дом.ru movix

Лекция 4. Функции

Что такое функция и как ее объявить?

Функция - это инструмент, необходимый для структурирования больших программ, уменьшения повторений, назначения имён подпрограммам, и изолирование подпрограмм друг от друга.

```
var sum = function(a, b) {
  return a + b;
};
console.log(sum(4, 5));

var sayHello = function() {
      alert("Hello, world!")
};

var empty = function() {};
```

1) ключевое слово function
2) набор параметров (a, b для функции sum), который может быть пустым (функция sayHello)
3) тело в функции в фигурных скобках
4) return - возвращает результат
5) return без выражения возвращает undefined

Объявление функции

1) Function Expression

```
var f = function(параметры) {
  // тело функции
};
```

2) Function Declaration

```
function f(параметры) {
  // тело функции
};
```

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

Объявление функции

1) Function Expression

```
sayHi("Мир");

var sayHi = function(name) {
  alert( "Привет, " + name );
}
```

2) Function Declaration

```
sayHi("Мир");

function sayHi(name) {
  alert( "Привет, " + name );
}
```

Локальные переменные. Области видимости

```
var x = "outside";

var f1 = function() {

    var x = "inside f1";

};

f1();

console.log(x); // "outside"
```

Переменные, созданные в коде функции, существуют локально только в коде функции и создаются заново при ее вызове.

Локальные переменные. Области видимости

```
var count = function() {
        for (var i = 0; i < 3; i++) {
                var j = i * 2;
        }

        alert(i);
        alert(j);
};
```

Блоки **if/else, switch, for, while, do..while** не влияют на область видимости переменных.

Переменные i,j не будут уничтожены по окончании цикла.

i=3, последнее значение i, при нём цикл перестал работать
j=4, последнее значение j, которое вычислил цикл

Внешние переменные. Области видимости

```
var x = "inside f1";

var f2 = function() {
    console.log(x); //  "inside f1"
    x = "inside f2";
    console.log(x); // "inside f2"
};

f2();

console.log(x); // "inside f2"
```

Переменные, объявленные на уровне всего скрипта, называют глобальными переменными.

Внешние переменные. Области видимости

```
var f2 = function() {
    x = "inside f2";
};
f2();

console.log(x); // "inside f2"
```

Неявное объявление глобальных переменных в старом стандарте JavaSctipt

Параметры функции

```
var showMessage = function(message) {
  message = "Hello, world!";
  console.log(message);
};

var message = "Buy, world!";
showMessage(message);
console.log(message);
```

Параметры функции - это обычные переменные, значения которым задаются до вызова функции, а не в ее коде.

Параметры функции. Необязательные аргументы

```
var logParam = function(key, value) {
  if (value === undefined) {
    value = "value is not defined";
  }
  key = key || "defaultKey";
  console.log(key + ": " + value);
};

logParam("a", 4);
logParam("a");
```

Функцию можно вызвать с любым количеством аргументов.

Если параметр не передан при вызове – он считается равным undefined.

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

Возврат значения

```
function isNumber(a) {
  if (typeof a === "number") {
    return true;
  };
  return false;
}

var num = prompt("Введите число");
if (isNumber(num)) {
  alert("Это число");
} else {
  alert("Это не число");
}
```

Для возврата значения используется директива **return**

1) может находиться в любом месте функции. Как только до неё доходит управление – функция завершается и значение передается обратно.
2) может также использоваться без значения, чтобы прекратить выполнение и выйти из функции

Возврат значения

```
var empty = function() {};

var empty1 = function() {
	return;
}

var empty2 = function() {
	return undefined;
}

empty() === empty2(); //верно
empty1() === empty2(); //верно
```

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

Условное объявление функции

```
var num = prompt("Введите число");

var check;

if (num % 2 === 0) {
  check = function() {
    alert("Число четное");
  }
} else {
  check = function() {
    alert("Число нечетное");
  }
}
check();
```

```
var num = prompt("Введите число");

var check = (num % 2 === 0) ?
  function() { alert("Число четное");  } :
  function() { alert("Число нечетное"); };

check();
```

Анонимные функции

```
var parityCheck = function(ask, yes, no) {
	if (prompt(ask) % 2 === 0) {
		yes();
	} else {
		no();
	}
};

parityCheck(
	"Введите число",
	function() { alert("Число четное");  },
	function() { alert("Число нечетное"); }
);
```

Функциональное выражение, которое не записывается в переменную, называют анонимной функцией.

