

什么是函数

什么是函数

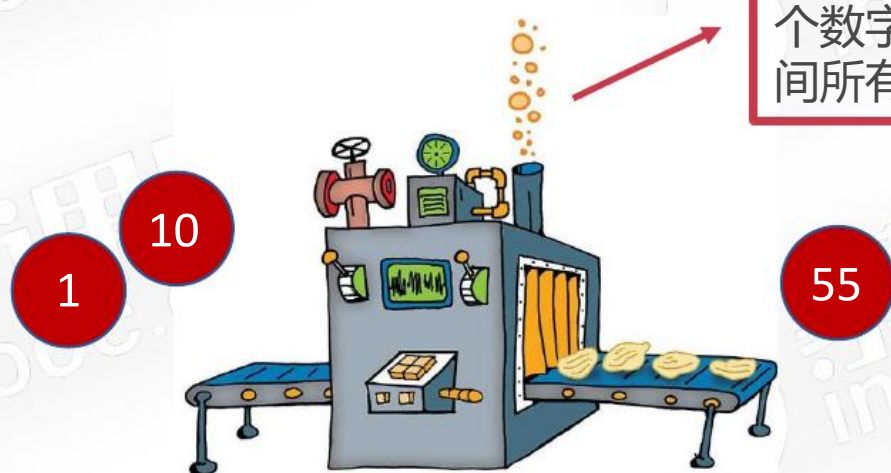
- ◆ 假如我们要分别计算1到10、5到12、14到35的整数和

```
for (var i = 1, sum = 0; i <= 10; i++) {  
    sum += i;  
}  
console.log(sum);
```

```
for (var i = 5, sum = 0; i <= 12; i++) {  
    sum += i;  
}  
console.log(sum);
```

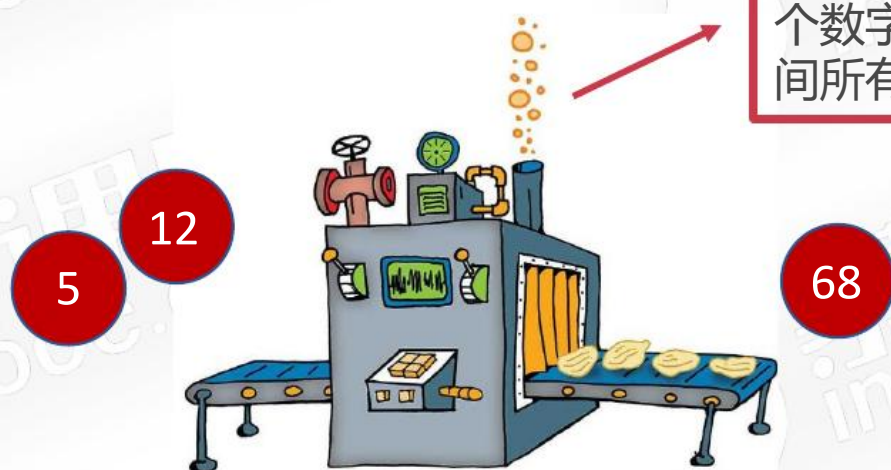
什么是函数

一个神奇的机器，送入机器两个数字，它将产生两个数字之间所有整数的和。



什么是函数

一个神奇的机器，送入机器两个数字，它将产生两个数字之间所有整数的和。



什么是函数



什么是函数

- ◆ 函数就是语句的封装，可以让这些代码方便地被复用
- ◆ 函数具有“一次定义，多次调用”的优点
- ◆ 使用函数，可以简化问题，让代码更具有可读性

函数的定义和调用

函数的定义

- ◆ 和变量类似，函数必须先定义然后才能使用
- ◆ 使用function关键字定义函数，function是“功能”的意思

函数的定义

function表示定义函数

函数名，函数名必须符合
JS标识符命名规则

圆括号中是形参列表，即使没有形参，也必须书写圆括号

```
function fun() {  
    // 函数体语句  
}
```

大括号中就是函数体语句

函数表达式

匿名函数

```
var fun = function () {  
    // 函数体语句  
}
```

函数的调用

- ◆ 执行函数体中的所有语句，就称为“调用函数”
- ◆ 调用函数非常简单，只需在函数名字后书写圆括号对即可

`fun()`

`// 调用函数`

圆括号中是实参列表，如果没有实参，也要书写圆括号

语句执行顺序

函数不调用内部语句
就不会执行

```
function fun() {  
  console.log('A');  
  console.log('B');  
  console.log('C');  
}
```

