In Python, dictionaries are a very powerful data type. They allow you to map keys to arbitrary values. The one major restriction is that the keys must be immutable. In this example, we will discuss a program designed to implement a simple contact list that maps names to phone numbers.

Dictionary Structure

Our contact list will initially consist of a dictionary that has strings as keys, which are the person's name, and strings as values, which is the associated person's phone number. So, we could create a simple contact list as follows:

```
1  contacts = {'Scott Rixner': '1-101-555-1234',
2              'Joe Warren': '1-102-555-5678',
3              'Jane Doe': '1-103-555-9012'}
```

A dictionary literal in Python consists of a sequence of key-value pairs enclosed by curly braces (`{}`). Each key value pair consists of the key, a colon, and then a value. The key-value pairs are separated by commas.

Value Lookup

In order to find a phone number in the contact list, you simply index into the dictionary with the name you want to look up:

```
1  def lookup(contacts, name):
2      """
3      Lookup name in contacts and return phone number.
4      If name is not in contacts, return an empty string.
5      """
6      if name in contacts:
7          return contacts[name]
8      else:
9          return ""
```

Notice that we first need to check if the name is actually a key in the dictionary (line 6). If you try to index into a dictionary with a key that does not exist, you will get an error. You could also use the **`get`** method to provide a default value in the case where the key does not exist in the dictionary:

```
1  def lookup2(contacts, name):
2      """
3      Lookup name in contacts and return phone number.
4      If name is not in contacts, return an empty string.
5      """
6      return contacts.get(name, "")
```

While this is shorter and potentially easier to read, it may not be what you want if you are going to do something more complicated than just return a default value when the key is not in the dictionary.

Iteration

You likely will want to print out the contact list. To do this, we need to iterate over the dictionary. If we just want the keys, we can iterate over the dictionary directly:

```
1  def print_contacts(contacts):
2      """
3      Print the names of the contacts in our contacts list.
4      """
5      for name in contacts:
6          print(name)
```

If we, instead, want to print out both the keys and values, we can use the **`items`** method to iterate over key-value pairs:

```
1  def print_contact_list(contacts):
2      """
3      Print the names and phone numbers of the contacts in
4      our contacts list.
5      """
6      for name, number in contacts.items():
7          print(name, ":", number)
```

Depending on what you are doing, it may be easier to iterate over the keys or the key-value pairs.

Order

Python dictionaries are not ordered. Regardless of how items are inserted into the dictionary, there is no notion of the keys being ordered. So, when you print out your contact list, they will not necessarily be in alphabetical order (if they are, you just got lucky). So, if you want things in order, you will have to sort them yourselves. To do that, you might get the keys, sort them, and then iterate over the sorted keys:

```
1  def print_ordered(contacts):
2      """
3      Print the names and phone numbers of the contacts
4      in our contacts list in alphabetical order.
5      """
6      keys = contacts.keys()
7      names = sorted(keys)
8      for name in names:
9          print(name, ":", contacts[name])
```

This makes use of the **`keys`** method of dictionaries to get the keys in the dictionary on line 6. Then, those keys are sorted using the **`sorted`** function, which returns a sorted list. We can then iterate over the sorted list to print the contacts in alphabetical order. Note that this will alphabetize the strings, so it is effectively sorting by first name. If you wanted to sort by last name, you would need to first split the name strings into first and last names and then sort by last names.

Update

Python dictionaries also allow you to add new key-value pairs and update the values associated with existing keys. The method is exactly the same for both actions:

```
1  def add_contact(contacts, name, number):
2      """
3      Add a new contact (name, number) to the contacts list.
4      """
5      if name in contacts:
6          print(name, "is already in contacts list!")
7      else:
8          contacts[name] = number
9
10 def update_contact(contacts, name, newnumber):
11     """
12     Update an existing contact's number in the contacts list.
13     """
14     if name in contacts:
15         contacts[name] = newnumber
16     else:
17         print(name, "is not in contacts list!")
```

In order to add a new key-value pair to the dictionary, you index the dictionary with the key on the left of the equals sign and the value will be the result of evaluating the expression on the right of the equals sign, as shown on line 8. Notice that we first check if the key is already in the dictionary, as we only want to add contacts with the **`add_contact`** function, so we first check to make sure the key was not already in the dictionary on line 5. Note that the syntax is *exactly* the same to update the value that is associated with a particular key, as shown on line 15. Again, we only want to change a contact's number with the **`update_contact`** function, so we first check if the key is already in the dictionary on line 14.

Because the syntax is the same, we could combine these two functions if we are willing to update the contact if it is already in the dictionary or add it if it is not:

```
1  def add_or_update_contact(contacts, name, number):
2      """
3      Add contact or update it if it is already in the contacts list.
4      """
5      contacts[name] = number
```

We encourage you to work with the contacts list constructed here, call the functions to see how they work and potentially modify the code to add more features to the contact list. This will help you to better understand how Python dictionaries work.

:

Mark as completed