ФУНКЦИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ

JavaScript

Модуль 4. Урок 1.

LUXOFT

LXFT LISTED NYSE

ОПРЕДЕЛЕНИЕ ФУНКЦИИ

Функция определяется с помощью ключевого слова, которое может использоваться в выражении определения функции или в операторе объявления функции.

Определение функции начинается с ключевого слова, за которым идут следующие компоненты:

- Идентификатор, который именует функцию
- Список разделенных запятыми параметров (ноль или более) в круглых скобках
- Ноль или более утверждений JavaScript в фигурных скобках

```
function <name>(<param>, <param>, ...) {
    <statements>
}
```

ФУНКЦИЯ КАК ГЛОБАЛЬНАЯ ПЕРЕМЕННАЯ

```
// these definitions are equal
function f() { console.log("i’m function"); }
f = function () { console.log("i’m function 2"); }
```

АРГУМЕНТЫ ФУНКЦИИ И МАССИВ ARGUMENTS

- Внутри любой функции доступен специальный массив arguments.
- Массив arguments позволяет получать значения аргументов, переданных функции, по номеру, а не по имени.

```
function f(x, y, z) {
  // First, verify that the right number of arguments was passed
  if (arguments.length != 3) {
    throw new Error("function f called with " + arguments.length +
    "arguments, but it expects 3 arguments.");
  }
  // Now do the actual function...
}
```

МАССИВ ARGUMENTS

```
function f() {
  console.log(arguments.length);
}
f() // 0
f(1,2,3) // 3

function f(x,y) {
 console.log(x,y);
 console.log(arguments.length);
 for(var i=0;i<arguments.length;i++) {
   console.log(arguments[i]);
 }
}
f(1,3) // 1 3
f(1,3,5,7)  // 1 3 5 7
```

ПРИМЕР: УНИВЕРСАЛЬНАЯ ФУНКЦИЯ SUM()

```
// универсальная функция суммирования
function sum() {
 var total = 0;
 for (var i=0;i<arguments.length;i++) {
  total += arguments[i];
 }
 return total;
}
// то же с использованием reduce
function sum() {
  return [].reduce.call(arguments,(x,y)=>x+y); }
```

ВЫЗОВ ФУНКЦИЙ

Функции JavaScript можно вызвать следующим образом:

- как функции
- как методы
- как конструкторы
- опосредованно через их методы call() и apply()

```
// function invocation expression
printprops({x:1});

// method invocation expression
o.m();

// constructor invocation
var o = new Object();

// indirect indirect
f.call(o);
```

ФУНКЦИИ КАК ДАННЫЕ

- В JavaScript функции — это не только элементы синтаксиса, но и значения.
- Функции могут быть назначены переменным, храниться в свойствах объектов или элементах массивов, передаваться в качестве аргументов другим функциям.

```
function square(x) { return x * x; }
var s = square;      // Now s refers to the same function that square does
square(4);           // => 16
s(4);                // => 16

var o = { square: function(x) { return x * x; }}; // An object literal
var y = o.square(16);                             // y equals 256
```

ПЕРЕДАЧА ФУНКЦИИ КАК ПАРАМЕТРА

```
function print(f) { console.log(f); }
print(1);

function print(f) { console.log(f()); }
print(function () { return 1; } ); // 1
print(()=>1); // 1
```

ФУНКЦИИ КАК МЕТОДЫ

- Метод — это функция, которая включена в свойство объекта и вызывается с помощью этого объекта.
- Ключевое слово this внутри метода указывает на объект, для которого был вызван этот метод.
- Для глобальной функции this указывает на глобальный объект (window).

```
var calculator = {
  operand1: 1,
  operand2: 1,
  compute: function() {
    this.result = this.operand1 +  this.operand2;
  }
};
calculator.compute();
calculator.result;  // => 2
```

МЕТОДЫ И СВОЙСТВА ОБЪЕКТА ФУНКЦИИ

- Функция может определять собственное свойство, которое будет «глобальным» для данной функции.
- Декларации функции обрабатываются до выполнения кода, поэтому назначение делается до объявления функции.

```
// Create and initialize the "static" variable.
// Function declarations are processed before code is executed, so
// we really can do this assignment before the function declaration.
uniqueInteger.counter = 0;

// Here's the function. It returns a different value each time
// it is called and uses a "static" property of itself to keep track
// of the last value it returned.
function uniqueInteger( ) {
  // Increment and return our "static" variable
  return uniqueInteger.counter++;
}
```