调用权移交给函数

函数体内所有语句执行完毕，
将语句执行权交还给主程序

```
console.log(1);  
console.log(2);  
console.log(3);  
fun();  
console.log(4);  
console.log(5);  
console.log(6);
```

函数声明的提升

- ◆ 和变量声明提升类似，函数声明也可以被提升

在预解析阶段会被提升

```
fun();  
  
function fun() {  
    alert("函数被执行");  
}
```

函数表达式不能提升

- ◆ 如果函数是用函数表达式的写法定义的，则没有提升特性

```
fun(); // 引发错误  
  
var fun = function () {  
    alert("函数被执行");  
}
```

函数优先提升

函数优先提升

变量声明提升，无法覆盖提升的函数

```
fun();           // 弹出B
```

```
var fun = function () {  
    alert('A');  
}
```

```
function fun() {  
    alert('B');  
}
```

```
fun();           // 弹出A
```

函数的参数和返回值

函数的参数

- ◆ 参数是函数内的一些**待定值**，在调用函数时，必须传入这些参数的**具体值**
- ◆ 函数的**参数可多可少**，函数可以没有参数，也可以有多个参数，多个参数之间需要用逗号隔开

函数的参数

```
function add(a, b) {  
  var sum = a + b;  
  console.log('两个数字的和是' + sum);  
}
```

圆括号中定义“形式参数”

```
add(3, 5);
```

调用函数时传入“实际参数”

“形实结合”

```
function add(a, b) {  
  var sum = a + b;  
  console.log('两个数字的和是' + sum);  
}  
add(3, 5);
```

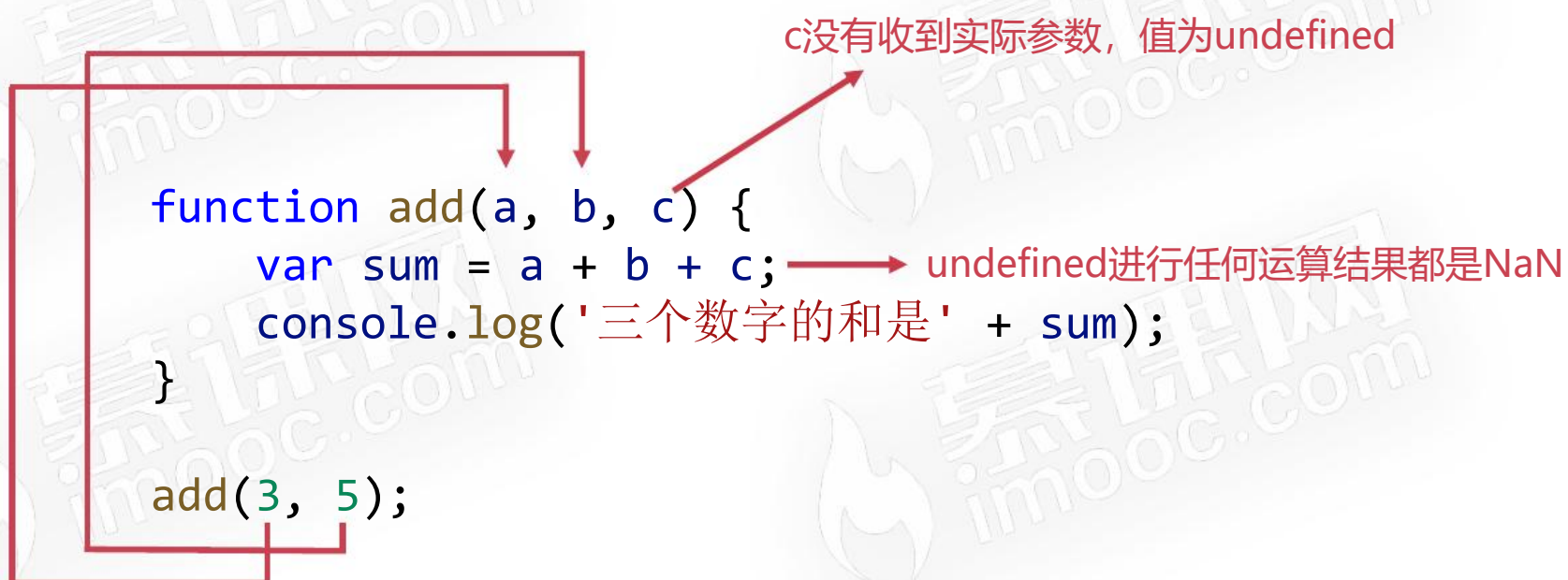
The diagram illustrates the 'call by value' mechanism. Red arrows show the flow of data: one arrow points from the parameter 'a' in the function definition to the value '3' in the function call, and another arrow points from the parameter 'b' to the value '5'. A third arrow points from the function call back to the function definition, indicating the execution flow.

