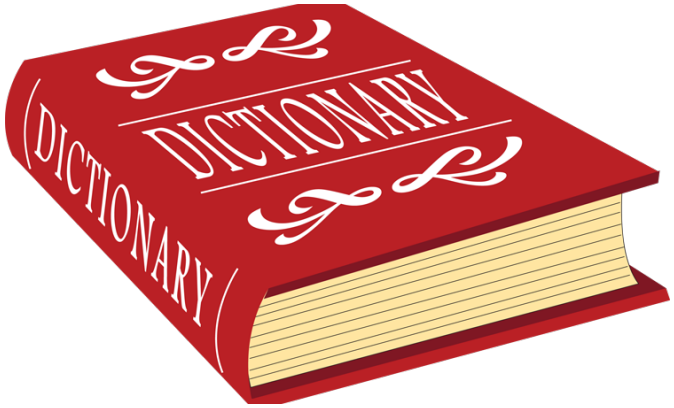
Today: Dictionaries and Files





Lists are *sequential* containers:

```
L = [42, 5, 6, 47]
```

0 1 2 3

elements are looked up by their **location**, or **index**, starting from 0

Dictionaries are *arbitrary* containers:

value value

```
d = {3: 47, 1: 5}
```

3 1

key key

elements (or *values*) are looked up by a **key** starting anywhere you want! **Keys** don't have to be ints!

Dictionaries are *arbitrary* containers:

value value

```
d = {'tiger': 1998, 'rabbit': 1999}
```

'tiger' 'rabbit'

key key

Elements (or *values*) are looked up by a **key** starting anywhere you want! **Keys** don't have to be ints!

What's up with d's data here?

Now I see the **key** to dictionaries' **value**...



Dictionaries

A ***dictionary*** is a set of **key/value** pairs

```
>>> d = {}
```
creates an empty dictionary, **d**

```
>>> d[1993] = 'rooster'
```
key value

```
>>> d[1996] = 'rat'
```
another key another value

Now I see the **key** to dictionaries' **value**...



馬 Horse	Jan 27, **1990** – Feb 14, 1991
羊 Goat	Feb 15, **1991** – Feb 3, 1992
猴 Monkey	Feb 4, **1992** – Jan 22, 1993
鷄 Rooster	Jan 23, **1993** – Feb 9, 1994
狗 Dog	Feb 10, **1994** – Jan 30 1995
猪 Pig	Jan 31, **1995** – Feb 18, 1996
鼠 Rat	Feb 19, **1996** – Feb 6, 1997
牛 Ox	Feb 7, **1997** – Jan 27, 1998
虎 Tiger	Jan 28, **1998** – Feb 15, 1999

Dictionaries

A ***dictionary*** is a set of **key/value** pairs

```
>>> print(d)
{1996:'rat', 1993:'rooster'}

>>> d[1996]
'rat'

>>> 1996 in d
True
>>> 1997 in d
False
```

in is *key* !

Now I see the **key** to dictionaries' **value**...



馬 Horse	Jan 27, **1990** – Feb 14, 1991
羊 Goat	Feb 15, **1991** – Feb 3, 1992
猴 Monkey	Feb 4, **1992** – Jan 22, 1993
鷄 Rooster	Jan 23, **1993** – Feb 9, 1994
狗 Dog	Feb 10, **1994** – Jan 30 1995
猪 Pig	Jan 31, **1995** – Feb 18, 1996
鼠 Rat	Feb 19, **1996** – Feb 6, 1997
牛 Ox	Feb 7, **1997** – Jan 27, 1998
虎 Tiger	Jan 28, **1998** – Feb 15, 1999

