Find Common Words with Mismatches

Write a Python program called `commoner.py` that takes exactly two positional arguments which should be text files that you will read and find words that are found to be in common. The program should also accept a `-m|--min_len` option (integer) which is the minimum length for a word to be included (so that we can avoid common short words like articles and "I", etc.) as well as a `-n|--hamming_distance` (integer) value that is the maximum allowed Hamming (edit) distance to consider two words to be the same. There should also be two options for debugging, one `-d|--debug` that turns on logging into a `-l|--logfile` option that defaults to `.log`. Lastly, the program should have a `-t|--table` option that indicates the output should be formatted into an ASCII table using the `tabulate` module (https://pypi.org/project/tabulate/); the default output (that is, without `-t`) should be tab-delimited text.

If there are no words found to be in common, print "No words in common." If there are words, print a header line with "word1," "word2", and "distance" as the column names. Then print each of the words, sorted by the pairs, along with their Hamming distances.

If either of the file inputs are not files, exit with an error and appropriate message. You could use the file handle `type` to `argparse` for the two file inputs as the test does not check for a specific error message. You also do not need to log the names of the input files, so there's that.

If the `--distance` is less than 0, exit with an error and message `--distance "{}" must be > 0`.

Logging

The logging tests do not look for specific messages, only that a non-empty log is created using a given name when `--debug` is present. You can use the same logging code from earlier assignments.

Expected Behavior

```
$ ./commoner.py
usage: commoner.py [-h] [-m int] [-n int] [-l str] [-d] [-t] FILE FILE
commoner.py: error: the following arguments are required: FILE
$ ./commoner.py -h
usage: commoner.py [-h] [-m int] [-n int] [-l str] [-d] [-t] FILE FILE

Find common words
```

```
positional arguments:
  FILE                  Input files

optional arguments:
  -h, --help            show this help message and exit
  -m int, --min_len int
                        Minimum length of words (default: 0)
  -n int, --hamming_distance int
                        Allowed Hamming distance (default: 0)
  -l str, --logfile str
                        Logfile name (default: .log)
  -d, --debug           Debug (default: False)
  -t, --table           Table output (default: False)
$ ./commoner.py data/fox.txt data/fox.txt
word1   word2   distance
brown   brown   0
dog dog 0
fox fox 0
jumps   jumps   0
lazy    lazy    0
over    over    0
quick   quick   0
the the 0
$ ./commoner.py data/fox.txt data/fox.txt -t
+---------+---------+------------+
| word1   | word2   |   distance |
|---------+---------+------------|
| brown   | brown   |          0 |
| dog     | dog     |          0 |
| fox     | fox     |          0 |
| jumps   | jumps   |          0 |
| lazy    | lazy    |          0 |
| over    | over    |          0 |
| quick   | quick   |          0 |
| the     | the     |          0 |
+---------+---------+------------+
$ ./commoner.py -t data/american.txt data/british.txt -m 5 -n 1
+-----------+-----------+------------+
| word1     | word2     |   distance |
|-----------+-----------+------------|
| about     | about     |          0 |
| analyze   | analyse   |          1 |
| faults    | faults    |          0 |
| forgot    | forgot    |          0 |
| generally | generally |          0 |
| improve   | improve   |          0 |
| license   | licence   |          1 |
| merits    | merits    |          0 |
| night     | night     |          0 |
| organize  | organise  |          1 |
| ourselves | ourselves |          0 |
| pretense  | pretence  |          1 |
| recognize | recognise |          1 |
| thoughts  | thoughts  |          0 |
| which     | which     |          0 |
| without   | without   |          0 |
+-----------+-----------+------------+
$ ./commoner.py data/american.txt data/british.txt -m 5 -n 1 | column -t
word1      word2      distance
about      about      0
analyze    analyse    1
faults     faults     0
forgot     forgot     0
generally  generally  0
improve    improve    0
license    licence    1
merits     merits     0
night      night      0
organize   organise   1
ourselves  ourselves  0
pretense   pretence   1
recognize  recognise  1
thoughts   thoughts   0
which      which      0
without    without    0
```

