# TextProcessing-Solutions

February 21, 2015

## 1 Workign with text

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

### 1.1 String methods

```
In [1]: # 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 [2]: 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 [3]: 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 [4]: paths = !‘echo $PATH‘
        print paths[0]
```

```
/bin/sh: /usr/local/bin:/Users/cliburn/git/julia/:/Developer/NVIDIA/CUDA-6.5/bin:/Users/cliburn/anaconda
```

```
In [5]: for path in paths[0].split(':'):
            print '=> '.join(path.strip().split('/'))
```

```
=> bin=> sh
=> usr=> local=> bin
=> Users=> cliburn=> git=> julia=>
=> Developer=> NVIDIA=> CUDA-6.5=> bin
=> Users=> cliburn=> anaconda=> bin
=> usr=> bin
=> bin
=> usr=> sbin
=> sbin
=> usr=> local=> bin
=> opt=> X11=> bin
=> usr=> texbin
No such file or directory
```

## 2 The string module

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

```
In [6]: from string import maketrans

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

        dna = 'gattaca'
        print dna.upper().translate(dna_to_rna).lower()
```

```
gauuaca
```

```
In [7]: # Incidentally the translate function is useful for getting rid of unwanted characters in a str

        from string import punctuation
        print punctuation
```

```
!"#$%&'()*+,-./:;<=>?@[\]^_‘{|}~
```

```
In [8]: 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 [9]: 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. Sometiems, however, regular expressiosn are extreemly useful. We will illustrate its use for motif finding in DNA seqeucnes.

See Regular Expression HOWTO and the re documnetation for details.

