1、了解背景知识：历史、现状、特点、应用场景

2、搭建开发环境，编写一个hello wrold

3、从变量和常量，用于临时存储数据

4、数据类型

5、运算符（操作数据）

6、逻辑结构（执行方式）

7、通用小程序

8、函数和对象

9、第三方的库、框架

10、实用的项目

一、JS(JavaScript解释器/引擎)

1、JS 概述

（1）历史现状

1995年，JS 出现在Netscaoe 的浏览器中

1997年JS提交给ECMA,出现了ECMAScript 标准规范。

2009年，JS遵循CommonJS规范，开始向服务器端发展。

(2）现状

既可以运行在客户端浏览器也可以运行在服务端浏览器NodeJS.

2、特点

JS脚本文件无需编译，由JavaScript引擎解释执行（解释性语言，编译一行执行一行）

弱类型语言

基于对象

跨平台性

3、应用场景

(1)编写在网页中，操作网页内容，实现浏览器端的各种交互效果、

(2)移动APP、

编写企业服务端应用、

创建WEB服务器、操作数据库等服务器端应用

2D绘图

2、JS的执行环境

（1）浏览器自带的JS解释器

（2）NOdeJS 下的JS 解释器（10.15.3LTS左侧稳定版。https://nodejs.org Nodjs下载地址）

在命令行下 输入 node -v 查看当前NodeJS 版本号

（3）执行JS代码

浏览器下：

创建角色文件嵌入到HTML 文件中

<script src='01-hello.js'></script>

NodeJS下：

node c:/xampp/…/01-hello.js 回车

①js 当中区分大小写

②js当中结束；号可加可不加，习惯上都加

③js 当中// …表示单行注释、/\*…\*/多行注释

4、变量（用于存放数据的容器）

X=1 Y=2

（1）声明变量（定义一个变量）

关键字 var x=1(在内存中创建空间，名称叫X ,把数据1 存放到

X中)

练习：声明多个变量，分别保存一个员工的姓名、编号、性别、生日、工资部门编号，并打印出来

（2）变量的命名规则

变量的名称由字母、数字、美元符号、下划线组成的；不能以数字开头。

多个连词之间命名（驼峰命名、下划线命名）

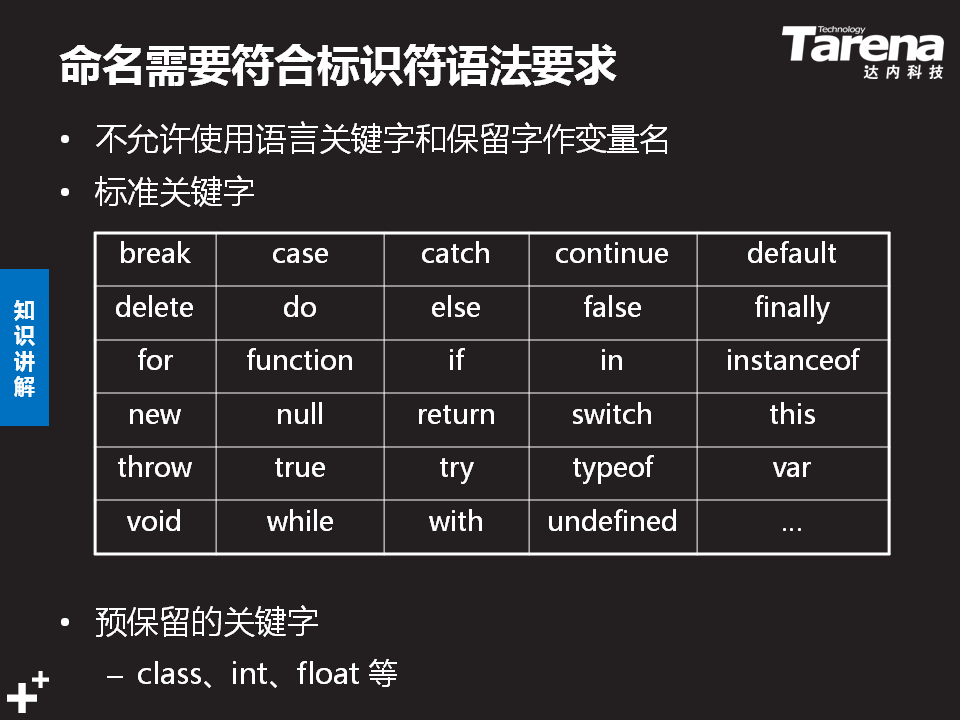
userName

userAge

userPassword

user\_name

user\_age

(3)var num;

