Object

AIT
Hi-tech team

НАШИ ПРАВИЛА

Включенная камера

Вопросы по поднятой руке

Не перебиваем друг друга

Все вопросы, не связанные с тематикой курса (орг-вопросы и т. д.), должны быть направлены куратору

Подготовьте свое рабочее окружение для возможной демонстрации экрана (закройте лишние соцсети и прочие приложения)

ЦЕЛЬ

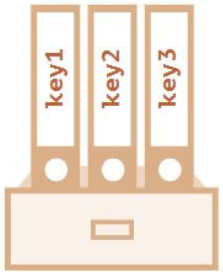
Научиться работать с объектом, узнать соответствующий синтаксис

ПЛАН ЗАНЯТИЯ

- Создание объекта
- Работа с ключами
- Деструктуризация

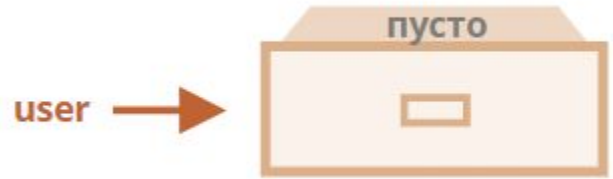
Объекты используются для хранения коллекций различных значений и более сложных сущностей.
В JavaScript объекты используются очень часто, это одна из основ языка.

Объект может быть создан с помощью фигурных скобок {…} с необязательным списком свойств.
Свойство – это пара «**ключ: значение**», где **ключ** – это строка (также называемая «именем свойства»), а **значение** может быть чем угодно.



Пустой объект можно создать, используя один из двух вариантов синтаксиса:

```
1  let user = new Object(); // синтаксис "конструктор объекта"
2  let user = {};  // синтаксис "литерал объекта"
```



При использовании литерального синтаксиса {...} мы сразу можем поместить в объект несколько свойств в виде пар «ключ: значение»:

```
let user = {     // объект
  name: "John",  // под ключом "name" хранится значение "John"
  age: 30        // под ключом "age" хранится значение 30
};
```

Если в объекте несколько свойств, то они перечисляются через запятую.
В объекте user сейчас находятся два свойства:

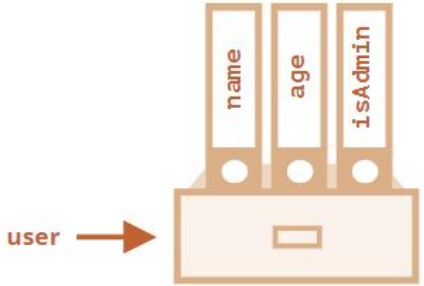
1. Первое свойство с именем "name" и значением "John".
2. Второе свойство с именем "age" и значением 30.

Для обращения к свойствам используется запись «через точку»:

```
1  // получаем свойства объекта:
2  alert( user.name ); // John
3  alert( user.age ); // 30
```

Значение может быть любого типа. Давайте добавим свойство с логическим значением:

```
1  user.isAdmin = true;
```



Имя свойства может состоять из нескольких слов, но тогда оно должно быть заключено в кавычки:

```
let user = {
  name: "John",
  age: 30,
  "likes birds": true  // имя свойства из нескольких слов должно быть в кавычках
};
```

Для свойств, имена которых состоят из нескольких слов, доступ к значению «через точку» не работает:

```
// это вызовет синтаксическую ошибку
user.likes birds = true
```

Для таких случаев существует альтернативный способ доступа к свойствам через квадратные скобки (**брекет-синтаксис**). Такой способ сработает с любым именем свойства:

```
let user = {};

// присваивание значения свойству
user["likes birds"] = true;

// получение значения свойства
alert(user["likes birds"]); // true

// удаление свойства
delete user["likes birds"];
```

Удаление свойств

Удалили из объекта свойство

```
const person = {
 name: 'John',
 age: 25,
 city: 'Example City'
};

delete person.age; // Удаление свойства 'age'
```

Проверка наличия свойств

Проверили наличие свойства при помощи оператора **in**

```
const person = {
 name: 'John',
 age: 25,
 city: 'Example City'
};

const hasAge = 'age' in person; // true
const hasGender = 'gender' in person; // false
```

