Chapter 07
JavaScript

Open Source SW Development
CSE22300

Javascript

What is Programming Language?

- **A programming language**
  - **A set of codes that we can use to give a computer instructions to follow.**
- **Popular and well-known programming languages**
  - **Include Java, C++, PHP, JavaScript and more.**
- **Modern programming languages share a large number of common concepts**
  - **Variables, arrays, loops, conditionals, and functions.**

Server-side Scripting

- **Server-side scripting is a web server technology**
- **Running a script directly on the web server**
  - **Generates dynamic HTML pages.**
  - **Interactive web sites that interface to databases.**
  - **The ability to highly customize the response**
  - **Written in languages such as Perl and PHP**
  - **The documents produced by server-side scripts may contain client-side scripts.**
  - **The user cannot see the script's source code**

Client-side Scripting

- **Client-side scripting are executed on client-side**
  - **By the user's web browser, instead of server-side.**
  - **Web authors write client-side scripts in languages such Client-side JavaScript or VBScript.**
  - **Web browser must understand the scripts.**
  - **The users may be able to see its source code**

JavaScript

- **Dynamic computer programming language**
  - **Lightweight and most commonly used as a part of web pages**
  - **Allows client-side script to interact with the user and make dynamic pages.**
  - **Interpreted programming language with object-oriented capabilities.**

Advantages

- **Less server interaction**
  - **Validates user input before sending the page off to the server.**
  - **Saves server traffic, which means less load on your server.**
- **Immediate feedback to the visitors**
  - **They don't have to wait for a page reload to see if they have forgotten to enter something.**
- **Increased interactivity**
  - **You can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard.**
- **Richer interfaces**
  - **You can use JavaScript to include such items as drag-and-drop components and sliders to give a Rich Interface to your site visitors.**

Disadvantages

- **Client-side JavaScript does not allow the reading or writing of files.**
  - **This has been kept for security reason.**

- **JavaScript doesn't have any multithreading or multiprocessor capabilities.**

- **However, server side script by Node.js**

Syntax

Script

- **JavaScript can be implemented using JavaScript statements that are placed HTML tags in a web page.**
  - **<script>... </script>**

- **Two Important Attributes**
  - **Language: This attribute specifies what scripting language**
  - **Type: This attribute indicates the scripting language**

```
<script language="javascript" type="text/javascript">
  JavaScript code
</script>
```

First Script Codes

- **document.write which writes a string into our HTML document.**

```
<html>
<body>
<script language="javascript" type="text/javascript">
   document.write ("Hello World!")
</script>
</body>
</html>
```

Whitespace and Line Breaks

- **JavaScript ignores**
  - **Spaces, tabs, and newlines that appear in JavaScript programs**

- **You can use spaces, tabs, and newlines freely**
  - **You are free to format and indent your programs**
  - **It makes the code easy to read and understand.**

Semicolons are Optional

```
var1 = 10
var2 = 20
```

```
var1 = 10; var2 = 20;
```

Case Sensitive

- **JavaScript is a case-sensitive language**
  - **This means that the language keywords, variables, function names, and any other identifiers must always be typed with a consistent capitalization of letters.**
- **So the identifiers Time and TIME will convey different meanings in JavaScript.**

Comments

- **JavaScript supports both C-style and C++-style comments**
  - **Any text between a // and the end of a line is treated as a comment**
  - **Any text between the characters /* and */ is treated as a comment. This may span multiple lines.**

- **JavaScript also recognizes the HTML comment opening sequence <!--.**
  - **JavaScript treats this as a single-line comment,**

- **The HTML comment closing sequence --> is not recognized by JavaScript**
  - **so it should be written as //-->.**

Comments

```
<script language="javascript" type="text/javascript">
<!--

// This is a comment. It is similar to comments in C++

/*
 * This is a multiline comment in JavaScript
 * It is very similar to comments in C Programming
 */
//-->
</script>
```

Variable

We focus on server side scripting with JavaScript

Variables

- **JavaScript has three primitive data types:**
  - **Numbers, e.g., 123, 120.50 etc.**
  - **Strings of text, e.g. "This text string" etc.**
  - **Boolean, e.g. true or false.**

