```
"""In order to only use reads that came from an original biological sample in an RNA seq experiment, PCR duplicates have
to be removed. In order to simplifiy this process, the RNA reads have been aligned to the reference genome. This allows
for us to find PCR duplicates by finding the actual 5' mapping position, UMI, and chromosome each read is located on. To find the 5' mapping position, you need to account for soft-clipping in the cigar string on the forward strand, and softclipping, matches, deletions, and skips for the reverse strand. Once these are all known, the reads have to be referenced to one another to see if they match. The "right" pcr duplicate then has to be chosen and written out to a new
file."""
import re
import argparse
import subprocess

def get_args():
    """ get file names of sam file, name of the known umis, and the character to indicate what to do with pcr duplicates
 """

    parser = argparse.ArgumentParser(description=' File name to be used')
    parser.add_argument('-f', '--file', type=str, help='File Name of sorted Sam File', required=True)
    parser.add_argument('-u', '--umi', type=str, help='File Name of known umis', required=True)
    parser.add_argument('-k', '--keep', type=str, help='Defines how to choose which duplicate is kept. Options are high'\
                                            'highest quality(q), first in file(1), or random(r)', \
                        required=False, default=1)
    return parser.parse_args()
args = get_args()
f = args.file
k = args.keep
u= args.umi

def umi(f):
    """Takes a file of known UMI's and adds them to a list"""
    return umis
# Test case: A file containing AAAAAAAA, output: ('AAAAAAAA')

def dflag(f):
```

```
    """Decodes bitwise flag for strand, first in pair, and returns true or false for each needed bit """
        return True
# Test case: 16  output: True

def dcigar(c,p):
    """Takes a cigar string and leftmost position and returns rightmost position using regex"""
    return p
#Test case: '3S66M2D3I24M2S', 3  output 97
def comp(h1,h2):
    """Uses two sam headers, finds the 5' start sites, compares umi's and returns True if they match and false if not"""

#Test Case:
# 'NS500451:154:HWKTMBGXX:1:11101:8846:5235-CAACTGGT^CAGTACTG;0^0  83 2  108949756  36 66M    =
108949700  -122'
# 'NS500451:154:HWKTMBGXX:1:11101:8846:5235-CAACTGGT^CAGTACTG;0^0  83 2  108949756  36 66M    =
108949700  -122'
# output: True

def dedupe1(f1):
    """Takes the initial Sam file, swaps leftmost position for 5' start and places leftmost at the end of the header
    f1 -- name of the original sam file
    returns new file """



# Testcase Sam File:
# Header 'NS500451:154:HWKTMBGXX:1:11101:8846:5235-CAACTGGT^CAGTACTG;0^0   83 2  108949756  36
66M    =  108949700  -122'
# Output 'NS500451:154:HWKTMBGXX:1:11101:8846:5235-CAACTGGT^CAGTACTG;0^0   83 2  108949822  36
66M    =  108949700  -122'
subprocess.call('samtools -view sort ("dd"+f1)', shell=True)
#calls bash commands from inside python, requires samtools to be loaded in talapas

def dedupe2(f2,umis,q):
    """Takes the new sorted Sam file, finds pcr duplicates, and chooses one to write based on user input
    f2 -- name of the sam file created by dedupe1 and samtools sort
    umis -- name of the tuple containing known umis produced from umi
    q -- user defined character that indicates what to do with PCR duplicates
```

```
    returns new sam file"""

#Testcase Input Headers
# 'NS500451:154:HWKTMBGXX:1:11101:8846:5235-CAACTGGT^CAGTACTG;0^0  83 2  108949822  36 66M    =
108949700  -122'
# 'NS500451:154:HWKTMBGXX:1:11101:8846:5235-CAACTGGT^CAGTACTG;0^0  83 2  108949822  36 66M    =
108949700  -122'
# output:
# 'NS500451:154:HWKTMBGXX:1:11101:8846:5235-CAACTGGT^CAGTACTG;0^0  83 2  108949822  36 66M    =
108949700  -122'
```