МЕТОДЫ И СВОЙСТВА ОБЪЕКТА ФУНКЦИИ

- Методы call() и apply() позволяют опосредованно вызвать функцию как метод какого-то другого объекта.
- Первый аргумент методов call() и apply() — это объект, для которого вызывается функция.
- Метод apply() аналогичен методу call(), за исключением того, что передаваемые функции аргументы определяются как массив.

```
f = function(a,b) { this.sum = a+b; }

f.call(o, 1, 2);
// or
f.apply(o, [1, 2]);

console.log(o); // {sum:3}

// Эти вызовы идентичны такому коду:
o.m= f;
o.m(1,2); // o.sum==3
delete o.m; // удаляем метод из объекта
```

ИСПОЛЬЗОВАНИЕ CALL ДЛЯ ВЫЗОВА ФУНКЦИИ

```
person = {name:"Vasya", age:30}
// создадим универсальную функцию, которую можно применить
// к любому объекту
function printer() {
  for (p in this) console.log(p+"="+this[p]);
}
printer.call(person)
printer.call({a:1})
// мы можем поместить функцию внутрь объекта
person.p = printer
person.p()
delete person.p
person.p()
printer.call({})
// или мы можем привязать (bind) print() к любому объекту
personPrinter = printer.bind(person)
```

МЕТОД BIND() – ПРИВЯЗКА ФУНКЦИИ К ОБЪЕКТУ

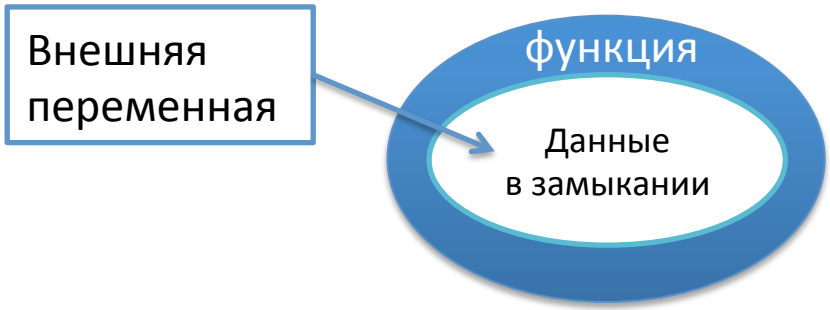
```
function f() {
   console.log(this.name); // this пока не привязан
}
f.call({name:"Vasya"})
fVasya = f.bind({name:"Vasya"})
// привязываем this к объекту {name: “Vasya”}

fVasya() // Vasya


person= {name:"Vasya"};
person.f = f  // тоже привязка
person.f(); // Vasya
delete person.f // удаляем

fVasya = f.bind(person)
fVasya()
```

МЕТОД BIND() – ПЕРЕДАЧА ПАРАМЕТРОВ

```
function f(surname) {
  console.log(this.name+" "+surname); // this пока не привязан
}

fVasya = f.bind({name:"Vasya"} , "Pupkin");
// передаем объект и параметр

fVasya() // Vasya Pupkin

function f(surname, title) {
  console.log(title+" "+this.name+" "+surname);
}
fVasya = f.bind({name:"Vasya"}, "Pupkin")
// привязали this и surname, а вот title остался не привязанным
fVasya("Sir"); // Sir Vasya Pupkin
```

ОБЛАСТЬ ВИДИМОСТИ И CLOSURES (ЗАМЫКАНИЯ)

- В JavaScript используется лексическая сфера действия. Это означает, что функции выполняются с использованием переменной области действия, которая действовала на момент их определения, а не той, которая действует на момент их вызова.
- В JavaScript объект-функция включает в себя не только код функции, но и ссылку на текущую цепочку областей видимости.
- Сочетание объекта-функции и цепочки областей видимости (набора связываний переменной), в которой решаются переменные функции, называется closure (замыкание).
- С технической точки зрения, все функции JavaScript являются closures: они представляют собой объекты и имеют связанную с ними цепочку областей видимости.