变量声明后未赋值，结果为undefined

弱类型语言，声明变量的时候不需要指定数据类型，后期可以存储任意的类型

性声明多个变量：(多个变量之间用逗号隔开)

var a=1,b=1,c

console.log(a,b,c);

练习：声明多个变量，分别保存语文、数学、总成绩：最后打印

var chinese=99,math=99,total

console.log(a,b,c);

(5）常量

一旦声明不能重新赋值

例如：圆周率、春节的日期

const PI=3.14

练习

声明变量保存圆周率的半径，声明常量保存圆周率；声明变量的面积和周长，并打印出来

var r=5;

const π=3.14;

var p=2\*π\*r

var s=π\*r\*r

console.log(r,π,p,s)

3、数据类型

数据范围：原始类型和引用类型

原始类型：数值型、字符串型、布尔型、未定义类型、空

（1）数值型

分为整形和浮点型

整形在内存中站四个字节，浮点型站八个字节

十进制1…9

八进制1…7 亿0开头 013->11

十六进制 以0x开头 0xa->10

a-f代表10-15 不区分大小写

0xff ->225

浮点型分为普通小数和指数型小数

3.14e6->3140000

3.14e-6 ->0.00000314

typeof 检测数据类型

（2）字符串类型

数据被引号包含就是字符串类型，不区分单双引号

查看任意一个字符串的unicode码

console.log('2'.charCodeAt()); //97

（3）布尔型

true/false

在程序中表示真或假，一般用于一些是否的结果，如是否注册，是否在售…

//var islogin=false;

//var isindex=true;

//console.log(islogin,typeof isindex);

(4)未定义型

声明了变量为赋值，结果就是undefined

(5)空---null

常和引用类型数据一起使用

4、数据类型转换

（1）隐式转换

通常是和一些运算符一起使用的

①数字+字符串 数字转换成字符串

var str1=1+"2"; //12

var str2=3+"4"; //34

console.log(str1,typeof str1);

②数字+布尔型 布尔型转换成数字true-->1 false->0

var str3=3+true //4

var str4=3+false //3

③字符串+布尔型 布尔型转换成字符串

var str5=“5”+true //5true

JS中加好（+）的作用

执行加法运算

执行字符串拼接

JS中减(-)、乘（\*）、除（/） 隐式换行

将运算符两端的数据转为数值型（自动调用Number）,如果转换失败，返回NaN(Not a Number),不是一个数字，

任何值和NaN执行减乘除运算符都会返回NaN.

练习：查看一下程序的运行结果

var num1=3,num2=true,num3='tedu';

console.log(num1+num2+num3); //4tedu

console.log(num2+num3+num1); //truetedu3

console.log(num3+num1+num2); //tedu3true

（2）强制转换

①将任意数据类型，强制转换为数值型

Number()

|  |  |
| --- | --- |
| Number('a') | //NaN |
| Number('1') | //1 |
| Number('1a') | //NaN |
| Number(undefined) | //NaN |
| Number(null) | //0 |
| Number(true) | //0 |

②将数据转换为整形

parseInt();

常用于将字符串或数字转为整形，其他的数据返回NaN；如果要转换的字符串以非数字开头，也返回NaN.

