

UNIT-4

JAVASCRIPT

Overview of JavaScript

JavaScript is a sequence of statements to be executed by the browser. It is most popular scripting language on the internet, and works in all major browsers, such as IE, FireFox, chrome, opera safari.

Prerequisite –HTML/XHTML

Origins

It is originally known as LiveScript, developed by Netscape. It become a joint venture of Netscape and Sun in 1995, and was renamed as JavaScript. It was standardized by the European computer Manufacturers Association as ECMA-262. ISO-16262. Current standard specifications can be found at

<http://www.ecma-international.org/publications/standardsEcma-262.htm>
Collections of JavaScript code scripts and not programs.

What is JavaScript?

1. JavaScript was designed to add interactivity to HTML pages.
2. JavaScript is a scripting language.
3. A scripting language is a lightweight programming language.
4. It is usually embedded directly into HTML pages.
5. JavaScript is an interpreted language (Scripts are executed without preliminary compilations)

JavaScript can be divided into three parts.

1. The Core :

It is a heart of the language, including its operators, expressions, statements and subprograms.

2. Client Side :

It is a collection of objects that support control of a browser and interactions with users.

Eg. With JavaScript an XHTML document can be made to be responsible to user inputs. Such as mouse clicks and keyboard use.

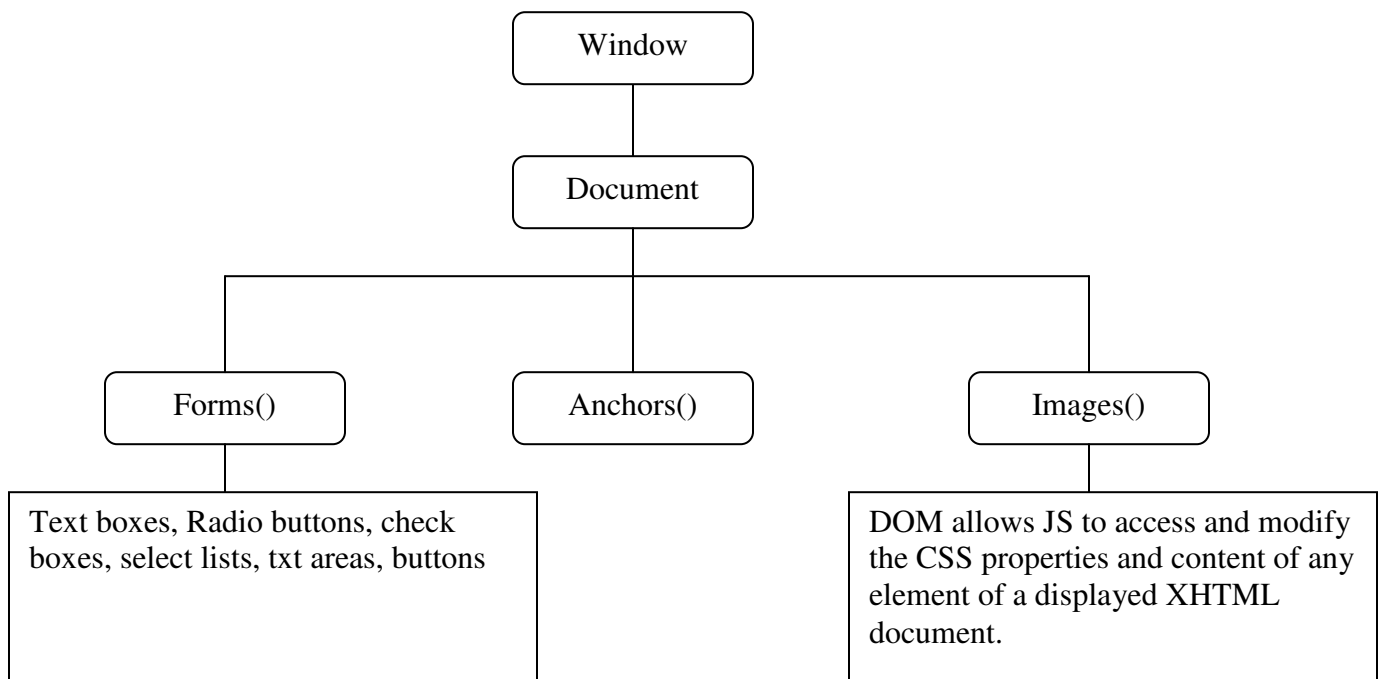
3. Server side :

It is a collection of objects that make the language useful on a Web server.
 Eg. To support communication with a DBMS.

