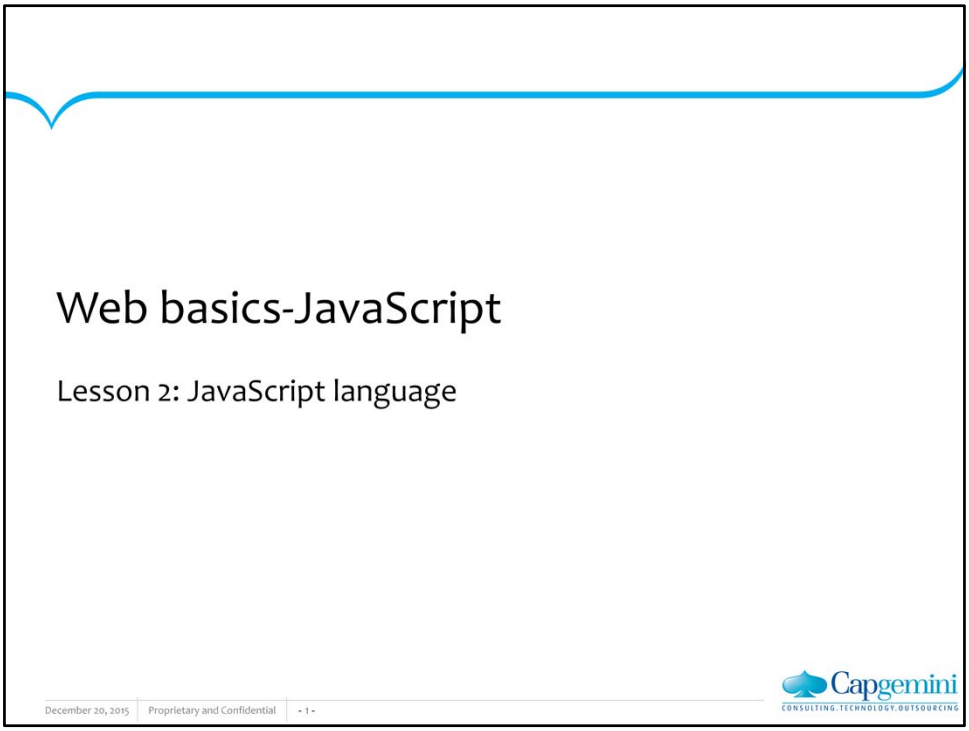
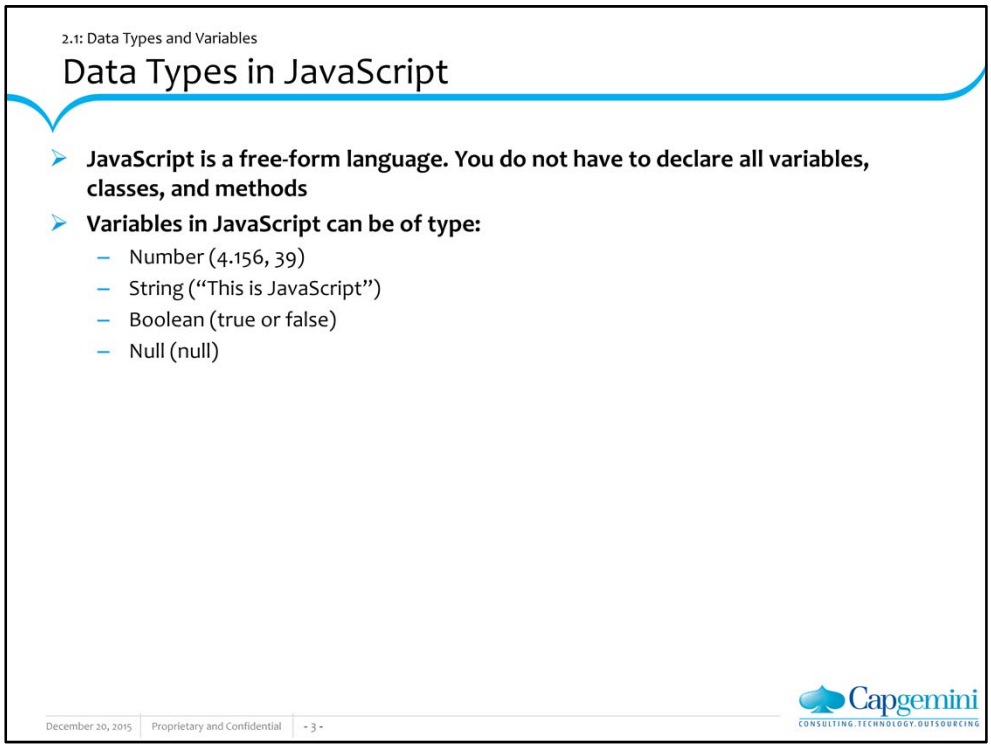

Web basics-JavaScript
Lesson 2: JavaScript language
December 20, 2015
Proprietary and Confidential
- 1 -
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Lesson Objectives

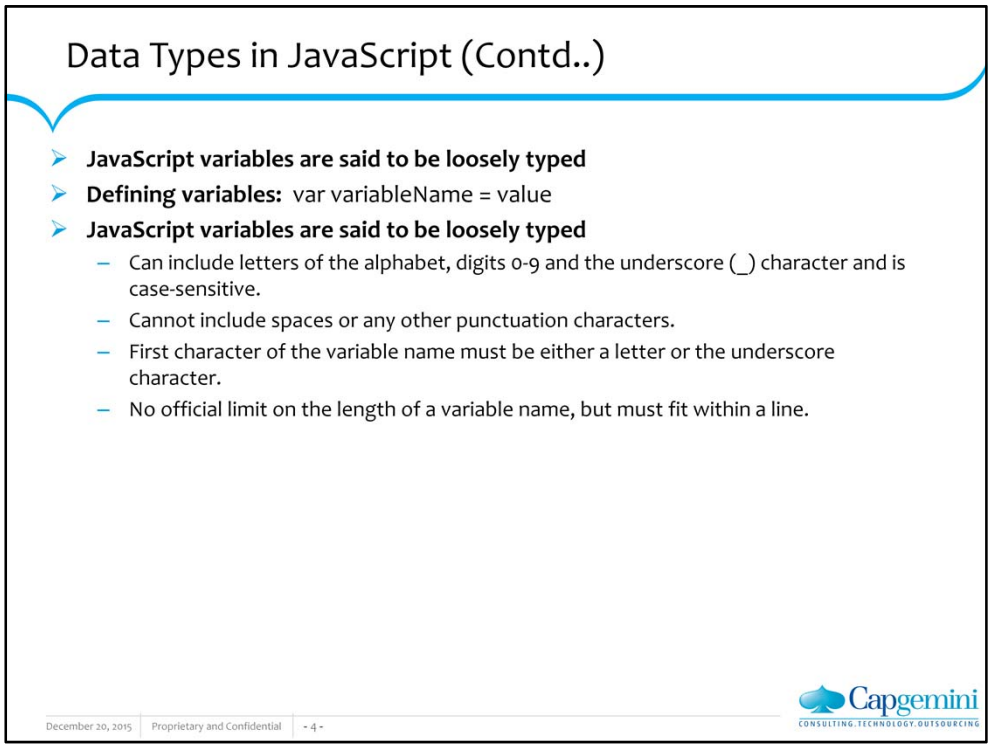
- Data Types and Variables
- JavaScript Operators
- Control Structures and Loops
- JavaScript Functions

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING



Data Types in JavaScript:

Although the number of data types is small, they are sufficient for the tasks that JavaScript performs. Notice that there is no distinction between integers and real numbers; both types are just numbers. JavaScript does not provide an explicit data type for a date. However, there are related functions and a built-in date object that enable the Web page designer to manage dates.



Defining Variables:

There are specific rules you must follow when choosing variable names.

Variable names can include letters of the alphabet, both upper and lowercase. They can also include the digits 0-9 and the underscore (_) character.

Variable names cannot include spaces or any other punctuation characters.

The first character of the variable name must be either a letter or the underscore character.

Variable names are case-sensitive; totalnum, Totalnum, and TotalNum are separate variable names.

There is no official limit on the length of a variable name, but it must fit within one line.

