# Объекты в JavaScript, их свойства и методы

# Пример

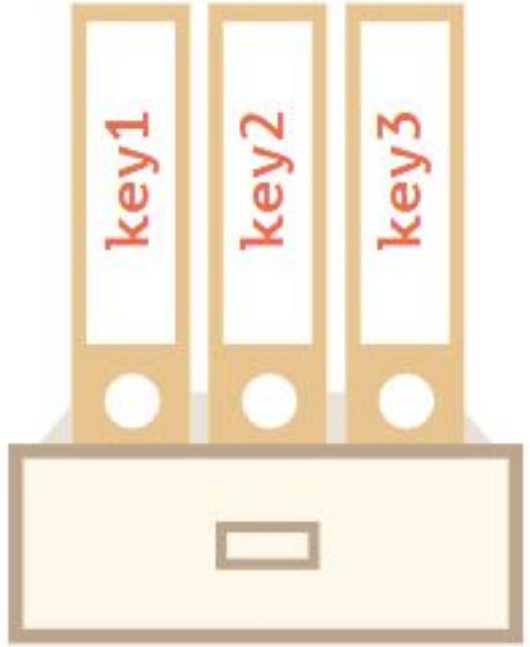
```
let arr = ['пн', 'вт', 'ср', 'чт', 'пт', 'сб',
  'вс'];
```

- **Объекты** создаются с помощью фигурных скобок { }, внутри которых пишутся элементы этого объекта в формате **ключ: значение.**

```
let obj = {1: 'пн', 2: 'вт', 3: 'ср', 4: 'чт',
  5: 'пт', 6: 'сб', 7: 'вс'};
console.log(obj[1]); // выведет 'пн'
```

# Объекты

- **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.

# Создание объекта

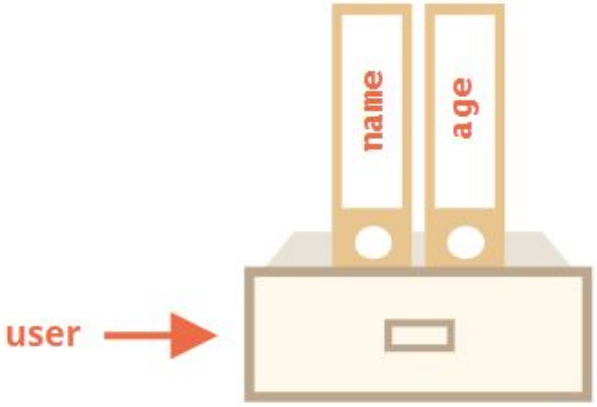
```
let user = new Object(); // синтаксис "конструктор объекта"
let user = {};  // синтаксис "литерал объекта"
```

- Обычно используют вариант с фигурными скобками {...}. Такое объявление называют литералом объекта или литеральной нотацией.


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

Для обращения к свойствам используется запись «через точку»

