HINDUSTAN
INSTITUTE OF TECHNOLOGY & SCIENCE
(DEEMED TO BE UNIVERSITY)

CSB4301 - WEB TECHNOLOGY

B.Tech – V Semester

UNIT II

Dr. Muthukumaran M
Associate Professor
School of Computing Sciences,
Department of Computer Science and Engineering

Functions

- A JavaScript function is a block of code designed to perform a particular task.
- A JavaScript function is executed when "something" invokes it (calls it).
- A JavaScript function is defined with the function keyword, followed by a name, followed by parentheses ().
- Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).
- The parentheses may include parameter names separated by commas:
- (parameter1, parameter2, ...)
- The code to be executed, by the function, is placed inside curly brackets: {}

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

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.

Why Functions?

You can reuse code: Define the code once, and use it many times.

You can use the same code many times with different arguments, to produce different results.

Function Invocation

The code inside the function will execute when "something" invokes (calls) the function:

- When an event occurs (when a user clicks a button)
- When it is invoked (called) from JavaScript code
- Automatically (self invoked)

Function Return

When JavaScript reaches a return statement, the function will stop executing.

If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement.

Functions often compute a return value. The return value is "returned" back to the "caller":

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

<h2>JavaScript Functions</h2>

<p>This example calls a function which performs a calculation
and returns the result:</p>

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

<script>
var x = myFunction(4, 3);
document.getElementById("demo").innerHTML = x;

function myFunction(a, b) {
  return a * b;
}
</script>

</body>
</html>
```

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

<h2>JavaScript Functions</h2>

<p>This example calls a function to convert from Fahrenheit to Celsius:</p>
<p id="demo"></p>

<script>
function toCelsius(f) {
  return (5/9) * (f-32);
}
document.getElementById("demo").innerHTML =
toCelsius(77);
</script>

</body>
</html>
```

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

<h2>JavaScript Functions</h2>

<p>Accessing a function without () will return the function definition instead of the function result:</p>
<p id="demo"></p>

<script>
function toCelsius(f) {
  return (5/9) * (f-32);
}
document.getElementById("demo").innerHTML =
toCelsius;
</script>

</body>
</html>
```

The () Operator Invokes the Function

Using the example above, toCelsius refers to the function object, and toCelsius() refers to the function result.

Accessing a function without () will return the function object instead of the function result.

Functions Used as Variable Values

Functions can be used the same way as you use variables, in all types of formulas, assignments, and calculations.

Instead of using a variable to store the return value of a function:

```
let x = toCelsius(77);
let text = "The temperature is " + x + " Celsius";
```

You can use the function directly, as a variable value:

```
let text = "The temperature is " + toCelsius(77) + " Celsius";
```

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

<h2>JavaScript Functions</h2>

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

<script>
document.getElementById("demo").innerHTML =
"The temperature is " + toCelsius(77) + " Celsius";

function toCelsius(fahrenheit) {
  return (5/9) * (fahrenheit-32);
}
</script>

</body>
</html>
```

Local Variables

Variables declared within a JavaScript function, become LOCAL to the function.

Local variables can only be accessed from within the function.

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Functions</h2>
<p>Outside myFunction() carName is undefined.</p>
<p id="demo1"></p>
<p id="demo2"></p>
<script>
myFunction();
function myFunction() {
  let carName = "Volvo";
  document.getElementById("demo1").innerHTML =
  typeof carName + " " + carName;
}
document.getElementById("demo2").innerHTML =
typeof carName;
</script>

</body>
</html>
```

Since local variables are only recognized inside their functions, variables with the same name can be used in different functions.

Local variables are created when a function starts, and deleted when the function is completed.

Arrays

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

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

<h2>JavaScript Arrays</h2>

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

<script>
const cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars;
</script>

</body>
</html>
```

What is an Array?

An array is a special variable, which can hold more than one value at a time.

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
et car1 = "Saab";
let car2 = "Volvo";
let car3 = "BMW";
```

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The solution is an array!

An array can hold many values under a single name, and you can access the values by referring to an index number.

Creating an Array

Using an array literal is the easiest way to create a JavaScript Array.

Syntax:

```
const array_name = [item1, item2, ...];
```

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

<h2>JavaScript Arrays</h2>

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

<script>
const cars = [
  "Saab",
  "Volvo",
  "BMW"
];
document.getElementById("demo").innerHTML = cars;
</script>

</body>
</html>
```

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

<h2>JavaScript Arrays</h2>

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

<script>
const cars = [];
cars[0]= "Saab";
cars[1]= "Volvo";
cars[2]= "BMW";
document.getElementById("demo").innerHTML = cars;
</script>

</body>
</html>
```

Using the JavaScript Keyword new

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

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

<h2>JavaScript Arrays</h2>

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

<script>
const cars = new Array("Saab", "Volvo", "BMW");
document.getElementById("demo").innerHTML = cars;
</script>

</body>
</html>
```

The two examples above do exactly the same.

