

Client Side Scripting Language (22519)

Unit 1 :
Basics of JavaScript Programming
(12 M)

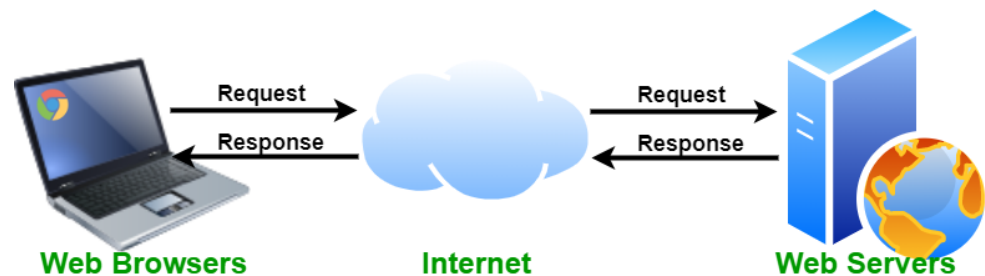
Basics of JavaScript Programming

- 1.1 Features of JavaScript
- 1.2 Object Name, Property, Method, Dot Syntax, Main Event
- 1.3 Values and Variables
- 1.4 Operators and Expressions
- 1.5 if statement , if...else. If...elseif, Nested if
- 1.6 switch... case statement
- 1.7 Loop statement
- 1.8 Querying and setting properties and Deleting properties,
Property Getters and Setters

Client side Scripting vs. Server Side Scripting

Client-side scripting requires browsers to run the scripts on the client machine but does not interact with the server while processing the client-side scripts

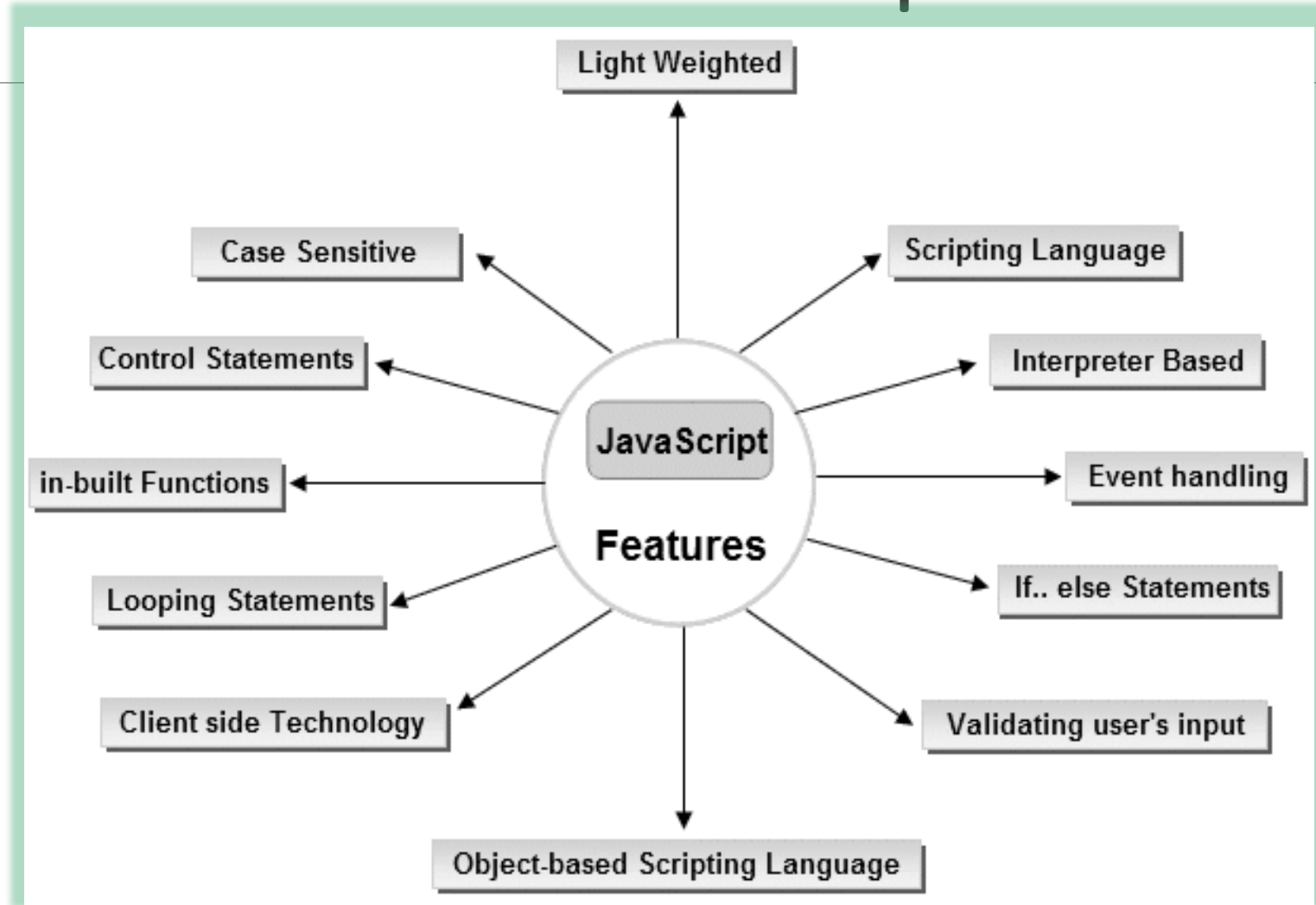
Server-side scripting involves server for its processing



Comparison

| | Server-side scripting | Client-side scripting |
|--------------------|---|--|
| Basic | Works in the back end which could not be visible at the client end. | Works at the front end and script are visible among the users. |
| Processing | Requires server interaction | Does not need interaction with the server |
| Languages involved | PHP, ASP.net, Ruby, Python | HTML, CSS, JavaScript |
| Affect | Could effectively customize the web pages and provide dynamic websites. | Can reduce the load to the server. |
| Security | Relatively secure | Insecure |

1.1 Features of JavaScript



1.2 Object Name, Property , Method, Dot Syntax, Main Event

- JavaScript is a Object based scripting language.
- A JavaScript object is a collection of named values.
- These named values are usually referred to as properties of the object.
- A JavaScript objects are collection of properties and methods.
 - ✓ A Methods is a function that is a member of an object.
 - ✓ A Property is a value or set of values that is the member of an object.

