JavaScript Base

Занятие 4. Функции

IT Education Academy

Содержание

- Для чего нужны функции?
- Способы создания (объявления) функций
- Вызов функций
- Области видимости переменных
- Разновидности функций

Для чего нужны функции?

Фу́нкция в программировании — это фрагмент программного кода (*подпрограмма*), к которому можно обратиться из другого места программы.

Они нужны **чтобы не повторять один и тот же код во многих местах**. 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.

Способы создания (объявления) функций

Пример объявления функции:

```
function sayHello() {
  alert( 'Hello, World!' );
}

sayHello();
sayHello();
```

Так тоже будет работать:

```
sayHello();
sayHello();

function sayHello() {
  alert( 'Hello, World!' );
}
```

Этот способ объявления функции называется «**Function Declaration**», создаются интерпретатором до выполнения кода.

Способы создания (объявления) функций

Так тоже будет работать:

```
var sayHi = function() {
  alert( 'Hello, World!' );
};

sayHi();
```

А так уже нет:

```
sayHi();

var sayHi = function() {
  alert( 'Hello, World!' );
};
```

Это альтернативный синтаксис для объявления функции, он называется «**Function Expression**» (функциональное выражение)

Способы создания (объявления) функций

Основное **отличие** между **Function Declaration** и **Function Expression**: функции, объявленные как **Function Declaration**, создаются интерпретатором **до** выполнения кода.

Это из-за того, что JavaScript перед запуском кода ищет в нём Function Declaration (их легко найти: они не являются частью выражений и начинаются со слова function) и обрабатывает их.

А Function Expression создаются в процессе выполнении выражения, в котором созданы, в данном случае — функция будет создана при операции присваивания `sayHi = function`...

Как правило, возможность Function Declaration вызвать функцию до объявления — это удобно, так как даёт больше свободы в том, как организовать свой код.

Выбор имени функции

Имя функции следует тем же правилам, что и имя переменной. Основное отличие — оно должно быть глаголом, т.к. функция — это действие.

Как правило, используются глагольные префиксы, обозначающие общий характер действия, после которых следует уточнение.

Функции, которые начинаются с «*show*» — что-то показывают:

```
showMessage(..);
```

Функция должна делать только то, что явно подразумевается её названием. И это должно быть одно действие.

Если оно сложное и подразумевает поддействия – может быть имеет смысл выделить их в отдельные функции? Зачастую это имеет смысл, чтобы лучше структурировать код.

Имена функций, которые используются *очень часто*, иногда делают сверхкороткими.

Например, во фреймворке jQuery есть функция $

Области видимости переменных

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

```
function showMessage() {
  var message = 'Hello, World!'; //
local variable

  alert( message );
}

showMessage();
alert( message ); // <-- Error
```

Более того:

```
var message = 'Hi!'; // variable
function showMessage() {
  var message = 'Hello, World!'; //
local variable

  alert( message ); // Hello, World!
}

showMessage();
alert( message ); // Hi!
```

Области видимости переменных

Функция может обратиться ко **внешней переменной**.

```
var userName = 'Ivan';

function showMessage() {
  var message = 'Hello ' + userName;
  alert(message);
}

showMessage(); // Hello Ivan
```

Доступ возможен не только на чтение, но и на запись.

```
var userName = 'Ivan';

function showMessage() {
  userName = 'John'; // global variable
  var message = 'Hello ' + userName;
  alert( message );
}

showMessage();
alert( userName ); // John
```

Параметры (агрументы) функции

При вызове фунции, ей можно передать параметры.

```
function showMessage(from, text) {
  from = '**' + from + '**'; //
changing local variable
  alert( from + ': ' + text );
}

var from = "Mary";
showMessage(from, "Hello");
alert( from ); // old from value
```

Параметры копируются в локальные переменные функции.

Например, в коде слева есть внешняя переменная **from**, значение которой при запуске функции копируется в параметр функции с тем же именем. Далее функция работает уже с параметром.

В объявлении функции мы должны обязательно объявить параметры.

Аргументы по умолчанию

Функцию можно вызвать с любым количеством аргументов.

Если параметр не передан при вызове — он считается равным **undefined**.

Например, функцию показа сообщения `showMessage(from, text)` можно вызвать с одним аргументом.

```
function showMessage(from, text) {
  text = text || 'No text here(';

  alert( from + ": " + text );
}
showMessage("Mary", "Hello!");
// Mary: Hello!
showMessage("Mary"); // Mary: No text here(
```

Аргументы по умолчанию

При объявлении функции необязательные аргументы, как правило, располагают в конце списка.

