Looking up English words in the dictionary

**Sequence**: a group of things that come one after the other

**Collection**: a group of (interesting) things brought together for some purpose

Is a sequence a collection?  
Is a collection a sequence?

Properties of sequences and collections

- Collections (list, tuple, string, set, dict)
  - Find their length with `len`
  - Check an element membership in the collection with `in`
  - Are iterables (one can iterate over their elements with a loop)

- Sequences (list, tuple, string)
  - Use indices to access elements, e.g. `myList[2]`
  - Use slice operations to access subsequences, e.g. `myList[2:5]`

- Mutable (list, set, dict): can be changed through object methods.
- Immutable (tuple, string): cannot be changed.

Python sequences and collections

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>a mutable sequence of arbitrary objects</td>
<td>[-100, &quot;blue&quot;, (1, 10), True]</td>
</tr>
<tr>
<td>tuple</td>
<td>an immutable sequence of arbitrary objects</td>
<td>(2017, &quot;Mar&quot;, 2)</td>
</tr>
<tr>
<td>string</td>
<td>an immutable sequence of characters</td>
<td>&quot;Go Wellesley!&quot;</td>
</tr>
<tr>
<td>set</td>
<td>an unordered collection of distinct objects</td>
<td>{1, 4, 5, 23}</td>
</tr>
<tr>
<td>dict</td>
<td>An unordered collection of key:value pairs, where keys are immutable and values are any Python objects</td>
<td>{&quot;orange&quot;: &quot;fruit&quot;, 3: &quot;March&quot;, &quot;even&quot;: [2,4,6,8]}</td>
</tr>
</tbody>
</table>

Note: `set` will be covered in the next lecture.
Dictionaries

A Python dictionary is a mutable collection that maps keys to values.

A dictionary is enclosed with curly brackets and contains comma-separated pairs. A pair is a colon-separated key and value.

```
daysOfMonth = {'Jan': 31, 'Feb': 28, 'Mar': 31, 'Apr': 30, ...
```

```
28: ['Feb'],
```

```
keys

keys: any immutable type such as numbers, strings, or tuples.

phones = {5558671234: 'Gal Gadot',
9996541212: 'Trevor Noah',
7811234567: 'Paula A. Johnson'}
```

```
```

```
friends = {('Harry', 'Potter'):['harry@hogwarts.edu', 'Gryffindor'],
('Cho', 'Chang'):['cho@hogwarts.edu', 'Ravenclaw'],
('Cedric', 'Diggory'):['ced@hogwarts.edu', 'Hufflepuff']
```

To notice:

In daysOfMonth, the key for December is missing, it will be added later in the slides.

```
keys cannot mutate

An analogy to P.O. box keys. If the key is damaged, one cannot retrieve the content.

values

values: any Python object (numbers, strings, lists, tuples, dicts, sets, even functions)

```
student = {'name': 'Georgia Dome', 'dorm': 'Munger Hall',
'year': 2019, 'CSMajor?': True }
```

```
functionNames = {'map': [double, square, pluralize],
'filter': [isEven, isPositive, contains] }
```

```
contributions = {'uma52': {2015: 10, 2016: 15},
'rid12': {2009: 5, 2010: 18, 2012: 4} }
```

Dictionaries
How to create dictionaries?

1. Literal dictionary: provide keys and pairs together, delimited with { }
   
   ```python
   In [1]: scrabbleDict = {'a': 1, 'b': 3, 'c': 3, 'd': 2, 'e': 1, 'f': 4, 'g': 2, 'h': 4, 'i': 1, 'j': 8, 'k': 5, 'l': 1, 'm': 3, 'n': 1, 'o': 1, 'p': 3, 'q': 10, 'r': 1, 's': 1, 't': 1, 'u': 1, 'v': 4, 'w': 4, 'x': 8, 'y': 4, 'z': 10}
   ```

2. Start with an empty dict and add key/pairs
   
   ```python
   In [2]: cart = {}  # an empty dict
   In [3]: cart['oreos'] = 3.99
   In [4]: cart['kiwis'] = 2.54
   In [5]: cart
   Out[5]: {'kiwis': 2.54, 'oreos': 3.99}
   ```

3. Applying the built-in function `dict` to a list of tuples:
   
   ```python
   In [5]: dict([('DEU', 49), ('ALB', 355), ('FRA', 33)])
   Out[5]: {'ALB': 355, 'DEU': 49, 'FRA': 33}
   ```

Dictionary Operations: subscripting

The value associated with a key is accessed using the same subscripting notation with square brackets used for list indexing:

```python
In [6]: daysOfMonth['Oct']
Out[6]: 31
```

```python
In [7]: friends[('Harry', 'Potter')]
Out[7]: ['harry@hogwarts.edu', 'Gryffindor']
```

