THE UNIVERSITY OF WESTERN AUSTRALIA

Lecture 21
Dictionary operations

Revision: Sequence Operations

Operator	Meaning
<seq> + <seq>	Concatenation
<seq> * <int-expr>	Repetition
<seq>[]	Indexing
len(<seq>)	Length
<seq>[:]	Slicing
for <var> in <seq>:	Iteration
<expr> in <seq>	Membership (Boolean)

Revision: List Operations

Method	Meaning
<list>.append(x)	Add element x to end of list.
<list>.sort()	Sort the list. A comparison function may be passed as a parameter. By default sorted in ascending order
<list>.reverse()	Reverse the list.
<list>.index(x)	Returns index of first occurrence of x.
<list>.insert(i, x)	Insert x into list at index i.
<list>.count(x)	Returns the number of occurrences of x in list.
<list>.remove(x)	Deletes the first occurrence of x in list.
<list>.pop(i)	Deletes the i^{th} element of the list and returns its value.

Revision: Tuples and Dictionaries

- A ***tuple*** is a sequence which looks like a list but uses `()` rather than `[]`.
- Tuples are sequences that are **immutable**, so are used to represent sequences that are not supposed to change.

- Python ***dictionaries*** are mappings.
- Dictionary is a collection that allows us to look up information associated with arbitrary keys.
- Dictionaries use key-value pairs.

Dictionary Operations

Method	Meaning
<key> in <dict>	Returns true if dictionary contains the specified key, false if it doesn't.
<dict>.keys()	Returns a sequence of keys.
<dict>.values()	Returns a sequence of values.
<dict>.items()	Returns a sequence of tuples (key, value) representing the key-value pairs (i.e., 2-tuples).
del <dict>[<key>]	Deletes the specified entry.
<dict>.clear()	Deletes all entries.
for <var> in <dict>:	Loop over the keys.
<dict>.get(<key>, <default>)	If dictionary has key, returns its value; otherwise returns default.
<dict>[<key>]	If dictionary has key, return its value; otherwise, exception raised

Dictionary Operations

```
>>> list(passwd.keys())
['guido', 'turing', 'bill']
>>> list(passwd.values())
['superprogrammer', 'genius', 'bluescreen']
>>> list(passwd.items())
[('guido', 'superprogrammer'), ('turing', 'genius'),
('bill', 'bluescreen')]
>>> "bill" in passwd
True
>>> "fred" in passwd
False
```

List of 2-tuples

Pythonic Ways to Use Dictionaries

Use **get()** and **setdefault()** with Dictionaries

Trying to access a dictionary key that doesn’t exist will result in an error:

- How to avoid an error if the accessed key does not exist?

```
# Unpythonic Example
>>> Pets = {‘monkeys’: 1}
 if 'cats' in Pets: # Check if 'cats' exists as a key
     print('I have', Pets['cats'], 'cats.’)
 else:
     print('I have 0 cats.')
```

Pythonic Ways to Use Dictionaries

Dictionaries have a **get()** method that allows you to specify a default value to return when a key doesn't exist in the dictionary.

```
# Pythonic Example
>>> Pets = {‘monkeys’: 1}
>>> print('I have ', Pets.get('cats', 0), ‘ cats.’)

‘I have 0 cats’.
```

Pythonic Ways to Use Dictionaries

```
>>> passwd.get("guido", "unknown")
'superprogrammer'
>>> passwd.get("fred", "unknown")
'unknown'
>>> passwd["fred"]
Traceback (most recent call last):
  File "<pyshell>", line 1, in <module>
KeyError: 'fred’
>>> passwd.clear()
>>> passwd
{}
```

Pythonic Ways to Use Dictionaries

Dictionaries also have a pythonic method **setdefault()** which helps set a default value if a key doesn't exist

```
# Pythonic Example

>>> Pets = {'monkeys': 2}
 >>> Pets.setdefault('cats', 1) # Does nothing if
'cats' exists.
>>> Pets['cats'] += 10
>>> Pets['cats']

11
```

Sets

Python provides another built-in type, called a **set**, which behaves like a collection of dictionary keys with no values. Set can have unique items only.