Object

In JavaScript, 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
- ✓ Objects are always objects

Types of Object

Built –in Objects

- Defined by JavaScript (such as Math, Date, String, Array)

User-Defined Objects

- Objects which user (we) create)

Host Objects

- Always available to JavaScript by browser environment (such as window, document, form)

Object Name

- Each object is uniquely identified by a name or ID.
- 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”.
Or By creating instance of Object
 3. Define an object constructor, and then create objects of the constructed type.

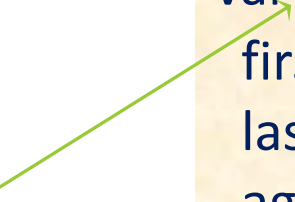
Using an Object Literal

- Easiest way to create a JavaScript Object.
- Using an object literal, you both define and create an object in one statement.
- An object literal is a list of name: value pairs (like age:10) inside curly braces {}.
- The following example creates a new JavaScript object with 3 properties:

Example:

**Person is
a object**

```
var person = {
  firstName: "Prasad",
  lastName: "Koyande",
  age: 10,
};
```



Example

```
<html>  
<body>  
<script>  
emp={id:"VP-179",name:"Prasad",salary:50000}  
document.write(emp.id+" "+emp.name+" "+emp.salary);  
</script>  
</body>  
</html>
```



OUTPUT

VP-179 Prasad 50000

Using “new” keyword

- **new keyword** is used to create object.

- Syntax: `var objectname=new Object();`

- Example:

```
var person = new Object();  
person.firstName = "Prasad";  
person.lastName = "Koyande";  
person.age = 10;
```

Example

```
<html>
<body>
<script>
var emp=new Object();
emp.id="VP-179";
emp.name="Prasad Koyande";
emp.salary=50000;
document.write(emp.id+" "+emp.name+" "+emp.salary);
</script>
</body>
</html>
```



OUTPUT

VP-179 Prasad Koyande
50000

By using Object Constructor

- Here, you need to create function with arguments.
- Each argument value can be assigned in the current object by using this keyword.
- The **this keyword** refers to the current object.

■ Example:

```
function person(firstName, lastName, age)
{
this. firstName = firstName;
this. lastName = lastName;
this. age = age;
}
p=new person("Prasad","Koyande",10);
document.write(p.firstName+" "+p.lastName+" "+p.age);
```

Example

```
<html> <body>  
<script>  
function emp(id,name,salary)  
{  
this.id=id;  
this.name=name;  
this.salary=salary;  
}  
e=new emp("VP-179","Prasad Koyande",5000);  
document.write(e.id+" "+e.name+" "+e.salary);  
</script>  
</body> </html>
```



OUTPUT

VP-179 Prasad Koyande 5000

Property

- 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.
- The syntax for accessing the property of an object is:

```
objectName.property // person.age
```

```
objectName["property"] // person["age"]
```

```
objectName[expression] // x = "age"; person[x]
```

Dot Operator

- The properties and methods associated with any object can be accessed by using `.` Operator.
- Example, `emp.id` or `op.add();`
- Also used to how to inteact with objects, methods, events and properties.
- Dot operator is also used to add new property.
- Example, `emp.designation="Lecturer";`

Accessing properties with dot operator

```
<html>  <body>
<h2>JavaScript Object Properties</h2>
<script>
var person = {
    firstname:"Prasad",
    lastname:"Koyande",
    age:10,
};
document.write(person.firstname+"<br>");
document.write(person.lastname);
</script>
</body>  </html>
```

OUTPUT



JavaScript Object Properties

Prasad

Koyande

Adding properties with dot operator

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

```
<script>
```

```
var person =
```

```
{
```

```
  firstname:"xyz",
```

```
  lastname:"abc",
```

```
  age:10
```

```
};
```

```
person.std = "Fifth";
```

```
document.write(person.firstname+" "+ "is in "+person.std+"  standard");
```

```
</script>
```

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



OUTPUT



xyz is in Fifth standard

Methods

- JavaScript methods are actions that can be performed on objects.
- A JavaScript function is a block of code designed to perform a particular task.
- A JavaScript function is defined with the **function keyword**, followed by a **name**, followed by parentheses **()**.
- The parentheses may include parameter names separated by commas:
(parameter1, parameter2, ...)

Methods

➤ The code to be executed, by the function, is placed inside curly brackets: {}

➤ Syntax:

```
function name(parameter1, parameter2, parameter3)
{
    // code to be executed
}
```

Methods-Example 1)

```
<html>
<body>
<script>
function op_add(p1, p2)
{
  return p1 + p2;
}
document.write("Addition is="+op_add(4, 5));
</script>
</body>
</html>
```



OUTPUT



Addition is=9

Methods- Example 2)

```
<script>
var person =
{
  firstname:"Prasad",
  lastname:"Koyande",
  Fullname:function()
  {
    return this.firstname+" "+this.lastname;
  }
};
document.write("Person Detail is="+person.Fullname());
</script>
```



OUTPUT

Person Detail is=Prasad
Koyande

Event

- ❖ An event is an action performed by user or web browser.
- ❖ In order to make a web pages more interactive, the script needs to be access the contents of the document and know when the user is interacting with it.
- ❖ Events may occur due to:
 - 1) a document loading
 - 2) user clicking on mouse button
 - 3) browser screen changing size

Here are some examples of HTML events:

- An HTML web page has finished loading
- An HTML input field was changed
- An HTML button was clicked

Event Handling

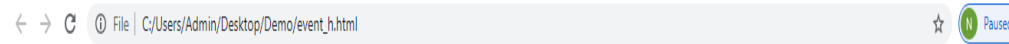
Event handlers can be used to handle, and verify, user input, user actions, and browser actions:

| Event | Description |
|-------------|--|
| onchange | An HTML element has been changed |
| onclick | The user clicks an HTML element |
| onmouseover | The user moves the mouse over an HTML element |
| onmouseout | The user moves the mouse away from an HTML element |
| onkeydown | The user pushes a keyboard key |
| onload | The browser has finished loading the page |

Example : Input (user clicking on button)

```
<html>
<head>
<script type="text/javascript">
function msg()
{
  alert("Hello CO5I students");
}
</script>
</head>
<body>
<center>
<p><h1>Welcome to Client-side scripting</h1></p>
<form>
<input type="button" value="click" onclick="msg()"/>
</form>
</body>
</html>
```

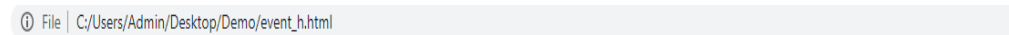
Example : Output



Welcome to Client-side scripting

click

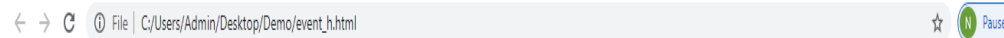
after clicking on **click** button



This page says
Hello IFSI students

OK

after clicking on **OK** button in alert



Welcome to Client-side scripting

click

Objects

- **Native Objects/ Built-in Objects**

are those objects supplied by JavaScript.

