**Лекція 7. Об'єкти як асоціативні масиви. Перебір властивостей об’єктів.** **Передача об’єктів за посиланням.**

1. Об'єкти як асоціативні масиви.

2. Перебір властивостей об’єктів.

3. Передача об’єктів за посиланням.

**1.Об'єкти як асоціативні масиви.**

Об'єкти в JavaScript поєднують в собі два важливих функціонала.

Перший - це асоціативний масив: структура, придатна для зберігання будь-яких даних. У цьому розділі ми розглянемо використання об'єктів саме як масивів.

Другий - мовні можливості для об'єктно-орієнтованого програмування.

**Асоціативні масиви**

Асоціативний масив - структура даних, в якій можна зберігати будь-які дані в форматі ключ-значення.

Її можна легко уявити як шафу з підписаними ящиками. Всі дані зберігаються в скриньках. На ім'я можна легко знайти ящик і взяти те значення, яке в ньому лежить.



На відміну від реальних шаф, в асоціативний масив можна в будь-який момент додати нові іменовані «ящики» або видалити існуючі. Далі ми побачимо приклади, як це робиться.

До речі, в інших мовах програмування таку структуру даних також називають «словник» і «хеш».

**Створення об'єктів**

Порожній об'єкт («порожню шафу») може бути створений одним з двох синтаксисів:

1. o = new Object();

2. o = {}; // пусті фігурні дужки

Зазвичай все користуються синтаксисом (2), тому що він коротше.

**Операції з об'єктом**

Об'єкт може містити в собі будь-які значення, які називаються властивостями об'єкта. Доступ до властивостей здійснюється на ім'я властивості (іноді говорять «по ключу»).

Наприклад, створимо об'єкт person для зберігання інформації про людину:

var person = {}; // поки пустий



Основні операції з об'єктами - це створення, отримання та видалення властивостей.

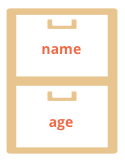
Для звернення до властивостей використовується запис «через крапку», виду об’єкт.властивість, наприклад:

// при присвоєнні властивості в об’єкті автоматично створюється "ящик"

// з ім’ям "name" і в нього записується вміст 'Вася'

person.name = 'Вася';

person.age = 25; // запишемо ще одну властивість: з ім’ям 'age' і значенням 25



Значення зберігаються «всередині» ящиків. Звернемо увагу - будь-які значення, будь-яких типів: число, рядок - не важливо.

Щоб прочитати їх - також звернемося через крапку:

alert( person.name + ': ' + person.age ); // "Вася: 25"

Видалення здійснюється оператором delete:

delete person.age;

Залишилась тільки властивість name:



Іноді буває потрібно перевірити, чи є в об'єкті властивість з певним ключем.

Для цього є особливий оператор: "in".

Його синтаксис: "prop" in obj, причому ім'я властивості - у вигляді рядка, наприклад:

if ("name" in person) {

alert( "Властивість name існує!" );

}

Втім, частіше використовується інший спосіб - порівняння значення з undefined.

**В JavaScript можна звернутися до будь-якої властивості об'єкта, навіть якщо її немає.**

Помилки не буде.

Але якщо властивість не існує, то повернеться спеціальне значення *undefined:*

var person = {};

alert( person.lalala ); // undefined, нема властивості з ключем lalala

Таким чином ми можемо легко перевірити існування властивості - отримавши його і порівнявши з undefined:

var person = {

name: "Василь"

};

alert( person.lalala === undefined ); // true, властивості немає

alert( person.name === undefined ); // false, властивість є.

**Різниця між перевірками in і === undefined**

Є два засоби для перевірки наявності властивості в об'єкті: перше - оператор in, друге - отримати його і порівняти з undefined.

Вони майже ідентичні, але є одна невелика різниця.

Справа в тому, що технічно можливо, що властивість є, а її значенням є undefined:

var obj = {};

obj.test = undefined; // додали властивість зі значенням undefined

// перевіримо наявність властивостей test і заздалегідь відсутнього blabla

alert( obj.test === undefined ); // true

alert( obj.blabla === undefined ); // true

... При цьому, як видно з коду, при простому порівнянні наявність такої властивості буде неможливо відрізнити від її відсутності.

