试题:(开卷 总分100分)

1. 程序分析题

1.

var a;

console.log(a); //打印undefined

var：定义变量也叫声明空间；

var a:则表示定义一个变量，变量名为a（相当于给声明的空间取一个名字a）,定义变量的目的是为了方便以后使用变量空间中存储的数据。

分号在javascript语法中表示一条语句结束（也可以不加，换行时系统自动补上）。

console.log()是系统提供的一个方法，表示将（）中的内容输出到控制台，（）中如果是字符串就直接打印，如果是变量分四种情况：

1. 如果变量没有声明没有初始化直接打印，console.log(a)程序报错；
2. 如果声明了变量但是没有初始化，打印undefined。
3. 如果没有声明变量但是初始化了，当代码运行到当前代码时才会出现变量提升（将var 变量名; 隐式提前到当前js代码块的最前面隐式声明成为一个全局变量），并且赋值，此时console.log(变量名)打印的是该变量空间中存储的数据；
4. 如果如果声明变量并且初始化了，也出现变量提升（将var 变量名; 隐式提前到当前代码块的最前面），和（3） 的不同的是直接有var修饰的变量只会提升到当前代码块的前面，可能是局部变量也可能是局部变量，当代码运行到当前代码的时候再赋值；此时console.log(变量名)打印的是变量空间中存储的数据。

由此可见，这一题声明了变量a但是没有初始化，如情况（2），打印undefined

2.

console.log(typeof [1, 2]); //打印object

console.log(typeof 'leipeng'); //打印string

var i = true;

console.log(typeof i); //打印boolean

console.log(typeof 1); //打印number

var a;

console.log(typeof a); //打印function

function a(){;};

console.log(typeof a); //打印function

console.log(typeof 'true'); //打印string

typeof()是系统提供的一种判断数据类型的方法，（）中写参数，

js有7大数据类型：number、string、Boolean、null、undefined、object、function。

typeof()返回值是有6种数据类型：number、string、boolean、null、undefined、function、object【对象、null、array数组】

也可表示成typeof+空格+参数，例如：typeof ‘666’;

console.log(typeof [1, 2]);

[1, 2]是数组直接量，是一个数组，返回值是object。 打印object

直接量，就是可以在程序中直接使用的数据。主要有字符串直接量，数字直接量，数组直接量，对象直接量，函数直接量、undefined直接量和null直接量。

console.log(typeof 'leipeng');

'leipeng'是字符串直接量，返回值是string。 打印string

var i = true;

var：定义变量也叫声明空间

var i:则表示定义一个变量，变量名为i（相当于给声明的空间取一个名字i）,定义变量的目的是为了方便以后使用变量空间中存储的数据。

=：表示赋值，把等号左边的数据赋给等号右边，此处表示把true这个boolean类型的数据赋给i这个变量

分号在javascript语法中表示一条语句结束（也可以不加，换行时系统自动补上）。

console.log(typeof i);

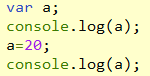
i是变量，代表它存储的数据true，true是boolean类型，所以打印boolean

console.log(typeof 1);

1是数字直接量，所以打印number

后面的代码涉及一个知识点变量提升与函数提升

1. 变量提升
2. 有var修饰,例如：

在程序进入当前代码块时，首先会所有有var修饰的变量隐式提前到当前代码块的最前面（可能是全局变量、也可能是局部变量，具体看当前代码块的位置）。所有这里将var a; 隐式提前，代码相当于

第一个打印时，声明了变量a但是没有初始化，打印undefined

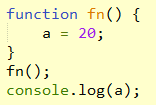
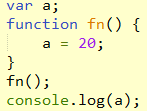
第二个打印时，声明了变量a并且初始化为20，打印20