- **JavaScript also defines two trivial data types**
  - **null and undefined**
  - **Difference?**

- **JavaScript supports a composite data type**
  - **Object**

Declare

- **Variables are declared with the var keyword as follows.**

```
var money;
var name;
```

Initialization

- **Variable initialization**
  - **Storing a value at the time of variable creation**

```
var name = "Ali";
var money;
money = 2000.50;
```

- **JavaScript is untyped language**
  - **A variable can hold a value of any data type.**

Scope

- **JavaScript variables have only two scopes.**
  - **Global Variables**
    A global variable has global scope which means it can be defined anywhere in your JavaScript code.
  - **Local Variables**
    A local variable will be visible only within a function where it is defined. Function parameters are always local to that function.

```
var myVar = "global"; // Declare a global variable
function checkscope( ) {
    var myVar = "local"; // Declare a local variable
    document.write(myVar);
}
```

Name

- **You should not use any of the reserved keywords**
  - **break or boolean variable names are not valid.**
- **Variable names should not start with a numeral (0-9)**
  - **They must begin with a letter or an underscore character**
  - **123test is an invalid variable name but _123test is a valid**
- **JavaScript variable names are case-sensitive**
  - **Name and name are two different variables.**

Reserved Keywords

abstract	else	Instanceof	switch
boolean	enum	int	synchronized
break	export	interface	this
byte	extends	long	throw
case	false	native	throws
catch	final	new	transient
char	finally	null	true
class	float	package	try
const	for	private	typeof
continue	function	protected	var
debugger	goto	public	void
default	if	return	volatile
delete	implements	short	while
do	import	static	with
double	in	super	

Operators

What is operator?

- **Let us take a simple expression 4 + 5 is equal to 9**
  - **Here 4 and 5 are called operands and '+' is called the operator**

- **JavaScript supports the following types of operators.**
  - **Arithmetic Operators**
  - **Comparison Operators**
  - **Logical (or Relational) Operators**
  - **Assignment Operators**
  - **Conditional (or ternary) Operators**

Arithmetic Operator

- **A holds 10 and B holds 20**

No	Operator and Description
1	+ (Addition) Adds two operands Ex: A + B will give 30
2	- (Subtraction) Subtracts the second operand from the first Ex: A - B will give -10
3	* (Multiplication) Multiply both operands Ex: A * B will give 200
4	/ (Division) Divide the numerator by the denominator Ex: B / A will give 2

Arithmetic Operator

- **A holds 10 and B holds 20**

No	Operator and Description
5	% (Modulus) Outputs the remainder of an integer division Ex: B % A will give 0
6	++ (Increment) Increases an integer value by one Ex: A++ will give 11
7	-- (Decrement) Decreases an integer value by one Ex: A-- will give 9

Comparison Operators

- **A holds 10 and B holds 20**

No	Operator and Description
1	== (Equal) Checks if the value of two operands are equal or not, if yes, then the condition becomes true. Ex: (A == B) is not true.
2	!= (Not Equal) Checks if the value of two operands are equal or not, if the values are not equal, then the condition becomes true. Ex: (A != B) is true.
3	> (Greater than) Checks if the value of the left operand is greater than the value of the right operand, if yes, then the condition becomes true. Ex: (A > B) is not true.

Comparison Operators

- **A holds 10 and B holds 20**

No	Operator and Description
4	< (Less than) Checks if the value of the left operand is less than the value of the right operand, if yes, then the condition becomes true. Ex: (A < B) is true.
5	>= (Greater than or Equal to) Checks if the value of the left operand is greater than or equal to the value of the right operand, if yes, then the condition becomes true. Ex: (A >= B) is not true.
6	<= (Less than or Equal to) Checks if the value of the left operand is less than or equal to the value of the right operand, if yes, then the condition becomes true. Ex: (A <= B) is true.

Logical Operators

- **A holds 10 and B holds 20**

No	Operator and Description
1	&& (Logical AND) If both the operands are non-zero, then the condition becomes true. Ex: (A && B) is true.
2	|| (Logical OR) If any of the two operands are non-zero, then the condition becomes true. Ex: (A || B) is true.
3	! (Logical NOT) Reverses the logical state of its operand. If a condition is true, then the Logical NOT operator will make it false. Ex: ! (A && B) is false.

