06 - Функции

1. Что такое функции?

Функция – это блок программного кода на языке JavaScript, который определяется один раз и может выполняться, или вызываться, многократно. Функции могут иметь **параметры**: определение функции может включать список идентификаторов, которые называются параметрами и играют роль локальных переменных в теле функции. При вызове функций им могут передаваться значения, или аргументы, соответствующие их параметрам. Функции часто используют свои аргументы для вычисления возвращаемого значения, которое является значением выражения вызова функции.
Функции в языке JavaScript являются объектами и могут использоваться разными способами. Например, функции могут присваиваться переменным и передаваться другим функциям. Поскольку функции являются объектами, **имеется возможность присваивать значения их свойствам** и даже вызывать их методы. Функция должна делать только то, что явно подразумевается ее названием. И это должно быть одно действие.

2. Определение функций

Определение функций выполняется с помощью ключевого слова function, которое может использоваться в **инструкциях объявления функций:**

```
function showMessage() {
	alert( 'Привет всем присутствующим!' );
}
```

или **выражениях определения функций**:

```
var sayHi = function(person) {
	alert( "Привет, " + person );
};

sayHi('Вася');
```

Инструкции объявления функций «поднимаются» в начало сценария или вмещающей их функции, благодаря чему объявленные таким способом функции могут вызываться в программном коде выше объявления. Это не относится к функциям, которые определяются в виде выражений: чтобы вызвать функцию, необходимо иметь возможность сослаться на нее, однако нельзя сослаться на функцию, которая определяется с помощью выражения, пока она не будет присвоена переменной. **Объявления переменных также поднимаются вверх** но операции присваивания значений этим переменным не поднимаются, поэтому функции, определяемые в виде выражений, не могут вызываться до того, как они будут определены.

Основное отличие между **объявлениями и выражениями функций**: функции, объявленные как **объявления** (Function Declaration), создаются интерпретатором до выполнения кода.

Если нет явной причины использовать **выражения определения функций** (Function Expression) – предпочитайте **инструкции объявления функций** (Function Declaration).

В любом случае определение функции начинается с ключевого слова function, за которым указываются следующие компоненты:

• Идентификатор, определяющий имя функции. Имя является обязательной частью инструкции объявления функции: оно будет использовано для создания новой переменной, которой будет присвоен объект новой функции. В выражениях определения функций имя может отсутствовать: при его наличии имя будет ссылаться на объект функции только в теле самой функции.

• Пара круглых скобок вокруг списка из нуля или более идентификаторов, разделенных запятыми. Эти идентификаторы будут определять **имена параметров функции** и в теле функции могут использоваться как **локальные переменные**.

• Пара фигурных скобок с нулем или более инструкций JavaScript внутри. Эти инструкции составляют тело функции: они выполняются при каждом вызове функции.

3. Локальные переменные

Функция может содержать локальные переменные, объявленные через var. Такие переменные видны только внутри функции:

```
function showMessage() {
	var message = 'Привет, я - Вася!'; // локальная переменная

	alert( message );
}

showMessage(); // 'Привет, я - Вася!'

alert( message ); // <-- будет ошибка, т.к. переменная видна только внутри
```

Блоки if/else, switch, for, while, do..while не влияют на область видимости переменных. При объявлении переменной в таких блоках, она всё равно будет видна во всей функции.

4. Глобальные переменные

Функция может обратиться ко внешней переменной, например:

```
var userName = 'Вася';
function showMessage() {
	var message = 'Привет, я ' + userName;
	alert(message);
}

showMessage(); // Привет, я Вася
```

Доступ возможен не только на чтение, но и на запись. При этом, так как переменная внешняя, то изменения будут видны и снаружи функции:

```
var userName = 'Вася';
function showMessage() {
      userName = 'Петя'; // (1) присвоение во внешнюю переменную

      var message = 'Привет, я ' + userName;
      alert( message );
}

showMessage();
alert( userName ); // Петя, значение внешней переменной изменено функцией
```

Конечно, если бы внутри функции, в строке (1), была бы объявлена своя локальная переменная var userName, то все обращения использовали бы её, и внешняя переменная осталась бы неизменной.

Переменные, объявленные на уровне всего скрипта, называют **«глобальными переменными».**

Делайте глобальными только те переменные, которые действительно имеют общее значение для вашего проекта, а нужные для решения конкретной задачи – пусть будут локальными в соответствующей функции.

5. Инструкция return

Большинство (но не все) функций содержат инструкцию return. Инструкция return завершает выполнение функции и выполняет возврат значения своего выражения (если указано) вызывающей программе. Если выражение в инструкции return отсутствует, она возвращает значение undefined. Если инструкция return отсутствует в функции, интерпретатор просто выполнит все инструкции в теле функции и вернет вызывающей программе значение undefined.

6. Вложенные функции

В JavaScript допускается вложение определений функций в другие функции. Например:

```
function hypotenuse(a, b) {

      function square(x) { return x*x; }

      return Math.sqrt(square(a) + square(b));

}
```

Особый интерес во вложенных функциях представляют правила видимости переменных: они могут обращаться к параметрам и переменным, объявленным во вмещающей функции (или функциях). Например, в определении выше внутренняя функция square() может читать и изменять параметры a и b, объявленные во внешней функции hypotenuse(). Эти правила видимости, действующие для вложенных функций, играют важную роль, и мы еще вернемся к ним.

