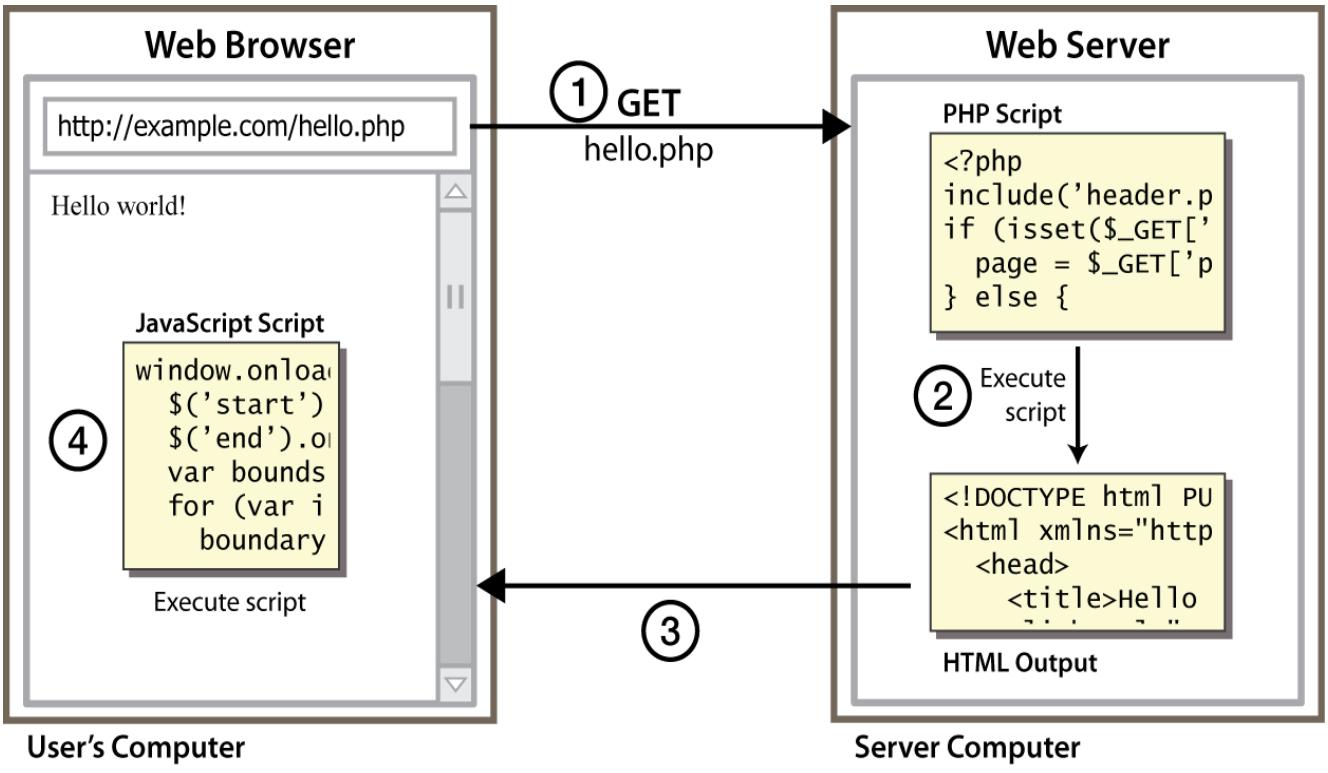
Introduction to Javascript

- JavaScript is a object-based scripting language and it is light weighted.
- It is first implemented by Netscape (with help from Sun Microsystems).
- JavaScript was created by Brendan Eich at Netscape in 1995 for the purpose of allowing code in web-pages (performing logical operation on client side).
- It is not compiled but translated.
- JavaScript Translator is responsible to translate the JavaScript code which is embedded in browser.
- Netscape first introduced a JavaScript interpreter in Navigator 2.
- The interpreter was an extra software component in the browser that was capable of interpreting JavaSript source code inside an HTML document.
- This means that web page developer need not have any other software other than a text editor to develop any web page.

Client Side Scripting



Why we Use JavaScript?

- Using HTML we can only design a web page but you can not run any logic on web browser like addition of two numbers, check any condition, looping statements (for, while), decision making statement (if-else) at client side.
- All these are not possible using HTML
- So to perform all these task at client side you need to use JavaScript.

Where it is used?

It is used to create interactive websites.

It is mainly used for:

- Client-side validation
- Dynamic drop-down menus
- Displaying data and time
- Build small but complete client side programs.
- Displaying popup windows and dialog boxes (like alert dialog box, confirm dialog box and prompt dialog box)
- Displaying clocks etc.

Why use client-side programming?

server-side programming (JSP) benefits:

- **security**: has access to server's private data; client can't see source code
- **compatibility**: not subject to browser compatibility issues
- **power**: can write files, open connections to servers, connect to databases, ...

Why use client-side programming?

JSP already allows us to create dynamic web pages. Why also use client-side scripting?

client-side scripting (JavaScript) benefits:

- **usability**: can modify a page without having to post back to the server (faster UI)
- **efficiency**: can make small, quick changes to page without waiting for server
- **event-driven**: can respond to user actions like clicks and key presses

What is Javascript?

a lightweight programming language ("scripting language")

- used to make web pages interactive
- insert dynamic text into HTML (ex: user name)
- **react to events** (ex: page load user click)
- get information about a user's computer (ex: browser type)
- perform calculations on user's computer (ex: form validation)
- A web standard (but not supported identically by all browsers)

NOT related to Java other than by name and some syntactic similarities

Javascript vs Java

•interpreted, not compiled

•more relaxed syntax and rules

- fewer and "looser" data types
- variables don't need to be declared
- errors often silent (few exceptions)