Bitwise Operators

- **A holds 2 and B holds 3**

No	Operator and Description
1	& (Bitwise AND) It performs a Boolean AND operation on each bit of its integer arguments. Ex: (A & B) is 2.
2	| (BitWise OR) It performs a Boolean OR operation on each bit of its integer arguments. Ex: (A | B) is 3.
3	^ (Bitwise XOR) It performs a Boolean exclusive OR operation on each bit of its integer arguments. Exclusive OR means that either operand one is true or operand two is true, but not both. Ex: (A ^ B) is 1.

Bitwise Operators

- **A holds 2 and B holds 3**

No	Operator and Description
4	~ (Bitwise Not) It is a unary operator and operates by reversing all the bits in the operand. Ex: (~B) is -4.
5	<< (Left Shift) It moves all the bits in its first operand to the left by the number of places specified in the second operand. New bits are filled with zeros. Shifting a value left by one position is equivalent to multiplying it by 2, shifting two positions is equivalent to multiplying by 4, and so on. Ex: (A << 1) is 4.
6	>> (Right Shift) Binary Right Shift Operator. The left operand's value is moved right by the number of bits specified by the right operand. Ex: (A >> 1) is 1.

Bitwise Operators

- **A holds 2 and B holds 3**

No	Operator and Description
7	>>> (Right shift with Zero) This operator is just like the >> operator, except that the bits shifted in on the left are always zero. Ex: (A >>> 1) is 1.

Assignment Operators

No	Operator and Description
1	= (Simple Assignment) Assigns values from the right side operand to the left side operand Ex: C = A + B will assign the value of A + B into C
2	+= (Add and Assignment) It adds the right operand to the left operand and assigns the result to the left operand. Ex: C += A is equivalent to C = C + A
3	-= (Subtract and Assignment) It subtracts the right operand from the left operand and assigns the result to the left operand. Ex: C -= A is equivalent to C = C - A

Assignment Operators

No	Operator and Description
4	*= (Multiply and Assignment) It multiplies the right operand with the left operand and assigns the result to the left operand. Ex: C *= A is equivalent to C = C * A
5	/= (Divide and Assignment) It divides the left operand with the right operand and assigns the result to the left operand. Ex: C /= A is equivalent to C = C / A
6	%= (Modules and Assignment) It takes modulus using two operands and assigns the result to the left operand. Ex: C %= A is equivalent to C = C % A

Conditional Operator

No	Operator and Description
1	? : (Conditional) If Condition is true? Then value X : Otherwise value Y

typeof Operator

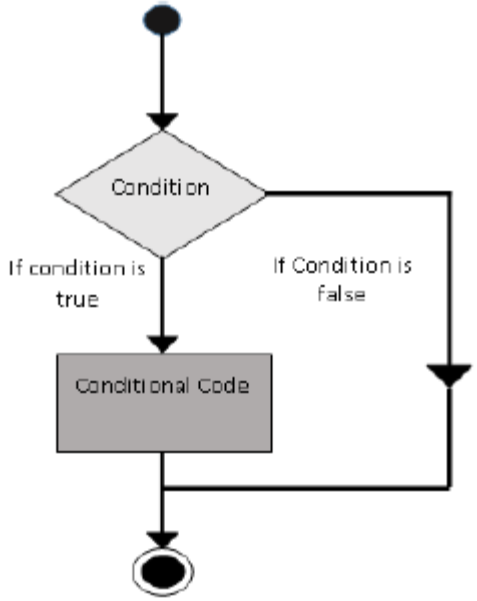
- **The typeof operator is a unary operator**
  - **Its value is a string indicating the data type of the operand**
  - **The typeof operator evaluates to "number", "string", or "boolean**

Type	String Returned by typeof
Number	"number"
String	"string"
Boolean	"boolean"
Object	"object"
Function	"function"
Undefined	"undefined"
Null	"object"

Conditional Statement

If else

- **if statement**
- **if ... else statement**
- **if ... else if ... statement**



