Arrays

Lesson Objectives

- Learn to use array in JavaScript programming.
- Learn useful array methods

Definition

- Array is a fundamental data structure that can hold multiple elements in a consecutive memory location.
- Memory location of an array is indexed, 0 being the first index for historical reason.

- In JavaScript, arrays are dynamic in both length and types of elements it can hold.

Declaring an Array

- Using array literal syntax

```
const numbers = [];
const fruits = ["Apple", "Banana", "Mango"];
```

- Array being an object type can also be created using new keyword

```
const numbers = new Array(6);
```

- Almost all the time, literal syntax is use.

Using an Array

- Array elements are numbered, starting with zero.
- We can get an element by its number in square brackets:

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

alert( fruits[0] ); // Apple
alert( fruits[1] ); // Orange
alert( fruits[2] ); // Plum
```

- We can replace an element:

```
fruits[2] = 'Pear'; // now ["Apple", "Orange", "Pear"]
```

- Or add a new one to the array:

```
fruits[3] = 'Lemon'; // now ["Apple", "Orange", "Pear", "Lemon"]
```

Size of an array

- In JavaScript, arrays have built-in property, `length`; which represents the current size of the array.
- The total count of the elements in the array is its `length`

```
let numbers = []
console.log(numbers.length); // 0
numbers = [1,2,3];
console.log(numbers.length) // 3
```

A word about “length”

- The `length` property automatically updates when we modify the array.
  - To be precise, it is not the count of values in the array, but the greatest numeric index plus one.
  - For instance, a single element with a large index gives a big length:

```
let fruits = [];
fruits[123] = "Apple";

alert( fruits.length ); // 124
```

- Note that we usually don’t use arrays like that.

Filling an Array

- Loops can be used to fill an array with some default values, usually for testing purposes.

```
const scores = [];
for (let i=0; i<10; i++){
       scores[i] = Math.ceil(Math.random()*100);
}

console.log(scores);
```

Exercise

- Write code to create an array named scores and fill it with 5 test scores 10, 20, 30, 40 and 50.
- Now write a function named findAverage, that takes an array as an argument and return average of the array values.
- Call findAverage function passing array you created in step1 and save the return result in a variable, average.
- Print the average, it should be 30 for this example.
- Create a second array filled with 10 random values between 0 to 10 and find the average of the array values.
- Make sure your program computes correct average for an array of any size.

Main Point

- Using an array, we can hold many elements under a single identifier, which eliminates the need for unique identifiers for every value. *Science of consciousness, during transcendence our bounded individual identity identifies with the unbounded cosmic identity*.

Looping through an array elements

- One of the oldest ways to cycle array items is the for loop over indexes:

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

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

- But for arrays there is another form of loop, `for..of`:

```
for (let fruit of fruits) {
  alert( fruit );
}
```

- The for..of doesn't give access to the index of the current element, just its value, but in most cases that's enough.
  - And it's shorter.
  - And avoids bugs that often occur from index errors at the end points
  - Favor for..of as default loop over arrays unless really need index

Array comparison

- Arrays are type Object
- When == or === operators are used on JavaScript objects, their references are compared

- **If array comparison is needed compare them item-by-item in a loop**.
  - Mocha has a very convenient assert.deepStrictEqual

Array methods

- JavaScript provides several useful methods that one can use to manipulate contents of an array.
    - Methods to add/remove array contents to/from beginning and end of an array.
    - Methods that allows to run a function on every element of the array.
    - Methods to sort and search
    - Methods to split and join and so on ...

`toString`

- Arrays have their own implementation of `toString` method that returns a comma-separated list of elements.
- For instance:

```
let arr = [1, 2, 3];

console.log( arr ); // [1,2,3]
console.log( arr.toString() === '1,2,3' ); // true
```

Add/Remove elements To/From the beginning

- Array in JavaScript has inbuilt methods that allow you to add/remove elements to/from the beginning of the array.
  - `shift`: extracts the first element of the array and returns it:

```
let fruits = ["Apple", "Orange", "Pear"];
console.log( fruits.shift() ); // remove Apple and alert it
console.log( fruits ); // Orange, Pear
```

  - `unshift`: add the element to the beginning of the array

```
let fruits = ["Orange", "Pear"];
fruits.unshift('Apple');
console.log( fruits ); // Apple, Orange, Pear
```

Add/Remove elements To/From the end

- Array in JavaScript has inbuilt methods that allow you to add/remove elements to/from the end of the array.
  - pop: extracts the last element of the array and return it.