Examples of these are Math, Date, String, Number, Array, Image, etc.

- **Host Objects**

are objects that are supplied to JavaScript by the browser environment. Examples of these are window, document, forms, etc.

- **User-Defined Objects**

are those that are defined by you, the programmer.

Math: Math Properties

| Math Property | Description |
|---------------|----------------------------------|
| SQRT2 | Returns square root of 2. |
| PI | Returns Π value. |
| E | Returns Euler's Constant. |
| LN2 | Returns natural logarithm of 2. |
| LN10 | Returns natural logarithm of 10. |
| LOG2E | Returns base 2 logarithm of E. |
| LOG10E | Returns 10 logarithm of E. |

Example : Math

```
<html>
<head>
<title>JavaScript Math Object Properties</title>
</head>
<body>
<script type="text/javascript">
var value1 = Math.E;
document.write("E Value is :" + value1 + "<br>");
var value3 = Math.LN10;
document.write("LN10 Value is :" + value3 + "<br>");
var value4 = Math.PI;
document.write("PI Value is :" + value4 + "<br>");
</script> </body>
</html>
```

**OUTPUT**

E Value is :2.718281828459045
LN10 Value is :2.302585092994046
PI Value is :3.141592653589793

Math: Methods

| Methods | Description |
|---------|---|
| abs() | Returns the absolute value of a number. |
| acos() | Returns the arccosine (in radians) of a number. |
| ceil() | Returns the smallest integer greater than or equal to a number. |
| cos() | Returns cosine of a number. |
| floor() | Returns the largest integer less than or equal to a number. |
| log() | Returns the natural logarithm (base E) of a number. |
| max() | Returns the largest of zero or more numbers. |
| min() | Returns the smallest of zero or more numbers. |
| pow() | Returns base to the exponent power, that is base exponent. |

Example : Math

```
<html>
<head>
<title>JavaScript Math Object Methods</title>
</head>
<body>
<script type="text/javascript">
var value = Math.abs(-20);
document.write("ABS Value : " + value + "<br>");
var value = Math.tan(5);
document.write("TAN Value : " + value + "<br>");
</script>
</body>
</html>
```



OUTPUT

ABS Value : 20
TAN Value : -3.380515006246586

Date

- Date is a data type.
- Date object manipulates date and time.
- Date() constructor takes no arguments.
- Date object allows you to get and set the year, month, day, hour, minute, second and millisecond fields.
- Syntax:
`var variable_name = new Date();`

Example:

```
var current_date = new Date();
```

Date

| Methods | Description |
|-------------------|---|
| Date() | Returns current date and time. |
| getDate() | Returns the day of the month. |
| getDay() | Returns the day of the week. |
| getFullYear() | Returns the year. |
| getHours() | Returns the hour. |
| getMinutes() | Returns the minutes. |
| getSeconds() | Returns the seconds. |
| getMilliseconds() | Returns the milliseconds. |
| getTime() | Returns the number of milliseconds since January 1, 1970 at 12:00 AM. |

Date

| Methods | Description |
|---------------------|--|
| getTimezoneOffset() | Returns the timezone offset in minutes for the current locale. |
| getMonth() | Returns the month. |
| setDate() | Sets the day of the month. |
| setFullYear() | Sets the full year. |
| setHours() | Sets the hours. |
| setMinutes() | Sets the minutes. |
| setSeconds() | Sets the seconds. |
| setMilliseconds() | Sets the milliseconds. |
| setTime() | Sets the number of milliseconds since January 1, 1970 at 12:00 AM. |

Date

| Methods | Description |
|------------------|--|
| setMonth() | Sets the month. |
| toDateString() | Returns the date portion of the Date as a human-readable string. |
| toLocaleString() | Returns the Date object as a string. |
| toGMTString() | Returns the Date object as a string in GMT timezone. |
| valueOf() | Returns the primitive value of a Date object. |

Example : Date

```

<html>
<body>
<h2>Date Methods</h2>
<script type="text/javascript">
var d = new Date();
document.write("<b>Locale String:</b> " + d.toLocaleString()+"<br>");
document.write("<b>Hours:</b> " + d.getHours()+"<br>");
document.write("<b>Day:</b> " + d.getDay()+"<br>");
document.write("<b>Month:</b> " + d.getMonth()+"<br>");
document.write("<b>FullYear:</b> " + d.getFullYear()+"<br>");
document.write("<b>Minutes:</b> " + d.getMinutes()+"<br>");
</script>
</body>
</html>

```

OUTPUT

Date Methods

Locale String: 7/3/2020, 5:23:19 PM

Hours: 17

Day: 5

Month: 6

FullYear: 2020

Minutes: 23

String

- String objects are used to work with text.
- It works with a series of characters.

Syntax:

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

Example:

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

Properties:

| Properties | Description |
|-------------|--|
| length | It returns the length of the string. |
| constructor | It returns the reference to the String function that created the object. |

String: Methods


| Methods | Description |
|---------------|--|
| charAt() | It returns the character at the specified index. |
| charCodeAt() | It returns the ASCII code of the character at the specified position. |
| concat() | It combines the text of two strings and returns a new string. |
| indexOf() | It returns the index within the calling String object. |
| match() | It is used to match a regular expression against a string. |
| replace() | It is used to replace the matched substring with a new substring. |
| search() | It executes the search for a match between a regular expression. |
| slice() | It extracts a session of a string and returns a new string. |
| split() | It splits a string object into an array of strings by separating the string into the substrings. |
| toLowerCase() | It returns the calling string value converted lower case. |
| toUpperCase() | Returns the calling string value converted to uppercase. |