Client side JavaScript is an XHTML embedded scripting language. We refer to every collection of JavaScript code as a script. An XHTML document can include any number of embedded scripts.

The HTML Document Object Model(DOM) is the browsers view of an HTML page as an object hierarchy, starting with the browser window itself and moving deeper into the page, including of the elements on the page and their attribute.

Fig: The HTML DOM



The top level object is window. The document object is a child of window and all the objects that appear on the page are descendants of the document object. These objects can have children of their own.

Eg. Form objects generally have several child objects , including textboxes, radio buttons and select menus.

JavaScript and java

- JavaScript and java is only related through syntax.
- JavaScript support for OOP is different from that of Java.
- JavaScript is dynamically typed.
- Java is strongly typed language. Types are all known at compile time and operand types are checked for compatibility. But variables in JavaScript need not be declared and are dynamically typed, making compile time type checking impossible.
- Objects in Java are static -> their collection of data number and methods is fixed at compile time.
- JavaScript objects are dynamic : The number of data members and methods of an object can change during execution.

Uses of JavaScript

Goal of JavaScript is to provide programming capability at both server and the client ends of a Web connection.

Client-side JavaScript is embedded in XHTML documents and is interpreted by the browser. This transfer of load from the often overloaded server to the normally under loaded client can obviously benefit all other clients. It cannot replace server side computations like file operations, database access, and networking.

JavaScript can be used as an alternative to Java applets. Java applets are downloaded separately from the XHTML documents that call them but JavaScript are integral part of XHTML document, so no secondary downloading is necessary. Java applets far better for graphics files scripts.

Interactions with users through form elements, such as buttons and menus, can be conveniently described in JavaScript. Because events such as button clicks and mouse movements are easily detected with JavaScript they can be used to trigger computations and provide feedback to the users.

Eg. When user moves the mouse cursor from a textbox, JavaScript can detect that movement and check the appropriateness of the text box's value. Even without forms, user interactions are both possible and simple to program. These interactions which take place in dialog windows, include getting input from the user and allowing the user to make choices through buttons. It is also easy to generate new content in the browser display dynamically.

Event driven computation

Event driven computation means that the actions often are executed in response to actions often are executed in response to actions of the users of doc, actions like mouse clicks and form submissions. This type of computation supports user interactions through XHTML form elements on the client display. One of the common uses of JS is client end

input data validation values entered by users will be checked before sending them to server for further processing. This becomes more efficient to perform input data checks and carry on this user dialog entirely on the client. This saves both server time and internet time.

Browsers and XHTML/JS documents.

It is an XHTML document does not include embedded scripts, the browser reads the lines of the document and renders its window according to the tags, attributes and content it finds when a JavaScript script is encountered in the doc, the browser uses its JS interpreter to execute the script. When the end of script reached, the browser goes back to reading the XHTML document and displaying its content.

JS scripts can appear in either part of an XHTML document, the head or the body, depending on the purpose of the script. Scripts that produce content only when requested or that react to user interactions are placed in the head of the document. -> Function definition and code associated with form elements such as buttons.

Scripts that are to be interpreted just once, when the interpreter finds them are placed in the document body. Accordingly, the interpreter notes the existence of scripts that appear in the head of a document, but it does not interpret them while processing the head. Scripts that are found in the body of a document are interpreted as they are found.

Object Orientation and JavaScript

JavaScript is object based language. It doesn't have classes. Its objects serve both as objects and as models of objects. JavaScript does not support class based inheritance as is supported in OO language. CTT-Java. But it supports prototype based inheritance i.e a technique that can be used to simulate some of the aspects of inheritance.

JavaScript does not support polymorphism. A polymorphic variable can reference related objects of different classes within the same class hierarchy. A method call through such a polymorphic variable can be dynamically bound to the method in the objects class.

JavaScript Objects

JavaScript objects are collection of prospectus, which corresponds to the members of classes in Java & C++. Each property is either a data property or a function or method property.