Перебор ключей

Перебор ключей при помощи цикла

```
const person = {
 name: 'John',
 age: 25,
 city: 'Example City'
};

// Перебор ключей
for (let key in person) {
 console.log(key); // 'name', 'age', 'city'
}
```

Перебор значений

Перебор значений при помощи цикла.

```
const person = {
 name: 'John',
 age: 25,
 city: 'Example City'
};

// Перебор значений
for (let key in person) {
 console.log(person[key]); // 'John', 25,
'Example City'
}
```

Получение значений/ключей

Получили массив ключей и массив значений

```
const person = {
 name: 'John',
 age: 25,
 city: 'Example City'
};

const keys = Object.keys(person); // ['name', 'age', 'city']
const values = Object.values(person); // ['John', 25, 'Example City']
```

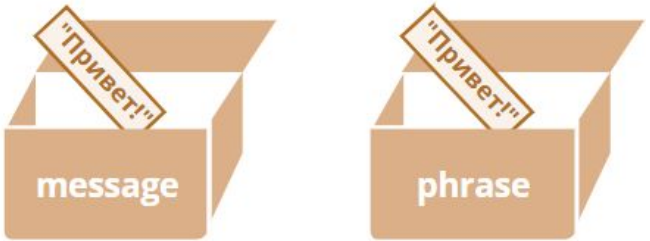
Reference type

```
let person = {
    firstName: 'John',
    lastName: 'Doe'
};
```

Одно из фундаментальных отличий объектов от примитивов заключается в том, что объекты хранятся и копируются «по ссылке», тогда как примитивные значения: строки, числа, логические значения и т.д. – всегда копируются «как целое значение».

```
1 let message = "Привет!";
2 let phrase = message;
```

В результате мы имеем две независимые переменные, каждая из которых хранит строку "Привет!".

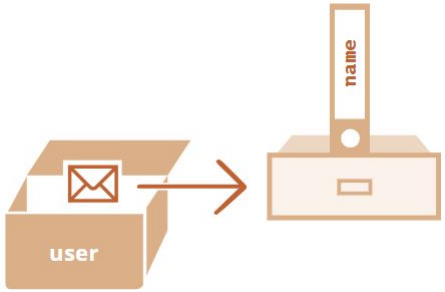


Объекты ведут себя иначе, чем примитивные типы.

Переменная, которой присвоен объект, хранит не сам объект, а его «адрес в памяти» – другими словами, «ссылку» на него.

```
let user = {
  name: "John"
};
```

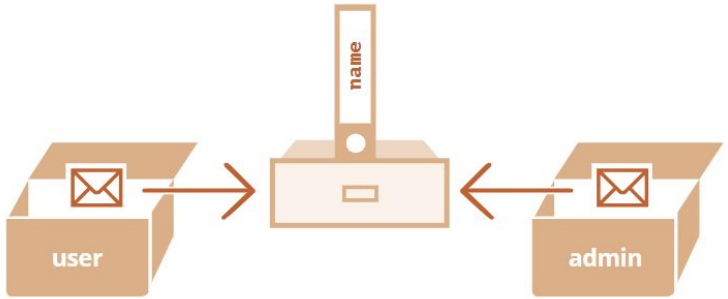
Вот как это на самом деле хранится в памяти:



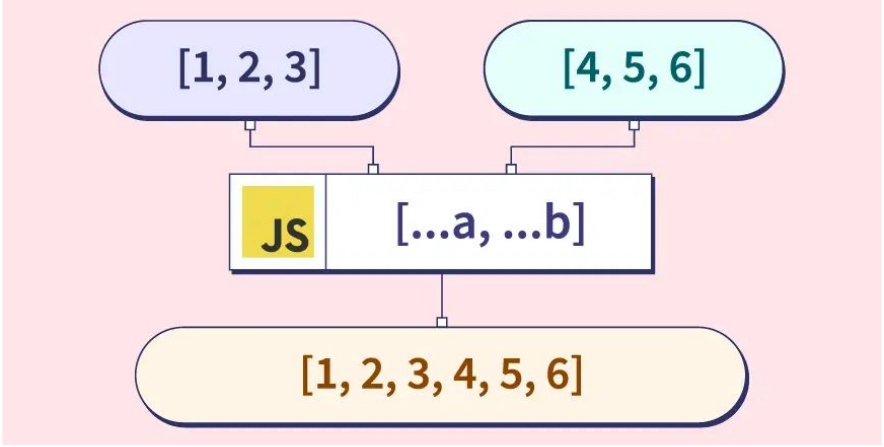
При копировании переменной объекта копируется ссылка, но сам объект не дублируется.

```
let user = { name: "John" };

let admin = user; // копируется ссылка
```