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

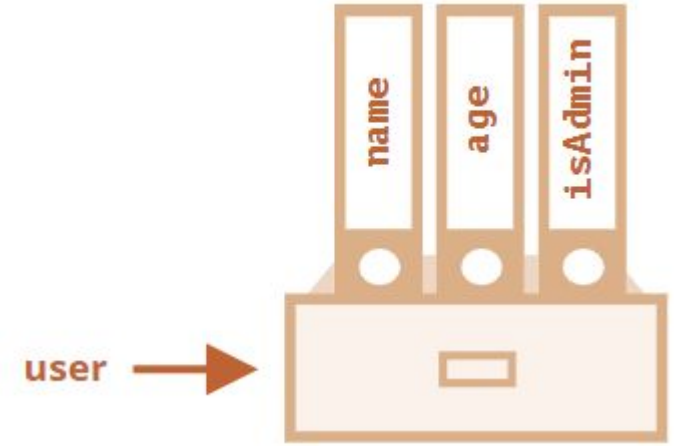


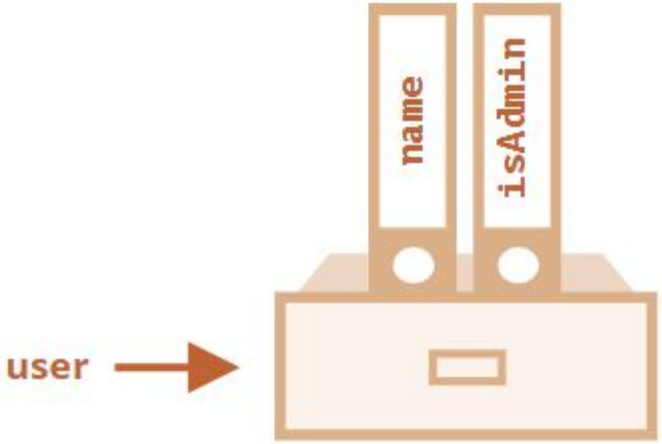
# Delete

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

Для удаления свойства мы можем использовать оператор **delete:**

```
delete user.age;
```



# Имя свойства

- 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:

```
let user = {};

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

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

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

# Объект, объявленный как константа, может быть изменён

- Дело в том, что объявление const защищает от изменений только саму переменную user, а не её содержимое.
- Определение const выдаст ошибку только если мы присвоим переменной другое значение: user=....

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

user.name = "Pete"; // (*)

alert(user.name); // Pete
```

# Пример

```
let user = {
  name: "John",
  age: 30
};

let key = prompt("Что вы хотите узнать о пользователе?", "name");

// доступ к свойству через переменную
alert( user[key] ); // John (если ввели "name")
```

```
let user = {
  name: "John",
  age: 30
};

let key = "name";
alert( user.key ); // undefined
```

# Пример

- Мы можем использовать квадратные скобки в литеральной нотации для создания вычисляемого свойства.

```
let fruit = prompt("Какой фрукт купить?", "apple");

let bag = {
  [fruit]: 5, // имя свойства будет взято из переменной fruit
};

alert( bag.apple ); // 5, если fruit="apple"
```

# Свойство из переменной

```
function makeUser(name, age) {
  return {
    name: name,
    age: age
    // ...другие свойства
  };
}

let user = makeUser("John", 30);
alert(user.name); // John
```

# Ограничения на имена свойств

- нет никаких ограничений к именам свойств. Они могут быть в виде строк или символов

```
// эти имена свойств допустимы
let obj = {
  for: 1,
  let: 2,
  return: 3
};

alert( obj.for + obj.let + obj.return );  // 6
```

```
let obj = {
  0: "Тест" // то же самое что и "0": "Тест"
};

// обе функции alert выведут одно и то же свойство (число 0 преобразуется в строку "0")
alert( obj["0"] ); // Тест
alert( obj[0] ); // Тест (то же свойство)
```

# Проверка существования свойства, оператор «in»

- Синтаксис оператора:

```
"key" in object
```

```
let user = { name: "John", age: 30 };

alert( "age" in user ); // true, user.age существует
alert( "blabla" in user ); // false, user.blabla не существует
```

# Цикл «for…in»

- Для перебора всех свойств объекта используется цикл for..in.
Этот цикл отличается от изученного ранее цикла for(;;).

```
for (key in object) {
  // тело цикла выполняется для каждого с
}
```

выведем все свойства объекта user:

```
let user = {
  name: "John",
  age: 30,
  isAdmin: true
};

for (let key in user) {
  // ключи
  alert( key );  // name, age, isAdmin
  // значения ключей
  alert( user[key] ); // John, 30, true
}
```

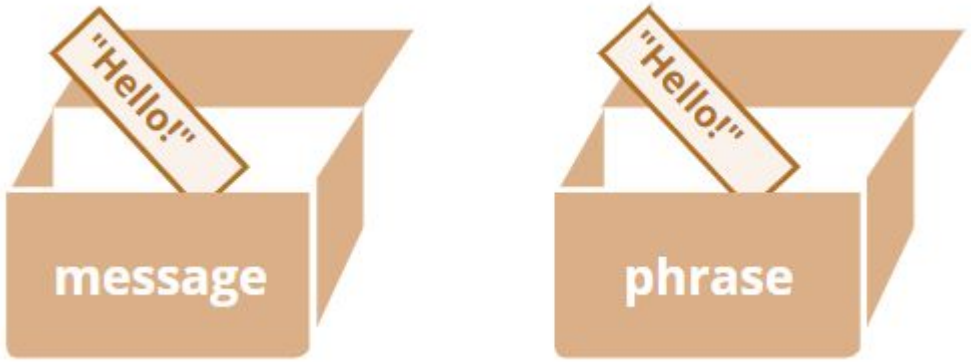
# Копирование объектов

- Одним из фундаментальных отличий объектов от примитивных типов данных является то, что они хранятся и копируются «по ссылке».

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

Объекты ведут себя иначе.

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

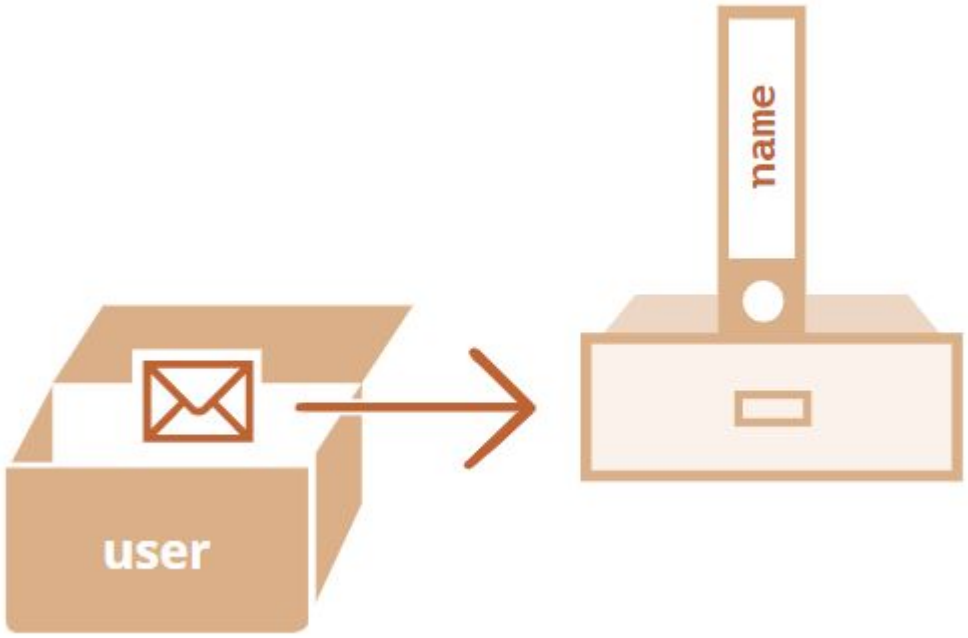


# Копирование обьектов

- **Переменная хранит не сам объект, а его «адрес в памяти», другими словами «ссылку» на него.**

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

Сам объект хранится где-то в памяти. А в переменной user лежит «ссылка» на эту область памяти.



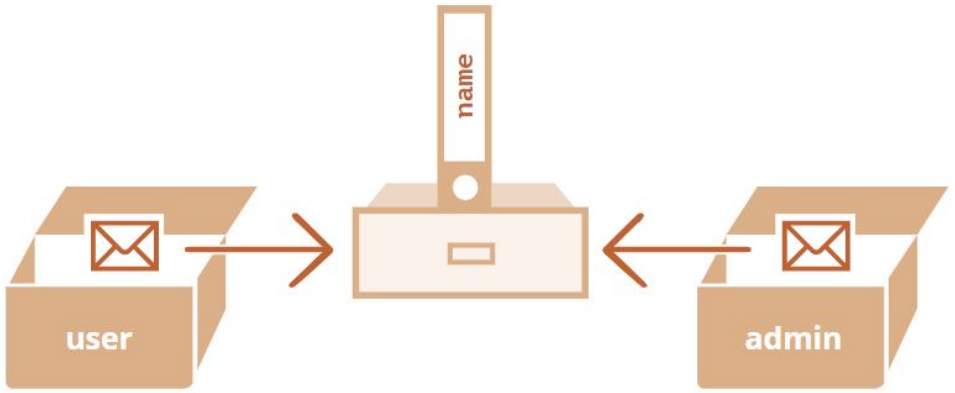
# Копирование обьектов

- **Когда переменная объекта копируется – копируется ссылка, сам же объект не дубл**

```
let user = { name: "Иван" };

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

Теперь у нас есть две переменные, каждая из которых содержит ссылку на один и тот же объект:



# Пример

```
let user = { name: 'Иван' };

let admin = user;

admin.name = 'Петя'; // изменено по ссылке из переменной "admin"

alert(user.name); // 'Петя', изменения видны по ссылке из переменной "user"
```

# Клонирование и объединение объектов

```
let user = {
  name: "Иван",
  age: 30
};

let clone = {}; // новый пустой объект

// скопируем все свойства user в него
for (let key in user) {
  clone[key] = user[key];
}

// теперь в переменной clone находится абсолютно независимый клон объекта
clone.name = "Пётр"; // изменим в нём данные

alert( user.name ); // в оригинальном объекте значение свойства `name` осталось прежним – Иван.
```

# Object.assign

```
Object.assign(dest, [src1, src2, src3...])
```

```
let user = { name: "Иван" };

let permissions1 = { canView: true };
let permissions2 = { canEdit: true };

// копируем все свойства из permissions1 и permissions2 в user
Object.assign(user, permissions1, permissions2);

// теперь user = { name: "Иван", canView: true, canEdit: true }
```

```
let user = { name: "Иван" };

Object.assign(user, { name: "Пётр" });

alert(user.name); // теперь user = { name: "Пётр" }
```

Если принимающий объект (user) уже имеет свойство с таким именем, оно будет перезаписано

# Ключевое слово «this» в методах

- Значение this – это объект «перед точкой», который использовался для вызова метода.

```
let user = {
  name: "Джон",
  age: 30,

  sayHi() {
    // this - это "текущий объект"
    alert(this.name);
  }

};

user.sayHi(); // Джон
```

во время выполнения кода user.sayHi() значением this будет являться user (ссылка на объект user).

```
let user = { name: "Джон" };
let admin = { name: "Админ" };

function sayHi() {
  alert( this.name );
}
// используем одну и ту же функцию в двух объектах
user.f = sayHi;
admin.f = sayHi;

// вызовы функции, приведённые ниже, имеют разное значение this
// "this" внутри функции является ссылкой на объект, который указан "перед точкой"
user.f(); // Джон  (this == user)
admin.f(); // Админ  (this == admin)

admin['f'](); // Админ (неважен способ доступа к методу - через точку или квадратные скобки)
```

# Типизация массивов и объектов в JavaScript

- посмотрим, что выведет оператор typeof для объекта и массива

```
console.log(typeof {}); // выведет 'object'
```

```
console.log(typeof []); // тоже выведет 'object'
```

ело в том, что на самом деле в JavaScript нет отдельного типа данных для массивов - каждый массив представляет собой частный случай объекта.

# Задача

- определите, что выведется на экран в консоль:

- console.log( typeof {a: 1, b: 2, c: 3} );

# Задача

- Не запуская код, определите, что выведется на экран в консоль:

- console.log( typeof [1, 2, 3] );

- Не запуская код, определите, что выведется на экран в консоль:

- let arr = [1, 2, 3];
- console.log( typeof arr );

# Работа с объектом Date в JavaScript

- Объект с датой создается :
- запишем созданный объект в переменную

```
new Date();
```

```
let date = new Date();
```

```
let date = new Date();

console.log(date.getFullYear()); // год
console.log(date.getMonth());    // месяц
console.log(date.getDate());     // день

console.log(date.getHours());    // часы
console.log(date.getMinutes());  // минуты
console.log(date.getSeconds());  // секунды
```