# Array

## 1.数组有length属性.表示数组中有多少个元素

## 2.用数组来做一个函数fizzBuzzArr,从1开始往一个数组里面添加指定数量的整数,当这个整数是3的倍数,添加’Fizz’而不是整个数字, 当这个整数是5的倍数,添加’Buzz’而不是整个数字, 当这个整数是3和5的倍数,添加’FizzBuzz’而不是整个数字,然后输出整个数组

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <meta name="viewport" content="width=device-width, initial-scale=1.0">      <title>Document</title>  </head>  <body>      <script>          let names = ["Jack","Angela","Becky","Tracy","Jessie","Mary","Paul"]          function welcomeGuests(guests){              guests.forEach(item => {                  console.log(`Welcome ${item}`);              });          }          // welcomeGuests(names);          console.log(names.includes("Jack")); // includes表示数组中是否包含指定元素      // fizzBuzzArr函数          let output =[]          function fizzBuzzArr(num) {              for(var i=1;i<num;i++)             {                 if(i%3===0 && i%5===0)                {                  output.push("FizzBuzz");                  continue;                }                 else if(i%3===0){                  output.push("Fizz");                  continue;                } else if(i%5===0)                {                  output.push("Buzz");                  continue;                }                output.push(i);             }             console.log(output);          }          fizzBuzzArr(31);      </script>  </body>  </html> |

### 效果

|  |
| --- |
|  |

## 3.还有另外一个版本FizzBuzz2(),刚刚开始的空的,然后每调用一次,就会多一个元素,然后3的倍数和5的倍数以及3和5的公倍数的处理同上

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <meta name="viewport" content="width=device-width, initial-scale=1.0">      <title>Document</title>  </head>  <body>      <script>          let names = ["Jack","Angela","Becky","Tracy","Jessie","Mary","Paul"]          function welcomeGuests(guests){              guests.forEach(item => {                  console.log(`Welcome ${item}`);              });          }          // welcomeGuests(names);          console.log(names.includes("Jack")); // includes表示数组中是否包含指定元素          // let name = prompt("input guest name")          // if(names.includes(name)){          //     alert(`Welcome ${name}!`)          // } else {          //     alert(`This person is not in the list!`)          // }          let output =[]          function fizzBuzzArr(num) {              for(var i=1;i<num;i++)             {                 if(i%3===0 && i%5===0)                {                  output.push("FizzBuzz");                  continue;                }                 else if(i%3===0){                  output.push("Fizz");                  continue;                } else if(i%5===0)                {                  output.push("Buzz");                  continue;                }                output.push(i);             }             console.log(output);          }          // fizzBuzzArr(31);  //另外一个版本fizzBuzz2          let output2=[]          let i=1          function fizzBuzz2()          {              if(i%3===0 && i%5===0)                {                  output2.push("FizzBuzz");                  i++;                }                 else if(i%3===0){                  output2.push("Fizz");                  i++;                } else if(i%5===0)                {                  output2.push("Buzz");                  i++;                }                else{                    output2.push(i);                    i++                }                console.log(output2);            }          fizzBuzz2();      </script>  </body>  </html> |

### 效果

|  |
| --- |
|  |

## 4.febonacci函数的实现,

### 实现逻辑

|  |
| --- |
|  |

### 实现代码

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <meta name="viewport" content="width=device-width, initial-scale=1.0">      <title>Document</title>  </head>  <body>      <script>          let names = ["Jack","Angela","Becky","Tracy","Jessie","Mary","Paul"]          function welcomeGuests(guests){              guests.forEach(item => {                  console.log(`Welcome ${item}`);              });          }          // welcomeGuests(names);          console.log(names.includes("Jack")); // includes表示数组中是否包含指定元素          // let name = prompt("input guest name")          // if(names.includes(name)){          //     alert(`Welcome ${name}!`)          // } else {          //     alert(`This person is not in the list!`)          // }          let output =[]          function fizzBuzzArr(num) {              for(var i=1;i<num;i++)             {                 if(i%3===0 && i%5===0)                {                  output.push("FizzBuzz");                  continue;                }                 else if(i%3===0){                  output.push("Fizz");                  continue;                } else if(i%5===0)                {                  output.push("Buzz");                  continue;                }                output.push(i);             }             console.log(output);          }          // fizzBuzzArr(31);          let output2=[]          let i=1          function fizzBuzz2()          {              if(i%3===0 && i%5===0)                {                  output2.push("FizzBuzz");                  i++;                }                 else if(i%3===0){                  output2.push("Fizz");                  i++;                } else if(i%5===0)                {                  output2.push("Buzz");                  i++;                }                else{                    output2.push(i);                    i++;                }                console.log(output2);            }          // fizzBuzz2();          function whoBuyLunch(names){            let index = Math.floor(Math.random()\*names.length);            return names[index];          }          // console.log(`${whoBuyLunch(names)} buy lunch for all today`);  //斐波那契数列生成器            function fibonacciGen2(n)          {  let fib=[]            if(n===1)            {              fib.push(0);              return fib;            }            if(n===2)            {             fib=[0,1];              return fib;            }            if(n>2){             fib=[0,1];             for(var i=2;i<n;i++)             {               fib.push((fib[i-2]+fib[i-1]))             }             return fib;            }          }      </script>  </body>  </html> |

