# JavaScript基本语法

# JavaScript 是什么



JavaScript 是一种脚本，一门编程语言，它可以在网页上实现复杂的功能，网页展现给你的不再是简单的静态信息，而是实时的内容更新，交互式的地图，2D/3D 动画，滚动播放的视频等等。

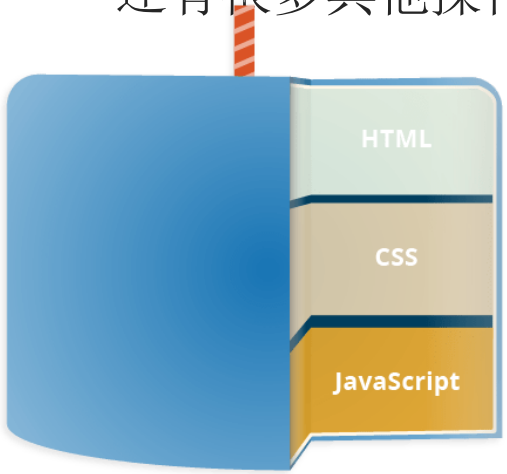
特点：

1. 一种解释性脚本语言（代码不进行预编译）。
2. 主要用来向 HTML 页面添加交互行为。
3. 可以直接嵌入 HTML 页面，但写成单独的 js 文件有利于结构和行为的分离。
4. 跨平台特性，在绝大多数浏览器的支持下，可以在多种平台下运行（如Windows、Linux、Mac、Android、iOS等）

# JavaScript 是什么



1. HTML是一种标记语言，用来结构化我们的网页内容并赋予内容含义，例如定义段落、标题和数据表，或在页面中嵌入图片和视频。
2. CSS 是一种样式规则语言，可将样式应用于 HTML 内容，例如设置背景颜色和字体，在多个列中布局内容。
3. JavaScript是一种脚本语言，可以用来创建动态更新的内容，控制多媒体，制作图像动画，还有很多其他操作。




Web技术三层蛋糕，这三层依次建立，秩序井然。

# JavaScript 是什么

JavaScript 相当简洁，却非常灵活。开发者基于 JavaScript 核心编写了大量实用工具，可以使开发工作事半功倍。包括：

1. 浏览器应用程序接口（API）—— 浏览器内置的 API 提供了丰富的功能，比如：动态创建 HTML 和设置 CSS 样式、从用户的摄像头采集处理视频流、生成 3D 图像与音频样本等等。
2. 第三方 API —— 让开发者可以在自己的站点中整合其他内容提供者提供的功能。
3. 第三方框架和库 —— 用来快速构建网站和应用。

浏览器 API 内建于 web 浏览器中，它们可以将数据从周边计算机环境中筛选出来，还可以做实用的复杂工作。例如：

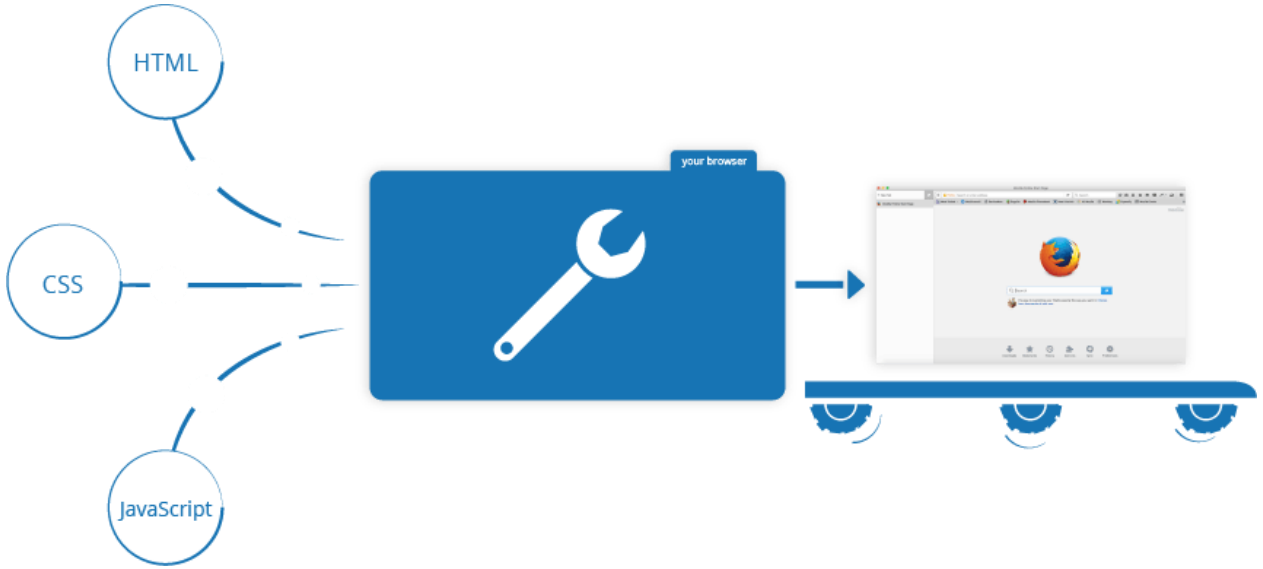
- 文档对象模型 API（DOM（Document Object Model）API） 能通过创建、移除和修改 HTML，为页面动态应用新样式等手段来操作 HTML 和 CSS。比如当某个页面出现了一个弹窗，或者显示了一些新内容，这就是 DOM 在运行。
- 地理位置 API（Geolocation API） 获取地理信息。这就是为什么谷歌地图可以找到你的位置，而且标示在地图上。
- 画布（Canvas） 和 WebGL API 可以创建生动的 2D 和 3D 图像。
- HTMLMediaElement 和 WebRTC 等影音类 API 让你可以利用多媒体做一些非常有趣的事，比如在网页中直接播放音乐和影片，或用自己的网络摄像头获取录像，然后在其他人的电脑上展示。

第三方 API 并没有默认嵌入浏览器中，一般要从网上取得它们的代码和信息。比如：

- Twitter API、新浪微博 API 可以在网站上展示最新推文之类。
- 谷歌地图 API、高德地图 API 可以在网站嵌入定制的地图等等

# JavaScript 在页面上做了什么？

- 浏览器在读取一个网页时，代码（HTML, CSS 和 JavaScript）将在一个运行环境（浏览器标签页）中得到执行。就像一间工厂，将原材料（代码）加工为一件产品（网页）。




