*Natural Language Procesing with Python Chapter 3*

Accessing Text

|  |  |
| --- | --- |
| Problem | Solution |
| Open a URL and read into a string | from urllib import urlopen  url = 'http://www.gutenberg.org/files/2554/2554.txt'  raw = urlopen(url).read() |
| Tokenize the string (now a list of words) **{Step 1}** | tokens = nltk.word\_tokenize(raw)  tokens[:10] >>> ['The', 'Project', 'Gutenberg', 'EBook', |
| Create an NLTK text object **{Step 2}** | text = nltk.Text(tokens)  text.collocations() |
| Find and reverse find (Python) | raw.find(‘'Petrovna') >>> 73649 (index number of first occurance)  raw.rfind( ‘'Petrovna') >>> 1065444 (index number of last occurance) |
| Cleaning HTML | Html = urlopen(url).read()  Raw = nltk.word\_html(html) ## beautiful soup maybe a better idea |
| Reading the contents of entire file | f = open(‘document.txt’)  raw = f.read() |
| Iterate through a file | f = open(‘document.txt’)  for line in f:  print line |
| Libraries for reading word and pdf documents | pypdf and pywin32. |
| Capturing user input | s = raw\_input("enter some text sucker: ")  enter some text sucker: hello my name is mark  print s >>> hello my name is mark |
| Create a new line | end = '\n'; mark = 'mark'; hatcher = 'hatcher'; print mark + end + hatcher;  mark  hatcher |
| How to include an apostrophe | circus = “Monty Python’s “ ## method 1 double quotes  circus = ‘Monty Python\’s‘ ## method 2 escape inside single quotes |
| Strings on two lines #1 | testString = "This is line 1"\  "and this is line 2"  print testString ## This is line 1and this is line 2 |
| Strings on two lines #2 | testString = ("This is line 1"  "and this is two")  #prints same as above. |
| Strings on two lines #3 [Eoln solution] | testString = """this is the first  and this the second"""  print testString  this is the first  and this the second |
| printing multiple items (commas) | testString = "hi there"  print testString, " dickhead"  hi there dickhead |
| Iterate through a string | A string is a sequence, so you can iterate through it. |
| Basic rules of slicing | Start includes start position. Finish is up to but not including the finish (finish = finish -1). Negative indexes are 1 based (i.e -1 is the last character) And postive indexes are zero based (0 is the first character). |
| Using in for a substring | hatcher = "hatcher"  'cher' in hatcher >> true |
| Dictionaries | most frequent items are at the start. |
| Getting help on a string | help(str) |
| Difference between strings and lists | Strings are not mutable. |

Unicode

|  |  |
| --- | --- |
| Problem | Solution |
| Octal, Hex | print('m') #prints m  print('\155') #prints m Octal  print('\x6d') #prints m Hexadecimal |
| Unicode (code point) | ***Each character is assigned a code point which is a number.*** Each unicode number is a 4 digit hex number. Some encoding only support a single byte per code point (ASCII and Latin 2) and can only represent a subset of Unicode. UTF-8 uses multiple bytes and represent all unicode characters. |
| Encoding | ***The process of translating a string of characters into its raw bytes form accoding to the desired encoding*** |
| Glyphs | Fonts are mappings from binary structures to ones that can be displayed on the screen or paper. |
| 1 Byte and 2 Hex pieces | 1 byte (8 bites) is 2^8 = 256 combinations. 2 hex digits are equal to one byte. (15 \* 16) + (1 x 15) |
| Assign a character using unicode | nacute = u'\u0144' |
| Assign a character using hex | something = '\xc5\x84' |
| Hex to unicode encodings | They do not map perfectly. The hex above is 329 and the unicode is 324 (5 digit offset) |
| Encode to raw bytes | nacute = u'\u0144'  nacuteUTF = nacute.encode('utf8') |
| Ascii / Latin-2 / Unicode | Ascii is from 0 to 127. Latin-2 is from 128 to 256 and Unicode is above above these. |
| unicode\_escape | This is a dummy encoding that converts all above 256 to \uXXXX and between 127 to 256 to \xXX |
| unicode\_escape (example) | import codecs  path = nltk.data.find('corpora/unicode\_samples/polish-lat2.txt')  f = codecs.open(path.path, encoding='latin2')  for line in f:  line = line.strip()  print line.encode('unicode\_escape') |
| Print a single polish character | import unicodedata  path = nltk.data.find('corpora/unicode\_samples/polish-lat2.txt')  lines = codecs.open(path.path, encoding='latin2').readlines()  line = lines[2]  print(line)  Niemców pod koniec II wojny światowej na Dolny Śląsk, zostały  print(line[5]) >>> ó  print(line[5].encode('unicode\_escape')) >> \xf3  char = (line[5]) |
| Print unicode details (from above) | print '%r U+%04x %s' % (char.encode('utf8'), ord(char), unicodedata.name(char))  '\xc3\xb3' U+00f3 LATIN SMALL LETTER O WITH ACUTE |
| Explanation of above (%r) | %r This is the raw string. If this is changed to %s then ó is printed |
| Explanation of above (U+%04x) | Ord(char) is 243. This is F3 it is formatted to have two leading zeros. |
| Setting inputing and editing specific encoding in a python file | You need to include as the first or second line:  ‘# -\*- coding utf-8 -\*- |

