Лабораторная работа №4

Введение в Angular.js 1

1. Задачи на лабораторную работу
   * Напишите следующие функции:
     1. Функция умножения с использованием каррирования.

Немного теории: *Каррирование - 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. Этот будет удобно применять в случае, когда первые несколько аргументов функции заранее известны, и указывать их при каждом вызове нет необходимости.*

function curry(a) {

return function (b) {

// в этом вызове аргумент a заменён на переданное в функцию curry значение

return a\*b;

};

}

var inc = curry(2);

alert(inc(5)); // Результат: 10

alert(inc(10)); // Результат: 20

alert(inc(3)); // Результат: 6

**Результат выполнения:**

* + 1. Функция вычисляющая факториал. Не использовать рекурсию. Использовать подход мемоизации.

Немного теории: *Мемоизация — сохранение результатов выполнения функций для предотвращения повторных вычислений. Это один из способов*[*оптимизации*](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D1%82%D0%B8%D0%BC%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0))*, применяемый для увеличения скорости выполнения*[*компьютерных программ*](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0)*. Перед вызовом функции проверяется, вызывалась ли функция ранее: если не вызывалась, функция вызывается и результат её выполнения сохраняется; если вызывалась, используется сохранённый результат.*

Сама функция вычисляющая факториал:

*C рекурсией:*

function fibonacci(n) {

  if (n === 0 || n === 1)

    return n;

  else

    return fibonacci(n - 1) + fibonacci(n - 2);

}

*Без рекурсии:*

function fibonacci (n){

var mass = [];

mass.push(1,1);

for(var i =2 ; i < n ; i++){

mass[i]= mass[i-1]+mass[i-2];

}

console.log(mass[n-1]);

return function(x){

return mass[x];

}

}

С использованием меморизации и без рекурсии:

function memoized (fn, keymaker) {

var lookupTable = {}, key;

keymaker || (keymaker = function (args) {

return JSON.stringify(args)

});

return function () {

var key = keymaker.call(this, arguments);

return lookupTable[key] || (

lookupTable[key] = fn.apply(this, arguments)

)

}

}

var memoizedFibonacci = memoized( function (n) {

var mass = [];

mass.push(1,1);

for(var i =2 ; i < n ; i++){

mass[i]= mass[i-1]+mass[i-2];

}

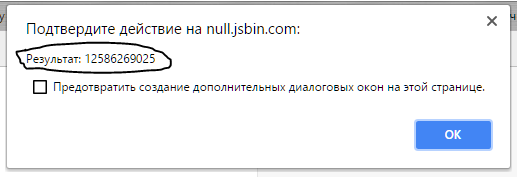
console.log(mass[n-1]);

return mass[n-1];

});

alert(“Результат: ” + memoizedFibonacci(50));

**Результат выполнения:**



1. Изучить документация по Angular.js
   * <https://www.youtube.com/watch?v=g41QNEqTE-E&list=PLIcAMDxr6tpqXzsd4AO0HehPCQtIf4TgP>
   * <http://campus.codeschool.com/courses/shaping-up-with-angular-js>
   * <https://thinkster.io/a-better-way-to-learn-angularjs>
2. Рассмотреть пример приложения todomvc.com с angular.js. Разобраться в компонентах приложения. Необходимо объяснить, что располагается в каком файле.
   * <http://todomvc.com/examples/angularjs/#/>
     1. <https://docs.angularjs.org/guide/concepts> <- обяъяснить зачем какая сущность необходима.
   * Нарисовать схему компонентов приложения <http://todomvc.com/examples/angularjs/#/>