# 100道前端笔面试题及答案

资料来自Duing（ID：duyi-duing）每日一题，作者：Duing

收集100天汇成此文

**1.【关于函数参数】下面代码的执行结果是什么**

```
function getInfo(member, year) {
    member.name = "Alice";
    year = "2020"
}
var person = { name: "Tom" }
var brithYear = "2010";
getInfo(person, brithYear);
console.log(person, brithYear);
```

答案：`{name:Alice},2010`

**2.【Event Loop】下面代码的执行结果是什么**

```
var p = new Promise(resolve => {
    console.log(4);
    resolve(5);
});
function func1() {
    console.log(1)
}
function func2() {
    setTimeout(() => {
        console.log(2)
    });
    func1();
    console.log(3);
    p.then(resolved => {
        console.log(resolved)
    }).then(() => {
        console.log(6)
    });
}
func2();
```

答案：`4 1 3 5 6 2`

解析：事件循环机制，主程序执行完，再执行微队列，再执行宏队列；Promise then会先进入微队列，setTimeout等进入宏队列

**3.下面的代码的执行结果是什么**

```
function Car(){
    this.name = "BMW"
    return {
        name:"maserati"
    }
}
var car = new Car();
console.log(car.name);
```

答案：`maserati`
解析：new构造函数产生对象时，有显示返回一个对象的情况，this就是该对象

**4.模拟实现placeholder作用**

```
<input type="text" name="user" value="请输入用户名" style="color:#999"
onfocus="if(this.value=='请输入用户名'){this.value =
'';this.style.color='#424242'}"
onblur="if(this.value == ''){this.value = '请输入用户
名';this.style.color='#999'}" >
```

**5.下面代码的执行结果是什么**

```
var a = 1;
var b = 2[a,b] = [b,a]
console.log(a,b)
```

答案：a是1，b是[undefined , 1]

解析：预编译和包装类对象的结果

**6.下面代码最后打印arr是什么**

```
var arr = [1,2,3,4,5];
arr.length = 1;
```

答案：[1]
解析：length会影响数组

**7.下面代码执行结果是什么**

```
var arr = [1+1,1*2,1/2];
console.log(arr);
```

答案：[2,2,0.5]

解析：数组保存的时候，每一位的表达式都会先运算

**8.下面代码的执行结果是**

```
var one = (false || {} || null);
var two = (null || false || "");
var three = ([] || 0 || true);
console.log(one,two,three);
```

答案：{},"",[]

解析：本题考查||运算符的用法。其实记住一条规律：||运算符返回的是能确认最终结果的值！

**9.下面代码执行结果是？**

```
var h5course = false;
var result = h5course / 0;
if (result) {
    console.log(result * 2 + '2' + 4)
} else {
    console.log(!result * 2 + '2' + '4')
}
```

答案："224"

解析：0/0,null/0,undefined/0,false/0都为NaN，其他正数/0为infinity，其他负数除以0位-infinity。!NAN为true，true*2为2

**10.下面代码执行结果是什么？**

```
var a = 0, b=0;
function A(a){
    A = function(b){
        alert(a + b++)
    }
    alert(a++)
}
A(1)
A(2)
```

答案：1，4

解析：深入预编译和闭包就会得出结果

**11.下面代码的执行结果是？**

```
var str = new Array(5).toString()
console.log(str);
```

答案：",,,,"

解析：new Array(n) 返回的是以length为n、每一位为空的数组。

所以正因为new Array有这样那样的问题，一般推荐使用字面量如：var arr = [5]

**12.下面代码的执行结果是什么？**

```
console.log(123..toString());
```

答案："123"

解析：包装类的原因，js会将123.通过new Number(123.)打包成包装类对象。调用的是Number.prototype.toString，所以new Number(123.).toString === Number.prototype.toString

**13.下面代码的执行结果是？**

```
var x = 1;
var y = 2;
function show() {
    var x = 3;
    return {
        x: x,
        fun: function (a, b) {
            x = a + b;
        }
    }
}
var obj = show();
obj.fun(x, y);
console.log(obj.x)
console.log(x)
```

答案：3，1

解析：考查知识点闭包和预编译

**14. 下面的代码执行结果是？**

```
var a = (true + false) > 2 + true;
console.log(a);
```

答案：false

解析：涉及知识点运算符的计算顺序和类型转换；

有()先算()，存在隐式类型转换true + false 变成1+0，

算数运算优先比较运算符，所以先算2+1

最后比较是：1>3 得出false

**15.说说运算符的优先级**

【小括号()】>【逻辑非!】>【算数*/%】>【算数+-】>【关系比较><==】>【逻辑与&&、逻辑或||】>【三目?:】>【赋值=】

大致是这样，具体可以查看这篇文档，里面有个表：https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/Operator_Precedence

