

## 浙江大学城市学院实验报告

课程名称 跨平台脚本开发技术

实验项目名称 实验一 JavaScript 简介

学生姓名 吴成洋 专业班级 软件工程 1404 学号 31401417

实验成绩          指导老师（签名）                      日期                 

注意：

- 务请保存好各自的源代码，以备后用。
- 请把作业保存为 pdf 上传到 BB 平台，请务必在截止日期前提交。

### 实验目的：

1. 熟悉 JavaScript 开发调试环境；
2. 掌握 JavaScript 的数据类型和基本流程控制。

### 实验内容：

#### 1、开发工具使用。（必做）

##### 1.1 安装开发工具 Node.js Chrome, VS Code

##### 1.2 node repl (Read Eval Print Loop) REPL 求值方式运行程序

`.help`

`.load prog.js`

`.exit`

##### 1.3 node 解释器方式运行教材上的你选任意程序,prog.js

`node prog.js`

##### 1.4 Chrome Devtools `ctrl+shift+j`

#### 2、阅读 JavaScript 指南相关内容（选做）

<https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Guide>

介绍

语法与数据类型

控制流与错误处理

循环与迭代

## 表达式和运算符

- 3、在 node.js 和 Chrome DevTool 平台上运行书中代码。（必做）
- 4、完成教材习题 p23-24 习题 1,2,3（必做）
- 5、阅读教材（前言，第 1 章，第 2 章）（必做）。结合像计算机科学家一样思考 python.pdf，说明下面的概念（选做）。

解释器 编译器

类型

求值

表达式

语句

语法错误

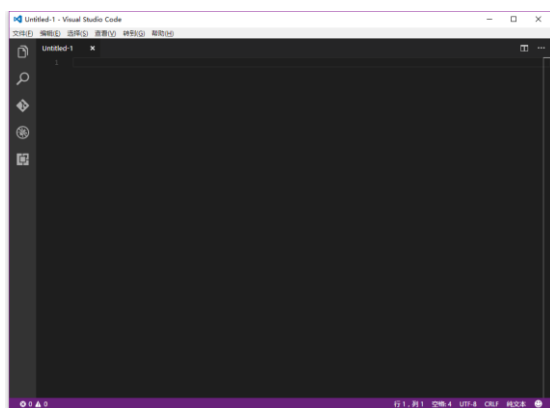
语义错误

- 6、给出代码，举例说明 JavaScript 是动态类型，弱类型。（选做）

## 实验步骤：

### 1、

VS Code



Node.js

```
C:\Windows\system32\cmd.exe
Microsoft Windows [版本 10.0.14393]
(c) 2016 Microsoft Corporation. 保留所有权利。

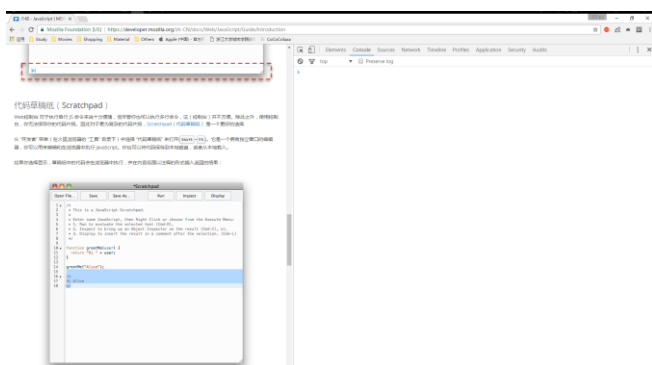
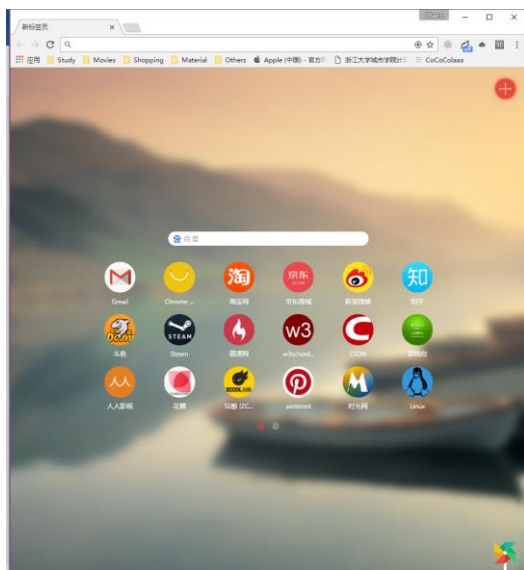
C:\Users\wcy62>npm -v
3.10.10

C:\Users\wcy62>node -v
v6.10.0

C:\Users\wcy62>node -h
Usage: node [options] [-e script | script.js] [arguments]
       node debug script.js [arguments]

Options:
  -v, --version          print Node.js version
  -e, --eval script       evaluate script
  -p, --print             evaluate script and print result
  -c, --check            syntax check script without executing
  -i, --interactive       always enter the REPL even if stdin
                        does not appear to be a terminal
  -r, --require module   module to preload (option can be repeated)
  --no-deprecation        silence deprecation warnings
  --trace-deprecation     show stack traces on deprecations
  --throw-deprecation     throw an exception when a deprecated function is used
  --no-warnings           silence all process warnings
  --trace-warnings        show stack traces on process warnings
  --trace-sync-io         show stack trace when use of sync IO
                        is detected after the first tick
  --trace-heap-objects   track heap object allocations for heap snapshots
                        process v8 profiler output generated
```

