ОБЪЕКТЫ

ТИПЫ ДАННЫХ:

- Примитивные типы
- Сложные типы

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

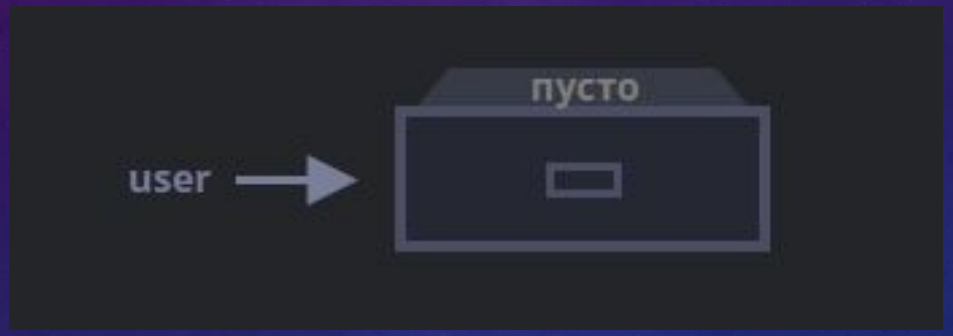
Объект может быть создан с помощью фигурных скобок { }

с необязательным списком *свойств*. Свойство – это пара «ключ: значение», где:

- ключ – это строка (также называемая «именем свойства»)
- значение – может быть чем угодно

Мы можем представить объект в виде ящика с подписанными папками. Каждый элемент данных хранится в своей папке, на которой написан ключ. По ключу папку легко найти, удалить или добавить в неё что-либо

```
let user = new Object(); // синтаксис «конструктор объекта»
```




```
let user = {}; // синтаксис "литерал объекта"
```

Задача:

Создать любой осмысленный объект с 5 осмысленными ключами и свойствами.

Вывести на экран попарно ключ и его значение

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

Инициализируем объект user с некоторыми ключами и значениями:

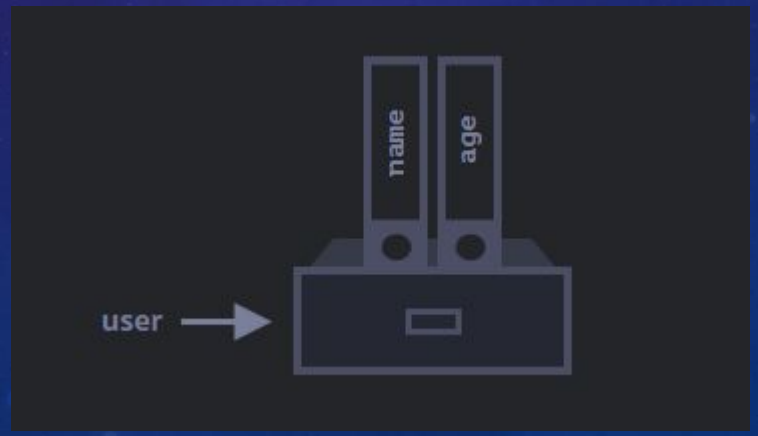
```
// объект
    let user = {
        name: "John",  // под ключом "name" хранится значение "John"
        age: 30 // под ключом "age" хранится значение 30
    };
```

У каждого свойства есть ключ (также называемый «имя» или «идентификатор»). После имени свойства следует двоеточие “:” , , и затем указывается значение свойства. Если в объекте несколько свойств, то они перечисляются через запятую.

В объекте user сейчас находятся два свойства:

- Первое свойство с именем “name” и значением “John”
- Второе свойство с именем “age” и значением 30

Можно сказать, что наш объект user - это ящик с двумя папками, подписанными «name» и «age».



Мы можем в любой момент добавить в него новые папки, удалить папки или прочитать содержимое любой папки.

Для обращения к свойствам используется запись «через точку»:

// получаем свойства объекта:

```
alert( user.name ); // John
alert( user.age ); // 30
```

Значение может быть любого типа.

Давайте добавим свойство с логическим значением:

```
user.secondName = “Smith”;
```

Для удаления свойства мы можем использовать оператор delete:

```
delete user.age;
```

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 };
```

КВАДРАТНЫЕ СКОБКИ

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

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

Рабочий способ:

```
let user = {};

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

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

Сейчас всё в порядке. Обратите внимание, что строка в квадратных скобках заключена в кавычки (подойдёт любой тип кавычек).

Квадратные скобки также позволяют обратиться к свойству, имя которого может быть результатом выражения. Например, имя свойства может храниться в переменной:

```
let key = "likes birds"; // то же самое, что и user["likes birds"] = true;

user[key] = true;
```

Здесь переменная `key` может быть вычислена во время выполнения кода или зависеть от пользовательского ввода. После этого мы используем её для доступа к свойству. Это даёт нам большую гибкость.

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")
```

Запись «через точку» такого не позволяет:

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

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

ВЫЧИСЛЯЕМЫЕ СВОЙСТВА

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

Пример:

```
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"
```

Смысл вычисляемого свойства прост: запись [fruit] означает, что имя свойства необходимо взять из переменной fruit

И если посетитель введёт слово “apple” , то в объекте bag теперь будет лежать свойство {apple: 5}

По сути, пример выше работает так же, как и следующий пример:

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

let bag = {}; // имя свойства будет взято из переменной fruit

bag[fruit] = 5;
```

СВОЙСТВО ИЗ ПЕРЕМЕННОЙ

В реальном коде часто нам необходимо использовать существующие переменные как значения для свойств с тем же именем.

```
function makeUser(name, age)
    { return
        {
            name: name,
            age: age // ...другие свойства
        };
    }
let user = makeUser("John", 30);
alert(user.name); // John
```

ПРОВЕРКА СУЩЕСТВОВАНИЯ СВОЙСТВА, ОПЕРАТОР «IN»

В отличие от многих других языков, особенность JavaScript-объектов в том, что можно получить доступ к любому свойству. Даже если свойства не существует – ошибки не будет!

При обращении к свойству, которого нет, возвращается undefined. Это позволяет просто проверить существование свойства:

```
let user = {};

alert( user.noSuchProperty === undefined ); // true означает "свойства нет"
```

ТАКЖЕ СУЩЕСТВУЕТ СПЕЦИАЛЬНЫЙ ОПЕРАТОР “IN” ДЛЯ ПРОВЕРКИ СУЩЕСТВОВАНИЯ СВОЙСТВА В ОБЪЕКТЕ.

Синтаксис оператора:

```
"key" in object
```

> Обратите внимание, что слева от оператора `in` должно быть *имя свойства*. Обычно это строка в кавычках.

Пример:

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

alert( "age" in user ); // true, user.age существует

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

Если мы опускаем кавычки, это значит, что мы указываем переменную, в которой находится имя свойства.

Например:

```
let user = { age: 30 };

let key = "age";
alert( key in user ); // true, имя свойства было взято из переменной key
```

Для чего вообще нужен оператор `in`? Разве недостаточно сравнения с `undefined`?

В большинстве случаев прекрасно сработает сравнение с `undefined`. Но есть особый случай, когда оно не подходит, и нужно использовать `"in"`.

Это когда свойство существует, но содержит значение `undefined`:

```
let obj = {
  test: undefined
};

alert( obj.test ); //  выведет undefined, значит свойство не существует?
alert( "test" in obj ); // true, свойство существует!
```

Задачи:

1. Создать объект Human. Статически инициализировать ему 5 свойств со значениями. Реализовать мини-интерфейс, реализующий следующие свойства:

1.1. Пользователь вводит ключ, программа отвечает существует такое свойство или нет.

1.2. Возможность добавлять новые ключи. Если при вводе,такой ключ будет существовать, то оповестить об этом пользователя.

1.3. Редактировать значения

1.4. Вывести объект

1.5. Удалить выбранное свойство

2. Создать массив с объектами. Реализовать мини-интерфейс, реализующий следующие свойства:

2.1. Вывод выбранного по индексу объекта

2.2. Добавление нового элемента в массив

2.3. Редактирование объекта

2.4.*С помощью бутстрапа вывести в таблицу выбранный объект, где каждое значение объекта в отдельной колонке, соответствующей имени ключа