1.修改this指向

题目描述

封装函数f，使f的this指向指定的对象

题解

```
解法一：apply修改this的作用域
function bindThis(f,oTarget){
    return function(){
        return f.apply(oTarget,arguments)
    }
}

解法二：call修改this的作用域
function bindThis(f,oTarget){
    return function(){
        return f.call(oTarget,...arguments)
    }
}

解法三：bind修改this的作用域
function bindThis(f,oTarget){
    return function(){
        return f.bind(oTarget,...arguments)()
    }
}

解法三衍生：直接返回bind函数
function bindThis(f,oTarget){
    return f.bind(oTarget)
}

解法四：自己写bind
function bindThis(f,oTarget){
    var args = Array.prototype.slice.call(arguments, 1) //从第二个参数截取
    return function(){
        return f.apply(oTarget,
Array.prototype.slice.call(arguments).concat(args));
    }
}
```

相关知识点

1. apply、call、bind区别

> apply、call、bind的作用都是修改执行上下文
>
> apply、call都是返回函数立即执行的结果，其中apply第二个参数之后是数组，call第二个参数之后是单个的值。
>
> bind返回的是函数，需要手动执行结果。第二个参数之后是单个的值。

2. apply详细介绍

1. 简介

> `apply()` 方法调用一个具有给定`this`值的函数，以及以一个数组（或类数组对象）的形式提供的参数。

2. 语法

```
func.apply(thisArg, [argsArray])

参数：
thisArg
必选的。在 func 函数运行时使用的 this 值。请注意，this可能不是该方法看到的实际值：如果这个函数处于非严格模式下，则指定为 null 或 undefined 时会自动替换为指向全局对象，原始值会被包装。
argsArray
可选的。一个数组或者类数组对象，其中的数组元素将作为单独的参数传给 func 函数。如果该参数的值为 null 或  undefined，则表示不需要传入任何参数。从ECMAScript 5 开始可以使用类数组对象。

返回值：
调用有指定this值和参数的函数的结果。
```

Function.prototype.apply() - JavaScript | MDN (mozilla.org)

3. call详细介绍

1. 简介

> `**call()**` 方法使用一个指定的 `this` 值和单独给出的一个或多个参数来调用一个函数。

2. 语法

```
function.call(thisArg, arg1, arg2, ...)

参数：
thisArg
可选的。在 function 函数运行时使用的 this 值。请注意，this可能不是该方法看到的实际值：如果这个函数处于非严格模式下，则指定为 null 或 undefined 时会自动替换为指向全局对象，原始值会被包装。
arg1, arg2, ...
指定的参数列表。

返回值：
使用调用者提供的 this 值和参数调用该函数的返回值。若该方法没有返回值，则返回undefined。
```

Function.prototype.call() - JavaScript | MDN (mozilla.org)

4. bind详细介绍

1. 简介

> bind()方法创建一个新的函数，在 `bind()` 被调用时，这个新函数的 `this` 被指定为 `bind()` 的第一个参数，而其余参数将作为新函数的参数，供调用时使用。

2. 语法

```
function.bind(thisArg[, arg1[, arg2[, ...]]])

参数：
thisArg
调用绑定函数时作为 this 参数传递给目标函数的值。 如果使用new运算符构造绑定函数，则忽略该值。当使用 bind 在 setTimeout 中创建一个函数（作为回调提供）时，作为 thisArg 传递的任何原始值都将转换为 object。如果 bind 函数的参数列表为空，或者thisArg是null或undefined，执行作用域的 this 将被视为新函数的 thisArg。
arg1, arg2, ...
当目标函数被调用时，被预置入绑定函数的参数列表中的参数。

返回值：
返回一个原函数的拷贝，并拥有指定的 this 值和初始参数。
```

3. bind有如下三个功能点

> 1. 改变原函数的 this 指向，即绑定上下文，返回原函数的拷贝
> 2. 当 绑定函数 被调用时，bind的额外参数将置于实参之前传递给被绑定的方法。
> 3. 注意，一个 绑定函数也能使用 new 操作符创建对象,这种行为就像把原函数当成构造器，thisArg 参数无效。也就是 new 操作符修改 this 指向的优先级更高。

4. 自己实现bind方法

> - 输入：接受一个或者多个参数，第一个是要绑定的上下文，额外参数当作绑定函数的前置参数。
> - 输出：返回原函数的拷贝，即返回一个函数，这个函数呢具备原函数的功能
>
> ```
> // 定义这个方法为myBind
> Function.prototype.myBind = function(thisArg) {
>   if (typeof this !== 'function') {
>     return;
>   }
>   var _self = this;
>   var args = Array.prototype.slice.call(arguments, 1) //从第二个参数截取
>   return function() {
>     return _self.apply(thisArg, args.concat(Array.prototype.slice.call(arguments))); // 注意参数的处理
>   }
> }
> ```

2.获取url参数

题目描述

> 获取 url 中的参数
>
> 1. 指定参数名称，返回该参数的值 或者 空字符串
> 2. 不指定参数名称，返回全部的参数对象 或者 {}
> 3. 如果存在多个同名参数，则返回数组
>
> 输入：http://www.nowcoder.com?key=1&key=2&key=3&test=4#hehe key
>
> 输出：[1, 2, 3]

题解

```
function getUrlParam(sUrl, sKey) {
        let newArr = []
        let newObj = {}
        // 获取?号后面#号前面的值
        let query = sUrl.split('#')[0].split('?')[1]
        // 如果query存在
        if (query) {
          let arr = query.split('&')//分解字符串
          for(let i = 0 ; i < arr.length; i++) {
            if (arr[i]) {
              arr[i] = arr[i].split('=')
              // 数组
              if (sKey !== undefined) {
                if(arr[i][0] === sKey) {
                  newArr.push(arr[i][1])
                }
              // 对象
              } else {
                if(arr[i][0] in newObj) {
                  newObj[arr[i][0]].push(arr[i][1])
                } else {
                  newObj[arr[i][0]] = [arr[i][1]]
                }
              }
            }
          }
          // 判断sKey有没有值
          if(sKey !== undefined) {
            switch(newArr.length) {
              case 0 : return '';break;
              case 1 : return newArr[0]; break;
              default: return newArr;break;
            }
          } else {
            return newObj
          }
        // 如果query不存在，判断sKey是否存在，如果存在就返回空对象，如果不存在就返回空字符串
        } else {
          return sKey !== undefined ? {} : ''
        }
      }
```

