Lab 7: Dictionaries

Feb 15, 2018

Main Event

1. Friendly directory

We’ve built friend directories before. This time we’ll add phone numbers, and make it searchable:

1. Build a directory of names and phone numbers. Continuously ask the user for their friends name and phone number; when the empty string is entered, stop.
2. Continuously ask the user for a name. If that name is in the directory, print the phone number; otherwise, print a notification. Again, when the empty string is entered, stop the interaction.

These two problems should be accomplished in separate loops. Your code should not crash if the user requests a name that hasn’t been entered.

Solution:

```
directory = {}

while True:
    name = input('Enter name (or stop): ')
    if name.lower() == 'stop':
        break
    directory[name] = int(input('Enter number: '))

while True:
    name = input('Enter name (or stop): ')
    if name.lower() == 'stop':
        break

    if name in directory:
        output = directory[name]
    else:
        output = 'does not exist'

    print(name, '->', output)
```

2. (More) Counting characters

Aww yea, character counting is back!

1. For a given string, print the number of times each character appears in that string. Your interaction should look exactly as follows:

   ```
   Enter a string: Bah bah sheep
   a -> 2
   e -> 2
     -> 2
   s -> 1
   h -> 3
   B -> 1
   p -> 1
   b -> 1
   ```

   Note that the order of the letters, in your count output, might be different, but you should maintain the formatting. The difference arises because there is no order to key extraction (you can think of keys as a set).

2. Update your code such that the letters are presented in alphabetic order. Stack Overflow may have some interesting ways of doing this, but before resorting to them, think it through on your own: Recall the data type we have discussed that is sortable. How can this data type be used to access values of your dictionary?

Solution:

```
# sentence = input('Enter a string: ')
sentence = 'Bah bah sheep'

counts = {}
for i in sentence:
    if i not in counts:
        counts[i] = 0
```

```
    counts[i] += 1

for i in counts:
    print(i, '->', counts[i])

print('alphabetic order:')
ordered = list(counts.keys())
ordered.sort()
for i in ordered:
    print(i, '->', counts[i])
```

3. Creation and inversion

This exercise requires the use of a helpers module. Remember, the module should be in the same directory as your Python implementation (for this lab/problem).

Dictionaries really only get interesting when you have a lot of data, and that data is added dynamically. This is usually the case when you're reading from files—something we haven't talked about yet; but this can be *faked* with random numbers—something that we have. The next two exercises will utilize this concept. In the first, you'll need to use the `genkey` function in helpers. The function returns a string that you can use as a key:

```
import helpers
usrdict = {}
key = helpers.genkey()
usrdict[key] = 0 # zero is just an example!
```

1. Add 100 elements to the dictionary. Their keys should come from `genkey` and their values should be a random number between zero and nine.

   After you have made your dictionary, use `ishundred` from the helpers module to test whether you actually have 100 elements:

   ```
   userdict = {}
   # ... updates to usrdict...
   helpers.ishundred(userdict)
   ```

   If this function prints “You’re okay!”, then you have completed the activity successfully. If, on the other hand you see a `ValueError`, you have done something wrong. Reassess your code that populates the dictionary to figure why you do not have 100 elements.

2. Invert the dictionary: the values should become keys, and the keys should become values. An example two-key dictionary:

   ```
   >>> print(original_dictionary)
   { 'aaa': 10, 'bbb': 12 }
   >>> print(inverted_dictionary)
   { 10: 'aaa', 12: 'bbb' }
   ```

   This example is simple. In reality, since the values are not unique, you'll have to come up with a way to associate values with all the of the keys that point to them. Remember, while dictionary keys have to be immutable, dictionary values don't—think of all the other data types we've used!

