Gültige Klammern

Aufgabenbeschreibung

Gegeben ist ein String `s`, der nur die Zeichen '(', ')', '{', '}', '[' und ']' enthält. Bestimme, ob der Eingabe-String gültig ist.

Ein Eingabe-String ist gültig, wenn:

1. Offene Klammern müssen in der richtigen Reihenfolge geschlossen werden.
2. Jede schließende Klammer muss das gleiche Typ haben wie ihre zuletzt geöffnete, nicht geschlossene Klammer.

```
const isValidBrackets = (s) => {


}
```

Beispiele

```
Eingabe: s = "()"
Ausgabe: true
```

```
Eingabe: s = "()[]{}"
Ausgabe: true
```

```
Eingabe: s = "(]"
Ausgabe: false
```

```
Eingabe: s = "([)]"
Ausgabe: false
```

```
Eingabe: s = "{[]}"
Ausgabe: true
```

```

```

```
Eingabe: s = "((()))"
Ausgabe: true
```

```
Eingabe: s = "{[()]}"
Ausgabe: true
```

```
Eingabe: s = "({[]})"
Ausgabe: true
```

```
Eingabe: s = "((("
Ausgabe: false
```

```
Eingabe: s = "{[([]{([({})])[{[()]}]}){([[])[{}})({()})}][({{}))[{[]}]]}
{[([{}])][(())]}{({{}})}([[]){{}})"
Ausgabe: true
```

Einschränkungen

- 1 <= s.length <= 10^4
- s besteht nur aus Klammern '()[]{}'.

Lösungsvorschlag

Dieser Lösungsvorschlag verwendet einen Stack und funktioniert wie folgt:

- Ein leerer Stack wird erstellt, um öffnende Klammern zu speichern.
- Die Eingabezeichenkette wird Zeichen für Zeichen durchlaufen.
- Wenn ein öffnendes Klammersymbol gefunden wird ('(', '{', '['), wird es auf den Stack gelegt.
- Bei einem schließenden Klammersymbol (')', '}', ']'):
  - Wenn der Stack leer ist, wird sofort `false` zurückgegeben, da eine schließende Klammer ohne entsprechende öffnende Klammer ungültig ist.
  - Andernfalls wird das oberste Element vom Stack entfernt.
  - Es wird geprüft, ob die entfernte öffnende Klammer zur aktuellen schließenden Klammer passt.
  - Wenn sie nicht passen, wird `false` zurückgegeben, da die Klammerung ungültig ist.
- Nach der Verarbeitung aller Zeichen wird überprüft, ob der Stack leer ist.
- Ein leerer Stack bedeutet, dass alle öffnenden Klammern geschlossen wurden.

Das musst lernen

Für diese Aufgabe verwendest du am besten einen `Stack`, allerdings bietet JavaScript, von Haus aus, keinen integrierten Stack an. Ein Stack zu erstellen ist aber nicht schwer. Schau dir die folgende Erklärung und Implementierung eines Stacks an. Die Implementierung kannst Du natürlich verwenden.

Ein Stack ist eine lineare Datenstruktur, die nach dem Prinzip "Last In, First Out" (LIFO) arbeitet. Das bedeutet, dass das zuletzt hinzugefügte Element als erstes entfernt wird, ähnlich wie bei einem Stapel von Büchern, bei dem das oberste Buch zuerst entfernt wird.

Eigenschaften und Methoden eines Stacks

Ein Stack hat folgende grundlegende Operationen:

- **Push**: Fügt ein Element oben auf den Stack.
- **Pop**: Entfernt das oberste Element vom Stack und gibt es zurück.
- **Peek**: Gibt das oberste Element des Stacks zurück, ohne es zu entfernen.
- **isEmpty**: Überprüft, ob der Stack leer ist.
- **size**: Gibt die Anzahl der Elemente im Stack zurück.
- **clear**: Entfernt alle Elemente aus dem Stack.

Implementierung eines Stacks in JavaScript

Hier ist ein Beispiel, wie man einen Stack in JavaScript implementieren kann:

```
class Stack {
  constructor() {
    // Ein Array für die Daten des Stacks
    this.items = [];
  }

  // Fügt ein Element oben auf den Stack
  push(element) {
    this.items.push(element);
  }

  // Entfernt das oberste Element vom Stack und gibt es zurück
  pop() {
    if (this.isEmpty()) {
      return 'Underflow';
    }
    return this.items.pop();
  }

  // Gibt das oberste Element des Stacks zurück, ohne es zu entfernen
  peek() {
    if (this.isEmpty()) {
      return 'No elements in Stack';
    }
    return this.items[this.items.length - 1];
  }

  // Überprüft, ob der Stack leer ist
```

```
  isEmpty() {
    return this.items.length === 0;
  }

  // Gibt die Anzahl der Elemente im Stack zurück
  size() {
    return this.items.length;
  }

  // Entfernt alle Elemente aus dem Stack
  clear() {
    this.items = [];
  }

  // Gibt den gesamten Stack als String zurück
  toString() {
    return this.items.join(" ");
  }
}

// Beispielverwendung
let stack = new Stack();
stack.push(10);
stack.push(20);
stack.push(30);
console.log(stack.toString());     // Ausgabe: 10 20 30
console.log(stack.peek());         // Ausgabe: 30
console.log(stack.pop());          // Ausgabe: 30
console.log(stack.toString());     // Ausgabe: 10 20
console.log(stack.isEmpty());      // Ausgabe: false
console.log(stack.size());         // Ausgabe: 2
stack.clear();
console.log(stack.toString());     // Ausgabe: (leere Zeichenkette)
console.log(stack.isEmpty());      // Ausgabe: true
```

Erklärung

- **Konstruktor**: Initialisiert den Stack mit einem leeren Array `items`.
- **push(element)**: Fügt ein neues Element oben auf den Stack.
- **pop()**: Entfernt und gibt das oberste Element zurück. Gibt `Underflow` zurück, wenn der Stack leer ist.
- **peek()**: Gibt das oberste Element zurück, ohne es zu entfernen. Gibt eine Nachricht zurück, wenn der Stack leer ist.
- **isEmpty()**: Überprüft, ob der Stack leer ist.
- **size()**: Gibt die Anzahl der Elemente im Stack zurück.
- **clear()**: Entfernt alle Elemente aus dem Stack.
- **printStack()**: Gibt alle Elemente des Stacks als String zurück.

Diese Implementierung zeigt, wie man einen Stack in JavaScript erstellt und verwendet, indem man grundlegende Methoden zur Verwaltung der Stack-Elemente bereitstellt.

Vorgehen

- Projekt auf GitHub anlegen
- dieses Dokument im Projekt mit aufnehmen
- Lösung als Pseudo-Code erstellen
- Pseudo-Code in JavaScript umsetzen
- alle 10 Testfälle aus dem Dokument durchführen und Ergebnisse vergleichen
- Zeitaufwand 90 Minuten

Hinweis

- Versuche die Lösung selbst zu entwicklen
- Verwende kein fertige Lösung, die KI generiert wurde