Javascript Πίνακες, Ουρές, Στοίβες

Επιμέλεια: Α.Δρακόπουλος

Πίνακες στην Javascript

Οι πίνακες (arrays) είναι ένα από τα πιο συχνά χρησιμοποιούμενα εργαλεία στη JavaScript για την αποθήκευση και διαχείριση πολλαπλών τιμών σε μία δομή. Ένας πίνακας είναι μια συλλογή από στοιχεία (τιμές), τα οποία αποθηκεύονται σε έναν ενιαίο χώρο μνήμης και κάθε στοιχείο έχει έναν δείκτη (index) που ξεκινά από το 0.

Η JavaScript προσφέρει μια σειρά από ενσωματωμένες μεθόδους και ιδιότητες που κάνουν τη διαχείριση των πινάκων πιο εύκολη και ευέλικτη. Παρακάτω παρουσιάζονται οι βασικές μέθοδοι και ιδιότητες των πινάκων, μαζί με παραδείγματα προγραμμάτων που δείχνουν τη χρήση τους.

1. Δημιουργία Πινάκων

Μπορούμε να δημιουργήσουμε έναν πίνακα με δύο βασικούς τρόπους:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Array Creation in JavaScript</title>
</head>
<body>

<script>
// Δημιουργία πίνακα χρησιμοποιώντας array literal.
let fruits = ["Apple", "Banana", "Mango"];

// Δημιουργία πίνακα χρησιμοποιώντας τον constructor Array.
let numbers = new Array(1, 2, 3, 4, 5);

console.log(fruits); // Εκτυπώνει ["Apple", "Banana", "Mango"]
console.log(numbers); // Εκτυπώνει [1, 2, 3, 4, 5]
</script>

</body>
</html>
```

2. Ιδιότητες Πινάκων

Η πιο βασική ιδιότητα των πινάκων είναι το length, το οποίο επιστρέφει τον αριθμό των στοιχείων του πίνακα.

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Array Length Property in JavaScript</title>
</head>
<body>

<script>
// Πίνακας με φρούτα.
let fruits = ["Apple", "Banana", "Mango"];

console.log(fruits.length); // Εκτυπώνει 3
</script>

</body>
</html>
```

3. Πρόσβαση σε Στοιχεία Πίνακα

Η πρόσβαση σε στοιχεία πίνακα γίνεται με βάση τον δείκτη (index).

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Accessing Array Elements in JavaScript</title>
</head>
<body>

<script>
// Πίνακας με αριθμούς.
let numbers = [10, 20, 30, 40, 50];

console.log(numbers[0]); // Εκτυπώνει 10 (πρώτο στοιχείο)
console.log(numbers[3]); // Εκτυπώνει 40 (τέταρτο στοιχείο)
</script>
```

```
</body>
</html>
```

4. push() και pop()

push(): Προσθέτει ένα ή περισσότερα στοιχεία στο τέλος του πίνακα.

pop(): Αφαιρεί το τελευταίο στοιχείο από τον πίνακα.

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Array Push and Pop in JavaScript</title>
</head>
<body>

<script>
let numbers = [1, 2, 3];

// Προσθήκη στοιχείων με την push().
numbers.push(4);
numbers.push(5);

console.log(numbers); // Εκτυπώνει [1, 2, 3, 4, 5]

// Αφαίρεση του τελευταίου στοιχείου με την pop().
numbers.pop();

console.log(numbers); // Εκτυπώνει [1, 2, 3, 4]
</script>

</body>
</html>
```

5. shift() και unshift()

shift(): Αφαιρεί το πρώτο στοιχείο από τον πίνακα.

unshift(): Προσθέτει ένα ή περισσότερα στοιχεία στην αρχή του πίνακα.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Array Shift and Unshift in JavaScript</title>
</head>
<body>

<script>
let fruits = ["Banana", "Mango", "Apple"];

// Αφαίρεση του πρώτου στοιχείου με την shift().
fruits.shift();

console.log(fruits); // Εκτυπώνει ["Mango", "Apple"]

// Προσθήκη νέου στοιχείου στην αρχή με την unshift().
fruits.unshift("Strawberry");

console.log(fruits); // Εκτυπώνει ["Strawberry", "Mango", "Apple"]
</script>

</body>
</html>
```

6. concat()

Η μέθοδος concat() χρησιμοποιείται για να συγχωνεύσουμε δύο ή περισσότερους πίνακες.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Array Concat in JavaScript</title>
</head>
```

```
<body>

<script>
let fruits = ["Apple", "Banana"];
let moreFruits = ["Mango", "Orange"];

// Συγχώνευση των δύο πινάκων.
let allFruits = fruits.concat(moreFruits);

console.log(allFruits); // Εκτυπώνει ["Apple", "Banana", "Mango", "Orange"]
</script>

</body>
</html>
```

7. slice() και splice()

slice(): Επιστρέφει ένα νέο πίνακα που περιέχει μέρος του αρχικού πίνακα.

splice(): Προσθέτει, αφαιρεί ή αντικαθιστά στοιχεία στον πίνακα.