Regular Expressions

|  |  |
| --- | --- |
| Problem | Solution |
| Create a wordlist | wordlist = [w for w in nltk.corpus.words.words('en') if w.islower()] |
| Get words that end in “ed” using **re.search** | import re  edWords = [w for w in wordlist if re.search(r'ed$',w)] |
| Count the number of patterns (for email or e-mail) | sum([1 for w in wordlist if re.search(r'e-?mail',w)]) |
| Find all (non-overlapping pieces) | word = 'supercalidfragislisticexpialidocious'  re.findall(r'[aeiou]',word)  ['u', 'e', 'a', 'i', 'a', 'i', 'i', 'i',……… |
| Find vowel frequencies | wsj = sorted(set(nltk.corpus.treebank.words()))  fd = nltk.FreqDist(vs for word in wsj for vs in re.findall(r'[aeiou]{2,}',word))  fd.items()[:10] >>> [('io', 549), ('ea', 476), ('ie', 331), ('ou', 329),….. |
| Deleting redundant parts of words. Matches: (1) initial vowel, final vowel (2), middle consanents (3). The RE is processed left to right. If any of the three parts and later parts are ignored. re.Findall() extracts the matching pieces and ‘ ‘,join to join them. | regexp = r'^[AEIOUaeiou]+|[AEIOUaeiou]+$|[^AEIOUaeiou]'  def compress(word):  pieces = re.findall(regexp, word)  return ''.join(pieces)  compress(‘mark’) >>> ‘mrk’ just misses the middle vowel  compress(‘amark’) >>> ‘amrk  compress(‘marko’) >>> ‘mrko’ |
| Parenthesis have a second function to select substrings to be extracted. | #the following matches the entire word. But the parentheses match the suffix. The parenthesis have two functions.  re.findall(r'^.\*(ing|ly|ed|ious|ies|ive|es|s|ment)$', 'processing')  ['ing']  re.findall(r'^.\*(?:ing|ly|ed|ious|ies|ive|es|s|ment)$', 'processing') ## put in **?:**  ['processing'] |
| If we want to split the words up into two, then we just add two pairs of parentheses. | re.findall(r'^(.\*)(ing|ly|ed|ious|ies|ive|es|s|ment)$', 'processing')  [('process', 'ing')] |
| Searching tokenized text Matching *“A XXXXX man”* ***Notes****: <.\*> matches any single token.* | from nltk.corpus import gutenberg  moby = nltk.Text(gutenberg.words('melville-moby\_dick.txt'))  moby.findall(r'<a>(<.\*>)<man>')  monied; nervous; dangerous; white; white;……. |
| Find 3 or more words starting with “l” | r”<l.\*>{3,}” |

Normalising Text

|  |  |
| --- | --- |
| Problem | Solution |
| What is normalising text | Converting it to lower case is the first step. Then removing affixes (stemming) then there is lemmarization. |
| Using a Porter stemmer | # to use a lancester stemmer use nltk.LancasterStemmer()  import nltk  porter = nltk.PorterStemmer()  tokens = nltk.word\_tokenize(raw) #raw previously defined as string  stemList = [porter.stem(word) for word in tokens]  stemList[:15]  ['DENNI', ':', 'Listen', ',', 'strang', 'women', 'lie', 'in', 'pond', 'distribut'  Was previously: ""DENNIS: Listen, strange women lying in ponds distributing |
| Indexing example (not documented) | ***Look on page 107*** |
| Lemmatization (removes affixes only if in the wordnet dictionary) Not as dramatic as stemming | lemmat = nltk.WordNetLemmatizer()  lemmas = [lemmat.lemmatize(word) for word in tokens] |
| Non standard words | Identifiers words not in a dictionary and then puts in them in special vocabulary. |