**16.下面代码执行结果是？**

```
var num = 3;
console.log(num.toString(2));
console.log(num.toFixed(2));
```

答案：11、3.00

解析：toString方法传入一个参数时，表示把一个数当成10进制转成对应的进制的数，这里3被当成10进制，变成2进制就是11；而toFixed方法把目标数变成保留指定位数的小数，注意会四舍五入！这里保留两位小数，即变成3.00

**17.下面代码的执行结果是什么？**

```
function a(x){
    return x*2
}
var a;
console.log(a);
```

答案：function a(x){return x*2}

**18.下面代码的运行结果是什么？**

```
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(2);
c.fun(2);
c.fun(3);
```

答案：

undefined 0 0 0

undefined 0 1 2

undefined 0 2 2

**19.下面代码的执行结果是？**

```
var a = []+[]+"alice".split("");
console.log(a);
```

答案："a,l,i,c,e"

解析：[]+[]为空字符串'',空字符串加数组，等于把数组变成字符串的形式，相当于数组调用了toString

**20.有两个变量a和b，其值为number类型且非NaN，不借助其他变量，完成a与b的交换**

答案：方式有很多种，下面是其中给一种

var a = 101;

var b = 102;

a = a + b;

b = a - b;

a = a - b;

console.log(a,b)

**21.下面代码的执行结果是？**

```
function Point(x, y) {
    this.x = x;
    this.y = y;
    this.moveTo = function (x, y) {
        this.x = x;
        this.y = y;
        console.log(this.x, this.y)
    }
}
var p1 = new Point(0, 0);
var p2 = { x: 0, y: 0 };
p1.moveTo(1, 1);
p1.moveTo.apply(p2, [10, 10])
```

答案：1,1 和 10,10

解析：this指向有两点，(1) 函数谁调用里面this指向谁，(2) apply call bind可改变函数执行时的this指向

**22.下面代码的执行结果是什么？**

```
var globalVar = 0;
(function (outerArg) {
    var outerVar = 456;
    (function (innerArg) {
        var innerVar = 10;
        console.log(globalVar, outerArg, outerVar, innerArg, innerVar);
    }(789))

}(123))
```

答案：0，123，456，789，10
解析：立即执行函数与普通函数执行没有区别

**23.下面代码执行结果是什么？**

```
console.log(1 + undefined);//NaN
console.log(1 + null);//1
console.log(null + undefined);//NaN
console.log(true + false);//1
console.log(1 + '2');//12
console.log(2 + []);//2
console.log(2 + { a: 1 });//2[object Object]
console.log([] + {});//[object Object]
console.log({} + []);//[object Object]
console.log(3 + {});//3[object Object]
console.log({} + 3);//[object Object]3
```

答案：如上

解析：知识点加号运算符存在隐式类型转换，其中加对象时，相当于数值+"[object Object]",充当字符串连接符

**24.下面代码的执行结果是什么？为什么**

```
var a = {},b={key:"b"},c={key:'c'};
a[b] = 123;
a[c] = 456;
console.log(a[b]);
```

答案：456
解析：在设置对象的属性时，如果属性传入是一个对象，js会隐式的讲变量变成对象对应的[object Object]
所以：a[b]和a[c]其实都相当于a["[object Object]"],所以他们操作的都是a对象的同一个属性，因此结果为456

**25.下面代码的执行结果是什么？**

```
for (var i = 0; i < 5; i++) {
    setTimeout(function () {
        console.log(i);
    }, i * 1000);
}
```

答案：5 5 5 5 5
解析:异步setTimeout是放在宏任务中，等执行完for循环之后再放到任务队列中执行
可以用let或者立即执行函数解决
方法一：

```
for(let i = 0;i<5;i++){
   setTimeout(function(){
      cosnole.log(i);
   },i1000)
}
```

*方法二：*

```
for (var i = 0; i < 5; i++) {
   (function(i){
      setTimeout(() => {
         console.log(i)//0 1 2 3 4
      }, i100);
   }(i))
}
```

**26.下面代码执行结果是？**

```
var user = "Alice"
function changeUser(){
    user  = "Tom";
    return;
    function user(){
        console.log('user function')
    }
}
changeUser();
console.log(user);
```