Dictionaries *can have different keys*

```
>>> d = {}
```
creates an empty dictionary, **d**

keys can be anything handled by *value*, not *reference!*

Strings CAN be keys.
Lists CANNOT be.

```
>>> d['rat'] = 1996
```
key value

`d.keys()`

`d.values()`

`d.items()`

```
>>> d['rooster'] = 1993
```
another key

another value

馬 Horse	Jan 27, **1990** – Feb 14, 1991
羊 Goat	Feb 15, **1991** – Feb 3, 1992
猴 Monkey	Feb 4, **1992** – Jan 22, 1993
鷄 Rooster	Jan 23, **1993** – Feb 9, 1994
狗 Dog	Feb 10, **1994** – Jan 30 1995
猪 Pig	Jan 31, **1995** – Feb 18, 1996
鼠 Rat	Feb 19, **1996** – Feb 6, 1997
牛 Ox	Feb 7, **1997** – Jan 27, 1998
虎 Tiger	Jan 28, **1998** – Feb 15, 1999

Dictionaries *can have different keys*

```
>>> d = {}
```
creates an empty dictionary, **d**

keys can be anything handled by *value*, not *reference!*

Strings CAN be keys.
Lists CANNOT be.

```
>>> d['rat'] = 1996
```

key value

d.keys()

d.values()

d.items()

```
>>> d['rooster'] = 1993
```

an ...other

value

d[42] = 1492
That's OK too!!!

馬 Horse	Jan 27, **1990** – Feb 14, 1991
羊 Goat	Feb 15, **1991** – Feb 3, 1992
猴 Monkey	Feb 4, **1992** – Jan 22, 1993
鷄 Rooster	Jan 23, **1993** – Feb 9, 1994
狗 Dog	Feb 10, **1994** – Jan 30 1995
猪 Pig	Jan 31, **1995** – Feb 18, 1996
鼠 Rat	Feb 19, **1996** – Feb 6, 1997
牛 Ox	Feb 7, **1997** – Jan 27, 1998
虎 Tiger	Jan 28, **1998** – Feb 15, 1999

Dictionaries are *in* !

```
z = {'rat': [1996,1984,1972,...],
     'ox': [1997,1985,1973,...],
     'tiger': [1998,...], ... }
```

Is 'dragon' a key in **z**?

```
if 'dragon' in z
```

Is 1969 in **z['dragon']**?

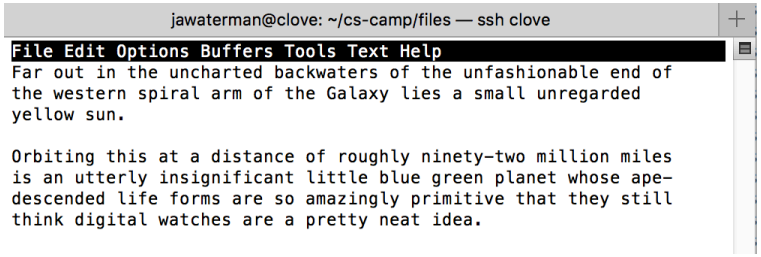
```
if  1969 in z['dragon']
```

In Python reading files is no problem…

Files

```
>>> f = open('a.txt')
```

opens the file and calls it **f**




```
>>> text = f.read()
```

reads the whole file into the string **text**

```
>>> f.close()
```

closes the file (optional but good practice)

```
>>> text
'Far out in the uncharted backwaters of …
```

```
>>> LoW = text.split()
[ 'Far', 'out', 'in', ... ]
```

text.split() returns a list of each "word"

```
def word_count(filename):
    """Counts # of words in file"""

    f = open(filename)
    text = f.read()
    f.close()

    LoW = text.split()
    print("There are", len(LoW), "words")
```

file handling

What if we wanted the number of ***different*** words in the file?

This would be the author's ***vocabulary size***, instead of the total word count.

Counting ***distinct*** words with a dictionary…



WOULD YOU LIKE THEM IN A HOUSE?

WOULD YOU LIKE THEN WITH A MOUSE?

I DO NOT LIKE THEM IN A HOUSE.

I DO NOT LIKE THEM WITH A MOUSE.
I DO NOT LIKE THEM HERE OR THERE.
I DO NOT LIKE THEM ANYWHERE.
I DO NOT LIKE GREEN EGGS AND HAM.
I DO NOT LIKE THEM, SAM-I-AM.

Which book? Which author?

