20230915

一、函数

定义：函数/方法/function/medthod 是包含应用程序的代码块/程序块，该程序块可以被其他程序通过函数名调用。

目的：提高代码的公用性，以及可维护性，代码复用性。

1.基本语法

1. 函数的声明

```
function student(形参){
    方法体
}
```

2. 函数的调用

   函数名(实参)

2.带参函数

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <!-- 调用 -->
    <!-- 实体参数（实参）：调用函数时，具体的值，多个实参 逗号分隔 -->
    <button onclick="myClick('jack','builder')">Click me</button>
</head>
<body>
    <script>

        //函数的命名必须小写
        // 形式参数（形参）：没有具体的值，只有变量名（自定义），多个变量用 逗号分隔
        // 在方法体中，可以直接使用形参

        function myClick(name,job) {
            alert('name:'+ name+'job:'+job);
        }

    </script>
</body>
</html>
```

3.函数返回值

关键字：return

格式：return 值

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>
    <script>
        //调用函数
        // 返回值：调用函数的代码 本身带值（为返回值）
        var resule = func01(2,3)
        console.log(resule)
        function func01(a,b) {
            var sum = a+b;
            return sum ;
        }

        //如果一个函数无返回值，那么该函数的返回值为undifiend
        function func02(a,b) {
            var sum = a+b;
        }
        var resule2 = func02(5,8)
        console.log(resule2)
    </script>
</body>
</html>
```

4.return语句

return+值：函数返回值

return；退出函数

```
<!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>

        function func01() {
            console.log('hello');
            return;//退出函数
            console.log('world');//不会被执行的代码
        }
        func01();
```

```
        function func02() {
            console.log('hello');
            return;
            console.log('world');
        }
        console.log(func02());
        //执行return后不管后面有没有代码，都直接退出函数
    </script>
</body>
</html>
```

5.函数调用案例

```
<!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>
        function func01() {
            func02();
        }
        function func02() {
            func03();
        }
        function func03() {
            console.log('run');
        }

        func01();
    </script>
</body>
</html>
```

6.调用案例2

函数中可以利用调用函数获取函数的返回值（数据在函数间的传导）

```
<!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>
        //函数返回值的传递
        function func01() {
            return func02();+1
```

```
        }
        function func02() {
            return func03();+2
        }
        function func03() {
            return 3;
        }

        var resule = func01();
        console.log(resule);
    </script>
</body>
</html>
```

7.单击触发执行函数案例

```
<!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>
    <button onclick="myClick()">Touch me</button>
    <input type="text" id="id_ipt_01">
    <script>
        function myClick(){


            var obj_ipt = document.getElementById('id_ipt_01');
            var text_ipt = obj_ipt.value;
            alert(text_ipt);
        }
    </script>
</body>
</html>
```

8.求最大值的函数

```
<!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>
    <input type="text" id="num1">
    <input type="text" id="num2">
    <input type="text" id="num3">
    <input type="text" id="num4" placeholder="最大值结果" readonly/>
```

```
    <br>
    <button onclick="myClick()">Click me</button>
</body>
<script>

   function myClick(){
        func01();
   }
    function func01(){
        var a = parseFloat(document.getElementById('num1').value);
        var b = parseFloat(document.getElementById('num2').value);
        var c = parseFloat(document.getElementById('num3').value);

        var result = document.getElementById('num4')

        var max=maxInto(a,b);
        max = maxInto(max,c);

        result.value=max
    }
    function maxInto(a,b){
        return a>b?a:b
    }
</script>
</html>
```

9.函数的递归

```
<!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>
        // var count = 0;
        // function init() {
        //     console.log(++count)
        //     init();
        // }
        // init();

        // 案例：求阶乘的函数
        function factorial(n) {
            if (n === 1) {
                return 1; // 阶乘的基本情况，0和1的阶乘都是1
            } else {
                return n * factorial(n - 1); // 递归调用
            }
        }

                // 使用示例
```

```
                console.log(factorial(170)); // 输出 120，因为5! = 5 * 4 * 3 * 2 *
1 = 120

        </script>
</body>
</html>
```

10函数表达式

另一种声明函数的方式

以给变量赋值的方式声明函数

var 函数名 = function（）{

方法体;

