学习总结

对两种学习模式的看法

来兄弟会已经有一个月了，现在我对这一个月的情况做一个总结。我是从线下班来的兄弟会，之前在线下班也是学习了 一个月，我先对两种模式做一个比较，在来兄弟连之前，接触编程不多，在线下班的学习中，每天老师都会将很多的东 西，老师通过屏幕敲一遍，然后给我们时间，让我们自己练习一遍，刚开始学习的内容比较简单，我还能跟的上老师的 节奏，当内容开始变难以后，感觉跟起来就有点吃力了，每天接受大量的知识，然而总感觉自己消化不了；兄弟会是另 一种学习模式，我们先在教官那里领取一个学习任务，然后通过自己查阅资料，搜集相关信息，然后自己解决问题，将 学习任务完成，然后再找教官，提交任务。我是比较倾向于兄弟会这种学习模式的，因为他避免了我在学习上陷入之前 的那种前面的知识没有完全学明白，后面的知识又大量涌来的恶性循环当中，在兄弟会，即使我学的比较慢，但每接受 一个新任务都是建立在之前内容学懂的基础之上的。知识相互之间是有联系的，当你前一个环节学明白了，后面的知识 学起来也会更容易一些。

对小组分享的感受

两天前轮到我们小组进行技术分享，我们小组分享的是数据结构和算法相关的知识，这是我之前没有接触过的东西，但 小组内有明确的分工，我需要讲一部分和栈有关的算法知识，为此我花了很多时间来准备，我首先确定了自己要讲什么， 然后查找相关的资料，便开始学习这些相关内容，在自己学会之后，我开始想到时候该怎么讲，该如何将自己要讲的内 容和小组的主题关联起来，在正式讲之前，一直感觉自己没有准备好，心里没底，等到正式讲的时候，很流畅的就讲下 来了。有了这次的经验，我相信在下次的分享中，我能够把自己要分享的东西讲的更明白，更流畅。

个人分享

每天下午，都有一个学员分享的环节，在这个环节中，每个学员都会分享一些关于技术方面的问题，可能是一些自己踩 过的坑，也可能是一些自己感觉非常有用的知识点，由于我的进度比较慢一点，其他人的分享总是可以给我之后的任务 给出一些或多或少的建议，我也会努力做好自己的分享。

接下来的打算

这两天看了看这段时间敲的代码，最大的感受就是敲的代码还是太少了，理解固然重要，但大量的练习也必不可少，在 之后的学习过程中，首先应该解决敲代码量的问题。

---

markdown语法学习

刻意练习

1.要有明确的目标（例：游戏技巧练习）

2.要有好的导师（例：练习弹钢琴，打球）

打破原来的思维

1.飞机加钢板的问题

2.利用手表找正北的问题

新了解的概念

1.数据的熵（混乱程度）

2.比特

当天学习的问题

1.阅读文档的效率过低

2.对各个软件的熟悉程度过低

今后学习模式

1.将每天分为几个部分，明确每部分的学习目标

2.每个部分学习结束之后，及时做这部分学习的总结并记录

3.解决小阶段遇到的问题

4.对整天的学习作总结，

学习过程中的问题

1.学习的目的性还是不够明确，对任务的理解还有待提高。 在看视频的过程中 ，感觉只是在刷，这样的效果不是很理想，在以后的类似学习中，必须要跟着视频中的内容，一起练习。

2.关于提交任务的问题。 来兄弟会也有两天时间了，目前还没有主动领取过学习任务，在这方面必须要想办法加强，现阶段的学习过程中，应该将理解各工具的工作原理放在首位，其次就是常用的命令，命令应该边练习边记忆。每次领取任务后，应该给自己定一个最后提交期限，时间到了就去提交，即使提交失败，也可以及时得到建议。

3.关于交流的问题 由于这两天没有提交任务，与教练的接触较少，今天旁观了同桌提交任务，发现教练目前提问的针对性主要在这两天学习内容的原理上，这一方面我也应该着重注意。