Example : String

```

<html>
  <body>
    <script type="text/javascript">
      var str = "A JavaScript";
      document.write("<b>Char At:</b> " + str.charAt(4)+"<br>");
      document.write("<b>CharCode At:</b> " + str.charCodeAt(0)+"<br>");
      document.write("<b>Index of:</b> " + str.indexOf("p")+"<br>");
      document.write("<b>Lower Case:</b> " + str.toLowerCase()+"<br>");
      document.write("<b>Upper Case:</b> " + str.toUpperCase()+"<br>");
    </script>
  </body>
</html>

```



Char At: v
CharCode At: 65
Index of: 10
Lower Case: a javascript
Upper Case: A JAVASCRIPT

Window

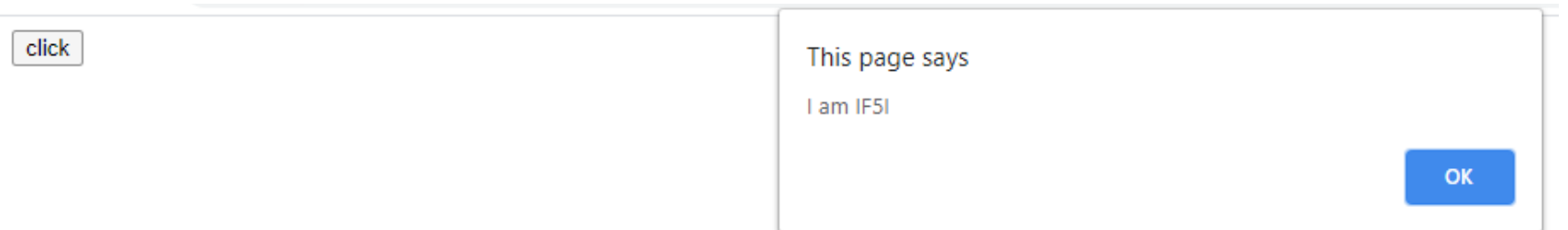
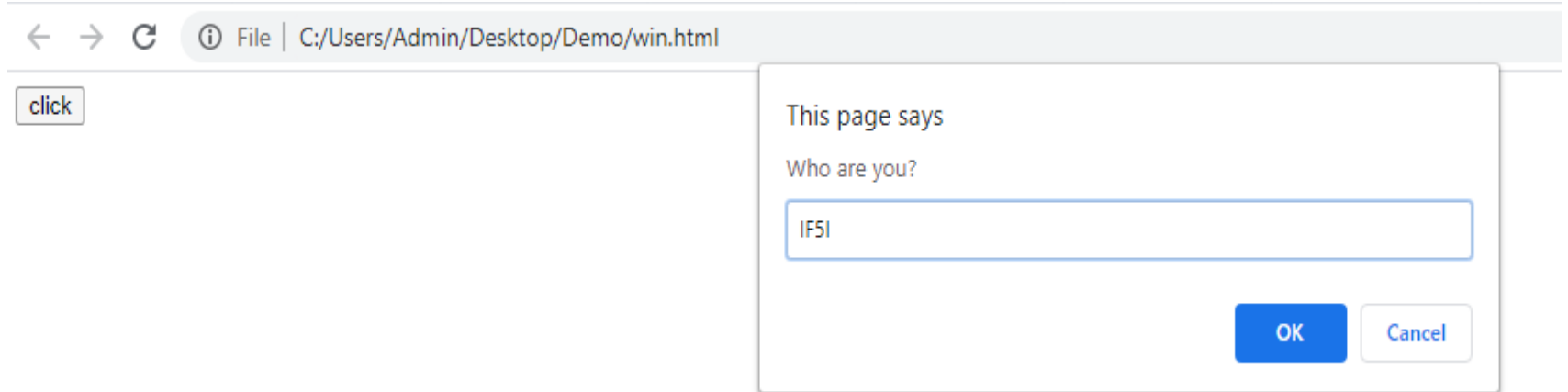
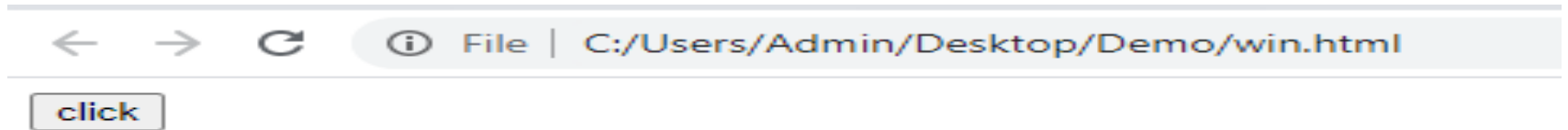
- ✓ The **window object** represents a window in browser.
- ✓ An object of window is created automatically by the browser.
- ✓ Window is the object of browser, **it is not the object of javascript.**

| Method | Description |
|-----------|---|
| alert() | displays the alert box containing message with ok button. |
| confirm() | displays the confirm dialog box containing message with ok and cancel button. |
| prompt() | displays a dialog box to get input from the user. |
| open() | opens the new window. |
| close() | closes the current window. |

Example : window

```
<script type="text/javascript">  
function msg()  
{  
var a= window.prompt("Who are you?");  
window.alert("I am "+a);  
}  
</script>  
<input type="button" value="click" onclick="msg()">
```

window: output



DOM getElementById() Method

- ✓ The `getElementById()` method returns the elements that has given ID which is passed to the function.
- ✓ This function is widely used in web designing to change the value of any particular element or get a particular element.
- ✓ Syntax: `document.getElementById(element_id) ;`

Parameter: This function accepts single parameter *element_id* which is used to hold the ID of element.

Return Value: It returns the object of given ID. If no element exists with given ID then it returns null.

DOM getElementById() Method

```

<html>
<body>
<p id="demo">Click the button to change the color of this paragraph.</p>
<button onclick="myFunction()">change color</button>
<script>
function myFunction()
{
  var x = document.getElementById("demo");
  x.style.color = "red";
}
</script>
</body>
</html>
  
```



Click the button to change the color of this paragraph.

change color

Click the button to change the color of this paragraph.

change color

1.3 Values and Variables

- ❑ A JavaScript variable is simply a name of storage location.
- ❑ Two types of variables in JavaScript :
 - local variable
 - global variable.

There are some rules while declaring a JavaScript variable (also known as identifiers).

- ❖ Name must start with a letter (a to z or A to Z), underscore(_), or dollar(\$) sign.
- ❖ After first letter we can use digits (0 to 9), for example value1.
- ❖ JavaScript variables are case sensitive, for example x and X are different variables.

