Шевяков В. И.

Методические материалы

JavaScript. Функции

План

Понятие функции

Функция – набор инструкций, который определяется единожды и может быть позднее вызван (использован в коде) любое количество раз. Главная цель создания функций: **избавление от дублирования кода**.

Синтаксис объявления функций

```
function идентификатор(арг_1, арг_2, ..., арг_n) {
    инструкция_1;
    инструкция_2;
    ...
    инструкция_n;

    return выражение;
}
```

Вначале идет ключевое слово **`function`**, после него имя функции, затем список параметров в скобках и тело функции – код, который выполняется при ее вызове. Оператор **`return`** указывает на то, что функция должна вернуть при завершении своего выполнения.

Возврат значения – это процесс передачи функцией результата в ту область кода, из которой она была вызвана. В момент, когда вызван оператор **`return`**, функция мгновенно завершает свою работу.

Весь код функции, который идет после вызова оператора return будет проигнорирован!

Название функции должно представлять собой **глагол** или **глагольное выражение** например: «showMessage» (показать сообщение).

В качестве примера, разработаем функцию для возведения числа в квадрат. Эта функция будет принимать в качестве аргумента число и возвращать результат в виде квадрата этого же числа.

```
function squareNumber(number) {
  let result = number * number;

  return result;
}
```

Разъяснение деталей синтаксиса объявления функции **squareNumber** приведены на рисунке 1.

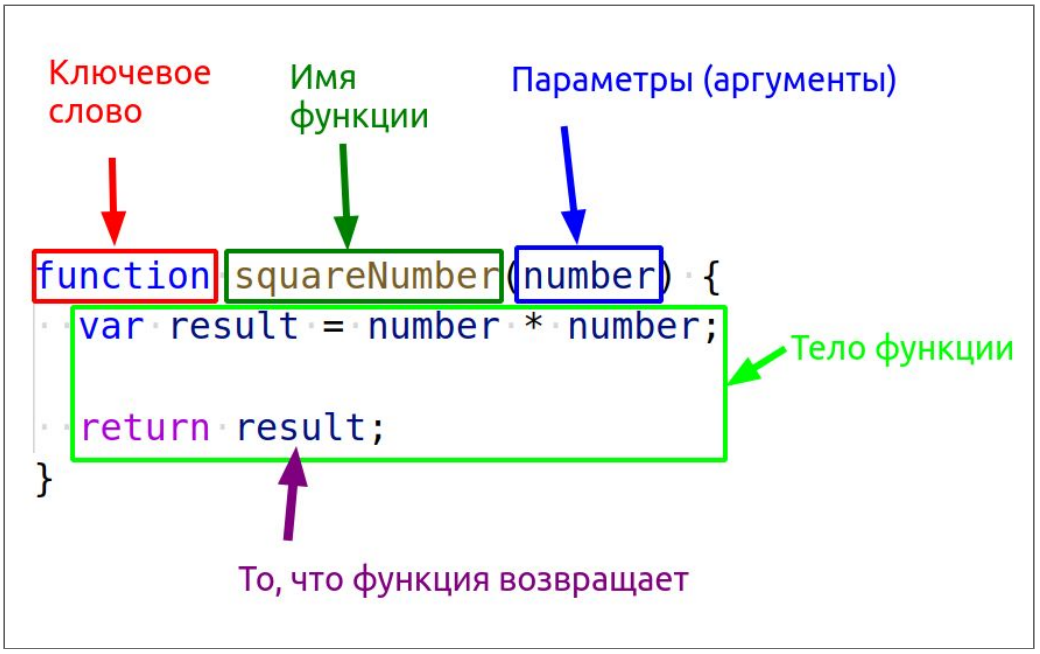

Рисунок 1 — Синтаксис объявления функции **squareNumber**

Синтаксис вызова функций

В общем виде синтаксис вызова функции выглядит следующим образом:

```
идентификатор(арг_1, арг_2, ..., арг_n)
```

Рассмотрим пример создания 3-х переменных и записи в них результатов выполнения функций возведения в степень аргументов 2, 5 и числа π.

```
let squareOf2 = squareNumber(2);
let squareOf5 = squareNumber(5);
let squareOfPI = squareNumber(Math.PI);
```

Однако, вызов функции является выражением, поэтому он может находится в любом месте кода, где ожидается выражение. Даже в скобках для передачи аргументов другой или такой же функции:

```
squareNumber(squareNumber(2)); // 16
```

В этом примере мы возвели число 2 в квадрат. Затем, возвели получившийся результат в квадрат. Таким образом получили результат 16.

При передаче функции меньшего количества аргументов, чем она принимает, **ошибки НЕ произойдет**. Вместо этого аргумент, который не был передан будет равен внутри функции значению **`undefined`**.

```
function sayHello(name) {
  console.log("Hello, " + name + "!");
}

sayHello("Tom"); // Hello, Tom!
sayHello(); // Hello, undefined!
```

Обратите внимание на то, что в функции `sayHello` **отсутствует оператор `return`**. Что же в этом случае возвращает функция? В такой ситуации функция возвращает значение **`undefined`**.

```
console.log(sayHello()); // undefined
```

Примеры встроенных функций

- `alert(message);`
- `prompt(message, default);`
- `confirm(question).`

Функции обратного вызова

Любая функция может быть передана другой в качестве аргумента. Для этого используется имя функции, которую мы хотим передать и использовать внутри другой. Такие функции называются **функциями обратного вызова** или **callback-функциями**.

Область видимости функции

Функция может содержать локальные переменные, объявленные через `var`. Такие переменные видны только внутри функции. Говорят Переменные, созданные с помощью оператора `var` имеют «функциональную область видимости».

```
var userName = "Alex";

function someFunction() {
  var userName = "Tom";
}

someFunction();

console.log(userName); // "Alex"
```