javascript.prac

Created	@November 19, 2024 6:40 PM
Class	MAT 201

1. Strings

Basics

- Strings are sequences of characters enclosed in **single** (`'`), **double** (`"`), or **backticks** (`` ` ``).
- Strings are **immutable** (cannot be changed after creation).

```
let greeting = "Hello, World!";
let name = 'Ali';
let message = `Welcome, ${name}!`; // Template Literal
```

Modern String Features

a. Template Literals

- Use backticks `` ` `` to embed variables directly with `${}`.

```
let name = "Ali";
let greeting = `Hello, ${name}!`;
console.log(greeting); // Output: Hello, Ali!
```

b. Multiline Strings

- Template literals allow multi-line strings without `\n`.

```
let poem = `
Roses are red,
Violets are blue,
JavaScript is awesome,
And so are you!
`;
console.log(poem);
```

add \n character whenever you require a line break to add a new line to a string.

String Methods

- `.startsWith()` and `.endsWith()`:

return value in boolean.

```
console.log("JavaScript".startsWith("Java")); // true
console.log("JavaScript".endsWith("Script")); // true
```

- `.includes()`:

```
console.log("Hello World".includes("World")); // true
```

- `.repeat()`:

```
console.log("Hi! ".repeat(3)); // Hi! Hi! Hi!
```

- `.padStart()` and `.padEnd()`:

```
console.log("5".padStart(3, "0")); // 005
console.log("5".padEnd(3, "0")); // 500
```

Syntax:

```
string.padStart(targetLength, padString)
```

Parameters:

- `targetLength`**: Total length jo string ki honi chahiye.**
- `padString` **(optional): Jo characters string ke shuru mein add karne hain. Default hai** `" "` **(space).**

Example:

```
let str = "5";
let padded = str.padStart(3, "0");
console.log(padded); // Output: "005"
```

Yahaan "5" ke shuru mein `0` **add kiya gaya, taa ke total length 3 ho jaye.**

- `.padStart()` string ke shuru mein characters add karta hai.
- `.padEnd()` string ke end mein characters add karta hai.
  Ye dono methods tab useful hain jab aapko fixed length ka output chahiye, jaise forms ya receipts design karte waqt.

String to Array

```
let char = "JavaScript".split("");
console.log(char); // ['J', 'a', 'v', 'a', 'S', 'c', 'r',
'i', 'p', 't']
```

JavaScript ki `.split()` method ek string ko tod kar ek array mein convert kar deti hai, aur yeh todne ke liye kisi specific separator (delimiter) ka use karti hai.

Syntax:

```
string.split(separator, limit)
```

Parameters:

1. `separator` (optional): Yeh wo character ya regular expression hai jiske basis par string ko todna hai. Agar separator specify na karein, toh poori string ek array ke andar as a single element rahegi.
2. `limit` (optional): Yeh integer value specify karti hai ke kitne elements array mein hone chahiye. (Jo baqi parts hain, unhein ignore kiya jayega.)

Example:

1. String ko spaces par todna:

```
let sentence = "Mujhe JavaScript seekhni hai";
let words = sentence.split(" ");
console.log(words);
// Output: ["Mujhe", "JavaScript", "seekhni", "hai"]
```

Yahaan string ko `" "` (space) ka separator use karke tod diya gaya aur ek array return hui.

---

2. Specific character par split karna:

```
javascript

let data = "Apple,Banana,Cherry";
let fruits = data.split(",");
console.log(fruits);
// Output: ["Apple", "Banana", "Cherry"]
```

Separator `","` ke base par string ko tod diya gaya.

---

3. Limit ka use karna:

```
let text = "Ali,Ahmed,Asim,Amna";
let limited = text.split(",", 2);
console.log(limited);
// Output: ["Ali", "Ahmed"]
```

Yahaan `limit` 2 hai, isliye sirf pehle do elements array mein aaye.

---

4. Separator ke bina use karna:

```
let name = "Ramila";
let characters = name.split();
console.log(characters);
// Output: ["Ramila"]
```

Yahaan separator nahi diya, toh poori string ek hi element ke roop mein array mein chali gayi.

---

5. Har character ko tod kar array banana:

Agar separator specify na karein ya ek empty string `""` ka use karein, toh string ke har character ko tod diya jata hai:

```
let word = "Hello";
let chars = word.split("");
console.log(chars);
// Output: ["H", "e", "l", "l", "o"]
```

Summary:

- `.split()` ka kaam hai string ko todna aur ek array return karna.
- **Separator:** Define karta hai kis basis par todna hai.
- **Limit:** Specify karta hai kitne elements chahiye.

Yeh method un scenarios mein helpful hai jahan strings ko process karte waqt todne ki zarurat ho, jaise:

- CSV (Comma Separated Values) files ko process karte waqt.
- Words count karte waqt.
- Strings ko individual characters mein todna.

Numbers

Basics

- Numbers in JavaScript are **floating-point values**.
- Use `Number` or `parseInt` / `parseFloat` for conversions.

1. `Number()`

- Yeh ek global function hai jo string ko ek **number** mein convert karta hai.
- Agar string ek valid number na ho, toh yeh `NaN` (Not-a-Number) return karta hai.
- Poora string evaluate hota hai. Agar koi invalid character ho, conversion fail ho jata hai.

Example:

```
let str = "123";
let num = Number(str);
console.log(num); // Output: 123 (as a number)

let invalidStr = "123abc";
console.log(Number(invalidStr)); // Output: NaN
```

2. `parseInt()`

- Yeh method string ko ek **integer** (poora number) mein convert karta hai.
- Yeh string ko **left se evaluate** karta hai aur sirf initial valid digits ko convert karta hai. Baqi characters ignore kiye jate hain.
- Agar string ka pehla character valid number na ho, toh yeh `NaN` return karta hai.
- Optional: Ek **radix** (base) specify karne ke liye second argument de sakte hain, jaise base 10 (decimal) ya base 16 (hexadecimal).

Example:

```
let str = "123px";
console.log(parseInt(str)); // Output: 123
```

```
let invalidStr = "px123";
console.log(parseInt(invalidStr)); // Output: NaN

// With radix:
let hexStr = "0xFF";
console.log(parseInt(hexStr, 16)); // Output: 255
```

Radix ka matlab hai **number system ka base**, jo define karta hai ke ek number ko kaise interpret kiya jaye. Yeh define karta hai ke ek number kis system mein likha gaya hai, jaise:

- **Base 10 (Decimal):** Yeh humara default number system hai jo 0-9 digits ka use karta hai.
- **Base 2 (Binary):** Yeh sirf 0 aur 1 ka use karta hai.
- **Base 8 (Octal):** Yeh 0 se 7 tak ki digits ka use karta hai.
- **Base 16 (Hexadecimal):** Yeh 0-9 aur A-F tak ki values ka use karta hai.

Radix in JavaScript

Jab hum `parseInt()` function ka use karte hain, toh optional second parameter (radix) specify kar sakte hain jo batata hai ke string kis base mein hai.

Syntax:

```
parseInt(string, radix)
```

Examples:

1. Default Base 10 (Decimal):

Agar radix specify na karein, aur string koi valid decimal number ho, toh JavaScript default **Base 10** ka use karega.

```
javascript
Copy code
let num = parseInt("123");
console.log(num); // Output: 123 (Decimal)
```

2. Base 2 (Binary):

Binary numbers sirf `0` aur `1` ka use karte hain. Radix 2 specify karne par string ko binary samjha jata hai.

```
javascript
Copy code
let binary = parseInt("1010", 2);
console.log(binary); // Output: 10 (Decimal)
```

3. Base 8 (Octal):

Octal numbers 0-7 digits ka use karte hain. Radix 8 specify karne par string ko octal samjha jata hai.

```
javascript
Copy code
let octal = parseInt("17", 8);
console.log(octal); // Output: 15 (Decimal)
```

4. Base 16 (Hexadecimal):

Hexadecimal numbers 0-9 aur A-F ka use karte hain (A = 10, B = 11, ... F = 15). Radix 16 specify karne par string ko hexadecimal samjha jata hai.

```
javascript
Copy code
let hex = parseInt("1F", 16);
console.log(hex); // Output: 31 (Decimal)
```

Radix Specify Karna Zaroori Hai

Jab radix clearly specify kiya jaye, toh yeh code ka behavior predictable banata hai. Agar radix specify na karein, toh kuch scenarios mein confusion ho sakti hai:

Example (Without Radix):

```
let num = parseInt("010");
console.log(num); // Output: 10 (Modern JS defaults to Base 10)
```

Kuch purane JavaScript engines `010` ko octal (Base 8) samajhte the aur output `8` hota tha. Isi confusion se bachne ke liye radix specify karna best practice hai.

Summary:

Radix define karta hai ke string kis base ke number mein hai:

- **Base 10 (Decimal):** Default, jo 0-9 digits ka use karta hai.
- **Base 2 (Binary):** Radix 2.
- **Base 8 (Octal):** Radix 8.
- **Base 16 (Hexadecimal):** Radix 16.

Best Practice: `parseInt()` ke sath radix zaroor specify karein, taa ke code ka behavior predictable ho.

3. `parseFloat()`

- Yeh method string ko ek **floating-point number** (decimal number) mein convert karta hai.
- Yeh bhi left se evaluate karta hai aur sirf initial valid digits ko convert karta hai. Baqi characters ignore kiye jate hain.
- Decimal points ko handle karne ke liye yeh `parseInt` se zyada useful hai.

Example:

```
javascript
Copy code
let str = "123.45px";
console.log(parseFloat(str)); // Output: 123.45

let invalidStr = "px123.45";
console.log(parseFloat(invalidStr)); // Output: NaN
```

Use Cases:

- `Number()`:
  - Jab complete string ek valid number ho.
  - Use karna straightforward aur clean hota hai.
- `parseInt()`:
  - Jab aapko string se **integer extract** karna ho (even with extra text).
  - Jab radix ka use karna ho.
- `parseFloat()`:
  - Jab aapko string se **floating-point number extract** karna ho.