7. Именование функций

В качестве имени функции может использоваться любой допустимый идентификатор. Старайтесь выбирать функциям достаточно описательные, но не длинные имена. Искусство сохранения баланса между краткостью и информативностью приходит с опытом. Правильно подобранные имена функций могут существенно повысить удобочитаемость (а значит, и простоту сопровождения) ваших программ. Чаще всего в качестве имен функций выбираются глаголы или фразы, начинающиеся с глаголов. По общепринятому соглашению имена функций начинаются со строчной буквы. Если имя состоит из нескольких слов, в соответствии с соглашением все слова, кроме первого, начинаются с прописной буквы, примерно так: likeThis(). Имена функций, которые, как предполагается, реализуют внутреннюю, скрытую от посторонних глаз функциональность, иногда начинаются **с символа подчеркивания**.

8. Вызов функций

Программный код, образующий тело функции, выполняется не в момент определения функции, а в момент ее вызова. Функции в языке JavaScript могут вызываться четырьмя способами:

• как функции,

• как методы,

• как конструкторы и

• косвенно, с помощью их методов call() и apply().

9. Определение собственных свойств функций

Функции в языке JavaScript являются не простыми значениями, а особой разновидностью объектов, благодаря чему функции могут иметь свойства. Когда функции требуется «статическая» переменная, значение которой должно сохраняться между ее вызовами, часто оказывается удобным использовать свойство объекта функции, позволяющее не занимать пространство имен определениями глобальных переменных. Например, предположим, что надо написать функцию, возвращающую уникальное целое число при каждом своем вызове. Функция никогда не должна возвращать одно и то же значение дважды. Чтобы обеспечить это, функция должна запоминать последнее возвращенное значение и сохранять его между ее вызовами. Эту информацию можно было бы хранить в глобальной переменной, но это было бы нежелательно, потому что данная информация используется только этой функцией. Лучше сохранять ее в свойстве объекта Function. Вот пример функции, которая возвращает уникальное целое значение при каждом вызове:

```
    // Инициализировать свойство counter объекта функции. Объявления функций
    // поднимаются вверх, поэтому мы можем выполнить это присваивание до объявления функции.

    uniqueInteger.counter = 0;

    // Эта функция возвращает разные целые числа при каждом вызове.
    //       Для       сохранения       следующего
```

возвращаемого значения она использует собственное свойство.

```
function uniqueInteger() {
    return uniqueInteger.counter++; // Увеличить и вернуть свойство counter
}
```

Функция куб числа. Создать функцию, которая возвращает куб числа

Меньше, больше. Разработать функции для нахождения меньшего, большего из двух чисел.

Положительное и отрицательное. Сформировать функцию, которая возвращает истину, если передаваемое значение положительное и ложь если отрицательное.

Функция калькулятор. Написать функцию, которая в зависимости от выбора пользователя вызывает функции сложения, произведения, вычитания, деления двух чисел (эти функции тоже нужно написать самостоятельно)

Вывод прямоугольника. Написать функцию, выводящую на экран прямоугольник с высотой N и шириной K.

Простое число. Написать функцию, которая проверяет: является ли некоторое число простым? Число называется простым, если оно делится без остатка только на себя и на единицу.

Факториал. Написать функцию, вычисляющую факториал заданного значения

Сумма чисел. Написать функцию, которая получает в качестве параметров 2 целых числа и возвращает сумму чисел из диапазона между ними.

Совершенное число. Число называется совершенным, если сумма всех его делителей равна ему самому. Напишите функцию поиска таких чисел во введенном интервале.

Округление числа. Написать функцию для округления заданного числа с заданной точностью.

Среднее арифметическое массива. Написать функцию, определяющую среднее арифметическое элементов передаваемого ей массива.

Универсальная функция счета. Написать функцию, определяющую сумму или количество положительных или отрицательных элементов передаваемого ей массива. В функцию надо передать массив и два флага - 1-й указывает функции считать сумму или кол-во, 2-й - работать с отрицательными или положительными элементами.

Сумма или кол-во. Написать функцию, определяющую сумму или количество нулевых элементов передаваемого ей массива.

Сумма или кол-во четных и нечетных. Написать функцию, определяющую сумму или количество четных и нечетных элементов передаваемого ей массива.

Минимум, максимум. Написать функцию, определяющую минимум и максимум (значение и номер элемента) элементов передаваемого ей массива.

Функция простых чисел. Написать функцию, возвращающую количество простых чисел в передаваемом ей массиве.

Кратность параметрам. Функция принимает три числа и показывает на экран все числа от нуля до тысячи, которые одновременно кратны всем трем параметрам. Если таких чисел нет, тогда функция возвращает нуль.

Комбинации. Функция в качестве параметра принимает массив чисел от 0 до 9 и показывает на экран все комбинации цифр, составляющих трехзначные числа из значений элементов массива.