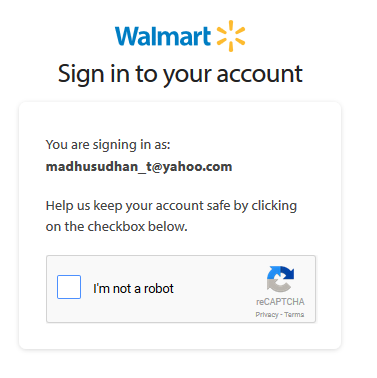
UFT One

# What are the advantages of automation?

* Time saving
* Quick
* Accuracy
* Confidence creation in management
* Reliable
* Save Cost
* Reusable
* Fast feedback to developers
* Minimize human testing errors

# Not automatable Tests

1. Login test case – I am not a robot – Captcha







1. Walmart Point of Sale (POS) – Scan the bar code and do billing

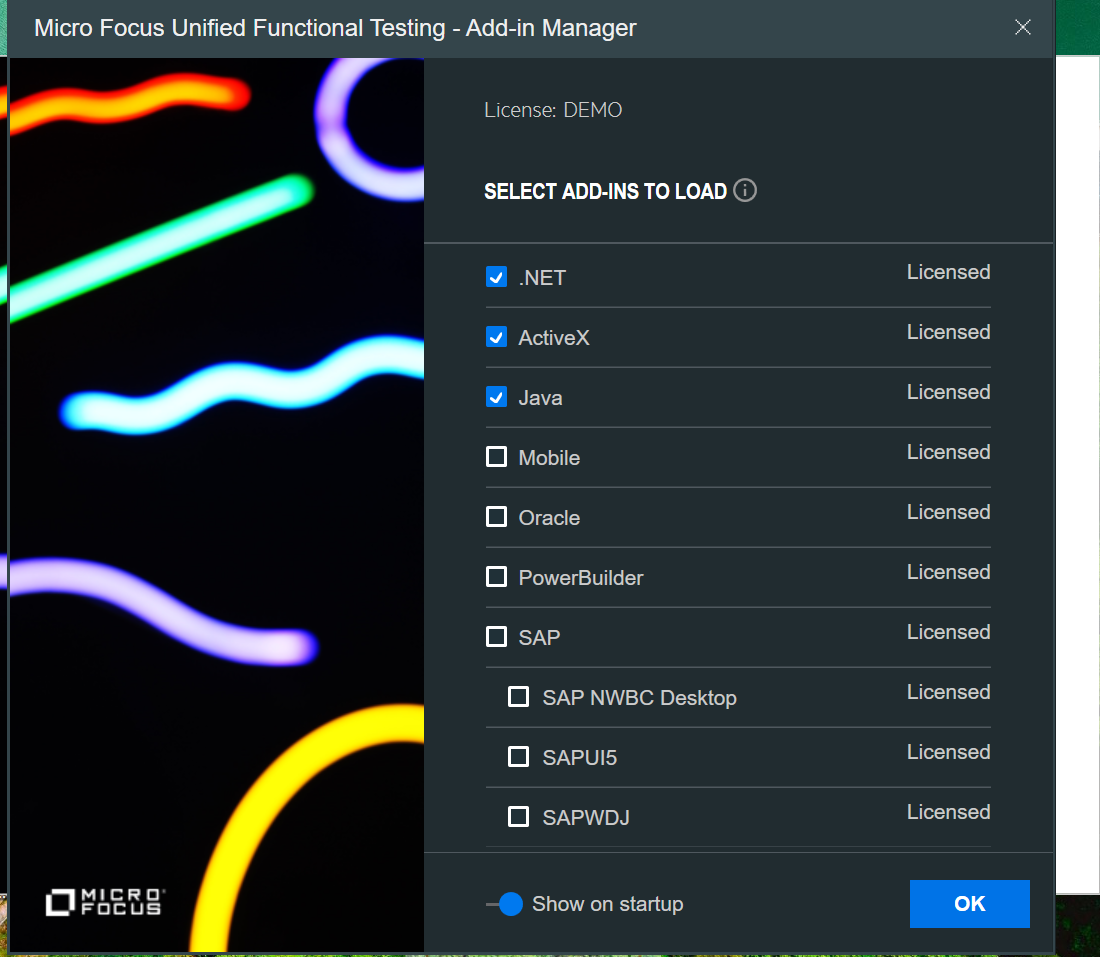


1. Interfaces testing – Print walmart bill
2. Withdraw $500 cash from ATM machine
3. If expected result is changing on real time situations (If no expected result, don’t automate)