Counting ***distinct*** words with a dictionary…

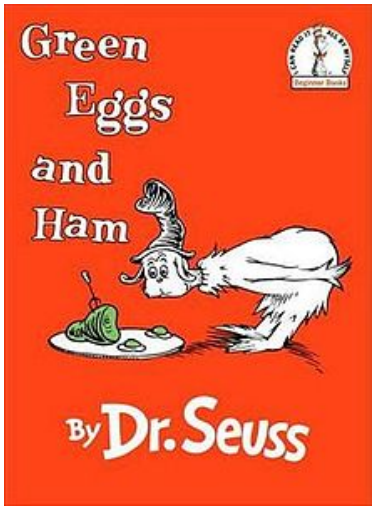


WOULD YOU LIKE THEM IN A HOUSE?

WOULD YOU LIKE THEN WITH A MOUSE?

I DO NOT LIKE THEM IN A HOUSE.

I DO NOT LIKE THEM WITH A MOUSE.
I DO NOT LIKE THEM HERE OR THERE.
I DO NOT LIKE THEM ANYWHERE.
I DO NOT LIKE GREEN EGGS AND HAM.
I DO NOT LIKE THEM, SAM-I-AM.



Vocabulary [edit]

Green Eggs and Ham is one of Seuss's "Beginner Books", written in a very simple vocabulary for beginning readers. The vocabulary of the text consists of just 50 different words[3] and was the result of a bet between Seuss and Bennett Cerf (Dr. Seuss's publisher)[3][4][5] that Seuss (after completing *The Cat in the Hat* using 236 words)[6] could not complete an entire book without exceeding that limit. The 50 words are: a, am, and, anywhere, are, be, boat, box, car, could, dark, do, eat, eggs, fox, goat, good, green, ham, here, house, I, if, in, let, like, may, me, mouse, not, on, or, rain, Sam, say, see, so, thank, that, the, them, there, they, train, tree, try, will, with, would, you.[3]

Counting ***distinct*** words with a dictionary...



wd wd wd wd wd ...
WOULD YOU LIKE THEM IN A HOUSE?
wd wd wd wd wd ...
WOULD YOU LIKE THEN WITH A MOUSE?
wd wd wd wd wd ...
I DO NOT LIKE THEM IN A HOUSE.
wd wd wd wd wd ...
I DO NOT LIKE THEM WITH A MOUSE.
I DO NOT LIKE THEM HERE OR THERE.

LoW

d keys values

```
for wd in LoW:
    if wd not in d:
        d[wd] = 1
    else:
        d[wd] += 1
```

```
def vocab_count(filename):
    """Vocabulary-counting program"""
    f = open( filename )
    text = f.read()
    f.close()

    LoW = text.split()
    print "There are", len(LoW), "words."

    d = {}

    for w in LoW:

        if w not in d:
            d[w] = 1

        else:
            d[w] += 1

    print "There are", len(d), "distinct words.\n"

    return d        # return d for later use by other code...
```

file handling

word counting

Tracking the number of occurrences of each word with a dictionary, d.

most/least common?

Vocabulary, anyone?

Shakespeare used ***31,534 different words***—and a grand total of 884,647 words —counting repetitions—(across all of his works)

http://www-math.cudenver.edu/~wbriggs/qr/shakespeare.html

Shakespearean coinages

gust
besmirch
unreal
superscript
watchdog
swagger

affined
rooky
attasked
out-villained

successful

unsuccessful

*There's one word from a **contemporary** author in the Oxford English Dictionary…*

Who?

What word?

http://www.pathguy.com/shakeswo.htm
http://www.shakespeare-online.com/biography/wordsinvented.html

Vocabulary, anyone?

Shakespeare used ***31,534 different words*** -- and a grand total of 884,647 words -- counting repetitions (across all of his works) http://www-math.cudenver.edu/~wbriggs/qr/shakespeare.html

Shakespearean coinages

gust	
besmirch	affined
unreal	rooky
superscript	attasked
watchdog	out-villained
swagger	
successful	unsuccessful

'Muggle' goes into Oxford English Dictionary

JK Rowling's word for non-wizards - "muggle" - has made it into the new edition of the Oxford English Dictionary (OED).

