

JavaScript Maps

A **Map** is a JavaScript object that stores data in **key-value pairs**.

- ✓ Keys can be **any type** (string, number, object, function)
 - ✓ Maintains **insertion order**
 - ✓ Faster than normal objects for large data
-

Example

```
let myMap = new Map();
```

```
myMap.set("name", "Jaspreet");
```

```
myMap.set("age", 20);
```

```
myMap.set(1, "one");
```

```
console.log(myMap);
```

Why Use Map?

- Keys of any data type
 - Easy iteration
 - Better performance than objects
 - Clean key-value structure
-

JS Map Methods

new Map()

Creates a new Map.

```
let map = new Map();
```

set(key, value)

Adds data to map.

```
map.set("city", "Delhi");
```

get(key)

Gets value by key.

```
map.get("city"); // Delhi
```

has(key)

Checks if key exists.

```
map.has("city"); // true
```

delete(key)

Removes key-value pair.

```
map.delete("city");
```

clear()

Deletes all entries.

```
map.clear();
```

size

Returns total entries.

```
map.size;
```

forEach()

Loop through map.

```
map.forEach((value, key) => {  
  console.log(key, value);  
});
```

keys()

Returns all keys.

```
map.keys();
```

values()

Returns all values.

```
map.values();
```

entries()

Returns key-value pairs.

```
map.entries();
```

Looping Through Map

Using for...of

```
for (let [key, value] of map) {  
  console.log(key, value);  
}
```

JS WeakMap

A **WeakMap** stores **key-value pairs**, where **keys must be objects**.

Example

```
let user1 = {name: "A"};  
let user2 = {name: "B"};  
  
let wm = new WeakMap();  
  
wm.set(user1, "Admin");
```

```
wm.set(user2, "User");
```

WeakMap Methods

set()

```
wm.set(obj, value);
```

get()

```
wm.get(obj);
```

has()

```
wm.has(obj);
```

delete()

```
wm.delete(obj);
```

JavaScript Loops

Loops are used to run code **again and again** until a condition is false.

1. for Loop

Used when number of iterations is known.

```
for (let i = 1; i <= 5; i++) {  
  console.log(i);  
}
```

2. while Loop

Runs while condition is true.

```
let i = 1;
```

```
while (i <= 5) {  
  console.log(i);  
  i++;  
}
```

3. do...while Loop

Runs **at least once**.

```
let i = 1;  
  
do {  
  console.log(i);  
  i++;  
} while (i <= 5);
```

4. for...in Loop

Used for **object keys**.

```
let user = { name: "A", age: 20 };  
  
for (let key in user) {  
  console.log(key, user[key]);  
}
```

5. for...of Loop

Used for **iterable objects** (arrays, strings, maps).

```
let arr = [10, 20, 30];  
  
for (let value of arr) {  
  console.log(value);}
```

JavaScript Regular Expressions (RegExp)

A **Regular Expression (RegExp)** is a **pattern** used to **search, match, or replace text** in strings.

Creating a RegExp

Method 1 (Literal)

```
let pattern = /hello/;
```

Method 2 (Object)

```
let pattern = new RegExp("hello");
```

JS RegExp Flags

Flags modify how the pattern works.

Flag	Meaning	Use
------	---------	-----

g	Global	Find all matches
i	Ignore case	Case-insensitive
m	Multiline	Match across lines
s	Dotall	Dot matches newline
u	Unicode	Unicode support
y	Sticky	Match from last index

Example

```
let text = "Hello hello";
```

```
let result = text.match(/hello/gi);
```

```
console.log(result);
```

Output

```
["Hello", "hello"]
```

JS RegExp Classes (Character Classes)

Used to match **groups of characters**.

Pattern Meaning

[abc] a or b or c

[0-9] any digit

[a-z] lowercase letters

[A-Z] uppercase letters

[^0-9] NOT a digit

Example

```
let text = "abc123";  
console.log(text.match(/[0-9]/g));
```

JS RegExp Metacharacters

Special characters with **special meaning**.

Metachar Meaning

. Any character

\d Digit

\D Non-digit

\w Word character

\W Non-word

\s Whitespace

\S Non-whitespace

Example

```
let text = "ID 45";  
console.log(text.match(/\d+/));
```

JS RegExp Assertions

Assertions **check position**, not characters.

Assertion Meaning

<code>^</code>	Start of string
<code>\$</code>	End of string
<code>\b</code>	Word boundary
<code>\B</code>	Not word boundary

Example

```
let text = "Hello world";  
console.log(/^Hello/.test(text)); // true  
console.log(/world$/.test(text)); // true
```

JS RegExp Quantifiers

Used to define **how many times** a character appears.

Quantifier Meaning

<code>*</code>	0 or more
<code>+</code>	1 or more
<code>?</code>	0 or 1
<code>{n}</code>	Exactly n
<code>{n,}</code>	At least n
<code>{n,m}</code>	Between n and m

Example

```
let text = "aaa";  
console.log(text.match(/a+/));
```

JS RegExp Patterns

Patterns define **what to search**.

OR operator

```
let text = "cat dog";  
  
console.log(text.match(/cat|dog/));
```

Grouping

```
let text = "apple pie";  
  
console.log(text.match(/(apple)/));
```

JS RegExp Objects

RegExp is a **built-in JavaScript object**.

```
let pattern = new RegExp("js", "i");  
  
console.log(pattern.test("JavaScript"));
```

JS RegExp Methods

1. test()

Returns **true or false**

```
let pattern = /hello/i;  
  
console.log(pattern.test("Hello world"));
```

2. exec()

Returns **matched result**

```
let pattern = /world/;  
  
let result = pattern.exec("Hello world");  
  
console.log(result);
```

3. match()

Used on strings

```
let text = "abc abc";
```

```
console.log(text.match(/abc/g));
```

4. search()

Returns index

```
let text = "Hello world";
```

```
console.log(text.search(/world/));
```

5. replace()

Replaces matched text

```
let text = "I like JS";
```

```
console.log(text.replace(/JS/, "JavaScript"));
```