4.关于学习方法 之前已经学习了一个月的java se，这两天学习的内容有些地方都是相通的，不能边学边忘，应该将所学的东西与之前的知识联系起来。

5.技术方面的问题 最近两天由于自己学习侧重点的问题，没有遇到这方面的问题。

6.关于markdown书写的问题 应该给自己每天书写markdown的内容，格式做一个合理的规划，这将很大的提升书写markdown的效率。

```
#include<stdio.h>

int max(int a, int b, int c)
{
    if(a>b&&a>c){
      return a;

    }else if(b>c){
      return b;
    }else{
      return c;


    }


}

int main()
{
    int a1,b1,c1;
    scanf("%d\n",&a1);
    scanf("%d\n",&b1);
    scanf("%d\n",&c1);

    printf("%d\n",max(a1,b1,c1));
     return 0;

}
```

问题

上述程序在运行过程键盘输入阶段，输入三个参数之后，必须再输入一个字符，然后再enter，程序才可以执行；

注意

p 与 *p 的区别；

```
#include<stdio.h>

int main()
{
     int a;
     scanf("%d",&a);
     int b;
     scanf("%d",&b);
     int c;
     scanf("%d",&c);

     if(a>b&&a>c){
        printf("%d\n",a);
     } else if(b>c){
        printf("%d\n",b);
     }
     else{
         printf("%d\n",c);
     }

     return 0;


  }
```

今天对昨天的代码进行了改造，昨天的问题得到了解决，问题出在哪呢？ 问题就处在%d后面的\n 上，当最后输入c的值后，\n自动切换到了下 一行，此时按下回车键，程序不能立即执行。

第一阶段任务总结

1.第一阶段的任务完成了，这一阶段的内容不是很难，我却用了很长的时间。我认为主要

原因还是目的性不强。

2.在提交任务的过程中，我了解到绝对路径，相对路径，linux的常用命令，vim的常用操作等是这一部分的重点，以后要继续练习。

第二个任务

第二个任务是JavaScript的学习，目前对我对这方面的知识还没有任何的了解和接触，接下来我需要通过看视频，查阅书籍等一切手段，尽快了解其相关运行实现机制，为自己掌握前端打好基础。

纯小白学习JavaScript

有关英语词汇的积累

由于编写程序主要用英文，所以，掌握一些常用的英语词汇是必要的。 如果这样做了，将会有许多好处， 尤其是在帮助自己学习理解编程语言方面。

html

html 是网页内容的载体，也是完成JavaScript学习的第一步，必须要首先掌握。 html 指超文本标签语言。 html 是通向 WEB 技术世界的钥匙。 html 不是一种编程语言，而是一种标记语言 标记语言是一套标记标签 html 使用标记标签来描述网页

html 的格式

```
<html>
<body>
<head>
</head>
<h1>标题</h1>

<p>段落</p>
</body>
</html>
```

CSS

CSS 是样式的表现

JavaScript

JavaScript 是用来实现网页上的特效处理

请不要把字符串、数值和布尔值声明为对象！

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<script >
    function validateForm() {

    var x = document.forms["myForm"]["fname"].value;


    if (x == "") {
        alert("邮箱必须填写! ");
        return false;
    }
    else {
        var y=x.indexOf("@");
        var yy=x.indexOf(".com")
        var z=x.substring(0,y);
        var tian=x.substring(y+1,yy);
        var txt = { "qq": {"name": "腾讯"},

                    "123": {"name": "abc"},

                    "163": {"name": "网易"}
        }



    }
    alert(txt[tina]);
}
</script>
</head>
<body>
<form name="myForm" onsubmit="return validateForm()" method="post">
邮箱:
<input type="text" name="fname">
<button>tijiao</button>
</form>
</body>
</html>
```

上述代码的作用

1.在表单中输入邮箱号，判断是邮箱是什么邮箱，属于什么公司； 2.上述代码存在的问题是将字符串声明为了对象。这将导致程序的可扩展性受到限制。

邮箱判断

<!DOCTYPE HTML>

邮箱:

时间戳代码

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>yshijianchuo</title>
<script type="text/javascript">



</script>
<style type="text/css">

#aaa{

   color: red;
   height: 400px;
   font-size: 50px;
}
#bbb{

   color: blue;
   height: 400px;
   font-size: 50px;
}

</style>
</head>
<body >
<p id="aaa"></p>
<p id="bbb"></p>
<script >
var shijangchuo= new Date();
```

```
document.getElementById('aaa').innerHTML=shijangchuo;
document.getElementById("bbb").innerHTML = shijangchuo.getTime();

</script>


</body>
</html>
```

