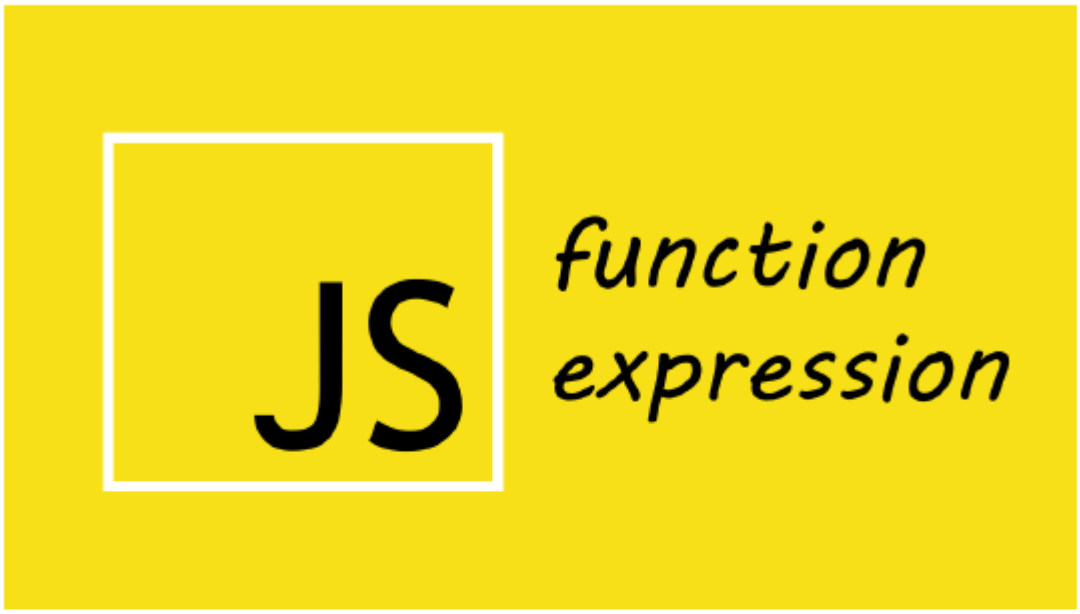
Лабораторная работа № 11. Функциональные выражения и стрелочные функции в JavaScript



На этом занятии изучим функциональные выражения и стрелочные функции. Рассмотрим почему не следует использовать традиционные функции, а также отличия между всеми этими способами создания функций.

Функциональные выражения

В JavaScript создавать функции можно не только посредством **объявления**, но также с помощью **функциональных выражений и стрелочных функций**.

Ключевое слово `function` также применяется для определения **функций в выражениях**.

Функциональное выражение (от английского *Function Expression*) очень похоже на обычное объявление функции:

```
function(a, b) {
  const sum = a + b;
  return sum;
}
```

Единственное отличие между ними только в том, что у функционального выражения может **отсутствовать имя**. Т.е. сразу после ключевого слова `function` идут круглые скобки, а в них параметры. Функциональные выражения без имени называются **анонимными функциями**.

```
function funcDeclaration() {
  return 'Обычное объявление функции';
}

const funcExpression = function() {
  return 'Функциональное выражение';
}
```

Здесь функциональное выражение мы присвоили переменной `funcExpression`. В итоге, у функционального выражения по сути будет имя (название переменной). Затем, используя эту переменную мы можем вызвать данную функцию:

```
funcExpression();
```

В этом примере присвоим переменной `sum` функциональное выражение. А затем вызовем данную функцию, используя эту переменную:

```
const sum = function(num1, num2) {
  return num1 + num2;
};

// вызов функции, используя переменную sum
sum(7, 4);
```

Если вы хотите внутри тела функции сослаться на эту же функцию, то можно создать **именованное функциональное выражение**:

```
const factorial = function factorialInner(num) {
  if (num <= 1) {
    return 1;
  }
  // использование factorialInner для вызова функции
  return factorialInner(num - 1) * num;
};

// выведем результат вызова функции factorial(5) в консоль
console.log(factorial(5)); // 120

// при попытке вызвать функцию по имени factorialInner получим ошибку
console.log(factorialInner(5)); // Uncaught ReferenceError: factorialInner is not defined
```

Вызов функции внутри себя используется для создания рекурсий. В этом примере именованное функциональное выражение имеет название `factorialInner`. По этому имени мы можем вызвать эту функцию внутри её же тела. Вне тела обратиться к этой функции по `factorialInner` нельзя.

При этом функциональное выражение присвоено переменной `factorial`, объявленной с помощью `const`. Используя эту переменную (т.е. `factorial`) мы можем вызвать данную функцию.

Но вызвать функцию внутри её тела можно не только по имени, но также с помощью свойства `arguments.callee`:

```
const factorial = function(num) {
  if (num <= 1) {
    return 1;
  }
  return arguments.callee(num - 1) * num;
};
```

Ещё очень часто функциональное выражение используется как колбэк-функция. Т.е. как функция, которая передаётся в качестве аргумента в другую функцию. И эта другая функция где-то внутри себя вызывает эту callback функцию.

Это связано с тем, что в этом случае не нужно создавать функцию с именем, когда вы хотите просто передать её в другую функцию. Можно использовать анонимную функцию.

Пример использования функционального выражение в вызове другой функции:

```
setTimeout(function() {
  console.log('Сообщение, которое будет выведено в консоль через 1 секунду!');
}, 1000);
```

В этом примере используется стандартная функция `setTimeout`, которая доступна как в браузере, так и в Node.js. Она принимает на вход колбэк-функцию и количество миллисекунд, через которые нужно вызвать эту колбэк-функцию. Здесь нет смысла давать имя вот этой функции. Достаточно просто использовать анонимное функциональное выражение.

Стрелочные функции

Стрелочные функции (от английского *arrow function*) – это функции, которые имеют немного другой более современный синтаксис. При создании стрелочных функциях не используется ключевое слово function. Появились стрелочные функции в стандарте ECMAScript 2016 (6 редакции).

Пример функции, выводящей в консоль среднее арифметическое двух чисел:

```
(num1, num2) => {
  const result = (num1 + num2) / 2;
  console.log(result);
}
```

У стрелочной функции нет имени. Начинается стрелочная функция сразу же с `()`, внутри которых при необходимости описываются параметры. Далее идёт специальная стрелочка, которая состоит из знака `=` и `>`. Этот специальный синтаксис как раз и делает эту функцию стрелочной. После этого идёт тело функции, внутри которого мы описываем действия, которая она будет выполнять при её вызове. В теле как в традиционной функции опционально с помощью `return` мы можем возвращать результат.

Как дать имя стрелочной функции? Точно также как анонимному функциональному выражению, т.е. путём его присваивания переменной.

```
const average = (num1, num2) => {
  const result = (num1 + num2) / 2;
  console.log(result);
}
```

В этом примере мы присвоили стрелочную функцию переменной `average`. То есть, по сути, дали ей имя.

После этого мы можем вызвать эту функцию используя данную переменную:

```
average(7, 5); // 6
```

Стрелочную функцию мы можем передать в качестве аргумента другой функции, т.е. использовать как колбэк-функцию:

```
setTimeout(() => {
  console.log('Это сообщение будет выведено в консоль через 1 секунду!');
}, 1000);
```

В этом примере у стрелочной функции нет параметров, поэтому здесь просто указываются круглые скобки.

В отличие от функционального выражения синтаксис стрелочной функции является более компактным. В основном это связано с тем, что он не содержит ключевое слово `function`.

Сокращение синтаксиса в стрелочных функциях

1. Если у стрелочной функции один параметр, то заключать его в круглые скобки не обязательно:

```
const greeting = name => {
  console.log(`Привет, ${name}`);
};
```

Но для удобства чтения стрелочной функции круглые скобки лучше не опускать:

```
const greeting = (name) => {
  console.log(`Привет, ${name}`);
};
```

2. Если тело функции состоит из одного выражения, значение которого нужно вернуть как результат выполнения функции, то фигурные скобки можно опустить:

```
const average = (num1, num2) => (num1 + num2) / 2;
```

В этом примере стрелочная функция возвращает результат выражения неявно, т.е. без необходимости использовать ключевого слово `return`.

Этот же примере без сокращенного варианта:

```
const average = (num1, num2) => {
  return (num1 + num2) / 2;
}
```

Данный вариант сокращения является очень популярным и довольно часто используется, т.к. позволяет уместить запись функции на одну строку.

Ещё примеры

Пример, в котором создадим стрелочную функцию, возвращающую массив определённой длины, заполненный случайными числами от 0 до 9.

```
const fillArr = (numElements) => {
  const arr = [];
  for (let i = 0; i < numElements; i++) {
    arr.push(parseInt(Math.random() * 10));
  }
  return arr;
};

// вызов функции fillArr
console.log(fillArr(5)); // [1, 4, 6, 4, 9]
```

Если стрелочная функция не имеет параметров, или их два и более, то круглые скобки в этом случае нужно писать обязательно:

```
// () - необходимо указывать при отсутствии параметров
const result = numElements = () => {
  console.log('Привет, мир!');
};

result(); // 'Привет, мир!'
```

Самовызывающаяся функция (IIFE)

Самовызывающаяся функция или IIFE - это функция, которая вызывается сразу же как только до неё дойдет интерпретатор кода.

Она используется для создания закрытой области видимости, и применяется в паттерне «модуль».

Для создания самовызывающейся функции, её необходимо обернуть в круглые скобки, а затем её вызвать, т.е. разместить ещё скобки, передав в них при необходимости аргументы.

```
// num1 и num2 - параметры самовызывающейся функции
// 7 и 4 - аргументы самовызывающейся функции
(function (num1, num2) {
  console.log(num1 + num2); // 11
})(7, 4);
```

Паттерн «модуль»:

```
const userInfo = (function() {
  // имя пользователя по умолчанию
  let name = 'Аноним';
```

```
  // возвращаем объект, состоящий из 2 функций
  return {
    getName: function () {
      return name;
    },
    setName: function (newName) {
      name = newName;
    },
  };
})();

console.log(userInfo.getName()); // 'Аноним'
console.log(userInfo.setName('Дима')); // 'Дима'
console.log(userInfo.getName()); // 'Дима'

// обратиться напрямую к переменной name нельзя, только через «публичные» методы
console.log(userInfo.name); // undefined
```

Почему не нужно использовать традиционные функции

1. Если вы создаёте функцию традиционным способом, то можете присвоить переменной (имени функции) новое значение:

```
function sum(a, b) {
  console.log(a + b);
}
sum(5, 3); // 8
sum = 7;
console.log(sum); // 7
```

Если функцию мы присвоим переменной, объявленной с помощью `const`, то затем присвоить новое значение этой переменной у нас уже не получится:

```
const sum = (a, b) => {
  console.log(a + b);
}
sum(5, 3); // 8
sum = 7; // Uncaught TypeError: Assignment to constant variable.
```

2. Функции, объявленные традиционным способом, **всплывают**, т.е. их можно использовать до их объявления:

```
sum(4, 3); // 7
function sum(a, b) {
  console.log(a + b);
}
```

При присвоении стрелочной функции или функционального выражения переменной, объявленной с помощью const или let, всплытие не происходит:

```
sum(4, 3); // Uncaught ReferenceError: Cannot access 'sum' before initialization
const sum = (a, b) => {
  console.log(a + b);
}
```