There is no need to use new Array().

Accessing Array Elements

You access an array element by referring to the index number:

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

<h2>JavaScript Arrays</h2>

<p>JavaScript array elements are accessed using numeric
indexes (starting from 0).</p>

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

<script>
const cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars[0];
</script>

</body>
</html>
```

Changing an Array Element

This statement changes the value of the first element in cars:

```
cars[0] = "Opel";
```

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

<h2>JavaScript Arrays</h2>

<p>JavaScript array elements are accessed using numeric
indexes (starting from 0).</p>

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

<script>
const cars = ["Saab", "Volvo", "BMW"];
cars[0] = "Opel";
document.getElementById("demo").innerHTML = cars;
</script>

</body>
</html>
```

Access the Full Array

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

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

<h2>JavaScript Arrays</h2>

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

<script>
const cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars;
</script>

</body>
</html>
```

Arrays are Objects

Arrays are a special type of objects. The typeof operator in JavaScript returns "object" for arrays.

But, JavaScript arrays are best described as arrays.

Arrays use numbers to access its "elements". In this example, person[0] returns John:

Arrays

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

<h2>JavaScript Arrays</h2>

<p>Arrays use numbers to access its elements.</p>

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

<script>
const person = ["John", "Doe", 46];
document.getElementById("demo").innerHTML =
person[0];
</script>

</body>
</html>
```

Objects use names to access its "members". In this example, *person.firstName* returns John:

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

<h2>JavaScript Objects</h2>
<p>JavaScript uses names to access object properties.</p>
<p id="demo"></p>

<script>
const person = {firstName:"John", lastName:"Doe", age:46};
document.getElementById("demo").innerHTML =
person.firstName;
</script>

</body>
</html>
```

Array Elements Can Be Objects

JavaScript variables can be objects. Arrays are special kinds of objects.

Because of this, you can have variables of different types in the same Array.

You can have objects in an Array. You can have functions in an Array. You can have arrays in an Array:

```
myArray[0] = Date.now;
myArray[1] = myFunction;
myArray[2] = myCars;
```

Array Properties

The real strength of JavaScript arrays are the built-in array properties and methods:

```
cars.length   // Returns the number of elements
cars.sort()   // Sorts the array
```

The length Property

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

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

<h2>JavaScript Arrays</h2>
<p>The length property returns the length of an array.</p>

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

<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.length;
</script>

</body>
</html>
```

The length property is always one more than the highest array index.

Accessing the First Array Element

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

<h2>JavaScript Arrays</h2>

<p>JavaScript array elements are accesses using numeric
indexes (starting from 0).</p>

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

<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits[0];
</script>

</body>
</html>
```

Accessing the Last Array Element

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

<h2>JavaScript Arrays</h2>

<p>JavaScript array elements are accesses using numeric
indexes (starting from 0).</p>

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

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML =
fruits[fruits.length-1];
</script>

</body>
</html>
```

The Difference Between Arrays and Objects

In JavaScript, arrays use numbered indexes.

In JavaScript, objects use named indexes.

When to Use Arrays. When to use Objects.

- JavaScript does not support associative arrays.
- You should use objects when you want the element names to be strings (text).
- You should use arrays when you want the element names to be numbers.

JavaScript Array Methods

Converting Arrays to Strings

The JavaScript method toString() converts an array to a string of (comma separated) array values.

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.toString();
```

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

<h2>JavaScript Array Methods</h2>
<h2>toString()</h2>
<p>The toString() method returns an array as a comma
separated string:</p>

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

<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML =
fruits.toString();
</script>

</body>
</html>
```

The join() method also joins all array elements into a string.

It behaves just like toString(), but in addition you can specify the separator:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.join(" * ");
```

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

<h2>JavaScript Array Methods</h2>
<h2>join()</h2>
<p>The join() method joins array elements into a string.</p>
<p>It this example we have used " * " as a separator between
the elements:</p>

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

<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.join(" *
");
</script>

</body>
</html>
```

Popping and Pushing

When you work with arrays, it is easy to remove elements and add new elements.

This is what popping and pushing is:

Popping items out of an array, or pushing items into an array.

