JavaScript

Урок №11: Основы.Функции.

Автор Бахшиллоев Бехзод
Ментор по Frontend в PRO UNITY

Функции

FUNCTION DECLARATION

```
function foo() {
   код
}
```

Создается до начала выполнения скрипта,можно вызвать перед объявлением

FUNCTION EXPRESSION

```
let foo = function() {
   код
}
```

Создается только тогда, когда доходит поток кода,можно вызвать только после объявления.

СТРЕЛОЧНЫЕ ФУНКЦИИ

```
() =>
```

Не имеет своего контекста (this)

Функции

Функция в языке программирования - это выражение или последовательность выражений, собранных в одном месте и названных одной переменной.
Предположим, мы имеем несколько выражений, которые планируются повторяться во многих местах нашего кода:

```
let a = 2;
let b = a +2;
console.log(‘Result is ’ + b); // ‘Result is 4’
```

Чтобы избавить программиста от необходимости писать эти строки каждый раз, когда нужно вызвать эти три выражения, существуют функции.

Функции являются основными «строительными блоками» программы.

Примеры встроенных функций вы уже видели – это **alert(message)**, **prompt(message, default)** и **confirm(question)**. Но можно создавать и свои.

Полезные ссылки: Функции

Функции. Объявление.

Чтобы указать javascript, что мы хотим объединить наши выражения в функцию, нужно использовать ключевое слово **function**.

Для того, чтобы потом можно было как-то вызвать нашу функцию, нужно дать ей имя (как и любой переменной) *function* **name**.

Кроме того, после имени функции мы должны указать круглые скобки (**в них могут быть аргументы**) ***function name()***.

После имени функции и круглых скобок следует тело функции - все те выражения, которые мы хотим вызвать при выполнении функции.

Тело функции заключается в фигурные скобки *function name()* **{ /* function body */ }.**

```
function foo () {
  let a = 2;
  let b = a +2;
  console.log(‘Result is ’ + b); // ‘Result is 4’
}
```

Функции. Вызов.

Когда мы хотим воспользоваться тем кодом который написан в функции мы ее должны **вызвать**.

Для того чтобы вызвать функцию нужно написать имя функции и круглые скобки без пробелов.

```
function foo () {
  let a = 2;
  let b = a +2;
  console.log(‘Result is ’ + b);
}

foo(); // ‘Result is 4’
```

Функции. Возвращаемое значение. Return.

Функция всегда производит какое-то действие. Любое действие имеет **результат**. Результат выполненной функции в JS определяется ключевым словом **return**.

Слово **return** определяет конечный результат выполнения функции.

Если **return** пропущено, функция вернет результат **undefined**.

Если после **return** указано выражение или переменная - функция вернет результат этого выражения или переменную.

function **foo**() {} *// результат - undefined*
function **foo**() { return; } *// результат - undefined*
function **foo**() { return 25; } *// результат - 25*
function **foo**() { return 5 + 10; } *// результат - 15*

Функции. Возвращаемое значение. Return.

Возвращаемое функцией значение может быть использовано как и любое другое значение в JavaScript: может быть присвоено в переменную, может быть прибавлено к строке или сложено с другим числом и т.д.

function **foo**() { let a = 5; return a * 2; }
let result = foo(); *// в переменную result присваивается значение 10*
15 + foo(); *// 15 + 10 → 25*
‘В этой комнате ’ + foo() + ‘ человек’
// "В этой комнтае 10 человек"

Функции. Методы объявления.

Function Declaration:

```
function foo() {}

test(); // 18

function test() { return 18 }
```

Нельзя объявлять функции внутри блоков при ‘use strict’!

```
if (a) {
    function goo() {return ‘works!’}
}
```

Function Expression:

```
var foo = function () {}

test(); // Error: test is not a function

var test = function (){ return 18 }

if (a) {
    var goo = function () {return ‘works!’};
}
```

Не сработает

```
if (a) {
    let goo = function () {return ‘works!’};
}

if (goo) goo();
```

Функции. Всплытие.

```
foo();
// 'I am a function!'
// ‘Executed’

function foo() {
    console.log('I am a function!');
    return 'Executed';
}
```

```
foo();
// Uncaught TypeError: foo is not a
function(...)

var foo = function () {
    console.log('I am a function!');
    return 'Executed';
};
```

Функции. Self invoked function.

```
(function () {
    console.log('self invoked function');
})(); // self invoked function
```

```
(function () {
    создали функцию
})(); // и тут же (на месте) её вызвали
```

```
(function () {
    console.log('self invoked function');
}()); // self invoked function
```

```
(function () {
    создали функцию
}()); // и тут же (на месте) её вызвали
```

```
(function (value) {
    console.log(value);
})(2016); // 2016
```

```
(function (value) {
   функция принимает аргумент value
})(2016); // 2016
```

Функции. Область видимости.

Область видимости - то место, где видна переменная.

Если переменная объявлена вне функций **(глобально)**, она видна в любом месте кода.

Если переменная объявлена внутри функции **(локальная переменная)**, то она видна только внутри этой функции, а также для внутренних функций.

Локальные переменные не объявленные через ключевые слова **let, var или const “перезаписывают”** глобальные переменные с таким же именем.