Ex: Weather.com 🡪 Check the temperature display

# UFT Introduction

## Addin manager window



The Add-in manager window lists all the add-ins installed in UFT, here we need to select the required add-in for the application.

If you can't find the Add-in Manager:

If the Add-in Manager doesn't show by default, enable this from the UFT Options. From within UFT, select Tools > Options > General > Startup Options > Select Display Add-in Manager on startup.

# VB Scripting

* **VB**Script stands for **V**isual **B**asic Scripting which forms a subset of Visual Basic for Applications (VBA). VBA is a product of Microsoft.
* VBScript is a lightweight scripting language, which has a lightning-fast interpreter.
* VBScript, for the most part, is case-insensitive. It has a very simple syntax, easy to learn and implement.

Writing the first VBScript

var1 = 10   
 var2 = 20  
 Sum = var1 + var2    
print "The Sum of two numbers var1 and var2 is "&Sum

## Comments in VB Script

Comments are used to document the program logic and the user information. There are two ways to provide comments in VBScript.

* + - 1. Any statement that starts with a Single Quote (‘) is treated as a comment.
      2. Any statement that starts with the keyword “REM”.

## Variables in VBScript

A variable is a named memory location used to hold a value that can be changed during the script execution. VBScript has only ONE fundamental data type, Variant.

### Rules for naming the variables

* Variable Name must begin with an alphabet.
* Variable names cannot exceed 255 characters.
* Variables Should NOT contain a period (.)
* Variable Names should be unique in the declared context.

Variables are declared using “dim” keyword. Since there is only ONE fundamental data type, all the declared variables are variant by default. Hence, a user **NEED NOT** mention the type of data during declaration.

Example:

Dim num1

## VBScript Constants

Constant is a named memory location used to hold a value that CANNOT be changed during the script execution. If a user tries to change a Constant Value, the Script execution ends up with an error. Constants are declared the same way the variables are declared.

Const Constant\_Name = Value

**dim** radius  
radius = 10  
**Const** pi = 3.14  
area = pi \* radius \* radius  
print area

## Operators in VBScript

* Arithmetic Operators
* Comparison Operators
* Logical (or Relational) Operators
* Concatenation Operators

### Arithmetic Operators

If A=5 and B=10

|  |  |  |
| --- | --- | --- |
| Operator | Description | Example |
| + | Adds two operands | A + B will give 15 |
| - | Subtracts second operand from the first | A - B will give -5 |
| \* | Multiply both operands | A \* B will give 50 |
| / | Divide numerator by denumerator | B / A will give 2 |
| % | Modulus Operator and remainder of after an integer division | B MOD A will give 0 |
| ^ | Exponentiation Operator | B ^ A will give 100000 |

### Comparision Operators

|  |  |  |
| --- | --- | --- |
| Operator | Description | Example |
| = | Checks if the value of two operands are equal or not, if yes then condition becomes true. | (A == B) is False. |
| <> | Checks if the value of two operands are equal or not, if values are not equal then condition becomes true. | (A <> B) is True. |
| > | Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true. | (A > B) is False. |
| < | Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true. | (A < B) is True. |
| >= | Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true. | (A >= B) is False. |
| <= | Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true. | (A <= B) is True. |

