第一题

```
def accum(s):
    # TOD
    pass

# accum("abcd") => "A-Bb-Ccc-Dddd"
# accum("cwAt") => "C-Ww-Aaa-Tttt"
```

这到题用到了字符串的字母大写、小写、字符串拼接、复制。用到的函数有 join 将列表中的内容按照指定字符连接成一个字符串，

- upper() 所有字母变大写 和lower() 所有字母小写
- 对于一个可迭代的（iterable）对象，enumerate将其组成一个索引序列，利用它可以同时获得索引和值
- enumerate多用于在for循环中得到计数

具体代码：

```
def asuum(s):
    return '-'.join(y.upper() + y.lower()* x for x,y in enumerate(s))
a = asuum('abcd')
print(a)
```

第二题

```
def duplicate_count():
    # TODO
    pass

# 实现将一串字符串返回该字符串有重复的的字母个数包括大小
# test.assert_equals(duplicate_count("abcde"), 0)
# test.assert_equals(duplicate_count("abcdea"), 1)
# test.assert_equals(duplicate_count("indivisibility"),
1)
```

这里用到了将所有字母都转成小写还有集合，和列表

具体代码：

```
def duplicate_count(text):
    text = text.lower()
    # 用以去重，和原文本进行对比
    texts = set(text)
    lists = []
    for i in texts:
        numbers = text.count(i)
        if numbers != 1:
            lists.append(numbers)
    return len(lists)
```

第三题

```
def descending_order(num):
    # TODO
    pass

# descending_order(21445) => 54421
# descending_order(145263) => 654321
# descending_order(1254859723) => 9875543221
```

具体代码：

```
def descending_Order(num):
   return int("".join(sorted(str(num), reverse=True)))
```

第四题

```
def error_printer(s):
   # TODO
   pass

# s="aaabbbbhaijjjm", error_printer(s) => "0/14"
# s="aaaxbbbbyyhwawiwjjjwwm", error_printer(s) => "8/22"
# 计算字符串有多少个在'abcdefghijkmlm'里
```

具体代码：

```
from re import sub
def printer_error(s):
   return "{}/{}".format(len(sub("[a-m]",'',s)),len(s))
```

我们讲一下sub 的用法

re.sub共有五个参数。

re.sub(pattern, repl, string, count=0, flags=0)

其中三个必选参数：pattern, repl, string

两个可选参数：count, flags

第一个参数：pattern

pattern，表示正则中的模式字符串，这个没太多要解释的。

第二个参数：repl

repl，就是replacement，被替换，的字符串的意思。

repl可以是字符串，也可以是函数。

repl是字符串

如果repl是字符串的话，其中的任何反斜杠转义字符，都会被处理的。

\n：会被处理为对应的换行符； \r：会被处理为回车符； 其他不能识别的转移字符，则只是被识别为普通的字符：比如\j，会被处理为j这个字母本身； 反斜杠加g以及中括号内一个名字，即：\g，对应着命名捕获

第三个参数：string

string，即表示要被处理，要被替换的那个string字符串。

第五题

```
def expanded_form(num):
    # TODO
    pass

# expanded_form(12) => '10 + 2'
# expanded_form(42) => '40 + 2'
# expanded_form(70304) => '70000 + 300 + 4'
```

具体代码：

```
def expanded_form(num):
    nums = str(num)
    x = []
    # 这里也可以用enumerate来做
    for i in range(0, len(nums)):
        if int(nums[i]) != 0:
            s = str(int(nums[i]) * (10 ** (len(nums) - i - 1)))
            x.append(s)
    return ' + '.join(x)
```

第六题

```
def order_weight(s):
    # TODO
    pass

# 计算每一个数字的累加和
# order_weight("2000 10003 1234000 44444444 9999 11 11 22 123") => "11 11 2000 10003 22 123 1234000 44444444 9999")
# 2000就是2, 10003就是4, 也就是这串数字的累加和
```

具体代码:

```
def order_weight(s):
    return ' '.join(sorted(sorted(s.split(' ')),key = lambda x:sum(int(c) for c in x)))
```

首先把这个数进行切割，然后用lambda算出这个数的累加和，在根据累加和进行排序。

第七题

```
def valid_parentheses(s):
  # TODO
  pass

# "()" => True
# ")(()))" => False
# "(" => False
# "(())((()())())" => True
# 判断是否是有效空号对
```

具体代码：

```
def valid_parentheses(string):
    cnt = 0
    for i in string:
        if i == "(":
            cnt+=1
        if i == ")":
            cnt-=1
        if cnt < 0:
            return False
    return True if cnt == 0 else False
```

这道题目题解很多，这是最简单的了，大家可以自行实现一个栈，当然也可以去我的github的exercise/python_data_structure/base_data_structure/stack.py下载，这道题目其实是一个标准的检验栈实现。

检测到一个左括号就压栈，检测到一个右括号就出栈，最后检查栈，若还有元素则False，没有则True。