}

```
<!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>

        var func01 = function(){
            console.log('func01');
        }

        func01();

        var func02 = function(a,b){

            return a>b?a:b;
        }

        console.log(func02(3,5));
    </script>
</body>
</html>
```

*11.Function类

又一种声明函数的方式

格式：var 函数名（对象名）= new Function（形参，方法体）

```
<!DOCTYPE html>
<html lang="en">
<head>
```

```
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script src="../../../js/vue.js"></script>
</head>
<body>
    <script>
        var func01 = new Function('a','b','var c=a+b;console.log(c);');

        func01(1,2)
    </script>
</body>
</html>
```

12.函数赋值

把函数看作对象，看作值，就可以赋给其他变量

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script src="../../../js/vue.js"></script>
</head>
<body>
    <script>
        function func01(a,b) {
            console.log(a>b?a:b)
        }
        // 函数赋值给变量
        var func02 = func01;

        func02(1,2)
    </script>
</body>
</html>
```

13.函数的声明提升

当使用非传统声明函数的方式时，函数的声明要求更严谨，必须先声明再调用

```
<!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>
```

```
        func01()
        func02()

        function func01() {
            console.log('func01');
        }

        var func02 = function () {
            console.log('func02');
        }
    </script>
</body>
</html>
```

*14.匿名函数的自执行

```
<!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>
        // 匿名函数
        // 自动执行
        ;(function(a,b){
            console.log(a+b);
        })(5,6)

    </script>

</body>
</html>
```

*15.匿名函数防止全局污染

将业务按成员，按需求，按组别模块化

某个匿名函数内 出现问题（代码内部 运行结果）不影响整个文件的执行

二、流程控制

1.if语句

1.单分支

格式

if（表达式）{

语句体

表达式为真语句体执行，反之语句体不执行

}

特点：

单分支语句，根据条件判断真假，可能执行，也可能不执行

允许省略{}，if关键字后面的一行代码作为方法体

2.双分支

格式：

if（condition）{

statement 1；

}else{

sta2；

}

condition的值为ture执行sta1，反之执行sta2

案例：

```
<body>
<button onclick="func01()">按钮</button>
<script>
function func01() {
var score = prompt("请输入一个分数");
// 判断是否取消
if (score == null) {
alert("操作已取消");
return;
}
//判断是否为空字符串
if (score == "") {
alert("分数不能为空");
return;
}
// 判断是否为数字
if (isNaN(score)) {
alert("必须输入一个数字");
return;
}
// score数据类型转换为number类型
score = parseInt(score);
// 判断分数是否及格
// 1.判断分数是否在0~100之间
if (score < 0 || score > 100) {
alert("必须在0~100之间");
```

```
} else {
// 2.判断分数是否及格
if (score >= 60) {
alert("pass");
} else {
alert("fail");
}
}
}
</script>
</body>
```

3.多分枝

格式

if（condition 1）{

sta1

}else if(condition2){

sta2

}else if(condition3){

sta 3

}...

else{

sta n

}

代码从上至下运行，如果condition为ture，则运行其对应的sta，执行一次后离开多分枝

```
var score = parseInt(prompt("请输入一个分数"));
// 判断成绩的分数段 >=90 A
if(score>=90){
console.log("A");
}else if(score>=80){ // 80~89分数段
console.log("B");
}else if(score>=70){
console.log("C");
}else if(score>=60){
console.log("D");
}else{
console.log("F");
}
```

2.switch语句

1.标准用法

```
格式:
switch(){
    case 1:
        sta1;
        break;
    case 2:
        sta2;
        break;
    case 3:
        sta3;
        break;
}
```

2.多个case执行相同语句

```
var month = parseInt(prompt("请输入一个月份"));
switch (month) {
case 3:
case 4:
case 5:
alert("春天");
break;
case 6:
case 7:
case 8:
alert("夏天");
break;
case 9:
case 10:
case 11:
alert("秋天");
break;
case 12:
case 1:
case 2:
alert("冬天");
break;
default:
alert("请输入标准月份");
}
```

3.循环

1.for循环

格式：

for（初始化；循环条件；迭代语句）{

　执行代码

}

2.while循环

格式： 初始化

while(循环条件){ 循环体 迭代语句 }