Але оператор in гарантує правильний результат:

var obj = {};

obj.test = undefined;

alert( "test" in obj ); // true

alert( "blabla" in obj ); // false

Як правило, в коді ми не будемо привласнювати undefined, щоб коректно працювали обидві перевірки. А в якості значення, що позначає невідомість і невизначеність, будемо використовувати null.

**Доступ через квадратні дужки**

Існує альтернативний синтаксис роботи з властивостями, що використовує квадратні дужки об’єкт['властивість']:

var person = {};

person['name'] = 'Вася'; // те ж, що і person.name = 'Вася'

Записи person['name'] і person.name ідентичні, але квадратні дужки дозволяють використовувати в якості імені властивості будь-який рядок:

var person = {};

person['улюблений стиль музики'] = 'Рок';

Таке привласнення було б неможливо «через крапку», так інтерпретатор після першого пробілу подумає, що властивість закінчилася, і далі видасть помилку:

person.улюблений стиль музики = 'Рок'; // ??? помилка

В обох випадках, **ім'я властивості має бути рядком**. Якщо використано значення іншого типу - JavaScript приведе його до рядка автоматично.

**Доступ до властивості через змінну**

Квадратні дужки також дозволяють звернутися до властивості, ім'я якої зберігається у змінній:

var person = {};

person.age = 25;

var key = 'age';

alert( person[key] ); // виведе person['age']

Взагалі, якщо ім'я властивості зберігається у змінній (var key = "age"), то єдиний спосіб до нього звернутися - це квадратні дужки person[key].

Доступ через крапку використовується, якщо ми на етапі написання програми вже знаємо назву властивості. А якщо вона буде визначено по ходу виконання, наприклад, введено відвідувачем і записано в змінну, то єдиний вибір - квадратні дужки.

**Оголошення з властивостями**

Об'єкт можна заповнити значеннями при створенні, вказавши їх у фігурних дужках: { ключ1: значення1, ключ2: значення2, ... }.

Такий синтаксис називається *літеральним* (англ. Literal).

Наступні два фрагмента коду створюють однаковий об'єкт:

var menuSetup = {

width: 300,

height: 200,

title: "Menu"

};

// те саме, що:

var menuSetup = {};

menuSetup.width = 300;

menuSetup.height = 200;

menuSetup.title = 'Menu';

Назви властивостей можна перераховувати як в лапках, так і без, якщо вони задовольняють обмеженням для імен змінних.

Наприклад:

var menuSetup = {

width: 300,

'height': 200,

"мама мила раму": true

};

Як значення можна тут же вказати й інший об'єкт:

var user = {

name: "Таня",

age: 25,

size: {

top: 90,

middle: 60,

bottom: 90

}

}

alert(user.name) // "Таня"

alert(user.size.top) // 90

Тут значенням властивості size є об'єкт {top: 90, middle: 60, bottom: 90 }.

**Висновки:**

* Об'єкти - це асоціативні масиви з додатковими можливостями:
* Доступ до елементів здійснюється:

- Безпосередньо по ключу obj.prop = 5

- Через змінну, в якій зберігається ключ:

var key = "prop";

obj[key] = 5

* Видалення ключів: delete obj.name.
* Існування властивості може перевіряти оператор in: if ("prop" in obj), як правило, працює і просто порівняння if (obj.prop !== undefined).

**2.Перебір властивостей об’єктів.**

Для перебору всіх властивостей з об'єкта використовується цикл за властивостями for..in. Ця синтаксична конструкція відрізняється від розглянутого раніше циклу for(;;).

**for..in**

Синтаксис:

for (key in obj) {

/\* ... робити щось з obj[key] ... \*/

}

При цьому for..in послідовно перебере властивості об'єкта obj, ім'я кожного властивості буде записано в key  і викликано тіло циклу.

**Оголошення змінної в циклі for (var key in obj)**

Допоміжну змінну key можна оголосити прямо в циклі:

for (var key in menu) {

// ...

}

Так іноді пишуть для стислості коду. Можна використовувати і будь-яка інша назва, крім key, наприклад for(var propName in menu).

Приклад ітерації за властивостями:

var menu = {

width: 300,

height: 200,

title: "Menu"

};

for (var key in menu) {

// цей код буде викликаний для кожної властивості об’єкта

// ..і виведе ім’я властивості та його значення

alert( "Ключ: " + key + " значення: " + menu[key] );

}

Зверніть увагу, ми використовували квадратні дужки menu[key]. Як вже говорилося, якщо ім'я властивості зберігається в змінній, то звернутися до неї можна тільки так, не через крапку.

**Кількість властивостей в об'єкті**

Як дізнатися, скільки властивостей зберігає об'єкт?

Готового методу для цього немає.

Самий крос-браузерний спосіб - це зробити цикл за властивостями і порахувати, ось так:

var menu = {

width: 300,

height: 200,

title: "Menu"

};

var counter = 0;

for (var key in menu) {

counter++;

}

alert( "Всього властивостей: " + counter );

У наступних розділах ми пройдемо масиви і познайомимося з іншим, більш коротким, викликом: Object.keys(menu).length.

**У якому порядку перебираються властивості?**

Для прикладу, розглянемо об'єкт, який задає список опцій для вибору країни:

var codes = {

// телефонні коди в форматі "код країни": "назва"

"37": "Білорусь",

"38": "Україна",

// ..,

"1": "США"

};

Тут ми припускаємо, що більшість відвідувачів з Білорусі, і тому починаємо з 37, це залежить від проекту.

При виборі телефонного коду ми хотіли б пропонувати варіанти, починаючи з першого. Зазвичай на основі списку генерується select, але тут нам важливо не це, а важливо інше.

**Чи правда, що при переборі** **for(key in codes)** **ключі** **key** **будуть перераховані саме в тому порядку, в якому задані?**

**За стандартом - немає. Але деяке правило про це, все ж, є.**

Правило каже, що якщо ім'я властивості - нечислових рядок, то такі ключі завжди перебираються в тому ж порядку, в якому присвоювалися. Так вийшло з історичних причин і змінити це складно: поламається багато готового коду.

З іншого боку, якщо ім'я властивості - число або числовий рядок, то всі сучасні браузери сортують такі властивості з метою внутрішньої оптимізації.

Наприклад, розглянемо об'єкт зі свідомо нечисловими властивостями:

var user = {

name: "Вася",

surname: "Петренко"

};

user.age = 25;

// порядок перебору відповідає порядку присвоєння властивості

for (var prop in user) {

alert( prop ); // name, surname, age

}

А тепер - що буде, якщо перебрати об'єкт з кодами?

var codes = {

// телефонні коди в форматі "код країни": "назва"

"37": "Білорусь",

"38": "Україна",

"1": "США"

};

for (var code in codes) alert( code ); // 1, 37, 38

При запуску цього коду в сучасному браузері ми побачимо, що на перше місце потрапив код США!

Порушення порядку виникло, тому що ключі чисельні. Інтерпретатор JavaScript бачить, що рядок насправді є числом і перетворює ключ в трохи інший внутрішній формат. Додатковим ефектом внутрішніх оптимізацій є сортування.

**А що, якщо ми хочемо, щоб порядок був саме таким, який ми задали?**

Це можливо. Можна застосувати невеликий хак, який полягає в тому, щоб зробити всі ключі нечисловими, наприклад, додамо в початок додатковий символ '+':

var codes = {

"+37": "Білорусь",

"+38": "Україна",

"+1": "США"

};

for (var code in codes) {

var value = codes[code];

code = +code; // ..якщо нам потрібно саме число, перетворимо: "+37" -> 37

alert( code + ": " + value ); // 37, 38, 1 у всіх браузерах

}

**Висновок:**

* Цикл по ключам: for (key in obj).
* Порядок перебору відповідає порядку оголошення для нечислових ключів, а числові - упорядковано (в сучасних браузерах).
* Якщо потрібно, щоб порядок перебору числових ключів відповідав їх оголошенню в об'єкті, то використовують трюк: числові ключі замінюють на схожі, але містять не тільки цифри. Наприклад, додають в початок +, як описано в прикладі вище, а потім, в процесі обробки, перетворять такі ключі в число.