1. 没有var修饰，例1：

在程序运行到变量赋值这一行的时候才会出现变量提升（隐式提升为全局变量）

所以在打印时变量a没有声明没有初始化，程序报错。

例2：

在fn()运行时，运行到代码a=20时，出现变量提升（隐式提升为全局变量）,会将var a; 隐式提升到函数体外面，代码相当于

所以在打印时变量a已经声明并且初始化了，打印20。

1. 函数提升

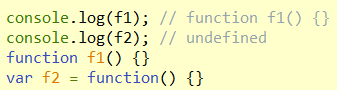
js中创建函数有两种方式：函数声明式和函数表达式。只有函数声明才存在函数提升！

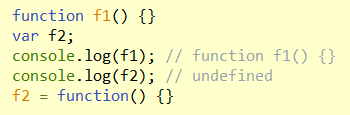
函数提升会把整个函数包括内容全部提升

函数声明式：function f1() {}

函数表达式：var f2 = function() {}

例如：

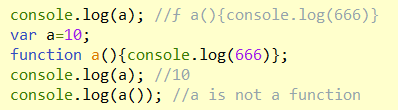
代码等同于：



1. 变量提升与函数提升

从上面我们知道函数和变量都会提升，但是谁在前谁在后呢？看例子吧！

例如：



由此可见，函数提升要比变量提升的优先级要高一些，且不会被变量声明覆盖，但是会在变量赋值之后被变量覆盖。（函数和变量的提升的位置，谁在前谁在后，并不影响结果）

所以接下来就利用上面的知识就好分析了，

var a;

console.log(typeof a); //打印function

function a(){;};

console.log(typeof a); //打印function

由于函数提升的优先级比变量提升高一些，代码相当于

function a(){;};

var a;

console.log(typeof a);

console.log(typeof a);

因为变量没有赋值，所以函数不会被变量覆盖，两次打印typeof a；a代表的都是函数，所以打印函数

console.log(typeof 'true'); 因为true用单引号引起来的，所以是字符串直接量，打印string

3.

for(i=0, j=0; i<4, j<6; i++, j++){

k = i + j;

}

console.log(k); //10

for(语句1；语句2；语句3){}：

语句1表示初始状态；

语句2表示每次执行时的判断条件；

语句3表示每次执行完后执行的操作，一般为++、--，++表示自加，--表示自减。++、--分为前缀形式和后缀形式，前缀形式先加减1在执行赋值，后缀形式先执行赋值再加减1

{}中写执行语句

for循环的判断语句中出现逗号运算符，表示最后一个条件不满足的时候才退出循环。

i<4, j<6相当于j<6; 所以i=4，j=4，k=4+4=8的时候,i,j都会自加，直到i=5,j=5,k=5+5=10;

下一步i=6,j=6不满足判断语句i<4 || j<6；退出循环。此时k=10, 打印10。

4.

var name = 'laruence';

function echo()

{

console.log(name);

}

function env()

{

var name = 'eve';

echo();

}

env();

这道题涉及到的知识点有全局变量、局部变量和作用域（作用域链）

1. 全局变量：最外层函数定义的变量拥有全局作用域，即对任何内部函数访问。
2. 局部变量：局部作用域一般只在固定的代码片段内可访问到，而对于函数外部是无法访问的。
3. 作用域链：根据在内部函数可以访问外部函数变量的这种机制（即它们在一条作用域链上）。内部函数访问一个变量时，首先在函数内部查找，没有就去它的外部函数查找（可以理解成，儿子没有钱了就问爸爸要，爸爸没有就问爷爷要，不能找兄弟要，因为已经分家了，即不在一条作用域链上）

需要注意的是，函数内部声明变量的时候，一定要使用var命令（即用函数表达式创建函数var f2 = function() {}）。如果不用的话（即用函数声明式创建函数function f1() {}），你实际上声明了一个全局变量！（出现函数提升）

这样来看这道题就比较清晰了!

var name = 'laruence'; 声明了一个全局变量name并且初始化为laruence。然后用函数声明式创建了两个函数（函数不调用就不管），然后调用函数echo(),该函数里有声明了一个局部变量name并且初始化为eve,然后调用函数echo()；该函数执行语句为console.log(name)，这个name到底指哪个呢，我们按照作用域链的方式查找，先看当前函数echo中有没有name，没有去函数外面找（找爸爸，不能找兄弟），刚好有var name = 'laruence'。所以打印laruence