- Sets provide methods and operators to compute common set operations
  - *E.g., Let's check whether set A is a subset of another set B*

```
>>> A = {monkeys', 'cats', 'dogs'} # create a set A
>>> B = {koalas', 'dogs'} # create a set B
>>> A <=B
```

Reference: 'Think Python-How to Think Like a Computer Scientist", Allen Downey, 2nd Edition, Version 2.4

Set Methods

We can use add() method to add an item to a set:

```
>>> B.add('parrot')
```

Operators	English	Meaning
|	union	Returns the union of two sets
&	intersection 交集	Returns the intersection of two sets
–	difference	Returns the difference between two sets

```
>>> print(A|B)
>>> print(A&B)
>>> print(A-B)
```

Read more: https://betterprogramming.pub/mathematical-set-operations-in-python-e065aac07413

Revision

Data Types	**Mutable**	**Sequence**
List	Yes	Yes
Tuple	No	Yes
String	No	Yes
Set	Yes	No
Dictionary	Yes	No

Example Program: Word Frequency

- We want to write a program that analyzes text documents and counts how many times each word appears in the document.
- This kind of analysis is sometimes used as a crude measure of the style similarity between two documents and is used by automatic indexing and archiving programs (like Internet search engines).

Algorithm

Steps:

1. Program introduction display
2. Read the file
3. Remove special characters and make it case insensitive
4. Split the text file in unique words
5. Count words appearances
6. Format the results to display
7. Display the results

Example Program: Word Frequency

- This is a multi-accumulator problem!
- We need a count for each word that appears in the document.
- We can use a loop that iterates over each word in the document, incrementing the appropriate accumulator.
- The catch: we will likely need hundreds, perhaps thousands of these accumulators!

Example Program: Word Frequency

- Let's use a dictionary where strings representing the words are the keys and the values are `int`s that count up how many times each word appears.
  - *The mapping is: <string> $\rightarrow$ <int>*
- To update the count for a particular word, `w`, we need something like:
  ```
  counts[w] += 1
  ```
- One problem – the first time we encounter a word it will not yet be in `counts`.

Example Program: Word Frequency

- Attempting to access a nonexistent key produces a run-time `KeyError`.

Pseudo–code

```
if w is already in counts:
   add one to the count for w
else:
   set count for w to 1
```

- How can this be implemented?

Example Program: Word Frequency

```
if w in counts:
   counts[w] += 1
else:
   counts[w] = 1
```

Can't do this in Python 2

- A more elegant approach:

```
counts[w] = counts.get(w, 0) + 1
```

- If `w` is not already in the dictionary, this `get` will return 0, and the result is that the entry for `w` is set to 1.

Example Program: Word Frequency

- The other tasks include
  - *Convert the text to lowercase (so occurrences of "Python" match "python")*
  - *Eliminate punctuation (so "python!" matches "python")*
  - *Split the text document into a sequence of words*

Example Program: Word Frequency

```
# get the sequence of words from the file
   fname = input("File to analyze: ")
   try:
     text = open(fname,'r').read()
   except IOError:
     print("Cannot open the file",fname)
     return
   text = text.lower()
   for ch in '!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~':
       text = text.replace(ch, ' ')
```

Example Program: Word Frequency

- Variable `text` has all the words in the file. Multiple spaces not a problem for `split()`

```
words = text.split()
```

- Loop through the words to build the counts dictionary

```
counts = {}
for w in words:
    counts[w] = counts.get(w,0) + 1
```

Example Program: Word Frequency

- print a list of words in alphabetical order with their associated counts

```
# get list of words that appear in document
# each word (i.e., key) is found only once!
uniqueWords = list(counts.keys())

# put list of words in alphabetical order
uniqueWords.sort()

# print words and associated counts
for w in uniqueWords:
    print(w, counts[w])
```

Example Program: Word Frequency

- This will probably not be very useful for large documents with many words that appear only a few times.
  - *Result will be a huge list*
- A more interesting analysis is to print out the counts for the *n* most frequent words in the document.
- To do this, we'll need to create a list that is sorted by counts (most to fewest), and then select the first *n* items.

