基础知识：
1.函数的定义方式：
    a.函数声明；function name() {};
    b.函数表达式；var func = function() {};
*注意：只有匿名函数表达式才简称函数表达式；
2.parseInt();
    把字符串中的有效"整数"内容取出来转换为Number类型，从左往右读，读到非整数即停止；
3.互相嵌套的函数，外面的函数不能访问里面函数的内容，里面的函数可以访问外面的函数的内容。即越往里权限越大：
eg:
```
    function test() {
        var a = 123;
        function demo() {
            var b = 456;
            console.log(a); //123
        }
        demo();
        console.log(b); //报错
    };
    test();
```
4.请使用css和html写一个三角形；（使用js实现好看的效果）
```
    .sjx{
        //width: 100px;
        //height: 100px;
        width: 0px;
        height: 0px;
        //background-color: yellow;
        border-top: 50px solid red;
        border-right: 50px solid orange; (transparent)
        border-bottom: 50px solid green; (transparent)
        border-right: 50px solid blue; (transparent)
    }
    <div class="sjx"></div>
```
5.父元素为一个div，宽高不固定，子元素是一个块元素，宽高已知，如何实现子元素在父元素内水平，垂直都居中？
方法1：相对定位（操作son）
```
    .father{
        width: 600px;
        height: 600px; //宽高不固定
        border: 1px solid yellow;
    }
    .son{
        width: 160px;
        height: 160px;
        border: 1px solid blue;
     ***margin: 0 auto; //左右居中
     ***position: relative;
     ***top: 50%;
     ***margin-top: -80px; //负的高度的一半，上下居中
    }
    <div class="father">
        <div class="son"></div>
```

```
    </div>
方法2: Flex弹性布局（操作father）
    .father{
        width: 600px;
        height: 600px; //宽高不固定
        border: 1px solid yellow;
     ***display: flex;
     ***align-items: center; //垂直方向居中
     ***justify-content: center; //水平方向居中
    }
    .son{
        width: 160px;
        height: 160px;
        border: 1px solid blue;
    }
    <div class="father">
        <div class="son"></div>
    </div>
方法3: transform属性（操作son）
    .father{
        width: 600px;
        height: 600px; //宽高不固定
        border: 1px solid yellow;
    }
    .son{
        width: 160px;
        height: 160px;
        border: 1px solid blue;
     ***position: relative;
     ***top: 50%;
     ***left: 50%;
     ***transform: translate(-50%, -50%); //改变中心点,向左向上移动自身宽高的一半
    }
    <div class="father">
        <div class="son"></div>
    </div>
6.请写出window.foo的值;
(window.foo || (window.foo = 'bar')); //bar
7.写一个函数,实现n的阶乘;
//使用递归的思想解决问题
//使用递归要注意两点: 1.找规律; 2.找出口;
//规律: n! = n * (n - 1)!
//出口: 0! = 1
    function mul(n) {
        if(n == 1 || n == 0) {
            return 1; //出口
        }
        return n * mul(n - 1); //规律,递归就是return公式实现循环计算;
    };
8.写一个函数,实现斐波那契数列;（第三位数等于前两位数之和）
eg:1 1 2 3 5 8...
    function fb(n) {
        if(n == 1 || n == 2) {
```

```
            return 1;
        }
        return fb(n - 1) + fb(n - 2)
    };
9.js: 单线程,解释性语言: 翻译一句执行一句;
```

预编译

```
基础知识:
1.imply global暗示全局变量: 即任何变量,如果变量未经声明就赋值,此变量为全局对象所有;
    eg: a = 123; ---> window.a = 123;
2.一切声明的全局变量,全是window的属性;
    eg: var a = 123; ---> window.a = 123;
3.window 就是全局的域;
    var a = 123;
    var b = 234;
    var c = 456;
    ==== 相当于 ====
    window {
        a : 123,
        b : 234,
        c : 456;
    }
    console.log(a); === console.log(window.a);
4.但凡在全局定义的变量window里面都有对应的属性;
5.预编译发生在函数执行的前一刻;
6.形参名,变量名,函数名一样,我们需要解决一个优先级顺序的问题,即谁覆盖谁的问题,执行顺序是怎样影响变量,影响函数,这就是预编译需要解决的问题;
eg:
    function fn(a) {
        console.log(a); //function a() {}
        var a = 123; //预编译提升声明变量,但是赋值还是需要看的,将a = 123代替AO对象里a属性的值
        console.log(a); //123
        function a() {} //预编译已经将其提升,所以不用再看
        console.log(a); //123
        var b = function() {} //var b不用看了,修改AO里的b,值为function() {}
        console.log(b); //function() {}
        function d() {}
        console.log(d); //function d() {}
    }
    fn(1);
```

```
运行三部曲:
    1.语法分析; (通篇扫描一遍,看看有没有语法错误,但不执行)
    2.预编译;
    3.解析执行; (解释一行执行一行)
```

```
function test(a,b) {
    console.log(a); //1
    c = 0;
    var c;
```

```
    a = 3;
    b = 2;
    console.log(b); //2
    function b() {}
    function d() {}
    console.log(b); //2
    console.log(d); //function d() {}
}
test(1);
---------------------------
预编译四部曲:
1.创建AO对象(执行期上下文);
    AO{

    }
2.找形参和变量声明;
(AO找的是局部范围里的,GO找的是全局范围里的),将形参和变量名作为AO属性名,值为undefined;
    AO{
        a : undefined,
        b : undefined,
    }
3.将实参值和形参相统一;
    AO{
        a : 1,
        b : undefined,
    }
4.在函数体里面找函数声明,值赋予函数体;
    AO{
        a : function a() {},
        b : undefined,
        d : function d() {},
    }
```

```
function test(a,b) {
    console.log(a); //function a() {}
    console.log(b); //undefined
    var b = 234;
    console.log(b); //234
    a = 123;
    console.log(a); //123
    function a() {}
    var a;
    var b = function() {}
    console.log(a); //123
    console.log(b); //function() {}
}
test(1);
```

```
1.函数声明整体提升: 不管是在函数声明之前调用函数还是在函数声明之后调用函数,其本质都是在函数声明之后调用
函数,他始终会把函数声明提升到逻辑的最前面;
    console.log(a); //function a() {}
    var a = 123;
```

```
    function a() {};
2.变量 声明提升: 只是将定义变量部分提升,赋值部分不提升;
    console.log(a); //undefined
    var a = 123;
3.GO全局状态下,预解析没有第三步,并且第一步是生成一个GO对象; GO === window
    var a = 123;
    function a() {};
    console.log(a); //123
4.