Для указания значения «*по умолчанию*», то есть, такого, которое используется, если аргумент не указан, используется два способа:

- Можно проверить, равен ли аргумент **undefined**, и если да — то записать в него значение по умолчанию.
- Использовать оператор **||**.

Второй способ считает, что аргумент отсутствует, если передана **пустая строка**, **0**, или вообще любое значение, которое в логическом контексте является **false**.

Если аргументов передано больше, чем надо, например `showMessage("Маша", "привет", 1, 2, 3)`, то ошибки не будет. Но, чтобы получить такие «лишние» аргументы, нужно будет прочитать их из специального объекта *arguments*.

Возврат значения

Функция может возвратить результат, который будет передан в вызвавший её код.

Для возврата значения используется директива **return**.

Она может находиться в любом месте функции. Как только до неё доходит управление — функция завершается и значение передается обратно.

В случае, когда функция не вернула значение или return был без аргументов, считается что она вернула **undefined**.

```
function checkAge(age) {
  if (age > 18) {
    return true;
  } else {
    return confirm('Are parents ok with this?
');
  }
}

var age = prompt('Your age?');

if (checkAge(age)) {
  alert( 'Access granted' );
} else {
  alert( 'Access denied' );
}
```

Использование объекта arguments

Доступ к аргументам может быть получен непосредственно (без обращения по именам объявленных параметров).

Ключевое слово **arguments** — это структура, где содержится список входных аргументов в порядке передачи при вызове. Общее количество аргументов содержится в **arguments.length**, а значение отдельного аргумента можно получить как **arguments[i]**.

```
function myConcat(separator) {
   var result = "";
   // iterate through arguments
   for (var i = 1; i < arguments.
length; i++) {
      result += arguments[i] +
separator;
   }
   return result;
}

myConcat(", ", "red", "orange",
"blue"); // "red, orange, blue, "
```

Свой порядок сортировки с помощью функций

Для указания своего порядка сортировки в метод **arr.sort(fn)** нужно передать функцию **fn** от двух элементов, которая умеет сравнивать их.

Внутренний алгоритм функции сортировки умеет сортировать любые массивы — апельсинов, яблок, пользователей, и тех и других и третьих — чего угодно. Но для этого ему нужно знать, как их сравнивать. Эту роль и выполняет **fn**.

Если эту функцию не указать, то элементы сортируются как строки.

```
function compareNumeric(a, b) {
  if (a > b) return 1;
  if (a < b) return -1;
}

var arr = [ 1, 2, 15 ];
arr.sort(compareNumeric);

alert(arr);  // 1, 2, 15
```

```
arr.sort(function(a, b) {
  return a - b;
});
```

Задача 1

Создайте функцию принимающую на вход 2 числа, складывающую их и выводящую результат в консоль.

Задача 2

Написать функцию `getStrings()`, принимающую массив в качестве аргумента, и возвращающую массив, в котором будут только строки из изначального.

```
console.log(getStrings([1, 'b', 3, 'c'])); // ['b', 'c']
console.log(getStrings(['say', 12, 'hello'])); // ['say', 'hello']
console.log(getStrings([33, 'Go!', 300])); // ['Go!']
```

Задача 3

Напишите функцию, рассчитывающую размер ипотечного аннуитетного платежа. Функция должна принимать размер кредита в евро, процентную ставку за период и количество периодов. Формула расчета:

$A = \frac{S * p}{1 - (1 + p)^{-n}}$ где **S** — величина (тело) кредита, **p** — величина процентной ставки за период (в долях), **n** — количество периодов.

$$a^{-n} = \frac{1}{a^n}$$

IT Education Academy

Задача 4

Написать функцию принимающую строки в качестве аргументов (arguments), и выводящую их в консоль пронумерованными.

```
log('Hello', 'World', 'again!'); /*
1. Hello
2. World
3. again! */
```

Задача 5

Есть массив `[1,5,23,346,2345,5,45,657,42,3]` отсортировать его в обратном порядке без использования **`reverse()`**.

Ссылки

- https://learn.javascript.ru/function-basics
- https://learn.javascript.ru/function-declaration-expression
- https://learn.javascript.ru/arguments-pseudoarray

Разное

- https://tproger.ru/articles/free-programming-books/#javascript
- https://habrahabr.ru/hub/javascript/
- https://code.org/learn

Вопросы?