JavaScript variables are said to be loosely typed. To declare a variable for a JavaScript program, you would write this:

```
var variablename = value ;
```

2.2: JavaScript Operators

Arithmetic Operator

Operator	Description	Example	Result
+	Addition	2 + 2	4
-	Subtraction	5 – 2	3
*	Multiplication	4 * 5	20
/	Division	5 / 2	2.5
%	Modulus	10 % 8	2
++	Increment	x = 5; x++	x = 6
--	Decrement	x = 5; x--	x = 4

2.2: JavaScript Operators

Comparison Operator

Operator	Description	Example	Result
==	is equal to	5 == 8	false
!=	is not equal	5 != 8	true
>	is greater than	5 > 8	false
<	is less than	5 <= 8	true
>=	is greater or equal	5 >= 8	false
<=	is less or equal	5 <= 8	true

2.2: JavaScript Operators

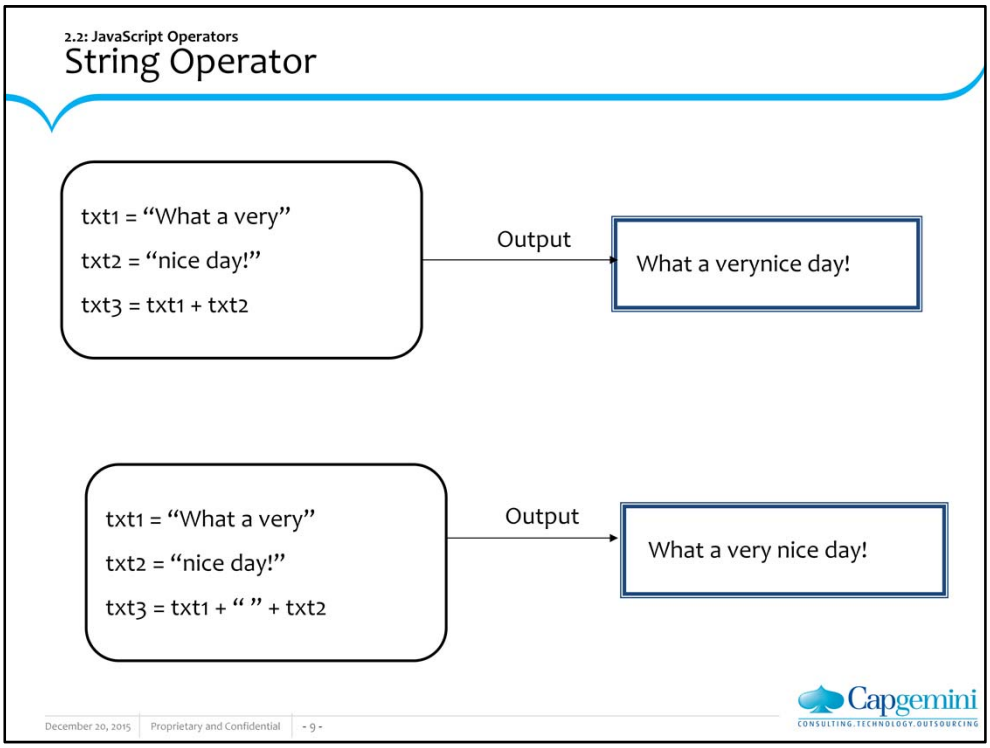
Assignment Operator

Operator	Example	Is same as
+=	x += y	x = x + y
-=	x -= y	x = x – y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y

2.2: JavaScript Operators

Logical Operator

Operator	Description	Example
&&	and	x = 6; y = 3 x < 10 && y > 1 returns true
||	or	x = 6; y = 3 x < 10 || y > 5 returns true
!	not	x = false !x returns true



String Operator:

```
txt1="What a very"
txt2="nice day!"
txt3=txt1+txt2
```

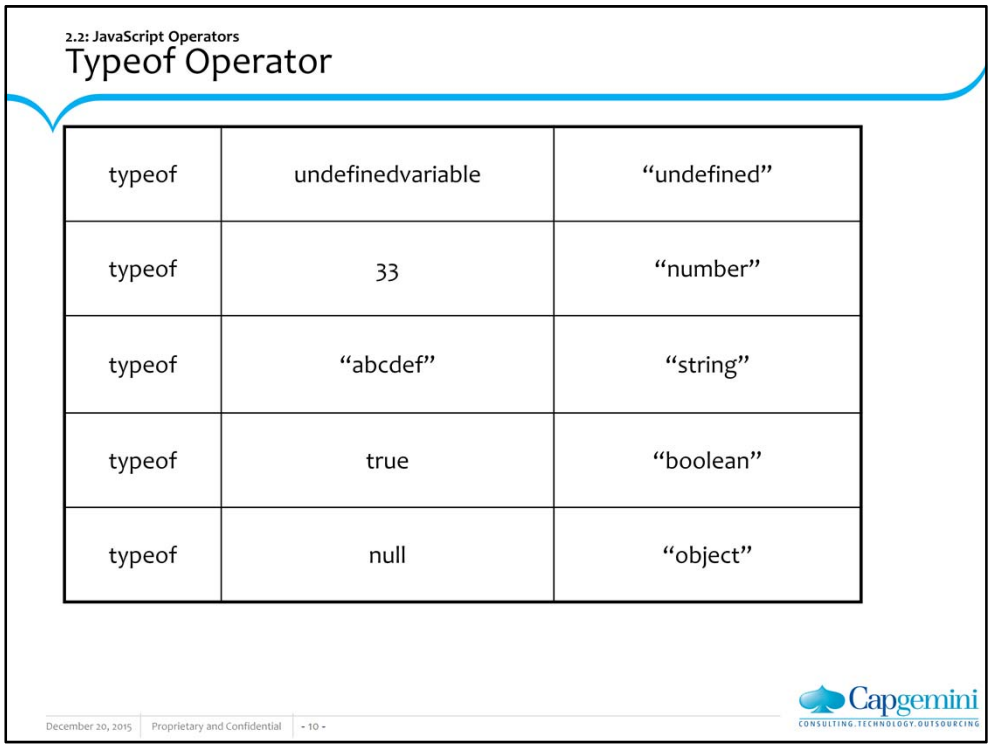
A string is most often a text, for example "Hello World!". To stick two or more string variables together, use the + operator.
The variable txt3 now contains "What a verynice day!". To add a space between two string variables, insert a space into the expression, OR in one of the strings.

```
txt1="What a very"
txt2="nice day!"
txt3=txt1+" "+txt2
```

Or

```
txt1="What a very "
txt2="nice day!"
txt3=txt1+txt2
```

The variable txt3 now contains "What a very nice day!".



Typeof Operator:

The typeof operator returns the type of data that its operand currently holds.

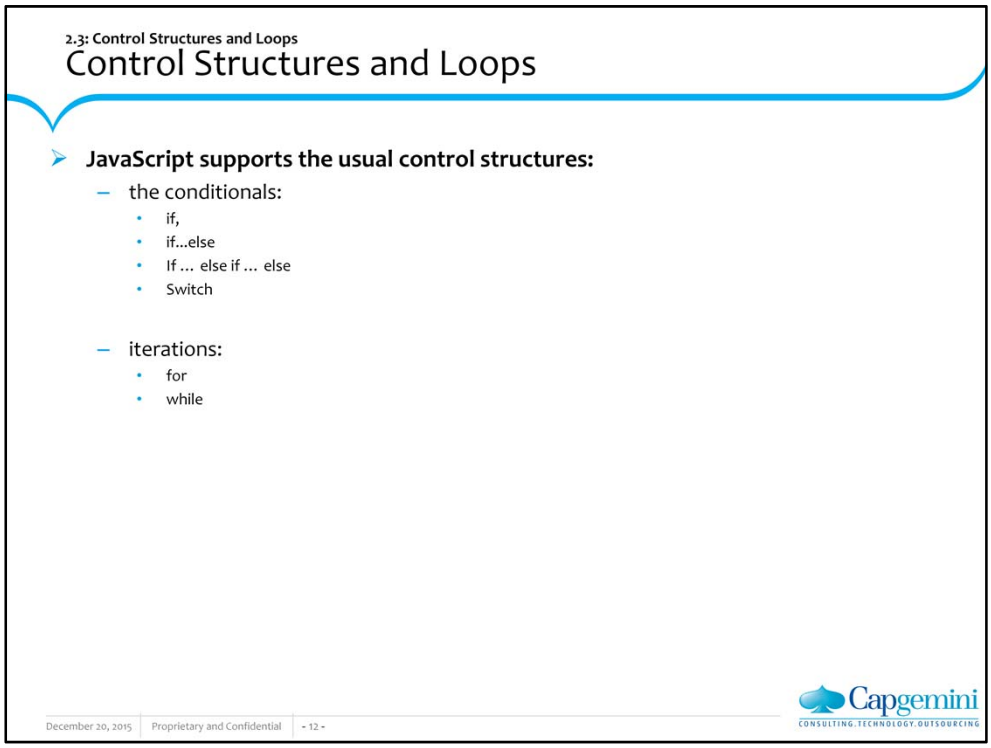
```
<HTML>
<HEAD>
<TITLE>Using typeof</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
<!-- var num1=20
 var str1="abc"
 var bool1=true
 var num2=null
 var var1;
 document.write("type of str1 : "+typeof(str1)+"<BR>")
 document.write("type of num1 : "+typeof(num1)+"<BR>")
document.write("type of bool1 : "+typeof(bool1)+"<BR>")
document.write("type of num2 : "+typeof(num2)+"<BR>") -->
</SCRIPT>
</BODY>
</HTML>
```