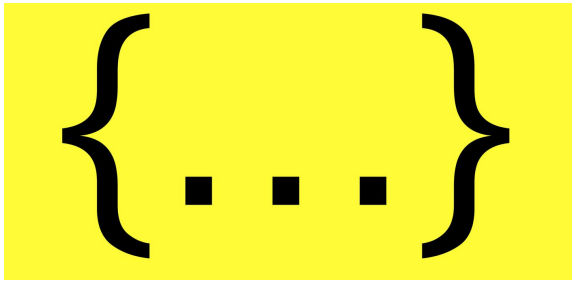


Деструктуризация



Оператор расширения - spread

В JavaScript оператор расширения (spread operator) представляет собой синтаксическую конструкцию, которая используется для создания копий массивов, объединения массивов, передачи аргументов функции и других подобных задач.



1. **Для массивов:**

В приведенном примере ...arr1 разворачивает элементы массива arr1 в новом массиве arr2.

```
const arr1 = [1, 2, 3];
const arr2 = [...arr1, 4, 5, 6];
console.log(arr2); // [1, 2, 3, 4, 5, 6]
```

2. Для объектов:

Здесь spread оператор используется для создания нового объекта obj2, который содержит все свойства из obj1, а также новые свойства.

```
const obj1 = { key1: 'value1', key2: 'value2' };
const obj2 = { ...obj1, key3: 'value3', key4: 'value4' };
console.log(obj2);
// { key1: 'value1', key2: 'value2', key3: 'value3', key4: 'value4' }
```

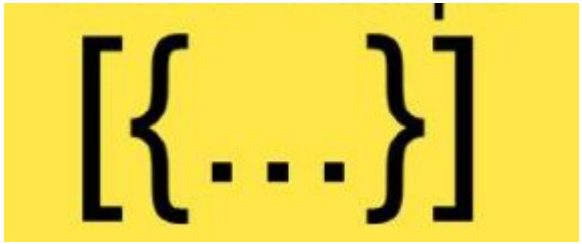
3. Передача аргументов функции

В данном примере spread оператор используется для передачи элементов массива в качестве аргументов функции sum.

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

const numbers = [1, 2, 3];
console.log(sum(...numbers)); // 6
```

Деструктурирующее присваивание



Деструктурирующее присваивание – это специальный синтаксис, который позволяет нам «распаковать» массивы или объекты в несколько переменных, так как иногда они более удобны.

Деструктуризация массива

```
1  // у нас есть массив с именем и фамилией
2  let arr = ["Ilya", "Kantor"];
3
4  // деструктурирующее присваивание
5  // записывает firstName = arr[0]
6  // и surname = arr[1]
7  let [firstName, surname] = arr;
8
9  alert(firstName); // Ilya
10 alert(surname);  // Kantor
```

Деструктурирующее присваивание ничего не уничтожает, его задача – только скопировать нужные значения в переменные.

Деструктуризация объекта

Деструктурирующее присваивание также работает с объектами.

```
let options = {
  title: "Menu",
  width: 100,
  height: 200
};

let {title, width, height} = options;

alert(title);  // Menu
alert(width);  // 100
alert(height); // 200
```

Для явного задания имени переменной используем синтаксис:

название ключа: имя переменной

```
let {title: optionName, width, height} = options;
```

Ваша новая IT-профессия – Ваш новый уровень жизни

Программирование с нуля в
немецкой школе AIT TR GmbH

AIT
Hi-tech team