JS

Functions

Exadel®

Функции

- функция - это объект (присваивать их переменным, добавлять собственные свойства и методы, возвращать из других функций, передавать в качестве аргументов)
- выполняется в рамках объекта (this); если объект задан не явно, то используется глобальный
- вызвать функцию в контексте дрдугого объекта можно с помощью методов call, apply.
- может быть анонимной
- выступают в качестве конструкторов для объектов
- позволяет создать локальную область видимости
- могут принимать любое количество аргументов (arguments)

Объявление функций

```
// funtion declaration
function sum (arg1, arg2){
    return arg1 + arg2;
}
```

```
// function expression
var mult = function(arg1, arg2){
    return arg1 * arg2;
}
```

Объявление функций

```
sum(1,2);  // => 3
mult(3,4); // => ERROR: undefined is not a function


// funtion declaration
function sum (arg1, arg2){
    return arg1 + arg2;
}

// function expression
var mult = function(arg1, arg2){
    return arg1 * arg2;
}
```

Scope функции

- каждый вызов функции создает свой scope, куда попадут аргументы функции и объявленные в ней локальные переменные
- каждая вложенная функция имеет доступ к scope «родительской» (scope chain), «родительская» же функция доступа к scope вложенной не имеет

  доступ из одного scope в другой осуществляется через свойство [[scope]]
- из любой функции доступны переменные из global scope

! не путать с контекстом выполнения (execution context)

Scope функции

```
var orderSum = 3.50;

function incrementOrderSum (increment){
    orderSum += increment;
}
```

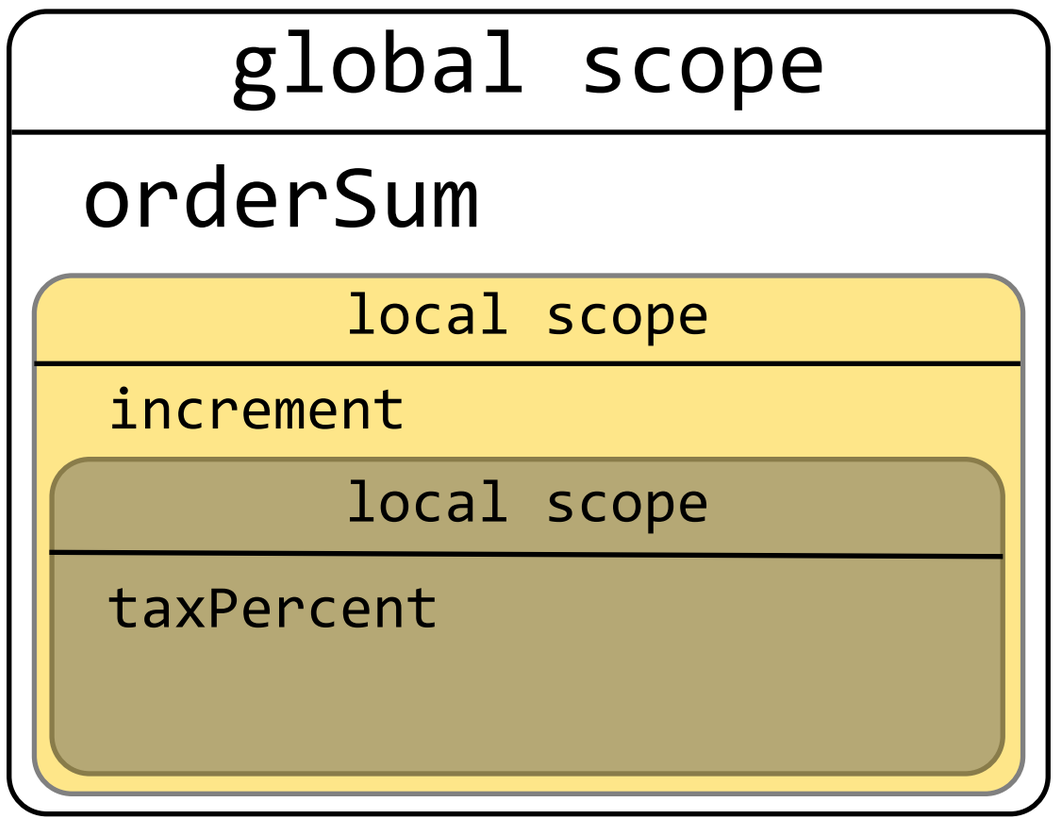
Scope функции

```
var orderSum = 3.50;

function incrementOrderSum (increment){
    orderSum += increment;

    function addTaxes(taxPercent){
        orderSum *= (1 + taxPercent);
    }
}
```

incrementOderSum( )



addTaxes( )

Scope функции

```
function sum (x){

    return function(y){
        return x + y;
    }
}

var sum5 = sum(5);

sum5(5);  // => 10
sum5(10); // => 15
sum5(15); // => 20
```

Scope функции

```
(function (){

    function test(){
        for(i = 0; i < 10; i++){
            console.log(i);
        }
    }

    for(var i = 0; i < 10; i++){
            test();
    }

})();

// => ?
```

Scope функции

```
(function (){

  var lis = document.getElementsByTagName(“li”);

  for(var i = 0; i < li.length; i++){
    lis[i].addEventListener(“click”, function(){
      alert(i);
    });
  }
})();

// => ?
```

Context функции

- контекст, в котором вызывается функция это то, на что указывает this внутри нее
- контекст может меняться при вызове, для этого можно использовать call, apply или просто вызвать функцию в ином контексте
- есть возможность связать функцию с определенным контекстом

Изменение контекста функции

```
var test = “global value”,
    obj  = {
      test      : “local value”,
      showTest : function(){

        return function(){
          return this.test;
        }
      }
    };


obj.showTest()();                                    // => ?
obj.showTest().call({test:“another value”}); // => ?
```

Удержание контекста функции

```
var man = {
      name        : “Остап Ибрагимович”,
      printName : function(){
        console.log(this.name);
      }
    };
```

```
document.body.addEventListener(“click”, man.printName);
// => ?
```

Удержание контекста функции

```
var man = {
      name        : “Остап Ибрагимович”,
      printName : function(){
        console.log(this.name);
      }
    };
```

```
document.body.addEventListener(“body”, function(){
    man.printName();
});
```

Удержание контекста функции

```
var man = {
      name        : “Остап Ибрагимович”,
      printName : function(){
        console.log(this.name);
      }
    };
```

```
man.printName = man.printName.bind(man);

document.body.addEventListener(“body”, man.printName);
```

Context функции

```
var a    = {},
    b    = {},
    func = function(){ console.log(this); };
```

```
func();                    // => ?
func.call(a);              // => ?
func.apply(b);             // => ?
func.bind(a).call(b);      // => ?
```