**Задачи по JavaScript:**

**Напишите функцию, которая будет проходить через массив целых чисел и выводить индекс каждого элемента с задержкой в 3 секунды.**

Самое популярное и неправильное решение разработчиков JavaScript выглядит следующим образом:

const arr = [10, 12, 15, 21];

for (var i = 0; i < arr.length; i++) {

setTimeout(function() {

console.log('The index of this number is: ' + i);

}, 3000);

}

В данном случае код будет выводить “*Index: 4, element: undefined*” (повторится 4 раза). Это происходит потому, что функция *setTimeout*является внутренней (вложенной) функцией, которая имеет доступ к переменным внешней функции. Во внешней функции есть цикл *for,* в котором инициализируется переменная *i* - индекс для прохода по массиву. Через 3 секунды выполняется код функции, выводится значение i, которое на конец цикла равно 4, потому что оно проходит через 0, 1, 2, 3, 4 и останавливается на 4.

Чтобы выполнить задание, необходимо быть знакомым с [замыканием функций](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Closures), [внутренним таймером](https://developer.mozilla.org/en-US/docs/Web/API/WindowOrWorkerGlobalScope/setTimeout)и [областью видимости](https://toddmotto.com/everything-you-wanted-to-know-about-javascript-scope/).

Правильная реализация выглядит примерно так:

const arr = [10, 12, 15, 21];

for (var i = 0; i < arr.length; i++) {

// pass in the variable i so that each function

// has access to the correct index

setTimeout(function(i\_local) {

return function() {

console.log('The index of this number is: ' + i\_local);

}

}(i), 3000);

}

Или так:

const arr = [10, 12, 15, 21];

for (let i = 0; i < arr.length; i++) {

// using the ES6 let syntax, it creates a new binding

// every single time the function is called

// read more here: http://exploringjs.com/es6/ch\_variables.html#sec\_let-const-loop-heads

setTimeout(function() {

console.log('The index of this number is: ' + i);

}, 3000);

}

**Реализуйте каррирование (англ. currying).**

Прежде всего, что такое каррирование в JavaScript? Это приём в функциональном программировании, позволяющий преобразовать функцию, заменив её несколько первых аргументов константными значениями, тем самым создав новую функцию с меньшим количеством аргументов на основе старой. Помогает писать более чистый и элегантный код.

**Пример использования:**

function addBase(base){

return function(num){

return base + num;

}

}

var addTen = addBase(10);

addTen(5); //15

addTen(80); //90

addTen(-5); //5

В данном примере, используя каррирование, вы прибавляете новое число к уже заданному.

**Реализация поставленной задачи:**

Вы можете применить каррирование к прототипу функции. Если нет аргументов для каррирования, вы просто возвращаете текущую функцию. Если есть аргументы, то переходите к следующим шагам:

* Объедините старые аргументы с новыми, используя args.concat(toArray(arguments))
* Передайте все аргументы в функцию, используя apply

Function.prototype.curry = function() {

if (arguments.length<1) {

return this; //nothing to curry. return function

}

var self = this;

var args = toArray(arguments);

return function() {

return self.apply(this, args.concat(toArray(arguments)));

}

}

function toArray(args) {

return Array.prototype.slice.call(args);

}

**Использование:**

Передайте аргументы в метод function.curry. Возвратится функция. Используйте её для дальнейшего каррирования.

function converter = function(factor, symbol, input){

return input \* factor + symbol;

}

var milesToKm = converter.curry(1.62, 'km');

mileToKm(3); //result here

var kgToLb = converter.curry(2.2, 'lb');

kgToLb(3); //result here

**Вопросы по JavaScript:**

**Как бы вы могли использовать Math.max для нахождения максимального элемента в массиве?**

Метод **Math.max()** возвращает наибольшее из нуля или более чисел. Используем  метод [Function.prototype.apply()](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Function/apply" \o "Метод apply() вызывает функцию с указанным значением this и аргументами, предоставленными в виде массива (либо массивоподобного объекта).) для нахождения максимального элемента в числовом массиве. Вызов getMaxOfArray([1, 2, 3]) эквивалентен вызову Math.max(1, 2, 3), однако вы можете использовать функцию getMaxOfArray() вместе с программно сконструированными массивами любого размера.

function getMax(arr){

return Math.max.apply(null, arr);

}

**Как бы вы сравнили два объекта в JavaScript?**

Используем [Lodash](https://lodash.com/docs/4.17.4" \l "isEqual" \t "_blank). Среди его функций есть isEqual(), которая насильно проверяет каждое значение ключа, используя [ECMAScript5](https://en.wikipedia.org/wiki/ECMAScript#Versions) и собственную оптимизацию, если они доступны в браузере.