```
In [10]: # 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 [11]:
```
# Suppose we want to replace motifs that start wtih 'ATA',
# followed by between 1 and 4 of any nucleotide, followed by 'CG'
# with a blank string of the same length

import re
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)])
```

```
ATGACCATGATTACGGATTCACTGGCCGTCGTTTTACAACGTCGTGACTGGGAAAACCCT
GGCGTTACCCAACTTAATCGCCTTGCAGCACATCCCCCTTTCGCCAGCTGGCGTA
 AAGAGGCCCGCACCGATCGCCCTTCCCAACAGTTGCGCAGCCTGAATGGCGAATGGCGC
TTTGCCTGGTTTCCGGCACCAGAAGCGGTGCCGGAAAGCTGGCTGGAGTGCGATCTTCCT
GAGGCCG         TCGTCCCCTCAAACTGGCAGATGCACGGTTACGATGCGCCCATC
TACACCAACGTGACCTATCCCATTACGGTCAATCCGCCGTTTGTTCCCACGGAGAATCCG
ACGGGTTGTTACTCGCTCACATTTAATGTTGATGAAAGCTGGCTACAGGAAGGCCAGACG
CGAATTATTTTTGATGGCGTTAACTCGGCGTTTCATCTGTGGTGCAACGGGCGCTGGGTC
GGTTACGGCCAGGACAGTCGTTTGCCGTCTGAATTTGACCTGAGCGCATTTTTACGCGCC
GGAGAAAACCGCCTCGCGGTGATGGTGCTGCGCTGGAGTGACGGCAGTTATCTGGAAGAT
CAGGATATGTGGCGGATGAGCGGCATTTTCCGTGACGTCTCGTTGCTGC        ACT
ACACAAATCAGCGATTTCCATGTTGCCACTCGCTTTAATGATGATTTCAGCCGCGCTGTA
CTGGAGGCTGAAGTTCAGATGTGCGGCGAGTTGCGTGACTACCTACGGGTAACAGTTTCT
TTATGGCAGGGTGAAACGCAGGTCGCCAGCGGCACCGCGCCTTTCGGCGGTGAAATTATC
GATGAGCGTGGTGGTTATGCCGATCGCGTCACACTACGTCTGAACGTCGAAAACCCGAAA
CTGTGGAGCGCCGAAATCCCGAATCTCTATCGTGCGGTGGTTGAACTGCACACCGCCGAC
GGCACGCTGATTGAAGCAGAAGCCTGCGATGTCGGTTTCCGCGAGGTGCGGATTGAAAAT
GGTCTGCTGCTGCTGAACGGCAAGCCGTTGCTGATTCGAGGCGTTAACCGTCACGAGCAT
CATCCTCTGCATGGTCAGGTCATGGATGAGCAGACGATGGTGCAGGATATCCTGCTGATG
AAGCAGAACAACTTTAACGCCGTGCGCTGTTCGCATTATCCGAACCATCCGCTGTGGTAC
ACGCTGTGCGACCGCTACGGCCTGTATGTGGTGGATGAAGCCAATATTGAAACCCACGGC
ATGGTGCCAATGAATCGTCTGACCGATGATCCGCGCTGGCTACCGGCGATGAGCGAACGC
GTAACGCGAATGGTGCAGCGCGATCGTAATCACCCGAGTGTGATCATCTGGTCGCTGGGG
AATGAATCAGGCCACGGCGCTAATCACGACGCGCTGTATCGCTGGATCAAATCTGTCGAT
CCTTCCCGCCCGGTGCAGTATGAAGGCGGCGGAGCCGACACCACGGCCACCGATATTATT
TGCCCGATGTACGCGCGCGTGGATGAAGACCAGCCCTTCCCGGCTGTGCCGAAATGGTCC
ATCAAAAAATGGCTTTCGCTACCTGGAGAGACGCGCCCGCTGATCCTTTGCGAATACGCC
CACGCGATGGGTAACAGTCTTGGCGGTTTCGCTAAATACTGGCAGGCGTTTCGTCAGTAT
CCCCGTTTACAGGGCGGCTTCGTCTGGGACTGGGTGGATCAGTCGCTGATTAAATATGAT
GAAAACGGCAACCCGTGGTCGGCTTACGGCGGTGATTTTGGCG         AACGATCGC
CAGTTCTGTATGAACGGTCTGGTCTTTGCCGACCGCACGCCGCATCCAGCGCTGACGGAA
GCAAAACACCAGCAGCAGTTTTTCCAGTTCCGTTTATCCGGGCAAACCATCGAAGTGACC
AGCGAATACCTGTTCCGTC             AGCTCCTGCACTGGATGGTGGCGCTGGAT
GGTAAGCCGCTGGCAAGCGGTGAAGTGCCTCTGGATGTCGCTCCACAAGGTAAACAGTTG
ATTGAACTGCCTGAACTACCGCAGCCGGAGAGCGCCGGGCAACTCTGGCTCACAGTACGC
GTAGTGCAACCGAACGCGACCGCATGGTCAGAAGCCGGGCACATCAGCGCCTGGCAGCAG
TGGCGTCTGGCGGAAAACCTCAGTGTGACGCTCCCCGCCGCGTCCCACGCCATCCCGCAT
CTGACCACCAGCGAAATGGATTTTTGCATCGAGCTGGGTA       TTGGCAATTTAAC
CGCCAGTCAGGCTTTCTTTCACAGATGTGGATTGGCGATAAAAAACAACTGCTGACGCCG
CTGCGCGATCAGTTCACCCGTGCACCGCTGG      ACATTGGCGTAAGTGAAGCGACC
CGCATTGACCCTAACGCCTGGGTCGAACGCTGGAAGGCGGCGGGCCATTACCAGGCCGAA
GCAGCGTTGTTGCAGTGCACGGCAGATACACTTGCTGATGCGGTGCTGATTACGACCGCT
CACGCGTGGCAGCATCAGGGGAAAACCTTATTTATCAGCCGGAAAACCTACCGGATTGAT
GGTAGTGGTCAAATGGCGATTACCGTTGATGTTGAAGTGGCGAGCG        CATCCG
GCGCGGATTGGCCTGAACTGCCAGCTGGCGCAGGTAGCAGAGCGGGTAAACTGGCTCGGA
TTAGGGCCGCAAGAAAACTATCCCGACCGCCTTACTGCCGCCTGTTTTGACCGCTGGGAT
CTGCCATTGTCAGACATGT        TACGTCTTCCCGAGCGAAAACGGTCTGCGCTGC
```

```
GGGACGCGCGAATTGAATTATGGCCCACACCAGTGGCGCGGCGACTTCCAGTTCAACATC
AGCCGCTACAGTCAACAGCAACTGATGGAAACCAGCCATCGCCATCTGCTGCACGCGGAA
GAAGGCACATGGCTGA          GTTTCCATATGGGGATTGGTGGCGACGACTCCTGG
AGCCCGTCAGTATCGGCGGAATTCCAGCTGAGCGCCGGTCGCTACCATTACCAGTTGGTC
```