The draft definition according to the dictionary's website says:

- ***Muggle:*** *invented by JK (Joanne Kathleen) Rowling (b. 1965), British author of children's fantasy fiction (see quot. 1997).*

  In the fiction of JK Rowling: a person who possesses no magical powers. Hence in allusive and extended uses: a person who lacks a particular skill or skills, or who is regarded as inferior in some way.

J. K. Rowling

http://www.pathguy.com/shakeswo.htm
http://www.shakespeare-online.com/biography/wordsinvented.html

Algorithmic authoring?

'Cause somethin' like he left knee and a harp," said he had to the whole school? The shouting and then some strange and Mrs. "Well, I know Hagrid; they spotted handkerchief and get him get rid of course, had a gigantic beet with her," he knew what to all he's

All the sky with the sun in the sun in the church where you're gone Lucy in my eyes. There beneath the girl with an hourglass And then the banker never wears a lot to hold your hand. Can't buy me tight, tight Owww! Love is love I can't hide,

Who is the author? What is the work? What is going on?

This is but ourselves. No, faith, My uncle! O royal bed of confession Of your rue for leave to nature; to this time I should weep for thy life is rotten before he is. have sworn 't. Or my blood. I have closely sent for nine; and unprofitable,

The Senators and the date of a written declaration that Purpose, they shall consist of nine States, shall not, when he shall have such Vacancies. The President pro tempore, in the Desire of a Qualification to the Speaker of the Senate. Article 6. When vacancies by the office upon probable

Markov Models

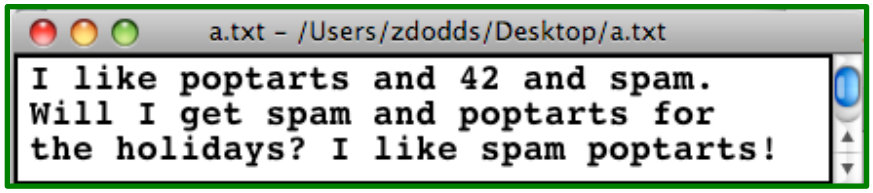
Techniques for modeling *any* sequence of natural data

speech, text, sensor data...

1st-order Markov Model
(defining property)

Each item depends ***only*** on the <u>one</u> immediately before it .

Our Markov Model



Original file

keys values

Markov Model

A dictionary!

```
{ '$':     ['I', 'Will', 'I'],
  'I':     ['like', 'get', 'like'],
  'like':
  'poptarts': ['and', 'for'],
  'and':   ['42', 'spam.', 'poptarts'],
  '42':    ['and'],
  'Will':  ['I'],
  'the':
  'spam':  ['and', 'poptarts!'],
  'get':   ['spam'],
  'for':   ['the']                    }
```

What are the keys?

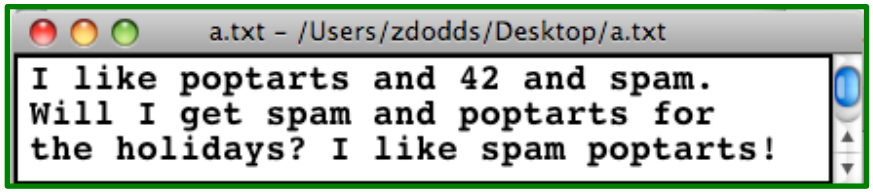
What are the values?

What are the missing values?

What is the '$'?

Why do some keys seem to be missing?

Our Markov Model



Original file

keys values

Markov Model

A dictionary!

What are the keys?

What are the values?

What are the missing values?

What is the '$'?

Why do some keys seem to be missing?

```
{ '$':      ['I', 'Will', 'I'],
  'I':      ['like', 'get', 'like'],
  'like':   ['poptarts', 'spam'],
  'poptarts': ['and', 'for'],
  'and':    ['42', 'spam.', 'poptarts'],
  '42':     ['and'],
  'Will':   ['I'],
  'the':    ['holidays?'],
  'spam':   ['and', 'poptarts!'],
  'get':    ['spam'],
  'for':    ['the']                    }
```

Model creation *in Python*

prev	*$*					
word	*I*	*like*	*spam*	*I*	*eat*	*poptarts*

```
d = {}
prev = '$'

for word in LoW:
    if prev not in d: d[prev] =  [word]
    else:             d[prev] += [word]

    prev = _______
```

Model creation *in Python*

d's final value

```
{'$': ['I'],
 'I': ['like', 'eat'],
 'like': ['spam'],
 'spam': ['I'],
 'eat': ['poptarts']}
```

prev	*$*					
word	*I*	*like*	*spam*	*I*	*eat*	*poptarts*

```
d = {}
prev = '$'

for word in LoW:
    if prev not in d: d[prev] =  [word]
    else:             d[prev] += [word]

    prev = word
```

Model creation:

1) start with the previous word, **`prev`** as **'$'**

2) for each next word, **`word`**, in the list of words, add it in ...

3) then change **`prev`** to **`word`** ...

(a) except if **word**[-1] was punctuation: change **`prev`** to...

Generating text:

1) start with **`prev`** as the **'$'** string

2) choose a **`word`** that follows **`prev`**, at random.

3) **`print(word, end = ' ')`**

4) **`prev`** gets set to either **`word`** or **'$'**

if **word**[-1] was punctuation: change **`prev`** to...

Markov Models are *generative!*

A key benefit of Markov Models is that they can ***generate*** feasible data!

Original file:

I like poptarts and 42 and spam.
Will I get spam and poptarts for
the holidays? I like spam poptarts!

Generated text:

I get spam poptarts! I like poptarts and 42 and spam. I like spam and 42 and 42 and 42 and spam. Will I like poptarts and 42 and poptarts and 42 and poptarts and 42 and 42 and poptarts and spam. I get spam and 42 and 42 and...



WMSCI 2005

Rooter: A Methodology for the Typical Unification of Access Points and Redundancy

Jeremy Stribling, Daniel Aguayo and Maxwell Krohn

http://pdos.csail.mit.edu/scigen/



Markov-generated submission accepted to WMSCI 2005

Rooter: A Methodology for the Typical Unification of Access Points and Redundancy

Jeremy Stribling, Daniel Aguayo and Maxwell Krohn

Abstract

Many physicists would agree that, had it not been for congestion control, the evaluation of web browsers might never have occurred. In fact, few hackers worldwide would disagree with the essential unification of voice-over-IP and public-private key pair. In order to solve this riddle, we confirm that SMPs can be made stochastic, cacheable, and interposable.

I. Introduction

Many scholars would agree that, had it not been for active networks, the simulation of Lamport clocks might never have occurred. The notion that end-users synchronize with the investigation of Markov models is rarely outdated. A theoretical grand challenge in theory is the important unification of virtual machines and real-time theory. To what extent can web browsers be constructed to achieve this purpose?