### Logical Operators

|  |  |  |
| --- | --- | --- |
| Operator | Description | Example |
| AND | Called Logical AND operator. If both the conditions are True, then Expression becomes True. | a<>0 AND b<>0 is False. |
| OR | Called Logical OR Operator. If any of the two conditions is True, then condition becomes True. | a<>0 OR b<>0 is true. |
| NOT | Called Logical NOT Operator. It reverses the logical state of its operand. If a condition is True, then the Logical NOT operator will make it False. | NOT(a<>0 OR b<>0) is false. |

# Conditions and Loops

VBScript provides the following types of decision making statements.

|  |  |
| --- | --- |
| Statement | Description |
| if statement | An **if** statement consists of a Boolean expression followed by one or more statements. |
| if..else statement | An **if else** statement consists of a Boolean expression followed by one or more statements. If the condition is True, the statements under the **If** statements are executed. If the condition is false, then the **Else** part of the script is Executed |
| if...elseif..else statement | An **if** statement followed by one or more **ElseIf**Statements, that consists of Boolean expressions and then followed by an optional else statement, which executes when all the condition becomes false. |
| nested if statements | An **if** or **elseif** statement inside another **if** or **elseif** statement(s). |
| switch statement | A **switch** statement allows a variable to be tested for equality against a list of values. |

### If Else

If(boolean\_expression) Then

Statement 1

.....

.....

Statement n

End If

Example:

**If** a>b **Then**  
    print "A is greater"  
**else**  
    print "B is greater"  
**End** **If**

### Switch Case

Select Case expression

Case expressionlist1

statement1

statement2

....

....

statement1n

Case expressionlist2

statement1

statement2

End Select

Example:

myvar = 1  
**Select** **Case** myvar  
    **Case** 1  
        print "Today is monday"  
    **Case** 2  
        print "Today is Tuesday"  
**End** **Select**

Practice:

Write a code to swap two numbers only if A is greater then B

# Loops

VBScript provides the following types of loops to handle looping requirements.

|  |  |
| --- | --- |
| Loop Type | Description |
| for loop | Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable. |
| for ..each loop | It is executed if there is at least one element in group and reiterated for each element in a group. |
| while..wend loop | It tests the condition before executing the loop body. |
| do..while loops | The do..While statements will be executed as long as condition is True.(i.e.,) The Loop should be repeated till the condition is False. |
| do..until loops | The do..Until statements will be executed as long as condition is False.(i.e.,) The Loop should be repeated till the condition is True. |

For counter = start To end [Step stepcount]

[statement 1]

[statement 2]

....

[statement n]

[Exit For]

[statement 11]

[statement 22]

....

[statement n]

Next

------------------------------

**For** i = 1 **To** 10 **Step** 1  
    print "Hello"  
**Next**

**-------------------------------------**

**Do** **While** i < 5  
            i = i + 1  
            print "The value of i is : "&i  
  **loop**

**-------------------------------------**

**Do** **Until** i>15  *'Condition is False.Hence loop will be executed*  
            i = i + 1  
    print "The value of i is : "&i  
 **loop**

**---------------------------------------**

**do**  
     *' Test value of Counter.*  
            Counter = Counter + 1   *' Increment Counter.*  
           print "The Current Value of the Counter is : " &Counter  
  **loop** **While** Counter < 15

Practise Code:

Write a program to display the sum of 5 terms of even natural numbers.  
Test Data :  
Input number of terms : 5  
*Expected Output* :  
The even numbers are :2 4 6 8 10  
The Sum of even Natural Number upto 5 terms : 30

# Functions and Procedures

VBScript procedures are used mainly for code organization and reuse. If you do not organize the code properly, the whole process of coding, debugging and managing the code will become really complex. So, you should organize or modularize the code carefully so that your code becomes easily manageable.

## Types of procedures

A procedure is a block of code that ideally performs a single function. A block of code that processes an input or handles a file is a good example of a function. There are two types of procedures in VBScript.

1. Sub procedure: The sub-procedure does not return a value.
2. Function procedure: The function procedure is used if you want to return a value.

### Sub Procedures

If you want to execute a series of statements without returning any value, then you can use sub procedures.

