MODULE 2

Introduction to JavaScript : Introduction to Scripting- Programming fundamentals of JavaScript -Obtaining User Input with prompt Dialogs-Arithmetic-Decision Making -Control Statements - Functions -Arrays -Objects -Document Object Model (DOM) -Form processing

JAVASCRIPT

- JavaScript is a scripting language, which is used to enhance the functionality and appearance of web pages
- JavaScript is a Client side scripting tool
- JavaScript is used in Web pages to improve the design, validate forms, detect browsers, create cookies, and much more.
- JavaScript is the most popular scripting language on the internet, and works in all major browsers, such as Internet Explorer, Mozilla, Firefox, Netscape, Opera
- JavaScript and Java are only related through syntax →JavaScript is dynamically typed →JavaScript's support for objects is very different

Java & JAVASCRIPT

- Java and JavaScript are two completely different languages in both concept and design!
- Java (developed by Sun Microsystems) is a powerful and much more complex programming language - in the same category as C and C++.
- Object model of javascript is quite different from java & C++
- Java is strongly typed and Javascript is Dynamically typed
- In Java ,types are all known at compile time, where as compile time type checking impossible
- Objects in Java are static (variables or function is shared between all instances of class)where as objects are dynamic

Uses of JAVASCRIPT

- JavaScript was designed to add interactivity to HTML
- JavaScript is a scripting language
- It is usually embedded directly into HTML pages
- JavaScript is an interpreted language
- provide programming capability at both the server and the client ends of a Web connection.
- Improve appearance especially graphics
- Site navigation
- Perform calculations
- Validation of input
- DOM allows Javascript to access &modify the style

First Example

```
<!DOCTYPE html>

<!-- Fig. 6.1: welcome.html -->
<!-- Displaying a line of text. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>A First Program in JavaScript</title>
      <script type = "text/javascript">

         document.writeln(
            "<h1>Welcome to JavaScript Programming!</h1>" );

      </script>
   </head><body></body>
</html>
```

Printing one line with separate statements

```
<!DOCTYPE html>

<!-- Fig. 6.2: welcome2.html -->
<!-- Printing one line with multiple statements. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>Printing a Line with Multiple Statements</title>
      <script type = "text/javascript">
         <!--
         document.write( "<h1 style = 'color: magenta'>" );
         document.write( "Welcome to JavaScript " +
            "Programming!</h1>" );
         // -->
      </script>
   </head><body></body>
</html>
```

Displaying Text in an Alert Dialog

```
<!DOCTYPE html>

<!-- Fig. 6.3: welcome3.html -->
<!-- Alert dialog displaying multiple lines. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>Printing Multiple Lines in a Dialog Box</title>
      <script type = "text/javascript">
         <!--
         window.alert( "Welcome to\nJavaScript\nProgramming!" );
         // -->
      </script>
   </head>
   <body>
      <p>Click Refresh (or Reload) to run this script again.</p>
   </body>
</html>
```

How to embed JavaScript in an HTML document

- When a JavaScript script is encountered in the HTML document, the browser uses its JavaScript interpreter to "execute" the script.
- Output from the script becomes the next markup to be rendered.
- When the end of the script is reached, the browser goes back to reading the HTML document and displaying its content.
- There are two different ways to embed JavaScript in an HTML document:
  - implicitly and explicitly.
- In explicit embedding, the JavaScript code physically resides in the XHTML document.
- In implicit embedding the JavaScript can be placed in its own file(.js file), separate from the XHTML document.
- When JavaScript scripts are explicitly embedded, they can appear in either part of an XHTML document—the head or the body
- 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.

How to embed JavaScript in an HTML document

- JavaScript are embedded in HTML documents , Either directly, as in • -

```
<script type = "text/javaScript">
    -- JavaScript script –
 </script>
```

- Or indirectly, as a file specified in the src attribute of

```
<script type = "text/javaScript"
        src = "myScript.js">
 </script>
```

JavaScript example with alert-Usage of Escape Sequence

- The backslash (\) in a string is an escape character.
- It indicates that a "special" character is to be used in the string.
- When a backslash is encountered in a string, the next character is combined with the backslash to form an escape sequence.

Escape sequence	Description
`\n`	*New line*—position the screen cursor at the beginning of the next line.
`\t`	*Horizontal tab*—move the screen cursor to the next tab stop.
`\\`	*Backslash*—used to represent a backslash character in a string.
`\"`	*Double quote*—used to represent a double-quote character in a string contained in double quotes. For example, `window.alert( "\"in double quotes\"" );` displays `"in double quotes"` in an `alert` dialog.
`\'`	*Single quote*—used to represent a single-quote character in a string. For example, `window.alert( '\'in single quotes\'' );` displays `'in single quotes'` in an `alert` dialog.

Some common escape sequences.

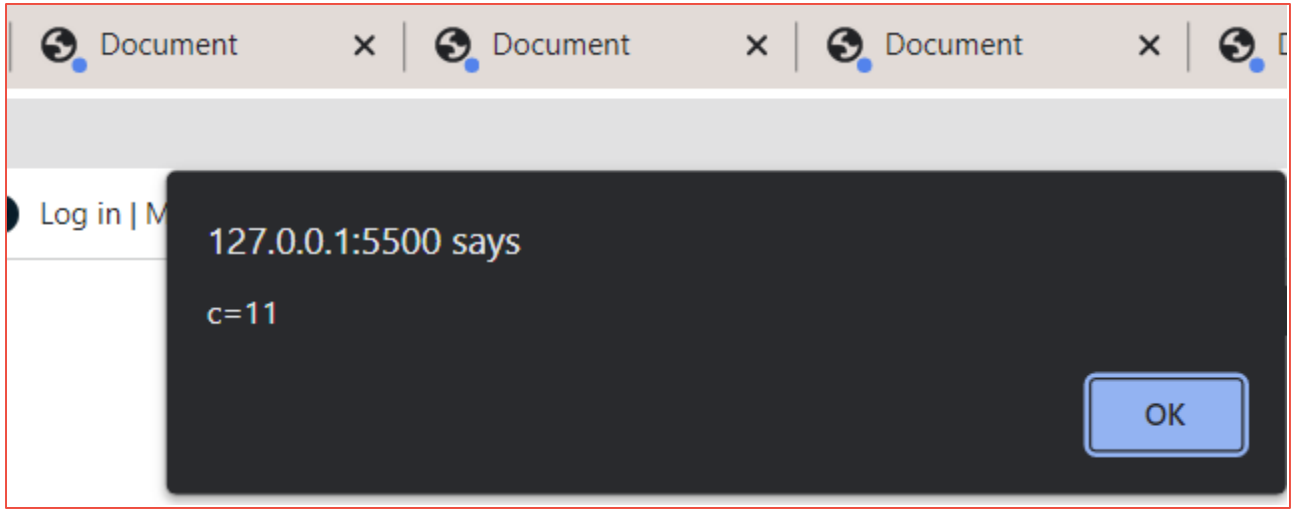
Programming fundamentals of JavaScript

GENERAL SYNTACTIC CHARACTERISTICS

- Identifier form: begin with a letter or underscore, followed by any number of
- letters, underscores, and digits
- Case sensitive
- 25 reserved words, plus future reserved words
- Comments: both // and /* ... */
- Scripts are usually hidden from browsers that do not include JavaScript
- interpreters by putting them in special comments
- <!--
- -- JavaScript script –
- //-->
- Semicolons can be a problem. They are “somewhat” optional.
- Problem : When the end of the line can be the end of a statement ,JavaScript puts a semicolon there
- When separated by semicolons, multiple statements on one line are allowed: a = 5; b = 6; c = a + b;

JavaScript example with alert, No semicolon

```
1   <!DOCTYPE html>
2   <html lang="en">
3   <head>···
9   </head>
10  <body>
11      <script type="text/javascript">
12          var a, b, c;
13          a = 5
14          b = 6
15          c = a + b
16          alert("c=" + c);
17      </script>
18  </body>
19  </html>
```



PRIMITIVES

- JavaScript provides different data types to hold different types of values.

1. Primitive Types
    - five primitive types:
        - Number,
        - String
        - Boolean,
        - Undefined,
        - Null
2. Non-Primitive Types
    - Objects such as functions and arrays are referred to as non-primitive values.

PRIMITIVES

- Primitive Types
- five primitive types: Number, String, Boolean, Undefined, or Null
- Number, String, and Boolean have wrapper objects (Number, String, and Boolean).
- wrapper objects
  - Wrapper objects are used to provide properties and methods that are convenient to use with the values of primitive types.
  - Anything that doesn't belong to any of these five primitive types is considered an object.
  - BUT each of these five primitive data types has a corresponding object constructor..
  - First as a primitive data type:

    var word = "hello";
  - And then as an object:

    var word = new String("hello");

PRIMITIVES and NonPrimitive Values

- The fundamental difference between primitives and non-primitives is that primitives are immutable and non-primitives are mutable.
- Primitives are known as being immutable data types because there is no way to change a primitive value once it gets created.

```
var s = 'This is a string.';
s[1] = 'H';
alert(s)   // 'This is a string.'
```

- Non-primitive values are mutable data types.
- The value of an object can be changed after it gets created.

```
var arr = [ 'one', 'two', 'three', 'four', 'five' ];
arr[1] = 'TWO';
alert(arr)      // [ 'one', 'TWO', 'three', 'four', 'five' ];
```

Numeric & String Literals,Other Primitives

- Numeric & String Literals
  - Numeric literals – like Java All numeric values are stored in double-precision floating point 72,7.2,.72,7E2,7e2 etc are valid
  - String literals are delimited by either ' or "
  - Can include escape sequences (e.g., \t)
  - All String literals are primitive values
- Other Primitive Types
- Boolean values are true and false.Used for evaluating Boolean Expressions
- The only Null value is null-means no value variable has not been explicitly declared or assigned a value.
- The only Undefined value is undefined-variable has been explicitly declared, but not assigned a value
- A variable declared without a value will have the value undefined. ex)var carName;

Declaring variables

- JavaScript is dynamically typed – any variable can be used for anything
- The interpreter determines the type of a particular occurrence of a variable
- Keywords are words that have special meaning in JavaScript.
- The keyword var at the beginning of the statement indicates that the word name is a variable.
- A variable is a location in the computer's memory where a value can be stored for use by a script.
- All variables have a name and value, and should be declared with a var statement before they're used in a script.
- Variables can be implicitly or explicitly declared

  var sum =0,today ="Monday", flag = false; var counter;

  - Variable names are case sensitive , They must begin with a letter or the underscore character

Identifiers and Case Sensitivity

- The name of a variable can be any valid identifier.
- An identifier is a series of characters consisting of letters, digits, underscores (_) and dollar signs ($) that does not begin with a digit and is not a reserved JavaScript keyword.
- Identifiers may not contain spaces.
- Some valid identifiers are Welcome, $value, _value, m_inputField1 and button7.
- The name 7button is not a valid identifier, because it begins with a digit, and the name ‘input field’ is not valid, because it contains a space.
- Remember that JavaScript is case sensitive—uppercase and lowercase letters are considered to be different characters, so name, Name and NAME are different identifiers.

Obtaining User Input with prompt Dialogs

Obtaining User Input with prompt Dialogs

- Scripting gives you the ability to generate part or all of a web page's content at the time it's shown to the user.
- A script can adapt the content based on input from the user or other variables, such as the time of day or the type of browser used by the client. Such web pages are said to be dynamic, as opposed to static, since their content has the ability to change.
- The script uses another predefined dialog box from the window object—a prompt dialog—which allows the user to enter a value that the script can use
- The JavaScript model for the HTML document is the Document object

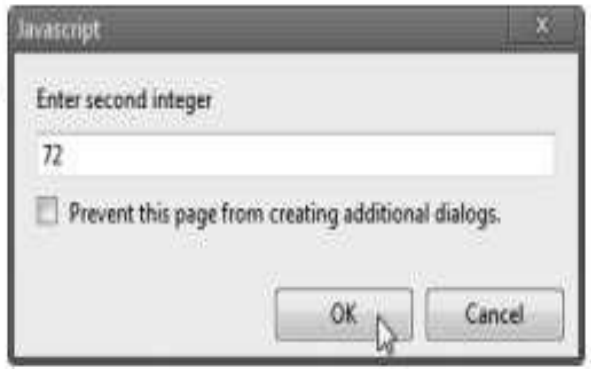
Obtaining User Input with prompt Dialogs

```
<!DOCTYPE html>

<!-- Fig. 6.5: welcome4.html -->
<!-- Prompt box used on a welcome screen -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>Using Prompt and Alert Boxes</title>
      <script type = "text/javascript">
         <!--
         var name; // string entered by the user

         // read the name from the prompt box as a string
         name = window.prompt( "Please enter your name" );

         document.writeln( "<h1>Hello " + name +
            ", welcome to JavaScript programming!</h1>" );
         // -->
      </script>
   </head><body></body>
</html>
```

Adding Integers

```
<!DOCTYPE html>

<!-- Fig. 6.7: addition.html -->
<!-- Addition script. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>An Addition Program</title>
      <script type = "text/javascript">
         <!--
         var firstNumber; // first string entered by user
         var secondNumber; // second string entered by user
         var number1; // first number to add
         var number2; // second number to add
         var sum; // sum of number1 and number2

         // read in first number from user as a string
         firstNumber = window.prompt( "Enter first integer" );

         // read in second number from user as a string
         secondNumber = window.prompt( "Enter second integer" );

         // convert numbers from strings to integers
         number1 = parseInt( firstNumber );
         number2 = parseInt( secondNumber );

         sum = number1 + number2; // add the numbers

         // display the results
         document.writeln( "<h1>The sum is " + sum + "</h1>" );
         // -->
      </script>
   </head><body></body>
</html>
```


Javascript
Enter first integer
45
OK
Cancel


Javascript
Enter second integer
72
Prevent this page from creating additional dialogs.
OK
Cancel


An Addition Program
file://
Other bookmarks
Sync Error
The sum is 117

Memory Concepts

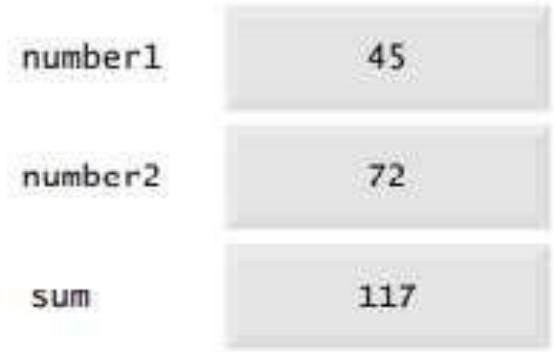
- Variable names such as number1, number2 and sum actually correspond to locations in the computer's memory.
- Every variable has a name, a type and a value
- Suppose the user entered the string 45 as the value for firstNumber. The script converts firstNumber to an integer, and the computer places the integer value 45 into location number1
- Whenever a value is placed in a memory location, the value replaces the previous value in that location.

The previous value is lost.

Memory locations after inputting values for variables **number1** and **number2**

Memory Concepts

- Once the script has obtained values for number1 and number2, it adds the values and places the sum into variable sum.
- JavaScript does not require variables to have a declared type before they can be used in a script.
- A variable in JavaScript can contain a value of any data type, and in many situations JavaScript automatically converts between values of different types for you.
- For this reason, JavaScript is referred to as a loosely typed language



Memory locations after calculating the `sum` of `number1` and `number2`.

Memory Concepts

- Unlike its predecessor languages C, C++ and Java, JavaScript does not require variables to have a declared type before they can be used in a script
- When a variable is declared in JavaScript, but is not given a value, the variable has an undefined value.
- Attempting to use the value of such a variable is normally a logic error.
- When variables are declared, they're not assigned values unless you specify them.
- Assigning the value null to a variable indicates that it does not contain a value.

Operators

Arithmetic Operators

Operator	Description	Example	Result
+	Addition	x=2 y=2 x+y	4
-	Subtraction	x=5 y=2 x-y	3
*	Multiplication	x=5 y=4 x*y	20
/	Division	15/5 5/2	3 2,5
%	Modulus (division remainder)	5%2 10%8 10%2	1 2 0
++	Increment	x=5 x++	x=6
--	Decrement	x=5 x--	x=4

Arithmetic Operators

JavaScript operation	Arithmetic operator	Algebraic expression	JavaScript expression
Addition	+	$f + 7$	`f + 7`
Subtraction	-	$p - c$	`p - c`
Multiplication	*	bm	`b * m`
Division	/	x/y or $\frac{x}{y}$ or $x \div y$	`x / y`
Remainder	%	r mod s	`r % s`

Operator(s)	Operation(s)	Order of evaluation (precedence)
*, / or %	Multiplication Division Remainder	Evaluated first. If there are several such operations, they're evaluated from left to right.
+ or -	Addition Subtraction	Evaluated last. If there are several such operations, they're evaluated from left to right.

Precedence of arithmetic operators.

Arithmetic Operators

Operator	Example	Called	Explanation
++	++a	preincrement	Increment a by 1, then use the new value of a in the expression in which a resides.
++	a++	postincrement	Use the current value of a in the expression in which a resides, then increment a by 1.
--	--b	predecrement	Decrement b by 1, then use the new value of b in the expression in which b resides.
--	b--	postdecrement	Use the current value of b in the expression in which b resides, then decrement b by 1.

Increment and decrement operators.

Operator Precedence

- JavaScript applies the operators in arithmetic expressions uses the following rules of operator precedence
- Multiplication, division and remainder operations are applied first. If an expression contains several multiplication, division and remainder operations, operators are applied from left to right.
- Multiplication, division and remainder operations are said to have the same level of precedence.
- Addition and subtraction operations are applied next. If an expression contains several addition and subtraction operations, operators are applied from left to right.
- Addition and subtraction operations have the same level of precedence.

Assignment Operators

Operator	Example	Is The Same As
=	x=y	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

Comparison Operators(relational &Equality Operators)

Operator	Description	Example
==	is equal to	5==8 returns false
===	is equal to (checks for both value and type)	x=5 y="5" x==y returns true x===y returns false
!=	is not equal	5!=8 returns true
>	is greater than	5>8 returns false
<	is less than	5<8 returns true
>=	is greater than or equal to	5>=8 returns false
<=	is less than or equal to	5<=8 returns true

Logical Operators

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

Logical Operators

expression1	expression2	expression1 || expression2
false	false	false
false	true	true
true	false	true
true	true	true

Truth table for the || (logical OR) operator.

- The && operator has a higher precedence than the || operator.
- Both operators associate from left to right. An expression containing && or || operators is evaluated only until truth or falsity is known.
- the || operator immediately returns true if the first operand is true. This performance feature for evaluation of logical AND and logical OR expressions is called short-circuit evaluation.

String Concatenation Operator

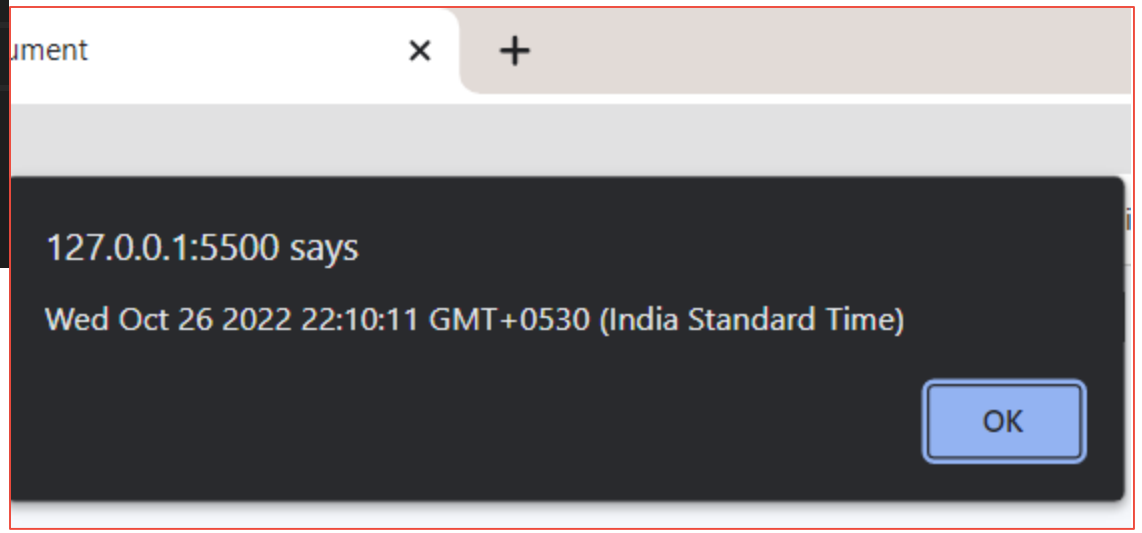
- the + operator used for string concatenation can convert other variable types to strings if necessary.
- Because string concatenation occurs between two strings, JavaScript must convert other variable types to strings before it can proceed with the operation.
- For example, if a variable age has an integer value equal to 21, then the expression "my age is " + age evaluates to the string "my age is 21".
- JavaScript converts the value of age to a string and concatenates it with the existing string literal "my age is ".

Creating and Using a New Date Object

- Create one with the new operator & Date constructor (no params)
- Local time methods of Date:
- toLocaleString– returns a string of the date
- getDate – returns the day of the month
- getMonth – returns the month of the year (0 – 11)
- getDay – returns the day of the week (0 – 6)
- getFullYear– returns the year
- getTime – returns the number of milliseconds since January 1, 1970
- getHours – returns the hour (0 – 23)
- getMinutes– returns the minutes (0 – 59)
- getMilliseconds– returns the milliseconds (0 – 999)

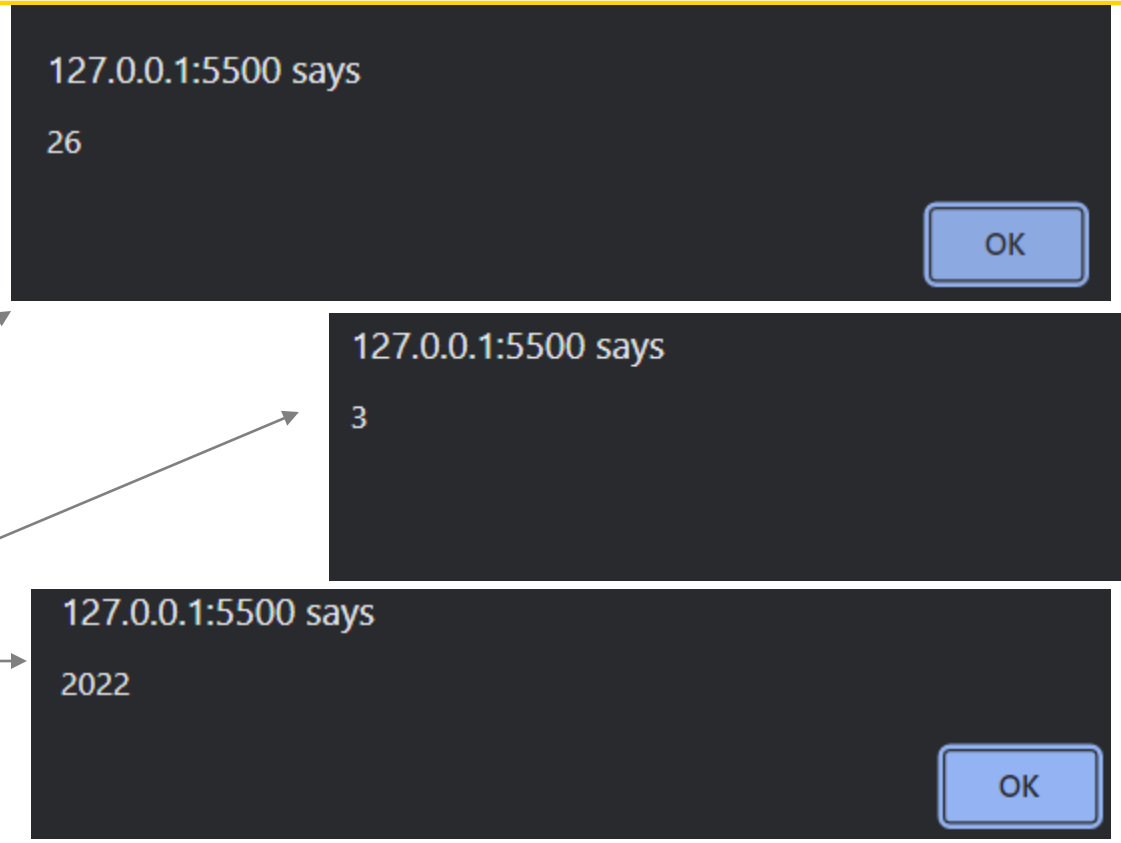
Creating and Using a New Date Object

```
1   <!DOCTYPE html>
2   <html lang="en">
3   <head>…
8   </head>
9   <body>
10      <script type="text/javascript">
11          var today=new Date();
12          alert(today);
13          </script>
14  </body>
15  </html>
```



Creating and Using a New Date Object

```
<script type="text/javascript">
        var today = new Date();
        alert(today);
        alert(today.getDate());
        alert(today.getMonth());
        alert(today.getDay());
        alert(today.getFullYear());
        alert(today.getTime())
</script>
```

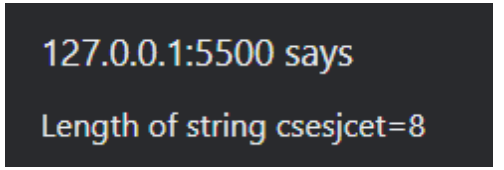


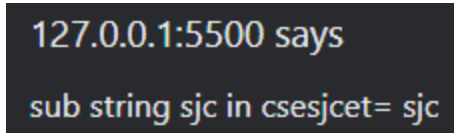
String properties and Methods

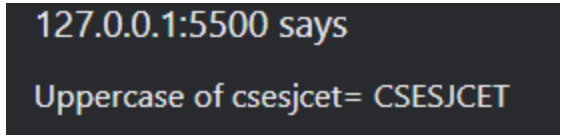
- String properties & methods:
- length e.g., var len = str1.length;
- var str="George";
- var len = str1.length; //len→6
- charAt(position) e.g., str.charAt(3) //r
- indexOf(string) e.g., str.indexOf('o') //2
- substring(from, to) e.g., str.substring(2,4) //org
- toLowerCase() e.g., str.toLowerCase() //george

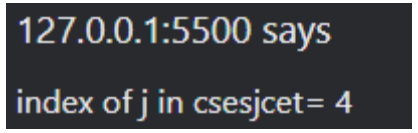
String properties and Methods

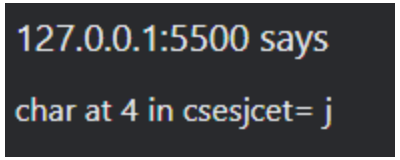
```
<script type="text/javascript">

    var str = "csesjcet";
    alert("Length of string csesjcet=" + str.length);
    alert("sub string sjc in csesjcet= " + str.substring(3, 6));
    alert("Uppercase of csesjcet= " + str.toUpperCase());
    alert("index of j in csesjcet= " + str.indexOf('j'));
    alert("char at 4 in csesjcet= " + str.charAt(4));

</script>
```











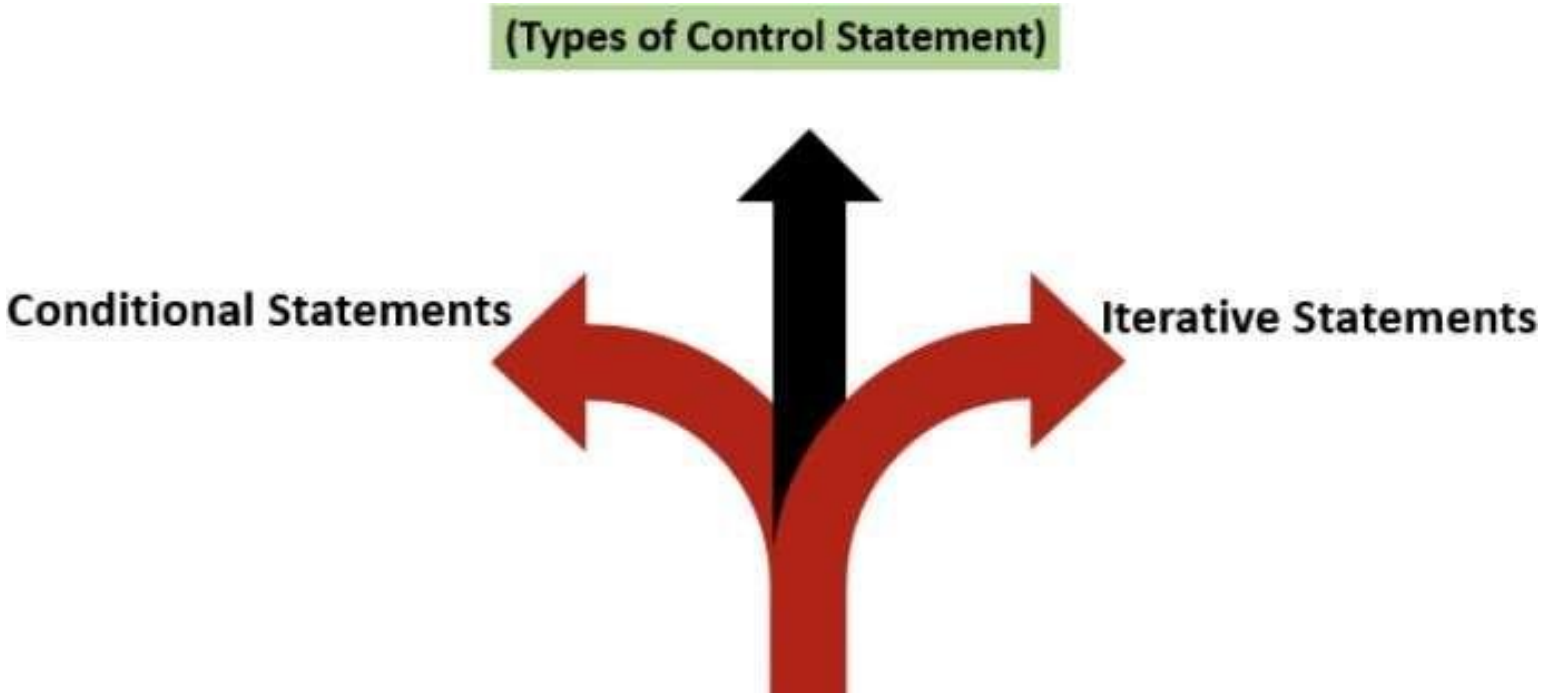
Precedence and Associativity

Operator	Associativity	Type
++ -- !	right to left	unary
* / %	left to right	multiplicative
+ -	left to right	additive
< <= > >=	left to right	relational
== != === !==	left to right	equality
&&	left to right	logical AND
||	left to right	logical OR
?:	right to left	conditional
= += -= *= /= %=	right to left	assignment

Precedence and associativity of the operators discussed so far.

Control Statements

Control statement



Decision Making

Conditional statement

- if statement - use this statement if you want to execute some code only if a specified condition is true
- if...else statement - use this statement if you want 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 if you want to select one of many blocks of code to be executed
- switch statement - use this statement if you want to select one of many blocks of code to be executed .

Conditional statement

- if (condition)

  {

  code to be executed if condition is true

  }
- JavaScript's if statement that allows a script to make a decision based on the truth or falsity of a condition.
- If the condition is met (i.e., the condition is true), the statement in the body of the if statement is executed.
- If the condition is not met (i.e., the condition is false), the statement in the body of the if statement is not executed.
- conditions in if statements can be formed by using the equality operators and relational operators

Conditional statement

- if (condition)

  {

  code to be executed if condition is true

  }
- JavaScript's if statement that allows a script to make a decision based on the truth or falsity of a condition.
- If the condition is met (i.e., the condition is true), the statement in the body of the if statement is executed.
- If the condition is not met (i.e., the condition is false), the statement in the body of the if statement is not executed.
- conditions in if statements can be formed by using the equality operators and relational operators

```
<!DOCTYPE html>

<!-- Fig. 6.14: welcome5.html -->
<!-- Using equality and relational operators. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>Using Relational Operators</title>
      <script type = "text/javascript">
         <!--
         var name; // string entered by the user
         var now = new Date();       // current date and time
         var hour = now.getHours(); // current hour (0-23)

         // read the name from the prompt box as a string
         name = window.prompt( "Please enter your name" );
```

```
      // determine whether it's morning
      if ( hour < 12 )
         document.write( "<h1>Good Morning, " );

      // determine whether the time is PM
      if ( hour >= 12 )
      {
         // convert to a 12-hour clock
         hour = hour - 12;

         // determine whether it is before 6 PM
         if ( hour < 6 )
            document.write( "<h1>Good Afternoon, " );

         // determine whether it is after 6 PM
         if ( hour >= 6 )
            document.write( "<h1>Good Evening, " );
      } // end if

      document.writeln( name +
         ", welcome to JavaScript programming!</h1>" );
      // -->
   </script>
</head><body></body>
</html>
```

IF ELSE

- if (condition)

  { code to be executed if condition is true

  }

  else

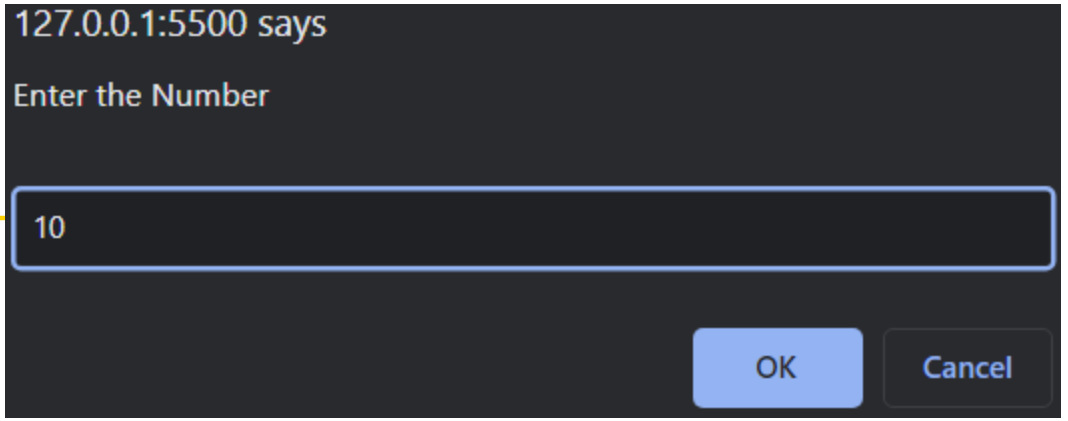
  { code to be executed if condition is not true

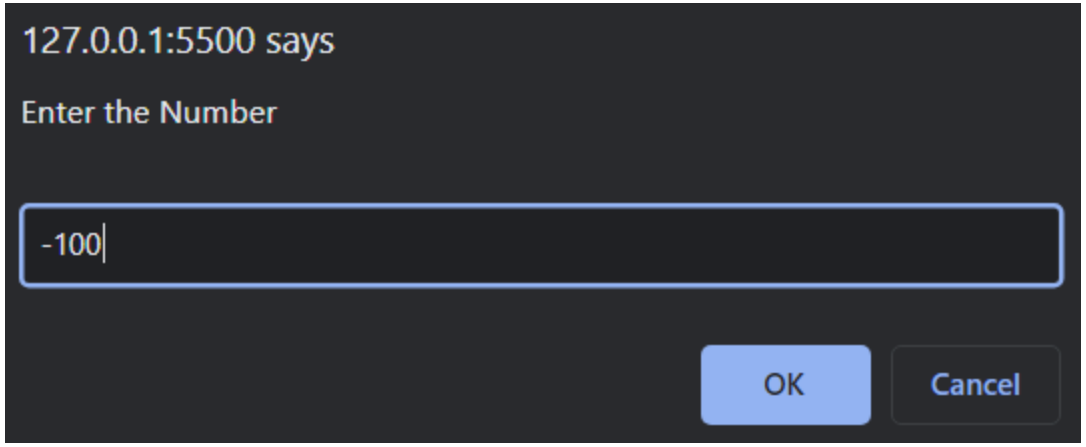
  }

- JavaScript's if statement that allows a script to make a decision based on the truth or falsity of a condition.
- If the condition is met (i.e., the condition is true), the statement in the body of the if statement is executed.
- If the condition is not met (i.e., the condition is false), the statement in the body of the else block is executed.

IF ELSE

```
<script type="text/javascript">
    var x = prompt("Enter the Number \n", "");
    if (x < 0) {
        alert("negative");
    }
    else {
        alert("positive");
    }
</script>
```





ELSE IF STATEMENT

- Use the else if statement to specify a new condition if the first condition is false.
- Syntax

```
if (condition1)
 { block of code to be executed if condition1 is true }
else if (condition2)
 { block of code to be executed if the condition1 is false and condition2 is true }
else
 { block of code to be executed if the condition1 is false and condition2 is false }
```

Conditional Operator (?:)

- JavaScript provides an operator, called the conditional operator (?:), that's closely related to the if...else statement.
- The operator ?: is JavaScript's only ternary operator—it takes three operands. The operands together with the ?: form a conditional expression.
- The first operand is a Boolean expression, the second is the value for the conditional expression if the expression evaluates to true and the third is the value for the conditional expression if the expression evaluates to false.

```
document.writeln( studentGrade >= 60 ? "Passed" : "Failed" );
```

- A conditional expression that evaluates to the string "Passed" if the condition studentGrade >= 60 is true and evaluates to the string "Failed" if the condition is false

Nested if else

```
If student's grade is greater than or equal to 90
    Print "A"
Else
    If student's grade is greater than or equal to 80
        Print "B"
    Else
        If student's grade is greater than or equal to 70
            Print "C"
        Else
            If student's grade is greater than or equal to 60
                Print "D"
            Else
                Print "F"
```

```
if ( studentGrade >= 90 )
   document.writeln( "A" );
else
   if ( studentGrade >= 80 )
      document.writeln( "B" );
   else
      if ( studentGrade >= 70 )
         document.writeln( "C" );
      else
         if ( studentGrade >= 60 )
            document.writeln( "D" );
         else
            document.writeln( "F" );
```

Nested if else

```
<!-- Arranging 3 numbers in order -->
<!doctype html>
<html lang="en">
<head>
 <title>nested if statement</title>
</head>
<body>
 <script language="javascript" type="text/javascript" >
 var a = parseInt(window.prompt("Enter Marks Sub1"));
 var b= parseInt(prompt("Enter Marks Sub2"));
 var c= parseInt(prompt("Enter Marks Sub3"));
 var avg=(a+b+c)/3;
 if(avg >= 70)
    alert("First With Distinction");
 else if(avg >=60 && avg <70)
    alert("First Class");
 else if(avg >=50 && avg <60)
    alert("Second Class");
 else if(avg >=40 && avg <50)
    alert("Third Class");
 else
    alert("Fail");
 </script>
</body>
</html
```

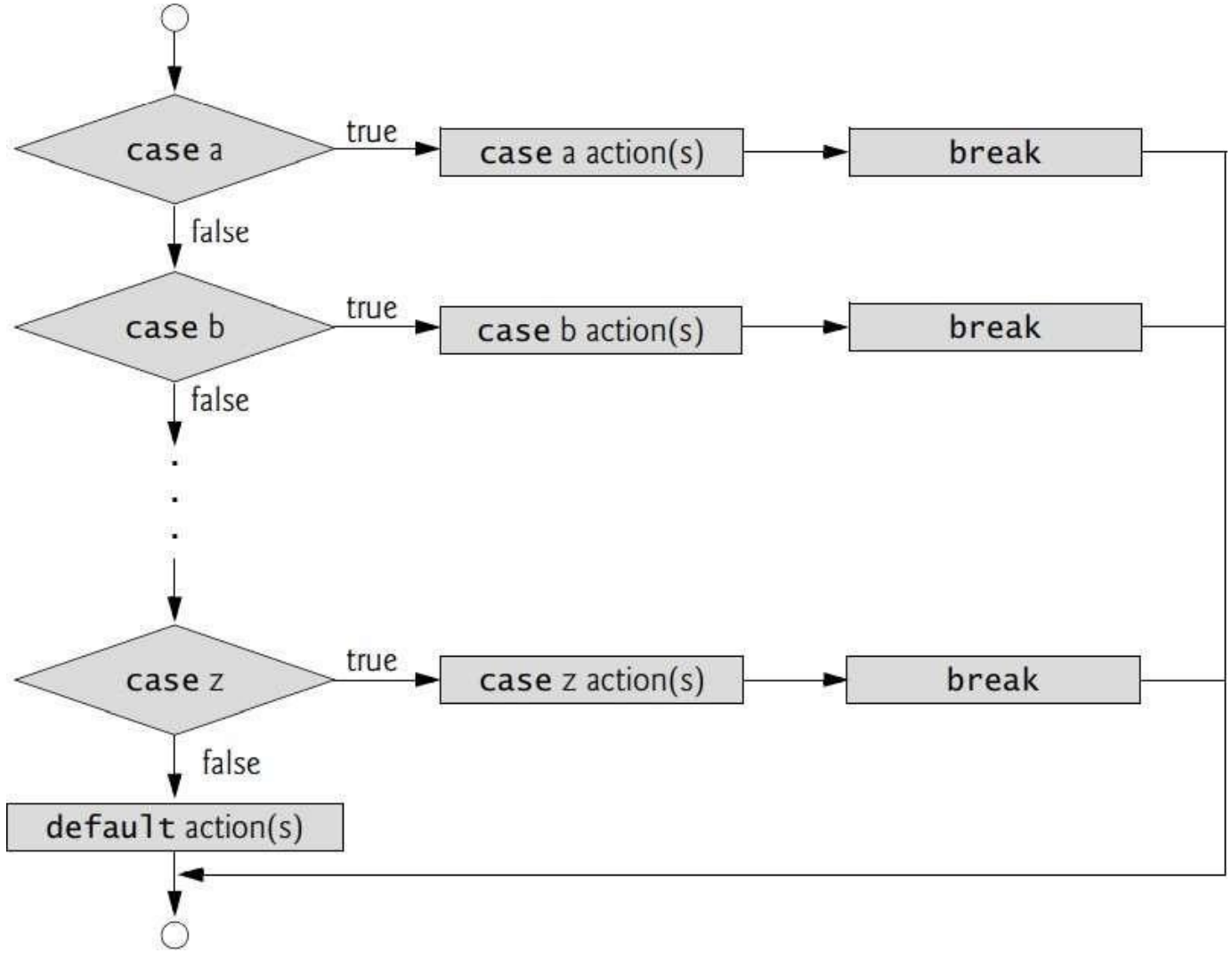
Switch statements

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

```
switch(expression)
        { case 1: code block
                  break; .....
        case n: code block
                  break;
        default: code block
}
```

- The switch expression is evaluated once.
- The value of the expression is compared with the values of each case.
- If there is a match, the associated block of code is executed

Switch statements

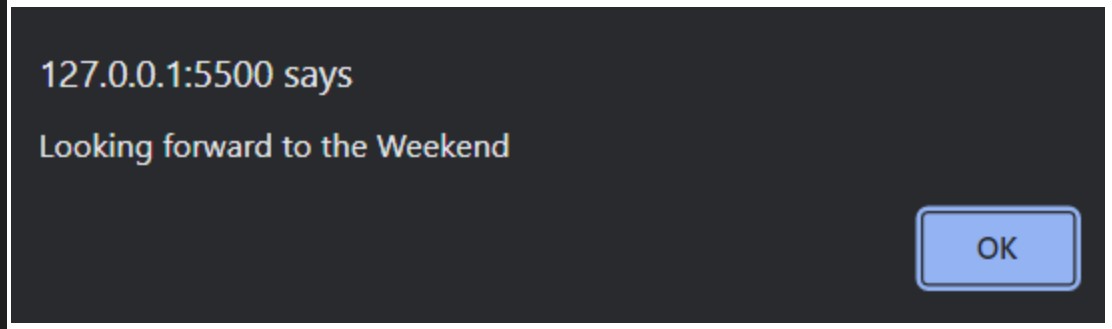


Switch statements


```
<script>
    var day = new Date().getDay();
    switch (day) {
        case 0:
            day = "Sunday";
            break;
        case 1:
            day = "Monday";
            break;
        case 2:
            day = "Tuesday";
            break;
        case 3:
            day = "Wednesday";
            break;
        case 4:
            day = "Thursday";
            break;
        case 5:
            day = "Friday";
            break;
        case 6:
            day = "Saturday";
    }
    alert(day);
</script>
```

Switch statements

```
<script>
    var day = new Date().getDay();
    switch (day) {
    case 4:
    case 5:
    text = "Soon it is Weekend";
    break;
    case 0:
    case 6:
    text = "It is Weekend";
    break;
    default:
    text = "Looking forward to the Weekend";
    }
    alert(text);
    </script>
```



Switch statements

```
<script language="javascript" type="text/javascript">
var n1 = parseInt(window.prompt("Enter the marks in Sub1"));
var n2 = parseInt(window.prompt("Enter the marks in Sub2"));
var n3 = parseInt(window.prompt("Enter the marks in Sub3"));
var avg= (n1+n2+n3)/3;

var ch = parseInt(avg/10); //parse to int

switch(ch)
{
case 10:
case 9:
case 8:
case 7: window.alert("Distinction");break;
case 6: window.alert("First Class");break;
case 5: window.alert("Second Class");break;
case 4: window.alert("Third Class ");break;
default: window.alert("Fail ");
}</script>
```

Loop Statements

Loop statements

- Different Kinds of Loops
- JavaScript supports different kinds of loops:
- for - loops through a block of code a number of times
  - For a loop, when you execute a loop for a fixed number of times, the loop does three specified tasks (Statement 1, Statement 2, Statement 3)
- for/in - loops through the properties of an object
- while - loops through a block of code while a specified condition is true
- do/while - also loops through a block of code while a specified condition is true
  - It will execute at least once even if the condition is to be not satisfied. If the condition is true, it will continue executing and once the condition becomes false, it stops the code execution.

For Loop statements

- for (statement 1; statement 2; statement 3)
  {
  code block to be executed
  }
- Statement 1 is executed before the loop (the code block) starts.
- Statement 2 defines the condition for running the loop (the code block).
- Statement 3 is executed each time after the loop (the code block) has been executed.

Example

```
for (i = 0; i < 5; i++)
 {
 text += "The number is " + i + "";
alert(text);
}
```

Examples using For Loop statements

a) Vary the control variable from 1 to 100 in increments of 1.

```
for ( var i = 1; i <= 100; ++i )
```

b) Vary the control variable from 100 to 1 in increments of -1 (i.e., *decrements* of 1).

```
for ( var i = 100; i >= 1; --i )
```

c) Vary the control variable from 7 to 77 in steps of 7.

```
for ( var i = 7; i <= 77; i += 7 )
```

d) Vary the control variable from 20 to 2 in steps of -2.

```
for ( var i = 20; i >= 2; i -= 2 )
```

Examples using For Loop statements

```
<!DOCTYPE html>
<html>
   <head>
      <meta charset = "utf-8">
      <title>Sum the Even Integers from 2 to 100</title>
      <script>

         var sum = 0;

         for ( var number = 2; number <= 100; number += 2 )
            sum += number;

         document.writeln( "The sum of the even integers " +
            "from 2 to 100 is " + sum );
      </script>
   </head><body></body>
</html>
```

Consider the following problem statement:

A person invests $1000.00 in a savings account yielding 5 percent interest. Assuming that all the interest is left on deposit, calculate and print the amount of money in the account at the end of each year for 10 years. Use the following formula to determine the amounts:

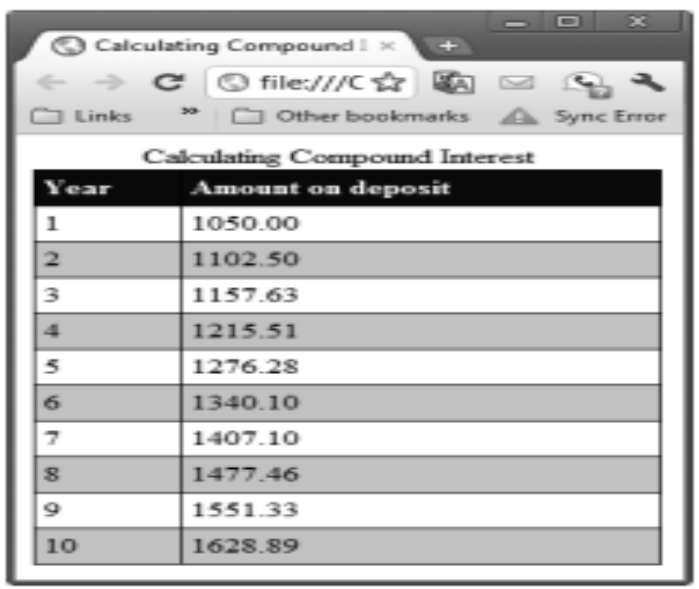
$$a = p\,(1 + r)^{n}$$

where

p is the original amount invested (i.e., the principal)
r is the annual interest rate
n is the number of years
a is the amount on deposit at the end of the nth year.

```
<!DOCTYPE html>

<!-- Fig. 8.6: Interest.html -->
<!-- Compound interest calculation with a for loop. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>Calculating Compound Interest</title>
      <style type = "text/css">
         table           { width: 300px;
                           border-collapse: collapse;
                           background-color: lightblue; }
         table, td, th { border: 1px solid black;
                           padding: 4px; }
         th              { text-align: left;
                           color: white;
                           background-color: darkblue; }
         tr.oddrow       { background-color: white; }
      </style>
      <script>

         var amount; // current amount of money
         var principal = 1000.00; // principal amount
         var rate = 0.05; // interest rate
```

```
         document.writeln("<table>" ); // begin the table
         document.writeln(
            "<caption>Calculating Compound Interest</caption>" );
         document.writeln(
            "<thead><tr><th>Year</th>" ); // year column heading
         document.writeln(
            "<th>Amount on deposit</th>" ); // amount column heading
         document.writeln( "</tr></thead><tbody>" );

         // output a table row for each year
         for ( var year = 1; year <= 10; ++year )
         {
            amount = principal * Math.pow( 1.0 + rate, year );

            if ( year % 2 !== 0 )
               document.writeln( "<tr class='oddrow'><td>" + year +
                  "</td><td>" + amount.toFixed(2) + "</td></tr>" );
            else
               document.writeln( "<tr><td>" + year +
                  "</td><td>" + amount.toFixed(2) + "</td></tr>" );
         } //end for

         document.writeln( "</tbody></table>" );

      </script>
   </head><body></body>
</html>
```



Calculating Compound Interest

Year	Amount on deposit
1	1050.00
2	1102.50
3	1157.63
4	1215.51
5	1276.28
6	1340.10
7	1407.10
8	1477.46
9	1551.33
10	1628.89

For..in statements

- The JavaScript for/in statement loops through the properties of an object:
- Example

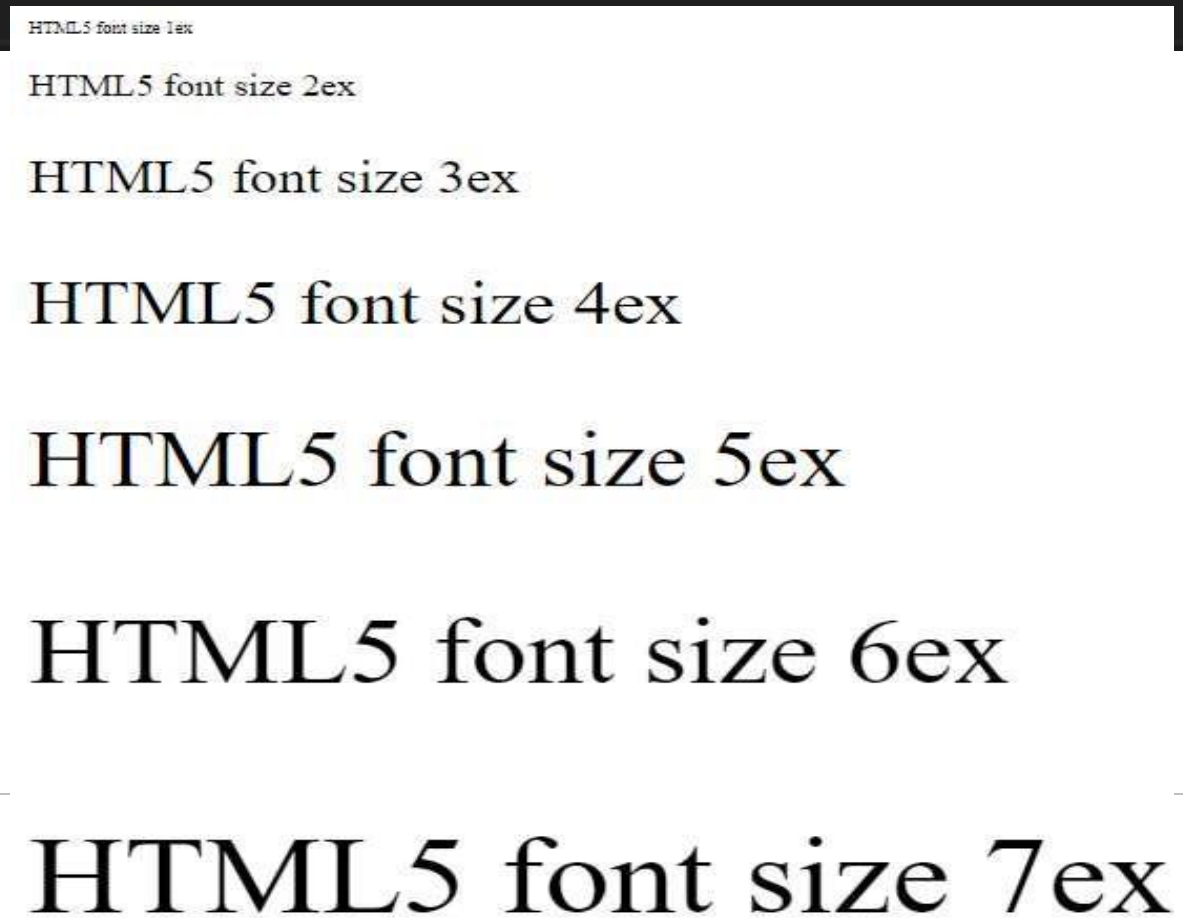
```
<script type="text/javascript">
    var x;
    var cars =["Maruthi","Toyoto",  "BMW"];
    for (x in cars)
    {
    alert(x+ "="+ cars[x]);
    }
</script>
```

While loop statements

- The while loop loops through a block of code as long as a specified condition is true.
  while (condition)
  { code block to be executed }

```
<script type="text/javascript">
    var i=0;
    while (i < 10)
    {
    alert("i="+i);
    i++;
    }
</script>
```

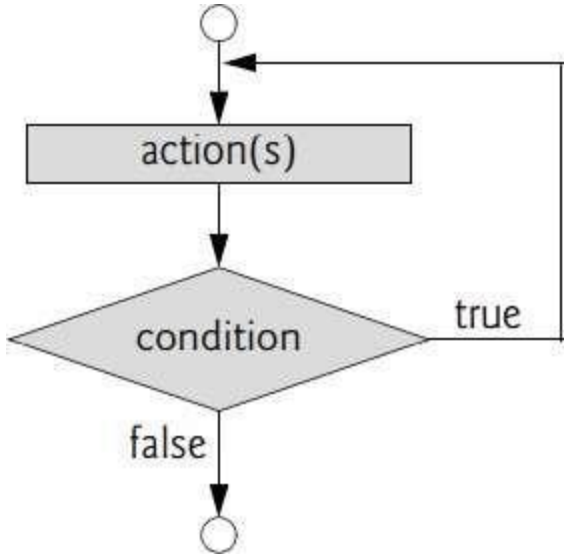
```
<script type="text/javascript">

    var counter = 1;
    // initialization
    while (counter <= 7) // repetition condition
    {
        document.writeln("<p style = 'font-size: " + counter + "ex'>HTML5 font size " + counter + "ex</p>");
        ++counter; // incremen
    }
</script>
```



Do ..While loop statements

- The do/while loop is a variant of the while loop.
- This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.
- Syntax

```
do
 { code block to be executed
 }
 while (condition);
```




```
<script>
var i=0;
do
{
alert("i="+i);
i++;
}while (i <= 3)
</script>
```

Break statements

- The break statement "jumps out" of a loop.
- It was used to "jump out" of a switch() statement
- break statement also be used to jump out of a loop.
- The break statement breaks the loop and continues executing the code after the loop (if any):
- Example

```
<script type="text/javascript">
var i,text="";
for (i = 0; i < 10; i++)
{
if (i === 3)
        { break; }
text += "The number is " + i;
        alert(text);
}
</script>
```

continue statement

- The continue statement "jumps over" one iteration in the loop.
- The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.
- This example skips the value of 3

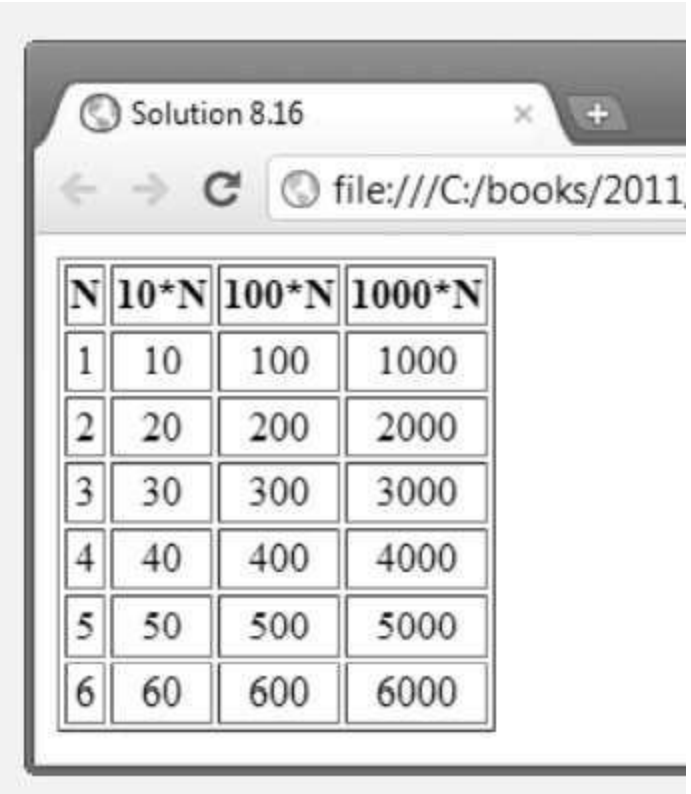
```
<script type="text/javascript">
var i,text="";
for (i = 0; i < 10; i++)
{
        if (i === 3)
         { continue; }
        text += "The number is " + i;
}
        alert(text);
</script>
```

Tutorial Questions

- A palindrome is a number or a text phrase that reads the same backward and forward. For example, each of the following five-digit integers is a palindrome: 12321, 55555, 45554 and 11611. Write a script that reads in a five-digit integer and determines whether it's a palindrome. If the number is not five digits long, display an alert dialog indicating the problem to the user. Allow the user to enter a new value after dismissing the alert dialog.
- Develop a script that will determine the gross pay for each of three employees. The company pays "straight time" for the first 40 hours worked by each employee and pays "time and a half" for all hours worked in excess of 40 hours. You're given a list of the employees of the company, the number of hours each employee worked last week and the hourly rate of each employee. Your script should input this information for each employee, determine the employee's gross pay and output HTML5 text that displays the employee's gross pay. Use prompt dialogs to input the data.

Tutorial Questions

- Write a script that uses looping to print the following table of values. Output the results in an HTML5 table. Use CSS to center the data in each column.



N	10*N	100*N	1000*N
1	10	100	1000
2	20	200	2000
3	30	300	3000
4	40	400	4000
5	50	500	5000
6	60	600	6000

Tutorial Questions

```
<!DOCTYPE html>

<html>
   <head>
      <meta charset = "utf-8">
      <title>Mystery Script</title>
      <script type = "text/javascript">
      <!--
         var y;
         var x = 1;
         var total = 0;

         while ( x <= 15 )
         {
            y = x * x * x;
            document.writeln( "<p>" + y + "</p>" );
            total += y;
            ++x;
         } // end while

         document.writeln( "<p>Total is " + total + "</p>" );
         //-->
      </script>
   </head><body></body>
</html>
```

Predict the output??

Functions

Functions

- function function_name([formal_parameters])

  { -- body – }

- Return value is the parameter of return ,If there is no return, or if the end of the function is reached, undefined is returned .If return has no parameter, undefined is returned
- Function call -Functions are invoked by writing the name of the function, followed by a left parenthesis, followed by a comma-separated list of zero or more arguments, followed by a right parenthesis
- If fun is the name of a function,

```
var a=fun();
        or
fun();    /* A call to fun */
```

Functions-example

```
// function returns a value
<script>
function myFunction(a, b)
 {
    return a * b;
}
var x = myFunction(4, 3);
alert(x);
</script>
```

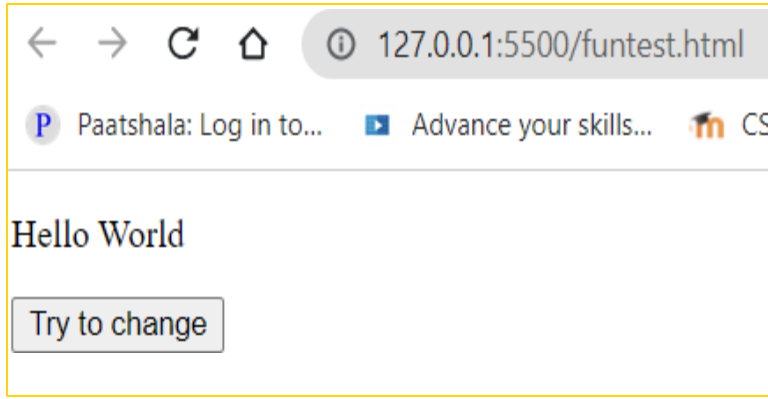
```
//invoking a function as a
function
<script>
function myFunction(a, b)
                {
            return a * b;
                }
alert(myFunction(10, 2));
</script>
```

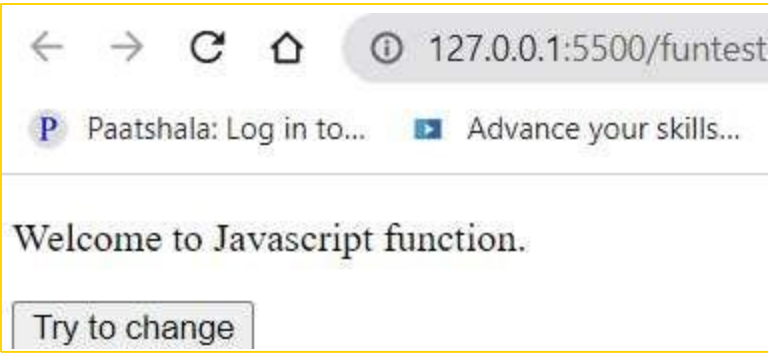
Functions

- The parameter values that appear in a call to a function are called actual parameters.
- The parameter names that appear in the header of a function definition, which correspond to the actual parameters in calls to the function, are called formal parameters.
- JavaScript uses the pass-by-value parameter-passing method.
- When a function is called, the values of the actual parameters specified in the call are copied into their corresponding formal parameters, which behave exactly like local variables.

Functions

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script>
        function test() {
            document.getElementById("hello").innerHTML = "Welcome to Javascript function.";
        }
    </script>
</head>
<body>
    <p id="hello">Hello World</p>
    <input type="button" onclick="test()" value="Try to change" />
</body>
</html>
```





Functions


Javascript Functions
Name: Smitha
Age: 42
submit
127.0.0.1:5500 says
Hello Smitha You are 42Yrs old
OK

Functions

```
<body>
    <p>Javascript Functions</p>
    <form name="form1" method="post" action="funtest2.html" >
        Name:<input type="text" name="fname" />
        <br />Age:<input type="text" name="age" />
        <br /><input type="submit" name="submit" value="submit" onclick="greeting()" />
        <br />
    </form>
    <script type="text/javascript">
        function greeting()
        {//document.getElementById("fname").value or var
            var name = document.form1.fname.value;
             var age = document.form1.age.value;
              alert('Hello ' + name + " You are " + age + "Yrs old");
              document.write('Hello ' + name + " You are " + age + "Yrs old")
        }
    </script>
</body>
```

Functions

- Scripts can also be placed in external files:
- External file:// myScript.js

```
function test()
    {
    document.getElementById("demo").innerHTML = “welcome to Javascript.";
    }
```

- External scripts are practical when the same code is used in many different web pages.
- JavaScript files have the file extension .js.
- To use an external script, put the name of the script file in the src (source) attribute of a <script> tag and embed in the html doc:

```
<script src="myScript.js"></script>
```

Compute the roots of a quadratic equation -example

```
//If the roots are imaginary, this script displays NaN
var a = prompt("Find the solution for ax^2 + bx +c \n 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", "");
document.write("a="+ a+"<br />");
document.write("b="+b+"<br />");
document.write("c="+c+"<br />");
// Compute the square root and denominator of the result
var root_part = Math.sqrt(b * b - 4.0 * a * c);
var denom = 2.0 * a;
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 />");
```

Example-Maximum of three numbers using user defined function

```
<body>
    <script type="text/javascript">

      var no1 = window.prompt("No 1","");
      var no2 = window.prompt("No 2","");
      var no3 = window.prompt("No 3","");
      var x=parseFloat(no1);
      var y=parseFloat(no2);
      var z=parseFloat(no3);
      var maxValue = maximum( x, y, z);
           document.writeln( "<p>First number: " + x + "</p>" +
           "<p>Second number: " + y + "</p>" +"<p>Third number: " + z + "</p>"
           +  "<p>Maximum is: " + maxValue + "</p>" )
    function maximum( x, y, z )
    {
     return Math.max( x, Math.max( y, z ) );
    }
</script>
</body>
```

Example-Random Number Generation

- Consider the following statement:

```
var randomValue = Math.random();
```

- Method random generates a floating-point value from 0.0 up to, but not including, 1.0.
- If random truly produces values at random, then every value in that range has an equal chance (or probability) of being chosen each time random is called.
- Math.floor determine the closest integer not greater than the argument's value—for example, Math.floor(1.75) is 1 and Math.floor(6.75) is 6

```
Math.floor( 1 + Math.random() * 6 )
```

- The preceding expression multiplies the result of a call to Math.random() by 6 to produce a value from 0.0 up to, but not including, 6.0. This is called scaling the range of the random numbers.

Example-Random Number Generation

```
<script type="text/javascript">
    var value;
    document.writeln("<p>Random Numbers</p><ol>");
    for (var i = 1; i <= 10; ++i) {
        value = Math.floor(1 + Math.random() * 6)

        document.writeln("<li>" + value + "</li>");
    } // end for
    document.writeln("</ol>");
</script>
```

Example Linear search

```
<script type="text/javascript">
    function Search()
    {  var a = new Array(50); // create an array
       // fill array with even integer values from 0 to 48
       for (var i = 0; i < a.length; ++i) {
        a[i] = 2 * i;
        document.getElementById("list1").innerHTML+=a[i]+" ";
       }
       var inputVal = document.getElementById("inputVal");
        var searchKey = parseInt(inputVal.value);
        var element = a.indexOf(searchKey);
        //The indexOf() method returns the first index (position) of a specified value.
        // get the result paragraph
        var result = document.getElementById("result");
        // get the search key from the input text field then perform the search
        if (element != -1) {
            result.innerHTML = "Found value in element " + element;
        } // end if
        else {
            result.innerHTML = "Value not found";
        }
      }  </script></head><body>
<form action="#" onsubmit="Search()">
    <label>Enter integer search key:
        <input id="inputVal" type="number"></label>
        <input id="searchButton" type="submit" value="Search">
        <p id="list1"></p>
    <p id="result"></p>
</form></body></html>
```

Enter integer search key: 24 Search

0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42

44 46 48 50 52 54 56 58 60 62 64 66 68 70 72

74 76 78 80 82 84 86 88 90 92 94 96 98

Found value in element 12

Example of function and array

```
<script type = "text/javascript"  >
function fun1 (my_list)
{
        alert(my_list);    //2,4,6,8
        var list2 = new Array(1, 3, 5);
        my_list[3] = 14;
        alert(my_list);  //2,4,6,14
        my_list = list2;
        alert(my_list);  //1,3,5
}
var list = new Array(2, 4, 6, 8);
fun1 (list);
   </script>
```

Global functions

- JavaScript provides nine standard global functions.
- We've already used parseInt, parseFloat and isFinite.
- The global functions are all part of JavaScript's Global object.
- The Global object contains all the global variables in the script, all the user-defined functions in the script and all the functions. Because global functions and userdefined functions are part of the Global object, some JavaScript programmers refer to these functions as methods.
- You do not need to use the Global object directly—JavaScript references it for you

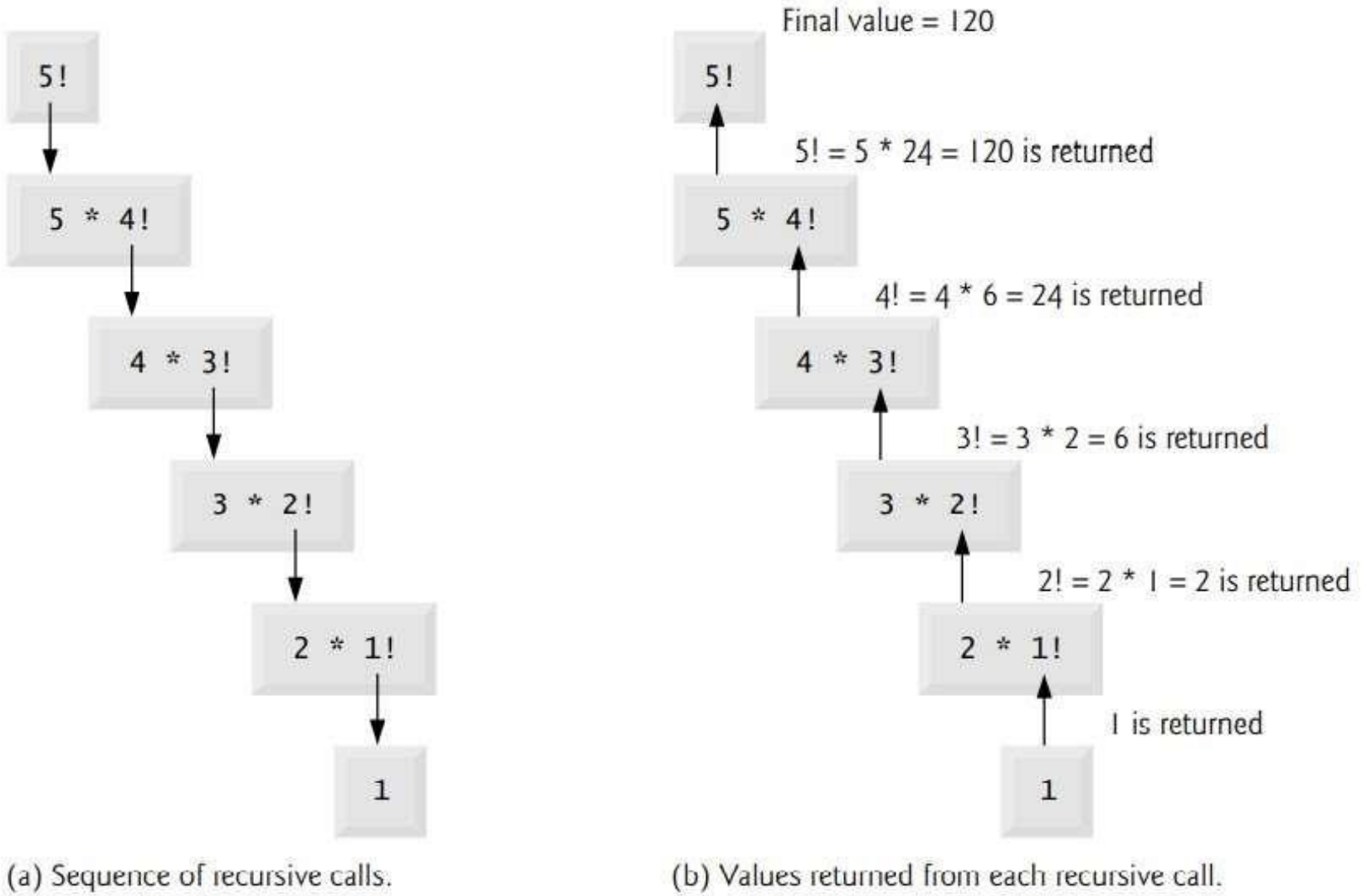
Global functions

Global function	Description
`isFinite`	Takes a numeric argument and returns `true` if the value of the argument is not `NaN`, `Number.POSITIVE_INFINITY` or `Number.NEGATIVE_INFINITY` (values that are not numbers or numbers outside the range that JavaScript supports)—otherwise, the function returns `false`.
`isNaN`	Takes a numeric argument and returns `true` if the value of the argument is not a number; otherwise, it returns `false`. The function is commonly used with the return value of `parseInt` or `parseFloat` to determine whether the result is a proper numeric value.
`parseFloat`	Takes a string argument and attempts to convert the *beginning* of the string into a floating-point value. If the conversion is unsuccessful, the function returns `NaN`; otherwise, it returns the converted value (e.g., `parseFloat( "abc123.45" )` returns `NaN`, and `parseFloat( "123.45abc" )` returns the value `123.45`).
`parseInt`	Takes a string argument and attempts to convert the beginning of the string into an integer value. If the conversion is unsuccessful, the function returns `NaN`; otherwise, it returns the converted value (for example, `parseInt( "abc123" )` returns `NaN`, and `parseInt( "123abc" )` returns the integer value `123`). This function takes an optional second argument, from 2 to 36, specifying the **radix** (or **base**) of the number. Base 2 indicates that the first argument string is in **binary** format, base 8 that it's in **octal** format and base 16 that it's in **hexadecimal** format. See Appendix E, for more information on binary, octal and hexadecimal numbers.

Recursion

- A recursive function is a function that calls itself, either directly, or indirectly through another function.
- The recursion step executes while the original call to the function is still open (i.e., it has not finished executing).
- The recursion step can result in many more recursive calls as the function divides each new subproblem into two conceptual pieces.
- For the recursion eventually to terminate, each time the function calls itself with a simpler version of the original problem, the sequence of smaller and smaller problems must converge on the base case.

Recursive Function -Factorial



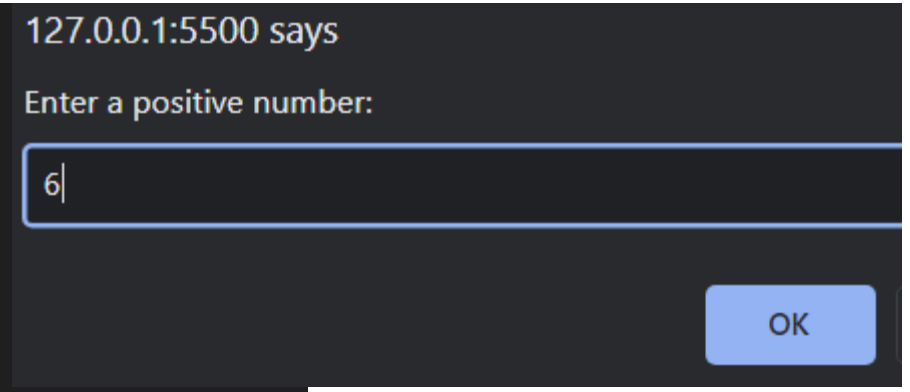
(a) Sequence of recursive calls.

(b) Values returned from each recursive call.

Recursive evaluation of 5!.

Recursive Function -Factorial

```
12      <script type="text/javascript">
13          function factorial(x) {
14              // if number is 0
15              if (x == 0) {
16                  return 1;
17              }
18              // if number is positive
19              else {
20                  return x * factorial(x - 1);
21              }
22          }
23          const num = prompt('Enter a positive number: ');
24          if (num >= 0) {
25              const result = factorial(num);
26              document.write("The factorial of" + num + " is " + result);
27          }
28      </script>
```



Recursion Vs Iteration

- Both iteration and recursion are based on a control statement:
  - Iteration uses a repetition statement (e.g., for, while or do…while);
  - recursion uses a selection statement (e.g., if, if…else or switch).
- Both iteration and recursion involve repetition:
  - Iteration explicitly uses a repetition statement;
  - recursion achieves repetition through repeated function calls.
- Iteration and recursion each involve a termination test:
  - Iteration terminates when the loop-continuation condition fails;
  - recursion terminates when a base case is recognized.
- Both iteration and recursion can occur infinitely:
  - An infinite loop occurs with iteration if the loop-continuation test never becomes false;
  - infinite recursion occurs if the recursion step does not reduce the problem each time via a sequence that converges on the base case or if the base case is incorrect.

ARRAYS

ARRAYS

- Arrays are data structures consisting of related data items.
- JavaScript arrays are “dynamic” entities in that they can change size after they’re created.
- An array is a group of memory locations that all have the same name and normally are of the same type (although this attribute is not required in JavaScript).
- To refer to a particular location or element in the array, we specify the name of the array and the position number of the particular element in the array
- The first element in every array is the zeroth element.
- Thus, the first element of array c is referred to as c[0], the second element as c[1].
- The position number in square brackets is called an index and must be an integer or an integer expression.
- If a program uses an expression as an index, then the expression is evaluated to determine the value of the index.

ARRAYS

- JavaScript arrays are used to store multiple values in a single variable.
- Example

```
var cars = [“Maruthi",“Toyoto","BMW"];
```

- An array can hold many values under a single name,
- Access the values of array by referring to an index number.
- Creating an Array Using an array literal we can create a JavaScript Array.

Syntax: var array_name = [item1, item2, ...];

- The following example also creates an Array, and assigns values to it:
- Example

```
var s = new Array(“AA", ”BB",“CC”);
var list = new Array(2, 4, 6, 8);
```

```
var n = [ 10, 20, 30, 40, 50 ];
```

- Uses a comma-separated initializer list enclosed in square brackets ([and]) to create a five element array with indices of 0, 1, 2, 3 and 4.
- The array size is determined by the number of values in the initializer list.

```
var n = new Array( 10, 20, 30, 40, 50 );
```

- Also creates a five-element array with indices of 0, 1, 2, 3 and 4.
- In this case, the initial values of the array elements are specified as arguments in the parentheses following new Array.
- The size of the array is determined by the number of values in parentheses

```
var n = [ 10, 20, , 40, 50 ];
```

- creates a five-element array in which the third element (n[2]) has the value undefined

Accessing elements of ARRAYS

- An array element can be accessed by referring to the index number.
- To access the value of the first element in s:

```
var name = s[0];
```

```
<script>
        var s = ["AA", "BB", "CC"];
        alert(s[0]+" "+s[1]+" "+s[2]);
</script>
```

- The length property of an array returns the length of an array (the number of array elements). Ex)

```
        var s = new Array("AA", "BB","CC");
        alert(s.length);
        //return 3
```

ARRAY properties and methods

- Adding Array Elements
  - To add a new element to an array is using the push method: •

    s.push("XX");

```
<script type="text/javascript">

    var s = ["AA", "BB", "CC"];
      s.push("XX");
      for (i = 0; i < s.length; i++) {
      alert(s[i])
    }

</script>
```

ARRAY properties and methods

- Adding Array Elements
- New element can also be added to an array using the length property:

```
var s = ["BB", "OO", "AA", "MM"];
s[s.length] = "LL"; //add as last element
```

- Adding elements with high indexes can create undefined "holes" in an array:

```
var s = ["BB", "OO", "AA", "MM"];
s[6] = "PP"
```

```
<script>
var s = ["AA","BB","CC"];
alert(s[0]+" "+s[1]+" "+s[2]);
alert(s.length);
s[s.length] = "LL";
s[6]="XX"
for (x in s)
 {
alert("Element"+x+ "="+ s[x]);
}
</script>
```

ARRAY properties and methods

- Popping Array Elements
- The pop() method removes the last element from an array:
- Example

```
var s = ["AA", "BB", "CC"];
s.pop();
// Removes the last element (“CC") from array s
```

- The pop() method returns the value that was "popped out":
- Example

```
var s = ["BB","OO","AA",“MM"];
var x = s.pop();
// the value of x is "MM"
```

ARRAY properties and methods

- Shifting is equivalent to popping, working on the first element instead of the last.
- The shift() method removes the first array element and "shifts" all other elements to a lower index.
- Example

```
var s = ["BB", "OO", "AA", “MM"];
 s.shift();
 // Removes the first element "BB" from ‘ s’
```

- The shift() method returns the string that was "shifted out":

Example

```
var s = ["BB", "OO", "AA", “MM"];
 alert(s.shift());  //dislays "BB"
```

ARRAY properties and methods

- The unshift() method adds a new element to an array (at the beginning), and "unshifts" older elements:
- Example

  var s = ["BB", "OO", "AA", “MM"];

  s.unshift(“PP"); // Adds a new element “PP" to s
- The unshift() method returns the new array length.
- Example

  var s = ["BB", "OO", "AA", “MM"];

  alert(s.unshift(“PP");

  // Returns 5

```
<script>
var s = ["BB", "OO", "AA", "MM"];
s.unshift("PP");
//add PP to s in the first position
for (x in s)
 {
alert("Element"+x+ "="+ s[x]);
}
</script>
```

ARRAY properties and methods

- Deleting Elements
- Since JavaScript arrays are objects, elements can be deleted by using the JavaScript operator delete:
- Example

```
var s = ["BB", "OO", "AA",“MM"];
delete s[0];
// Changes the first element in s to undefined
```

- Using delete may leave undefined holes in the array. Use pop() or shift() instead.

ARRAY properties and methods

- Splicing an Array
- The splice() method can be used to add new items to an array:
- Example

```
var fruits = ["Banana","Orange", "Apple", "Mango"];
fruits.splice(2, 0, "Lemon", "Kiwi");
```

- The first parameter (2) defines the position where new elements should be added (spliced in).
- The second parameter (0) defines how many elements should be removed.
- The rest of the parameters ("Lemon" , "Kiwi") define the new elements to be added.

ARRAY properties and methods

```
<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(2, 0, "Lemon", "Kiwi");
for (x in fruits)
 {
document.write(fruits[x]+"<br/>");
}
</script>
```

Banana
Orange
Lemon
Kiwi
Apple
Mango

ARRAY properties and methods

- Merging an Array
- concat() method creates a new array by merging (concatenating) existing arrays:
- Example (Merging Two Arrays)

```
var girls = [“ammu",“annu"];
var boys = ["Emil",“richu","Linu"];
 var mystds = girls.concat(boys);   //Concatenates (joins) girls and boys
```

- The concat() method does not change the existing arrays. It always returns a new array.The concat() method can take any number of array arguments:
- Example (Merging Three Arrays)

```
 var arr1 = ["cilie","Lona"]; var arr2 = ["Emil","Tobias","Linus"];
var arr3 = ["Robin","Morgan"];
 var myChildren = arr1.concat(arr2, arr3);  // Concatenates arr1 with arr2 and arr3
```

ARRAY properties and methods

- The concat() method can also take values as arguments:
- Example (Merging an Array with Values)

```
var arr1 = ["Cecilie", "Lone"];
    var myChildren = arr1.concat(["Emil", "asi", "Linu"]);
```

- slice() method slices out a piece of an array into a new array.
- slices out a part of an array starting from array element 1 ("Orange"):

```
var fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
    var citrus = fruits.slice(1);
```

- slice() method creates a new array. It does not remove any elements from the source array.

ARRAY properties and methods

- This example slices out a part of an array starting from array element 3 ("Apple"):

```
var fruits = ["Banana","Orange","Lemon","Apple","Mango"];
    var citrus = fruits.slice(3);
```

- The slice() method can take two arguments like slice(1, 3).
- The method then selects elements from the start argument, and up to (but not including) the end argument.

```
    var fruits = ["Banana","Orange","Lemon","Apple","Mango"];
    var citrus = fruits.slice(1,3);
```

- The sort() method sorts an array alphabetically:

```
    var fruits = ["Banana","Orange","Apple","Mango"];
    fruits.sort();
    // Sorts the elements of fruits
```

ARRAY properties and methods

- Numeric Sort
- By default, the sort() function sorts values as strings.
- This works well for strings .
- However, if numbers are sorted as strings, "25" is bigger than "100", because "2" is bigger than "1".
- Because of this, the sort() method will produce incorrect result when sorting numbers
- Use compare function for numeric sorting.A function that defines an alternative sort order. • The function should return a negative, zero, or positive value, depending on the arguments, like: function(a, b)

```
{
return a-b;
}
```

ARRAY properties and methods

- When the sort() method compares two values,itsends the values to the compare function,and sorts the values according to the returned (negative, zero, positive) value.
- Example:
- When comparing 40 and 100, the sort() method calls the compare function(40,100).The function calculates 40-100, and returns -60 (a negative value).The sort function will sort 40 as a value lower than 100.

```
<SCRIPT TYPE="text/javascript">
var points = [40, 100, 1, 5, 25, 10];
//asceding order
alert(points.sort(function(a,b){return a-b}));
//descending order
alert(points.sort(function(a,b){return b-a}));
//highest value in array
alert("Highest value"+points[0]);
</SCRIPT>
```

Reverse Arrays

- The reverse() method reverses the elements in an array.
- use it to sort an array in descending order:

```
var fruits = ["Banana", "Orange", "Apple","Mango"];
        fruits.sort();
        // Sorts the elements of fruits
         fruits.reverse();
        // Reverses the order of the elements
```

Associate Arrays

- Arrays with named indexes are called associative arrays (or hashes).
- JavaScript does not support arrays with named indexes.
- In JavaScript,arrays always use numbered indexes.
- Example

```
var person = [];
person[0] = "John";
person[1] = "Doe";
person[2] = 46;
var x = person.length;
 // person.length will return 3
var y = person[0];
// person[0] will return "John"
```

Associate Arrays

- If you use named indexes, JavaScript will redefine the array to a standard object.
- After that, some array methods and properties will produce incorrect results.
- Example:

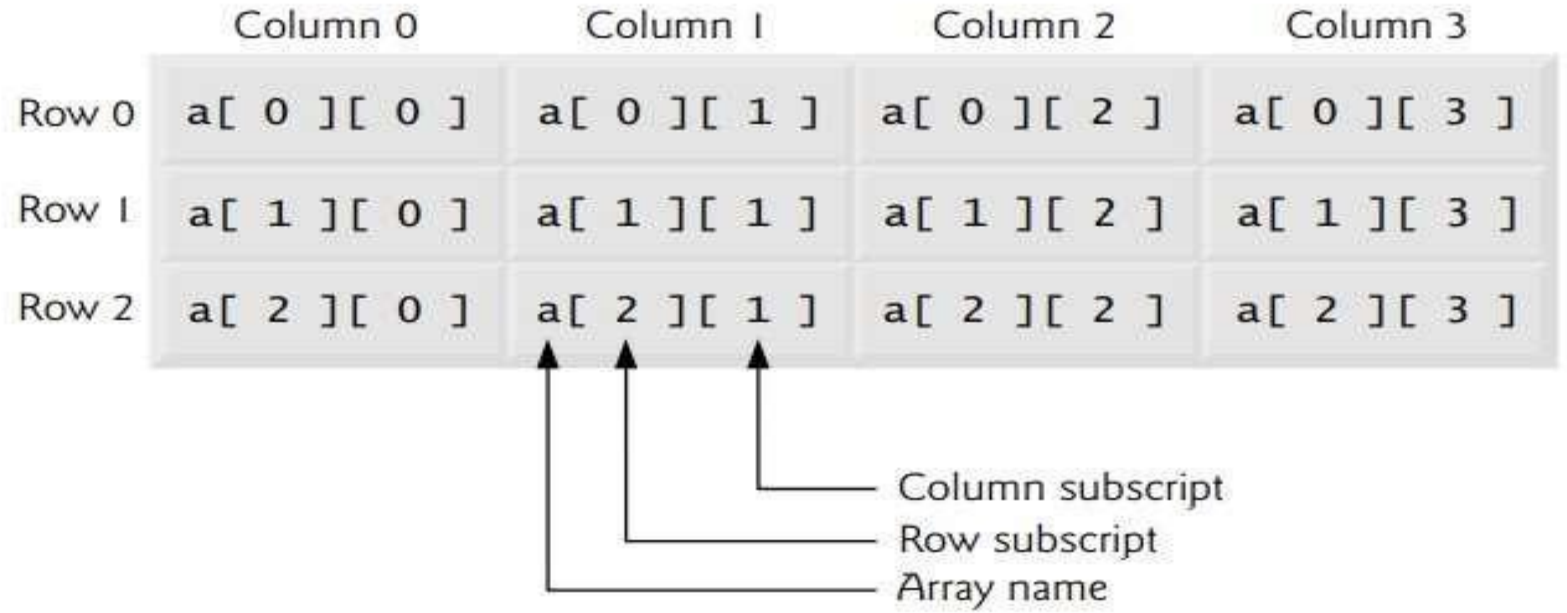
```
var person = [];
person["firstName"] = "John";
person["lastName"] = "Doe";
person["age"] = 46;

var x = person.length;
// person.length will return 0
var y = person[0];
// person[0] will return undefined
```

Multi Dimensional Arrays

- Multidimensional arrays with two indices are often used to represent tables of values consisting of information arranged in rows and columns
- Arrays that require two indices to identify a particular element are called two-dimensional arrays.
- Multidimensional arrays can have more than two dimensions.
- JavaScript does not support multidimensional arrays directly, but it does allow you to specify arrays whose elements are also arrays.
- An array with m rows and n columns is called an m-by-n array.

Multi Dimensional Arrays

- Every element in array a is identified by an element name of the form a[row][column]—a is the name of the array, and row and column are the indices that uniquely identify the row and column, respectively, of each element in a.
- The element names in row 0 all have a first index of 0; the element names in column 3 all have a second index of 3.



Two-dimensional array with three rows and four columns.

Arrays of One dimensional Arrays

- Multidimensional arrays can be initialized in declarations like a one-dimensional array. Array b with two rows and two columns could be declared and initialized with the statement .Two-dimensional array with three rows and four columns.

```
var b = [ [ 1, 2 ], [ 3, 4 ] ];
```

- The values are grouped by row in square brackets.
- The array [1, 2] initializes element b[0], and the array [3, 4] initializes element b[1]. So 1 and 2 initialize b[0][0] and b[0][1], respectively. Similarly, 3 and 4 initialize b[1][0] and b[1][1], respectively.
- interpreter determines the number of rows by counting the number of subinitializer lists— arrays nested within the outermost array. The interpreter determines the number of columns in each row by counting the number of values in the subarray that initializes the

Two-Dimensional Arrays with Rows of Different Lengths and new

- The rows of a two-dimensional array can vary in length. The declaration var b = [[1, 2], [3, 4, 5]];
- creates array b with row 0 containing two elements (1 and 2) and row 1 containing three elements (3, 4 and 5).
- Creating Two-Dimensional Arrays with new
- A multidimensional array in which each row has a different number of columns can be allocated dynamically, as follows:

```
var b;
b = new Array( 2 ); // allocate two rows
b[ 0 ] = new Array( 5 ); // allocate columns for row 0
b[ 1 ] = new Array( 3 ); // allocate columns for row 1
```

- The preceding code creates a two-dimensional array with two rows. Row 0 has five columns, and row 1 has three columns.

Two-Dimensional Arrays with Rows of Different Lengths and new

```
<script>
    var myNumbers = [[1, 2], // row 0
                     [3], // row 1
                     [4, 5, 6]]; // row 2
    for (var i = 0; i < myNumbers.length; ++i)
    {
        for (var j = 0; j < myNumbers[i].length; ++j)
        {
            document.writeln(myNumbers[i][j]);
        }
        document.writeln("<br/>");
    }
</script>
```

```
1 2
3
4 5 6
```

OBJECTS

OBJECTS

- In JavaScript,object is the king.
- Almost "everything" is an object.
- Booleans can be objects (if defined with the new keyword)
- Numbers can be objects (if defined with the new keyword)
- Strings can be objects (if defined with the new keyword)
- Dates are always objects
- Maths are always objects
- Regular expressions are always objects
- Arrays are always objects
- Functions are always objects
- All JavaScript values, except primitives, are objects.

OBJECTS

- With JavaScript, you can define and create your own objects.
- There are different ways to create new objects:
    1) Define and create a single object, using an object literal.
    2) Define and create a single object, with the keyword new.
- JavaScript objects are written with curly braces. Object properties are written as name:value pairs, separated by commas.
- The following example creates a new JavaScript object with four properties:

  var person = {firstName:“xyz”, lastName:“pqr", age:50, eyeColor:"blue"};
- The object (person) in the example above has 4 properties: firstName, lastName, age, and eyeColor.
- A JavaScript object is a collection of named values
- The named values, in JavaScript objects, are called properties.

OBJECT Creation using new

- The following example also creates a new JavaScript object with four properties:
- Properties are the values associated with a JavaScript object.
- A JavaScript object is a collection of unordered properties.
- Properties can usually be changed, added, and deleted, but some are read only
- Example

```
var person = new Object();
person.firstName = “xyz";
person.lastName = “pqr";
person.age = 50;
person.eyeColor = "blue";
```

OBJECT Creation using new

- The following example also creates a new JavaScript object with four properties:
- Properties are the values associated with a JavaScript object.
- A JavaScript object is a collection of unordered properties.
- Properties can usually be changed, added, and deleted, but some are read only
- Example

```
var person = new Object();
person.firstName = “xyz";
person.lastName = “pqr";
person.age = 50;
person.eyeColor = "blue";
```

Add new OBJECT properties & Accessing OBJECT properties

- You can add new properties to an existing object by simply giving it a value.
- Assume that the person object already exists - you can then give it new properties:
- Example person.city = ”kochi";
- syntax for accessing the property of an object is:

objectName.property ----- person.age

or

objectName["property"] -----person["age"]

or

objectName[expression] ------- x = "age"; person[x]

Accessing OBJECT properties

```
person.firstname + " is " + person.age + " years old.";
        or
person["firstname"] + " is " + person["age"] + " years old.";
```

- The JavaScript for...in statement loops through the properties of an object.
- Syntax

```
for (variable in object)
{
code to be executed
 }
```

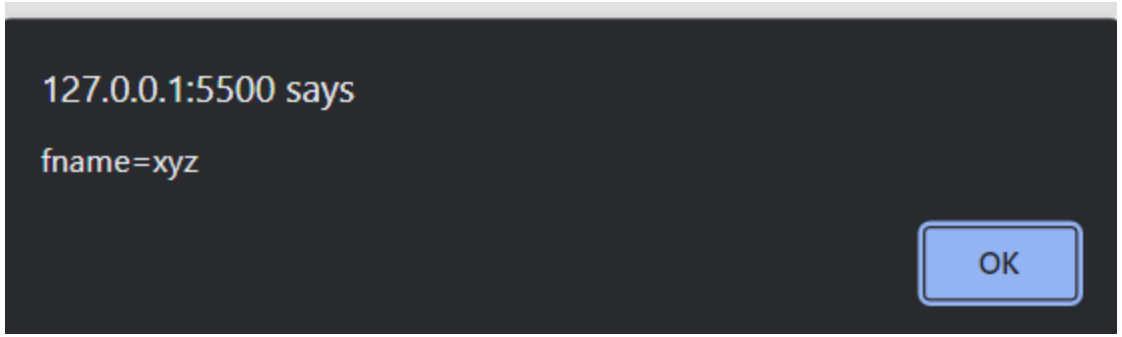
- The block of code inside of the for...in loop will be executed once for each property.

Looping through the OBJECT

JavaScript for...in Loop

- Looping through the properties of an object:
- Example

```
<script type="text/javascript">
    var person = {fname:"xyz", lname:"pqr", age:25};
    for (x in person)
    {
    alert(x+ "="+ person[x]);
    }
    </script>
```



Delete the properties of OBJECT

- The delete keyword deletes a property from an object:
- Example

```
var person = {fname:“xyz", lname:“pqr", age:50, city:“kochi"};
delete person.age;
        or
delete person["age"];
```

- The delete keyword deletes both the value of the property and the property itself.
- After deletion, the property cannot be used before it is added back again.
- The delete operator is designed to be used on object properties.
- It has no effect on variables or functions

STRINGS, NUMBERS, AND BOOLEANS AS OBJECTS

- When a JavaScript variable is declared with the keyword "new", the variable is created as an object:

```
var x = new String();
 // Declares x as a String object
var y = new Number();
 // Declares y as a Number object
var z = new Boolean();
// Declares z as a Boolean object
```

- Avoid String, Number, and Boolean objects.

Math Object methods

Method	Description	Examples

`abs( x )`	Absolute value of x.	`abs( 7.2 )` is `7.2`
		`abs( 0 )` is `0`
		`abs( -5.6 )` is `5.6`

`ceil( x )`	Rounds x to the smallest integer not less than x.	`ceil( 9.2 )` is `10` `ceil( -9.8 )` is `-9.0`
`cos( x )`	Trigonometric cosine of x (x in radians).	`cos( 0 )` is `1`
`exp( x )`	Exponential method e^x.	`exp( 1 )` is `2.71828` `exp( 2 )` is `7.38906`
`floor( x )`	Rounds x to the largest integer not greater than x.	`floor( 9.2 )` is `9` `floor( -9.8 )` is `-10.0`
`log( x )`	Natural logarithm of x (base *e*).	`log( 2.718282 )` is `1` `log( 7.389056 )` is `2`

Math Object methods

`max( x, y )`	Larger value of x and y.	`max( 2.3, 12.7 )` is `12.7` `max( -2.3, -12.7 )` is `-2.3`
`min( x, y )`	Smaller value of x and y.	`min( 2.3, 12.7 )` is `2.3` `min( -2.3, -12.7 )` is `-12.7`
`pow( x, y )`	x raised to power y (x^y).	`pow( 2, 7 )` is `128` `pow( 9, .5 )` is `3.0`
`round( x )`	Rounds x to the closest integer.	`round( 9.75 )` is `10` `round( 9.25 )` is `9`
`sin( x )`	Trigonometric sine of x (x in radians).	`sin( 0 )` is `0`
`sqrt( x )`	Square root of x.	`sqrt( 900 )` is `30` `sqrt( 9 )` is `3`
`tan( x )`	Trigonometric tangent of x (x in radians).	`tan( 0 )` is `0`

String Object methods

Method	Description
`charAt( index )`	Returns a string containing the character at the specified *index*. If there's no character at the *index*, `charAt` returns an empty string. The first character is located at *index* 0.
`charCodeAt( index )`	Returns the Unicode value of the character at the specified *index*, or `NaN` (not a number) if there's no character at that *index*.
`concat( string )`	Concatenates its argument to the end of the string on which the method is invoked. The original string is not modified; instead a new `String` is returned. This method is the same as adding two strings with the string-concatenation operator + (e.g., `s1.concat(s2)` is the same as `s1 + s2`).
`fromCharCode( value1, value2, ...)`	Converts a list of Unicode values into a string containing the corresponding characters.

String Object methods

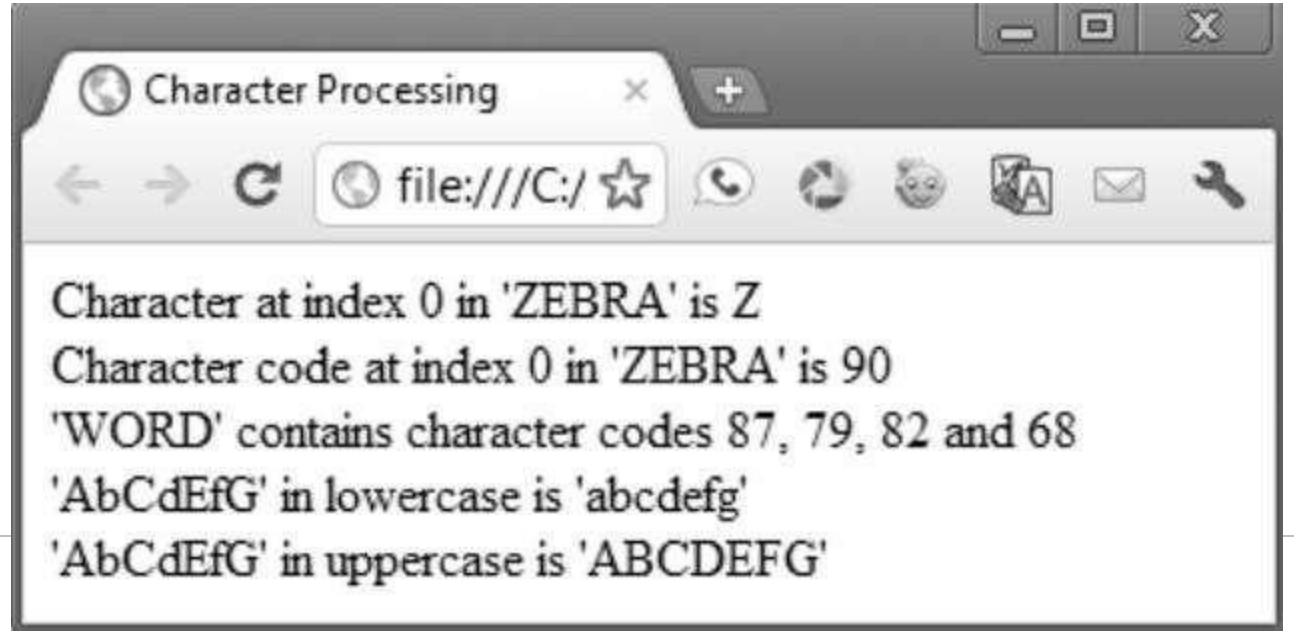
`indexOf(` *substring*, *index* `)`	Searches for the *first* occurrence of *substring* starting from position *index* in the string that invokes the method. The method returns the starting index of *substring* in the source string or –1 if *substring* is not found. If the *index* argument is not provided, the method begins searching from index 0 in the source string.
`lastIndexOf(` *substring*, *index* `)`	Searches for the *last* occurrence of *substring* starting from position *index* and searching toward the beginning of the string that invokes the method. The method returns the starting index of *substring* in the source string or –1 if *substring* is not found. If the *index* argument is not provided, the method begins searching from the *end* of the source string.
`replace(` *searchString*, *replaceString* `)`	Searches for the substring *searchString*, replaces the first occurrence with *replaceString* and returns the modified string, or returns the original string if no replacement was made.
`slice(` *start*, *end* `)`	Returns a string containing the portion of the string from index *start* through index *end*. If the *end* index is not specified, the method returns a string from the *start* index to the end of the source string. A negative *end* index specifies an offset from the end of the string, starting from a position one past the end of the last character (so –1 indicates the last character position in the string).

String Object methods

`split(` *string* `)`	Splits the source string into an array of strings (tokens), where its *string* argument specifies the delimiter (i.e., the characters that indicate the end of each token in the source string).
`substr(` *start*, *length* `)`	Returns a string containing *length* characters starting from index *start* in the source string. If *length* is not specified, a string containing characters from *start* to the end of the source string is returned.
`substring(` *start*, *end* `)`	Returns a string containing the characters from index *start* up to but not including index *end* in the source string.
`toLowerCase()`	Returns a string in which all uppercase letters are converted to lowercase letters. Non-letter characters are not changed.
`toUpperCase()`	Returns a string in which all lowercase letters are converted to uppercase letters. Non-letter characters are not changed.

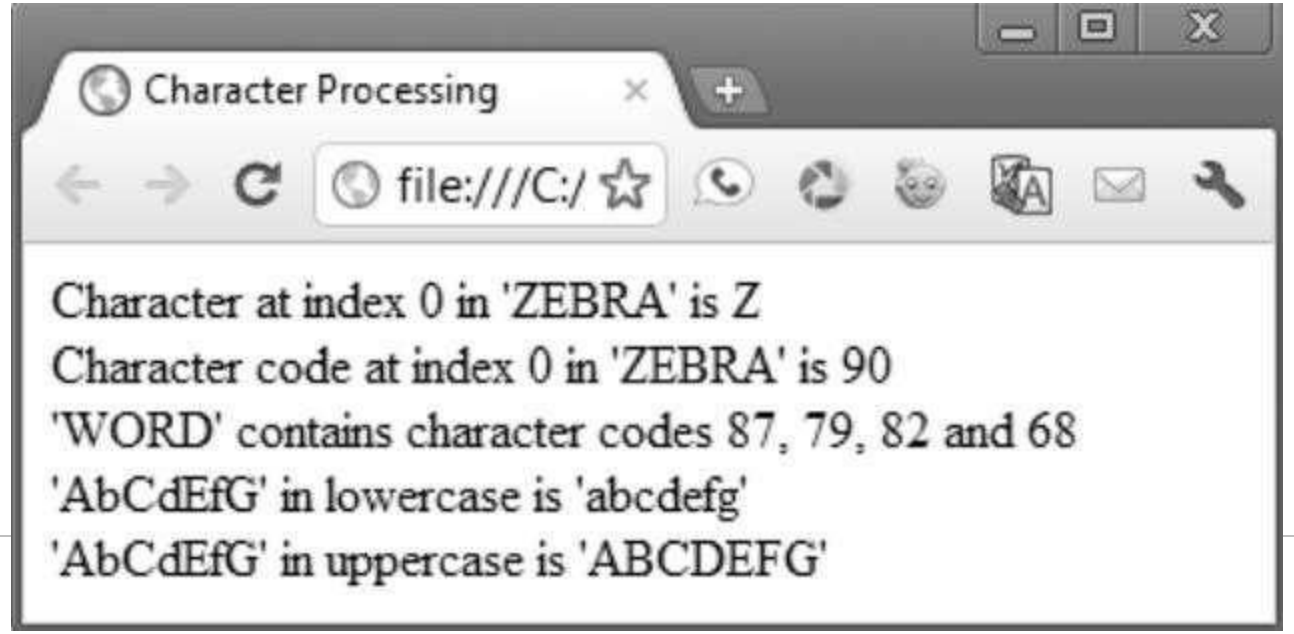
Character processing methods

- String object's character-processing methods, including:
  - charAt—returns the character at a specific position
  - charCodeAt—returns the Unicode value of the character at a specific position
  - fromCharCode—returns a string created from a series of Unicode values
  - toLowerCase—returns the lowercase version of a string
  - toUpperCase—returns the uppercase version of a string



Character processing methods

- String object's character-processing methods, including:
  - charAt—returns the character at a specific position
  - charCodeAt—returns the Unicode value of the character at a specific position
  - fromCharCode—returns a string created from a series of Unicode values
  - toLowerCase—returns the lowercase version of a string
  - toUpperCase—returns the uppercase version of a string



Character processing methods

Character processing methods