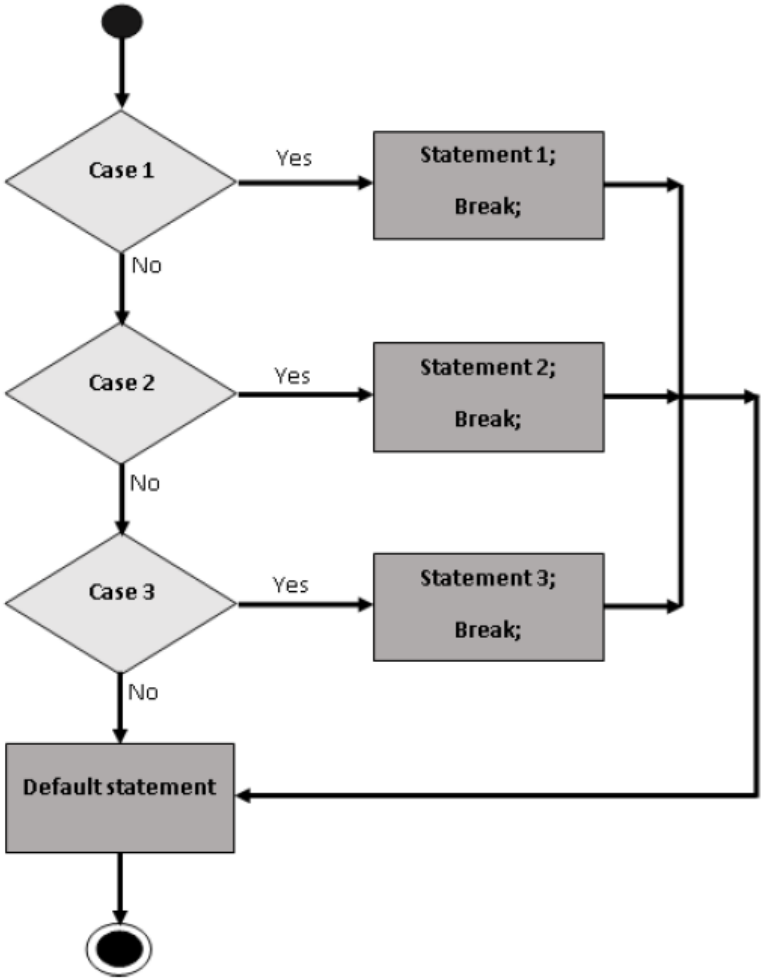
If else

- **The ‘if’ statement is the fundamental control statement that allows JavaScript to make decisions and execute statements conditionally.**

```
if (expression 1){
   Statement(s) to be executed if expression 1 is true
}else if (expression 2){
   Statement(s) to be executed if expression 2 is true
}else if (expression 3){
   Statement(s) to be executed if expression 3 is true
}else{
   Statement(s) to be executed if no expression is true
}
```

Switch

- **Efficient way for multiway branch**



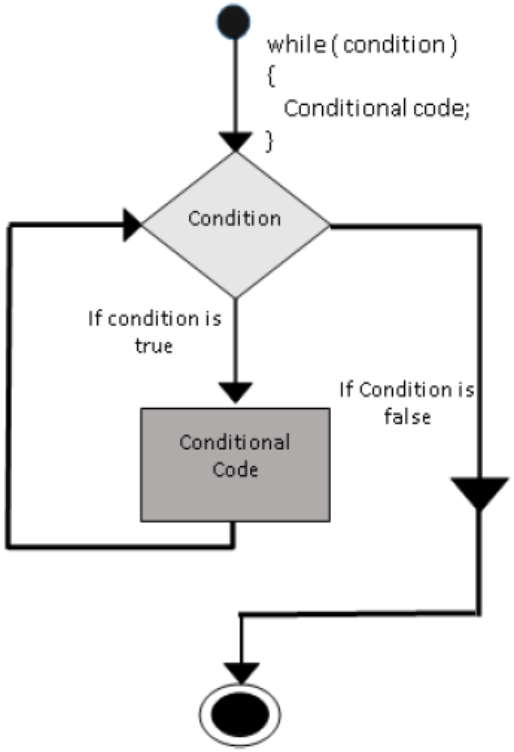
Switch

- **The objective of a switch statement is to give an expression to evaluate and several different statements to execute based on the value of the expression.**

```
switch (expression)
{
  case condition 1: statement(s)
                    break;
  case condition 2: statement(s)
                    break;
   ...
  case condition n: statement(s)
                    break;
  default: statement(s)
}
```

