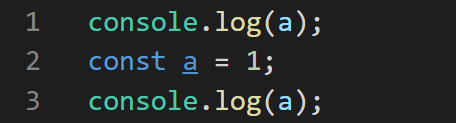
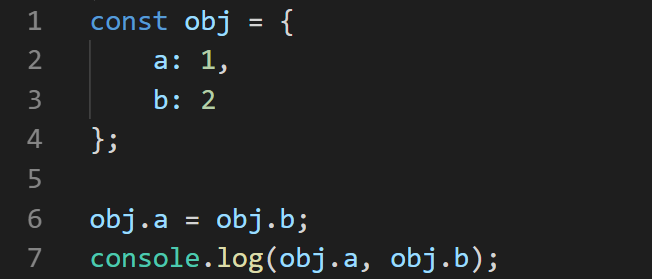
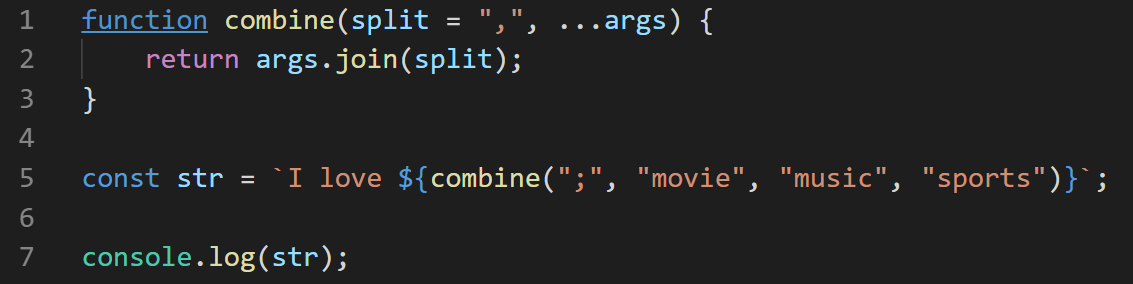
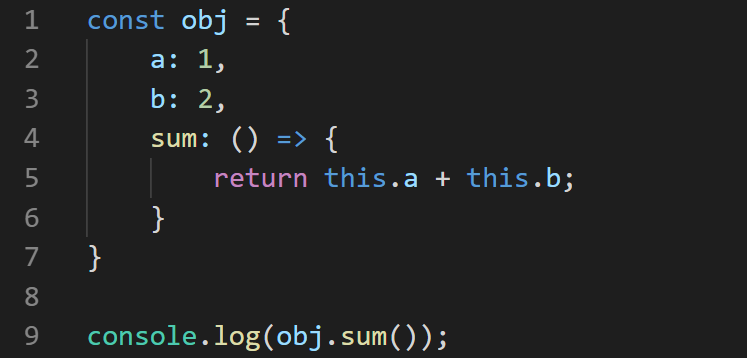
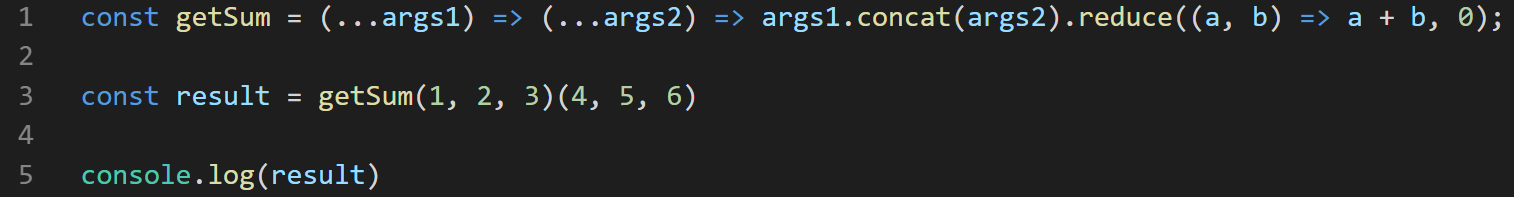
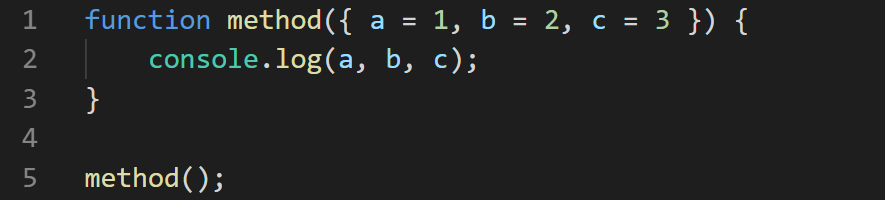
# ES6 结业考试题

1. **选择题（均为单选题，每小题2分，共24题， 48分）**
2. 下面代码的输出结果正确的是（ A ）  
     
   A. 第1行报错  
   B. 第2行报错  
   C. 第3行报错  
   D. undefined 1
3. 下面代码的输出结果正确的是（ B ）  
     
   A. 第6行报错  
   B. 2 2  
   C. 1 2  
   D. 第7行报错
4. 下面的代码输出结果正确的是（ B ）  
     
   A. I love movie, music, sports  
   B. I love movie; music; sports  
   C. I love ;, movie, music, sports  
   D. 报错
5. 下面的代码输出结果是（ C ）  
     
   A. 3  
   B. undefined  
   C. NaN undefined + undefined  
   D. null
6. 下面的代码，输出结果正确的是（ D ）  
     
   A. 0  
   B. NaN

C. 报错  
D. 21

1. ES6中类的特点**不包含**（ B ）  
   A. 不能当作普通函数调用  
   B. 类声明不会被提升  
   C. 类中的代码自动处于严格模式  
   D. 类中的实例方法可以被枚举
2. 如果A和B是两个类，A是B的父类，B是A的子类，则下面的描述正确的是（ A ）  
   A. 可以描述为：B是A  
   B. A和B拥有同一个原型  
   C. A的实例的原型链上有构造函数B  
   D. B的实例的原型链上有构造函数A
3. 下面的代码，输出结果正确的是（ C ）  
     
   A. undefined undefined undefined  
   B. 1 2 3  
   C. 报错  
   D. null null null
4. 对于ES6中的符号Symbol，说法**错误**的是（ B ）  
   A. 符号主要用于产生一个唯一的值  
   B. Symbol可以当作构造函数调用  
   C. 符号可以共享  
   D. ES6规定了一些知名符号，它们每个有着特殊的作用
5. ES6使用一种标准的模式Promise来处理异步，它将一个异步任务分为（ A ）  
   A. unsettle 和 settled 两个阶段  
   B. pending 和 resolved 两种状态  
   C. resolved 和 rejected 两种状态  
   D. thenable 和 catchable 两个阶段
6. 如何理解Promise的构造函数参数（ C ）  
   A. 描述如何进行后续处理  
   B. 描述settled阶段如何处理  
   C. 描述在unsettled阶段如何处理  
   D. 如何通向settled阶段不由该参数负责
7. 如何理解Promise对象的then方法（ C ）  
   A. 通过then方法可以让Promise通向settled阶段  
   B. 通过then方法可以让Promise通向resolved状态  
   C. 通过then方法可以向Promise注册thenable或catchable处理函数  
   D. 通过then方法可以向状态传递数据
8. 下面哪种做法**不能**让Promise进入rejected状态（ D ）  
   A. 在Promise构造函数的参数方法中抛出错误  
   B. 在Promise构造函数的参数方法中调用reject()  
   C. 在Promise构造函数的参数方法中代码报错  
   D. 调用Promise的catch方法
9. 当你需要同时发送多个请求，并把所有响应的结果进行合并，你应该选择（ B ）  
   A. Promise的串联  
   B. Promise.all  
   C. Promise.race  
   D. 使用async和await
10. 关于async和await说法正确的是（ B ）  
    A. async修饰的方法内必须使用await  
    B. async修饰的方法必定返回Promise对象  
    C. await必须等待一个Promise对象  
    D. await语句之后的代码可能仍然是同步代码
11. Fetch Api相比于XHR的优势包括（ A ）  
    A. 功能分割更加精细  
    B. 摒弃了ajax模式  
    C. 使用的是回调模式  
    D. 使用的是事件模式
12. 对于文件上传表单字段，说法正确的是（ B ）  
    A. 表单字段名必须是file  
    B. 表单字段名需要和服务器进行约定  
    C. 表单字段内容是文件名  
    D. 表单格式必须是multipart/form-data
13. 下面的哪种对象**不是**可迭代对象（ A ）  
    A. Object  
    B. Array  
    C. NodeList  
    D. String
14. 对于yield关键字理解正确的是（ A ）  
    A. yield后面的表达式表示迭代的结果  
    B. yield会导致浏览器运行阻塞  
    C. yield语句的返回结果是后面表达式的结果  
    D. yield语句可以用到普通函数中
15. 关于生成器函数理解正确的是（ D ）  
    A. 生成器函数调用后会运行到第一个yield位置停止  
    B. 生成器函数本质上就是迭代器  
    C. 生成器函数返回的是一个Promise对象  
    D. 生成器函数返回的一定是一个可迭代对象
16. 关于set和map集合的说法**正确**的是（ A ）  
    A. set集合没有filter方法  
    B. map集合的值（value）是唯一的  
    C. map集合在迭代时，每次迭代得到的是一个普通对象  
    D. 用set构造函数初始化集合内容时，参数必须是一个数组
17. 关于Reflect的说法，正确的是（ C ）  
    A. Reflect只能用于代理中  
    B. Reflect 出现后，过去的相关语法已被淘汰  
    C. Reflect使用api的形式暴露JS底层实现  
    D. Reflect.defineProperty和Object.defineProperty完全一样
18. 如果要从一个学生数组中找到姓名为“李华”的学生对象，最合适的API是（ C ）  
    A. includes  
    B. indexOf  
    C. find  
    D. findIndex
19. 关于ArrayBuffer和类型化数组，说法正确的是（ D ）  
    A. ArrayBuffer可以存储任何类型的数据  
    B. 类型化数组可以存储任何类型的数据  
    C. ArrayBuffer只能通过类型化数组操作数据  
    D. 类型化数组可以降低内存消耗