```
// 死循环
// while(true){
// console.log("test ok");
// }
var i=1;
while(i<=5){
console.log("test ok");
i++;
}
```

注意： 没有迭代语句 可能死循环 如果第一次循环条件不满足，则一次也不执行

```
var i=10;
while(i<=5){
console.log("test ok");
i++;
}
```

3.do-while循环

格式： 初始化 do{ 循环体 迭代语句 }while(循环条件)

```
var i=10;
do{
console.log("test ok");
i++;
}while(i<=5)
```

特点：先执行一次，再判断条件（至少执行一次循环体）

4.break关键字

```
// 执行break,跳出循环(退出整个循环)
for(var i=1;i<=5;i++){
if(i==3){
break;
}
console.log(i);
}
```

5.countinue关键字

```
// continue:跳出本次循环 继续执行下次循环
for(var i=1;i<=5;i++){
if(i==3){
continue;
}
console.log(i);
}
```

作业

1.

```
<!DOCTYPE html>
<html>
<head>
    <title>奇数偶数判断</title>
</head>
<body>
    <script>
        // 获取用户输入的数字
        var userInput = prompt("请输入一个数字：");

        // 将用户输入的字符串转换为数字
        var number = parseFloat(userInput);

        // 判断数字是奇数还是偶数
        if (isNaN(number)) {
            alert("请输入有效的数字！");
        } else {
            if (number % 2 === 0) {
                alert(number + " 是偶数。");
            } else {
                alert(number + " 是奇数。");
            }

            // 控制台输出
            console.log(number % 2 === 0 ? number + " 是偶数。" : number + " 是奇
数。");
        }
    </script>
</body>
</html>
```

2.

```
<!DOCTYPE html>
<html>
<head>
    <title>整除判断</title>
</head>
<body>
    <script>
```

```
        // 获取用户输入的数字 a 和 b
        var inputA = prompt("请输入数字 a：");
        var inputB = prompt("请输入数字 b：");

        // 将用户输入的字符串转换为数字
        var a = parseFloat(inputA);
        var b = parseFloat(inputB);

        // 判断是否能被整除
        if (isNaN(a) || isNaN(b)) {
            alert("请输入有效的数字！");
        } else {
            if (b === 0) {
                alert("除数不能为零！");
            } else if (a % b === 0) {
                alert(a + " 能够被 " + b + " 整除。");
            } else {
                alert(a + " 不能被 " + b + " 整除。");
            }
        }
    </script>
</body>
</html>
```

3.

```
<!DOCTYPE html>
<html>
<head>
    <title>三角形类型判断</title>
</head>
<body>
    <script>
        // 获取用户输入的三个边长
        var sideA = parseFloat(prompt("请输入第一条边的长度："));
        var sideB = parseFloat(prompt("请输入第二条边的长度："));
        var sideC = parseFloat(prompt("请输入第三条边的长度："));

        // 判断是否构成三角形
        if (isNaN(sideA) || isNaN(sideB) || isNaN(sideC) || sideA <= 0 || sideB
<= 0 || sideC <= 0) {
            alert("请输入有效的正数作为边长！");
        } else if (sideA + sideB <= sideC || sideA + sideC <= sideB || sideB +
sideC <= sideA) {
            alert("这些边长无法构成三角形！");
        } else {
            // 判断三角形类型
            if (sideA === sideB && sideB === sideC) {
                alert("这是一个等边三角形。");
            } else if (sideA === sideB || sideA === sideC || sideB === sideC) {
                alert("这是一个等腰三角形。");
            } else {
                alert("这是一个普通三角形。");
            }
        }
    </script>
```

```
</body>
</html>
```

4.

```
<!DOCTYPE html>
<html>
<head>
    <title>闰年判断</title>
</head>
<body>
    <script>
        // 获取用户输入的年份
        var year = parseInt(prompt("请输入一个年份："));

        // 判断是否为闰年
        if (isNaN(year)) {
            alert("请输入有效的年份！");
        } else if ((year % 4 === 0 && year % 100 !== 0) || year % 400 === 0) {
            alert(year + " 年是闰年。");
        } else {
            alert(year + " 年不是闰年。");
        }
    </script>
</body>
</html>
```

5.

