

# Оператор IN

in – это специальный оператор для работы с объектами. С помощью него можно как проверить наличие свойства в объекте, так и перебрать все свойства объекта!

```
var object = {  
    property : 'value',  
    color : 'red',  
    'this is property too' : 'some value',  
    2 : 42  
}
```

Проверяем наличие свойства в объекте

```
if ( propertyName in object){  
    alert('property '+propertyName+' was found');  
}
```

Перебираем все свойства в объекте

```
for ( key in object){  
    console.log(key + ':' + object [ key ] );  
}
```

Всегда используйте **IN** вместо  
проверки на **undefined**



# Объекты в JavaScript

ОБЪЕКТЫ КАК АССОЦИАТИВНЫЕ МАССИВЫ

# Объекты как ассоциативные массивы

Ассоциативный массив – структура данных, в которой можно хранить любые данные в формате ключ-значение

Фигурные скобки –  
создают объект

Разделитель ключом  
и значением

Значение свойства

Разделитель между  
свойствами

Ключ, имя свойства  
объекта

- Свойством объекта может быть строка, либо число
- Значением свойства может выступать любой тип данных
- Все свойства должны быть разделены запятыми

Ключи объекта  
всегда приводятся к строке

# Объекты как ассоциативные массивы

Для получения доступа к объекту используется следующий синтаксис

```
Object . property
```

Стандартный способ обращения к свойству

```
Object [ property ]
```

Данный способ позволяет получить доступ к свойствам, имена которых содержат спец. символы, числа, либо динамически

Свойства объекта также могут быть созданы «налету». После того как объект был создан, к нему можно добавить новые свойства

```
Object.property = value;
```

Для удаления свойства из объекта воспользуйтесь оператором **delete**

```
delete object.property
```



# Объекты в JavaScript

КОПИРОВАНИЕ И КЛОНИРОВАНИЕ

# Копирование и клонирование

```
var message = 'Hello people!';  
var hello = message;  
  
message = 'new message';  
  
console.log(message);  
console.log(hello);
```

Копирование (передача)  
по значению



Копирование (передача)  
по ссылке

```
var message = {  
  title: 'hello',  
  body: 'hello people'  
};  
var hello = message;  
hello.title = 'new Title';  
  
console.log(message);  
console.log(hello);
```





# Копирование и клонирование

Для того чтоб создать полноценную и независимую копию объекта, необходимо прибегнуть к хитрости, и, по сути, скопировать примитивы

```
var message = {  
  title: 'hello',  
  body: 'hello people'  
};  
var hello = {};
```

Создаем объекты

```
for(key in message){  
  hello[key] = message[key];  
}
```

Создаем свойства  
и копируем в них  
значения

```
hello.title = 'new title';
```

Изменяем нужные  
нам свойства

```
console.log(message);  
console.log(hello);
```

Если же свойства объектов, в свою очередь, могут хранить ссылки на другие объекты, то нужно обойти такие подобъекты и тоже клонировать их. Это называют «глубоким» клонированием.



# Объекты в JavaScript

МЕТОДЫ ОБЪЕКТА



# Методы объекта

Объект также можно воспринимать как некую самодостаточную **сущность**.

Метод объекта, это по сути функция. Для создания метода, необходимо объявить свойство и присвоить в него функцию

```
var user = {  
  name : 'Alex',  
  age : 28  
  
  getAge : function(){  
    return user.age;  
  }  
}
```



user . getAge()



user [ "getAge" ] ()

! Методы, так же как и свойства, могут быть добавлены или удалены в любой момент времени !

# Методы объекта. Еще раз про контекст

Для полноценной работы метод должен **иметь доступ** к **данным объекта**

Контекст описывает в какой «области» используется данная функция, и следовательно какие ресурсы ей доступны

```
var user = {  
  name : 'Alex',  
  age : 28  
  
  getAge : function(){  
    return this . age;  
  }  
}
```

**this** всегда указывает  
на текущий объект

**this** всегда есть у  
любой функции

# Методы объекта. Еще раз про контекст

Любая функция может иметь в себе **this**. Совершенно неважно, объявлена ли она в объекте или отдельно от него.

