Programming Homework 1 Instructions (Read First)

In lecture and in a practical, we saw an implementation of the naive exact matching algorithm:

```
def naive(p, t):
    occurrences = []
    for i in range(len(t) - len(p) + 1):  # loop over alignments
        match = True
        for j in range(len(p)):  # loop over characters
            if t[i+j] != p[j]:  # compare characters
                match = False
                break
        if match:
            occurrences.append(i)  # all chars matched; record
    return occurrences

```

...and we saw a function that takes a DNA string and returns its reverse complement:

```
def reverseComplement(s):
    complement = {'A': 'T', 'C': 'G', 'G': 'C', 'T': 'A', 'N': 'N'}
    t = ''
    for base in s:
        t = complement[base] + t
    return t

```

...and we saw a function that parses a DNA reference genome from a file in the FASTA format.

```
def readGenome(filename):
    genome = ''
    with open(filename, 'r') as f:
        for line in f:
            # ignore header line with genome information
            if not line[0] == '>':
                genome += line.rstrip()
    return genome

```

...and we saw a function that parses the read and quality strings from a FASTQ file containing sequencing reads.

```
def readFastq(filename):
    sequences = []
    qualities = []
    with open(filename) as fh:
        while True:
            fh.readline()  # skip name line
            seq = fh.readline().rstrip()  # read base sequence
            fh.readline()  # skip placeholder line
            qual = fh.readline().rstrip() # base quality line
            if len(seq) == 0:
                break
            sequences.append(seq)
            qualities.append(qual)
    return sequences, qualities

```

First, implement a version of the naive exact matching algorithm that is *strand-aware*. That is, instead of looking only for occurrences of P in T, additionally look for occurrences of the*reverse complement* of P in T. If P is ACT, your function should find occurrences of both ACTand its reverse complement AGT in T.

If P and its reverse complement are identical (e.g. AACGTT), then a given match offset should be reported only once. So if your new function is called naive_with_rc, then the old naivefunction and your new naive_with_rc function should return the same results when P equals its reverse complement.

Hint: See this notebook for a few examples you can use to test your naive_with_rc function.

Next, download and parse the lambda virus genome, at: https://d28rh4a8wq0iu5.cloudfront.net/ads1/data/lambda_virus.fa