Local Variables

A JavaScript local variable is declared inside block or function.

It is accessible within the function or block only.

For example:

```
<script>  
function abc()  
{  
var x=10;  
}  
</script>
```

Value of x is
10

```
<script>  
If(10<13)  
{  
var y=20;//JavaScript local variable  
}  
</script>
```


Global Variables

A JavaScript global variable is accessible from any function.

A variable i.e. declared outside the function or declared with window object is known as global variable.

For example:

```
<html>
<body>
<script>
var data=200; //global variable
function a()
{
document.write(data);
}
```

```
function b()
{
document.write(data);
}
a(); //calling JavaScript function
b();
</script>
</body>
</html>
```

200 200

Global Variables

A JavaScript global variable is declared outside the function or declared with window object.

It can be accessed from any function.

For example:

```
<html>
<body>
<script>
var value=50; //global variable
function a()
{
alert(value);
}
```

```
a();
</script>
</body>
</html>
```

This page says

50

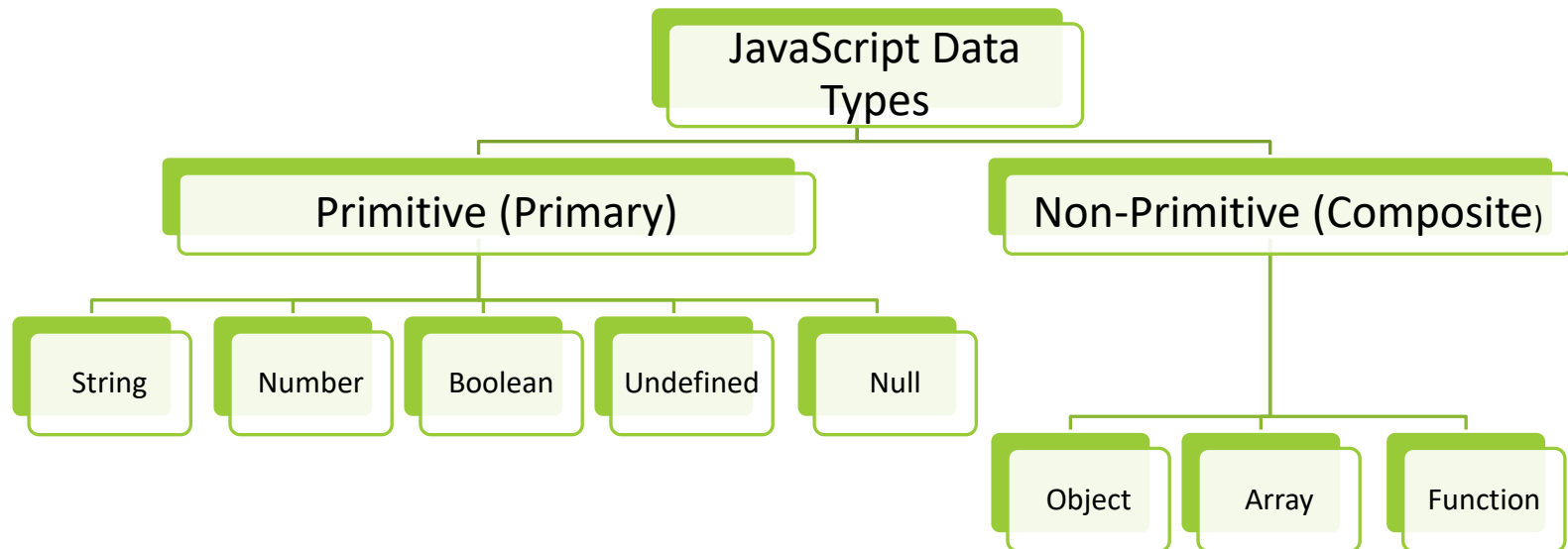
OK

Data Types

- ✓ JavaScript provides different **data types** to hold different types of values.
- ✓ There are two types of data types in JavaScript:
 1. Primitive data type
 2. Non-primitive (reference) data type/ Composite Data Types
- ✓ JavaScript is a **dynamic type language**, means you don't need to specify type of the variable.
- ✓ You need to use **var** here to specify the data type.
- ✓ It can hold any type of values such as numbers, strings etc.
- ✓ For example:

```
var a=40;//holding number  
var b="Info Technology"//holding string
```

Data Types



Data Types: Primitive

Primitive data types can hold only one value at a time.

1) The String Data Type

The *string* data type is used to represent textual data (i.e. sequences of characters).

Strings are created using single or double quotes surrounding one or more characters, as shown below:

```
var a = 'Welcome'; // using single quotes
```

```
var b = "Welcome"; // using double quotes
```

Data Types: Primitive

2) The Number Data Type

- ✓ The *number* data type is used to represent positive or negative numbers with or without decimal place.
- ✓ The Number data type also includes some special values which are: Infinity, -Infinity, NaN
- ✓ Example,

```
var a = 25; // integer
```

```
var b = 80.5; // floating-point number
```

```
var c = 4.25e+6; // exponential notation, same as 4.25e6 or 4250000
```

```
var d = 4.25e-6; // exponential notation, same as 0.00000425
```

Data Types: Primitive

3) The Boolean Data Type

✓ The Boolean data type can hold only two values: True/False

✓ Example,

```
var a = 2, b = 5, c = 10;
```

```
alert(b > a) // Output: true
```

```
alert(b > c) // Output: false
```

Data Types: Primitive

4) The Undefined Data Type

- ✓ The undefined data type can only have one value-the special value “undefined”.
- ✓ If a variable has been declared, but has not been assigned a value, has the value “undefined”.
- ✓ Example,

```
var a;
```

```
var b = “Welcome”;
```

```
alert(a) // Output: undefined
```

```
alert(b) // Output: Welcome
```


Data Types: Primitive

5) The Null Data Type

- ✓ A Null value means that there is no value.
- ✓ It is not equivalent to an empty string (" ") or zero or it is simply nothing.
- ✓ Example,

```
var a = null;
```

```
alert(a); // Output: null
```

```
var b = "Hello World!"
```

```
alert(b); // Output: Hello World!
```

```
b = null;
```

```
alert(b) // Output: null
```

Data Types: Non-primitive

1) The Object Data Type

- ✓ a complex data type that allows you to store collections of data.
- ✓ An object contains properties, defined as a key-value pair.
- ✓ A property key (name) is always a string, but the value can be any data type, like strings, numbers, Boolean, or complex data types like arrays, function and other objects.
- ✓ Example,

```
var car =
```

```
{ "modal": "SUZUKI", "color": "WHITE", "model": 2019 }
```

Data Types: Non-primitive

2) The Array Data Type

- ✓ An array is a type of object used for storing multiple values in single variable.
- ✓ Each value (also called an element) in an array has a numeric position, known as its index, and it may contain data of any data type-numbers, strings, Booleans, functions, objects, and even other arrays.
- ✓ The array index starts from 0, so that the first array element is arr [0].
- ✓ The simplest way to create an array is by specifying the array elements as a comma-separated list enclosed by square brackets, as shown in the example below:
- ✓ `var cities = ["London", "Paris", "New York"];`
- ✓ `alert(cities[2]);` // Output: New York

Data Types: Non-primitive

3) The Function Data Type

- ✓ The function is callable object that executes a block of code.
- ✓ Since functions are objects, so it is possible to assign them to variables, as shown in the example below:

```
var ab = function()
```

```
{
```

```
  return "Welcome";
```

```
}
```

```
alert(typeof ab);//output: function
```

```
alert(ab());//output:Welcome
```

Example : Non- Primitive

```
<html>
<body>
<h1>JavaScript Array</h1>
<script>
var stringArray = ["one", "two", "three"];
var mixedArray = [1, "two", "three", 4];
document.write(stringArray+"<br>");
document.write( mixedArray);
</script>
</body>
</html>
```

OUTPUT

JavaScript Array

one,two,three

1,two,three,4

Values/Literals

- ✓ They are **types** that can be assigned a single **literal** value such as the number 5.7, or a string of characters such as "hello".
- ✓ **Types of Literals:**
 - Array Literal
 - Integer Literal
 - Floating number Literal
 - Boolean Literal (include True and False)
 - Object Literal
 - String Literal

Array Literal

- ✓ an array literal is a list of expressions, each of which represents an array element, enclosed in a pair of square brackets ' [] ' .
- ✓ When an array is created using an array literal, it is initialized with the specified values as its elements, and its length is set to the number of arguments specified.
- ✓ Creating an empty array :

```
var tv = [ ];
```

Creating an array with four elements.

```
var tv = ["LG", "Samsung", "Sony", "Panasonic"]
```

Array Literal

✓ Comma in array literals:

- In the following example, the length of the array is four, and tv[0] and tv[2] are undefined.

```
var tv = [ , "Samsung" , , "Panasonic"]
```

- This array has one empty element in the middle and two elements with values. (tv[0] is "LG", tv[1] is set to undefined, and tv[2] is "Sony")

```
Var tv = ["LG" , , "Sony" , ]
```


Integer Literal

An **integer** must have at least one digit (0-9).

- No comma or blanks are allowed within an integer.
- It does not contain any fractional part.
- It can be either positive or negative if no sign precedes it is assumed to be positive.

In JavaScript, integers can be expressed in three different bases.

1. Decimal (base 10)

Example: 123, -20, 12345

Decimal numbers can be made with the digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 and there will be no leading zeros.

2. Hexadecimal (base 16)

Example: 7b, -14, 3039

Hexadecimal numbers can be made with the digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 and letters A, B, C, D, E, F or a, b, c, d, e, f. A leading 0x or 0X indicates the number is hexadecimal.

3. Octal (base 8)

Example: 173, -24, 30071

Octal numbers can be made with the digits 0, 1, 2, 3, 4, 5, 6, 7. A leading 0 indicates the number is octal.

Floating Number Literal

A floating number has the following parts.

- A decimal integer.
- A decimal point ('.').
- A fraction.
- An exponent.

The exponent part is an "e" or "E" followed by an integer, which can be signed (preceded by "+" or "-").

Example of some floating numbers :

- 8.2935
- -14.72
- 12.4e3 [Equivalent to 12.4×10^3]
- 4E-3 [Equivalent to $4 \times 10^{-3} \Rightarrow .004$]

Object Literal

An object literal is zero or more pairs of comma-separated list of property names and associated values, enclosed by a pair of curly braces.

In JavaScript an object literal is declared as follows:

1. An object literal without properties:

```
var userObject = {}
```

2. An object literal with a few properties :

```
var student = {  
  First-name : "Suresy",  
  Last-name : "Rayy",  
  Roll-No : 12  
};
```

Syntax Rules

- There is a colon (:) between property name and value.
- A comma separates each property name/value from the next.
- There will be no comma after the last property name/value pair.

String Literal

- JavaScript has its own way to deal with string literals.
- A string literal is zero or more characters, either enclosed in single quotation (') marks or double **quotation** (") marks. You can also use + operator to join strings.
- The following are the examples of string literals :

```
string1 = "w3resource.com"  
string1 = 'w3resource.com'
```

```
string1 = "1000"
```

- In addition to ordinary characters, you can include special characters in strings, as shown in the following.

```
string1 = "First line. \n Second line."
```

Comments

- ✓ The JavaScript comments are meaningful way to deliver message.
- ✓ It is used to add information about the code, warnings or suggestions so that end user can easily interpret the code.
- ✓ The JavaScript comment is ignored by the JavaScript engine i.e. embedded in the browser.

Types of JavaScript Comments

There are two types of comments in JavaScript.

1. Single-line Comment

It is represented by double forward slashes (//).
It can be used before and after the statement.

```
<script>  
// It is single line comment  
document.write("hello javascript");  
</script>
```

Types of JavaScript Comments

There are two types of comments in JavaScript.

2. Multi-line Comment

It can be used to add single as well as multi line comments.

It is represented by forward slash with asterisk then asterisk with forward slash.

```
<script>
```

```
/* It is multi line comment.
```

```
It will not be displayed */
```

```
document.write("example of javascript multiline comment");
```

```
</script>
```

1.4 Operators and Expression

JavaScript operators are symbols that are used to perform operations on operands.

