<u>Slim</u>









Arrow Functions

```
JS main.js

// ES5 function
function addition(a, b) {
   return a + b;
}
// ES6 arrow function
const addition = (a, b) ⇒ a + b;
```

Explanation : Arrow functions provide a concise syntax for writing functions, especially useful for **short**, **one-line** operations.

```
Slim toumi
```





Template Literals

```
const name = 'John';
const greeting = `Hello, ${name}!`;

console.log(greeting);

//result: Hello, John!
```

Explanation: Template literals allow embedding expressions inside strings, providing a cleaner and more readable way to concatenate strings.







Destructuring Assignment

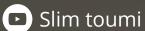
```
const person = { name: 'Alice', age: 25 };
// Extracting properties
const { name, age } = person;

console.log('Name :',name,' Age :',age)

//result: Name : Alice Age : 25
```

Explanation: **Destructuring** assignment simplifies the **extraction of values** from **objects** or **arrays** into **individual** variables.







Spread Operator

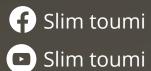
```
JS main.js

const numbers = [1, 2, 3];
const newNumbers = [...numbers, 4, 5];

console.log('newNumbers :',newNumbers)

//result:newNumbers :[1,2,3,4,5]
```

Explanation: The spread operator allows for the expansion of elements, making it handy for creating new arrays or objects based on existing ones.





Rest Parameter

```
const sum = (...numbers) \Rightarrow {
  return numbers.reduce((acc, num) \Rightarrow {
    return acc + num;
  }, 0);
};

console.log(sum(1, 2, 3));
// result: 6
```

Explanation: The **rest parameter** allows functions to accept an **indefinite** number of arguments as an **array**, simplifying parameter handling.

```
Slim toumi
```





Async / Await

```
const API = 'https://api.example.com';
const fetchData = async () => {
  try {
    const result = await fetch(`${API}/data`);
    const data = await result.json();
    console.log(data);
} catch (error) {
    console.error(error);
}
};
```

Explanation: Async/await is a syntax for handling **asynchronous** code more concisely, providing a cleaner alternative to working with Promises.

Slim toumi





Map & Set

```
JS main.js

// Creating a Map with a key-value pair
const numberMap = new Map().set('one', 1);

// Creating a Set with unique numbers
const unique = new Set([1, 2, 3, 2, 1]);

unique.forEach(number => console.log(number));

// Output: 1

// 2

// 3
```

Explanation: Map and Set are new data structures introduced in ES6.

Map is an ordered collection of key-value pairs,
and Set is a collection of unique values.

```
Slim toumi
```





Slim

Default Parameters

```
const greet = (name = 'Guest') => {
  return `Hello, ${name}!`;
};
console.log(greet());
// Output: Hello, Guest!

console.log(greet('John'));
// Output: Hello, John!
```

Explanation: Default parameters provide values for function parameters if none are provided, improving flexibility and reducing the need for explicit checks.







Modules

```
JS main.js

// Exporting module
export const myFunction = () => {...};

// Importing module
import { myFunction } from './myModule';
```

Explanation: ES6 modules provide a **clean** and **organized** way to structure and **import/export** code, improving **maintainability** and **reusability**.







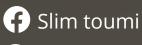
map Methode

```
JS main.js

const numbers = [1, 2, 3, 4, 5];
const doubled = numbers.map(num => num * 2);

console.log(double)
// Result: [2, 4, 6, 8, 10]
```

Explanation: The **map** method in JavaScript is used to create a **new array** by applying a provided function to **each** element of an **existing** array.







filter Methode

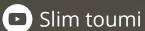
```
JS main.js

const numbers = [1, 2, 3, 4, 5];
const evens = numbers.filter(num => num % 2 === 0);

console.log(evens)
// Result: [2, 4]
```

Explanation: The **filter** method is used to create a **new array** containing only the elements that satisfy a **specified** condition.







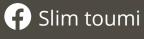
reduce Methode

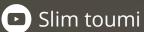
```
JS main.js

const data = [1, 2, 3, 4, 5];
const sum = data.reduce((acc, num) => acc + num, 0);

console.log(sum)
// Result: 15
```

Explanation: The **reduce** method is used to **accumulate** the elements of an array into a **single** value







S

Hopefully You Found It Usefull!

Be sure to save this post so you can come back to it later,

like

Comment

Share





