

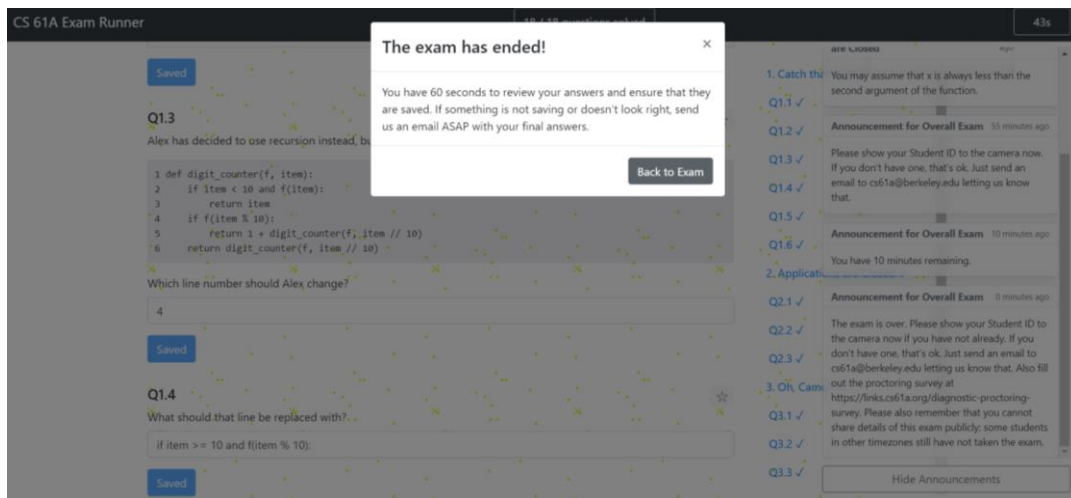
# Diagnostic Recall

Diagnostic 一共就三个 problem。digit\_counter(f,item); application and closed (太简单以至于忘记了题目考的是什么函数和什么考点); is\_camel\_number(n);

但是每个 problem 都有 local defined function 和 recursion，我不是很熟练。

而且都是填空题或者 debug 题。对方给出一段代码。我首先要花时间去理解函数的意思与题目的要求，然后要阅读理解对方的思路！之后再找题目中的问题，去 debug！

我发觉自己就是过于忙于 code interpreter 里测试了，导致很多小的 question 没有梳理思路。经常反复去读题干和读代码去理解思路，然后发现了一段不对劲的代码，就修改一下看着，然后疯狂 test samples。这样耽误了很多时间去试错了！这样的问题在于没有带入具体的例子，去测试他给出代码的各行意义！我应该带入一些例子，然后人脑逐步编译的！



比如这个 recursion-question

最后 10 分钟也没搞清楚那个 recursion 的思路，实在想不出来 digit\_counter(f,item)这第一个 problem 的 recursion-question 的代码是什么思路，为什么有两个 return 一个 return 1+digit\_counter(f,item//10)，而另一个是直接不要 1 了，recursive case 不就应该 func(item)=1+func(item//10)吗？这个对应 iteration-question 的 count=count+1 啊！

考前 5 分钟一直在想这个问题，时间到了我还是没想出来！考完 5 分钟内，我把 f 换成 is\_even，即 is\_even=lambda x: x%2==0；再把 item 换成 4343 这个数了，然后突然就懂了！

item%10 是 3！所以这个 3 肯定不会让 digit\_counter+1 的！而是直接返回 digit\_counter(f,item//10)，所以 Line 6 是没错的！

然后到最高位数字 4 的时候，item=4，这个 base case 应该 return 1！因为位数是 1，肯定不是 return item！

## Problem1 Catch that Bug!

第一个 Problem 有三个小 question，分别是 iteration, recursion 和 local defined function。

Recursion 的问题如之前所述，不多总结了。第一个 problem 的 local defined function 也遇到困难了，虽然写出来了，但是我瞎试出的答案。

那里的 helper(0,0)改成 helper(1,0)，我也只是抓住了(item//x)%10 这一个细节去理解代码的，完全没有去代入例子理解代码书写思路！纯粹是在 online coder 里 test 各种 samples，这样是非常有问题的！

## Problem2. Applications are Closed

第二个 problem 倒是很简单，填空还是 debug 我给忘记了，直接 5 分钟内搞定了 problem2

## Problem3. Camel Sequence

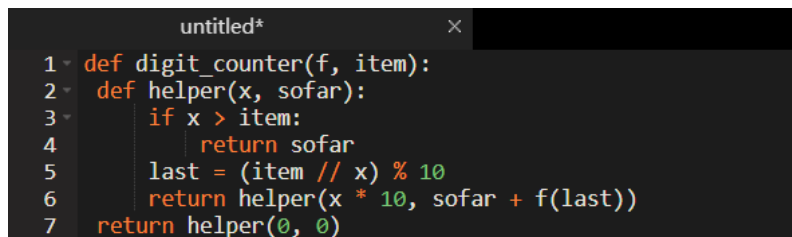
第三个 problem 的 `is_camel_number(n)` 也用了 helper 的 func, `helper(n,thank)`，我直接给看懵了，不知道 `thank` 是啥，为什么传入一个 `thank`，后来纯看代码发觉 `thank` 像是 `True` or `False` 的 Boolean Type。而且也花了不少时间，有 10 分钟才搞清楚挖出的坑是什么类型的语句，然后再用 10 分钟才填写完毕并且 online coder 测试完毕。

而且这个 `is_camel_number(n)` 我得出来的时候还是带入了 `n=12` 这个例子才理清思路的！那是考试临时起意代入的一个数，没有发觉其他题目也应该这样，所以其他题目继续死磕以至于时间到了还没写完。

看来带入例子进行对方代码理解和填空，真的极其重要！不能光看字母代码去理解思路！更不能随便修改个别看着不对劲的代码，然后疯狂 test samples!

## Review

对于瞎蒙的 Q1.3 `digit_counter(f,item)` 的 local defined function version，现在利用考完反思 Q1.2 的代入数值分析出题人思路的方法，来重新解决这个 Q1.3 的思路



```
untitled* x
1 def digit_counter(f, item):
2     def helper(x,sofar):
3         if x > item:
4             return sofar
5         last = (item // x) % 10
6         return helper(x * 10, sofar + f(last))
7     return helper(0, 0)
```

需要 debug 的代码是这样的

我代入 `f=is_even`，"`>>>is_even=lambda x: x%2==0`"，然后"`>>>digit_counter(is_even, 4343)`"

然后 7 分钟就理清这个题目的思路了！具体过程如下：

4343 是 item 之后，先是 `helper(0,0)`

If `x>item` 无效，进入 `last` 那句，然后发现 `//x` 出错，而且 `item%10` 显然是要取余数，然后后面的 `f(last)` 显然是要判断这个 digit 是否 `f(digit)==True`，所以初始的 `x` 是 1。因此 `last=3`，`is_even(3)` 是 `False`，之后就 `return helper(10,0+False)` 了，可能 `sofar+False` 很费解，不过在 terminal 试了一下 `0+False=0`，`0+True=1`。

所以返回为 `helper(10,0)`，`last=(4343//10)%10`，显然是取第二位的 4，`last=4`。`return helper(100,0+True)` 即 `helper(100,1)` 新的 `x=100`，新的 `sofar=1`

`Helper(100,1)` 中，`x=100>4343` 仍然是 `False`，所以这个 Base Case 跳过，`last=3`，`return helper(1000, 1+0)`

`Helper(1000,1)` 中，跳过 Base Case，`last=(4343//1000)%10=4%10=4`，最后一个位了。因此 `return(10000, 1+1)`

对于 `helper(10000,2)`，“if `x>item`： ”的判断即为“`x=10000>item=4343`” 结果是 `True`，因此 `return sofar`，而 `sofar` 就是 `helper` 传入的第二个 argument，2！这就是 item 中满足 `is_even` 的 digit 的数量，也是 `digit_counter(f,item)` 的返回值！

至此成功解决问题，而 `sofar+f(last)` 传入以 `sofar` 为参数的 `helper` 函数，其实就是 `sofar=sofar+f(last)`，而 `f(last)` 就起到了如果 `last` is even 那么 `count+1` 的功能！