Example 2.1 typeof operator (typeof.html)

Demo

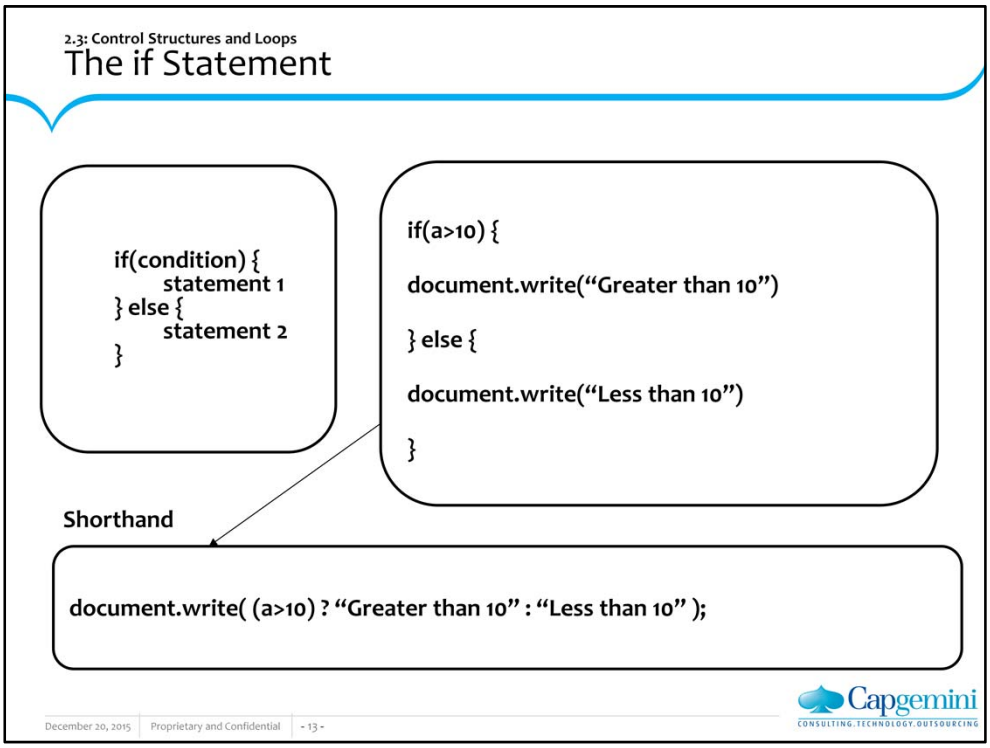
- **Typeof_ex.html**

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING



Control Structures and Loops

Conditional statements are used to perform different actions based on different conditions. Loops execute a block of code for specified number of times or while a specified condition is true.



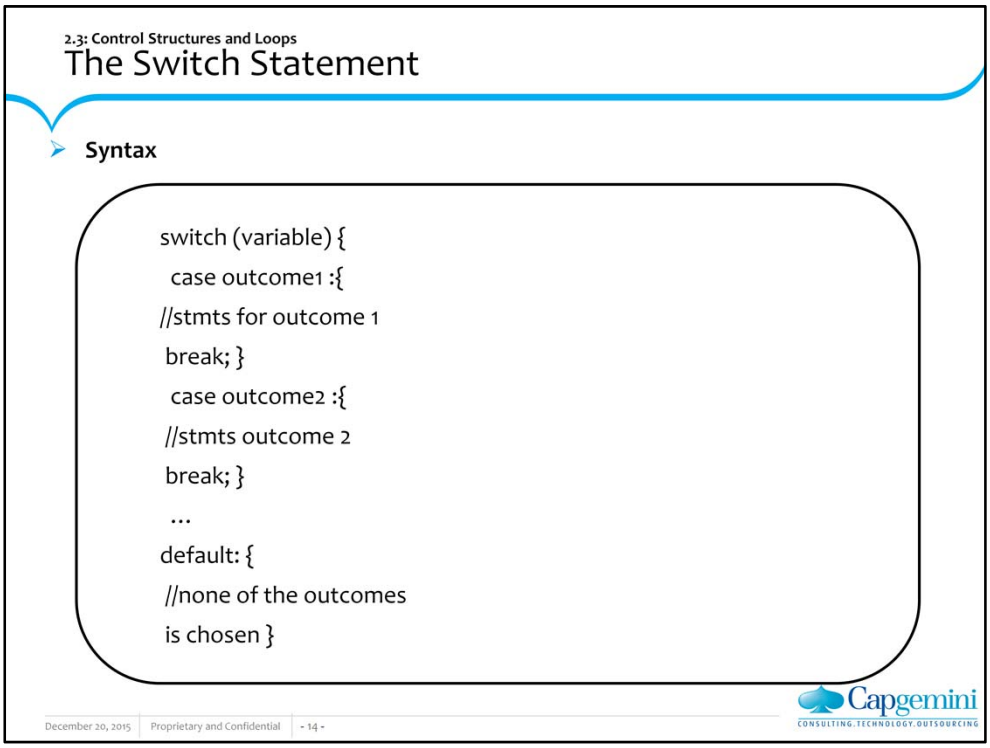
The if Statement:

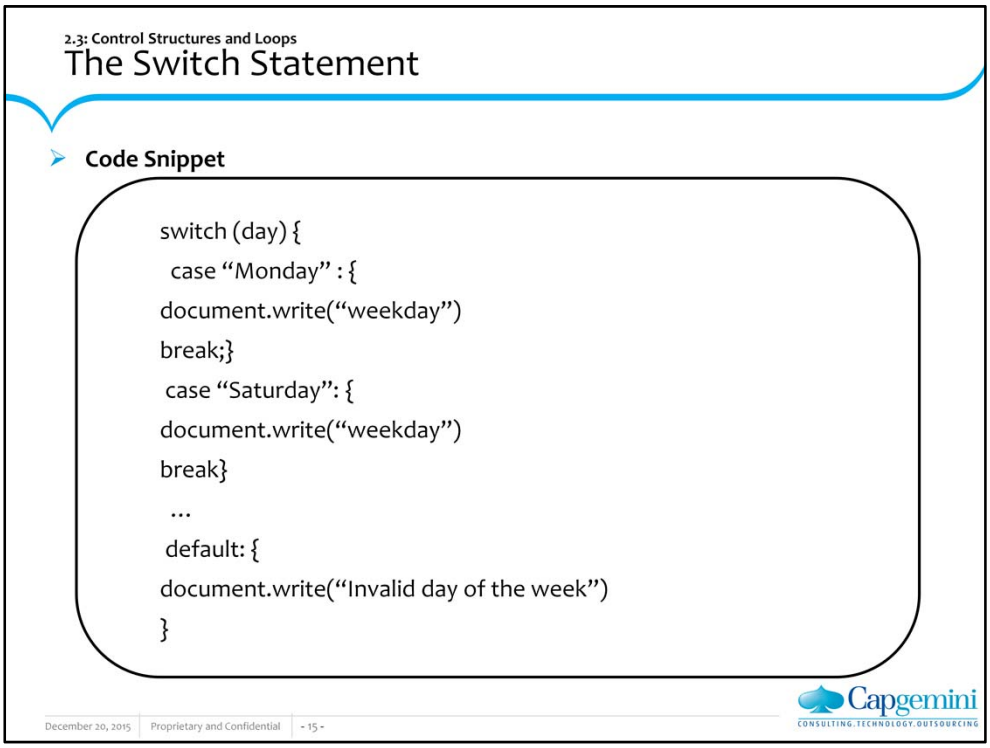
The condition is any JavaScript expression that evaluates to the Boolean type, either true or false.

