Dictionaries in Python

Dictionaries are Python's built-in data type for storing and manipulating correspondence relationships. A dictionary contains key-value pairs, where both the keys and values are objects. The purpose of a dictionary is to provide an abstraction for storing and retrieving values that are indexed not by consecutive integers, but by descriptive keys.

Strings commonly serve as keys, because strings are our conventional representation for names of things. This dictionary literal gives the values of various Roman numerals.

```
>>> numerals = {'I': 1.0, 'V': 5, 'X': 10}
```

Looking up values by their keys uses the element selection operator that we previously applied to sequences.

```
>>> numerals['X']
10
```

A dictionary can have at most one value for each key. Adding new key-value pairs and changing the existing value for a key can both be achieved with assignment statements.

```
>>> numerals['I'] = 1
>>> numerals['L'] = 50
>>> numerals
{'I': 1, 'X': 10, 'L': 50, 'V': 5}
```

Notice that `'L'` was not added to the end of the output above. Dictionaries are unordered collections of key-value pairs. When we print a dictionary, the keys and values are rendered in some order, but as users of the language we cannot predict what that order will be.

Dictionaries can appear in environment diagrams as well.

```
1  numerals = {'I': 1, 'V': 5, 'X': 10}
```

```
2  numerals['L'] = 50
```

Edit code

< Back Step 1 of 2 Forward >

The dictionary abstraction also supports various methods of iterating of the contents of the dictionary as a whole. The methods `keys`, `values`, and `items` all return iterable values.

```
>>> sum(numerals.values())
66
```

A list of key-value pairs can be converted into a dictionary by calling the `dict` constructor function.

```
>>> dict([(3, 9), (4, 16), (5, 25)])
{3: 9, 4: 16, 5: 25}
```

Dictionaries do have some restrictions:

- A key of a dictionary cannot be an object of a mutable built-in type.
- There can be at most one value for a given key.

This first restriction is tied to the underlying implementation of dictionaries in Python. The details of this implementation are not a topic of this text. Intuitively, consider that the key tells Python where to find that key-value pair in memory; if the key changes, the location of the pair may be lost.

The second restriction is a consequence of the dictionary abstraction, which is designed to store and retrieve values for keys. We can only retrieve *the* value for a key if at most one such value exists in the dictionary.

A useful method implemented by dictionaries is `get`, which returns either the value for a key, if the key is present, or a default value. The arguments to `get` are the key and the default value.

```
>>> numerals.get('A', 0)
0
>>> numerals.get('V', 0)
5
```

Dictionaries also have a comprehension syntax analogous to those of lists and generator expressions. Evaluating a dictionary comprehension yields a new dictionary object.

```
>>> {x: x*x for x in range(3,6)}
{3: 9, 4: 16, 5: 25}
```

Source :http://inst.eecs.berkeley.edu/~cs61A/book/chapters/objects.html#dictionaries