## Методы объектов и контекст исполнения функции

1. Создайте объект calculator с методами:
   1. read() запрашивает prompt для двух значений и сохраняет их как свойства объекта x, y
   2. sum() возвращает сумму этих двух значений
   3. multi() возвращает произведение этих двух значений
   4. diff() возвращает разницу
   5. div() возвращает частное

|  |
| --- |
| calculator.read();  alert( calculator.sum() );  alert( calculator.multi() ); |

1. Создайте объект coffeeMachine со свойством message: ‘Your coffee is ready!’ и методом start(), при вызове которого – coffeeMachine.start() – через 3 секунды появляется окно с сообщением, записанным в свойстве объекта message.
2. Создайте функцию hello(), которая выводит приветствие пользователю. Создайте два объекта, содержащие поля firstname, lastname. Используя метод **call** и функцию hello() приветствуйте каждого из пользователей персонально.
3. Создайте объект counter с методами увеличения, уменьшения значения счетчика и методом возврата текущего значения. Используйте концепцию *chaining* для создания цепочки вызовов.

|  |
| --- |
| var current = counter.inc().inc().dec().inc().dec().getValue();  alert(current); // 1 |

1. Создайте объект с данными: x, y и методами: getSum, getDiff, getMulti, getDiv. Методы объекта ничего не реализуют, а только выводят в alert сообщения вида ‘1 + 1 = 2’ или ‘1 / 0 = Infinity’. Для расчетов все методы используют функционал ранее созданного калькулятора.

|  |
| --- |
| alert(me.getSum(1, 1));  alert(me.getDiv(1, 0)); |

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

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

1. Создайте объект со свойством delay и методами appendTo и appendText. Метод appendTo с помощью jQuery добавляет абзац в контейнер, переданный в параметре метода. Метод appendText может дописывает текст в добавленный элемент. Создайте массив строк и запустите цикл по этому массиву. С периодичностью, определенной в свойстве delay, в текст добавленного html-элемента добавляется соответствующий по порядку элемент массива. Учтите, что для доступа к вашему элементу не должен производиться поиск по DOM-дереву. [Live preview](https://www.screencast.com/t/TBioBXcXKvpm)
2. Есть следующий код:

|  |
| --- |
| var country = {  name: 'Ukraine',  language: 'ukrainian',  capital: {  name: 'Kyiv',  population: 2907817,  area: 847.66  } };  function format(start, end) {  console.log(start + this.name + end); } |

Допишите код, чтобы в консоли браузера появились строки, которые написаны в комментариях:

|  |
| --- |
| format.call(/\* Ваш код \*/); // Ukraine format.apply(/\* Ваш код \*/); // [Ukraine] format.call(/\* Ваш код \*/); // Kyiv format.apply(/\* Ваш код \*/); // Kyiv  format.apply(/\* Ваш код \*/); // undefined |

1. Создайте объект user с полем name. Создайте функцию format с параметрами start и end:

|  |
| --- |
| function format(start, end) {  console.log(start + this.name + end); } |

Привяжите функцию format() к объекту user таким образом, чтобы ее вызов возвращал отформатированное имя пользователя

|  |
| --- |
| userFormat('<<<', '>>>'); // <<<John>>> |

Реализуйте 2 версии текущего задания, используя:  
1. Анонимную функцию;  
2. Метод bind().

1. Напишите функцию concat, которая соединяет две строки, разделенные каким-то символом: разделитель и строки передаются в параметрах функции. Используя карринг, создайте новую функцию hello, которая которая выводит приветствие тому, кто передан в ее параметре:

|  |
| --- |
| hello('World'); // Hello World  hello('John'); // Hello John |