- 在 HTML 和 CSS 集合组装成一个网页后，浏览器的 JavaScript 引擎将执行 JavaScript 代码。这保证了当 JavaScript 开始运行之前，网页的结构和样式已经就位。
- 因为 JavaScript 最普遍的用处是通过 DOM API动态修改 HTML 和 CSS 来更新用户界面（user interface）。如果 JavaScript 在 HTML 和 CSS 就位之前加载运行，就会引发错误。

# JavaScript示例

- Website演示
- JS文件中所用到的函数均属于文档对象模型（DOM）API

# 运行方式

1. 直接在浏览器的控制台编写运行
2. 编写一个独立的js文件，然后在html文件中引入
3. 用开发工具如 Webstorm 或 VSCode 编写独立的js文件，由安装好的 Node.js 解释执行

# JavaScript 是什么

## JavaScript的组成

- ECMAScript:

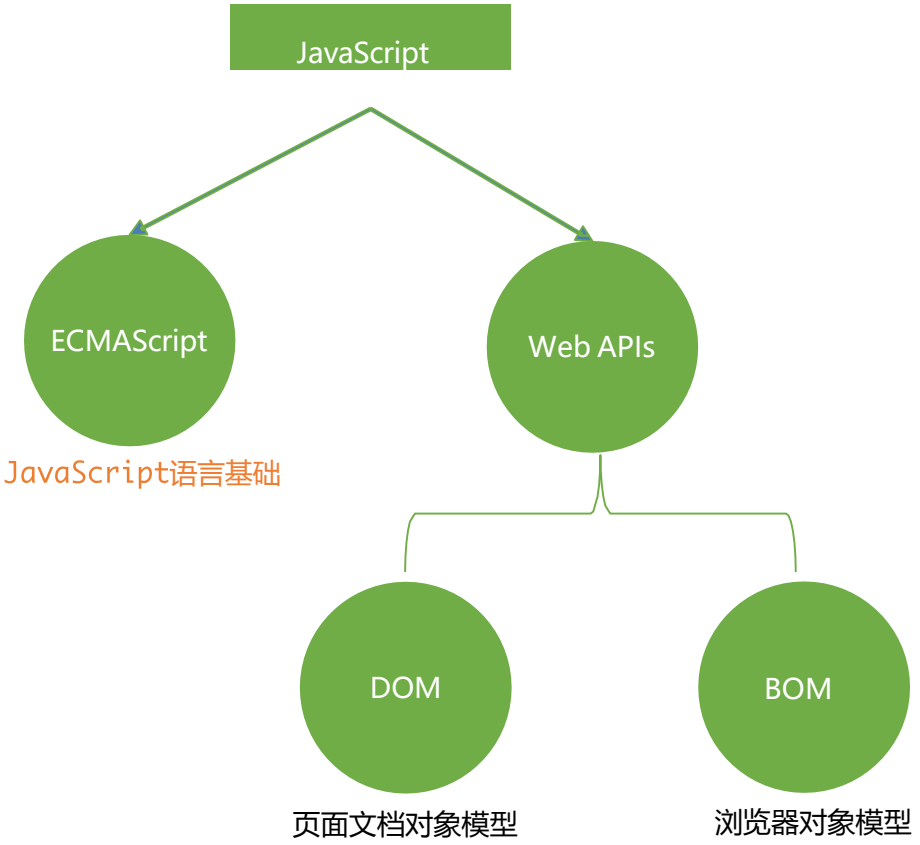
  规定了JS基础语法核心知识

  比如：数据类型、变量、分支语句、循环语句、对象等等。

- Web APIs:

  - DOM： 文档对象模型， 操作文档，比如对页面元素进行移动、大小、添加删除等操作
  - BOM： 浏览器对象模型， 操作浏览器，比如页面弹窗，检测窗口宽度、存储数据到浏览器等等

**权威网站： MDN**



# JavaScript 书写位置

# JavaScript 书写位置

## 1. 内部 JavaScript

直接写在html文件里，用script标签包住。

**规范**：script标签写在</body>上面。

拓展：alert(“Hello, JS”)页面弹出警告对话框。

**注意**：将 <script> 放在HTML文件的底部附近的原因是浏览器会按照代码在文件中的顺序加载 HTML。 如果先加载的 JavaScript 期望修改其下方的 HTML，那么它可能由于 HTML 尚未被加载而失效。

因此，将 JavaScript 代码放在 HTML页面的底部附近通常是最好的策略。

# JavaScript 书写位置

## 2. 外部 JavaScript

代码写在以. js 结尾的文件里

**语法：**通过script标签，引入到html页面中。

```
<body>
    <!-- 通过src引入外部js文件 -->
    <script src="my.js"></script>
</body>
```

**注意：**script标签中间无需写代码，否则会被忽略！

外部JavaScript会使代码更加有序，更易于复用，且没有了脚本的混合，HTML 也会更加易读，因此这是个好的习惯。

# JavaScript 书写位置

## 3. 内联 JavaScript

代码写在标签内部

**语法：**

```
<body>
    <button onclick="alert('Hi, 你好呀！')">点击一下</button>
</body>
```

# JavaScript 注释

## 两种JavaScript注释方法

1. 单行注释
   - 符号：//
   - 作用：//右边这一行的代码会被忽略
   - 快捷键：ctrl+/

2. 块注释
   - 符号：/* */
   - 作用：在/* 和 */之间的所有内容都会被忽略
   - 快捷键：shift+alt+A

```
<script>
  // 这种是单行注释的语法
  // 一次只能注释一行
  // 可以重复注释
</script>
```

```
<script>
  /* 这种的是多行注释的语法 */
  /*
     更常见的多行注释是这种写法
     在些可以任意换行
     多少行都可以
  */
</script>
```

# JavaScript 结束符

## JavaScript结束符

- 代表语句结束
- 英文分号;
- 可写可不写（现在不写结束符的越来越多）
- 换行符（回车）会被识别成结束符
- 因此在实际开发中有许多人主张书写 JavaScript 代码时省略结束符
- 但为了风格统一，要写结束符就每句都写，要么每句都不写

```
<script>
  alert(1);
  alert(2);
</script>
```

```
<script>
  alert(1)
  alert(2)
</script>
```

# JavaScript 输入输出语法

**语法：**人和计算机打交道的规则约定，要按照约定的规则去写，程序员需要操控计算机，需要计算机能看懂。

1. 输出语法：

```
document.write('要输出的内容');
```

- 向body内输出内容
- 如果输出的内容写的是标签，也会被解析成网页元素

```
alert('要输出的内容');
```

- 页面弹出警告对话框

```
console.log('控制台打印')
```

- 控制台输出语法，程序员调试使用

2. 输入语法：

```
prompt('请输入你的年龄：')
```

- 显示一个对话框，对话框中包含一条文字信息，用来提示用户输入文字

# JavaScript 输入输出语句

## 输入和输出练习

需求：

浏览器中弹出对话框： 你好~

页面中打印输出：自己的姓名

```
<body>
    <script>
      alert('你好~')
      let name = prompt('输入自己的姓名: ')
      document.write(name)
    </script>
  </body>
```

# 字面量

在计算机科学中，字面量（literal）是在计算机中描述事/物的。

比如：

- 1000 就是：数字字面量
- ’你好’就是：字符串字面量
- [ ]就是：数组字面量
- { }就是：对象字面量等等

# JavaScript 介绍小结

## 1. JavaScript是什么?

JavaScript是一门编程语言，可以实现很多的网页交互效果。

## 2. JavaScript 书写位置?

- 内联JavaScript
- 内部JavaScript写到</body>标签上方
- 外部JavaScript- -- 通过src引入html页面中，但是<script>标签不要写内容，否则会被忽略

## 3. JavaScript 的注释?

- 单行注释 //
- 多行注释 /* */

## 4. JavaScript 的结束符?

- 分号；  可以加也可以不加，可以按照团队约定
- 注意换行默认为结束符

## 5. JavaScript 输入输出语句?

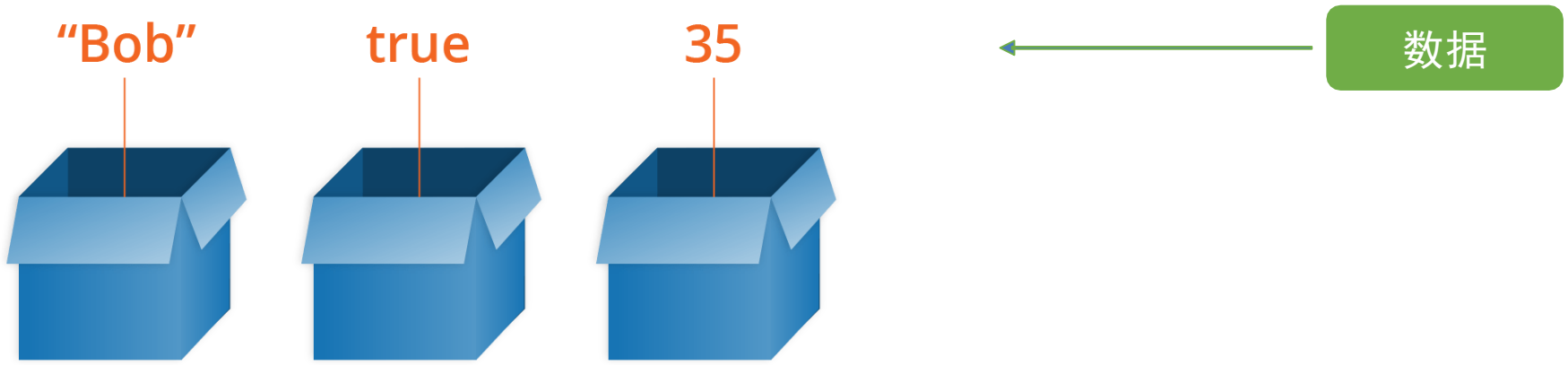
- 输入：prompt()
- 输出：alert()  document.write()  console.log()

# 变量（Variable）是什么?



**变量是存储值的容器。要声明一个变量，先输入关键字 let 或 var，然后输入合适的名称。** JavaScript 对大小写敏感，myVariable 和 myvariable 是不同的。

```
let myVariable; //变量声明
```




上述语句行末的分号表示当前语句结束，不过只有在单行内需要分割多条语句时，这个分号才是必须的。然而，一些人认为每条语句末尾加分号是一种好的风格

# 变量的基本使用

## 2. 变量赋值:

定义了一个变量后，你就能够初始化它（赋值）。

```
myVariable = '李雷';
```

也可以将定义、赋值操作写在同一行：

```
let myVariable = '李雷';
```

可以直接通过变量名取得变量的值：

```
myVariable;
```

变量在赋值后是可以更改的：

```
let myVariable = '李雷';
myVariable = '韩梅梅';
```

# 变量的练习

需求：

1.浏览器中弹出对话框： 请输入姓名

2.页面中输出：刚才输入的姓名

分析：

①：输入：用户输入框： prompt()

②：内部处理：保存数据

③：输出： 页面打印 document.write()

# 变量命名规则与规范

**规则：**必须遵守，不遵守报错

**规范：**建议，不遵守不会报错，但不符合业内通识

## 1. 规则：

- 不能用关键字
  - ✓ 关键字：有特殊含义的字符，JavaScript 内置的一些英语词汇。例如：let、var、if、for等
- 只能用下划线、字母、数字、$组成，且数字不能开头
- 字母严格区分大小写，如 Age 和 age 是不同的变量

## 2. 规范：

- 起名要有意义
- 遵守小驼峰命名法
  - ✓ 第一个单词首字母小写，后面每个单词首字母大写。例：userName

# 变量命名规则与规范

以下哪些是合法的变量名？

| 变量名 | 是否报错 | 是否符合规范 |
| --- | --- | --- |
| 21age | | |
| _age | | |
| user-name | | |
| username | | |
| userName | | |
| let | | |
| na@me | | |
| $age | | |

# 变量的练习

**需求**：让用户输入自己的名字、年龄、性别，再输出到网页

分析：

弹出输入框（prompt）：  请输入您的姓名：输入用变量保存起来；

弹出输入框（prompt）：  请输入您的年龄：输入用变量保存起来；

弹出输入框（prompt）：  请输入您的性别：输入用变量保存起来；

页面分别输出(document.write)刚才的3个变量。

```
<body>
    <script>
      let name = prompt('请输入您的姓名：')
      let age = prompt('请输入您的年龄：')
      let gender = prompt('请输入您的性别：')
      document.write('姓名：' + name + '，年龄：' + age + '，性别：' + gender)
    </script>
  </body>
```

# 小结


1. 为什么需要变量？
   - 变量无处不在，因为我们一些数据需要保存，所以需要变量
2. 变量是什么？
   - 变量就是一个容器，用来存放数据的。方便我们以后使用里面的数据
3. 变量的本质是什么？
   - 变量是内存里的一块空间，用来存储数据。
4. 变量怎么使用的？
   - 我们使用变量的时候，一定要声明变量，然后赋值
   - 声明变量本质是去内存申请空间。

# 变量声明-let和var的区别

**let 和 var 区别：**

let 为了解决 var 的一些问题。

var 声明：

- 可以先使用，再声明 (不合理)
- var 声明过的变量可以重复声明(不合理)
- 比如变量提升、全局变量、没有块级作用域等等

结论：

以后声明变量我们统一使用 let。

# 变量数据类型



| 变量 | 解释 | 示例 |
| --- | --- | --- |
| String | 字符串（一串文本）：字符串的值必须用引号（单双均可，必须成对）扩起来。 | `let myVariable = '李雷';` |
| Number | 数字：无需引号。 | `let myVariable = 10;` |
| Boolean | 布尔值（真 / 假）：`true` / `false` 是 JS 里的特殊关键字，无需引号。 | `let myVariable = true;` |
| Array | 数组：用于在单一引用中存储多个值的结构。 | `let myVariable = [1, '李雷', '韩梅梅', 10];`<br>元素引用方法：`myVariable[0]`，`myVariable[1]` ..... |
| Object | 对象：JavaScript 里一切皆对象，一切皆可储存在变量里。这一点要牢记于心。 | `let myVariable = document.querySelector('h1');`<br>以及上面所有示例都是对象。 |

# 数组的基本使用

```
let 数组名 = [数据1, 数据2, ..., 数据n]
```

let names=['张三','李四','刘七']

- 数组是按顺序保存，所以每个数据都有自己的编号
- 计算机中的编号从0开始，所以张三的编号为0，李四编号为1，以此类推
- 在数组中，数据的编号也叫**索引或下标**
- 数组可以存储任意类型的数据

# 数组的基本使用

## 取值语法

```
数组名[下标]
```

```
let names = ['张三', '李四', '刘七']
names[0]//张三
names[1]//李四
```

- 通过下标取数据
- 取出来是什么类型的，就根据这种类型特点来访问

```
let names = ['张三', '李四', '刘七']
console.log(names[0]) //张三
console.log(names[1]) //李四
console.log(names.length)//3
```

- 元素：数组中保存的每个数据都叫数组元素；
- 下标：数组中数据的编号；
- 长度：数组中数据的个数，通过数组的length属性获得；

# 数据类型

**JS 数据类型整体分为两大类：**

- 基本数据类型
- 引用数据类型

基本数据类型

Number 数字型
String 字符串型
Boolean 布尔型
undefined 未定义型
null 空类型

引用数据类型

Object 对象
Function 函数
Array 数组

# 数据类型 – 数字类型（Number）

整数、小数、正数、负数。

```
let score = 100; // 正整数
let price = 12.345; // 小数
let temperature = -40; // 负数
```

JavaScript 中的正数、负数、小数等统一称为 数字类型。

JS 是弱数据类型，变量到底属于那种类型，只有赋值之后，我们才能确认

Java是强数据类型 例如 int a = 3; 必须是整数

# 数据类型 – 字符串类型（String）

通过单引号（’’）、双引号（″″）或反引号(`)包裹的数据都叫字符串，单引号和双引号没有本质上的区别，推荐用单引号。

```
let user_name = '小明'; // 使用单引号
let gender = "男"; // 使用双引号
let str = '123'; // 看上去是数字，但是用引号包裹了就成了字符串了
let str1 = ''; // 这种情况叫空字符串
```

注意事项：

1. 无论单引号或是双引号必须成对使用
2. 单引号/双引号可以互相嵌套，但是不可以自己嵌套自己（口诀：外双内单，或者外单内双）
3. 必要时可以使用转义符 \，输出单引号或双引号

# 数据类型 – 字符串类型（String）

## 字符串拼接:

```
document.write( ‘Java’ +’ Script’ );


let a =  ‘J’ ;
let b =  ‘S’ ;
document.write(a + b);
```

# 模板字符串

## 1. 作用

- 拼接字符串和变量
- 在没有它之前，要拼接变量比较麻烦

```
document.write('大家好，我叫' + name + '，今年' + age + '岁')
```

## 2. 符号

- ``
- 在英文输入模式下按键盘的tab键上方那个键（1左边那个键）
- 内容拼接变量时，用 ${} 包住变量

```
document.write(`大家好，我叫${name}，今年${age}岁`)
```

# 小结


1. JavaScript中什么样数据我们知道是字符串类型？
   - 只要用 单引号、双引号、反引号包含起来的就是字符串类型
2. 字符串拼接比较麻烦，我们可以使用什么来解决这个问题？
   - 模板字符串， 可以让我们拼接字符串更简便
3. 模板字符串使用注意事项：
   - 用什么符号包含数据？
     - ✓ 反引号``
   - 用什么来使用变量？
     - ✓ ${变量名}

```
document.write(`Hello, my name is ${name}, I'm ${age} years old.`)
```

# 数据类型 – 布尔类型（Boolean)

表示肯定或否定时在计算机中对应的是布尔类型数据。

它有两个固定的值 true 和 false，表示肯定的数据用 true（真），表示否定的数据用 false（假）。

```
let flag = true
flag = false
```

布尔值（真 / 假）： true/false 是 JS 里的特殊关键字，无需引号。

# 数据类型 – 未定义类型（undefined）



未定义是比较特殊的类型，只有一个值 undefined。

**什么情况出现未定义类型?**

只声明变量，不赋值的情况下，变量的默认值为 undefined，一般很少【直接】为某个变量赋值为 undefined。

```
let age    // 声明变量但是未赋值
document.write(age)  // 输出 undefined
```

**工作中的使用场景:**

开发中经常声明一个变量，等待传送过来的数据。

如果我们不知道这个数据是否传递过来，此时可以通过检测这个变量是不是undefined，就能够判断用户是否有数据传递过来。

# 数据类型 –未定义类型（undefined）

未定义是比较特殊的类型，只有一个值 undefined。

总结使用情况如下：

| 情况 | 说明 | 结果 |
| --- | --- | --- |
| let age ; console.log (age) | 只声明不赋值 | undefined |
| console.log(age) | 不声明不赋值 直接使用 | 报错 |
| age = 10; console.log (age) | 不声明只赋值 | 10 （不提倡） |

