❖ JavaScript Arrays

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

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

- **Creating an Array**

Syntax:

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

- **Access the Elements of an Array**

✓ You refer to an array element by referring to the **index number**.

✓ This statement accesses the value of the first element in cars:

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

✓ This statement modifies the first element in cars:

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

Ex:

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

- **Access the Full Array**

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

Ex:

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

- **Array Properties and Methods**

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

✓ The length Property

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

Ex:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.length;                          // the length of fruits is 4
```

✓ Looping Array Elements

The best way to loop through an array, is using a "for" loop:

Ex:

```
<p id="demo"></p>
<script>
      var fruits, text, fLen, i;
      fruits = ["Banana", "Orange", "Apple", "Mango"];
      fLen = fruits.length;
      text = "<ul>";
      for (i = 0; i < fLen; i++) {
          text += "<li>" + fruits[i] + "</li>";
      }
      text += "<ul>";
</script>
```

✓ Adding Array Elements

The easiest way to add a new element to an array is using the push method:

Ex:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.push("Lemon");        // adds a new element (Lemon) to fruits
```

New element can also be added to an array using the length property:

Ex:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits[fruits.length] = "Lemon";
```

- **Associative Arrays**

  ✓ Many programming languages support arrays with named indexes.

  ✓ Arrays with named indexes are called associative arrays (or hashes).

  ✓ JavaScript does **not** support arrays with named indexes.

  ✓ In JavaScript, **arrays** always use **numbered indexes**.

  Ex:

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

❖ JavaScript Array Methods

- **Converting Arrays to Strings**

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

  Ex:

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

  Result: `Banana,Orange,Apple,Mango`

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

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

  Ex:

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

  Result: `Banana * Orange * Apple * Mango`

- **Popping and Pushing**

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

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

  Ex:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.pop();          // Removes the last element ("Mango") from fruit
```

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

  Ex:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
var x = fruits.pop();      // the value of x is "Mango"
```

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

  Ex:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.push("Kiwi");       //  Adds a new element ("Kiwi") to fruits
```

  The push() method returns the new array length:

  Ex:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
var x = fruits.push("Kiwi");   //  the value of x is 5
```

- **Changing Elements**

  Array elements are accessed using their **index number**:

  Ex:

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

- **Deleting Elements**

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

  Ex:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
delete fruits[0];

 // Changes the first element in fruits to undefined
```

- **Splicing an Array**

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

  Ex:

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

  Result: 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**.

- **Merging (Concatenating) Arrays**

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

  Ex:

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

  // Concatenates (joins) myGirls and myBoys
```

- **Sorting an Array**

  The **sort()** method sorts an array alphabetically:

  Ex:

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

  output**:** Apple,Banana,Mango,Orange

- **Reversing an Array**

  The **reverse()** method reverses the elements in an array.

  Ex:

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

  output: Orange,Mango,Banana,Apple

❖ JavaScript Popup Boxes

JavaScript has three kind of popup boxes: Alert box, Confirm box, and Prompt box.

- **Alert Box**

✓ An alert box is often used if you want to make sure information comes through to the user.

✓ When an alert box pops up, the user will have to click "OK" to proceed.

Syntax:

```
window.alert("sometext");
```

The **window.alert()** method can be written without the window prefix.

Ex:

```
alert("I am an alert box!");
```

✓ To display line breaks inside a popup box, use a back-slash followed by the character n.

Ex:

```
alert("Hello\nHow are you?");
```

- **Confirm Box**

✓ A confirm box is often used if you want the user to verify or accept something.

✓ When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed.

✓ If the user clicks "OK", the box returns true. If the user clicks "Cancel", the box returns false.

Syntax:

```
window.confirm("sometext");
```

The **window.confirm()** method can be written without the window prefix.

Ex:

```
if (confirm("Press a button!") == true) {
    txt = "You pressed OK!";
} else {
    txt = "You pressed Cancel!";
}
```

- **Prompt Box**

✓ A prompt box is often used if you want the user to input a value before entering a page.

✓ When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value.

✓ If the user clicks "OK" the box returns the input value. If the user clicks "Cancel" the box returns null.

Syntax:

```
window.prompt("sometext","defaultText");
```

The **window.prompt()** method can be written without the window prefix.

Ex:

```
var person = prompt("Please enter your name", "Harry Potter");
if (person == null || person == "") {
    txt = "User cancelled the prompt.";
} else {
    txt = "Hello " + person + "! How are you today?";
}
```

❖ Timing Events

- ✓ The window object allows execution of code at specified time intervals.
- ✓ These time intervals are called timing events.
- ✓ The two key methods to use with JavaScript are:
  - ➢ **setTimeout(*function, milliseconds*)**
    Executes a function, after waiting a specified number of milliseconds.
  - ➢ **setInterval(*function, milliseconds*)**
    Same as setTimeout(), but repeats the execution of the function continuously.

- **The setTimeout() Method**

```
window.setTimeout(function, milliseconds);
```

- ✓ The **window.setTimeout()** method can be written without the window prefix.
- ✓ The first parameter is a function to be executed.
- ✓ The second parameter indicates the number of milliseconds before execution.

Ex:

```
<button onclick="setTimeout(myFunction, 3000)">Try it</button>

<script>
function myFunction() {
    alert('Hello');
}
</script>
```

- **The setInterval() Method**

  ✓ The setInterval() method repeats a given function at every given time-interval.

```
window.setInterval(function, milliseconds);
```

  ✓ The **window.setInterval()** method can be written without the window prefix.

  ✓ The first parameter is the function to be executed.

  ✓ The second parameter indicates the length of the time-interval between each execution.

  Ex:

```
var myVar = setInterval(myTimer, 1000);
function myTimer() {
    var d = new Date();
    document.getElementById("demo").innerHTML =
d.toLocaleTimeString();
}
```