**This Next Part is Key**

A dictionary is similar to a list, but you access values by looking up a *key* instead of an index. A key can be any string or number. Dictionaries are enclosed in curly braces, like so:

d = {'key1' : 1, 'key2' : 2, 'key3' : 3}

This is a dictionary called d with three *key-value pairs*. The key 'key1' points to the value 1, 'key2' to 2, and so on.

Dictionaries are great for things like phone books (pairing a name with a phone number), login pages (pairing an e-mail address with a username), and more!

|  |
| --- |
| # Assigning a dictionary with three key-value pairs to residents:  residents = {'Puffin' : 104, 'Sloth' : 105, 'Burmese Python' : 106}  print residents['Puffin'] # Prints Puffin's room number  # Your code here!  print residents['Sloth']  print residents['Burmese Python'] |

Print the values stored under the 'Sloth' and 'Burmese Python' *keys*. Accessing dictionary values by key is just like accessing list values by index:

residents['Puffin']# Gets the value 104

Check the Hint if you need help!

**New Entries**

Like Lists, Dictionaries are *mutable*. This means they can be changed after they are created. One advantage of this is that we can add new *key/value pairs* to the dictionary after it is created like so:

dict\_name[new\_key] = new\_value

An empty pair of curly braces {} is an empty dictionary, just like an empty pair of [] is an empty list.

The length len() of a dictionary is the number of key-value pairs it has. Each pair counts only once, even if the value is a list. (That's right: you can put lists *inside*dictionaries!)

**New Entries**

Like Lists, Dictionaries are *mutable*. This means they can be changed after they are created. One advantage of this is that we can add new *key/value pairs* to the dictionary after it is created like so:

dict\_name[new\_key] = new\_value

An empty pair of curly braces {} is an empty dictionary, just like an empty pair of [] is an empty list.

The length len() of a dictionary is the number of key-value pairs it has. Each pair counts only once, even if the value is a list. (That's right: you can put lists *inside*dictionaries!)

# Changing Your Mind

Because dictionaries are mutable, they can be changed in many ways. Items can be removed from a dictionary with the delcommand:

del dict\_name[key\_name]

will remove the key key\_name and its associated value from the dictionary.

A new value can be associated with a key by assigning a value to the key, like so:

dict\_name[key] = new\_value

# It's Dangerous to Go Alone! Take This

Let's go over a few last notes about dictionaries

my\_dict = { "fish": ["c", "a", "r", "p"], "cash": -4483, "luck": "good" } print my\_dict["fish"][0]

1. In the example above, we created a dictionary that holds many types of values.
2. The key "fish" has a list, the key "cash" has an int, and the key "luck"has a string.
3. Finally, we print the letter "c". When we access a value in the dictionary like my\_dict["fish"], we have direct access to that value (which happens to be a list). We can access the item at index 0in the list stored by the key "fish".

You can use methods with a list stored in a dictionary as follows:

dict\_name['list\_key'].method()

For example, since 'backpack' is a key in our dictionary inventory you can delete 'dagger' from the corresponding list like this:

inventory['backpack'].remove('dagge