TextProcessing

February 21, 2015

1 Working with text

One of Python's strengths is the ease of working with text. Here are some examples.

1.1 String methods

```
In [10]: # multi-line strings use triple quotes
         s = """
         it was the best of times,
         it was the worst of times,
         it was the age of wisdom,
         it was the age of foolishness,
         it was the epoch of belief,
         it was the epoch of incredulity,
         it was the season of Light,
         it was the season of Darkness,
         it was the spring of hope,
         it was the winter of despair,
         """

         print s.count('of')
         print s.find('wisdom')
         print s.find('foolsihness')
```

```
10
72
-1
```

```
In [11]: print s.upper()
```

```
IT WAS THE BEST OF TIMES,
IT WAS THE WORST OF TIMES,
IT WAS THE AGE OF WISDOM,
IT WAS THE AGE OF FOOLISHNESS,
IT WAS THE EPOCH OF BELIEF,
IT WAS THE EPOCH OF INCREDULITY,
IT WAS THE SEASON OF LIGHT,
IT WAS THE SEASON OF DARKNESS,
IT WAS THE SPRING OF HOPE,
IT WAS THE WINTER OF DESPAIR,
```

```
In [12]: print s.replace('was', 'might have been')
```

```
it might have been the best of times,
it might have been the worst of times,
it might have been the age of wisdom,
it might have been the age of foolishness,
it might have been the epoch of belief,
it might have been the epoch of incredulity,
it might have been the season of Light,
it might have been the season of Darkness,
it might have been the spring of hope,
it might have been the winter of despair,
```

1.2 Splitting and joining strings

```
In [13]: paths = !‘echo $PATH‘
         print paths[0]

/bin/sh: 1: /home/bitnami/anaconda/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr

In [14]: for path in paths[0].split(’:’):
             print ’=> ’.join(path.strip().split(’/’))

=> bin=> sh
1
=> home=> bitnami=> anaconda=> bin
=> usr=> local=> sbin
=> usr=> local=> bin
=> usr=> sbin
=> usr=> bin
=> sbin
=> bin
=> usr=> games
=> usr=> local=> games
not found
```

2 The string module

The string module provides a very useful `maketrans` function. It is easier to show than to explain what this does.

```
In [15]: from string import maketrans

         dna_to_rna = maketrans(’ACTG’, ’ACUG’)

         dna = ’gattaca’
         print dna.upper().translate(dna_to_rna).lower()

gauuaca

In [16]: # Incidentally the translate function is useful for getting rid of unwanted characters in a st

         from string import punctuation
         print punctuation

!"#$%&’()*+,-./:;<=>?@[\]^_‘{|}~
```

```
In [17]: import os

         # Alice in Wonderland from Project Gutenberg

         if not os.path.exists('alice.txt'):
             ! wget http://www.gutenberg.org/cache/epub/11/pg11.txt -O alice.txt

In [18]: from collections import Counter

         # Remove
         alice = open('alice.txt').read()
         words = alice.translate(None, punctuation).lower().split()
         word_counts = Counter(words)
         for item in word_counts.most_common(10):
             print item
         print 'alice', word_counts['alice']
```

```
('the', 1804)
('and', 912)
('to', 801)
('a', 684)
('of', 625)
('it', 541)
('she', 538)
('said', 462)
('you', 429)
('in', 428)
alice 385
```

2.1 Regular expressions

Regular expressions are a domain specific language for flexible text processing. It is a useful tool, but can be hard to deciper unless you use it often. Where possible, use string methods in preference to regular expressions. Sometimes, however, regular expressions are extremely useful. We will illustrate their use for motif finding in DNA sequences.