答案：Alice

解析：原因很简答，在执行changeUser函数的时候，预编译将里头函数user提升了，后面改变user = "Tom"，改变的是changeUser函数里面的局部变量。

**27.下面代码执行的结果是什么？**

```
(function(){
    var user = author = "Alice";
}());
console.log(author);
console.log(user);
```

答案：Alice、Uncaught ReferenceError: user is not defined

解析：全局里面没有user，所以访问时会报错。而author是暗示全局变量，是有值的。

**28.下面代码的执行结果是？**

```
console.log(0 || 1);
console.log(1 || 2);
console.log(0 && 1);
console.log(1 && 2);
```

答案是：1 1 0 2

解析：||或运算，前面为真，返回前面的，前面为false，返回后面的

&&与运算符 前面为假，返回前面的，前面为真，返回后面的

**29.下面代码的执行结果是什么？**

```
console.log(1 + '2' + '3');
console.log(1 + +'2' + '3');
console.log(1 + -'1' + '2');
console.log(+'1' + '2' + '3');
console.log('A' - 'B' + "2");
console.log("A" - "B" + 2);
```

答案：'123' '33' '02' '123' 'NaN2' NaN
解析：（1）任何数据类型+字符串都为字符串；（2）一元正负可以将其变成数值型；
（3）非数字类型或者非数字型字符串字符串相减，值为NaN

**30.下面代码的执行结果是？**

```
(function(){
    try{
        throw new Error
    }catch(x){
        console.log(x)//Error
        var x = 1;
        var y = 2;
        console.log(x)//1
    }
    console.log(x);//undefined
    console.log(y);//2
})();
```

答案: 1 undefined 2

解析：最开始执行匿名立即执行函数参数的作用域中，由于预编译将变量x和y提升了值为undefined

当执行catch函数时，里面有自己的形参变量x，这里的值是错误信息：Error,后面赋值为1；所以里面catch中先打印Error后打印1

y被赋值为2;

所以外面打印的值：x为undefined，y为改变的值2

**31.下面代码的执行结果是？**

```
var a = 10;
function fn() {
    console.log(this.a)
}
var obj = {
    a: 5,
    method: function (fn) {
        fn();//自己执行的，所有this指向window，打印出10
        arguments[0]();
    }
}
obj.method(fn, 1)
```

答案：10 undefined

解析：函数自己执行里面this指向window，

函数传入另外一个函数作为参数，通过arguments[0]的形式调用，相当于arguments.0(),arguments.0就代表该函数，所以里面this指向arguments。

再比如：

```
var length = 10;
function fn() {
    console.log(this.length)
}
var obj = {
    length: 5,
    method: function (fn) {
        fn();//自己执行的，所有this指向window，打印出10
        arguments[0]();//相当于arguments.0(),这里arguments的length为2，并且arguments.0中的0就是传入的fn，所以执行函数，相当于arguments调用。打印出2
    }
}
obj.method(fn, 1)
```

答案是：10 2

**32.下面代码的执行结果**

```
var fn = function a(){
    a = 1;
    console.log(typeof a)
}
fn();//function
console.log(typeof a)//undefined
```

答案：function undefined

解析：关于有名函数表达式需要注意2点：

（1）有名函数表达式的函数名（a）在函数体内还代表一个函数

（2）在函数体外部是会自动忽略函数的名称，即函数体外面是不能访问到（a）

（3）在函数体内部，函数名（a）不能再被修改

**33.说说this和$(this)在jQuery中有什么不同？**

答案：this是js关键字中的一个，表示当前DOM元素；而$(this)返回的一个jQuery对象，可以调用jQuery中封装的方法

**34.下面代码的执行结果是什么？**

```
function fn1() {
    return {
        str: 'hello'
    }
}
function fn2() {
    return
    {
        str: 'hello'
    }
}
console.log(fn1());
console.log(fn2());
```

答案：{str: "hello"}  undefined
解析：js语言特性，如果语句表达是完整的，换行后默认会在语句后面加入';'

**35.如何获取函数实参和形参的个数？**

答案：

```
function fn(a,b){
    //1. 获取实参个数的方式：
    console.log(arguments.length);//5
    //2. 获取形参个数的方式：
    console.log(arguments.callee.length)//2
    arguments.callee===fn,所以也可以用fn.length获取形参个数
    console.log(fn.length)//2
}
fn(1,2,3,4,5);
```