③ 将数据转换为浮点型

parseFloat()

和parseInt的用法基本一致，只是转的数据式浮点型

④数值型和布尔型转为字符串型

toString()

//var num=10;

//var str=num.toString(); //'10'

var bool=true;

var str=bool.toString();

console.log(str,typeof str);

5运算符

表达式：由运算符连接到操作数据，所组成的形式。

运算符分为：

①算术运算符

+（加） -（减） \*（乘） /（除）

%（取余）console.log(3%2);

console.log(2%3);

console.log(-2%3);

//一个值是否为奇数

console.log(27%2);

++（自增）在原来的基础之上加一

console.log(num++);先打印num的值，在执行自增

console.log(++num);先执行自增，在打印num的值

--（自减）在原来的基础之上减一

console.log(num-- + --num);//2

//先取出num的值3，然后执行自减，num的值变成了2。在执行自减，num的值变成了1，然后取出num的值，也就是1.

var a='1'

//自增隐式将数据转为数值型

a++

console.log(a,typeof a)

②比较运算符

包含：> < >= <= != ==(判断)

===(全等于) ！==全不等于

返回一个布尔型的值

==只是返回两个值是否相等

===不仅比较值，还会比较类型是否相等

如果数据类型不同，会发生数据类型转换

console.log(3>'10');字符串转成数字10

‘3’>'10'比较的是unicode码(代码查询方法console.log('h'.charCodeAt())）

3>'10a' //false

3<'10a' //false

3=='10a' //false

//'10a'转成数值型，调用Number,返回NaN.

NaN和任何职比较(< > >= <= == ===)都返回false

NaN==NaN返回false

③逻辑运算符

或者 （||）

关联的两个条件只需要满足其一，结果是true,否则false

并且(&&)

关联的两个条件都是true,结果是true,否则false

//80~90以下

var score=85;

console.log(score>=80 && score<90);

逻辑短路

&& 当归各条件为false的时候，就不需要在执行第二个条件

|| 当过一个条件为true 的时候，就不需要在执行第二个条件

逻辑短路无需关注最后的结果是true还是false,重点是看是否执行第二个表达式

练习以下程序是否会报错

var num=3;

num>5 && console.log(a);

num<1 || console,log(a);

练习声明变量保存一个人的年龄，如果满18岁，打印‘成年人’

var age=18;

age>=18 && console.log('成年人')

非（反向)!

!false ->true !true->false

console.log(!!(3>1));

④位运算符(了解)

在执行运算的时候，计算机会把数据转换成二进制进行运算。

1 10 11 100 101 110 111 1000 1001 1010

2 4 8

25 ->16+8+1

->10000+1000+1 -> 11001

按位与（&） binary

上下两位都是1，结果是1，否则是0

console.log(5&7);

/\*

101

111

——

101

\*/

按位或（|）上下两位含有1，结果是1，否则是0

console.log(7|9);

/\*

0111

1001

————

1111

\*/

console.log(13|20)

/\*

01101

10100

————

11111

\*/

按位异活（）上下两位不同为1，相同位0

console.log(4^7);

/\*

100

111

\_\_

011\*/

console.log(5^8);

/\*

0101

1000

\_\_\_\_

1101

\*/

按位右移（>>）删除二进制的最后一位，每次缩小到原来的一半或更多

按位左移（<<）在二进制的最后添加0，成被的增加

⑤赋值运算符

= += -= \*= /= %=

a=1

a++ a=a+1 a+=1

如果a是字符串，+=变成了字符串拼接

//神声明变量保存商品的价格，价格在原来基础之上打八折，然后打印价格

var price=120;

price\*=0.8;

console.log(price)

⑥三目运算符

条件表达式？表达式1 ：表达式2

A ? B: C;

如果A成立，则执行B,否则执行C

如果条件表达式为true，执行表达式1

如果条件表达式为false,执行表达式2

（由三个表达式或数据组成的形式）

var age=17;

age>=18?console.log('成年人'):console.log('未成年人');

//声明两个变量分别保存用户名和密码，如果用户是root，并且密码是123456，打印‘登陆成功’，否则打印‘登录失败’

var uname='root';

var poswrd='123456';

var res=(uname=='root'&&poswrd=='123456')?'登陆成功':'登录失败';

console.log(res);

练习：（1）声明变量保存一个年份，判断这个年分是否为闰年，结合逻辑短路，如果是闰年，打印闰年

闰年：4年一个闰年（能被4整除），并且不能被100整除，或者能被400整除

var year=2000;