See Regular Expression HOWTO and the re documnetation for details.

```
In [19]: # Here is the E Coli DNA sequnce for the beta-D-galactosidase enzyme.

         gene = """
         >ENA|BAE76126|BAE76126.1 Escherichia coli str. K-12 substr. W3110 beta-D-galactosidase
         ATGACCATGATTACGGATTCACTGGCCGTCGTTTTACAACGTCGTGACTGGGAAAACCCT
         GGCGTTACCCAACTTAATCGCCTTGCAGCACATCCCCCTTTCGCCAGCTGGCGTAATAGC
         GAAGAGGCCCGCACCGATCGCCCTTCCCAACAGTTGCGCAGCCTGAATGGCGAATGGCGC
         TTTGCCTGGTTTCCGGCACCAGAAGCGGTGCCGGAAAGCTGGCTGGAGTGCGATCTTCCT
         GAGGCCGATACTGTCGTCGTCCCCTCAAACTGGCAGATGCACGGTTACGATGCGCCCATC
         TACACCAACGTGACCTATCCCATTACGGTCAATCCGCCGTTTGTTCCCACGGAGAATCCG
         ACGGGTTGTTACTCGCTCACATTTAATGTTGATGAAAGCTGGCTACAGGAAGGCCAGACG
         CGAATTATTTTTGATGGCGTTAACTCGGCGTTTCATCTGTGGTGCAACGGGCGCTGGGTC
         GGTTACGGCCAGGACAGTCGTTTGCCGTCTGAATTTGACCTGAGCGCATTTTTACGCGCC
         GGAGAAAACCGCCTCGCGGTGATGGTGCTGCGCTGGAGTGACGGCAGTTATCTGGAAGAT
         CAGGATATGTGGCGGATGAGCGGCATTTTCCGTGACGTCTCGTTGCTGCATAAACCGACT
         ACACAAATCAGCGATTTCCATGTTGCCACTCGCTTTAATGATGATTTCAGCCGCGCTGTA
         CTGGAGGCTGAAGTTCAGATGTGCGGCGAGTTGCGTGACTACCTACGGGTAACAGTTTCT
         TTATGGCAGGGTGAAACGCAGGTCGCCAGCGGCACCGCGCCTTTCGGCGGTGAAATTATC
```

```
GATGAGCGTGGTGGTTATGCCGATCGCGTCACACTACGTCTGAACGTCGAAAACCCGAAA
CTGTGGAGCGCCGAAATCCCGAATCTCTATCGTGCGGTGGTTGAACTGCACACCGCCGAC
GGCACGCTGATTGAAGCAGAAGCCTGCGATGTCGGTTTCCGCGAGGTGCGGATTGAAAAT
GGTCTGCTGCTGCTGAACGGCAAGCCGTTGCTGATTCGAGGCGTTAACCGTCACGAGCAT
CATCCTCTGCATGGTCAGGTCATGGATGAGCAGACGATGGTGCAGGATATCCTGCTGATG
AAGCAGAACAACTTTAACGCCGTGCGCTGTTCGCATTATCCGAACCATCCGCTGTGGTAC
ACGCTGTGCGACCGCTACGGCCTGTATGTGGTGGATGAAGCCAATATTGAAACCCACGGC
ATGGTGCCAATGAATCGTCTGACCGATGATCCGCGCTGGCTACCGGCGATGAGCGAACGC
GTAACGCGAATGGTGCAGCGCGATCGTAATCACCCGAGTGTGATCATCTGGTCGCTGGGG
AATGAATCAGGCCACGGCGCTAATCACGACGCGCTGTATCGCTGGATCAAATCTGTCGAT
CCTTCCCGCCCGGTGCAGTATGAAGGCGGCGGAGCCGACACCACGGCCACCGATATTATT
TGCCCGATGTACGCGCGCGTGGATGAAGACCAGCCCTTCCCGGCTGTGCCGAAATGGTCC
ATCAAAAAATGGCTTTCGCTACCTGGAGAGACGCGCCCGCTGATCCTTTGCGAATACGCC
CACGCGATGGGTAACAGTCTTGGCGGTTTCGCTAAATACTGGCAGGCGTTTCGTCAGTAT
CCCCGTTTACAGGGCGGCTTCGTCTGGGACTGGGTGGATCAGTCGCTGATTAAATATGAT
GAAAACGGCAACCCGTGGTCGGCTTACGGCGGTGATTTTGGCGATACGCCGAACGATCGC
CAGTTCTGTATGAACGGTCTGGTCTTTGCCGACCGCACGCCGCATCCAGCGCTGACGGAA
GCAAAACACCAGCAGCAGTTTTTCCAGTTCCGTTTATCCGGGCAAACCATCGAAGTGACC
AGCGAATACCTGTTCCGTCATAGCGATAACGAGCTCCTGCACTGGATGGTGGCGCTGGAT
GGTAAGCCGCTGGCAAGCGGTGAAGTGCCTCTGGATGTCGCTCCACAAGGTAAACAGTTG
ATTGAACTGCCTGAACTACCGCAGCCGGAGAGCGCCGGGCAACTCTGGCTCACAGTACGC
GTAGTGCAACCGAACGCGACCGCATGGTCAGAAGCCGGGCACATCAGCGCCTGGCAGCAG
TGGCGTCTGGCGGAAAACCTCAGTGTGACGCTCCCCGCCGCGTCCCACGCCATCCCGCAT
CTGACCACCAGCGAAATGGATTTTTGCATCGAGCTGGGTAATAAGCGTTGGCAATTTAAC
CGCCAGTCAGGCTTTCTTTCACAGATGTGGATTGGCGATAAAAAACAACTGCTGACGCCG
CTGCGCGATCAGTTCACCCGTGCACCGCTGGATAACGACATTGGCGTAAGTGAAGCGACC
CGCATTGACCCTAACGCCTGGGTCGAACGCTGGAAGGCGGCGGGCCATTACCAGGCCGAA
GCAGCGTTGTTGCAGTGCACGGCAGATACACTTGCTGATGCGGTGCTGATTACGACCGCT
CACGCGTGGCAGCATCAGGGGAAAACCTTATTTATCAGCCGGAAAACCTACCGGATTGAT
GGTAGTGGTCAAATGGCGATTACCGTTGATGTTGAAGTGGCGAGCGATACACCGCATCCG
GCGCGGATTGGCCTGAACTGCCAGCTGGCGCAGGTAGCAGAGCGGGTAAACTGGCTCGGA
TTAGGGCCGCAAGAAAACTATCCCGACCGCCTTACTGCCGCCTGTTTTGACCGCTGGGAT
CTGCCATTGTCAGACATGTATACCCCGTACGTCTTCCCGAGCGAAAACGGTCTGCGCTGC
GGGACGCGCGAATTGAATTATGGCCCACACCAGTGGCGCGGCGACTTCCAGTTCAACATC
AGCCGCTACAGTCAACAGCAACTGATGGAAACCAGCCATCGCCATCTGCTGCACGCGGAA
GAAGGCACATGGCTGAATATCGACGGTTTCCATATGGGGATTGGTGGCGACGACTCCTGG
AGCCCGTCAGTATCGGCGGAATTCCAGCTGAGCGCCGGTCGCTACCATTACCAGTTGGTC
TGGTGTCAAAAATAA
"""
```