1. Data Properties

- a. Primitive Values(Non object Types)
- b. Reference to other objects

2. Method Properties –methods.

Primitives are non object types and are used as they can be implemented directly in hardware resulting in faster operations on their values. These are accessed directly-like scalar types in java & C++ called value types.

All objects in a JavaScript programs are directly accessed through variables. Such a variable is like a reference in java. The properties of an object are referenced by attaching the name of the property to the variable that references the object.

Eg. If myCar variable referencing an object that has the property engine, the engine property can be referenced with myCar.engine.

The root object in JavaScript is object. It is ancestor through prototype inheritance, of all objects. Object is most generic of all objects, having some methods but no data properties. All other objects are specializations of object, and all inherit its methods.

JavaScript object appears both internally and externally as a list of property/value pairs. Properties are names values are data values of functions. All functions are objects and are referenced through variables. The collection of properties of JavaScript is dynamic –Properties can be added or deleted at any time.

General syntactic Characteristics

1. JavaScript are embedded either directly or indirectly in XHTML documents.
2. Scripts can appear directly as the content of a <script> tag.
3. The type attribute of <script> must be set to “text/JavaScript”.
4. The JavaScript can be indirectly embedded in an XHTML document using the src attribute of a <script> tag, whose value is name of a file that contains the script.

Eg. <script type=”text/JavaScript” src=”tst_number.js”>
</script>

Closing tag is required even if script element has src attribute included.

The indirect method of embedding JavaScript in XHTML has advantages of

- 1) Hiding the script from the browser user.
- 2) It also avoids the problem of hiding scripts from older browsers.

- 3) It is good to separate the computation provided by JavaScript from the layout and presentation provided by XHTML and CSS respectively.

But it is sometimes not convenient and cumbersome to place all JavaScript code in separate file JavaScript identifiers or names are similar to programming languages.

1. must begin with (-), or a letter. Subsequent characters may be letters, underscores or digits.
2. No length limitations for identifiers.
3. Case sensitive
4. No uppercase letters.

Reserved words are

break	delete	function	return	typeof
case	do	if	switch	var
catch	else	in	this	void
continue	finally	instanceof	throw	while
default for	new	try	with	

JavaScript has large collection of predefined words

alert
open
java
self

Comments in JavaScript

// - Single line
/* */ -Multiple line

Two issues regarding embedding JavaScript in XHTML documents.

- 1) There are some browsers still in use that recognize the <script> tag but do not have JS interpreters. These browsers will ignore the contents of the script element and cause no problems.
- 2) There are still a few browsers in use that are so old they do not recognize <script> tag. These browsers will display the contents of the script elements as if it were just text.

Therefore it has been customary to enclose the contents of all script elements in XHTML comments to avoid this problem.

XHTML validator also has a problem with embedded JS. When embedded JS happens to include recognizable tags.

For eg
 in output of JS-they often cause validation errors.

Therefore we have to enclose embedded JS in XHTML comments.

XHTML comment introduction (<! - -) works as a hiding prelude to JS code.

Syntax for closing a comment that encloses JS code is different. It is usual XHTML comment closer but it must be on its own line and preceded by two slashes.

```
Eg. <!--
      -- JS ---
      //-->
```

Many more problem are associated with putting embedded JavaScript in comments in XHTML document.

Solution : Put JavaScript scripts of significant style in separate files.

Use of ; in JS is unusual

When EOL coincides with end of statement, the interpreter effectively inserts a semicolon there, but this leads to problems.

Eg.

```
return
```

```
x;
```

Interpreter puts; after return making x an illegal orphan.

Therefore put JS statements on its own line when possible and terminate each statement with a semicolon. If stmt does not fit in one line, break the stmt at a place that will ensure that the first line does not have the form of a complete statement.

```
<?xml version = "1.0 encoding = "utf-8"?>
<!DOCTYPE html PUBLIC "-//w3c//DTD XHTML 1.1//EN"
  http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd>
<! - -hello.html
```

A trivial hello world example of XHTML/JavaScript

```

- ->
<html xmlns = "http://www.w3.org/1999/xhtml".>
<head>
  <title> Hello World</title>
</head>
<body>
  <script type = "text/javascript">
    <!--
      Document.write("Hello, fellow Web programmers!");
    //-->
  </script>
</body>
</html>

```

Primitives, Operations and Expressions:

The primitive data types, operations and expressions of JavaScript.

