# Question 1

The following code snippet can be used to find the past perfect constructions from Whitman's *Leaves of Grass*.

```
>>> import re
>>> from nltk.corpus import gutenberg
>>> leaves_of_grass = gutenberg.raw('whitman-leaves.txt')
>>> past_perfect = re.compile(r'had\s\w+\'d')
>>> past_perfect_constructions = past_perfect.findall(leaves_of_grass)
```

In all, there are 5 past perfect constructions in the text. They are:

```
["had form'd", "had receiv'd", "had receiv'd", "had father'd", "had conceiv'd"]
```

# Question 2

1. `r'\w+er\b|\w+est\b'`

2. `r'\w+ment\b'`

3. `r'\bin\w+|\bim\w+|\bun\w+|\bnon\w+'`

4. `r'\w+ly\s\w{1,7}\b'`

Yes, the first three regular expressions will capture words that do not have affixes.

I tried the above mentioned regular expressions on the *Leaves of Grass* text in Project Gutenberg. In the first regular expression searching for words with comparative/superlative suffix *-er*, *-est*, words for which they are not suffixes such as 'water', 'rest' are captured as well. In the second regular expression searching for words with the suffix *-ment*, words for which it's not the suffix such as 'monument', 'garment', etc. are captured as well. In the third regular expression searching for words beginning with the given prefixes, words for which they are not the prefixes such as 'immense', 'impel', 'under' etc. are captured as well. This shows that the given patterns are not sufficient to capture the corresponding properties.

# Question 3

1. The word pattern I looked for is the words with suffix *-ing*, which are the present participle forms of a verb.

2. 
```
r'\w+ing\b'
```

3. 
```
>>> import re
>>> from nltk.corpus import gutenberg
>>> leaves_of_grass = gutenberg.raw('whitman-leaves.txt')
>>> participle = re.compile(r'\w+ing\b')
>>> participle_constructions = re.findall(leaves_of_grass)
```

   I used Whitman's *Leaves of Grass* text from Project Gutenberg for this exervise. The present participle words are those forms of a verb which end in *-ing*. In the second line of the snippet, 'gutenberg' module was imported, from which the file 'whitman-leaves.txt' was extracted as a raw text file. On this text, the regular expression was explained. Upon running the above code snippet, it was observed that while present participle words, i.e verbs ending in *-ing* were found, few other words in the text such as 'sing', 'spring', etc. were also captured by the regular expression, which are not present participles.

# Question 4

```
>>> import re
>>> from nltk.corpus import gutenberg
>>> text = gutenberg.raw("whitman-leaves.txt")
>>> old_orthography = re.compile(r'\w+\'d\b')
>>> modern_orthography = re.compile(r'\w+ed\b')
>>> old_orthography_seq = re.compile(r'\'d')
>>> normalized_text = old_orthography_seq.sub('ed', text)
```

The above code snippet replaces the *'d* suffix in the old orthography text to *-ed* in the normalized text.

```
>>> print(len(re.findall(old_orthography, normalized_text)))
```

From the above code snippet, whose output was '0', it can be concluded that all *'d* in Leaves of Grass are replaced by *-ed* suffix.
Few of the words with the suffix *-ed* in the new text are:

```
['pleased',
 'formed',
 'Pondered',
 'pondered',
 'answered',
 'Waged',
 'deferred',
 'Cabined'...]
```

# Question 5

First, the text was searched upon to find apostrophe's in the middle of a word. Such words were identified, and from among those, the cases where apostrophe is preceded by 'o', followed by 's' were eliminated. The following code snippet is useful to extract the apostrophe-containing words.

```
>>> import re
>>> from nltk.corpus import gutenberg
>>> text = gutenberg.raw("whitman-leaves.txt")
>>> middle_apostrophe = re.compile(r"\w+\'\w+")
>>> print(re.findall(middle_apostrophe, text))
```

From the remaining words, another instance of an apostrophe which needed normalization was *eé* where the apostrophe replaces a "v". The regular expression which denotes such words is **r'\w+e\'e\w+'**. The code snippet below is useful to extract such words:

```
>>> apostrophe = re.compile(r"\w+e\'e\w+")
>>> re.findall(apostrophe, text)
```

