## 复习

条件分支语句：

if语句

三元表达式： 条件表达式?值1：值2；

switch语句

|  |
| --- |
| 1. switch (表达式) { 2. case 值1： 3. 执行体; 4. break; 5. case 值2： 6. 执行体; 7. break; 8. …… 9. default: 10. 执行体; 11. break; 12. } |

循环语句

for (循环变量初始值; 条件表达式; 变量的改变) {

循环语句；

}

js语句；

# 循环语句

## do while

do while:后测试循环语句。直接执行循环语句，然后再进行出口判断，满足条件继续循环；如果条件为假跳出do while，执行后面其他的语句。

循环变量必须书写在循环语句外面。

|  |
| --- |
| 1. do { 2. 循环语句; 3. }while (条件表达式) |

|  |
| --- |
| 1. // 后测试循环语句。先执行循环语句 2. // 注意循环变量必须声明在循环语句外面 3. var i = 3; 4. do { 5. console.log(i); 6. i += 4; 7. }while (i < 15) 8. console.log("结束"); 9. /\* 10. i=3;直接就执行循环语句输出3;i+=4出口判断 11. i=7;i<15真，继续执行循环语句输出7；i+=4 12. i=11;i<15真，继续执行循环语句输出11；i+=4 13. i=15;i<15假立即跳出do while语句 14. 执行后面其他的 15. 输出“结束” 16. \*/ |

do while循环语句至少会执行一次。

|  |
| --- |
| 1. // do while循环语句至少会执行一次 2. // 循环变量必须书写在外面 3. var i = 3; 4. do { 5. console.log(++ i); 6. i += 4; 7. }while (i < 4); 8. /\* 9. i=3;直接执行循环语句输出4；i+=4 10. i=8;出口判断i<4假立即跳出do while语句 11. \*/ |

## while

while语句：前测试循环语句。先判断条件，为真才执行循环语句

注意：循环变量必须书写在循环语句外面

|  |
| --- |
| 1. // 注意：循环变量必须书写在外面 2. var i = 1; 3. while (i < 10) { 4. console.log(i); 5. i += 4; 6. } 7. console.log("结束"); |

while语句都可以改为for语句

|  |
| --- |
| 1. // 改为for 2. for (var i = 1; i < 10; i += 4 ) { 3. console.log(i); 4. } 5. console.log("结束"); |

while语句中输出语句位置非常重要。

|  |
| --- |
| 1. // 注意：循环变量必须书写在外面 2. var i = 1; 3. while (i < 10) { 4. i += 4; 5. console.log(i); 6. } 7. console.log(i + "结束"); 8. /\* 9. i=1;i<10真;i+=4;输出5 10. i=5;i<10真;i+=4;输出9 11. i=9;i<10真;i+=4;输出13 12. i=13;i<10假立即跳出 13. 输出13结束 14. \*/ |

## break

break: 我已经找到了一个满足条件的结果，不再继续往下执行

|  |
| --- |
| 1. // break:执行break立即跳出循环语句 2. for (var i = 1 ; i < 200 ; i += 2) { 3. // 能被3整除 4. if (i % 3 === 0) { 5. console.log(i); 6. break; 7. } 8. } 9. console.log("结束"); 10. /\* 11. i=1;i<200真;i%3===0假；i+=2 12. i=3;i<200真;i%3===0真输出3;执行break立即跳出当前所在层的for语句 13. 输出”结束“ 14. \*/ |

作用：在循环语句for,do while,while中遇见break,只能结束break所在层的循环不能控制外层循环。

|  |
| --- |
| 1. for (var i = 1; i <= 3 ; i ++ ) { 2. for (var j = 1; j <= 3; j ++) { 3. // 进行限制 4. if (j == 2) { 5. console.log(i,j); 6. // break只能控制j不能控制i 7. break; 8. } 9. } 10. } 11. console.log(j + "结束"); |

break控制外层循环：给外层循环添加标签，break引用

|  |
| --- |
| 1. waiceng: for (var i = 1; i <= 3 ; i ++ ) { 2. for (var j = 1; j <= 3; j ++) { 3. // 进行限制 4. if (j == 2) { 5. console.log(i,j); 6. // break只能控制i不能控制j 7. break waiceng; 8. } 9. } 10. } 11. console.log(i + "结束"); 12. console.log(j + "结束"); 13. /\* 14. vari=1;i<=3真；varj=1;j<=3真；j==2假；j++ 15. j=2;j<=3真；j==2真；输出1,2执行break waiceng;跳出i循环 16. 输出1结束 17. 输出2结束 18. \*/ |

