递归

2种常见的查找方式:

1. 使用while循环,伪代码如下:

```
    def look_for_key(main_box):
        pile = main_box.make_a_pile_to_look_through()
        while pile is not empty:
            box = pile.grab_a_box()
            for item in box:
                if item.is_a_box():
                    pile.append(item)
                elif item.is_a_key():
                    print "found the key!"
```

2. 使用递归,伪代码如下:

```
def look_for_key(box):
    for item in box:
        if item.is_a_box():
            look_for_key(item)
        elif item.is_a_key():
            print "found the key!"
```

comment:

> 递归只是让解决方案更清晰，并没有性能上的优势。实际上，在有些情况下，使用循环的性能更好。Leigh Caldwell在Stack Overflow上说的一句话：“如果使用循环，程序的性能可能高；如果使用递归，程序可能更容易理解。如何选择要看什么对你来说更重要。

基线条件和递归条件

1. A wrong example:

(伪代码如下)

```
def countdown(i):
    print i
    countdown(i-1)
```

2. Corrected(添加了基线条件)

(**The program can stop now!**)

(伪代码如下)

```
def countdown(i):
    print i
    if i <= 0:
        return
    else:
    countdown(i-1)
```

Stack(栈)

call stack(调用栈)

(**include 递归调用栈**)

2 operations:

1. pop(弹出栈)
2. push(压入栈)

> 栈包含未完成的函数调用，每个函数调用都包含还未检查完的盒子(e.g.)使用栈很方便,因为你无需自己跟踪盒子堆(e.g.)——栈替你这样做了。 使用栈虽然很方便，但是也要付出代价：存储详尽的信息可能占用大量的内存。每个函数调用都要占用一定的内存，如果栈很高，就意味着计算机存储了大量函数调用的信息。在这种情况下，你有两种选择。

1. 重新编写代码，转而使用循环。
2. 使用尾递归。这是一个高级递归主题，不在本书的讨论范围内(not included here)。另外，并非所有的语言都支持尾递归。

comments:

1. 递归指的是调用自己的函数。
2. 每个递归函数都有两个条件：基线条件和递归条件。
3. 栈有两种操作：压入和弹出。
4. 所有函数调用都进入调用栈。
5. 调用栈可能很长，这将占用大量的内存。