正则判断手机号

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<script >
    function validateForm() {
        var x = document.forms["myForm"]["fname"].value;
        var patt1=/\d/g;
        var yyy =x.match(patt1);
        var z=x.substring(0);
        var xxx=z.split("");
        if(xxx.length==11){
           alert(xxx);
        if(yyy[0]==1&&yyy.length==11){
            alert("号码正确");
        }
        if(yyy[0]!=1||yyy.length!=11){
           alert("号码错误");
        }
        }else{
           alert("重新输入");
        }
    }
</script>
</head>
<body>
<form name="myForm" onsubmit="return validateForm()" method="post">
手机号:
<input type="text" name="fname">
</form>
</body>
</html>
```

。

这个任务完成的较为轻松，用的时间也不多，原因是教官明确告诉我用正则，如果知道了用哪些知识来完成一项任务，那么完成任务的过程是很容易的，以后作任务的时候，可以多想要用到哪些知识，这样可以提高学习进度。

服务器创建

```
'use strict';

var
    fs = require('fs'),
    url = require('url'),
    path = require('path'),
    http = require('http');

var workDir = path.resolve('.');

// 组合完整的文件路径
var filePath = path.join(workDir, 'pub', 'index.html');

// 从命令行参数获取root目录，默认是当前目录:
var root = path.resolve(process.argv[2] || '.');

console.log('Static root dir: ' + root);

// 创建服务器:
var server = http.createServer(function (request, response) {
    // 获得URL的path
    var pathname = url.parse(request.url).pathname;
    // 获得对应的本地文件路径
    var filepath = path.join(root, pathname);
    // 获取文件状态:
    fs.stat(filepath, function (err, stats) {
        if (!err && stats.isFile()) {
            // 没有出错并且文件存在:
            console.log('200 ' + request.url);
            // 发送200响应:
            response.writeHead(200);
            // 将文件流导向response:
            fs.createReadStream(filepath).pipe(response);
        } else {
            // 出错了或者文件不存在:
            console.log('404 ' + request.url);
            // 发送404响应:
            response.writeHead(404);
            response.end('404 Not Found');
```

```
        }
    });
});

server.listen(8080);

console.log('Server is running at http://127.0.0.1:8080/');
```

矩形的绘制

```
'use strict';
var fs = require('fs');
var k = 10;
var  t=  10;
var f = "%";
var q=1;
test(k,t,f);

function test(k,t,f) {

    if(q==1){
        for(var i = 1;i<=k;i++){

            for(var j = 1; j<=t;j++){
            process.stdout.write("    " + f);
            }

         console.log();
        }
    }
    if (q!=1){
        for(var j = 1; j<=t;j++){
            process.stdout.write("    " + f);
            }
            console.log();
            for(var i = 1;i<=k-2;i++){
                process.stdout.write("    " + f);
                for(var j = 1; j<=t-2;j++){
                    process.stdout.write("        " );
                }
            process.stdout.write("     " + f);
            console.log();
            }
         for(var j = 1; j<=t;j++){
            process.stdout.write("    " + f);
         }
     }
}
```

回形的绘制

```
'use strict';
var a=8;//长
var b=8;//宽
var c= 4;//内长
var d =4;//内宽
var f ='.';
test (a,b,c,d,f);
function test(a,b,c,d,f){
    for(var i=1;i<b+1;i++){
        if(i==1||i==b){//  1  ,  8

            for(var j=1;j<a+1;j++){
                process.stdout.write(' '+f);
            }
        }
        else{
            process.stdout.write(' '+f);
            if(i==(b-d)/2+1||i==(b-d)/2+d){ //nei   s   x
                for(var q=1;q<(a-c-2)/2+1;q++){
                    process.stdout.write('  ');
                }
                for(var w=1;w<c+1;w++){
                process.stdout.write(' '+f);
                }
                for(var e=1;e<(a-c-2)/2+1;e++){
                    process.stdout.write('  ');
                }
                process.stdout.write(' '+f);
            }
            if(i>(b-d)/2+1&&i<(b-d)/2+d){//nei   z
                for(var q=1;q<(a-c-2)/2+1;q++){
                    process.stdout.write('  ');
                }
                process.stdout.write(' '+f);
                for(var w =1;w<c-1;w++){
                    process.stdout.write('  ');
                }
                process.stdout.write(' '+f);
                for(var e=1;e<(a-c-2)/2+1;e++){
                    process.stdout.write('  ');
                }
                process.stdout.write(' '+f);
            }
            if(i>1&&i<(b-d)/2+1){//s
```