**36.下面代码执行结果是？**

```
function Foo() {
    getName = function () {
        alert(1)
    }
    return this;
}
Foo.getName = function () {
    alert(2)
```

```
}
Foo.prototype.getName = function () {
    alert(3)
}
var getName = function () {
    alert(4)
}
function getName() {
    alert(5)
}
// 分析此时GO里有的东西：{
// getName: function getName() {alert(4)},
// Foo: function Foo(){getName = function () {alert(1)}  return this; }
// }
// 并且Foo的静态方法中有个Foo.getName = function () {alert(2)}
// 原型上还有个方法getName = function () {alert(3)}
// 下面的结果是？
Foo.getName();//2
getName()//4
Foo().getName()//Foo(),返回this为window，并且GO中的getName变成alert(1),所以这里是：1
getName()//1
new Foo.getName()//将Foo.getName函数作为构造函数执行，所以弹出：2
new Foo().getName();//调用的是原型上的方法 所以打印： 3
new new Foo().getName()//这里是js运算符的优先级问题，相当于new((new Foo()).getName)();答案输出：3
```

答案：如上

**37.下面代码的执行结果是？**

```
var user = 'alice';
function firstFn(){
    console.log(user);
}
function secondFn(){
    var user = 'tom';
    firstFn();
}
firstFn();
```

答案:'alice'

解析：变量查找是顺着作用域链查找的，例子中，secondFn的作用域没有加到firstFn，

需要注意的是，看是否有加入到某个作用域链中，主要是看该函数的定义位置，而不是使用位置。

**38.如何阻止a标签的默认跳转事件？**

答案：

```
let tagA = document.getElementsByTagName('a')[0]
taga.onclick = function (e) {
    //w3c
    // e.preventDefault();
    //IE独有的
    // e.returnValue = false;
    //都可以用
    return false;
}
```

**39.如何让一个元素不显示**

答案：

```
let odiv = document.getElementsByClassName("div")[0]
// 方法1
odiv.style.display = 'none'
// 方法2
odiv.style.visibility = "hidden"
// 方法3
odiv.styly.opacity = 0
```

**40.下面代码的执行结果是？**

```
let arr = [1, 2, 3, 4]
arr[10] = 10
arr[20] = 20
console.log(arr[15])
```

答案：undefined

解析：可以溢出读和写，溢出读时值为undefined，溢出写时，中间空位值为undefined

**41.下面代码的执行结果是什么？**

```
console.log(typeof undefined == typeof NULL)
```

答案：true

解析：js大小写敏感,NULL不是null,如果是typeof null返回的是object类型

**42.下面代码的执行结果是**

```
function fn(xx){
    this.x = xx
    return this
}
var x = fn(5)
var y = fn(10)
console.log(x.x)
console.log(y.x)
```

答案：undefined 10

解析：在fn(5)执行完，window.x === window;在fn(10)执行的时候，window.x = 10，window.y === window;

因此x.x其实就是new Number(10).x,所以打印出undefined，y.x其实就是window.x ，所以打印出10

**43.下面代码的执行结果什么？**

```
var x = 5;
function fn(x){
    x = 8
    console.log(x)
}
fn()
console.log(x)
```

答案：8、5

解析：fn打印的x是局部变量x，所以先打印8，外面打印全局window的x，所以打印出5

**44.下面代码的执行结果是**

```
(function(x){
    return (function(y){
        console.log(x)
    })(2);
})(1)
```

答案：1

解析：第一个立即执行函数执行的时候，ao上有个x值为1，然后return第二个立即执行函数执行，这里还能访问到x，所以打印出1

**45.下面代码的输出结果是什么？**

```
var result = (function f(n) {
    return (n > 1) ? n * f(n - 1) : n
})(10)
console.log(result)
```

答案：10的阶层，即3628800

**46.下面代码的执行结果是什么，为什么？**

```
function fn(){
    var a = b = {}
    a.name = 'alice'
    b.age = 10
    console.log(b.name,a.age)
}
console.log(typeof a)
console.log(typeof b)
```

答案：alice 10 undefined object

解析：引用值的赋值是地址，地址指向一个对象。暗示全局变量属于window所有

**47.下面代码的执行结果是？**

```
console.log(+true)
console.log(!'str')
```

答案：1，false

解析：一元正负默认隐式调用Number()；！默认取反后调用Boolean(),即!'str'相当于Boolean('str')

**48.下面代码的执行结果是？**

```
function Person(firstName,lastName){
    this.firstName = firstName
    this.lastName = lastName
}
let p1 = new Person('Lewis','Carroll')
let p2 =  Person('Lewis','Carroll')
console.log(p1)
console.log(p2)
```

答案：Person{fristName: 'Lewis', lastName: 'Carroll'}, undefined

**49.下面代码的执行结果是什么？**

```
function show(){
    "use strict";
    // name = "alice";
    age = 12;
    // console.log(`${name}今年${age}岁了`)
}
show()
```

答案：严格模式下，变量必须声明后再赋值，否则ReferenceError

解析：经过测试发现，如果严格模式下变量没声明就赋值为字符串，居然不报错。暂时待进一步验证

**50.下面代码的执行结果是？**

```
(function(){
    console.log(1)
    setTimeout(() => {
        console.log(2)
    }, 1000);
    setTimeout(() => {
        console.log(3)
    }, 0);
    console.log(4)
})()
```

答案：1 4 3 2

解析：考查的是执行队列的问题

**51.下面代码，当点击页面button时，event.target打印的是什么？**

```
<div onclick="console.log('first div',event.target)">
    <div onclick="console.log('second div')">
        <button onclick="console.log('button')">click me!</button>
    </div>
</div>
```

答案：event.target打印的是--> click me!

解析：点击tutton时，由于事件冒泡会传递到父级元素触发相同事件。而event.target记录的是事件的目标源头

**52.下面的代码执行结果是什么？**

```
function checkAge(data) {
    if (data === { age: 18 }) {
        console.log('我今年18岁了')
    } else if (data == { age: 18 }) {
        console.log('我今年成年了')
    } else {
        console.log('emmm...')
    }
}
checkAge({ age: 18 })
```

答案：emmm...

解析：引用值对比的是地址

**53.下面代码的执行结果是？**

```
let count = 0
console.log(count++)
console.log(++count)
console.log(count)
```

答案：0 2 2

解析：考查++运算符，如果++在后，先执行语句再+1；如果++在前面，先将数进行+1 然后在执行语句

**54.如何判断一个变量是数组**

```
let arr = []
//方法1
arr.contructor === Array
//方法2
arr instanceof Array
//方法3
Array.isArray(arr)
```

**55.数组中的push pop unshift shift分别有什么用途？**

push方法：给数组末尾添加一位或多位，返回变化后的数组长度

pop方法 ：给数组末尾移除一位，返回移除的元素

unshift ：给数组前面添加一位或者多位，返回变化后的数组长度

shift ：移除数组最前位，返回移除的元素

注意：以上方法都会改变原始数组

**56.下面代码的执行结果是？**

```
(function(){
    var x = y = 12
})()
alert(y)
```

答案：12

解析:暗示全局变量,即变量未经声明就赋值,该变量属于window

**57.下面代码的执行结果是什么？**

```
var x = 'global'
function fn1(){
    console.log(x)
}
function fn2(){
    var x = 'fn2'
    fn1()
}
fn2();
```

答案：global

解析：作用域是静态的，不取决于函数执行位置，而是取决于函数定义所在位置。

**58.下面代码的执行结果是？**

```
var a = 1
var b = a
b++
console.log(a, b)
```

答案：1，2
解析：原始值赋值是值的拷贝

**59.下面代码的执行结果是？**

```
var a = [1,2,3,4]
var b = a;
a.push(5)
console.log(a,b)
```

答案：[1,2,3,4,5] [1,2,3,4,5]
解析：引用值的赋值是地址的拷贝，指向的是同一个空间

**60.下面代码的执行结果是？**

```
var x = 5;
function F(){
    x = 12;
    console.log(x);
    console.log(this.x);
    var x;
    console.log(x)
}
new F()
```

答案：12 undefined 12

解析：考点一 new关键字；考点二 预编译

**61.下面代码的执行结果是什么？**

```
var arr = [1,2,3,4]
delete arr[1]
console.log(arr.length)
delete arr[3]
console.log(arr.length)
```