Sub procedures start and end with Sub and End Sub statements respectively. Sub procedures can take arguments, but cannot return a value. Sub procedures may or may not take input.

**Sub** outputMessage()  
 print "Welcome"  
**End** **Sub**

### Functions

If you want to execute a series of statements and return a value, then you need to use function procedures, commonly known as function.

Function procedures start and end with Function and End Function statements respectively.

A function procedure may or may not take input.

Function procedures return a value by assigning the value to its name.

**Function** findArea(radius)  
   **const** pi=3.14  
   area = pi\*radius\*radius  
   findArea = area  
**End** **Function**

## ByRef and ByVal Parameters

You can pass VBScript arguments to the procedures by reference or by value.

If you do not specify anything when you call a procedure, then the argument/arguments are passed by reference by default.

If you the changes made to the arguments to persist even after the procedure is called, then you need to pass the VBScript arguments by reference.

When an argument is passed by value, any changes that the called procedure makes to the value of the variable do not persist after the procedure is called.

The keywords ByRef and ByVal are used to pass arguments by reference and by value respectively.

**Dim** x  
 x = 2  
**call** findArea(x)  
**msgbox** x  
   
**Function** findArea(**Byref** value)  
   value = value + 1  
   findArea = value  
**End** **Function**

# InBuilt Functions in VBScript

Commonly used inbuilt functions in VBScript

|  |  |
| --- | --- |
| Function | Description |
| Date | Returns the current system date |
| DateDiff | Returns the number of intervals between two dates |
| Day | Returns a number that represents the day of the month |
| Month | Returns a number that represents the month of the year |
| Year | Returns a number that represents the year |
| Cint | Converts an expression to a variant of subtype Integer |
| CStr | Converts an expression to a variant of subtype String |
| Array | Returns a variant containing an array |
| IsArray | Returns a Boolean value that indicates whether a specified variable is an array |
| LBound | Returns the smallest subscript for the indicated dimension of an array |
| UBound | Returns the largest subscript for the indicated dimension of an array |
| InStr | Returns the position of the first occurrence of one string within another |
| InstrRev | Returns the position of the first occurrence of one string within another. The search begins at the last character of the string |
| LCase/UCase | Coverts to lowercase/uppercase |
| Left/Right | Returns a specified number of characters from the left/Right side of a string |
| Len | Returns the number of characters in a string |
| LTrim/RTrin/Trim | Removes spaces on the string |
| Replace | Replaces a specified part of a string with another string a specified number of times |
| strComp | Compares two strings and returns a value that represents the result of the comparison |
| strReverse | Reverses a string |

# Arrays in VBScript

Sometimes, we have to hold more than one value in a single variable at a time. When a series of values is stored in a single variable, then it is known as an **array variable**.

Arrays are declared the same way a variable has been declared except that the declaration of an array variable uses parenthesis.

**Array Declaration:**

**Dim** arr1()  
**Dim** arr2(5)  
**Dim** arr3  
arr3 = **Array**("Bike","Cars","Bus")

The values are assigned to the array by specifying array index value against each one of the values to be assigned.

aar(0) = "user1"  
aar(1)="user2"  
aar(2)="user3"

## Multi Dimentional Arrays

Arrays are not just limited to single dimension and can have a maximum of 60 dimensions. Two-dimension arrays are the most commonly used ones.

**Dim** studentsDetails(3,2)  
studentsDetails(0,0) = "S1"  
studentsDetails(0,1) = "John"  
studentsDetails(1,0) = "S2"  
studentsDetails(1,1) = "Eva"  
studentsDetails(2,0) = "S3"  
studentsDetails(2,1) = "Robert"

## Redim Statement

ReDim Statement is used to declare dynamic-array variables and allocate or reallocate storage space.

**Dim** a()  
a(0) = "a"  
**redim** a(5)  
a(0) = "XYZ"  
a(1) = "ABC"  
a(2) = "BBB"  
  
**redim** **preserve** a(7)  
a(3) = "ooo"  
a(4)="qqq"  
a(5)="PPP"  
a(6)="ZZZ"

## Dynamic Arrays

Size of a dynamic array can be modified at runtime.

