**============================= ТЕОРІЯ ==============================**

**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’єкти призначені для опису властивостей та функціональних можливостей сутностей, з якими працюємо у програмі.

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

**{ властивість1: значення1,**

**властивість2: значення2,**

**...**

**властивістьN: значенняN**

**}**

Приклад

|  |  |
| --- | --- |
| Опис користувача | Програмна реалізація |
| Властивості   * ім’я * прізвище * вік | var user = {  name: "Іван",  surname: "Петров",  age: 25  }; |

Приклад

|  |  |
| --- | --- |
| Опис учня | Програмна реалізація |
| Властивості   * Ім’я * Клас * Оцінки з 3-х предметів | var pupil = {  name: "Ivan",  class\_: "8B",  marks: [10, 11, 12],  } |

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.

//Опис 1 (одразу вказуємо властивості і їх значення)

var test1 = {

**field1**:123,

**field2**:true

}

//Опис 2 (спочатку об’єкт порожній, а вже потім додаємо властивості

var test1 = {}; 🡨 створення порожнього об’єкту

test1.**field1** = 123; 🡨 додали нову властивість **field1**

test1[**'field2'**] = true; 🡨 додали нову властивість **field2**

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

**delete об’єкт.властивість**

// Приклад

**delete** test1.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 = {

name:”Ivan”

};

if ("name" in person) { 🡨 "name" in person == true (бо властивість "name" існує)

alert ("Властивість ім’я існує!");

}

if ("age" in person) { 🡨 "age" in person == false (бо властивість "age" не існує)

alert ("Властивість вік 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] ...**

**}**

Приклад

var user = {

name: "Іван",

surname: "Петров",

age: 25

};

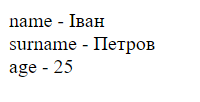
for (var **propertyName** in user) {

document.write(**propertyName** + " - " + user[**propertyName**]);

document.write("<br>");

}

Результат



Задача. Для рядка тексту вивести на екран кількість входжень кожної літери (побудувати об’єкт, у якому поля – літери, а значення – кількість входжень).

function getDiffLetters(text) {

var diffLetters = { };

for (var i = 0; i < text.length; i++) {

if (text[i] in diffLetters) {

diffLetters[text[i]]++;

}

else {

diffLetters[text[i]] = 1; }

}

return diffLetters;

}

//-----------------

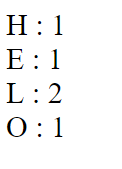
var letters = getDiffLetters("HELLO");

for (var c in letters) {

document.write(c + " : " + letters[c]);

document.write("<br>");

}



***Методи об’єктів***

Об’єкти реальної дійсності характеризуються не тільки деякими властивостями а й функціональними можливостями (діями, що може виконати сам об'єкт, або можна виконати над об’єктом). Функціональні можливості об’єкта описують за допомогою методів (властивостей – функцій).

Приклад.

|  |  |
| --- | --- |
|  |  |
| Властивості   * ім’я * прізвище * вік   **Функціональні можливості**   * **вивести привітання** * **попрощатися** | var user = {  //------ Властивості ------  name: "Іван",  surname: "Петров",  age: 25,  //------- Методи ------  sayHi: function () {  alert('Привіт!');  },  sayBye: function () {  alert('До побачення!');  }  }; |

Виклик методів здійснюється як і властивостей за допомогою оператора крапка «.»

|  |  |
| --- | --- |
| Загальний вигляд | Приклад |
| Об’єкт . метод(*список фактичних параметрів*) | user.sayHi();  user.sayBye(); |

Властивості-функції (*методи*) об'єктів можна додавати і видаляти в будь-який момент, в тому числі і явним присвоєнням:

//---- Опис об’єкта ------

var user = {

   name: 'Василь'

};

//------ Додавання методу -------

user.sayHi = function () {

   alert('Привіт!');

};

user[‘satHi’]= 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**);

   }

};

user.name=’Ivan’

var s=12;

// 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.

Приклад.

|  |  |
| --- | --- |
| Опис учня | Програмна реалізація |
| Властивості   * Ім’я * Клас * Оцінки з 3-х предметів   Методи (дії)   * Знаходження середнього * Перетворення у string | var pupil = {  name: "Ivan",  class\_: "8B",  marks: [10, 11, 12],    toString: function () {  return this.name + ", " +  this.marks + ", оцінки:" +  this.marks;  },  getAverage: function () {  var s = 0;  for (var i = 0; i < this.marks.length; i++)  {  s += this.marks[i];  }  return s / this.marks.length;  } |

**-----------------------------------------------------------------------------**

**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:

var 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

Приклад. Прямокутник

<script>

function Rect(a, b) {

//---- Унікальні (у кожного об'єкта вони свої)

this.a = a;

this.b = b;

}

//------ Статичні поля (спільні для усіх)

Rect.Scale = 10;

//------- Властивості-методи

Rect.prototype.S = function () {

return this.a \* this.b \* Rect.Scale;

}

//---------------

var obj = new Rect(2,3);

document.write("s=" + obj.S());

Rect.c = 6;

var obj2 = new Rect(1, 5);

document.write("<br>s2=" + obj2.S());

//-----------------

Rect.prototype.P = function () {

return 2 \* (this.a + this.b);

}

//--------------------

document.write("<br>p2=" + obj2.P());

=========================== ПРАКТИКА =======================

Задача. Створити об’єкт «Діапазон»

|  |  |
| --- | --- |
| Діапазон |  |
| Поля (властивості) | Нижня межа  Верхня межа |
| Методи (дії) | Перевірка того, чи належить діапазону  Генерування випадкового числа з вказаного діапазону |

Задача. Створити об’єкт «Клієнт»

|  |  |
| --- | --- |
| Клієнт |  |
| Поля(властивості) | ПІБ  Номер рахунку  Кількість коштів |
| Методи (дії) | Зняття грошей з рахунку  Додавання грошей на рахунок |

Задача. Створити об’єкт «Гральний кубик»

|  |  |
| --- | --- |
| Гральний кубик |  |
| Поля(властивості) | Кількість граней |
| Методи (дії) | Перевірка того, чи належить діапазону  Генерування випадкового числа з вказаного діапазону |

<script>

var gameCube = {

//------ Властивості-поля ------

edgesCount:6,

//------- Властивості-Методи ------

isNumberCorrect: function (number) {

return (number >= 1 && number <= this.edgesCount);

},

getRandomNumber: function () {

return 1 + Math.floor(Math.random() \* 6);

}

};

//------------------

var score1 = gameCube.getRandomNumber();

var score2 = gameCube.getRandomNumber();

document.write(`${score1} - ${score2}`);

</script>

//------------------------------------- Реалізації з використанням функції-конструктора

//===================================

function Dice(faceNumber,attemptNumber) {

//------ Всередині функції-коструктора описуємо поля

this.faceNumber=faceNumber

this.attemptNumber=attemptNumber

}

//----------- Методи додаємо у prototype

Dice.prototype.toString=function(){

return `faceN=${this.faceNumber} , attemptN=${this.attemptNumber}`

}

Dice.prototype.getRandomNumber=function(){

return 1+Math.floor(Math.random()\*this.faceNumber)

}

Dice.prototype.getScore=function(){

let s=0;

for (let i = 0; i < this.attemptNumber; i++) {

s+=this.getRandomNumber()

}

return s/this.attemptNumber

}

//------------------------------

let dice= new Dice(6,3)

let score1 = dice.getScore()

document.write(`Score1 = ${score1}`)

Задача. Створити об’єкт «Рулетка»

|  |  |
| --- | --- |
| Рулетка |  |
| Поля(властивості) | Масив, у якому зберігаються бали ігрового барабану |
| Методи (дії) | Метод для випадкового визначення кількості балів  Виведення ігрового барабану на екран (у формі таблиці) |

Задача. Створити об’єкт «Тир». У масиві зберігаються 1, якщо у цьому квадраті є заєць і 0 в іншому випадку.

|  |  |
| --- | --- |
| Тир |  |
| Поля(властивості) | Масив, у якому зберігається поле з зайцями |
| Методи (дії) | Метод пострілу (задається позиція пострілу)  Виведення ігрового поля |

Задача. Створити об’єкт «Авто».

|  |  |
| --- | --- |
| Авто |  |
| Поля(властивості) | Марка  Розмір бака  Кількість наявних літрів  Кількість місць  Кількість пасажирів |
| Методи (дії) | Заправка на вказану кількість літрів  Виведення кількості пасажирів  Додавання пасажирів  Висадка пасажирів |