Front-end

JavaScript

JS. Functions

- Объявление функций
- Анонимные функции
- Стрелочные функции
- Псевдомассив arguments
- Значения по умолчанию
- Оператор return

JS. Functions

JavaScript **функция** - это блок кода, предназначенный для выполнения определенной задачи.

JavaScript функция выполняется, когда ее вызывают (вызвает) "что-то".

Функция – это фрагмент кода, который можно выполнить многократно в разных частях программы. Т.е. одни и те же действия много раз с разными исходными значениями.

JS. Functions

Синтаксис функции

Функция определяется с помощью ключевого кода function, за которым следует имя и скобки ().

Имена функций могут содержать буквы, цифры, подчеркивания и знаки доллара (те же правила, что и для переменных).

Скобки могут включать имена параметров, разделенные запятыми:

(parameter1, parameter2, ...)

Код, который будет выполнен функцией, помещается в фигурные скобки: {}

JS. Functions

Синтаксис функции

Функцию стоит называть так, чтобы название объясняло её действие.

Обычно имя функции пишут стилем camelCase и используют в начале глагол. Например createBearCounter, getBearCalculator, sendBearPolice.

В JavaScript есть два типа функций по признаку имени.

Функция называется именованной, еслиу неё есть имя.

function namedFunction() {}

Противоположность именованным функциям — анонимные. У таких имени нет

JS. Functions

Кроме этого, функции позволяют структурировать код. Например, если перед вами стоит какая-то задача, то чтобы её проще написать, её можно разбить на подзадачи и оформить каждую из них в виде функции. А затем уже используя их написать финальный код. Вдобавок к этому в такой код будет проще вносить различные изменения и добавлять новые возможности.

JavaScript позволяет создавать функцию различными способами:

- Function Declaration (просто объявить функцию в коде)
- Function Expression (создать функциональное выражение)
- Arrow Function (стрелочная функция)

JS. Functions

Function Declaration

Объявление и вызов функции

Операции с функцией в JavaScript можно разделить на 2 этапа:

- объявление (создание) функции;
- вызов (выполнение) этой функции.

1. Объявление функции. Написание функции посредством Function Declaration начинается с ключевого слова function. После чего указывается имя, круглые скобки, внутрь которых при необходимости помещаются параметры, и тело, заключённое в фигурные скобки.

В теле пишутся те действия, которые вы хотите выполнить с помощью этой функции.

JS. Functions

Function Declaration

При этом функции рекомендуется давать осмысленное название, т.е. такое, что было понятно, что она делает только исходя из её имени. Кроме этого, не нужно делать так, чтобы одна функция выполняла разные задачи. Лучше создать много различных функций, каждая из которых будет выполнять одну строго определённую задачу.

JS. Functions

Function Declaration

Параметры предназначены для получения значений, переданных в функцию, по имени. Их именование осуществляется также как переменных:

```
function getFullName(firstname, lastname) {

    console.log(`${firstname} ${lastname}`);

}
```

Параметры ведут себя как переменные и в теле функции мы имеем доступ к ним. Значения этих переменных (в данном случае firstname и lastname) определяются в момент вызова функции. Обратиться к ним вне функции нельзя.

Если параметры не нужны, то круглые скобки всё равно указываются.

JS. Functions

Function Declaration

2. Вызов функции. Объявленная функция сама по себе не выполняется. Запуск функции выполняется посредством её вызова.

```
// вызов функции
getFullName('Андрей', 'Иванов')
```

При этом когда мы объявляем функцию с именем, мы тем самым по сути создаём новую переменную с этим названием. Эта переменная будет функцией. Для вызова функции необходимо указать её имя и две круглые скобки, в которых при необходимости ей можно передать аргументы. Отделение одного аргумента от другого выполняется с помощью запятой.

JS. Functions

JavaScript функции являются объектами первого класса (First-class Objects)

Объект первого класса (first class object или first class citizen) это объект, который может быть передан как аргумент функции, возвращён из функции или присвоен переменной.

Функции в JavaScript полностью соответствуют этому определению.

