JAVASCRIPT

Follow the book 'HTML/DHTML/Javascript/Perl CGI' by Ivan Bayross along with this handout.

JAVASCRIPT

Introduction

- JavaScript is a compact, object-based scripting language for developing client Internet applications.
- JavaScript was designed to add interactivity to HTML pages.
- JavaScript is a scripting language - a scripting language is a lightweight programming language.
- A JavaScript is lines of executable computer code.
- A JavaScript is usually embedded directly in HTML pages
- JavaScript is an interpreted language (means that scripts execute without preliminary compilation).
- Everyone can use JavaScript without purchasing a license.
- All major browsers, like Netscape and Internet Explorer, support JavaScript.
- Javascript was developed by Netscape as *Live Script* - changed to *JavaScript* when endorsed by Sun 1993, version 1.0 released with Netscape 2.0.
- JavaScript is a powerful scripting language that is also capable of performing extensive form data collecting and form processing functions.

Comparing Javascript with Java

- *While comparing javascript with java first lets see the basic differences between them*

Javascript	Java
1. Javascript is a small, lightweight programming language.	1. Java is a full-blown powerful, sophisticated programming language.
2. Developed by Netscape communications.	2. Developed by Sun Microsystems.
3. Scripting language interpreted at runtime.	3. True programming compiled to byte-code.
4. Object-based, has limited number of built-in objects.	4. Object-oriented, can create their own classes.
5. Not fully extensible.	5. Fully extensible.
6. Code integrated with, and embedded in HTML.	6. Applies distinct from HTML (accessed from HTML pages).
7. Variables type must not be declared.	7. Variables type must be declared.
8. Javascript source code can be viewed by everybody using “view source command”.	8. Your Java source is hidden because it's only the compiled byte-code, which the browser uses, but this is not a *guarantee* of security.

- *Now lets see the similarities between java and javascript:*
    1. Both can be used for enhancing the capabilities of the web pages.
    2. Both can run on the client machine - i.e. the machine where you have your browser, not the server where the page came from.
    3. Both have some level of security built in to guard against malicious use. No computer system is ever 100% secure unless it's isolated in a locked room surrounded by armed guards, but the developers of Java and JavaScript have taken some care in their security.

Netscape & Java Script:

Java script is scripting language created by Netscape hence JavaScript works best with the Netscape suite of client and server products. The Netscape client 'browser' is called Netscape communicator. The default scripting language that Netscape communicator understands is java Script. Netscape server product is called Netscape commerce server. The default scripting language that Netscape commerce server understands is JavaScript.

Database Connectivity:

Netscape has a product called 'Live wire', which permits server side, JavaScript code, to connect to Relational Database management systems (RDBMS) like oracle, My SQL server, My SQL etc. 'Live wire' database drivers also support a no of non-relational database.

JavaScript Uses

- The most frequent uses of javascript are:
    - For displaying the clock
    - Used for creating Drop-down menus.
    - Showing Alert messages.
    - Displaying Popup windows.
    - Validating HTML Form Data.

Advantages of JavaScript

- Cross Browser supporting:
    - This means that the javascript codes are supported by various browsers like Internet Explorer, Netscape Navigator, and Mozilla etc.

- Platform Independent:
  - Javascript codes can be executed in any platform like Linux, Microsoft Windows and many other operating systems.
- Lightweight for fast downloading:
  - The javascript codes runs directly in the client machine. The code should be downloaded all the way from server to the client and this time duration is very minimum and the executing the codes of javascript is also very fast.
- Validating Data:
  - The data can be validated in the two different way:
    - Validating in server side or in server machine.
    - Validating in client side or in client machine.
  - In this two different types of validation of data the second one is much more faster, and this is done through javascript.
- Sophisticated User Interfaces:
  - By using javascript you can create a user interactive interfaces that can communicate with the user and the Browser.
- In-Built software:
  - To you don't need any extra tools to write JavaScript, any plain text or HTML editor will do, so there's no expensive development software to buy.
- Easy to Learn:
  - The javascript programmer should know the minimal syntax of javascript since it supports many syntax and data types as C and C++.
  - It's also an easy language to learn, and there's a thriving and supportive online community of JavaScript developers and information resources.
- Designed for programming User-Events:
  - Javascript supports Object/Event based programming. So the code written in javascript can easily be break down into sub-modules.

Disadvantages of JavaScript:

- Launching an application on the client computer.

  - Javascript is not used to create stand-alone application; it is only used to add some functionality in any web page.
- Reading or writing files:
  - Javascript cannot read and write files into the client machines. It can only be used as a utility language to develop any web site.
- Retrieving the text contents of HTML pages or files from the server.
- Reading and Writing files to the server:
  - Javascript can read and write to any file in the server as well.
- Sending secret e-mails from Web site visitors to you:
  - Javascript cannot be used to send email to the visitors or user of the web site. This can be done only with the server side scripting.

- Cannot create Database Application:
  - By using javascript you cannot connect the web site to the database. For this you need to use server-side scripting.
- Browser Compatibility Issues:
  - Not all browsers support the javascript codes. The browser may support javascript as a whole, but may not support the codes or lines of codes written in javascript and may interpret differently.
- Javascript does not implement multiprocessing or multithreading.
- Use printers or other devices on the user's system or the client-side LAN.
- Javascript has limitations of writing in a client machine. It can only write the cookie in client machine that is also of a certain size i.e. 4K.

JavaScript Myths

JavaScript is a new technology that is rapidly changing. It is not yet well understood and is the subject of a fair bit of misinformation and confusion.

1. **JavaScript Is Not Java Simplified**

One of the most common misconceptions about JavaScript is that it is a "simplified version" of Java, the programming language from Sun Microsystems. Other than an incomplete syntactic

resemblance and the fact that both Java and JavaScript can deliver "executable content" over networks, the two languages are entirely unrelated. The similarity of names is purely a marketing ploy (the language was originally called LiveScript, and its name was changed to JavaScript at the last minute).