(year%4==0&&year%100！=0||year%400==0)&&console.log('闰年');

（再不加括号的情况下，此处先执行&&，在执行||）

6、浏览器端函数

①js 文本中alert();//弹出警示框

alert('web');//(alert警示框)

alert('tedu.cn')//(多次警示框，弹出顺序，一个接一个)

在html中插入js 文本，在浏览器运行

<script src='browser.js'></script>

②//弹出提示框（输入框）

prompt()弹出提示框，需要使用变量来接受输入的值；值得类型是字符串

//把输入的值存储到变量中

var str=prompt('input your name');//(prompt提示框)

console.log(str,typeof str);

在html中插入js 文本，在浏览器运行

<script src='browser.js'></script>

//练习弹出两次提示框，分别输入两个数字，计算两个值相加：把相加的结果一警示框形式弹出

var num1=prompt('input first number')

var num2=prompt('input second number')

//把num1和num2转换为数值型

num1=Number(num1);

num2=Number(num2);

//弹出相加的结果

alert(num1+num2);

7、逻辑结构————流程控制

程序=数据+算法（数据是用来计算）

程序分为：顺序执行、选择执行、循环执行

一、选择执行：

（1）if语句(当前面的条件，)

if(条件表达式){

语句；

}

如果条件表达式为true,会执行后边的语句

if ( A ){ B;}

如果满足条件A，则执行B过程;

在条件表达中，有一些默认值转成false

0 、NaN 、'' 、undefined、null

var str='';

if(!str){

str='这家伙很懒，什么也没留下';

}

console.log(str);

if(str==''){

str='这家伙很懒，什么也没留下';

}

//练习：声明变量保存个人签名，如果签名内容为空，给签名赋值什么也没留下，打印变量

（2）if-else语句

if（条件表达式）{

语句1;

}else{

语句2;

}

如果条件表达式为true,执行语句1，否则条件表达式为false,执行语句2

如果if或者else后只有一行语句，则后边的大括号可以省略

if(A){B}else{C};

如果满足条件A则执行过程B，反之执行过程C;

//练习：声明变量保存性别的值（1/0），如果是1，打印’男‘；否则打印女

var sex=1

//0-false 1-true

if(sex){

console.log('男');

}else{

console.log('女');

}

练习使用弹出提示框分别输入商品的单价和数量，计算商品的总价满500打八折；使用变量保存账户余额，如果余额足够支付，打印pay success，否则打印pay error 07-exercise.html 07-exercise.js

<script src="07-exercise.js"></script>（html中）

var str=prompt('input price');

var str1=prompt('input count');

//str=Number(str);

//str1=Number(str1);

//alert(str\*str1)

//计算总价

var total=str\*str1;

if(total>=500){total\*0.8;}

console.log('total');

var money=600;

//如果余额足够支付，支付成功否则支付失败

if(money>=total){console.log('pay success');

}else{console.log('pay error');}

（3）if-else嵌套

用于判断多种情况

if(条件表达式1){

语句1；

}else …if(条件表达式n){

语句n；

}else{

语句n+1;//以上所有的条件表达式都是false

}

if (A){B;}else if(D){C;}else{E;}

如果满足条件A,则执行过程B;反之如果不满足条件A,但满足条件D是则执行过程C,反之若A条件，B条件都不满足，则执行过程E;

练习

//判断一个人的政治面貌

var type='少先队员';

if (type=='党员')

{console.log('此人为党员');

}else if (type=='团员')

{console.log('此人为团员');

}else if (type=='群众')

{console.log('此人为群众');

}else {console.log('此人为非法的政治面貌');

}

(4)switch-case语句(转换)

是一种特殊的分支语句，可以根据一个表达式的值来选择执行不同的程序

switch(表达式){

case 值1://如果表达式的值是值1

语句1;

break(//结束);

………

case 值n:

语句n;

break;

default:语句n+1;//以上所有的表达式和值比较都是false

}

注意：在case中，表达式和值比较是用的是全等于比较，要求值和类型都满足星期代码

switch （A）{case B: C; break;

case D:E;break;

default:F;}

当表达式A的值为B，则执行程序C,结束该程序；

当表达式A的值为D是，则执行程序E,结束该程序；如果以上值都不与表达式A满足时，则执行程序F;

要求对比if-else嵌套和switch-case语句

相同点：两者都可以用于多项分支语句

不同点：if-else既可以判断相等，也可以判断不等情况，适用范围广；switch-case只能用于全等于的比较，在结构上更为清晰，执行效率相对高。

练习：使用switch-case语句完成成绩评判

原理：让分数除以10，然后取整，在使用语句。

var scocr=10;

var srt=scocr/10;

parseInt(srt)

switch (srt)

{

case 6:

console.log('及格');

break;

case 7:

console.log('中等');

break;

case 8:

console.log('良好');

break;

case 9:

case 10:

console.log('优秀');

break;

default :

console.log('不合格');

}

8、循环相关

一遍一遍又一遍的执行相同或相似的代码。

循环的要素

循环的条件：重复的次数

循环体：执行的代码

[（1）while循环](02/411while.js)

while(循环条件){

循环体;

}

先给定条件，后执行循环体；

break 可以结束任何形式的循环，break执行后，后边所以得循环体和循环条件都不在执行

//打印1-10之间所有的整数

var i=1;

while(true){console.log(i);

//当i为10的时候，结束循环

//后边的循环体和循环条件都不执行

if (i==10)

{break;

}

i++;

}

练习：声明变量保存1个数字，无限循环弹出提示框，输入数字；如果输入的比保存的大，警示框提示'big',如果输入的比保存的小，警示框提示输入'small',否则提示'right'\*/

//03—break.html循环前声明变量保存数字，无限循环的循环中弹出提示框，获取输入的值，在和之前的变量比较if-elae嵌入套

var a=25;

while (true)

{var str=prompt('input your b');

//查看输入的值和之前保存的值对比

//如果输入的值大于保存的值

if (str>a){

alert('big');

}else if (str<a){

alert('small');

}else {

alert('right');

break;}

}

（2）do-while循环

do{

循环体；

}while(循环条件)；

区别：不管循环条件是否为true都会先执行一次循环体

//打印1-10所有的整数

var i=1;

do

{console.log(i);

i++;

}

while (i<=10);

//阶乘 5！=5\*4\*3\*2\*1

var i=1;

var sum=1;

do

{str=sum\*=i;

i++;

}

while (i<=5);

console.log(str);

(3)for循环

for(初始值;循环条件;增量){

循环体；

}

①执行初始值

②判断循环条件

③如果循环条件是true执行循环体，是false结束循环体

④如果执行完循环体，执行增量

⑤重新执行第二步

//练习1-10所有的整数

/\*for (var i=1;i<=10;i++ )

{console.log(i)

}\*/

//字符串拼接

var str=''

for (i=1;i<=5 ;i++)

{//每次产生的\*拼接到str中

str+='\*'

}console.log(str)

break和continue

break:结束循环，后续不在执行其他的循环

continue:跳过本次循环，继续执行下一次循环

//计算1-100之间所有偶数的和，遇到奇数跳过

for (var i=1,sum=0;i<=100 ;i++ )

{if(i%2==1){continue;}

sum+=i

}console.log(sum)

（4）循环嵌套

while、do-while、for循环三者之间可以相互嵌套

结构1

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

/\*

\*\*\*\*\*

for (var i=1,str=' ';i<=5 ;i++ )

{//每次循环产生一个\*，把他们拼接到字符串str

str+='\*'

}console.log(str)\*/

//外层循环，控制9行————j

for (var j=1;j<=9 ;j++ ){

// 内层循环，控制9列————i

for (var i=1,str=' ';i<=9 ;i++ )

{//每次循环产生一个\*，把他们拼接到字符串str

str+='\*'

}console.log(str)

}

练习：1、打印99乘法表/

2、打印本世纪前10个闰年（打印到第十个，提前结束循环break）

3、计算1-100之间所有整数和，当和大于4000的时候，提前结束循环，打印总和和当前i的值

打印九九乘法表

for (var i=1;i<=9 ;i++ )