相关知识点

1. url的组成部分

> https://user:pass@www.baidu.com:80/index.html?type=1&name=2#haha
> http/https 是协议
> user:pass@ 是登录认证
> www.baidu.com 是域名，服务器地址
> :80 是端口号
> /index.html 是请求资源文件路径
> ?type=1&name=2 是查询字符串，携带参数，给服务器传的内容。
> #haha 是哈希，片段标识符

2. split方法

> 字符串分割成数组的方法，里面的参数是以什么分割，如果不传就是空字符串为分割，返回值是一个数组。

3. query部分可以使用正则

3.dom节点查找

题目描述

> 查找两个节点的最近的一个共同父节点，可以包括节点自身
>
> 输入描述:
> oNode1 和 oNode2 在同一文档中，且不会为相同的节点

题解

```
function commonParentNode(oNode1, oNode2) {
    if(oNode1.contains(oNode2)) {
        return oNode1
    } else {
        return commonParentNode(oNode1.parentNode,oNode2)
    }
}
```

相关知识点

- contains API

> 查看dom元素包含关系，包含返回true，不包含返回false
> Node.contains - Web API 接口参考 | MDN (mozilla.org)

- 递归(参考数据结构树)

4.根据包名，在指定空间中创建对象

题目描述

> 根据包名，在指定空间中创建对象
> 输入描述:
> namespace({a: {test: 1, b: 2}}, ‘a.b.c.d’)
> 输出描述:
> {a: {test: 1, b: {c: {d: {}}}}}
>
> 命名空间
> 命名空间是一种代码封装技术,代码中的每个成员，都是自己的活动空间，彼此互不干扰。
>
> 首先理解题意，namespace($1, $2)函数中第一个参数是现有命名空间中的对象，而第二个参数是需要此空间实现的结构（即对象的包含关系），可知是a包含b包含c包含d；

题解

```
function namespace(oNamespace, sPackage) {
    let scope = sPackage.split('.')//拆分数组
    let ns = oNamespace//建立对象引用
    for(let i = 0; i< scope.length; i++) {
        // 如果对象中没有该元素，或者不是对象，那么就置为空对象
       if(!ns.hasOwnProperty(scope[i]) ||
Object.prototype.toString.call(ns[scope[i]]) !== '[object Object]') {
            ns[scope[i]] = {}
        }
        // 然后继续往下找
        ns = ns[scope[i]]
    }
    return oNamespace
}
```

相关知识点

- 判断对象的自身是否有某属性(hasOwnProperty)
- hasOwnProperty / typeof / in / instanceof 的区别

  - hasOwnProperty 是判断对象自身有没有某属性，不包含原型链的方法。
  - in 是判断对象在自身和原型链上有没有该方法。
  - instanceof 是判断对象在原型链上有没有该方法。
  - typeof 判断操作数的类型，但是null也会判断为"object"

- 准确判断某值的类型

  Object.prototype.toString.call(123) === “[object Number]”
  Object.prototype.toString.call(‘aaa’) === “[object String]”
  Object.prototype.toString.call(true) === “[object Boolean]”
  Object.prototype.toString.call(undefined) === “[object Undefined]”
  Object.prototype.toString.call(null) === ‘[object Null]’
  Object.prototype.toString.call({}) === ‘[object Object]’
  Object.prototype.toString.call([]) === ‘[object Array]’
  Object.prototype.toString.call(Math) === “[object Math]”
  Object.prototype.toString.call(new Date()) === “[object Date]”
  Object.prototype.toString.call(new RegExp) === “[object RegExp]”

- 递归(对象的嵌套参考数据结构中的树)
- JS命名空间（namespace） - 简书 (jianshu.com)

5.数组去重

题目描述

为 Array 对象添加一个去除重复项的方法
输入
[false, true, undefined, null, NaN, 0, 1, {}, {}, ‘a’, ‘a’, NaN]
输出
[false, true, undefined, null, NaN, 0, 1, {}, {}, ‘a’]

题解

```
// 方法一：终极思路
Array.prototype.uniq = function () {
    return [...new Set(this)]
}

// 方法二：普通思路，遍历之后比较值
Array.prototype.uniq = function () {
    let arr = []
    let flag = true
    this.forEach(value => {
        // == -1 有两种情况，一种是NaN，一种是有相同值
        if(arr.indexOf(value) === -1) {//等于-1证明数组arr中没有添加
            // 如果是NaN
            if(value !== value) {//判断是否为NAN，因为精度的原因，NaN === NaN返回false
                // flag是标记，第一个NaN就进，之后的就不进去
                if(flag){
                  arr.push(value)
                  flag = false
                }
            } else {
                arr.push(value)
            }
        }
    })
    return arr
}
```

相关知识点

- uniq方法中的this指向哪里?

> Array构造函数的原型方法中的this指的是数组实例。

- Set的特性

> Set存储的成员是唯一的，不是重复的，如果有重复会自动过滤掉.
> ES6（七）—— Set & Map

- (NaN === NaN) => false

> NaN： is not a number，不等于自己
> typeof NaN => number
> Object.prototype.toString.call(NaN) => “[object Number]”
> ES6 新增方法：Number.isNaN() 用来判断是否属于数字

6.斐波那契数列

题目描述

> 用 JavaScript 实现斐波那契数列函数,返回第n个斐波那契数。 f(1) = 1, f(2) = 1 等

题解

```
// 方法一：递归思路
function fibonacci(n) {
    if(n === 0) return 0
    if(n === 1 || n === 2) return 1
    return fibonacci(n-1) + fibonacci(n-2)
}

// 方法二：迭代思路
function fibonacci(n) {
    let num1 = 1
    let num2 = 1
    let sum = 0
    for(let i = 3; i <= n; i++) {
        sum = num1 + num2
        num1 = num2
        num2 = sum
    }
    return sum
}

// 上面写法可以过oj，但是如果数字大点就超级慢，使用缓存很可
// 方法三：递归优化思路
function fibonacci(n,cache = {}) {
    // 有缓存就直接读缓存
    if(n in cache) return cache[n]
    if(n === 1 || n === 2) {
        cache[n] = 1
        return 1
    }
    // 没有缓存算完之后存入缓存
    let temp = fibonacci(n-1, cache) + fibonacci(n-2,cache)
    cache[n] = temp
    return temp
}
```

相关知识点

- 递归

7.时间格式化输出

题目描述

> 按所给的时间格式输出指定的时间
> 格式说明
> 对于 2014.09.05 13:14:20
> yyyy: 年份，2014
> yy: 年份，14
> MM: 月份，补满两位，09
> M: 月份, 9
> dd: 日期，补满两位，05
> d: 日期, 5