答案：4 4
解析：此处delete会移除arr数组中的第1位数据，虽然删除后该位置为empty，但数组的length不会变

**62.下面代码的执行结果是什么？**

```
var res = (12).toFixed(2)
console.log(res)
```

答案：'12.00'
解析：涉及到数字和包装类知识。如果是12.toFixed(2)就报错，相当于12.xxx！

**63.下面代码的执行结果是什么？**

```
console.log(12 + '5')
console.log(12 - '5')
```

答案：125  7

解析：+号运算符中有一个数是字符串时，则加号作用是字符串连接。-号运算符会将两边通过Number()转成数字再进行减法操作

**64.下面代码的执行结果是什么？**

```
var a = new Array(3);
var b = [undefined,undefined,undefined];
var c = [null,null,null];
var d = [false,false,false];
var e = [0,0,0];
console.log(a.join('-'));
console.log(b.join('-'));
console.log(c.join('-'));
console.log(d.join('-'));
console.log(e.join('-'));
```

答案：

--

--

--

false-false-false

0-0-0

解析：数组的join方法会将数组成员安装指定字符连接；特别的，如果join中数组成员是null或者undefined就会当成空串

**65.如何获取一个大于等于0且小于等于9的随机整数？**

```
console.log(Math.floor(Math.random()*10))
```

**66.下面代码的执行结果是什么？**

```
function sum(a, b) {
    return a + b;
}
console.log(sum(1, '2'));
```

答案："12"

解析：字符串拼接

**67.哪些对象没有原型？**

答案：（1）基础的顶层对象；（2）通过Object.create(null)创建的对象

**68.下面代码的执行结果是什么？**

```
var obj = {
  a: 1,
  b: 2,
  a: 3
}
console.log(obj)
```

答案：{a:3,b:2}

解析：非严格模式对象里头变量可重复，但会后面的会覆盖前面的

**69.下面HTML代码的执行结果是什么？**

```
<div onclick="console.log('div')">
    <button onclick="console.log('btn')">click me!!</button>
</div>
```

答案：btn div

解析：结构上非视觉上嵌套的元素存在事件冒泡功能，即从子元素冒泡向父元素

**70.下面代码的执行结果是？**

```
console.log(typeof typeof 12)
```

答案：'string'

解析：任意类型经过第一次typeof之后返回类型都是字符串

**71.下面代码的执行结果是什么？**

```
var arr = [1, 2, 3];
arr[10] = 4;
console.log(arr);
```

答案：[1, 2, 3, empty * 7, 4]

解析：数组可以溢出写，中间没有内容部分为空（empty），读取时值为undefined

**72.下面代码的执行结果是什么？**

```
String.prototype.showInfo = ()=>{
    return 'This is showInfo!'
}
let str = 'aaa';
let res = str.showInfo();
console.log(res);
```

答案：'This is showInfo!'

解析：包装类和原型

**73.下面代码的执行结果是什么？**

```
console.log(!!undefined)
console.log(!!null)
console.log(!!'')
console.log(!!0)
console.log(!!1)
```

答案：false false false false true
解析：!!运算符用于将值变成布尔值

**74.下面代码的执行结果是什么？**

```
console.log(1>2>3);
console.log(3>2>1);
console.log(1<2<3);
console.log(3<2<1);
```

答案：false false true true
解析：比较运算符，按顺序从左往右比较，将前面的结果跟后一个比较，有数值会优先转成数字

**75.下面代码的执行结果是什么？**

```
let  arr = [1,2,3];
arr.unshift('a');
arr.push('b');
let newArrr = [4,...arr,5];
console.log(newArrr);
```

答案：[4,'a',1,2,3,'b',5]
解析：考查数组的基本使用，以及扩展运算符的基本使用

**76.下面代码的执行结果是什么？**

```
console.log('12'/0);
console.log(0/0);
console.log(true == 1);
console.log(NaN === NaN);
```

答案：infinity NaN true false
解析：考查运算符

77.下面代码的执行结果是什么？

```
let a = 1;
a = (++a, a++);
console.log(a);
a = (a++, ++a);
console.log(a);
```

答案：2 4

解析：考查++运算符和逗号运算符。

++在前就先++在执行该语句，++在后就先执行完该语句在++

逗号运算符，会返回逗号后面的结果，当然前面的语句也会执行

**78.js中定义函数的几种方式：**

(1) 函数声明

function fn1(){}

(2)函数表达式

let fn2 = function(){}

(3)通过Function构造函数

let fn3 = new Function('a','b','return a+ b')

Function最后一个参数表示函数体，其余参数作为形参

**79.下面代码的执行结果是什么？**

```
function Person(){}
console.log(Person.prototype === Function);
console.log(Person.prototype === Function.prototype);
console.log(Person.__proto__ === Function.prototype);
console.log(new Person().__proto__ === Person.prototype);
```

答案：false false true true

解析：关于原型。原型是function对象的属性，它定义了构造函数构造出对象的共有祖先

**80.下面代码的执行结果是什么？**

```
function f(){
    f = 12;
    console.log(f);
}
console.log(f);
f();
console.log(f);
```

答案：f(){}  12  12
解析：暗示全局变量

**81.下面代码的执行结果是什么？**

```
var fn = (function(temp){
    arguments[0] = 'Hello';
    return function(){
        console.log(temp,arguments[0])
    }
})('Mike')
fn('Alice');
```

答案：Hello,Alice
解析：考查立即执行函数，以及形参与arguments相互映射

**82.下面代码的执行结果是什么?**

```
if(function fn(){}){
    console.log(typeof fn)
}
console.log(typeof fn)
```

答案：'undefined' 'undefined'

解析：在if语句的小括号中，js会自动将里面的东西通过Boolean()变成对应的布尔值，所以里头函数声明并不能再其他地方使用该函数

**83.下面代码的执行结果是什么？**

```
console.log(1..toString())
console.log(NaN.toString())
console.log(typeof temp)
console.log(1 ++)
```

答案：前三个分别是：'1' 'NaN' 'undefined'；

但最后一个语法有误：Uncaught SyntaxError: Invalid left-hand side expression in postfix operation

解析：由于js执行的时候会先扫一遍看有没有语法错误，然后在一行一行的执行js语句。如果语法有误时，并不会执行其他语句。所以这里会出现SyntaxError

**84.说说call和apply函数**

（1）call和apply都是Function.prototype上定义的函数，任何一个函数都有call和apply方法

（2）当函数调用和call或apply方法时，call和apply会改变函数里头的this指向，然后执行该函数

（3）call和apply唯一不同在于传参形式，

call第一个参数是this指向，第二个开始分别对应函数的参数，

apply第一个参数也是this指向，第二个参数是数组，数组里头分别对应形参

（4）另外与call和apply相似的函数，bind()。该函数只是绑定函数的this指向，不会立即执行该函数。

**85.下面代码的执行结果是什么？**

```
var obj = {
    name: 'obj'
}
var bar = {
    name:'bar',
    show:function(){
        var name = 'foo'
        console.log(this.name)
    }
}
bar.show.call(obj);
```

答案：'obj'

**86.下面代码的执行结果是什么？**

```
var val = 1
var fn = function(){
    var val = 2;
    return function(){
        console.log(this.val)
        console.log(val)
    }
}
fn()()
```

答案：1 2
解析：fn()执行返回function(){},然后再执行，执行时里面this为window，所以this.val是1，val是局部变量val为2

**87.下面代码执行结果是什么？**

```
var a = function () {
    this.b = 2;
}
a.prototype.b = 20;
b = new a();
console.log(b.b);
var b = 1000;
a();
console.log(this.b)
```

答案：2，2

解析：b被new出来的时候，里面自己保存了一个b，所以不用去原型上找，即第一个b.b打印出2

当a()执行的时候，由于里面this指向window，所以执行了window.b=2；所以最后全局答应this.b就是window.b,即打印出2

**88.下面代码的执行结果是什么？**

```
var a = { n: 1 }
var b = a
a.m = a = { m: 2 }
console.log(a)
console.log(b)
```

答案:a为{m:2}，b为{n:1,m:{m:2}}
解析：这题需要特别注意，在js在读取a.m = a = { m: 2 }的时候，是从左往右；执行赋值是从右往左的。所以先开始a和b指向的空间{n:1}变成{n:1,m:{m:2}}，而a被重新赋值为{m:2}，所以最终解雇结果a为{m:2}，b为{n:1,m:{m:2}}

**89.下面代码的执行结果是什么？**

```
function Foo(){}
var f1 = new Foo()
console.log(f1.constructor)
Foo.prototype = {}
var f2 = new Foo()
console.log(f2.constructor)
```

答案：Foo(){}，Object(){}
解析：需要注意constructor是定义在原型上的东西，所以它会先找到自己的原型，然后再看constructor指向谁。
因此，在没改变原型前new出来的实例f1的constructor指向Foo，改变构造函数的原型后再new出来的对象f2的constructor就是Object(){}

**90.下面代码的执行结果是什么？**

```
function Person(){
    this.name = 'alice'
    return true;
}
var p = new Person();
console.log(p)
```