In [20]:
```
# Suppose we want to replace motifs that start with 'ATA',
# followed by between 1 and 4 of any nucleotide, followed by 'CG'
# with a blank string of the same length

from toolz import partition

def replace(match):
    return ' ' * len(match.group(0))

# convert FASTA into single DNA sequence
dna = ''.join(line for line in gene.strip().split('\n')
              if not line.startswith('>'))
pattern = 'ATA.{1,4}CG'
modified_dna = re.sub(pattern, replace, dna)
```

```
    # pretty print modified sequence
    linewidth = 60
    print '\n'.join(['',join(line) for line
                    in partition(linewidth, modified_dna)])
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)

<ipython-input-20-d4424a9550aa> in <module>()
     12                if not line.startswith('>'))
     13 pattern = 'ATA.{1,4}CG'
---> 14 modified_dna = re.sub(pattern, replace, dna)
     15
     16 # pretty print modified sequence


NameError: name 're' is not defined
```

2.2 The NLTK toolkit

If you will be doing statistical natural language processing or significant amounts of machhine learning on natural text, check out the Natural Language Toolkit.

2.3 Exercises

1. Write a function to find the complementary strand given a DNA sequence. For example
Given ATCGTTA Return TAGCAAT
Note: The following are complementary bases A|T, C|G.

```
In []: # YOUR CODE HERE
```

2. Write a regular expression that matches the following:

- Phone numbers with the format: (919)-1234567 (i.e. (123)-9876543 should match but not 234-1234567 or (123)-666666)
- Email addresss john.doe@duke.edu (i.e. steve@gmail.com should match but not steve@gmail)
- DNA seqences with the motif A-C-T-G where - indicates 0 or 1 other nucleotide (any of A,C,T or G)

```
In []: # YOUR CODE HERE
```

3. Download 'Pride and Prejudice' by Jane Austin from Project Gutenbrrg.

- Remove all punctuation and covert to lower case
- Count how many times the word 'married' appears
- Count how often the word 'daughter' and 'married' appear in the same 10-word window

```
In []: # YOUR CODE HERE
```

4. Download "The Gutenberg Webster's Unabridged Dictionary" from Project Gutenbrrg

- First extract all defined words (109561 words)
- Count the number of *defined* English words containing 3 or more vowels (aeiou)
- Find all longest palindromes (a palindrome is a word that is spelt the same forwards as backwards - e.g. 'deified')

```
In []: # YOUR CODE HERE
```