```
                for(var r=1;r<a-1;r++){
                    process.stdout.write('  ');
                }
                process.stdout.write(' '+f);
            }
            if(i>(b-d)/2+d&&i<b){//x
                for(var r=1;r<a-1;r++){
                    process.stdout.write('  ');
                }
                process.stdout.write(' '+f);
            }

        }
        console.log();
    }

}
```

自定义函数处理字符串

```
'use strict';
var str = "ertysssu";
var count = 0;
var f = '+';
var arr=[];
var ss='';
function aa(str){

    for (var i in str){
        count++;
    }

}

function bb(){
    for(var j=0;j<=count-1;j++){
        arr[j]=str[j];
    }
}

function cc(f){
    for(var i =0;i<=count-1;i++){
        if(i<count-1){
            var s =arr[i]+f;

        }else{
            var s =arr[i];
        }
```

```
    ss=ss+s;
    }
}
function dd(){

}
aa(str);
bb();
cc(f);
console.log(count);
console.log(arr);
console.log(ss);
//1。字符串长度，，2。分割字符串，，3。通过指定字符合并数组。4。字符串搜索。5。截取
```

强化逻辑阶段

1.最近的编程一直在做一件程序员最喜欢做的事情，那便是不写注释，导致打开一段程序，想要了解具体功能，必须要浏览一遍程序，在以后的编程中，要养成写注释的习惯；2.程序的变量参数等可以集中申明，这在后期中的维护会有很大的帮助； 3.将一个问题进行拆分，拆分的越细，程序越好编写，逻辑训练便是加强拆分问题的能力； 4.思考问题要考虑到所有可能发生的情况。这样会最大的降低程序出现bug的可能性。

服务器

```
var net = require('net');
var clientList=[];

var server = net.createServer(function(socket){
    clientList.push(socket);
    socket.write('成功连接到服务器 \r\n');

    socket.on('data',function(data){
        console.log(data.toString());
   broadcast(data);
    });
    socket.on('end',function(){

   for(var j=0;j<=clientList.length;j++){
          if(socket.end()==clientList[j]){
             clientList.splice(j,1);
       }
   }
   //   var a = clientList.indexOf(socket);
   //
```

```
   //            clientList.splice(a,1);


    });

});
function broadcast(data){
       for(var i=0;i<clientList.length;i++){
          clientList[i].write(data);
       }
}

server.listen(1337,'127.0.0.1');
```

客户端

```
var net = require('net');
var hostname=process.argv[2];
var port=process.argv[3];
var client = net.connect({host:hostname,port:port},
    function(){
   console.log('连接服务器');
   process.stdin.setEncoding('utf8');
   process.stdin.on('readable',function(){
       var chunk = process.stdin.read();
       if(chunk!=null){
           client.write('data:'+chunk);
       }
   });
});
client.on('data',function(data){
    console.log(data.toString());
});
client.on('end',function(){
    console.log('服务器已断开');
});
```

声明变量

```
#include<stdio.h>
int aa = 8;//全局变量
int main()
{
    int i=1;
    int j=2;
    int k=3;
```

