Dictionaries

25 Nov 2010
CMPT140
Dr. Sean Ho
Trinity Western University

TRINITY WESTERN UNIVERSITY

What's on for today

- Dictionaries
  - Keys and values
  - Basic methods: .keys(), .values(), .items()
  - Iterating through dictionaries
  - Other methods: len(), del, in, .get(), .copy()
  - Application: hinting
  - Application: word frequencies

Dictionaries

- Python dictionaries are mutable, unsorted containers holding associative key-value pairs
- Create a dictionary with curly braces {}:
  - **appleInv = {'Fuji': 10, 'Gala': 5, 'Spartan': 7}**
- Index a dictionary using a key:
  - **appleInv['Fuji'] # returns 10**
- Values can be any object and may mix types:
  - **appleInv['Rome'] = range(3)**
- Keys can be any immutable type (even tuples!):
  - **appleInv[('BC', 'Red Delicious')] = 12**

keys() and values()

- All dictionaries have the following methods:
  - keys(): returns an iterable view of all the keys
    - list(appleInv.keys())

    ['Fuji', 'Spartan', 'Rome', 'Gala',
    ('BC', 'Red Delicious')]
  - values(): returns an iterable view of values
    - list(appleInv.values())

    [10, 7, [0, 1, 2], 5, 12]
- Dictionaries are unsorted!
  - Although the order of keys() and values() will correspond if the dictionary isn't modified

Iterating through dictionaries

- for iterates over the keys in a dictionary
- To print our apple inventory:

```
for appleType in appleInv:
    print( "We have %s %s apples" %
        ( appleInv[appleType], appleType ) )
```

- Output:
  - We have 10 Fuji apples.
  - We have 7 Spartan apples.
  - We have [0, 1, 2] Rome apples.
  - We have 5 Gala apples.
  - We have 12 ('BC', 'Red Delicious') apples.

Other dictionary methods

- len(appleInv)
- del appleInv['Fuji']
- 'Fuji' in appleInv
- appleInv.get('Braeburn', 0)
  - Return default value if key is not in dictionary
- list(appleInv.items())
  - Returns a copy of the dictionary as a list of (key, value) tuples
- appleInv.copy()
  - Shallow copy (only to first level of depth)

Dictionary application: hinting

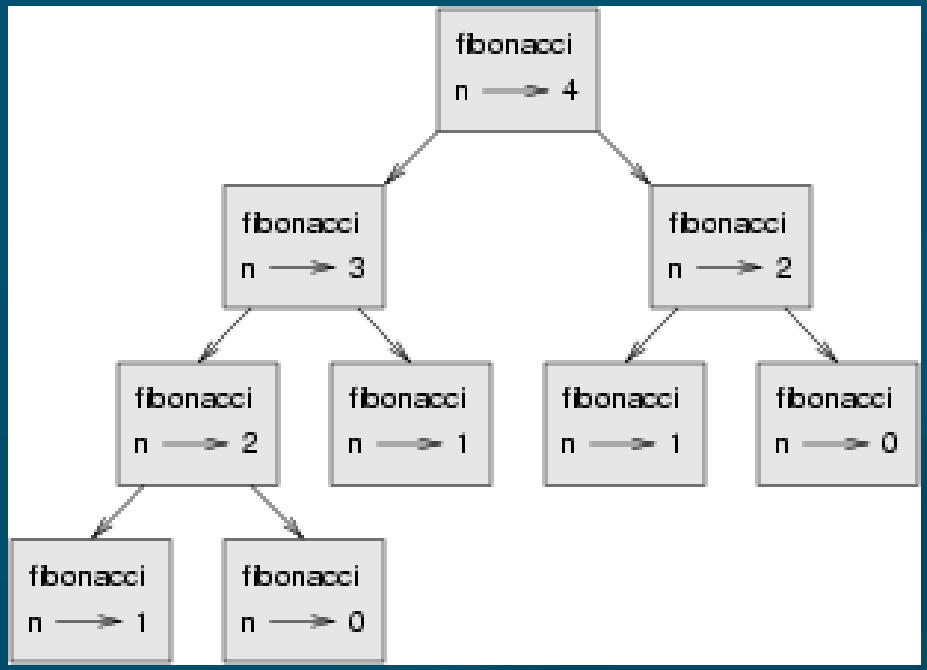
- Hinting: save (cache) previously-calculated values for future use
- Fibonacci example from last time:

```
def fib(n):
    if n == 0 or n == 1:
        return 1
    return fib(n-1) + fib(n-2)
```

- This is very slow and inefficient!
  - Try fib(28), fib(29), fib(30), ...
- Fibonacci numbers get very big very fast

Fibonacci revisited

- The call-graph for fib() shows that, e.g, fib(2) gets recalculated many times:



Approximately n^2 calls in the graph

- If we save the value of fib(2) the first time it's calculated, we can reuse that hint

Hinting Fibonacci

- Use a dictionary to store precalculated hints:
  - Key is n; value is fib(n)
  - When we calculate a fib(), add it to the dict
  - Before calculating a fib(), check to see if it's already in the dictionary of hints
  - Base cases are in the initial hint dictionary

```
fibHints = {0:1, 1:1}
def hFib(n):
    if n in fibHints:
        return fibHints[n]
    fibHints[n] = hFib(n-2) + hFib(n-1)
    return fibHints[n]
```

Iterative Fibonacci

- Actually, we don't need recursion to solve Fibonacci:

```
def iFib(n):
    current = 1
    parent = 1
    grandparent = 0
    for i in range( int(n) ):
        current = grandparent + parent
        grandparent = parent
        parent = current
    return current
```

- hFib() just illustrates the concept of hinting

Application: word frequency

- Another application: count how many times each word shows up in a block of text
- If we were counting letters instead, we could use a list, since there are only 26 letters
  - But # unique words is unknown!
- Each key is a word; the value is its frequency
- Read file one word at a time
  - Increment the value associated with the given word
  - (If word not in dictionary, use 0 as value)

Word frequency: pseudocode

- Open file for reading:
  - Read one line at a time:
    - Normalize: convert to lowercase and replace all punctuation with spaces
    - Split into words
    - For each word:
      - Increment the count for that word
- Sort word list
- Output most frequently used words

Word freq: helper functions

- String methods:
  - myStr.split() splits on whitespace
  - myStr.replace(oldstr, newstr) replaces all occurences of oldstr with newstr
  - (The tokenize library has more!)
- sorted() returns a sorted copy of any list:
    - **sorted([5,2,3,1,4])**
  - Sort a dictionary: return a list of keys, sorted by value
    - **sorted(appleInv, key=appleInv.get)**

wordfreq.py

- See wordfreq.py for complete program
- Filename is hard-coded as “input.txt”
  - Canadian Charter: charter.txt
- Sorts in ascending order of frequency
- Prints the 20 most frequently-used words