Words with this pattern *-eve-* in the text are:

```
["wheresoe'er", "whoe'er", "whoe'er", "ne'er", "whoe'er", "whate'er"]
```

The following code snippet replaces the *e'e* in the old orthography text to *-eve-* in the normalized text.

```
>>> old_orthography_seq = r"e\'e"
>>> normalized_text = old_orthography_seq.sub('eve', text)
>>> print(len(re.findall(old_orthography, normalized_text)))
```

From the above code snippet, whose output was '0', it can be concluded that all *e'e* in Leaves of Grass are replaced by *-eee* and the final output is

# Question 6

```
>>> untokenized = ' '.join(w for w in nltk.corpus.treebank.words()
                           if not w.startswith('*') and w is not '0')
>>> pattern = r"[A-Za-z]+|--|[.,?]|\$ [0-9,]+[.]?[0-9]* (?:million|billion|thousand)"
>>> tokenized = nltk.regexp_tokenize(text=untokenized, pattern=pattern)
>>> money_expressions = [token for token in tokenized if re.search(r'\$', token)]
```

The above code snippet looks for money expressions of the form "$abcd.efgh units", which is expressed in the 'pattern' argument for *regexp_tokenize()*, and also stored in the pattern variable. Finally, the money expressions are searched in the list of tokens through the '$' symbol. This particular regular expression also looks for decimal numbers in money expressions, which the given regular expression does not.

The first 100 money expressions are:

```
['$ 1.5 billion', '$ 352.7 billion', '$ 212 million', '$ 295 million',
'$ 370 million', '$ 101 million', '$ 5.29 billion', '$ 5.39 billion',
'$ 68 billion', '$ 71 million', '$ 50.45 billion', '$ 50.38 billion',
'$ 2 billion', '$ 2.29 billion', '$ 2.25 billion', '$ 2.2 billion',
'$ 250 million', '$ 55 million', '$ 737.5 million', '$ 2.5 billion',
'$ 49 million', '$ 190 million', '$ 195 million', '$ 55 million',
'$ 140 million', '$ 245 million', '$ 72.7 million', '$ 100 million',
'$ 120 million', '$ 98.3 million', '$ 15 million', '$ 30 million',
'$ 53 million', '$ 19.3 million', '$ 5.9 million', '$ 20.5 million',
'$ 64 billion', '$ 38.3 billion', '$ 23 million', '$ 10.2 million',
'$ 37.3 million', '$ 1.5 billion', '$ 701 million', '$ 570 million',
'$ 900 billion', '$ 2.4 billion', '$ 1.82 billion', '$ 84.29 billion',
'$ 2.2 billion', '$ 86.12 billion', '$ 188 million', '$ 188 million',
'$ 236.74 billion', '$ 236.79 billion', '$ 415.6 billion', '$ 415.8 billion',
'$ 109.73 billion', '$ 127.03 billion', '$ 234.4 billion', '$ 497.34 billion',
'$ 191.9 billion', '$ 99.1 billion', '$ 88 billion', '$ 19 million',
'$ 450 million', '$ 2 billion', '$ 2.44 million', '$ 85.1 million',
'$ 5.57 billion', '$ 705.6 million', '$ 150 million', '$ 2.4 billion',
'$ 8 million', '$ 8 billion', '$ 273.5 million', '$ 70.7 million',
'$ 89.9 million', '$ 4.8 million', '$ 225.6 million', '$ 11 million',
'$ 7 million', '$ 120 million', '$ 32.8 million', '$ 28.6 million',
'$ 29.3 million', '$ 28.4 million', '$ 280 million', '$ 35.7 million',
'$ 2 million', '$ 3 million', '$ 43 million', '$ 35.7 million',
'$ 3.1 million', '$ 7.5 million', '$ 133.7 million', '$ 94 million',
'$ 16 billion', '$ 75 million', '$ 150 million', '$ 5 million']
```