```
        for(int a=0;a<2;a++){
            int b=4;
        }

        for(int c=0;c<2;c++){
       int d=5;
        }
    }
```

在gdb中运行 p &aa 0x555555755010

p &a 0x7fffffffde54

p &c 0x7fffffffde58

p &i 0x7fffffffde5c

p &j 0x7fffffffde60

p &k 0x7fffffffde64

p &d 0x7fffffffde68

p &b 0x7fffffffde6c

上述打印结果说明了什么问题?

关于变量

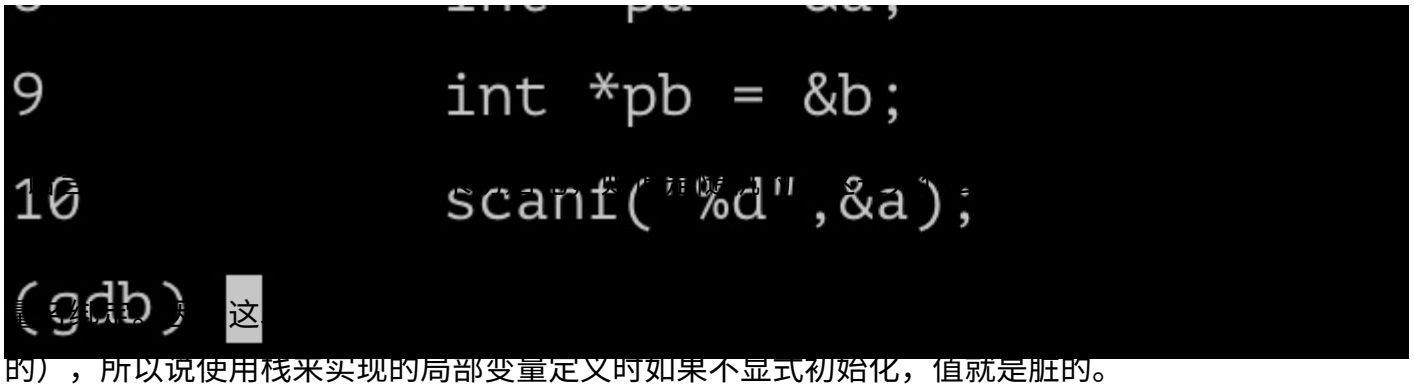
```
4       int main() {
(gdb) p aa
$1 = 0
(gdb) p c
$2 = 0
(gdb) p &a
$3 = (int *) 0x7fffffffe3d0
(gdb) p a
$4 = 1431652400
(gdb) p b
$5 = 21845
(gdb) l
1       #include<stdio.h>
2       int aa;
3
4       int main() {
5           int a;
6           int b;
7           static int c;
8           int *pa = &a;
```

全局变量

局部变量

局部变量

静态变量



的），所以说使用栈来实现的局部变量定义时如果不显式初始化，值就是脏的。

c语言中只有局部变量在未赋初值时，才是随机数，全局变量和静态变量未赋初值时编译器会自动将其初始化为0。局部变量是分配在堆栈上的，而全局变量和静态变量是分配在数据段中的。这个跟程序的内存分配是有关系的。

中缀表达式

3 / 5 + 8

18 - 9 * （ 4 + 3 ） 运算符在操作数中间就是中缀表达式

后缀表达式

3 5 / 8 +

18 9 4 3 + * - 这两个后缀是由上面的中缀转化而来，运算符在操作数后面就是后缀表达式

中缀转为后缀

中缀转化为后缀的规则；把每个运算符都移到它的两个操作数的后面，然后删除所有的括号即可；

后缀表达式的求值

将中缀表达式转换成等价的后缀表达式后，求值时，不需要再考虑运算符的优先级，只需从左到右扫描一遍后缀表达式即可。具体求值步骤为：从左到右扫描后缀表达式，遇到运 算符就把表达式中该运算符前面两个操作数取出并运算，然后把结果带回后缀表达式；继续 扫描直到后缀表达式最后一个表达式。

后缀表达式的求值的算法

设置一个栈，开始时栈为空，然后从左到右扫描后缀表达式，若遇操作数，则进栈；若遇运算符，则从栈中退出两个元素，先退出的放到运算符的右边，后退出的放到运算符左边 ，运算后的结果再进栈，直到后缀表达式扫描完毕。此时栈中仅有一个元素，即为运算的结 果。 例如，求后缀表达式：1 2 + 8 2 - 7 4 - / * 的值，栈的变化如下。

步骤	元素	说明
1	1	1进栈
2	1 2	2进栈
3		遇+号退栈2和1
4	3	1+2=3的结果3进栈
5	3 8	8进栈
6	3 8 2	2进栈
7	3	遇-号退栈2和8
8	3 6	8-2=6的结果6进栈
9	3 6 7	7进栈
10	3 6 7 4	4进栈
11	3 6	遇-号退栈4和7
12	3 6 3	7-4=3的结果3进栈
13	3	遇/号退栈3和6
14	3 2	6/3=2的结果2进栈
15		遇*号退栈2和3
16	6	3*2=6进栈
17	6	扫描完毕运行结束

中缀表达式转换为后缀表达式的算法

遇到数字输出，（ 和 运算符 入栈 ， 当遇到） 时，出栈到上一个（，将运算符输出；

---

git常用命令

1. git init 初始化
2. git config --global user.name XXX 设置用户名
3. git config --global user.email XXX 设置用户邮箱
4. git config --list 查看用户信息
5. git add 添加暂存区
6. git commit –m “XXXX” 提交到本地仓库
7. git status 查看项目状态

8. git push origin 本地分支名:远程分支名 推送到远程仓库

9. git pull 拉取远程仓库

创建分支(1.本地 git branch 分支名 2.远程分支1.可以再github创建 2.可以在本地把分支推到远程仓库)

删除分支（1.本地 git branch -d（D强制删除） 分支名 2.远程 git push origin --delete 分支名）

查看所有分支（git branch -a ）

切换分支（git checkout 分支名 ）

合并分支（git merge orgin/分支名）本地 就得说fetch从把同步远程分支更新到本地

比较fetch加merge 和 pull 。

结果上来看 确实 fetch 加merge 等于 pull

但是 fetch是让你本地所关联的远程端的commit id 版本号更新到了最新，然后我在本地在合并远程。

git pull 会将本地库更新至远程库的最新状态

禅意花园