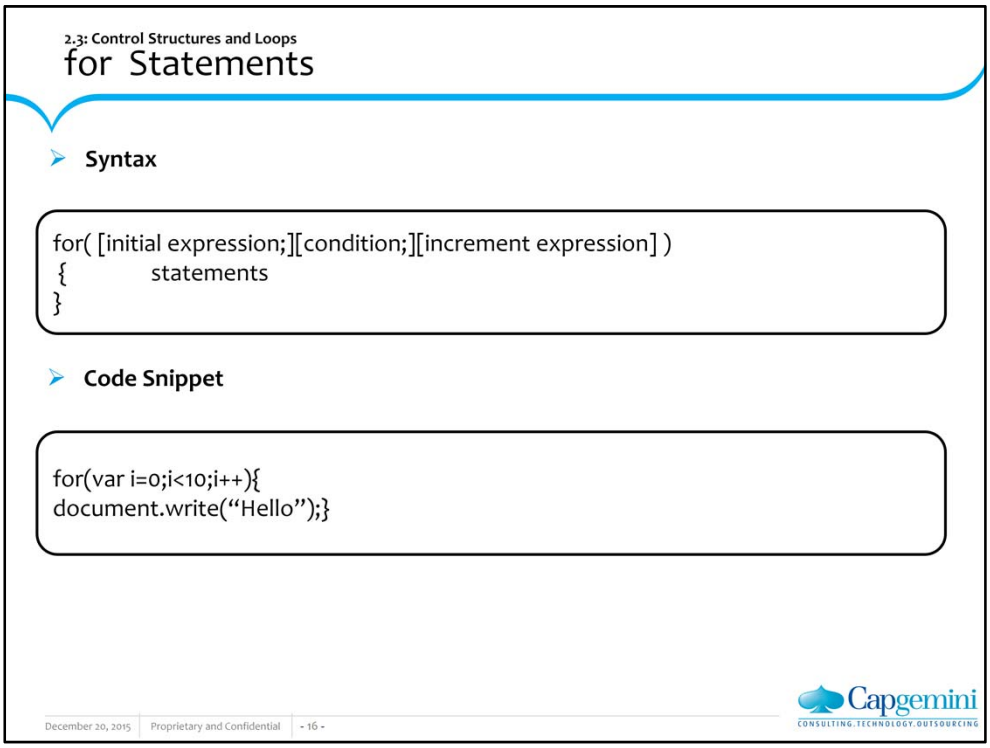
The example as shown on the slide.

A shorthand method can also be used for these types of statements, where ? indicates the 'if' portion and : indicates the 'else' portion. This statement is equivalent to the previous example:

The equivalent shorthand method is also seen on the slide.


2.3: Control Structures and Loops
The Switch Statement
Syntax
switch (variable) {
case outcome1 :{
//stmts for outcome 1
break; }
case outcome2 :{
//stmts outcome 2
break; }
...
default: {
//none of the outcomes
is chosen }
December 20, 2015
Proprietary and Confidential
- 14 -
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING


2.3: Control Structures and Loops
The Switch Statement
Code Snippet
switch (day) {
case "Monday" : {
document.write("weekday")
break;}
case "Saturday": {
document.write("weekday")
break}
...
default: {
document.write("Invalid day of the week")
}
December 20, 2015
Proprietary and Confidential
- 15 -
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING



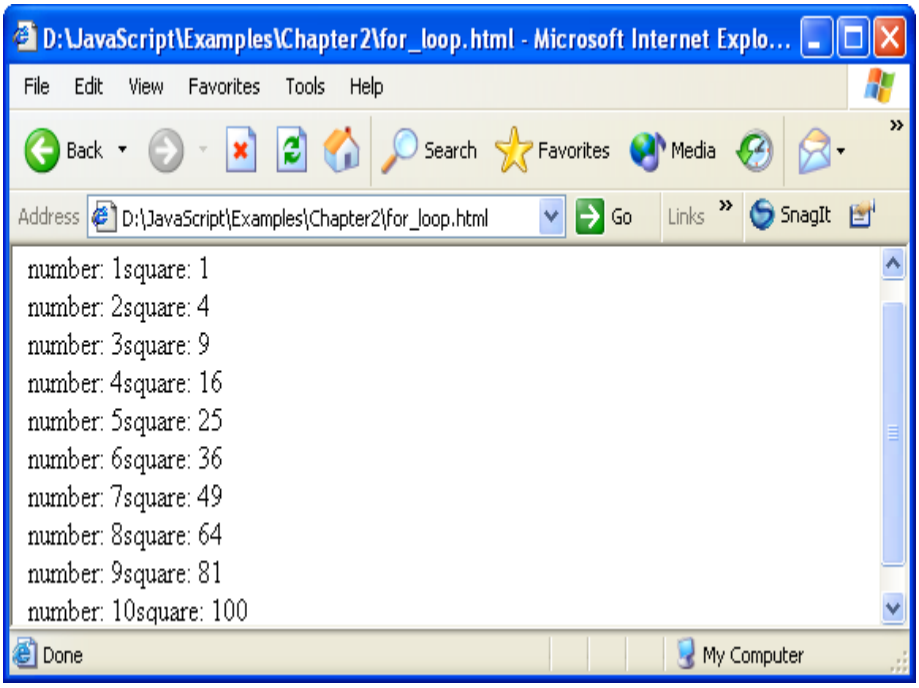
Looping Statements

The 'for' Statement:

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
<!-- hide script
for (i=1; i<=10; i++)
{
  sq=i*i
  document.write("number: " + i + "square: " + sq + "<BR>")
}
// end script hiding -->
</SCRIPT>
</HEAD>
<BODY></BODY></HTML>
```

Example 2.2 For Construct (for_loop.html)

And it produces the output as:



The 'while' Statement

The while statement continues to repeat the loop as long as the condition is true. The syntax for the while statement is as follows:

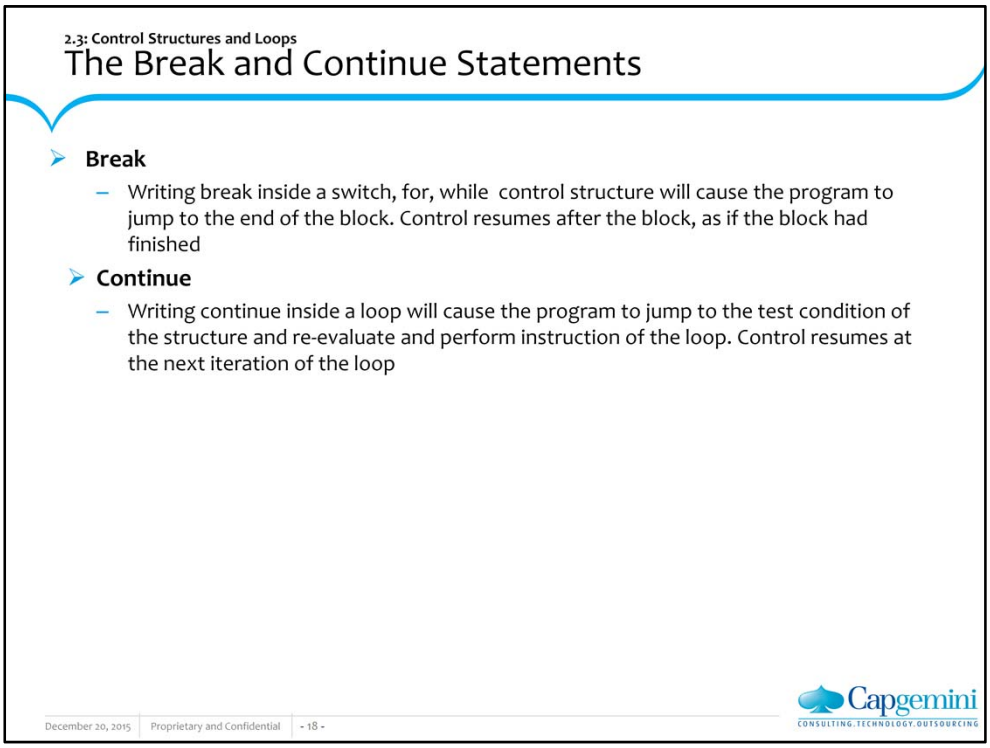
while (condition):

```
{
  statements
}
```

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
<!-- hide script
i=1
while (i<=10)
{
  sq=I*i
  document.write("number: " + i + "square: " + sq + "<BR>")
  i++
}
// end script hiding -->
</SCRIPT>
</HEAD>
<BODY></BODY>
</HTML>
```

Example 2.3 While Construct

And it produces the output as in the previous screen shot.