ОБЛАСТЬ ВИДИМОСТИ И CLOSURES (ЗАМЫКАНИЯ)

Замыкание глобальной переменной:

```
var x = "global";
function f() {
  function g() { console.log(x); }
  g();
}
f(); // печатает “global”
```

Замыкание локальной переменной:

```
var x = "global";
function f() {
  var x = "local";
  function g() { console.log(x); }
  g();
}
f(); // печатает “local”
```

Замыкание параметра:

```
function f(x) {
  function g() { console.log(x); }
  g();
}
f("param"); // печатает “param”
```

ЗАМЫКАНИЯ

```
var x = "global";
function f() {
 var x = "local";
 return function() { console.log(x); }
}
f() // f () { console.log(x); }
f()() // local
```

```
function f(x) {
 return function() { console.log(x); }
}
f("hello")
f("hello")() // hello

g = f("hello")
g() // hello
```

СОЗДАНИЕ ИНКАПСУЛИРОВАННОГО ЗНАЧЕНИЯ ЧЕРЕЗ ЗАМЫКАНИЕ

```
// getFullName keeps name and surname in closure
getPerson = function(name, surname) {
 return {
  getFullName: function() {
   return name+" "+surname;
  }
 }
}
person = getPerson("Petr", "Petrov")
person.getFullName()
person.name = "sdsd"
person.getFullName()
getPerson("Petr", "Petrov").getFullName()
```

ОБЛАСТЬ ВИДИМОСТИ И CLOSURES (ЗАМЫКАНИЯ)

```
function counter() {
  var n = 0;
  return {
    count: function() { return n++; },
    reset: function() { n = 0; }
    };
}

var c = counter(), d = counter();    // Create two counters
c.count()                            // => 0
d.count()                            // => 0: they count independently
c.reset()                            // reset() and count() methods share state
c.count()                            // => 0: because we reset c
d.count()                            // => 1: d was not reset
```

КОНСТРУКТОР FUNCTION()

- Функции могут быть определены с помощью конструктора Function()

```
var f = new Function("x", "y", "return x*y;");
// equivalent to a function definition
var f = function(x, y) { return x*y; };
```

- Последний аргумент — текст тела функции: он содержит произвольные утверждения JavaScript, разделяемые точкой с запятой.
- Все остальные аргументы для конструктора — это строки, указываются имена параметров для функции.

УНИВЕРСАЛЬНЫЙ КАЛЬКУЛЯТОР

```
var f = new Function("x","y","return x"+op+"y;")
function getCalculator(op) {
  return new Function("x","y","return x"+op+"y;");
}
summator = getCalculator("+");
summator(1,4)
minusator = getCalculator("-");
minusator(6,1)
```

СТРЕЛОЧНАЯ ФУНКЦИЯ

```
f = v => v + 1;
```

```
// аналог в обычном варианте
var f = function (v) { return v + 1; })
```

Пример использования:

```
var arr = [1,2,3];
arr.forEach(i=>console.log(i));
```

СТРЕЛОЧНАЯ ФУНКЦИЯ С НЕСКОЛЬКИМИ ПАРАМЕТРАМИ

```
f = (x,y) => x+y;
f(1,2) === 3;
```

ФУНКЦИЯ СТРЕЛКИ С ТЕЛОМ ФУНКЦИИ

```
f = (x,y) => {
  console.log(x,y);
  return x+y;
}
```

ФУНКЦИИ МАССИВОВ: MAP/FILTER/REDUCE

```
// array: map/filter/reduce
let arr = [1,2,3];
arr.find(x=>x>2);
arr.findIndex(x=>x>2);

let a = ["first","second","third"];
a.filter(x=>x.length>5);
a.map(x=>x.length); // [5,6,5]
a.map(x=>x.length).reduce((x,y)=>x+y); // 16
a.map(x=>x.length).reduce((x,y)=>x+y)/a.length; // avg length
a.forEach(x=>console.log(x.length)); // 5 6 5
```

ПРИМЕР ПРИМЕНЕНИЯ MAP/FILTER/REDUCE

```
// find persons elder 30 and print their names
persons = [{name:"Ivan",age:25}, {name:"Mary",age:35},
 {name:"Stas", age:33}];
persons.filter(p=>p.age>30)
  .map(p=>p.name)
  .forEach(n=>console.log(n)); // Mary Stas

// print average age of persons
persons.map(p=>p.age).reduce((a,b)=>a+b)/persons.length // 31
```