Summaries

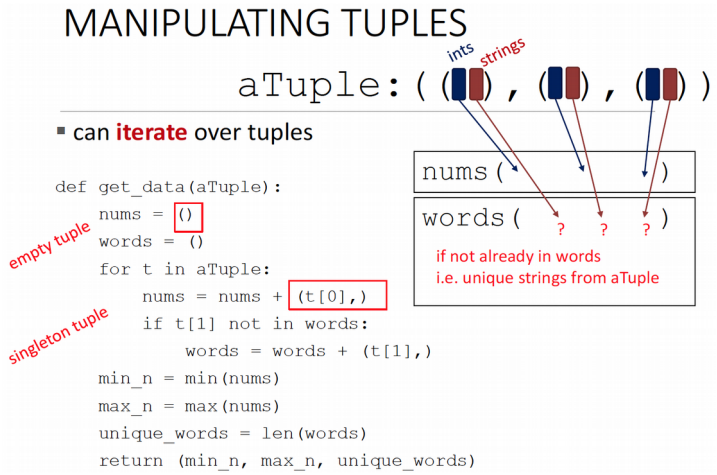
Lec 5 讲了元组、列表等数据结构。注意列表 list 的排序方法 sort()和 sorted()的区别。注意列表的拷贝和 list_a=list_b 实际上是设置别名的问题。
Lec 6 讲了迭代 iteration 和递归 recursion 的相同作用和不同特点，包括是如何判定程序结束的。（还没看完 Lec 6）

Lec 5

元组 Tuples

```
# 如何直接交换两个数
(x, y) = (y, x)

# 整数乘法
x = 10
y = 3
x / y # 3.33333
x // y # 3
```

Lec5-image-1.jpg



关于 OPERATIONS ON LISTS - REMOVE 的，for x in xxx 最好保持 xxx 不要变，比如 copy 一份，否则会有意想不到的 bug
python 特有的 split 和 join 操作（C 语言木有），还有排序（注意 sorted 和 sort 函数的区别），reverse

然后讲了一个列表 list，操作后改变与否的问题。比如用 sort 就是把 list 本身排序，sort 不返回值；sorted 不改变 list 本身，二是返回一个排序好的 new_list
还有 list_new = list 实质上是设置别名（两个不同的指针指向同一块区域），用 list_new 操作等价于 list 操作
除非用 list_new = list[:]或者 list.copy()
这也是个坑，我也踩过，列表这种多个元素组成的数组型数据结构，直接赋值很多语言都是设置别名 alias，要手动调用 copy 才是复制
类似的用法，opencv 中的矩阵也是如此

Lec 6

ITERATION vs. RECURSION

```
def factorial_iter(n):
    prod = 1
    for i in range(1,n+1):
        prod *= i
    return prod
```

```
def factorial(n):
    if n == 1:
        return 1
    else:
        return n*factorial(n-1)
```

- recursion may be simpler, more intuitive
- recursion may be efficient from programmer POV
- recursion may not be efficient from computer POV

数学归纳法
MATHEMATICAL INDUCTION
§ To prove a statement indexed on integers is true for all values of n:
◦Prove it is true when n is smallest value (e.g. n = 0 or n = 1)
◦Then prove that if it is true for an arbitrary value of n, one can show that it must be true for n+1

写一点有用的东西，递归、迭代、数学归纳法（recursion、iteration、induction）
其实本科 C 语言老师也是这么讲的，，，吧。。。经典的递归当然是斐波拉切数列、还有阶乘
张老师按照递推公式讲的，比如 factorial(n) = factorial(n-1) * n；另一方面，张老师毕竟硬核 C 语言，指针啥的很熟（毕竟研究生室友是微信老总张小龙），张老师讲，递归有个递归栈，一层一层压栈，递归返回的过程则是一层一层反方向出栈。看这些内容，神书 SICP 里面也提到了，栈的问题。如果一个递归函数输入某参数，导致压多深的栈，也解不出来，那么就会超过被分配的内存空间，俗称栈溢出（Stack Overflow）。

MIT Python 这门课则是讲了递归下一步的问题是本次问题的子问题（更小，更有解）即，最终步骤 b = 1 时，不需要递归，直接返回值；在后面的 b > 1 的时候，必须能够到达递归最底层 b = 1

函数 **mult** called with b = 1 has no recursive call and stops
函数 **mult** called with b > 1 makes a recursive call with a smaller version of b; must eventually reach call with b = 1

For recursive case, we can assume that mult correctly returns an answer for problems of size smaller than b, then by the addiSon step, it must also return a correct answer for problem of size b

