**ООП в JavaScript**

**1. Об’єкти**

Об'єкти в JavaScript поєднують в собі два важливих функціонали. Перший - це асоціативний масив: структура, придатна для зберігання будь-яких даних; другий - можливості для об'єктно-орієнтованого програмування.

Порожній об'єкт може бути оголошено за допомогою одного з двох синтаксисів: за допомогою конструктора та за допомогою літералу.

**var змінна = new Object (); // за допомогою конструктора**

**var змінна = {}; // за допомогою літералу**

Об'єкт можна заповнити значеннями при створенні, вказавши їх у фігурних дужках:

**{властивість1: значення1, властивість2: значення2, ... властивістьN: значенняN}**

// Приклад

var user = {

name: "Іван",

surname: "Петров",

age: 25

};

Доступ до властивостей (полів) здійснюється за іменем властивості. Для звертання до властивостей використовується уточнюючий запис через оператор «крапка»: об'єкт.властивість. Ім'я властивості при уточнюючому записі повинне бути правильним ідентифікатором.

Також існує альтернативний синтаксис звертання до властивості за індексом: об'єкт['властивість']. Звертання user['name'] еквівалентне user.name. Як індекс можна використати змінну: об'єкт[property]. Змінна property повинна містити ім'я властивості. При звертанні через індекс можна використовувати будь-який рядок. Якщо використано значення іншого типу, то воно буде автоматично приведене (перетворено) до рядка.

Звертання до властивості через оператор «крапка» використовується, якщо ми на етапі написання програми вже знаємо назву властивості. А якщо назва властивості буде визначена тільки в ході виконання програми, наприклад, введена користувачем і записана в змінну, то єдиний вихід звертання за індексом.

Основні операції над об'єктами:

Присвоєння значення властивості виконується як звичайне присвоєння:

**об’єкт.властивість = значення;**

Якщо в об’єкта дана властивість вже є, то змінюється її значення, якщо ж властивості немає, то вона додається. В результаті виконання наступного коду створюються два ідентичні об’єкти.

var test1 = {field1:123, field2:true},

var test2 = {};

test2.field1 = 123;

test2['field2'] = true;

Видалення властивостей за іменем здійснюється за допомогою спеціального оператора **delete**:

**delete об’єкт.властивість**

// Приклад

delete test2.field2;

Якщо такої властивості в об’єкта нема, то оператор не робить нічого.

Перевірка існування властивості з певним іменем здійснюється за допомогою оператора **in**. Його синтаксис наступний:

**"властивість" in об’єкт**

// Приклад

var person = {};

if ("name" in person) {

alert ("Властивість ім’я існує!");

}

Також можна використовувати інший спосіб - порівняння значення з undefined. В JavaScript можна звернутися до будь-якої властивості об'єкта, навіть якщо її не існує. Якщо властивість не існує, то повернеться значення undefined. Порівняння з undefined не працює, якщо значення властивості рівне undefined. А оператор in гарантує завжди правильний результат.

Для перебору всіх властивостей об'єкта використовується цикл за властивостями:

**for (властивість in об’єкт) {**

**// що робити з об’єкт[властивість] ...**

**}**

При оголошенні об'єкта можна вказати властивість-функцію, наприклад:

var user = {

   name: 'Василь',

   // Метод

  sayHi: function () {

     alert ('Привіт!');

   }

};

// Виклик методу

user.sayHi();

Властивості-функції називають *методами* об'єктів. Їх можна додавати і видаляти в будь-який момент, в тому числі і явним присвоєнням:

var user = {

   name: 'Василь'

};

user.sayHi = function () {

   alert('Привіт!');

};

// Виклик методу

user.sayHi();

Для повноцінної роботи метод повинен мати доступ до даних об'єкта. Зокрема, метод user.sayHi() може вивести ім'я користувача. Для доступу до об'єкта з методу використовується ключове слово **this**. Значення this називається *контекстом виклику* і буде визначено в момент виклику функції. Значенням this є об'єкт, в контексті якого викликаний метод, наприклад:

var user = {

   name: 'Василь',

   sayHi: function () {

    alert (this.name);

   }

};

// Виклик методу

user.sayHi();

Тут при виклику функції user.sayHi() в this буде зберігатися посилання на поточний об'єкт user.

В даному випадку замість this можна було б і використовувати і змінну: alert(user.name), але об'єкт user може бути кудись переданий, змінна user перезаписана і т.п. Використання this гарантує, що функція працює саме з тим об'єктом, в контексті якого викликана.

**Методи для управління властивостями об’єкта**

Метод Object.defineProperty дозволяє оголосити властивість об’єкта та налаштувати його особливі аспекти. Це основний метод для управління властивостями у JavaScript.

Синтаксис:

**Object.defineProperty(obj, prop, descriptor)**

Аргументи:

**obj** – об’єкт, для якого оголошується властивість

**prop** – Ім'я властивості, яке потрібно оголосити або модифікувати

**descriptor** – об’єкт, який описує поведінку властивості та який може містити наступні поля:

1. **value** - значення властивості (за замовчуванням undefined). Наприклад:

var user = {};

// 1. простое присваивание

user.name = "Вася";

// 2. указание значения через дескриптор

Object.defineProperty(user, "name", { value: "Вася" });

1. **writable** - значення властивості можна змінювати, якщо true (за замовчуванням false). Наприклад:

var user = {};

Object.defineProperty(user, "name", {

value: "Вася",

writable: false, // запретить присвоение "user.name="

});

1. **configurable** - якщо true, то властивість можна видаляти, а також змінювати його в подальшому за допомогою нових викликів **defineProperty** (за замовчуванням false). Наприклад:

var user = {};

Object.defineProperty(user, "name", {

value: "Вася",

configurable: false // заборонити видалення "delete user.name"

});

1. **enumerable** - якщо true, то властивість братиме участь в переборі for..in. За замовчуванням false. Наприклад:

var user = {

name: "Вася",

toString: function() { return this.name; }

};

// відмічаємо toString як такий, що не підлягає перебору в for..in

Object.defineProperty(user, "toString", {enumerable: false});

// Приклад використання

for(var key in user) alert(key); // name

1. **get** - функція, яка повертає значення властивості. За замовчуванням undefined. Наприклад:

var user = {

firstName: "Вася",

surname: "Петров"

};

Object.defineProperty(user, "fullName", {

get: function() {

return this.firstName + ' ' + this.surname;

}

});

// Приклад використання

alert(user.fullName); // Вася Петров

1. **set** - функція, яка записує значення властивості. За замовчуванням undefined. Наприклад:

var user = {

firstName: "Вася",

surname: "Петров"

};

Object.defineProperty(user, "fullName", {

set: function(value) {

let split = value.split(' ');

this.firstName = split[0];

this.surname = split[1];

}

});

// Приклад використання

user.fullName = "Петя Іванов";

alert( user.firstName ); // Петя

alert( user.surname ); // Іванов

***Зауваження***. JavaScript дозволяє також визначати геттери та сеттери безпосередньо у літералах. Наприклад:

var user = {

firstName: "Вася",

surname: "Петров",

get fullName() {

return this.firstName + ' ' + this.surname;

},

set fullName(value) {

var split = value.split(' ');

this.firstName = split[0];

this.surname = split[1];

}

};

// Приклад використання

alert(user.fullName); // Вася Петров (із геттера)

user.fullName = "Петя Іванов";

alert(user.firstName); // Петя (задав сеттер)

alert(user.surname); // Іванов (задав сеттер)

**2. Конструктори**

Оголошення об’єкта за допомогою літералу {...} дозволяє створити один об'єкт. Але зазвичай потрібно створювати багато однотипних об'єктів. Для цього використовують функції, викликаючи їх за допомогою спеціального оператора **new**. У JavaScript конструктором стає будь-яка функція, викликана через оператор **new**.

Наприклад:

function Animal(name) {

this.name = name; // властивість, для подання назви тваринки

}

var animal = new Animal("їжачок");

В результаті виклику new Animal("їжачок"); отримуємо такий об'єкт:

animal = { name: "їжачок" }

Технічно, будь-яку функцію можна викликати за допомогою оператора new. Але при цьому вона працює дещо іншим чином.

Алгоритм роботи функції, запущеної через оператор **new**:

1. Автоматично створюється новий порожній об'єкт.
2. Ключове слово **this** отримує посилання на цей об'єкт.
3. Функція виконується. Як правило, вона модифікує **this**, додає методи, властивості.
4. Повертається **this**.

Так, наприклад, при виклику new Animal відбувається наступне:

function Animal(name) {

// this = {}

// в this записуються поля, властивості та методи

this.name = name;

// return this

}

Як правило, конструктори нічого не повертають. Їх завдання - записати все, що потрібно, в **this**, який автоматично стане результатом. У JavaScript дозволено явний виклик **return**. Такі функції-конструктори називаються *фабричними*. У фабричних функціях-конструкторах застосовується просте правило:

* при виклику **return** з об'єктом, буде повернуто цей об’єкт, а не **this**;
* при виклику **return** з примітивним значенням, воно буде відкинуте.

Наприклад:

function BigAnimal() {

this.name = "Миша";

return { name: "Годзила" }; // повертаємо об’єкт

}

alert( new BigAnimal().name ); // Годзила // отримали об’єкт замість this

function BigAnimal() {

this.name = "Миша";

return "Годзила"; // повертаємо примітив

}

alert( new BigAnimal().name ); // Миша // отримали this

Окрім властивостей конструктор може містити також методи.

Наприклад, new User(name) створює об'єкт із заданим значенням властивості name і методом sayHi:

function User(name) {

this.name = name;

this.sayHi = function() {

alert( "Моє ім’я: " + this.name );

};

}

// Використання

var ivan = new User("Іван");

ivan.sayHi(); // Моє ім’я: Іван

**Поліморфні конструктори та фабричні конструктори**

У JavaScript відсутній поліморфізм функцій. Це означає, що не можна оголошувати декілька функцій із однаковим іменем. Натомість у JavaScript будь-яка функція може бути викликана із довільними фактичними параметрами. Таким чином функції у JavaScript в деякій мірі поліморфні відносно параметрів.

Тому у випадку, коли об’єкт потрібно створювати різними способами, одним зі способів реалізувати таку можливість у JavaScript є оголошення поліморфної функції-конструктора із параметрами, які будуть по різному оброблятися.

Наприклад:

function User(name, age) {

if (arguments.length == 0) { // конструктора без параметрів

this.name = name || 'Анонім';

} else {

if (arguments.length == 1 && arguments[0] instanceof User) { // конструктор копіювання

this.name = arguments[0].name;

this.age = arguments[0].age;

}

else { // конструктор з параметрами

this.name = name;

this.age = age;

}

}

this.sayHi = function () {

alert(this.name);

};

}

// Використання

var guest = new User();

guest.sayHi(); // Анонім

var vasil = new User('Вася', 25);

vasil.sayHi(); // Вася

var teska = new User(vasil);

teska.sayHi(); // Вася

У JavaScript, як правило, оголошують конструктор із одним параметром - об’єктом. Наприклад:

function User(userData) {

if (userData) {

this.name = userData.name;

this.age = userData.age;

} else {

this.name = 'Анонім';

}

this.sayHi = function () {

alert(this.name)

};

}

// Використання

var guest = new User();

guest.sayHi(); // Анонім

var knownUser = new User({ name: 'Вася', age: 25 });

knownUser.sayHi(); // Вася

**Статичні поля та методи. Фабричні статичні функції**

Поля та методи, які не прив'язані до конкретного екземпляру класу, називають «статичними». У JavaScript їх записують безпосередньо в сам конструктор.

Наприклад у коді нижче використовуються статичні поля Article.count і Article.DEFAULT\_FORMAT:

function Article() {

Article.count++;

}

Article.count = 0; // статичне поле‐змінна

Article.DEFAULT\_FORMAT = "html"; // статичне поле‐константа

// використання

var article1 = new Article();

var article2 = new Article();

alert(Article.count); // 2

Аналогічним чином у JavaScript визначаються і статичні методи. У JavaScript у статичних методах дозволяється використання ключового слова this. При цьому слід пам’ятати, що у статичному методі ключове слово this відповідає не якомусь конкретному об’єкту, а класу загалом. Наприклад створимо для Article статичний метод Article.showCount():

Article.showCount = function() {

alert( this.count ); // тут this.count те саме що й Article.count

}

// використання

new Article();

new Article();

Article.showCount(); // 2

Статичні методи можна використовувати як альтернативу поліморфним конструкторам для того, щоб уникнути розбору формальних параметрів. При цьому підході замість розбору формальних параметрів у конструкторі потрібно для кожного випадку створювати відповідний статичний метод.

Наприклад:

function User() {

this.sayHi = function () {

alert(this.name)

};

}

User.createAnonymous = function () { // аналог конструктора без параметрів

var user = new User();

user.name = 'Анонім';

return user;

}

User.createFromData = function (userData) { // аналог конструктора з параметрами

var user = new User();

user.name = userData.name;

user.age = userData.age;

return user;

}

// Використання

var guest = User.createAnonymous();

guest.sayHi(); // Анонім

var knownUser = User.createFromData({ name: 'Вася', age: 25 });

knownUser.sayHi(); // Вася

**3. Інкапсуляція**

Оголошені у конструкторі локальні змінні і локальні вкладені функції доступні тільки в контексті (всередині, *private*) функції-конструктора, а ті властивості і методи, які оголошені в контексті виклику (тобто із ключовим словом this) доступні на рівні об’єкта (ззовні, *public*).

**4. Успадкування на прототипах**

**Прототип об’єкта. Класи на прототипах**