Dim arrayvalue(3)

Msgbox lbound(arrayvalue)

Msgbox ubound(arrayvalue)

**Redim**Statement is used to redefine the size of an Array. When the array is declared without any size, then it can be declared again using Redim with the feasibility of specifying the size of an array.

**The preserve** keyword is used to preserve the contents of a current array when the size of an array gets changed.

**Dim** a(1)  
**ReDim** a(2)  
a(2)="aa"  
**msgbox** a(2)

The above code throws error as we are trying to resize the static array.

Write a program to copy an array to another by iterating the array.

# FSO(File System Object)

FSO Objects help the developers to work with drives, folders and files.

## Getting file details

**Dim** fso, f  
**Set** fso = **CreateObject**("Scripting.FileSystemObject")  
**Set** f = fso.GetFile("C:\Users\MEGHANA\Desktop\download.png")  
print f.DateCreated     
print f.DateLastAccessed  
print f.DateLastModified  
print f.Drive  
print f.Name

## Reading and Writing from a file

**Const** ForReading = 1, ForWriting = 2  
**Dim** fso  
filePath = "D:\Training\Cookies\Sample2.txt"  
**Set** fso = **CreateObject**("Scripting.FileSystemObject")  
**Set** f = fso.OpenTextFile(filePath, ForWriting, **True**)   
f.**Write** "Hello world!"   
  
**Set** f = fso.OpenTextFile(filePath, ForReading)  
**msgBox** f.ReadAll  
f.Close  
fso.DeleteFile(filePath)

# Excel Object Model

Excel Objects are mainly referred to as objects that provide support to the Coders to work and deal with the Excel Sheets.

## Creating an Excel sheet

Set obj = createobject("Excel.Application")  *'Creating an Excel Object*  
obj.visible=True                 *'Making an Excel Object visible*  
obj.Workbooks.Add()       *'Adding a Workbook to Excel Sheet*  
obj.Worksheets(1).Cells(1,1) = "Hello!!!"  
obj.ActiveWorkbook.SaveAs "D:\Training\Cookies\newexcelfile.xlsx"   *'Saving a Workbook*  
obj.Quit       *'Exit from Excel Application*  
Set obj=Nothing

## Writing in Existing File

**Set** obj = **createobject**("Excel.Application")  *'Creating an Excel Object*  
obj.visible=**True**       
obj.Workbooks.Open("D:\Training\Cookies\newexcelfile.xlsx")  
obj.Worksheets(1).Cells(1,2) = "Hello UFT"  
obj.ActiveWorkbook.Save  
obj.Quit       *'Exit from Excel Application*  
**Set** obj=**Nothing**

## Reading Data from Excel

**Dim** objExcel, objWorkbook, objWorksheet, RowsCount  
**Set** objExcel = **CreateObject**("Excel.Application")  
**Set** objWorkbook = objExcel.Workbooks.Open("D:\Training\Cookies\newexcelfile1.xlsx")  
**Set** objWorksheet = objWorkbook.Worksheets(1)  
  
Count = objWorksheet.usedRange.Columns.Count  
**msgbox** Count  
**For** i = 1 **To** Count **Step** 1  
    **msgbox** objWorksheet.Cells(1, i)  
**Next**  
objExcel.Quit  
**Set** objWorksheet = **Nothing**  
**Set** objWorkbook = **Nothing**  
**Set** objExcel = **Nothing**

# VBScript Regular Expressions

Regular Expressions is a sequence of characters that forms a pattern, which is mainly used for search and replace. The purpose of creating a pattern is to match specific strings, so that the developer can extract characters based on conditions and replace certain characters.

## Regex Object

RegExp object helps the user to match the pattern of strings and the properties and methods help us to work with Regular Expressions easily.