Primitive Types:

Pure primitive types : Number, String, Boolean, Undefined and null.

JavaScript includes predefined objects that are closely related to the number, string and Boolean types named number, string and Boolean.

These are wrapper objects. Each contains a property that stores a value of the corresponding primitive type. The purpose of the wrapper object is to provide properties and methods that are convenient for use with values of the primitive types.

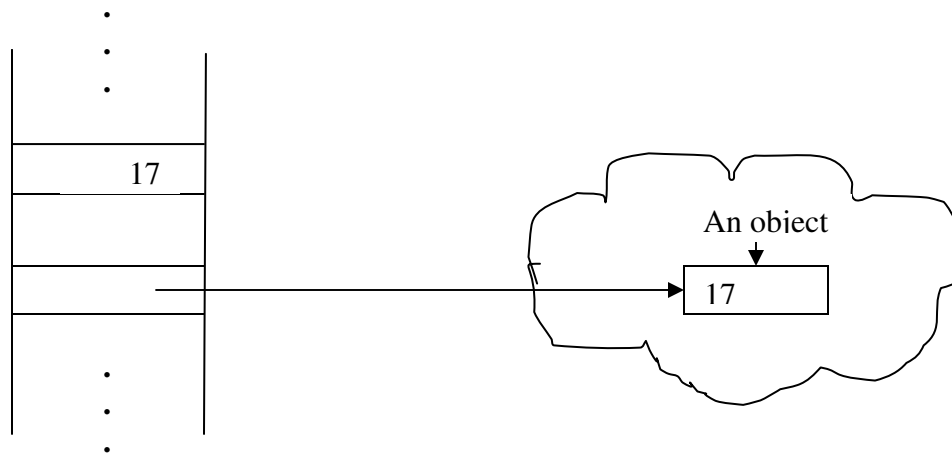
In case of numbers : Properties are more useful.

In case of string : Methods are more useful.

Because JavaScript coerces values between the number type and number objects and between the string type and objects, the methods of number and string can be used on variables of the corresponding primitive types.

Difference between primitives and objects :

Fig:



Prim is a primitive variable with value 17 and obj is a number object whose property value is 17. Fig shows how they are stored.

Numeric and String literals:

All numeric literals are values of type number. The numeric values of JavaScript are represented internally in double precision floating point form, Numeric values in JavaScript are called numbers because of single numeric data type. Literal numbers in a script can have forms of either integers or floating point values. Integer literals are strings of digits.

Floating point literals can have decimal points or exponents or both.

Legal numeric literals: 72, 7.2, .72, 72, 7E2, 7e2, .7e2, 7.e2, 7.2E-2.

Integers in Hexadecimal form 0x or 0X. String Literal: Sequence of 0 or more characters delimited by either single quotes or double quotes. They can include characters specified with escape sequences, such as \n and \t. If you want an actual single quote character in a string literal that is delimited by single quotes, embedded single quote must be preceded by a backslash.

'You\' re the most freckly person I\'ve ever met'

"D:\\bookfiles" -> Jo embed\

'' or "" -> Null string

Other primitive types:

Null Type : Value of this type is nullise (reserved word) novalue. Variable is null if explicitly declared. Any attempt to use the value of a variable which is declared as null will cause a runtime error.

Undefined: Is not a reserved word i.e if a variable is explicitly declared, but not arranged a value. If value of undefined variable is displayed the word 'undefined' is displayed.

Boolean : true/false. These are computed as a result of evaluating a relational/Boolean expression.

Declaring Variables:

Dynamically typed :-> means that a variable can be used for anything and variables are not typed only the values are typed. Variables can have the value of any primitive type or it can be a reference to any object. The type of value of a particular appearance of a variable in a program is determined by the interpreter. i.e interpreter converts the type of a value to whatever is needed for the context in which it appears.

Variable Declaration:

By assigning a value, by listing it in a declaration statement that begins with a reserved word var.

