**Arrays**

There exists a special data structure named Array, to store ordered collections.

**[Declaration](https://javascript.info/array" \l "declaration)**

There are two syntaxes for creating an empty array:

let arr = new Array();

let arr = [];

Almost all the time, the second syntax is used. We can supply initial elements in the brackets:

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

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"]

The total count of the elements in the array is its length:

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

alert( fruits.length ); // 3

We can also use alert to show the whole array.

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

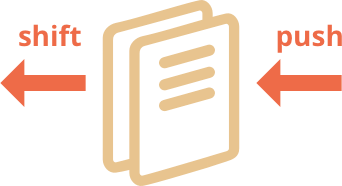
alert( fruits ); // Apple,Orange,Plum

An array can store elements of any type.

**[Methods pop/push, shift/unshift](https://javascript.info/array" \l "methods-pop-push-shift-unshift)**

A [queue](https://en.wikipedia.org/wiki/Queue_(abstract_data_type)) is one of most common uses of an array. In computer science, this means an ordered collection of elements which supports two operations:

* **push** appends an element to the end.
* **shift** get an element from the beginning, advancing the queue, so that the 2nd element becomes the 1st.



Arrays support both operations.

In practice we meet it very often. For example, a queue of messages that need to be shown on-screen.

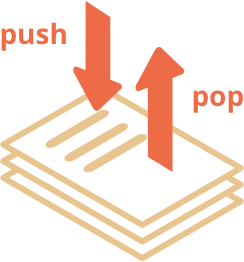
There’s another use case for arrays – the data structure named [stack](https://en.wikipedia.org/wiki/Stack_(abstract_data_type)).

It supports two operations:

* **push** adds an element to the end.
* **pop** takes an element at the end.

So new elements are added or taken always from the “end”.

A stack is usually illustrated as a pack of cards: new cards are added to the top or taken from the top:



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

Arrays in JavaScript can work both as a queue and as a stack. They allow to add/remove elements both to/from the beginning or the end.

**Methods that work with the end of the array:**

**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

**push**

Append the element to the end of the array:

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

fruits.push("Pear");

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

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

**Methods that work with the beginning of the array:**

**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

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 );

**[Loops](https://javascript.info/array" \l "loops)**

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] );

}

**[A word about “length”](https://javascript.info/array" \l "a-word-about-length)**

The length property automatically updates when we modify the array. To be precise, it is actually 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.

Another interesting thing about the length property is that it’s writable.

If we increase it manually, nothing interesting happens. But if we decrease it, the array is truncated. The process is irreversible, here’s the example:

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

So, the simplest way to clear the array is: arr.length=0.

**[toString](https://javascript.info/array" \l "tostring)**

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

For instance:

let arr = [1, 2, 3];

alert( arr ); // 1,2,3

alert( String(arr) === '1,2,3' ); // true

**[Summary](https://javascript.info/array" \l "summary)**

Array is a special kind of objects, suited to store and manage ordered data items.

* The declaration:
* // square brackets (usual)
* let arr = [item1, item2...];
* The length property is the array length or, to be precise, its last numeric index plus one. It is auto-adjusted by array methods.
* If we shorten length manually, the array is truncated.

We can use an array as a deque with the following operations:

* push(...items) adds items to the end.
* pop() removes the element from the end and returns it.
* shift() removes the element from the beginning and returns it.
* unshift(...items) adds items to the beginning.

To loop over the elements of the array:

* for(let i=0; i<arr.length; i++) – works fastest, old-browser-compatible.

**Questions and Exercise**

* 1. What is this code going to show?

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

// push a new value into the "copy"

fruits.push("Banana");

// what's in fruits?

alert( fruits.length ); // ?

4

1. 2. Create an array styles with items “Jazz” and “Blues”.

Let styles=[“Jazz”,”Blues”]

1. Append “Rock-n-Roll” to the end.

Styles.push(“RockNRoll”);

1. Replace the value in the middle by “Classics”. Your code for finding the middle value should work for any arrays with odd length.

Styles(Math.round(styles.length-1)/2)=“Classics”

1. Strip off the first value of the array and show it.

Alert(styles.shift());

1. Prepend Rap and Reggie to the array.

The array in the process:

Jazz, Blues

Jazz, Bues, Rock-n-Roll

Jazz, Classics, Rock-n-Roll

Classics, Rock-n-Roll

Rap, Reggie, Classics, Rock-n-Roll

* 1. Write the function sumInput() that:
* Asks the user for values using prompt and stores the values in the array.
* Finishes asking when the user enters a non-numeric value, an empty string, or presses “Cancel”.
* Calculates and returns the sum of array items.

P.S. A zero 0 is a valid number, please don’t stop the input on zero.

function sumInput()

{

Let data=0;

Let answer=0;

Do

{

Answer=+prompt()

Data.push(answer);

}while(number(answer)=NaN)

Data.pop()

Let sum=0;

For()

4. The input is an array of numbers, e.g. arr = [1, -2, 3, 4, -9, 6].

The task is: find the contiguous subarray of arr with the maximal sum of items.

Write the function getMaxSubSum(arr) that will find return that sum.

For instance:

getMaxSubSum([-1, 2, 3, -9]) = 5 (the sum of highlighted items)

getMaxSubSum([2, -1, 2, 3, -9]) = 6

getMaxSubSum([-1, 2, 3, -9, 11]) = 11

getMaxSubSum([-2, -1, 1, 2]) = 3

getMaxSubSum([100, -9, 2, -3, 5]) = 100

getMaxSubSum([1, 2, 3]) = 6 (take all)

If all items are negative, it means that we take none (the subarray is empty), so the sum is zero:

getMaxSubSum([-1, -2, -3]) = 0

**Additional Exercises:**

* 1. Create a program that fills a 10 element array with 10 random numbers between 1 and 100. Display these numbers. Add two buttons to your program. When the first is clicked find the largest number in your array. When the second is clicked find the smallest.
  2. Create a program that displays the numbers 1 to 100 and their squares.
  3. Add a third button to your program from exercise #1. When clicked display the average of all the numbers in the array.
  4. Add a fourth button and a textbox to the program from exercise #1. When clicked display the position of the number entered into the textbox in the array. Display -1 if the number is not in the list.

**Arrays and Functions**

1. Create and test a function that takes an array of integers and returns their average.
2. Create a function that takes an array of random numbers and returns the largest number.
3. Create another function that takes an array of random numbers and returns the smallest number.
4. Create a third function that takes an array of random numbers and a boolean value. If the boolean value passed as an arguement is false find and return the largest number, otherwise return the smallest.
5. Create a function that takes an array of words of words and returns a random one from the list.
6. Create a function that takes a word and and letter as arguments. It then returns the number of times that letter appears in the word.
7. Create a function that takes a String representing a sentence and returns the number of words in it.
8. Create a function that takes two arrays of Strings and returns **true** if any word in the first array is found in the second array, otherwise it returns **false**.