1.完成《计算机导论》练习题2.2.1，2.2.2，2.2.3，2.2.4，2.2.5，2.2.6.

编写以下函数，其中 in_num 为输入数，in_base 为输入进制，out_num 为输出数，out_base 为输出进制。可以完成 36 进制内整数的任意转换

```
def BaseConvert(in_num, in_base, out_num, out_base):
    temp = 0
    for i in in_num[:]:
        if type(i) == str:
            i = ord(i) - ord('A') + 10
        temp = temp * in_base + i
    while temp > 0:
        if temp % out_base > 9:
            out_num = [chr(temp % out_base - 10 + ord('A'))] + out_num
        else:
            out_num = [temp % out_base] + out_num
        temp //= out_base

    return out_num
```

2.2.1

```
print(BaseConvert([7, 8], 10, [], 2))
```

Shell:

```
[1, 0, 0, 1, 1, 1, 0]

Process finished with exit code 0
```

2.2.2

```
print(BaseConvert([1, 0, 1, 1, 0, 1], 2, [], 10))
```

Shell:

```
[4, 5]

Process finished with exit code 0
```

2.2.3

```
print(BaseConvert([3, 5, 8], 10, [], 16))
print(BaseConvert([3, 5, 8], 10, [], 8))
```

Shell:

```
[1, 6, 6]
[5, 4, 6]

Process finished with exit code 0
```

2.2.4

```
print(BaseConvert([1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1], 2, [], 10))
print(BaseConvert([1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1], 2, [], 16))
```

Shell:

```
[2, 4, 7, 3]
[9, 'A', 9]

Process finished with exit code 0
```

2.2.5

```
print(BaseConvert(['A', 'A', 0, 'C'], 16, [], 10))
print(BaseConvert(['A', 'A', 0, 'C'], 16, [], 2))
print(BaseConvert(['A', 'A', 0, 'C'], 16, [], 8))
```

Shell:

```
[4, 3, 5, 3, 2]
[1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0]
[1, 2, 5, 0, 1, 4]

Process finished with exit code 0
```

2.2.6

```
print(BaseConvert([1, 2, 3], 8, [], 2))
print(BaseConvert([1, 2, 3], 8, [], 10))
print(BaseConvert([1, 2, 3], 8, [], 16))
```

Shell:

```
[1, 0, 1, 0, 0, 1, 1]
[8, 3]
[5, 3]

Process finished with exit code 0
```

2.完成《计算机导论》练习题2.2.7，2.2.8，2.2.9，2.2.10，2.2.11，2.2.12.

2.2.7

选 B， 证明如下

考虑对于一个数 x，其 n 进制下位数满足：

$$digit_x = log_n x + 1$$

那么，由对数函数的增减性可知当 $x \geqslant 1$ 时道$log_{10} x \leq log_2 x$

因此，十进制整数 d 的位数小于等于二进制数 b 的位数

2.2.8

甲采用的二进制，乙采用的八进制，丙采用的十六进制，丁采用的十进制

2.2.9

按照第一问修改函数，其中 in_num 为输入数，in_base 为输入进制，out_num 为输出数，out_base 为输出进制。可以完成 36 进制内浮点数转换（保留 10 位小数）

```
def BaseFloatConvert(in_num, in_base, out_num, out_base):
    in_num_int, in_num_fra = in_num.split('.')
    out_num_int = []
    out_num_fra = []
    temp_int = 0
    for i in list(in_num_int)[:]:
        if ord(i) >= ord('A'):
            i = ord(i) - ord('A') + 10
        else:
            i = ord(i) - ord('0')
        temp_int = temp_int * in_base + i
    while temp_int > 0:
        if temp_int % out_base > 9:
            out_num_int = [chr(temp_int % out_base - 10 + ord('A'))] + out_num_int
        else:
            out_num_int = [temp_int % out_base] + out_num_int
        temp_int //= out_base
    temp_float = 0
    for i in list(in_num_fra)[::-1]:
        if ord(i) >= ord('A'):
            i = ord(i) - ord('A') + 10
        else:
            i = ord(i) - ord('0')
        temp_float = (temp_float + i) / in_base
    for i in range(10):
        if int(temp_float * out_base) > 9:
            out_num_fra += [chr(int(temp_float * out_base) - 10 + ord('A'))]
        else:
            out_num_fra += [int(temp_float * out_base)]
        temp_float = temp_float * out_base - int(temp_float * out_base)

    out_num_int = str(out_num_int)
    out_num_fra = str(out_num_fra)
    out_num = out_num_int + '.' + out_num_fra
    return out_num
```

第一个问

```
print(BaseFloatConvert("10010101.0111", 2, [], 10))
```

Shell:

```
[1, 4, 9].[4, 3, 7, 5, 0, 0, 0, 0, 0, 0]

Process finished with exit code 0
```

