第三题：

这道题的代码主要分为三个部分：

（1） 函数 smith_waterman()的定义；该函数复现了局部比对算法 smith_waterman()的过程，输入的为两个待比对的字符串，输出为

（2） 函数 local_trackback()的定义；该函数则是以上面的 smith_waterman()函数的输出结果为输出，打印回溯矩阵；

（注：在编程的时候，比较常用的思维的模式是“模块化思维”，即将一个具体的问题，分解成不同的独立模块，不同的模块用不同的函数封装实现，然后按顺序的调用函数即可。）

（3） 以及测试代码，输入给定的字符串，去看两个函数的输出的结果；

在理解本题代码之前，需要首先明白局部比对算法 smith waterman 的主要的算法的过程和原理。

一、局部比对算法 smith waterman 原理

（一）算法目的

smith waterman 算法是一种局部比对的算法，和全局比对不同，其主要的目的是找到两个序列中，具有最高相似性的局部片段，而非考虑整体的相似性。

（二）主要公式

该算法的打分矩阵为 Hij（如下）：

$$H_{ij} = \max \begin{cases} H_{i-1,j-1} + s(a_i, b_j), \\ H_{i-1,j} - W_1, \\ H_{i,j-1} - W_1, \\ 0 \end{cases}$$

Hij 的值取的是后面四个计算结果的最大值。Hi-1,j-1，表示的是 Hij 在矩阵中的左斜上角的值，s(ai,bj)表示的是延伸的置换矩阵，分为两种情况，正确匹配或错配。而 Hi-1,j 表示的是 Hij 在矩阵中左边的值，Hi,j-1 表示的是 Hij 在矩阵中上边的值。W1 表示的是空位的罚分，即如果两个序列之间对应位置没有比对上，而是比对到了空位上，应该罚分多少。

我们可以根据比对的实际情况，人为的设定置换矩阵和空位罚分的值。比如下面就是一个示例，这里我们规定，如果两个序列中的对应碱基相同，则加 3 分，如果错误匹配，则罚 3 分，而如果对应到空位，则罚 2 分。

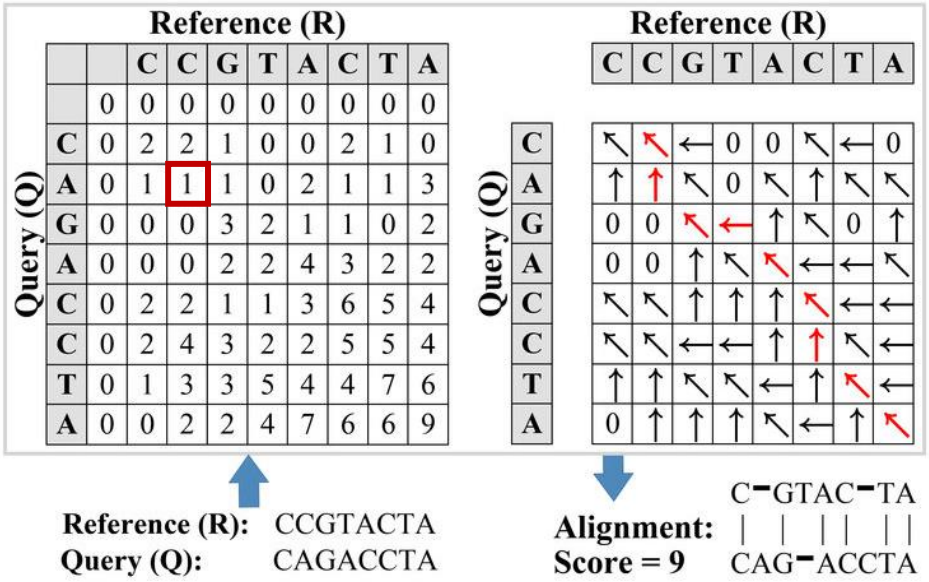
$$\text{置换矩阵：} s(a_i, b_j) = \begin{cases} +3, & a_i = b_j \\ -3, & a_i \neq b_j \end{cases}$$

$$\text{空位罚分：} W_k = kW_1, W_1 = 2$$

我们按照这种方式，逐个的遍历矩阵计算 Hij 的值，从而求出得分最大的那条路径。而该路径所对应的序列匹配的结果，就是最优的局部比对的结果。

下面介绍具体的实现过程。

（三）实现过程



1、打分

假设我们有两条序列，R 序列：CCGTACTA；Q 序列：CAGACCTA；

首先，我们将这两条序列空两格，填入上述矩阵中。先初始化该矩阵的第一（行）列【gap】，即全为 0（注：如果是全局比对的打分矩阵，初始化不一定为 0）。我们规定，正确匹配为 2，错配罚分为 1，空位罚分为 1，这样的打分矩阵。