The Break statement:

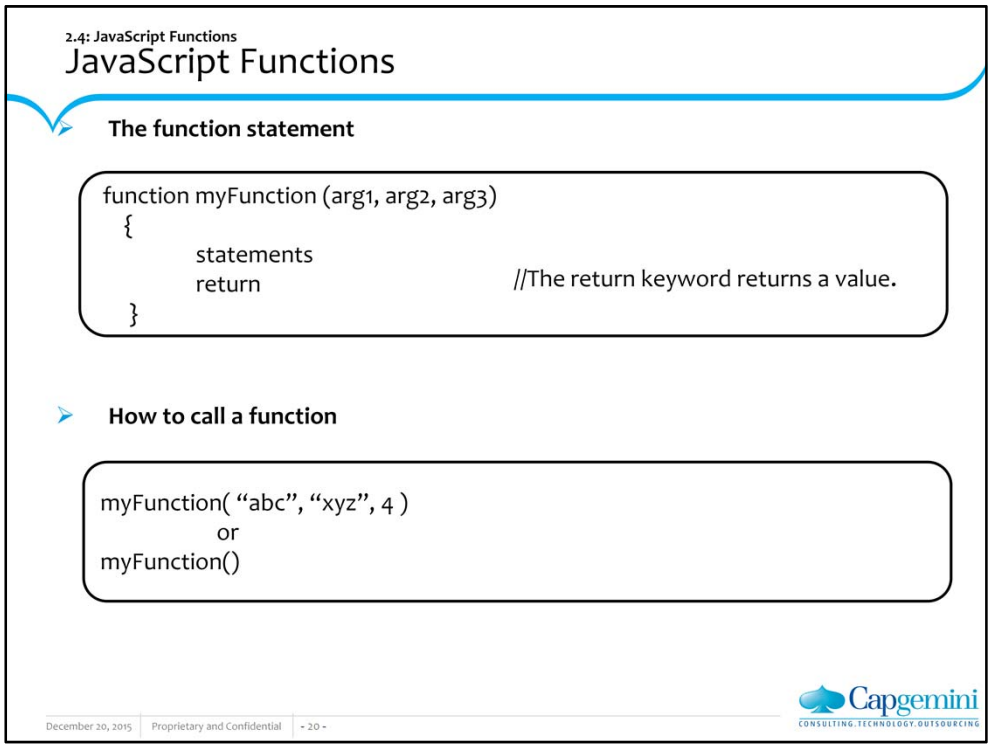
This statement is used to break out of the current 'for' or 'while' loop. Control resumes after the loop, as if it had finished.

The Continue Statement:

This statement continues a 'for' or 'while' loop without executing the rest of the loop. Control resumes at the next iteration of the loop.

Demo

- For_ex.html



The Function Statement:

A function contains some code that will be executed by an event or a call to that function. A function is a set of statements. You can reuse functions within the same script, or in other documents. You define functions at the beginning of a file (in the head section), and call them later in the document.

The syntax of a typical function is as follows:

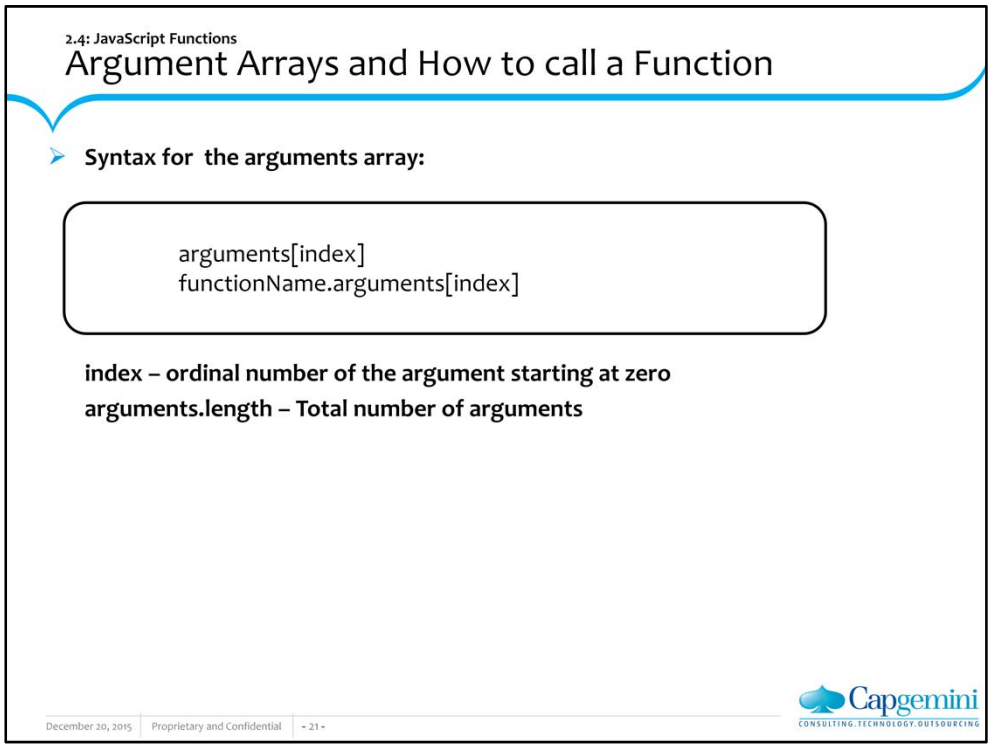
```
function myfunction(arg1, arg2, arg3)

{
  statements
}
```

How to Call a Function?

A function is not executed before it is called. You can call a function containing arguments:

```
myfunction("abc","xyz",4)
or without arguments:
myfunction()
```



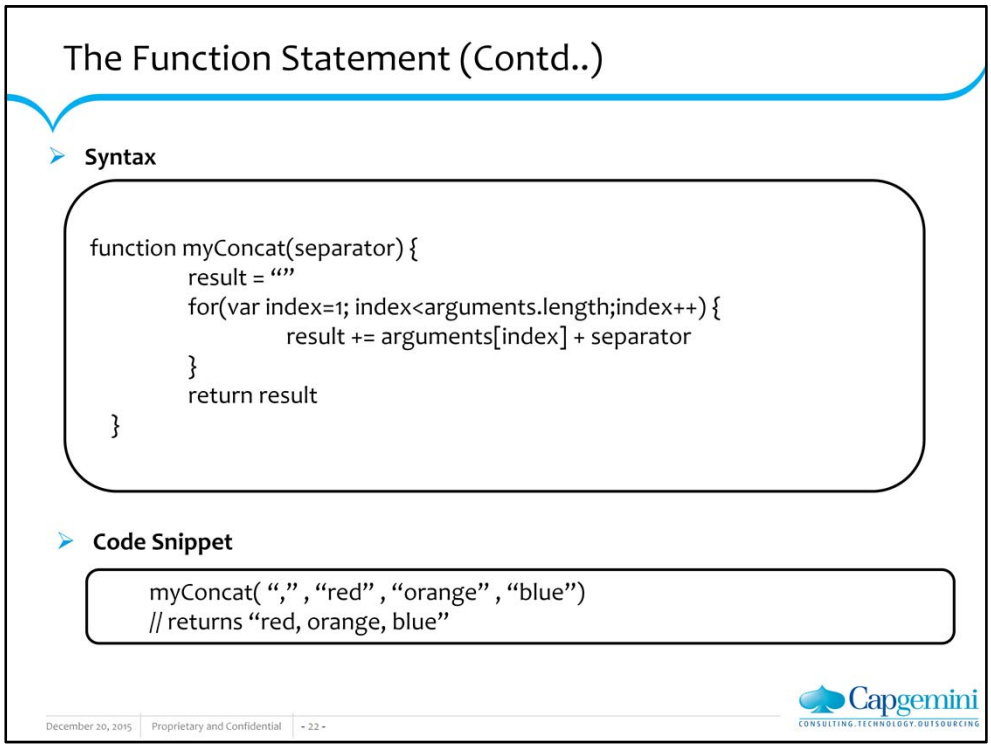
Using the arguments Array

The arguments of a function are maintained in an array. Within a function, you can address the parameters passed to it as follows:

arguments[*i*]

