DePaul University
College of Computing and Digital Media

CSC 243 NOTES
Fall 2020: Amber Settle

Monday, October 12, 2020

Announcements

- Questions about the fourth quiz or third assignment?
- The fourth assignment is due Thursday, October 15th at 9 pm – questions?
- The midterm exam will be next week
    - Study guide posted to D2L
    - Questions?

Dictionaries

Suppose you want to store employee records in a collection and access the employee records using the employees' social security numbers.

The list collection is not really useful for this problem because list items are accessed using an index, i.e. a position in a collection.

For this problem we need a collection type that allows access to stored items using user-defined "indices", which are referred to as keys.

```
>>> employees = {'022-83-8795': 'Allison
Andover', '987-65-4321': 'Benjamin Benton',
'123-45-6789': 'Cassidy Cantor', '874-74-1532':
'Daniel Danton'}
>>> employees['123-45-6789']
'Cassidy Cantor'
```

A **dictionary**, also called a map, is a collection type that stores key-value pairs.

In the employee dictionary above, '022-83-8795': 'Allison Andover' is a **key-value pair**.

The **key** '022-83-8795' is the "index" of the stored object and is used to access the object. The key must be an object of an immutable type (like a string or number or tuple).

The following are the **dictionary collection type properties**:

- Items in the collection are accessed by key, not index (e.g. offset)
- Items in the collection are not ordered, and no ordering assumption can be made

- Dictionaries are mutable, like lists, meaning they can be modified to contain different items.
- Dictionaries have dynamic size, meaning they can grow or shrink

Dictionary methods

The dictionary type is different from the list type and supports different methods. For example, consider the following:

```
>>> d = {'Mo': 'Monday', 'Tu': 'Tuesday', 'We':
'Wednesday', 'Th': 'Thursday', 'Fr': 'Friday', 'Sa':
'Saturday', 'Su': 'Sunday'}
>>> d
{'Fr': 'Friday', 'Mo': 'Monday', 'Tu': 'Tuesday',
'We': 'Wednesday', 'Su': 'Sunday', 'Th': 'Thursday',
'Sa': 'Saturday'}
>>> d['We']
'Wednesday'
>>> 'Su' in d
True
>>> d.keys()
dict_keys(['Fr', 'Mo', 'Tu', 'We', 'Su', 'Th', 'Sa'])
>>> d.values()
dict_values(['Friday', 'Monday', 'Tuesday',
'Wednesday', 'Sunday', 'Thursday', 'Saturday'])
>>> d.items()
dict_items([('Fr', 'Friday'), ('Mo', 'Monday'),
('Tu', 'Tuesday'), ('We', 'Wednesday'), ('Su',
'Sunday'), ('Th', 'Thursday'), ('Sa', 'Saturday')])
>>> type(d.keys())
<class 'dict_keys'>
>>> del(d['Sa'])
>>> d
{'Fr': 'Friday', 'Mo': 'Monday', 'Tu': 'Tuesday',
'We': 'Wednesday', 'Su': 'Sunday', 'Th': 'Thursday'}
>>> d.clear()
>>> d
{}
```

As usual, you can learn about the methods supported by the dictionary type by typing:

```
>>> help(dict)
Help on class dict in module builtins:
[…]
```

Dictionary problems

To better understand the dictionary type, we'll do some sample problems.

Problem 1: Write a function **copyDict**() that takes a dictionary as a parameter and makes a new copy of the dictionary.

The function could be used as follows:

```
>>> d1 = {'Annika': 1, 'Erin': 16, 'Amber': 52,
'Chad': 50}
>>> d2 = copyDict(d1)
>>> d2
{'Annika': 1, 'Erin': 16, 'Amber': 52, 'Chad':
50}
>>> id(d1)
1675591942720
>>> id(d2)
1675591942504
>>> d1 == d2
True
>>>
```

Problem 2: Write a function **inverse**() that takes as a parameter a dictionary and builds and returns the inverse dictionary. The inverse dictionary is one that flips the keys and values in the dictionary. You can assume that the values in the dictionary are valid as keys and that there are no duplicate values in the dictionary.

The function could be used as follows:

```
>>> d1 = {'Annika': 'tuna', 'Amber': 'chocolate',
'Chad': 'beer', 'Erin': 'chips'}
>>> d2 = inverseDict(d1)
>>> d2
{'tuna': 'Annika', 'chocolate': 'Amber', 'beer':
'Chad', 'chips': 'Erin'}
>>> d1
{'Annika': 'tuna', 'Amber': 'chocolate', 'Chad':
'beer', 'Erin': 'chips'}
>>>
```

Problem 3: Write a function **wordFreq**() that takes as a parameter a file name (i.e. a string object). The function should compute the frequency of each word in the file and then return a dictionary with keys representing words in the file and values representing the frequency of the word. You should use the file **sample.txt** to test your code. Remove all punctuation and capitalization from the words before you process them to ensure accurate results. The function would be used as follows:

```
>>> d = wordFreq('sample.txt')
>>> d
{'this': 2, 'is': 1, 'a': 2, 'test': 1, 'file': 1,
'how': 2, 'many': 2, 'words': 2, 'are': 2, 'of': 2,
'length': 2, 'one': 1, 'three': 1, 'we': 1, 'should':
1, 'figure': 1, 'it': 1, 'out': 1, 'can': 1,
'function': 1, 'do': 1}
>>> d = wordFreq('duplicates.txt')
>>> d
{'this': 1, 'file': 1, 'has': 1, 'duplicates': 2,
'for': 1, 'example': 1, 'several': 1, 'words': 2,
'appear': 1, 'more': 1, 'than': 1, 'once': 1,
'something': 1, 'with': 1, 'that': 1, 'is': 1,
'repeated': 1, 'should': 1, 'return': 1, 'true': 1,
'when': 1, 'given': 1, 'to': 1, 'the': 1, 'function':
1}
>>>
```

Problem 4: Write a function called **duplicates**() that returns True if there are any duplicate words in the file and False if there are no duplicates in the file. The function must call wordFreq() to get a dictionary with the frequency of each word in the file and then use that dictionary to determine what value to return.

```
>>> ans = hasDuplicates('sample.txt')
>>> ans
True
>>> ans = hasDuplicates('duplicates.txt')
>>> ans
True
>>> ans = hasDuplicates('noDuplicates.txt')
>>> ans
False
>>>
```

Problem 5: Suppose you want to find a count of each element in a two-dimensional list. How do you have to modify the above solution to do that?