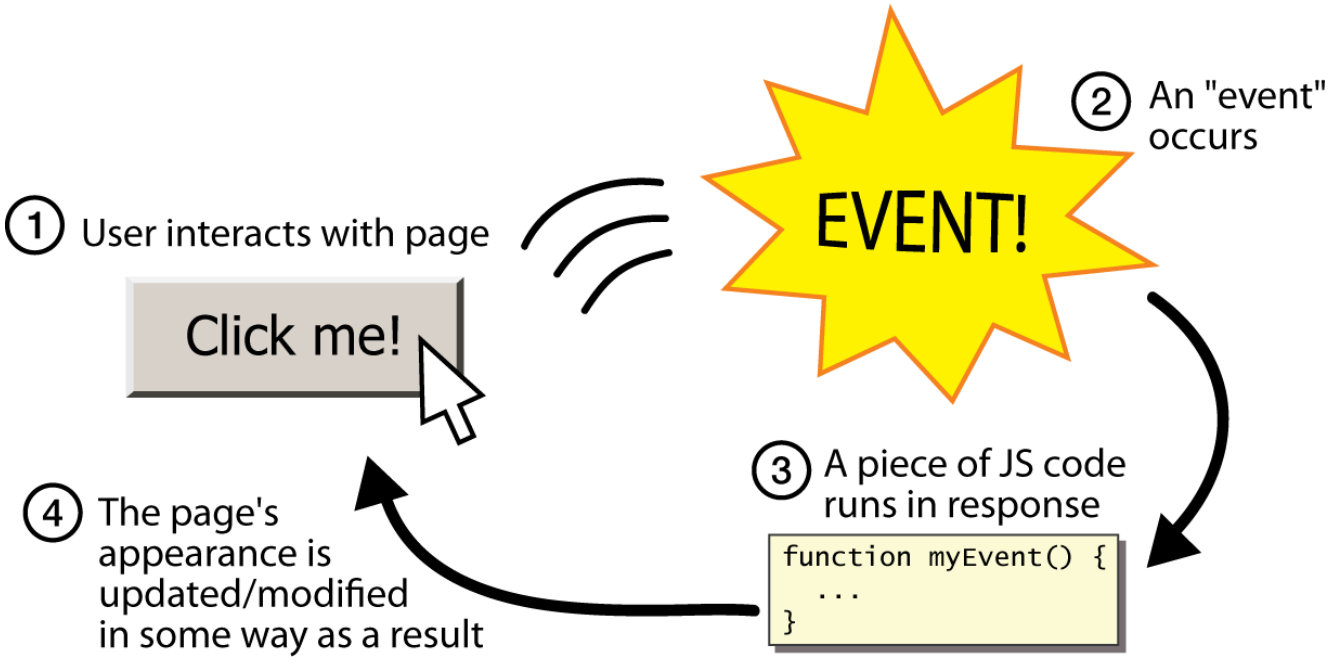
•key construct is the function rather than the class.

•Functions are used in many situations

•contained within a web page and integrates with its HTML/CSS content

Javascript vs Java

Event-driven programming



Features of JavaScript

- JavaScript is a object-based scripting language.
- Giving the user more control over the browser.
- Detecting the user's browser and OS,
- JavaScript is a scripting language and it is not java.
- JavaScript is interpreter based scripting language.
- JavaScript is case sensitive.
- JavaScript is object based language as it provides predefined objects.
- Most of the javascript control statements syntax is same as syntax of control statements in C language.

Uses of JavaScript

- Client-side validation
- Dynamic drop-down menus
- Displaying data and time
- Validate user input in an HTML form before sending the data to a server.
- Build forms that respond to user input without accessing a server.
- Change the appearance of HTML documents and dynamically write HTML into separate Windows.
- Open and close new windows or frames.
- Manipulate HTML "layers" including hiding, moving, and allowing the user to drag them around a browser window.
- Displaying popup windows and dialog boxes (like alert dialog box, confirm dialog box and prompt dialog box)

Limitations of JavaScript

We cannot treat JavaScript as a full-fledged programming language. It lacks the following important features when Javascript run by brower –

- Client-side JavaScript does not allow the reading or writing of files.
- JavaScript cannot be used for networking applications because there is no such support available.
- JavaScript doesn't have any multi-threading or multiprocessor capabilities.

Once again, JavaScript is a lightweight, interpreted programming language that allows you to build interactivity into otherwise static HTML pages.

History

- JavaScript was first implemented by Netscape (with help from Sun Microsystems).
- JavaScript was created by Brendan Eich at Netscape in 1995 for the purpose of allowing code in web-pages (performing logical operation on client side).

Writing JavaScript code

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

```
<script ...>
  JavaScript code
</script>
```


JavaScript code inside body tag

- JavaScript code is placed in the <body> section of an HTML page.

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

     <h1>A Web Page</h1>
     <p id="demo">A Paragraph</p>
     <button type="button" onclick="myFunction()">Try
                 it</button>
     <script>
         function myFunction() {
             document.getElementById("demo").innerHTML =
                     "Paragraph changed.";
         }
     </script>

 </body>
</html>
```

JavaScript code inside head tag

- JavaScript code is placed in the <head> section of an HTML page.

```
<!DOCTYPE html>
<html>
  <head>
     <script>
         function myFunction() {
             document.getElementById("demo").innerHTML =
                     "Paragraph changed.";
         }
     </script>
 <head>
 <body>
     <h1>A Web Page</h1>
     <p id="demo">A Paragraph</p>
     <button type="button" onclick="myFunction()">Try
                 it</button>
 </body>
</html>
```


External JavaScript file

- Scripts can also be placed in external files:
- JavaScript files have the file extension **.js**.
- External file: myScript.js

```
function myFunction() {
  document.getElementById("demo").innerHTML =
                        "Paragraph changed.";
}
```

External JavaScript file

- To use an external script, put the name of the script file in the src (source) attribute of a <script> tag:
- Example

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

- You can place an external script reference in <head> or <body> as you like.
- The script will behave as if it was located exactly where the <script> tag is located.

Course : Javascript

Comment

- Comment is a statement which is not processed by javascript engine.
- It is useful to explain the code that is written for what purpose.
- There are two types of comments in JavaScript

  -> Single-line Comment

  -> Multi-line Comment

Variable

- JavaScript variables are containers for storing data value
- Variable in java script can store any type of value.
- Javascript did not provide any data types for declaring variables
- Java script is loosely typed language.
- We can use a variable directly without declaring it.
- var and let keyword can be used to declare variable.

Variable

- **To declare single variable**

```
let message;
 message = 'Hello'; // store the string
```

- **To declare multiple variables in one line:**