While Loop

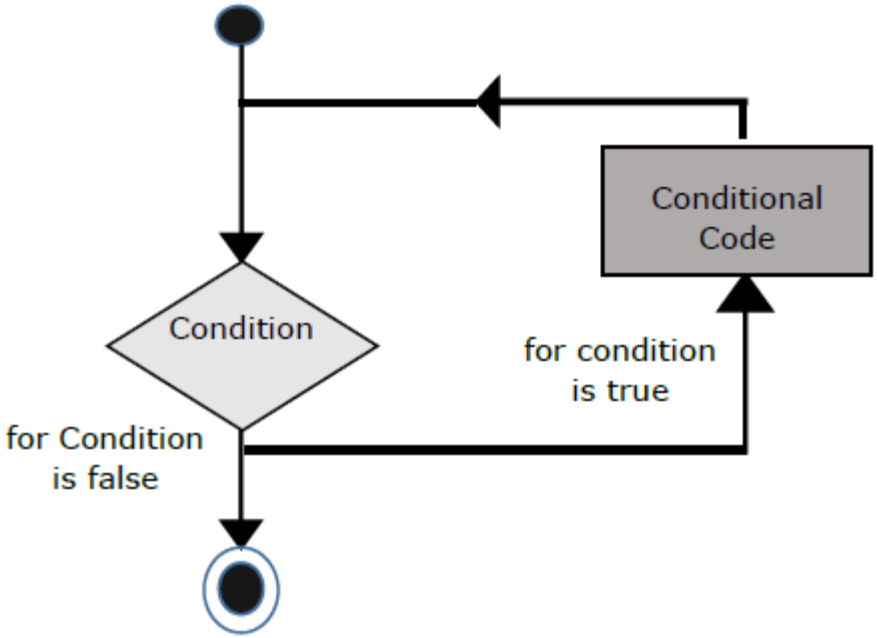
- **Execute a statement or code block repeatedly**




```
while (expression){
   Statement(s) to be executed if expression is true
}
```

For loop

- **Loop initialization**
- **Test statement**
- **Iteration statement**




```
for (initialization; test condition; iteration statement){
     Statement(s) to be executed if test condition is true
}
```

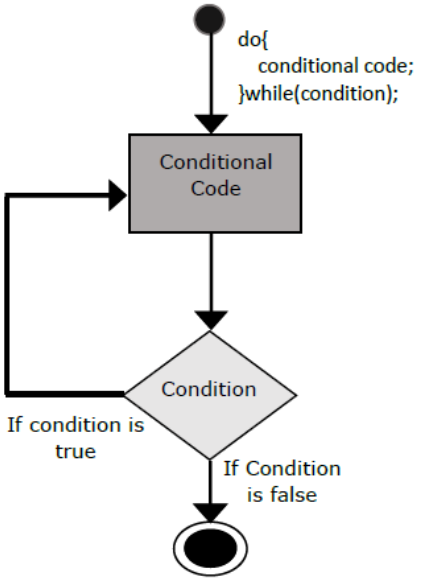
For-In loop

- **Loop through an object's properties.**

```
for (variablename in object){
  statement or block to execute
}
```

Do While Loop

- **Loop will always be executed at least once, even if the condition is false**




```
do{
   Statement(s) to be executed;
} while (expression);
```

Loop Control

- **Break**
  - **Breaks out of loop completely**
- **Continue**
  - **Start immediately the next iteration of the loop and skip the remaining code block**

Function

Function

- **Reusable code which can be called anywhere**
  - **Eliminates the need of writing the same code again and again**
  - **Divides a big program into a number of small and manageable functions**

- **Definition**
  - **function keyword**
  - **list of parameter**
  - **statement**

```
function functionname(parameter-list)
{
  statements
}
```

```
function sayHello()
{
   alert("Hello there");
}
```

Calling a Function

- **To invoke a function**
  - **Write the name of that function**

```
function sayHello()
{
   console.log ("Hello there!");
}

sayHello();
```

Parameters

- **Multiple parameters separated by comma**

```
function sayHello(name, age)
{
   console.log (name + " is " + age + " years old.");
}
sayHello('Zara', 7);
```

Return Statement

- **Return a value from a function**

```
function concatenate(first, last)
{
   var full;
   full = first + last;
    return full;
}
function secondFunction()
{
   var result;
   result = concatenate('Zara', 'Ali');
   console.log (result );
}
secondFunction();
```

Function() Constructor

- **Define your function dynamically using Function() constructor along with the new operator**
  - **anonymous functions**

```
var variablename = new Function(Arg1, Arg2..., "Function Body");
```

```
var func = new Function("x", "y", "return x*y;");
function secondFunction() {
    var result;
    result = func(10,20);
    console.log(result);
}
secondFunction();
```

Function() Literals

- **Another new way of defining functions.**
  - **Unnamed function**

```
var variablename = function(Argument List){
   Function Body
};
```

```
var func = function(x,y){ return x*y };
function secondFunction() {
   var result;
   result = func(10,20);
   console.log ( result );
}
secondFunction();
```

Objects

Objects

- **JavaScript is an OOP language**
- **Objects are composed of properties and methods**

- **Object Properties**
  - **Can be any of the three primitive data types, or any of the abstract data types, such as another object**
  - **Are used internally in the object's methods, but can also be globally visible variables**

- **Object Methods**
  - **Functions that let the object do something**

User-Defined Objects

- **new operator**
  - **create an instance of an object**

```
var employee = new Object();
var books = new Array("C++", "Perl", "Java");
var day = new Date("August 15, 1947");
```

- **Constructor**
  - **Creates and initializes an object**
  - **JavaScript provides a special constructor function**
    - **Object()**

```
var book = new Object();
book.subject = "Perl";
book.author = "Mohtashim";
console.log("Book name is : " + book.subject + "<br>");
console.log("Book author is : " + book.author + "<br>");
```

User-Defined Objects

- **User-Defined Constructor**
  - **create an instance of an object**

```
function book(title, author) {
    this.title = title;
    this.author = author;
}
var myBook = new book("Perl", "Mohtashim");
console.log("Book title is : " + myBook.title);
console.log("Book author is : " + myBook.author);
```

User-Defined Objects

- **Defining Methods**
  - **create an instance of an object**

```
function addPrice(amount) {
   this.price = amount;
}
function book(title, author) {
   this.title = title;
   this.author = author;
   this.addPrice = addPrice;
}
var myBook = new book("Perl", "Mohtashim");
myBook.addPrice(100);
console.log("Book title is : " + myBook.title);
console.log("Book author is : " + myBook.author);
console.log("Book price is : " + myBook.price);
```

Number

Number

- **Number Object**
  - **Represents numerical date, either integers or floating-point numbers**
- **Syntax**

```
var val = new Number(number);
```

Number

- **Property**

Property	Description
MAX_VALUE	The largest possible value a number
MIN_VALUE	The smallest possible value a number
NaN	Equal to a value that is not a number
NEGATIVE_INFINITY	A value that is less than MIN_VALUE
POSITIVE_INFINITY	A value that is greater than MAX_VALUE
prototype	The prototype property allows you to add properties and methods
constructor	It returns a reference to the Number function that created the instance's prototype

Number

- **Methods**

Method	Description
toExponential()	Represents a number in exponential notation.
toFixed()	Represents a number in fixed-point notation.
toLocaleString()	Converts a number to a string by using the current or specified locale.
toPrecision()	Represents a number either in exponential or fixed-point notation with a specified number of digits.
toString()	Returns the string representation of the number's value.
valueOf()	Returns the number's value.

Boolean

Boolean

- **Represents two values, either "true" or "false".**
- **Value parameter is omitted or is 0, -0, null, false, NaN, undefined, or the empty string (""),**
  - **False**

```
var val = new Boolean(value)
```

Boolean

- **Property**

Property	Description
prototype	The prototype property allows you to add properties and methods
constructor	It returns a reference to the Number function that created the instance's prototype

String

String

- **String Object**
  - **JavaScript's string primitive data type with a number of helper methods**
- **Syntax**

```
var val = new String(string);
```

String

- **Property**

Property	Description
length	Returns the length of the string
prototype	The prototype property allows you to add properties and methods
constructor	It returns a reference to the Number function that created the instance's prototype

- **Length**

```
var str = new String( "This is string" );
console.log("str.length is:" + str.length);
```

String

- **Methods**

Method	Description
charAt()	Returns the character at the specified index.
charCodeAt()	Returns a number indicating the Unicode value of the character at the given index
concat()	Combines the text of two strings and returns a new string
indexOf()	Returns the index within the calling String object of the first occurrence of the specified value, or -1 if not found
lastIndexOf()	Returns the index within the calling String object of the last occurrence of the specified value, or -1 if not found
localeCompare()	Returns a number indicating whether a reference string comes before or after or is the same as the given string in sorted order

String

- **Methods**

Method	Description
match()	Used to match a regular expression against a string
replace()	Used to find a match between a regular expression and a string, and to replace the matched substring with a new substring
search()	Executes the search for a match between a regular expression and a specified string
slice()	Extracts a section of a string and returns a new string
split()	Splits a String object into an array of strings by separating the string into substrings.
substr()	Returns the characters in a string beginning at the specified location through the specified number of characters.

String

- **Methods**

Method	Description
substring()	Returns the characters in a string between two indexes into the string
toLocaleLowerCase()	The characters within a string are converted to lower case while respecting the current locale.
toLocaleUpperCase()	The characters within a string are converted to upper case while respecting the current locale
toLowerCase()	Returns the calling string value converted to lower case
toString()	Returns a string representing the specified object.
toUpperCase()	Returns the calling string value converted to uppercase
valueOf()	Returns the primitive value of the specified object

String

- **charAt()**
  - **Returns the character from the specified index**

```
string.charAt(index)
```

```
var str = new String( "This is string" );
console.log("str.charAt(0) is:" + str.charAt(0));
```

- **Concat()**
  - **Returns a single concatenated string**

```
string.concat(string2, string3[, ..., stringN]);
```

```
var str1 = new String( "This is string one" );
var str2 = new String( "This is string two" );
var str3 = str1.concat( str2 );
console.log("Concatenated String :" + str3);
```

String

- **indexOf()**
  - **Returns the index of the found occurrence, otherwise -1 if not found**

```
string.indexOf(searchValue[, fromIndex])
```

```
var str1 = new String( "This is string one" );
var index = str1.indexOf( "string" );
console.log("indexOf found String :" + index );
```

- **lastIndexOf()**
  - **Returns the index of the last found occurrence, otherwise -1 if not found**

```
string.lastIndexOf(searchValue[, fromIndex])
```

```
var str1 = new String( "This is string one and again string" );
var index = str1.lastIndexOf( "string" );
console.log("lastIndexOf found String :" + index );
```

String

- **match()**
  - **Retrieve the matches when matching a string against a regular expression**

```
string.match ( param )
```

```
var str = "For more information, see Chapter 3.4.5.1";
var re = /Chapter/;
var found = str.match( re );
console.log(found );
```

- **replace()**
  - **Replaces the matched substring with a new substring.**

```
string.replace(regexp/substr, newSubStr/function[, flags]);
```

```
var str = "For more information, see Chapter 3.4.5.1";
var re = /Chapter/;
var found = str.replace( re , “aaa”);
console.log(found );
```

String

- **slice()**
  - **Extracts a section of a string and returns a new string**

```
string.slice( beginslice [, endSlice] );
```

```
var str = "Apples are round, and apples are juicy.";
var sliced = str.slice(3, -2);
console.log( sliced );
```

- **split()**
  - **Splits a String object into an array of strings by separating the string into substrings.**

```
string.split([separator][, limit]);
```

```
var str = "Apples are round, and apples are juicy.";
var splitted = str.split(" ", 3);
console.log ( splitted );
```

String

- **substr()**
  - **returns the characters in a string beginning at the specified location through the specified number of characters**

```
string.substr(start[, length]);
```

```
var str = "Apples are round, and apples are juicy.";
console.log ("(1,2): " + str.substr(1,2));
console.log ("(-2,2): " + str.substr(-2,2));
console.log ("(1): " + str.substr(1));
console.log ("(-20, 2): " + str.substr(-20,2));
consloe.log("(20, 2): " + str.substr(20,2));
```

String

- **toLocaleLowerCase()**
  - **Returns a string in lowercase with the current locale.**

```
string.toLocaleLowerCase( )
```

```
var str = "Apples are round, and Apples are Juicy.";
console.log(str.toLocaleLowerCase( ));
```

Array

Array

- **Array Object**
  - **Lets you store multiple values in a single variable**

```
var fruits = new Array( "apple", "orange", "mango" );
var fruits = [ "apple", "orange", "mango" ];
```

Array

- **Property**

Property	Description
length	Reflects the number of elements in an array
input	This property is only present in arrays created by regular expression matches
index	The property represents the zero-based index of the match in the string
prototype	The prototype property allows you to add properties and methods
constructor	Returns a reference to the array function that created the object.

Array

- **Method**

Method	Description
concat()	Returns a new array comprised of this array joined with other array(s) and/or value(s).
every()	Returns true if every element in this array satisfies the provided testing function.
filter()	Creates a new array with all of the elements of this array for which the provided filtering function returns true.
forEach()	Calls a function for each element in the array.
indexOf()	Returns the first (least) index of an element within the array equal to the specified value, or - 1 if none is found.
join()	Joins all elements of an array into a string.

Array

- **Method**

Method	Description
lastIndexOf()	Returns the last (greatest) index of an element within the array equal to the specified value, or - 1 if none is found.
map()	Creates a new array with the results of calling a provided function on every element in this array.
pop()	Removes the last element from an array and returns that element.
push()	Adds one or more elements to the end of an array and returns the new length of the array.
reduce()	Apply a function simultaneously against two values of the array (from left-to-right) as to reduce it to a single value.
reduceRight()	Apply a function simultaneously against two values of the array (from right-to-left) as to reduce it to a single value.

Array

- **Method**

Method	Description
reverse()	Reverses the order of the elements of an array -- the first becomes the last, and the last becomes the first.
shift()	Removes the first element from an array and returns that element.
slice()	Extracts a section of an array and returns a new array.
some()	Returns true if at least one element in this array satisfies the provided testing function.
toSource()	Represents the source code of an object
sort()	Sorts the elements of an array.

Array

- **Method**

Method	Description
splice()	Adds and/or removes elements from an array.
toString()	Returns a string representing the array and its elements.
unshift()	Adds one or more elements to the front of an array and returns the new length of the array.

Array

- **concat()**
  - **Returns a new array comprised of this array joined with two or more arrays.**

```
array.concat(value1, value2, ..., valueN);
```

```
var alpha = ["a", "b", "c"];
var numeric = [1, 2, 3];
var alphaNumeric = alpha.concat(numeric);
console.log("alphaNumeric : " + alphaNumeric );
```

- **IndexOf()**
  - **Returns the first index at which a given element can be found in the array**

```
array.indexOf(searchElement[, fromIndex]);
```

Array

- **pop()**
  - **Removes the last element from an array and returns that element.**

```
Array.pop();
```

```
var numbers = [1, 4, 9];
var element = numbers.pop();
console.log("element is : " + element );
```

- **push()**
  - **Appends the given element(s) in the last of the array and returns the length of the new array**

```
Array.push();
```

```
var numbers = new Array(1, 4, 9);
var length = numbers.push(10);
console.log("new numbers is : " + numbers );
```

Array

- **reverse()**
  - **Reverses the element of an array**

```
array.reverse();
```

```
var arr = [0, 1, 2, 3].reverse();
console.log ("Reversed array is : " + arr );
```

- **shift()**
  - **Removes the first element from an array and returns that element**

```
array.shift();
```

```
var element = [105, 1, 2, 3].shift();
console.log ("Removed element is : " + element );
```

Array

- **unshift()**
  - **Adds one or more elements to the beginning of an array and returns the new length of the array**

```
array.unshift( element1, ..., elementN );
```

```
var arr = new Array("orange", "mango", "banana", "sugar");
var length = arr.unshift("water");
console.log("Returned array is : " + arr );
console.log("Length of the array is : " + length );
```