Занятие 11. Тема «Объекты в JavaScript»

На прошлых занятиях мы познакомились со строками и массивами — научились хранить текст и создавать упорядоченные списки данных. Это уже открыло для нас множество возможностей, например, мы могли собрать информацию о человеке в массив: фамилию, имя, отчество, дату рождения и даже логическое значение, женат ли он.

Но есть загвоздка: запомнить, что именно хранится под индексом `2`, не так-то просто. А если полей станет ещё больше, и вовсе легко запутаться.

Сегодня мы попробуем хранить такую информацию удобнее:

- Познакомимся с объектами и принципом поиска значения по ключу — основой большинства современных структур данных.
- Узнаем, как извлекать нужную информацию по имени поля (ключа), а не по номеру.
- Разберёмся, какие типы данных можно использовать в качестве ключей и почему не любые из них подойдут.
- Научимся добавлять, изменять и удалять свойства объекта.
- Поработаем с реальными задачами: создадим объект книги, соберём массив товаров и посчитаем стоимость заказа.

Глоссарий к одиннадцатому занятию

Object (Объект)

В объектно-ориентированном программировании (ООП) объект — это экземпляр класса, то есть переменная сложного типа. Мы узнаем об этом подробнее на занятии 25, но уже сейчас их можно использовать в JavaScript без предварительного проектирования классов. Объекты позволяют описывать реальные предметы и явления через набор свойств и действий, то есть через переменные и функции.

Field (Поле)

Поле в ООП — это переменная внутри объекта. Она содержит данные, характеризующие объект, например: имя, возраст или координаты.

Method (Метод)

Метод в ООП — это функция, принадлежащая объекту. С его помощью объект может выполнять действия, например: выводить сообщение, рассчитывать стоимость или переключать режим.

Key (Ключ)

Ключ — это имя свойства в объекте. С его помощью мы можем обратиться к нужному значению внутри объекта. Ключ всегда записывается как строка. Например, в паре `name: "Иван"` слово `name` — это ключ.

Осторожно! Слово `key` имеет много значений. В программировании оно может означать и клавишу на клавиатуре, и элемент в шифровании, и часть ассоциативной структуры данных.

Value (Значение)
Значение — это содержимое, связанное с ключом. В JavaScript значением может быть что угодно: число, строка, массив, объект или даже функция. Например, в `name: "Иван"` строка `"Иван"` — это значение.

Hash (Мешанина, крошево)

Хеш — это специальное число, которое вычисляется по строке или другому объекту, чтобы ускорить поиск и сравнение. Например, строку `"JavaScript"` можно преобразовать в число вроде `875401` , и уже его сравнивать с другими хешами. Это позволяет не перебирать символы по одному. Мы не будем углубляться в алгоритмы хеширования, в конце занятия есть ссылки, если захочется узнать об этом больше.

Потренироваться в переводах и лучше запомнить определения каждого термина мы советуем в приложении Quizlet. Чтобы открыть карточки к занятию, перейдите по этой ссылке:
https://quizlet.com/ru/1049445832/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%
%D0%B2-javascript-flash-cards/?i=688d2e&x=1qqt

Программисты не стали изобретать для кода (искусственного языка) новые слова — они заимствовали слова из реальной жизни (естественного языка) и дали им новые технические смыслы. Казалось бы, удобно — всё уже знакомо. Но на практике это создаёт трудности. Когда мы слышим слова «объект», «ключ» или «метод», мы представляем стол, замок или способ действовать, а в коде они означают нечто иное.

Историческая справка — клавиша!

Все знают, что такое клавиша. Но знаете ли вы, откуда произошло это слово?

Оно образовано от латинского clavis, что означает «ключ». Причём здесь ключ? — спросите вы. Всё просто: в старинных музыкальных инструментах, например, в органе, клавиши действительно выполняли функцию ключа — они открывали доступ воздуху в органные трубы. А трубы, в свою очередь, отвечали за извлечение звуков разной высоты.

На органах X века клавиши часто подписывались буквами латинского алфавита, чтобы музыкант ориентировался не на ноты, а на последовательность букв. Это упрощало игру и позволяло не путаться в

последовательность букв. Это упрощало игру и позволяло не путаться в звуковом ряду.

Позже такая буквенная запись перешла с инструмента на бумагу. Так появилась музыкальная нотация, где в начале строки ставился ключ — специальный символ, определяющий, какие ноты будут звучать на каких линейках. Слово сохранилось, а вместе с ним и его смысл: ключ как способ расшифровки, открытия и управления.

Давайте начнём не с синтаксиса, а с образов. Сначала — о том, что мы вообще называем объектом в жизни, а потом — как это понятие используется в программировании.

Что такое объект?

Вы сидите за столом и слушаете или читаете урок. Стол твёрдый, его можно потрогать. Урок — нет, но его можно услышать, прочитать, запомнить или придумать самостоятельно. Ещё в школе нас приучают делить мир на предметы и явления. У стола есть такие характеристики, как материал, высота, ширина и длина. У урока тоже есть длина — его продолжительность. А ещё содержание, цель и результат.

Согласитесь, перечисленные критерии не позволили бы с точностью воссоздать именно этот стол. Но они помогли бы подобрать подходящий, например, достаточно высокий, чтобы за ним можно было удобно сидеть, и с деревянной поверхностью, на которой работала бы мышка. Мы описали лишь те свойства, которые важны для конкретной задачи.

То же самое происходит в программировании. Когда мы описываем объект, то не стремимся перечислить всё возможное. Мы выбираем только те характеристики и действия, которые имеют значение в данном контексте. Такое упрощённое, но достаточное описание и называют объектом.

Более того, с объектами можно не только хранить свойства, но и выполнять действия. Если вы слушаете урок в записи, то можете остановить воспроизведение, перемотать или начать сначала. Если стол стоит не там, где удобно, вы можете его передвинуть. Эти действия — то, что объект может делать. В программировании их называют методами, а характеристики — полями.

Рассмотрим простой пример. Машина проехала путь `s` с постоянной скоростью `v`. Сколько времени она потратила на дорогу?

```
let s = 250;  // km
const v = 50; // km/h
let t = s / v;
console.log(t, " часов");
```

А если у нас много машин? У каждой свои скорость, расстояние и время в пути. Можно создать три отдельных массива: `speed`, `distance`, `time`. Но тогда легко запутаться, где чья скорость и сколько ехала именно вторая машина.

А можно сделать проще: создать объект «машина», который сам хранит свои параметры, и собрать массив уже из таких объектов. Каждая машина будет знать свои скорость, путь и время. Так устроено большинство программ: они работают не с абстрактными наборами данных, а с объектами, описывающими реальные сущности.

Мы уже работали с объектами

Один из первых объектов, с которым мы познакомились в JavaScript, это `console`. Он умеет выводить сообщения и ошибки. Например:

```
console.log("Привет, мир!");
console.error("Что-то пошло не так...");
```

Здесь `log` и `error` — это методы объекта `console`, то есть действия, которые он может выполнять.

Как создать свой объект?

В JavaScript есть как минимум три способа создать объект. Рассмотрим каждый по порядку.

1. Некоторые типы — уже объекты

Строки и массивы — это тоже объекты. У них есть свойства (например, `length`) и методы (например, `slice`).

```
let s = "А я всё понял!";
console.log(s.length);
console.log(s.slice(4));
```

2. Через конструктор и оператор `new`

Можно использовать встроенные классы JavaScript, например, для получения текущей даты:

```
let dt = new Date();
console.log(dt.getDate());
```

Здесь `Date` — встроенный объект, а `new` создаёт его новый экземпляр.

3. Через литерал объекта

Самым удобным способом будет использование литерала объекта — фигурные скобки с парами ключ: "значение".

```
const human = {
  "name": "Константин",
  "date of birth": new Date(2000, 1, 1),
  "height": 185,
  "welcome": () => console.log("Привет!")
};
```

Объекты хороши тем, что позволяют объединить связанные данные в одну сущность, как папка объединяет документы, шкаф — вещи, а посудомойка — тарелки.

Синтаксис объекта

Объект создаётся с помощью фигурных скобок `{}` . Внутри пишутся пары:

- ключ — имя свойства (обычно строка);
- значение — данные или функция;
- пары отделяются запятыми;
- между ключом и значением ставится двоеточие.

Пример:

```
const book = {
title: "1984",
author: "Джордж Оруэлл"
};
```

JavaScript автоматически превращает ключи в строки. Поэтому, если имя ключа — допустимое имя переменной, кавычки можно опустить:

```
const example = {
 name: "Анна",      // ключ без кавычек
 "full name": "Анна Иванова" // ключ с пробелом — кавычки обязательны
};
```

Обратите внимание! Вычисляемые выражения не могут быть ключами, и, например, массив нельзя использовать как ключ без приведения к строке.

```
const human = {
  name: "Константин",
  "date of birth": new Date(2000, 1, 1),
  height: 185,
  welcome: () => console.log("Привет!")
};
```

В реальных задачах нам редко бывает нужен весь объект целиком. Чаще только один его элемент. Например, представим библиотеку: у нас есть объект, где каждый ключ — это название книги, а значение — её содержание. Мы же не просим библиотекаря выдать весь архив, а просто называем нужную книгу.

То же самое и в JavaScript: объект может хранить множество данных, но доступ к ним можно получить по одному ключу — быстро и точно.

Как получить данные из объекта?

1. Через квадратные скобки

Поскольку объект — это семантически массив, хоть и ассоциативный, можно обращаться к значениям его полей через квадратные скобки по ключу-строке:

```
console.log(human["name"]);  // "Константин"
console.log(human["date of birth"]);
```

Этот способ универсален — в кавычках указывается ключ в виде строки. Подходит для любых имён, даже с пробелами и спецсимволами.

2. Через точку

Если имя ключа соответствует правилам написания переменных (то есть без пробелов, начинается с буквы и т.д.), можно использовать точечную запись:

```
console.log(human.name); // "Константин"
```

Важно! Если ключ содержит пробел или символы, не разрешённые в именах переменных, такой способ вызовет ошибку:

```
console.log(human.date of birth); // синтаксическая ошибка
```

А если ключа нет?

В отличие от обращения к несуществующей переменной, которое вызывает ошибку и прерывает выполнение программы, попытка обратиться к несуществующему ключу объекта просто возвращает значение `undefined`. Это удобно: программа продолжит работать, даже если такого свойства нет.

К счастью, `undefined` — ложное значение. То есть в условиях `if` или `hile` оно будет интерпретироваться как `false`. Но важно помнить: `undefined` не равно `false`, это разные типы данных и сравнивать их стоит осторожно.

```
console.log(human.best_friend);  // undefined
```

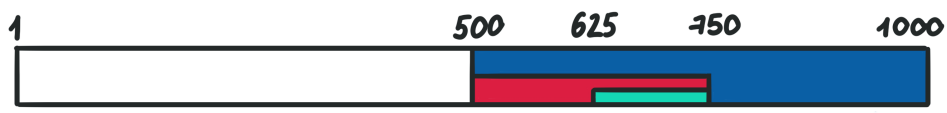
Почему может возникнуть `undefined` при обращении к ключу? Обычно по одной из двух причин:

- вы опечатались в названии ключа;
- вы ещё не добавили это свойство в объект.

Под капотом: как JavaScript ищет значения по ключу?

Представим классическую задачку. У вашего друга есть книга на 1000 страниц, и вы не знаете, какую страницу он читал сегодня утром. Он предлагает вам угадать. Сколько вопросов, на которые можно ответить «да» или «нет», вам потребуется, чтобы точно найти нужную страницу?

Если спрашивать по порядку «Ты читаешь первую страницу? Или вторую?», это займёт много времени. Гораздо быстрее делить варианты пополам. Сначала «Ты прочитал больше половины?» — и сразу отбрасываем 500 страниц. Потом делим оставшиеся и снова уточняем. Так шаг за шагом нужная страница найдётся всего за 10 вопросов. Это и есть бинарный поиск.



Бинарный поиск применяется, когда данные отсортированы. Например, если у нас есть массив, отсортированный по возрастанию, мы можем быстро найти, есть ли в нём нужное число, без полного перебора.

Если пойти дальше, из отсортированного массива можно построить бинарное дерево поиска. У него есть корень — «средний» элемент. Все значения меньше него располагаются слева, все те, что больше — справа. Это ещё один способ ускорить поиск.

А как обстоит дело с объектами?

В JavaScript объект — это ассоциативный массив, но его ключи не упорядочены. Нам не гарантируется, что они будут расположены в каком-то определённом порядке, даже если мы их добавили последовательно. В большинстве реализаций ключи перебираются в порядке их добавления, но это не правило, а поведение конкретного движка.

Поэтому, в отличие от массивов, с объектами нельзя применять бинарный поиск. Но он и не нужен: внутри JavaScript значение по ключу ищется не перебором, а через более быстрые механизмы, к примеру, хеш-таблицы. Это позволяет получать доступ к свойству за доли секунды, вне зависимости от количества полей в объекте.

Объект в JavaScript — это не просто хранилище данных. Он гибок: его можно дополнять, обновлять и даже очищать. В отличие от массивов, где структура фиксирована, объект можно менять на ходу, добавляя новые свойства, редактируя старые и удаляя ненужные.

Давайте посмотрим, как это работает на практике.

Как изменять или добавлять поля?

В JavaScript добавление и изменение свойств объекта происходит одинаково — с помощью обычного присваивания по ключу:

- через точку;
- через квадратные скобки;
- с помощью метода `OObject.defineProperty(obj, prop, descriptor)` (подробно — в следующем занятии).

Если ключ уже существует, значение будет заменено. Если ключа не было, будет создана новая пара.

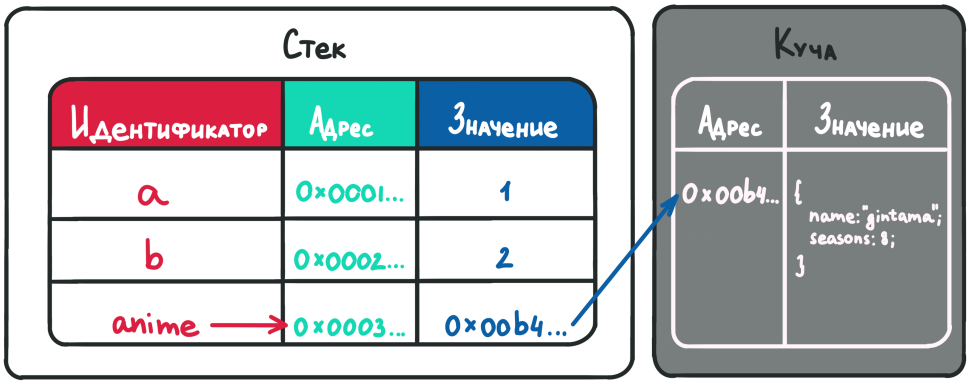
```
human["name"] = "Константин Константинович";
human.work = "Слесарь-сантехник";
```

Выведем объект в консоль и посмотрим, как изменился Константин.

«Константин, как ты мог?... Human же был константой?!»

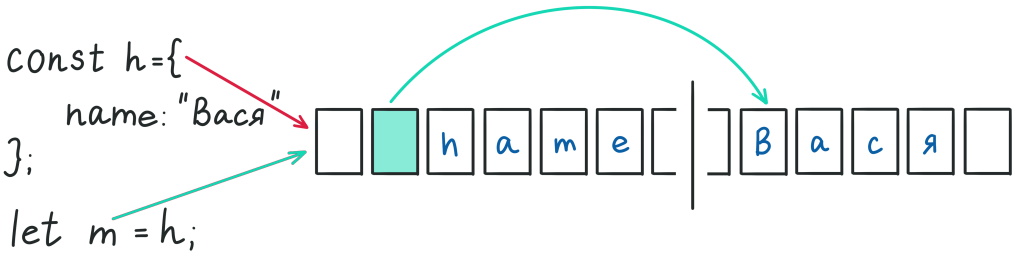
Мы подошли к очень важному моменту. Все переменные в JavaScript, которые хранят простые типы, создаются на стеке — особой области памяти для относительно продолжительного содержания данных с сохранением порядка их создания.

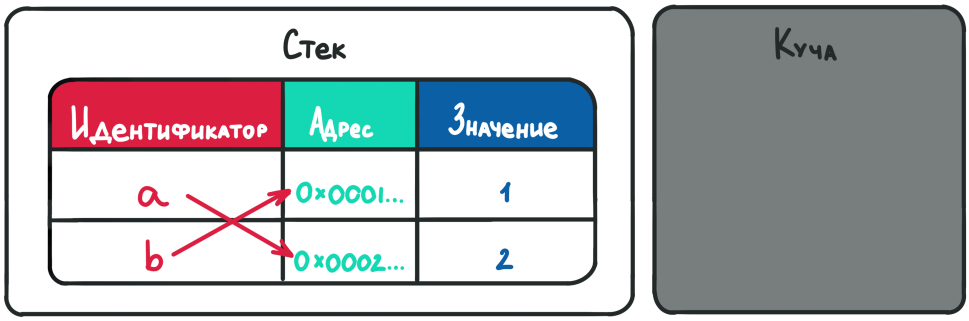
А сложные типы данных (массивы, объекты, функции) хранятся по ссылке в куче — области для короткоживущих данных, порядок расположения которых не важен, важны только адреса.

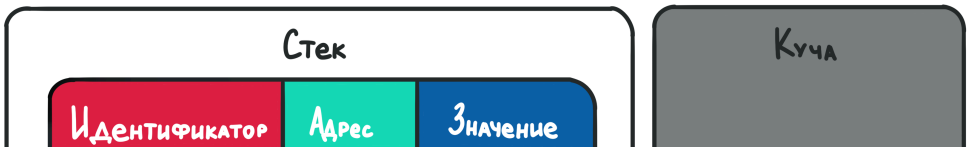


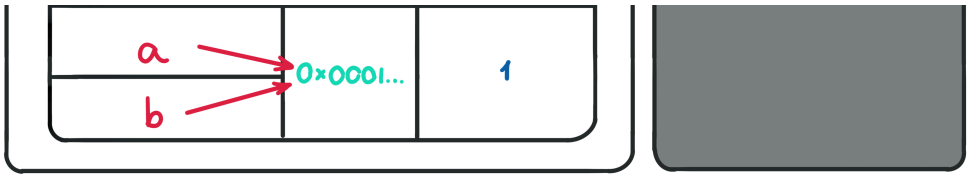
Это означает, что при создании новой переменной возможны два сценария:

- Если переменной присваивается новое значение — например, строка, число, массив или объект — в памяти выделяется место под это значение, а переменная получает адрес первого байта этой области.
- Если же переменная создаётся как результат присваивания уже существующей переменной, никакой новой области не создаётся — просто записывается тот же адрес. Так появляются две ссылки на один и тот же объект.





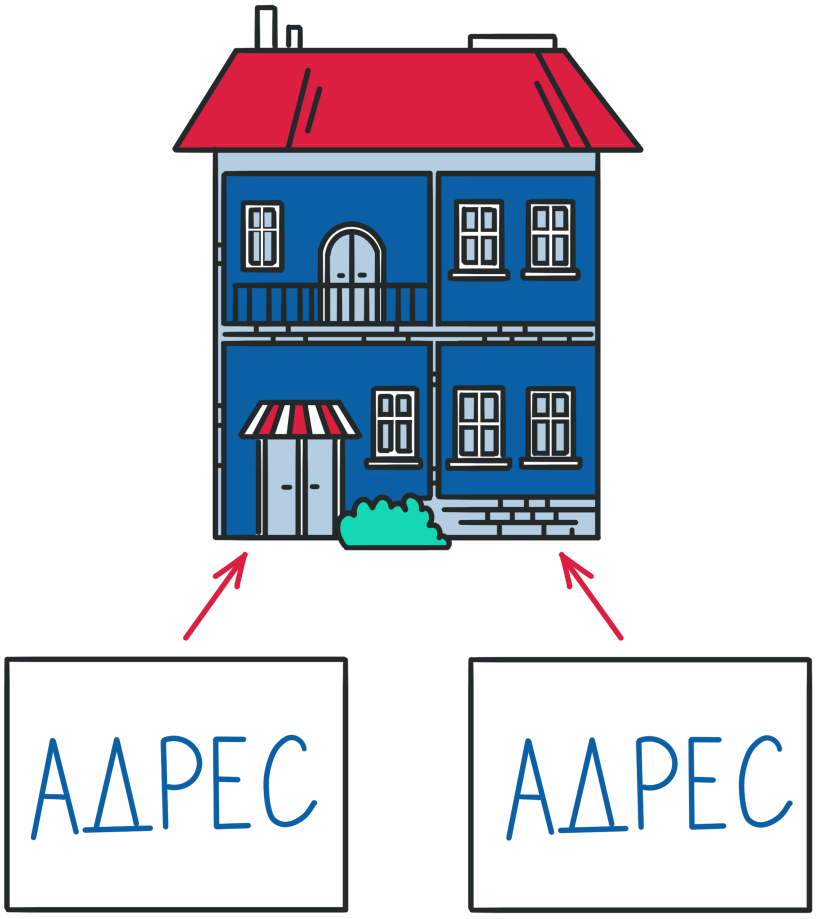




Это как два пульта от одного телевизора. Если вы подключили второй пульт, вы не создали новый телевизор, а просто получили ещё один способ управлять тем же самым устройством. Хотите переключить канал? Пожалуйста, с любого пульта.

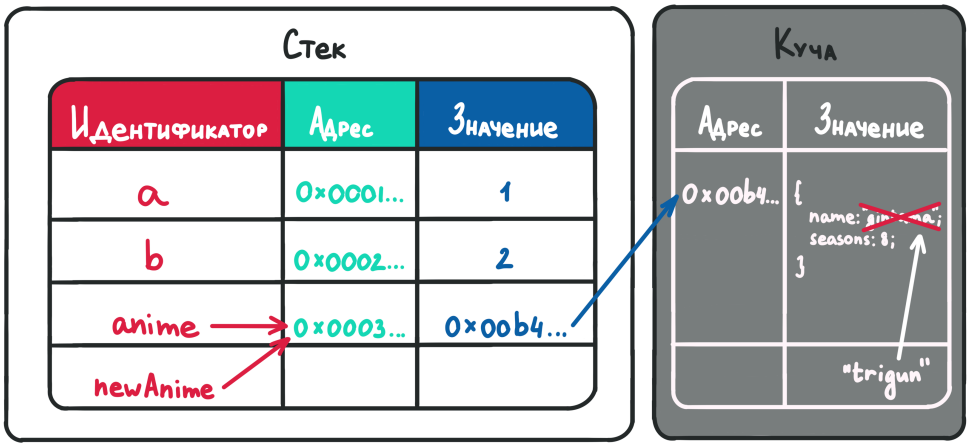
То же самое происходит и с объектами. Если вы присвоили объект другой переменной, то теперь у вас две переменные, которые управляют одним и тем же объектом. Изменение через одну ссылку будет видно и через другую.

Можно представить это и так: переменные — это бумажки с адресами, а объект — дом. Если у вас две бумажки с одним и тем же адресом, вы оба раза придёте в одно и то же место.



А что насчет `const` ?

Когда мы используем спецификатор `const` , мы фиксируем саму ссылку, то есть не можем переназначить переменную на другой объект. Но содержимое объекта (его поля) менять можно. Константа в данном случае — это адрес, а не «всё, что по этому адресу».



А если мы хотим запретить любые изменения?

Тогда используем специальный метод:

```
Object.freeze(human);
```

После этого любые попытки изменить или удалить свойства этого объекта будут проигнорированы. Попробуйте: переименуйте Константина и проверьте, что получится.

Удаление ключей

Допустим, Константин уволился и больше не работает сантехником. Удалим это свойство:

```
delete human.work;
```

Но, если вы предварительно заморозили объект с помощью `Object.freeze()` , удаление не произойдёт. Попробуйте на своём примере: создайте объект, заморозьте его, а потом попытайтесь что-нибудь удалить.

Присвоить. Буквально

Если мы присваиваем один объект другому:

```
let foreign_car = {
  brand: 'Mercedes'
};
let local_car = foreign_car;
local_car.brand = 'Lada';
console.log(foreign_car);
```

Мы не создаём копию: обе переменные ссылаются на один и тот же объект. Изменения через одну переменную будут видны через другую.

Как сделать копию объекта?

`Object.assign` позволяет создать новый объект или объединить их. Вторую его особенность мы рассмотрим в следующий раз.

```
// куда скопировать, что скопировать - не запускайте этот код!
let result = Object.assign(target, source);
```

Пример:

```
let foreign_car = {
  brand: 'Mercedes'
};
let local_car = Object.assign({}, foreign_car);
local_car.brand = 'Lada';
console.log(foreign_car, local_car);
```

Теперь это два разных объекта: один — «Мерседес», другой — «Лада».

Какие ключи можно использовать?

В JavaScript ключи всегда преобразуются в строки. Даже если мы укажем число, булевое значение или `null`, они будут автоматически превращены в строки:

```
const example = {
  10: "десять", // станет "10"
  true: "истина", // станет "true"
  null: "ничего" // станет "null"
};
```

Это может удивить, особенно если вы раньше работали с Python, где в словаре `(dict)` можно использовать ключи разных типов.

Сравнение на равенство

Сравнение на равенство в JavaScript работает только для примитивных типов: чисел, строк, булевых значений и т.д. Это мы уже встречали на первом занятии.

А вот с объектами всё не так просто. При сравнении двух объектов оператор `===` проверяет не содержимое, а ссылки, то есть равны ли адреса, на которые ссылаются переменные. Даже если два объекта выглядят одинаково, они считаются разными, если были созданы независимо.

Чтобы сравнить содержимое объектов, нужно писать специальную функцию. Казалось бы, задача типовая, но до сих пор не существует универсального решения, которое подходило бы для всех случаев.

Почему так?

Потому что в реальной жизни нам не всегда нужна полная идентичность. Вспомните стол в начале занятия. Когда вы за ним сидите, важно, чтобы он был подходящей высоты, с удобной поверхностью и достаточно широким, чтобы поместился ноутбук. Вас не интересует, сколько у него точно миллиметров по краю или кто его изготовил.

Так же и с объектами: иногда достаточно сравнить 2-3 свойства, а остальное можно опустить. Полное совпадение требуется далеко не всегда.

Как написать функцию для глубокого сравнения объектов, обсудим в одном из следующих занятий.

Массив объектов. Вложенные объекты

Объекты можно объединять в массивы. Это удобно, когда у вас есть несколько однотипных сущностей, например, студентов:

```
const students = [
{
  name: 'Ася',
  marks: [5,4,5,3,4,3,5],
},
{
  name: 'Вася',
  marks: [2,3,2,4,3,3,4,2]
}
];
```

Чтобы узнать имя первого студента, достаточно обратиться к первому элементу массива, а затем — к полю `name`:

```
console.log(students[0].name);
```

Такой приём является одним из самых частых в практике — массив объектов с доступом по индексу и по ключу.

Но и это ещё не всё. Объекты могут содержать другие объекты, и таких уровней вложенности может быть сколько угодно:

```
const car = {
coordinates: {
  lat: 55,
  lon: 37
},
velocity: 50
};
console.log(car.coordinates.lat);
```

Здесь внутри объекта `car` находится вложенный `coordinates`, а внутри него — уже конкретные значения широты и долготы. Мы пошагово добрались до нужного значения с помощью цепочки точек.

Как вы понимаете, это только начало. Вы получили инструмент, с помощью которого можно строить многоуровневые конструкции, способные описывать самые разные и сложные объекты — от заказов в интернет-магазине до данных о погоде в разных странах.

Использование методов

Вернёмся к нашему знакомому:

```
const human = {
    "name": "Константин",
    "date of birth": new Date(2000, 1, 1),
    "height": 185,
    "welcome": () => console.log("Привет!"),
    "goodbye": function() {console.log('Пока!')}
```

Как мы могли заметить, ключу `welcome` соответствует стрелочная функция, а ключу `goodbye` — обычная. И та, и другая — это методы объекта, то есть действия, которые он может выполнять.

Методы — это просто функции, которые приписаны объекту как одни из его свойств. Мы можем вызывать их, добавлять, менять и даже присваивать после создания объекта.

```
human.welcome(); // "Привет!"
human.goodbye(); // "Пока!"
```

Вы уже умеете создавать объекты, добавлять в них поля, вызывать методы. Всё выглядит просто, пока не появляются первые баги. Давайте заглянем туда, где чаще всего спотыкаются новички. Ошибки простые, но коварные — разберём их заранее, чтобы не наступать на эти грабли в коде.

Типичные ошибки

Тип ошибки 1: Использование точки для доступа к ключу с пробелом

При попытке обратиться к ключу объекта, содержащему пробелы, через точку возникнет синтаксическая ошибка.

Пример ошибки:

```
const cat = {
  name: 'Мурзик',
  'любимая еда': 'рыбка'
};

console.log(cat.любимая еда) // Ошибка!
```

Объяснение:

JavaScript ожидает после точки имя свойства (ключ) без специальных символов — по сути, идентификатор как название переменной или функции. Ключ с пробелом можно получить только через квадратные скобки.

Решение:

```
cat["любимая еда"]; // Правильно
```

Тип ошибки 2: Сравнение объектов с помощью операторов `==` и `===`

Даже если объекты выглядят одинаково, они считаются разными, потому что в памяти это различные сущности.

Пример ошибки:

```
const a = { name: "Алиса" };
const b = { name: "Алиса" };
const c = b;

console.log(a === b); // false - разные объекты
console.log(c === b); // true - разные ссылки на один объект
```

Объяснение:

Даже если объекты выглядят одинаково, они могут быть разными сущностями в памяти. Сравниваются не значения, а ссылки на них.

Решение:

Не используйте `==` и `===` длля сравнения объектов на равенство. В практикуме приведено решение этой задачи и оно значительно сложнее, чем в одну строку.

Для сравнения содержимого объектов нужно проверять каждое свойство отдельно.

Практикум

Задача 1: Ассоциативный массив

Ситуация:
Вам стало жалко библиотекаря, и вы решили помочь: написать простую программу, которая подскажет, где искать нужную книгу на полке.

Задача:
Создайте структуру данных, которая позволяет быстро определить местоположение книги по её названию.

Шаги реализации:

1. Выбрать подходящий тип данных, объект отлично подойдёт.
2. Написать для примера 3 книги.
3. Показать библиотекарю, как добавлять и менять книги в каталоге.

Реализация:

```
const catalog = {
  'Физики шутят': 'Второй зал, седьмой шкаф, пятая полка',
  'Приключения Электроника': 'Первый зал, десятый шкаф, вторая
полка',
  'Сборник задач для поступающих в ВУЗы': 'Третий зал, восьмой
шкаф, семнадцатая полка'
};
console.log(catalog['Физики шутят']);
```

Задача 2: Массив объектов

Ситуация:

Вы — веб-мастер интернет-магазина. Настало время написать корзину, которая умеет считать итоговую сумму заказа.

Задача: НСоздайте массив продуктов. Каждый продукт — объект с полями:

- name — название товара;
- price — цена за единицу;

- quantity — количество.

Посчитайте общую стоимость заказа.

Шаги реализации:

1. Выбрать подходящий тип данных. Подойдёт позиционный массив, каждым элементом которого является объект.
2. Описать пример структуры данных. Создать результат `= 0` .
3. Перебрать массив поэлементно, умножая количество на цену и прибавляя произведение к результату.

Реализация:

```
const cart = [
    {
      name: 'Розы алые',
      price: 75,
      count: 1000000
    },
    {
      name: 'Ленты белые',
      price: 10,
      count: 333333
    }
];
let result = 0;
for (let product of cart) {
  result += product.price * product.count;
}
console.log(result, ' рублей');
```

Сегодня мы:

- познакомились с понятиями «объект», «поле», «метод»;
- научились добавлять, изменять и удалять свойства объекта;
- узнали, как извлекать нужную информацию по имени поля (ключа);
- поработали с реальными задачами, которые очень удобно решать с применением объектов.

Тестирование

(1 балл) 1. Какой синтаксис используется для создания объекта?
Выберите один вариант ответа.

- Квадратные скобки `[]`
- Фигурные скобки `{}`
- Круглые скобки `()`
- Угловые скобки `<>`

Правильный ответ: 2) Фигурные скобки `{}`

(2 балла) Что произойдёт при таком коде:

```
const a = { name: "Алиса" };
const b = a;
b.name = "Вася";
```

- Изменения не повлияют на объект a
- Будет создан новый объект b с такими же значениями
- Изменения повлияют на объект a
- JavaScript выдаст ошибку

Правильный ответ: 3) Изменения повлияют на объект a

(2 балла) 3. Как правильно обратиться к ключу, содержащему пробел?
Выберите один вариант ответа.

- `object.ключ с пробелом`
- `object("ключ с пробелом")`
- `object["ключ с пробелом"]`
- `object<ключ с пробелом>`
- К ключу с пробелом обращаться нельзя

Правильный ответ: 3) `object["ключ с пробелом"]`

(2 балла) 4. Как добавить новое свойство в объект `my_object` ?
Выберите **все** подходящие варианты ответа.

- `my_object = 'новое'`
- `my_object.newProperty = 'значение'`
- `new my_object.property = 'значение'`
- `my_object.add("property")`
- `my_object['new_property'] = 'значение'`

Правильные ответы:
2) `my_object.newProperty = 'значение'`
5) `my_object['new_property'] = 'значение'`

(2 балла) 5. Что делает оператор `delete` ?
Выберите один вариант ответа.

- Удаляет переменную полностью
- Удаляет объект из памяти
- Удаляет указанное свойство объекта
- Очищает все свойства объекта

Правильный ответ: 3) Удаляет указанное свойство объекта

(3 балла) 6. Чем массив отличается от объекта?
Выберите **все** подходящие варианты ответа.

- Массив содержит только строки
- Массив — это тоже объект, но с числовыми ключами
- Объекты работают только со строковыми ключами

Правильные ответы:

Правильные ответы:

2) Массив — это тоже объект, но с числовыми ключами

3) Объекты работают только со строковыми ключами

Конвертация баллов в отметку:

- **11–12 баллов** — отметка **«5»**
- **8–10 баллов** — отметка **«4»**
- **5–7 баллов** — отметка **«3»**
- **≤ 4 баллов** — отметка **«2»**

Домашняя работа

Описание задачи:

Вы подрабатываете помощником в школьной библиотеке. Сегодня вас попросили внести изменения в карточку одной из книг в электронной базе данных. Работа важная, ведь библиотека должна всегда отображать актуальную информацию, чтобы ученики могли быстро находить нужные книги!

Указания:

Создайте объект `libraryBook`, в котором будут следующие свойства:

- `title` — название книги;
- `author` — автор;
- `genre` — жанр;
- `year` — год издания;
- `available` — наличие в библиотеке (булевое значение: `true` или `false`).

После создания объекта:

1. Добавьте новое поле `rating` — оценка книги по версии читателей.
2. Измените значение поля `available` на противоположное.
3. Удалите поле `genre`, так как библиотекарь решил временно не указывать жанры.

Ожидаемый результат:

Финальный объект должен отразить все изменения и быть выведен в консоль:

```
{
  title: "Преступление и наказание",
  author: "Ф. М. Достоевский",
  year: 1866,
  available: false,
  rating: 4.8
}
```

Критерии оценивания:

2 балла — правильная структура объекта с нужными полями.

3 балла — корректное добавление, изменение и удаление свойств.

2 балла — использование `delete` для удаления жанра.

3 балла — чистота и читаемость кода (отступы, имена переменных, комментарии при необходимости).

Решение:

```
// Шаг 1: создаём объект с нужными свойствами
let libraryBook = {
  title: "Преступление и наказание",
  author: "Ф. М. Достоевский",
  genre: "Роман",
  year: 1866,
  available: true
};

// Шаг 2: добавляем рейтинг книги
libraryBook.rating = 4.8;

// Шаг 3: меняем доступность на противоположную
libraryBook.available = !libraryBook.available;

// Шаг 4: удаляем жанр, как просил библиотекарь
delete libraryBook.genre;

// Шаг 5: выводим финальный объект в консоль
console.log(libraryBook);
```

Практическая работа

Уровень 1: Базовый

Задача:

Создайте объект `book`, в котором будут следующие свойства:

- title: `"Мастер и Маргарита"`
- author: `"Михаил Булгаков"`
- year: `1967`

Выведите в консоль название книги и её автора.

Решение:

Для создания объекта используется литерал объекта — фигурные скобки `{}`. Внутри указываются пары ключ–значение, разделённые запятой. Подсказка: для доступа к свойствам объекта используют точечную нотацию: `object.key`.

Ответ:

```
const book = {
    title: "Мастер и Маргарита",
    author: "Михаил Булгаков",
```

```
    year: 1967
};

console.log(book.title);
console.log(book.author);
```

Уровень 2: Средний

Задача:

1. Создайте ассоциативный массив (объект) геометрических фигур.

- Ключ — название фигуры
- Значение — объект с параметрами фигуры (у каждой фигуры — свои)

2. Добавьте в объект:

- круг с радиусом,
- прямоугольник с длиной и шириной,
- треугольник с основанием и высотой.

3. Покажите, как можно посчитать площадь прямоугольника.

Решение:

```
const figures = {
  round: {
    d: 20
  },
  rectangle: {
    a: 6,
    b: 10
  },
  triangle: {
    leg1: 4,
    hypotenuse: 5,
    leg2: 3
  }
};
console.log(figures.rectangle.a * figures.rectangle.b);
```

Уровень 3: Продвинутый

Задача:

У вас есть массив данных с камер видеонаблюдения, состоящий из объектов, описывающих проезжающие мимо автомобили. У авто есть номер и скорость. Допустимая скорость на участке — 90 км/ч, но камера имеет погрешность измерения примерно 10 км/ч. Вывести на печать нарушителей.

Решение:

1. Описать массив.
2. Перебрать массив циклом по номеру элемента.

3. Вывести номера и скорости автомобилей, превышающих разрешенные показатели с допустимой погрешностью.