> HH: 24制小时，补满两位，13
> H: 24制小时，13
> hh: 12制小时，补满两位，01
> h: 12制小时，1
> mm: 分钟，补满两位，14
> m: 分钟，14
> ss: 秒，补满两位，20
> s: 秒，20
> w: 星期，为 [‘日’, ‘一’, ‘二’, ‘三’, ‘四’, ‘五’, ‘六’] 中的某一个，本 demo 结果为 五
>
> 输入
> formatDate(new Date(1409894060000), ‘yyyy-MM-dd HH:mm:ss 星期w’)
>
> 输出
> 2014-09-05 13:14:20 星期五

题解

```
function formatDate(t,str) {
    let year = ''+t.getFullYear()
    let month = t.getMonth() + 1
    let day = t.getDate()
    let hour = t.getHours()
    let minutes = t.getMinutes()
    let second = t.getSeconds()
    let week = ['日','一','二','三','四','五','六']
    let date = {
      'yyyy': year,
      'yy': year.slice(2),
      'MM': ten(month),
      'M': month,
      'dd': ten(day),
      'd': day,
      'HH': ten(hour),
      'H': hour,
      'hh': ten(hour % 12),
      'h': hour % 12,
      'mm': ten(minutes),
      'm': minutes,
      'ss': ten(second),
      's': second,
      'w': week[t.getDay()]
    }
    for(let key in date) {
      str = str.replace(key,date[key])
    }
    return str
  }

  // 不足10的前面要加0
  let ten = num  => num >= 10 ? num : '0' + num
```

相关知识点

- 获取年月日周时分秒的系统API
- 格式统一处理
- 字符串替换 (replace)

8.获取字符串的长度

题目描述

> 如果第二个参数 bUnicode255For1 === true，则所有字符长度为 1
> 否则如果字符 Unicode 编码 > 255 则长度为 2
> 输入
> hello world, 牛客', false
> 输出 17

题解

```
function strLength(s, bUnicode255For1) {
    if(bUnicode255For1) return s.length;
    let len = s.length
    for(let i = 0; i < s.length; i++) {//为什么用i<len会报错--因为len会改变
        if(s[i].charCodeAt() > 255) len++
    }
    return len
}
```

相关知识点

- 获取字符的 Unicode 编码 API —— str.charCodeAt()

9.邮箱字符串判断

题目描述

> 判断输入是否是正确的邮箱格式
>
> 1.不限制长度
>
> 2.不限制大小写
>
> 3.邮箱开头必须是数字或字符串
>
> 4.邮箱中可以使用字母、数字、点号、下划线、减号，但是不能连写点号、下划线、减号，如 abc_-de@q_.q.com
>
> 5.@符号前后不能为点号、下划线、减号

题解

```
// ^ 表示开头
// [] 表示匹配字符的范围
// \w 表示正常符号 [0-9a-zA-Z_]
// \. 是对任意符.进行转义，表示字符.
// + 表示前面的表达式，一次到多次
function isAvailableEmail(sEmail) {
    return /^[\w\.]+@\w+\.\w+/.test(sEmail)
}
```

相关知识点

- 邮箱格式
- 正则表达式的规则和匹配

10.颜色字符串转换

题目描述

> 将 rgb 颜色字符串转换为十六进制的形式，如 rgb(255, 255, 255) 转为 #ffffff
>
> rgb 中每个 , 后面的空格数量不固定
> 十六进制表达式使用六位小写字母
> 如果输入不符合 rgb 格式，返回原始输入
> 输入：‘rgb(255, 255, 255)’
> 输出：#ffffff

题解

```
function rgb2hex(sRGB) {
    // 正则匹配获取三个数值
    let reg = sRGB.match(/^rgb\((\d+),\s*(\d+),\s*(\d+)\)/)
    if(!reg) return sRGB;//输入不符合规范
    // 字符串拼接
    let str = '#'
    for(let i = 1; i < reg.length; i++) {
      // 将字符串转成数字
      let m = parseInt(reg[i])
      if (m >= 0 && m <= 255) {
        // 然后转化成16进制
        str += (m >= 16 ? m.toString(16) : '0' + m.toString(16))
      } else {//输入不符合规范
        return sRGB
      }
    }
    return str
  }
```

相关知识点

- toString的进制转换

  > 颜色是16进制，所以toString(16)可以得到结果

- 字符串中如何截取数字(不限于正则)

11.将字符串转换为驼峰格式

题目描述

> css 中经常有类似 background-image 这种通过 - 连接的字符，通过 javascript 设置样式的时候需要将这种样式转换成 backgroundImage 驼峰格式，请完成此转换功能
>
> 以 - 为分隔符，将第二个起的非空单词首字母转为大写
> -webkit-border-image 转换后的结果为 webkitBorderImage
> 输入： ‘font-size’
> 输出： fontSize

题解

```
function cssStyle2DomStyle(sName) {
    let arr = sName.split('-')
    for(let i = (arr[0] ? 1 : 2); i < arr.length; i++) {
        arr[i] = arr[i].slice(0,1).toUpperCase()+arr[i].slice(1)
    }
    return arr.join('')
}
```

相关知识点

- 数组常用方法

  - split/join
  - toUpperCase() —— toUpperCase将小写字符转成大写，toLowerCase将大写字符转成小写
  - slice

12.字符串字符统计

题目描述

> 统计字符串中每个字符的出现频率，返回一个 Object，key 为统计字符，value 为出现频率
>
> 不限制 key 的顺序
> 输入的字符串参数不会为空
> 忽略空白字
> 输入 : ‘hello world’
> 输出： {h: 1, e: 1, l: 3, o: 2, w: 1, r: 1, d: 1}

题解

```
function count(str) {
    let obj = {}
    for (let i = 0; i < str.length; i++) {
        // 去掉空白字符
        if(str[i] !== ' ') {
            // 如果有该属性就+1，没有就设置值为1
            obj[str[i]] = obj.hasOwnProperty(str[i]) ? obj[str[i]] + 1 : 1
        }
    }
    return obj
}
```

相关知识点

- 对象赋值

13.加粗文字

题目描述

> 使用一个标签将“牛客网”三个字加粗显示

题解

```
<!--方法一：html-->
<p><strong>牛客网</strong>，程序员必备求职神器</p>

<!--方法二：js-->
let p = document.getElementByTagName('p')
let text = p.innterHTML
p.innterHTML = text.replace('牛客网','<strong>牛客网</strong>')
```