答案为 149.4375

第二个问

```
print(BaseFloatConvert("645.75", 10, [], 8))
```

Shell:

```
[1, 2, 0, 5].[6, 0, 0, 0, 0, 0, 0, 0, 0, 0]

Process finished with exit code 0
```

答案为 1205.6

2.2.10

分配 8 个袋子，胡萝卜个数分别为 1, 2, 4, 8 , 16, 32, 64, 128, 256, 512 个。就可以保证题目的要求。

考虑任何一个小于 1024 的整数的二进制形式，其每一位只有 1 或者 0 两种状态，而上面胡萝卜的分配办法可以满足任何一位都可以选择是 0 或是 1，即可以表示 1024 内的任何一个整数

2.2.11

使用第一问题中的函数，打印 36 进制内的转换表：

```
 6 : [7, 3]
 7 : [9, 'B']
 8 : ['C', 9]
 9 : ['F', 'D']
 10 : [1, 3, 7]
 11 : [1, 7, 7]
 12 : [1, 'B', 'D']
 13 : [2, 0, 9]
 14 : [2, 5, 'B']
 15 : [2, 'B', 3]
 16 : [3, 1, 1]
 17 : [3, 7, 5]
 18 : [3, 'D', 'F']
 19 : [4, 4, 'F']
 20 : [4, 'C', 5]
 21 : [5, 4, 1]
 22 : [5, 'C', 3]
 23 : [6, 4, 'B']
 24 : [6, 'D', 9]
 25 : [7, 6, 'D']
 26 : [8, 0, 7]
 27 : [8, 'A', 7]
 28 : [9, 4, 'D']
 29 : [9, 'F', 9]
 30 : ['A', 'A', 'B']
 31 : ['B', 6, 3]
 32 : ['C', 2, 1]
 33 : ['C', 'E', 5]
 34 : ['D', 'A', 'F']
 35 : ['E', 7, 'F']
 36 : ['F', 5, 5]
```

对照发现。当 8 进制时成立，因此这个数是 8 进制数

它的十进制表示法 201

2.2.12

添加以下代码来实现对每一次数的打印的计算（由于函数本身的“缺陷”——输入是一个特别的列表，代码显得有些复杂，再加上 Python3 比较阴间的 map 与 zip 返回对象为迭代器）

```
for i in range(6, 37):
    somme = 0
    zipped = list(zip(BaseConvert([5, 1, 2], i, [], 10), BaseConvert([5, 6, 3], i, [],
```

```
10)))
    for j in zipped:
        somme = somme * 10 + sum(j)
    print(i, ':', BaseConvert(list(map(int, list(str(somme)))), 10, [], i))
```

可以发现打印结果为

```
6 : [1, 5, 1, 5]
7 : [1, 4, 0, 5]                    22 : ['A', 7, 5]
8 : [1, 2, 7, 5]                    23 : ['A', 7, 5]
9 : [1, 1, 7, 5]                    24 : ['A', 7, 5]
10 : [1, 0, 7, 5]                   25 : ['A', 7, 5]
11 : ['A', 7, 5]                    26 : ['A', 7, 5]
12 : ['A', 7, 5]                    27 : ['A', 7, 5]
13 : ['A', 7, 5]                    28 : ['A', 7, 5]
14 : [5, 8, 'A']                    29 : ['A', 7, 5]
15 : ['A', 7, 5]                    30 : ['A', 7, 5]
16 : ['A', 7, 5]                    31 : ['A', 7, 5]
17 : ['A', 7, 5]                    32 : ['A', 7, 5]
18 : ['A', 7, 5]                    33 : ['A', 7, 5]
19 : ['A', 7, 5]                    34 : ['A', 7, 5]
20 : ['A', 7, 5]                    35 : ['A', 7, 5]
21 : ['A', 7, 5]                    36 : ['A', 7, 5]
```

对照发现 7 进制就是答案

反思这个代码，发现输入和输出都不便于阅读。尽管对于自己使用而言，这个不是很大的问题，但如果要提供给他人，这绝对不是一个好的代码，因此，接下来的编写中会修改输入和输出的方式。

3.完成《计算机导论》程序练习 2.2.1

第一二题使用的函数在输入上和输出上格式没有控制好。按照书本适当改写，做一些改进，输入输出都是字符串。

```
def BaseConvert(in_num, in_base):
    in_num = list(in_num)
    out_num = 0
    for i in in_num[:]:
        i = ord(i) - ord('0')
        out_num = out_num * in_base + i

    return str(out_num)


num, base = input("Please enter the number and its base:").split()
base = int(base)
print("The decimal number is", BaseConvert(num, base))
```

Case0:

```
Please enter the number and its base:100 2
The decimal number is 4

Process finished with exit code 0
```

Case1:

```
Please enter the number and its base:145 8
The decimal number is 101

Process finished with exit code 0
```

Case2:

```
Please enter the number and its base:123 6
The decimal number is 51

Process finished with exit code 0
```

4.完成《计算机导论》程序练习 2.2.2

```
def BaseConvert(in_num, out_base):
    in_num = int(in_num)
    out_num = []
    while in_num > 0:
        out_num = [in_num % out_base] + out_num
        in_num //= out_base

    return ''.join(list(map(str, out_num)))


num, base = input("Please enter a decimal number and the base you want:").split()
base = int(base)
print('The number in base %d is %s:' % (base, BaseConvert(num, base)))
```

Case0:

```
Please enter a decimal number and the base you want:100 2
The number in base 2 is 1100100:

Process finished with exit code 0
```

Case1:

```
Please enter a decimal number and the base you want:123 8
The number in base 8 is 173:

Process finished with exit code 0
```

Case2:

```
Please enter a decimal number and the base you want:1564 6
The number in base 6 is 11124:

Process finished with exit code 0
```

5.根据《编程导论》2.4 节中针对 for 的讨论，完成以下任务：

a.说明 Python 中 `for i in range(a, b ,step)` 与 `for(i = a ; i < b; i = i + step)` 之间的差异

b.解释 Python 中 `for i in range(len(L))` 与 `i = 0; while i < len(L): i+= 1` 结构的差异

c.解释 `for i in L`（L 是一个列表）是如何遍历的。假设 L 在循环中被改变了，接下去是新列表还是旧列表被遍历？它比较像 `for i in range(a, b ,step)` 结构还是 `while i < len(L)` 结构？说说你的理由。

a.

第一部分：从本质上来论述这两条语句循环的差异

在 Python 中，该语句的具体含义是：

创建一个<range>对象（通过.__next__()方法就可以发现<range>对象并不是迭代器，而是一种特殊的可迭代对象）。随后，系统将会为 `range(a, b ,step)` 之结果创建一个迭代器，之后为其迭代器所提供的每一项将会给 i 赋值，之后执行一次子语句，并且与迭代器返回顺序一致。当抛出异常 *StopIteration* 时，终止循环。

在 C 语言中，该语句的具体含义是：

给变量 i 赋值为 a，之后计算表达式 `i < b`，如果结果为 false（0），则跳转到 for 语句外；如果返回 true（1），就进入循环语句块，并在语句块结束执行后执行一次 `i = i + step` 语句，再次计算表达式 `i < b`，如果结果为 false（0），则跳转到 for 语句外；如果返回 true（1），就进入循环语句块，重复以上过程。

第二部分：其本质差异引起的特性：

1.在 Python 中，对于输入的 a， b， step，一旦创建了对应的<range>对象后，在之后循环中无论怎样修改 a，b，step，这个<range>对象都不会随之改变（不妨想象成向函数中输入了参数，已经返回数值了，这个时候无论怎么修改传入值，已经返回的数值不能改变），这也导致其在循环过程中上限，步长不能改变；在 C 中，由于过程中都是在对变量进行赋值与运算，因此可以在循环过程中修改上限，步长。

2.在 Python 中，其迭代器不断给 i 赋值，结束标志是迭代器抛出异常；在 C 中，每次将计算表达式，表达式保持为真时运行，为假时结束。

b.

对于语句 `for i in range(len(L))` 而言，由上述可知创建了相应的<range>对象与迭代器，之后为其迭代器所提供的每一项将会给 i 赋值，之后执行一次子语句，并且与迭代器返回顺序一致。当抛出异常 *StopIteration* 时，终止循环；对于语句 `while i < len(L): i+= 1` 而言，每次都会重复地检验表达式（这个表达式会被多次运算，所有运算符均如此），如果表达式结果为真则执行其中语句，为假则终止循环。

那么这就意味着，语句 `for i in range(len(L))` 中无论如何改变 L，都会像 a 中的一样，不能改变循环的次数和结果；而在语句 `while i < len(L): i+= 1` 中，由于每次都会重复地检验表达式，其中的函数自然也会被运算，这就意味着如果在循环中修改了 L，表达式中 len(L)也会变化，循环的次数有可能发生改变，结果也有可能改变。

c.

对于<list>对象，由于它不具备.__next__()方法，因此不是迭代器。系统会对 L 创建一个迭代器，之后为其迭代器所提供的每一项将会给 i 赋值，之后执行一次子语句，并且与迭代器返回顺序一致。当抛出异常 *StopIteration* 时，终止循环，如此遍历了 L 中的每一项。要补充的是，对于具有.__iter__()方法的对象，for 循环都可以把这个对象转化为迭代器。

而对于可变序列（如列表）在使用时，转化为迭代器的时候并不会复制该列表，而是保留这个迭代器与列表的关联：在程序内部有一个计数器用来跟踪可变序列下一个使用的项（这里这个项用于赋值 i），每次迭代都会返回这个计数器指向的一项并且使得计数器递增，当计数器的数值超过序列长度时立刻抛出异常 *StopIteration*。这意味着一些改变序列长度的做法都有可能改变迭代的次数和结果。例如，在循环过程中，删除了这个列表当前或者之前项，由于下一项计数器的数值变为了当前处理项目的编号，会使得下一项的被跳过。又或者是在循环过程中在当前项的前面插入新的一项，那么当前项又会被处理一次。所以，相对于是对新列表进行了遍历。

尽管这个过程与的 `while i < len(L)` 原理完全不同，但在循环效果上，两者都可能随着 L 的变化而改变循环次数，因而我认为这两者相似。

6.根据以下两种要求，完成《编程导论》习题2.14

a.请不要用递归完成此程序

b.【可以挑战，不做不扣分】使用递归完成此程序

a.

```
def merge(L1, L2):
    ret = []
    i = 0
    j = 0
    while i < len(L1) and j < len(L2):
        if L1[i] > L2[j]:
            ret.append(L2[j])
            j += 1
        else:
            ret.append(L1[i])
            i += 1
    if i == len(L1):
        ret += L2[j:]
    elif j == len(L2):
        ret += L1[i:]

    return ret
```

b

```
def merge(L1, L2):
    ret = []
    if len(L1) < len(L2):
        L1, L2 = L2, L1
    if len(L2) == 1:
        for i in range(len(L1)):
            if L1[i] < L2[0]:
                ret += [L1[i]]
            else:
                ret = ret + L2 + L1[i:]
                break
        else:
            ret += L2
    else:
        ret += merge(L1[:len(L1) // 2], L2[:len(L2) // 2])
        merge1 = len(ret) - 1
        merge2 = len(ret)
        # these are the point of merge, it's necessary to check the value
        ret += merge(L1[len(L1) // 2:], L2[len(L2) // 2:])
        if ret[merge1] > ret[merge2]:
            ret = ret[:merge1] + merge([ret[merge1]], ret[merge2:])

    return ret
```

7.根据《编程导论》2.5.1 节中对排序算法的介绍，完成以下任务：

a.编写选择排序和插入排序的 python 程序并验证。

b.假设输入的列表是反向的，从大到小，例如`N = 10000;L = [i for i in range(N, 0, -1)]`，请分析两种排序算法所需要的比较次数。哪个算法比较快。

a.

```
def selection_sort(L):
    for i in range(len(L) - 1):
        _min = i
        for j in range(i + 1, len(L)):
            if L[j] < L[_min]:
                _min = j
        L[_min], L[i] = L[i], L[_min]
    return L


def insertion(L, i):
    a = L[i + 1]
    for index in range(i + 1):
        if L[index] > a:
            break
    else:
        index = i + 1
    for k in range(i + 1, index, -1):
        L[k] = L[k - 1]
    L[index] = a
    return L
def insertion_sort(L):
    for i in range(len(L) - 1):
        L = insertion(L, i)
    return L
```

先测试选择排序，这里，选择生成一串随机的序列

```
L = random.sample(range(1, 1000), 10)
print(selection_sort(L))
```

Shell:

```
[72, 137, 159, 277, 527, 532, 606, 626, 668, 715]

Process finished with exit code 0
```

在测试插入排序

```
L = random.sample(range(1, 1000), 10)
print(insertion_sort(L))
```

Shell:

```
[5, 240, 258, 332, 368, 498, 522, 858, 971, 980]

Process finished with exit code 0
```

b.

对于题目中的 N，当采用以上写法的选择排序时，比较次数为$\frac{N(N-1)}{2}$次，当 N = 10000 时，比较了 49,995,000 次；

当采用以上写法时插入排序的时候（注意它每次比较都是从整个列表第一项开始），比较次数为 N – 1 次，当 N = 10000 时，比较了 9999 次，就比较次数而言，插入排序比选择排序次数少些。但是这并不意味着插入排序会比选择排序快，采用以下方法检测：

```
import time
start = time.time()
......
end = time.time()
```

我们可以检测一下，发现选择排序花费了 1.5915935039520264s，插入排序花费了 1.8715002536773682s，虽然相当，但也发现即插入排序比选择排序略微慢一些。这有可能是因为这种写法的插入排序移动数据花了更多的时间(这或许是 Python 本身的“缺陷”)：这种情况下插入排序中元素移动次数为$\frac{(N-1)(N+4)}{2}$次而选择排序的列表元素移动次数为$3 \times (N-1)$次）。