### 老师的版本,原理是一样的

|  |
| --- |
|  |

### 效果

# 扩展:JavaScript中数组的常见方法

数组常用方法：push pop [unshift](https://so.csdn.net/so/search?q=unshift&spm=1001.2101.3001.7020) shift splice reduce concat slice reverse join indexOf lastIndexOf sort every some filter map forEach

a. 改变原数组的方法：push pop unshift shift splice reverse sort

b. 不改变原数组的方法：reduce concat slice join [indexOf](https://so.csdn.net/so/search?q=indexOf&spm=1001.2101.3001.7020) lastIndexOf every some filter map forEach

## 1. push: 在数组的尾部添加一项或者多项，返回添加数据后数组的长度

let array = ['1', '2', '3']

let res = array.push('4')

console.log(array); *// ['1', '2', '3', '4']*

let res1 = array.push('5', '6', '7')

console.log(array); *//  ['1', '2', '3', '4', '5', '6', '7']*

console.log(res1); *// 7*

## 2. pop: 删除数组的末尾项，返回被删除的项；如果数组是空数组，返回undefined

let array = ['1', '2', '3']

let res = array.pop()

console.log(array); *//  ['1', '2']*

console.log(res); *// '3'*

let array1 = []

let res1 = array1.pop()

console.log(array1); *// []*

console.log(res1); *// undefined*

## 3. unshift: 在数组的头部添加一项或者多项，返回添加数据后数组的长度

let array = ['1', '2', '3']

let res = array.unshift('4')

console.log(array); *//  ['4', '1', '2', '3']*

let res1 = array.unshift('5', '6', '7')

console.log(array); *//  ['5', '6', '7', '4', '1', '2', '3']*

console.log(res1); *// 7*

## 4. shift: 删除数组的首项，返回被删除的项；如果数组是空数组，返回undefined

let array = ['1', '2', '3']

let res = array.shift()

console.log(array); *//  ['2', '3']*

console.log(res); *// '1'*

let array1 = []

let res1 = array1.shift()

console.log(array1); *// []*

console.log(res1); *// undefined*

## 5. splice: 增删改，返回被删除的项组成的数组 splice(开始下标，删除元素的数量，新增元素1，新增元素2，新增元素N)

let array = ['1', '2', '3', '4']

let res = array.splice(1)

console.log(array); *//  ['1']*

console.log(res); *//  ['2', '3', '4']*

let array1 = ['1', '2', '3', '4']

let res1 = array1.splice(1, 2)

console.log(array1); *//  ['1', '4']*

console.log(res1); *// ['2', '3']*

let array2 = ['1', '2', '3', '4']

let res2 = array2.splice(1, 1, '3', '4')

console.log(array2); *// ['1', '3', '4', '3', '4']*

console.log(res2); *// ['2']*

let array3 = ['1', '2', '3', '4']

let res3 = array3.splice(1, 3, '3', '4')

console.log(array3); *// ['1', '3', '4']*

console.log(res3); *// ['2', '3', '4']*

## 6. reduce(callbackFn([accumulator,](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Array/reduce#accumulator)[currentValue](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Array/reduce#currentvalue)[accumulator,](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Array/reduce#accumulator), [currentIndex](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Array/reduce#currentindex)), initalValue)

数组的每个元素上执行callBackFn函数，函数返回值作为下次调用callbackFn的入参accumulator

eg: reduce逐个遍历数组元素，每一步都将当前元素的值与前一步的结果相加

let array = [1, 2, 3, 4]

let sum = array.reduce((curSum, curVal) => {

return curSum + curVal

}, 0)

console.log('sum==', sum); *// 10*

## 7. concat: 拼接2个或多个数组，返回新数组，不改变原有数组

let array = [1, 2, 3, 4]

let array1 = [5, 6, 7, 8]

let array2 = [9, 0]

let newArray = array.concat(array1, array2)

console.log('newArray==', newArray); *// [1, 2, 3, 4, 5, 6, 7, 8, 9, 0]*

console.log('array==', array); *// [1, 2, 3, 4]*

console.log('array1==', array1); *// [5, 6, 7, 8]*

console.log('array2==', array2); *// [9, 0]*

## 8. slice: 返回截取出来的新数组 slice(start, end)     范围：左闭右开

let array = [1, 2, 3, 4, 5, 6, 7, 8]