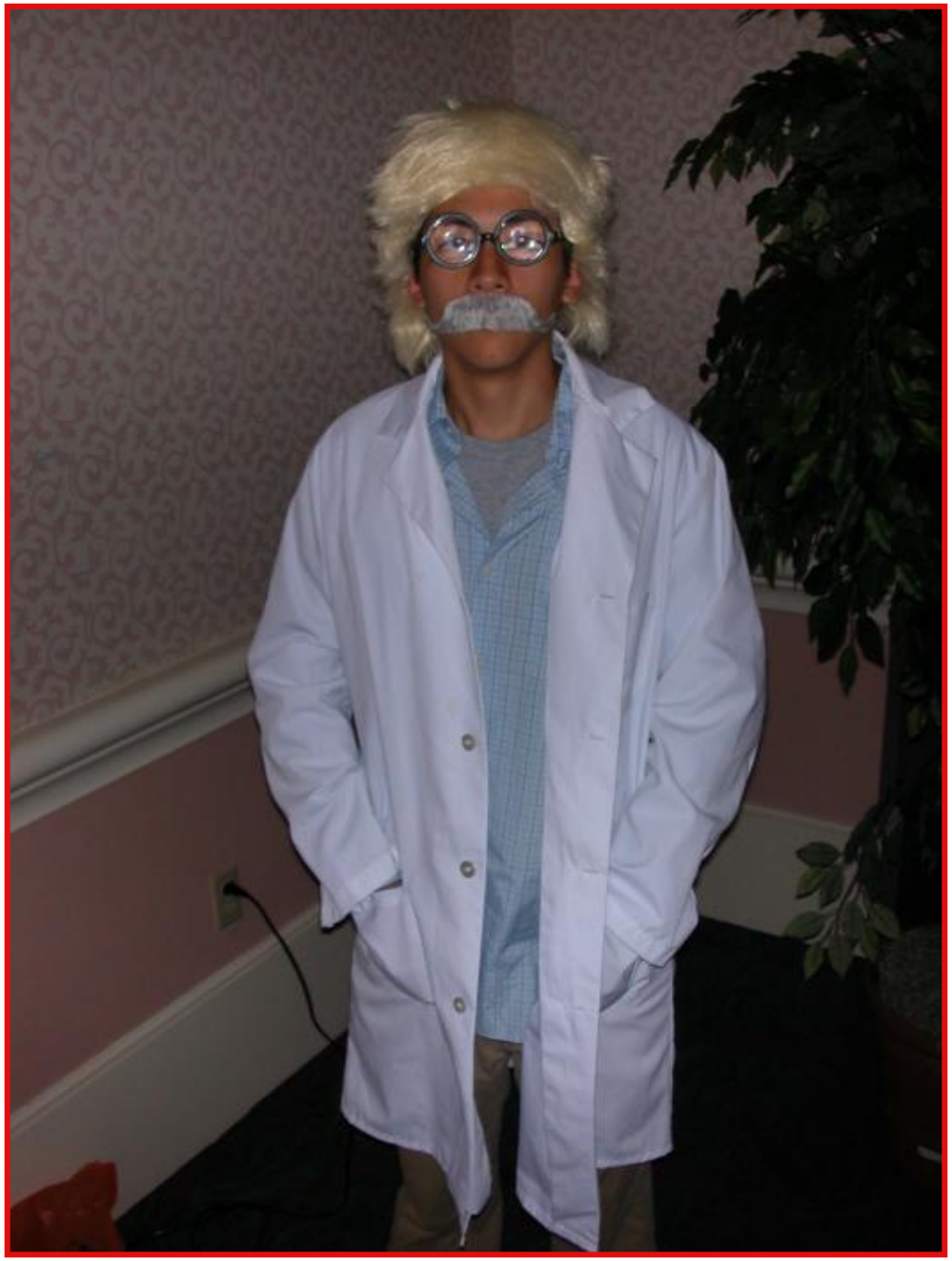
Certainly, the usual methods for the emulation of Smalltalk that paved the way for the investigation of rasterization do not apply in this area. In the opinions of many, despite the fact that conventional wisdom states that this grand challenge is continuously answered by the study of access points, we

The rest of this paper is organized as follows. For starters, we motivate the need for fiber-optic cables. We place our work in context with the prior work in this area. To address this obstacle, we disprove that even though the much-tauted autonomous algorithm for the construction of digital-to-analog converters by Jones [10] is NP-complete, object-oriented languages can be made signed, decentralized, and signed. Along these same lines, to accomplish this mission, we concentrate our efforts on showing that the famous ubiquitous algorithm for the exploration of robots by Sato et al. runs in $\Omega((n + \log n))$ time [22]. In the end, we conclude.

II. Architecture

Our research is principled. Consider the early methodology by Martin and Smith; our model is similar, but will actually overcome this grand challenge. Despite the fact that such a claim at first glance seems unexpected, it is buffetted by previous work in the field. Any significant development of secure theory will clearly require that the acclaimed real-time algorithm for the refinement of write-ahead logging by Edward Feigenbaum et al. [15] is impossible; our application is no different. This may or may not actually hold in reality

this was ***more*** than a first-order model…



and, *scene*...!

in costume

Thesis deadlines?

Other papers due?

Let Python write the rest of your papers for you…

… and you're still the author!