Eg.

```
var counter, //undefined
index,
pi=3.14,
quarterback = "Elway",
stop_flag = true;
```

Numeric Operators :

Binary Operators: +, -, *, /, %

Unary operators: +, -, --, ++

Prefix and postfix are not always equivalent.

If a=7

(++a)*3 ans ->24

(a++)*3 ans ->21

All numeric operations are done in double precision floating point.

Precedence rules:

Specify which operator is evaluated first when two operators with different precedence are adjacent in an expression.

Associativity rules :

Specify which operator is evaluated first when two operators with same precedence are adjacent in an expression.

<u>Operator</u>	<u>Associativity</u>
++, --, unary-, unary+	Right
*, /, %	Left
Binary+, Binary-	left

Precedence : Precedence and associativity of the numeric operators.

Eg. var a=2,

b=4,

c,

d;

c=3+a*b; c=11

d=b/a/2 d=1

(a+b)*c

Math Object:

Provides a collection of properties of Number objects and methods that operate on number objects. Math object has methods for trigonometric functions - sine, cos etc.

floor - to truncate a number
 round - to round a number
 max - to return largest of 2 given no.

Number Object:

Includes a collection of useful properties that have constant values. These properties are referenced through number.

Eg. Number.MIN_VALUE.

Properties of Number:

Property	Meaning
MAX_value	Largest representable number
MIN_value	Largest representable number
NaN	Not a Number
Positive_Infinity	Special value to represent infinity.
NEGATIVE_INFINITY	Special value to represent -ve infinity.
PI	The value of π

Any arithmetic operation that results in an error(for eg. division by zero) or that produces a value that can be represented as a double precision floating point number such as one that is too large(overflow), returns the value “not a number”, displayed as NaN.

If NaN is compared to any number or with itself for equality it fails.

isNAN() -> Is a predicate function used to determine whether a variables NaN value.

Returns true if variable has the NaN value.

tostring ->Number objects inherits it from objects but overrides converts number through which it is called to a string.

```

var price = 427,
str_price;
...

```

```
str_price = price.toString();
```

numeric primitives and number objects are always coerced to the other when necessary toString can be called through a numeric primitive.

String Catenation Operator:

JavaScript strings are not stored treated as arrays of characters but as unit scalar values. String catenation is specified with the operator denoted by + sign

Eg: The value of first is “Freddie” is The value of the following expression is “Freddie Freeloader”:

```
first + “freeloader”
```

Implicit type conversion:

Coercions – are different implicit type conversions done by JavaScript interpreter ie when a value of one type is used in a position that requires a value of a different type, JS attempts to convert the value to the type that is required.

Most commonly used ex involve primitive string number value.

If either operator of a + is a string, the operator is interpreted as a string catenation operator. If the other operand is not a string, it is coerced to a string.

Eg. “August” +1977 or 1977+ “August”.

```
7 * ”3” =>21
```

If 3 was replaced by “August” then conversion would produce NaN.

Null is 0 when used as number.

Undefined is interpreted as NaN when used as a number. Number 0 is false and other numbers are true when interpreted as a boolean.

When string is interpreted as a Boolean the empty string is false and other strings are true.

If a special value NaN is interpreted as a Boolean, it is false.

If undefined is used as a Boolean, it is false.

Null is false when interpreted as a Boolean, true=1 and false=0 when interpreted as a number.

Explicit type Conversion:

Several ways to force type conversions primarily between strings and numbers. Strings that contain numbers can be converted to numbers with the string constructor.

```
Var str_value = string(value);
```

Eg. var num = 6;

```
var str_value = num.toString(); =6
```

```
var str_value_binary = num.toString(2); = 110
```

```
var number = Number(a string);    - using Number constructor.
```

```
Var number = astring - 0;
```

But the number in the string cannot be followed by any char except a space. Conversion will not work even if it has ','

parseInt and parseFloat operate on strings given as parameters.

parseInt - Searches the string for an integer literal. If found at the beginning of string it is converted to number and returned.

If not found NaN is returned.

parseFloat- Same as above except it searches for floating point literal, which could have a decimal point or an exponent or both.

In both if numeric values are followed by any nondigit character, it converts it without any problem.