```
let user = 'John', age = 25, message = 'Hello';
```

- **The multi-line variant is a bit longer, but easier to read:**

```
let user = 'John’;
let age = 25;
let message = 'Hello'
```


Variable

- **Multiple variables can be declared in multiline style:**

```
let user = 'John',
age = 25,
message = 'Hello';
```

- **Or even in the “comma-first” style:**

```
let user = 'John'
, age = 25
, message = 'Hello';
```

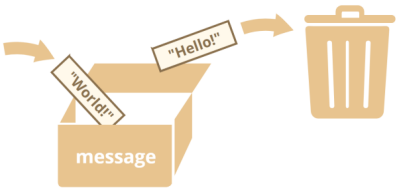
How variable works?

let message = “Hello”

The variable message can be imagined as a box labeled "message" with the value "Hello!" in it:

We can also change it as many times as we want:

message = 'World!'; // value changed



Difference between let and var :

- **let** variables cannot be used before declaration line whereas var variable can be used.
- **let** can be only available inside the **scope** it's declared.
- **var** can be accessed outside the **scope**.

```
function varTest() {
  var x = 1;
  if (true) {
    var x = 2;  // same variable!
    console.log(x);  // 2
  }
  console.log(x);  // 2
}
```

```
function letTest() {
  let x = 1;
  if (true) {
    let x = 2;  // different variable
    console.log(x);  // 2
  }
  console.log(x);  // 1
}
```

Types of variables

Types of Variable in JavaScript

-> Local Variable

-> Global Variable

Local variables

- A variable which is declared inside block or function is called local variable.
- It is accessible within the function or block only.
- For example:

```
<script>
    function abc() {
        let x=10;  //local variable
    }
</script>
```

or

```
<script>
    if(10<13) {
        let y=20;//javascript local variable
    }
</script>
```

Global variable

- A 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:

```
<script>
        var value=10;//global variable

        function a() {
            alert(value);
        }

        function b() {
            alert(value);
        }
</script>
```


Declaring global through window object

- The best way to declare global variable in javascript is through the window object.
- For example:

```
window.value=20;
```

- It can be declared inside any function and can be accessed from any function.

```
function m() {
    window.value=200; //declaring global variable by window object
}
function n() {
    alert(window.value); //accessing global variable from other function
}
```

Rules to declare a variable

- 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.
- Reserve words are not allowed
- No space inbetween is allowed
- No special symbols (except _ and $) are allowed

Constants

- Variables declared using const are called “constants”.
- They cannot be reassigned. An attempt to do so would cause an error.
- To declare a constant (unchanging) variable, use const instead of let
- Example
  const myBirthday = '18.04.1982';
  myBirthday = '01.01.2001'; // error, can't reassign the constant!
- When a programmer is sure that a variable will never change, they can declare it with const to guarantee and clearly communicate that fact to everyone.

Constants

- There is a widespread practice to use constants as aliases for difficult-to-remember values that are known prior to execution.
- Such constants are named using capital letters and underscores.
- For instance, let's make constants for colors in so-called "web" (hexadecimal) format:

```
const COLOR_RED = "#F00";
const COLOR_GREEN = "#0F0";
const COLOR_BLUE = "#00F";
const COLOR_ORANGE = "#FF7F00";

// ...when we need to pick a color
let color = COLOR_ORANGE;
```


Data types - Number

- The number type represents both integer and floating point numbers.

```
let n = 123;

n = 12.345;
```

- There are many operations for numbers, e.g. multiplication *, division /, addition +, subtraction -, and so on.
- Besides regular numbers, there are so-called "special numeric values" which also belong to this data type:

-> Infinity and
-> NaN.

Data types - Number

- Infinity represents the mathematical Infinity ∞.
- It is a special value that's greater than any number.
- We can get it as a result of division by zero:

  alert(1 / 0); // Infinity

- Or just reference it directly:

  alert(Infinity); // Infinity

Data types - Number

- NaN represents a computational error.
- It is a result of an incorrect or an undefined mathematical operation.
- Example

  alert("not a number" / 2); // NaN, such division is erroneous

  alert("not a number" / 2 + 5); // NaN

Data types - String

- A string in JavaScript must be surrounded by quotes.
- In JavaScript, there are 3 types of quotes.

  -> Double quotes: "Hello".
  -> Single quotes: 'Hello'.
  -> Backticks: `Hello`.

- Example

```
let str = "Hello";
let str2 = 'Single quotes are ok too';
let phrase = `can embed ${str}`;
let name = "John";
alert( `Hello, ${name}!` ); //embed a variable  Hello, John!
alert( `the result is ${1 + 2}` ); //embed an expression
                                   // the result is 3
```


Data types - Boolean

- The boolean type has only two values: true and false

```
let nameFieldChecked = true; // yes, name field is checked
let ageFieldChecked = false; // no, age field is not checked
```

- Boolean values also come as a result of comparisons:

```
let isGreater = 4 > 1;

alert( isGreater ); // true (the comparison result is "yes"
```

Data types – “null” value

- The special null value does not belong to any of the types.
- It forms a separate type ie object of its own which contains only the null value:

```
let age = null;
```

- null is not a “reference to a non-existing object” or a “null pointer” like in some other languages.
- It’s just a special value which represents “nothing”, “empty” or “value unknown”.

Data types – “undefined” value

- The special value undefined also stands apart. It makes a type of its own, just like null.
- The meaning of undefined is “value is not assigned”.
- If a variable is declared, but not assigned, then its value is undefined:

```
let x;
alert(x); // shows "undefined"
```

- Technically, it is possible to assign undefined to any variable:

```
let x = 123;
x = undefined;
alert(x); // "undefined"
```

typeof

- The typeof operator returns the type of the argument.
- It’s useful when we want to process values of different types differently or just want to do a quick check.
- It supports two forms of syntax:

  -> As an operator: typeof x.

  -> As a function: typeof(x).

typeof

- The call to typeof x returns a string with the type name