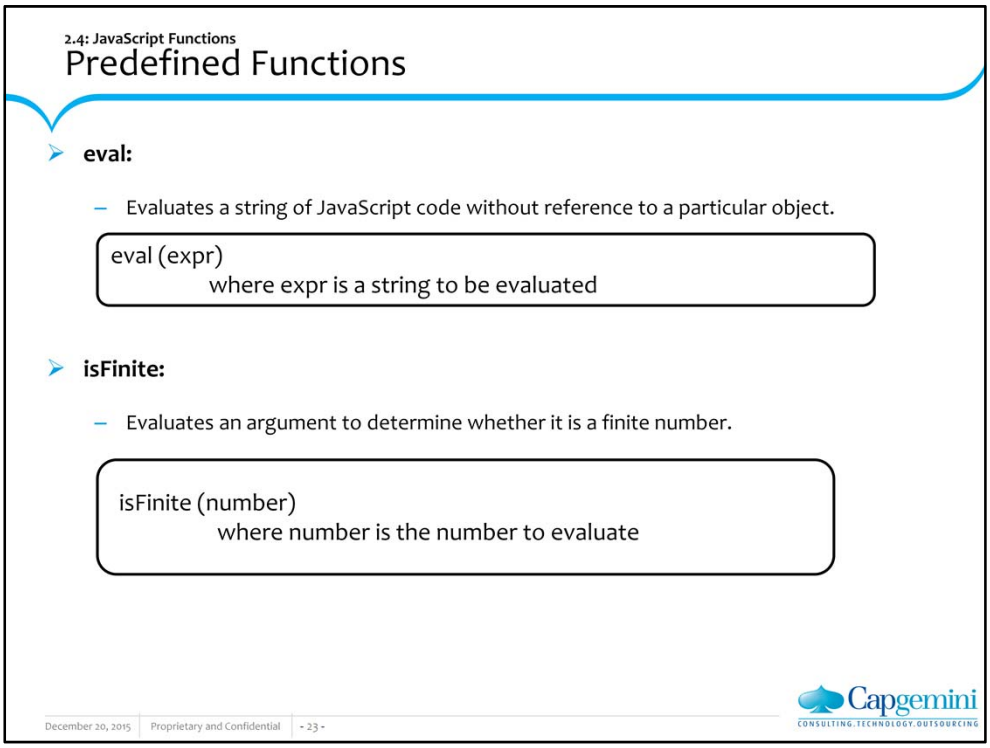
functionName.arguments[i]

where i is the ordinal number of the argument, starting at zero. So, the first argument passed to a function would be arguments[0]. The total number of arguments is indicated by arguments.length. Using the arguments array, you can call a function with more arguments than it is formally declared to accept. This is often useful if you don't know in advance how many arguments will be passed to the function. You can use arguments.length to determine the number of arguments actually passed to the function, and then treat each argument using the arguments array.



For example, consider a function that concatenates several strings. The only formal argument for the function is a string that specifies the characters that separate the items to concatenate. The function is defined as shown on the slide.
You can pass any number of arguments to this function, and it creates a list using each argument as an item in the list.

```
// returns "red, orange, blue, "
myConcat(", ","red","orange","blue")
// returns "elephant; giraffe; lion; cheetah;"
myConcat("; ","elephant","giraffe","lion", "cheetah")
// returns "sage. basil. oregano. pepper. parsley. "
myConcat(". ","sage","basil","oregano", "pepper", "parsley")
```

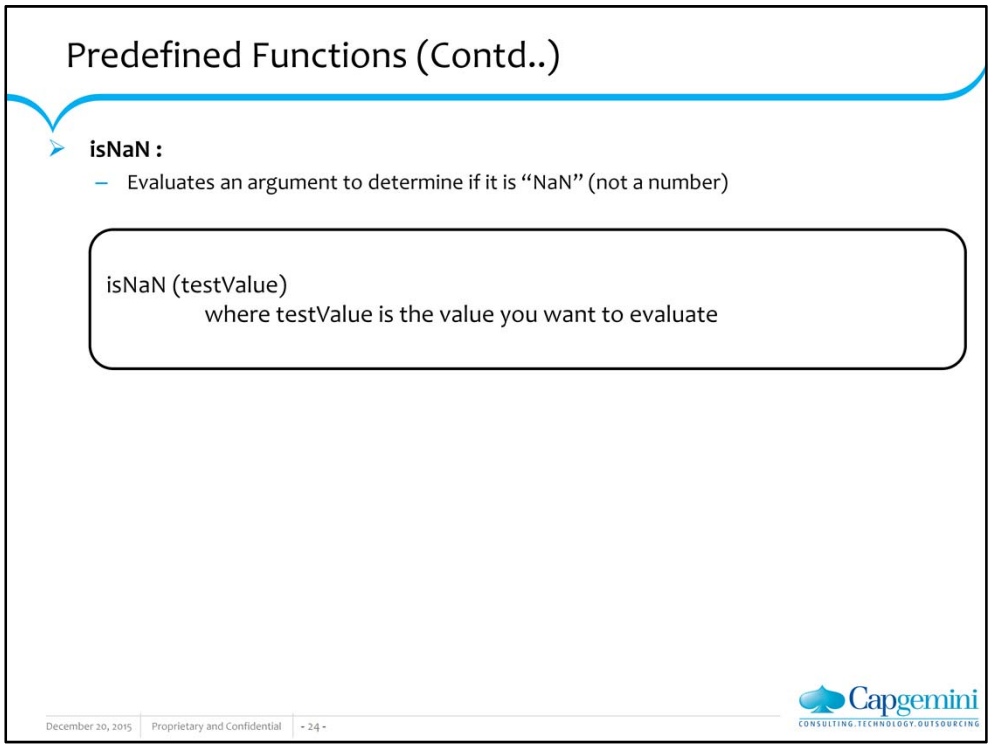


Predefined Functions:
JavaScript has several top-level predefined functions:
Eval, isFinite, isNaN, parseInt and parseFloat, Number and String

eval Function
The eval function evaluates a string of JavaScript code without reference to a particular object. The syntax of eval is:
eval(*expr*) where expr is a string to be evaluated.
If the string represents an expression, eval evaluates the expression. If the argument represents one or more JavaScript statements, eval performs the statements. Do not call eval to evaluate an arithmetic expression; JavaScript evaluates arithmetic expressions automatically.

isFinite Function
The isFinite function evaluates an argument to determine whether it is a finite number. The syntax of isFinite is:
isFinite(*number*) where number is the number to evaluate.
If the argument is NaN, positive infinity or negative infinity, this method returns false, otherwise it returns true. The following code checks client input to determine whether it is a finite number.

```
if(isFinite(ClientInput) == true)
{
  /* take specific steps */
}
```



isNaN Functions:

The isNaN function evaluates an argument to determine if it is "NaN" (not a number). The syntax of isNaN is:

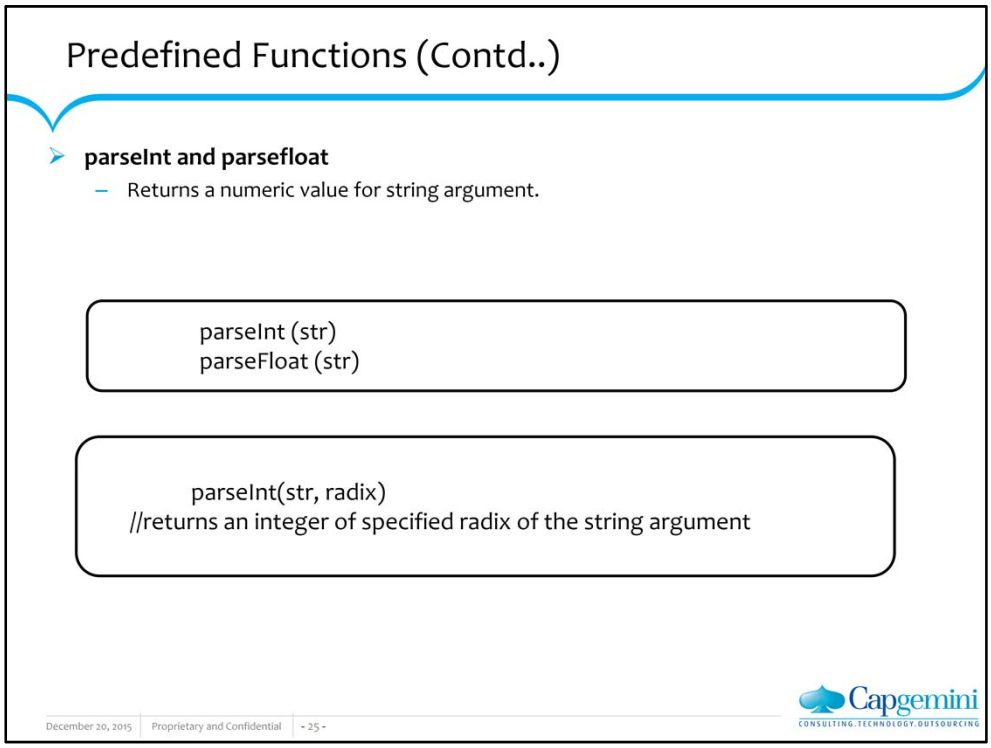
```
isNaN(testValue)
```

where testValue is the value you want to evaluate.

The parseFloat and parseInt functions return "NaN" when they evaluate a value that is not a number. isNaN returns true if passed "NaN," and false otherwise.

The following code evaluates floatValue to determine if it is a number and then calls a procedure accordingly:

```
floatValue=parseFloat(toFloat)
        if (isNaN(floatValue)) {
            notFloat()
          } else {
            isFloat()
          }
```



parseInt and parseFloat Functions:

The two "parse" functions, parseInt and parseFloat, return a numeric value when given a string as an argument.

The syntax of parseFloat is:

parseFloat(*str*)

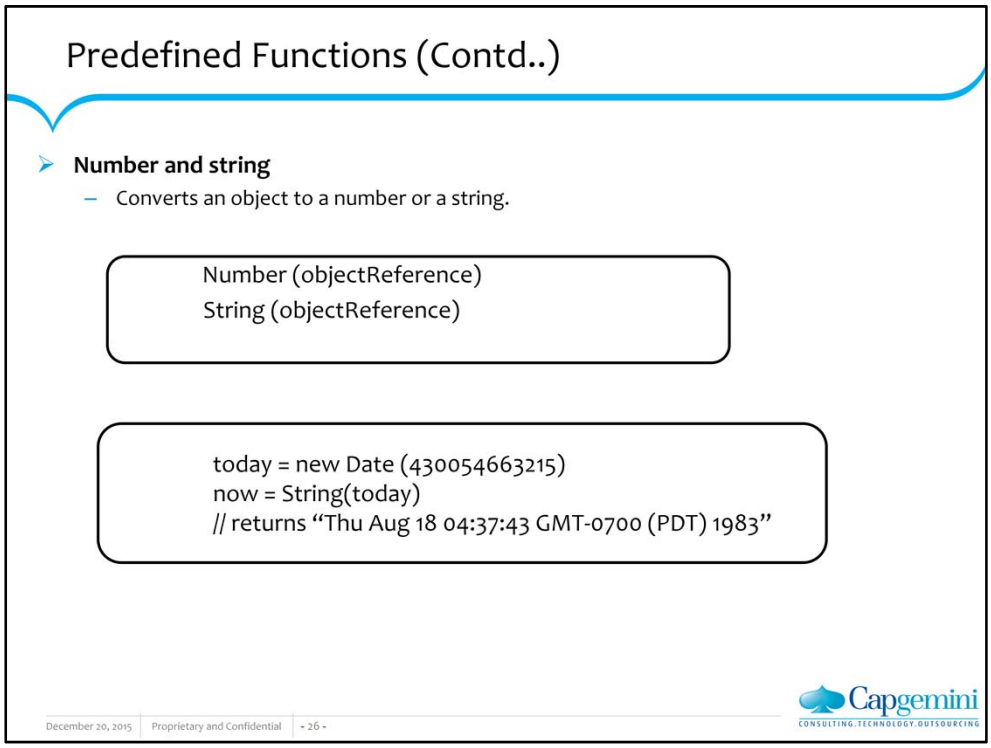
where parseFloat parses its argument, the string str, and attempts to return a floating-point number. If it encounters a character other than a sign (+ or -), a numeral (0-9), a decimal point, or an exponent, then it returns the value up to that point and ignores that character and all succeeding characters. If the first character cannot be converted to a number, it returns "NaN" (not a number).

The syntax of parseInt is:

parseInt(str [, radix])

where parseInt parses its first argument, the string str, and attempts to return an integer of the specified radix (base), indicated by the second, optional argument, radix. For example, a radix of ten indicates to convert to a decimal number, eight octal, sixteen hexadecimal, and so on. For radixes above ten, the letters of the alphabet indicate numerals greater than nine. For example, for hexadecimal numbers (base 16), A through F are used.

If parseInt encounters a character that is not a numeral in the specified radix, it ignores it and all succeeding characters and returns the integer value parsed up to that point. If the first character cannot be converted to a number in the specified radix, it returns "NaN." The parseInt function truncates the string to integer values.



Number and String Functions:

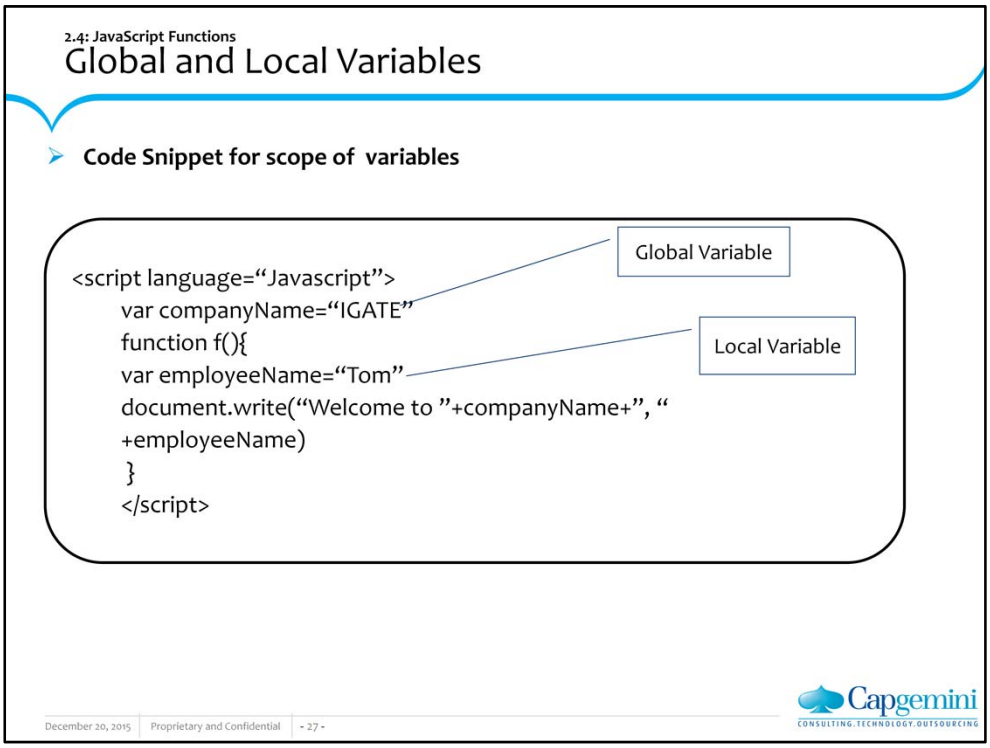
The Number and String functions let you convert an object to a number or a string. The syntax of these functions is:

Number(*objRef*)

String(*objRef*)

where objRef is an object reference. The following example converts the Date object to a readable string.

```
D = new Date (430054663215)
// The following returns
// "Thu Aug 18 04:37:43 GMT-0700 (Pacific Daylight Time) 1983"
x = String(D)
```



Scope of Variables:

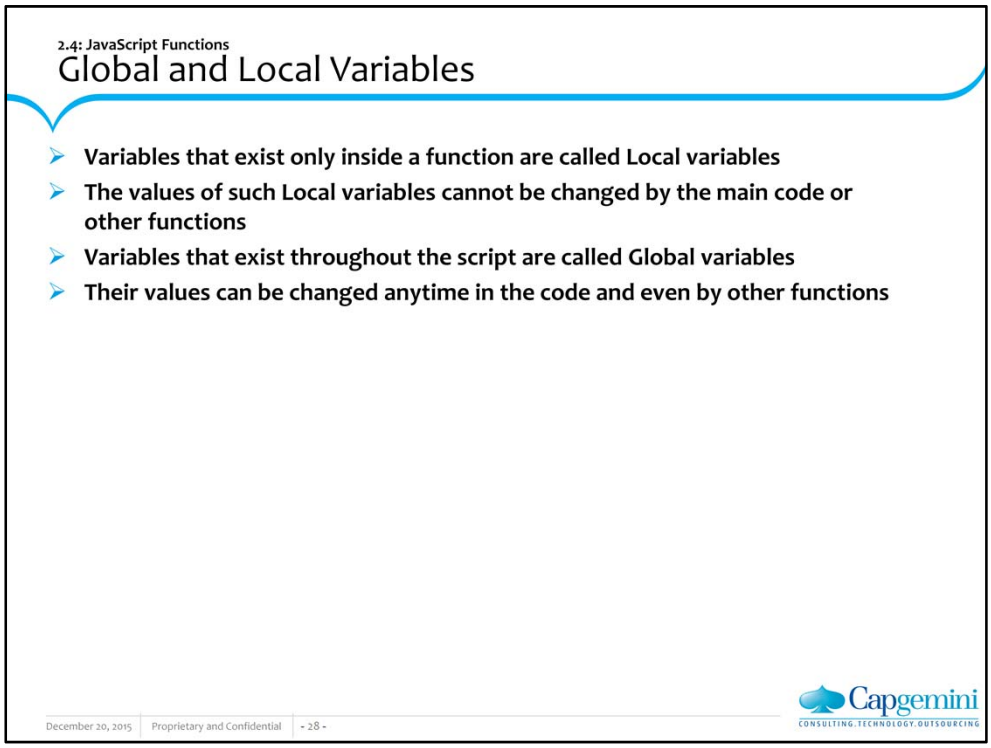
JavaScript supports two variable scopes:

Global variables
Local variables

The local variable applies only within a function and limits the scope of the variable to that function. To declare a local variable, the variable name must be preceded by var, as shown following:

```
var MaxValue=0;
```

Any variable declaration that is not within a function, is treated as a global variable. The syntax to declare a global variable is the same as that for local variable.



Using Global and Local Variables:

You can choose between local and Global variables by using the following guidelines :

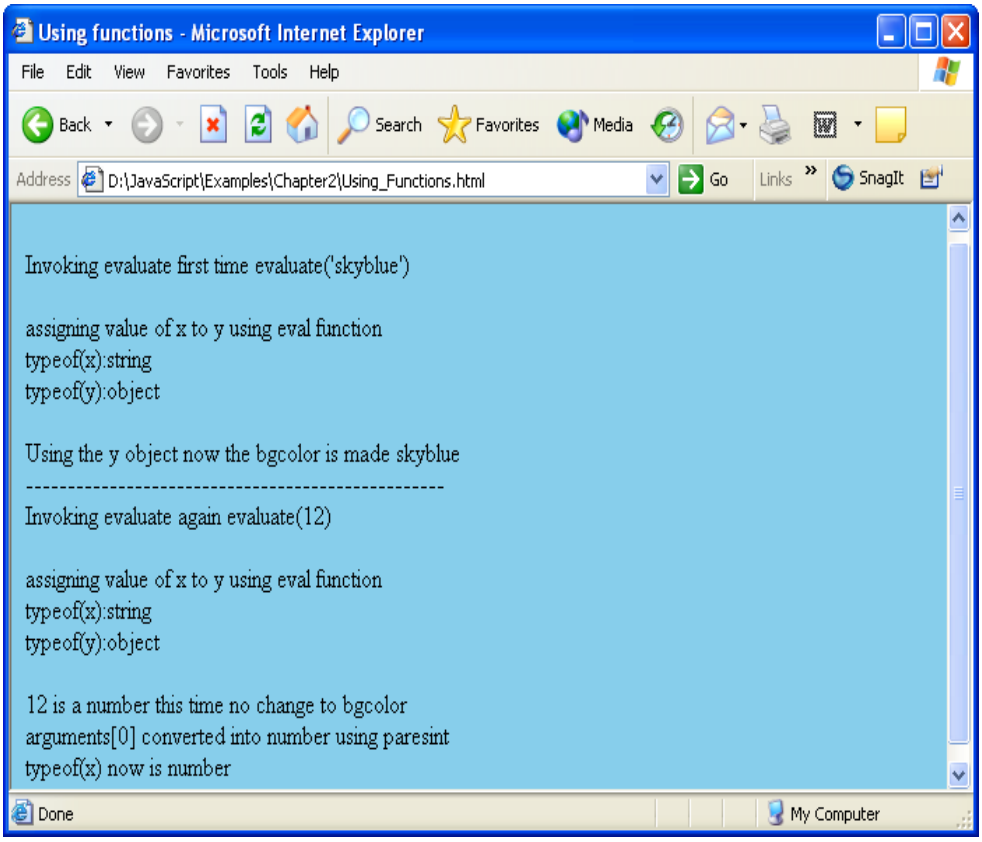
If the value of a variable is meant to be used by any part of the program, both inside and outside functions, the variable should be declared outside any function. This has the effect of making it global and modifiable by any part of the program. The best place to declare global variables is in the <head> block of the HTML document.

If the variable is needed only within a particular function, the variable should be declared inside that function.

If you want the value of a variable to be modified only by the main script of a single function, but you need to use it in another function, pass the variable as an argument to that function. This has the effect of making a copy of the variable and assigning its value to the argument. As the function works and modifies its own copy of the variable, it will not effect the original. Argument variables are automatically declared as local to that function. Even if the argument has the same name as the variable being passed, making changes to it does not effect the variable that was passed. The only exception to this is objects. When an object is passed as an argument, it is passed by reference as opposed to being passed by value. Instead of making a copy of the object, the function uses the original object. Changes made to an object's properties within the function have an effect on the original object.

```
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
<TITLE>Using functions</TITLE>
<script language="Javascript">
<!--
  function evaluate()
   {
     var x;
      var y;
    x = "document“
    document.write("<br>assigning value of x to y using eval function");
    y = eval(x);
    document.write("<br>typeof(x):");
    document.write(typeof(x)+"<br>");
    document.write("typeof(y):");
    document.write(typeof(y)+"<br>");
     if(isNaN(arguments[0]))
    {
        document.write("<br>Using the y object now the bgcolor is made
                                    "+arguments[0] );
         y.bgColor=arguments[0];
    }
    else
    {
      document.write("<br>"+arguments[0]+" is a number this time no
                                change to bgcolor");
      x=parseInt(arguments[0]);
      document.write("<br> arguments[0] converted into number using
                               parseInt ");
      document.write("<br>typeof(x) now is ");
      document.write(typeof(x)+"<br>");
    }
  }
-->
</script></HEAD><BODY><script>
document.write("<br>Invoking evaluate first time evaluate('skyblue')<br>");
evaluate("skyblue");
document.write("<BR>");
for(var i=0;i<50;i++)
document.write("-");
document.write("<br>Invoking evaluate again evaluate(12)<br>");
evaluate(12);
</script>
</BODY>
</HTML>
```

Example 2.5 Demo of creating functions and predefined function (Using_Functions.html)
And it produces the output as:



Demo

- If_ex.html
- Switch_ex.html
- Break_con_ex.html
- Fun_ex.html
- Num_string_fun.html

Lab

- The JavaScript language

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Summary

- **Data Types & Variables**
  - Numbers, Strings, Boolean, and Null
- **Operators & Expressions**
- **Functions**
- **Predefined Functions**
  - eval, isFinite, isNAN, parseInt & parseFloat, Number & String
- **Global and Local Functions**

Answers

1. Option 0
2. True

Review Question

- **Question 1: Which of the following two variable scopes is supported by JavaScript:**
  - Global, Local
  - Functional, Non functional
  - Static, Dynamic

- **Question 2: The eval function evaluates a string of JavaScript code without reference to a particular object.**
  - True/False

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Answers

1. 4
2. 5
3. 1
4. 2
5. 3

Review Question: Match the Following

1. Loop statements	1. isNan
2. Arithmetic operators	2. +=, -=
3. Predefined function	3. &&,||
4. Assignment operators	4. For, While, Switch
5. Logical operators	5. ++, --, %, *

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING