Patterns/variations: Lambert/Zelle

V. Rego

Nov 12, 2015

Week 12, Examples 2

# Dictionaries

```
#1.py

''' Look up the word "fish" in an English dictionary. You don't start at the
first page (so it's not like moving
through a list to get an explanation of "fish"). There is no sequence. So
also when you want
Joseph's phone number. It's somewhere in the book, you don't care where. You
just need to get
to it through "Joseph" just like you need to get to the meaning of "fish"
through "fish".

For this we need a Python DICTIONARY. It organizes info by ASSOCIATION and
not POSITION.
In computer science the data structures based on association are also called
TABLES or
ASSOCIATION LISTS.

If you give Python a KEY it looks up a dictionary and gives you a data
value. So you can say
"Joseph" and "fish" are keys to the phone number and meaning, respectively.

A Python dictionary has ENTRIES. Each entry is a pair <key: value>

Examples:

  Phone book:  phone = { "Joseph": 494-1234, "Eggbert": 494-5127}
  General Info:  info = {"Name": Joseph, "Age": 22}

1. Notice the curly braces.

2. Dictionaries are MUTABLE. You can change the value associated with a key.
   Example:  phone["Joseph"] = "345-6789", and then the dictionary changes

3. The keys are data of any immutable type, usually strings or integers;
   the associated value
   can also be any type.

4. You may feel that entries in a Dictionary are ordered in some way. Don't
count on it. Python does its own ordering and reordering to get an efficient structure.

```python
def wait():
    x = input()

# Adding and replacing keys
info = {} # empty dictionary
wait()
info['name'] = 'Larry'
info['occupation'] = 'unintentional comedian'
wait()
print(info)
wait()
info['occupation'] = 'Bishop of Chicago'
wait()
print(info)
```

5. When key is absent from the dictionary, both key and associated value are inserted

6. When key is present, the associated value is replaced

7. You can think of a key as an index or subscript that lets you go to the associated value

```python
# Accessing values
print(info['name'])

#print(info['job']) # should give a traceback/error because "job" is not a key
```

8. If you are not sure if a key exists, you can check using dictionary method "has_key".

9. But there is a better way. Use the method `get()`.
get("job", None)
    -- takes 2 arguments, a possible key and a default value. If key exists
    in dictionary, the
    associated value is returned. If key is absent, the second argument
    (the default value you
    gave to get() is returned. In this case it is "None">

""

print(info.get("job", None))

#Deleting keys (i.e., remove a dictionary entry). Use the method pop().

""" It needs a key and an optional default value as arguments(parameters). If key exists, it
is deleted and associated value is returned; else, the default value is
returned. If key does not exist and pop() is not given a default argument, Python raises an error.

Let's delete two keys and print their associated values.""

print(" ")
wait()

print (info.pop("job", None))

print(info.pop("occupation"))

print(info)

print(" ")

#Traversing a dictionary.

#when you use a for-loop, the loop var is bound to each key in some
unspecified order

info = {"Larry":"clown", "Moe":"clown",
"Curly":"clown","Daffy":"duck","Bugs":"Bunny",
    "Elmer":"clever hunter"}

for key in info:
    print(key, info[key])    # so you see, key acts as a
subscript/index

#Notice how the order is different from what you thought it might be via the
definition
# Important: If you have duplicate keys, because of its own order Python will select and use one.

# Tuples/dictionaries
# 2.py

Lists, strings, tuples are all examples of sequences. See p 343 of text for sequence operators
(+, len(), *, :, etc. etc.)

A TUPLE looks exactly like a list but is immutable. So use round brackets
"(" , ")" instead of square
"[", "]" so we can tell the difference

Rule: if at any time you need a list and you know its structure will not change, make it a tuple. More efficient.

```python
def wait():
    x = input()

trees = ("pine", "walnut", "oak", "cedar")

wait()
print (trees)

water = ("ocean", "lake", "river", "stream", "puddle", "glass")

wait()
print (water)

nature = trees + water

wait()
print (nature)

land = ["plain", "mound", "hill", "mountain"]
      # notice "land" is a list, not a tuple

wait()
print (land)

wait()
print (tuple(land))  # we make it a tuple
```

# a bit more on dictionaries
```python
#3.py

''' If you don't want to use a for-loop (as done at end of 1.py) to access keys and associated values, you can use the method "items()"
'''

def wait():
    x = input()

grades = {95:"A+", 90:"A", 85:"B+", 80:"B", 75:"C+", 70:"C"}
wait()
print(grades.items())
#Notice how entries become tuples inside the list; you can now use a tuple of variables in a for-loop to traverse that "items()" list
wait()
for (key, value) in grades.items():
    print(key, value)

#Note: if you need a special ordering of keys, get list of keys using the "keys()" method and process it to get the order you want. For example you can sort it and then traverse it in alphabetical order. Try it for home-work. Small example below.

info = {
    "Larry":"clown",  "Moe":"clown",  
    "Curly":"clown",  "Daffy":"duck",  "Bugs":"Bunny",  
    "Elmer":"clever hunter"}
wait()
theKeys = list(info.keys())
theKeys.sort()
for key in theKeys:
    print(key, info[key])

#________________________________________________________________________________________

Do a help(dict) to see full documentation
```
Let d represent dictionary. Here are usual operations.

len(d)                             number of entries
aDict[key]                         use to insert new key, replace
value, get value of existing key   
d.get(key, [default])             return value if key exists; else
return default; if no default and   key does not exist, then error
if no default and key does not
exist, then error

d.pop(key, [default])             delete key and return value; if
key does not exist, return default.
exist, then error

list(d.keys())                    return list of keys
list(d.values())                  return list of values
list(d.items())                   return list of (key, value)
tuples

d.has_key(key)                     True if key exists; else False
for key in d                      key is bound to each key in d, in
unspecified order for traversal...

#___________________ octal to binary converter_____________________________
4.py
...

1. Binary number system: you have only 0 and 1, and every integer you know
must be represented with
0 and 1

Decimal 16 = Binary 10000

Decimal 17 = Binary 10001

Decimal 2 = Binary 10

0,1 are the binary digits. Likewise, 0,1,...........,9 are the decimal digits.

In binary, the next number after 00 and 01 is 10, which is really 2 in
decimal, i.e, 1*2^1 + 0*2^0 = 2
In decimal, the next number after 0,1,2,.....,9 is 10.

2. Decimal number system: you grew up with these. All integers are made up with
   0,1,2,3,4,5,6,7,8,9
   To get decimal ten you use 10

3. Octal number system: here you only have the numbers 0,1,2,3,4,5,6,7
   To get decimal eight you use 10
   Why does it evaluate to decimal 8?
   \[ 1\times8^1 + 0\times8^0 = 8 \text{ (decimal)} \]

A program to convert Octal numbers to binary. We'll use a dictionary. Because we "look up"
this dictionary during conversion, its called a LOOKUP TABLE

```python
oct_to_bin_table = {"0":"000","1":"001","2":"010","3":"011",
    "4":"100","5":"101","6":"110","7":"111"}

def convert(number,table):
    """ Builds and returns the base 2 representation of octal number """
    binary = " "
    for digit in number:
        binary = binary + table[digit]
    return binary

def main():
    n = eval(input("How many octal numbers? "))
    for k in range(n):
        octal_number = str(input())
        print(convert(octal_number,oct_to_bin_table))

main()
```

#
```
''' Try the following program on any text file. Make sure it ends right so the read can terminate.

#_________________ program that counts word frequencies in a text file__________________
#5.py

def byFreq(pair):
    return(pair[1])            #takes a pair as input and returns second parameter

def main():
    print("Analyze word frequency in a text file and report the n most frequent words.\

")

    symbols = '!"#$%&()*+,-./:;<=>?@[\^_]`{|}~'

    #get the words from file and sequence them
    fname = input("File to analyze: ")
    text = open(fname,"r").read()                      #reads whole file
    text = text.lower()                                #convert all to lower case
    for ch in symbols:
        text = text.replace(ch, " ")
            #remove symbols, replace by spaces
    words = text.split()                               #get list of words

    # Now make a dictionary of word counts
    counts = { }                                       #each word will have a count
    for w in words:
        counts[w] = counts.get(w,0) + 1
    # Get the n most frequent words
    n = eval(input("For how many words do you want a frequency count?"))
```

items = list(counts.items())
# items becomes a list of
tuples: [("the",25), ("it",10), ....]

items.sort()
# item list gets sorted in
alphabetical order based on word
# but we want the highest freq
word first, and next highest second, etc.

items.sort(key=byFreq, reverse = True)
# sort based on freq, high to
low

for i in range(n):
# output only top n
    word, count = items[i]
    print("{0:<15}{1:>5}".format(word, count))

if __name__ == "__main__": main()
# Simplified version of Weizenbaum's program

Functions share a common pool of data: hedges and qualifiers. These do not change, so use tuples.

```python
import random

hedges = ("Please tell me more.",
          "Many of my patients tell me the same thing.",
          "Please continue.",
          "This is not unusual; it would be good to understand more.")

qualifiers = ("Why do you say that ",
              "You seem to think that ",
              "Can you explain why ",
              "It would be helpful to understand why you feel that ")

replacements = {"I":"you", "me":"you", "My": "Your", "my":"your",
                "we":"you", "us":"you", "mine":"yours"}
```

The other set of data connects first-person pronouns and second-person pronouns so one can be replaced by the other. So "I will be replaced by "you" when the doctor responds. A dictionary is best for this.

```python
def reply(sentence):
    """ Builds and returns a reply to the sentence. """
    which_sentence = random.randint(1,5)

    if (which_sentence == 1):
        return (random.choice(hedges))
    else:
        return((random.choice(qualifiers) + changePerson(sentence)))
```

Function changePerson extracts a list of words from the person's sentence string. Any pronoun
key in the replacements dictionary is replaced by its pronoun/value. The new
list is built,
converted to a string and returned.''

def changePerson(sentence):
    """ Replaces first person pronouns with second person pronouns."""

    words = sentence.split()
    replyWords = [ ]

    for w in words:
        replyWords.append(replacements.get(w, w))

    return ( " ".join(replyWords) )

'''The main program prints a greeting, displays a prompt and waits for the
user to input.
This is usually interactive (keyboard input) but in this main program we'll
just read the
user input from a file to save typing in class. You can modify it.'''

def main():
    """ Handles the interaction between patient and doctor. """

    fname = input("File that contains patient's part of conversation? ")
    f = open(fname, "r")

    print(" ")
    print("Good morning! I hope you are well today.")
    print("What can I do for you?")
    print(" ")

    while True:
        sentence = f.readline()
        if (sentence.upper() == "QUIT\n"):#remember readline() takes in
            f.close()
            print("Have a wonderful day!")
            break
        print(">> ",sentence)
        print(reply(sentence),"\n")

main()