那么我们图中红色框（C-A 不匹配）框的那个值，可以有以下四种情况：

（1） 斜上角：2+（-1）=1（负一为错配罚分，二为斜上角值）

（2） 上面：2-1=1（一为空位罚分）

（3） 左侧：1-1=0（一位空位罚分）

（4） 0

所以综合这四种情况，取最大值 1，作为该位置的最终的得分。

依次类推，我们可以得到这个矩阵中任何一个方格中的值。

2、回溯最优路径，计算 alignment score

我们从**矩阵的最大值**的位置出发，回溯。回溯的方向是临近的三个值中最大的值的方向，当这三个值有相等值时，则分枝进行，**一直到 0 为止**。也就是我们上面图中的红色的线。

（1） 对角线的那条线：表示两者正确匹配；

（2） “↑”：表示 Q 序列在该位置是 gap；

（3） “←”：表示 R 序列在该位置是 gap；

即对于 Q 序列和 R 序列，最优的匹配的结果是：

C - G T A C - T A

C A G - A C C T A

其中，正确匹配的碱基的数量为 6，错配为 0，gap 的数量为 3。

所以，最终计算得到最优的匹配结果的 alignment score 的值为：6*2-0-3=9（也是打分矩阵中的最大值）。

二、代码实现的解释

在该段代码之初，作者首先 import 了 numpy 模块（可以用这个模块中现成的函数），并将其简称为“np”。因此，接下来如果要调用这个模块中的函数，直接 np.zeros(),括号里面是要输入进函数的参数。

```
import numpy as np
```

（一） smith_waterman()函数

同样地，这里我们之前讲过很多次，使用 def 关键词，定义了名为“swith_waterman”的函数。

输入的参数有五个：

（1） s1/s2：候选进行局部比对的两条序列；

（2） match：正确匹配得分；这里默认为 1；

（3） mismatch：错配罚分；这里默认为-1；

（4） gap：空位罚分；这里默认为-1；

注：在多参数的情况下，这些参数的顺序是固定的。如第 4 个输入参数表示的就是 mismatch 的值；后面三个参数（有默认值）可以为空，但是如果有值，则程序是按照顺序对应的。

```
def smith_waterman(s1,s2,match=1,mismatch=-1,gap=-1):
```

接下来，我们初始化比对矩阵和回溯矩阵。

以输入的两条序列的长度+1，分别作为矩阵的维度，构建了两个 n*m 维的矩阵，并将矩阵，以 0 值作为填充（np.zeros((n,m))）。

```
#Initialize Alignment Matrix with 0
n = len(s1)+1 #number of rows
m = len(s2)+1 #number of columns
matrix = np.zeros((n,m))
#init tracebackmatrix
traceback = np.zeros((n,m))
```

		X	X	X
	0	0	0	0
X	0	0	0	0
X	0	0	0	0

接下来就是两个嵌套的 for 循环。所谓嵌套的 for 循环，就是外面的循环取一个值，进到你们的 for 循环，里面的 for 循环把所有的值都取完了，循环跳出；然后外面的 for 循环再取下一个值，带到里面，里面的 for 循环再转啊转，以此类推。最后，到外面的 for 循环也值取完了之后，整个 for 循环跳出，循环结束（无论有多少个嵌套的 for 循环，都是同样的道理，不过我们在编程的时候，会尽量避免使用 for 循环嵌套，因为程序运算极慢，对内存需求很大，程序会卡死）。

```
for i,a1 in enumerate(s1):
    for j,a2 in enumerate(s2):
```

那么，这里的 enumerarte()函数是什么意思呢？当你不知道这段代码什么意思的时候，最直接的办法就是使用 print 输出，看输出的是什么？我们可以看到 enumerate()函数把字符串 s1 的索引赋值给了 i，将索引所对应的字符赋值给了 a1，则每一次循环，则是从字符串中取出其索引及其所对应的字符。如此，直到字符串所有的字符读完。

```
>>> s1 = "ATAGCCCGAA"
>>> for i,a1 in enumerate(s1):
...     print(i,"\t",a1)
...
0        A
1        T
2        A
3        G
4        C
5        C
6        C
7        G
8        A
9        A
```

