[笔记][LIKE-JS][3-JavaScript基础-数组和函数]

JavaScript

---

041. JS基础-数组和函数-for循环练习(掌握)

```
<script>
    // 1 - 100 偶数和、奇数和
    var evenSum = oddSum = 0;
    for (i = 1; i <= 100; i++) {
        if (i % 2 === 0) {
            evenSum += i;
        } else {
```

```
                oddSum += i;
            }
        }
        document.writeln('偶数和为：', evenSum);
        document.write('<br>');
        document.writeln('奇数和为：', oddSum);
    </script>
```

---

042. JS基础-数组-基本介绍(掌握)

创建数组的方式：

- `Array` 构造函数
- 数组字面量

```
        // 1. 使用 Array 构造函数
        var arr1 = new Array();  // 创建了一个空数组，可以放无限个元素
        var arr2 = new Array(20); // 创建了一个长度为 20 的数组
        var arr3 = new Array(6, 7, 9); // 创建带数据的数组

        console.log(arr1);
        console.log(arr2);
        console.log(arr3);

        // 2. 使用数组字面量
        var arr4 = [];
        var arr5 = [30];  // 字面量这样放，30 不是数组长度，是里面的数据

        console.log(arr4);
        console.log(arr5);
```

---

043. JS基础-数组-基本使用(掌握)

常用操作

- `length` 属性
  注意：如果数组长度改小了，从后面删除元素。
- 索引取值

```
<script>
    var arr1 = [30];
    arr1.length = 100;
    console.log(arr1);

    var arr2 = ['张三', 18, '男', '篮球'];
    console.log(arr2.length);  // 4

    console.log(arr2[0]); // 张三
    console.log(arr2[arr2.length]); // undefined
</script>
```

- 遍历数组

```
<script>
    var numbers = [10, 1, 19, 22, 73];
    for (var i = 0; i < numbers.length; i++) {
        if (numbers[i] % 2 === 0) {
            console.log(numbers[i]);
        }
    }
</script>
```

---

044. JS基础-数组-基本使用-下(掌握)

去除数组中的零，放到新数组中。

```
<script>
    var arr = ['张三', 0, 12, 0, '男'];
```

```
    var newArr = [];

    for (var i = 0; i < arr.length; i++) {
        if (arr[i] !== 0) {
            newArr[newArr.length] = arr[i];
        }
    }

    console.log(newArr);

</script>
```

045. JS基础-数组-常用方法(掌握)

- `join(seperator)`
  连接成字符串，默认以逗号分隔
  不会改变原数组
- `push` 和 `pop`
  数组是一个栈结构，是有底的，后进先出，与之对应的是先进先出的队列结构
  `push()` 接收任意数量的参数，把它们逐个添加到数组末尾，并返回修改后数组的长度；
  `pop()` 数组末尾移除最后一项，减少数组的 `length` 值，然后返回移除的项。

> `Python` 的列表有 `append` 追加元素，`extend` 追加可迭代对象，`insert` 在某位置插入，`pop` 弹出某位置元素。除了 `pop` 其它都不会返回值。

046. JS基础-数组-常用方法2(掌握)

- `shift()` 删除原数组第一项，并返回元素的值，如果数组为空，返回 `undefined`

- `unshift()` 将参数添加到原数组开头，并返回数组的长度。
  可以接收多个参数
- `reverse()` 反转数组项。改变原数组，并返回反转后的数据。

> `reverse()` 跟 `Python` 的 `list.reverse()` 一样。

- `concat()` 将参数添加到原数组。
  先 `copy` 一个当前数组，然后将接收到的参数添加到这个 `copy` 数组的末尾，最后返回新构建的数组。
  在没有给 `concat()` 方法传递参数的情况下，它只是复制当前数组并返回。

```
<script>
    var arr = [1, 3, 5, 7, 9];
    var copyArr = arr.concat(11, '王小二', ['张三', '里斯']);
    console.log(arr);
    console.log(copyArr);
</script>
```

`push` 方法与 `concat` 方法的区别：

- `push` 方法会改变原数组，`concat` 方法不会改变原数组
- `push` 方法返回新数组的长度，`concat` 方法返回一个新数组
- `push` 方法按原样追加元素，`concat` 方法会拆一层数组

- `indexOf`：接收两个参数，要查找的项和查找起点的索引。默认从开头向末尾查找，查找第一个匹配的项。
- `lastIndexOf`：参数同上，默认从末尾开始向开头查找，查找第一个匹配的项。
- 这两个方法都返回元素的索引，没找到返回 `-1`。
- 在比较的时候，会使用全等操作符。

---

047. JS基础-函数-基本使用(掌握)

函数：完成特定功能的一段代码，可以重复执行。
作用：简化代码、封装可复用代码

1. 函数声明方式

```
function 函数名(参数1, 参数2, ...) {
    函数体;
}
```

2. 函数表达式声明方式

```
var add = function () {
  console.log(1 + 3);
};

add();
```

3. 使用 `Function` 构造函数

```
var add2 = new Function('console.log(1 + 3);');

add2();
```

> 个人理解：函数表达式就是搞一个匿名函数，然后给它绑定一个名称而已。

---

048. JS基础-函数-参数和arguments对象(掌握)

函数的参数：

- 形参
- 实参

实参个数可以多于形参个数，不会报错。
如果缺少参数，则会变成 `undefined`。

求出多个数的和

- 使用数组
- 使用 `arguments` 对象
  它不是数组，与数组类似，拥有 `length` 属性，其它所有属性和方法都不具备。它里面存放着传入函数中的所有参数。它还有一个 `callee` 属性，该属性是一个指针，指向拥有这个 `arguments` 对象的函数。

```
<script>
    function sum() {
        // arguments 对象
        // console.log(arguments);

        var value = 0;
        for (var i = 0; i < arguments.length; i++) {
            value += arguments[i];
        }
        console.log(value); // 60
    }

    sum(10, 20, 30);
</script>
```

---

049. JS基础-函数-return返回值(掌握)

函数有 `length` 属性，存放着形参个数。
`函数名.length`

`return` 后面的语句不会被执行，返回执行结果

- 如果函数没有返回值，返回值默认为 `undefined`

- 如果直接 `return;`，返回值默认为 `undefined`

`Math.min(x, y)` 取最小值

---

050. JS基础-函数-相关补充(掌握)

`JavaScript` 解析器会首先把当前作用域的函数声明提前到整个作用域的最前面。

> 所以可以先使用函数，再定义函数，不会报错。
> 这跟 `Python` 不一样，`Python` 里面必须先定义再使用。

但是通过函数表达式声明的函数，不会这样做。
所以不能先使用通过表达式声明的函数，再定义它。

匿名函数：没有命名的函数
作用：

- 绑定事件

```
document.onclick = function () {
    alert('点我干嘛？');
};
```

- 定时器

```
setInterval(function () {
    console.log(111);
}, 1000);
```

- 自调用函数

```
(function () {
    alert('我会被执行！');
```

```
})();
```

格式

```
function (参数列表) {
    函数体;
}
```

---

051. JS基础-函数-回调函数(掌握)

回调函数：通过函数调用的函数。
如果你把函数的指针（地址）作为参数传递给另外一个函数，当这个指针被用来调用其所指向的函数时，我们就说这是回调函数。
一般用于递归。

```
<script>
    // 模拟计算器
    // 总控方法
    function fn(num1, num2, func) {
        return func(num1, num2);
    }

    // 加减乘除函数
    function add(a, b) {
        return a + b;
    }

    function sub(a, b) {
        return a - b;
    }

    function mul(a, b) {
        return a * b;
    }

    function div(a, b) {
```

```
            return a / b;
        }

        console.log(fn(10, 20, add));
        console.log(fn(10, 20, sub));
        console.log(fn(10, 20, mul));
        console.log(fn(10, 20, div));
        /*
        30
        -10
        200
        0.5
         */

    </script>
```

递归示例

```
    <script>
        function fib(n) {
            if (n === 1 || n === 2) return 1;
            return fib(n - 1) + fib(n - 2);
        }
    </script>
```

---

052. JS基础-变量作用域-上(掌握)

没有块级作用域这个概念，花括号内定义的变量，在花括号外面也能够访问。

全局变量：定义在 `script` 或者不属于某个函数的变量就是全局变量。
局部变量：在函数内部定义的变量就是局部变量。

注意

- 函数内部可以访问到函数所属的外部作用域的变量（作用域链）
- 不使用 `var` 声明的变量是全局变量，不推荐使用

> 这里老师可能讲的有点问题，我搜了一下，不带 `var` 不是这样的。
> 参见：https://www.cnblogs.com/juejiangWalter/p/5725220.html
>
> 事实上是对属性赋值操作。首先，它会尝试在当前作用域链中解析 `num`； 如果在任何当前作用域链中找到 `num`，则会执行对 `num` 属性赋值； 如果没有找到 `num`，它才会在全局对象（即当前作用域链的最顶层对象，如 `window` 对象）中创造 `num` 属性并赋值。
> 注意！它并不是声明了一个全局变量，而是创建了一个全局对象的属性。

```
<script>
    function test() {
        b = 777;
    }
    // 注意：必须先执行一遍，否则报错！
    test();
    console.log(b);
</script>
```

- 变量退出作用域之后会销毁，全局变量关闭网页或浏览器才会销毁

---

053. JS基础-变量作用域-下(掌握)

`JS` 与其它语言的不一样：

- 没有块级作用域，大括号内部的变量依然是全局变量
- `for` 循环中的循环变量，在 `for` 语句外部仍然可以被访问到

> 注意： `JS` 跟 `Python` 在这一点上也是一样的。
> `Python` 的缩进不会形成块级作用域。
> `for` 的循环变量在语句外面也能被访问。

面试题

```
var num = 10;
fun();
function fun() {
    console.log(num);
    var num = 20;
}
// 答案：undefined
```

这里会将声明提前。
等价于下面的代码：

```
var num = 10;
fun();
function fun() {
    var num;  // 声明提前
    console.log(num);
    num = 20;
}
// 答案：undefined
```

> 其实跟 `Python` 一样，在函数体内的所有变量赋值，都使该变量变成局部变量，但是不会对它进行赋值。

```
var a = 18;
f1();
function f1() {
    var b = 9;
    console.log(a);
    console.log(b);
    var a = '123';
}
// undefined
// 9
```

最后一题：

```
f1();
console.log(c);
console.log(b);
console.log(a);
function f1() {
    var a = b = c = 9;  // a 局部，b、c 全局
    console.log(a);
    console.log(b);
    console.log(c);
}

// 9
// 9
// 9
// 9
// 9
// 报错
```

完成于 `20190225`