## 函数--function

> 当我们有一段代码，在不同的地 方可以多次调用，

  那么我们可以把这样的代码封装 一个函数中，然后通过函数的调 用来完成复用。

        function 函数名称（）{

            函数体，要重复利用的代码

        }

        注\*函数名称的规则同变量名称

        -调用函数：

        函数名称（）；

> 为了增加函数的复用性，我们  可以通过\*\*参数\*\*来让函数变得 更加强大，不变的是过程，

  可变的是过程操作的数据。

- 参数：

    > 函数在执行过程中接受的      外部传入的数据，这个数      据在函数每次调用的时候      传入

    > 为了能够函数在执行中，      调用到传入的数据，函        数在定义的时候可以定义      一个\*\*形参\*\*来接收他

- 形参：

    > 形式上的参数，和变量类      似，形参就是函数为接收      到数据创建的变量。

            例如：fn（al，bl）

                 fn (11,12)

    > 形参定义在函数中（），      多个形参使用逗号分隔，      \*\*该参数变量属于该函数      ，其他人不能访问\*\*。

- 实参：

    > 我们在调用函数的时候可以      根据具体的需求传入不同的      数据给这个函数参数变量，      这个过程我们称为传参，我      们把传给函数形参的数据成      为：实参

    - 函数执行过程中实际的参数

    - 传参过程其实就是变量赋值

    - 多个不同的形参与实参是一一对应的关系

#### 函数参数的个数

> 理论上函数的参数个数255个，但 是我们在定义函数的时候通常是  不会定义这么多的参数，一般最好 不要超过4个

  不过在某些时候会有特殊情况：把传入的数据进行相加

        function add(x, y, z) {

            console.log(x + y + z);

        }

- 如果实参的个数与形参个数

  不一样，那么就意味着形参上变量  不是每个都赋值了，形参默认值是  ：undefined

        例如：fn（x,y,z）;

             fn (0,1);

        那么这个 z 就默认为undefined

- 如果实参在使用的时候并不确定个数-不定参

    - 那么我们就需要对该形参做一些处理（给每一个形参做一个判断：必须是数字，并且不是NaN,不是Infinity）

            例如：

            // function add(x, y, z) {

                //     if (typeof x != 'number' || x != x || x == Infinity) {

                //         x = 0;

                //     }

                //     if (typeof y != 'number' || y != y || y == Infinity) {

                //         y = 0;

                //     }

                //     if (typeof z != 'number' || z != z || z == Infinity) {

                //         z = 0;

                //     }

                //     console.log(x + y + z);

                // }

                function add(x, y, z) {

                    x = x || 0;

                    y = y || 0;

                    z = z || 0;

                    console.log(x + y + z);

                }

                add(1, 2);

             不定个数的参数如果太多，形参定义就比较麻烦

- 使用数组

    - 声明一个变量，来存储数组里面的值

              // function add(arr) {

                  //     var result = 0;

                  //     for (var i=0; i<arr.length; i+=1) {

                  //         result += arr[i];

                  //     }

                  //     console.log(result);

                  // }

- 在JS中也考虑到了这样的需求，所以在函数有一个类似数组的东西去给我们调用

    - 每当函数调用的时候。系统会      在函数的内部自动创建一个变      量：\*\*argumentts\*\*，他会      把当前这次调用传入的实参赋      值给arguments，               arguments是一个集合（类似      数组）

          例如：

          function add() {

                  console.log(arguments);

                  var result = 0;

                  for (var i=0; i<arguments.length; i+=1) {

                      result += arguments[i];

                  }

                  console.log(result);

              }

              add(1,2,3,7,4,3,3);

#### 函数返回值

- 函数执行完成以后返回给外部调用的结果，用：return 值 =>表示

    - 提高函数的复用性、灵活性

            例如：

            function add(x, y) {

                    // console.log(x + y);

                    return x + y;

                }

                var a = add(1, 2);

                console.log(a);

                document.body.innerHTML = add(10, 20);