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Array Slice and Splice in JavaScript</title>
</head>
<body>

<script>
let numbers = [1, 2, 3, 4, 5];

// Χρήση της slice() για να πάρουμε μέρος του πίνακα.
let slicedNumbers = numbers.slice(1, 3);
console.log(slicedNumbers); // Εκτυπώνει [2, 3]

// Χρήση της splice() για να αφαιρέσουμε στοιχεία και να προσθέσουμε νέα.
numbers.splice(2, 1, 10); // Αφαιρεί 1 στοιχείο στη θέση 2 και προσθέτει το 10.
console.log(numbers); // Εκτυπώνει [1, 2, 10, 4, 5]
</script>
```

```
</body>
</html>
```

8. forEach()

Η forEach() εκτελεί μια συνάρτηση για κάθε στοιχείο του πίνακα.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Array forEach in JavaScript</title>
</head>
<body>

<script>
let fruits = ["Apple", "Banana", "Mango"];

// Εκτέλεση της συνάρτησης για κάθε στοιχείο του πίνακα.
fruits.forEach(function(fruit) {
    console.log(fruit);
});
// Εκτυπώνει "Apple", "Banana", "Mango"
</script>

</body>
</html>
```

9. map()

Η μέθοδος map() δημιουργεί έναν νέο πίνακα, εφαρμόζοντας μια συνάρτηση σε κάθε στοιχείο του πίνακα.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
    <title>Array Map in JavaScript</title>
</head>
<body>

<script>
let numbers = [1, 2, 3, 4, 5];

// Χρήση της map() για να δημιουργήσουμε έναν νέο πίνακα με διπλασιασμένα στοιχεία.
let doubledNumbers = numbers.map(function(number) {
    return number * 2;
});

console.log(doubledNumbers); // Εκτυπώνει [2, 4, 6, 8, 10]
</

</script>

</body>
</html>
```

10. filter()

Η μέθοδος filter() δημιουργεί έναν νέο πίνακα με όλα τα στοιχεία που πληρούν μια συνθήκη που καθορίζεται από μια συνάρτηση.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Array Filter in JavaScript</title>
</head>
<body>

<script>
let numbers = [1, 2, 3, 4, 5, 6];

// Χρήση της filter() για να πάρουμε τους αριθμούς που είναι μεγαλύτεροι από 3.
let filteredNumbers = numbers.filter(function(number) {
    return number > 3;
});
```

```
console.log(filteredNumbers); // Εκτυπώνει [4, 5, 6]
</script>

</body>
</html>
```

11. reduce()

Η μέθοδος reduce() εφαρμόζει μια συνάρτηση σε κάθε στοιχείο του πίνακα (από αριστερά προς τα δεξιά), ώστε να το "μειώσει" σε μια μόνο τιμή.

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Array Reduce in JavaScript</title>
</head>
<body>

<script>
let numbers = [1, 2, 3, 4, 5];

// Χρήση της reduce() για να βρούμε το άθροισμα όλων των στοιχείων του πίνακα.
let sum = numbers.reduce(function(total, current) {
   return total + current;
}, 0);

console.log(sum); // Εκτυπώνει 15
</script>

</body>
</html>
```

12. find() και findIndex()

find(): Επιστρέφει το πρώτο στοιχείο που πληροί μια συνθήκη.

findIndex(): Επιστρέφει τον δείκτη του πρώτου στοιχείου που πληροί μια συνθήκη.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Array Find and FindIndex in JavaScript</title>
</head>
<body>

<script>
let numbers = [5, 12, 8, 130, 44];

// Χρήση της find() για να βρούμε τον πρώτο αριθμό μεγαλύτερο από 10.
let found = numbers.find(function(number) {
    return number > 10;
});

console.log(found); // Εκτυπώνει 12

// Χρήση της findIndex() για να βρούμε τον δείκτη του πρώτου αριθμού μεγαλύτερου από 10.
let foundIndex = numbers.findIndex(function(number) {
    return number > 10;
});

console.log(foundIndex); // Εκτυπώνει 1
</script>

</body>
</html>
```

