Lecture 8 - Dictionaries

Week 4 Wednesday

Miles Chen, PhD

Adapted from Chapter 11 of Think Python by Allen B Downey

Additional content on Dictionaries adapted from "Whirlwind Tour of Python" by Jake VanderPlas

I'm skipping ahead to chapter 11. Chapter 9 has some good exercises working with strings and loops. You can expect to see its exercises in the next homework. Chapter 10 covers lists, which I covered in a few previous lectures.

Dictionaries

A dictionary is a mapping

dictionaries (dicts) are unordered mappings of keys to values.

If you are coming from r, you can think of them as named vectors, except like lists, they can contain different types of data

The *normal* way to create dictionaries are with curly braces `{}` and colons `:`

In [1]:
```
people = {'adam':25 , 'bob': 19, 'carl': 30}
```

In [2]:
```
people
```

Out[2]:
```
{'adam': 25, 'bob': 19, 'carl': 30}
```

Dictionaries can also be created by calling dict after zipping two lists together:

In [3]:
```
people2 = dict( zip( ['adam','bob','carl'] , [25, 19, 30] ) )
```

In [4]:
```
 zip( ['adam','bob','carl'] , [25, 19, 30] )  # output of a zip function
```

Out[4]:
```
<zip at 0x1e04c6472c8>
```

In [5]:
```
people == people2
```

Out[5]:
```
True
```

You can then access the value in a dictionary by using the key.

In [6]:
```
people['bob']
```

Out[6]:
```
19
```

In [7]:
```
people.get('bob') # can also be done with method get()
```

Out[7]:
```
19
```

In [8]:
```
people['joe'] # if you ask for a key that doesn't exist you get an error
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-8-30624aee624f> in <module>
----> 1 people['joe'] # if you ask for a key that doesn't exist you get an err
or

KeyError: 'joe'
```

In [9]:
```
print(people.get('joe') ) # if you use get() and it does not find, returns None
```

```
None
```

In [10]:
```
people.get('joe', 0) # You can also specify a default value to return if the key is
not found
```

Out[10]:
```
0
```

In [11]:
```
d = {2:[20, 4, 5], 1:10}  # keys can be numeric, values can also be lists
```

In [12]:
```
d[2]
```

Out[12]:
```
[20, 4, 5]
```

In [13]:
```
d[1]
```

Out[13]:
```
10
```

Dictionaries are inherrently unordered, so you cannot use numeric indexes. If you provide a number, that number needs be a key in the dictionary.

In [14]:
```
people[0]
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-14-ea81c3364c87> in <module>
----> 1 people[0]

KeyError: 0
```

The `in` operator applies to the keys. If you want to check the existence of a value, you'll have to use the `dict.values()` view object.

```
In [15]: 'adam' in people
```

```
Out[15]: True
```

```
In [16]: 19 in people
```

```
Out[16]: False
```

```
In [17]: 19 in people.values()
```

```
Out[17]: True
```

The `in` operator uses different algorithms for lists and dictionaries. For lists, it searches the elements of the list in order. As the list gets longer, the search time gets longer in direct proportion.

Python dictionaries use a data structure called a hashtable that has a remarkable property: the in operator takes about the same amount of time no matter how many items are in the dictionary.

You can use key mapping to create new entries in the dictionary too. You can also use it to modify the value associated with a key.

In [18]:
```
people
```

Out[18]:
```
{'adam': 25, 'bob': 19, 'carl': 30}
```

In [19]:
```
people['derek'] = 33  # new entry
people['adam'] = 26   # modifies existing key-value pair
```

In [20]:
```
people
```

Out[20]:
```
{'adam': 26, 'bob': 19, 'carl': 30, 'derek': 33}
```

To remove a key, use del

In [21]:
```
del people['carl']
```

In [22]:
```
people
```

Out[22]:
```
{'adam': 26, 'bob': 19, 'derek': 33}
```

In [23]:
```
people.pop()  # pop method requires a key that exists in the dictionary
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-23-80e89430a19e> in <module>
----> 1 people.pop()  # pop method requires a key that exists in the dictionar
y

TypeError: pop expected at least 1 arguments, got 0
```