# 数据类型 – null（空类型）

null 表示值为 空

```
let obj = null
```

**null 和 undefined 区别：**

1. undefined 表示没有赋值
2. null 表示赋值了，但是内容为空

**null 开发中的使用场景：**

将来有个变量里面存放的是一个对象，但是对象还没创建好，可以先给个null

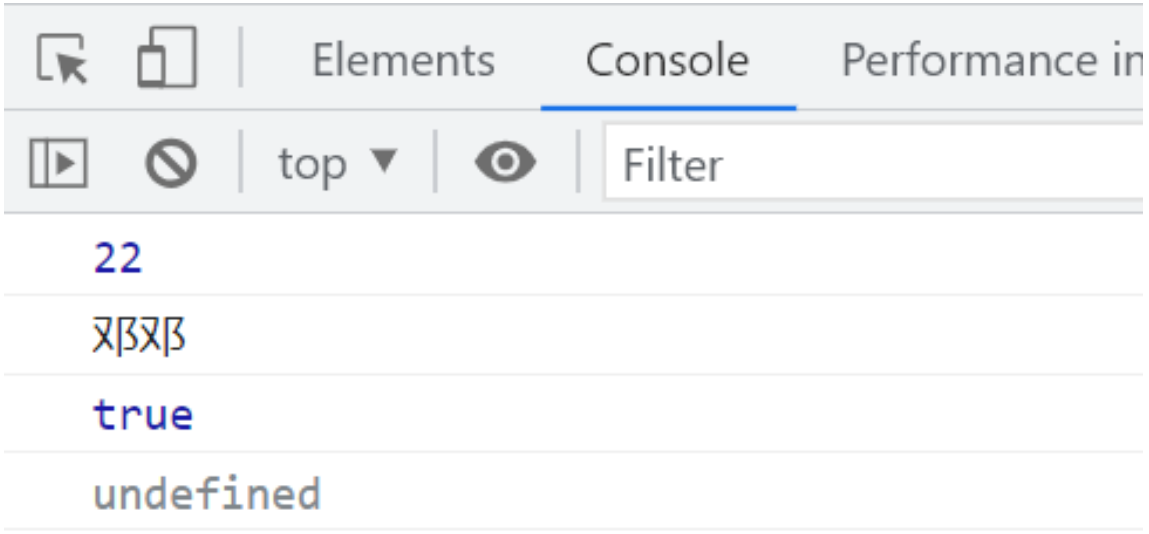
# 小结


1. 布尔数据类型有几个值？
   - true 和 false
2. 什么时候出现未定义数据类型？以后开发场景是？
   - 定义变量未给值就是 undefined
   - 如果检测变量是undefined就说明没有值传递过来
3. null是什么类型？ 开发场景是？
   - 空类型
   - 如果一个变量里面确定存放的是对象，如果还没准备好对象，可以放个null

# 控制台输出语句和检测数据类型

1. **控制台输出语句：**

```
let age = 22
let username = '邓邓'
let flag = true
let las
console.log(age)
console.log(username)
console.log(flag)
console.log(las)
```

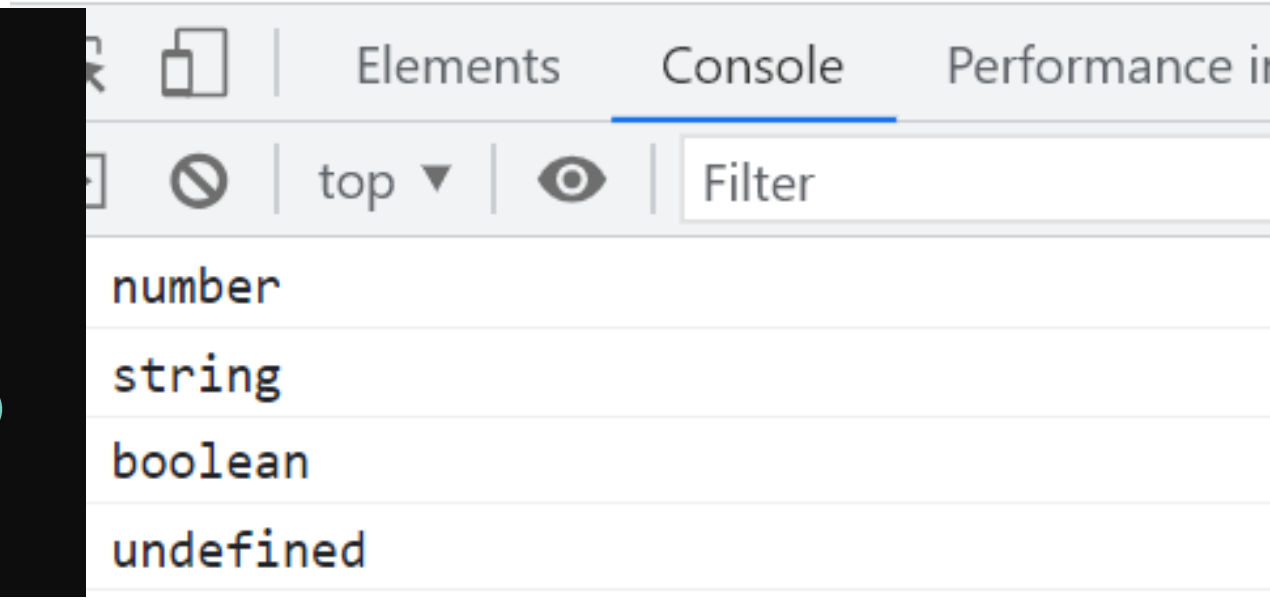



- 控制台语句经常用于测试结果来使用。
- 可以看出数字型和布尔型颜色为蓝色，字符串和undefined颜色为灰色

# 控制台输出语句和检测数据类型

## 通过 typeof 关键字检测数据类型

```
let age = 22
let username = '邓邓'
let flag = true
let las
console.log(typeof age)
console.log(typeof username)
console.log(typeof flag)
console.log(typeof las)
```



# 类型转换

## 为什么需要类型转换？

JavaScript是弱数据类型： JavaScript也不知道变量到底属于那种数据类型，只有赋值了才清楚。

使用表单、prompt 获取过来的数据默认是字符串类型的，此时就不能直接简单的进行加法运算。

```
console.log('10000' + '2000')  // 输出结果  100002000
```

