task17

问答

1.函数声明和函数表达式有什么区别（*）

```
function functionName(){};  //函数声明
var functionName = function(){};   //函数表达式
//区别：函数声明可以提前，函数表达式不可提前
```

2.什么是变量的声明前置？什么是函数的声明前置（**）

```
//函数外变量声明前置
console.log(a);
var a = 0;
//等于
var a;
console.log(a);
a = 0;

//函数内变量声明前置
function functionName(){
console.log(a);
var a = 0;
};
//等于
function functionName(){
var a;
console.log(a);
a = 0;
};

//函数的声明前置
function();
function functionName(){};
//等于
function functionName(){};
function();
```

3.arguments 是什么（*）

arguments是一个类数组对象。数组中包含了调用arguments的函数的参数。arguments只在函数体内部生效。

```
//比如
function a(x,y,z){
return arguments[1];
}
//此函数 return y
```

4.函数的重载怎样实现（**）

JavasScript不支持函数的重载，但是有变通的办法，可以利用JavaScript中的特殊对象arguments来模拟函数重载

```
function functionName(){
    var x = 0;
    for(var i = 0;i < arguments.length; i++){
        x = x + arguments[i];
    }
    return x;
}
console.log(functionName());
```

5.立即执行函数表达式是什么？有什么作用（*）

表达式有2种形式:

```
(function (){})()
(function (){}())

/*作用：
1.模拟块作用域，JS没有块作用域（block），只有函数作用域，在同时调用多个库的情况下，很容易造成对象或者变量的覆盖，使用立即执行函数可以创建独立的作用域，不互相影响。
2.解决闭包冲突
3.模拟单例*/
```

6.什么是函数的作用域链（**）

作用域指的是变量或函数能够生效的范围，分为全局作用域和局部作用域；变量在函数外部声明或者声明的时候没有带var就是全局变量，反之为布局变量。

作用域链就是函数对象内可以通过代码访问的属性，以及一系列仅供JavaScript引擎访问的内部属性。其中一个内部属性是[[Scope]]，由ECMA-262标准第三版定义，该内部属性包含了函数被创建的作用域中对象的集合，这个集合被称为函数的作用域链，它决定了哪些数据能被函数访问。

当代码在一个环境中执行时，会创建变量对象的一个作用域链。作用域链的用途是保证对执行环境有权访问的所有变量和函数的有序访问。作用域链的前端始终都是当前执行代码所在环境的变量对象。如果这个环境是函数，则将其活动对象作为变量对象。活动对象最开始只

包含一个变量，就是arguments对象，作用域中的下一个变量来自包含环境，在下一个变量对象则来自下一个包含环境。这样，一直延续到全局执行环境；全局执行环境的变量对象始终都是作用域链中的最后一个对象。

代码

1.以下代码输出什么？（难度**）

```
     function getInfo(name, age, sex){
            console.log('name:',name);
            console.log('age:', age);
            console.log('sex:', sex);
            console.log(arguments);
            arguments[0] = 'valley';
            console.log('name', name);
        }
        getInfo('hunger', 28, '男');
/*
name: hunger
age: 28
sex: 男
["hunger", 28, "男"]
name valley
*/
getInfo('hunger', 28);
/*
name: hunger
age: 28
sex: undefined
["hunger", 28]
name valley
 */
getInfo('男');
/*
name: 男
age: undefined
sex: undefined
["男"]
name valley
 */
```

2.写一个函数，返回参数的平方和？如（难度**）

```
   function sumOfSquares(){
   for(var i=0;i<arguments.length;i++{
```

```
   var sum = 0;
   sum += arguments[i]*arguments[i];
   return sum;
   }
   }
   sumOfSquares(2,3,4);   // 29
   sumOfSquares(1,3);   // 10
```

3.如下代码的输出？为什么 （难度* ）

```
 console.log(a);
    var a = 1;
    console.log(b);
    //undefined  a未赋值
    //报错       b未声明
```

4.如下代码的输出？为什么 （难度* ）

```
sayName('world');
sayAge(10);
function sayName(name){
    console.log('hello ', name);
}
var sayAge = function(age){
    console.log(age);
};

//hello world
//报错，因为sayAge(10);之前未声明sayAge是一个函数
```

5.如下代码的输出？为什么 （难度** ）

```
function fn(){}
var fn = 3;
console.log(fn);
//3，因为函数fn 被变量fn=3 覆盖
```

6.如下代码的输出？为什么 （难度* ）

```
function fn(fn2){
   console.log(fn2);
   var fn2 = 3;
   console.log(fn2);
   console.log(fn);
```

```
   function fn2(){
        console.log('fnnn2');
    }
 }
fn(10);

//提升变量声明和函数声明后写成：
function fn(fn2){
   var fn2;
   function fn2(){
        console.log('fnnn2');
    }
   console.log(fn2);
   fn2 = 3;
   console.log(fn2);
   console.log(fn);
 }
fn(10);
//输出
function fn2(){
        console.log('fnnn2');
    }
    3
function fn(fn2){
   console.log(fn2);
   var fn2 = 3;
   console.log(fn2);
   console.log(fn);
   function fn2(){
        console.log('fnnn2');
    }
 }
```

7.如下代码的输出？为什么 （难度*）

```
var fn = 1;
function fn(fn){
     console.log(fn);
}
console.log(fn(fn));
//报错，因为fn最后是一个等于1的全局变量
```

8.如下代码的输出？为什么 （难度**）

```
//作用域
console.log(j);
```

```
console.log(i);
for(var i=0; i<10; i++){
    var j = 100;
}
console.log(i);
console.log(j);
//undefined   i未赋值
//undefined   j未赋值
//10          for循环最后i=10
//100         var j = 100;
```

9.如下代码的输出？为什么（难度**）

```
fn();
var i = 10;
var fn = 20;
console.log(i);
function fn(){
    console.log(i);
    var i = 99;
    fn2();
    console.log(i);
    function fn2(){
        i = 100;
    }
}
//等于
var i;
var fn;
function fn(){
    var i;
    function fn2(){
        i = 100;
    }
    console.log(i);
    i = 99;
    fn2();
    console.log(i);

}
 fn();
i = 10;
fn = 20;
console.log(i);
//最后输出
//undefined
//100
```

```
//10
```

10.如下代码的输出？为什么 （难度*）

```
var say = 0;
(function say(n){
    console.log(n);
    if(n<3) return;
    say(n-1);
}( 10 ));
console.log(say);

//先分解中间那个立即执行函数表达式，等于
var say = 0;
(function say(n){
    console.log(n);
    if(n<3) return;
    say(n-1);
}
say(10);)
console.log(say);

//把所有声明提前，变成
var say;
(function say(n){
    console.log(n); //第一次10，之后每次减1，直到2
    if(n<3) return; //最后return出一个2，结束循环
    say(n-1);
}
say(10);)
say=0;
console.log(say);  //这里为0
//最后就是10到2，接一个0: 10,9,8,7,6,5,4,3,2,0
```