|  |  |  |
| --- | --- | --- |
|  |  |  |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |

Институт Информационных Технологий

Кафедра ИППО

**ОТЧЕТ**

по лабораторной работе №3

по дисциплине

«Проектирование сред программирования и их компонентов»

Выполнил студент группы ИКМО-01-17 Алешин К.В.

Принял ассистент Гречин А.А.

Выполнено «\_\_\_» \_\_\_\_\_\_\_\_\_ 2018 г.

Зачтено «\_\_\_» \_\_\_\_\_\_\_\_\_ 2018 г.

Москва 2018

1. **Создайте калькулятор**

Создайте объект calculator с тремя методами:

* read() запрашивает prompt два значения и сохраняет их как свойства объекта
* sum() возвращает сумму этих двух значений
* mul() возвращает произведение этих двух значений

var calculator = {

...ваш код...

}

calculator.read();

alert( calculator.sum() );

alert( calculator.mul() );

### 

### 

### 

1. **Цепочка вызовов**

Есть объект «лестница» ladder:

var ladder = {

step: 0,

up: function() { // вверх по лестнице

this.step++;

},

down: function() { // вниз по лестнице

this.step--;

},

showStep: function() { // вывести текущую ступеньку

alert( this.step );

}

};

Сейчас, если нужно последовательно вызвать несколько методов объекта, это можно сделать так:

ladder.up();

ladder.up();

ladder.down();

ladder.showStep(); // 1

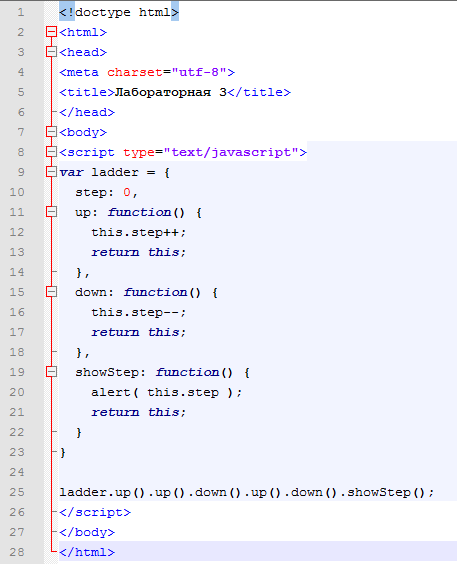
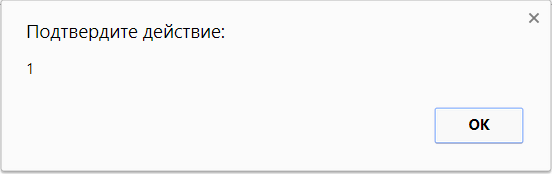
Модифицируйте код методов объекта, чтобы вызовы можно было делать цепочкой, вот так:

ladder.up().up().down().up().down().showStep(); // 1

Как видно, такая запись содержит «меньше букв» и может быть более наглядной.

Такой подход называется «чейнинг» (chaining) и используется, например, во фреймворке jQuery.

Решение состоит в том, чтобы каждый раз возвращать текущий объект. Это делается добавлением return this в конце каждого метода:

1. **Сумма произвольного количества скобок**

Напишите функцию sum, которая будет работать так:

sum(1)(2) == 3; // 1 + 2

sum(1)(2)(3) == 6; // 1 + 2 + 3

sum(5)(-1)(2) == 6

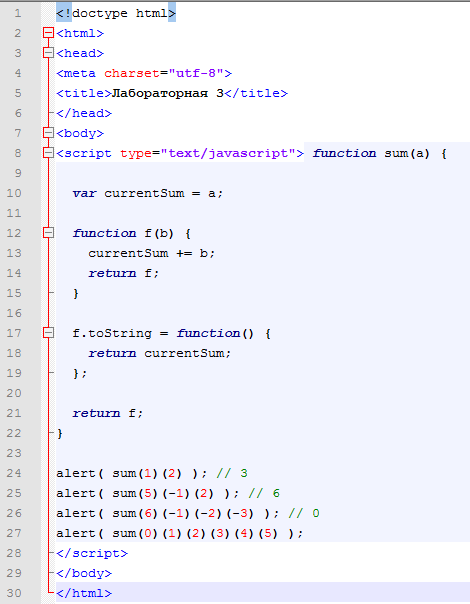
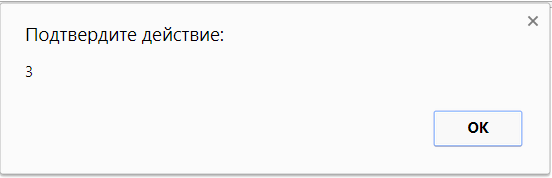
sum(6)(-1)(-2)(-3) == 0

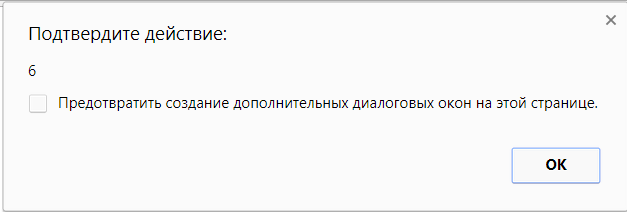
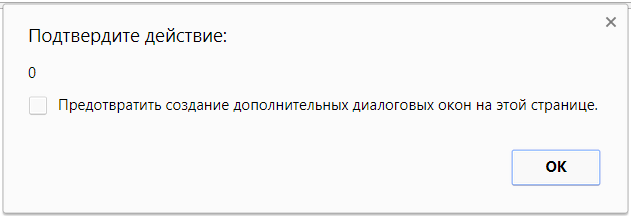
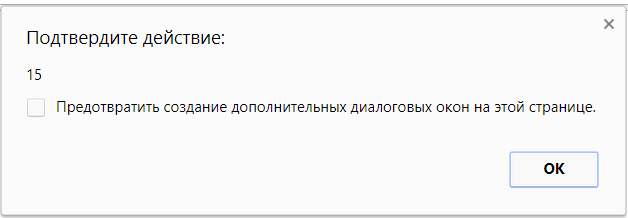
sum(0)(1)(2)(3)(4)(5) == 15

Количество скобок может быть любым.

Функция, которая возвращается sum, должна накапливать значение при каждом вызове.

Удобнее всего хранить его в замыкании, в переменной currentSum. Каждый вызов прибавляет к ней очередное значение:

Эта f используется при следующем вызове, опять возвратит себя, и так сколько нужно раз. Затем, при использовании в строчном или численном контексте – сработает toString, который вернет текущую сумму currentSum.

1. **Создать Calculator при помощи конструктора**

Напишите функцию-конструктор Calculator, которая создает объект с тремя методами:

Метод read() запрашивает два значения при помощи prompt и запоминает их в свойствах объекта.

Метод sum() возвращает сумму запомненных свойств.

Метод mul() возвращает произведение запомненных свойств.

Пример использования:

var calculator = new Calculator();

calculator.read();

alert( "Сумма=" + calculator.sum() );

alert( "Произведение=" + calculator.mul() );

### 

### 

1. **Создайте калькулятор**

Напишите конструктор Calculator, который создаёт расширяемые объекты-калькуляторы.

Эта задача состоит из двух частей, которые можно решать одна за другой.

Первый шаг задачи: вызов calculate(str) принимает строку, например «1 + 2», с жёстко заданным форматом «ЧИСЛО операция ЧИСЛО» (по одному пробелу вокруг операции), и возвращает результат. Понимает плюс + и минус -.

Пример использования:

var calc = new Calculator;

alert( calc.calculate("3 + 7") ); // 10

Второй шаг – добавить калькулятору метод addMethod(name, func), который учит калькулятор новой операции. Он получает имя операции name и функцию от двух аргументов func(a,b), которая должна её реализовывать.

Например, добавим операции умножить \*, поделить / и возвести в степень \*\*:

var powerCalc = new Calculator;

powerCalc.addMethod("\*", function(a, b) {

return a \* b;

});

powerCalc.addMethod("/", function(a, b) {

return a / b;

});

powerCalc.addMethod("\*\*", function(a, b) {

return Math.pow(a, b);

});

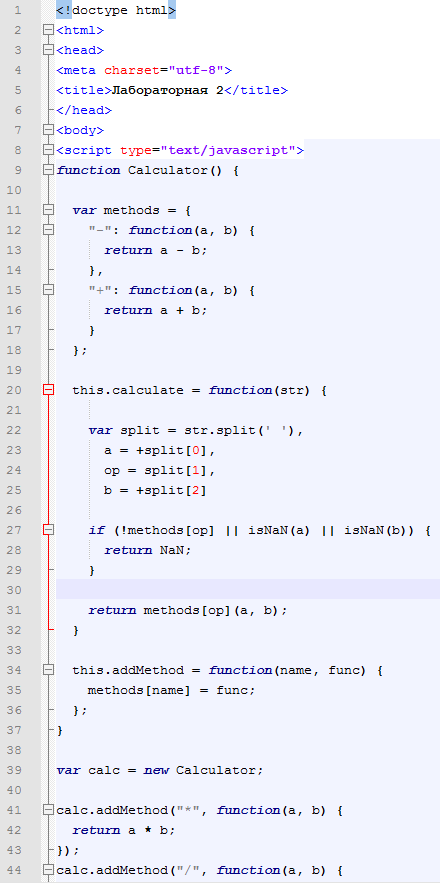
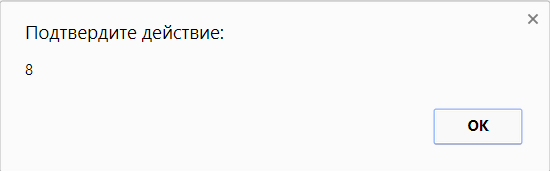
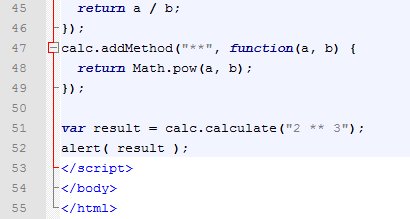
var result = powerCalc.calculate("2 \*\* 3");

alert( result ); // 8

Поддержка скобок и сложных математических выражений в этой задаче не требуется.

Числа и операции могут состоять из нескольких символов. Между ними ровно один пробел.

Предусмотрите обработку ошибок. Какая она должна быть – решите сами.

Все проверки и преобразование к числу производятся в методе calculate.

1. **Решето Эратосфена**

Целое число, большее 1, называется простым, если оно не делится нацело ни на какое другое, кроме себя и 1.

Древний алгоритм «Решето Эратосфена» для поиска всех простых чисел до n выглядит так:

Создать список последовательных чисел от 2 до n: 2, 3, 4, ..., n.

Пусть p=2, это первое простое число.

Зачеркнуть все последующие числа в списке с разницей в p, т.е. 2\*p, 3\*p, 4\*p и т.д. В случае p=2это будут 4,6,8....

Поменять значение p на первое не зачеркнутое число после p.

Повторить шаги 3-4 пока p2 < n.

Все оставшиеся не зачеркнутыми числа – простые.

Реализуйте «Решето Эратосфена» в JavaScript, используя массив.

Найдите все простые числа до 100 и выведите их сумму.