```
<script>
    var letters = "abcdefghijklmnopqrstuvwxyzabcdefghijklm";
    function buttonPressed() {
        var inputField = document.getElementById("inputField");
        document.getElementById("results").innerHTML =
            "<p>First occurrence is located at index " + letters.indexOf(inputField.value)
            + "</p>" + "<p>Last occurrence is located at index " + letters.lastIndexOf(inputField.value)
            + "</p>" + "<p>First occurrence from index 12 is located at index " + letters.indexOf(inputField.value, 12)
            + "</p>" + "<p>Last occurrence from index 12 is located at index " + letters.lastIndexOf(inputField.value, 12)
            + "</p>";
    }
    function start() {
        var searchButton = document.getElementById("searchButton");
        searchButton.addEventListener("click", buttonPressed, false);
    } // end function start
    window.addEventListener("load", start, false);
</script>
<body>
    <form id="searchForm">
        <h1>The string to search is:
            abcdefghijklmnopqrstuvwxyzabcdefghijklm</h1>
        <p>Enter the substring to search for
            <input id="inputField" type="search">
            <input id="searchButton" type="button" value="Search">
        </p>
        <div id="results"></div>
    </form>
</body>
```

Date Object Methods

Method	Description
`getDate()` `getUTCDate()`	Returns a number from 1 to 31 representing the day of the month in local time or UTC.
`getDay()` `getUTCDay()`	Returns a number from 0 (Sunday) to 6 (Saturday) representing the day of the week in local time or UTC.
`getFullYear()` `getUTCFullYear()`	Returns the year as a four-digit number in local time or UTC.
`getHours()` `getUTCHours()`	Returns a number from 0 to 23 representing hours since midnight in local time or UTC.
`getMilliseconds()` `getUTCMilliSeconds()`	Returns a number from 0 to 999 representing the number of milliseconds in local time or UTC, respectively. The time is stored in hours, minutes, seconds and milliseconds.
`getMinutes()` `getUTCMinutes()`	Returns a number from 0 to 59 representing the minutes for the time in local time or UTC.
`getMonth()` `getUTCMonth()`	Returns a number from 0 (January) to 11 (December) representing the month in local time or UTC.
`getSeconds()` `getUTCSeconds()`	Returns a number from 0 to 59 representing the seconds for the time in local time or UTC.

Date Object Methods

Method	Description
`getTime()`	Returns the number of milliseconds between January 1, 1970, and the time in the `Date` object.
`getTimezoneOffset()`	Returns the difference in minutes between the current time on the local computer and UTC (Coordinated Universal Time).
`setDate( val )` `setUTCDate( val )`	Sets the day of the month (1 to 31) in local time or UTC.
`setFullYear( y, m, d )` `setUTCFullYear( y, m, d )`	Sets the year in local time or UTC. The second and third arguments representing the month and the date are optional. If an optional argument is not specified, the current value in the `Date` object is used.
`setHours( h, m, s, ms )` `setUTCHours( h, m, s, ms )`	Sets the hour in local time or UTC. The second, third and fourth arguments, representing the minutes, seconds and milliseconds, are optional. If an optional argument is not specified, the current value in the `Date` object is used.
`setMilliSeconds( ms )` `setUTCMilliseconds( ms )`	Sets the number of milliseconds in local time or UTC.

Date Object Methods

Method	Description
`getTime()`	Returns the number of milliseconds between January 1, 1970, and the time in the `Date` object.
`getTimezoneOffset()`	Returns the difference in minutes between the current time on the local computer and UTC (Coordinated Universal Time).
`setDate( val )` `setUTCDate( val )`	Sets the day of the month (1 to 31) in local time or UTC.
`setFullYear( y, m, d )` `setUTCFullYear( y, m, d )`	Sets the year in local time or UTC. The second and third arguments representing the month and the date are optional. If an optional argument is not specified, the current value in the `Date` object is used.
`setHours( h, m, s, ms )` `setUTCHours( h, m, s, ms )`	Sets the hour in local time or UTC. The second, third and fourth arguments, representing the minutes, seconds and milliseconds, are optional. If an optional argument is not specified, the current value in the `Date` object is used.
`setMilliSeconds( ms )` `setUTCMilliseconds( ms )`	Sets the number of milliseconds in local time or UTC.

Date Object Methods

`setUTCHours( h, m, s, ms )`	fourth arguments, representing the minutes, seconds and milliseconds, are optional. If an optional argument is not specified, the current value in the `Date` object is used.
`setMilliSeconds( ms )` `setUTCMilliseconds( ms )`	Sets the number of milliseconds in local time or UTC.
`setMinutes( m, s, ms )` `setUTCMinutes( m, s, ms )`	Sets the minute in local time or UTC. The second and third arguments, representing the seconds and milliseconds, are optional. If an optional argument is not specified, the current value in the `Date` object is used.
`setMonth( m, d )` `setUTCMonth( m, d )`	Sets the month in local time or UTC. The second argument, representing the date, is optional. If the optional argument is not specified, the current date value in the `Date` object is used.
`setSeconds( s, ms )` `setUTCSeconds( s, ms )`	Sets the seconds in local time or UTC. The second argument, representing the milliseconds, is optional. If this argument is not specified, the current milliseconds value in the `Date` object is used.
`setTime( ms )`	Sets the time based on its argument—the number of elapsed milliseconds since January 1, 1970.

Date Object Methods

`toLocaleString()`	Returns a string representation of the date and time in a form specific to the computer's locale. For example, September 13, 2007, at 3:42:22 PM is represented as *09/13/07 15:47:22* in the United States and *13/09/07 15:47:22* in Europe.
`toUTCString()`	Returns a string representation of the date and time in the form: *15 Sep 2007 15:47:22 UTC.*
`toString()`	Returns a string representation of the date and time in a form specific to the locale of the computer (*Mon Sep 17 15:47:22 EDT 2007* in the United States).
`valueOf()`	The time in number of milliseconds since midnight, January 1, 1970. (Same as `getTime`.)

Boolean and Number Objects

- JavaScript provides the Boolean and Number objects as object wrappers for boolean true/ false values and numbers, respectively.
- These wrappers define methods and properties useful in manipulating boolean values and numbers.
- When a JavaScript program requires a boolean value, JavaScript automatically creates a Boolean object to store the value.

```
var b = new Boolean( booleanValue );
```

- The booleanValue specifies whether the Boolean object should contain true or false.
- If booleanValue is false, 0, null, Number.NaN or an empty string (""), or if no argument is supplied, the new Boolean object contains false.

`toString()`	Returns the string "`true`" if the value of the `Boolean` object is `true`; otherwise, returns the string "`false`".
`valueOf()`	Returns the value `true` if the `Boolean` object is `true`; otherwise, returns `false`.

Boolean and Number Objects

- JavaScript automatically creates Number objects to store numeric values in a script. You can create a Number object with the statement

```
var n = new Number( numericValue );
```

- The constructor argument numericValue is the number to store in the object.
- We can explicitly create Number objects, normally the JavaScript interpreter creates them as needed

Boolean and Number Objects

`toString(` *radix* `)`	Returns the string representation of the number. The optional *radix* argument (a number from 2 to 36) specifies the number's base. Radix 2 results in the *binary* representation, 8 in the *octal* representation, 10 in the *decimal* representation and 16 in the *hexadecimal* representation. See Appendix E, Number Systems, for an explanation of the binary, octal, decimal and hexadecimal number systems.
`valueOf()`	Returns the numeric value.
`Number.MAX_VALUE`	The largest value that can be stored in a JavaScript program.
`Number.MIN_VALUE`	The smallest value that can be stored in a JavaScript program.
`Number.NaN`	*Not a number*—a value returned from an arithmetic expression that doesn't result in a number (e.g., `parseInt("hello")` cannot convert the string `"hello"` to a number, so `parseInt` would return `Number.NaN`.) To determine whether a value is `NaN`, test the result with function `isNaN`, which returns `true` if the value is `NaN`; otherwise, it returns `false`.
`Number.NEGATIVE_INFINITY`	A value less than `-Number.MAX_VALUE`.
`Number.POSITIVE_INFINITY`	A value greater than `Number.MAX_VALUE`.

Document Objects

- The document object, which we've used extensively, is provided by the browser and allows JavaScript code to manipulate the current document in the browser.
- The document object has several properties and methods, such as method document.getElementByID

Method	Description
`getElementById( id )`	Returns the HTML5 element whose `id` attribute matches *id*.
`getElementByTagName( tagName )`	Returns an array of the HTML5 elements with the specified *tagName*.

DOCUMENT OBJECT MODEL

The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."

Document Object Model

- The DOM gives you scripting access to all the elements on a web page.
- Inside the browser, the whole web page—paragraphs, forms, tables, etc.—is represented in an object hierarchy.
- Using DOM,we can dynamically create, modify and remove elements in the page.
- JavaScript can change all the HTML elements in the page
- JavaScript can change all the HTML attributes in the page
- JavaScript can change all the CSS styles in the page
- JavaScript can remove existing HTML elements and attributes
- JavaScript can add new HTML elements and attributes
- JavaScript can react to all existing HTML events in the page
- JavaScript can create new HTML events in the page

DOM nodes and Trees

- The document's getElementById method is the simplest way to access a specific element in a page. The method returns objects called DOM nodes.
- Every piece of an HTML5 page (elements, attributes, text, etc.) is modeled in the web browser by a DOM node. All the nodes in a document make up the page's DOM tree, which describes the relationships among elements.
- Nodes are related to each other through child-parent relationships.
- An HTML5 element inside another element is said to be its child—the containing element is known as the parent.
- A node can have multiple children but only one parent. Nodes with the same parent node are referred to as siblings.

DOM nodes and Trees

- The head and body nodes are siblings, since they're both children of the html node.
- The head contains the meta and title nodes
- The body node contains nodes representing each of the elements in the document's body element.
- The li nodes are children of the ul node, since they're nested inside it.

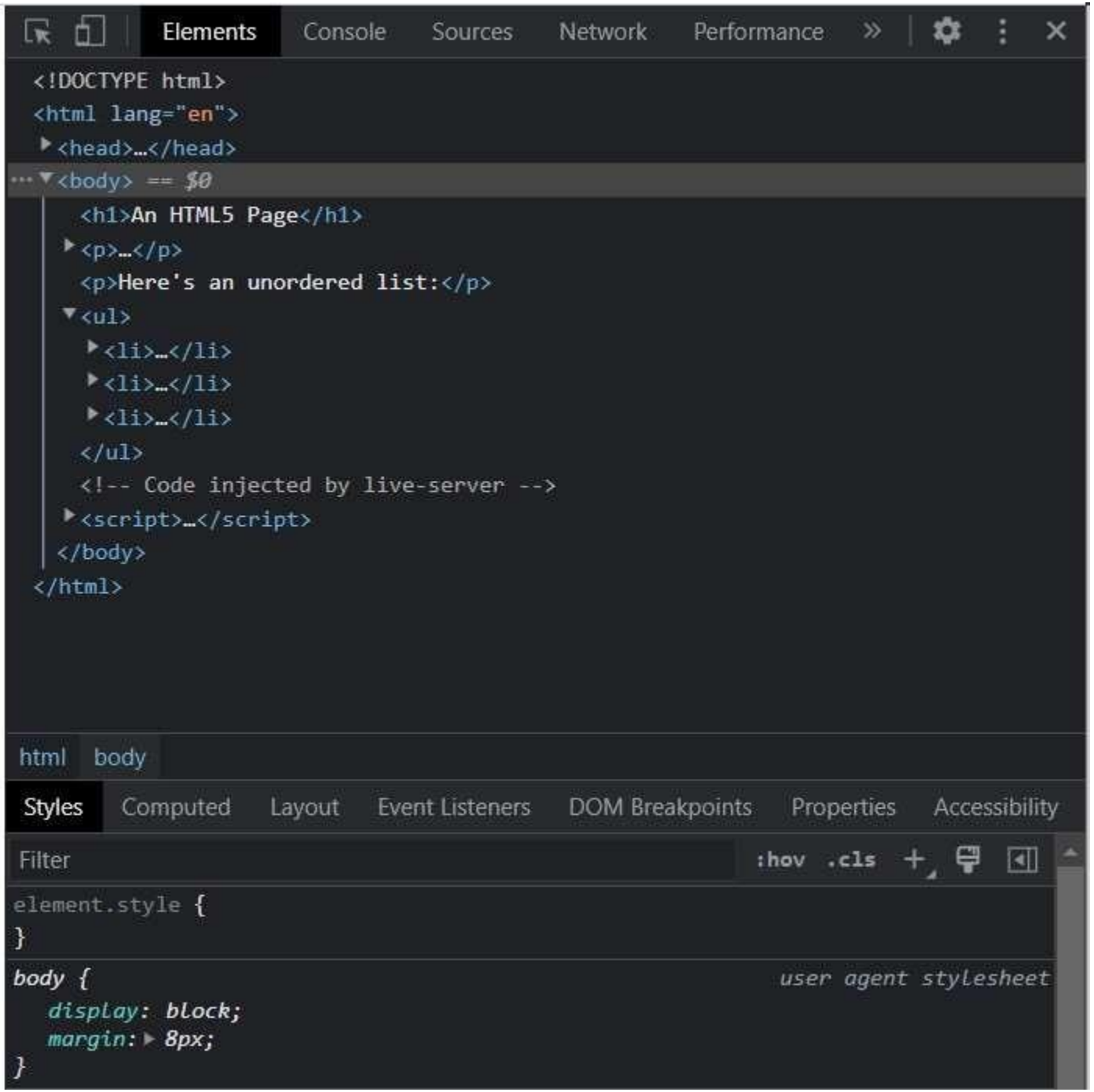
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <h1>An HTML5 Page</h1>
 <p>This page contains some basic HTML5 elements. The DOM tree
 for the document contains a DOM node for every element</p>
 <p>Here's an unordered list:</p>
<ul>
 <li>One</li>
 <li>Two</li>
 <li>Three</li>
 </ul>
</body>
</html>
```

An HTML5 Page

This page contains some basic HTML5 elements. The DOM tree for the document contains a DOM node for every element

Here's an unordered list:

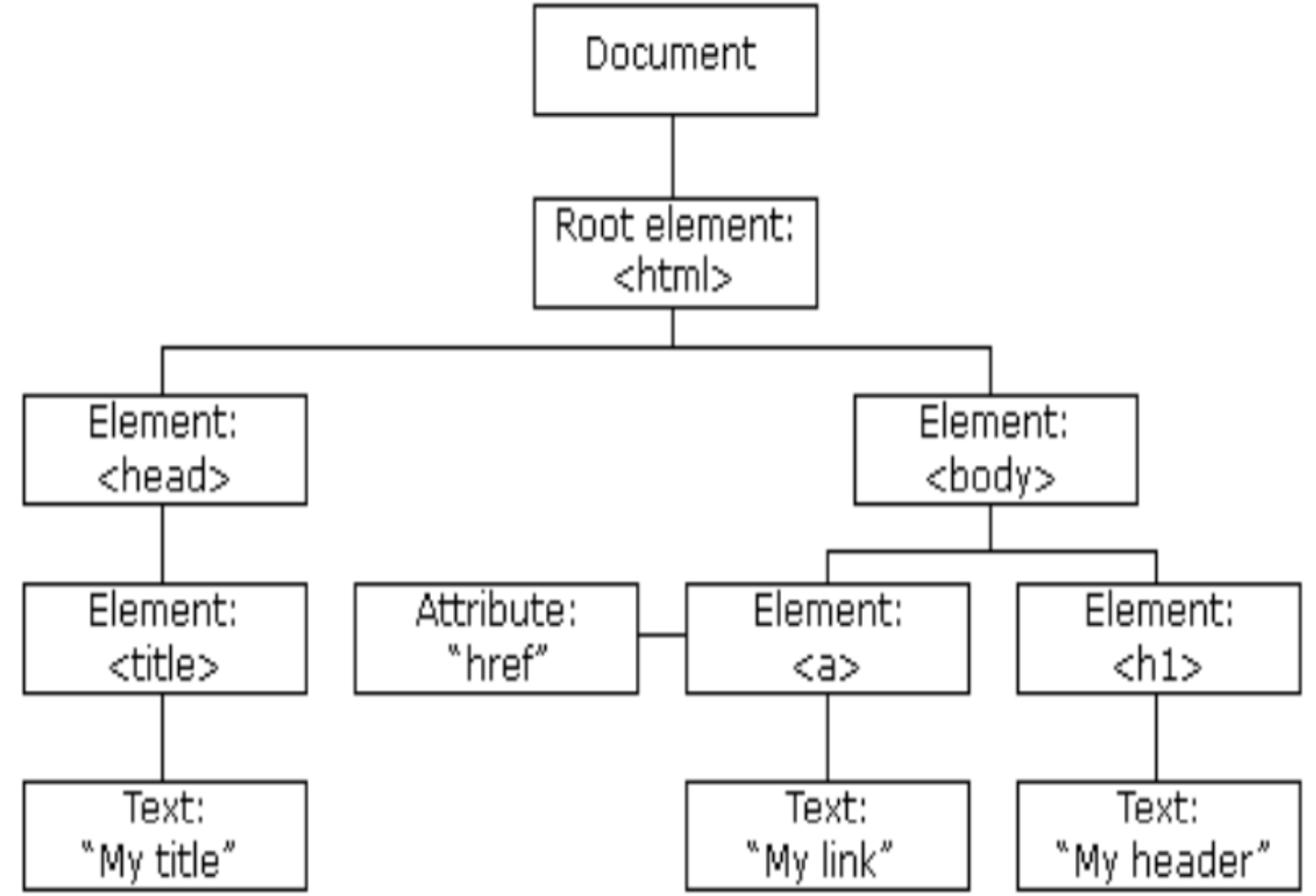
- One
- Two
- Three

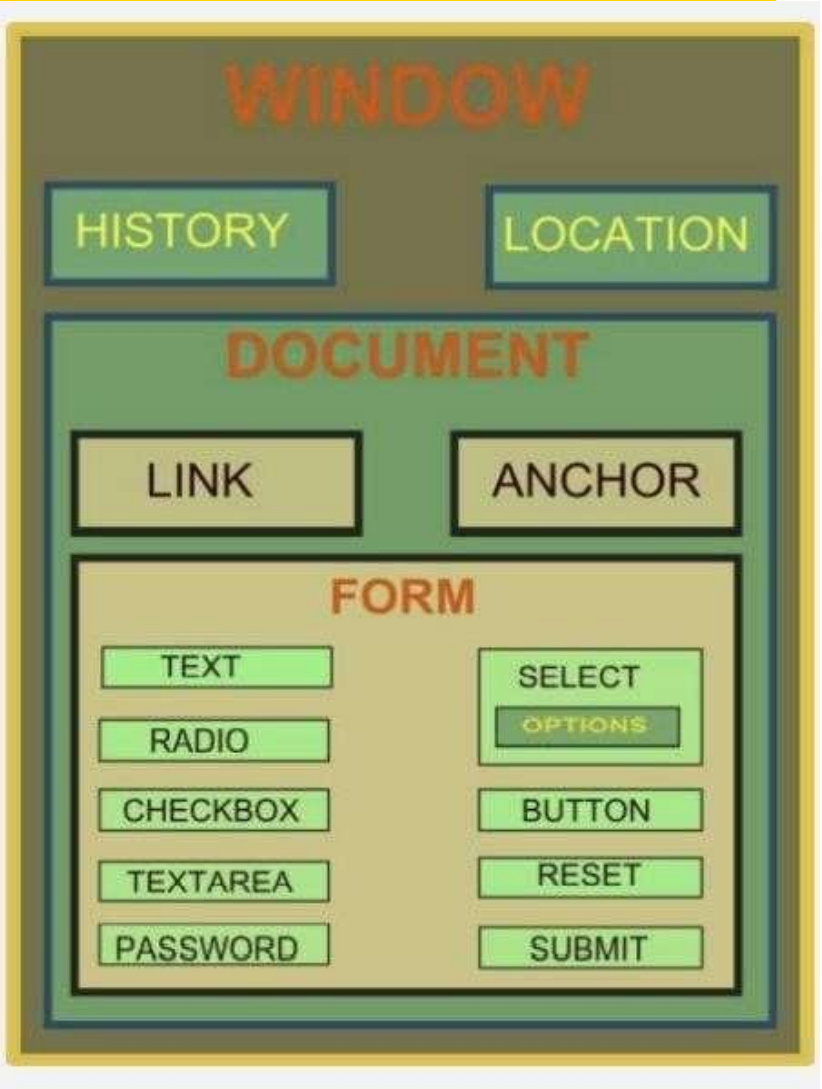


DOM

- DOM is a model for describing HTML documents
- A common set of properties/methods to access everything in a web document
- When a web page is loaded, the browser creates a Document Object Model of the page.
- The HTML DOM model is constructed as a tree of Objects:
- Objects are in a hierarchy
- The window is the parent for a given web page
- Document is the child with the objects that are most commonly manipulated
- The HTML DOM is a standard for how to get, change, add, or delete HTML elements.
- HTML DOM methods are actions you can perform (on HTML Elements).
- HTML DOM properties are values (of HTML Elements) that you can set or change.