## Chrome



## 2.

介绍：JavaScript 是一门跨平台、面向对象的轻量级脚本语言。在宿主环境（例如 web 浏览器）中，JavaScript 能够通过其所连接的环境提供的编程接口进行控制。

语法与数据类型：JavaScript 语言可以识别下面 7 种不同类型的值：

六种 原型 数据类型:

Boolean. 布尔值, true 和 false.

null. 一个表明 null 值的特殊关键字。JavaScript 是大小写敏感的, 因此 null 与 Null、NULL 或其他变量完全不同。

undefined. 变量未定义时的属性。

Number. 表示数字, 例如: 42 或者 3.14159。

String. 表示字符串, 例如: "Howdy"

Symbol ( 在 ECMAScript 6 中新添加的类型)。一种数据类型, 它的实例是唯一且不可改变的。

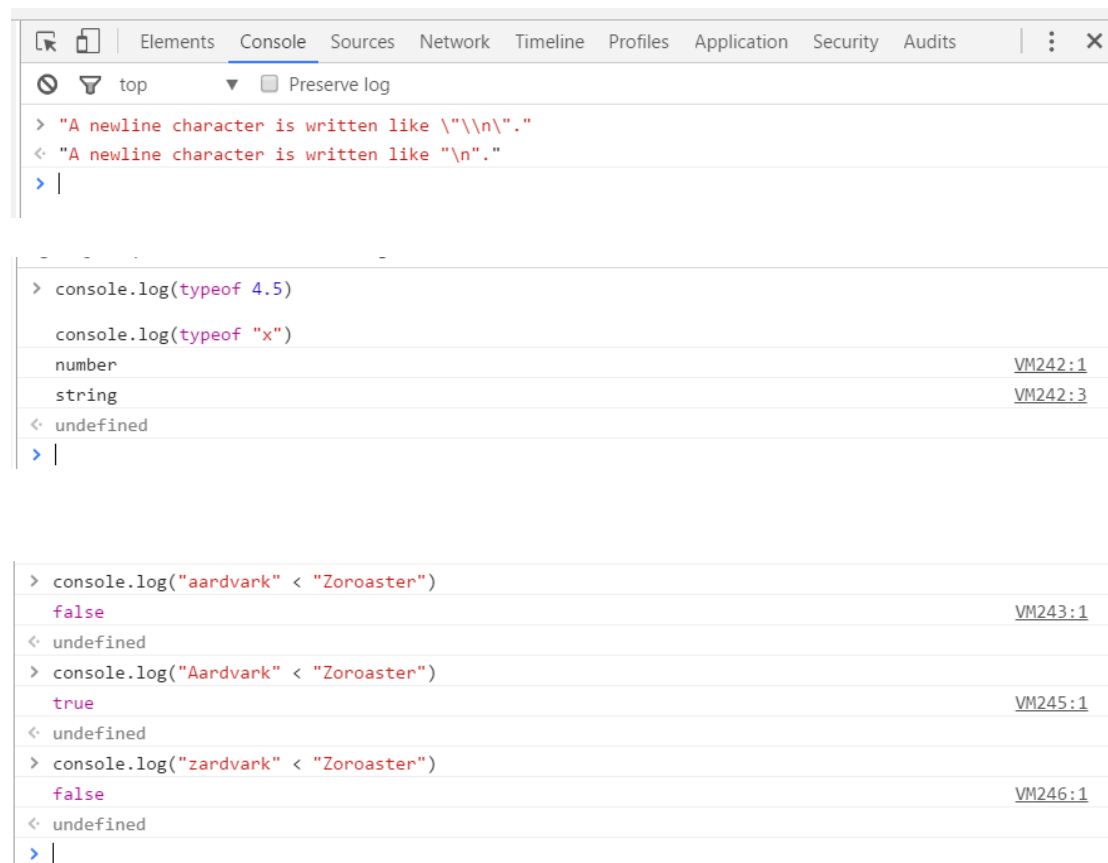
以及 Object 对象

控制流与错误处理

循环与迭代

表达式和运算符

### 3.



```
Elements Console Sources Network Timeline Profiles Application Security Audits
top ▼ Preserve log

> "A newline character is written like "\\n\"."
< "A newline character is written like "\\n\"."
> |

- - - - -

> console.log(typeof 4.5)
console.log(typeof "x")
number VM242:1
string VM242:3
< undefined
> |

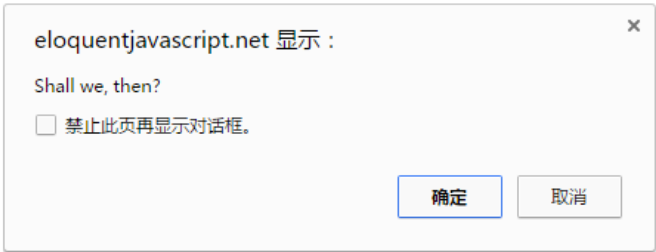
- - - - -

> console.log("aardvark" < "Zoroaster")
false VM243:1
< undefined
> console.log("Aardvark" < "Zoroaster")
true VM245:1
< undefined
> console.log("zardvark" < "Zoroaster")
false VM246:1
< undefined
> |
```

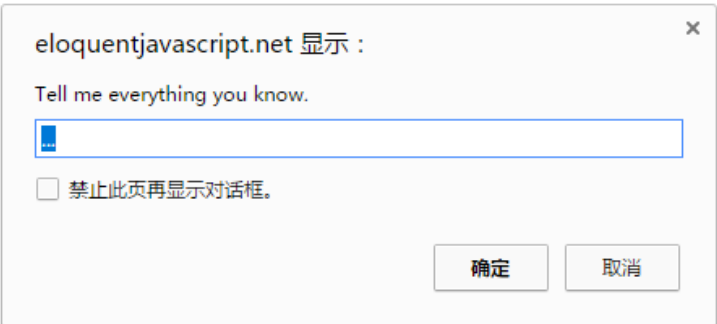
> console.log(NaN == NaN)	
false	VM248:1
< undefined	
>	
