Instructor Notes:

Add instructor notes here.

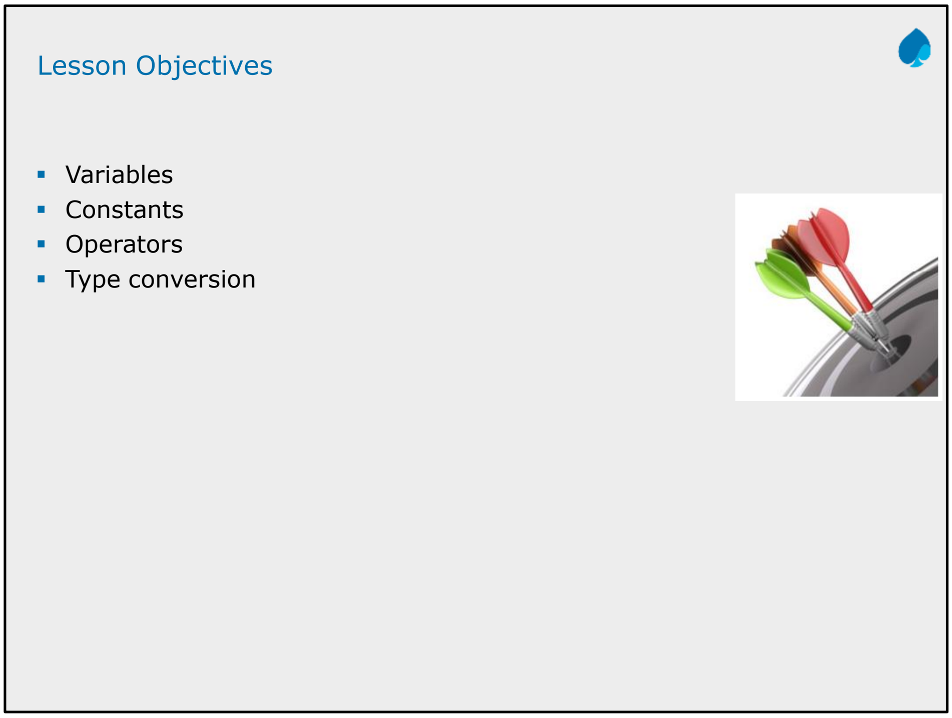
# VBScript

Lesson 2: Variables ,Constants and Operators in VBScript

Capgemini

Instructor Notes:

Explain the lesson coverage

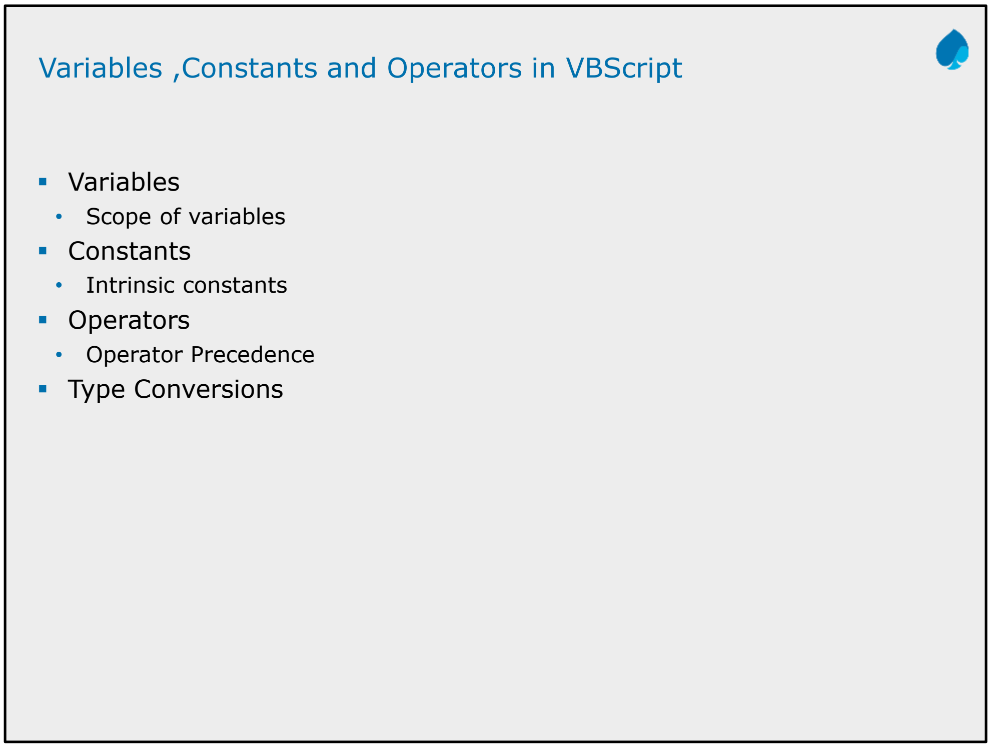


## Lesson Objectives

- Variables
- Constants
- Operators
- Type conversion

Instructor Notes:

Additional notes for instructor

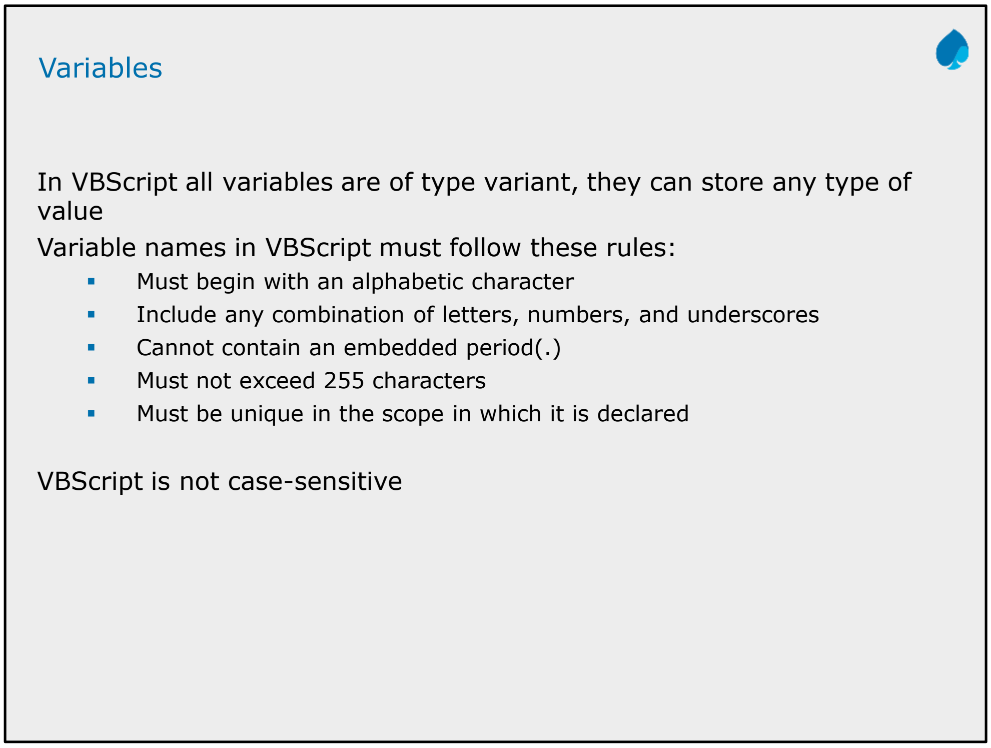


## Variables ,Constants and Operators in VBScript

- Variables
  - Scope of variables
- Constants
  - Intrinsic constants
- Operators
  - Operator Precedence
- Type Conversions

Add the notes here.

Instructor Notes:



## Variables

In VBScript all variables are of type variant, they can store any type of value

Variable names in VBScript must follow these rules:

- Must begin with an alphabetic character
- Include any combination of letters, numbers, and underscores
- Cannot contain an embedded period(.)
- Must not exceed 255 characters
- Must be unique in the scope in which it is declared

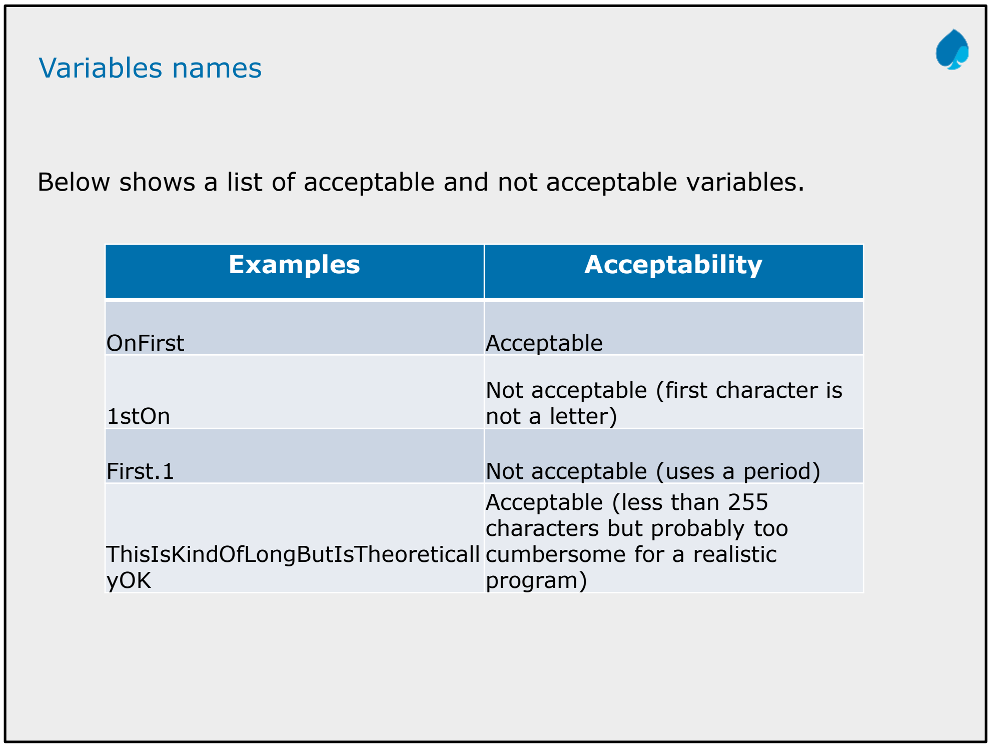
VBScript is not case-sensitive

**Variables**

Variables in VBScript hold information (values). Whenever you use a variable, VBScript sets up an area in the computer's memory to store the information.

To VBScript, MyName and myname are the same variable name.

Instructor Notes:



## Variables names

Below shows a list of acceptable and not acceptable variables.

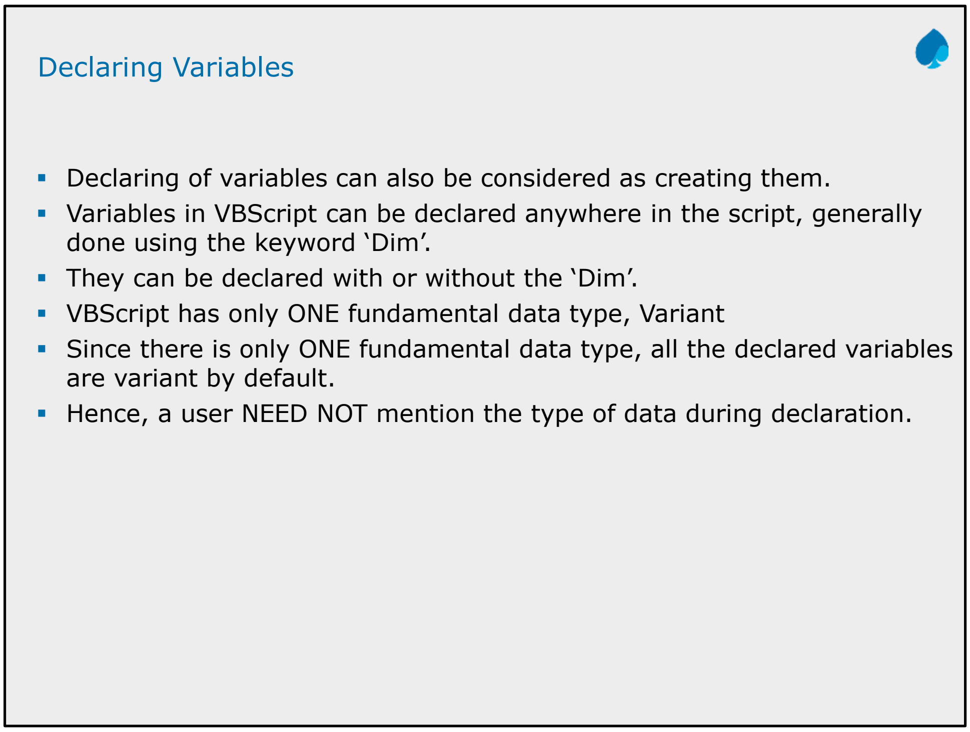
| Examples | Acceptability |
|---|---|
| OnFirst | Acceptable |
| 1stOn | Not acceptable (first character is not a letter) |
| First.1 | Not acceptable (uses a period) |
| ThisIsKindOfLongButIsTheoreticallyOK | Acceptable (less than 255 characters but probably too cumbersome for a realistic program) |

All characters in a variable name are significant. Base is a different variable from Base1, and both are different from Base_1.

You can't use names reserved by VBScript for variable names; for example, Sub is not acceptable as a variable name.

However, you can embed reserved words within a variable's name. For example, Substitute is a perfectly acceptable variable name.

Instructor Notes:



## Declaring Variables

- Declaring of variables can also be considered as creating them.
- Variables in VBScript can be declared anywhere in the script, generally done using the keyword ‘Dim’.
- They can be declared with or without the ‘Dim’.
- VBScript has only ONE fundamental data type, Variant
- 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.

You can also declare variables by using its name in a script. Like this:

**empname=“Sandra"**

Now you have also created a variable. The name of the variable is “empname".
However, this method is not a good practice, because you can misspell the variable name later in your script, and that can cause strange results when your script is running.
If you misspell for example the “empname" variable to “empnime", the script will automatically create a new variable called “empnime”.
To prevent your script from doing this, you can use the Option Explicit statement. This statement forces you to declare all your variables.

Instructor Notes:

## Option Explicit

- To ensure that, your VBScript programs declare its variables before you can use them, use option explicit directive

```
<SCRIPT LANGUAGE="VBSCRIPT">
        Option Explicit
 <SCRIPT>
```

