# 函数

# 学习目标

## 能够使用函数封装一段代码重复使用

## 能够使用return让函数有返回值

## 能够使用arguments获取函数的所有实际参数

## 能够写出匿名函数和自调用函数的基本结构

# 函数的定义

## 导入

在前面我们学习了如何用\*好打印矩形，打印5行5列的代码可能是下面这样的:

|  |
| --- |
| for (var i = 0; i < 5; i++) {  var row = "";  for (var j = 0; j < 5; j++) {  str+="\*"+' ';  }  console.log(row);  } |

但是，假设我要打印 5行6列，7行8列，9行10列呢？在代码中，就会出现都是打印矩形这个功能的重复代码？那我们如何来做呢？

## 函数的定义

所谓的函数就是 函数是一段代码，它只定义一次，但可以被执行或调用任意次

语法结构:

|  |
| --- |
| function 函数名() {  函数体  } |

调用:

|  |
| --- |
| 函数名(); |

举例:

|  |
| --- |
| function getSum() {  Var num1 = 2;  Var num2 = 3;  var sum = num1 + num2;  console.log(sum)  }  getSum() |

## 总结

* 函数的作用: 代码的重复使用
* 定义函数的关键字: function
* 函数的函数名是自定义的，但是要符合小驼峰命名法
* 函数的函数名不能重复，一旦重复，后面的函数会把前面的函数覆盖

# 函数的参数

## 导入

刚才我们知道了，把5行5列的代码，放到一个函数格式中，调用就可以打印出矩形，

|  |
| --- |
| function square() {  for (var i = 0; i < 5; i++) {  var row = "";  for (var j = 0; j < 5; j++) {  row += "\*" + ' ';  }  console.log(row);  }  }  square(); |

但是，假设我要打印 5行6列，7行8列，9行10列呢？在代码中，就会出现都是打印矩形这个功能的重复代码？那我们如何来做呢？

## 语法格式

语法格式：

|  |
| --- |
| function 函数名(形式参数1,形式参数2…..) {  函数体  } |

调用

|  |
| --- |
| 函数名(实际参数1,实际参数2); |

形式参数：也称为形参 形式上参数运算的参数，在 函数声明 的小括号中

实际参数：也称为实参 实际上参与运算的参数，在 函数调用 的小括号中

## 总结

* 形式参数只是占位符，不参与代码的实际运算，形式参数实际上相当于变量，要符合变量的命名格式
* 实际参数是真实参与实际运算的参数，实际参数的个数不能超过形式参数的个数

# 函数的返回值

## 导入

在前面，我们练习for循环的时候，我们做过求某一个数的阶乘的问题，比如求4的阶乘，代码大概是下面这样的：

|  |
| --- |
| var fac = 1;  for (var i = 1; i <=4; i++) {  fac = fac \* i;  }  console.log(fac); |

但是 如果我要计算 4!+3!+2!,我就要把他们的阶乘结果获取到进行求和？怎么办呢？这时候就要用到函数的返回值

## 语法

函数体内部的return语句，表示返回。JavaScript遇到return语句，就直接返回return后面那个表达式的值。也就是说，return语句所带的那个表达式，就是函数的返回值。

语法结构:

|  |
| --- |
| function 函数名(参数, 参数, 参数) {  函数体  return 值  } |

调用:

|  |
| --- |
| var 变量 = 函数名(实参, 实参, 实参); |

## 总结

函数体内部的return语句，表示返回。JavaScript 引擎遇到return语句，就直接返回return后面的那个表达式的值，后面即使还有语句，也不会得到执行。也就是说，return语句所带的那个表达式，就是函数的返回值。return语句不是必需的，如果没有的话，该函数就不返回任何值，或者说返回undefined。

# arguments对象

## 导入

在前面，我们已经计算过2个数的和，它的代码可能是这样的：

|  |
| --- |
| function sum(num1, num2) {  var result = num1 + num2;  return result;  } |

但是，如果要计算任意三个数的和，任意四个数的和，任意n个数的和，我们就没法去定义函数的参数了，这时候就需要arguments对象来解决。

## 语法

JavaScript 允许函数有不定数目的参数，所以需要一种机制，可以在函数体内部读取所有参数。

arguments对象包含了函数运行时的所有实参，arguments[0]就是第一个参数，arguments[1]就是第二个参数，以此类推。这个对象只有在函数体内部，才可以使用。

|  |
| --- |
| var f = function () {  console.log(arguments[0]);// 1  console.log(arguments[1]);// 2  console.log(arguments[2]);// 3  }  f(1, 2, 3) |

## 总结

需要注意的是，虽然arguments很像数组，但它是一个对象。

# 函数的其他定义方式

## 导入

上面我们使用函数声明的方式，定义了一些函数，但是在我们的实际工作中，难免还可以见到其他一些定义函数的方式，为了让大家能认识其他开发者的代码，下面我们来介绍一下。

|  |
| --- |
| Function 函数名() {  函数体  } |

## 语法

**函数表达式**

|  |
| --- |
| var 变量 = function(参数, 参数, ...) {  函数体  return 值  } |
| 调用:  变量(实参, 实参); |

**自调用函数**

|  |
| --- |
| (function (形参, 形参) {    函数体  })(实参, 实参) |
| 自调用函数不能写return  自调用函数只能被调用一次 |

## 总结

在我们日常工作用，函数声明和函数表达式都比较常用

如果是函数表达式，function 后面不用写名字，表达式中的变量就可以认为是函数名