* **Pattern** − The Pattern method represents a string that is used to define the regular expression and it should be set before using the regular expression object.
* **IgnoreCase** − A Boolean property that represents if the regular expression should be tested against all possible matches in a string if true or false. If not specified explicitly, IgnoreCase value is set to False.
* **Global** − A Boolean property that represents if the regular expression should be tested against all possible matches in a string. If not specified explicitly, Global value is set to False.

## Methods

* **Test**(search-string) − The Test method takes a string as its argument and returns True if the regular expression can successfully be matched against the string, otherwise False is returned.
* **Replace**(search-string, replace-string) − The Replace method takes 2 parameters. If the search is successful then it replaces that match with the replace-string, and the new string is returned. If there are no matches then the original search-string is returned.
* **Execute**(search-string) − The Execute method works like Replace, except that it returns a Matches collection object, containing a Match object for each successful match. It doesn't modify the original string.

|  |  |
| --- | --- |
| Pattern | Description |
| ^ | Matches only the beginning of a string. |
| $ | Match only the end of a string. |
| [xyz] | Match any of the character class enclosed within the character set. |
| [^xyz] | Matches any of the character class that are NOT enclosed within the character set. |
| \w | Match any word character class. Equivalent to [a-zA-Z\_0-9] |
| \W | Match any non-word character class. Equivalent to [^a-zA-Z\_0-9] |
| \d | Match any digit class. Equivalent to [0-9]. |
| \D | Match any non-digit character class. Equivalent to [^0-9]. |

Example:

strid = "99-00110"  
**Set** re = **New** RegExp  
**With** re  
.Pattern    = "\d{2}-\d{5}"  
.IgnoreCase = **False**  
.**Global**     = **False**  
**End** **With**  
  
*' Test method returns TRUE if a match is found*  
**If** re.Test( strid ) **Then**  
print strid&" is a valid ID"  
**Else**  
print strid&" is NOT a valid ID"  
**End** **If**  
  
**Set** re = **Nothing**

Example:

strid = "KA12 PP-1234"

Pattern: "\w{2}\d{2}\s\w{2}-\d{4}"

# Error Handling

Error Handling is a very useful mechanism of programming languages like VBScript in order to deal with errors and to continue the execution of the program even after the occurrence of errors inside a program.

**Error Prevention** is an aspect of Error handling which means taking effective measures inside a script in order to avoid the occurrence of errors.

Types of Error:

Syntax Errors

Runtime Errors

Logical Errors

## Error Handling in VBScript

### On Error Resume Next

**Moves the control of the cursor to the next line of the error statement.**

**Example:**

In this case, the division is by 0 and if you do not want your script to get stuck due to this error then you put ***‘On Error Resume Next’*** at the top of your script as shown below.

**Dim**  num2, div, num1  
div = 0  
num1 = **InputBox**("Enter")  
num2 = 10  
**On** **error** **resume** **next**  
div = num2/num1  
**If** div=0 **Then**  
    **msgbox** "Please enter something other than zero"  
**else**  
    **msgbox** "Division: "&div  
**End** **If**

### Err Object

This method is basically used to capture the **details of the Error**. If you want to know more about the Error like Number, description, etc., then you can do so by accessing the properties of this Object.

**Following is the list of properties of Err Object with their details:**

**Number**: This will tell you the error number i.e. the integer value of the type of error that occurred.

**Description**: This will tell you about the error i.e. the description of the error.

**Raise**: This will let you raise the specific error by mentioning its number.

**Clear**: This will clear the error i.e. will set to error handler to nothing.

**msgbox** "Number of the Error and Description is "&**Err**.**Number**&" "&**Err**.**Description**  
    **Err**.**Clear**

### Err.Raise

The Err. Raise method allows you to generate a runtime error. where *ErrorNumber* is the numeric code for the error you’d like to generate.

#### Some common error types

|  |  |
| --- | --- |
| Error Number | Description |
| 5 | Invalid procedure call |
| 6 | Overflow |
| 7 | Out of memory |
| 9 | Subscript out of range |
| 11 | Division by zero |
| 13 | Type mismatch |