function test() {
    var a = b = 123;
    console.log(window.a); //undefined GO里面没有a定义
    console.log(window.b); //123
    console.log(a); //123
    console.log(b); //123 AO里面没有的话往上找GO
}
test();
----------
GO{
    b : undefined,123
}
AO{
    a : undefined,123
}
5.
console.log(test); //function test() {} 和下面打印出来的fun指的不一样
function test(test) {
    console.log(test); //function test() {}
    var test = 234;
    console.log(test); //234
    function test() {}
}
test(1);
----------
GO{
    test : function test() {}
}
AO{
    test ; undefined,1,function test() {},234
}
6.
var global = 100;
function fn() {
    console.log(global); //100 先去AO里面找,AO里面没有,再去GO上找
}
fn();
---------------
GO{
    global : undefined,100
    fn : function fn() {},
}
AO{
```

```
}
7.
global = 100;
function fn() {
    console.log(global); //undefined
    global = 200; //可以改变AO里对应属性的值
    console.log(global); //200
    var global = 300;
}
fn();
console.log(global); //100
var global;
--------------
GO{
    global : undefined,100
    fn : function fn() {}
}
AO{
    global : undefined,200,300
}
8.
function test() {
    console.log(b); //undefined
    if(a) { //a = undefined,所以b = 100不执行
        var b = 100; //AO 提升
    }
    console.log(b); //undefined
    c = 234;
    console.log(c); //234
}
var a;
test();
console.log(a); //undefined
a = 10;
console.log(a); //10
console.log(c); //234
-------------------------
GO{
    a : undefined,10
    c : 234,
    test : function test() {}
}
AO{
    b : undefined
}
9.
function bar() {
    return foo; //function foo() {} 在完成预编译第四步后进来走程序直接return,必然返回function
    foo = 10;
    function foo() {}
    var foo = 11;
}
console.log(bar());
```

```
10.
console.log(bar());
function bar() {
    console.log(foo); //function foo() {}
    foo = 10;
    console.log(foo); //10
    function foo() {}
    var foo = 11;
    return foo; //11
}
11.
a = 100;
function demo(e) {
    function e() {}
    arguments[0] = 2; //实参列表,传参与传参的形参位相映射即 e = 2
    console.log(e); //2
    if(a) { //a = undefined,所以if里面的语句不执行
        var b = 123;
        function c() {

        }
    }
    var c;
    a = 10;
    var a;
    console.log(b); //undefined
    f = 123;
    console.log(c); //理想状态下是function,因为规定里刚刚新添if语句里面不能定义函数
    console,log(a); //10
}
var a;
demo(1);
console.log(a); //100
console.log(f); //123
-------------
GO{
    a : undefined,100
    demo : function demo() {}
    f : 123
}
AO{
    e : undefined,1,function e() {},2
    b : undefined
    c : undefined,function c() {}
    a : undefined,10
}
12.
var str = false + 1;
document.write(str); //1
13.
var demo = false == 1; //将false == 1的结果赋给demo
document.write(demo); //false
14.
```

只有一种情况使用没有定义的变量是不报错的,就是typeof(a) ---> 返回的是字符串类型的undefined;
typeof(null) ---> "object"
(null本来是原始值,但在typeof这里被认为是给对象占位的,所以认为null是个对象)
15.
```
if(typeof(a) && -true + (+undefined) + "") { //有数学符号的一般都转化为数字,转不了的为NaN
    //结果为: “undefined” && "NaN"
    document.write('基础扎实'); //能够打印出来
}
```
16.
```
if(11 + "11" * 2 == 33) { //不管乘法两边有什么东西,都是要转换成数字的,"11" * 2 = 22
    document.write('基础扎实');
}
```
17.
```
!!" " + !!"" - !!false || document.write('你觉得能打印,你就是猪');
```
非非空格字符串 --> 变成布尔值,变成ture + false - false = 1 || ...
18.写一个函数,功能是告知你所选定的小动物的叫声;
19.写一个函数,实现加法计数器;
20.定义一组函数,输入数字,逆转并输出汉子形式;