<div>🔍 📄   Elements Console Sources Network Timeline Profiles Application Security Audits   ⋮ ✕</div> <div>🚫 🔍 top ▼ <input type="checkbox"/> Preserve log</div>	
> console.log(true && false)	
console.log(false    true)	
console.log(false    false)	
console.log(true && true)	
false	VM252:1
true	VM252:2
false	VM252:3
true	VM252:4
< undefined	
>	
> console.log(true ? 1 : 2);	
1	VM254:1
< undefined	
> console.log(1+1==2 ? 1 : 2);	
1	VM265:1
< undefined	
>	
> console.log(8 * null)	
console.log("5" - 1)	
console.log("5" + 1)	
console.log("five" * 2)	
console.log(false == 0)	
0	VM267:1
4	VM267:3
51	VM267:5
NaN	VM267:7
true	VM267:9
< undefined	
>	
<div>— — — — —</div>	
> console.log(null == undefined);	
console.log(null == 0);	
true	VM269:1
false	VM269:3
< undefined	
>	

```
> console.log(Math.max(2, 4));  
4 VM399:1  
< undefined  
> console.log(Math.min(2, 4));  
2 VM400:1  
< undefined  
>
```

```
> confirm("Shall we, then?");  
>
```



```
> prompt("Tell me everything you know.", "...");  
>
```



```
> var result = 1;  
var counter = 0;  
while (counter < 10) {  
  result = result * 2;  
  counter = counter + 1;  
}  
console.log(result);  
1024 VM510:7  
< undefined  
>
```

```
> var number = 0;
  while (number <= 12) {
    console.log(number);
    number = number + 2;
  }
```

