Q3} Explain With Example the methods of Array and String.

JavaScript Array Methods

Converting Arrays to Strings

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

Example

```
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:

Example

```
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.

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:

Example

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

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

Example

```
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):

Example

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

The push() method returns the new array length:

Example

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

Shifting Elements

Shifting is equivalent to popping, working on the first element instead of the last.

The shift() method removes the first array element and "shifts" all other elements to a lower index.

Example

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

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

Example

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

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

Example

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

The unshift() method returns the new array length.

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.unshift("Lemon");    // Returns 5
```

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….

Example

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

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

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits[fruits.length] = "Kiwi";          // Appends "Kiwi" to fruits
```

Deleting Elements

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

Example

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

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

Splicing an Array

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

Example

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

The first parameter (2) defines the position where new elements should be added (spliced in).

The second parameter (0) defines how many elements should be removed.

The rest of the parameters ("Lemon" , "Kiwi") define the new elements to be added.

The splice() method returns an array with the deleted items:

Example

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

Using splice() to Remove Elements

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

Example

```
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.

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:

Example (Merging Two Arrays)

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

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

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

Example (Merging Three Arrays)

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

The concat() method can also take strings as arguments:

Example (Merging an Array with Values)

```
var arr1 = ["Emil", "Tobias", "Linus"];
var myChildren = arr1.concat("Peter");
```

Slicing an Array

The slice() method slices out a piece of an array into a new array.

This example slices out a part of an array starting from array element 1 ("Orange"):

Example

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

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

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

Example

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

The slice() method can take two arguments like slice(1, 3).

The method then selects elements from the start argument, and up to (but not including) the end argument.

Example

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

If the end argument is omitted, like in the first examples, the slice() method slices out the rest of the array.

Example

```
var fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
var citrus = fruits.slice(2);
```

Automatic toString()

JavaScript automatically converts an array to a comma separated string when a primitive value is expected.

This is always the case when you try to output an array.

These two examples will produce the same result:

Example

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

Example

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

AllJAvaScript objects have a toString() method.

JavaScript String Methods

String methods help you to work with strings.

String Methods and Properties

Primitive values, like "John Doe", cannot have properties or methods (because they are not objects).

But with JavaScript, methods and properties are also available to primitive values, because JavaScript treats primitive values as objects when executing methods and properties.

String Length

The length property returns the length of a string:

Example

```
var txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
var sln = txt.length;
```

Finding a String in a String

The indexOf() method returns the index of (the position of) the first occurrence of a specified text in a string:

Example

```
var str = "Please locate where 'locate' occurs!";
var pos = str.indexOf("locate");
```

JavaScript counts positions from zero.
0 is the first position in a string, 1 is the second, 2 ia third..

The lastIndexOf() method returns the index of the last occurrence of a specified text in a string:

Example

```
var str = "Please locate where 'locate' occurs!";
var pos = str.lastIndexOf("locate");
```

Both indexOf(), and lastIndexOf() return -1 if the text is not found.

Example

```
var str = "Please locate where 'locate' occurs!";
var pos = str.lastIndexOf("John");
```

Both methods accept a second parameter as the starting position for the search:

Example

```
var str = "Please locate where 'locate' occurs!";
var pos = str.indexOf("locate", 15);
```

The lastIndexOf() methods searches backwards (from the end to the beginning), meaning: if the second parameter is 15, the search starts at position 15, and searches to the beginning of the string.

Example

```
var str = "Please locate where 'locate' occurs!";
var pos = str.lastIndexOf("locate", 15);
```

Searching for a String in a String

The search() method searches a string for a specified value and returns the position of the match:

Example

```
var str = "Please locate where 'locate' occurs!";
var pos = str.search("locate");
```

The two methods, indexOf() and search(), are equal?

They accept the same arguments (parameters), and return the same value?

The two methods are NOT equal. These are the differences:

- The search() method cannot take a second start position argument.
- The indexOf() method cannot take powerful search values (regular expressions).

You will learn more about regular expressions in a later chapter.

Extracting String Parts

There are 3 methods for extracting a part of a string:

- slice(start, end)
- substring(start, end)
- substr(start, length)

The slice() Method

slice() extracts a part of a string and returns the extracted part in a new string.

The method takes 2 parameters: the start position, and the end position (end not included).

This example slices out a portion of a string from position 7 to position 12 (13-1):

Example

```
var str = "Apple, Banana, Kiwi";
var res = str.slice(7, 13);
```

The result of res will be:

```
Banana
```

If a parameter is negative, the position is counted from the end of the string.

This example slices out a portion of a string from position -12 to position -6:

Example

```
var str = "Apple, Banana, Kiwi";
var res = str.slice(-12, -6);
```

The result of res will be:

```
Banana
```

If you omit the second parameter, the method will slice out the rest of the string:

Example

```
var res = str.slice(7);
```

or, counting from the end:

Example

```
var res = str.slice(-12);
```

The substring() Method

substring() is similar to slice().

The difference is that substring() cannot accept negative indexes.

Example

```
var str = "Apple, Banana, Kiwi";
var res = str.substring(7, 13);
```

The result of res will be:

Banana

If you omit the second parameter, substring() will slice out the rest of the string.

The substr() Method

substr() is similar to slice().

The difference is that the second parameter specifies the length of the extracted part.

Example

```
var str = "Apple, Banana, Kiwi";
var res = str.substr(7, 6);
```

The result of res will be:

Banana

If you omit the second parameter, substr() will slice out the rest of the string.

Example

```
var str = "Apple, Banana, Kiwi";
var res = str.substr(7);
```

The result of res will be:

Banana, Kiwi

If the first parameter is negative, the position counts from the end of the string.

Example

```
var str = "Apple, Banana, Kiwi";
var res = str.substr(-4);
```

The result of res will be:

```
Kiwi
```

Replacing String Content

The replace() method replaces a specified value with another value in a string:

Example

```
str = "Please visit Microsoft!";
var n = str.replace("Microsoft", "W3Schools");
```

The replace() method does not change the string it is called on. It returns a new string.

By default, the replace() method replaces only the first match:

Example

```
str = "Please visit Microsoft and Microsoft!";
var n = str.replace("Microsoft", "Schools");
```

By default, the replace() method is case sensitive. Writing MICROSOFT (with upper-case) will not work:

Example

```
str = "Please visit Microsoft!";
var n = str.replace("MICROSOFT", "Schools");
```

To replace case insensitive, use a regular expression with an /i flag (insensitive):

Example

```
str = "Please visit Microsoft!";
var n = str.replace(/MICROSOFT/i, "Schools");
```

To replace all matches, use a regular expression with a /g flag (global match):

Example

```
str = "Please visit Microsoft and Microsoft!";
var n = str.replace(/Microsoft/g, "Schools");
```

Converting to Upper and Lower Case

A string is converted to upper case with toUpperCase():

Example

```
var text1 = "Hello World!";       // String
var text2 = text1.toUpperCase();  // text2 is text1 converted to upper
```

A string is converted to lower case with toLowerCase():

Example

```
var text1 = "Hello World!";       // String
var text2 = text1.toLowerCase();  // text2 is text1 converted to lower
```

The concat() Method

concat() joins two or more strings:

Example

```
var text1 = "Hello";
var text2 = "World";
var text3 = text1.concat(" ", text2);
```

The concat() method can be used instead of the plus operator. These two lines do the same:

Example

```
var text = "Hello" + " " + "World!";
var text = "Hello".concat(" ", "World!");
```

All string methods return a new string. They don’t modify the original string. Strings are immutable. Strings cannot br changed , only replaced.

String.trim()

The trim() method removes whitespace from both sides of a string:

Example

```
var str = "       Hello World!        ";
alert(str.trim());
```

If you need to support IE 8, you can use replace() with a regular expression instead:

Example

```
var str = "       Hello World!        ";
alert(str.replace(/^[\s\uFEFF\xA0]+|[\s\uFEFF\xA0]+$/g, ''));
```

You can also use the replace solution above to add a trim function to the JavaScript String.prototype:

Example

```
if (!String.prototype.trim) {
  String.prototype.trim = function () {
    return this.replace(/^[\s\uFEFF\xA0]+|[\s\uFEFF\xA0]+$/g, '');
  };
}
var str = "       Hello World!        ";
alert(str.trim());
```

Extracting String Characters

There are 3 methods for extracting string characters:

- charAt(position)
- charCodeAt(position)
- Property access [ ]

The charAt() Method

The charAt() method returns the character at a specified index (position) in a string:

Example

```
var str = "HELLO WORLD";
str.charAt(0);            // returns H
```

The charCodeAt() Method

The charCodeAt() method returns the unicode of the character at a specified index in a string:

The method returns a UTF-16 code (an integer between 0 and 65535).

Example

```
var str = "HELLO WORLD";

str.charCodeAt(0);         // returns 72
```

Converting a String to an Array

A string can be converted to an array with the split() method:

Example

```
var txt = "a,b,c,d,e";   // String
txt.split(",");          // Split on commas
txt.split(" ");          // Split on spaces
txt.split("|");          // Split on pipe
```

If the separator is omitted, the returned array will contain the whole string in index [0].

If the separator is "", the returned array will be an array of single characters:

Example

```
var txt = "Hello";       // String
txt.split("");           // Split in characters
```