String properties and methods:

Differences between string objects and string type have little effects on scripts since JS coerces primitive string values to and from string objects. String methods

can always be used through string primitive values, as if the values were objects. The string object includes one property length and a large collection of methods

```
Var str= "Nandini";
```

```
Var len =str.length;
```

```
len = 6
```

str.length = primitive variable but can be treated as an object.

When str is being used with the length property JS implicitly builds a temporary string object with a property whose value is that of the primitive variable. After the II statement is executed, the temporary string object is discarded.

Method	Parameters	Result
charAt	A number	Returns the character in the String object that is at the specified position.
indexOf	One-character string	Returns the position in the string object of the parameter
substring	Two numbers	Returns the substring of the String object from the first parameter position to the second
toLowerCase	None	Converts any uppercase letters in the string to lowercase
toUpperCase	None	Converts any lowercase letters in the string to uppercase

Eg. `var str = "Nandini";`
`Str.charAt(2) = n`
`Str.indexOf('d') =3`
`Str.substring(2,4)` is 'ndi'
`Str.toLowerCase()` is 'nandini'

The typeof Operator :

The `typeof` operator returns the type of its single operand. It evaluates to "number", "string" or "Boolean". If the operand is of primitive type Number, String or Boolean respectively.

If operand is an object or null, `typeof` evaluates to "object".

This implies the fundamental characteristics of JS-objects do not have types. If the operand is a variable that has not been assigned a value, `typeof` evaluates to "undefined", reflecting the fact that variables themselves are not typed. It always returns a string. The operand for `typeof` can be placed in parenthesis or may not be in parenthesis. i.e

`typeof X` and `typeof(x)` are equivalent.

Assignment Statements:

It is same as C language.

JavaScript has two kinds of values-primitives and objects. A variable can refer to a primitive value like 17 or an object as shown in 4.1. Objects are allocated on the heap and variables that refer to them are essentially reference variables. When used to refer to an object, a variable stores an address only.

Therefore assigning the address of an object to a variable is fundamentally different from assigning a primitive value to a variable.

Date Object:

It is required to create objects to represent specific date and time and manipulate them. These capabilities are available in JS through the date object and its methods.

A Date object is created naturally with the `new` operator and the `Date` constructor, which has several forms.

Eg. `var today = new Date();`

Date and time properties of a Date object are in two forms local and coordinated Unusual Time.

(UTC).

Method	Returns
toLocaleString	A string of the Date information
getDate	The day of the month
getMonth	The Month of the year, as a number in the range of 0 to 11
getDay	The day of the week, as a number in the range of 0 to 6
getFullYear	The Year
getTime	The number of milliseconds since January 1, 1970
getHours	The number of the hour, as a number in the range of 0 to 23
getMinutes	The number of the minute, as a number in the range of 0 to 59
getSeconds	The number of the second, as a number in the range of 0 to 59
getMilliseconds	The number of the millisecond, as a number in the range of 0 to 999

Screen Output and Keyboard Input:

A JavaScript is interpreted when the browser finds the script in the body of the XHTML document. Thus the normal screen for the JavaScript is the same as the screen in which the content of the host XHTML document is displayed. JS models the XHTML document with the document object. The window in which the browser displays an XHTML document is modeled with the window object. It includes two properties document and window.

Document -> Refers to document object.

Window -> self referential and refers to window object.

Methods -> write

Write is used to create script o/p, which is dynamically created XHTML document content.

Eg. document.write("The result is : “, result,”
”);

Because write is used to create XHTML code, the only useful punctuations in its parameter is in the form of XHTML tags. Therefore the parameter to write often includes `
` writeLn methods implementing adds “\n” to its parameter. As browsers neglects line breaks when displaying XHTML, it has no effect on the output.

Window object is JS model for the browser window. It includes three methods that create dialog boxes for three specific kinds of user interactions. The default object for JS is window object currently being displayed, so calls to these methods need not include an object reference.

Alert method opens a dialog window and displays its parameter in that window. It also displays an OK button. The parameter string to alert is not XHTML code, it is plain text.

Therefore the string parameter to alert may include \n but never should include `
`.

(Fig 4.3)

Confirm method opens a dialog window in which it displays its string parameter, along with two buttons OK and Cancel. Confirm returns a Boolean value that indicates the users button input

True -> for OK

False-> for cancel.