形参和实参个数不同的情况

```
function add(a, b) {  
  var sum = a + b;  
  console.log('两个数字的和是' + sum);  
}  
add(3, 5, 8);
```

The diagram shows a function call with three arguments: `add(3, 5, 8);`. Red arrows connect the first two arguments, '3' and '5', to the parameters 'a' and 'b' in the function definition. A third arrow points from the third argument, '8', to the text '没有形参接收它' (No formal parameter receives it), illustrating that the extra argument is ignored because there are no more parameters to receive it.

形参和实参个数不同的情况



本节开头遇见的案例

- ◆ 假如我们要分别计算1到10、5到12、14到35的整数和

```
for (var i = 1, sum = 0; i <= 10; i++) {  
    sum += i;  
}  
console.log(sum);  
  
for (var i = 5, sum = 0; i <= 12; i++) {  
    sum += i;  
}  
console.log(sum);
```

arguments

- ◆ 函数内arguments表示它接收到的实参列表，它是一个类数组对象
- ◆ 类数组对象：所有属性均为从0开始的自然数序列，并且有length属性，和数组类似可以用方括号书写下标访问对象的某个属性值，但是不能调用数组的方法

函数的返回值

- ◆ 函数体内可以使用return关键字表示“函数的返回值”

```
function sum(a, b) {  
    return a + b;  
}
```

函数的返回值

```
var result = sum(3, 5);
```

函数的返回值可以被变量接收

函数的返回值

- ◆ 调用一个有返回值的函数，可以被当做一个普通值，从而可以出现在任何可以书写值的地方

```
function sum(a, b) {  
    return a + b;  
}  
var result = sum(3, 4) * sum(2, 6);
```

函数的返回值

- ◆ 调用一个有返回值的函数，可以被当做一个普通值，从而可以出现在任何可以书写值的地方

```
function sum(a, b) {  
    return a + b;  
}  
var result = sum(3, sum(4, 5));
```

遇见return即退出函数

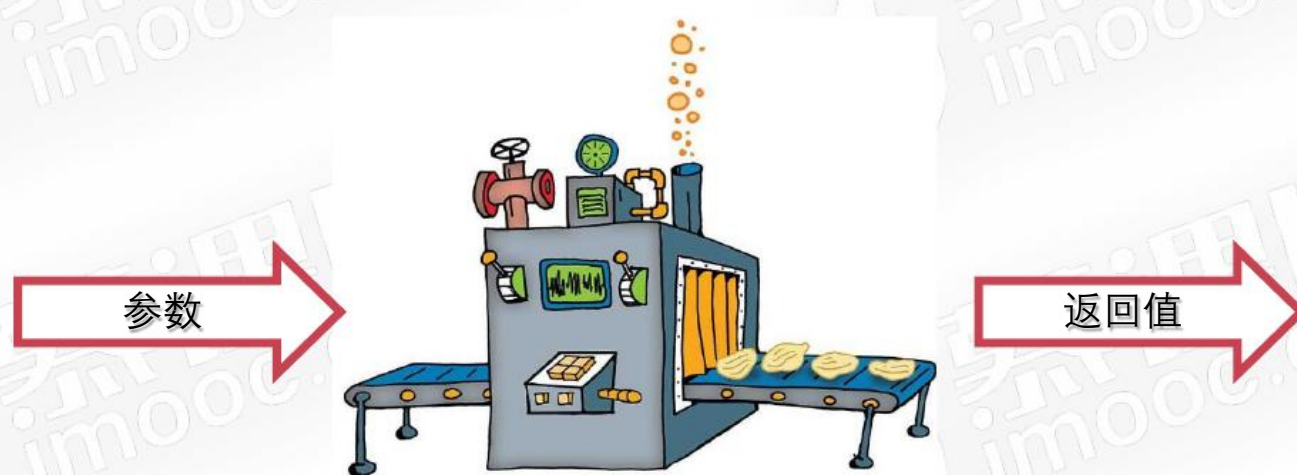
- ◆ 调用函数时，一旦遇见return语句则会立即退出函数，将执行权交还给调用者

```
function fun() {  
    console.log('A');  
    return 'B';  
    console.log('C');  
}  
console.log(1);  
var char = fun();  
console.log(char);  
console.log(2);
```

遇见return即退出函数

- ◆ 结合if语句的时候，往往不需要写else分支了
- ◆ 比如题目：请编写一个函数，判断一个数字是否是偶数

函数像一个“小工厂”



谢谢