Demultiplex Part 3

Final Python Script and Output

Code needs to:

- Incorporate feedback from peer code reviews
- Utilize appropriate functions (perhaps you want to `import bioinfo`???)
- Sufficiently comment your code/use docstrings/use type annotations on functions
- Use unit tests on functions/entire algorithm to ensure it works properly
- Create a useful report for the end user of your code
- Use `argparse` to "generalize" your code
- Be mindful of "simple" things you can do to optimize your code
- Follow the specifications laid out in Assignment the First for the code

Modules that I decided to use

```
import bioinfo
import argparse
import gzip
```

High Level Function (including examples)

```
def rev_comp(sequence: str) -> str:

'''Takes a DNA sequence (index) and returns the reverse complement'''

        pair = {"A":"T", "T":"A", "G":"C", "C":"G", "N":"N"}

        reverse_comp = ""

        reverse = sequence[::-1]

        for base in reverse:

                reverse_comp += pair[base]
```

```
        return reverse_comp


#sample test!

#print(rev_comp("CAT"))

#input: CAT

#output: ATG
```

```
def edit_header(R1_header: str, R4_header: str, R2_seq: str, R3_seq: str) -> tuple[str, str]:

'''Takes a header, adds two indexes to the end, returns a new header'''

        R1_header += " " + R2_seq + "-" + R3_seq

        R4_header += " " + R2_seq + "-" + R3_seq

        return R1_header, R4_header

#sample test!

#print(edit_header("Header1", "Header2", "AAAA", "TTTT"))

#input: Header1, Header2, AAAA, TTTT

#output: Header1 AAAA-TTTT, Header2 AAAA-TTTT
```

```
def check_q(R2_qscores: str, R3_qscores: str, qscore_thresh: int) -> bool:

'''Takes the quality scores of a record in reads 2 and 3, averages them, then checks them against a quality score threshold. Returns True if the score is above the threshold, False if it is below'''

        R2 = bioinfo.qual_score(R2_qscores)

        R3 = bioinfo.qual_score(R3_qscores)

        if R2 < qscore_thresh or R3 < qscore_thresh:

                return True
```

```
        else:
                return False
#sample test!
#print(check_q("#AJAAA#","JJJJE#A", 20))
#Input: #AJAAA#, JJJJE#A, 20
#Output: True
```

Initialized sets, variables, and dictionaries at beggining

matched_indexes = set() - set of 24 known indexes from 2017 data

unknown = 0 - where all indexes that have low quality scores, an N base, or aren't in the list of 24 go

matched_dict: dict = {} - where all indexes with high quality scores, no N bases, are matched and in the list of 24 go

hopped_dict: dict = {} - where all indexes that have high quality scores, no N bases, in the list of 24 but are NOT matched to each other go

file_handles: dict = {}

Flow of Code:

- Started by converting matched index text file (matched_indexes.txt) to a set that would be used later for comparison
- Opened all 4 input files, specified "rt" so they would be read as Text not Binary
- Read 4 lines (one record) from each file at the same time
  - Specified that if it encountered an empty line to break the loop, means we reached the end of the file
- Reverse complemented read 3 to match read 2. If the indexes are not hopped, the reverse complement of read 3 should be identical to read 2
- Added the index pair to the end of each header line in reads 1 and 4

- Then checked to see if both index 1 and index 2 were in the set of 24 known indexes
  - if either of them was not, then the record was sent to the unknown file
- Compared converted quality scores to the specified quality score threshold
  - if it passed the threshold the record moved on to the final checkpoint
  - I decided to use a quality score threshold of 20. At Q20, 99% of the read is accurate. 1% inaccuracy is still quite large when using such a large data set, but 20 seemed like a good cutoff to not loose too much data.
- Checked to see if index 1 and index 2 were identical
  - if they were not, that means index hopping had occurred and the record was sent to the hopped file
- If the record made it past all of those checkpoints that means the indexes were known, high quality, and matched

Data Output

I had the script print out the matched and hopped dictionaries to see the indexes that were matched and hopped and how many times each index occurred in those dictionaries. I also printed out the raw number and percent of unknown reads, hopped reads, and matched reads

Unknown:

```
Percent = 8.76%
Amount = 31828861
```

Hopped:

```
Percent = 0.187%
Amount = 679459
```

Matched:

```
Percent = 91.05%
Amount = 330738415
```