0	VM508:3
2	VM508:3
4	VM508:3
6	VM508:3
8	VM508:3
10	VM508:3
12	VM508:3
< 14	
>	

#### 4. 习题 1

```
var num = "";

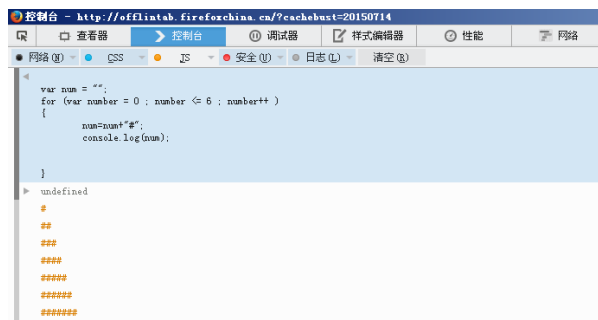
for (var number = 0 ; number <= 6 ; number++ )

{

    num=num+"#";

    console.log(num);

}
```



#### 4. 习题 2

```
for (var number = 1 ; number <= 100 ; number++ )

{

    if ( number % 3 == 0)

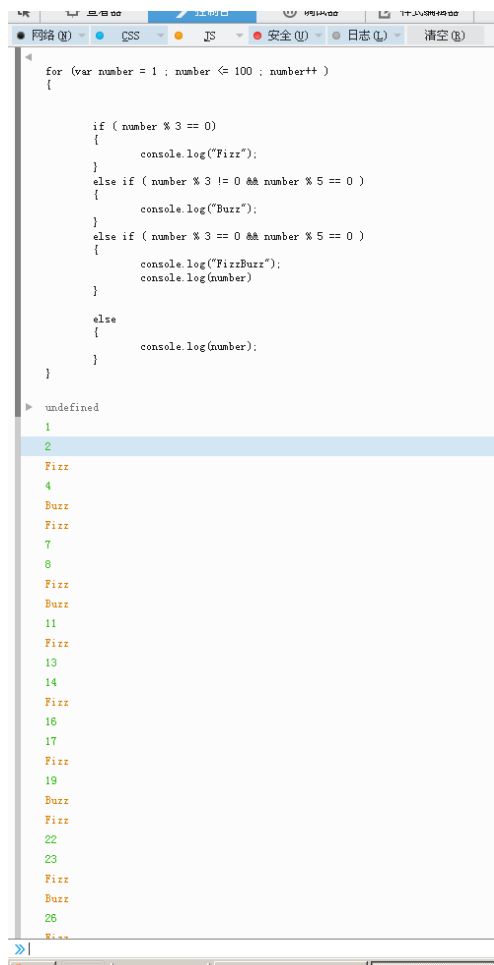
    {

        console.log("Fizz");

    }

    else if ( number % 3 != 0 && number % 5 == 0 )
```

```
{  
    console.log("Buzz");  
}  
else if ( number % 3 == 0 && number % 5 == 0 )  
{  
    console.log("FizzBuzz");  
    console.log(number)  
}  
else  
{  
    console.log(number);  
}  
}
```



```
for (var number = 1 ; number <= 100 ; number++ )  
{  
    if ( number % 3 == 0 )  
    {  
        console.log("Fizz");  
    }  
    else if ( number % 3 != 0 && number % 5 == 0 )  
    {  
        console.log("Buzz");  
    }  
    else if ( number % 3 == 0 && number % 5 == 0 )  
    {  
        console.log("FizzBuzz");  
        console.log(number)  
    }  
    else  
    {  
        console.log(number);  
    }  
}
```

undefined

1

2

Fizz

4

Buzz

Fizz

7

8

Fizz

Buzz

11

Fizz

13

14

Fizz

16

17

Fizz

Buzz

Fizz

22

23

Fizz

Buzz

26

...



#### 4. 习题 3

```
var size = 8;

var print = "";

for ( var y = 0 ; y < size ; y++ ){

    for( var x = 0 ; x < size ; x++ ){

        if( (x+y) % 2 == 0 )

            print = print + " " ;

        else

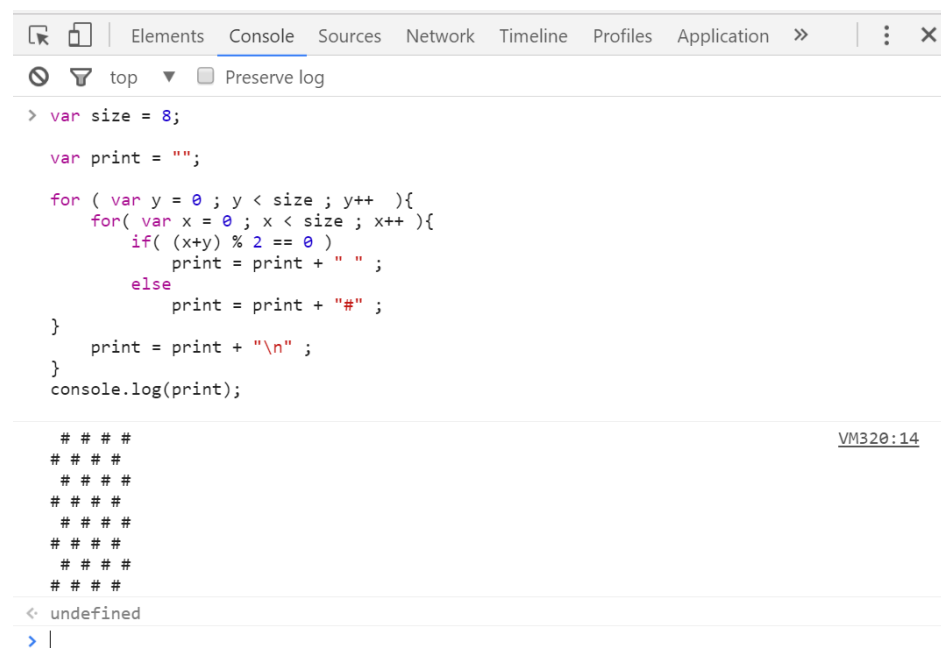
            print = print + "#" ;

    }

    print = print + "\n" ;

}

console.log(print);
```



```
> var size = 8;

var print = "";

for ( var y = 0 ; y < size ; y++ ){
  for( var x = 0 ; x < size ; x++ ){
    if( (x+y) % 2 == 0 )
      print = print + " " ;
    else
      print = print + "#" ;
  }
  print = print + "\n" ;
}
console.log(print);

# # # #
# # # #
# # # #
# # # #
# # # #
# # # #
# # # #
# # # #

VM320:14
<> undefined
> |
```

#### 5. 已读

6. JavaScript 是动态类型，弱类型，下面是对于相关概念的介绍。

强类型：偏向于不容忍隐式类型转换。譬如说 `haskell` 的 `int` 就不能变成 `double`

弱类型：偏向于容忍隐式类型转换。譬如说 `C` 语言的 `int` 可以变成 `double`

静态类型：编译的时候就知道了每一个变量的类型，因为类型错误而不能做的事情是语法错误。

动态类型：编译的时候不知道每一个变量的类型，因为类型错误而不能做的事情是运行时错误。譬如说你不能对一个数字 `a` 写 `a[10]` 当数组用。

弱类型：

```
> "1"+2  
'12'
```

动态类型

```
>>> a = 1  
>>> type(a)  
<type 'int'>  
>>> a = "s"  
>>> type(a)  
<type 'str'>
```