5.

var a = '' + 3; //声明了一个变量a,将等号右边运算的结果赋给a,引号中什么都没有表示空字符串，空字符串+数字：表示字符串的拼接，将数字类型转化成字符串类型，结果为’3’，即a=’3’

var b = 4; //声明了一个变量b,将等号右边运算的结果赋给b,即b初始化为4

console.log(typeof a); //打印a的类型，a=’3’,是字符串类型，打印string

console.log(a + b); //打印a+b的结果，a是字符串类型，b是数字类型，字符串+数字：把数字转化成字符串在拼接即’3’+’4’=34;所以打印34

console.log(a - b); // a是字符串类型，b是数字类型。字符串-数字：把减号左右两边的字符串转化成数字类型在进行减法运算，即3-4=-1；所以打印-1

var foo = "11" + 2 + "1"; //声明了一个变量foo,将等号右边运算的结果赋给foo.+左右有一边是string类型，执行字符串的拼接,把数字2转化成字符串类型’2’,再执行拼接操作’11’+’2’+’1’=’1121’,所以foo初始化为’1121’

console.log(foo); //打印变量foo的值1121

console.log(typeof foo); //因为foo的值’1121’是字符串类型，所以打印string

这道题主要考的是算数运算符(+、-、\*、/、%、++、--)

+运算符：

如果左右两边都是number类型,执行加法运算。如2+3=5;

如果左右有一边或两边都是string类型，执行字符串的拼接。如’3’+3=’33’;

其他运算符在操作非number类型的数据时，都会把数据类型转换成number在运算。

6.

var x=8; //声明一个变量为x并且初始化为一个数据类型的直接量8

var objA = {

x:'good',//对象中有一个属性叫x,x的值为一个字符串直接量good

y:32 //对象中有一个属性叫y,y的值为一个数字直接量32

} //声明一个变量为objA并且初始化为一个对象object类型的字面量（{}表示一个对象），且对象中有两个属性，

function add(x,y){ //声明一个函数叫add,并且设置两个形式参数x,y （函数不调用就不看内容）

console.log(x.y+y);

}

function fn(x,y){ //声明一个函数叫fn,并且设置两个形式参数x,y

x.y=5;

console.log(x.y);

y(x,3);

}

fn(objA,add); //调用函数fn（函数调用：在函数名后面加小括号，如fn()）,并给fn传入两个实参objA和add，函数相 当于

// fn(x,y){

// var x=objA; 在函数内先声明第一个形参x，给第一个形参x初始化为第一个实参objA对象

// var y=add; 然后在函数内再声明第二个形参y，第二个形参y初始化为第二个实参add函数

// x.y=5; 因为变量x代表的是objA对象，所以使用点语法调用对象的属性y,并通过等号给objA的属性y赋新值为数字类型的数据5

// console.log(x.y); 因为变量x代表的是objA对象，所以使用点语法调用对象的属性y,所以这里打印的是objA的属性y,因为上面赋了新值，新值5会把旧值覆盖掉。以后再调用该属性，获取的是最后一次修改的值。 所以打印5

// y(x,3); 因为变量y代表的是add函数,函数通过小括号()调用，并给函数add传入两个实参x即objA和数字3，函数相当于

// add(x,y){

// var x=objA; 在函数内先声明第一个形参x，给第一个形参x初始化为第一个实参objA对象

// var y=3; 然后在函数内再声明第二个形参y，第二个形参y初始化为第二个实参数字3

// console.log(x.y+y); 打印x.y+y的值。因为变量x代表的是objA对象，所以使用点语法调用对象的属性y,所以这里打印的是objA的属性y,因为上面赋了新值，新值5会把旧值覆盖掉。以后再调用该属性，获取的是最后一次修改的值5。所以结果为5+y,因为变量y代表的是第二个实参数字3，5+3=8,所以结果打印8.