Рекурсия

```
function pow(x, n) {
/* пока n != 1, сводить вычисление
  pow(x, n) к pow(x, n-1) */
        if (n != 1) {
                return x * pow(x, n - 1);
        } else {
                return x;
        }
}

alert(pow(2, 3)); // 8
```

В теле функции могут быть вызваны другие функции для выполнения подзадач.

Частный случай подвызова – рекурсия, т.е. когда функция вызывает сама себя.

pow(2, 4) = 2 * pow(2, 3)
pow(2, 3) = 2 * pow(2, 2)
pow(2, 2) = 2 * pow(2, 1)
pow(2, 1) = 2

Рекурсия

```
function pow(x, n) {
/* пока n != 1, сводить вычисление
  pow(x, n) к pow(x, n-1) */
        if (n != 1) {
                return x * pow(x, n - 1);
        } else {
                return x;
        }
}

alert(pow(2, 3)); // 8
```

Описание примера:
«функция pow рекурсивно вызывает сама себя» до n == 1

Базис рекурсии - значение, на котором рекурсия заканчивается (1 в примере)

Глубиной рекурсии - общее количество вложенных вызовов (3 в примере). Максимальная глубина рекурсии в браузерах ограничена, точно можно рассчитывать на 10000 вложенных вызовов, но некоторые интерпретаторы допускают и больше.

Контекст выполнения, стек

У каждого вызова функции есть свой «контекст выполнения» (execution context).

Контекст выполнения – это служебная информация, которая соответствует текущему запуску функции. Она включает в себя локальные переменные функции и конкретное место в коде, на котором находится интерпретатор.

При любом вложенном вызове JavaScript запоминает текущий контекст выполнения в специальной внутренней структуре данных – «стеке контекстов».

Замыкания

```
function outerFn(myArg) {
  var myVar;
  function innerFn() {
    //имеет доступ к myVar и myArg
  }
}
```

Замыкание – это функция вместе со всеми внешними переменными, которые ей доступны

innerFn - вложенная функция

Переменные myArg и myVar продолжают существовать и остаются доступными внутренней функцией даже после того, как внешняя функция, в которой они определены, была исполнена

Замыкания

```
function createCounter() {
  var numberOfCalls = 0;
  return function() {
    return ++numberOfCalls;
  }
}

var fn = createCounter();

fn(); //1
fn(); //2
fn(); //3
```

Функция, возвращаемая createCounter, использует переменную numberOfCalls, которая сохраняет нужное значение между ее вызовами (вместо того, чтобы сразу прекратить своё существование с возвратом createCounter).

Такие «вложенные» функции в JavaScript называют замыканиями — они «замыкают» на себя переменные и аргументы функции, внутри которой определены.

Замыкания

```
(function() {
  …
})();
```

```
var fn = (function() {
  var numberOfCalls = 0;
  return function() {
    return ++numberOfCalls;
  }
})();
```

Замыкания

```
function multiplier(factor) {
    return function(number) {
        return number * factor;
    };
}

var twice = multiplier(2);
console.log(twice(5)); // 10
```

Замыкания. Очень хитрый пример

```
var arr = [10, 12, 15, 21];

for (var i = 0; i < arr.length; i++) {

  setTimeout(function() {

    console.log('Index: ' + i + ', element: ' + arr[i]);

  }, 3000);

}
```

Что будет выведено в консоль?