此时需要转换变量的数据类型。

通俗来说，就是把一种数据类型的变量转换成我们需要的数据类型。

# 隐式转换

某些运算符被执行时，系统内部自动将数据类型进行转换，这种转换称为隐式转换。

**规则：**

- + 号两边只要有一个是字符串，都会把另外一个转成字符串
- 除了+以外的算术运算符 比如 - * / 等都会把数据转成数字类型

**缺点：**

- 转换类型不明确，靠经验才能总结

**小技巧：**

- +号作为正号解析可以转换成Number

```
<script>
    console.log(11 + 11)
    console.log('11' + 11)
    console.log(11 - 11)
    console.log('11' - 11)
    console.log(1 * 1)
    console.log('1' * 1)
    console.log(typeof '123')
    console.log(typeof +'123')
    console.log(+'11' + 11)
</script>
```

```
22
1111
0
0
1
1
string
number
22
```

# 显式转换

编写程序时过度依靠系统内部的隐式转换是不严禁的，因为隐式转换规律并不清晰，大多是靠经验总结的规律。为了避免因隐式转换带来的问题，通常根逻辑需要对数据进行显示转换。

## 转换为数字型

- Number(数据)
  - ✓ 转成数字类型
  - ✓ 如果字符串内容里有非数字，转换失败时结果为 NaN（Not a Number）即不是一个数字
  - ✓ NaN也是number类型的数据，代表非数字
- parseInt(数据)
  - 只保留整数
- parseFloat(数据)
  - 可以保留小数

## 转换为字符型:

- String(数据)
- 变量.toString(进制)

# 类型转换

**输入2个数，计算两者的和，打印到页面中**

```
<body>
    <script>
      let a1 = prompt('第一个数: ')
      let a2 = prompt('第二个数: ')
      let sum = Number(a1) + Number(a2)
      document.write(a1 + '+' + a2 + '=' + sum)
    </script>
  </body>
```

# JavaScript运算符



| 运算符 | 解释 | 符号 | 示例 |
| --- | --- | --- | --- |
| 加 | 将两个数字相加，或拼接两个字符串。 | `+` | `6 + 9;`<br>`"Hello " + "world!";` |
| 减、乘、除 | 这些运算符操作与基础算术一致。只是乘法写作星号，除法写作斜杠。 | `-`, `*`, `/` | `9 - 3;`<br>`8 * 2; //乘法在 JS 中是一个星号`<br>`9 / 3;` |
| 赋值运算符 | 为变量赋值（你之前已经见过这个符号了） | `=` | `let myVariable = '李雷';` |
| 等于 | 测试两个值是否相等，并返回一个 `true`/`false` （布尔）值。 | `===` | `let myVariable = 3;`<br>`myVariable === 4; //`<br>`false` |
| 不等于 | 和等于运算符相反，测试两个值是否不相等，并返回一个 `true`/`false` （布尔）值。 | `!==` | `let myVariable = 3;`<br>`myVariable !== 3; //`<br>`false` |
| 取非 | 返回逻辑相反的值，比如当前值为真，则返回 `false`。 | `!` | 原式为真，但经取非后值为 `false`：<br>`let myVariable = 3;`<br>`!(myVariable === 3); //`<br>`false` |

“3” + “5” = ?

3 + 5 = ?

# 运算符

- **算术运算符**
- **赋值运算符**
- **一元运算符**
- **比较运算符**
- **逻辑运算符**
- **运算符优先级**

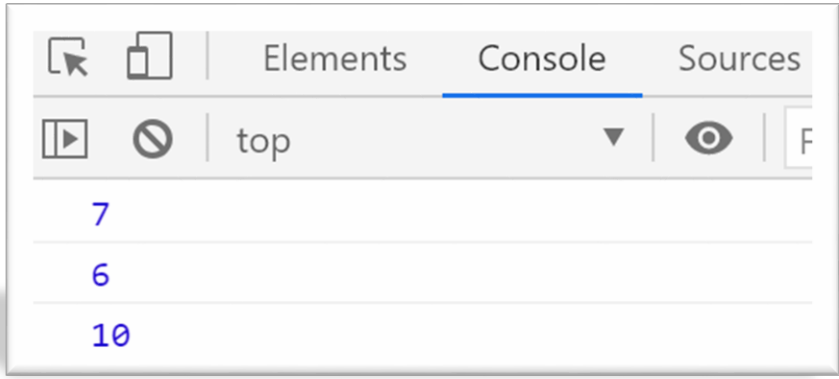
# 算术运算符

数学运算符也叫算术运算符，主要包括加、减、乘、除、取余（求模）。

- +：求和
- -：求差
- *：求积
- /：求商
- %：取模（取余数）

```
console.log(1 + 2 * 3)
console.log(10 - 8 / 2)
console.log(2 % 5 + 4 * 2)
```

优先级：先乘除后加减，有括号先算括号里面的。



# 赋值运算符

= ：将等号右边的值赋予给左边，要求左边必须是一个容器。

其他赋值运算符：

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

```
<script>
    let num = 1
    num += 1
    console.log(num)  // 结果是 2
</script>
```

使用这些运算符可以在对变量赋值时进行快速操作。

# 一元运算符

**自增：**

符号：++

作用：让变量的值 +1

**自减：**

符号：--

作用：让变量的值 -1

# 一元运算符

**前置自增：**

```
let num = 1
++num       //让num的值加 1变 2
```

每执行1次，当前变量数值加1
其作用相当于 num += 1

**后置自增：**

```
let num = 1
num++       //让num的值加 1变 2
```

每执行1次，当前变量数值加1
其作用相当于 num += 1

**前置自增和后置自增单独使用没有任何区别。**

# 一元运算符

**前置自增和后置自增如果参与运算就有区别了：**

- 前置自增：先自加再使用（记忆口诀：++在前，先加）

```
let i = 1
console.log(++i + 2)  //结果是 4
//  注意：i是 2
//  i先自加 1,变成2之后，在和后面的2相加
```

- 后置自增：先使用再自加（记忆口诀：++在后，后加）

```
let i = 1
console.log(i++ + 2)  //结果是 3
// 注意：   此时的 i是 1
// 先和2相加,先运算输出完毕后，i再自加是2
```