1. Arithmetic Operators
2. Comparison (Relational) Operators
3. Bitwise Operators
4. Logical Operators
5. Assignment Operators
6. Special Operators

Arithmetic Operator

✓ used to perform arithmetic operations on the operands.

| Operator | Description | Example |
|----------|----------------|--|
| + | Addition | $10+20 = 30$ |
| - | Subtraction | $20-10 = 10$ |
| * | Multiplication | $10*20 = 200$ |
| / | Division | $20/10 = 2$ |
| % | Modulus | $20\%10 = 0$ |
| ++ | Increment | <code>var a=10; a++;</code> Now a = 11 |
| -- | Decrement | <code>var a=10; a--;</code> Now a = 9 |

Comparison Operator

➤ compares the two operands

| Operator | Description | Example |
|----------|------------------------------------|-----------------|
| == | Is equal to | 10==20 = false |
| === | Identical (equal and of same type) | 10===20 = false |
| != | Not equal to | 10!=20 = true |
| !== | Not Identical | 20!==20 = false |
| > | Greater than | 20>10 = true |
| >= | Greater than or equal to | 20>=10 = true |
| < | Less than | 20<10 = false |
| <= | Less than or equal to | 20<=10 = false |

Bitwise Operator

✓ The bitwise operators perform bitwise operations on operands.

| Operator | Description | Example |
|----------|-------------------------------|---------------------------------|
| & | Bitwise AND | 5 & 1 = 1 0101 & 0001 = 0001 |
| | Bitwise OR | 5 1 = 5 0101 0001 = 0101 |
| ^ | Bitwise XOR | 5 ^ 1 = 4 0101 ^ 0001 = 0100 |
| ~ | Bitwise NOT | ~ 5 = 10 → ~0101 = 1010 |
| << | Bitwise Left Shift | 5 << 1 = 10 → 0101 << 1 = 1010 |
| >> | Bitwise Right Shift | 5 >> 1 = 2 → 0101 >> 1 = 0010 |
| >>> | Bitwise Right Shift with Zero | 5 >>> 1 = 2 → 0101 >>> 1 = 0010 |

Logical Operator

| Operator | Description | Example |
|----------|-------------|---|
| && | Logical AND | $(10==20 \ \&\& \ 20==33) = \text{false}$ |
| | Logical OR | $(10==20 \ \ 20==33) = \text{false}$ |
| ! | Logical Not | $!(10==20) = \text{true}$ |

Assignment Operator

| Operator | Description | Example |
|----------|---------------------|------------------------------|
| = | Assign | 10+10 = 20 |
| += | Add and assign | var a=10; a+=20; Now a = 30 |
| -= | Subtract and assign | var a=20; a-=10; Now a = 10 |
| *= | Multiply and assign | var a=10; a*=20; Now a = 200 |
| /= | Divide and assign | var a=10; a/=2; Now a = 5 |
| %= | Modulus and assign | var a=10; a%=2; Now a = 0 |

Special Operator

| Operator | Description |
|------------|--|
| (?:) | Conditional Operator returns value based on the condition. It is like if-else. |
| , | Comma Operator allows multiple expressions to be evaluated as single statement |
| delete | Delete Operator deletes a property from the object |
| in | In Operator checks if object has the given property |
| instanceof | checks if the object is an instance of given type |
| new | creates an instance (object) |
| typeof | checks the type of object |
| void | it discards the expression's return value |

Expression

- ✓ Any unit of code that can be evaluated to a value is an expression.
- ✓ Since expressions produce values, they can appear anywhere in a program where JavaScript expects a value such as the arguments of a function invocation.
- ✓ Types of Expression:
 1. Primary Expression
 2. Object and Array Initializers
 3. Property Access Expressions
 4. Function Definition Expression
 5. Invocation Expression

Primary Expression

- ✓ Primary expressions refer to stand alone expressions such as literal values, certain keywords and variable values.

'hello world'; // A string literal

23; // A numeric literal

true; // Boolean value true

sum; // Value of variable sum

this; // A keyword that evaluates to the current object.

Object and Array Initializers

- ✓ Object and array initializers are expressions whose value is a newly created object or array.
- ✓ Object initializer expressions uses curly brackets, and each subexpression is prefixed with a property name and a colon.
- ✓ Example, `var emp={ name:"Yogita", branch:"IF"};`

OR

```
var person={ };  
person.name="Yogita";  
person.branch="IF";
```

- ✓ An array initializer is a comma-separated list of expressions surrounded with a square brackets.
- ✓ Example, `var tv=["LG", "Samsung"];`

Property Access Expressions

- ✓ A property access expression evaluates to the value of an object property or an array element.
- ✓ JavaScript defines two syntaxes for property access:

```
expression.identifier;  
expression[identifier];
```

- ✓ Exmample,

```
emp.firstName;  
emp[lastName];
```

Function Definition Expression

- ✓ A function expression is part of a variable assignment expression and may or may not contain a name.
- ✓ Since this type of function appears after the assignment operator =, it is evaluated as an expression.
- ✓ Function expressions are typically used to assign a function to a variable.
- ✓ Function expressions are evaluated only when the interpreter reaches the line of code where function expressions are located.

```
var sq=function (x)
{ return x*x;
}
```

Invocation Expressions

- ✓ An *invocation expression* is JavaScript's syntax for calling (or executing) a function or method.
- ✓ It starts with a function expression that identifies the function to be called.
- ✓ The function expression is followed by an open parenthesis, a comma-separated list of zero or more argument expressions, and a close parenthesis.
- ✓ When an invocation expression is evaluated, the function expression is evaluated first, and then the argument expressions are evaluated to produce a list of argument values.

Invocation Expressions

`f(0)` // *f* is the function expression; *0* is the argument expression.

`Math.max(x,y,z)` // *Math.max* is the function; *x*, *y* and *z* are the arguments.

`a.sort()` // *a.sort* is the function; there are no arguments.

1.5 if statement(Conditional)

- ✓ Conditional statements are used to perform different actions based on different conditions.
- ✓ In JavaScript we have the following conditional statements:
 - Use `if` to specify a block of code to be executed, if a specified condition is true
 - Use `else` to specify a block of code to be executed, if the same condition is false
 - Use `else if` to specify a new condition to test, if the first condition is false
 - Use `switch` to specify many alternative blocks of code to be executed

The if Statement

- ✓ Use **if** statement to specify a block of JavaScript code to be executed if a condition is true.

✓ Syntax:

```
if (condition)
{
    //block of code to be executed if the condition is true
}
```

✓ Example:

```
<html>
<body>
<script>
if (new Date().getHours() < 18)
{
    document.write("Good day!");
}
</script>
</body>
</html>
```

The else Statement

- ✓ Use **else** statement to specify a block of code to be executed if the condition is false.
- ✓ Syntax:

```
if (condition)
{
    // block of code to be executed if the condition
    is true
} else
{
    // block of code to be executed if the condition
    is false
}
```


The else Statement-Example

```
<html> <body>
<script>
if (new Date().getHours() < 18)
{
    document.write("Good day!");
}
else
{
    document.write("Good Evening!");
}
</script>
</body> </html>
```

The else if Statement

- ✓ Use **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
}
```

The else if Statement-Example

```
<html>
<body>
<script>
  var greeting;
  var time = new Date().getHours();
  if (time < 10)
  {
    greeting = "Good morning";
  }
  else if (time < 20)
  {
```

```
    greeting = "Good day";
  }
  else
  {
    greeting = "Good evening";
  }
  document.write(greeting);
</script>
</body>
</html>
```

The switch case Statement

- ✓ The switch statement is used to perform different actions based on different conditions.
- ✓ It is used to select one of many code blocks to be executed.
- ✓ Syntax:

```

switch(expression)
{
    case x:
        // code block
        break;
    case y:
        // code block
        break;
    default:
        // code block
}
  
```

This is how it works:

- 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.
- If there is no match, the default code block is executed.

1.6 The switch case-Example

```
<html>
<body>
<script>
var day;
switch (new Date().getDay())
{
  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";
}
document.write("Today is " + day);
</script>
</body>

</html>
```

default keyword

- ✓ default keyword specifies the code to run if there is no case match.
- ✓ The `getDay()` method returns the weekday as a number between 0 and 6.

If today is neither Saturday (6) nor Sunday (0), write a default message.

```
switch (new Date().getDay())
{
  case 6:
    text = "Today is Saturday";
    break;
  case 0:
    text = "Today is Sunday";
    break;
  default:
    text = "Looking forward to the Weekend";
}
```

1.7 JavaScript Loop Statement

The **JavaScript loops** are used *to iterate the piece of code* using for, while, do while or for-in loops.

There are four types of loops in JavaScript.

- ✓ for loop
- ✓ while loop
- ✓ do-while loop
- ✓ for-in loop

for Loop

✓ The JavaScript for loop *iterates the elements for the fixed number of times*. It should be used if number of iteration is known.

✓ Syntax:

```
for (initialization; condition; increment)
{
Code to be executed
}
```

✓ Example:

```
<script>
for (i=0; i<=10; i=i+2)
{
document.write(i + "<br/>")
}
</script>
```


do while Loop

- ✓ 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:

```
<script>
var i=21;
do{
document.write(i +"<br/>");
i++;
}while (i<=25);
</script>
```

✓ Example:

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

while Loop

✓ The **JavaScript while** loop loops through a block of code as long as a specified condition is true.

✓ Syntax:

```
while ( condition)
{
Code to be executed
}
```

✓ Example:

```
var i=11;
while (i<=20)
{
document.write(i + "<br/>");
i++;
}
```

For-in Loop

- ✓ The for..in statement loops through the properties of an object.
- ✓ The block of code inside the loop will be executed once for each property.

- ✓ Syntax:

```
for (variable_name in object)
{
Code to be executed
}
```

- ✓ Example:

```
<script type = "text/javascript">
var lang = { first : "C", second : "Java",third : "Python", fourth :
"PHP"};
    for (prog in lang)
    {
        document.write(lang[prog] + "<br >");
    }
</script>
```

```
C
Java
Python
PHP
```

break statement

- ✓ break statement breaks the loop and continues executing the code after the loop.
- ✓ The break statement can also be used to jump out of a loop.

✓ Example:

```
var text = "";  
var i;  
for (i = 0; i < 10; i++)  
{  
    if (i === 4)  
    { break;  
    }  
    text =text + "The number is " + i + "<br>";  
}  
document.write(text);  
</script>
```

The number is 0
The number is 1
The number is 2
The number is 3

continue statement

- ✓ Continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.
- ✓ Example:

```
var text = "";  
var i;  
for (i = 0; i <=6; i++)  
{  
    if (i === 4)  
        {continue;  
    }  
    text =text + "The number is " + i + "<br>";  
}  
document.write(text);  
</script>
```

```
The number is 0  
The number is 1  
The number is 2  
The number is 3  
The number is 5  
The number is 6
```

1.8 Querying and Setting Properties

- ✓ To obtain the value of a property, use . (dot) operator or square[] bracket.
- ✓ The left hand side should be an expression whose value is an object.
- ✓ If using dot (.) operator, the right-hand must be a simple identifier that names the property.
- ✓ If using square brackets, the value within the brackets must be an expression that evaluates to a string that contains the desired property name.

1.8 Querying and Setting Properties

✓ Example,

```
var name=author.lastname;
```

```
//get the "lastname " property of the book.
```

```
var title=book["main title"];
```

```
//get the "main title" property of the book.
```

✓ To create or set a property, use a dot or square brackets as you would to query the property, but put them on the left-hand side of an assignment expression:

✓ Example, `book.price=250;`

```
//create or set a property of price.
```

```
book["main title"]="JavaScript"
```

```
//set the "main title" property.
```

Deleting properties

The **delete** operator deletes a property from an object.

The delete operator deletes both the value of the property and the property itself.

Syntax:

```
delete var_name.property;
```

Example, delete person.name; or

```
delete person["name"];
```


Deleting properties

```
<html>
<body>
<script>
var a={name:"Priti",age:35};
document.write(a.name+" "+a.age+"<br>");
delete a.age; //delete property
document.write(a.name+" "+a.age);
</script>
</body>
</html>
```

Priti 35
Priti undefined

Property getter and setter

- ✓ Also known as Javascript assessors.
- ✓ Getters and setters allow you to control how important variables are accessed and updated in your code.
- ✓ JavaScript can secure better data quality when using getters and setters.

JavaScript Function or Getter?

```
<script>
// Create an object:
var person = {  firstName: "Chirag",  lastName : "Shetty",
               fullName : function()
               {
                 return this.firstName + " " + this.lastName;
               }
               };
document.write(person.fullName());
</script>
```

This example access fullName as a function: person.fullName().

JavaScript Function or Getter?

```
<script>
// Create an object:
var person = { firstName: "Yash ", lastName : "Desai",
              get fullName()
              {
                return this.firstName + " " + this.lastName;
              }
              };

// Display data from the object using a getter
document.write(person.fullName);
</script>
```

This example fullName as a property: person.fullName.

**Thank
You**

