## Функции

1. Напишите функцию, которая в строке, состоящей из чисел, разделенных пробелом, находит максимальное и минимальное и возвращает их

|  |
| --- |
| var result = highAndLow("4 5 29 54 4 0 -214 542 -64 1 -3 6 -6");  console.log( result ); // "542 -214" |

1. Напишите функцию *checkNumber*, которая получает на вход как параметр массив любых значений и возвращает *true*, если все элементы - числа, и *false* - если в массиве хотя бы одно не число. Для проверки массива используйте метод [every](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Array/every) или [some](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Array/some)

|  |
| --- |
| console.log( checkNumber([1, 2, '3', 'a']) ); // false  console.log( checkNumber([1, 2, '3', '4']) ); // true |

1. Напишите функцию, которая *возвращает* куб переданного числа, аналог Math.pow(x, 3) – a) используя цикл b) используя рекурсию:

|  |
| --- |
| console.log( cube(2) ); // 8 |

1. Напишите функцию **extraCube**, которая принимает в качестве параметра или число, или массив числовых значений и *возвращает* либо куб числа, либо массив кубов, в зависимости от типа входящего параметра (**typeof**). Для расчета куба числа вместо встроенного метода Math.pow используйте собственную функцию.

|  |
| --- |
| console.log( extraCube(2) ); // 8  console.log( extraCube([0, 1, 2, 3]) ); // [0, 1, 8, 27] |

1. Придумайте алгоритм расчета суммы всех фактических параметров функции с использованием только рекурсии:

|  |
| --- |
| console.log( sum(1, 2, 3, 4, 5) ); // 15 |

1. Дан массив, элементами которого могут быть любые значения, включая другие массивы. Напишите функцию flat, которая рекурсивно “разворачивает” переданный массив на глубину depth, указанную в параметре функции. По умолчанию flat поднимает элементы на одну глубину (то есть если depth не указан).

|  |
| --- |
| var data1 = [1, 2, [3, 4, [5, 6]]];  flat(data1); // [1, 2, 3, 4, [5, 6]]  var data2 = [1, 2, [3, 4, [5, 6, [7, 8, [9, 10]]]]];  console.log( flat(data2, 1) ); // [1, 2, 3, 4, [5, 6, [7, 8, [9, 10]]]]  console.log( flat(data2, 2) ); // [1, 2, 3, 4, 5, 6, [7, 8, [9, 10]]]  console.log( flat(data2, 3) ); // [1, 2, 3, 4, 5, 6, 7, 8, [9, 10]]  console.log( flat(data2, 4) ); // [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  console.log( flat(data2, 0) ); // [1, 2, [3, 4, [5, 6, [7, 8, [9, 10]]]]]  console.log( flat(data2, 5) ); // [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] |

## Замыкания

1. Используя замыкание, напишите функцию createTimer, которая использует метод **performance.now()** для получения текущей временн*о*й метки и служит для замера времени выполнения другого кода:

|  |
| --- |
| var timer = createTimer();  alert('!'); // код, время выполнения которого нужно измерить  alert( timer() ); // время в мкс от начала выполнения createTimer() до момента вызова timer() |

1. Используя замыкания, создайте функцию createAdder(), которая принимает на вход любой примитивный параметр и возвращает функцию, которая добавляет к первому параметру второй. Функция работает по следующему принципу:

|  |
| --- |
| var hello = createAdder('Hello, ');  alert( hello('John') ); // Hello, John  alert( hello('Harry') ); // Hello, Harry  var plus = createAdder(5);  alert( plus(1) ); // 6  alert( plus(5) ); // 10 |

## [JS Quiz](https://docs.google.com/forms/d/e/1FAIpQLSdI6jod1To6R7hc7oITofOCJ6KnrRP1_nBd4MyuYAk2Tt0WeA/viewform)