```
let fruits = ["Apple", "Orange", "Pear"];
console.log( fruits.pop() ); // remove "Pear" and log it
console.log( fruits ); // Apple, Orange
```

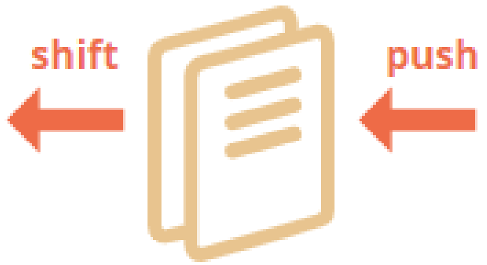
  - push: append element to the end of the array.

```
let fruits = ["Apple", "Orange"];
fruits.push("Pear");
console.log( fruits ); // Apple, Orange, Pear
```

  - The call `fruits.push(...)` is equal to `fruits[fruits.length] = ...`

Array as a queue

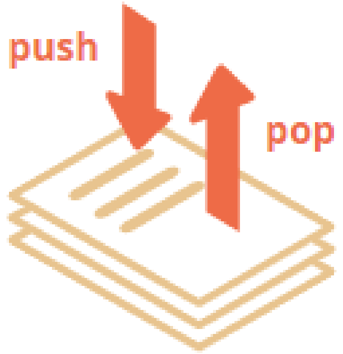
- A queue is one of the most common uses of an array.
  - In computer science, this means an ordered collection of elements which supports two operations:
    - `push` (enqueue) appends an element to the end
    - `shift` (dequeue) get an element from the beginning, advancing the queue, so that 2nd element becomes the 1st.



  - For queues, we have FIFO (First-In-First-Out) principle.
  - In practice we need it very often.
    - For example, a queue of messages that need to be shown on-screen.

Array as a stack

- There’s another use case for arrays – the data structure named stack.
- It supports two operations:
  - push adds an element to the end.
  - pop takes an element from the end.
- So new elements are added or taken always from the “end”.



- For stacks, the latest pushed item is received first, that’s also called LIFO (Last-In-First-Out) principle

Exercises

Given an expression array exp, write a program to examine whether the pairs and the orders of “{“, “}”, “(“, “)”, “[“, “]” are correct in exp.

Example:

Input: exp = ["(", ")", "[", "{", "}", "]"]

Output: Balanced

Input: exp = ["[", "(", "]", ")"]

Output: Not Balanced

Array methods

- Arrays provides a lot of methods.
  - We already know methods that add and remove from the beginning and the end.
  - There are few other methods to add/remove items.
- There are also methods for
  - Iterating through an array elements
    - `forEach`
  - Searching in array
    - `indexOf`, `lastIndexOf`, `includes`, `find`, `findIndex`, `filter`
  - Transforming an array
    - `map`, `sort(fn)`, `reverse`, `join`, `reduce`

splice

- The syntax is:

```
arr.splice(start[, deleteCount, elem1, ..., elemN])
```

- The `arr.splice` method is a swiss army knife for arrays.
  - It can remove/ replace array elements.
  - where to start
  - how many to delete
  - elements to insert
- insert elements without any removals.
  - set `deleteCount` to 0:
- Negative start means position from end of array
- See examples: *lecture_codes/arrays/splice.**

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

- It returns a new array copying 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.

- We can also call it without arguments: `arr.slice()` creates a copy of arr.
  - That’s often used to obtain a copy for further transformations that should not affect the original array.

- See example: *lecture_codes/arrays/slice_demo*

`arr.concat(arg1, arg2...)`

- Creates a new array that includes values from other arrays and additional items