Testing

This test suite is going to mix unit tests *inside* your `commoner.py` program with integration tests in `test.py`. You will need to copy your `dist` function from `13-hamm` and add this (probably just after the `dist` function):

```
def test_dist():
    """dist ok"""

    tests = [('foo', 'boo', 1), ('foo', 'faa', 2), ('foo', 'foobar', 3),
             ('TAGGGCAATCATCCGAG', 'ACCGTCAGTAATGCTAC',
              9), ('TAGGGCAATCATCCGG', 'ACCGTCAGTAATGCTAC', 10)]

    for s1, s2, n in tests:
        d = dist(s1, s2)
        assert d == n
```

You will also need to define a function `def uniq_words(file, min_len):` that takes a file – or open file handle! – and a minimum length. Paste this test below your function definition:

```
def test_uniq_words():
    """Test uniq_words"""

    s1 = '?foo, "bar", FOO: $fa,'
    s2 = '%Apple.; -Pear. ;bANAna!!!'

    assert uniq_words(io.StringIO(s1), 0) == set(['foo', 'bar', 'fa'])

    assert uniq_words(io.StringIO(s1), 3) == set(['foo', 'bar'])

    assert uniq_words(io.StringIO(s2), 0) == set(['apple', 'pear', 'banana'])

    assert uniq_words(io.StringIO(s2), 4) == set(['apple', 'pear', 'banana'])

    assert uniq_words(io.StringIO(s2), 5) == set(['apple', 'banana'])
```

Note that this test is mocking the idea of a file handle; that is, the source for the words will be a file(handle), but for purposes of the test I just want to pass something that can pretend to be a filehandle. Notice how we can use a `for` loop over an `io.String` just like we can an `open` file:

```
>>> import io
>>> file = io.StringIO('foo\nbar baz\nquux!')
>>> for i, line in enumerate(file):
...   print(i, line, end='')
...
0 foo
1 bar baz
2 quux!
```

Lastly define a `def common(words1, words2, distance):` function that takes two lists of words and a maximum Hamming distance and returns a list of tuples containing the two words and the actual distance between the two words if that distance is less than or equal to the maximum allowed. Copy this function just below it.

```
def test_common():
    w1 = ['foo', 'bar', 'quux']
    w2 = ['bar', 'baz', 'faa']

    assert common(w1, w2, 0) == [('bar', 'bar', 0)]
    assert common(w1, w2, 1) == [('bar', 'bar', 0), ('bar', 'baz', 1)]

    assert common(w1, w2, 2) == [('bar', 'bar', 0), ('bar', 'baz', 1),
                                 ('bar', 'faa', 2), ('foo', 'faa', 2)]
```

Once you have written the above functions, I don't think it's helping too much to show you my logic:

```
words1 = uniq_words(fh1, args.min_len)
words2 = uniq_words(fh2, args.min_len)
common_words = common(words1, words2, distance)
```

I think it would help you to think about first getting a unique set of words of the correct length from each file. Then use those words to find the ones in common. Were we not concerned about the Hamming distance, we could use a `set` for the words and do `words1.intersection(words2)`, but we have to instead to a pair-wise comparison of every `word1` to every `word2`! That is most easily accomplished by using `itertools.product`.

Test Suite

The Makefile's `test` target is `pytest -v commoner.py test.py`. Notice how it's looking in both your `commoner.py` program for `test_` functions as well as the `test.py`. Again, the point here is to build small, testable functions inside your program and integrate the tests directly into the program. Then `test.py` is used to ensure that the *user interface* works; that is, your program generates a usage, it emits error codes on errors, it honors the expected flags and arguments, etc.

```
$ make test
pytest -v commoner.py test.py
============================= test session starts ==============================
platform darwin -- Python 3.6.8, pytest-4.2.0, py-1.7.0, pluggy-0.8.1 -- /anaconda3/bin/pyth
cachedir: .pytest_cache
rootdir: /Users/kyclark/work/worked_examples/15-commoner, inifile:
plugins: remotedata-0.3.1, openfiles-0.3.2, doctestplus-0.2.0, arraydiff-0.3
collected 7 items

commoner.py::test_dist PASSED                                            [ 14%]
commoner.py::test_uniq_words PASSED                                      [ 28%]
commoner.py::test_common PASSED                                          [ 42%]
test.py::test_usage PASSED                                               [ 57%]
test.py::test_bad_n PASSED                                               [ 71%]
test.py::test_bad_input PASSED                                           [ 85%]
test.py::test_runs_ok PASSED                                             [100%]
```

```
=========================== 7 passed in 2.94 seconds ============================
```