```
typeof undefined       // "undefined"
typeof 0               // "number"
typeof true            // "boolean"
typeof "foo"           // "string"
typeof Symbol("id")    // "symbol"
typeof Math            // "object"  (1)
typeof null            // "object"  (2)
typeof alert           // "function"  (3)
```

Arithmetic Operators

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation (ES2016)
/	Division
%	Modulus (Remainder)
++	Increment
--	Decrement

Assignment Operators

Operator	Example	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
<<=	x <<= y	x = x << y
>>=	x >>= y	x = x >> y
>>>=	x >>>= y	x = x >>> y
&=	x &= y	x = x & y
^=	x ^= y	x = x ^ y
|=	x |= y	x = x | y
**=	x **= y	x = x ** y

Comparison Operators

Given that `x = 5`

Operator	Description	Comparing	Returns
==	equal to	x == 8	false
		x == 5	true
		x == "5"	true
===	equal value and equal type	x === 5	true
		x === "5"	false
!=	not equal	x != 8	true
!==	not equal value or not equal type	x !== 5	false
		x !== "5"	true
		x !== 8	true
>	greater than	x > 8	false
<	less than	x < 8	true
>=	greater than or equal to	x >= 8	false
<=	less than or equal to	x <= 8	true

Logical Operators

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

Conditional Operator

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

- Syntax

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

- Example

```
var voteable = (age < 18) ? "Too young":"Old enough";
```


Comparing different types

Case	Value
2 < 12	true
2 < "12"	true
2 < "John"	false
2 > "John"	false
2 == "John"	false
"2" < "12"	false
"2" > "12"	true
"2" == "12"	false

Output statement

- JavaScript can "display" data in different ways:
  - Writing into an HTML element, using innerHTML.
  - Writing into the HTML output using document.write().
  - Writing into an alert box, using window.alert().
  - Writing into the browser console, using console.log()

Using innerHTML

- To access an HTML element, JavaScript can use the document.getElementById(id) method.
- The id attribute defines the HTML element.
- The innerHTML property defines the HTML content

```
<body>
    <h1>My First Web Page</h1>
    <p>My First Paragraph</p>

    <p id="demo"></p>

    <script>
      document.getElementById("demo").innerHTML = 5 + 6;
    </script>
</body>
```

Using document.write()

- Write some text directly to the HTML document.

```
document.write("Hello World!");
```

- The write() method is mostly used for testing.
- If it is used after an HTML document is fully loaded, it will delete all existing HTML.

Using window.alert()

- The alert() method displays an alert box with a specified message and an OK button.

```
Window.alert("Hello! I am an alert box!!");
```

- An alert box is often used if you want to make sure information comes through to the user.
- The alert box takes the focus away from the current window, and forces the browser to read the message.
- Do not overuse this method, as it prevents the user from accessing other parts of the page until the box is closed.

Using console.log()

- The **console.log()** is a function which is used to print any kind of variables defined before in it or to just print any message that needs to be displayed to the user.

    - `console.log(5 + 6);`

- `console.log(5 + 6);`

- `It is usually used while debugging.`

Conditional statement

- The if statement is used in JavaScript to execute the code if condition is true or false.

- There are three forms of if statement.
    - → if Statement
    - → if else statement
    - → if else if statement

Conditional statement - if

- Use if to specify a block of code to be executed, if a specified condition is true

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

- Example

```
if (hour < 18) {
    greeting = "Good day";
}
```


Conditional statement – else

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

```
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
}
```

Conditional statement – else

- Example

```
if (age < 18) {
    msg = "Not eligible to vote";
} else {
    msg = "Eligible to vote";
}
```


Conditional statement – if .. else .. if

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

```
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 statement – if .. else .. if

- Example

```
if (time < 10) {
        greeting = "Good morning";
} else if (time < 20) {
        greeting = "Good day";
} else {
        greeting = "Good evening";
}
```


Conditional statement – switch

- The switch statement is used to perform different actions based on different conditions

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

Conditional statement – switch

- Example

```
switch (grade) {
        case 'A': msg = "Good job”;
                  break;
        case 'B': msg = "Pretty good";
                  break;
        case 'C': msg = "Passed";
                  break;
        case 'D': msg = "Not so good";
                  break;
         case 'F': msg = "Failed";
                  break;
         default: msg = "Unknown grade";
    }
```


Conditional statement – switch

- Switch cases use strict comparison (===).
- The values must be of the same type to match.
- A strict comparison can only be true if the operands are of the same type.

Conditional statement – switch

In this example there will be no match for x:

```
var x = "0";
switch (x) {
  case 0:
    text = "Off";
    break;
  case 1:
    text = "On";
    break;
  default:
    text = "No value found";
}
```


Loop statements – while

- The purpose of a while loop is to execute a statement or code block repeatedly as long as an expression is true.

- Once the expression becomes false, the loop terminates.

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

Loop statements – while

- Example – Code to display natural number from 1 to 10.

```
i=1;
while (i <=10) {
    console.log(i)
    i++;
}
```


Loop statements – do..while

- 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.

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

Loop statements – do..while

- Example - Code to display natural number from 1 to 10

```
i=1;
  do {
     console.log(i);
     i++;
 }while (i <= 10);
```


Loop statements – for

- The 'for' loop is the most compact form of looping.

- It includes three important parts –

  - The **loop initialization** where we initialize our counter to a starting value. The initialization statement is executed before the loop begins.

  - The **test statement** which will test if a given condition is true or not. If the condition is true, then the code given inside the loop will be executed, otherwise the control will come out of the loop.

  - The **iteration statement** where you can increase or decrease your counter.

Loop statements – for

- The 'for' loop is the most compact form of looping.
- It includes three important parts –
  - The **loop initialization** where we initialize our counter to a starting value. The initialization statement is executed before the loop begins.
  - The **test statement** which will test if a given condition is true or not. If the condition is true, then the code given inside the loop will be executed, otherwise the control will come out of the loop.
  - The **iteration statement** where you can increase or decrease your counter.