```python
In [8]: phones[5558671234]
Out[8]: 'Gal Gadot'
```

```python
In [9]: functionNames['map'][0]
Out[9]: double
```

```python
In [10]: contributions['rid12'][2010]
Out[10]: 18
```

Dictionary Operations: check with `in` before accessing

Subscripting a dictionary with an invalid key raises a `KeyError`:

```python
In [12]: daysOfMonth['October']
                  -------------------------
            KeyError
Traceback (most recent call last)
<ipython-input-4-3d32324d55ec> in <module>()
     1 ----> 1 daysOfMonth['October']
            ^
            KeyError: 'October'
```

One way to avoid such errors is to use `in` to check if a key exists

```python
In [13]: 'Oct' in daysOfMonth
Out[13]: True
```

```python
In [14]: 'October' in daysOfMonth
Out[14]: False
```

Mutability in Dictionaries

Dictionaries are mutable

- We can add or remove key-value pairs.
- We can change the value associated with a key.

```python
In [15]: daysOfMonth['Feb'] = 29  # change for leap year
In [16]: daysOfMonth['Dec'] = 31  # add new key and value
```

However, dictionary keys must be immutable

Eg: a list or a dict cannot be a key (only immutable values such as numbers, strings, and tuples)
Memory diagram for a dictionary

```python
student = {'name': 'Georgia Dome', 'dorm': 'Munger Hall',
'section': 2, 'year': 2019, 'CSMajor?': True}
```

Dictionaries are Mutable: change value for key

The value associated with a key can be changed by combining subscript and assignment notation:

```python
student['dorm'] = 'Cazenove Hall'
```

Dictionaries are Mutable: add key/value pair

A new key/value pair can be added by assigning to a key not already in the dictionary:

```python
student['psets'] = [95, 87, 73, 92]
```

Dictionaries are Mutable: remove key/value pair

A key/value pair can be removed by the `pop` method, which returns the old value in addition to removing the key/value pair:

```python
student.pop('section')
```

Notice how the “keys” are shown similarly to the indices in sequences. However, they are not ordered in any meaningful way.
Dictionaries are Mutable: 

**update**

An existing dictionary can be updated with new key/pair values through the `update` method.

In [15]: contributions


In [17]: contributions.update(newContributions)  # no output

In [18]: contributions

**Dictionary Methods: get**

The `get` method is an alternative to using subscripting to get the value associated with a key in a dictionary. It takes two arguments:

1. The key
2. A default value to use if the key is not in the dictionary

In [22]: daysOfMonth.get('Oct', 'unknown')
Out[22]: 31

In [23]: daysOfMonth.get('OCT', 'unknown')
Out[23]: 'unknown'

It is possible to use `get` without a second argument (it is optional). In this case, if the key doesn’t exist, `get` will return None. To see it, we need to print the value:

In [24]: print daysOfMonth.get('OCT')
None

**Dictionary Methods: keys, values, items**

The `keys`, `values`, and `items` method invocations on a dictionary return, respectively, lists of keys, values, and key/value pairs for a dictionary:

In [25]: daysOfMonth.keys()

In [26]: daysOfMonth.values()

In [27]: daysOfMonth.items()

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Result</th>
<th>Mutates dict?</th>
</tr>
</thead>
<tbody>
<tr>
<td>keys</td>
<td>Returns all keys as a list</td>
<td>No</td>
</tr>
<tr>
<td>values</td>
<td>Returns all values as a list</td>
<td>No</td>
</tr>
<tr>
<td>items</td>
<td>Returns a list of (key, value) pairs</td>
<td>No</td>
</tr>
<tr>
<td>get(key[, val])</td>
<td>Returns corresponding value if key in dict, else returns val.</td>
<td>No</td>
</tr>
<tr>
<td>has_key(key)</td>
<td>Returns true if key in dict, else false</td>
<td>No</td>
</tr>
<tr>
<td>pop(key)</td>
<td>Removes key+val from dict and return val.</td>
<td>Yes</td>
</tr>
<tr>
<td>update(dict1)</td>
<td>Adds new key-value pairs to the dict, replaces existing key-value pairs.</td>
<td>Yes</td>
</tr>
<tr>
<td>clear</td>
<td>Removes all items from the dict.</td>
<td>Yes</td>
</tr>
<tr>
<td>iterkeys, iteritems, itervalues</td>
<td>Similar to keys, values, items, but more efficient for big dictionaries, because use “iterators”.</td>
<td>No</td>
</tr>
</tbody>
</table>

*The notation [, val] means an optional second parameter. This can be omitted.*
Iterating over a Dictionary [1]

**Concepts in this slide:**
There are different ways to iterate over the key/values of a dictionary.

**keys, items, and values** are useful for accessing the content of dictionaries:

**In [28]:**
```python
for num in phones.keys():
    print num, phones[num]
```

5558671234 Gal Gadot
9996541212 Trevor Noah
7811234567 Paula A. Johnson

**In [29]:**
```python
for number, name in phones.items():
    print "Call {} at {}.".format(name, number)
```

Call Gal Gadot at 5558671234.
Call Trevor Noah at 9996541212.
Call Paula A. Johnson at 7811234567.

**To notice:**
The method `items` returns a list of tuples, thus, we can use tuple assignment to assign to the key and value at the same time (see In[29]).

We're using the method format to print the formatted phrases.

**Concepts in this slide:**
Avoid using the method `keys` to iterate over the keys of the dictionary.

**In [30]:**
```python
for num in phones.keys():
    print num, phones[num]
```

This behaves the same as for `num in phones.keys()` in slide 14-20. It is the recommended way to iterate over a dictionary and we will use it from now on. Don't make it a habit to use the method `keys` with a `for` loop. That method is useful in some situations, but not during iteration.

**In [31]:**
```python
for name in phones.values():
    print "Call {}!".format(name)
```

Call Gal Gadot
Call Trevor Noah
Call Paula A. Johnson

**Concepts in this slide:**
There are different ways to iterate over the key/values of a dictionary.

**Iteration: Best Practices**

The methods `keys, items, and values` always generate a list. For a big dictionary, this will occupy additional space in memory (that is, in addition to storing the dictionary, we'll have to store a separate list of keys too).

**In [32]:**
```python
type(daysOfMonth.keys())
```

Out[47]: list

In such cases, it's better to use the special methods: `iterkeys, iteritems, and itervalues`. They don't create an explicit list (as we can notice by using `type` to check for its output).

**In [33]:**
```python
type(daysOfMonth.iterkeys())
```

Out[48]: dictionary-keyiterator
Membership in Dictionaries

When looking up a key in a dictionary, always use the following idiom:

```python
if someKey in someDict
```

and never write the following:

```python
if someKey in someDict.keys()
```

A dictionary is optimized to find a key instantly, that's a reason for using dictionaries. In a list, you have to search to be able to establish membership. For a long list, that will take time, especially if you do it within a LOOP, where this search will be repeated many times.

---

**Summary**

1. A dictionary is a new Python data type that is a kind of collection. It differs from lists because it stores together pairs of keys and values. We use keys to access values.
2. Keys are always immutable (numbers, strings, and tuples), while values can be any Python object.
3. There are at least three different ways to create a dictionary, but the most common one is to start with an empty `dict` and add keys and values while iterating over some other data sequence.
4. Dictionaries are mutable, we can change the values through their keys, add new key/value pairs, and remove existing ones. Two examples of methods that mutate the dictionary are `pop` and `update`.
5. An important method that avoids encountering the `KeyError` (in case the key doesn't exist) is `get`, which can be used with one or two arguments.
6. The methods keys, values, and items return the corresponding lists, but we discourage their use when working with large dictionaries.
7. **Don't write:** `key in myDict.keys()`. Only write: `key in myDict`.

---

**Mutability with hash**

When trying to use a mutable value as key for a dictionary, we'll get an error:

```python
In [34]: daysOfMonth[{'Feb', 2015}] = 28
TypeError: unhashable type: 'list'
```

What does this error mean? It turns out, Python stores keys of a dictionary as hash values, generated by the `hash` function. This is why dictionaries are also known as hashtables, especially in other programming languages.

```python
In [35]: hash("Wellesley")
Out[35]: 1371402960993349759

In [36]: hash((2015,10))
Out[36]: 3711745792089893406

In [37]: hash(1234)
Out[37]: 1234
```

Only immutable objects have hash values. We'll get an error for mutable objects.

```python
In [38]: hash([1,2,3])
TypeError: unhashable type: 'list'
```

---

**Test your knowledge**

1. What is the main difference between data types that are sequences and those that are collections?
2. Would you need to use `range` to generate indices to access the elements of a dictionary? Explain.
3. Which has to be unique: the keys or the values of the dictionary?
4. When iterating over the values of a dictionary as in 15-22, is it possible to access the keys too? Explain. Which of the dictionaries defined in 15-6, 8 would be a good example to make your point.
5. The diagram in 15-13 shows two boxes for assigning values to the `freqDict[word]`. How can you replace the `if` statement and those two assignments by one of the dictionary methods you learned?
6. In lecture 13 we discussed how the function sorted could sort different sequence types, always returning a list. Try out if we can use sorted with a dictionary and see what happens.
7. **Challenge**: Use `sorted` and a `lambda` to sort first by days and then by name for the `daysOfMonth` dict in 15-6. The result should look like: `{'Feb', 28}, {'Apr', 30}, {'Jun', 30}, {'Nov', 30}, {'Oct', 30}, {'Sep', 30}, {'Aug', 31}, {'Dec', 31}, ...`