python
powered

DM536
Introduction to Programming

Peter Schneider-Kamp

petersk@imada.sdu.dk

http://imada.sdu.dk/~petersk/DM536/

SELECTING DATA STRUCTURES

Reading and Cleaning Words

1. read file given as argument
2. break lines into words
3. strip whitespace & punctuation
4. convert to lower-case letters

- import module sys for command line arguments sys.argv
- Example: import sys; print sys.argv

- import module string for punctuation
- Example: import string; print string.punctuation

- use translate(None, deletechars) to remove punctuation
- Example: "Hello World!".translate(None, "ol")

Word Frequency in E-Books

1. use program on Project Gutenberg e-book
2. skip over beginning & end of ebook (marked "***")
3. count total number of words
4. count number of times each word is used
5. print 20 most frequently used words

- use Boolean flag to indicate when to start
- use list to gather all words (and count total number)
- use dictionary to count number of times each word is used
- use tuple comparison to sort words

Optional Parameters

- have seen functions that take variable length argument list

- also possible to make some parameters optional
- in this case, default value has to be supplied by programmer
- Example:

```
def print_most_common(hist, num = 10):
    t = most_common(hist)
    print "The most common", num, "words are:"
    for n, word in t[:num]:
        print word, "\t", n
print_most_common(freq, 20)
```

Dictionary Subtraction

1. find all words that do NOT occur in other word list

- to this end, subtract dictionaries from each other
- **Idea:** new dictionary containing with keys only in first dict
- Implementation:

```
def subtract(d1, d2):
    d = {}
    for key in d1:
        if key not in d2:
            d[key] = None
    return d
```

Random Number Generation

- to work with random numbers, import module random
- Example: import random

- function random() returns random float from 0.0 to < 1.0
- Example: for i in range(10): print random.random()

- function randint(a, b) returns random integer in range(a,b+1)
- Example: for i in range(10): print random.randint(1,10)

- function choice(seq) returns random element of a sequence
- Example: random.choice("Slartibartfast")

  random.choice([23, 42, -3.0])

Random Words

1. choose random word from histogram according to frequency

- how to ensure random choice w.r.t. frequency?
- **Idea 1:** create list with n copies of word with frequency n
- Implementation:

```
def random_word(h):
    t = []
    for word, n in h.items():
        t.extend([word] * n)
    return random.choice(t)
```

- works, but very inefficient!

Random Words

- **Idea 2:** use list with cumulative sum of frequencies
- Implementation:

```
def random_word(h):
    words = h.keys();  sum = 0;  cum = []
    for word in words:  sum += h[word];  cum.append(sum)
    num = random.randint(1, cum[-1]); low = 0; high = len(cum)-1
    while low < high:
        mid = (low+high) / 2
        if num <= cum[mid]:  high = mid
        elif num > cum[mid]:  low = mid+1
    return words[low]
```

Markov Analysis

1. generate more meaningful random texts

- word order in texts is not random
- markov analysis maps a finite number of words (prefix) to all possible following words (suffix)

- how to represent the prefixes?

- how to represent the collection of possible suffixes?

- how to represent the mapping from prefixes to suffixes?

Data Structures

- for mapping, we clearly use a dictionary

- for prefixes, we need to be able to "shift" them (list?)
- we also need to use them as dictionary keys
- thus, we use tuples to present prefixes (+ slicing and "*")

- for suffixes, we need to add elements (list? dictionary?)
- we also need to efficiently generate random word (list?)
- tradeoff space vs time
  - dictionary uses less space and easy to add
  - list uses less time for generating a word
  - can change representation before generation

Debugging Hard Bugs

- bugs can be hard to find

- four popular strategies
  1. reading: re-read your code, check that it is right!
  2. running: make changes, experiment with outcome
  3. ruminating: take time to think it over (and over)
  4. retreating: revert to a known-to-be-good version

- often combination of these strategies needed
- always good to view debugging as scientific experiment

FILE HANDLING

Persistence

- persistent = keeping (some) data stored during runs
- transient = beginning from input data each time over

- most programs so far have been transient

- examples of persistent programs:
  - operating systems
  - web servers
  - most app(lication)s on recent Android, iOS, and Mac OS X

- text files are easiest way to save some program state
- alternatively, program states can be saved in databases