## 2.2 The NLTK toolkit

If you will be doing statitical natural language processing or significant amounts of machhine learning on natrual 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 [12]: # YOUR CODE HERE

         def complement(dna):
             """Return compelementary strand given DNA sequence."""
             import string
             table = string.maketrans('actgACTG', 'tgacTGAC')
             return dna.translate(table)

         print complement('ATCGTTA')
```

```
TAGCAAT
```

**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 [13]: # YOUR CODE HERE

         phone_pat = re.compile(r'\(\d{3}\)-\d{7}')

         for s in ['(123)-9876543', '234-1234567', '123)-666666)']:
             m = phone_pat.match(s)
             if m:
                 print 'Mathced', s
             else:
                 print 'Not matched', s
```

```
Mathced (123)-9876543
Not matched 234-1234567
Not matched 123)-666666)
```

Note: This is just for practice - actual email validators should not be using regular expressions because the rules for a valid eamil are insanely complex, and should probably be checked with a *parser*.

```
In [14]: email_pat = re.compile(r'[\w]+[\.[\w]+]?@([\w]+\.)+[\w]+')

         for s in ['johm@', 'john.doe@duke.edu', 'steve@gmail.com', 'steve@gmail']:
```

```
            m = email_pat.match(s)
            if m:
                print 'Mathced', s
            else:
                print 'Not matched', s
```

```
Not matched johm@
Mathced john.doe@duke.edu
Mathced steve@gmail.com
Not matched steve@gmail
```

```
In [15]: motif_pat = re.compile(r'A.?C.?T.?G')

         for s in ['GATTACA', 'ACTG', 'AACCTTGG', 'AAACCCTTTGGG']:
             m = motif_pat.match(s)
             if m:
                 print 'Mathced', s
             else:
                 print 'Not matched', s
```

```
Not matched GATTACA
Mathced ACTG
Mathced AACCTTGG
Not matched AAACCCTTTGGG
```

**3**. Download 'Pride and Prejudice' by Jane Austem 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 [16]: # YOUR CODE HERE

         if not os.path.exists('pride_and_prejudice.txt'):
             ! curl 'http://www.gutenberg.org/cache/epub/1342/pg1342.txt' > 'pride_and_prejudice.txt'
```

```
In [17]: import string

         with open('pride_and_prejudice.txt') as f:
             s = f.read()
             s = s.lower().translate(None, string.punctuation)

             words = s.split()
             size = 10
             windows = list(partition(size, words))
             print "'daughter' and 'married' appera %d times in the same 10-word window" % \
                 sum('daughter' in window and 'married' in window for window in windows)
             print "The word 'married' appears %d times" % s.count('married')
```

```
'daughter' and 'married' appera 5 times in the same 10-word window
The word 'married' appears 61 times
```

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

- First extract all defined words (109561 words) - oops I cannot replicate this number
- Count the number of *defined* English words containing 3 or more vowels (aeiou)

- Find all longest palindrome (a palindrome is a word that is spelt the same forwards as backwards - e.g. 'deified')

In [18]:
```
# YOUR CODE HERE

# If you look at the plain text file,
# it is quite hard to figure out how to extract a defined word.
# We have more luck wiht the HTNL file.

if not os.path.exists('websters.html'):
    ! curl 'www.gutenberg.org/cache/epub/29765/pg29765.html' > 'websters.html'
```

In [19]:
```
! head -n 400 websters.html | tail -n 30
```

In [20]:
```
# Notice that in the HTML, word definitions have the structure <p id="xxxxxx">WORD</br> or <p

text = open('websters.html').read()
word = re.compile(r'<p id="id\d+">([A-Z]+)[<br/>|\r\n+]')

words = word.findall(text)
count = 0
for word in words:
    if word.count('A') + word.count('E') + word.count('I') + word.count('O') + word.count('U')
        count += 1

print "Number of words is %d" % len(words)
print "Number of words with 3 or  more vowels is %d" % count

palindromes = [word for word in words if word == word[::-1]]
lengths = map(len, palindromes)
max_len = max(lengths)
print "Longest palindromes are", [p for p in palindromes if len(p) == max_len]
```

```
Number of words is 103020
Number of words with 3 or  more vowels is 69210
Longest palindromes are ['MALAYALAM']
```

In [20]: