hillel IT school

Lesson 9

Трохи кодінгу

1. Дано масив з елементами різних типів. Створити функцію яка вираховує середнє арифметичне лише числових елементів даного масиву.
2. Написати функцію заповнення даними користувача двомірного масиву. Довжину основного масиву і внутрішніх масивів задає користувач. Значення всіх елементів всіх масивів задає користувач.
3. Створити функцію, яка прибирає з рядка всі символи, які ми передали другим аргументом. 'func(" hello world", ['l', 'd'])' поверне нам "heo wor". Вихідний рядок та символи для видалення задає користувач.

План заняття

- Стрілочні функції;
- Відмінності стрілочних функцій від звичайних;
- Функція вищого порядку;
- Функція-коллбек;
- Стек викликів;
- Рекурсія;

Arrow function

Існує ще один дуже простий та лаконічний синтаксис для створення функцій, який часто краще, ніж Function Expression

Синтаксис:

```
const func = (arg1, arg2, ...argN) => expression;
const double = n => n * 2;
const sayHi = () => alert("Hello!");

const sum = (a, b) => {
  const result = a - b;
  return result;
}
```

Arrow function

1. У стрілочних функцій немає this. Якщо відбувається звернення до цього, його значення береться зовні.
2. Arrow function не можна використовувати з new.
3. Arrow function не можуть бути використані як конструктори. Вони не можуть бути викликані з new.
4. Arrow function не мають «arguments»

Higher-Order Function

Що таке функціональне програмування?

Якщо просто, то це підхід до програмування, при використанні якого функції можна передавати іншим функціям як параметри і використовувати функції ролі значень, що повертаються іншими функціями. Займаючись функціональним програмуванням, ми проектуємо архітектуру програми та пишемо код з використанням функцій.

У JavaScript, як і в інших мовах, що підтримують функціональне програмування, функція є об'єктами. Функції можна передавати як параметри іншим функціям. Такі функції, передані іншим, зазвичай виступають ролі функцій зворотного виклику (коллбеков). Функції можна призначати змінним, зберігати в масивах, тощо. Саме тому функції JS — це об'єкти першого класу.

Функцією вищого порядку називається така функція, яка отримує функції як аргумент або повертає функцію у вигляді вихідного значення. Наприклад, вбудовані функції JavaScript Array.prototype.map, Array.prototype.filter та Array.prototype.reduce є функціями вищого порядку.

Рекурсія

Рекурсія – це паттерн, який є корисним у ситуаціях, коли завдання може бути розділена на кілька завдань того ж роду, але простіших. Або коли завдання може бути спрощене до простої дії плюс простіший варіант того ж завдання. Або, щоб працювати з певними структурами даних.

Коли функція викликає саму себе - це називається рекурсія.

Пам’ятайте що будь яку рекурсію можна замінити на цикли, але це не означає простіше рішення!

У рекурсії завжди є:

1. База - якась умова виходу;
2. Крок - дія яка виконується з даними, та виклик самої себе. Наступні кроки спрощують його;
3. Глибина рекурсії - максимальна кількість вкладених викликів (включаючи перший);

Максимальна глибина рекурсії обмежена рушієм JavaScript. Ми можемо покластися, що вона може дорівнювати 10000, деякі рушії дозволяють отримати більшу глибину, але 100000, ймовірно, не підтримується більшістю з них.

Контекст виконання

Інформація про процес виконання викликаної функції зберігається у контексті виконання.

Контекст виконання – це внутрішня структура даних, яка містить деталі про виконання функції: де зараз керуючий потік, поточні змінні, значення this і кілька інших внутрішніх деталей.

Один виклик функції має рівно один контекст виконання, пов’язаний з ним.

Коли функція робить вкладений виклик, відбувається наступне:

- Поточна функція зупиняється.
- Контекст виконання, пов’язаний з нею, запам’ятовується в спеціальній структурі даних, що називається стек контекстів виконання.
- Вкладений виклик виконується.
- Після закінчення, старий контекст виконання витягується з стека, і зовнішня функція відновлюється з того місця, де вона зупинилася.

Контекст виконання

Щоб зробити вкладений виклик, JavaScript пам’ятає контекст поточного виконання в стеці контексту виконання.

Тобто стек викликів заповнюється так:

- Контекст n
- Контекст 2
- Контекст 1

Вихід

Коли у базі рекурсії ми отримаємо true (немає більше вкладених викликів) то ми вже маємо усі результати викликів і ми їдемо у зворотньому напрямку. Оскільки функція завершується, то контекст виконання більше не потрібний, тому він видаляється з пам’яті. Попередній контекст відновлюється з вершини стека:

- значення - операція - з результатом фунції
- x * результат pow(x, n - 1)
- x * результат pow(x, n - 1)

Зберігання контекстів потребує пам’яті.

Рекурсивний обхід

Коли у базі рекурсії ми отримаємо true (немає більше вкладених викликів) то ми вже маємо усі результати викликів і ми їдемо у зворотньому напрямку. Оскільки функція завершується, то контекст виконання більше не потрібний, тому він видаляється з пам’яті. Попередній контекст відновлюється з вершини стека:

- значення - операція - з результатом фунції
- x * результат pow(x, n - 1)
- x * результат pow(x, n - 1)

Зберігання контекстів потребує пам’яті.

Дякую за увагу