Popping

The pop() method removes the last element from an array:

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

<h2>JavaScript Array Methods</h2>
<h2>pop()</h2>
<p>The pop() method removes the last element from an
array.</p>

<p id="demo1"></p>
<p id="demo2"></p>

<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo1").innerHTML = fruits;
fruits.pop();
document.getElementById("demo2").innerHTML = fruits;
</script>

</body>
</html>
```

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

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

<h2>JavaScript Array Methods</h2>
<h2>pop()</h2>
<p>The pop() method removes the last element from an array.</p>
<p>The return value of the pop() method is the removed item.</p>

<p id="demo1"></p>
<p id="demo2"></p>
<p id="demo3"></p>

<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo1").innerHTML = fruits;
document.getElementById("demo2").innerHTML = fruits.pop();
document.getElementById("demo3").innerHTML = fruits;
</script>

</body>
</html>
```

Pushing

The push() method adds a new element to an array (at the end):

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

<h2>JavaScript Array Methods</h2>
<h2>push()</h2>
<p>The push() method appends a new element to an array.</p>

<button onclick="myFunction()">Try it</button>
<p id="demo"></p>

<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits;

function myFunction() {
  fruits.push("Kiwi");
  document.getElementById("demo").innerHTML = fruits;
}
</script>

</body>
</html>
```

The push() method returns the new array length:

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

<h2>JavaScript Array Methods</h2>
<h2>push()</h2>
<p>The push() method returns the new array length.</p>

<button onclick="myFunction()">Try it</button>
<p id="demo1"></p>
<p id="demo2"></p>
<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo1").innerHTML = fruits;

function myFunction() {
  document.getElementById("demo1").innerHTML = fruits.push("Kiwi");
  document.getElementById("demo2").innerHTML = fruits;
}
</script>
</body>
</html>
```

Changing Elements

Array elements are accessed using their index number:

Array indexes start with 0:

[0] is the first array element
[1] is the second
[2] is the third ...

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

<h2>JavaScript Array Methods</h2>
<p>Array elements are accessed using their index number:</p>

<p id="demo1"></p>
<p id="demo2"></p>

<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo1").innerHTML = fruits;
fruits[0] = "Kiwi";
document.getElementById("demo2").innerHTML = fruits;
</script>

</body>
</html>
```

The length property provides an easy way to append a new element to an array:

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

<h2>JavaScript Array Methods</h2>
<p>The length property provides an easy way to append new elements to an array without using the push() method.</p>

<button onclick="myFunction()">Try it</button>
<p id="demo"></p>

<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits;

function myFunction() {
  fruits[fruits.length] = "Kiwi";
  document.getElementById("demo").innerHTML = fruits;
}
</script>

</body>
</html>
```

Deleting Elements

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

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

<h2>JavaScript Array Methods</h2>
<p>Deleting elements leaves undefined holes in an array.</p>
<p id="demo1"></p>
<p id="demo2"></p>
<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];

document.getElementById("demo1").innerHTML =
"The first fruit is: " + fruits[0];

delete fruits[0];

document.getElementById("demo2").innerHTML =
"The first fruit is: " + fruits[0];
</script>

</body>
</html>
```

Splicing an Array

The splice() method can be used to add new items to an array:

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

- The first parameter (2) defines the position where new elements should be added (spliced in).
- The second parameter (0) defines how many elements should be removed.
- The rest of the parameters ("Lemon" , "Kiwi") define the new elements to be added.
- The splice() method returns an array with the deleted items:

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Array Methods</h2>
<h2>splice()</h2>
<p>The splice() method adds new elements to an array.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo1"></p>
<p id="demo2"></p>
<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo1").innerHTML = "Original Array:<br>" + fruits;
function myFunction() {
  fruits.splice(2, 0, "Lemon", "Kiwi");
  document.getElementById("demo2").innerHTML = "New Array:<br>" + fruits;
}
</script>

</body>
</html>
```

Using splice() to Remove Elements

With clever parameter setting, you can use splice() to remove elements without leaving "holes" in the array:

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

- 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.
- The rest of the parameters are omitted. No new elements will be added.

Merging (Concatenating) Arrays

The concat() method creates a new array by merging (concatenating) existing arrays:

```
const myGirls = ["Cecilie", "Lone"];
const myBoys = ["Emil", "Tobias", "Linus"];

// Concatenate (join) myGirls and myBoys
const myChildren = myGirls.concat(myBoys);
```

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

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

<h2>JavaScript Array Methods</h2>
<h2>concat()</h2>
<p>The concat() method is used to merge (concatenate)
arrays:</p>

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

<script>
const myGirls = ["Cecilie", "Lone"];
const myBoys = ["Emil", "Tobias", "Linus"];
const myChildren = myGirls.concat(myBoys);

document.getElementById("demo").innerHTML = myChildren;
</script>

</body>
</html>
```

The concat() method can take any number of array arguments:

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

<h2>JavaScript Array Methods</h2>
<h2>concat()</h2>
<p>The concat() method is used to merge (concatenate) arrays:</p>

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

<script>
const array1 = ["Cecilie", "Lone"];
const array2 = ["Emil", "Tobias", "Linus"];
const array3 = ["Robin", "Morgan"];

const myChildren = array1.concat(array2, array3);

document.getElementById("demo").innerHTML = myChildren;
</script>

</body>
</html>
```