String Methods

Methods - actions that can be performed on objects.

Format

stringName.**method( )**

String Methods

let msg = " Hello ";

str.trim()

Trims whitespaces from both ends of string & returns a new one.

```
> let msg = "   Hello   ";
<· undefined
> msg.trim();
<· 'Hello'
> msg
<· '   Hello   '
```

output : "Hello", but value of msg remains same.

JS app.js > ...

```
// let msg = "    he  llo     ";

let password = prompt("set your password");
let newPass = password.trim();
console.log(newPass);
```

```
> str
<· '    hello    '
> str.trim();
<· 'hello'
> str
<· '    hello    '
>
```

String are Immutable in JS

No changes can be made to strings.

Whenever we do try to make a change, a new string is created and old one remains same.

```
> let msg = "apna    ";
<· undefined
> msg.trim();
<· 'apna'
> let str = msg.trim();
<· undefined
> str
<· 'apna'
> msg
<· 'apna    '
>
```

String Methods

```
let str = "Random string";
```

```
str.toUpperCase( )        "RANDOM STRING"

str.toLowerCase( )        "random string"
```

top Filter

```
> name
<· 'Apna College'
> name.toLowerCase();
<· 'apna college'
> name.toLowerCase
<· ƒ toLowerCase() { [native code] }
> name.toUpperCase();
<· 'APNA COLLEGE'
>
```

```
let name = "Apna College";
let msg = "error";

console.log(msg.toUpperCase)
```

String Methods with Arguments

Argument is a some value that we pass to the method.

Format

stringName.method(arg)

indexOf

Returns the first index of occurrence of some value in string. Or gives -1 if not found.

```
let str = "IloveCoding";

str.indexOf("love")      // 1

str.indexOf("J")         // -1 (not found)

str.indexOf("o")         // 2 (only 1 index)
```


```
msg
'ILoveCoding'
msg.indexOf("Love");
1
msg.indexOf("love");
-1
msg.indexOf("o");
2
```

Method Chaining

Using one method after another. Order of execution will be left to right.

```
str.toUpperCase( ).trim( )
```

```
let msg = "    hello   ";
// let newMsg = msg.trim();
// console.log("after trim : ", newMsg);
// newMsg = newMsg.toUpperCase();
// console.log("after uppercase : ", newMsg);
let newMsg = msg.trim().toUpperCase();
```

slice

Returns a part of the original string as a new string.

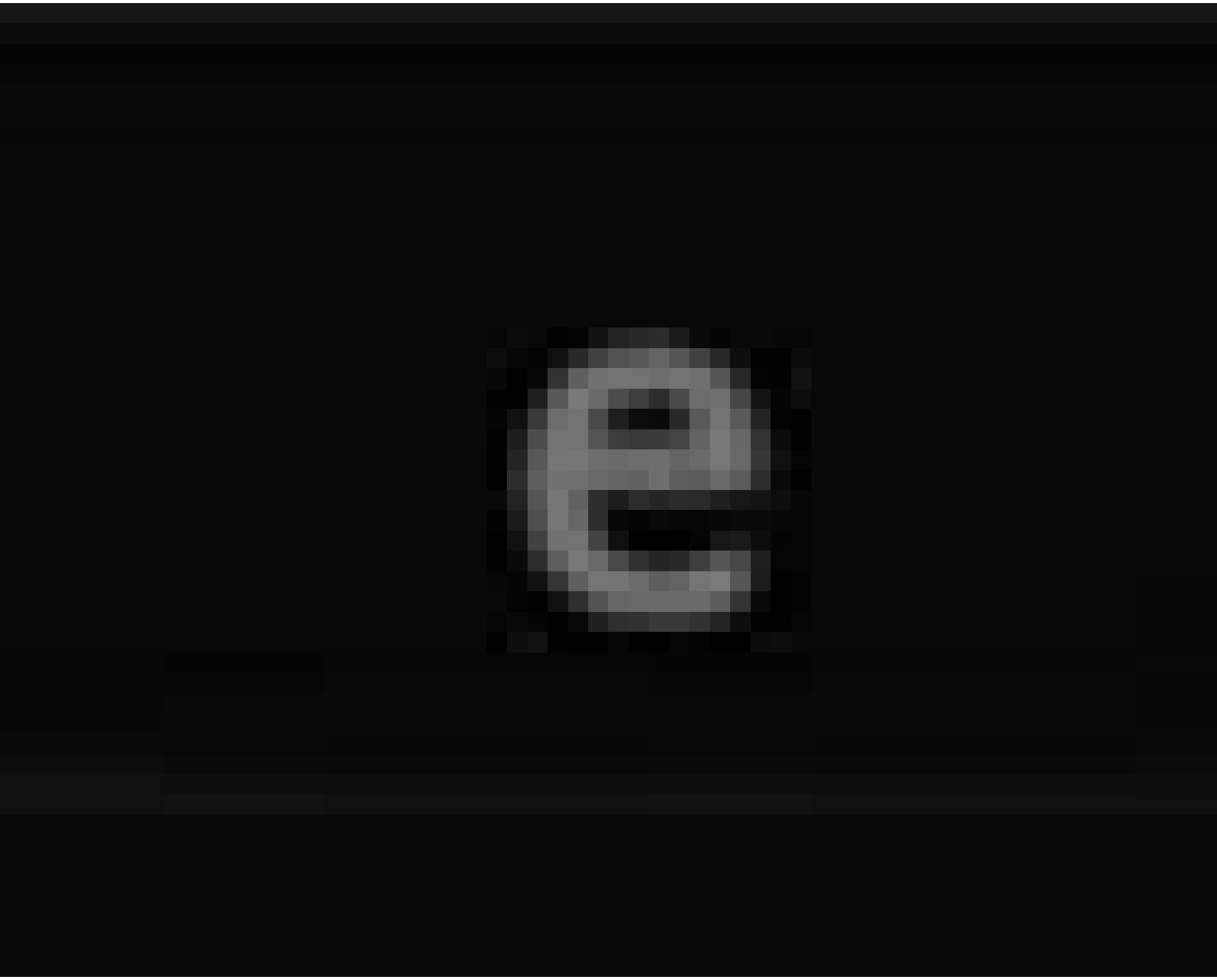
```
let str = "IloveCoding";

    str.slice(5)                    // "Coding"

    str.slice(1, 4)                 // "love"

    str.slice(-num)   =   str.slice(length-num)
```

JS app.js > msg

```
1    let msg = "hello";
2    console.log(msg.slice(0, 4));
3
```

JS app.js > ...

```
1   let msg = "apnacollege";
2   console.log(msg.slice(-1)); //11-1 => 10
3
```



replace

Searches a value in the string & returns a new string with the value replaced.

```
let str = "IloveCoding";

    str.slice("love", "do")    // "IdoCoding"

    str.slice("o", "x")        // "IlxveCoding"
```

```
> msg
<· 'IloveCoding'
> msg.replace("love", "do");
<· 'IdoCoding'
> msg
<· 'IloveCoding'
> msg.replace('o', 'x');
<· 'IlxveCoding'
>
```

repeat

Returns a string with the number of copies of a string

```
let str = "Mango";

    str.repeat(3)            // "MangoMangoMango"
```

```
> fruit
<· 'mango'
> fruit.repeat(5);
<· 'mangomangomangomangomango'
>
```

Practice Qs

Qs. For the Give String :

```
let msg = "help!";
```

Trim it & convert it to uppercase.

Qs. For the String -> let name = "ApnaCollege", predict the output for following :

name.slice(4, 9)

name.indexOf("na")

name.replace("Apna","Our")

Qs. Separate the "College" part in above string & replace 'l' with 't' in it.

```
> name.slice(4, 9);
<· 'Colle'
> name.indexOf("na");
<· 2
> name.replace("Apna", "Our");
<· 'OurCollege'
> name.slice(4)
<· 'College'
> name.slice(4).replace('l', 't')
<· 'Cotlege'
> let newStr = name.slice(4).replace('l', 't');
<· undefined
> newStr
<· 'Cotlege'
> newStr.replace('l', 't');
<· 'Cottege'
>
```

```
> name
<· 'ApnaCollege'
> name.slice(4).replace('l', 't').replace('l', 't');
<· 'Cottege'
>
```

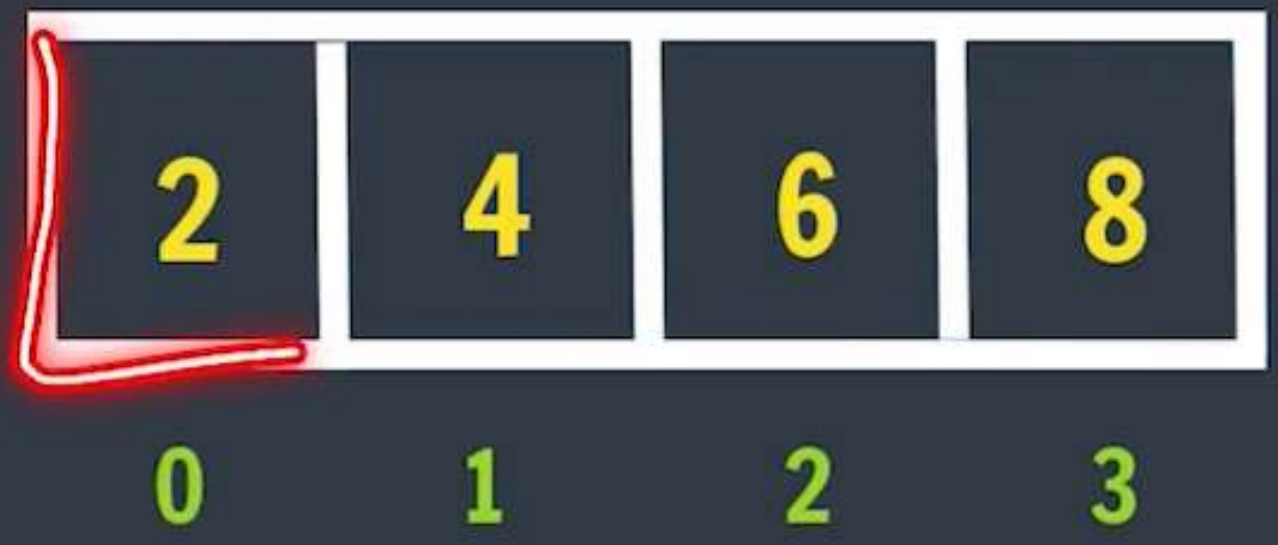
Array (Data Structure)

Linear collection of things

```
let students = ["aman", "shradha", "rajat"]
```

```
let students = ["aman", "shradha", "rajat"];
```

Visualizing Array




```
let nums = [ 2, 4, 6, 8 ];
```

```
let nums = [2, 4, 6, 8];
undefined
nums
▶ (4) [2, 4, 6, 8]
let name = "shradha";
undefined
name[0]
's'
nums[0]
2
nums[1]
4
nums[2]
6
nums[3]
8
nums[4]
undefined
```

```
> nums.length
<· 4
> typeof nums
<· 'object'
>
```

Creating Arrays

```
let marks = [99, 85, 93, 76, 62];
let names = ["adam", "bob", "catlyn"];
let info = ["aman", 25, 6.1];   //mixed array

//empty array
let newArr = [];
```

```
let info = ["shradha", 23, 89.9];
undefined
info
(3) ['shradha', 23, 89.9]
```

```
> empArr[0]
< undefined
> info.length
< 3
> empArr.length
< 0
> [].length
< 0
> [1, 2, 3, 4].length
< 4
```

```
> info[0]
<· 'shradha'
> info[0][0]
<· 's'
> info[0][1]
<· 'h'
```

Arrays are Mutable

```
> let fruits = ["mango", "apple", "litchi"];
<· undefined
> fruits[0] = "banana";
<· 'banana'
> fruits
<· ▶ (3) ['banana', 'apple', 'litchi']
```