In [24]:
```
people.pop('adam')
```

Out[24]:
```
26
```

In [25]:
```
print(people)
```

```
{'bob': 19, 'derek': 33}
```

`dict.update()` can be used to add more keys from another dictionary

In [26]:
```
peopleA = {'adam':25 , 'bob': 19, 'carl': 30}
```

In [27]:
```
peopleB = {'dave':35 , 'earl': 22, 'fred': 27}
```

In [28]:
```
peopleA.update(peopleB)
```

In [29]:
```
peopleA
```

Out[29]:
```
{'adam': 25, 'bob': 19, 'carl': 30, 'dave': 35, 'earl': 22, 'fred': 27}
```

If the dictionary used to update has keys that exist in the first dictionary, the keys will be overwritten with the updated keys.

In [30]:
```
peopleA
```

Out[30]:
```
{'adam': 25, 'bob': 19, 'carl': 30, 'dave': 35, 'earl': 22, 'fred': 27}
```

In [31]:
```
peopleC = {'fred':99 , 'gary': 18}
```

In [32]:
```
peopleA.update(peopleC)
```

In [33]:
```
peopleA
```

Out[33]:
```
{'adam': 25,
 'bob': 19,
 'carl': 30,
 'dave': 35,
 'earl': 22,
 'fred': 99,
 'gary': 18}
```

Dictionary view objects

Dictionaries support dynamic view objects. This means that the values in the view objects change when the dictionary changes.

the view objects are

- `dict.keys()`
- `dict.values()`
- `dict.items()`

In [34]:
```
people = {'adam':25 , 'bob': 19, 'carl': 30}
```

In [35]:
```
people
```

Out[35]:
```
{'adam': 25, 'bob': 19, 'carl': 30}
```

In [36]:
```
names = people.keys()
ages = people.values()
```

In [37]:
```
names
```

Out[37]:
```
dict_keys(['adam', 'bob', 'carl'])
```

In [38]:
```
ages
```

Out[38]:
```
dict_values([25, 19, 30])
```

In [39]:
```
# I create a new key-value pair in the dictionary
people['ed'] = 40
```

In [40]:
```
# without redefining what names or ages are, the view object updates
names
```

Out[40]:
```
dict_keys(['adam', 'bob', 'carl', 'ed'])
```

In [41]:
```
ages
```

Out[41]:
```
dict_values([25, 19, 30, 40])
```

view objects support only a few functions: `len()` or `in`

If you need to do more, you can convert them to a list or other iterable type, but you'll lose the dynamic quality

In [42]:
```
len(ages)
```

Out[42]:
```
4
```

In [43]:
```
35 in ages
```

Out[43]:
```
False
```

In [44]:
```
age_list = list(ages)
```

In [45]:
```
age_list
```

Out[45]:
```
[25, 19, 30, 40]
```

In [46]:
```
# add a new key-value pair in the dictionary
people['frank'] = 29
```

In [47]:
```
ages # the view object is dynamic
```

Out[47]:
```
dict_values([25, 19, 30, 40, 29])
```

In [48]:
```
age_list # the list created earlier is not
```

Out[48]:
```
[25, 19, 30, 40]
```

In [49]:
```
ages[3]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-49-76eef9137dc8> in <module>
----> 1 ages[3]

TypeError: 'dict_values' object does not support indexing
```

In [50]:
```
ages['bob']
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-50-82752bbd5bc6> in <module>
----> 1 ages['bob']

TypeError: 'dict_values' object is not subscriptable
```

In [51]:
```
dic_items = people.items()
```

In [52]:
```
dic_items
```

Out[52]:
```
dict_items([('adam', 25), ('bob', 19), ('carl', 30), ('ed', 40), ('frank', 2
9)])
```

Application: Using a dictionary as a collection of counters

You are given a string and you want to count how many times each letter appears.

There are a few ways we can do this.

1. You could create 26 variables, one for each letter of the alphabet. Then you could traverse the string and, for each character, increment the corresponding counter, probably using a chained conditional.

2. You could create a list with 26 elements. Then you could convert each character to a number (using the built-in function ord), use the number as an index into the list, and increment the appropriate counter.

3. You could create a dictionary with characters as keys and counters as the corresponding values. The first time you see a character, you would add an item to the dictionary. After that you would increment the value of an existing item.

