***JS模块导出基本语法：***

🡪在模块文件中，export关键字表示此函数可以被外界引用。默认情况下，模块中的函数是无法被其他JS文件所引用，只能在模块内使用。从而保证了模块内的变量不会污染外部的全局变量。

使用方式是,可以将export放在任何变量,函数或类声明的前面,从而允许他们从模块导出。

**例如导出普通数据类型，变量前面加上export关键字：**

写法1:

export var name = "Hui";  
export let weChatPublic = "Hui";  
export const time = 2018;

写法2，在最后进行导出：

var name = "Hui";  
let weChatPublic = "Hui";  
const time = 2018;  
  
export {name, weChatPublic, time}

**例如导出函数，函数前面加上export关键字**

写法1：

export function sum (num1, num2) {  
     return num1+num2;  
}

写法2，在最后进行导出：

function sum(num1,num2){  
      return num1+num2;  
}  
// 之后将其导出  
export sum;

**例如导出类，类前面加上export关键字：**

写法1:

export class People {  
     constructor(name,age){  
        this.name = name;  
        this.age = age;  
     }  
     info(){  
        return `${this.name}${this.age}`  
     }  
}

写法2，在最后进行导出：

class People {  
     constructor(name,age){  
        this.name = name;  
        this.age = age;  
     }  
     info(){  
        return `${this.name}${this.age}`  
     }  
}

export People

**-🡪我们可以使用as关键字来指定变量,函数,或者类在模块外应该被称为什么名称来达到导出时候重命名的效果**

例如：

function sum(num1,num2){  
    return num1+num2;  
}  
export {sum as add} // # 本地名称为sum, 使用as关键字从而在导出的时候为add.同时在未来导入的时候，也需要使用add作为标示名进行导入。

-🡪 **default关键字。export 之后加上default意指默认接口的意思，在一个文件里面只能使用一次default关键字,该default可以指定某一个变量，也可以将多个变量集合申明一次。 并且被default修饰的导出，在未来引入时不需要使用{}来放入对应的名字。只需要任意一个名字来接收对象即可。**

例子1：

let a = “Apple”;

export default {a}; # 导出默认变量 a

import a from “./example.js” # 不需要花括号也会导出模块中的a变量。在本文件中通过变量a接收。

import anyname from “./example.js” # 不需要花括号也会导出模块中的a变量。在本文件中通过任意一变量名也能接收。

例子2：

let a = “Apple”;

function test(name) {

console.log (“….”);

}

export default {a, test}; # 导出默认变量 a

import a from “./example.js” # 不需要花括号也会导出模块中的默认变量。在本文件中通过变量a对象接收, 该a对象含有模块中所有的默认导出变量。a.a为Apple, a.test为模块中的test方法。

***JS模块导入基本语法：***

import {identifer1, indentifer2} from "./example.js"

import语句的两个部分组成分别是，要导入的标识符和标识符应当从那个模块导入。另外,导入的标识符的顺序可以是任意位置,**但是导入的标识符(也就是大括号里面的变量)与export（本模块中定义）暴露出的变量名必须是一致的。除非该模块暴露的变量已经被default申明。**

**当模块中导入一个绑定时, 它就好像使用了const定义的一样,也就是自动默认使用严格模式。你无法定义另一个同名变量(包括导入另一个同名绑定， 对导出的变量重新赋值),也无法在import语句前使用标识符或改变绑定的值**

例如：

import {sum} from "./example.js"  
console.log (sum (1,2)); // 3  
sum = 1; # 非法，无法对导出的变量进行重新赋值，除非调用模块提供的方法，由该模块的方法来改变模块提供的值

**将模块作为新的命名空间进行整体导入：**

特殊情况下,可以导入整个模块作为一个单一的对象,然后所有的导出都可以作为对象的属性使用。

例如:

// 导入一整个模块  
import \* as example from "./exportExample.js"  
console.log (example.sum(1, example.time));  
consoole.log(example.multiply(1,2));// multiply与sum函数功能一样

在上面这段代码中,从本地模块的exportExample.js导出所有的绑定被加载到一个被称作为example的对象中,模块中指定了export关键字的sum()函数,multiply()函数和time之后作为example的属性被访问,这种导入格式被称为命名空间导入,因为exportExample.js文件中不存在example对象,所以它被作为exportExample.js中所有导出成员的命名空间对象而被创建。

**-🡪我们可以使用as关键字来指定变量,函数,或者类在模块外应该被称为什么名称来达到导出时候重命名的效果**

例如：

import {add as sum} from "./exportExample.js" #导入函数add, 重命名为sum  
console.log (sum (1,2)); // 3  
console.log (typeof add); // undefined, 导入重命名后便没有add标识符

**-🡪我们可以引入默认变量的同时，也引入指定名字的变量。**

例如：

import react, {Component} from ‘react’;

将为本js文件引入默认变量到react对象中，并且额外引入模块中定义的Component变量

**注意：export和import的一个重要的限制是, 他们必须在其他语句和函数之外使用。不允许在条件语句中使用，不允许动态加载。**

**注意：导入模块的时候，使用.以及..表示采用相对路径导入，系统将从本文件所在文件夹为起点进行相对路径搜索。如果不使用.以及..开头的相对路径寻找，则在React的环境下，代表从node\_modeuls/中开始寻找。**