Замикання

№ уроку:	11	**Курс:**	JavaScript Базовий

Засоби навчання:	Visual Studio Code Web Browser

Огляд, мета та призначення уроку

Навчитися використовувати замикання, зрозуміти внутрішній механізм роботи замикань, який ґрунтується на лексичному оточенні.

Вивчивши матеріал цього заняття, учень зможе:

- Працювати з globalthis.
- Розуміти, що таке контекст виконання та стек виклику.
- Розуміти, що таке лексичне оточення та як воно використовується під час роздільної здатності ідентифікаторів.
- Використовувати замикання та розуміти їхній внутрішній механізм роботи.

Зміст уроку

1. Глобальний об'єкт.
2. Контекст виконання (Execution Context).
3. Лексичне оточення (Lexical Environment).
4. Замикання (Closures).

Резюме

- Код, який виконується на JavaScript, розташований у контексті виконання глобального об'єкта, який у браузері представлений об'єктом windows, а в Node.JS – об'єктом global.

  Під час написання сценаріїв, які покладаються на глобальний об'єкт і працюватимуть у різних середовищах виконання, для посилання на глобальний об'єкт можна використовувати змінну **globalThis. globalthis** – стандартизоване ім'я глобального об'єкта.
- Змінні, які створені за допомогою **var** і функції, визначені в **script**, стають властивостями глобального об'єкта. Якщо в сценарії необхідно створити ідентифікатор, який доступний у будь-якій частині поточного сценарію або в інших сценаріях, такий ідентифікатор потрібно зробити властивістю глобального об'єкта.
- Варто уникати глобальних змін і функцій, оскільки можуть виникнути конфлікти, які пов'язані з тим, що інші сценарії почнуть визначати глобальні змінні або функції з таким самим ім'ям, як створені раніше, що може спричинити непередбачувану поведінку.
- **Контекст виконання** – спеціальний механізм, який використовується для опису оточення, в якому виконується JavaScript-код (для збереження оголошених змінних, this та інших важливих станів для роботи коду). Будь-який виконуваний код виконується в контексті виконання.
- **Стек виконання (execution stack)** – зберігає всі контексти виконання, які були створені у процесі роботи коду. Коли функція запускається, її контекст виконання додається в стек.

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

Коли функція завершує свою роботу, її контекст видаляється зі стека.

- **Можна виділити три типи контекстів виконання:**
  - **Global Execution Context** – створюється один раз під час запуску сценарію.
  - **Functional Execution Context** – створюється під час кожного запуску функції.
  - **Eval Function Execution Context** – код, який виконується через eval.

- **Контекст виконання створюється у два етапи:**
  1. **Creation Phase** – створюються та ініціалізуються компоненти контексту.
  2. **Execution Phase** – виконується код.

- **Контекст виконання містить:**
  - **Лексичне оточення (LexicalEnvironment)** – для визначення, який ідентифікатор з яким значенням зв'язаний.
  - **Оточення змінних (VariableEnvironment)** – для змінних var.

- **Лексичне оточення містить:**
  - **EnvironemtRecord** – містить визначення змінних і функцій.
  - **OuterEnv** – зовнішнє оточення на основі стека виклику.
  - **ThisBinding** – контекст (this) виконуваного коду, який пов'язаний із цим лексичним оточенням.
- **Замикання (closure)** – це функція та лексичне оточення, в якому ця функція була створена.

```
function makeCounter() {
        let counter = 0;

        function increment() {return counter += 1;
        }
    return increment;
}
```

Під час запуску функції makeCounter буде створено лексичне оточення, в якому буде розташована змінна counter. Кожна функція містить внутрішнє посилання на лексичне оточення, у якому вона була створена. Коли функція increment повертається з функції makeCounter, лексичне оточення makeCounter не може бути видалено з оперативної пам'яті, бо на нього посилається функція increment.

Якщо повторно запустити функцію makeCounter, буде створено нове лексичне оточення. Нова функція increment буде містити в собі посилання на це оточення. Функція та пов'язані з нею лексичні оточення є замиканням.

Замикання можуть бути причиною більшої витрати пам'яті, якщо в замиканні зберігається досить великий ланцюжок лексичних оточень.

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

- Замикання часто використовуються під час реалізації різних шаблонів кодування у JavaScript. Наприклад, для реалізації шаблону модуль – для створення об'єкта з відкритими та закритими властивостями й методами.

- **IIFE** (inmidiatly invoked function expression) – функція, яка запускається відразу після оголошення. Прийом, який дає змогу визначити блок коду, в якому створені ідентифікатори будуть локальними, а не глобальними, оскільки, створюючи змінну функції, вона розташована в лексичному оточенні цієї функції. IIFE широко використовувалися в попередніх версіях JavaScript до введення модулів та let і давала змогу легко контролювати кількість створюваних глобальних змінних.

Закріплення матеріалу

- Що таке globalThis?
- Що таке контекст виконання? Чим він відрізняється від контексту функції?
- Що таке лексичне оточення?
- Що таке IIFE? Навіщо використовується ця конструкція?
- Що таке замикання? Опишіть принцип роботи замикань.

Самостійна діяльність учня

Виконайте завдання у директорії Exercises\Tasks\014 Closures. Текст завдань розташований у коментарях, у тегах script.

Рекомендовані ресурси

ECMAScript 2022 Language Specification. Executable Code and Execution Context
https://tc39.es/ecma262/#sec-executable-code-and-execution-contexts

Замикання
https://developer.mozilla.org/ru/docs/Web/JavaScript/Closures

IIFE
https://developer.mozilla.org/ru/docs/Glossary/IIFE

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