Arrays are Mutable

```
> let fruits = ["mango", "apple", "litchi"];
<· undefined
> fruits[0] = "banana";
<· 'banana'
> fruits
<· ▶ (3) ['banana', 'apple', 'litchi']
```

```
> let name = "rohit";
<· undefined
> name[0] = 'm';
<· 'm'
> name
<· 'rohit'
> let fruits = ["mango", "apple", "litchi"];
<· undefined
> fruits
<· ▶ (3) ['mango', 'apple', 'litchi']
> fruits[0] = "banana";
<· 'banana'
> fruits
<· ▶ (3) ['banana', 'apple', 'litchi']
> fruits[1] = "pineapple";
<· 'pineapple'
> fruits
<· ▶ (3) ['banana', 'pineapple', 'litchi']
> fruits[10] = "papaya";
<· 'papaya'
> fruits
<· ▶ (11) ['banana', 'pineapple', 'litchi', empty × 7, 'papaya']
```

Array Methods

Push : add to end

Unshift : add to start

Pop : delete from end & returns it

Shift : delete from start & returns it

Array Methods

Push : add to end

Unshift : add to start

Pop : delete from end & returns it

Shift : delete from start & returns it

```
> cars
<· ▶ (4) ['audi', 'bmw', 'xuv', 'maruti']
> cars.push("toyota");
<· 5
> cars
<· ▶ (5) ['audi', 'bmw', 'xuv', 'maruti', 'toyota']
>
```

```
> cars.push("ferrari");
<· 6
> cars
<· ▶ (6) ['audi', 'bmw', 'xuv', 'maruti', 'toyota', 'ferrari']
>
```

```
> cars.pop();
<· 'ferrari'
> cars
<· ▶ (5) ['audi', 'bmw', 'xuv', 'maruti', 'toyota']
> cars.pop();
<· 'toyota'
> cars
<· ▼ (4) ['audi', 'bmw', 'xuv', 'maruti']
      0: "audi"
      1: "bmw"
      2: "xuv"
```

```
> cars
<· ▶ (4) ['audi', 'bmw', 'xuv', 'maruti']
> cars.unshift("toyota");
<· 5
> cars
<· ▶ (5) ['toyota', 'audi', 'bmw', 'xuv', 'maruti']
> cars.unshift("ferrari");
<· 6
> cars
<· ▶ (6) ['ferrari', 'toyota', 'audi', 'bmw', 'xuv', 'maruti']
> cars.shift();
<· 'ferrari'
> cars
<· ▶ (5) ['toyota', 'audi', 'bmw', 'xuv', 'maruti']
```

```
> let followers = ["a", "b", "c"];
<· undefined
> let blocked = followers.shift();
<· undefined
> followers
<· ▶ (2) ['b', 'c']
> blocked
<· 'a'
```

Practice Qs

Qs. For the given start state of an array, change it to final form using methods.

start : `['january', 'july', 'march', 'august']`

final : `['july', 'june', 'march', 'august']`

```
> let months = ["january", "july", "march", "august"];
<· undefined
> months
<· ▶ (4) ['january', 'july', 'march', 'august']
> months.shift();
<· 'january'
> months
<· ▶ (3) ['july', 'march', 'august']
> months.shift();
<· 'july'
> months
<· ▶ (2) ['march', 'august']
> months.unshift("june");
<· 3
> months
<· ▶ (3) ['june', 'march', 'august']
> months.unshift("july");
<· 4
> months
<· ▶ (4) ['july', 'june', 'march', 'august']
```

Array Methods

```
> let primary = ["red", "yellow", "blue"];
```

indexOf : returns index of something

```
> primary.indexOf("yellow");
<· 1
> primary.indexOf("green");
<· -1
> primary.indexOf("Yellow");
<· -1
```


includes : search for a value

```
> primary.includes("red");
<· true
> primary.includes("green");
<· false
```

```
> cars
<· ▶ (4) ['audi', 'bmw', 'xuv', 'maruti']
> cars.indexOf("bmw");
<· 1
> cars.indexOf("xuv");
<· 2
> cars.indexOf("XUV");
<· -1
> let marks = [99, 89, 67, 42, 100];
<· undefined
> marks.indexOf(100);
<· 4
> marks.indexOf(97);
<· -1
>
```

```
> cars
<· ▶ (4) ['audi', 'bmw', 'xuv', 'maruti']
> cars.includes("audi");
<· true
> cars.includes("toyota");
<· false
>
```

Array Methods

```
> let primary = ["red", "yellow", "blue"];
<· undefined
> let secondary = ["orange", "green", "violet"];
<· undefined
```

concat : merge 2 arrays

```
> primary.concat(secondary);
<· ▶ (6) ['red', 'yellow', 'blue', 'orange', 'green', 'violet']
```

reverse : reverse an array

```
> primary.reverse();
<· ▶ (3) ['blue', 'yellow', 'red']
```

```
> let primary = ["red", "yellow", "blue"];
<· undefined
> let secondary = ["orange", "green", "violet"];
<· undefined
> primary.concat(secondary);
<· ▶ (6) ['red', 'yellow', 'blue', 'orange', 'green', 'violet']
> primary
<· ▶ (3) ['red', 'yellow', 'blue']
> secondary
<· ▶ (3) ['orange', 'green', 'violet']
> let allColors = primary.concat(secondary);
<· undefined
> allColors
<· ▶ (6) ['red', 'yellow', 'blue', 'orange', 'green', 'violet']
> secondary.concat(primary);
<· ▶ (6) ['orange', 'green', 'violet', 'red', 'yellow', 'blue']
>
```

```
> cars
<· ▶ (4) ['audi', 'bmw', 'xuv', 'maruti']
> cars.reverse();
<· ▶ (4) ['maruti', 'xuv', 'bmw', 'audi']
> cars
<· ▶ (4) ['maruti', 'xuv', 'bmw', 'audi']
```

Array Methods

slice : copies a portion of an array

```
> let colors = ["red", "yellow", "blue", "orange", "pink", "white"];
```

```
> colors.slice()
<· ▶ (6) ['red', 'yellow', 'blue', 'orange', 'pink', 'white']
> colors.slice(2);
<· ▶ (4) ['blue', 'orange', 'pink', 'white']
> colors.slice(2, 3);
<· ▶ ['blue']
> colors.slice(-2);
<· ▶ (2) ['pink', 'white']
```

```
> cars
<· ▶ (4) ['audi', 'bmw', 'xuv', 'maruti']
> cars.slice();
<· ▶ (4) ['audi', 'bmw', 'xuv', 'maruti']
> cars.slice(1);
<· ▶ (3) ['bmw', 'xuv', 'maruti']
> cars.slice(1, 3);
<· ▶ (2) ['bmw', 'xuv']
> cars.slice(3)l
⊗ Uncaught SyntaxError: Unexpected identifier 'l'
> cars.slice(3);
<· ▶ ['maruti']
> cars.slice(cars.length-1);
<· ▶ ['maruti']
> cars.slice(5);
<· ▶ []
                    ƒ(?start, ?end)
> cars.slice(cars.length);

<· []
```

```
> cars
<· ▶ (4) ['audi', 'bmw', 'xuv', 'maruti']
> cars.slice(-1)
<· ▶ ['maruti']
> cars.slice(-2)
<· ▶ (2) ['xuv', 'maruti']
> cars.slice(-3)
<· ▶ (3) ['bmw', 'xuv', 'maruti']
> cars.slice(-4)
<· ▶ (4) ['audi', 'bmw', 'xuv', 'maruti']
> cars.slice(-5)
<· ▶ (4) ['audi', 'bmw', 'xuv', 'maruti']
>
```

Array Methods

```
> let colors = ["red", "yellow", "blue", "orange", "pink", "white"];
```

splice : removes / replaces / add elements in place

splice(start, deleteCount, item0...itemN)

```
> colors.splice(4);
<· ▶ (2) ['pink', 'white']
> colors
<· ▶ (4) ['red', 'yellow', 'blue', 'orange']
> colors.splice(0, 1);
<· ▶ ['red']
> colors
<· ▶ (3) ['yellow', 'blue', 'orange']
> colors.splice(0, 1, "black", "grey");
<· ▶ ['yellow']
> colors
<· ▶ (4) ['black', 'grey', 'blue', 'orange']
```

```
> cars
<· ▶ (4) ['audi', 'bmw', 'xuv', 'maruti']
> cars.splice(3);
<· ▶ ['maruti']
> cars
<· ▶ (3) ['audi', 'bmw', 'xuv']
> cars.splice(0,1);
<· ▶ ['audi']
> cars
<· ▶ (2) ['bmw', 'xuv']
> cars.push("maruti");
  cars.push("ferrari");
<· 4
> cars
<· ▶ (4) ['bmw', 'xuv', 'maruti', 'ferrari']
> cars.splice(1, 2);
<· ▶ (2) ['xuv', 'maruti']
> cars
<· ▶ (2) ['bmw', 'ferrari']
> cars.splice(0, 0, "toyota", "xuv", "maruti");
<· ▶ []
> cars
<· ▶ (5) ['toyota', 'xuv', 'maruti', 'bmw', 'ferrari']
```

```
> cars
<· ▶ (5) ['toyota', 'xuv', 'maruti', 'bmw', 'ferrari']
> cars.splice(1, 0, "mercedes");
<· ▶ []
> cars
<· ▼ (6) ['toyota', 'mercedes', 'xuv', 'maruti', 'bmw', 'ferrari']
      0: "toyota"
      1: "mercedes"
      2: "xuv"
      3: "maruti"
      4: "bmw"
      5: "ferrari"
      length: 6
    ▶ [[Prototype]]: Array(0)
```

```
> cars
<· ▶ (5) ['toyota', 'xuv', 'maruti', 'bmw', 'ferrari']
> cars.splice(1, 0, "mercedes");
<· ▶ []
> cars
<· ▶ (6) ['toyota', 'mercedes', 'xuv', 'maruti', 'bmw', 'ferrari']
> cars.splice(1, 1, "gwagon");
<· ▶ ['mercedes']
> cars
<· ▶ (6) ['toyota', 'gwagon', 'xuv', 'maruti', 'bmw', 'ferrari']
```

Array Methods

ascending
descending

sort : sorts an array

```
> let days = ["monday", "sunday", "wednesday", "tuesday"];
<· undefined
> days.sort();
<· ▶ (4) ['monday', 'sunday', 'tuesday', 'wednesday']
```

```
> let squares = [25, 16, 4, 49, 36, 9]
<· undefined
> squares.sort();
<· ▶ (6) [16, 25, 36, 4, 49, 9]
```

```
> cars
<· ▶ (6) ['toyota', 'gwagon', 'xuv', 'maruti', 'bmw', 'ferrari']
> cars.sort();
<· ▶ (6) ['bmw', 'ferrari', 'gwagon', 'maruti', 'toyota', 'xuv']
> let chars = ['b', 'd', 'e', 'a'];
<· undefined
> chars.sort();
<· ▶ (4) ['a', 'b', 'd', 'e']
> let marks = [99, 89, 67, 42, 100];
<· undefined
> marks.sort();
<· ▶ (5) [100, 42, 67, 89, 99]
>
```

Practice Qs

Qs. For the given start state of an array, change it to final form using splice.

start : `['january', 'july', 'march', 'august']`

final : `['july', 'june', 'march', 'august']`

Qs. Return the index of the "javascript" from the given array, if it was reversed.

```
['c', 'c++', 'html', 'javascript', 'python', 'java', 'c#', 'sql']
```

0 1 2 3 4 5 6

```
> let lang = ["c", "c++", "html", "javascript", "python", "java", "c#"];
<· undefined
> lang.push("sql");
<· 8
> lang
<· ▶ (8) ['c', 'c++', 'html', 'javascript', 'python', 'java', 'c#', 'sql']
> lang.reverse()
<· ▶ (8) ['sql', 'c#', 'java', 'python', 'javascript', 'html', 'c++', 'c']
> lang.reverse()
<· ▶ (8) ['c', 'c++', 'html', 'javascript', 'python', 'java', 'c#', 'sql']
> lang
<· ▶ (8) ['c', 'c++', 'html', 'javascript', 'python', 'java', 'c#', 'sql']
> lang.reverse().indexOf("javascript");
<· 4
```

Array References

```
> [1] === [1]
<· false
```

```
> [1] == [1]
<· false
```

address in memory

```
> "name" == "name"
<· true
> "name" === "name"
<· true
> [1] === [1]
<· false
> [1] == [1]
<· false
> [] == []
<· false
> [] === []
<· false
>
```

Array References

```
> let arr = ['a', 'b'];
<· undefined
```

```
> let arrCopy = arr;
<· undefined
```

```
> arrCopy
<· ▶ (2) ['a', 'b']
```

```
> arrCopy.push('c');
<· 3
```

```
> arr
<· ▶ (3) ['a', 'b', 'c']
```

```
> arr == arrCopy
<· true
```

```
> let arr = ['a', 'b', 'c'];
<· undefined
> let arrCopy = arr;
<· undefined
> arr == arrCopy
<· true
> arr === arrCopy
<· true
> arr.push('d');
<· 4
> arr
<· ▶ (4) ['a', 'b', 'c', 'd']
> arrCopy
<· ▶ (4) ['a', 'b', 'c', 'd']
> arrCopy.pop();
<· 'd'
> arrCopy
<· ▶ (3) ['a', 'b', 'c']
> arr
<· ▶ (3) ['a', 'b', 'c']
```

Constant Arrays

```
const arr = [1, 2, 3, 4];
```

```
let a = 5;
```

```
const a = 5
```

```
> const arr = [1, 2, 3];
<· undefined
> arr
<· ▶ (3) [1, 2, 3]
> arr.push(4);
<· 4
> arr
<· ▶ (4) [1, 2, 3, 4]
> arr.pop();
<· 4
> arr
<· ▶ (3) [1, 2, 3]
> arr = [1, 2, 3];
⊗ ▶ Uncaught TypeError: Assignment to constant variable.
    at <anonymous>:1:5
```

Nested Arrays

array of arrays

```
> let nums = [ [2, 4], [3, 6], [4, 8] ];
<· undefined
> nums
<· ▼ (3) [Array(2), Array(2), Array(2)] i
    ▶ 0: (2) [2, 4]
    ▶ 1: (2) [3, 6]
    ▶ 2: (2) [4, 8]
      length: 3
    ▶ [[Prototype]]: Array(0)
```

```
> let nums = [ [1, 2], [3, 4], [4, 5] ];
<· undefined
> let nums = [ [2, 4], [3, 6], [4, 8] ];
<· undefined
> nums
<· ▶ (3) [Array(2), Array(2), Array(2)]
> nums.length
<· 3
> nums[0]
<· ▶ (2) [2, 4]
> nums[0].length
<· 2
> nums[0][0]
<· 2
```

Nested Arrays

```
> let nums = [ [2, 4], [3, 6], [4, 8] ];
```

	0,0	0,1	
	2	4	
1,0	3	6	1,1
2,0	4	8	2,1

rows = arrays = 3
cols = indv.

nums[0][0] //2

```
> let game = [ ['X', null, 'O'], [null, 'X', null], ['O', null, 'X'] ];
<· undefined
> game
<· ▼ (3) [Array(3), Array(3), Array(3)] i
    ▶ 0: (3) ['X', null, 'O']
    ▶ 1: (3) [null, 'X', null]
    ▶ 2: (3) ['O', null, 'X']
      length: 3
    ▶ [[Prototype]]: Array(0)
> game[0]
<· ▶ (3) ['X', null, 'O']
> game[0][1]
<· null
> game[0][1] = 'O';
<· 'O'
> game
<· ▼ (3) [Array(3), Array(3), Array(3)] i
    ▶ 0: (3) ['X', 'O', 'O']
    ▶ 1: (3) [null, 'X', null]
    ▶ 2: (3) ['O', null, 'X']
      length: 3
    ▶ [[Prototype]]: Array(0)
```