (e.g. `ATTACKMODE`, `STRING`, `WINDOWS`, `MAC`, `$_BUTTON_ENABLED`). See the full command and variable reference.