## 1.4continue

continue(继续)：这个结果不是我想要的，赶快测试下一个。

|  |
| --- |
| 1. // continue结束该次循环，立即执行变量改变 2. for (var i = 1; i < 10 ; i += 2) { 3. if (i % 3 === 0) { 4. continue; 5. } 6. console.log(i); 7. } 8. console.log(i + "结束"); 9. /\* 10. vari=1;i<10真；执行i=1这次循环i%3===0假输出1; 11. i=3;i<10真；执行i=3这次循环i%3===0真；执行continue，立即结束i=3这次循环 12. i=5;i<10真；执行i=5这次循环i%3===0假输出5; 13. i=7;i<10真；执行i=7这次循环i%3===0假输出7; 14. i=9;i<10真；执行i=9这次循环i%3===0真执行continue;立即结束该次循环 15. i=11;i<10假立即跳出i循环 16. 输出11结束 17. \*/ |

continue:遇见continue结束该次循环，立即进入下一次循环

continue也是只能结束所在层的这次循环，不能控制外层循环

|  |
| --- |
| 1. for (var i = 1; i <= 3; i ++) { 2. for (var j = 1; j <= 3; j ++) { 3. if (j == 2) { 4. // continue只能控制j 5. continue; 6. } 7. console.log(i,j); 8. } 9. } 10. console.log(j); 11. /\* 12. vari=1;i<=3真；varj=1;j<=3真；j==2假；输出1,1;j++ 13. j=2;j<=3真；j==2真遇见continue结束该次循环;j++ 14. j=3;j<=3真；j==2假；输出1,3;j++ 15. j=4;j<=3假立即跳出j循环 16. i=2;i<=3真；varj=1;j<=3真；j==2假；输出2,1;j++ 17. j=2;j<=3真；j==2真遇见continue结束该次循环;j++ 18. j=3;j<=3真；j==2假；输出2,3;j++ 19. j=4;j<=3假立即跳出j循环 20. i=3;i<=3真；varj=1;j<=3真；j==2假；输出3,1;j++ 21. j=2;j<=3真；j==2真遇见continue结束该次循环;j++ 22. j=3;j<=3真；j==2假；输出2,3;j++ 23. j=4;j<=3假立即跳出j循环 24. i=4;i<=3假立即跳出i循环 25. 输出4 26. \*/ |

continue想控制外层，也是通过标签绑定

|  |
| --- |
| 1. waiceng: for (var i = 1; i <= 3; i ++) { 2. for (var j = 1; j <= 3; j ++) { 3. if (j == 2) { 4. // continue只能控制i 5. continue waiceng; 6. } 7. console.log(i,j); 8. } 9. } 10. console.log(i); |

break 和 continue可以简化我们的计算过程（指的是计算机的计算过程）

案例：用户输入数字，判断该数是不是质数

|  |
| --- |
| 1. // 的到数字8 2. var num = parseInt(prompt("请输入数字")); 3. for (var i = 2; i < num ; i ++) { 4. // 如果找到了一个约数循环就结束 5. if (num % i === 0) { 6. console.log(num + "不是质数"); 7. break; 8. } 9. } |

案例：输出2-100以内所有的质数。

|  |
| --- |
| 1. // 一一列举2-100以内所有的数字 2. waiceng: for (var i = 2; i <= 100 ; i ++) { 3. // i=20,判断i是不是质数，是输出；不是赶快测试i++ 4. // 一旦找到i除了1和本身之外的一个约数，断定i不是质数 5. for (var j = 2; j < i; j ++) { 6. // 判断j是不是i的约数 7. if (i % j === 0) { 8. // 找到了一个约数，断定i不是我们想要的质数，赶快测试i++ 9. continue waiceng; 10. } 11. } 12. console.log(i + "是质数"); 13. } |

# 二、函数

## 2.1 函数的声明

函数就是功能。

我们可以自己封装一些语句在函数内部，函数就具有了某一种特定的功能。

声明函数：

使用关键字function 空格隔开函数名(参数) {},在大括号内部封装语句，表示一个整体

|  |
| --- |
| 1. function 函数名(参数) { 2. 语句； 3. } |

函数声明语句并不会执行，只是告诉我们函数可以实现某一种功能，内部有哪些语句。

|  |
| --- |
| 1. // 声明函数 2. function xiyiji() { 3. console.log("接水"); 4. console.log("洗衣服"); 5. console.log("漂洗"); 6. console.log("甩干"); 7. console.log("烘干"); 8. } |

函数名命名规则和变量相同。

函数优点1：将一些重复性的语句封装在函数内部，可以声明一次多次调用。简化书写

## 2.2 调用

函数名();

函数是一个整体，在调用时，内部语句都会执行。

函数执行的位置和声明没有位置没有关系只和调用位置有关。

|  |
| --- |
| 1. console.log(1); 2. // 声明函数 3. function xiyiji() { 4. console.log("接水"); 5. console.log("洗衣服"); 6. console.log("漂洗"); 7. console.log("甩干"); 8. console.log("烘干"); 9. } 10. // 调用，函数内部语句都会执行 11. xiyiji(); 12. console.log(2); 13. // 函数声明一次可以多次调用 14. xiyiji(); |



## 2.3 函数的参数

函数可以帮我们封装一些代码，代码可以重复调用，函数留了一个接口，就是我们的参数，可以通过参数的变化让我们的函数发生不同作用。

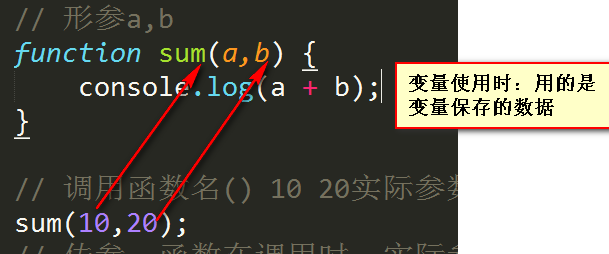
参数都是变量：命名规则与变量一样。

调用过程中给函数传递参数的过程就是一个给变量赋值的过程。

形式参数：在函数声明语句中小括号内部书写的是形式参数，简称形参

实际参数：在函数调用语句中小括号内部书写的是实际参数，简称实参

传参：在函数调用时，将实际参数赋值给形式参数的过程。



形参也有数据类型，是动态变化，根据实际参数赋值动态变化的。

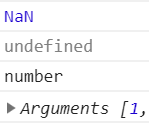
参数个数问题：

arguments: 由实际参数组成的类数组对象。

实际参数个数多于形式参数，多于实际参数直接舍弃。

实际参数个数少于形式参数，优先将实参赋值前面的形参，没有赋值是undefined。

|  |
| --- |
| 1. sum(1); |



函数的优点2：函数有参数，相当于给我们提供一个API接口，我们可以通过接口去调用函数，执行不同的操作，后面封装函数的时候，只需要了解API的用途就够了，就是传参之后有什么结果，不用去了解函数里面的构造。不论是自己的函数还是用的别人封装好的函数，都只需要知道怎么用就够了。

## 2.4 return

return返回值

return也可以接收参数（变量），作为函数的返回值。return并不会输出，想看输出console.log（）

|  |
| --- |
| 1. // 声明函数 2. function sum(a,b) { 3. // return 4; 4. // 只要调用函数可以的到a+b结果 5. return a + b; 6. } 7. console.log(sum(1,4)); |

返回值相当于将我们的函数变成了一个表达式。利用这个特性，我们可以将函数作为一个实际参数，传递给另外一个函数。

|  |
| --- |
| 1. console.log(sum(10,sum(1,19))); |

return特点：函数内部遇见return,直接返回

|  |
| --- |
| 1. // 函数内部遇见return直接返回值 2. function fun() { 3. console.log(1); 4. console.log(2); 5. console.log(3); 6. return; 7. console.log(4); 8. } 9. // 函数调用 10. fun(); |

函数优点3：函数内部书写return，可以让函数作为参数，传递，有利于模块化编程

## 2.4 模块化编程

人类从古至今，习惯将事情分工，将一些内容做成一些公共模块，模块可以重复反复使用。

模块化编程：将一些基础的公共的部分单独封装到一个函数内，可以多次被调用。

注意：模块化编程，可以让我们的程序更加优化，各个小模块要尽量功能单一，提高重复使用率。

案例：输出100以内的质数，模块化编程。

案例：输出100以内的完美数

|  |
| --- |
| 1. /\* 2. 找100以内的完美数：（一个数的约数除了它本身外其他约数和还等于这个数）。 3. 比如：6 = 1 + 2 + 3 6完美数 4. 8 != 1 + 2 + 4 8不是完美数 5. 输出100以内所有的完美数： 6. 判断任何数字是不是完美数 7. 一个数字除了本身之外其他约数和 8. \*/ 9. /\* 10. \*函数1：一个数字约数和（不包含本身） 11. \*参数： 一个数字 12. \*返回值：约数和 13. \*/ 14. function yueshuhe(n) { 15. // 求n约数和 16. // 累加器 17. var sum = 0; 18. for (var i = 1; i < n; i ++) { 19. // 约数和 20. if (n % i === 0) { 21. sum += i; 22. } 23. } 24. // 返回值 25. return sum; 26. } 27. /\* 28. \*判断任何数字是不是完美数 29. \*参数： 一个数字 30. \*返回值：布尔值 31. \*/ 32. function isWms(num) { 33. // 判断num是不是完美数 34. if (num === yueshuhe(num)) { 35. return true; 36. }else { 37. return false; 38. } 39. } 40. // 输出100以内所有的完美数 41. for (var i = 1; i <= 100; i ++) { 42. // i是完美数，输出 43. if (isWms(i)) { 44. console.log(i); 45. } 46. } |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |