**Лабораторная работа №5. Функции и замыкания**

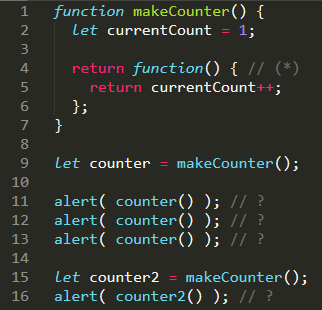
*Глобальный объект. Замыкания, внутренняя работа функции, лексическое окружение. Каррирование. Генераторы.*

**Задание:** изучите теорию и решите задачи.

**Задачи:**

1. Что выведет alert в примерах? Поясните почему так? На что ссылается [[Environment]] функций? Что будет содержать LexicalEnvironment при запуске функций? Что хранится в counter? Когда будет вызвана функция (\*)?

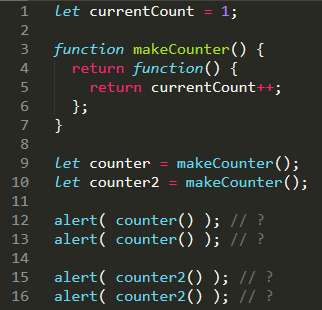
Вариант 1.



// Локальная переменная, доступная только внутри makeCounter

Вложенная функция , которая  При каждом вызове возвращает текущее значение currentCount. Затем увеличивает currentCount на 1 (currentCount++). Функция makeCounter создаёт замыкание, в котором переменная currentCount сохраняет своё состояние. В момент вызова makeCounter() создаётся лексическое окружение, для хранения его переменных и аргументов. Таким образом, два разных счётчика (counter и counter2) имеют свои собственные значения currentCount.

Вариант 2.



В этом варианте переменная currentCountv2 является глобальной, и обе функции (counterv2 и counter2v2) используют одну и ту же переменную.

• Это означает, что оба счётчика будут изменять одно и то же значение currentCountv2, что приводит к последовательному увеличению счётчика.

Во втором варианте обе функции имеют доступ к одной и той же глобальной переменной currentCountv2, что приводит к совместному использованию её значения.

**Замыкание** возникает здесь потому, что возвращаемая функция сохраняет **ссылку на currentCount.** [Замыкание](https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%BC%D1%8B%D0%BA%D0%B0%D0%BD%D0%B8%D0%B5_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) – это функция, которая запоминает свои внешние переменные и может получить к ним доступ.

[[Environment]] — это внутреннее свойство функции в JavaScript, которое ссылается на лексическое окружение, в котором функция была создана. Это окружение содержит все переменные, доступные в момент создания функции.

Как и все лексические окружения, оно содержит две вещи:

Environment Record с локальными переменными. Ссылка на внешнее окружение, которая устанавливается в значение [[Environment]] функции.

В counter хранится функция, возвращаемая из makeCounter. Эта функция замыкает переменную currentCount, которая была объявлена в makeCounter. Таким образом, при каждом вызове counter() будет возвращаться текущее значение currentCount, а затем оно будет увеличиваться на 1.

Вызов функции: alert(counter()); // 1

1. Реализуйте каррированную функцию, которая рассчитывает объем прямоугольного параллелепипеда. Выполните преобразование функции для неоднократного расчёта объема прямоугольных параллелепипедов, у которых одно ребро одинаковой длины.

[*Каррирование*](https://ru.wikipedia.org/wiki/%D0%9A%D0%B0%D1%80%D1%80%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) или карринг (currying) в функциональном программирование – это преобразование функции с множеством аргументов в набор вложенных функций с одним аргументом. При вызове каррированной функции с передачей ей одного аргумента, она возвращает новую функцию, которая ожидает поступления следующего аргумента. Каррирование позволяет фиксировать некоторые значения (например, length = 10) и использовать их в будущих вычислениях, что может быть удобно, если один из параметров остается неизменным.

1. Пользователь управляет движением объекта, вводя в модальное окно команды left, right, up, down. Объект совершает 10 шагов в заданном направлении (т.е. высчитываются и выводятся в консоль соответствующие координаты) и запрашивает новую команду. Разработайте генератор, который возвращает координаты объекта, согласно заданному направлению движения.

Обычные функции возвращают только одно-единственное значение (или ничего). Генераторы могут порождать (yield) множество значений одно за другим позволяют легко создавать потоки данных.

Основным методом генератора является next(). При вызове он запускает выполнение кода до ближайшей инструкции yield . По достижении yield выполнение функции приостанавливается, а соответствующее значение – возвращается во внешний код:

Результатом метода next() всегда является объект с двумя свойствами:

* value: значение из yield.
* done: true, если выполнение функции завершено, иначе false.

1. Какие переменные и функции сохраняются в глобальный объект window? Получите значения всех созданных переменных и функции, которые хранятся в глобальном объекте window. Переопределите переменные через глобальный объект.

Глобальный объект предоставляет переменные и функции, доступные в любом месте программы. В браузере он называется window. Ко всем свойствам глобального объекта можно обращаться напрямую:

alert("Привет");

// это то же самое, что и

window.alert("Привет");

В браузере глобальные функции и переменные, объявленные с помощью var (не let/const), становятся свойствами глобального объекта. **Переменные с let и const** не становятся свойствами window, поэтому они недоступны через window. Они создаются в глобальной области видимости, но не прикрепляются к window. мы присваиваем 100 свойству window.globalVar. Так как globalVar — это переменная, связанная с window, присваивание window.globalVar = 100 изменяет и globalVar. Поэтому console.log(globalVar); выводит 100. Здесь мы присваиваем свойству window.globalFunc новую функцию. Теперь свойство window.globalFunc указывает на новую функцию, которая выводит 'Переопределенная глобальная функция'. Это переопределяет глобальную функцию globalFunc, так как она тоже является свойством window.