ISTA 350 Regex, More Worksheet Name:

Write a function called `dog_words` that takes a filename. Use a regex to return a list of all occurrences of words that start with `'dog'`. A dog word is immediately preceded by whitespace or the beginning of string and is immediately followed by whitespace, `'.'`, `','`, `':'`, or the end of string. It is all letters, or `'dog'` followed by a hyphen, followed by more letters. Use a negative lookahead assertion to avoid returning any words starting with `'dogcatcher'`. The syntax for a negative lookahead assertion is `(?!…)` where … is the pattern to avoid. The positive lookahead assertion syntax is `(?=…)` and the positive lookbehind is `(?<=…)`. Recall the following special sequences/metacharacters: `\A` – start of string, `\Z` – end of string, `[]` – character sets, `\s` – whitespace, `()` – grouping, `|` – or, `?` – zero or one occurrence, and `*` – zero or more occurrences.

Given a variable `list_of_strings`, use a `for` loop to make a list that contains the first three letters of all of the strings in `list_of_strings`. Turn it into a list comprehension.

E.g. `['batman', 'starlord'] -> ['bat', 'sta']`.

Given a variable `list_of_ints`, use a `for` loop to make a list that contains the squares of all of the numbers in `list_of_ints`. Turn it into a list comprehension.

E.g. `[1, 2, 3] -> [1, 4, 9]`.

If you can't understand what a spec is asking for, you can't fulfill it. So we're going to practice that. Here is the specification for `init` from hw1:

```
class WatchList:
```

`init`: This magic method takes a filename that defaults to the empty string. Initialize an instance variable called `bills` to a dictionary that maps each of the five denominations of interest, represented as strings (i.e. `'5'`, `'10'`, etc.), to an empty list. If a filename was passed in, each line of the file will represent a bill that we want to add to our watch list dictionary and will be in the format `'<serial_number> <denomination>\n'`. Look at one of the bill files in a text editor to see specific examples. Append the serial number for each bill in the file to the appropriate list in the dictionary. A Boolean instance variable called `is_sorted` indicates whether or not the lists in the dictionary are sorted. Assume that the bill files are not sorted. Finally, an instance variable called `validator` holds a compiled regular expression that will be used to check for valid serial numbers (see the **Introduction** above for the rules governing serial numbers).

Draw a `WatchList` with two $10 bills in it, a $20 bill, and three $100 bills.

Hw2 asks you to implement a parse tree in two different ways. The `WatchListLinked` class will build its parse tree using the `Node` class you implemented in the last worksheet (see the next page for a memory refresher). Here is the spec for its `init`:

`class WatchListLinked:`

`init`: This magic method takes a filename that defaults to the empty string. Initialize an instance variable called `root` to a node that has 5 children. The `datum` in the node should be the `None` object (let your default arg do the work, don't pass it in). The data in the 5 children are the 5 denomination strings (i.e. `'5', '10'`, etc.), respectively. The children have no children of their own. If a filename was passed in, each line of the file will represent a bill that we want to add to our watch list dictionary and will be in the format `'<serial_number> <denomination>\n'`. Look at one of the bill files in a text editor to see specific examples. Insert each of the bills into the watch list. Finally, an instance variable called `validator` holds a compiled regular expression that will be used to check for valid serial numbers (use the same regex as you did in hw1).

Draw an empty watch list. Draw a watch list that has had the following strings parsed and inserted: `"rock 5", "rocket 5", "rock'n'roll 5", "rocker 5", "rocking 5"`. The last character in each string will have as one of its children an empty node, i.e. its datum is `None` and it has no children. This is how we store the fact that we have reached the end of a string so that we can store, for instance, both the strings `"app"` and `"apple"`.

Recall the `Node` class that we created on the last worksheet:

```
class Node:
    def __init__(self, datum=None):
        self.datum = datum
        self.children = []
```

Write code that creates an empty `WatchListLinked` instance and inserts the strings `"abc 5"` and `"abef 5"` into it.

Here is the spec for class `WatchListDict`:

`init`: This magic method takes a filename that defaults to the empty string. Initialize an instance variable called `root` to a dictionary that maps each of the 5 denomination strings (i.e. `'5', '10'`, etc.) to an empty dictionary. If a filename was passed in, each line of the file will represent a bill that we want to add to our watch list dictionary and will be in the format `'<serial_number> <denomination>\n'`. Look at one of the bill files in a text editor to see specific examples. Insert each of the bills into the watch list. Finally, an instance variable called `validator` holds a compiled regular expression that will be used to check for valid serial numbers (use the same regex as you did in hw1).

Draw an empty watch list. The terminator for a string will be a dictionary that maps `None` to `None`. Draw a watch list that has had the following strings parsed and inserted: `"abc 5", "abef 5", "abcd 5", "abce 5", "abcde 5"`.