答案：Person {name: "alice"}

解析：构造函数构造对象时，如果构造函数本身显示return 原始值，会自动忽略，任然返回构造出来的this，如果显示返回引用值，那么this会被覆盖

**91.下面代码的执行结果是什么？**

```
var val = 1;
val.length = 2;
console.log(val.length)
```

答案：undefined
解析：调用原始类型的什么属性，其实是js内部将原始值变成包装类对象，执行完之后删除该对象。所以val.length为undefined

**92.下面代码的执行结果是什么**

```
var fn = function a(){
    console.log(a)
    a = 1;
    console.log(a)
}
fn();
console.log(a)
```

答案：function a(){...} function a(){...} Uncaught ReferenceError: a is not defined
解析：具名函数表达式的特点，函数名可以在函数内部访问，但是无法被修改。

**93.下面代码的执行结果是什么？**

```
function a() {
    console.log(1)
}
function b() {
    console.log(2)
}
function c() {
    console.log(3)
}
function d() {
    console.log(4)
}
false && a();//&&前面为false后面不会再看，所以这里不执行a()
true || b();//||前面为true后面不会再看，所以这里不执行b()
false || c();//||前面为false后面会再看，所以这里执行c()，打印出3
true && d();//&&前面为true后面会再看，所以这里执行d()，打印出4
```

答案：3 4

解析：主要&&和||运算符的特点，原则是根据前面的真假值来判断是否还要执行后面的表达式

**94.下面代码的执行结果是什么？**

```
function fn(a){
    let a = 10;
    console.log(a)
}
fn(1)
```

答案：Uncaught SyntaxError: Identifier 'a' has already been declared
解析：函数内，用let或者const声明的变量，不能与形参名相同，否则会报错

**95.下面代码的执行结果是什么？**

```
function fn1(){
    var a = 1;
}
function fn2(){
    return;
}
console.log(fn1(),fn2())
```

答案：undefined undefined
解析：没有返回的函数，默认返回undefined；只写return的就相当于没写返回，也是undefined

**96.手写圣杯继承**

```
var inherit = (function () {
    function F() { };
    return function (Target, Origin) {
        F.prototype = Origin.prototype;
        Target.prototype = new F();
        Target.prototype.constructor = Target;
        Target.prototype._super = Origin.prototype;
    }
}());
```

**97.封装一个函数mul，用于计算n的阶层**

```
function mul(n){
    if(n==0||n==1){
        return 1;
    }
    return n*arguments.callee(n-1);
}
```

**98.封装一个函数bytesLengh，用于查找任意字符串的字节长度**

```
function bytesLengh(str) {
    var count = str.length;
    for (var i = 0; i < str.length; i++) {
        if (str.charCodeAt(i) > 255){
            count++;
        }
    }
    return count;
}
charCodeAt(n)：返回字符串第n位字符的unicode编码，大于255字节长度为2，小于为1
```

**99.封装一个方法insertAfter,功能类似insertBefore**

```
Element.prototype.insertAfter = function (targetNode, afterNode) {
    var beforeNode = afterNode.nextElementSibling;
    if (beforeNode) {
        this.appendChild(targetNode);
    }
    this.insertBefore(targetNode, beforeNode);
}
```

**100.手动封装异步加载js的方法asyncLoadScript**

```
function asyncLoadScript(url, callback) {
    var script = document.createElement('script');
    script.type = 'text/javascript';
    if (script.readyState) {//readyState是IE中的状态码
        script.onreadystatechange = function () {
            //绑定监听状态码的事件，IE状态码变成complete或者loaded，表示该元素加载完
            if (script.readyState == "complete" || script.readyState ==
"loaded") {
                callback();//回调函数，当script加载完后调用
            }
        }
    } else {
        //非IE 用onload事件，表示当script加载完时
        script.onload = function () {
            callback();//回调函数，当script加载完后调用
        }
    }
    script.src = url;//放在这，是为了避免IE立即加载完，立即加载完就不再触发
onreadystatechange
    document.head.appendChild(script);//加载到页面中去
}
//=============test code=========================
asyncLoadScript('./js/tools.js', function () {
    //code
    console.log('按照加载完了：' + url + '文件')
});
```

注：

- 很多题目在原本题目的基础上进行修改，原题可关注微信公众号查阅：Duing
- 本文已发布于简书：100道前端笔面试题及答案，如有侵权请第一时间联系删除
- 文章来源自Duing，如需转载请联系Duing（ID：duyi-duing）


《扫码关注
关注我们的人都是可爱的