JS

1

NEW JAVASCRIPT FEATURES

JS

2

ARRAY.WITH()

immutable ES 2023

```
const arr = ['apple', 'banana', 'mango'];
// without .with()
// const newArr = [...arr];
// newArr[0] = 'orange';

const newArr = arr.with(0, 'orange');
console.log(arr);
console.log(newArr);
// arr [ 'apple', 'banana', 'mango']
// newArr [ 'orange', 'banana', 'mango']
```

support: 83.52%

JS

3

ARRAY.TOSORTED()

immutable ES 2023

```
const arr = ['banana', 'apple', 'mango'];
const sortedArr = arr.toSorted();
```

```
console.log(arr);
console.log(sortedArr);
// arr ['banana', 'apple', 'mango']
// sortedArr ['apple', 'banana', 'mango']
```

support: 83.52%

JS

4

ARRAY.TOREVERSED()

immutable ES 2023

```
const arr = ['apple', 'banana', 'mango'];
const reversedArr = arr.toReversed();

console.log(arr);
console.log(reversedArr);
// arr ['apple', 'banana', 'mango']
// reversedArr ['mango', 'banana', 'apple']
```

support: 83.52%

JS

5

ARRAY.TOSPLICED()

immutable ES 2023

```
const arr = ['apple', 'banana', 'mango'];
const splicedArr = arr.toSpliced(0, 1, 'kiwi');

console.log(arr);
console.log(splicedArr);
// arr  ['apple', 'banana', 'mango']
// splicedArr ['kiwi', 'banana', 'mango']
```

support: 83.52%

JS

6

.FINDLASTINDEX()
.FINDLAST()

ES 2023

```
const arr = [{ name: 'apple' }, { name: 'kiwi' }];
// without .findLast()
// [...arr].reverse().find(obj => obj.name === 'kiwi');

console.log(arr.findLast(obj => obj.name === 'kiwi'));
console.log(arr.findLastIndex(obj => obj.name === 'kiwi'));
// {name: 'kiwi'}
// 1
```

support: 89%

JS

7

PREVIOUS METHODS PREPARED US FOR NEW TYPES!!

RECORD & TUPLE

Not yet part of the specification.

JS

8

RECORD

Record is analogous to an Object in JavaScript with the exception that the Record is not an Object but a deeply immutable primitive value.

```
const user = #{
    name: 'John',
    surname: 'Doe',
    age: 25,
};

user.name = 'Alex';
// TypeError, cannot assign to read only property
```

JS

9

TUPLE

Tuple is like an Array but is a deeply immutable primitive value.

```
const ages = #[20, 21, 22, 23];
ages[0] === 1; // TypeError!
console.log(ages)
// Tuple {0: 20, 1: 21, 2: 22, 3: 23}
```

JS

10

RECORD & TUPLE

Record and Tuple use only primitive values including other records or tuples

```
// TypeError!
const users = #[{ name: 'John' }, { name: 'Alex' }];
// Right
const users = #[#{ name: 'John' }, #{ name: 'Alex' }];

const user = #{
    name: 'John',
    contacts: #{  // Right
      email: 'user@gmail.com'
  }
    // contacts: {} // TypeError!
};
```

JS

11

PRIVATE CLASS FIELDS

ES 2022

```
class Human {
  #name = 'John';
  setName(name) {
    this.#name = name;
  }
  getName() {
    return this.#name;
  }
}

const human = new Human();
human.#name = 'Amy'; // Error!
human.setName('Amy'); // Right
console.log(human.getName()); // Right
```

support: 91.6%

JS

12

ERROR HANDLING - CAUSE

ES 2022

```
try {
  someCodeWithErr();
} catch (error) {
  throw new Error('This is the result of another error',
    { cause: error }
  );
}
```

support: 89.68%

JS

13

TOP LEVEL AWAIT

ES 2022

```
// Some cases:
const strings = await import(
  `/i18n/${navigator.language}`
);
const connection = await dbConnector();
```

support: 89.93%

JS

14

ARRAY/STRING.AT()

ES 2022

```
const arr = [1, 2, 3, 4, 5];
// without at()
console.log(arr[arr.length - 1]); //5
console.log(arr.slice(-1)[0]); //5

console.log(arr.at(-1)); //5
```

support: 89.47%

JS

15

Don’t forget to

LIKE, COMMENT, REPOST