```
<!DOCTYPE html>
<html>
<head>
    <title>计算1到100的和</title>
</head>
<body>
    <script>
        // 初始化总和
        var sum = 0;

        // 计算1到100的和
        for (var i = 1; i <= 100; i++) {
            sum += i;
        }

        // 输出结果
        console.log("1 到 100 的和为：" + sum);
        alert("1 到 100 的和为：" + sum);
    </script>
</body>
</html>
```

6.

```
<!DOCTYPE html>
<html>
<head>
```

```
    <title>计算能被3整除的数字之和</title>
</head>
<body>
    <script>
        // 初始化总和
        var sum = 0;

        // 计算1到100中能被3整除的数字之和
        for (var i = 1; i <= 100; i++) {
            if (i % 3 === 0) {
                sum += i;
            }
        }

        // 输出结果
        console.log("1 到 100 中能被 3 整除的数字之和为：" + sum);
        alert("1 到 100 中能被 3 整除的数字之和为：" + sum);
    </script>
</body>
</html>
```

7.

```
<!DOCTYPE html>
<html>
<head>
    <title>计算能被3整除和5整除的数字之和</title>
</head>
<body>
    <script>
        // 初始化总和
        var sum = 0;

        // 计算1到100中能被3整除和5整除的数字之和
        for (var i = 1; i <= 100; i++) {
            if (i % 3 === 0 && i % 5 === 0) {
                sum += i;
            }
        }

        // 输出结果
        console.log("1 到 100 中能被 3 整除和 5 整除的数字之和为：" + sum);
        alert("1 到 100 中能被 3 整除和 5 整除的数字之和为：" + sum);
    </script>
</body>
</html>
```

8.

```
 <!DOCTYPE html>
<html>
<head>
    <title>九九乘法表</title>
</head>
<body>
```

```
    <script>
        // 初始化乘法表字符串
        var multiplicationTable = "";

        // 嵌套循环生成九九乘法表
        for (var i = 1; i <= 9; i++) {
            for (var j = 1; j <= 9; j++) {
                multiplicationTable += i + " × " + j + " = " + (i * j) + "\t";
            }
            multiplicationTable += "\n"; // 换行
        }

        // 输出乘法表
        console.log("九九乘法表：");
        console.log(multiplicationTable);

        // 在页面上显示乘法表
        document.write("<pre>" + multiplicationTable + "</pre>");
    </script>
</body>
</html>
```

9.

```
<!DOCTYPE html>
<html>
<head>
    <title>生成不重复的三位数</title>
</head>
<body>
    <script>
        // 初始化计数器
        var count = 0;

        // 嵌套循环生成不重复的三位数
        for (var i = 1; i <= 4; i++) {
            for (var j = 1; j <= 4; j++) {
                for (var k = 1; k <= 4; k++) {
                    // 确保数字不重复
                    if (i !== j && i !== k && j !== k) {
                        var number = i * 100 + j * 10 + k;
                        console.log(number);
                        count++;
                    }
                }
            }
        }

        // 输出个数
        console.log("不重复的三位数个数：" + count);
    </script>
</body>
</html>
```

10.

```html
<!DOCTYPE html>
<html>
<head>
    <title>兔子繁殖问题</title>
</head>
<body>
    <script>
        function calculateRabbitTotal(n) {
            if (n <= 0) {
                return 0;
            }

            if (n === 1 || n === 2) {
                return 1;
            }

            var fib1 = 1; // 1月的兔子数量
            var fib2 = 1; // 2月的兔子数量
            var total = 0;

            for (var i = 3; i <= n; i++) {
                total = fib1 + fib2; // 当前月的兔子数量
                fib1 = fib2; // 更新1月的兔子数量
                fib2 = total; // 更新2月的兔子数量
            }

            return total;
        }

        // 输入月份 n
        var n = parseInt(prompt("请输入月份 n："));

        if (isNaN(n) || n < 1) {
            alert("请输入有效的正整数月份。");
        } else {
            var rabbitTotal = calculateRabbitTotal(n);
            alert("第 " + n + " 个月的兔子总数为：" + rabbitTotal);
        }
    </script>
</body>
</html>
```