// }

// }

console.log(objA); //打印对象objA，会把对象的数据类型Object和所有属性、方法即原型打印出来，由于属性x没有修改过，所以与原来一样为x的值为一个字符串直接量good；因为属性y赋了新值，新值会把旧值覆盖掉。以后再调用该属性，获取的是最后一次修改的值。 y的值为5；对象的原型也是对象，所以原型对象\_\_proto\_\_的值为Object。

//所以结果打印 Object

// x: "good"

// y: 5

// \_\_proto\_\_: Object

对象：

JavaScript 中的所有事物都是对象：字符串、数值、数组、函数... 对象只是带有属性和方法的特殊数据类型，属性是与对象相关的值。对象中属性的存储方式为键值对即key:value，属性与属性之间用逗号给开。除了属性对象中还可以有方法（即函数），方法一般用来描述对象的行为。对象通过点语法来调用属性与方法如：obj.name,obj.fn();也可以用中括号调用属性如：obj[name];当然创建属性与方法也可以用这两种方式如：添加属性age,obj.age=10或者obj["age"]=10

创建对象的方式：

## 通过字面量方式创建

## 方法：将成员信息写到{}中，并且赋值给一个变量。此时的这个变量就是一个对象

## 例如：var person = {name: 'jack',sex: 'girl'};

## 通过构造函数创建

## 方法：方法：var person=new 函数名()，通过该方法创建对象时，会自动执行该构造函数

## 例如： function Person() {

this.name = 'jack';

this.sex = 'girl';

}

var child = new person();

## 通过object方式创建

## 方法：方法：通过object构造器new一个对象，再丰富成员信息

## 例如： var person=new Object();

person.name='jack';

## person.sex='girl';

## 通过object.create方式创建

例如：var p={name:'jack',sex:'girl'};

## var object1=Object.create(p);

7.

function changeObjectProperty(o) { //创建一个函数changeObjectProperty，并且设置一个形参为o // (函数不调用就不管，等到调用的时候再具体分析)

o.siteUrl = "http://www.csser.com/";

o = new Object();

o.siteUrl = "http://www.popcg.com/";

}

var CSSer = new Object(); //使用var声明一个变量CSSer，变量CSSer初始化为一个对象，该对象是使用new关键字后面加构造函数名加小括号，即创建对象的实例。

changeObjectProperty(CSSer); //通过()调用函数changeObjectProperty,并传入实参CSSer这个对象

// changeObjectProperty(o) { 函数相当于这样

// var o=CSSer; 函数体内部首先声明形参o这个变量，再将实参CSSer赋给形参o,现在o代表的就是一个对象了；

// o.siteUrl = "http://www.csser.com/"; 对象通过点语法给o代表的对象CSSer添加一个属性siteUrl，并赋值为一个字符串

// o = new Object(); 用new关键字后面加函数名加小括号，创建一个新的对象，并赋给o这个变量，新值会把旧值覆盖掉。以后再访问该变量，获取的是最后一次修改的值。 即现在o代表的是新对象，不代表CSSer对象了

// o.siteUrl = "http://www.popcg.com/"; 给o代表的这个新的对象通过点语法添加一个属性siteUrl，并赋值为一个字符串

// }

console.log(CSSer.siteUrl); //打印对象CSSer的siteUrl属性的值,因为上面在调用函数的时候给对象CSSer添加了siteUrl为字符串http://www.csser.com/；而后面新创建了一个对象，给新对象也添加了siteUrl属性，但是两个对象所操作的空间不同，对象的名字也不同，所以互不影响。打印 http://www.csser.com/

8.

var num=5;

function func1(){

var num=3;

var age =4;j7hhb

function func2(){

console.log(num);

var num ='ivan';

function func3(){

age =6;

}

func3();

console.log(num)；

console.log(age);

}

func2();

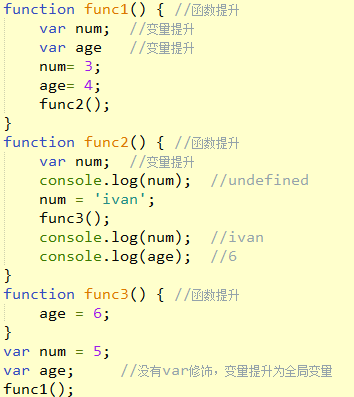
}

func1();

与第2题一样涉及到变量提升与函数提升，通过上面变量与函数提升的知识，我们知道：

1. 没有var 修饰的变量会提升为全局变量
2. 有var 修饰的变量会提升到当前代码块开始部分
3. 使用函数声明式创建的函数，整体函数都会提升到全局
4. 函数提升要比变量提升的优先级要高一些，且不会被变量声明覆盖，但是会在变量赋值之后被变量覆盖。(变量提升和函数提升的位置不影响，我喜欢把函数提升到前面)

所以上面代码相当于：



函数不调用就先不看，所以代码从var num=5;开始执行

var num=5; 使用var关键字声明一个全局变量num，使用等号，将等号右边的数据类型5赋给等号左边的变量num，这样过程称为变量赋值，由于是第一次赋值，又称为初始化

var age因为是没有var修饰的变量提升，所以先不管。因为没有var修饰的变量出现变量提升是代码运行到那一行的时候才会出现，现在先不管。而又var修饰的变量出现变量提升是在执行到当前代码块开始的时候

通过()调用函数func1,现在我们看第一个函数，函数体内部使用var也定义了一个叫num的变量并初始化为3，但是是局部变量，只在当前这个函数中起作用；还使用var定义了一个叫age的变量并初始化为4，也是局部变量，作用域为当前这个函数。因为使用var的关系，所以在进入这个函数的时候就出现了变量提升。

通过()调用函数func2, 现在我们看第二个函数，第一步打印一个叫num的变量，由于下面有一个使用var 修饰的变量，使用出现变量提升到第一行。但是打印的时候num还没有初始化，所以打印结果为undefined。然后给这个函数里的局部变量num赋值为一个字符串类型的数据ivan

通过()调用函数func3, 现在我们看第三个函数，直接给一个没有声明的变量age赋值为6，因为没有var 修饰，所以提升为全局变量。

回到函数func2的后面两句，先打印num，当访问一个变量的时候现在函数里面找，找不到再到同一作用域链的外部函数找。因为这里函数里面有一个局部变量num值为ivan,所以打印ivan;再打印age，因为这个函数里面没有，所以去函数外面找，正好有age这个变量，并且在上面调用函数func3的时候给这个全局变量赋值为6，所以打印6.

9.

var fn1 = 'ivan';

var name ='good';

var fn1 = function(y){

y();

}

function fn1(x){

x(name);

}

function fn2(x){

console.log(x);

console.log(name);

var name = 'hello';

console.log(name);

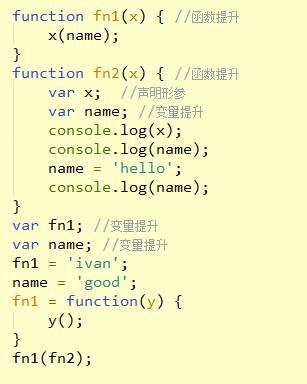
}

fn1(fn2);

与第2题一样涉及到变量提升与函数提升，通过上面变量与函数提升的知识，我们知道：

1. 没有var 修饰的变量会提升为全局变量
2. 有var 修饰的变量会提升到当前代码块开始部分
3. 使用函数声明式创建的函数，整体函数都会提升到全局
4. 函数提升要比变量提升的优先级要高一些，且不会被变量声明覆盖，但是会在变量赋值之后被变量覆盖。(变量提升和函数提升的位置不影响，我习惯把函数提升到前面)

所以上面代码相当于：



代码运行前，先把所有有var修饰的fn1、name变量声明、使用函数声明式创建的函数fn1、fn2提升，然后在逐行分析。函数不调用就先不看。

然后先给fn1赋值为字符串ivan,给name赋值为字符串good;再给fn1赋新值为一个匿名函数。调用fn1并传入一个参数为函数fn2，那么fn1具体是指哪个呢？

由于新值会把旧值覆盖掉。以后再访问该变量，获取的是最后一次修改的值。所以fn1赋新值为一个匿名函数会把fn1初始值ivan覆盖。又因为在变量赋值后，函数会被变量覆盖，所以fn1最后指的是变量fn1，而变量fn1最后赋值的是匿名函数

在匿名函数中传入函数fn2，然后调用实参即调用函数fn2

函数进入时，有var修饰的变量name发生变量提升；然后打印x, 因为fn2函数中定义的一个形参，但是在调用的时候没有传入实参，所以在代码第一行会声明形参但没有初始化，打印结果为undefined。然后打印name,因为变量提升了但没有初始化，打印结果为undefined。然后给name赋值为字符串hello,再打印name，打印结果为hello

10.

var buttons = [{name:'b1'},{name:'b2'},{name:'b3'}];

function bind(){

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

buttons[i].onclick = function() {

console.log(i);

}

}

};

bind();

buttons[0].onclick();//3

buttons[1].onclick();//3

buttons[2].onclick();//3

click事件 ：鼠标点击事件,事件会在对象被点击时发生。

使用var关键字声明了一个变量命名为buttons，该变量空间中存储的是一个数组，长度为3，有三个对象类型的元素，每一个对象中都含有一个命名为name的成员，其属性值分别为b1，b2，b3

function bind(){} 使用函数声明式创建函数，会发生函数提升，函数不调用就不看

bind(); 函数名+()调用函数，执行函数bind里的代码块。

使用for r循环遍历整个数组，给每一个元素都添加了click点击事件，点击后执行函数，打印i的值。不点击或者不调用这个函数

For循环结束后，依次调用数组中每个元素点击事件的函数，执行该函数代码块，打印i的值。

因为前面已经调用了bind函数，for循环里遍历完整个数组也已经执行完毕了，所以最后i的值等于数组的长度3，而在for循环中只是给每一个元素添加了点击事件这个函数，并没有调用函数，函数不调用就不访问i的值，所以最后三次调用函数的时候，再去访问i的值，就变成for循环结束后i的值3，所以打印的结果都是3。

11.

function fun(n,o) {

console.log(o)

return {

fun:function(m){

return fun(m,n);

}

};

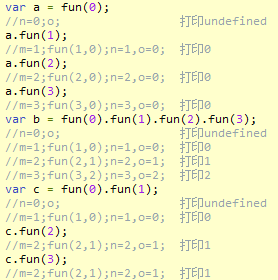
}

var a = fun(0); a.fun(1); a.fun(2); a.fun(3);

var b = fun(0).fun(1).fun(2).fun(3);

var c = fun(0).fun(1); c.fun(2); c.fun(3);

打印结果如下：



先分析一下这个函数fun的代码结构，有两个形参n、o打印o的值，返回值为一个对象，该对象有一个属性fun，该属性的值是一个匿名函数，有一个形参m,这个匿名函数的返回值在调用最外层的函数fun，并且传入了两个实参m、n；

1. Var a=fun(0),调用最外层函数，并且传入一个参数0，进入函数代码块，首先用var 声明两个形参，此时第一个参数n=0,第二个参数o没有赋值，所以打印unfined
2. fun(1), 因为外层函数的返回值是一个对象，这里通过对象调用方法，并传入一个实参1，此时m=1,然后返回值继续调用外层函数fun,并将m、n的值1，0作为形参传入即fun(1,0),此时n=1,o=0,打印o的值，所以打印0
3. a.fun(2), 因为外层函数的返回值是一个对象，这里通过对象调用方法，并传入一个实参2，此时m=2,然后返回值继续调用外层函数fun,并将m、n的值2，0作为形参传入即fun(2,0),此时n=2,o=0,打印o的值，所以打印0
4. a.fun(3), 因为外层函数的返回值是一个对象，这里通过对象调用方法，并传入一个实参2，此时m=2,然后返回值继续调用外层函数fun,并将m、n的值2，0作为形参传入即fun(2,0),此时n=2,o=0,打印o的值，所以打印0
5. var b = fun(0).fun(1).fun(2).fun(3);使用var声明一个变量为b,并将等号右边的调用的结果赋给b
6. fun(0), 调用最外层函数，并且传入一个参数0，进入函数代码块，首先用var 声明两个形参，此时第一个参数n=0,第二个参数o没有赋值，所以打印unfined
7. .fun(1), 因为fun(0)的返回值是一个对象，这里通过对象调用方法，并传入一个实参1，此时m=1,然后返回值继续调用外层函数fun,并将m、n的值1，0作为形参传入即fun(1,0),此时n=1,o=0,打印o的值，所以打印0
8. .fun(2), 因为fun(1)的返回值是一个对象，这里通过对象调用方法，并传入一个实参2，此时m=2,然后返回值继续调用外层函数fun,并将m、n的值2，1作为形参传入即fun(2,1),此时n=2,o=1,打印o的值，所以打印1
9. .fun(3), 因为fun(2)的返回值是一个对象，这里通过对象调用方法，并传入一个实参3，此时m=3,然后返回值继续调用外层函数fun,并将m、n的值3，2作为形参传入即fun(3,2),此时n=3,o=2,打印o的值，所以打印2
10. var c = fun(0).fun(1);使用var声明一个变量为c,并将等号右边的调用的结果赋给c
11. fun(0), 调用最外层函数，并且传入一个参数0，进入函数代码块，首先用var 声明两个形参，此时第一个参数n=0,第二个参数o没有赋值，所以打印unfined
12. .fun(1), 因为fun(0)的返回值是一个对象，这里通过对象调用方法，并传入一个实参1，此时m=1,然后返回值继续调用外层函数fun,并将m、n的值1，0作为形参传入即fun(1,0),此时n=1,o=0,打印o的值，所以打印0
13. .fun(2), 因为fun(0).fun(1)的返回值是一个对象，这里通过对象调用方法，并传入一个实参2，此时m=2,然后返回值继续调用外层函数fun,并将m、n的值2，1作为形参传入即fun(2,1),此时n=2,o=1,打印o的值，所以打印1
14. .fun(3), 因为fun(0).fun(1)的返回值是一个对象，这里通过对象调用方法，并传入一个实参3，此时m=3,然后返回值继续调用外层函数fun,并将m、n的值3，1作为形参传入即fun(3,2),此时n=3,o=1,打印o的值，所以打印1

12.

var name = 'lili';

var obj = {

name: 'liming',

prop: {

name: 'ivan',

getname: function() {

return this.name;

}

}

};

console.log(obj.prop.getname());//ivan

var test = obj.prop.getname;

console.log(test()); //lili

这道题主要考的是this关键字的指代

我的理解是,找this的指代，先找离this最近的function,是谁调用的这个function，this就是谁，如果没有，就指window.

在箭头函数()=>{}中出现的this，都指向window

现在再来分析

var name = 'lili';使用var关键字创建了一个全局变量name，并赋值为一个字符串lili

var obj = {}，使用var关键字创建了一个全局变量obj, 赋值为一个对象字面量；

obj这个对象里面有一个属性叫name，name的值为一个字符串liming,另一个属性叫prop,prop的值为一个对象；prop这个对象中也有一个叫name的属性，name的值为ivan,还有一个叫getname的属性，属性值为一个函数，这个函数的代码块执行的代码为返回this.name;

console.log(obj.prop.getname());打印obj.prop.getname()即打印this.name;的值，按照上面this关键字的理解，先找里this最近的function，是getname这个属性值，再看调用这个函数的对象，是prop对象，所以打印的是prop.name，结果为ivan.

