Функції-конструктори

№ уроку:	7	**Курс:**	JavaScript Базовий

Засоби навчання:	Visual Studio Code Web Browser

Огляд, мета та призначення уроку

Вивчити способи створення безліч об'єктів з однаковою структурою з використанням конструкторів. Вивчити механізми успадкування через прототипи. Навчитися виносити методи об'єктів у прототипи.

Вивчивши матеріал цього заняття, учень зможе:

- Створювати та використовувати функції-конструктори.
- Працювати з прототипами для ефективного використання оперативної пам'яті.
- Розуміти принципи успадкування через прототипи.
- Використовувати hasOwnProperty та ключове слово instanceof.

Зміст уроку

1. Функція-фабрика.
2. Функція-конструктор.
3. Робота з прототипами.
4. For...in та прототипи.
5. Ключове слово instanceof.

Резюме

- Для створення об'єктів у JavaScript не потрібне визначення типу даних. Щоб створювати безліч об'єктів з однаковою структурою, можна застосовувати **функції-конструктори** або **функції-фабрики**.

- **Функція-фабрика** – функція, яка створює, конфігурує та повертає об'єкт. Водночас якщо створити кілька об'єктів через одну й ту ж функцію, структура об'єктів буде однаковою. Такий підхід дає змогу позбутися дублювання коду й об'єднати всі операції, які пов'язані з конфігурацією об'єкта в одній функції. Одним з головних недоліків такого підходу є те, що за наявності методів в об'єктах, які створюються, усі методи будуть дублюватися в кожному екземплярі та займати більше оперативної пам'яті.

- **Функція-конструктор** – функція для створення та ініціалізації об'єктів з однаковою структурою.
  За угодою функції мають іменуватися з великої літери, а під час виклику функції перед функцією-конструктором необхідно використовувати ключове слово **new**.

- Функцію-конструктор можна розглядати як аналог класів у суворо типізованих мовах програмування. Для створення нового «типу даних» у JavaScript використовуються функції-конструктори (починаючи з версії ECMAScript 2015, можна використовувати ключове слово **class**, яке докладніше розглядатиметься на наступному уроці).

CyberBionic Systematics ®
19 Evgenia Sverstiuka Str., 5 floor
Kyiv, Ukraine

Tel.:0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

- У тілі функції-конструктора робота має бути орієнтована те що, щоб задати структуру новому об'єкту. Доступ до нового об'єкта відбувається через ключове слово **this**. Наприкінці значення this неявно повертається.

- Ключове слово **new** перед викликом конструктора відповідає за створення нового порожнього об'єкта та встановлення цього об'єкта як контекст для функції-конструктора (контекст функції – значення ключового слова this). Якщо викликати функцію-конструктор без ключового слова this, то контекстом функції буде глобальний об'єкт або undefined (якщо код виконується в суворому режимі (strict mode)). Також у тілі конструктора можна визначити додаткові перевірки, щоб конструктор спрацьовував коректно, навіть якщо був викликаний без new. Якщо функція іменується з великої літери (угоди для назви конструкторів), то буде хорошою практикою запустити цю функцію з ключовим словом new, навіть якщо функція підтримує виклик без цього ключового слова.

- Кожна функція JavaScript пов'язана з об'єктом – **прототипом**. Наприклад, function Test() {} має прототип. Щоб отримати доступ до нього, потрібно звернутися до властивості Test.prototype.
  Прототип використовується під час створення нового об'єкта, ключове слово new після створення об'єкта пов'язує цей об'єкт з прототипом тієї функції, яка вказується після ключового слова new.
  У такий спосіб якщо за допомогою конструктора створити, наприклад, 10 екземплярів, усі вони будуть пов'язані з одним і тим же прототипом.

- Під час використання оператора «.» для отримання доступу до членів об'єкта відбувається пошук цього члена в об'єкті. Якщо пошук не дасть результату, то він триває у зв'язаному прототипі. Якщо прототип має свій прототип, то пошук може продовжуватися далі.
  Загальною практикою є **винесення у прототип методів**. Оскільки методи не змінюють свою роботу від об'єкта до об'єкта, який створений через конструктор, немає сенсу витрачати оперативну пам'ять для зберігання методів у кожному об'єкті. Метод виноситься в прототип. Коли на об'єкті відбувається виклик цього методу, то інтерпретатор після неуспішної спроби знайти метод в об'єкті переходить у прототип і викликає метод на прототипі.

- Якщо звернутися до екземпляра та записати властивість, яка вже є в прототипі, то вона з'являється лише у вказаному екземплярі, а прототип залишається без змін. Отримати доступ до прототипу можна лише через операції читання. Операції запису змінюватимуть екземпляр, з яким йде взаємодія.

- **Instanceof** – ключове слово для перевірки наявності в ланцюжку прототипів вказаної функції. Приклад використання: variable instanceof MyConstructor, якщо змінна з іменем variable була створена через конструктор MyConstructor, то ключове слово поверне значення true, інакше false.

Закріплення матеріалу

- Що таке функція-фабрика?
- Що таке функція-конструктор?
- Що станеться, якщо запустити конструктор без ключового слова new?
- Які є угоди щодо іменування конструктора?
- За що відповідає ключове слово new під час використання його спільно з конструкторами?
- Що таке прототип? Опишіть сценарії використання прототипу.
- Для чого використовується ключове слово instanceof?

CyberBionic systematics

CyberBionic Systematics ®
19 Evgenia Sverstiuka Str., 5 floor
Kyiv, Ukraine

Tel.:0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

Самостійна діяльність учня

Виконайте завдання у директорії Tasks\010 Constructor Function. Текст завдань розташований у коментарях, у тегах script.

Рекомендовані ресурси

Функції-конструктори
https://www.w3schools.com/js/js_object_constructors.asp

Прототипи об'єктів
https://developer.mozilla.org/ru/docs/Learn/JavaScript/Objects/Object_prototypes

CyberBionic Systematics ®
19 Evgenia Sverstiuka Str., 5 floor
Kyiv, Ukraine

Tel.:0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com