13. some() και every()

some(): Ελέγχει αν τουλάχιστον ένα στοιχείο του πίνακα πληροί μια συνθήκη.

every(): Ελέγχει αν όλα τα στοιχεία του πίνακα πληρούν μια συνθήκη.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
```

```
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Array Some and Every in JavaScript</title>
</head>
<body>

<script>
let numbers = [1, 2, 3, 4, 5];

// Χρήση της some() για να ελέγξουμε αν υπάρχει τουλάχιστον ένας αριθμός μεγαλύτερος από 4.
let someResult = numbers.some(function(number) {
    return number > 4;
});
console.log(someResult); // Εκτυπώνει true

// Χρήση της every() για να ελέγξουμε αν όλοι οι αριθμοί είναι μικρότεροι από 6.
let everyResult = numbers.every(function(number) {
    return number < 6;
});
console.log(everyResult); // Εκτυπώνει true
</script>

</body>
</html>
```

14. sort()

Η sort() ταξινομεί τα στοιχεία του πίνακα είτε αλφαβητικά είτε αριθμητικά. Για την αριθμητική ταξινόμηση, πρέπει να δώσουμε μια συνάρτηση σύγκρισης.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Array Sort in JavaScript</title>
</head>
<body>

<script>
let numbers = [40, 100, 1, 5, 25, 10];

// Ταξινόμηση σε αύξουσα σειρά με συνάρτηση σύγκρισης.
```

```
numbers.sort(function(a, b) {
   return a - b;
});

console.log(numbers); // Εκτυπώνει [1, 5, 10, 25, 40, 100]
</script>

</body>
</html>
```

15. reverse()

Η reverse() αντιστρέφει τη σειρά των στοιχείων του πίνακα.

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Array Reverse in JavaScript</title>
</head>
<body>

<script>
let numbers = [1, 2, 3, 4, 5];

// Αντιστροφή της σειράς των στοιχείων.
numbers.reverse();

console.log(numbers); // Εκτυπώνει [5, 4, 3, 2, 1]
</script>

</body>
</html>
```

Συμπέρασμα

Οι πίνακες στη JavaScript αποτελούν έναν από τους πιο χρήσιμους τύπους δεδομένων και προσφέρουν μια ευρεία γκάμα μεθόδων για τη διαχείρισή τους. Οι παραπάνω μέθοδοι καλύπτουν τις πιο βασικές λειτουργίες, όπως η προσθήκη, αφαίρεση, ταξινόμηση και

φιλτράρισμα των στοιχείων ενός πίνακα. Με τη σωστή χρήση αυτών των μεθόδων, μπορούμε να διαχειριστούμε τα δεδομένα αποτελεσματικά και να υλοποιήσουμε σύνθετους αλγόριθμους και λειτουργίες.

Queue σε JavaScript

Για να υλοποιήσουμε μια ουρά (queue) σε JavaScript που θα εκτελείται μέσα από έναν browser, θα πρέπει να χρησιμοποιήσουμε τις δομές δεδομένων που παρέχει η JavaScript, όπως τα arrays (πίνακες), καθώς και HTML για να αλληλεπιδράσουμε με τον χρήστη. Ακολουθεί ένα παράδειγμα ολοκληρωμένου προγράμματος με αναλυτικά σχόλια και περιγραφή της λειτουργίας.

Βήματα υλοποίησης:

1. HTML για τη δημιουργία ενός interface (φορμών και κουμπιών) ώστε ο χρήστης να μπορεί να προσθέτει και να αφαιρεί στοιχεία από τη ουρά.

2. JavaScript για την υλοποίηση των λειτουργιών της ουράς (enqueue, dequeue) και για την ενημέρωση του περιεχομένου της ιστοσελίδας.

3. CSS για βασική μορφοποίηση της σελίδας (προαιρετικά).

Ολοκληρωμένο Πρόγραμμα

HTML (index.html)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Queue Implementation in JavaScript</title>
    <style>
        body {
            font-family: Arial, sans-serif;
        }
        .queue-container {
```

```
            width: 300px;
            margin: 0 auto;
            padding: 20px;
            border: 1px solid #ccc;
            border-radius: 5px;
            text-align: center;
        }
        .queue-display {
            margin: 10px 0;
            padding: 10px;
            border: 1px solid #000;
            min-height: 50px;
        }
        button {
            padding: 10px;
            margin: 5px;
        }
    </style>
</head>
<body>

<div class="queue-container">
    <h2>Queue Implementation</h2>

    <!-- Input field to add an element to the queue -->
    <input type="text" id="queue-input" placeholder="Enter element" />
    <button onclick="enqueue()">Enqueue</button>
    <button onclick="dequeue()">Dequeue</button>

    <h3>Queue Elements:</h3>
    <!-- Div to display the current state of the queue -->
    <div id="queue-display" class="queue-display">Queue is empty</div>
</div>

<script src="queue.js"></script>
</body>
</html>
```

JavaScript (queue.js)

```
// Δημιουργούμε μια κλάση Queue για να διαχειριστούμε τη ουρά.
class Queue {
    constructor() {
```

```
        this.items = []; // Αρχικοποιούμε έναν κενό πίνακα για τα στοιχεία της ουράς.
    }

    // Προσθήκη στοιχείου στη ουρά (enqueue).
    enqueue(element) {
        this.items.push(element); // Τοποθετούμε το νέο στοιχείο στο τέλος της ουράς.
    }

    // Αφαίρεση στοιχείου από τη ουρά (dequeue).
    dequeue() {
        if(this.isEmpty()) {
            return "Queue is empty"; // Επιστρέφει μήνυμα αν η ουρά είναι άδεια.
        }
        return this.items.shift(); // Αφαιρούμε το πρώτο στοιχείο της ουράς.
    }

    // Έλεγχος αν η ουρά είναι άδεια.
    isEmpty() {
        return this.items.length === 0; // Επιστρέφει true αν η ουρά δεν έχει στοιχεία.
    }

    // Επιστρέφει όλα τα στοιχεία της ουράς σε μορφή συμβολοσειράς.
    printQueue() {
        if(this.isEmpty()) {
            return "Queue is empty"; // Αν δεν υπάρχουν στοιχεία, επιστρέφουμε κατάλληλο μήνυμα.
        }
        return this.items.join(" -> "); // Επιστρέφουμε τα στοιχεία χωρισμένα με το βέλος.
    }
}

// Δημιουργούμε μια νέα ουρά (queue).
const queue = new Queue();

// Συνάρτηση για να προσθέσουμε ένα στοιχείο στη ουρά μέσω της διεπαφής του χρήστη.
function enqueue() {
    const inputElement = document.getElementById("queue-input");
    const element = inputElement.value; // Λαμβάνουμε το στοιχείο από το πεδίο εισόδου.

    if(element) {
        queue.enqueue(element); // Προσθέτουμε το στοιχείο στη ουρά.
        inputElement.value = ""; // Καθαρίζουμε το πεδίο εισόδου μετά την προσθήκη.
    }

    displayQueue(); // Ενημερώνουμε την εμφάνιση της ουράς.
```

```
}

// Συνάρτηση για να αφαιρέσουμε το πρώτο στοιχείο από τη ουρά μέσω της διεπαφής.
function dequeue() {
    queue.dequeue(); // Αφαιρούμε το πρώτο στοιχείο από τη ουρά.
    displayQueue(); // Ενημερώνουμε την εμφάνιση της ουράς.
}

// Συνάρτηση που εμφανίζει τα στοιχεία της ουράς στην οθόνη.
function displayQueue() {
    const displayElement = document.getElementById("queue-display");
    displayElement.textContent = queue.printQueue(); // Ενημερώνουμε το div με το περιεχόμενο της ουράς.
}
```

Περιγραφή της υλοποίησης

1. HTML: Η HTML δημιουργεί το περιβάλλον χρήστη όπου ο χρήστης μπορεί να εισάγει στοιχεία μέσω ενός input πεδίου και να τα προσθέτει στη ουρά (enqueue) ή να αφαιρεί στοιχεία (dequeue). Επίσης, υπάρχει ένα div όπου θα εμφανίζονται τα στοιχεία της ουράς.

2. CSS: Η βασική μορφοποίηση της σελίδας γίνεται με χρήση CSS για να φαίνεται καλύτερα η διαμόρφωση του περιβάλλοντος χρήστη.

3. JavaScript:

Η κλάση Queue υλοποιεί τη βασική δομή δεδομένων ουράς με τις εξής μεθόδους:

enqueue(element): Προσθέτει ένα στοιχείο στη ουρά.

dequeue(): Αφαιρεί το πρώτο στοιχείο από τη ουρά.

isEmpty(): Ελέγχει αν η ουρά είναι άδεια.

printQueue(): Επιστρέφει τα στοιχεία της ουράς σε συμβολοσειρά για εμφάνιση.

Οι συναρτήσεις enqueue και dequeue ενημερώνουν το περιεχόμενο της ουράς και εμφανίζουν τα στοιχεία στο div με id queue-display.

Τρόπος λειτουργίας

1. Ο χρήστης πληκτρολογεί ένα στοιχείο στο πεδίο εισόδου και πατά το κουμπί Enqueue για να το προσθέσει στη ουρά.

2. Όταν πατηθεί το κουμπί Dequeue, αφαιρείται το πρώτο στοιχείο που είχε προστεθεί στη ουρά.

3. Σε κάθε ενέργεια, η ουρά εμφανίζεται οπτικά στην οθόνη με τα στοιχεία της διατεταγμένα.

Αυτό το πρόγραμμα είναι ένα απλό και κατανοητό παράδειγμα για τη λειτουργία μιας ουράς και πώς αυτή μπορεί να αλληλεπιδράσει με τον χρήστη σε ένα περιβάλλον browser.

Stack σε JavaScript

Για να υλοποιήσουμε μια στοίβα (stack) σε JavaScript και να αλληλεπιδρούμε με αυτήν μέσω ενός περιβάλλοντος browser, θα χρησιμοποιήσουμε την ίδια προσέγγιση όπως με τη ουρά (queue). Δηλαδή, θα δημιουργήσουμε μια δομή Stack σε JavaScript και θα παρέχουμε ένα απλό HTML interface για την προσθήκη (push) και αφαίρεση (pop) στοιχείων από τη στοίβα.

Ολοκληρωμένο Πρόγραμμα

HTML (index.html)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Stack Implementation in JavaScript</title>
    <style>
        body {
            font-family: Arial, sans-serif;
        }
        .stack-container {
            width: 300px;
            margin: 0 auto;
            padding: 20px;
            border: 1px solid #ccc;
            border-radius: 5px;
            text-align: center;
        }
        .stack-display {
            margin: 10px 0;
            padding: 10px;
            border: 1px solid #000;
            min-height: 50px;
            max-height: 200px;
            overflow-y: auto;
        }
        button {
            padding: 10px;
```

```
            margin: 5px;
        }
    </style>
</head>
<body>

<div class="stack-container">
    <h2>Stack Implementation</h2>

    <!-- Input field to add an element to the stack -->
    <input type="text" id="stack-input" placeholder="Enter element" />
    <button onclick="push()">Push</button>
    <button onclick="pop()">Pop</button>

    <h3>Stack Elements:</h3>
    <!-- Div to display the current state of the stack -->
    <div id="stack-display" class="stack-display">Stack is empty</div>
</div>

<script src="stack.js"></script>
</body>
</html>
```

JavaScript (stack.js)

```
// Δημιουργούμε μια κλάση Stack για να διαχειριστούμε τη στοίβα.
class Stack {
    constructor() {
        this.items = []; // Αρχικοποιούμε έναν κενό πίνακα για τα στοιχεία της στοίβας.
    }

    // Προσθήκη στοιχείου στη στοίβα (push).
    push(element) {
        this.items.push(element); // Προσθέτουμε το στοιχείο στο τέλος του πίνακα (στην κορυφή
της στοίβας).
    }

    // Αφαίρεση του στοιχείου από την κορυφή της στοίβας (pop).
    pop() {
        if (this.isEmpty()) {
            return "Stack is empty"; // Επιστρέφει μήνυμα αν η στοίβα είναι άδεια.
        }
```

```
        return this.items.pop(); // Αφαιρούμε το τελευταίο στοιχείο από τον πίνακα (κορυφή της στοίβας).
    }

    // Επιστροφή του στοιχείου στην κορυφή της στοίβας χωρίς αφαίρεση (peek).
    peek() {
        if (this.isEmpty()) {
            return "Stack is empty"; // Επιστρέφει μήνυμα αν η στοίβα είναι άδεια.
        }
        return this.items[this.items.length - 1]; // Επιστρέφει το τελευταίο στοιχείο του πίνακα.
    }

    // Έλεγχος αν η στοίβα είναι άδεια.
    isEmpty() {
        return this.items.length === 0; // Επιστρέφει true αν η στοίβα είναι άδεια.
    }

    // Επιστροφή όλων των στοιχείων της στοίβας σε μορφή συμβολοσειράς.
    printStack() {
        if (this.isEmpty()) {
            return "Stack is empty"; // Αν η στοίβα είναι άδεια, επιστρέφουμε το κατάλληλο μήνυμα.
        }
        return this.items.join(" -> "); // Επιστρέφει τα στοιχεία της στοίβας χωρισμένα με βέλη.
    }
}

// Δημιουργούμε μια νέα στοίβα (stack).
const stack = new Stack();

// Συνάρτηση για να προσθέσουμε ένα στοιχείο στη στοίβα μέσω της διεπαφής του χρήστη.
function push() {
    const inputElement = document.getElementById("stack-input");
    const element = inputElement.value; // Λαμβάνουμε το στοιχείο από το πεδίο εισόδου.

    if (element) {
        stack.push(element); // Προσθέτουμε το στοιχείο στη στοίβα.
        inputElement.value = ""; // Καθαρίζουμε το πεδίο εισόδου μετά την προσθήκη.
    }

    displayStack(); // Ενημερώνουμε την εμφάνιση της στοίβας.
}

// Συνάρτηση για να αφαιρέσουμε το στοιχείο από την κορυφή της στοίβας μέσω της διεπαφής.
function pop() {
```

```
    stack.pop(); // Αφαιρούμε το στοιχείο από την κορυφή της στοίβας.
    displayStack(); // Ενημερώνουμε την εμφάνιση της στοίβας.
}

// Συνάρτηση που εμφανίζει τα στοιχεία της στοίβας στην οθόνη.
function displayStack() {
    const displayElement = document.getElementById("stack-display");
    displayElement.textContent = stack.printStack(); // Ενημερώνουμε το div με το περιεχόμενο
της στοίβας.
}
```

Περιγραφή της υλοποίησης

1. HTML: Η HTML δομή παρέχει το interface όπου ο χρήστης μπορεί να εισάγει στοιχεία μέσω ενός input πεδίου και να τα προσθέτει στη στοίβα (push) ή να αφαιρεί τα στοιχεία από την κορυφή της στοίβας (pop). Ένα div με id stack-display χρησιμοποιείται για την εμφάνιση της τρέχουσας κατάστασης της στοίβας.

2. CSS: Το CSS προσθέτει βασική μορφοποίηση για να βελτιώσει την οπτική παρουσίαση του interface.

3. JavaScript:

Η κλάση Stack υλοποιεί τη βασική λειτουργικότητα της στοίβας με τις εξής μεθόδους:

push(element): Προσθέτει ένα στοιχείο στην κορυφή της στοίβας.

pop(): Αφαιρεί το στοιχείο από την κορυφή της στοίβας.

peek(): Επιστρέφει το στοιχείο στην κορυφή χωρίς να το αφαιρεί.

isEmpty(): Ελέγχει αν η στοίβα είναι άδεια.

printStack(): Επιστρέφει τα στοιχεία της στοίβας σε μορφή συμβολοσειράς για εμφάνιση.

Οι συναρτήσεις push και pop χρησιμοποιούνται για την προσθήκη και αφαίρεση στοιχείων από τη στοίβα αντίστοιχα, ενώ η συνάρτηση displayStack ενημερώνει το div με την τρέχουσα κατάσταση της στοίβας.

Παράδειγμα Λειτουργίας

1. Ο χρήστης εισάγει ένα στοιχείο στο πεδίο εισόδου και πατά το κουμπί Push για να το προσθέσει στη στοίβα. Το στοιχείο προστίθεται στην κορυφή της στοίβας.

2. Όταν πατηθεί το κουμπί Pop, αφαιρείται το στοιχείο που βρίσκεται στην κορυφή της στοίβας.

3. Σε κάθε ενέργεια, η στοίβα εμφανίζεται δυναμικά στο div stack-display με τα στοιχεία της διατεταγμένα από την βάση έως την κορυφή.

Συμπέρασμα

Αυτό το πρόγραμμα παρουσιάζει με απλό και κατανοητό τρόπο πώς μπορούμε να υλοποιήσουμε μια στοίβα (stack) σε JavaScript και να αλληλεπιδρούμε με αυτήν μέσω ενός περιβάλλοντος browser. Είναι κατάλληλο για επίδειξη της λειτουργίας της στοίβας και μπορεί να επεκταθεί περαιτέρω για πιο σύνθετες εφαρμογές.