# 比较运算符

> ：　左边是否大于右边

<：　左边是否小于右边

>=：　左边是否大于或等于右边

<=：　左边是否小于或等于右边

==：　左右两边是否相等

===：左右两边是否类型和值都相等

!==：左右两边是否不全等

比较结果为boolean类型，即只会得到true或false

# 比较运算符

比较细节：

- 字符串比较，是比较的字符对应的ASCII码，从左往右依次比较。
- NaN不等于任何值，包括它本身
- 尽量不要比较小数，因为小数有精度问题
- 不同类型之间比较会发生隐式转换：
  - 最终把数据隐式转换转成number类型再比较
  - 所以开发中，如果进行准确的比较使用=== 或者 !==

# 比较运算符

**= 和 == 和 === 怎么区别？**

= 是赋值

== 是判断 只要求值相等，不要求数据类型一样即可返回true

=== 是全等 要求值和数据类型都一样返回的才是true

开发中，请使用===

# 逻辑运算符

| 符号 | 名称 | 特点 | 口诀 |
| --- | --- | --- | --- |
| && | 逻辑与 | 符号两边都为true 结果才为true | 一假则假 |
| \|\| | 逻辑或 | 符号两边有一个<br>true就为true | 一真则真 |
| ! | 逻辑非 | true变false<br>false变true | 真变假，假变真 |

逻辑运算符里的短路：只存在于 && 和 || 中，当满足一定条件会让右边代码不执行

| 符号 | 短路条件 |
| --- | --- |
| && | 左边为false就短路 |
| \|\| | 左边为true就短路 |

原因：通过左边能得到整个式子的结果，因此没必要再判断右边

运算结果：无论 && 还是 ||，运算结果都是最后被执行的表达式值，一般用在变量赋值

# 逻辑运算符

```
console.log(false && 20)  // false
console.log(5 < 3 && 20) // flase
console.log(undefined && 20) // undefined
console.log(null && 20) // null
console.log(0 && 20) // 0
console.log(10 && 20) // 20
```

```
console.log(false || 20)  // 20
console.log(5 < 3 || 20) // 20
console.log(undefined || 20) // 20
console.log(null || 20) // 20
console.log(0 || 20) // 20
console.log(10 || 20) // 10
```

# 逻辑运算符

## 示例：判断一个数是4的倍数，且不是100的倍数

需求：用户输入一个，判断这个数能被4整除，但是不能被100整除

分析：

①：用户输入

②：控制台： 是否能被4整除并且100整除

```
<body>
    <script>
      let num = prompt('请输入一个数字: ')
      if (num % 4 === 0 && num % 100 === 0) {
        alert('Yes!')
      } else {
        alert('No!')
      }
    </script>
  </body>
```

# 运算符优先级

掌握运算符优先级，能判断运算符执行的顺序

| 优先级 | 运算符 | 顺序 |
| --- | --- | --- |
| 1 | 小括号 | () |
| 2 | 一元运算符 | ++ -- ! |
| 3 | 算数运算符 | 先 * / % 后 + - |
| 4 | 关系运算符 | > >= < <= |
| 5 | 相等运算符 | == != === !== |
| 6 | 逻辑运算符 | 先 && 后 \|\| |
| 7 | 赋值运算符 | = |
| 8 | 逗号运算符 | , |

- 一元运算符里面的逻辑非优先级很高
- 逻辑与比逻辑或优先级高

# 运算符优先级

## 练习

```
let a = 3 > 5 && 2 < 7 && 3 == 4
console.log(a);

let b = 3 <= 4 || 3 > 1 || 3 != 2
console.log(b);

let c = 2 === "2"
console.log(c);

let d = !c || b && a
console.log(d);
```