Solution:

```
import random
import helpers

dictionary = {}
while len(dictionary) < 100:
    key = labhelpers.genkey()
    dictionary[key] = random.randrange(10)

print('Do you have 100 elements?', labhelpers.ishundred(dictionary))

inverted = {}
for key in dictionary:
    value = dictionary[key]
    if value not in inverted:
        inverted[value] = []
    inverted[value].append(key)

for i in inverted:
    print(i, inverted[i])
```

4. Dragon breeder

Here, we'll be analyzing a random hoard of dragons (technically, a "weyr").

First, we breed

Each dragon is going to have a color, age, size, and breed. Possible breeds and color options are:

Color	Breed
black	Antipodean Opaleye
white	Chinese Fireball
red	Common Welsh Green
blue	Hebridean Black
	Hungarian Horntail
	Norwegian Ridgeback
	Peruvian Vipertooth
	Romanian Longhorn
	Swedish Short-Snout
	Ukrainian Ironbelly

These lists can be accessed from the helpers module as `helpers.dragons.colors` and `helpers.dragons.breeds`, respectively.

Dragon ages can range from 0 to 99; their size between 200 and 399. A random dragon might look as follows:

```
{'color': 'black', 'age': 1, 'size': 215, 'breed': 'Hungarian Horntail
```

Your tasks:

- Populate a list with 300 random dragons.
- When the user gives you an attribute and a value, tell the user how many dragons fit that description. For example:

  ```
  What do you want to know: color black
  500 black dragons
  ```

  When taking input, assume whatever separator and “stop” word (to break the loop) you like.
- Find the largest dragon in each breed.
- Dragon breeder statistics:

- What is the average age of your weyr?
- The average size?
- Breed distribution?

Solution:

```
import random
import helpers

population = 300

# 1 (populate the list)
weyr = []
for i in range(population):
    dragon = {
        'age':   random.randrange(100),
        'size':  random.randrange(200, 400),
        'color': random.choice(helpers.dragons.colors),
        'breed': random.choice(helpers.dragons.breeds),
    }
    weyr.append(dragon)

# 2 (attribute lookup)
while True:
    request = input('What do you want to know: ')
    if request.lower() == 'stop':
        break
    (key, value) = request.split() # https://tinyurl.com/nyou7oy
    if key == 'age' or key == 'size':
        value = int(value)

    count = 0
    for dragon in weyr:
        if key in dragon:
            count += dragon[key] == value
    plural = '' if count == 1 else 's'

    print(count, 'dragon' + plural)

# 3 (largest dragon in each breed)
largest = {}
for dragon in weyr:
    breed = dragon['breed']
    if breed in largest:
        d = largest[breed]
        if d['size'] < dragon['size']:
            continue
    largest[breed] = dragon

for dragon in largest.values():
    print(*dragon.values(), sep=',') # https://tinyurl.com/k6vpdxb

# 4 (dragon breeder stats)
ages = []
sizes = []
breed_pcts = {}
for dragon in weyr:
    ages.append(dragon['age'])
    sizes.append(dragon['size'])

    breed = dragon['breed']
    if breed not in breed_pcts: # Tired of doing this? Checkout defaul
        breed_pcts[breed] = 0
    breed_pcts[breed] += 1

print('Average age:', round(sum(ages) / len(ages)), 2)
print('Average size:', round(sum(sizes) / len(sizes), 2))
breeds = sum(breed_pcts.values())
for (breed, count) in breed_pcts.items():
    print(breed, round(count / breeds, 2))
```

Additional Practice

1. Fight club

Now that you've warmed up, it's time to get fancy. So far you've basically built an imitation of a database system, which can be used to store anything from financial data to Facebook likes. We now bring them to life! Each battle should start with the full set of 300 dragons.

1. The dragons are free. Any dragon caught in between two larger dragons will be eaten—how many dragons are left after the slaughter?

2. Dragons of the same species will not attack each other. A dragon that is twice as large as another can kill it without needing a partner. How many dragons remain?
3. Kill dragons by age until there are less than 100 dragons. How many dragons are left?

Top challenge

Let’s do some population statistics. Run the battles at least 100 times on 100 different populations of dragons. Which kind of battle is statistically most detrimental to the population?