let newArray = array.slice(2, 5)

console.log('newArray', newArray); *// [3, 4, 5]*

console.log('array', array); *// [1, 2, 3, 4, 5, 6, 7, 8]*

## 9. reverse：数组翻转，返回翻转后的数组，改变原数组

let array = [1, 2, 3, 4]

let newArray = array.reverse()

console.log('newArray', newArray); *// [4, 3, 2, 1]*

console.log('array', array); *// [4, 3, 2, 1]*

## 10. join: 将所有的数组元素用连接符拼接起来，返回字符串 join(连接符)

let array = [1, 2, 3, 4]

let newArray = array.join()

console.log('newArray', newArray); *// 1,2,3,4*

let newArray1 = array.join("&")

console.log('newArray1', newArray1); *// 1&2&3&4*

let newArray2 = array.join("")

console.log('newArray2', newArray2); *// 1234*

let newArray3 = array.join(" ")

console.log('newArray3', newArray3); *// 1 2 3 4*

## 11. indexOf: 从前往后 查找当前元素在数组中的位置 如果有返回下标 无返回-1

let array = [1, 2, 3, 4]

let index = array.indexOf(2)

console.log('index', index); *// 1*

let index1 = array.indexOf(9)

console.log('index1', index1); *// -1*

## 12. lastIndexOf: 从后往前 查找当前元素在数组中的位置 如果有返回下标 无返回-1

## 13. sort: 数组排序 改变原数组

let array = [1, 12, 3, 44, 15]

let newArray = array.sort() *// 默认字符串的排序规则*

console.log('newArray', newArray); *// [1, 12, 15, 3, 44]*

console.log('array', array); *// [1, 12, 15, 3, 44]*

let array1 = [1, 12, 3, 44, 15]

let newArray1 = array1.sort((a, b) => a - b) *// 从小到大*

console.log('newArray1', newArray1); *// [1, 3, 12, 15, 44]*

let array2 = [1, 12, 3, 44, 15]

let newArray2 = array2.sort((a, b) => b - a) *// 从大到小*

console.log('newArray2', newArray2); *// [44, 15, 12, 3, 1]*

## 14. every: 数组每项都为真 最终为真 返回的是布尔值

let array = [1, 12, 3, 44, 15]

let res = array.every((item, index) => item > 0)

console.log('res===', res); *// true*

console.log('array==', array); *// [1, 12, 3, 44, 15]*

## 15. some: 数组只要某一项为真 最终为真 返回的是布尔值

let array = [1, 12, 3, 44, 15]

let res = array.some((item, index) => item > 40)

console.log('res===', res); *// true*

console.log('array==', array); *// [1, 12, 3, 44, 15]*

## 16. filter: 对数组进行过滤 返回符合条件的新数组

let array = [1, 12, 3, 44, 15]

let res = array.filter((item, index) => item > 10)

console.log('res===', res); *// [12, 44, 15]*

console.log('array==', array); *// [1, 12, 3, 44, 15]*

## 17. map: 对数组的每个成员做一些操作 返回操作后的新数组

let array = [1, 12, 3, 44, 15]

let res = array.map((item, index) => item + 1)

console.log('res===', res); *// [2, 13, 4, 45, 16]*

console.log('array==', array); *// [1, 12, 3, 44, 15]*

## 18. forEach:  对数组的每个成员做一些操作 没有返回值 需要自己定义新变量承接结果

let array = [1, 12, 3, 44, 15]

let newArray = []

let res = array.forEach((item, index) => {

newArray.push(item + 1)

})

console.log('res===', res); *// undefined*

console.log('array==', array); *// [1, 12, 3, 44, 15]*

console.log('newArray==', newArray); *// [2, 13, 4, 45, 16]*

# ES6 新增方法:

## Array.of() 返回一个由参数组成的数组

let arr = Array.of(1,2,3); // [1,2,3]

## Array.from() 将参数对象转换为数组,不改变原对象，而是返回一个新的数组对象

对象的要求:

* 1. 拥有 length 属性
  2. 可迭代对象,即必须要有[Symbol.iterator]接口的数据类型结构 例如：Set，Map,NodeList，字符串等

参数:

* 1. (必需) 要转化为数组的对象。
  2. (可选) 类似于 map 方法，即对于每个元素进行的处理，并会在处理后放入返回的新的数组中
  3. (可选) 用来绑定执行第二参数方法时的this指向.

let a = {0: '1', 1: '2', 2: '3', 'length': 3};

let arr = Array.from(a); //['1','2','3']

// 字符串

let arr = Array.from('array'); //['a','r','r','a','y']

// Set、Map

let arrset = Array.from(new Set(['func', window])); //['func', window]

let arrMap = Array.from(new Map([1,2],[3,4],[5,6])); // [[1,2], [3,4],[5,6]]

// 类 map 方法的使用

let theArr = Array.from([1,2,3], x => ++x); // [2,3,4]

## copyWithin() 数组的指定位置复制[改变原数组、ES6]

arr.copyWithin(index, start, end)

参数:

1. index(必需): 开始替换元素的位置，负值表示从后面查起。
2. start(可选): 从该位置读取数据，默认值为0，负值表示从后面查起。
3. end(可选): 到该位置停止读取，默认为数组长度，负值表示从后面查起，但实际上复制的是到这位之前的元素。

无返回值

使用场景:

let arr = [1,2,3,4,5];

arr.copyWithin(0,2,4); //复制了几个元素，就替代几个元素

console.log(arr); // [3,4,3,4,5]

## fill()数组的填充 [改变原数组、ES6]

arr.fill(value, start, end)

参数:

1. value(必需): 用来填充的值。
2. start(可选): 填充的起始位置，默认为0。
3. end(可选)：填充的结束为止，默认为 数组的长度。

无返回值

使用场景:

let arr = new Array(3);

arr.fill(1);

console.log(arr); //[1,1,1]

arr.fill(2,1,3);

console.log(arr); //[1,2,2]

## toString() / toLocaleString()[不改变原数组]

arr.toString() / arr.toLocaleString();

无参数

返回值：转换后的字符串。

两个与join有点类似，只是 toLocaleString相当于先调用数组元素的 toLocaleString方法之后再进行join,toString()用逗号隔开数组元素拼接为字符串。

使用场景:

let arr = [1,2,3];

let arrStr = arr.toString();

console.log(arrStr); //"1,2,3"

let arrLocale = [new Date(),1,2];

let LocaleStr = arrLocale.toLocaleString();

console.log(LocaleStr); //"2019/4/16 下午5:05:43,1,2"

## includes() 查找数组是否包含某个元素。[不改变原数组, ES7]

arr.includes(value);

参数:

1. value(必需)：要查找的元素值
2. fromIndex(可选): 从此处开始查找，接受负值，默认为0，若超过数组长度则函数返回-1。

返回值：布尔型

与indexOf的区别:

1. indexOf 不能识别 NaN,但 includes 可以。
2. 当我们只需要查找是否含有时，若返回索引可能是0的情况，不方便我们直接进行一些判断操作。

let arr = [1,NaN, 100,'42'];

console.log(arr.includes(1)); //true

console.log(arr.includes(NaN)); //true

console.log(arr.includes(1,3)); //false

## flat()深度遍历展开数组

arr.flat(depth);

参数: depth(可选): 提取嵌套数组的结构深度，默认为1。

返回值：展开后的新数组。

let arr = [1,2,[3,4,[5,6]]]

let one = arr.flat();

console.log(one); //默认值为1， 所以只能展开一层嵌套 [1,2,3,4,[5,6]]

let two = arr.flat(2);

console.log(two); //[1,2,3,4,5,6]

// 若不清楚有多少层嵌套，可以直接用 Infinity 设置，就可全部展开

let inf = arr.flat(Infinity);

console.log(inf); //[1,2,3,4,5,6]

// flat方法会移除数组中的空白项

let arr2 = [1,2,3,,5];

console.log(arr2.flat()); //[1,2,3,5]

分享一个可以用这个解决的面试题，还是通过这道题发现原来有flat这个方法 :joy:

题目: 将一个多维数组扁平化并去重，之后得到一个升序数组。

let arr = [1,2,3,4,5,[2,4,5,8,[44,88,1,3,4,8,5,7,6,[123],111],15],2,8,7];

let newArr = Array.from(new Set(arr.flat(Infinity))).sort((a,b) => a - b)

console.log(newArr); //[1, 2, 3, 4, 5, 6, 7, 8, 15, 44, 88, 111, 123]

## keys() 遍历键名 / values() 遍历键值/ entries() 遍历键值对

arr.keys() / arr.values() / arr.entries()

无参数

返回值: 一个可遍历的数组对象[Array Iterator],所以我们是不可以直接打印这个对象的，而是可以遍历它。

let arr = [9,8,7,6,5,4,3,2,1]

for(let index of arr.keys()){

console.log(index); //0,1,2,3,4...依次打印

}

for(let value of arr.values()){

console.log(value); //9,8,7,6,5....依次打印

}

for(let [index, value] of arr.entries()){

console.log(index,value); // 0,9 1,8 2,7

}

当然也可以手动调用遍历器

let arr =["one","two","three"];

let result = arr.entries();

console.log(result.next().value); //[0, "one"]

console.log(result.next().value); //[1,"two"]

console.log(result.next().value); //[2, "three"]