Python Gradaute - Parsing SwissProt

> “Without requirements or design, programming is the art of adding bugs to an empty text file.” - Louis Srygley

Create a Python program called “swisstake.py” that processes a SwissProt-formatted file as a positional argument. It should have a *required* `-k|--keyword` argument of the keyword to match in the “keyword” field of the input record in order to determine which sequences to “take” (hence the name). It should also have an *optional* `-s|--skip` argument to “skip” records with given taxa (which could be many so `nargs='+'`), as well as an *optional* `-o|--output` argument to where to write the output in FASTA format (default “out.fa”).

If the given input file is not a file, it should die with ‘ “XXX” is not a file’.

```
$ ./swisstake.py -h
usage: swisstake.py [-h] [-s STR [STR ...]] -k STR [-o FILE] FILE

Filter Swissprot file for keywords, taxa

positional arguments:
  FILE                  Uniprot file

optional arguments:
  -h, --help            show this help message and exit
  -s STR [STR ...], --skip STR [STR ...]
                        Skip taxa (default: )
  -k STR, --keyword STR
                        Take on keyword (default: None)
  -o FILE, --output FILE
                        Output filename (default: out.fa)
$ ./swisstake.py swiss.txt
usage: swisstake.py [-h] [-s STR [STR ...]] -k STR [-o FILE] FILE
swisstake.py: error: the following arguments are required: -k/--keyword
$ ./swisstake.py -k proteome foo
"foo" is not a file
$ ./swisstake.py swiss.txt -k "complete proteome" -s Metazoa FUNGI viridiplantae
Processing "swiss.txt"
Done, skipped 14 and took 1. See output in "out.fa".
$ ./swisstake.py swiss.txt -k "complete proteome" -s metazoa fungi
Processing "swiss.txt"
Done, skipped 13 and took 2. See output in "out.fa".
```

BioPython SwissProt Record

A FASTA record had three attributes: ID, description, and sequence. A SwissProt record has considerably more which will make sense once you look at the file.

There are at least two ways I've found to parse a SwissProt record. One is use `SeqIO.parse(fh, 'swiss')` which gives you a record very similar to a FASTA record which has an `annotations` attribute which is a dictionary that looks like this:

```
>>> import pprint
>>> pp = pprint.PrettyPrinter(indent=4)
>>> pp.pprint(rec.annotations)
{   'accessions': ['P13813'],
    'date': '01-JAN-1990',
    'date_last_annotation_update': '20-JAN-2016',
    'date_last_sequence_update': '01-JAN-1990',
    'entry_version': 42,
    'keywords': ['Malaria', 'Repeat'],
    'ncbi_taxid': ['5850'],
    'organism': 'Plasmodium knowlesi',
    'protein_existence': 2,
    'references': [   Reference(title='Cloning and characterization of an abundant Plasmodiu
    'sequence_version': 1,
    'taxonomy': [   'Eukaryota',
                    'Alveolata',
                    'Apicomplexa',
                    'Aconoidasida',
                    'Haemosporida',
                    'Plasmodiidae',
                    'Plasmodium',
                    'Plasmodium (Plasmodium)']}
```

The other way is use the `Bio.SwissProt` module which has attributes for the same kind of information though sometimes called slightly different names, e.g.:

```
>>> sw1.organism_classification
['Eukaryota', 'Alveolata', 'Apicomplexa', 'Aconoidasida', 'Haemosporida', 'Plasmodiidae', 'P
```

Cf:

- https://biopython.readthedocs.io/en/latest/Tutorial/chapter_uniprot.html)
- http://biopython.org/DIST/docs/api/Bio.SwissProt.Record-class.html)

However you choose to parse, you should be able to pass the tests. FWIW, I used the first method.

Sets

We've talked about dictionaries quite a bit, and for this exercise I think you'll want something that is a natural extension of a dictionary called `set()` which is just a dictionary where the values are all `1`. If you have two lists, you can test for equality:

```
>>> a = ['foo', 'bar']
>>> b = ['foo', 'bar']
>>> a == b
True
>>> c = ['bar', 'foo']
>>> a == c
False
```

The list `c` has the same members but in a different order, so the lists definitely are not the same; however, if all you cared about what if the two lists shared the same items, you could sort them:

```
>>> sorted(a) == sorted(c)
True
```

But what if you wanted to know if there was some overlap. Clearly you can't use equality:

```
>>> d = ['foo', 'bar', 'baz']
>>> a == d
False
```

You have to individually check each element of `a` to see if they are in `d`:

```
>>> [e for e in a if e in d]
['foo', 'bar']
>>> [e for e in d if e in a]
['foo', 'bar']
>>> any([e for e in a if e in d])
True
```

That is the "intersection" of the two lists. The "difference" would be:

```
>>> [e for e in a if e not in d]
[]
>>> [e for e in d if e not in a]
['baz']
```

The "union" would be everything in both lists, which we can't easily do in one line of code; however, if we convert these lists to sets, then we can do all those calculations easily:

```
>>> a = set(['foo', 'bar'])
>>> d = set(['foo', 'bar', 'baz'])
>>> a.union(d)
set(['baz', 'foo', 'bar'])
>>> a.intersection(d)
set(['foo', 'bar'])
>>> a.difference(d)
set([])
>>> d.difference(a)
set(['baz'])
```

Keep this in mind when you are trying to find if there is an intersection of the taxa you are given with the taxa that are in the record.

Test Suite

A passing test suite looks like this:

```
$ make test
python3 -m pytest -v 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/07-grad-swissprot, inifile:
plugins: remotedata-0.3.1, openfiles-0.3.2, doctestplus-0.2.0, arraydiff-0.3
collected 3 items

test.py::test_usage PASSED                                               [ 33%]
test.py::test_bad_input PASSED                                           [ 66%]
test.py::test_good_input1 PASSED                                         [100%]

=========================== 3 passed in 1.75 seconds ===========================
```