答案是false，此时发生了逻辑与中断

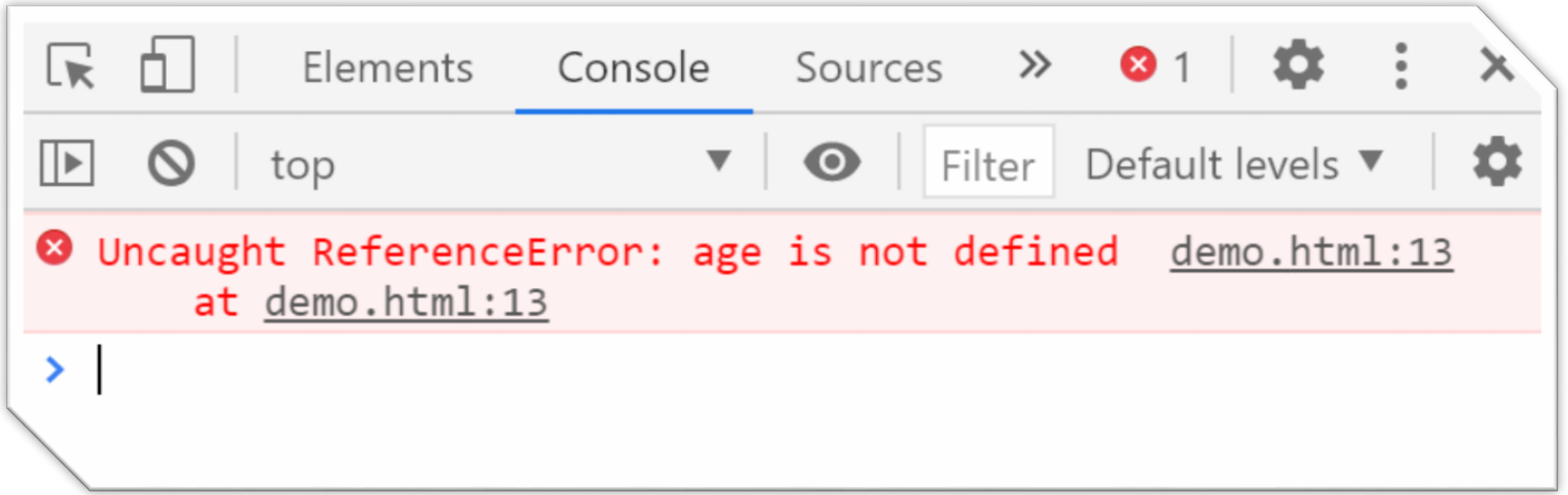
答案是true，此时发生了逻辑或中断

答案是false 数据类型不匹配

答案是true，此时发生了逻辑或中断

# 常见错误

1、下面可能出现的原因是什么？



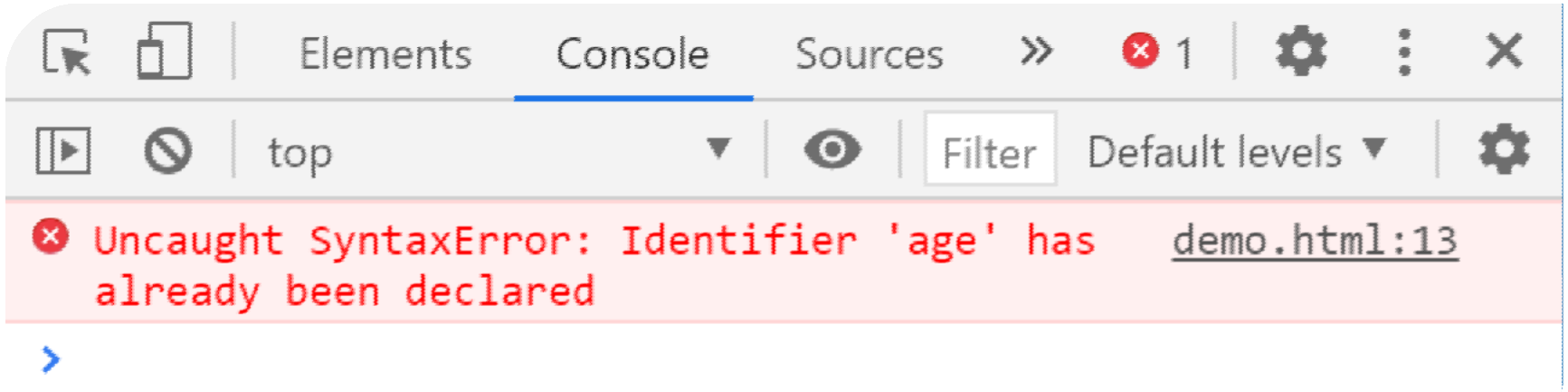

分析：

1. 提示 age变量没有定义过
2. 很可能 age 变量没有声明和赋值
3. 或者我们输出变量名和声明的变量不一致引起的。

```
10  <body>
11      <script>
12          let aeg = 10
13          console.log(age);
14      </script>
15  </body>
```

# 常见错误

2. 下面可能出现的原因是什么？



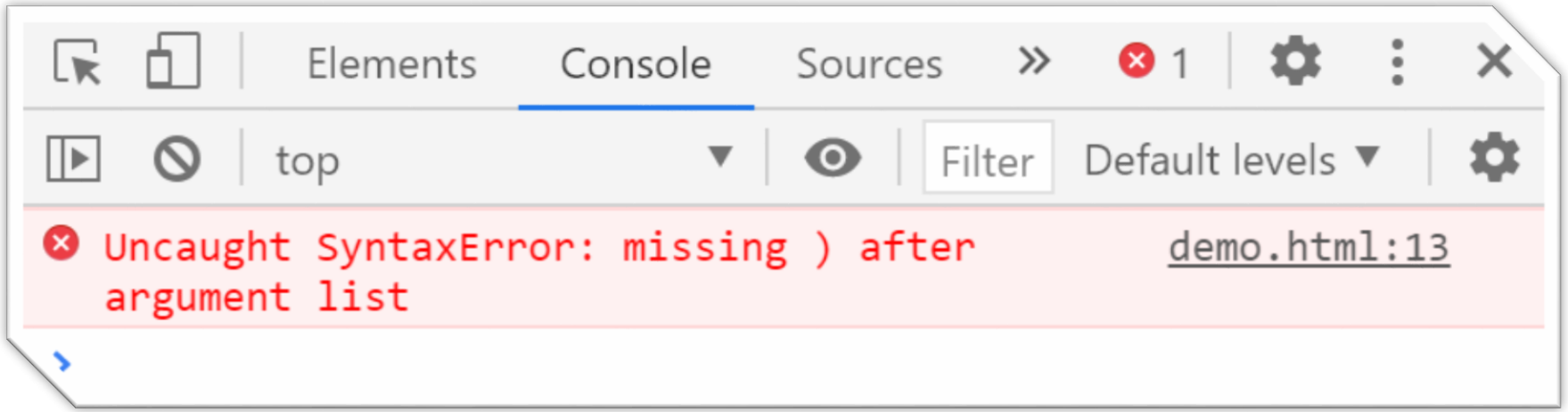

分析：

1. 提示 “age” 已经声明
2. 很大概率是因为使用let 重复声明了一个变量。
3. 注意let 变量不允许多次声明同一个变量

```
10  <body>
11      <script>
12          let age = 10
13          let age = 20
14      </script>
15  </body>
```

# 常见错误

3. 下面可能出现的原因是什么？




分析：

1. 提示 参数少了 ）
2. 很大概率是小括号不匹配

```
10  <body>
11      <script>
12          let age = 18
13          document.write(`我今年已经${age}岁啦`
14      </script>
15  </body>
```

# 常见错误

```
<body>
    <script>
        let num1 = prompt('请输入第一个数')
        let num2 = prompt('请输入第二个数')
        alert(num1 + num2)
    </script>
</body>
```

分析：

1. 出现字符相加的问题
2. prompt 如果出现相加，记得要转为数字型

# 总结

#js拥有动态类型

```
let x;  // 此时x是undefined
let x = 1;  // 此时x是数字
let x = "Alex"  // 此时x是字符串


# NaN 表示不是一个数字  （Not a Number）
```

数值Number

js不分整型与浮点型都是数值Number

# 总结

#常用方法

```
parseInt("123")  // 返回123
parseInt("ABC")  // 返回NaN,NaN属性是代表非数字值的特殊值。该属性用于指示某个值不是数字。
parseFloat("123.456")  // 返回123.456
```

字符串String

```
let a = "Hello"
let b = "world;
let c = a + b;
console.log(c);  // 得到Helloworld
```

# 总结

```
// 普通字符串
`这是普通字符串！`
// 多行文本
`这是多行的
文本`
// 字符串中嵌入变量
var name = "jason", time = "today";
`Hello ${name}, how are you ${time}?`
```

``是反引号，不是单引号

# 总结

#布尔值(Boolean)，区别于Python，true和false都是小写。

```
let a = true;
let b = false;
```

""(空字符串)、0、null、undefined、NaN都是false。

null和undefined

- null表示值是空，一般在需要指定或清空一个变量时才会使用，如 name=null;
- undefined表示当声明一个变量但未初始化时，该变量的默认值是undefined。还有就是函数无明确的返回值时，返回的也是undefined。