JavaScript and Java do, however, make a good team. The two languages have disjoint sets of capabilities. JavaScript can control browser behavior and content but cannot draw graphics or perform networking. Java has no control over the browser as a whole, but can do graphics, networking, and multithreading.

2. **JavaScript Is Not Simple**

JavaScript is touted as a "scripting language" instead of a "programming language," the implication being that scripting languages are simpler, that they are programming languages for nonprogrammers. Indeed, JavaScript appears at first glance to be a fairly simple language, perhaps of the same complexity as BASIC. Further experience with JavaScript, however, reveals complexities that are not readily apparent. For example, the use of objects as arguments to functions requires a careful understanding of the difference between passing arguments "by value" and passing arguments "by reference." There are also a number of tricky details to understand about converting data values from one type to another in JavaScript.

This is not to say that JavaScript is beyond the reach of nonprogrammers. It *will* be useful to nonprogrammers, but only for limited, cookbook-style tasks. For better or worse, complete mastery of JavaScript requires sophisticated programming skills.

3. **JavaScript is just a clowny version of Java**

JavaScript is actually totally independent of Java. The only thing it shares is slightly similar syntax. The reason why it has Java in its name is due to marketing reasons. "The language's name is the result of a co-marketing deal between Netscape and Sun, in exchange for Netscape bundling Sun's Java runtime with their then-dominant browser." JavaScript is a brand new language that shares much more with functional languages like Lisp or even Scheme than with Java.

4. **JavaScript is loosely defined and unpredictable**

JavaScript is very specifically defined. Google's V8 Engine adheres to ECMA-262, which has no room for misinterpretation. V8 and other JavaScript engines have literally thousands of test cases to verify that they stick *exactly* to the specification. When people say that JavaScript has a lot of inconsistency, they are probably confusing JavaScript with DOM APIs* which are notoriously changeable between browsers -- namely older versions of Internet Explorer.

*DOM APIs are basically an API for JavaScript to interact with the browser, for example, adding new text to a webpage, or calling a function when the user clicks on a button.

5. **JavaScript is very slow**

JavaScript is actually quite fast. It is much optimized and keeps getting more and more. A ton of companies have HUGE vested interest in optimizing JavaScript and JavaScript performance increases all the time. Browsers live and die by their JS execution performance so there is a lot of original research in this area. Granted, it is not as fast as a compiled language like C, but we don't need it to be.

6. **JavaScript is a crappy language and only suited for small browser tricks**

JavaScript is actually an awesome dynamic language that can definitely stand tall. JavaScript features prototypal inheritance, which is hard for a lot of people, trained in classical inheritance to wrap their heads around. This often leads to claims that JavaScript is not object oriented, doesn't support complex programs, etc.

Well, actually JavaScript does support classical inheritance as well, and with a few coding patterns, people trained in languages like C++ or Java should be ok. JavaScript is a dynamic language and this lends itself to a lot of cool behavior.

Adding Javascript codes

- You have to focus on three major steps while adding javascript codes in HTML document.
  1. Use a **<script>** tag to tell the browser that you are using javascript.

     Example:

     <script language = ”Javascript”**>**
  2. Write the javascript codes.

     Example:

```
<script language = ”Javascript”>
    // javascript codes HERE
</script>
```

3. Test the scripts.

- While writing the javascript codes there can be many types of errors like:
  - Human Errors.
  - Browser compatibility issues.
  - Incorrect behavior based on the different operating systems.
- So when using javascript, be sure that you test your scripts out on a wide variety of systems and setups.
- You can add javascript codes between the **<head>** tag as well as in the **<body>** tag. And you can have multiple **<script>** tag in a single web page, but you must have the closing **</script>** tag of each opening tag.

First Example in Javascript

```
<Script language = “javascript”>
    <!- -
        document.write (“Hello World”);
    // - - >
</script>
```

- The first line <script language = “javascript”> tells the browser that the code written between the <script> and </script> tags are the javascript codes.
- Some browsers doesn’t support scripts, and may display the scripts as the HTML content in the web-page.
- To prevent this you must write the javascript codes in between <! - - and // - ->.
- The two forward slashes at the end of the line are the javascript comments.
- You cannot put this slashes to the starting of the comment line like (// <!- -), because the older browser will display it.
- Now the above code will display **Hello World** in the browser.

Using External JavaScript Files

- While using external javascript files in a web-page, the javascript files must have the three main features:
  - First, the file that you are importing must be a valid javascript file.
  - Second, the file must have the extension "**.js**".
  - Lastly, you must know the location of the file.
- Here is the example of using external javascript files in a web-page.

```
<HTML>
    <HEAD>
        <Script src = "myExample.js"></script>
        <TILTE></TITLE>
    </HEAD>
        <BODY>
        </BODY>
<HTML>
```

- Here myExample.js contains the javascript codes and whatever is written in the file will be displayed in the browser.
- In this condition the myExample.js file and the above HTML file must be in the same location.
- For implementing the external javascript file you must have at least two files: one for HTML codes and other for javascript file.

Points to Ponder in Javascript

- Optional semicolon (;) to end the statement.
- Some browser doesn't support javascript codes, and display the script as the content of the web-page.
- To prevent this we have to use the HTML comments.
- To use JavaScript functions the code needs to follow the line:

  <SCRIPT language = "JavaScript"> and closed with **</SCRIPT>.**
- JavaScript uses both // and /* */ to indicate comments.
- JavaScript supports data types: numeric, string, Boolean, and null.
- JavaScript contains three user interfaces: **alert** (message), **prompt** (message, [Default]), **confirm**(message).

- JavaScript does not have built-in support for arrays, but does allow for building of arrays.
- JavaScript supports operators: +, -, *, /, %, ++, --, =, +=, -=, *=, /=, %=, &&, ||, !, &, |, ^.
- JavaScript is case sensitive.
- It's a Object-based language.

Case Sensitivity

- Javascript is a case-sensitive language.
- This means the Upper and Lower case of any variables or methods may effect.
- The variable in Uppercase is different then the variable declared in Lowercase.
- For Example:

  myVar, myvar, MyVar and MYVAR are four different types of variables.
- The same rules occur in the name of function as well as in objects of javascript.

Optional Semicolons

- The semicolon (;) identifies the end of the statement in javascript.
- But it is not always necessary to put semicolon at the end of statement.
- If you are writing two different statements in two different lines without semicolon than it is valid in javascript.
- But if you are writing the two different statements in same single line then you have to keep semicolon after the end of the first statement.
- Examples:

  var a=5

  var b=2

  The above statement are valid in javascript since javascript has optional semicolon to end the statement.

  var a=5; var b=5

  This is also valid in javascript

  var a=5;

  var b=2;

  These two statements are also valid in javascript.

Reserved Words

- JavaScript have some reserved words that cannot be used as variable, objects or a function.
- These words are used for some reserved purpose in JavaScript.
- ***The lists of reserved words are listed below:***

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

Data types

- A computer works with values. It manipulates values and produces some kind of result.
- Depending on the program, it may be limited as to what types of values it can work with.
- The types of values a program can work with are known as its *data types*.
- JavaScript supports many types of data.
- Data types can be broken into two broad categories:
  1. *Primitive Data types*
     - A *primitive data type* is a data type that stores a single value, such as numbers and strings.
  2. *Composite Data types*

- A *composite* data type, for the purposes of JavaScript, is the same thing as an *object*.
- It is a data type that can consist of multiple values grouped together in some way.
- JavaScript treats objects as *associative arrays*.
- An associative array is an array that can have its elements associated with names in addition to their numeric index position in the array.

Primitive Data types

- There are three primitive data type, javascript provides to us:
  - *Numbers*
    - Numbers are the easist of the data types to understand.
    - They represent numeric values.
    - The simplest type of number is an integer.
    - It is a number with no fractional component.
    - JavaScript can handle integer values between 2^{53} and -2^{53}. Some JavaScript functions, however, cannot handle extremely large or small numbers, so it is probably a good idea to try to keep your numbers in the range of 2^{31} and -2^{31}.
    - Another commonly used type of number is a floating-point number.
    - A *floating-point* number is a number that is assumed to have a decimal point.
    - Being a computing language, JavaScript also has the ability to handle hexadecimal (four bit) and octal (three bit) numbers.

  - *Strings*
    - A *string* is sequence of valid characters within a given character set.
    - It is normally used to represent text. A string literal is defined by enclosing it in matching single or double quotes.
    - If you create a string with nothing in it, it is called an empty string and is normally created by two quote marks with nothing between them.
    - Some characters that you may want in a string may not exist on the keyboard, or may be special characters that can't appear as themselves in a string.
    - In order to put these characters in a string, you need to use an *escape sequence* to represent the character. An escape sequence is a character or numeric value

representing a character that is preceeded by a backslash (\ ) to indicate that it is a special character.

- Some escaped characters are as follows:

Escape Sequence	Character
\0	(blackslash zero) The Null character (\u0000). If you've taken COBOL, you may recognize this as COBOL's low-value, used to populate fields with a clearly defined invalid value that tests less than anything else.
\b	Backspace.
\t	Tab. Tabs behave erratically on the Web and are best avoided, but sometimes you need them.
\n	New line (\u0000a). Inserts a line break at the point specified. It is a combination of the carriage return (\r) and the form feed (\f).
\"	Double quote.
\'	Single quote, or an apostrophe, such as in can\'t.

- *Boolean values*
  - There are two boolean values, true and false.
  - These are normally the result of a logical comparison in your code that returns a true/false or yes/no result, such as:

    a = = b
  - This statment reads: does the value of variable a equal the value of variable b?
  - If you need to use boolean values in computations, JavaScript will automatically convert true to 1 and false to 0.
  - The conversion can also work the other way.
  - When testing for the result of a comparison, JavaScript will treat any non-zero value as true, and a zero value as false.
  - JavaScript will also test to false when you test for the existence of something that does not exist.

Composite Data types

- All composite data types can be treated as objects, but we normally categorize them by their purpose as a data type.
- For composite data types we will look at objects, including some special pre-defined objects that JavaScript provides, as well as functions and arrays.
    - *Objects:*
        - An *object* is a collection of named values, called the *properties* of that object.
        - Functions associated with an object are referred to as the *methods* of that object.
        - Properties and methods of objects are referred to with a dot.notation that starts with the name of the object and ends with the name of the property.
        - Objects in JavaScript can be treated as associative arrays.
        - JavaScript has many predefined objects, such as a **Date** object and a **Math** object.

    - *Functions:*
        - A *function* is a piece of code, predefined or written by the person creating the JavaScript, that is executed based on a call to it by name.
        - Any JavaScript that is not inside a function (or is not associated with some even attribute or hyperlink attribute) is executed the moment the Web browser reaches it when first parsing the document.
        - They also allow for the same piece of code to be re-used many times in the same document, since functions allow the section of code they contain to be referred to by name.
        - A function is a data type in JavaScript.
        - This means that they can be treated as containing values that can be changed.
    - *Arrays:*
        - An *Array* is an ordered collection of data values.
        - In some programming languages, arrays have very specific limitations.
        - In JavaScript, an array is just an object that has an *index* to refer to its contents.
        - In other words, the fields in the array are numbered, and you can refer to the number position of the field.
        - The array index is included in square brackets immediately after the array name.

- In JavaScript, the array index starts with zero, so the first element in an array would be arrayName[0], and the third would be arrayName[2].
- JavaScript does not have multi-dimensional arrays, but you can nest them, which is to say, an array element can contain another array.
- You access them listing the array numbers starting for the outmost array and working inward.
- Therefore, the third element (position 2) of or inside the ninth element (position 8) would be arrayName[8][2].

Variable - Introduction

- Variable is a name that can be used to store values.
- It is a name assigned to a location in computer's memory to store data.
- These variables can take different values but one at a time and the value can be changed during the execution of program.
- Variable names may consist of uppercase character, lowercase character and underscore.
- *Rules to giving the name of variable are as:*
  1. First character should be a letter or underscore.
  2. The variable name cannot be a keyword.
  3. Uppercase and lowercase letters are significant for example code, Code, CODE are three different variables. But variables are in lowercase.
- **Note:** The Netscape supports the ($) sign as the first character in variable but the Internet Explorer does not supports.

Typing

- Javascript is a untyped language.
- This means you can use variables directly where you want to use it.
- But in javascript the variables are declared with the **var** keyword.
- This keyword declares all types of variables and you can use a same variable as string, as Integer, as Number and any type of Objects.

- For Example:

```
var a=”Mr. ABC”;

a=12345;
```

- The both statements are valid in javascript, but in C and C++ the statements are not valid.
- The both statements are valid due to javascript is untyped language.

Local and Global Variables

- Javascript supports both local as well as global variables.

  Local Variables:

  - The variables which are defined within a body of the function or block is local to that function or block only is called local variable.
  - They have no presence outside the function.
  - The values of such Local variables cannot be changed by the main code or other functions.
  - Example:

```
<script language=”javascript”>
        function testLocal()
        {
                var a =5;
                alert(“The local value of a is: ” + a);
        }
</script>
```

  Global Variables:

  - The variables that are declared outside the function and is used inside the function is called global variables.
  - The global variable has the same data type and same name throughout the program.
  - It is useful to declare the variable global when the variable has constant value throughout the program.
  - These are the variables that can be used throughout the scripts and the value of which can be changed.

- ***Example:***

```
<script language="javascript">
    Var a=10;
    function testGlobal_1()
    {
        alert("The local value of a is: " + a);
        //displays the value of a as 10
    }
    function testGlobal_2()
    {
        alert("The Global value of a is:" + a);
        //displays the value of a as 10
    }
</script>
```

- Here the value of a is global and is accessed within both functions testGlobal_1 and testGlobal_2.

What is variable scoping?

- *Local variables* exist only inside a particular function hence they have ***Local Scope***.
- *Global variables* on the other hand are present throughout the script and their values can be accessed by any function. Thus, they have ***Global Scope***.

Constants

- You can create a read-only, named constant with the const keyword.
- The syntax of a constant identifier is the same as for a variable identifier: it must start with a letter or underscore and can contain alphabetic, numeric, or underscore characters.
- *Example:*

```
const prefix = '212';
```

- A constant cannot change value through assignment or be re-declared while the script is running.
- The scope rules for constants are the same as those for variables, except that the **const** keyword is always required, even for global constants.

- If the keyword is omitted, the identifier is assumed to represent a **var**.
- You cannot declare a constant at the same scope as a function or variable with the same name as the function or variable.
- *For example:*

```
//THIS WILL CAUSE AN ERROR
    function f{};
    const f = 5;
//THIS WILL CAUSE AN ERROR ALSO
    function f
    {
    const g=5;
    var g;
    }
```

Garbage Collection

- is a way of reclaiming the memory occupied by objects that are no longer in use.
- In C and C++, garbage collection is manual--the programmer explicitly decides when to free memory for reuse.
- In Java, on the other hand, garbage collection is handled automatically--the system can detect when objects are no longer in use and free them appropriately.
- JavaScript also supports automatic garbage collection.
- In Internet Explorer 3.0, garbage collection is implemented in a technically sound way and you don't have to understand any of its details.
- It is enough to know that when your objects are no longer in use, the memory they occupy will automatically be reclaimed by the system.
- Navigator 4.0 will also have a perfectly transparent garbage collection scheme like this.
- Unfortunately, garbage collection in earlier versions of Navigator is less than perfect.
- In Navigator 3.0, it is pretty good, but requires you to be aware of a couple of issues.
- In Navigator 2.0, garbage collection is seriously flawed, and you must take a number of steps to avoid crashing the browser.
- [***Note:*** Study self if you want to more about Garbage Collection]

Expression

- An *expression* is any valid set of literals, variables, operators, and expressions that evaluates to a single value:
  - the value can be a number,
  - a string, or
  - a logical value.
- Conceptually, there are two types of expressions:
  - those that assign a value to a variable, and
  - those that simply have a value.
- For example, the expression x = 7 is an expression that assigns x the value seven.
- This expression itself evaluates to seven. Such expressions use *assignment operators.*
- On the other hand, the expression 3 + 4 simply evaluates to seven; it does not perform an assignment.
- The operators used in such expressions are referred to simply as *operators*.
- *JavaScript has the following types of expressions:*
  1. *Arithmetic:* evaluates to a number, for example 3.14159.
  2. *String:* evaluates to a character string, for example, "Fred" or "234".
  3. *Logical:* evaluates to true or false.
  4. *Object:* evaluates to an object.

Operators

- Javascript has different types of operator, which can be classified by several criteria, such as:
  1. **Number of Operands**
     - In this type of category, javascript supports 3 basic types of Operators:
       - ***Unary***
         - Converts a single expression into a single more complex expression.
         - A unary operator requires a single operand, either before or after the operator.
         - For Example:

```
i--;
++i;
```

- ***Binary***
  - Combine two expressions into a single, more complex expression.
  - A binary operator requires two operands, one before the operator and one after the operator.
  - For Example:

    var x = 13 + 14
- ***Ternary***
  - Also known as conditional operator.
  - The conditional operator is the only JavaScript operator that takes three operands.
  - The operator can have one of two values based on a condition.
  - *The syntax is:*

    condition **?** val_1 **:** val_2
  - If condition is true, the operator has the value of val_1. Otherwise it has the value of val_2.
  - You can use the conditional operator anywhere you would use a standard operator.
  - *For example:*

    status = (age >= 18) **?** "adult" **:** "minor"
  - This statement assigns the value "adult" to the variable status if age is 18 or more.
  - Otherwise, it assigns the value "minor" to status.

2. Number of Operation.

- Javascript supports many types of operators according to this classification.
- They can be categorized as:

- ***Comparison Operators***
  - A comparison operator compares its operands and returns a logical value based on whether the comparison is true.
  - The operands can be numerical, string, logical, or object values.
  - Strings are compared based on standard lexicographical ordering, using Unicode values.
  - The following table describes the comparison operators.

Operator	Description	Example
Equal (==)	Returns true if the operands are equal. If the two operands are not of the same type, JavaScript attempts to convert the operands to an appropriate type for the comparison.	3 == var1 "3" == var1 3 == '3'
Not equal (!=)	Returns true if the operands are not equal. If the two operands are not of the same type, JavaScript attempts to convert the operands to an appropriate type for the comparison.	var1 != 4 var2 != "3"
Strict equal (===)	Returns true if the operands are equal and of the same type.	3 === var1
Strict not equal (!==)	Returns true if the operands are not equal and/or not of the same type.	var1 !== "3" 3 !== '3'
Greater than (>)	Returns true if the left operand is greater than the right operand.	var2 > var1
Greater than or equal (>=)	Returns true if the left operand is greater than or equal to the right operand.	var2 >= var1 var1 >= 3
Less than (<)	Returns true if the left operand is less than the right operand.	var1 < var2
Less than or equal (<=)	Returns true if the left operand is less than or equal to the right operand.	var1 <= var2 var2 <= 5

- ***Arithmetic Operators***
  - Arithmetic operators take numerical values as their operands and return a single numerical value.
  - The standard arithmetic operators are addition(+), subtraction(-), multiplication(*), and division(/).
  - These operators work as they do in most other programming languages, except the / operator returns a floating-point division in JavaScript.
  - In addition, JavaScript provides the arithmetic operators listed in the following table.

Operator	Description	Example
%(Modulus)	Binary operator. Returns the integer remainder of dividing the two operands.	12 % 5 returns 2.
++(Increment)	Unary operator. Adds one to its operand. If used as a prefix operator (++x), returns the value of its operand after adding one; if used as a postfix operator (x++), returns the value of its operand before adding one.	If x is 3, then ++x sets x to 4 and returns 4, whereas x++ sets x to 4 and returns 3.
--(Decrement)	Unary operator. Subtracts one to its operand. The return value is analogous to that for the increment operator.	If x is 3, then --x sets x to 2 and returns 2, whereas x++ sets x to 2 and returns 3.
-(Unary negation)	Unary operator. Returns the negation of its operand.	If x is 3, then -x returns -3.

- ***Assignment Operators***
  - An assignment operator assigns a value to its left operand based on the value of its right operand.
  - The basic assignment operator is equal (=), which assigns the value of its right operand to its left operand.
  - That is, x = y assigns the value of y to x.
  - The other assignment operators are shorthand for standard operations, as shown in the following table.

Shorthand operator	Meaning
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
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
x |= y	x = x | y

- ***Bitwise Operators***
  - Bitwise operators treat their operands as a set of 32 bits (zeros and ones), rather than as decimal, hexadecimal, or octal numbers.
  - For example, the decimal number nine has a binary representation of 1001.
  - Bitwise operators perform their operations on such binary representations, but they return standard JavaScript numerical values.
  - The following table summarizes JavaScript's bitwise operators.

Operator	Usage	Description
Bitwise AND	*a* & *b*	Returns a one in each bit position for which the corresponding bits of both operands are ones.
Bitwise OR	*a* | *b*	Returns a one in each bit position for which the corresponding bits of either or both operands are ones.
Bitwise XOR	*a* ^ *b*	Returns a one in each bit position for which the corresponding bits of either but not both operands are ones.
Bitwise NOT	~ *a*	Inverts the bits of its operand.
Left shift	*a* << *b*	Shifts a in binary representation b bits to left, shifting in zeros from the right.
Sign-propagating right shift	*a* >> *b*	Shifts a in binary representation b bits to right, discarding bits shifted off.
Zero-fill right shift	*a* >>> *b*	Shifts a in binary representation b bits to the right, discarding bits shifted off, and shifting in zeros from the left.

- ***Logical Operators***
  - Logical operators are typically used with Boolean (logical) values; when they are, they return a Boolean value.
  - However, the && and || operators actually return the value of one of the specified operands, so if these operators are used with non-Boolean values, they may return a non-Boolean value.
  - The logical operators are described in the following table.

Operator	Usage	Description
&&	expr1 && expr2	(Logical AND) Returns expr1 if it can be converted to false; otherwise, returns expr2. Thus, when used with Boolean values, && returns true if both operands are true; otherwise, returns false.
||	expr1 || expr2	(Logical OR) Returns expr1 if it can be converted to true; otherwise, returns expr2. Thus, when used with Boolean values, || returns true if either operand is true; if both are false, returns false.
!	!expr	(Logical NOT) Returns false if its single operand can be converted to true; otherwise, returns true.

Conditional Operator

JavaScript also contains a conditional operator that assigns a value to a variable based on some condition.

Syntax

```
variablename=(condition)?value1:value2
```

Example

```
greeting=(visitor=="PRES")?"Dear President ":"Dear ";
```

If the variable **visitor** has the value of "PRES", then the variable **greeting** will be assigned the value "Dear President " else it will be assigned "Dear".

Conditional statements are used to perform different actions based on different conditions.

Conditional Statements

Very often when you write code, you want to perform different actions for different decisions. You can use conditional statements in your code to do this.

In JavaScript we have the following conditional statements:

- **if statement** - use this statement to execute some code only if a specified condition is true
- **if...else statement** - use this statement to execute some code if the condition is true and another code if the condition is false
- **if...else if....else statement** - use this statement to select one of many blocks of code to be executed
- **switch statement** - use this statement to select one of many blocks of code to be executed

---

If Statement

Use the if statement to execute some code only if a specified condition is true.

Syntax

```
if (condition)
 {
 code to be executed if condition is true
 }
```

Note that if is written in lowercase letters. Using uppercase letters (IF) will generate a JavaScript error!

Example

```
<script type="text/javascript">
//Write a "Good morning" greeting if
//the time is less than 10

var d=new Date();
var time=d.getHours();

if (time<10)
  {
  document.write("<b>Good morning</b>");
  }
</script>
```

Notice that there is no ..else.. in this syntax. You tell the browser to execute some code **only if the specified condition is true**.

If...else Statement

Use the if....else statement to execute some code if a condition is true and another code if the condition is not true.

Syntax

```
if (condition)
  {
  code to be executed if condition is true
  }
else
  {
  code to be executed if condition is not true
  }
```

```
<script type="text/javascript">
//If the time is less than 10, you will get a "Good morning" greeting.
//Otherwise you will get a "Good day" greeting.

var d = new Date();
var time = d.getHours();
```

```
if (time < 10)
  {
  document.write("Good morning!");
  }
else
  {
  document.write("Good day!");
  }
</script>
```

If...else if...else Statement

Use the if....else if...else statement to select one of several blocks of code to be executed.

Syntax

```
if (condition1)
  {
  code to be executed if condition1 is true
  }
else if (condition2)
  {
  code to be executed if condition2 is true
  }
else
  {
  code to be executed if neither condition1 nor condition2 is true
  }
```

```
<script type="text/javascript">
var d = new Date()
var time = d.getHours()
if (time<10)
  {
  document.write("<b>Good morning</b>");
  }
else if (time>10 && time<16)
  {
  document.write("<b>Good day</b>");
  }
```

```
else
 {
 document.write("<b>Hello World!</b>");
 }
</script>
```

Conditional statements are used to perform different actions based on different conditions.

The JavaScript Switch Statement

Use the switch statement to select one of many blocks of code to be executed.

Syntax

```
switch(n)
{
case 1:
 execute code block 1
 break;
case 2:
 execute code block 2
 break;
default:
 code to be executed if n is different from case 1 and 2
}
```

This is how it works: First we have a single expression *n* (most often a variable), that is evaluated once. The value of the expression is then compared with the values for each case in the structure. If there is a match, the block of code associated with that case is executed. Use **break** to prevent the code from running into the next case automatically.

```
<script type="text/javascript">
//You will receive a different greeting based
//on what day it is. Note that Sunday=0,
//Monday=1, Tuesday=2, etc.

var d=new Date();
var theDay=d.getDay();
switch (theDay)
{
```

```
case 5:
  document.write("Finally Friday");
  break;
case 6:
  document.write("Super Saturday");
  break;
case 0:
  document.write("Sleepy Sunday");
  break;
default:
  document.write("I'm looking forward to this weekend!");
}
</script>
```

JavaScript has three kinds of popup boxes: Alert box, Confirm box, and Prompt box.

Alert Box

An alert box is often used if you want to make sure information comes through to the user.

When an alert box pops up, the user will have to click "OK" to proceed.

Syntax

```
alert("sometext");
```

```
<html>
<head>
<script type="text/javascript">
function show_alert()
{
alert("I am an alert box!");
}
</script>
</head>
<body>

<input type="button" onclick="show_alert()" value="Show alert box" />
```

```
</body>
</html>
```

Confirm Box

A confirm box is often used if you want the user to verify or accept something.

When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed.

If the user clicks "OK", the box returns true. If the user clicks "Cancel", the box returns false.

Syntax

```
confirm("sometext");
```

```
<html>
<head>
<script type="text/javascript">
function show_confirm()
{
var r=confirm("Press a button");
if (r==true)
  {
  alert("You pressed OK!");
  }
else
  {
  alert("You pressed Cancel!");
  }
}
</script>
</head>
<body>

<input type="button" onclick="show_confirm()" value="Show confirm box" />

</body>
</html>
```

Prompt Box

A prompt box is often used if you want the user to input a value before entering a page.

When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value.

If the user clicks "OK" the box returns the input value. If the user clicks "Cancel" the box returns null.

Syntax

```
prompt("sometext","defaultvalue");
```

```
<html>
<head>
<script type="text/javascript">
function show_prompt()
{
var name=prompt("Please enter your name","Harry Potter");
if (name!=null && name!="")
  {
  document.write("Hello " + name + "! How are you today?");
  }
}
</script>
</head>
<body>

<input type="button" onclick="show_prompt()" value="Show prompt box" />

</body>
</html>
```

- ***Special Operators***
  - Javascript provides the following Special Operators:
    1. *Comma Operator:*
       - The comma operator (,) simply evaluates both of its operands and returns the value of the second operand.
       - This operator is primarily used inside a for loop, to allow multiple variables to be updated each time through the loop.

- For example, if a is a 2-dimensional array with 10 elements on a side, the following code uses the comma operator to increment two variables at once.
- The code prints the values of the diagonal elements in the array:

```
for(i=0,j=9;i<=9;i++,j--)
document.write("a["+i+","+j+"]= " + a[i*10 +j])
```

- **[Note:** Two-dimensional arrays are not yet supported.**]**
- This example emulates a two-dimensional array using a one-dimensional array.

2. *delete*
   - The delete operator deletes an object, an object's property, or an element at a specified index in an array.
   - The syntax is:
   - You can use the delete operator to delete variables declared implicitly but not those declared with the var statement.
   - If the delete operator succeeds, it sets the property or element to undefined.
   - The delete operator returns true if the operation is possible; it returns false if the operation is not possible.
   - Example:

```
x=42
var y= 43
delete x //returns true (can delete if declared implicitly)
delete y   // returns false (cannot delete if declared with var)
```

3. *new*
   - You can use the new operator to create an instance of a user-defined object type or of one of the predefined object types Array, Boolean, Date, Function, Image, Number, Object, Option, RegExp, or String.
   - On the server, you can also use it with DbPool, Lock, File, or SendMail.
   - *The syntax for new operator is as follow:*

objectName = **new** *objectType* (*param1*...[,*paramN*])

JavaScript Loops

Often when you write code, you want the same block of code to run over and over again in a row. Instead of adding several almost equal lines in a script we can use loops to perform a task like this.

In JavaScript, there are two different kind of loops:

- **for** - loops through a block of code a specified number of times
- **while** - loops through a block of code while a specified condition is true

---

The for Loop

The for loop is used when you know in advance how many times the script should run.

Syntax

```
for (variable=startvalue;variable<=endvalue;variable=variable+increment)
{
code to be executed
}
```

Example

The example below defines a loop that starts with i=0. The loop will continue to run as long as **i** is less than, or equal to 5. **i** will increase by 1 each time the loop runs.

Note: The increment parameter could also be negative, and the <= could be any comparing statement.

```
<html>
<body>
<script type="text/javascript">
var i=0;
for (i=0;i<=5;i++)
{
document.write("The number is " + i);
document.write("<br />");
}
</script>
</body>
</html>
```

Loops execute a block of code a specified number of times, or while a specified condition is true.

The while Loop

The while loop loops through a block of code while a specified condition is true.

Syntax

```
while (variable<=endvalue)
 {
 code to be executed
 }
```

Note: The <= could be any comparing operator.

Example

The example below defines a loop that starts with i=0. The loop will continue to run as long as **i** is less than, or equal to 5. **i** will increase by 1 each time the loop runs:

```
<html>
<body>
<script type="text/javascript">
var i=0;
while (i<=5)
 {
 document.write("The number is " + i);
 document.write("<br />");
 i++;
 }
</script>
</body>
</html>
```

The do...while Loop

The do...while loop is a variant of the while loop. This loop will execute the block of code ONCE, and then it will repeat the loop as long as the specified condition is true.

Syntax

```
do
 {
 code to be executed
 }
while (variable<=endvalue);
```

Example

The example below uses a do...while loop. The do...while loop will always be executed at least once, even if the condition is false, because the statements are executed before the condition is tested:

```
<html>
<body>
<script type="text/javascript">
var i=0;
do
 {
 document.write("The number is " + i);
 document.write("<br />");
 i++;
 }
while (i<=5);
</script>
</body>
</html>
```

The break Statement

The break statement will break the loop and continue executing the code that follows after the loop (if any).

Example

```
<html>
<body>
<script type="text/javascript">
var i=0;
for (i=0;i<=10;i++)
 {
 if (i==3)
  {
  break;
  }
 document.write("The number is " + i);
 document.write("<br />");
 }
</script>
</body>
</html>
```

The continue Statement

The continue statement will break the current loop and continue with the next value.

```
<html>
<body>
<script type="text/javascript">
var i=0
for (i=0;i<=10;i++)
 {
 if (i==3)
  {
  continue;
  }
 document.write("The number is " + i);
 document.write("<br />");
 }
</script>
</body>
</html>
```

Javascript Events & Event Handlers

Events are keyboard, mouse or other actions that can be detected by JavaScript.

Every element on a web page has certain events which can trigger a JavaScript. For example, we can use the onClick event of a button element to indicate that a function will run when a user clicks on the button. We define the events in the HTML tags.

Examples of events:

- A mouse click
- A web page or an image loading
- Mousing over a hot spot on the web page
- Selecting an input field in an HTML form
- Submitting an HTML form
- A keystroke

Note: Events are used in combination with functions, and the function will not be executed before the event occurs!

Events Handlers

Event Handlers are very powerful and useful mechanism. They are *JavaScript code that are not added inside the <script> tags, but rather, inside the html tags,* that execute JavaScript when something happens, such as pressing a button, moving your mouse over a link, submitting a form etc.

To allow you to run your bits of code when these events occur, JavaScript provides ***event handlers***. All the event handlers in JavaScript start with the word on, and each event handler deals with a certain type of event. Here's a list of all the event handlers in JavaScript, along with the objects they apply to and the events that trigger them:

The basic syntax of these event handlers is:

name_of_handler="JavaScript code here"

For example:

```
<a href="http://google.com" onClick="alert('hello!')">Google</a>
```

When the above link is clicked, the user will first see an alert message before being taken to Google.

Different event handlers with with different HTML tags. For example, while "onclick" can be inserted into most HTML tags to respond to that tag's onclick action, something like "onload" only works inside the

<body> and <img> tags. Below are some of the most commonly used event handlers supported by JavaScript:

Event Handlers:

Event handler	Applies to:	Triggered when:
onAbort	Image	The loading of the image is cancelled.
onBlur	Button, Checkbox, FileUpload, Layer, Password, Radio, Reset, Select, Submit, Text, TextArea, Window	The object in question loses focus (e.g. by clicking outside it or pressing theTAB key).
onChange	FileUpload, Select, Text, TextArea	The data in the form element is changed by the user.
onClick	Button, Document, Checkbox, Link, Radio, Reset, Submit	The object is clicked on.
onDblClick	Document, Link	The object is double-clicked on.
onDragDrop	Window	An icon is dragged and dropped into the browser.
onError	Image, Window	A JavaScript error occurs.
onFocus	Button, Checkbox, FileUpload, Layer, Password, Radio, Reset, Select, Submit, Text, TextArea, Window	The object in question gains focus (e.g. by clicking on it or pressing the TAB key).
onKeyDown	Document, Image, Link, TextArea	The user presses a key.
onKeyPress	Document, Image, Link, TextArea	The user presses or holds down a key.
onKeyUp	Document, Image, Link, TextArea	The user releases a key.
onLoad	Image, Window	The whole page has finished loading.
onMouseDown	Button, Document, Link	The user presses a mouse button.
onMouseMove	None	The user moves the mouse.
onMouseOut	Image, Link	The user moves the mouse away from the object.

onMouseOver	Image, Link	The user moves the mouse over the object.
onMouseUp	Button, Document, Link	The user releases a mouse button.
onMove	Window	The user moves the browser window or frame.
onReset	Form	The user clicks the form's Reset button.
onResize	Window	The user resizes the browser window or frame.
onSelect	Text, Textarea	The user selects text within the field.
onSubmit	Form	The user clicks the form's Submit button.
onUnload	Window	The user leaves the page.

Using an event handler

To use an event handler, you usually place the event handler name within the HTML tag of the object you want to work with, followed by ="*SomeJavaScriptCode*", where *SomeJavaScriptCode* is the JavaScript you would like to execute when the event occurs.

For example:

```
<input type="submit" name="clickme" value="Click Me!" onclick="alert('Thank You!')"/>
```

JavaScript Functions

- A function is really nothing more than a named block of code.
- It is a statement block that has been assigned a name.

- Functions are small groups of instructions that are carried out when they are called upon.
- It is a reusable code-block that will be executed by an event, or when the function is called.
- A function typically contains a set of commands for a specific purpose, which you want to run at a certain time.
- Each function in a script is given a unique name.
- The layout of a function always follows the same format, including these three things:
  - The word "**function**"
    - **Function** tells the browser "Do these instructions when this name is called upon.".
  - The function's name, followed by parentheses (which may or may not be empty)
    - Function's name must be unique and every function name is contains the parentheses, which may be empty if the function does not take any parameter.
  - Curly braces containing the function's code.
    - The curly brackets (**{ }**)are used to contain the set of instructions.
- A script can contain any number of functions, according to the terms and conditions.

Defining Functions:

- The functions in javascript is defined by using the **function** keyword followed by function-name and the parentheses.

- *The Syntax of defining the functions:*

```
function function-Name(Argument lists...)
{
        /*
                some lines of codes
                return;
        */
}
```

- The **function** here is a keyword and should be explictly written in-front of the function name.
- The function is followed by the function name, which should be unique and should follow all the naming conventions that any function name should follow.

- The parentheses should be attached with every function name and in-between the parantheses you should lists the parameters if the function contains any parameter.
- If not then you must include an empty parentheses, but you should include parantheses.
- Then every function contains some lines of codes and these codes are included within the curly braces.
- If the function returns any value then it is returned with the keyword **return** from the function.

Invoking Functions

- Functions do not run automatically. When the page loads, each function waits quietly until it is told to run.
- To run a function you must *call* it.
- This means sending an instruction to the function telling it to run.
- *There are two common ways to call a function:*
  - From an *event handler* and
  - From another function.
  - *Calling Functions from Event-handlers*
    - An event handler is a command, which calls a function when an event happens, such as the user clicking a button.
    - The command is written in the format **on*Event***, where ***Event*** is the name for a specific action.
    - *Here are some common examples:*
    - User clicks a button (**onClick**):

      <input type="button" value="Click Here" **onClick**="doSomething();">
    - User places their cursor in a text field (**onFocus**):

      <input type="text" **onFocus**="doSomething();">
  - *Calling Functions from another Functions*
    - Functions can also call other functions. Simply enter the name of the function to be called, with its parentheses.

    - ***Example:***

      function doSomething()

```
{
        doSomethingElse(); //this line calls the another function
}
function doSomethingElse()
{
        //the function codes go here
}
```

The Function constructor

- JavaScript has a Function constructor which allows you create a variable and assign it a function as a value.
- *Example:*

```
var q = new Function('a', 'b', 'return a / b;');
```

- The Function constructor takes any number of arguments, each of which must be defined as a string, and the last of which must be the body of the code for the function.
- A function defined this way can still be invoked by the variable named, res = q(20, 5).
- But technically it has no name. Instead the function name is a variable that contains a reference to that function as a value.
- Such function are referred to as *anonymous* functions, since there is no way to refer to them by name.
- The benefit of declaring a function this way is that it is reparsed each time it is run, meaning, you could replace the strings that define the arguments and function body with variables, dynamically assign values to these variables, and thus rewrite the function each time it ran.
- Since the function is reparsed each time, it can really slow down processing if it is run repeatedly.

Function Properties

- As an object, a function has other properties as well.
- One is the length property, which specifies how many arguments the function is expecting.
- As with objects, you can also define properties for functions.
- This allows you to create values that persist across repeated calls to the function.

- To create a new property, you can just assign it a value.
- Since the function declaration is parsed before the code in the script is executed, we can put the initialization statement outside of the function.
- **Example:**

```
q.counter = 0;
function q (x)
{
        q.counter ++;
        //statements of the function
}
```

- Each time function q runs, it will increment the property q.counter by one.

What is HTML Editor?

A **HTML editor** is an authoring software program that is used to create content for web sites. HTML software is easy to use since it has a feature that is known as WYSIWYG.

It is a software application for creating web pages. Although the HTML markup of a web page can be written with any text editor, specialized HTML editors can offer convenience and added functionality. For example, many HTML editors work not only with HTML, but also with related technologies such as CSS, XML and JavaScript.

When you design web pages you want to use editor features that are simple to understand. You can buy and download HTML management software from the Internet as well as templates that will help you create web pages for your business or personal use.
HTML editors are also great for creating tables; building borders around images and changing background color in no time at all. You can easily change the design of your website in just a few minutes to reflect your business style.

Examples:

- Notepad, Notepad+, TextEdit(Mac)

WYSIWYG - ***W****hat* ***Y****ou* ***S****ee* ***I****s* ***W****hat* ***Y****ou* ***G****et*. The term is used in computing to describe a system in which content (text and graphics) displayed onscreen during editing appears in a form exactly corresponding to its appearance when printed or displayed as a finished product, which might be a printed document, web page, or slide presentation.

WYSIWYG implies a user interface that allows the user to view something very similar to the end result while the document is being created. In general WYSIWYG implies the ability to directly manipulate the layout of a document without having to type or remember names of layout commands. The actual meaning depends on the user's perspective.

Examples:

- Adobe Dreamweaver (With this software user can view the output as he writes the HTML code.)

HTML Convertor

A software program that is used to convert basic text files to HTML code. Different software tools can be used for the conversion of HTML. In Adobe Dreamweaver too the user can make required design with Drag-and-Drop mouse event and the software will automatically write the corresponding HTML code. The Adobe Photoshop also provide this facility, The user can save the sliced image in web format which will automatically convert the image to HTML files that can be opened with the browsers.

The office software packages also provide the facility to save the documents in web format, which means they are converting the normal documents in web format.

~