Example Program: Word Frequency

- We can start by getting a list of key-value pairs using the `items` method for dictionaries.

```
pairs = list(count.items())
```

- `pairs` will be a list of tuples like

```
[('foo', 5), ('bar', 7), ('spam', 376)]
```

- If we try to sort them with `pairs.sort()`, they will be in ascending order of first component, i.e., dictionary order of the words.

```
[('bar', 7), ('foo', 5), ('spam', 376)]
```

Example Program: Word Frequency

- Not what we wanted.
- To sort the items by frequency, we need a function that will take a tuple (here, 2-tuple) and return the second term, i.e., count.

```
def byCount(pair):
    return pair[1]
```

- To sort the list by frequency:

```
pairs.sort(key=byCount)
```

- Similarly, we can do the following which we have discussed in earlier lecture

```
pairs.sort(key=lambda x:x[1],reverse=True)
```

Example Program: Word Frequency

- We're getting there!
- What if have multiple words with the same number of occurrences? We'd like them to print in alphabetical order.
- That is, we want the list of pairs primarily sorted by count but sorted alphabetically within each level.

Example Program: Word Frequency

- Looking at the documentation for sort, it says this method performs a "*stable* sort **in place**".
  - *"In place" means the method modifies the list that it is applied to, rather than producing a new list.*
  - *Stable means equivalent items (equal keys) stay in the same relative position to each other as they were in the original list.*

Example Program: Word Frequency

- If all the words were in alphabetical order before sorting them by frequency, words with the same frequency will be in alphabetical order!
- We just need to sort the list twice – first by words, then by frequency.

```
pairs.sort()                          # orders pairs alphabetically
pairs.sort(key = byCount, reverse = True) # orders by count
```

- Setting `reverse` to `True` tells Python to sort the list in reverse order.

Example Program: Word Frequency

- Now we are ready to print a report of the *n* most frequent words.
- Here, the loop index `i` is used to get the next pair from the list of items.
- That pair is unpacked into its `word` and `count` components.
- The word is then printed left-justified in fifteen spaces, followed by the count right-justified in five spaces.

Example Program: Word Frequency

```
for i in range(n):
    word, count = pairs[i]
    print(f"{word :<15}{count :>5}“)
```

Example Program: Word Frequency

```
# A program to count word frequencies in text file
def byCount(pair):  # service function, select second of pair
    return pair[1]

def main():
    print("This program counts word frequency in a file and")
    print(”prints a report on the n most frequent words.\n")
    # get the sequence of words from the file
    fname = input("File to analyze: ")
    text = open(fname,'r').read()
    text = text.lower()
     for ch in '!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~':
        text = text.replace(ch, ' ’)
```

Example Program: Word Frequency

```
words = text.split()

# construct a dictionary of word counts
counts = {}
for w in words:
    counts[w] = counts.get(w,0) + 1
# output analysis of n most frequent words.
n = int(input("Output analysis of how many words? "))
items = list(counts.items())  # word-count pair list
items.sort()   # alphabetic sort
items.sort(key=byCount, reverse=True)
for i in range(n):
    word, count = items[i]
    print(f"{word:<15}{count :>5}“)
```

Summary

- We completed looking at Python lists, noting that many of the functions are actually methods that change the input list, esp. append and sort.
- We looked at tuples, as a special sort of list.
- We looked at dictionaries, as a mapping from keys to values which is not restricted to the order of items