Zaawansowany Programista JS

Dzień 3

Plan

- Struktury Danych
  - Tablica obiektów
  - Obiekt obiektów
  - Zamiana obiektu obiektów na tablicę obiektów przy uzyciu `Object.keys()` i `Object.values()`
- Praca z localStorage
  - serializacja danych do formatu JSON przy uzyciu `JSON.parse()` i `JSON.stringify()`
  - praca z localStorage (dodawanie, usuwanie, edytowanie)
- Tworzenie aplikacji z wykorzystaniem formularzy, listy i localStorage

---

Struktury Danych

Struktury danych to sposoby przechowywania i organizowania danych w programowaniu. W JavaScript istnieje kilka rodzajów struktur danych, a każda z nich ma swoje własne cechy i zastosowania.

Tablice (Arrays) - są to kolekcje wartości uporządkowanej sekwencji. Elementy w tablicy są indeksowane od 0 i są przechowywane w porządku, w którym zostały dodane. Można dodawać, usuwać i modyfikować elementy w tablicy za pomocą różnych metod, takich jak push, pop, shift, unshift, splice, itp.

Obiekty (Objects) - są to kolekcje kluczy i wartości. Klucze są ciągami znaków, a wartości mogą być dowolnymi typami danych JavaScript, w tym tablicami i innymi obiektami. Można dodawać, usuwać i modyfikować pary klucz-wartość w obiekcie za pomocą różnych metod, takich jak Object.keys, Object.values, Object.entries, itp.

Struktury danych są bardzo ważne w programowaniu, ponieważ pozwalają na skuteczne zarządzanie danymi i wykonywanie różnych operacji. W JavaScript istnieją także biblioteki i narzędzia, które umożliwiają pracę z bardziej zaawansowanymi strukturami danych, takimi jak drzewa i grafy.

Tablica obiektów

Tablica obiektów to tablica, która przechowuje obiekty JavaScript jako swoje elementy. Każdy element w tablicy obiektów jest obiektem, który składa się z właściwości i metod.

Aby stworzyć tablicę obiektów w JavaScript, można po prostu utworzyć nową tablicę i przypisać do niej wiele obiektów za pomocą operatora []. Przykładowo, jeśli chcemy utworzyć tablicę obiektów reprezentujących użytkowników, możemy to zrobić w następujący sposób:

```
let users = [
  { name: "Jan", age: 25, city: "Warszawa" },
  { name: "Anna", age: 30, city: "Kraków" },
  { name: "Piotr", age: 40, city: "Gdańsk" }
];
```

W tym przykładzie, users jest tablicą, która składa się z trzech obiektów, z których każdy reprezentuje użytkownika i ma trzy właściwości: name, age i city. Możemy uzyskać dostęp do poszczególnych obiektów w tablicy, odwołując się do ich indeksów, np. `users[0]` zwróci pierwszy obiekt w tablicy, czyli `{ name: "Jan", age: 25, city: "Warszawa" }`.

Tablice obiektów są bardzo przydatne w programowaniu, ponieważ pozwalają na łatwe przechowywanie i przetwarzanie zbiorów danych, takich jak dane użytkowników, produkty w sklepie internetowym czy komentarze na stronie internetowej. Dzięki nim możemy łatwo sortować, filtrować i przeszukiwać duże ilości danych, co jest niezbędne w wielu aplikacjach internetowych i mobilnych.

Obiekt obiektów

Obiekt obiektów w JavaScript to obiekt, który zawiera inne obiekty jako swoje właściwości. Możemy go utworzyć, przypisując do właściwości obiektów inne obiekty lub tworząc je dynamicznie w trakcie działania programu.

Przykładowo, można utworzyć obiekt employees, który zawiera obiekty reprezentujące pracowników i ich dane, takie jak imię, nazwisko i stanowisko

```
let employees = {
  employee1: { firstName: "Jan", lastName: "Kowalski", position: "Programista" },
  employee2: { firstName: "Anna", lastName: "Nowak", position: "Projektant" },
  employee3: { firstName: "Piotr", lastName: "Mickiewicz", position: "Tester" }
};
```

W tym przykładzie, employees jest obiektem, który ma trzy właściwości: `employee1`, `employee2` i `employee3`, z których każda jest obiektem reprezentującym pracownika.

Możemy uzyskać dostęp do poszczególnych obiektów w obiekcie obiektów, odwołując się do ich właściwości za pomocą operatora . lub [], np. `employees.employee1` lub `employees['employee1']` zwróci obiekt reprezentujący pierwszego pracownika.

Obiekty obiektów są przydatne w programowaniu, gdy chcemy grupować obiekty w logiczne zbiory, takie jak lista pracowników w firmie czy produkty w sklepie internetowym. Dzięki nim możemy łatwo uzyskać dostęp do poszczególnych obiektów w kolekcji i wykonywać na nich różne operacje, takie jak sortowanie, filtrowanie czy przeszukiwanie.

Zamiana obiektu obiektów na tablicę obiektów przy uzyciu `Object.keys()` i `Object.values()`

Aby zamienić obiekt obiektów na tablicę obiektów w JavaScript, możemy użyć metody `Object.values()`, która zwraca wartości wszystkich właściwości obiektu jako tablicę. Możemy następnie przetworzyć tę tablicę i stworzyć z niej nową tablicę obiektów.

Przykładowo, jeśli mamy obiekt employees, który zawiera obiekty reprezentujące pracowników i ich dane, jak w poprzednim przykładzie, możemy przekształcić go na tablicę obiektów w następujący sposób

```
let employees = {
  employee1: { firstName: "Jan", lastName: "Kowalski", position: "Programista" },
  employee2: { firstName: "Anna", lastName: "Nowak", position: "Projektant" },
  employee3: { firstName: "Piotr", lastName: "Mickiewicz", position: "Tester" }
};

let employeesArray = Object.values(employees);
```

W tym przykładzie, `employeesArray` będzie tablicą, która składa się z trzech obiektów, z których każdy reprezentuje pracownika i ma trzy właściwości: `firstName`, `lastName` i `position`. Możemy uzyskać dostęp do poszczególnych obiektów w tablicy, odwołując się do ich indeksów, np. `employeesArray[0]` zwróci obiekt reprezentujący pierwszego pracownika.

Metoda `Object.keys()` działa podobnie jak `Object.values()`, ale zwraca tablicę zawierającą nazwy wszystkich właściwości obiektu. Można ją wykorzystać do uzyskania dostępu do poszczególnych obiektów w obiekcie obiektów za pomocą ich nazw.

Przykładowo, jeśli mamy obiekt `employees`, jak w poprzednim przykładzie, możemy wykorzystać metodę `Object.keys()` do uzyskania nazw wszystkich pracowników i iterować po nich, aby uzyskać dostęp do ich obiektów

```
let employees = {
  employee1: { firstName: "Jan", lastName: "Kowalski", position: "Programista" },
  employee2: { firstName: "Anna", lastName: "Nowak", position: "Projektant" },
  employee3: { firstName: "Piotr", lastName: "Mickiewicz", position: "Tester" }
};

let keys = Object.keys(employees);
for (let key in keys) {
  let employeeObject = employees[key];
  console.log(`${employeeObject.firstName} ${employeeObject.lastName}
(${employeeObject.position})`);
}
```

W tym przykładzie, keys będzie tablicą zawierającą trzy nazwy właściwości: employee1, employee2 i employee3. W pętli `for...in` iterujemy po tych nazwach, uzyskując dostęp do obiektów pracowników za pomocą ich nazw i wyświetlamy ich dane.

Praca z localStorage

`localStorage` to obiekt globalny dostępny w przeglądarkach internetowych, który pozwala na przechowywanie danych na dłuższy czas, nawet po zamknięciu przeglądarki i ponownym uruchomieniu komputera. Dane przechowywane w localStorage są dostępne dla wszystkich stron internetowych, które korzystają z tej samej domeny.

Serializacja danych do formatu JSON przy uzyciu `JSON.parse()` i `JSON.stringify()`

`JSON.parse()` służy do konwertowania tekstu w formacie JSON na obiekt JavaScript. Przykładowo, jeśli mamy tekst JSON reprezentujący obiekt person:

```
let jsonText = '{"name": "Jan Kowalski", "age": 30}';
// Możemy przekonwertować ten tekst na obiekt JavaScript za pomocą metody
JSON.parse():

let person = JSON.parse(jsonText);
console.log(person); // wyświetli obiekt { name: "Jan Kowalski", age: 30 }
```

`JSON.stringify()` służy do konwertowania obiektu JavaScript na tekst w formacie JSON. Przykładowo, jeśli mamy obiekt person

```
let person = { name: "Jan Kowalski", age: 30 };
```

Możemy przekonwertować ten obiekt na tekst w formacie JSON za pomocą metody `JSON.stringify()`:

```
let jsonText = JSON.stringify(person);
console.log(jsonText); // wyświetli tekst '{"name": "Jan Kowalski", "age": 30}'
```

Metody `JSON.parse()` i `JSON.stringify()` są szczególnie przydatne, gdy chcemy przesyłać dane między serwerem a przeglądarką w formacie JSON. Dzięki nim, możemy przesyłać obiekty JavaScript w formie tekstu i odbierać je z powrotem jako obiekty JavaScript.

Praca z localStorage (dodawanie, usuwanie, edytowanie)

localStorage działa w oparciu o pary klucz-wartość. Możemy używać metod `setItem()` i `getItem()` do zapisywania i odczytywania danych z localStorage.

Przykładowo, aby zapisać wartość do localStorage, możemy użyć metody `setItem()`

```
localStorage.setItem("name", "Jan Kowalski");
```

W tym przykładzie, zapisujemy wartość "Jan Kowalski" pod kluczem `name`. Możemy teraz odczytać tę wartość za pomocą metody `getItem()`

```
let name = localStorage.getItem("name");
console.log(name); // wyświetli "Jan Kowalski"
```

Możemy również usunąć wartość z localStorage za pomocą metody `removeItem()`

```
localStorage.removeItem("name");
```

W tym przykładzie, usuwamy wartość o kluczu `name` z localStorage.

Uwaga: localStorage przechowuje dane w postaci tekstowej, więc przed zapisaniem wartości, które nie są tekstowe (np. obiektów JavaScript), musimy je przekonwertować na tekst. Możemy to zrobić za pomocą metody `JSON.stringify()` przed zapisaniem i `JSON.parse()` po odczytaniu

```
let user = { name: "Jan Kowalski", age: 30 };
localStorage.setItem("user", JSON.stringify(user));


let storedUser = JSON.parse(localStorage.getItem("user"));
console.log(storedUser); // wyświetli obiekt { name: "Jan Kowalski", age: 30 }
```

Metody `setItem()`, `getItem()` i `removeItem()` są dostępne w większości współczesnych przeglądarek internetowych.

Tworzenie aplikacji z wykorzystaniem formularzy, listy i localStorage

```
Napisz Chat który będzie spełniał następujące zadania

1. W HTML stwórz formularz i listę
2. Formularz ma zawierać 2 pola input: author i message
3. Formularz w momencie wysłania powinien zwalidować czy pole author nie jest puste i
czy pole message nie jest krótsze niz 2 znaki
4. W momencie poprawnej walidacli formularza, dodaj element do listy
5. Wykorzystaj localStorage aby móc odczytywać i zapisywać dane do pamięci
przeglądarki.
```