```
let arr = [1, 2];

// create an array from: arr and [3,4]
console.log( arr.concat([3, 4]) ); // 1,2,3,4

// create an array from: arr and [3,4] and [5,6]
console.log( arr.concat([3, 4], [5, 6]) ); // 1,2,3,4,5,6

// create an array from: arr and [3,4], then add values 5 and 6
console.log( arr.concat([3, 4], 5, 6) ); // 1,2,3,4,5,6
```

`indexOf` / `lastIndexOf` and `includes`

- Syntax
  - `arr.indexOf(item)` – looks for `item` and returns the index where it was found, otherwise `-1`.
  - `arr.lastIndexOf(item)`– same, but looks for from right to left.
  - `arr.includes(item)`– looks for `item` starting from index from, returns `true` if found.
- Note that the methods use `===` comparison.
  - So, if we look for `false`, it finds exactly `false` and not the `zero`.
- See examples: *lecture_slides/arrays/indexOf.js*

reverse()

- The method arr.reverse() reverses the order of elements in arr.

```
let arr = [1, 3, 4, 5, 2];
arr.reverse();

console.log( arr ); // 2, 5, 4, 3, 1
```

`join([separator])`

- The `join()` method creates and returns a new string by concatenating all of the elements in an array separated by commas or a specified separator string.

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

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

console.log(str); // Bilbo,Gandalf,Nazgul

str = arr.join("-");

console.log(str); //Bilbo-Gandalf-Nazgul
```

Exercise

- Write a function, invert, that will reverse an array and output the reversed elements as a string with an optional separator.

const myArray = [“Sam”, “am”, “I”];

invert(myArray, “<<>>”) → “I<<>>am<<>>Sam”

Or

invert(myArray) → “I am Sam”

Callback functions (revisited)

- A callback is a function passed as an argument to another function.

```
function myDisplayer(result) {
    console.log(`Result of the calculation is ${result}`);
}

function myCalculator(num1, num2, myCallback) {
    let sum = num1 + num2;
    myCallback(sum);
}

myCalculator(5, 5, myDisplayer);
```

`forEach`

- The `arr.forEach` method allows to run a function for every element of the array.
- General syntax:

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

- Example

```
function display(item) {
    console.log(item);
}
// for each element call display
["Bilbo", "Gandalf", "Nazgul"].forEach(display);

// ["Bilbo", "Gandalf", "Nazgul"].forEach(item=>console.log(item));
```

sort([compareFunction])

- The call to `arr.sort()` sorts the array in place, changing its element order.
  - It also returns the sorted array, but the returned value is usually ignored, as the original itself is modified.
- The items are sorted lexicographically by default
  - To use your own sorting order, we need to supply a comparator function as the argument to the `sort()` method.

```
function comparator(a, b) {
    if (a > b) return 1; // if the first value is greater than the second
    if (a === b) return 0; // if values are equal
    if (a < b) return -1; // if the first value is less than the second
  }
```

- See examples: *lecture_notes/arrays/sort*

Exercise

- Refactor number sorting example to use anonymous function and then to use arrow function.

Main Point

- JavaScript arrays provide helper methods that make array programming easier. Use these methods to accomplish more by doing less. *Science of consciousness, when we are in harmony with the natural laws, our actions require less effort and hence we can achieve more by doing less*.

Multidimensional arrays

- Arrays can have items that are also arrays.
  - We can use it for multidimensional arrays, for example to store matrices:

```
let matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
  ];

  alert( matrix[1][1] ); // 5, the central element
```

Accessing elements

```
let matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]];

console.log(matrix);

for (let i = 0; i < matrix.length; i++) {
    for (let j = 0; j < matrix[i].length; j++) {
        console.log(matrix[i][j]);
    }
}
```

Exercise

- Write a function that accepts a two-dimensional array of numbers and return sum of all the elements in the array.

Assignment

- Reading
  - References
- assignments.docx

  Chapter 10, programming assignments (2 to 7, 10)

References

- Arrays (javascript.info)
- Array methods (javascript.info)