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


Loop statements – for

- Example – Code to display natural number from 1 to 10

```
for(count = 1; count <= 10; count++) {
    console.log(count);
 }
```

Array

- An array is a special variable, which can hold more than one value at a time.
- Creating an Array - Using an array literal is the easiest way to create a JavaScript Array.
- Syntax:

  var array_name = [item1, item2, ...];
- Example

  var cars = ["Saab", "Volvo", "BMW"];

Array

- Array declaration can span multiple lines:
  - Example

```
var cars = [
  "Saab",
  "Volvo",
  "BMW"
];
```

- Array can be created using new keyword:
  - Example

```
var cars = new Array("Saab", "Volvo", "BMW");
```

Accessing elements of Array

- You access an array element by referring to the index number.
- name[0] is the first element. name[1] is the second element.
- Array indexes start with 0.
- This statement accesses the value of the first element in cars:
  - var name = cars[0];
- Example

```
var cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars[0];
```


Accessing elements of Array

- Access the Full Array

  With JavaScript, the full array can be accessed by referring to the array name:

```
var cars = ["Saab", "Volvo", "BMW"];
console.log(cars);
```

- An array can store elements of any type.

```
let arr = [ 'Apple', { name: 'John' }, true, function() { alert('hello'); } ];

// get the object at index 1 and then show its name
alert( arr[1].name ); // John

// get the function at index 3 and run it
arr[3](); // hello
```

Cycling array elements using for loop

- Accessing all array elements :

```
let arr = ["Apple", "Orange", "Pear"];

for (let i = 0; i < arr.length; i++) {
  alert( arr[i] );
}
```

- Accessing all array elements using for each :

```
let fruits = ["Apple", "Orange", "Plum"];

// iterates over array elements
for (let fruit of fruits) {
  alert( fruit );
}
```


Cycling array elements using for loop

- Technically, because arrays are objects, it is also possible to use for..in:

```
let arr = ["Apple", "Orange", "Pear"];

for (let key in arr) {
  alert( arr[key] ); // Apple, Orange, Pear
}
```

length property

- Length actually does not give the count of values in the array, but the greatest numeric index plus one example :

```
let fruits = [];
fruits[123] = "Apple";
alert( fruits.length ); // 124
```

- length property is that it's writable.

  If we increase it manually, nothing interesting happens. But if we decrease it, the array is truncated.

```
let arr = [1, 2, 3, 4, 5];
arr.length = 2; // truncate to 2 elements
alert( arr ); // [1, 2]
arr.length = 5; // return length back
alert( arr[3] ); // undefined: the values do not return
```


Array object methods

- Push

  Append the element to the end of the array:

```
let fruits = ["Apple", "Orange"];

fruits.push("Pear");

alert( fruits ); // Apple, Orange, Pear
```

- Pop

  Extracts the last element of the array and returns it:

```
let fruits = ["Apple", "Orange", "Pear"];

alert( fruits.pop() ); // remove "Pear" and alert it

alert( fruits ); // Apple, Orange
```

Array object methods

- shift

  Extracts the first element of the array and returns it:

```
let fruits = ["Apple", "Orange", "Pear"];

alert( fruits.shift() ); // remove Apple and alert it

alert( fruits ); // Orange, Pear
```

- unshift

  Add the element to the beginning of the array:

```
let fruits = ["Orange", "Pear"];

fruits.unshift('Apple');

alert( fruits ); // Apple, Orange, Pear
```


Array object methods

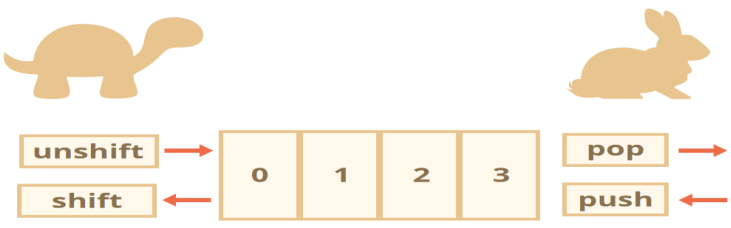
- Adding multiple elements :

  Methods push and unshift can add multiple elements at once:

```
let fruits = ["Apple"];

fruits.push("Orange", "Peach");
fruits.unshift("Pineapple", "Lemon");

// ["Pineapple", "Lemon", "Apple", "Orange", "Peach"]
alert( fruits );
```



Other Array object methods

- Deleteing element

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
delete fruits[0];          // Changes the first element in fruits to undefined
```

- Converting array to string

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
console.log(fruits.toString())  //Banana,Orange,Apple,Mango
```

- Merging two arrays (concat method)

Creates a new array by merging (concatenating) existing arrays:

```
var myGirls = ["Cecilie", "Lone"];
var myBoys = ["Emil", "Tobias", "Linus"];
var myChildren = myGirls.concat(myBoys);
```

The concat() method does not change the existing arrays. It always returns a new array.

Splice method

- Removing elements using splice

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(0, 1);        // Removes the first element of fruits
```

- The first parameter (0) defines the position where new elements should be added (spliced in).
- The second parameter (1) defines how many elements should be removed.

Splice method

- Syntax splice(start_index,not of deletions, new replace data)
- Removing and adding new elements using splice

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

- 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.

slice method

```
arr.slice([start], [end])
```

- slice returns a new array copying to it all items from index start to end (not including end).
- Both start and end can be negative, in that case position from array end is assumed.

```
let arr = ["t", "e", "s", "t"];

alert( arr.slice(1, 3) ); // e,s (copy from 1 to 3)

alert( arr.slice(-2) ); // s,t (copy from -2 till the end)
```

Iterate: forEach

- The **arr.forEach** method allows to run a function for every element of the array.

```
arr.forEach(function(item, index, array) {
  // ... do something with item
});
```

- Following code shows each element of the array:

  - ["Bilbo", "Gandalf", "Nazgul"].forEach(console.log);

  - ["Bilbo", "Gandalf", "Nazgul"].forEach((item, index, array) => {
    alert(`${item} is at index ${index} in ${array}`);
    });

Iterate: forEach

```
let names = ["Anil","Kiran","Suresh","Ramesh","Vishala"];
names.forEach(display);


   function display(item,index,arr){
    console.log(`Value in index ${index} is ${item}`)
}
```

Searching in array

- You can search an element in array by using **indexOf, lastIndexOf and includes** methods

- arr.indexOf(item, from)
  - ➢ – looks for item starting from index from, and returns the index where it was found, otherwise -1.

- arr.lastIndexOf(item, from)
  - ➢ – same, but looks for from right to left.

- arr.includes(item, from)
  - ➢ – looks for item starting from index from, returns true if found.

Searching in array

- Examples :

```
let arr = [1, 0, false];

alert( arr.indexOf(0) ); // 1
alert( arr.indexOf(false) ); // 2
alert( arr.indexOf(null) ); // -1

alert( arr.includes(1) ); // true
```

find and findIndex methods

- These methods are used to search in array of objects
- Syntax :

```
let result = arr.find(function(item, index, array) {
  // if true is returned, item is returned and iteration is stopped
  // for falsy scenario returns undefined
});
```

- The function is called for elements of the array, one after another:
  - item is the element.
  - index is its index.
  - array is the array itself.
- If it returns true, the search is stopped, the item is returned. If nothing found, undefined is returned.

find and findIndex methods

- These methods are used to search in array of objects
- Syntax :

```
let result = arr.find(function(item, index, array) {
  // if true is returned, item is returned and iteration is stopped
  // for falsy scenario returns undefined
});
```

- The function is called for elements of the array, one after another:
  - item is the element.
  - index is its index.
  - array is the array itself.
- If it returns true, the search is stopped, the item is returned. If nothing found, undefined is returned.

find and findIndex methods

- Example

```
let users = [
  {id: 1, name: "John"},
  {id: 2, name: "Pete"},
  {id: 3, name: "Mary"}
];

let user = users.find(item => item.id == 1);

alert(user.name); // John
```


Filter methods

- The find method looks for a single (first) element that makes the function return true whereas if there may be many occurances, we can use arr.filter(fn).

```
let results = arr.filter(function(item, index, array) {
  // if true item is pushed to results and the iteration
  // continuesreturns empty array if nothing found
});
```

Filter methods

- Example:

```
let users = [
  {id: 1, name: "John"},
  {id: 2, name: "Pete"},
  {id: 3, name: "Mary"}
];

// returns array of the first two users
let someUsers = users.filter(item => item.id < 3);

alert(someUsers.length); // 2
```


map methods

- The arr.map method calls the function for each element of the array and returns the array of results.

```
let result = arr.map(function(item, index, array) {
  // returns the new value instead of item
});
```

- Example that transform each element into its length:

```
let lengths = ["Bilbo", "Gandalf", "Nazgul"].map(item => item.length);
alert(lengths); // 5,7,6
```

sort method

- The sort method sorts the array in place, changing its element order.

- It returns the sorted array, but the returned value is usually ignored, as arr itself is modified.

- By default items are sorted as strings.

```
let arr = [ 1, 2, 15 ];
// the method reorders the content of arr
arr.sort();
console.log( arr );  // 1, 15, 2
```


sort method

- To use our own sorting order, we need to supply a function as the argument of arr.sort().

- The function should compare two arbitrary values and return.

```
function compareNumeric(a, b) {
  if (a > b) return 1;
  if (a == b) return 0;
  if (a < b) return -1;
}
let arr = [ 1, 2, 15 ];
arr.sort(compareNumeric);
console.log(arr);  // 1, 2, 15
```

sort method

```
let arr = [ {Rollno:1001,Name:'Ravi'},
            {Rollno:1005,Name:'Kiran'},
            {Rollno:1003,Name:'Anil'},
            {Rollno:1002,Name:'Govind'}
          ];

  arr.sort((v1,v2)=>{
    if(v1.Name > v2.Name) return 1;
    else if(v1.Name == v2.Name) return 0;
    else return -1;
   });
```


split method

- The split method splits the string into an array by the given delimiter.
- Example 1 :

```
let names = 'Bilbo, Gandalf, Nazgul';
let arr = names.split(', ');
for (let name of arr) {
  alert( `A message to ${name}.` ); // A message to Bilbo..
}
```

- Example 2 :

```
//If delimiter is empty then it would split the string
  into an array of letters:
let str = "test";
alert( str.split('') ); // t,e,s,t
```

split method

- The split method has an optional second numeric argument that limits on the array length.

- If it is provided, then the extra elements are ignored.

```
let st = 'Bilbo, Gandalf, Nazgul, Saruman’;
let arr = st.split(', ', 2);

console.log(arr); // Bilbo, Gandalf
```


join method

- The join method creates a string of array items joined by glue between them.

```
For instance:

let arr = ['Bilbo', 'Gandalf', 'Nazgul'];

let str = arr.join(';'); // glue the array into a
                            string using ;

alert( str ); // Bilbo;Gandalf;Nazgul
```

Map

- Map is a collection of keyed data items similar to Objects
- Map allows keys of any type.
- Methods and properties are:
  - new Map() – creates the map.
  - map.set(key, value) – stores the value by the key.
  - map.get(key) - returns the value by the key, undefined if key doesn't exist in map.
  - map.has(key) – returns true if the key exists, false otherwise.
  - map.delete(key) – removes the value by the key.
  - map.clear() – removes everything from the map.
  - map.size – returns the current element count.

Map

- Example

```
let map = new Map();

map.set('1', 'str1');   // a string key
map.set(1, 'num1');     // a numeric key
map.set(true, 'bool1'); // a boolean key

console.log( map.get(1)   ); // 'num1'
console.log( map.get('1') ); // 'str1'

console.log( map.size ); // 3
```

Iteration over Map

- For looping over a map, there are 3 methods:

```
map.keys()      – returns an iterable for keys,
map.values()    – returns an iterable for values,
map.entries()   – returns an iterable for entries
                  [key, value]
```


Iteration over Map

```
let recipeMap = new Map([
  ['cucumber', 500],
  ['tomatoes', 350],
  ['onion',    50]
]);

// iterate over keys (vegetables)
for (let vegetable of recipeMap.keys()) {
  alert(vegetable); // cucumber, tomatoes, onion
}

// iterate over values (amounts)
for (let amount of recipeMap.values()) {
  alert(amount); // 500, 350, 50
}
```

Function

- A function is a block of code designed to perform a particular task.
- Function is executed when "something" invokes it (calls it).

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

- Example

```
function product(p1, p2) {
  return p1 * p2;   // The function returns the product of p1 and p2
}
```


Function

- Function parameters are listed inside the parentheses () in the function definition.
- Function arguments are the values received by the function when it is invoked.
- Inside the function, the arguments (the parameters) behave as local variables.

Function - Local variables

- A variable declared inside a function is only visible inside that function.

- For example:

```
function showMessage() {
    let message = "Hello, I'm JavaScript!"; // local variable
    alert( message );
}

showMessage(); // Hello, I'm JavaScript!
alert( message ); // <-- Error! The variable is local to the function
```


Function – Outer variables

- The function has full access to the outer variable.
- It can modify it as well.
- Example

```
let userName = 'John';
function showMessage() {
    userName = "Bob"; // (1) changed the outer variable
    let message = 'Hello, ' + userName;
    alert(message);
}

alert( userName ); // John before the function call
showMessage();
alert( userName ); // Bob, the value was modified by the function
```

Function – Outer variables

- If a same-named variable is declared inside the function then it shadows the outer one.
- For instance, in the code below the function uses the local userName. The outer one is ignored:

```
let userName = 'John';
function showMessage() {
    let userName = "Bob"; // declare a local variable
    let message = 'Hello, ' + userName; // Bob
    alert(message);
}
// the function will create and use its own userName
showMessage();
alert( userName ); // John, unchanged, the function did not access the outer variable
```


Function – Parameter

- We can pass arbitrary data to functions using parameters (also called function arguments) .
- In the example below, the function has two parameters: from and text.

```
function showMessage(from, text) { // arguments: from, text
        alert(from + ': ' + text);
}
showMessage('Ann', 'Hello!'); // Ann: Hello! (*)
showMessage('Ann', "What's up?"); // Ann: What's up? (**)
```

- When the function is called in lines (*) and (**), the given values are copied to local variables from and text.

Function – Default Parameter

- Default function parameters allow named parameters to be initialized with default values if no value or undefined is passed.

```
function multiply(a, b = 1) {
  return a * b;
}

console.log(multiply(5, 2));
// expected output: 10

console.log(multiply(5));
// expected output: 5
```


Function – Returning a value

- A function can return a value back into the calling code as the result.
- Example

```
function sum(a, b) {
return a + b;
}

let result = sum(1, 2);
alert( result ); // 3
```

Function expressions

- There is another syntax for creating a function that is called a Function Expression.

```
let sayHi = function() {
      alert( "Hello" );
};
```

- We can copy a function to another variable:

```
function sayHi() {          // (1) create
   alert( "Hello" );
}
let func = sayHi;       // (2) copy
func(); // Hello        // (3) run the copy (it works)!
sayHi(); // Hello       //     this still works too
```


Arrow Functions

- An arrow function expression is a syntactically compact alternative to a regular function expression.

Regular function

```
let hello = function() {
  return "Hello World!";
}
```

Arrow function

```
let hello = ()  => {
  return "Hello World!";
}
```

Regular function

```
let sum=function(a,b) {
  return a+b;
}
```

Arrow function

```
let hello = (a,b)  => {
  return a+b;
}
```

Arrow Functions

- An arrow function expression is a syntactically compact alternative to a regular function expression.

Regular function

```
let hello = function() {
  return "Hello World!";
}
```

Arrow function

```
let hello = ()  => {
  return "Hello World!";
}
```

Regular function

```
let sum=function(a,b) {
  return a+b;
}
```

Arrow function

```
let hello = (a,b)  => {
  return a+b;
}
```


Arrow Functions

- If function has only one statement then arrow function can be written as

Arrow function

```
let hello = (a,b)  => {
  return a+b;
}
```

Single line arrow function

```
let sum=(a,b) => a+b;
```

Multi-line Arrow Functions

- Arrow functions can be of more then one line.

```
let sum = (a, b) => {  // the curly brace opens a multiline function
  let result = a + b;
  return result;          // if we use curly braces, then we need an explicit "return"
};

alert( sum(1, 2) ); // 3
```


Callback Functions

- In JavaScript, functions are objects and we can pass objects to functions as parameters.
- A callback function is a function that is passed as an argument to another function, to be “called back” at a later time.

```
function display(sm) {
  console.log(“Sum : “+sm);
}

function findSum(a,b,dispSum){
        let c = a+b;
  dispSum(c);
  }
```

```
function main() {
        let x = 100;
  let y = 200;
  findSum(x,y,display);
  }
```

Callback Functions

```
    function display(sm) {
      console.log("Sum : "+sm);
}


function findSum(a,b,dispSum){
            let c = a+b;
       dispSum(c);
       }

    function main() {
            let x = 100;
       let y = 200;
       findSum(x,y,display);
       }
```

```
function findSum(a,b,dispSum){
          let c = a+b;
          dispSum(c);
     }

   function main() {
          let x = 100;
    let y = 200;
    findSum(x,y, function(sm) {
                    console.log("Sum : "+sm);
                   });
     }
```


Callback Functions

```
nction findSum(a,b,dispSum){
        let c = a+b;
        dispSum(c);
   }

 function main() {
        let x = 100;
  let y = 200;
  findSum(x,y, function(sm) {
        console.log("Sum : "+sm);
    });
   }
```

```
function findSum(a,b,dispSum){
        let c = a+b;
        dispSum(c);
   }

 function main() {
        let x = 100;
  let y = 200;
  findSum(x,y, (sm) => {
                    console.log("Sum : "+sm);
                   });
   }
```

Synchronous and Asynchronous code

- Synchronous basically means that you can only execute one thing at a time.

- Asynchronous means that you can execute multiple things at a time and you don't have to finish executing the current thing in order to move on to next one.

Synchronous code

```
console.log('Before');
console.log('Reading user details from database');
sleepFor(3000);
console.log('User details read');
console.log('After');

function sleepFor( sleepDuration ){
    var now = new Date().getTime();
    while(new Date().getTime() < now + sleepDuration){
     /* do nothing */
    }
}
```

Asynchronous code

```
console.log('Before');
console.log('Reading user details from database');
setTimeout(()=>{
                    console.log('User details read');
                },2000);
console.log('After');
```


Asynchronous code

```
console.log('Before');
let user = getUser(1);
console.log("User : "+user);
console.log('After');

function getUser(id) {
   setTimeout(() => {
      console.log('Reading user details....');
      return({id:1001,name:'Amit'});
   },2000);
}
```

Asynchronous code