相关知识点

- 获取元素及元素内容
- 加粗标签

14.段落标识

题目描述

> 请将下面这句话以段落的形式展示在浏览器中——“牛客网是一个专注于程序员的学习和成长的专业平台。”

题解

```
<!--方法一：html-->
<p>牛客网是一个专注于程序员的学习和成长的专业平台。</p>

<!--方法二：js-->
let p = document.createElement('p')
p.innerHTML = '牛客网是一个专注于程序员的学习和成长的专业平台。'
document.querySelector('body').append(p)
```

相关知识点

- 创建标签createElement
- 将元素添加到body中 append()

15.设置文字颜色

题目描述

> 请使用嵌入样式将所有p标签设置为红色文字

题解

```
<!--方法一：行内样式-->
<p style="color:red;">欢迎来到牛客网</p>
<p style="color:red;">在这里，我们为你提供了IT名企的笔试面试题库</p>
<p style="color:red;">在这里，我们以题会友</p>
<p style="color:red;">QQ群号：272820159</p>

<!--方法二：css样式-->
<style>
    p {
        color: red;
    }
</style>
<p>欢迎来到牛客网</p>
<p>在这里，我们为你提供了IT名企的笔试面试题库</p>
<p>在这里，我们以题会友</p>
<p>QQ群号：272820159</p>

<!--方法一：js-->
let p = document.querySelectorAll('p')
for(let i = 0; i < p.length; i++) {
    p[i].style.color = 'red'
}
```

相关知识点

- 获取所有p元素：querySelectorAll
- 给元素设置style样式：dom.style.color

16.查找数组元素位置

题目描述

> 找出元素 item 在给定数组 arr 中的位置
> 输出描述:
> 如果数组中存在 item，则返回元素在数组中的位置，否则返回 -1
> 输入：[1, 2, 3, 4], 3
> 输出：2

题解

```
// 方法一：简单遍历
function indexOf(arr, item) {
    for(let i = 0; i < arr.length; i++) {
        if(arr[i] === item) return i
    }
    return -1
}

// 方法二： ES6新增数组方法
function indexOf(arr, item) {
    return arr.findIndex(val => val === item)
}
```

相关知识点

- 数组遍历
- 函数返回值

17.数组求和

题目描述

> 计算给定数组 arr 中所有元素的总和
> 输入描述:
> 数组中的元素均为 Number 类型
> 输入： [1, 2, 3, 4]
> 输出： 10

题解

```
// 方法一：简单方法，普通for循环这里不多加介绍
function sum(arr) {
    let count = 0
    arr.forEach((value, index) => {
        count+=value
    })
    return count
}

// 方法二：reduce（函数式编程）
function sum(arr) {
    return arr.reduce((prev, item) => item + prev,0)
```

```
}

//方法三：eval
function sum(arr) {
    return eval(arr.join("+"));
};
```

相关知识点

- 数组遍历
- reduce

18.移除数组中的元素

题目描述

> 移除数组 arr 中的所有值与 item 相等的元素。不要直接修改数组 arr，结果返回新的数组
> 输入 : [1, 2, 3, 4, 2], 2
> 输出 : [1, 3, 4]

题解

```
// 方法一：简单遍历
function remove(arr, item) {
    let newArr = []
    arr.forEach(value =>{
        if(value !== item) newArr.push(value)
    })
    return newArr
}

// 方法二：filter过滤方法
function remove(arr, item) {
    return arr.filter(val => val !== item)
}

// 方法三：新数组中使用splice删除
function remove(arr, item) {
    let newArr = arr.slice(0)
    for(let i = newArr.length - 1; i >= 0 ; i--) {
        if(newArr[i] === item) newArr.splice(i, 1)
    }
    return newArr
}
```

相关知识点

- 数组方法哪些是在原数组中改的，哪些是返回新数组的?

> 返回新数组的API
> slice \ map \ filter \ reduce \ concat ...
> 返回原数组的API
> push \ unshift \ shift \ pop \ splice \ sort \ reverse ...

19.移除数组中的元素

题目描述

> 移除数组 arr 中的所有值与 item 相等的元素，直接在给定的 arr 数组上进行操作，并将结果返回
> 输入: [1, 2, 2, 3, 4, 2, 2], 2
> 输出: [1, 3, 4]

题解

```
// 倒着遍历不用考虑数组长度
function removeWithoutCopy(arr, item) {
    for(let i = arr.length - 1; i >= 0; i--) {
        if(item === arr[i]) arr.splice(i,1)
    }
    return arr
}
//方法二：利用indexOf找出是否有和item相等的元素
function removeWithoutCopy(arr, item) {
    while(arr.indexOf(item) != -1){
        arr.splice(arr.indexOf(item),1);
    }
    return arr;
}
```

相关知识点

- 遍历 + 修改数组长度

20.添加元素

题目描述

> 在数组 arr 末尾添加元素 item。不要直接修改数组 arr，结果返回新的数组
> 输入: [1, 2, 3, 4], 10
> 输出: [1, 2, 3, 4, 10]

题解

```
// 方法一：简单迭代
function append(arr, item) {
    let newArr = []
    arr.forEach(val => newArr.push(val))
    newArr.push(item)
    return newArr
}

// 方法二：slice
function append(arr, item) {
    let arr1 = arr.slice(0)
    arr1.push(item)
    return arr1
}
```

```
/// 方法三：concat
function append(arr, item) {
    return arr.concat(item)
}
```

相关知识点

- 合并新元素，返回新数组

> JavaScript常用的数组方法
>
> - 截取方法中，字符串有三种方法slice / substring / substr ，数组方法有两个slice / splice
>   字符串的slice 和 substring 是要开始和结束的索引，substr 是要开始索引和长度
>   数组的slice是要开始和结束索引，但是splice是要开始索引和长度
> - 搜索元素方法中，数组和字符串都有indexOf方法，但是字符串多出来两种方法charAt和charCodeAt
>   其中indexOf是返回索引，charAt是返回索引对应的值，charCodeAt是返回对应值的ASCII码值。
> - 数组的遍历有4中方法，map,every,foreach,some,filter
>   其中foreach开始就停不下来，全部遍历。every遍历一个就判断一下，true就继续遍历下一个，false就跳出。map就是边遍历边运算。some返回的是布尔值，符合就是true，不符合就是false。filter返回的是符合元素组成的数组。
> - 增加数组元素，前面unshift，后面push
>   移除数组元素，前面shift，后面pop
> - 数组和字符串都有concat方法，各自连接各自的，是数组就连接到数组，字符串就连接成字符串
> - 比较重要的两个就是数组和字符串之间的转化的两个方法
>   join是数组转字符串，split是字符串转数组