Понятие "объекты первого рода (или класса - элемент, который можно передать в функцию, возвратить из функции и присвоить переменной (или константе)

JS. Functions

Function

Параметры и аргументы

Параметры – это по сути переменные, которые описываются в круглых скобках на этапе объявления функции. Параметры доступны только внутри функции, получить доступ к ним снаружи нельзя. Значения параметры получают в момент вызова функции, т.е. посредством аргументов.

Аргументы – это значения, которые мы передаём в функцию в момент её вызова.

JS. Functions

При вызове функции в JavaScript количество аргументов не обязательно должно совпадать с количеством параметров. Если аргумент не передан, а мы хотим его получить с помощью параметра, то он будет иметь значение undefined.

Передача аргументов примитивных типов осуществляется по значению. Т.е. значение переменной не изменится снаружи, если мы изменим её значение внутри функции.

```
let a = 7;
let b = 5;

function sum(a, b) {
    // изменяем параметр внутри функции
    a *= 2; // => 14
    console.log(a + b);
}

sum(a, b); // => 19

console.log(a); // => 7
```

JS. Functions

Передача значения по ссылке

Очень важно отметить при работе с функциями, что когда в качестве аргумента мы указываем ссылочный тип, то его изменение внутри функции изменит его и снаружи:

```
// объявим переменную users и присвоим ей массив пользователей
const users = ['vanya', 'fedya', 'nastya'];

// объявим функцию changeUserName
function changeUserName(users) {
  // изменим значение
  users[0] = 'Иван';
}

// вызовем функцию changeUserName
changeUserName(users);

// выводим значение в консоль
console.log(users[0]); // Иван
```

В этом примере переменные ссылаются на один и тот же объект в памяти. И когда мы изменяем объект внутри функции он изменится.

При этом не рекомендуется изменять внешние относительно функции переменные. Т.е. если на вход функции мы передаём объект и хотим изменить его, то лучше создать копию этого объекта и изменять уже его. Таким образом мы не изменим внешний объект и код функции останется чистой.

JS. Functions

Локальные и внешние переменные

Переменные, объявленные внутри функции, называются локальными. Они не доступны вне функции. По сути это переменные, которые действуют только внутри функции.

При этом, когда мы обращаемся к переменной и её нет внутри функции, она берётся снаружи. Переменные объявленные вне функции являются по отношению к ней внешними.

Все переменные, созданные внутри функции и её параметры, как мы уже отмечали выше, являются её локальными переменными. Они доступны только внутри неё, а также в других функциях, вложенных в неё, если там нет переменных с такими же именами.

JS. Functions

Работа с аргументами через arguments

Получить доступ к аргументам в JavaScript можно не только с помощью параметров, но также посредством специального массивоподобного (псево-массив) объекта arguments, доступ к которому у нас имеется внутри функции. Кроме этого он также позволяет получить количество переданных аргументов.

Доступ к аргументам через arguments выполняется точно также как к элементам обычного массива, т.е. по порядковому номеру

```
let a = 7;
let b = 5;

function sum(a, b) {
    console.log(arguments)
}

sum(a, b);
```

JS. Functions

Частая ошибка – попытка применить методы массивов к arguments. Это невозможно.

```
function sayHi() {
    let a = arguments.shift(); // => arguments.shift is not a function
}

sayHi(1);
```

Дело в том, что arguments – это не массив Array.

В действительности, это обычный объект, просто ключи числовые и есть length. На этом сходство заканчивается. Никаких особых методов у него нет, и методы массивов он тоже не поддерживает.

Впрочем, никто не мешает сделать обычный массив из arguments.

JS. Functions

Функция – это объект

Функция в JavaScript – это определённый тип объектов, которые можно вызывать. А если функция является объектом, то у неё как у любого объекта имеются свойства. Убедиться в этом очень просто. Для этого можно воспользоваться методом console.dir() и передать ему в качестве аргумента функцию.

JS. Functions

Возврат значения

Функция всегда возвращает значение, даже если мы не указываем это явно. По умолчанию она возвращает значение undefined.