\_.isEqual(object, other);

**В чем разница между собственными объектами и объектами хоста?**

**Объекты хоста**: объекты типа window, XmlHttpRequest, узлы DOM и т. д., предоставляющиеся средой браузера. Они отличаются от встроенных объектов, потому что не все окружения имеют одинаковые объекты хоста. Если JavaScript работает за пределами браузера, например, на языке сценариев на стороне сервера, например, в Node.js, будут доступны разные объекты хоста.

**Объекты пользователя**: объекты, определенные в JavaScript-коде. Таким образом, инициализированная переменная в вашем коде будет объектом пользователя.

**(function() {\*code\*})(); Для чего используется такая конструкция? Напишите несколько примеров создания.**

Это называется [Immediately-Invoked Function Expression](http://benalman.com/news/2010/11/immediately-invoked-function-expression/" \t "_blank) (IIFE) функцией, которая вызывается сразу после создания. Используется для того, чтобы некоторые переменные не попали во внешнее окружение. Также используется для создания локального замыкания.

**Примеры создания:**

(function(){ /\* code \*/ })(); // Обычная IIFE

// Поставить функцию в состояние "значение"

true && function(){ /\* code \*/ }();

0,function(){ /\* code \*/ }();

// Можно создать, используя оператор в качестве префикса

!function(){ /\* code \*/ }(); // Facebook style

~function(){ /\* code \*/ }();

-function(){ /\* code \*/ }();

+function(){ /\* code \*/ }();

// Используем new

new function(){ /\* code \*/ }

new function(){ /\* code \*/ }() // Если необходимы аргументы, то ставим скобки

**С помощью какой конструкции можно управлять потоком выполнения скрипта  и отслеживать ошибки?**

Это возможно с помощью try{}...catch{}

try{

var number = 9;

alert( number.toString(2) );

//...

alert('Without exception!');

} catch(e) {

alert('Exception!');

}

**Правда/ложь**

В этой части статьи будет задано 10 вопросов, на которые нужно однозначно ответить: ложно утверждение или нет.

**Вопрос**: 'false' ложно?  
**Ответ**: Нет, т.к. только пустая строка является ложной.

**Вопрос**: ' ' ложно?  
**Ответ**: Эта строка не является пустой, т.к. в ней есть пробел. Из этого следует, что ' ' не ложно.

**Вопрос**: Что насчёт {} ?  
**Ответ**: Истинно, т.к. это объект. Объект без свойств не может быть ложным.

**Вопрос**: Что можете сказать о [] ?  
**Ответ**: Это выражение также истинно, т.к. это объект массива.

**Вопрос**: Если вы утверждаете, что '' (пустая строка) является ложным утверждением, тогда что насчёт newString(' ')? // newString(пустая строка)  
**Ответ**: Несмотря на то, что вы передаёте пустую строку в string constructor, конструктор создаёт объект string. Точнее, экземпляр объекта string. Он становится объектом. Следовательно, это не ложь.

**Вопрос**:  Что можете сказать о newBoolean(false)?  
**Ответ**: Истинно, потому что конструктор создаёт экземпляр объекта boolean, который является объектом.

**Вопрос**: Boolean(function(){})  
**Ответ**: true, если вы передадите истинное значение в Boolean.

**Вопрос**: Boolean(/foo/)  
**Ответ**: true

**Вопрос**: true % 1  
**Ответ**: 0. Если вы хотите найти остаток от true, true становится единицей, а остаток от деления единицы на единицу равен 0.

**Вопрос**: ' ' % 1 //пустая строка % 1  
**Ответ**: 0