Let's use the dictionary approach:

In [53]:
```
def histogram(string):
    d = {}
    for character in string:
        if character not in d:
            d[character] = 1
        else:
            d[character] += 1   # += 1 means increment by 1
    return d
```

In [54]:
```
h = histogram("brontosaurus")
h
```

Out[54]:
```
{'b': 1, 'r': 2, 'o': 2, 'n': 1, 't': 1, 's': 2, 'a': 1, 'u': 2}
```

Iterating over a dictionary

In [55]:
```
for key in h:
    print(key, h[key])
```

```
b 1
r 2
o 2
n 1
t 1
s 2
a 1
u 2
```

It might appear like the letters are arranged in order of appearance, but this is not always the case. You cannot count on dictionary keys to be sorted in any meaningful way.

If you need them to appear in alphabetical order you can use `sorted()` on the dictionary

In [56]:
```
for key in sorted(h):
    print(key, h[key])
```

```
a 1
b 1
n 1
o 2
r 2
s 2
t 1
u 2
```

Reverse Lookup Search

Dictionaries are designed to return values when you provide the key.

If you need to find the key associated with a particular value, it's a bit harder and requires us to perform a search.

In [57]:
```
def reverse_lookup(d, v):
    for k in d:
        if d[k] == v:
            return k
    raise LookupError("Value does not appear in dictionary")
```

In [58]:
```
h
```

Out[58]:
```
{'b': 1, 'r': 2, 'o': 2, 'n': 1, 't': 1, 's': 2, 'a': 1, 'u': 2}
```

In [59]:
```
reverse_lookup(h, 2)
```

Out[59]:
```
'r'
```

In [60]:
```
reverse_lookup(h, 4)
```

```
---------------------------------------------------------------------------
LookupError                               Traceback (most recent call last)
<ipython-input-60-56d6f6d320d9> in <module>
----> 1 reverse_lookup(h, 4)

<ipython-input-57-29f8656ab757> in reverse_lookup(d, v)
      3         if d[k] == v:
      4             return k
----> 5     raise LookupError("Value does not appear in dictionary")

LookupError: Value does not appear in dictionary
```

The `raise` statement

The raise statement can be used to handle errors.

In our code we tell Python to raise a Lookup Exception with a message to the user. There are several types of exceptions that exist.

https://docs.python.org/3/library/exceptions.html (https://docs.python.org/3/library/exceptions.html)

Dictionaries and lists

Lists can appear as values in a dictionary.

For example, if you are given a dictionary that maps from letters to frequencies, you might want to invert it; that is, create a dictionary that maps from frequencies to letters. Since there might be several letters with the same frequency, each value in the inverted dictionary should be a list of letters.

In [61]:
```
def invert_dict(d):
    inverse = dict()
    for key in d:
        val = d[key]
        if val not in inverse:
            inverse[val] = [key]
        else:
            inverse[val].append(key)
    return inverse
```

In [62]:
```
h = histogram("parrot")
```

In [63]:
```
h
```

Out[63]:
```
{'p': 1, 'a': 1, 'r': 2, 'o': 1, 't': 1}
```

In [64]:
```
inverse = invert_dict(h)
```

In [65]:
```
inverse
```

Out[65]:
```
{1: ['p', 'a', 'o', 't'], 2: ['r']}
```

A few more notes about keys

Lists can be values in a dictionary, but they cannot be keys. Only immutable objects are hashable and can be keys, so mutable objects like lists are not allowed to be used as keys.

In [66]:
```
t = [1, 2, 3]
```

In [67]:
```
d = {}
```

In [68]:
```
d[t] = "won't work"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-68-f12a7a2240a2> in <module>
----> 1 d[t] = "won't work"

TypeError: unhashable type: 'list'
```

In [69]:
```
# you can even use boolean values ask keys
d = {True: "yes", False: "no", "d": 4}
```

In [70]:
```
d["d"]
```

Out[70]:
```
4
```

In [71]:
```
d[True]
```

Out[71]:
```
'yes'
```

Duplicate Keys

Python will not produce an error if you create a dictionary with duplicated keys.

However, looking up the value will no longer be predictable.

In [72]:
```
d = {"a":1, "b":10, "a":2, "b":0, "a": 3}
```

In [73]:
```
d["b"]
```

Out[73]: 0

In [74]:
```
d["a"]
```

Out[74]: 3