{for (var j=1,str=' '; j<=i;j++ )

{str+=i+'\*'+j+'='+j\*i+' ';

}console.log(str)

}

2、打印本世纪前10个闰年（打印到第十个，提前结束循环break）

/\*for (var i=2000,sum=0;i<2100 ;i++ )

{if (i%4==0&&i%100!=0||i%400==0)

{

console.log(i);

sum++;

if (sum==10)

{break

}

}

}\*/

计算1-100之间所有整数和，当和大于4000的时候，提前结束循环，打印总和和当前i的值

for (var i=1,sum=0;i<=100 ;i++ )

{sum+=i

if (sum>4000)

{console.log(sum,i);break

}

}

9函数

parseInt()/parseFloat()………

分为系统函数和自定义函数

（1）自定义函数

function:功能体，函数。用于封装反复执行的代码，可以接受若干个数据，返回处理的结果。

①创建一个普通的函数

function函数名称（）{

函数体；//封装的反复执行的代码

}

调用

函数名称()

练习：使用函数封装1+2的函数结果

/\*function sha(){

console.log(1+2);

}

sha()

sha()

sha()\*/

//练习使用函数封装计算1-100之间所有整数的和，并调用三次

/\*function kk(){

for (i=1,sum=0;i<=100 ;i++ )

{sum+=i;

}

console.log(sum);

}

kk();\*/

②创建带有参数的函数

function 函数名称(参数列表//形参——>形式上的参数,用于接收数据){

函数体

}

调用

函数名称

函数名称（参数列表）；//实际的参数——>传递的数据

参数列表：可以是0个或者多个，之间用多个逗号隔开；创建时的参数成为形参，调用时的参数成为实参，实参会赋值给形参

形参本质可以理解为一个变量，只是未赋值，默认结果是undefined.

//创建带有参数的函数

//计算任意两个数字相加得和

/\*function add(num1,num2){

//在调用的时候，实参的第一个值会赋给num1,实参的第二个值赋给num2;

console.log(num1+num2）

}

add(2,3);\*/

//创建函数，传递一个参数，计算1~任意数字之间的和

/\*function hh(n) {

for (i=1,sum=0; i<=n; i++)

{sum+=i

}

console.log(sum)

}

hh()\*/

//创建函数，传递2个函数，计算任意两个年份之间的闰年个数。n1-n2

function kama(n1,n2){

for (var i=n1,sum=0;i<=n2 ;i++ )

{if (i%4==0&&i%100!=0||i%400==0)

{

sum++;

}

}console.log(sum)

}

kama(2000,2100)

kama(1900,2000)

kama(1862,1962)

③带有返回值的函数

function 函数名称(参数列表){

函数体；

return 值//（返回值）；

}

调用

函数名称（参数列表）

return表示函数执行后，返回的结果；可以保存下来

注意事项：

一、如果没有return和return后没有值，结果都是undefined

二、一旦有return出现后，return后的所有代码都不会执行

//计算任意三个数字相加得和，并把结果返回

/\*function add (a,b,c){

return a+b+c;

}

//把函数的返回结果保存到变量中

var num1=add(3,29,12);

console.log(num1)

//练习：创建一个函数，返回任意两个数字中的最大值。

function getmax (a,b){

if (a>b)

{return(a)

}else

{return(d)

}return max

}var num=getmax(12,9)

console.log(num)

return a>d ? a:d;\*/

//创建函数getmax2,返回任意三个数字中的最大值

/\*function getmax2(a,b,c)

{var max= a>b?a:b;

return max>c?max:c;}

var big=getmax2(25,7,15)

console.log(big)\*/

//创建函数getstatus,传递状态码，返回对应的中文码

//1等待付款 2等待发货 3运输中 4已签收 5已取消 其他无法追踪

/\*function getstataus (a){

if (a==1)

{return '等待付款'

}else if (a==2)

{return '等待发货'

}else if (a==3)

{return '运输中'

}else if (a==4)

{return '已签收'

}else if (a==5)