For example, we want it to produce something like this:

```
>>> d = itemFreq([[1, 3, 1, 4, 1], [5, 1, 3], [3, 4,
7, 1, 2]])
>>> d
{1: 5, 3: 3, 4: 2, 5: 1, 7: 1, 2: 1}
>>> d = itemFreq([['one', 2, 'three'], [4, 5, 6],
['seven', 7.7, 8, 'nine']])
>>> d
{'one': 1, 2: 1, 'three': 1, 4: 1, 5: 1, 6: 1,
'seven': 1, 7.7: 1, 8: 1, 'nine': 1}
>>>
```

See the solutions in **collectSols.py**.

More collection types

Python supports two more collection types: **sets** and **tuples**.

The set collection type

The set type is a collection type used to store an unordered collection of immutable values.

One use of sets is in removing duplicates from sequences since sets do not contain duplicates by definition.

Consider the following examples:

```
>>> grains = {'rice', 'wheat', 'corn', 'rye', 'oat',
'wheat', 'millet'}
>>> grains
{'wheat', 'corn', 'rye', 'millet', 'oat', 'rice'}
>>> type(grains)
<class 'set'>
>>> fruit = {}
>>> type(fruit)
<class 'dict'>
>>> fruit = set()
>>> fruit
set()
>>> fruit.add("apple")
>>> fruit
{'apple'}
```

It supports functions that manipulate sets, including set membership, set intersection, union, difference, etc.

- **Set membership** (in): Returns True if the item is in the set and False otherwise
- The **union** (|): The set produced by combining the elements of the two sets
- The **difference** (&): The set produced by considering only the items that are in both sets

Consider the following examples:

```
>>> s = {1, 2, 3}
>>> t = {2, 3, 4}
>>> s | t
{1, 2, 3, 4}
>>> s&t
{2, 3}
>>> s^t
{1, 4}
>>> s-t
{1}
>>> t-s
{4}
>>> (s-t) | (t-s)
{1, 4}
```

The tuple collection type

The tuple type is the same as a list except that **a tuple is immutable**, i.e. a tuple cannot be modified.

Consider the following example:

```
>>> t = (1,2)
>>> t
(1, 2)
>>> type(t)
<class 'tuple'>
>>> t[0]
1
>>> t[1] = 3
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    t[1] = 3
TypeError: 'tuple' object does not support item
assignment
```

The **tuple collection type properties** are:

- Items in the collection are ordered
- Items in the collection are accessed using an index (offset)
- Like strings and unlike lists, tuples are immutable
- Tuples have a fixed length, since they cannot change

  Because tuple objects are immutable, they can be used as dictionary keys.

To learn more, write:

```
>>> help(tuple)
Help on class tuple in module builtins:

class tuple(object)
[…]
```

Collection problems

Practice problem 1: Implement a function **lookup**() that provides the phonebook lookup feature.

The function takes as a parameter a dictionary representing a phonebook. In the dictionary, tuples containing first and last names of individuals (the keys) are mapped to strings containing phone numbers (the values).

Your function should provide a simple user interface through which a user can enter the first and last name of an individual and obtain the phone number assigned to that individual. It should prompt the user and lookup values until an empty first name and last name are entered, at which point the function should stop.

For example, it would be used as follows:

```
>>> phonebook = {('Kat', 'Elam'): '(312) 123-4567',\
           ('Django', 'Settle'): '(773) 987-6543',\
           ('Cookie', 'Settle'): '(520) 454-6677'}
>>> lookup(phonebook)
Enter the first name: Kat
Enter the last name: Elam
(312) 123-4567
Enter the first name: Cookie
Enter the last name: Settle
(520) 454-6677
Enter the first name: Amber
Enter the last name: Settle
Unlisted number
Enter the first name:
Enter the last name:
Goodbye!
```

Practice problem 2: Re-write the word frequency problem into a different function **uniqueWords**() that returns an alphabetical list of the unique words in the file.

```
>>> lst = uniqueWords('sample.txt')
>>> lst
['a', 'are', 'can', 'do', 'figure', 'file',
'function', 'how', 'is', 'it', 'length', 'many',
'of', 'one', 'out', 'should', 'test', 'this',
'three', 'we', 'words']
```

Hint: Use a set to find the unique words.

Practice problem 3: Re-write the 2D list frequency problem into a different function **uniqueItems**() that returns a sorted list of the unique values in a 2D list.

```
>>> lst = uniqueItems([[1, 3, 1, 4, 1], [5, 1, 3],
[3, 4, 7, 1, 2]])
>>> lst
[1, 2, 3, 4, 5, 7]
>>> lst = uniqueItems([['Amber', 'Chad'],
['Gertrude', 'Prudence'], ['Amber', 'Erin',
'Andre']])
>>> lst
['Amber', 'Andre', 'Chad', 'Erin', 'Gertrude',
'Prudence']
>>>
```

See the solutions in **collectSols.py**.