Problem 1. (*Watson-Crick Complement*) Implement the function `wc_complement()` in `wc_complement.py` that takes a DNA string as argument and returns its *Watson-Crick complement*: replace A with T, C with G, and vice versa. You may assume that the characters in the DNA string are all in upper case.

```
$ python wc_complement.py ACTGACG
TGACTGC
```

Problem 2. (*Domain Type*) Implement the function `domain_type()` in `domain_type.py` that returns the domain type of the URL specified as argument. For example, the domain type of the URL `http://www.swamiiyer.net/cs110/` is `net`. You may assume that the URL starts with `'http://'` and ends with `'/'`. Hint: use the `str` methods `find()` and `split()`.

```
$ python domain_type.py http://www.swamiiyer.net/cs110/
net
```

Problem 3. (*Password Checker*) Implement the function `is_valid()` in `password_checker.py` that returns `True` if the given password string meets the following specifications, and `False` otherwise:

- At least eight characters long.
- Contains at least one digit (0-9).
- Contains at least one uppercase letter.
- Contains at least one lowercase letter.
- Contains at least one character that is neither a letter nor a number.

Hint: use the `str` methods `isdigit()`, `isupper()`, `islower()`, and `isalnum()`.

```
$ python password_checker.py Abcde1fg
False
$ python password_checker.py Abcde1@g
True
```

Problem 4. (*Set Distance*) The *Jaccard index* measures the similarity between finite sample sets, and is defined as the size of the intersection divided by the size of the union of the sample sets:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}.$$

Note that $0 \leq J(A,B) \leq 1$. The *Jaccard distance*, which measures dissimilarity between sample sets, is complementary to the Jaccard index and is obtained by subtracting the Jaccard index from 1:

$$d_J(A,B) = 1 - J(A,B)$$

.
Implement the functions `jaccard_index()` and `jaccard_distance()` in `set_distance.py` that take two sets A and B as arguments and return their Jaccard index and Jaccard distance, respectively. Hint: use the `set` methods `intersection()` and `union()`.

```
$ python set_distance.py "b, c" "a, b, c, d"
0.5
$ python set_distance.py "7, 3, 2, 4, 1" "4, 1, 9, 7, 5"
0.571428571429
```

Problem 5. (*Word Frequencies*) Implement the function `count_word_frequencies()` in `word_frequencies.py` that takes a list of words as argument and returns a dictionary whose keys are the words from the list and values are the corresponding frequencies. Also implement the function `write_word_frequencies()` that takes a dictionary as argument and writes (in reverse order of values) the key-value pairs of the dictionary to standard output, one per line, and with a `' -> '` between a key and the corresponding value. Hint: Use `dict` method `setdefault()` for the first part and use `word_frequencies.keys()` for the second part.

```
$ python word_frequencies.py
it was the best of times it was the worst of times
<ctrl-d>
of -> 2
it -> 2
times -> 2
the -> 2
was -> 2
worst -> 1
best -> 1
```

Files to Submit

1. `wc_complement.py`
2. `domain_type.py`
3. `password_checker.py`
4. `set_distance.py`
5. `word_frequencies.py`

Before you submit:

- Make sure your programs meet the input and output specifications by running the following command on the terminal:

  ```
  $ python run_tests.py -v [<problems>]
  ```

  where the optional argument `<problems>` lists the problems (`Problem1`, `Problem2`, etc.) you want to test; all the problems are tested if no argument is given.

- Make sure your programs meet the style requirements by running the following command on the terminal:

  ```
  $ pep8 <program>
  ```

  where `<program>` is the `.py` file whose style you want to check.