{return '已取消'

}else{return '无法追踪'}

}var str=getstataus(2)

console.log(str)\*/

/\*function getstataus(code){

switch (code)

{

case 1:return '等待付款';

case 2:return '等待发货';

case 3:return '运输中';

case 4:return '已签收';

case 5:return '已取消';

case 6:return '无法追踪';

}

}

var n=getstataus(5);

console.log(n);

\*/

//创建函数getdays传递任意一个年份，返回对应的的天数

/\*function getdays(year){

if (year%4==0&&year&100!=0||year%400==0)

{return 366

}return 365

}

var str=getdays(2000)

console.log(str)\*/

//判断是否为为闰年，是返回true,不是返回false

/\*function isrun(year){

if (year%4==0&&year%100!=0||year%400==0)

{return true;

}return false;

}

//var res=isrun(2000);

//console.log(res);

//根据年分返回天数

function getdays2(year){

//判断年份year是否为闰年

if (isrun(year))

{return 366;

}

return 365;

}

var d=getdays2(2020);

console.log(d)\*/

//1-100所有整数得和

//计算1-任意数字之间所有整数阶乘的和

//传递一个参数

//步骤一创建函数getjc，传递一个参数，计算任意数字的阶乘

//步骤二创建函数getsum，传递一个参数，计算1~任意数字之间所有整数的和

function getjc(n){

for ( var i=1,ride=1;i<=n ; i++)

{ride\*=i

}return ride

}

//var str=getjc(8);

//console.log(str);

function getsum (n){

for (var i=1,sum=0;i<=n ;i++ )

//i代表中间数字所有的整数

//获取数字的阶乘getjc(i)

{//sum+=i；把所有数字加到sum

sum+=getjc(i)//把所有数字的阶乘加到sum

}

return sum;

}

var str=getsum(15);

console.log(str)

10、变量作用域

（1）作用域

变量和函数的可访问范围，分为两种

函数作用域（局部）：在函数中使用关键字var声明的变量，只能在函数的内部访问 。

全局作用域：在函数的外部使用var声明的变量,可以在任意的位置访问。

①局部可访问全局变量，但全局不可访问局部变量

②在局部下可修改全局变量

③在函数内使用var声明的变量是局部变量，不使用var声明是全局变量

（2）变量的声明提升

js程序执行前，会将使用var声明的变量提升到所在作用域的最前边，赋值还是在原来的位置

//声明变量

/\*

console.log(a);

var a=1;

function fn(){

console.log(b)

var b=2;

}

fn()

\*/

/\*var c=3;

function fun(){

console.log(c);

c=5;

}

fun()\*/

//参数

/\*function myfun(n){

//相当于是函数内部的变量，只能函数内部使用

console.log(n)

}

myfun(7);

console.log(n);\*/

var d=5;

function foo(d){

//d始终是局部变量，不能被外部访问，形参在原来的基础之上加3，和全局的d没有关系

d=d+3;

//d=11

}

foo(8);

console.log(d);

11、函数的作用域

函数的可访问范围，也分为全局作用域和函数作用域

（1）函数声明的提升

和变量声明提升一样，js程序执行前，会将function创建的函数提升到所在作用域的最前边，调用可以在任意合法的位置

/\*var a=7;

function myfun(){

//var a=5;

function foo(){

//console.log(2);

console.log(a);

}

foo()

}

myfun();

//foo();\*/

12递归调用

在函数内部调用自身

递归要有结束的条件，结合return来使用。

0

//弹出三次警示框结束

var i=0;

function fn(n){

i++;

alert('long ago')

//当i为3的时候，结束

if (i==3)

{//结束函数，不是为了返回某个结果

return;

}

//在函数内部调用自身\_\_递归

fn();

}

fn();

斐波那契数列

1 1 2 3 5 8 13 21 34

n1 n2

n1 n2

n1 n2

a1=1 a

a1=1 a2=1 an=an-1+an-2

练习：复习内容，整理思维导图

创建函数，计算斐波那契数列的第n项；使用递归和普通函数两种方法完成。

递归：

/\*//获取斐波那契数列的第n 项

function fib(n){

//当n=1或者n=2的时候直接返回1

if (n==1||n==2)