```
console.log('Before');
getUser(1,(user)=> {
    console.log("User : ",user);
});
console.log('After');

function getUser(id,callback) {
    setTimeout(() => {
        console.log('Reading user details....');
        callback({id:1001,name:'Amit'});
    },2000);
}
```


Promise

Objects

- Object is an entity having state and behavior (properties and method).
- JavaScript is an object-based language.
- JavaScript is template based not class based.
- Here, we don't create class to get the object. But, we directly create objects.

Creating Objects

- There are 3 ways to create objects.

1) By object literal

1) By creating instance of Object directly (using new keyword)

1) By using an object constructor (using new keyword)

Creating Object by object literal

- Syntax:

object={property1:value1,property2:value2.....propertyN:valueN}

- Property and value is separated by : (colon).

- Example

```
emp={id:102,name:"Shyam Kumar",salary:40000}
console.log(emp.id+" "+emp.name+" "+emp.salary);
```


Creating Object by instance of Object

- Syntax:

var objectname=new Object();

Here, new keyword is used to create object.

- Example:

```
var emp=new Object();
emp.id=101;
emp.name="Ravi Malik";
emp.salary=50000;
console.log(emp.id+" "+emp.name+" "+emp.salary);
```

Creating Object by 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 emp(id,name,salary){
   this.id=id;
   this.name=name;
   this.salary=salary;
}
e=new emp(103,"Vimal Jaiswal",30000);
console.log(e.id+" "+e.name+" "+e.salary);
```


Adding, modifying, removing and accessing property

```
let user = {
  name: “Kiran",
  age: 30
};
```

- **Adding new property to user**
  - user.branch = “CS”;
- **Modifying property**
  - user.name = “Kiran Kumar”;
- **Deleting the property from user**
  - delete user.name
- **Accessing property**
  - console.log(user.name);
  - console.log(user[“name”]

Reading a non-existing property just returns undefined.

In operator

- **in** operator is used to check whether property exists or not
- Syntax :
  - "key" in object
- Example
  - let user = { name: "John", age: 30 };
  - alert("age" in user); // true, user.age exists
  - alert("blabla" in user); // false, user.blabla doesn't exist

For..in loop

- **for..in** is used to walk over all keys of an object
- **Syntax**:

```
for (key in object) {
  // executes the body for each key among object properties
}
```

- Example:

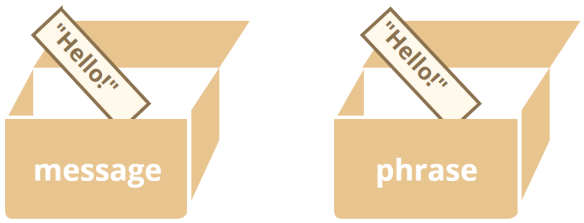
```
let user = {
  name: "John",
  age: 30,
  isAdmin: true
};
```

```
for (let key in user) {
        // keys
        alert( key );  // name, age, isAdmin
        // values for the keys
        alert( user[key] ); // John, 30, true
}
```

Object referencing

- For primitive data type data's are copied from one variable to another variable
- For instance:

```
let message = "Hello!";
let phrase = message;
```



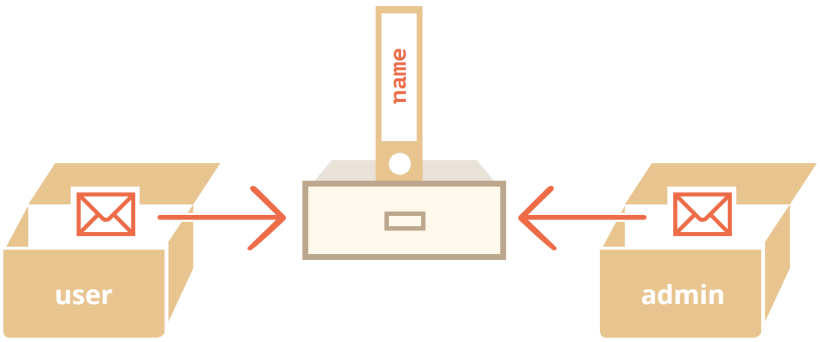
- As a result we have two independent variables, each one is storing the string "Hello!".

Copying object

- When an object variable is copied – the reference is copied, the object is not duplicated.

```
let user = { name: "John" };
let admin = user; // copy the reference
```

- Now we have two variables, each one with the reference to the same object.



Comparing by reference

- The equality == and strict equality === operators can be used to compare to objects.
- Two objects are equal only if both references are pointing to the same object.
- Here two variables reference the same object, thus they are equal:

```
let a = {};
let b = a; // copy the reference
alert( a == b ); // true, both variables reference the same object
alert( a === b ); // true
```

- And here two independent objects are not equal, even though both are empty:

```
let a = {};
let b = {}; // two independent objects
alert( a == b ); // false
```


Cloning object

- Following is the code to clone an object

```
let user = {
  name: "John",
  age: 30
};
let clone = {}; // the new empty object
// let's copy all user properties into it
for (let key in user) {
  clone[key] = user[key];
}
// now clone is a fully independent object with the same content
clone.name = "Pete"; // changed the data in it
alert( user.name ); // still John in the original object
```

Object methods

- Objects can have behavior i.e methods.

```
let user = {
  name: "John",
  age: 30
};

user.sayHi = function() {
  alert("Hello!");
};

user.sayHi(); // Hello!
```


Method shorthand

- There exists a shorter syntax for methods in an object literal:

```
// these objects do the same
user = {
  sayHi: function() {
    alert("Hello");
  }
};

// method shorthand looks better, right?
user = {
  sayHi() { // same as "sayHi: function()"
    alert("Hello");
  }
};
```

“this” in methods

- Object methods can access the information stored in the object using this reference.

```
let user = {
  name: "John",
  age: 30,

  sayHi() {
    // "this" is the "current object"
    alert(this.name);
  }
};
user.sayHi(); // John
```


“this” in methods

- Technically, it’s also possible to access the object without this, by referencing it via the outer variable:

```
let user = {
        name: "John",
        age: 30,

        sayHi() {
          alert(user.name); // "user" instead of "this"
        }
};
```