二、**填空题（每空1分，共12空， 12分）**

1. 过去，一个函数总是可以有两种使用方式：直接调用和使用关键字new，这模糊了函数的功能，ES6之后，我们可以通过 **new.target** 来判断使用者使用了何种方式调用函数。
2. 在使用解构时，我们可以通过符号  **:**  来修改解构到变量的名称
3. 将下面的代码补充完整

/\*\*

 \* 请求指定的地址，得到服务器返回的json对象

 \* @param {\*} url

 \*/

const req =  async (url) => {

    return await fetch(url).then( resp  => resp.json());

}

//请求students.json，输出得到的结果

req("students.json"). then (resp => console.log(resp))

1. ES6对函数做了大量改进，其中， **参数默认值** 解决了参数默认值的问题， **剩余参数** 解决了参数数量不固定的问题
2. Promise可以从unsettled阶段推向settled阶段，调用resolve函数可以进入 **resolved** 状态，调用reject函数或 **抛出错误** 可以进入 **rejected** 状态
3. 一个可迭代对象，必定满足 **属性名是知名，属性值是生成迭代器的函数** ，即包含一个知名符号Symbol. **iterator**

**三、简答题（共2题，每题10分， 20分）**

1. 阐述Promise解决了什么问题，它如何解决的，async和await又解决了什么问题？

Promise没有消除回调，但是让回调变得可控

ES6将一件可能发生异步操作的事情，分为两个阶段

1. unsettled：未解决阶段，表示事情还在进行处理，没有发生通向结果的那件事

2. settled：已解决阶段，表示事情已经有了结果，不管是好是坏，事情无法逆转

unsettled阶段拥有控制何时通向settled阶段的能力

Promise将事情分为三个状态：pending、resolved、rejected

pending：挂起，处于unsettled阶段

resolved：已处理，settled，出现正常结果

rejected：已拒绝，settled，通常有一个错误

async和await，简化了Promise的使用。

能将异步代码，写成同步代码的样子。

async函数自动返回一个Promise对象，在async函数里写的代码，相当于写在Promise构造函数里，return的值为Promise的resolve的值。

await要在async函数中使用，在await写的代码，相当于写在了Promise.then中。

1. 阐述什么是迭代器、可迭代协议、可迭代对象、生成器

**迭代器**

- 迭代器应该具有得到下一个数据的能力，应该具有判断是否还有后续数据的能力

**可迭代协议**

可迭代协议允许 JavaScript 对象去定义或定制它们的迭代行为, 例如（定义）在一个 for..of 结构中什么值可以被循环（得到）。

**可迭代对象**

ES6规定，如果一个对象具有知名符号属性 Symbol.iterator，且属性值是一个创建迭代器的函数，则该对象是可迭代的。

**生成器**

帮助实现生成迭代器的函数。

四、**编程题（共2题，每题10分，20分）**

1. 有一个字符串数组urls，里面存放了很多的请求地址，现在需要同时向这些地址发送ajax请求，并把每次请求的JSON结果按照响应的先后顺序依次保存到一个新数组results里面，当所有请求完成后，把新数组results输出

results数组中每一项的对象格式如下：

{

requestTime: 日期对象, //开始请求的时间

data: 对象, //服务器响应的结果

responseTime: 日期对象 //响应的时间

}

// 请求地址数组

const urls = [

    'https://api.github.com/users',

    'https://api.github.com/users/mojombo/repos'

]

// 存放promise数组，以便Promise.all使用

const proArr = [];

// 结果数组

const results = [];

urls.forEach(url => {

    const result = {}; //结果对象

    result.requestTime = new Date(); //请求时间

    const pro = fetch(url); //请求

    proArr.push(pro);

    // 注册后续处理函数

    pro.then(resp =>{

        result.responseTime = new Date(); //响应时间

        // console.log(resp);

        return resp.json();

    }).then(data => {

        // console.log(data);

        result.data = data;

        results.push(result);

    })

})

const proAll = Promise.all(proArr);

proAll.then(() => {

    console.log(results);

},err =>{

    console.log(err);

})

1. 编写一个函数createDomProxy，该函数返回一个DOM对象的代理，该代理可以将DOM对象以on开头的属性（即事件）变为Promise对象，可以让后续的开发者这样使用这个函数createDomProxy：

async function test() {

    const div = document.getElementById("test"); //得到某个div dom

    const divProxy = createDomProxy(div); //创建它的代理

    while (true) { //使用一个死循环，是为了不断的监听它被点击

        const e = await divProxy.onclick //等待它被点击，e是事件参数

        console.log("div被点击了", e) //被点击后运行的代码

    }

}

test();

function createDomProxy(dom) {

    const domProxy = new Proxy(dom,{

        get(target,key){

            // console.log(target, key)

            // 如果key是以'on'开头

            if (key.startsWith('on')) {

                return new Promise((resolve, reject) => {

                    dom[key] = function (e) {

                        resolve(e);

                    }

                })

            }

            else {

                return dom[key];

            }

        }

    })

    return domProxy;

}