Eg. `var question = confirm("Do you want to continue this download?");`

After the user responds to one of the button in the confirm dialog window script can test the variable, question and react accordingly.

Prompt method creates a dialog window that contains a text box which is used to collect a string of input from the user, which prompt returns as its value. The window also includes two buttons, OK and Cancel, prompt takes two parameters the string that prompts the user for input and a default string in case the user does not type a string before pressing one of the two buttons. In many cases an empty string is used for the default input.

Eg. `name = prompt("What is your name?");`

(fig 4.5)

Alert, prompt and confirm cause the browser to wait for a user response.

Alert – OK

Prompt-OK, Cancel

Confirm-OK, Cancel.

Eg.

```
<html>
  <head>
    </title> roots.html</title>
  </head>
  <body>
    <script type =”text/javascript” src=”roots.js”>
      </script>
  </body>
</html>
```

```
//roots.js
```

```
// Compute the real roots of a given quadratic equation. If the roots are imaginary,
//this script displays NaN, because that is what results from taking the square root of
//a negative number.
```

```
//Get the coefficients of the equation of the equation from the user
```

```
var a = prompt(“What is the value of ‘a’?\n”, “”);
```

```
var b = prompt(“What is the value of ‘b’?\n”, “”);
```

```
var c = prompt(“What is the value of ‘c’?\n”, “”);
```

```
// Compute Square root
```

```
var root_part = Math.sqrt(b * b – 4.0 * a * c);
```

```
var denom = 2.0 * a;
```

```
//Compute and Display
```

```
Var root1 = (-b + root_part) / denom;
```

```
Var root2 = (-b - root_part) / denom;
```

```
document.write(“The first root is :”,root1, “<br/>”);
```

```
document.write(“The second root is :”,root2, “<br/>”);
```

Control Statements:

Control expression control the order of execution of statements.

Compound statements in JavaScript are syntactic constructs for sequences of statements whose execution they control. Compound statement sequence of statements delimited by braces.

Control construct is a control statement whose execution it controls.

Compound statements are not allowed to create local variables.

Control Expressions.

Control statements flow.

Eg. primitive values, relational expression and compound expressions. Result of evaluating a control expression is Boolean value true or false.

For strings.

true - string

false-null string

For number

True- any number

False-0

Operation	Operator
Is equal to	==
Is not equal to	!=
Is less than	<
Is greater than	>
Is less than or equal to	<=
Is greater than or equal to	>=
Is strictly equal to	===
Is strictly not equal to	!==

If two operands are not of the same type and operator is neither === or !==, JS will convert to a single type.

Eg. If one is string and other is number, JS will convert string to a number.

If one operand is Boolean and other is not, then Boolean is converted to a number.(1 for true, 0 for false)

=== and !== disallow type conversion of either operand.

Eg. '3' === 3 evaluated as false.

Comparisons of variables that refer objects are not useful. Eg. If a and b refer different objects, a == b is false but a === b is true if a and b reference to the same object

JS has &&(AND), ||(OR) and !(NOT)

The properties of the Boolean object are different from primitive values true and false. If a Boolean object is used as a conditional expression, it evaluates to true if it has any value other than null or undefined.

Operator precedence and Associativity:

Selection Statements_ (If then and if-then-else)

```

if(a>b)
    document.write("a is greater than b <br/>");
else
    {
        a=b;
        document.write("a was not greater than b<br/>");
        "Now they are equal <br/>");
    }

```

The switch statement

JS has switch statement like C.

```

Switch(expression)
{
    Case value-1; //statement(s)
    Case value-2; //statement(s)
    .
    .
    .
    [default : // statement(s)]
}

```



Statement sequence or a compound statement.

Control expression could evaluate to a number, string or a Boolean value, case labels also can be numbers strings or Boolean and different case values can be numbers strings or Boolean and different case values can be of different types.

Eg.

Loop statements

```
while(control expression)
    statement or compound statement
```

```
for(initial expression, control expression; increment expression)
    statement or compound statement
```

Eg.

```
do statement or compound statement
while(control expression)
```

Eg.

```
Do {
    count ++;
    sum = sum +(sum*count);
}while count <=50;
```