var test = obj.prop.getname; 使用var关键字创建了一个变量test，并将obj.prop.getname赋为test变量，即现在test变量代表的是getname这个函数

console.log(test()); 打印调用test()函数的值，发现test外面没有function，所以this指代window，window.name访问全局变量name的值，结果为lili

1. 问答题

14.谈谈你对JavaScript中原型对象的理解.(理解,用法,特点,优点)

1. 简谈我对原型对象的理解：

原型对象是指和对象相关联的另外一个对象; 函数对象都有一个prototype 属性，这个属性指向函数的原型对象；每个对象会从原型对象继承属性和方法；实体对象的原型对象是构造函数，构造函数的原型对象是Object内置对象，Object往上就是null了；上面一层层的原型关系就构成了原型链，对象会继承原型链上允许继承的全部属性和方法；对象在调用属性或方法时，首先在自己内部查找对应的属性和方法，找到就结束；如果自己内部找不到，就在原型对象上面找，找到就结束；如果原型对象上找不到，就继续沿着原型的原型往上找，直到找到为止；如果整个原型链上都找不到需要的属性则返回undefined；如果整个原型链上找不到需要的方法，则提示函数不存在，代码报错；

1. prototype与\_\_proto\_\_：

JavaScript中万物皆对象，但对象之间也是有区别的。分为函数对象和普通对象。

函数对象可以创建普通对象，普通对象没法创建函数对象

在 JavaScript 中，每当定义一个对象（函数也是对象）时候，对象中都会包含一些预定义的属性。其中每个函数对象都有一个prototype 属性，这个属性指向函数的原型对象。所有的对象obj都具有\_\_**proto\_\_**属性(null和undefined除外),而且指向创造obj对象的函数对象的prototype属性，但只有函数对象才有 prototype 属性；普通对象是没有的；

prototype与*\_*proto*\_*的区别：prototype表明实例自身的属性，*\_proto\_*表明实例的原型是谁

1. 原型链：

JavaScript中，每个对象都会在内部生成一个\_\_proto\_\_ 属性，当我们访问一个对象属性时，如果这个对象不存在就回去proto 指向的对象里面找，一层一层找下去，这就是JavaScript原型链的概念。

new Demo().\_\_proto\_\_ == Demo.prototype ==> Demo.prototype.\_\_proto\_\_ ==> Object.prototype ==> Object.prototype.\_\_proto\_\_ ==> null

4．通过函数对象的prototype属性，给这个函数对象添加方法，例如：

Demo. prototype.fn=function(){}

只要使用Demo构造函数创建的对象即new Demo()，都可以直接通过点语法调用使用上面的方法。

prototype属性指向函数的原型对象,主要是为了实现继承;

所有实例对象需要共享的属性和方法，都放在原型对象里面；那些不需要共享的属性和方法，就放在构造函数里面。

如果我们在穿件一下同类型，或者有继承关系的对象时，使用原型对象会减少很多代码。

5. 构造器constructor

constructor 属性返回对创建此对象的函数对象的引用。

请看以下例子:

function a(){};

函数a是由Function创造出来,那么它的constructor指向的Function,a.prototype是由new a()方式创造出来,那么a.prototype.constructor理应指向a

那么a.prototype.\_\_proto\_\_.constructor指向谁?

1, a.prototype指向a的一个实例,而且属于普通对象

2, \_\_proto\_\_定义为:指向创造obj对象的函数对象的prototype属性,所以看下谁创造了a.prototype,因为a.prototype是普通对象,类型为object,那么是Object创造了它,

3, 那么显而易见a.prototype.\_\_proto\_\_指向了Object.prototype

4, 那么题目简化为Object.prototype.constructor指向谁

5, 继续分解题目,Object.prototype为基本对象,那么就是Object创造了它,那么它的constructor就指向了Object

由原型链的知识我们知道，尽头为 Object.prototype.\_\_proto\_\_===null