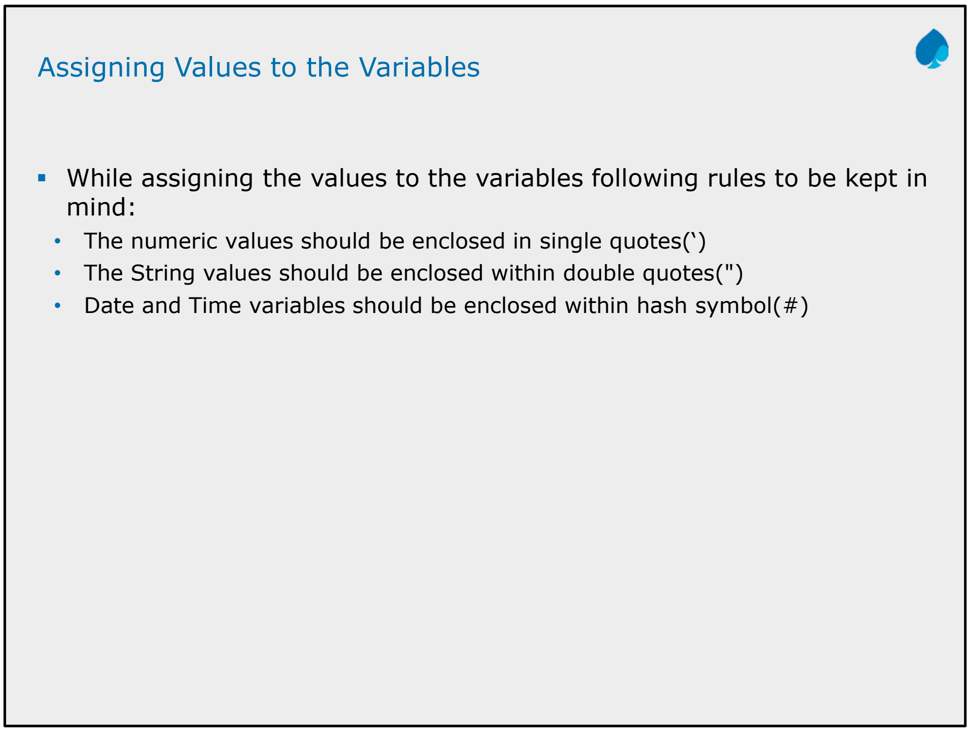
- For Example:
- Since Option Explicit is mentioned the variable needs to be declared

```
Option Explicit
Dim empname
empname=“Sandra”
```

**Explicit Declaration**

After VBScript processes an Option Explicit statement, it will no longer allow you to use a variable unless you declare it first. If you try to use a variable without declaring it, an error message will appear. Finally, the first time you use a variable, VBScript temporarily assigns it the default value "empty." This basically means that the variable doesn't contain a value. The "empty" value disappears the moment you assign a value to the variable.
You shouldn't hesitate to assign initial values to your variables. Otherwise, you risk creating a breeding ground for hard-to-find bugs. It is therefore quite common to use the first few statements in an event procedure to initialize the variables.

Instructor Notes:



## Assigning Values to the Variables

- While assigning the values to the variables following rules to be kept in mind:
  - The numeric values should be enclosed in single quotes(‘)
  - The String values should be enclosed within double quotes(")
  - Date and Time variables should be enclosed within hash symbol(#)

**Examples :**

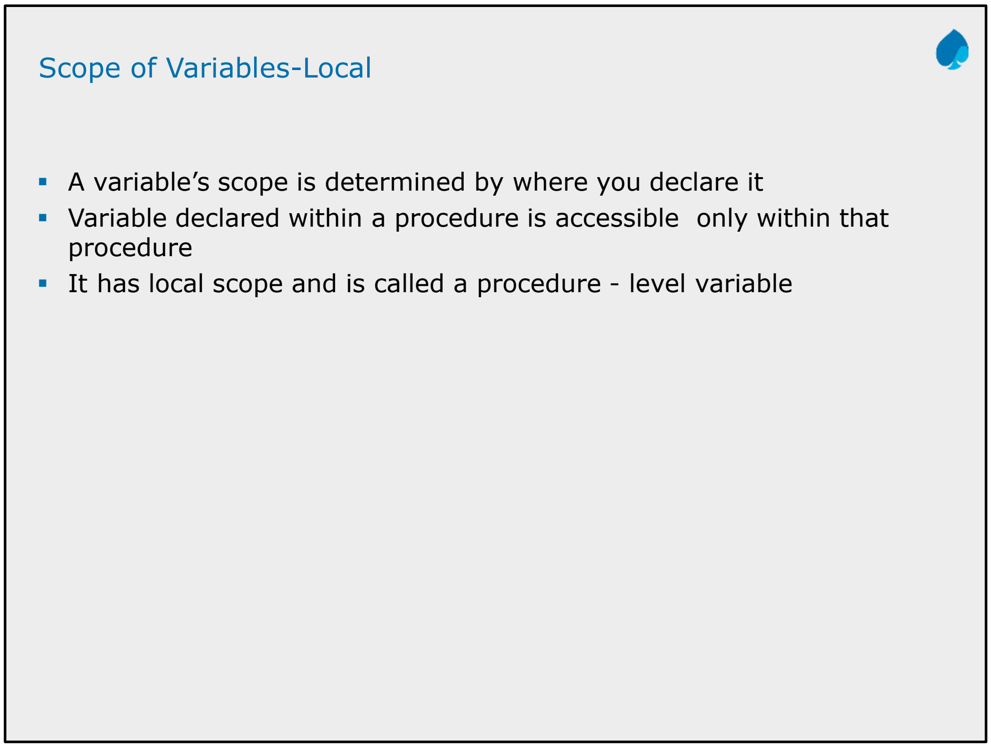
' Below Example, The value 20 is assigned to the variable.
  Number1 = 20 '

A String Value ‘Come’ is assigned to the variable StrValue.
StrValue = “Come”

' The date 01/01/2020 is assigned to the variable DToday.
   DToday = #01/03/2010#

' A Specific Time Stamp is assigned to a variable Time1
  Time1 = #12:20:24 PM#

Instructor Notes:



## Scope of Variables-Local

- A variable’s scope is determined by where you declare it
- Variable declared within a procedure is accessible  only within that procedure
- It has local scope and is called a procedure - level variable

**Sharing values within the procedure**

When programmers discuss the availability of a variable used in one part of the program to the other parts of the program, they refer to the scope of variables.  An event procedure will not normally have access to the value of a variable changed in another event procedure. As always, it is not a good programming practice to rely on defaults. If you want to be sure that a variable is local within an event procedure, use the Dim statement inside the event procedure to declare all its variables.

Instructor Notes:

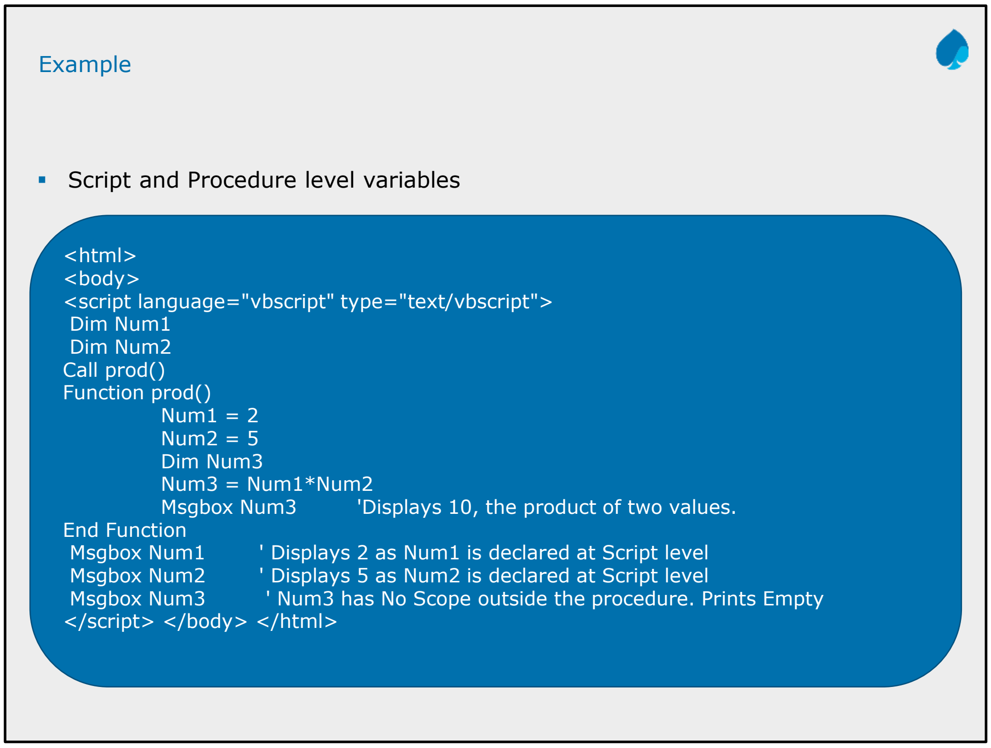
## Scope of Variables –Global

- Variable declared outside a procedure, will be recognizable to all the procedures in your script.
- This is a script - level variable, and it has script-level scope.

**Sharing values across procedures**

Occasionally you will want the value of a variable to be available to all the VBScript code for your Web page. For example, if an application is designed to perform a calculation involving a single interest rate at a time, that rate should be available to all the procedures for the page. Variables that are shared in this way are called script-level variables. Just as with the Option Explicit statement, you put the declaration statements for script-level variables outside any event procedures.

Instructor Notes:



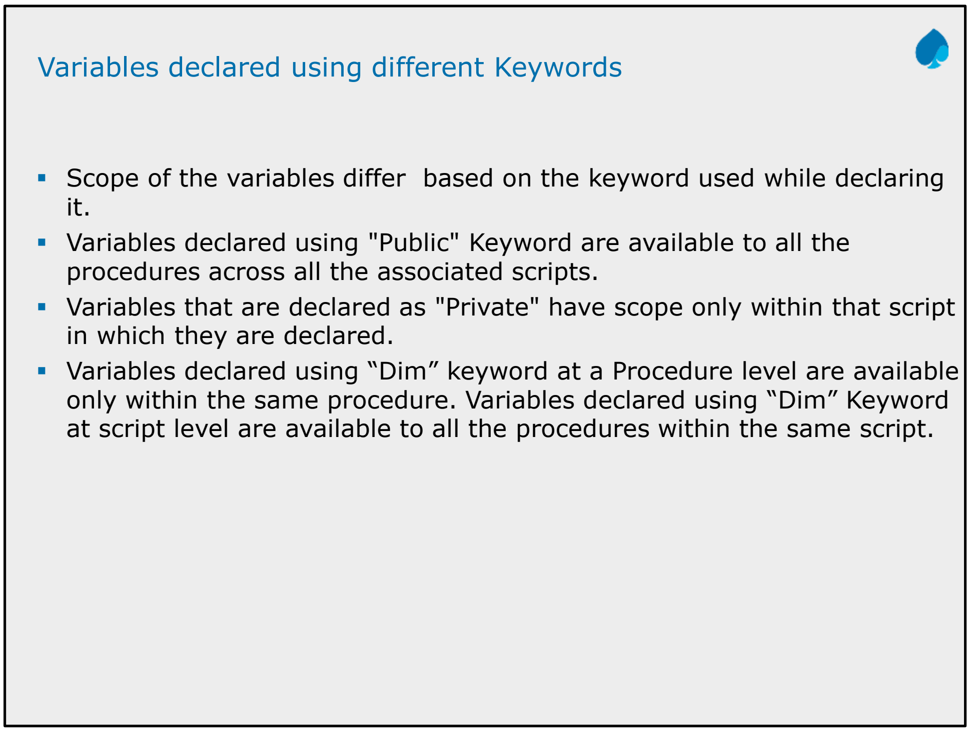
## Example

- Script and Procedure level variables

```
<html>
<body>
<script language="vbscript" type="text/vbscript">
 Dim Num1
 Dim Num2
Call prod()
Function prod()
               Num1 = 2
               Num2 = 5
               Dim Num3
               Num3 = Num1*Num2
               Msgbox Num3         'Displays 10, the product of two values.
End Function
 Msgbox Num1        ' Displays 2 as Num1 is declared at Script level
 Msgbox Num2        ' Displays 5 as Num2 is declared at Script level
 Msgbox Num3         ' Num3 has No Scope outside the procedure. Prints Empty
</script> </body> </html>
```

In the above example:
the values of Num1 and Num2 are declared at script level while Num3 is declared at procedure level.

Instructor Notes:



## Variables declared using different Keywords

- Scope of the variables differ  based on the keyword used while declaring it.
- Variables declared using "Public" Keyword are available to all the procedures across all the associated scripts.
- Variables that are declared as "Private" have scope only within that script in which they are declared.
- Variables declared using “Dim” keyword at a Procedure level are available only within the same procedure. Variables declared using “Dim” Keyword at script level are available to all the procedures within the same script.

Variables declared using "Public" Keyword are available to all the procedures across all the associated scripts. When declaring a variable of type "public", Dim keyword is replaced by "Public

Instructor Notes:

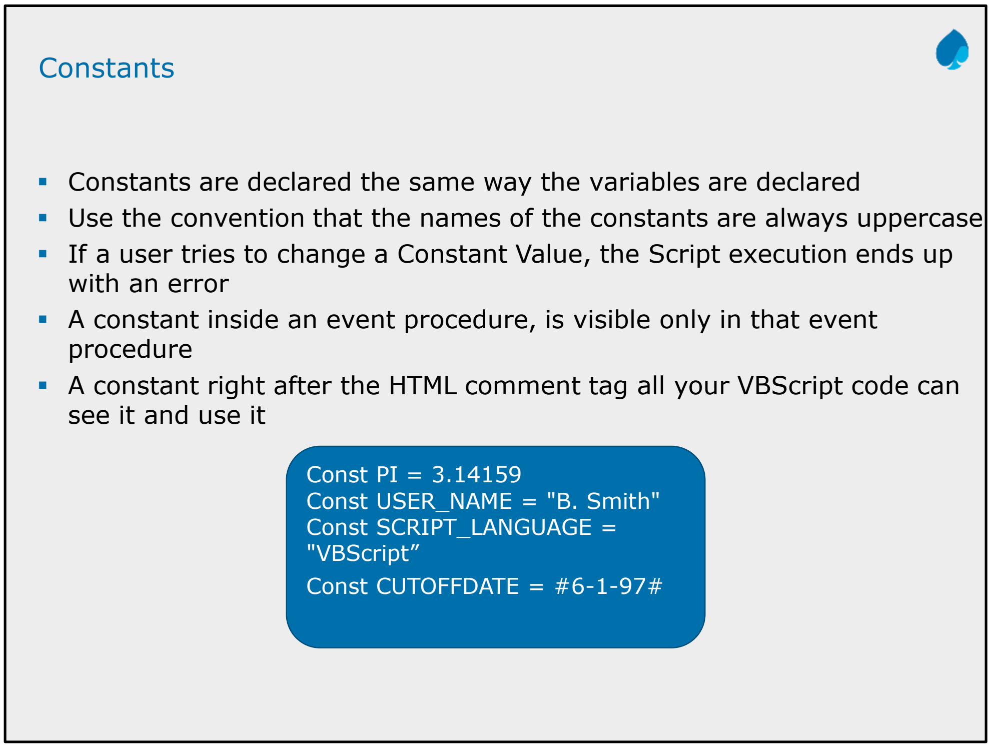
Additional notes for instructor

## Demo

- Demo on Using different declaration of Variables

Add the notes here.

Instructor Notes:



## Constants

- Constants are declared the same way the variables are declared
- Use the convention that the names of the constants are always uppercase
- If a user tries to change a Constant Value, the Script execution ends up with an error
- A constant inside an event procedure, is visible only in that event procedure
- A constant right after the HTML comment tag all your VBScript code can see it and use it

```
Const PI = 3.14159
Const USER_NAME = "B. Smith"
Const SCRIPT_LANGUAGE =
"VBScript”
Const CUTOFFDATE = #6-1-97#
```

**Constants**

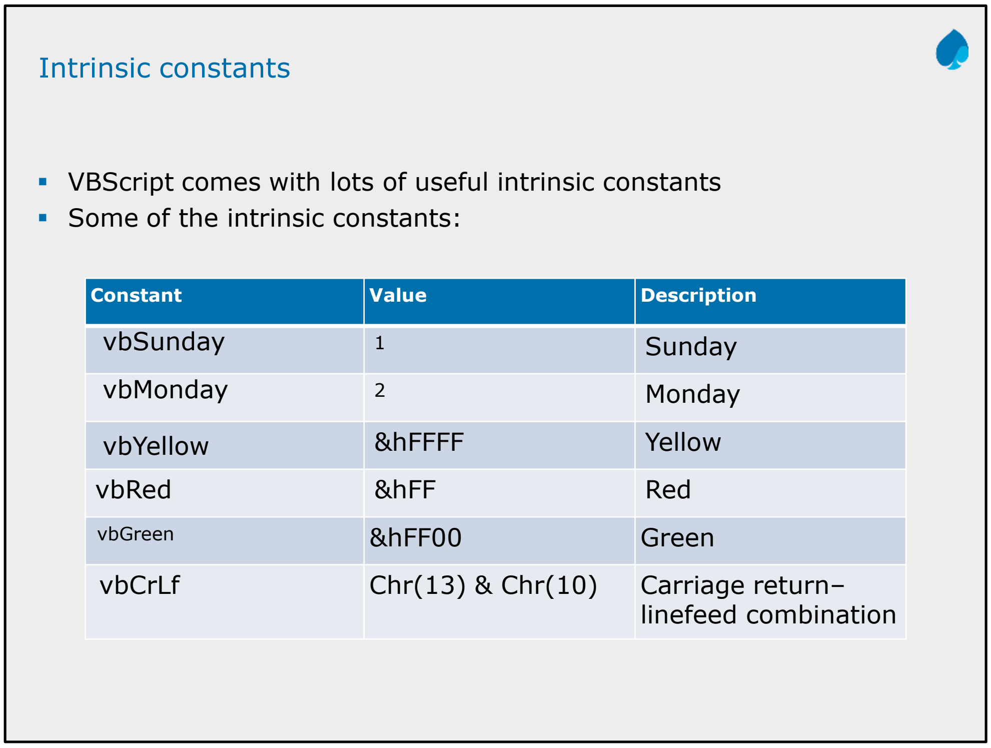
The keyword for telling VBScript that something is a constant is simply const, as shown in the following example:

```
Const PI = 3.14159
```

You might also set up string constants using the same keyword:

```
Const USER_NAME = "B. Smith"
Const SCRIPT_LANGUAGE = "VBScript"
```

Instructor Notes:



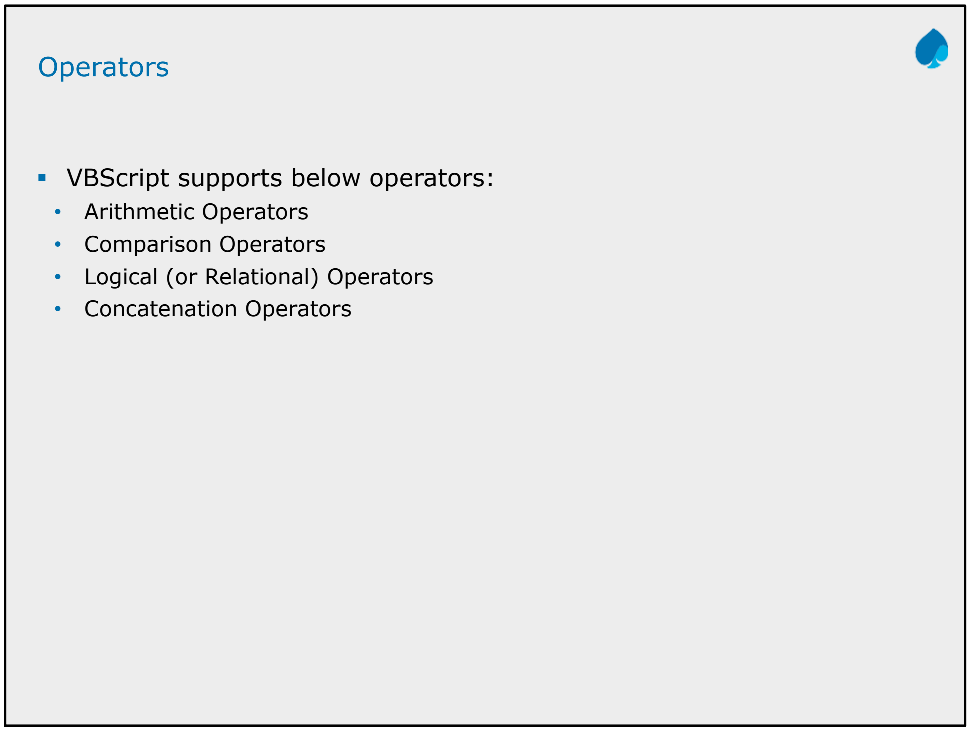
## Intrinsic constants

- VBScript comes with lots of useful intrinsic constants
- Some of the intrinsic constants:

| Constant | Value | Description |
|---|---|---|
| vbSunday | 1 | Sunday |
| vbMonday | 2 | Monday |
| vbYellow | &hFFFF | Yellow |
| vbRed | &hFF | Red |
| vbGreen | &hFF00 | Green |
| vbCrLf | Chr(13) & Chr(10) | Carriage return–linefeed combination |

VBScript comes with lots of useful built-in constants. These are constants such as vbRed and vbYellow for colors, and vbSunday and vbMonday for date functions.
VBScript does not follow the convention that built-in constants are uppercase.

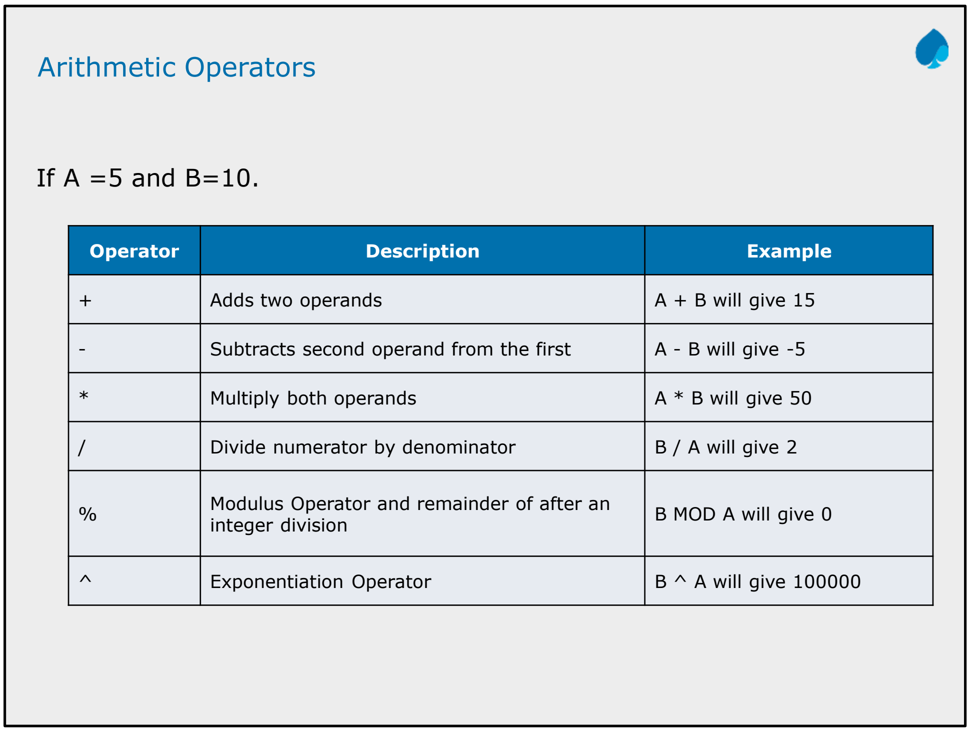
Instructor Notes:



## Operators

- VBScript supports below operators:
  - Arithmetic Operators
  - Comparison Operators
  - Logical (or Relational) Operators
  - Concatenation Operators

Instructor Notes:



## Arithmetic Operators

If A =5 and B=10.

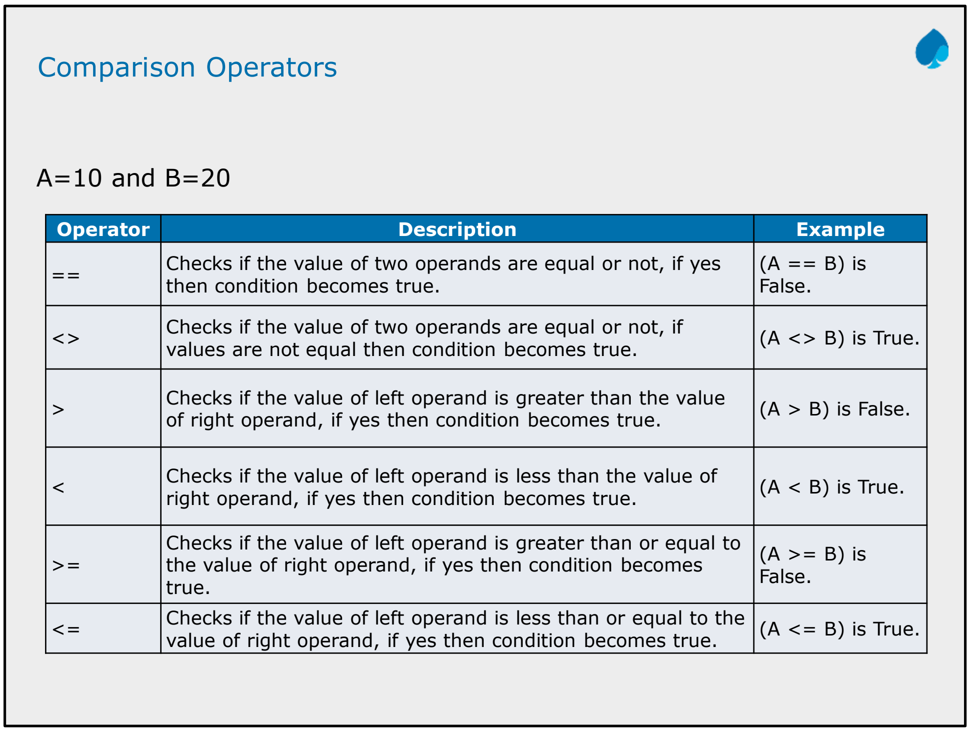
| Operator | Description | Example |
|---|---|---|
| + | Adds two operands | A + B will give 15 |
| - | Subtracts second operand from the first | A - B will give -5 |
| * | Multiply both operands | A * B will give 50 |
| / | Divide numerator by denominator | B / A will give 2 |
| % | Modulus Operator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

The ordinary division symbol (/) gives you a value that has a decimal point. The integer division symbol (\) throws away the remainder in order to give you an integer.
For example, 7\3 = 2.

Since ordinary division always produces a number with a decimal point, use integer division or the Mod operator if you really want to work with integers. The Mod operator gives you the remainder after integer division.
For example, 7 Mod 3 = 1.

When one integer perfectly divides another, there is no remainder, so the Mod operator gives 0: 8 Mod 4 = 0.
The term for a combination of numbers, variables, and operators from which VBScript can extract a value is numeric expression

Instructor Notes:



## Comparison Operators

A=10 and B=20

| 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. |

As an example, suppose you want to prevent a "divide-by-zero" error when a user enters data in a text box. Use a fragment like this:

```
Do
Number = InputBox("Please enter a nonzero number.")
Loop Until Number <> 0
```

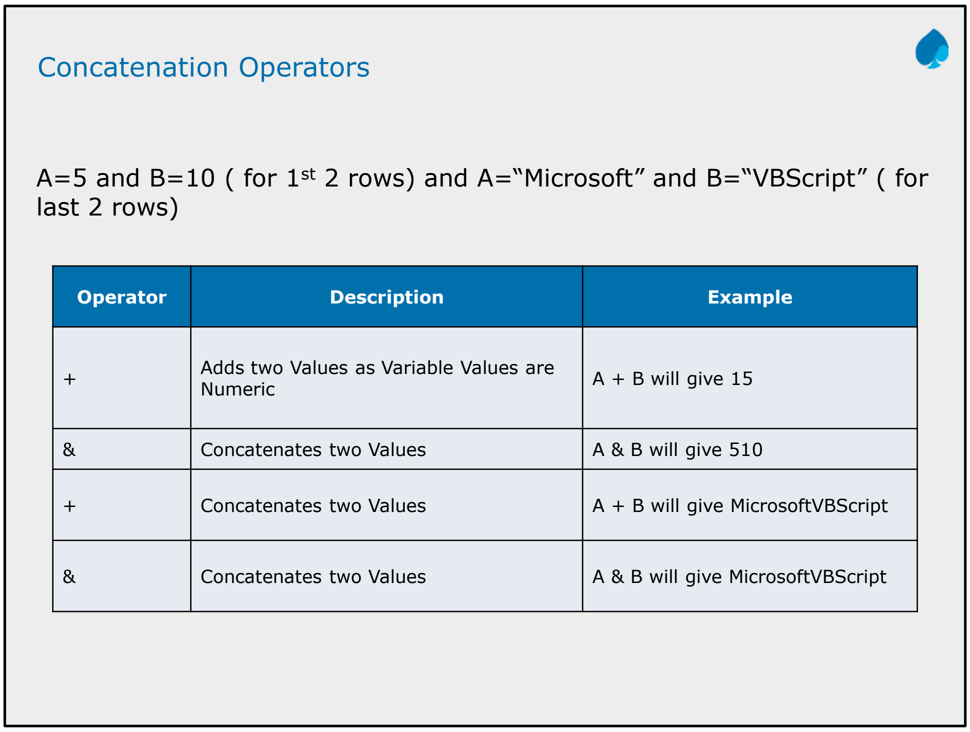
Instructor Notes:

## Logical Operators:

A=10 and B=0

| 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 are True then condition becomes true. | a<>0 OR b<>0 is true. |
| NOT | Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false. | NOT(a<>0 OR b<>0) is false. |
| XOR | Called Logical Exclusion. It is the combination of NOT and OR Operator. If one, and only one, of the expressions evaluates to True, result is True. | (a<>0 XOR b<>0) is false. |

Instructor Notes:

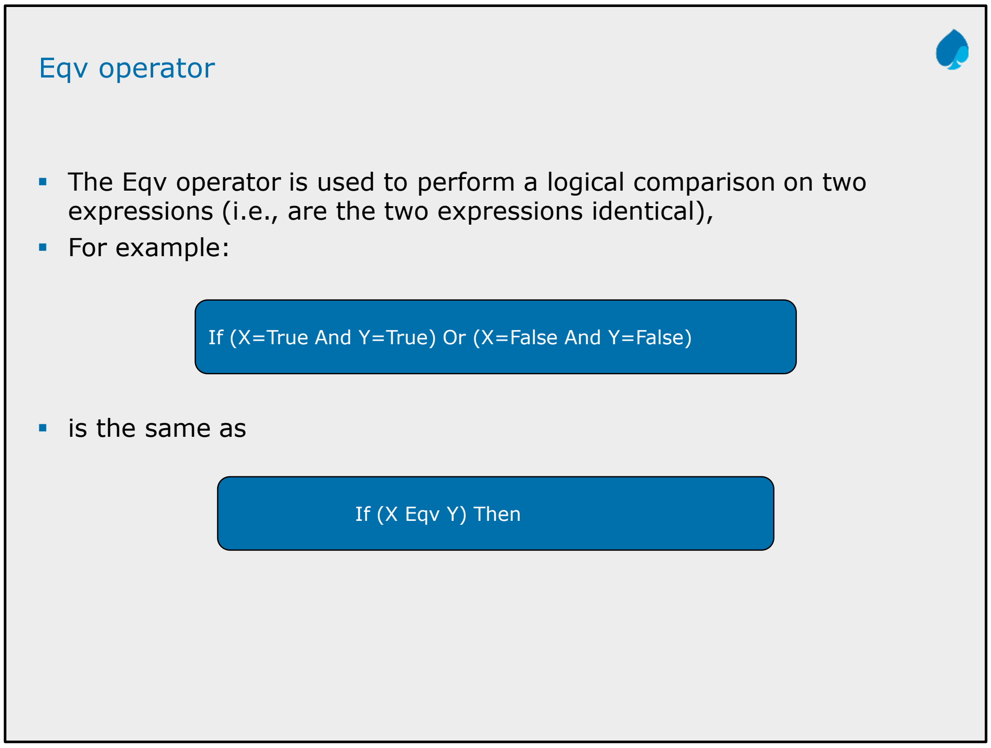


## Concatenation Operators

A=5 and B=10 ( for 1st 2 rows) and A=“Microsoft” and B=“VBScript” ( for last 2 rows)

| Operator | Description | Example |
|---|---|---|
| + | Adds two Values as Variable Values are Numeric | A + B will give 15 |
| & | Concatenates two Values | A & B will give 510 |
| + | Concatenates two Values | A + B will give MicrosoftVBScript |
| & | Concatenates two Values | A & B will give MicrosoftVBScript |

Instructor Notes:



## Eqv operator

- The Eqv operator is used to perform a logical comparison on two expressions (i.e., are the two expressions identical),
- For example:

```
If (X=True And Y=True) Or (X=False And Y=False)
```

- is the same as

```
If (X Eqv Y) Then
```

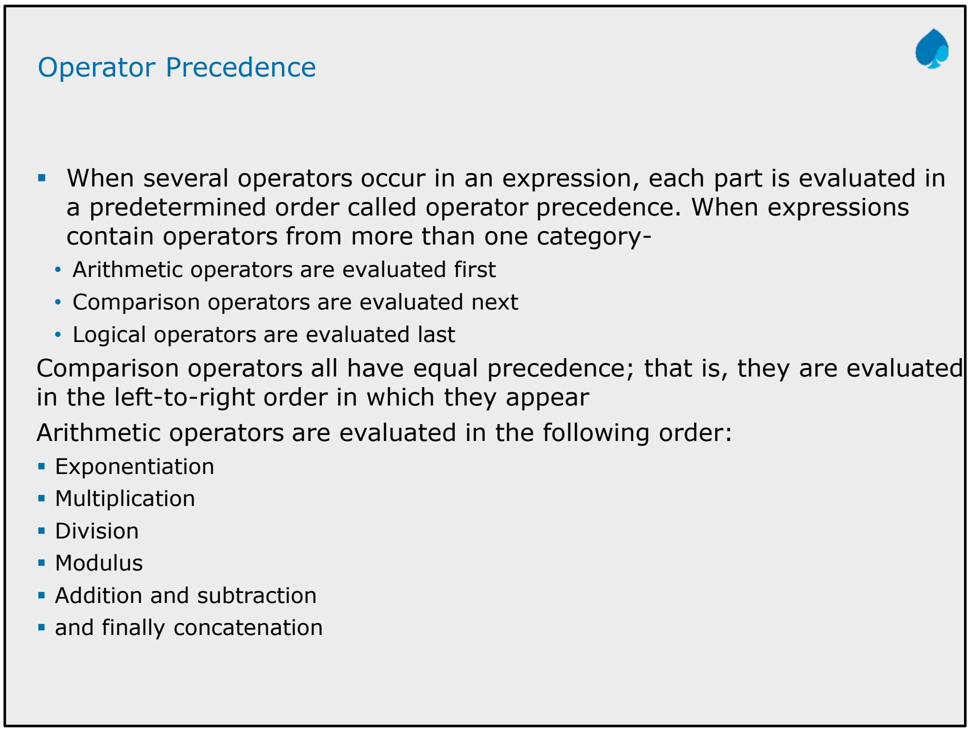
Instructor Notes:

## Eqv Operator –Examples

The truth table for Eqv is as follows:

| Expr1 | Expr2 | Expr1 EQV Expr2 |
|---|---|---|
| **True** | True | True |
| **True** | False | False |
| **False** | True | False |
| **False** | False | True |

Instructor Notes:



## Operator Precedence

- When several operators occur in an expression, each part is evaluated in a predetermined order called operator precedence. When expressions contain operators from more than one category-
  - Arithmetic operators are evaluated first
  - Comparison operators are evaluated next
  - Logical operators are evaluated last

Comparison operators all have equal precedence; that is, they are evaluated in the left-to-right order in which they appear

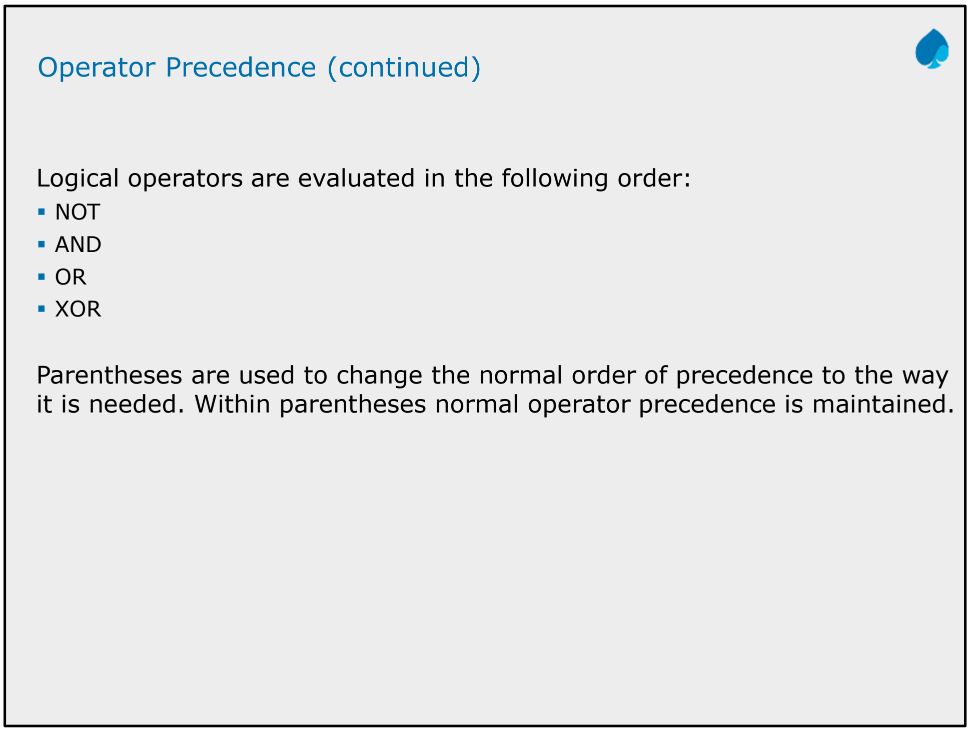
Arithmetic operators are evaluated in the following order:

- Exponentiation
- Multiplication
- Division
- Modulus
- Addition and subtraction
- and finally concatenation

For example, if you try to calculate the expression a = 5-4*3/5^6, what you expect as the result?

The result will be 4.999232
How? The exponentiation comes first, then come multiplication and division and finally comes subtraction.
So the above expression gets calculated like this: 5-4*3/(5^6) --> 5-(4*3)/30--> 5-(12/30) --> 5-.04 --> 4.999232

Instructor Notes:



## Operator Precedence (continued)

Logical operators are evaluated in the following order:

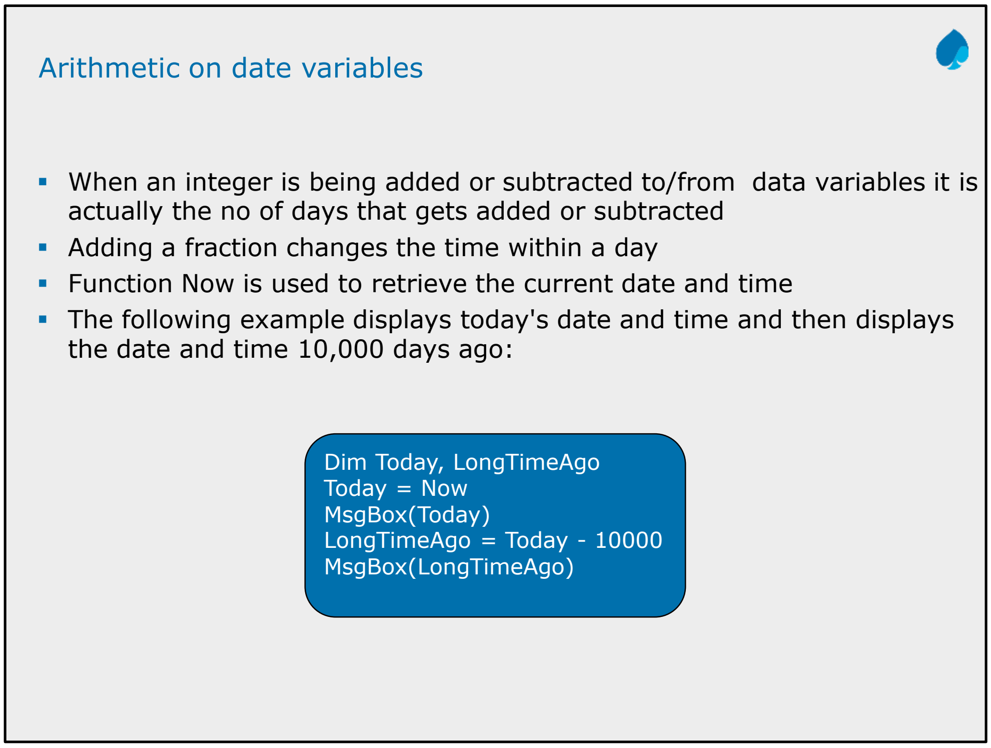
- NOT
- AND
- OR
- XOR

Parentheses are used to change the normal order of precedence to the way it is needed. Within parentheses normal operator precedence is maintained.

In the earlier example Suppose, you want to calculate 5-4 first, then you should write the expression as a = (5-4)*3/5^6.

Now you get the value of as a as 1*3/5^6 --> 1*3/30-->3/30--> 0.1

Instructor Notes:



## Arithmetic on date variables

- When an integer is being added or subtracted to/from  data variables it is actually the no of days that gets added or subtracted
- Adding a fraction changes the time within a day
- Function Now is used to retrieve the current date and time
- The following example displays today's date and time and then displays the date and time 10,000 days ago:

```
Dim Today, LongTimeAgo
Today = Now
MsgBox(Today)
LongTimeAgo = Today - 10000
MsgBox(LongTimeAgo)
```

**Arithmetic on date variables**

VBScript makes it easy to do calculations with date variables. If you add or subtract an integer, you add or subtract that many days.
Adding a fraction changes the time within a day. You can use the function Now to retrieve the current date and time.

Instructor Notes:

Additional notes for instructor

## Demo

- Demo on Operator precedence
- Demo on Arithmetic on date variable

Add the notes here.

Instructor Notes:

## Type conversions

- The Syntax used to carry on type conversions in VBScript is:

***Variable_name = Conversion_function_name(expression)***

Some of the Conversion function names are:

1. Cint : Converts an expression to an Integer, rounding if necessary
2. CLng : Converts an expression to a long integer, rounding if necessary
3. CSng : Converts an expression to a single-precision number
4. CDbl : Converts an expression to a double-precision number
5. Ccur : Converts an expression to a variable of type Currency
6. CStr : Converts an expression to a string
7. Cbool : Converts an expression to a Boolean value (True or False)
8. Cbyte : Converts an expression to a variable of type Byte
9. Cdate : Converts a string to a variable of type Date

For example, the following line ensures that VBScript regards the zip code as a text string and not as a number:

ZipCode = CStr(22203)

Instructor Notes:

Additional notes for instructor

## Demo

- Demo on Type conversions

Add the notes here.

Instructor Notes:

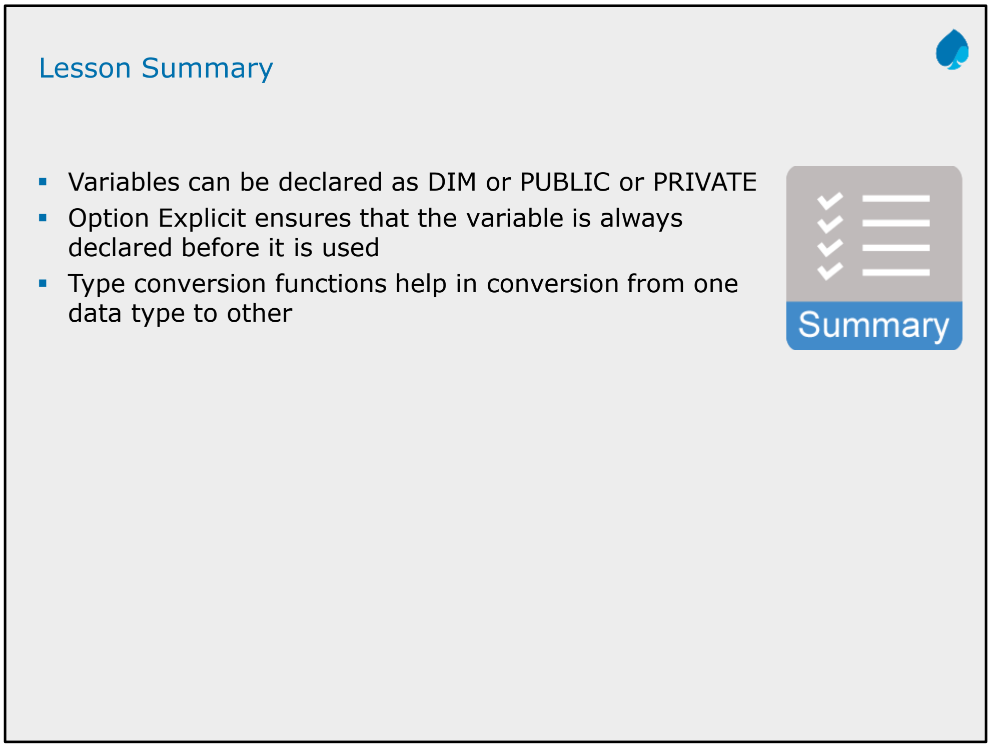
Additional notes for instructor

## Lab

Lab 2

Instructor Notes:

Additional notes for instructor



## Lesson Summary

- Variables can be declared as DIM or PUBLIC or PRIVATE
- Option Explicit ensures that the variable is always declared before it is used
- Type conversion functions help in conversion from one data type to other

Summary

Instructor Notes:

1. False
2. Logical
3. Now

## Review - Questions

- Question1: Intrinsic Constants should always be in upper case. State True or False.
- Question 2:________________Operators are operated at the end.
- Question 3: _______ is used to retrieve the current date and time.