```
body{
  font:12px/16px ;
  color#555;
  background: url(bg_left.gif) repeat-y #fff;
  margin:0;

}
abbr{
  border:0 none;
}
.page-wrapper{
  margin: 0;
  padding: 0;
}
.intro{
  width: 303px;
}
h1{
    margin:0;
```

```
    width:303px;
    height:198px;
    background: url(h1.jpg) no-repeat top left;
    text-indent: 100%;
    overflow: hidden;
    white-space: nowrap;
}
h2{
  margin:0 0 9px 0;
  width:303px;
  height:37px;
  background:url(h2.gif) no-repeat top left;
  text-indent: 100%;
  overflow: hidden;
  white-space: nowrap;
}
.summary p{
  margin: 0;
  padding: 8px 30px 4px 18px;
}
.summary p:first-child{
  font-weight: bold;
  color:#777;
}

.preamble h3{
  text-indent: 100%;
  overflow: hidden;
  white-space: nowrap;
  background:url(the_road.jpg) no-repeat top left;
  margin: 5px 0 0;
  width: 303px;
  height: 36px;
}
.preamble p{
  margin: 0;
  padding: 8px 30px 4px 18px;
  font-weight: bold;
  color:#777;
}
.supporting{
  position: absolute;
  top:70px;
  left:303px;
  width: 320px;
  margin: 0;
  padding: 0;
  background: url(support_bg.gif) repeat-y top right;
}
.supporting h3,.supporting p{
  margin: 0;
```

```
  padding: 0 40px 10px 0;
  color:rgb(54, 52, 52);
}
.supporting h3{
  text-indent: 100%;
  overflow: hidden;
  white-space: nowrap;
  opacity: 0;
  margin: 0 40px 5px 0;
  height:34px;
  border-bottom: 1px solid rgb(255, 255, 255);
}
.explanation{
  background: url(so_what.jpg) no-repeat top left;
  margin-right: 14px;
}
.participation{
  background: url(parti.gif) no-repeat top left;
}
.benefits {
  background: url(bene.gif) no-repeat top left;
}
.requirements {
  background: url(req.gif) no-repeat top left;
}
.sidebar{
  position: absolute;
  top:130px;
  left: 608px;
  width: 200px;
  margin:0;
  padding: 0;
  font-weight: bold;
  color:#777;
}
.sidebar h3{
  text-indent: 100%;
  overflow: hidden;
  white-space: nowrap;

  margin: 5px 0 0;
  width: 303px;
  height: 36px;
}
.sidebar h3.select{
  background:url(select.jpg) no-repeat top left;
}
.sidebar h3.archives{
  background:url(archives.jpg) no-repeat top left;
}
.sidebar h3.resources{
```

```
  background:url(resources.jpg) no-repeat top left;
}
footer{
  background: url(foot.jpg);
  margin-left: -14px;
  margin-right:14px;
}
ul{
  margin: 7px 0 7px 20px;
  padding: 0;
}
li{
  display: block;
  list-style-type: disc;
  margin-block-start: 1em;
  margin-block-end: 1em;
  margin-inline-start: 0px;
  margin-inline-end: 0px;
  padding-inline-start: 12px;
}
a{
  color:rgb(119, 114, 114);
}
footer a{
  color: black;
}
a:hover{
  color:red;
}
```