Аргументы функции также считаются локальными переменными и доступны только внутри функции и всем внутренним функциям.

Функции. Область видимости.

```
var a = ‘global variable’,
      b = 2016;

function foo() {
    var a = ‘local variable’;
    console.log(a, b)
}

foo() // “local variable”, 2016
console.log(a) // “global variable”
```

Функции. Область видимости.

```
var cat = “I’m a global cat!” // 3. потом ищет в глобальном окружении

function findCat(cat) { // 2. потом она ищет внутри аргументов
    var cat = ‘I am a local cat!’; // 1. сначала функция ищет переменную внутри себя

    return cat;
}
```

Функции. Аргументы.

Обычно функции дают возможность выполнять один и тот же код, но с разными значениями - то есть вместо конкретных **(определенных)** значений используются **переменные**.
Во время создания функции мы используем переменные, а во время вызова функции - **конкретные значения (то, чему станет равна переменная).**

Чтобы что-то передать внутрь функции, мы используем **аргументы**.
Аргументы передаются внутри круглых скобок.
Аргументы - это обычные переменные, которые еще не определены (undefined); их особенность в том, **что они становятся чем-то определенным во время вызова функции.**

Функции. Аргументы.

Рассмотрим функцию:

```
function greeting() {
    let name = “Denis”;
    console.log(`Hello ${name}`);
}
```

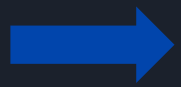
Она однообразна и не универсальна, так как не позволяет как-то изменить переменные внутри её тела.

Функции. Аргументы.

Изменим нашу функцию так, чтобы вместо **name = “Denis”** мы могли использовать **name**, равное любому числу.

Для этого нам нужно **name** вынести в аргумент, а функцию вызывать как **greeting**(“Maks”) или **greeting**(“Denis”) и тд.

```
function greeting() {
    let name = “Denis”;
    console.log(`Hello ${name}`);
}
```



```
function greeting(name) {
    console.log(`Hello ${name}`);
}

greeting(“Denis”); // Hello Denis
greeting(“Maks”) // Hello Maks
```

Функции. Аргументы.

Аргументов может быть больше одного и они разделяются запятой:

```
function calc(a, b) {
    return a + b;
}

calc(2, 3); // 5
calc(23, 42) // 65
```

Если вы при создании функции объявили аргумент, а при вызове функции его не передали, то переменная, соответствующая аргументу, будет равна **undefined**

calc(2) // 2 + undefined -> **NaN**

Функции. Аргументы. Проверка.

Часто наши функции очень строго зависят от того какого типа или с какими значениями они принимают аргументы, для того чтобы избежать ошибок при расчетах нам нужно проверять правильные ли данные мы принимаем на входе:

```
function calc(a, b) {
    if (typeof a !== “number” || isNaN(a)) return console.log(“error”);
    if (typeof b !== “number” || isNaN(b)) return console.log(“error”);

    return a + b;
}

calc(); // error
calc(23) // error
```

Функции. Аргументы. Значения по умолчанию.

Очень часто бывает когда нам нужно подставить какое то значение переменной если оно не было передано.

```
function calc(a = 0, b = 0) {
    if (typeof a !== “number” || isNaN(a)) return console.log(“error”);
    if (typeof a !== “number” || isNaN(a)) return console.log(“error”);

    return a + b;
}

calc(); // 0
calc(23) // 23
```

Полезные ссылки: Аргументы по умолчанию

Функции. Аргументы. Arguments.

Иногда мы не знаем, с каким количеством аргументов может быть вызвана функция. Не часто, но иногда это случается.

Для подобных случаев внутри любой функции доступно специальное свойство **arguments** - оно содержит список всех значений, с которыми вызвана функция.

Даже если вы создали функцию, которая не содержит ни одного аргумента, то при вызове функции с параметрами, все они будут содержатся в списке **arguments**.

function foo() { return **arguments**; }
foo(); ***// []***
foo(1, 2, 3); ***// [1, 2, 3]***
foo('Hello', 'By-by'); ***// ['Hello', 'By-by']***
foo({car: 'Lexus', price: 64000}); ***// [{car: 'Lexus', price: 64000}]***

Задачи:

1. Создать функцию **multiply**, которая будет принимать **любое количество чисел** и возвращать их произведение: *multiply(1,2,3) = 6 (1*2*3)*

   Если нет ни одного аргумента, вернуть ноль: multiply() // 0

2. Создать функцию, которая принимает строку и возвращает строку-перевертыш: *reverseString('test') // "tset".*

3. Создать функцию, которая в качестве аргумента принимает строку из букв и возвращает строку, где каждый символ разделен пробелом и заменен на юникод-значение символа:

   getCodeStringFromText('hello') // "104 101 108 108 111"

   подсказка: для получения кода используйте специальный метод

4. Создать функцию угадай число. Она принимает число от 1-10 **(обязательно проверить что число не больше 10 и не меньше 0)**. Генерирует рандомное число от 1-10 и сравнивает с заданным числом если они совпали то возвращает *"Вы выиграли"* если нет то *"Вы не угадали ваше число 8 а выпало число 5".* Числа в строке указаны как пример вы подставляете реальные числа.