Для явного указания значения, которое должна возвращать функция используется инструкция return. При этом значение или выражение, результат которого должна вернуть функция задаётся после этого ключевого слова.

Инструкции, расположенные после return никогда не выполняются

JS. Functions

Функция, которая возвращает функцию

В качестве результата функции мы можем также возвращать функцию.

Например:

```
function outer(a) {
    return function (b) {
        return a * b;
    };
}

// в one будет находиться функция, которую возвращает outer(3)
const one = outer(3);
// в two будет находиться функция, которую возвращает outer(4)
const two = outer(4);
// выведем в консоль результат вызова функции one(5)
console.log(one(5)); // 15
// выведем в консоль результат вызова функции two(5)
console.log(two(5)); // 20
```

JS. Functions

Вызовы функции outer(3) и outer(4) возвращают одну и туже функцию, но первая запомнила, что a = 3, а вторая - что a = 4. Это происходит из-за того, что функции в JavaScript «запоминают» окружение, в котором они были созданы. Этот приём довольно часто применяется на практике. Так как с помощью него мы можем, например, на основе одной функции создать другие, которые нужны.

В примере, приведённом выше, мы могли также не создавать дополнительные константы one и two. А вызвать сразу после вызова первой функции вторую.

```
// выведем в консоль результат вызова функции one(5)

console.log(outer(3)(5)); // 15

// выведем в консоль результат вызова функции two(5)

console.log(outer(4)(5)); // 20
```

При создании таких конструкций нет ограничений по уровню вложенности, но с точки зрения разумности этим лучше не злоупотреблять.

JS. Functions

Функцию, приведённую в коде мы можем также создать и так:

```
function outer(a) {
    function inner(b) {
        return a * b;
    }
    return inner;
}

//   Кроме этого в качестве результата мы можем также возвратить внешнюю функцию:
function funcA() {
    return 'Привет!';
}

function funcB() {
    return funcA;
}

console.log(funcB()()); // Привет!
```

JS. Functions

Function Expression

Данный синтаксис позволяет нам создавать новую функцию в середине любого выражения.

Это выглядит следующим образом:

```
const sayHi = function () {
    console.log("Привет");
};

sayHi();
```

Здесь мы можем видеть переменную sayHi, получающую значение, новую функцию, созданную как function() { alert("Привет"); }.

После ключевого слова function нет имени. Для Function Expression допускается его отсутствие.

JS. Functions

Arrow Function (стрелочная функция)

Стрелочные функции строятся по единой схеме, при этом структура функций может быть, в особых случаях, упрощена. Базовая структура стрелочной функции выглядит так:

```
(argument1,... argumentN) => {

        // тело функции

}
```

```
const arrowFunc = () => {
    console.log('Arrow func');
};

arrowFunc();
```

У стрелочных функций есть два основных преимущества перед традиционными функциями. Первое — это очень удобный и компактный синтаксис.

JS. Functions

Arrow Function (стрелочная функция)

Это очень похоже на то, как устроены обычные функции, главные отличия заключаются в том, что здесь опущено ключевое слово function и добавлена стрелка после списка аргументов.

Если 1 аргумент, круглые скобки можно убрать

Если тело функции состоит из одного выражения, значение которого нужно вернуть как результат выполнения функции, то фигурные скобки можно опустить, return также не нужен

```
const arrowFunc = name => console.log(name); // => John

arrowFunc('John');
```

JS. Functions

Значение параметров функции по умолчанию

Параметрам функции можно присвоить дефолтное значение. Оно будет использоваться, когда при вызове функции этому параметру не будет задано значение с помощью аргумента:

```
function setBgColor(selector, color = 'green') {
    console.log(`Селектор: ${selector}`)
    console.log(`Цвет: ${color}`)
}

setBgColor('body');
```

```
function setBgColor(selector, color) {
    // в ранних версиях
    color = color || 'green';

    console.log(`Селектор: ${selector}`)
    console.log(`Цвет: ${color}`)
}

setBgColor('body');
```