1.完成《编程导论》习题3.2

```
def find(iteration, key):
    ret_position = []
    for i in range(len(iteration)):
        if iteration[i] == key:
            ret_position.append(i)

    return ret_position
```

检测时，添加以下代码检测字符串：

```
iteration, key = input().split()
print(find(iteration, key))
```

添加以下代码检测列表：

```
iteration = [2, 4, 6, 'e']
key = 'f'
print(find(iteration, key))
```

Shell:

Case0:

```
thisisatest t
[0, 7, 10]

Process finished with exit code 0
```

Case1:

```
[]

Process finished with exit code 0
```

2.完成《编程导论》习题3.3

Shell:

```
8


Process finished with exit code 0
```

其中，a，b，c 为全局变量，仅有 d 为局部变量

3.完成《编程导论》习题3.6

Shell:

```
Finally, a is: 18 and b is 2


Process finished with exit code 0
```

其中，a，b 是整段代码的全局变量；x，y 是局部变量，但在函数 fun()中相当于 fun()的全局变量，可以被 F()访问与修改。

4.完成《编程导论》习题3.10

```
def total_reverse(l):
    ret = []
    for word in l[::-1]:
        temp_word = ""
        for index in range(-1, -len(word) - 1, -1):
            temp_word += word[index]
        ret.append(temp_word)
    return ret


L = ["It is", "very very", "funny", "!"]
print(total_reverse(L))
```

Shell:

```
['!', 'ynnuf', 'yrev yrev', 'si tI']

Process finished with exit code 0
```

5.完成《编程导论》习题3.16

```
L = ["%d * %d = %d" % (x, y, x*y) for x in range(1, 10) for y in range(1, 10) if x<=y]
```

Shell:

```
['1 * 1 = 1', '1 * 2 = 2', '1 * 3 = 3', '1 * 4 = 4', '1 * 5 = 5', '1 * 6 = 6', '1 * 7 = 7', '1 * 8 = 8', '1 * 9 = 9', '2 * 2 = 4', '2 * 3 = 6', '2 * 4 = 8', '2 * 5 = 10', '2 * 6 = 12', '2 * 7 = 14', '2 * 8 = 16', '2 * 9 = 18', '3 * 3 = 9', '3 * 4 = 12', '3 * 5 = 15', '3 * 6 = 18', '3 * 7 = 21', '3 * 8 = 24', '3 * 9 = 27', '4 * 4 = 16', '4 * 5 = 20', '4 * 6 = 24', '4 * 7 = 28', '4 * 8 = 32', '4 * 9 = 36', '5 * 5 = 25', '5 * 6 = 30', '5 * 7 = 35', '5 * 8 = 40', '5 * 9 = 45', '6 * 6 = 36', '6 * 7 = 42', '6 * 8 = 48', '6 * 9 = 54', '7 * 7 = 49', '7 * 8 = 56', '7 * 9 = 63', '8 * 8 = 64', '8 * 9 = 72', '9 * 9 = 81']

Process finished with exit code 0
```

6.完成《编程导论》习题3.17

```
text = "My house is full of flowers"
L = [letter[0] for letter in text.split()]
print(L)
```

Shell:

```
['M', 'h', 'i', 'f', 'o', 'f']

Process finished with exit code 0
```

7.《编程导论》3.4.2 中的第二部分：使用三种不同方法删除列表元素的执行时间，现在我们要删除的是字符串的元素。请分析三种方法的执行时间和优劣。

首先完成一个函数 remove(s,x):删除字符串 s 中第一次出现的 x

```
def remove(s,x):
    for i in range(len(s)):
        if s[i] == x:
            return s[:i] + s[i + 1:]
    return s
```

请仿照书本的三种方法，完成 removeall((s,x)函数：返回一个字符串是 s 剔除所有出现的 x

并做实验测试执行时间：

```
s = "1" * 1000000 + "0" * 1000
# 时间开始
a = removeall(s, "0")
# 时间结束
print(len(a))   # 不要直接打印 a，因为 a 太长了，此外还需要打印执行时间
```

按照第一种方法：

```
def removeall_1(s, t):
    while t in s:
        s = remove(s, t)
    return s


s = "1" * 1000000 + "0" * 1000
start = time.time()
a = removeall_1(s, "0")
elapsed = time.time() - start
print(len(a))
print(elapsed)
```

Shell:

```
1000000
27.959524869918823

Process finished with exit code 0
```

按照第二种方法：

```
def removeall_2(s, t):
    i = 0
    while i < len(s):
        if s[i] == t:
            s = remove(s, t)
        else:
            i += 1
    return s

s = "1" * 1000000 + "0" * 1000
start = time.time()
a = removeall_2(s, "0")
```

```
elapsed = time.time() - start
print(len(a))
print(elapsed)
```

Shell:

```
1000000
26.921823501586914

Process finished with exit code 0
```

按照第三种方法：

```
def removeall_3(s, t):
    s1 = ""
    for e in s:
        if e != t:
            s1 += e
    return s1

s = "1" * 1000000 + "0" * 1000
start = time.time()
a = removeall_3(s, "0")
elapsed = time.time() - start
print(len(a))
print(elapsed)
```

Shell:

```
1000000
0.07301640510559082

Process finished with exit code 0
```

8.电脑猜数字游戏

这个地方的解答可能不可靠!!!!!!! 因为这是一个很复杂的问题，要解决的话，可能需要你仔细去思考下怎么做。

玩家提供一个 4 位长度的数字，例如 5924（假设数字不重复）。电脑持续猜这个数字，每一次猜后会知道有几个数字位置完全相同，有几个数字位置不同但是数字出现。
这两种情况用几个 A 和几个 B 来表示。例如，一开始输入的秘密数字是 5924,你的电脑程序一遍遍猜这个秘密。每一次都会计算电脑猜的数字和真正的秘密相似度有多少相似(返回几个 A 几个 B).请写个 Python 程序能比较少次数猜出正确的数字。
如下是示例程序输出的例子，电脑第一次猜 9802,这个与 5924,没有任何数字位置相同，所以是 0A,有 2 个数字(9,2)出现但是位置不对，所以是 2B,结果就是 0A2B.然后电脑想了想，程序猜了第二个数 8956,返回 1A0B(9 是位置对，5 是数字存在但位置不对。然后程序再猜 8064,返回 1A0B,如此，猜到第 5 次，就抓到正确的秘密：

```
Now enter your secret number, and the computer will try to guess it: 5924

Guess 1 times: 9802. Ansser: 0 A, 2 B
Guess 2 times: 8956, Ansser: 1 A, 1 B
Guess 3 times: 8064, Ansser: 1 A, 0 B
Guess 4 times: 8125, Ansser: 1 A, 1 B
Guess 5 times: 5924, Ansser: 4 A, 0 B
```

a.请尝试分析对于 4 位长度的数字，至多（最坏情况）需要猜多少次才能得到正确答案？对于任意 n 位长度的数字呢？
*b.把这题改变一点，假设数字可以重复，计算 AB 时，先算有几 A,剩下的数再看有几 B.例如秘密是 4454,电脑猜 5445 而言，第二个 04 给了个 A,然后 5*45 和 4“54 比较，其中有 2 个(5,4)出现，所以是 2B.由于秘密只有一个 5,注意不要重复计算.5.*

a.

在最好的情况下，至多只需要猜 7 次就可以得到正确答案。

根据已有的计算数据（http://slovesnov.users.sourceforge.net/index.php?bullscows），在最优解的情况下，决策树分布为：

algorithm - crushBullsCows nodes - 5269 turns from recent moves - 95.7%															
		first turn response													total
		0.0	0.1	0.2	0.3	0.4	1.0	1.1	1.2	1.3	2.0	2.1	2.2	3.0	4.0	
layer	1														1	1
	2	1					1	1								3
	3	2	4	3	4	3	4	7	3	2	4	1	2			39
	4	65	75	57	63	6	81	77	62	5	50	43	4	22		610
	5	208	532	511	177		306	422	143	1	122	28		2		2452
	6	84	783	685	20		88	213	8		4					1885
	7		46	4												50
total numbers		360	1440	1260	264	9	480	720	216	8	180	72	6	24	1	5040
total turns		1812	7992	6930	1269	33	2396	3719	1020	31	846	315	22	98	1	26 484
average		5.033	5.55	5.5	4.807	3.667	4.992	5.165	4.722	3.875	4.7	4.375	3.667	4.083	1	5.255

因此可以得知最好情况是 7 次。

也同时可以得知，书上的那个代码并不是最好的代码，中间某些地方，对于 n 大一点的时候，可能存在可以优化的部分。

对于 n 的情况，由于每一层决策需要极大量的计算，我们只能在数学层面上估计一个平均数值，然后，根据统计学上的泊松分布去推测。

首先，我们可以知道，因为没有重复的数字，所以一定可以得出 n≤10。

我们首先假设对于一个 n 位数，得到的回复种类有 r 种，不重复的数字组合有 k 个，那么我们首先可以有：

$$k = \frac{10!}{(10-n)!}$$

然后，对于每一层决策树，可以粗略估计每层可以解决$(r-1)^{n-1}$种情况。

同时，还要考虑最坏情况下验证的数字个数经可能的少（即每次最坏仅仅验证一个数字）

另外，能够推理的，也就直接去推理了，而不是去估计

当 $d=1$ 时，r = 2，$k=10$ 每层分别解决 1 种情况，因此最差情况下要猜 10 次。

当 $d=2$ 时，r = 5，$k=90$ 每层分别解决 1 种，4 种，16 种，64 种，最后还剩下 6 种没有解决情况，因此最差情况下，可以得出平均猜测次数$(1\times1+4\times2+16\times3+64\times4+5\times5)\div90=3.76$次，经过泊松分布的拟合（这个模型来自于 n=4 时比较完备的数据），取概率≤0.1 的最近整数，下同），推算出极端情况下可能需要猜测 6 次才能猜中。然而，由于每次最坏情况仅能验证一个数字（在验证过程中另一个一定要么是位置正确，要么是数正确），完成数字验证就需要 8 次，最后猜对应该是 9 次。因此最坏应当取两者最大值，也就是 9 次。

当 $d=3$ 时，r = 9，$k=720$ 每层分别解决 1 种，8 种，64 种，512 种，最后还剩下 135 种没有解决情况，因此最差情况下，可以得出平均猜测次数$(1\times1+8\times2+64\times3+512\times4+135\times5)\div720=4.07$次，经过泊松分布的拟合，推算出极端情况下需要猜测 7 次才能猜中。然后，由于每次最坏情况仅仅能够验证一个数字，完成数字验证就需要 7 次，最后猜对最好就是 8 次。因此最坏应当取两者最大，也就是 8 次。

当 $d\geqslant4$ 时，就不应当从数字验证方面去看，因为每次数字验证并不能保证对于许多位置正确的数都可以验证其位置是否正确。

当 $d=5$ 时，r = 20，$k=30240$ 每层分别解决 1 种，19 种，361 种，6859 种，最后还剩下 23000 种没有解决情况，因此最差情况下，可以得出平均猜测次数$(1\times1+19\times2+361\times3+6859\times4+23000\times5)\div30240=4.75$次，经过泊松分布的拟合，推算出极端情况下需要猜测 7 次才能猜中。

当 $d=6$ 时，r = 25，$k=151200$ 每层分别解决 1 种，24 种，576 种，13824 种，最后还剩下 136775 种没有解决情况，因此最差情况下，可以得出平均猜测次数$(1\times1+24\times2+576\times3+13824\times4+136775\times5)\div151200=4.9$次，经过泊松分布的拟合，推算出极端情况下需要猜测 8 次才能猜中。

当 $d=7$ 时，r = 26，$k=604800$ 每层分别解决 1 种，25 种，625 种，15625 种，390625 种，最后还剩下 136775 种没有解决情况，因此最差情况下，可以得出平均猜测次数$(1\times1+24\times2+576\times3+13824\times4+390625\times5+136775\times6)\div604800=4.7$次，经过泊松分布的拟合，推算出极端情况下需要猜测 8 次才能猜中。

当 $d=8$ 时，r = 24，$k=1814400$ 每层分别解决 1 种，23 种，529 种，12167 种，279841 种，最后还剩下 1521839 种没有解决情况，因此最差情况下，可以得出平均猜测次数$(1\times1+23\times2+529\times3+12167\times4+279841\times5+1521839\times6)\div1814400=5.83$次，经过泊松分布的拟合，推算出极端情况下需要猜测 9 次才能猜中。

对于接下来的 n，可以考虑逻辑推理的方式。

当 $d=9$ 时，r = 19，每次最优可以排除 2 种情况，推算出极端情况下需要猜测 9 次才能猜中。

当 $d=10$ 时，r = 10，每次仅能排除 1 种排序情况，推算出极端情况下需要猜测 10 次才能猜中。

b.

主要是利用了字典的特性让计数

```
import random


def generateALL(length):
    L = []
    for i in range(10 ** length):
        t1 = str(i)
        len_t1 = len(t1)
        for j in range(length - len_t1):
            t1 = "0" + t1
        L.append(t1)
```

```
    return (L)

def guess_number_rand(L):
    if len(L) == 1:
        return L[0]
    i = random.randint(0, len(L) - 1)
    return L[i]


def verifyAB(SL, GL):
    SL_num = {}
    GL_num = {}
    for number in SL:
        SL_num[number] = 0
        GL_num[number] = 0
    for number in SL:
        SL_num[number] += 1
    if len(SL) != len(GL):
        print("ERROR in verifyAB")
        return ([])
    a = 0
    for i in range(len(GL)):
        if GL[i] in SL and GL_num[GL[i]] < SL_num[GL[i]]:
            GL_num[GL[i]] += 1
    b = sum(GL_num.values())
    for i in range(len(GL)):
        if GL[i] == SL[i]:
            a += 1
            b -= 1

    return [a, b]


def prune(Legal, G, Ans):
    L = []
    for e in Legal:
        a = verifyAB(e, G)
        if a == Ans:
            L.append(e)
    return L


if __name__ == "__main__":
    print("猜数字游戏，几个 A 代表有几个正确，并且位置也对，几个 B 来表示有及格正确但是位置不对的")
    for j in range(1):
        while True:
            S = input("你想让计算机猜几个数字（eg. 5）")
            SL = input("请输入你的秘密数字（数字可以重复）。计算机将要对你的数字猜测")
```

```
            if S.isdigit() and SL.isdigit():
                digit_num = int(S)
                if digit_num < 10 and len(SL) == digit_num:
                    break
        Legal_list = generateALL(digit_num)
        print("第一次将有 %d 个可能数" % (len(Legal_list)))
        Answer = [0, 0]
        i = 0
        while Answer[0] < digit_num:
            G1 = guess_number_rand(Legal_list)
            Answer = verifyAB(SL, G1)
            Legal_list = prune(Legal_list, G1, Answer)
            i += 1
            print("第%d次猜测：%s， 答案：%dA，%dB。剩有%d个可能的数" % (i, G1, Answer[0],
Answer[1], len(Legal_list)))
```