Tokenizing Text *(with Regular Expressions)*

|  |  |
| --- | --- |
| Tokenizing (using white space) | re.split(r' ', raw) # raw is previously defined.  As a newline can split a word. This results in '**very\nwell'** being a token although it shold be “very” “well” |
| Tokenizing (using spaces, tabs and new lines [at least one of these] | re.split(r'[ \t\n]+', raw) #this can be written using \s which means match any whitespace although this does not cope with new lines.  re.split('r\s+', raw) |
| Python class for word charcters | [a-zA-Z0-9\_] == \w. The complement to this is \W |
| Complement of split (findall | re.split(r’\W+’,raw) #splits according to non-characterrs  re.findall(‘r’w+’, raw) #finds (opposite of split) according to characters |
| Puntuation | There are other problems such as hyphens and punctuation. Quotes are another problem. |
| Comprehensive Example | text = 'That U.S.A. poster-print costs $12.40...'  pattern = r'''(?x) # set flag to allow verbose regexps  ([A-Z]\.)+ # abbreviations, e.g. U.S.A.  | \w+(-\w+)\* # words with optional internal hyphens  | \$?\d+(\.\d+)?%? # currency and percentages, e.g. $12.40, 82%  | \.\.\. # ellipsis  | [][.,;"'?():-\_`] # these are separate tokens  '''  nltk.regexp\_tokenize(text,pattern)  ['That', 'U.S.A.', 'poster-print', 'costs', '$12.40', '...']  Note: The special verbose flat (?x) tells Python to strip out embedded white space and comments |
| Tokenization issues | No single solution works perfectly. Best to have a manual solution to compare it with. |

Segmentation

|  |  |
| --- | --- |
| Problem | Solution |
| Sentence segmentation | Made difficult because of the periods in acronyms. Following is an example of an NLTK solution:  sent\_tokenizer = nltk.data.load('tokenizers/punkt/english.pickle')  text = nltk.corpus.gutenberg.raw('chesterton-thursday.txt')  sentences = sent\_tokenizer.tokenizer(text)  ***See section 6.2 for another approach to sentence segmentation.*** |
| What is normalising text | Converting it to lower case is the first step. Then removing affixes (stemming) then there is lemmarization. |

String Printing (see format specfication mini language)

|  |  |
| --- | --- |
| Problem | Solution |
| Format specifiers | |  |  | | --- | --- | | Specifier | Meaning | | b | Binary. Outputs the number in base 2 | | c | Character. Converts the integer to the corresponding unicode character | | d | Decimal integer. Outputs the number in base 10 | | o | Octal format. | | x | Hex format | | s | String | | e | Exponent notation. | | s | string | | f | Fixed point number | |
| Print a number to four decimal places in 25 spaces, left justified followed by a string | number = 314159.6343443  print '%-25.4f %s' % (number, 'markTheKoala')  314159.6343 markTheKoala |
| Print each item on the same line (add a comma) | for word in sentWords:  print(word), |
| Same as the above, right justified number. String in 20 spaces | print '%25.4f %20s' % (number, 'markTheKoala')  314159.6343 markTheKoala |
| Print a number with leading zeros | number = 1000  print '%010d' % (number) >>> 0000001000 |
| Printing using conversion specifiers without print() | "%s wants a %s for %s" %("mark", "sandwich", "lunch")  'mark wants a sandwich for lunch' |
| Using tempates | template = "Mark wants a %s right now"  menu = ["headjob", "million dollars", "taxi"]  for word in menu:  print template % word  Mark wants a headjob right now >>> and on and on |
| Setting width with a variable | width = 30  "Rande likes %\*s" % (width, "fat chicks")  'Rande likes fat chicks' |
| Percentage signs (double percent) | "this is %f%% off" % 50  'this is 50.000000% off' |
| Using module textwrap (this wraps a line ) not clear though (page 120) | from textwrap import fill  text = "After all is said and done, Hachiko and Mark the Koala climbs up trees and then goes to sleep for a while"  print(fill(text))  After all is said and done, Hachiko and Mark the Koala climbs up trees  and then goes to sleep for a while |
|  |  |
|  |  |