**3. Передача об’єктів за посиланням.**

Фундаментальним відзнакою об'єктів від примітивів, є їх зберігання і копіювання «за посиланням».

**Копіювання за значенням**

Звичайні значення: рядки, числа, булеві значення, null/undefined при присвоєнні змінних копіюються цілком або, як кажуть, «за значенням».

var message = "Привіт!";

var phrase = message;

В результаті такого копіювання вийшли дві повністю незалежні змінні, в кожній з яких зберігається значення "Привіт!".



**Копіювання за посиланням**

З об'єктами - все не так.

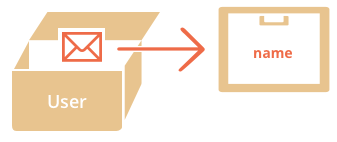
**В змінній, якій присвоєно об'єкт, зберігається не сам об'єкт, а «адреса його місця в пам'яті», іншими словами - «посилання» на нього.**

Ось як виглядає змінна, якій присвоєно об'єкт:

var user = {

name: "Вася"

};



Увага: об'єкт - поза змінної. В змінній - лише «адреса» (посилання) для нього.

При копіюванні змінної з об'єктом - копіюється ця посилання, а об'єкт як і раніше залишається в єдиному екземплярі.

Наприклад:

var user = { name: "Вася" }; // в змінній - посилання

var admin = user; // скопіювали посилання

Отримали дві змінні, в яких знаходяться посилання на один і той же об'єкт:



**Так як об'єкт всього один, то зміни через будь-яку змінну видно в інших змінних:**

var user = { name: 'Вася' };

var admin = user;

admin.name = 'Петя'; // змінили данні через admin

alert(user.name); // 'Петя', зміни видимі в user

**Змінна з об'єктом як «ключ» до сейфу з даними**

Ще одна аналогія: змінна, в яку присвоєно об'єкт, насправді зберігає не самі дані, а ключ до сейфа, де вони зберігаються.

При копіюванні її, виходить що ми зробили копію ключа, але сейф і раніше один.

**Клонування об'єктів**

Іноді, на практиці - дуже рідко, потрібно скопіювати об'єкт цілком, створити саме повну незалежну копію, «клон» об'єкта.

Що ж, можна зробити і це. Для цього потрібно пройти по об'єкту, дістати дані і скопіювати на рівні примітивів.

Приблизно так:

var user = {

name: "Вася",

age: 30

};

var clone = {}; // новий пустий об’єкт

// скопіюємо в нього всі властивості user

for (var key in user) {

clone[key] = user[key];

}

// тепер clone - повністю незалежна копія

clone.name = "Петя"; // змінили дані в clone

alert( user.name ); // як і раніше "Вася"

У цьому коді кожна властивість об'єкта user копіюється в clone. Якщо припустити, що вони примітивні, то кожна копіюється по значенню і ми як раз отримаємо повний клон.

Якщо ж властивості об'єктів, в свою чергу, можуть зберігати посилання на інші об'єкти, то потрібно перебрати такі підоб'єкти і теж склонувати їх. Це називають «глибоким» клонуванням.

**Висновок:**

* Об'єкт присвоюється і копіюється «за посиланням». Тобто, в змінній зберігається не сам об'єкт а, умовно кажучи, адреса в пам'яті, де він знаходиться.
* Якщо змінна-об'єкт скопійована або передана в функцію, то копіюється саме це посилання, а об'єкт залишається один в пам'яті.

Це - одне з ключових відмінностей об'єкта від примітиву (числа, рядки ...), який при присвоєнні якраз копіюється «за значенням», тобто повністю.