Значение **this** называется **контекстом** вызова и будет **определено в момент вызова функции**.

```
function getAge(){  
  return this.age;  
}
```

```
var user = { firstName: "Вася", age: 20 };  
var admin = { firstName: "Админ", age: 15};
```

```
user.f = getAge;  
admin.d = getAge;
```

```
user.f();  
admin.d();
```

# Методы объекта. toString и valueOf

Так как JavaScript не строго типизированный язык, в нем существует понятие «приведение типа». С приведением примитивов вы уже знакомы, теперь стоит обратить внимание на объекты.

**toString** – метод который вызывается при попытке работать с объектом, как со строкой  
**valueOf** – метод который вызывается при попытке работать с объектом как с числом, если этот метод не найден, будет вызван метод **toString**

```
var user = {  
  name : 'Alex',  
  age : 28  
  toString: function(){  
    return this.name;  
  },  
  valueOf: function(){  
    return this.age;  
  }  
}
```

Все объекты, включая встроенные,  
имеют свои реализации  
метода toString



# Объекты в JavaScript

ФУНКЦИЯ КОНСТРУТОР

# Функция конструктор

Обычный синтаксис {...} позволяет создать один объект. Но зачастую нужно создать много однотипных объектов.

Для этого используют «функции-конструкторы», запуская их при помощи специального оператора `new`.

```
function Book(title, author, price){  
  this.title = title;  
  this.author = author;  
  this.price = price;  
  
  this.getPrice = function (){  
    return this.price + '$';  
  };  
}
```

```
var boor1 = new Book('JS', 'Alex', 125);  
var boor2 = new Book('PHP', 'Alex', 250);
```

```
book1.getPrice();  
book2.getPrice();
```



# Функция конструктор

В функции-конструкторе бывает удобно объявить вспомогательные локальные переменные и вложенные функции, которые будут видны только внутри:

```
function Book(title, author, price){  
  var currency = '$';  
  function getLink()  
    return 'Книга:' + title + '. Автор: ' + author;  
};  
this.showLink(){  
  console.log(getLink());  
};  
this.getPrice = function ()  
  return this.price + currency;  
};  
}
```

```
var boor1 = new Book('JS', 'Alex', 125);  
var boor2 = new Book('PHP', 'Alex', 250);  
  
book1.showLink();  
book2.showLink();  
  
book1.getPrice();  
book2.getPrice();
```



# Функция конструктор

Довольно часто необходимо обратиться из локального метода к самому объекту. Но так как контекст разный, **this** внутри локальной функции не указывает на нужный нам объект...

```
function Book(title, author, price){  
  function getLink() {  
    return 'Книга:' + this.title + '. Автор: ' + this.author;  
  };  
}
```

! Вот такой код приведет к ошибке! !

# Функция конструктор

Одним из способов решить данную проблему, является сохранение контекста в переменную, для использования в замыкании

```
function Book(title, author, price){  
    var self = this;  
    function getLink() {  
        return 'Книга:' + self.title + '. Автор: ' + self.author;  
    };  
}
```

# Функция конструктор

Методы и свойства, которые не привязаны к конкретному экземпляру объекта, называют «статическими». Их записывают прямо в саму функцию-конструктор

```
function Book(title, author, price){  
  Book.type = 'soft';  
  Book.getType = function(){  
    return this.type;  
  }  
}
```

Константы, яркий пример  
использования статических  
свойств

# Функция конструктор

Фабричный статический метод – это статический метод, который служит для создания новых объектов, различными способами.

```
function User(){  
    this.sayHello = function(){  
        console.log('hello '+this.name);  
    }  
}
```

```
User.createGuest = function(){  
    var user = new User();  
    user.name = 'guest';  
    return user;  
}
```

```
User.createFromData = function(data){  
    var user = new User();  
    user.name = data.name;  
    return user;  
}
```