1 Tuples

A tuple is a new kind of object that is similar to a list but is immutable.

We create a tuple with round brackets:

```python
>>> t = ('one', 'two', 'three')
>>> t
('one', 'two', 'three')
>>> type(t)
<type 'tuple'>
```

and index it with square brackets (like a list):

```python
>>> t[0]
'one'
>>> t[2]
'three'
>>> t[:2]
('one', 'two')
```

Q. What does it mean for tuples to be immutable?

A.

2 Dictionaries

Consider the following problem domain. We need to represent years and the total North American fossil fuel CO2 emissions for those years.

Q. How could we do this?

A.

Q. How would we add entries to this data structure?

A.

Q. How would we edit the emissions value for a particular year?

A.

There is a better way: a new type of object called dictionary.
2.1 Dictionary Basics

A dictionary keeps track of associations between keys and values. Just like a regular dictionary where a key is a word and the value is the definition of that word.

This makes look-ups and some other operations very easy!

Form to define a dictionary:

```python
my_dict = {key1: value1, key2: value2, ...}
```

Form to look up a key's value:

```python
my_dict[key]  # This expression assumes key is present in the dictionary.
```

Form to add or update a key-value pair:

```python
my_dict[key] = value
```

- `my_dict` is the name of the dictionary object.
- `key` is any immutable object (string, int, tuple).
- `value` is any object.

Let's look at some dictionary examples:

```python
>>> emissions_by_year = {1799: 1, 1800: 70, 1801: 74, 1802: 82, 1902: 215630, 2002: 1733297}
>>> # Look up the emissions for the given year
>>> emissions_by_year[1801]
74
>>> # Add another year to the dictionary
>>> emissions_by_year[1950] = 734914
>>> emissions_by_year
```

Q. Why does the following expression cause an error?

```python
d[[1, 2, 3]] = 77
```

A. 

Q. Are dictionaries mutable?

A.
2.2 Using Dictionaries

We'll use `emissions_by_year` to illustrate what we can do with a dictionary:

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value/Output/Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>emissions_by_year[2009] = 1000000</code></td>
<td></td>
</tr>
<tr>
<td><code>emissions_by_year[2009] = 10</code></td>
<td></td>
</tr>
<tr>
<td><code>1950 in emissions_by_year</code></td>
<td></td>
</tr>
<tr>
<td><code>del emissions_by_year[1950]</code></td>
<td></td>
</tr>
<tr>
<td><code>len(emissions_by_year)</code></td>
<td></td>
</tr>
<tr>
<td><code>for key in emissions_by_year:</code></td>
<td></td>
</tr>
<tr>
<td>\hspace{1cm} <code>print(key)</code></td>
<td></td>
</tr>
</tbody>
</table>

Q. When we iterate through a dictionary, the keys come out in an unexpected order. Why??

A. Dictionaries are unordered. The order that the keys are traversed (when you loop through) is not specified: there is no guarantee that it will be in the order that they were added.

**Dictionary Methods:** Dictionaries have many methods that we can use. Use `dir(dict)` or browse [http://docs.python.org/](http://docs.python.org/) to see them all. Here are the most common:

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value/Output/Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>emissions_by_year.keys()</code></td>
<td></td>
</tr>
<tr>
<td><code>emissions_by_year.values()</code></td>
<td></td>
</tr>
<tr>
<td><code>emissions_by_year.items()</code></td>
<td></td>
</tr>
<tr>
<td><code>emissions_by_year.get(1802)</code></td>
<td></td>
</tr>
<tr>
<td><code>emissions_by_year.get(1805)</code></td>
<td></td>
</tr>
<tr>
<td><code>emissions_by_year.setdefault(1805, 42)</code></td>
<td></td>
</tr>
<tr>
<td><code>more_emissions = {1802: 100, 1809: 111}</code></td>
<td></td>
</tr>
<tr>
<td><code>emissions_by_year.update(more_emissions)</code></td>
<td></td>
</tr>
</tbody>
</table>
2.3 Iterating Through A Dictionary

Suppose we have the following dictionary associating phone numbers with people (for example, a caller ID program might work somewhat like this):

```python
phone = {'555-7632': 'Paul', '555-9832': 'Andrew', '555-6677': 'Dan', '555-9823': 'Michael', '555-6342': 'Cathy', '555-7343': 'Diane', '555-1234': 'Anya', '555-4321': 'Anya'}
```

Q. How can we iterate through the keys?
A.

Q. How can we iterate through the values?
A.

Q. How can we iterate through the key-value pairs?
A.

3 Let’s Practice!

Consider the dictionary that maps telephone numbers to names, such as `phone` above. Design and implement the following functions:

- `get_name(number, phone_dict)`: Given a phone number `number` that is a key in the dictionary `phone_dict`, return the corresponding name. If the number doesn’t exist, return an error message.

- `get_number(lookup_name, phone_dict)`: Given a name `lookup_name` that is a value in the dictionary `phone_dict`, return the key(s) associated with that value as a list of phone number(s).

- `invert(d)`: Return a new dictionary that has keys equal to the values of `d`, and values equal to the keys of `d`. Assume that the values in `d` are immutable. Assume that each value only occurs once.

- `invert_general(d)`: Return a new dictionary that has keys equal to the values of `d`, and values equal to the keys of `d`. Assume that the values in `d` are immutable. Assume that each value may occur more than once.
def get_name(number, phone_dict):
    """ (str, dict of {str: str}) -> str

    Return the name in phone_dict that has number as key. Return
    an error message if number does not exist.
    """

def get_number(lookup_name, phone_dict):
def invert(my_dict):

def invert_general(my_dict):