Python Dictionaries: Translate DNA/RNA to AA

Write a Python program called `translate.py` that translates a given DNA/RNA sequence to amino acids. The input sequence is expected to be a positional argument. The codon table is expected for the `-c` or `--codons` option and should be a valid, readable file. The `-o` or `--output` option (default `'out.txt'`) will name a file to write the output.

When run with no inputs, the program should print a brief usage:

```
$ ./translate.py
usage: translate.py [-h] -c FILE [-o FILE] str
translate.py: error: the following arguments are required: str, -c/--codons
```

Or a longer usage for the `-h` or `--help` flags:

```
$ ./translate.py -h
usage: translate.py [-h] -c FILE [-o FILE] str

Translate DNA/RNA to proteins

positional arguments:
  str                   DNA/RNA sequence

optional arguments:
  -h, --help            show this help message and exit
  -c FILE, --codons FILE
                        A file with codon translations (default: None)
  -o FILE, --outfile FILE
                        Output filename (default: out.txt)
```

The program should reject a `--codons` argument that fails to name a file:

```
$ ./translate.py -c lkdflk ACTG
usage: translate.py [-h] -c FILE [-o FILE] str
translate.py: error: argument -c/--codons: can't open 'lkdflk': \
[Errno 2] No such file or directory: 'lkdflk'
```

If given good input, write the results to the proper output file. Note here that the input string is lowercase:

```
$ ./translate.py -c codons.dna gaactacaccgttctcctggt
Output written to "out.txt".
```

And now the `out.txt` file should contain the following:

```
$ cat out.txt
ELHRSPG
```

Note that the `-o` flag should make the program write the output to a different file. Note here that the input sequence is uppercase:

```
$ ./translate.py UGGCCAUGGCGCCCAGAACUG --codons codons.rna -o foo.txt
Output written to "foo.txt".
```

Your program will be given the wrong codon table for a given sequence type, e.g., a DNA table for an RNA sequence:

```
$ ./translate.py UGGCCAUGGCGCCCAGAACUG --codons codons.dna
Output written to "out.txt".
```

This means that your program will be unable to find some of the codons in which case it should print a `-` for a missing codon:

```
$ cat out.txt
-P-RPE-
```

If you are creating a dictionary from the codon table, e.g.:

```
$ head -3 codons.rna
AAA K
AAC N
AAG K
```

Such that you have something like this:

```
>>> codons = dict(AAA='K', AAC='N', AAG='K')
```

Everything is fine as long as you ask for codons that are defined but will fail at runtime if you ask for a codon that does not exist:

```
>>> codons['AAC']
'N'
>>> codons['AAT']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'AAT'
```

What dictionary access method will prevent you from creating an exception if you ask for a key that doesn't exist and will also allow you to set a default value?

Extracting k-mers

A k-mer is a `k`-length sequence of contigous characters in a string (a "mer" like "polymer"). Here is a piece of code that you can use to extract all the 3-mers from the sequence `seq`:

```
>>> seq = 'actga'
>>> k = 3
>>> for codon in [seq[i:i + k] for i in range(0, len(seq) - k + 1)]:
...     print(codon)
...
act
ctg
tga
```

You will then use the `codon` as a key to lookup the amino acid translation from the `codons` dictionary you build using the input file.

Tests

A passing test suite looks like this:

```
$ make test
pytest -xv test.py
============================= test session starts ==============================
...
collected 10 items

test.py::test_exists PASSED                                              [ 10%]
test.py::test_usage PASSED                                               [ 20%]
test.py::test_no_args PASSED                                             [ 30%]
test.py::test_missing_input PASSED                                       [ 40%]
test.py::test_missing_codons PASSED                                      [ 50%]
test.py::test_bad_codon_file PASSED                                      [ 60%]
test.py::test_good_input1 PASSED                                         [ 70%]
test.py::test_good_input2 PASSED                                         [ 80%]
test.py::test_good_input3 PASSED                                         [ 90%]
test.py::test_good_input4 PASSED                                         [100%]

============================== 10 passed in 0.50s ==============================
```