DOM nodes and Trees





FINDING & CHANGING HTML ELEMENTS

Method	Description
document.getElementById(id)	Find an element by element id
document.getElementsByTagName(name)	Find elements by tag name
document.getElementsByClassName(name)	Find elements by class name
element.innerHTML = new html content	Change the inner HTML of an element
element.attribute = new value	Change the attribute value of an HTML element
element.setAttribute(attribute, value)	Change the attribute value of an HTML element
element.style.property = new style	Change the style of an HTML element

ADDING AND DELETING HTML ELEMENTS

Method	Description
document.createElement(*element*)	Create an HTML element
document.removeChild(*element*)	Remove an HTML element
document.appendChild(*element*)	Add an HTML element
document.replaceChild(*new, old*)	Replace an HTML element
document.write(*text*)	Write into the HTML output stream

CHANGING THE VALUE OF AN ATTRIBUTE

- To change the value of an HTML attribute, use this syntax:

document.getElementById(id).attribute = new value

```
<!DOCTYPE html>
<html><body><img id="clg" src="sjc.jpg">
<script>
    document.getElementById("clg").src ="sjclogo.jpg";
</script>
</body>
</html>
```

CHANGING THE HTML Style

- To change the style of an HTML element, use this syntax:

document.getElementById(id).style.property = new style

```
<html>
<body>
<p id="p1">Hello World!</p>
<script>
        document.getElementById("p1").style.color = "blue";
</script>
<p>The paragraph above was changed by a script.</p>
</body>
</html>
```

DOM EVENTS

- HTML DOM allows JavaScript to react to HTML events:
- JavaScript can execute a statement (typically, call a function)
- A JavaScript can be executed when an event occurs, like when a user clicks on an HTML element.
- To execute code when a user clicks on an element, add JavaScript code to an HTML event attribute:
- when an event occurs

<… oneventname="javascript stmt;">

<BODY … onload="func();">

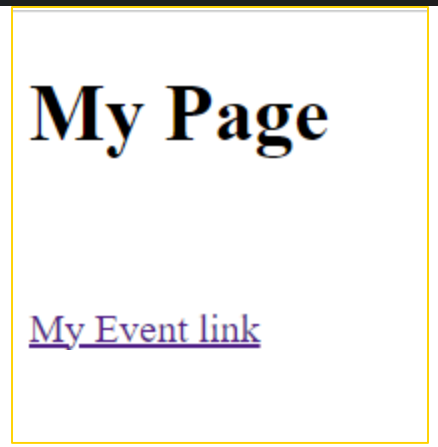
<INPUT type="submit" name="submit" onsubmit="fun();">

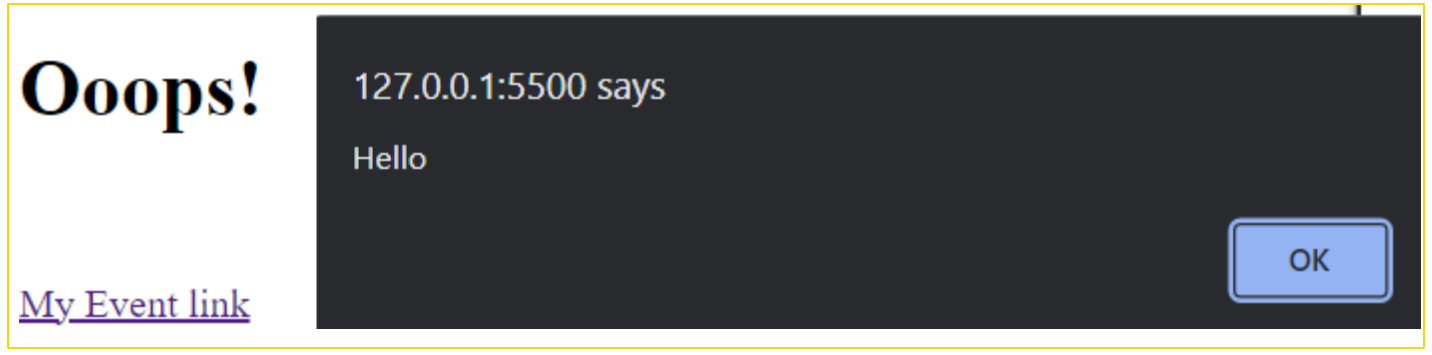
DOM EVENTS

- onsubmit - call when submit button is clicked
- onclick - call when this button is clicked
- onreset - call when the reset button is clicked
- onload - call after page loads
- onmouseover - call when mouse pointer enters image area
- onmouseout - call when mouse pointer leaves image area
- onfocus - call when control receives focus
- onblur - call when a control loses focus
- onchange - call when a control loses focus and the value of its contents has changed
- many more

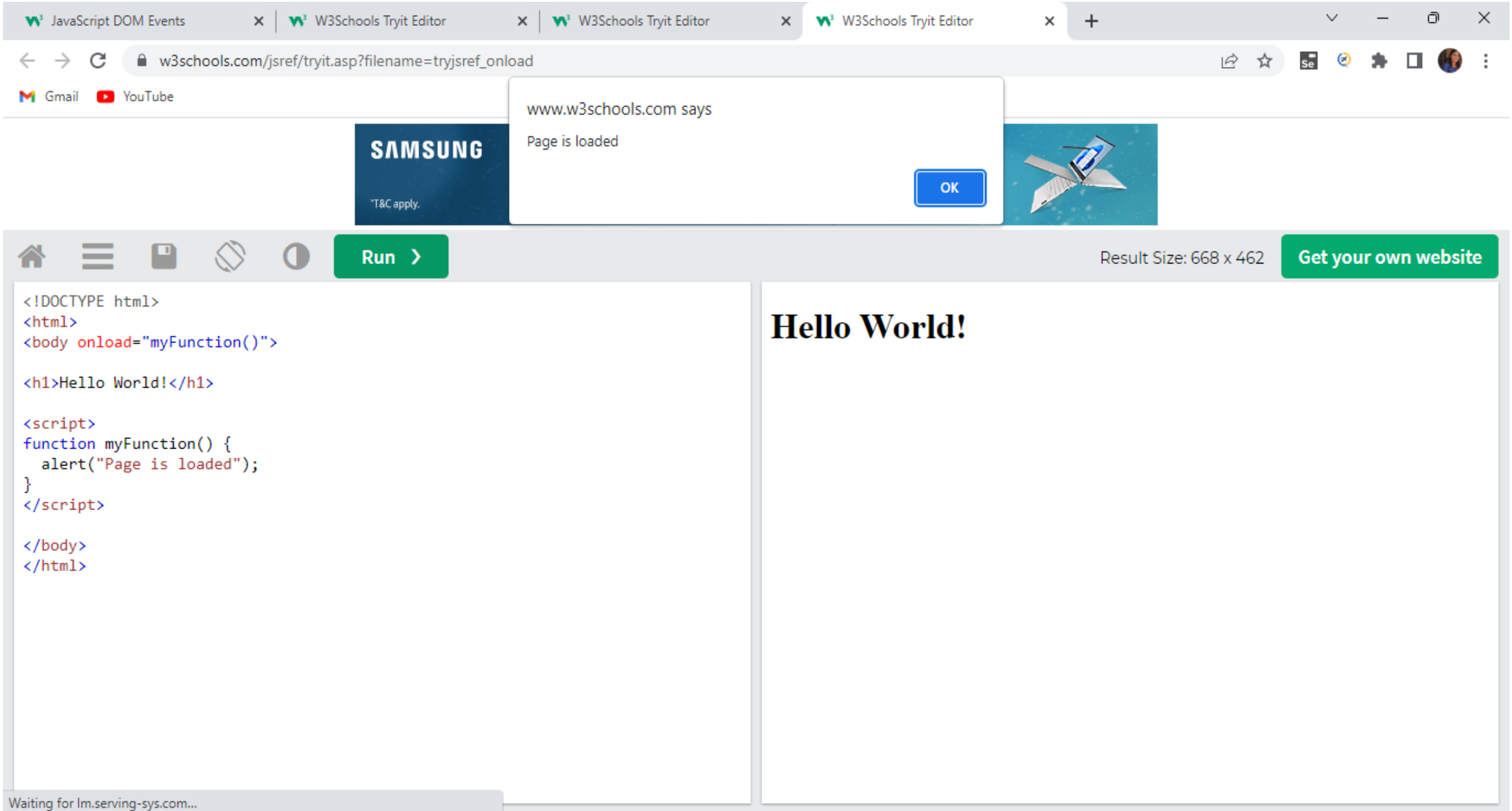
Event handling using DOM –onclick and onMouseover

```
<!DOCTYPE html>
<html lang="en">
<head> ···
</head>
<body>
    <h1 onclick="this.innerHTML = 'Ooops!'">My Page</h1>
    <br /><br />
    <a href="jtest1.html" onMouseover="window.alert('Hello');">My Event link</a>
</body>
```





Event handling using DOM –onload()

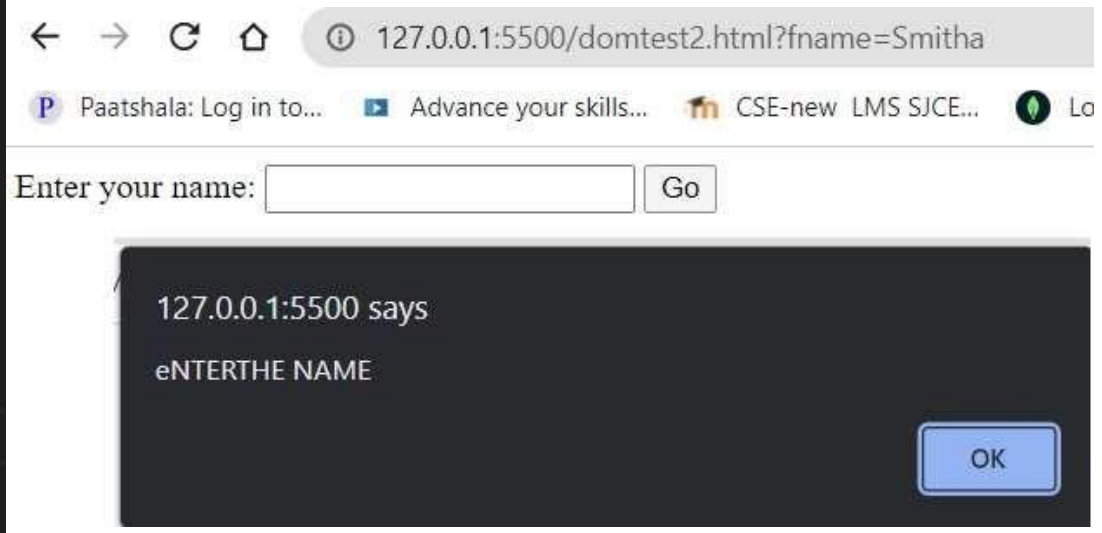


Form Handling –onclick event

```
<!DOCTYPE html>
<html lang="en">
<head>···
</head>
<body>
    <form name="f1">
        Enter your name:<input type="text" name="fname">
        <input type="button" value= "Go" onClick="window.alert ('Hello ' + document.f1.fname.value);">
        </form>
  </body> </html>
```

Form Handling –onclick event

```
<script>
    function valname() {
    if(document.f1.fname.value=="")
    {window.alert('eNTERTHE NAME');
    }
    else
    {window.alert('Hello ' +document.f1.fname.value); }
    }
    </script></head>
    <body><form name="f1" onSubmit="valname()">
    Enter your name: <input type="text" name="fname">
    <input type="submit" value= "Go" >
    </form></body></html>
```



Run ›

Result Size: 668 x 462

Get your own w

```
om the list.</p>

onchange="myFunction()">
>Audi</option>
BMW</option>
des">Mercedes</option>
">Volvo</option>


ew car, a function is triggered which outputs the
car.</p>



{
ElementById("mySelect").value;
yId("demo").innerHTML = "You selected: " + x;
```

Select a new car from the list.

Mercedes

When you select a new car, a function is triggered which outputs the value of the selected ca

You selected: Mercedes

Form Handling –onchange event

```
<script>
    function valphone()
    {
        if ((document.f1.phone.value != ""))
        {
            if (isNaN(document.f1.phone.value))
            { window.alert('eNTER Numbers for phone field'); }
            else {
                window.alert("Valid Phone "); }
        }
        else
        {
            window.alert("Enter Phone ");
        }
    }
</script></head><body>
<form name="f1">
    Enter your phone Number:
    <input type="text" name="phone" onchange="valphone()">
    <input type="submit" value="SUBMIT">
</form>
</body>
```

Node in DOM

- Any element on a page including text & whitespaces of a DOM structure is known as "NODE."
- Nodes can be between HTML Tags.
- Nodes available in DOM:
  - node.childNodes
  - node.firstChild
  - node.lastChild
  - node.parentNode
  - node.nextSibling
  - node.previousSibling

Adding and Removing Nodes (HTML Elements) **in DOM**

- To add a new element to the HTML DOM, you must create the element (element node) first, and then append it to an existing element.

```
<body ><h2>JavaScript HTML DOM</h2>
    <p>Add a new HTML Element.</p>

    <div id="div1">
    <p id="p1">This is a paragraph.</p>
    <p id="p2">This is another paragraph.</p>
    </div>

    <script>
    const para = document.createElement("p");
    const node = document.createTextNode("This is new PARA.");
    para.appendChild(node);
    const element = document.getElementById("div1");
    element.appendChild(para);
    </script>
</body>
```

JavaScript HTML DOM

Add a new HTML Element.

This is a paragraph.

This is another paragraph.

This is new PARA.

Adding and Removing Nodes (HTML Elements) **in DOM**

- This code creates a new <p> element:

  const para = document.createElement("p");

- To add text to the <p> element, you must create a text node first. This code creates a text node: const node = document.createTextNode("This is a new paragraph.");
- Then you must append the text node to the <p> element:

  para.appendChild(node);

- Finally you must append the new element to an existing element.This code finds an existing element:

  const element = document.getElementById("div1");

- This code appends the new element to the existing element:

  element.appendChild(para);

Insert before in Node

- This code appends the new element to the existing element:

element.appendChild(para);

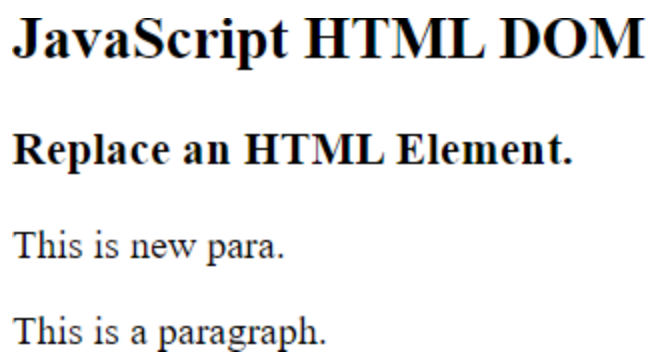

```
<h2>JavaScript HTML DOM</h2>
<p>Add a new HTML Element.</p>

<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>

<script>
const para = document.createElement("p");
const node = document.createTextNode("This is new.");
para.appendChild(node);

const element = document.getElementById("div1");
const child = document.getElementById("p1");
element.insertBefore(para,child);
</script>
```

Remove element in Node


```
<body>
    <h2>JavaScript HTML DOM</h2>
<h3>Remove an HTML Element.</h3>
<div>
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>

<button onclick="myFunction()">Remove Element</button>

<script>
function myFunction() {
document.getElementById("p1").remove();
}
</script>
</body>
```

Replacing child element in Node

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript HTML DOM</h2>
<h3>Replace an HTML Element.</h3>

<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is a paragraph.</p>
</div>

<script>
const parent = document.getElementById("div1");
const child = document.getElementById("p1");
const para = document.createElement("p");
const node = document.createTextNode("This is new para.");
para.appendChild(node);
parent.replaceChild(para,child);
</script>

</body>
</html>
```



DOM Collection

- The Document Object Model contains several collections, which are groups of related objects on a page.
- DOM collections are accessed as properties of DOM objects such as the document object or a DOM node.
- The document object has properties containing the
  - images collection
  - links collection
  - forms collection
  - anchors collection
- These collections contain all the elements of the corresponding type on the page

Thank You