Preparation for Midterm 2

Thomas Schwarz, SJ
Marquette University

String Processing

- String processing patterns
  - Substituting in strings
    - Use a dictionary to contain substitutions

File Processing

- File processing patterns
  - Count the frequency for the beginning letter of a word
  - Create a counter object
  - Open file with “with”
  - Use a for loop to go through every line
  - Use split and a for loop to pass through every word
  - Canonicalize the word
  - Add word to counter
  - Return the tablef

File Processing

```
import collections

def count_start_word(filename):
    ctr = collections.Counter()
    with open(filename, encoding="latin-1") as infile:
        for line in infile:
            for word in line.split():
                word = word.lower().strip("[&(*,.;:'\"'`?!")
                if word:  ##we could have stripped
                          ##the word to nothing
                    ctr[word[0]] += 1
return ctr.most_common()
```

Test Yourself

- Alter the preceding code to find the frequencies of the last letters in word

Test Yourself

- You just need to change the counter to count word[-1].

```
def count_end_word(filename):
    ctr = collections.Counter()
    with open(filename, encoding="latin-1") as infile:
        for line in infile:
            for word in line.split():
                word = word.lower().strip("[&(*,.;:'\"'`?!")
                if word:  ##we could have stripped the word to
nothing
                    ctr[word[-1]] += 1
    return ctr.most_common()
```

Memoization

- A more complicated example:
  - How many stamps are needed to make various postages
  - Assume we have three types of stamps: ones, fours, and fives
  - To lick 27 cents, we can:
    - five 5c + 2 1c stamps: total of 7 stamps
    - four 5c, one 4c and 3 1c stamps: total of 8 stamps
    - three 5c and three 4c stamps: total of six stamps

Memoization

- Given an amount *n* we want to calculate the minimum number of stamps needed
- An inane method: Try out all possibilities

```
def inane(n):
    best_seen = n
    for ones in range(n+1):
        for fours in range(n+1):
            for fives in range(n+1):
                if ones*1+fours*4+fives*5 == n:
                    if ones+fours+fives < best_seen:
                        best_seen = ones+fours+fives
    return best_seen
```

Memoization

- Why is it inane?
  - It takes $n \times n \times n$ iterations to try out all possibilities

Memoization

- A better way?
  - Given an amount *n*, we can try out the best ways to lick *n*-5, *n*-4, and *n*-1, corresponding to deciding to first lick a five cent, a four cent, and a one cent stamp
  - Find the best among those three possibilities and add one to it, because we already licked one stamp
  - $$\texttt{lick}(n) = \min\left(\begin{cases} 0 & \texttt{if } n = 0 \\ 1 + \texttt{lick}(n-5) & \texttt{if } n \geq 5 \\ 1 + \texttt{lick}(n-4) & \texttt{if } n \geq 4 \\ 1 + \texttt{lick}(n-1) & \texttt{if } n \geq 1 \end{cases}\right)$$

Memoization

- This gives immediately a nice recursive implementation

```
def lick(amount):
    if amount == 0:
        return 0
    if amount < 4:
        return amount
    if amount == 4:
        return 1
    if amount == 5:
        return 1
    return min([lick(amount-1), lick(amount-4), lick(amount-5)])+1
```

Just some special base cases

Memoization

- But this recursive version is very slow
  - That is because the same value `link(x)` can be calculated many times
  - So we put the intermediate results into a dictionary to remember them
  - This is called memoization

Memoization

- Memoization:
  - Maintains a **cache** of intermediate results
  - If the function is called on a value:
    - We first check whether the value is in the cache
    - Otherwise: we calculate it and put it into the cache

Memoization

```
lick_dictionary = {0:0, 1:1, 2:2, 3:3, 4:1, 5:1}

def mlick(amount):
    if amount in lick_dictionary:
        return lick_dictionary[amount]
    else:
        result = min([mlick(amount-1), mlick(amount-4), mlick(amount-5)])+1
        lick_dictionary[amount] = result
        return result
```