补充：数组知识点

> **Array.prototype**
>
> Array.prototype 属性表示构造函数的原型，并允许您向所有Array对象添加新的属性和方法。

```
Array.prototype 属性表示构造函数的原型，并允许您向所有Array对象添加新的属性和方法。

    /*
 如果JavaScript本身不提供 first() 方法，
 添加一个返回数组的第一个元素的新方法。
 */

 if(!Array.prototype.first) {
     Array.prototype.first = function() {
         return this[0];
     }
 }


Array.prototype本身也是一个 Array

    Array.isArray(Array.prototype); // true
 //属性
```

```
Array.prototype.constructor
//所有的数组实例都继承了这个属性，它的值就是 Array，表明了所有的数组都是由 Array 构造出来的。
Array.prototype.length
//上面说了，因为 Array.prototype 也是个数组，所以它也有 length 属性，这个值为 0，因为它是个空数组。
```

- 判断是不是数组的方式

  Array.isArray( );

  - 静态方法，是数组构造函数的方法
  - obj是需要检测的值，如果是数组，返回true，否则返回false

    ```
    // 下面的函数调用都返回 true
    Array.isArray([]);
    Array.isArray([1]);
    Array.isArray(new Array());
    // 鲜为人知的事实：其实 Array.prototype 也是一个数组。
    Array.isArray(Array.prototype);

    // 下面的函数调用都返回 false
    Array.isArray();
    Array.isArray({});
    Array.isArray(null);
    Array.isArray(undefined);
    Array.isArray(17);
    Array.isArray('Array');
    Array.isArray(true);
    Array.isArray(false);
    Array.isArray({ __proto__: Array.prototype });、
    ```

  - 存在兼容问题(IE8及以下不支持)

    ```
    //Polyfill
    //假如不存在 Array.isArray()，则在其他代码之前运行下面的代码将创建该方法。
    if (!Array.isArray) {
      Array.isArray = function(arg) {
        return Object.prototype.toString.call(arg) === '[object Array]';
      };
    }
    ```

- (Object.prototype)toString.call(arr) -> [object Array]

  - 转化成字符串是"[object Array]",可以作为判断条件。
  - Object.prototype.toString.call(obj).slice(8,-1); -> ==='Array'
  - ({}).toString.call(function(){}).slice(8,-1); -> ==='Function'
  - slice截取，前面的是从第八个开始，截取到倒数的第二个。

- instanceof

  对象 instanceof 数据类型

  - console.log(obj instanceof Array);

- （不严谨）多个页面进行判断，会有问题

```
//iframe
  /*B页面嵌套到A页面中，每个页面都有一个top属性，top属性一直都指向A页面的window，所以在A页面定义的函数fn，暴露在B页面的全局环境中，在B页面中也可以调用。*/
  top.fn(); //(就是调用页面A的fn函数)

//那么问题来了：
B页面：   top.fn([]);
A页面：   function fn(arr){
            console.log(arr instanceof Array);
            }  //此时会成为false

//要直接访问A页面，访问B页面会报错，因为只打开的页面的top指向自己的window，此时调用了没有定义的函数。
//防止被嵌套：if( top != window){
      top.location.href = 'inner-B.html';} //跳转到自己的地址
```

- 数组长度
  - 数组的length属性总是比数组中定义的最后一个元素的下标大一，表示数组中元素的个数。
  - 数组的length属性在创建数组的时候初始化，在添加新元素的时候数组长度改变

```
//如果函数中没有参数，a为空数组
var a = new Array();   // a.length 被初始化为 0

//如果函数参数是一个，参数表示函数的长度
var b = new Array(10); // b.length 被初始化为 10

//如果函数参数是两个及以上，参数表示数组内容
var c = new Array("one", "two", "three");  // c.length 被初始化为 3
c[3] = "four";         // c.length 被更新为 4
c[10] = "blastoff";    // c.length 变为 11
```

  - 设置属性length的值可以改变数组的大小，设置值小则被从后截断，设置值大则剩下的值都为undefined

```
var a = new Array("one", "two", "three");
a.length = 2;  //["one", "two"]
a.length = 5;  //["one", "two", undefined × 3]
```

- **遍历数组**

map

var new_array = array.map (function(value,index,array){ },thisArg);

  - 遍历数组，能够将数组转化为一个新的数组，新数组的值由map方法回调函数的返回值决定。
  - 回调函数的第一个参数是数组的值，第二个参数是索引,第三个参数是被调用的数组。thisArg可选，执行 callback函数时 使用的this值。

- 如果 thisArg 参数有值，则每次 callback 函数被调用的时候，this都会指向thisArg参数上的这个对象。如果省略了 thisArg参数,或者赋值为null或undefined，则 this 指向全局对象 。
- 返回值是新的数组

```
 var arrNew = arr.map(function(value,index){
     console.log('索引是'+index+",内容是："+value+);
   })

 let numbers = [1, 5, 10, 15];
 let roots = numbers.map(function(x) {
      return x * 2;
   });
// roots is now [2, 10, 20, 30]
// numbers is still [1, 5, 10, 15]
```

- 求数组中每个元素的平方根

```
var numbers = [1, 4, 9];
var roots = numbers.map(Math.sqrt);
/* roots的值为[1, 2, 3], numbers的值仍为[1, 4, 9] */
```

- 使用 map 重新格式化数组中的对象

```
var kvArray = [{key:1, value:10},
               {key:2, value:20},
               {key:3, value: 30}];
var reformattedArray = kvArray.map(function(obj){
  //obj指的是每一个数组元素，是一个对象
   var rObj = {};
   rObj[obj.key] = obj.value;
   return rObj;
});
// reformattedArray is now [{1:10}, {2:20}, {3:30}],
// kvArray is still [{key:1, value:10}, {key:2, value:20}, {key:3,
value: 30}]
```

- 将数组中的单词转换成对应的复数形式

```
var words = ["foot", "goose", "moose", "kangaroo"];
//定义函数
function fuzzyPlural(single) {
  //所有的o变成e
  var result = single.replace(/o/g, 'e');
  if( single === 'kangaroo'){
    result += 'se';
  }
  return result;
}
//遍历每一个元素
console.log(words.map(fuzzyPlural));
// ["feet", "geese", "meese", "kangareese"]
```