{return 1;

}

//当前这一项的值等于前两项相加的和

return fib(n-1)+fib(n-2)

}

console.log(fib(10))

/\*fib(4)+fib(3)

fib(3)+fib(2)+fib(2)+fib(1)

fib(2)+fib(1)+fib(2)+fib(2)+fib(1)

\*/

普通函数表示：

function fib(n){

//首先第一项和第二项的值都是1

var n1=1;

var n2=2;

//如果要获取的第n项的值，需要从第三项开始，不断往后移动

for (var i=3;i<n ;i++ )

{//每移动一次，当前的n1的值为上一次n2的值，当前的n2为上一次n2+n2的值

var c=n1;

n1=n2;

n2=c+n2;

}

//当前n1的值就是这一项的值

return n2;

}console.log(fib(50));

13.匿名函数

没有名称的函数 function(){}

(1)创建函数

函数声明方式

function fn(){}

函数表达式

var fun=function(参数列表){函数体}

此时的变量fun就是函数名称

对比fun和 fun()的区别

fun()是函数的调用，返回函数的调用结果

fun是函数的名称，对应的是函数的整体结构

函数声明和函数表达式创建函数的区别

函数声明存在函数提升，调用可以在人已合法位置

函数表达式不存在函数提升，必须先创建，在调用

练习：使用函数表达式创建函数，计算任意两个数字之间所有整数的和，并返回结果

var getsum=function (a,b){

for (var i=a,sum=0 ;i<=b ;i++ )

{sum+=i

}

return sum;

}

console.log(getsum(2,15));

（2）匿名函数自调用

目的：创建局部函数作用域，防止污染全局

(function(形参列表){

函数体重的变量和创建的函数只能在内部使用

})(实参列表);

防止污染群全局

//轮播图

//匿名函数自调用

(function(a){

//局部变量，不会对全局造成污染

var num1=1;

console.log(a);

})(2);

//就买专辑

（3）回调函数

将匿名函数一实参形式传递，这个匿名函数就成为‘回调函数’；此时的形参就是函数名称，如果要调用匿名函数，只需要 形参名称

functiuon(a){

//a就是传递的匿名函数的名称

a();//执行传递的匿名函数函数体中的代码

}

fn(function(){…});

练习创建函数，传递两个参数，实参都是以匿名函数的行书传递，在匿名函数中分别返回数字；计算两个数字想家的和并打印出来

function fun(a,b){

var num1=a()

var num2=b()

console.log(num1+num2)};

fun((function(){

return 1}),function(){return 3;

})

14、全局函数

parseInt()将数据转会整型

parseFloat()将数据转回浮点型

encodeURI()对一个URL中的中文进行编码

decodeURI()对已经编码的URL进行解码

isNaN检测一个值是否为NaN，是———>true,不是——>false

//判断一个值是否为NaN

/\*console.log(isNaN(1+undefined));

console.log(1+true)\*/

isFinite()检测一个值是否为有限值是->true,不是->false

1/0 Infinity无穷 无限

//检测一个值是否为有限值

//console.log(isFinite(2/3))

eval（‘1+2’）求字符串中表达式的值

var url='http://www.codeboy.com/seach?keyword=电脑';

//对一个url中文进行编码

var str=encodeURI(url)//编码一个网址

console.log(str);

//对一个url进行解码

var str2=decodeURI(str);

console.log(str2)

练习：使用弹出提示框，输入一组运算，使用eval来执行该运算，并打印

var str=prompt('input' );

console.log(str);

alert(eval(str));

<script src='6-global.js'></script>

11对象

是应用类型数据

对象：是一组属性和方法（功能）的集合。

哪些是对象？

一个手机：属性：品牌、尺寸。方法：打电话，发短信

一个人：属性、年龄、姓名。方法：工作，说话

万物皆对象

（1）JS中的对象

内置对象：JS提供的

宿主对象：根据JS不同的执行环境来划分

自定义对象：自己创建的对象

属性：

方法：