一个回文序列的例子

如果 str 回文，则 str 第一个 char 和最后一个 char 相同，并且去头去尾也是回文的。

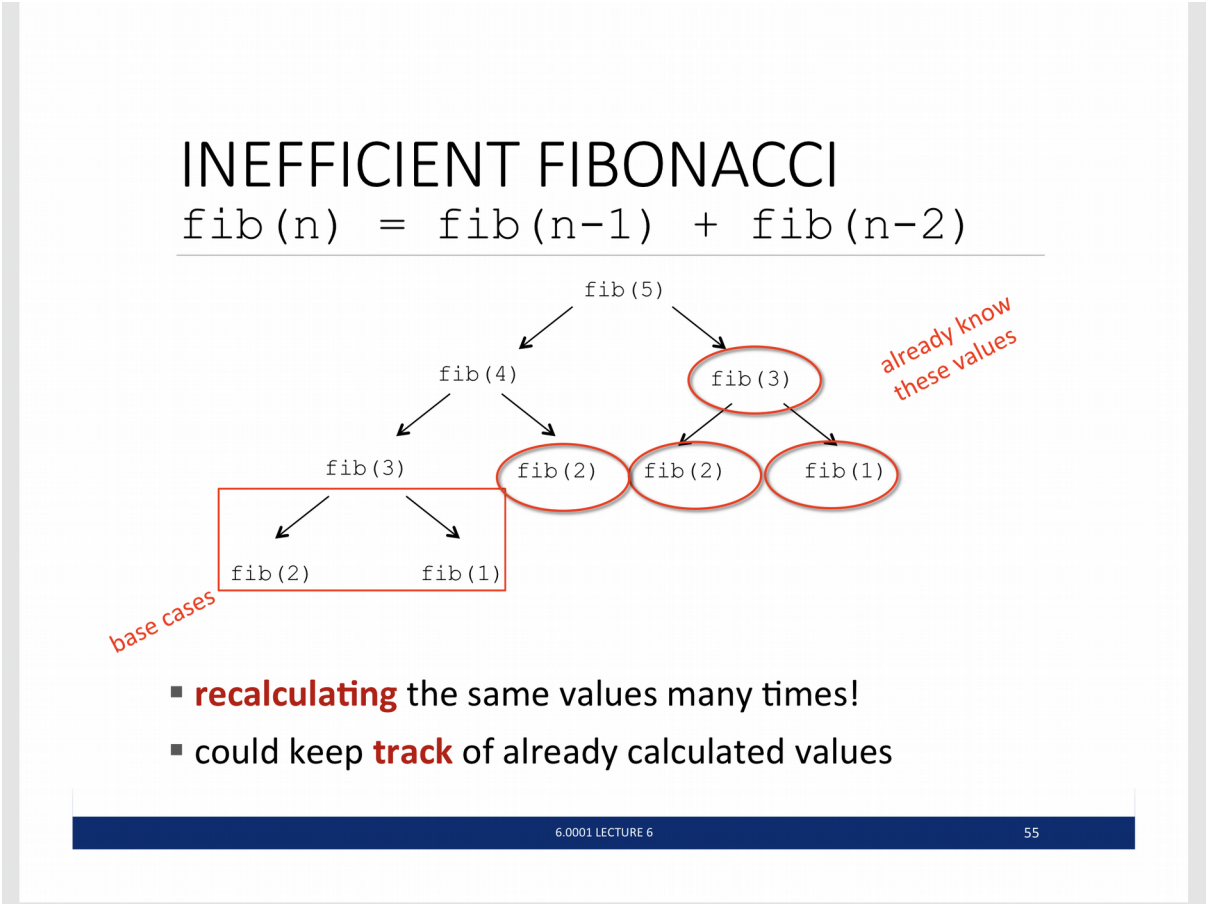
```
def isPalindrome(s):
    def toChars(s):
        s = s.lower()
        ans = ''
        for c in s:
            if c in 'abcdefghijklmnopqrstuvwxyz':
                ans = ans + c
            return ans

    def isPal(s):
        if len(s) <= 1:
            return True
        else:
            return s[0] == s[-1] and isPal(s[1:-1])

    return isPal(toChars(s))

print(isPalindrome("iloveshixixihsevoli"))
```

分而制之
DIVIDE AND CONQUER

§ an example of a “divide and conquer” algorithm § solve a hard problem by breaking it into a set of subproblems such that:

◦ sub-problems are easier to solve than the original ◦ soluSons of the sub-problems can be combined to solve the original

字典 dictionary
结构化存储?
字典.keys()和.values()不保证顺序
key 键值必须唯一
其他，看看课件就好，其实也不难，注意语法。



字典的键值对应用，存储递归代中重复使用的结果

```
def fib_efficient(n, d):
    if n in d:
        return d[n]
    else:
        ans = fib_efficient(n - 1, d) + fib_efficient(n - 2, d)
        d[n] = ans
        return ans


d = {1: 1, 2: 2}
print(fib_efficient(5, d))
```

效率提升
EFFICIENCY GAINS
fib(34)要 11405773 次递归，而 fib_efficient(34)只要 65 次。
§ Calling fib(34) results in 11,405,773 recursive calls to the procedure § Calling fib_efficient(34) results in 65 recursive calls to the procedure § Using dicSonaries to capture intermediate results can be very efficient