- 如何让一个string使用map方法获取字符串中每个字符所对应的ASCII码组成的数组

```
var map = Array.prototype.map
var a = map.call("Hello World", function(x) {
  return x.charCodeAt(0);
})
// a的值为[72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100]
```

- 如何遍历querySelectorAll得到动态对象集合

```
var elems = document.querySelectorAll('select option:checked');
var values = Array.prototype.map.call(elems, function(obj) {
  return obj.value;
});
```

- 反转字符串

```
var str = '12345';
Array.prototype.map.call(str, function(x) {
  return x;
}).reverse().join('');

// Output: '54321'
```

- parseInt等有多个参数的函数要注意

```
var a = ['1','2','3'];
var b = a.map(function(x){
  return parseInt(x);
 }); //[1,2,3]

 // 下面的语句返回什么呢:
  ["1", "2", "3"].map(parseInt);
  // 你可能觉的会是[1, 2, 3]
  // 但实际的结果是 [1, NaN, NaN]

  //parseInt有两个参数，第二个数是进制数
  //所以以上函数主要执行的是
  var a  = ['1','2','3'];
  var b = a.map(function(ele,index,array){
    return parseInt(ele,index);
    //第三个参数parseInt会忽视，但第二个参数不会
    // parseInt把传过来的索引值当成进制数来使用
    //而第二个参数假如经过 Number 函数转换后为 0 或 NaN，则将会忽略
    //parseInt(1,0);       1
    //parseInt(2,1);       NaN
    //parseInt(3,2);       NaN
  })

  //解决如下
  function returnInt(element){
    return parseInt(element,10);
  }
  ["1", "2", "3"].map(returnInt);
  // 返回[1,2,3]
```

- 兼容问题 IE8及以下不支持

```
// 实现 ECMA-262, Edition 5, 15.4.4.19
// 参考: http://es5.github.com/#x15.4.4.19
// 兼容代码
if (!Array.prototype.map) {
  Array.prototype.map = function(callback, thisArg) {

    var T, A, k;
    if (this == null) {
      throw new TypeError(" this is null or not defined");
    }
    // 1. 将O赋值为调用map方法的数组.
    var O = Object(this);
    // 2.将len赋值为数组O的长度.
    var len = O.length >>> 0;
    // 3.如果callback不是函数,则抛出TypeError异常.
    if (Object.prototype.toString.call(callback) != "[object
Function]") {
      throw new TypeError(callback + " is not a function");
    }
    // 4. 如果参数thisArg有值,则将T赋值为thisArg;否则T为undefined.
    if (thisArg) {
      T = thisArg;
    }
    // 5. 创建新数组A,长度为原数组O长度len
    A = new Array(len);
    // 6. 将k赋值为0
    k = 0;
    // 7. 当 k < len 时,执行循环.
    while(k < len) {
      var kValue, mappedValue;
      //遍历O,k为原数组索引
      if (k in O) {
        //kValue为索引k对应的值.
        kValue = O[ k ];
        // 执行callback,this指向T,参数有三个.分别是kValue:值,k:索引,O:原
数组.
        mappedValue = callback.call(T, kValue, k, O);
        // 返回值添加到新数组A中.
        A[ k ] = mappedValue;
      }
      // k自增1
      k++;
    }
    // 8. 返回新数组A
    return A;
  };
}
```

####

forEach

array.forEach(function(value,index,array){}，thisArg);

- 遍历数组，跑起来就停不下来，调用就会遍历整个数组，无法中断循环
- 回调函数的第一个参数表示数组中的每一个元素，第二个表示索引号,第三个表示正在操作的数组，可选

- 返回值undefined

```
//实例
function logArrayElements(element, index, array) {
    console.log("a[" + index + "] = " + element);
}

// 注意索引2被跳过了，因为在数组的这个位置没有项
[2, 5, ,9].forEach(logArrayElements);
// a[0] = 2
// a[1] = 5
// a[3] = 9

[2, 5,"" ,9].forEach(logArrayElements);
// a[0] = 2
// a[1] = 5
// a[2] =
// a[3] = 9
```

- 使用thisArg

```
function Counter() {
    this.sum = 0;
    this.count = 0;
}

Counter.prototype.add = function(array) {
    array.forEach(function(entry) {
        this.sum += entry;
        ++this.count;
    }, this);
    //console.log(this);
};

var obj = new Counter();
obj.add([1, 3, 5, 7]);
obj.count;
// 4 === (1+1+1+1)
obj.sum;
// 16 === (1+3+5+7)
```

- 兼容问题 IE8及以下不支持

```
// Production steps of ECMA-262, Edition 5, 15.4.4.18
// Reference: http://es5.github.io/#x15.4.4.18
if (!Array.prototype.forEach){
  Array.prototype.forEach = function(callback, thisArg) {

    var T, k;
    if (this == null) {
      throw new TypeError(' this is null or not defined');
    }
    // 1. Let O be the result of calling toObject() passing the
    // |this| value as the argument.
    var O = Object(this);
    // 2. Let lenValue be the result of calling the Get() internal
```

```
    // method of O with the argument "length".
    // 3. Let len be toUint32(lenValue).
    var len = O.length >>> 0;
    // 4. If isCallable(callback) is false, throw a TypeError
exception.
    // See: http://es5.github.com/#x9.11
    if (typeof callback !== "function") {
      throw new TypeError(callback + ' is not a function');
    }
    // 5. If thisArg was supplied, let T be thisArg; else let
    // T be undefined.
    if (arguments.length > 1) {
      T = thisArg;
    }
    // 6. Let k be 0
    k = 0;
    // 7. Repeat, while k < len
    while (k < len) {
      var kValue;
      // a. Let Pk be ToString(k).
      //    This is implicit for LHS operands of the in operator
      // b. Let kPresent be the result of calling the HasProperty
      //    internal method of O with argument Pk.
      //    This step can be combined with c
      // c. If kPresent is true, then
      if (k in O) {
        // i. Let kValue be the result of calling the Get internal
        // method of O with argument Pk.
        kValue = O[k];
        // ii. Call the Call internal method of callback with T as
        // the this value and argument list containing kValue, k,
and O.
        callback.call(T, kValue, k, O);
      }
      // d. Increase k by 1.
      k++;
    }
    // 8. return undefined
  };
}
```