下面的这一段代码，其实是复现了我们前面提到的打分取 Hij 值的过程。按照打分矩阵，逐行进行赋分。

```
if a1==a2:
    score1 = matrix[i,j] + match
else:
    score1 = matrix[i,j] + mismatch
score2 = matrix[i-1,j] + gap #oben
score3 = matrix[i,j-1] + gap #links
score = np.max([score1,score2,score3,0])
matrix[i+1,j+1] = score
```

I=0,j=0,1,2,3……

		X	X	X
	0	0	0	0
X	0	c	0	0
X	0	0	0	0

I=1,j=0,1,2,3……

		X	X	X
	c	a	d	q
X	0	0	0	0
X	0	c	0	0

I=2,j=0,1,2,3……

		X	X	X
	0	0	0	0
X	0	s	f	e
X	0	0	0	0
……				

注：黄色表示 i，j 的取值位置，而红色区域表示程序运行的过程中，实际赋值的地方。

怎样进行赋分呢？还是四种情况：

（1） 斜上角：score1

- 正确匹配：matrix[i,j]+match
- 错配：matrix[i,j]+mismatch

（2） 上面：score2

Matrix[i,j+1]-空位罚分

（3） 左侧：score3

Matrix[i+1,j]-空位罚分

（4） 0

i,j	I,j+1
i+1,j	i+1,j+1

所以，最终 i+1,j+1 的值，取的是这四个值中的最大值（np.max）。然后，将其赋值给 matrix[i+1,j+1]。

```
if a1==a2:
    score1 = matrix[i,j] + match
else:
    score1 = matrix[i,j] + mismatch
score2 = matrix[i-1,j] + gap #oben
score3 = matrix[i,j-1] + gap #links
score = np.max([score1,score2,score3,0])
matrix[i+1,j+1] = score
```

然后接下来，就是记录回溯的时候的方向。score1、score2 和 score3 分别代表的是三个方向，我们回溯的时候是往值最大的那个方向。所以，这个时候就需要记录这样的值。

这里作者使用了 np.argmax()这个函数，这函数的作用是记录，列表中最大的那个值的索引。如果 score1 在这个三个值中最大，则 direction 的值为 0，如果 score2 在这个三个值中最大，则 direction 的值为 1，如果 score3 在这个三个值中最大，则 direction 的值为 2，如果是 0 最大，则 direction 的值为 3。

```
direction = np.argmax([score1,score2,score3])
traceback[i+1,j+1] = direction
```

```
>>> import numpy as np
>>> a = np.argmax([1,2,3])
>>> a
2
>>> a = np.argmax([1,4,3])
>>> a
1
>>> a = np.argmax([5,4,9])
>>> a
2
>>> a = np.argmax([5,4,2])
>>> a
0
```

于是，经过上面的两个 for 循环的遍历，我们得到了两个 matrix，一个 matrix 是我们的打分的矩阵，一个 matrix 是标记回溯方向的回溯矩阵。

其中，局部比对的最优的 alignment score 为打分矩阵中的最大值。作者将这三者作为函数 swith_waterman()的输出。

```
optimal_score = np.max(matrix)
return (optimal_score, matrix, traceback)
```

所以这里总结一下该函数的作用。
该函数的输入为待比对的两条序列。输出为这两条序列局部比对的最优的结果的 score 值，打分矩阵和标记回溯方向的回溯矩阵。

（二） local_track()函数

同样地，这里我们之前讲过很多次，使用 def 关键词，定义了名为“local_traceback()”的函数。s1 和 s2 为输入的两条不对的序列，matrix 为函数 smith_waterman()的输出结果的打分矩阵，traceback 为函数 smith_waterman()的输出的追溯矩阵。

```
def local_traceback(s1,s2,matrix,traceback):
```

我们先将待输出的字符串初始化为空。

```
r1 = ""
r2 = ""
```

indices = np.where(matrix==matrix.max()) 这行代码的意思，是取 matrix 打分矩阵中，最大值的 index，由于是一个二维的矩阵，所以其位置为坐标形式。在我们的示例矩阵中，有两个位置都为 3，一个 index 为[5,3],另一个 index 为[6,5]。

```
>>> np.where(matrix==matrix.max())
(array([5, 6]), array([3, 5]))
>>> matrix
array([[ 0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  1.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  2.,  1.],
       [ 0.,  1.,  0.,  0.,  0.,  1.,  1.],
       [ 0.,  0.,  2.,  1.,  0.,  0.,  0.],
       [ 0.,  0.,  1.,  3.,  2.,  1.,  0.],
       [ 0.,  0.,  0.,  2.,  2.,  3.,  2.]])
```

注意这边 np.where()是把坐标分别还存储的。无论有多少个位置都为最大值，其结果都为两个 array。第一个 array 存储这些位置横坐标的信息，第二个 array 存储这些位置的纵坐标的信息。

```
>>> matrix
array([[ 0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  1.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  2.,  1.],
       [ 0.,  1.,  0.,  0.,  0.,  1.,  1.],
       [ 0.,  0.,  2.,  1.,  0.,  0.,  0.],
       [ 0.,  0.,  1.,  3.,  2.,  1.,  0.],
       [ 0.,  0.,  0.,  2.,  2.,  3.,  2.]])
>>> indices = np.where(matrix==matrix.max())
>>> indices
(array([5, 6]), array([3, 5]))
>>> indices = np.where(matrix==matrix.min())
>>> indices
(array([0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 4,
       4, 4, 4, 4, 5, 5, 5, 6, 6, 6]), array([0, 1, 2, 3, 4, 5, 6, 0, 1, 2, 3, 5, 6, 0, 1, 2, 3, 4, 0, 2, 3, 4, 0,
       1, 4, 5, 6, 0, 1, 6, 0, 1, 2]))
```

所以，这里也比较好理解了。这里的 i，j 取得是第一个最大值的横纵坐标。

```
i = indices[0][0]
j = indices[1][0]
```

得到最大值的横纵坐标之后，我们从最大值的位置，开始向前向上回溯。
我们前面提到：
如果 score1 在这个三个值中最大，则 direction 的值为 0，如果 score2 在这个三个值中最大，

则 direction 的值为 1，如果 score3 在这个三个值中最大，则 direction 的值为 2。如果是 0 最大，则 direction 的值为 3。

所以这四种取值，意味着不同的方向。

下面就到了这个 while 循环：while 循环的判断条件是打分矩阵中的值大于 0，当遍历到的打分矩阵的值为 0 时，循环终止。

在循环体中，有三种条件的选择：

（1） 如果 traceback 矩阵中的值为 0（即对应到上面 score1 的值最大，斜对角的方向），则两个序列的某个对应碱基（分别对应的序列 s1 和 s2 的 i, j 的位置）匹配或错配。

（2） 如果 traceback 矩阵中的值为 1 或为 2，则序列比对时存在 gap，对应的存在 gap 的位置用“-”拼接。

```
    while matrix[i,j]>0:
        #Diagonal Step
        if traceback[i,j]==0:
            r1 += s1[i-1]
            r2 += s2[j-1]
            i -= 1
            j -= 1
        #Gap in Seq2
        elif traceback[i,j]==1:
            r1 += s1[i-1]
            r2 += "-"
            i -= 1
        #Gap in Seq2
        elif traceback[i,j]==2:
            r1 += "-"
            r2 += s2[j-1]
            j -= 1
```

最后，当矩阵遍历完成之后，逆序打印输出最优的两条序列的匹配的结果。

```
print(r1[::-1])
print(r2[::-1])
```

这边的 r1[::-1]的意思是，反向输出序列 r1。两个冒号分隔的是三个元素，前两个元素为空，则默认是序列的全部的首和尾，最后一个元素表示的是步长，为“负”则指的是逆序。

```
>>> r1="ATATACC"
>>> print(r1[::-1])
CCATATA
>>> print(r1[1:4:1])
TAT
```

（三） 测试代码

```
#Test Algorithm
s1 = "ACGTTC"
s2 = "GTTACC"
score,matrix,traceback = smith_waterman(s1,s2,
            match=1,mismatch=-1,gap=-1)
print("Optimal Score: %d" % score)
local_traceback(s1,s2,matrix,traceback)
```

最后就是这一段测试代码，输入的待比对序列 s1 和 s2。我们以 match=1，mismatch=-1，以及 gap 为-1 的打分矩阵，将其作为 smith_waterman()的输入，输出的变量分别赋值为 score、

matrix 和 traceback。
同样的，使用 print（）函数格式化打印 score 值。其中%d 指的是按照整数的格式输出数值型变量。
然后将 smith_waterman()的输出作为 local_traceback 的输入，打印序列比对的结果。