####

every

- 根据当前回调函数的返回值决定是否进行下一次循环
  - 如果没有return true，则只是执行一次
  - 回调函数的返回值为true，继续循环
  - 返回值是false，停止循环
- 第一个参数是数组中的每一个元素 第二个参数表示索引号
- 兼容问题 IE8及以下不支持

```
//兼容问题
if (!Array.prototype.every){
  Array.prototype.every = function(fun /*, thisArg */){
    'use strict';
```

```
    if (this === void 0 || this === null)
      throw new TypeError();

    var t = Object(this);
    var len = t.length >>> 0;
    if (typeof fun !== 'function')
        throw new TypeError();

    var thisArg = arguments.length >= 2 ? arguments[1] : void 0;
    for (var i = 0; i < len; i++){
      if (i in t && !fun.call(thisArg, t[i], i, t))
        return false;
    }
    return true;
  };
}
```

####

some(返回布尔)

array.some(callback[,thisArg])

- callback 被调用时传入三个参数：元素的值，元素的索引，被遍历的数组
- *thisArg参数,将会把它传给被调用的callback，作为this 值。否则，在非严格模式下将会是全局对象，严格模式下是undefined*
- 数组中如果有一个满足条件，返回true，否则返回false

```
//callback
function isBigEnough(element, index, array){
  return (element >= 10);
}
var passed = [2, 5, 8, 1, 4].some(isBigEnough);
// passed is false
passed = [12, 5, 8, 1, 4].some(isBigEnough);
// passed is true
```

- some有兼容问题，IE8及以下不支持

```
//兼容代码
if (!Array.prototype.some){
  Array.prototype.some = function(fun /*, thisArg */){
    'use strict';
    if (this === void 0 || this === null)
       throw new TypeError();

    var t = Object(this);
    var len = t.length >>> 0;
    if (typeof fun !== 'function')
      throw new TypeError();

    var thisArg = arguments.length >= 2 ? arguments[1] : void 0;
    for (var i = 0; i < len; i++){
      if (i in t && fun.call(thisArg, t[i], i, t))
        return true;
```

```
    }
    return false;
  };
}
```

####

filter

var new_array = arr.filter(callback[, thisArg])

- 判断数组中的每一项是否都满足条件，所有满足条件的则返回新数组
- callback 用来测试数组的每个元素的函数。调用时使用参数 (element, index, array)返回true表示保留该元素（通过测试），false则不保留
- *thisArg 可选，执行callback时的用于this的值*
- 这些概念去看some

```
function isBigEnough(element) {
  return element >= 10;
}
var filtered = [12, 5, 8, 130, 44].filter(isBigEnough);
// filtered is [12, 130, 44]

//另一种写法
var arr = [12, 5, 8, 130, 44];
var filt = arr.filter(function(element){
  return element >= 10;
})//filt is [12, 130, 44]
```

- filter有兼容问题，IE8及以下不支持

```
//兼容代码
//假定 fn.call 等价于 Function.prototype.call 的初始值，且
Array.prototype.push 拥有它的初始值。
if (!Array.prototype.filter)
{
  Array.prototype.filter = function(fun /*, thisArg */)
  {
    "use strict";

    if (this === void 0 || this === null)
      throw new TypeError();

    var t = Object(this);
    var len = t.length >>> 0;
    if (typeof fun !== "function")
      throw new TypeError();

    var res = [];
    var thisArg = arguments.length >= 2 ? arguments[1] : void 0;
    for (var i = 0; i < len; i++)
    {
      if (i in t)
      {
        var val = t[i];
```

```
        // NOTE: Technically this should Object.defineProperty at
        //       the next index, as push can be affected by
        //       properties on Object.prototype and
Array.prototype.
        //       But that method's new, and collisions should be
        //       rare, so use the more-compatible alternative.
        if (fun.call(thisArg, val, i, t))
          res.push(val);
      }
    }

    return res;
  };
}
```

- **截取数组**

slice(索引)

array.slice(start,end)

- start
  - 正数，数组片段开始截取的下标
  - 负数，从数组尾部开始算起，-1是最后一个，-2是倒数第二个
  - 截取一个生成新数组，不影响原来数组，参数开始从0开始，从-1结束
- end
  - 结束的后一个元素的数组下标
  - 没有指定参数，则默认是到数组结束
  - 如果是负数，就是从末尾开始算起
- 截取一个生成新数组，不影响原来数组，包含从start到end(不包括该元素)指定的array元素

```
var a = [1,2,3,4,5];
a.slice(0,3);    // 返回 [1,2,3]
a.slice(3);      // 返回 [4,5]
a.slice(1,-1);   // 返回 [2,3,4]
a.slice(-3,-2);  // 返回 [3];
//IE 4存在的Bug: 返回[1,2,3]
```

splice(长度，可替换)

array.splice(start, deleteCount, value, ...)

- 参数有start，deleteCount，options(替换)
  - start开始插入和(或)删除的数组元素的下标
  - deleteCount 从start开始（包括start）要删除的元素个数。参数可选，如果没有指定，就默认到结尾的所有元素
  - value,... 要插入的数组值，从start所指的下标处开始插入
- 返回值是截取到的数组

```
//定义一个数组
```

```
var arr = [10,20,30,40,50,60,70,80,90];
//三个参数的情况  从索引为三的地方数三个，替换这三个数
var result = arr.splice(3,3,100,200,300);
//arr = [10,20,30,100,200,300,70,80,90];
//result = [40,50,60];
//两个参数的情况  从索引为三的地方截取三个
var result1 = arr.splice(3,3);
//arr = [10,20,30,70,80,90];
//result = [100,200,300];
//一个参数的情况，是从索引为三的地方一直到最后截取
var result2 = arr.splice(3);
//arr = [70,80,90]
//result2 = [10,20,30]
```

- **连接数组**

join(字符串)

var string = array.join()

- 如果没有参数，默认用逗号作为分割符
- 如果有参数，则参数是用于分隔数组元素的字符或字符串
- 返回字符串，通过把array每个元素转换成字符串，用参数连接起来
- 可以用String对象的split()方法进行相反的操作，把字符串根据参数分隔成数组

```
var a = new Array(1, 2, 3, "testing");
//a = [1,2,3,testing]
var s = a.join("+");  // s 是字符串"1+2+3+testing"
```

concat

var new_array = array1.concat('array2','array3')

- 参数至少是一个
- 返回一个新数组
- 如果操作的参数是一个数组，那么添加的是数组中的元素，而不是数组

```
var a = [1,2,3];
a.concat(4, 5)          //返回 [1,2,3,4,5]
a.concat([4,5]);        //返回 [1,2,3,4,5]
a.concat([4,5],[6,7])   //返回 [1,2,3,4,5,6,7]
a.concat(4, [5,[6,7]])  //返回 [1,2,3,4,5,[6,7]]
```

- **添加元素**

push

array.push(value,...)

- 要添加到array的末尾，可以是一个也可以是多个
- 返回值是添加后的数组的长度

- pop()方法和push()方法可以提供先进后出的栈的功能
- 在对象中添加元素

```
var obj = {
    length: 0,
    addElem: function addElem(elem){
        [].push.call(this, elem);
    }
};
obj.addElem({});
obj.addElem({});
console.log(obj.length);
// → 2
```

unshift

array.unshift(value, ...)

- 参数是要添加到头部的一个或多个值
- 返回数组的新长度

移除元素

pop

array.pop()

- 删除的是数组中的最后一个元素,数组长度-1
- 返回值是删除的元素
- 如果数组为空，则数组长度不变，返回undefined
- pop()方法和push()方法可以提供先进后出的栈的功能

shift

array.shift()

- 移除的是数组中的第一个元素，其余的向前移
- 返回的是移除元素的值
- 如果是空数组，则不进行任何操作，返回undefined

数组排序

sort

arr.sort(compareFunction)

- compareFunction 可选。是用来指定按什么顺序进行排序的函数，可选
- 返回排序后的数组
- 如果不传参数，将按照字母（字符编码）顺序最数组进行排序，所以要把数组中的元素转化为字符串以便进行比较
- 如过按照别的顺序进行排序，就要提供比较函数（参数a,b）
    - a < b 排序后a在b之前，就返回一个小于0的值

- a = b 返回0
- a > b 排序后a在b之后，返回一个大于0的值

```
// 按照数字顺序排序的排序函数
//a-b 表示升序排列
function  sortAscending(a, b) { return a - b; }
//b-a 表示降序排列
function  sortDescending(a,b) { return b - a; }

var a = new Array(33, 4, 1111, 222);
// 按照字母顺序的排序
a.sort();              // 结果为: 1111, 222, 33, 4
// 按照数字顺序的排序
a.sort(sortAscending);  //结果为: 4, 33, 222, 1111
a.sort(sortDescending); //结果为：1111，222，33，4
```

reverse

array.reverse( )

- 颠倒数组中元素的顺序，不创建新数组

```
var a = new Array(1, 2, 3);
// a = [1,2,3]    a[0] == 1, a[2] == 3;
a.reverse(  );
//Now  a = [3,2,1]  a[0] == 3, a[2] == 1;
```

查找数组

indexOf(返回索引)

arr.indexOf(searchElement)

arr.indexOf(searchElement[, fromIndex = 0])

- searchElement 要查找的元素
- fromIndex 开始查找的位置
  - fromIndex >= length，意味着不会在数组里查找，返回-1
  - 负值，-1从最后一个开始查找，-2从倒数第二个开始找
- 返回值如果找到了元素就返回元素在数组中的索引位置，若没有找到则返回-1

```
var array = [2, 5, 9];
array.indexOf(2);     // 0
array.indexOf(7);     // -1
array.indexOf(9, 2);  // 2
array.indexOf(2, -1); // -1
array.indexOf(2, -3); // 0
```

- 找出指定元素出现的所有位置

```
var indices = [];
var array = ['a', 'b', 'a', 'c', 'a', 'd'];
var element = 'a';
```

```
//判断元素在不在数组里面
var idx = array.indexOf(element);
//如果元素在数组里面，就循环
while (idx != -1) {
  //把索引推入新数组中
  indices.push(idx);
  //从找到元素的下一个索引开始继续查找
  idx = array.indexOf(element, idx + 1);
}
console.log(indices);
// [0, 2, 4]
```

- 判断一个元素是否在数组里，不在则更新数组

```
//定义一个函数
function update(vegs, veg) {
  //如果数组中不存在
    if (vegs.indexOf(veg) === -1) {
      //在数组中添加元素
        vegs.push(veg);
      console.log('New vegs is :' + vegs);
      //如果在数组中存在
    } else if (vegs.indexOf(veg) > -1) {
      //这个元素已经存在在数组中
        console.log(veg + ' already exists in the vegs.');
    }
}

var vegs = ['potato', 'tomato', 'chillies', 'green-pepper'];
 update(vegs, 'spinach');
  // New vegs is : potato,tomato,chillies,green-papper,spinach
 update(vegs, 'spinach');
  // spinach already exists in the vegs.
```

- 兼容问题 IE8及以下不兼容

```
  //兼容代码
  // Production steps of ECMA-262, Edition 5, 15.4.4.14
  // Reference: http://es5.github.io/#x15.4.4.14
  if (!Array.prototype.indexOf) {
    Array.prototype.indexOf = function(searchElement, fromIndex) {

      var k;
      // 1. Let O be the result of calling ToObject passing
      //    the this value as the argument.
      if (this == null) {
        throw new TypeError('"this" is null or not defined');
      }
      var O = Object(this);
      // 2. Let lenValue be the result of calling the Get
      //    internal method of O with the argument "length".
      // 3. Let len be ToUint32(lenValue).
      var len = O.length >>> 0;
      // 4. If len is 0, return -1.
      if (len === 0) {
```

```
        return -1;
      }
      // 5. If argument fromIndex was passed let n be
      //    ToInteger(fromIndex); else let n be 0.
      var n = +fromIndex || 0;
      if (Math.abs(n) === Infinity) {
        n = 0;
      }
      // 6. If n >= len, return -1.
      if (n >= len) {
        return -1;
      }
      // 7. If n >= 0, then Let k be n.
      // 8. Else, n<0, Let k be len - abs(n).
      //    If k is less than 0, then let k be 0.
      k = Math.max(n >= 0 ? n : len - Math.abs(n), 0);
      // 9. Repeat, while k < len
      while (k < len) {
        // a. Let Pk be ToString(k).
        //   This is implicit for LHS operands of the in operator
        // b. Let kPresent be the result of calling the
        //    HasProperty internal method of O with argument Pk.
        //   This step can be combined with c
        // c. If kPresent is true, then
        //    i.  Let elementK be the result of calling the Get
        //        internal method of O with the argument
ToString(k).
        //   ii.  Let same be the result of applying the
        //        Strict Equality Comparison Algorithm to
        //        searchElement and elementK.
        //  iii.  If same is true, return k.
        if (k in O && O[k] === searchElement) {
          return k;
        }
        k++;
      }
      return -1;
    };
  }
```

####