CSCI 303

Introduction to Data Science

4 - Python Dictionaries

Preview

In [1]:
```
# initializing a dictionary
d = {'one' : 1, 'two' : 2, 'three' : 3, 'four' : 4}

# outputting the entire dictionary
print(d)

# if 'three' (the value) is in the dictionary
if 'three' in d:
    print(d['three']) # then print the key (think index) associated with that val

# for loop to iterate through the dictionary.
# .items() gives us a key/value pair of the dictionary!
# k -> key
# v -> value
for k, v in d.items():
    print(k, 'spells', v)
```

```
{'one': 1, 'two': 2, 'three': 3, 'four': 4}
3
one spells 1
two spells 2
three spells 3
four spells 4
```

Dictionaries

- Implement the Map abstract data type
- Underlying implementation uses hashtables
- aka "associative arrays":
  - Indexed by *keys*, not integers
  - Keys must be immutable (strings, numbers, tuples of immutable types)

Basic Usage

Literals:

```
mydict =
    {<key1> : <value1>,
     <key2> : <value2>,
     ...}
```

Access/modify values via `[]`:

```
v = mydict[somekey]

mydict[somekey] = newvalue
```

In [2]:

```
# note Python allows multiple lines for things enclosed in
# delimiters like {}, [], ().
oddstuff = { 'test'   : 1234,
             7.45     : ['a', 'b', 'c'],
             (1,2,3) : (4,5,6)}

# print the value associated with the key 7.45.
print(oddstuff[7.45])

# the key 7.45 is now assocaited with 'hello'
oddstuff[7.45] = 'hello'
print(oddstuff[7.45])

# print the entire dictionary as before. Notice that 7.45 is now associated with
oddstuff
```

```
['a', 'b', 'c']
hello
```

Out[2]:

```
{'test': 1234, 7.45: 'hello', (1, 2, 3): (4, 5, 6)}
```

Keys/values don't have to be literals:

In [3]:

```
x = 1234
y = 'hello'
d = {x : y}
d
```

Out[3]:

```
{1234: 'hello'}
```

Creating Dictionaries

There are many ways to create a pre-populated dictionary:

- `{ key1 : value1, key2 : value2, ...}`
- `dict(key1 = value1, key2 = value2, ...)`
- `dict([(key1, value1), ...])`

Adding to Dictionaries

To add a key/value pair, simply assign to the key as if it were already there:

In [4]:
```
# create our empty dictionary
mydict = {}

# assign the key 'I am' to be associated with 'Groot'
mydict['I am'] = 'Groot'

# and output that dictionary
mydict
```

Out[4]:
```
{'I am': 'Groot'}
```

You can obtain an empty dictionary using `{}` or `dict()`, and build up:

In [5]:
```
# using dict() to initialize an empty dictionary
cubes = dict()

# loop through each x from [0,4] (the key) and the value will be the key cubed.
for x in range(5):
    cubes[x] = x ** 3
cubes
```

Out[5]:
```
{0: 0, 1: 1, 2: 8, 3: 27, 4: 64}
```

Removing From Dictionaries

Use `del` to remove a key/value pair from a dictionary:

In [6]:
```
# initialize our dictionary
mydict = {'one' : 1, 'two' : 2, 'three' : 3, 'four' : 4}

# and delete both 'one' and 'four' from that dictionary
del mydict['one']
del mydict['four']
mydict
```

Out[6]:
```
{'two': 2, 'three': 3}
```

Querying Dictionaries

Use the usual `in` or `not in` operators to test for the presence of keys:

In [7]:
```
# since we just deleted 'one' from our dictionary it is now not in it!
if 'one' in mydict:
    print(mydict['one'])
else:
    print('nope')
```

```
nope
```

Loops on Dictionaries

The usual `for` loop iterates on *keys* of a dictionary:

In [8]:
```
# intialize our dictionary
mydict = {'one': 1, 'two': 2, 'three': 3, 'four': 4}

# usual for loop to iterate on the keys of our dictionary
for key in mydict:
    print(key, "->", mydict[key])
```

```
one -> 1
two -> 2
three -> 3
four -> 4
```

Note the lack of ordering of keys...

Generally, if you want both keys and their values, it is better to loop on the `items` collection of the dictionary:

In [9]:
```
mydict.items()
```

Out[9]:
```
dict_items([('one', 1), ('two', 2), ('three', 3), ('four', 4)])
```

In [10]:
```
for k, v in mydict.items():
    print(k, "->", v)
```

```
one -> 1
two -> 2
three -> 3
four -> 4
```

To iterate over keys *in sorted order by keys*, sort the keys first and then iterate:

In [11]:
```
# sorted in alphabetical order so by key so our dictionary will result in the bel
for key in sorted(mydict.keys()):
    print(key, "->", mydict[key])
```

```
four -> 4
one -> 1
three -> 3
two -> 2
```

Dictionary Comprehensions

As with lists and tuples, dictionaries can be created using *comprehensions*.

```
{ <key> : <value> for <item> in <iterable> }
```

Compare:

In [12]:
```
# what we did before
cubes = dict()
for x in range(5):
    cubes[x] = x ** 3
cubes
```

Out[12]:
```
{0: 0, 1: 1, 2: 8, 3: 27, 4: 64}
```

In [13]:
```
# more efficient way to iterate through a dictionary
cubes = { x : x ** 3 for x in range(5) }
cubes
```

Out[13]:
```
{0: 0, 1: 1, 2: 8, 3: 27, 4: 64}
```

For another example, we can easily invert (an invertible) mapping:

In [14]:
```
# using .items() for a comprehension
cuberoots = { v : k for k, v in cubes.items() }
cuberoots
```

Out[14]:
```
{0: 0, 1: 1, 8: 2, 27: 3, 64: 4}
```

Zip

Sometimes we want to create dictionaries from pairs of sequences.

For example, suppose we are reading in a file that contains a sequence of points on one line, followed by some computed values on those points on a second line:

```
0 1 2 3 4 5 \n

17 109 32 4 88 15 \n
```

If we read these in one line at a time and split on spaces (we'll see how to do this later), we get two collections:

In [15]:
```
x = [0, 1, 2, 3, 4, 5]
y = [17, 109, 32, 4, 88, 15]
```

To turn these into a dictionary, we could use several approaches.

Here's what you might think of first, coming from another language:

In [16]:
```
# creates an empty dictionary
fn = {}
# iterates through that above x list
for i in range(len(x)):
    # and the respective key of our new dictionary becomes the respective value o
    # and the respective value of our new dictionary becomes the respective value
    fn[x[i]] = y[i]
fn
```

Out[16]:
```
{0: 17, 1: 109, 2: 32, 3: 4, 4: 88, 5: 15}
```

We can improve things a bit with the `enumerate` function, which returns pairs of indices and values from a

sequence:

In [17]:
```
# create an empty dictionary
fn = {}
# create our dictionary by iterating through the x list using the enumerate funct
for i, xval in enumerate(x):
    fn[xval] = y[i]
fn
```

Out[17]:
```
{0: 17, 1: 109, 2: 32, 3: 4, 4: 88, 5: 15}
```

or even better, using a comprehension:

In [18]:
```
# xval is the key of the dictionary
# each value in our y list is the value of the dictionary
# and then iterate through the x list to create our dictionary
fn = { xval : y[i] for i, xval in enumerate(x) }
fn
```

Out[18]:
```
{0: 17, 1: 109, 2: 32, 3: 4, 4: 88, 5: 15}
```

Perhaps the most "Pythonic" approach uses `zip`.

What `zip` does to pairs of iterable objects:

In [19]:
```
print(x)
print(y)

# how zip works is that the first values of the lists are paired together, then t
list(zip(x, y))
```

```
[0, 1, 2, 3, 4, 5]
[17, 109, 32, 4, 88, 15]
```

Out[19]:
```
[(0, 17), (1, 109), (2, 32), (3, 4), (4, 88), (5, 15)]
```

We can pass this directly to the `dict` constructor:

In [20]:
```
# tada! Everything we did above can be done by calling dict on our zip creation
fn = dict(zip(x,y))
fn
```

Out[20]:
```
{0: 17, 1: 109, 2: 32, 3: 4, 4: 88, 5: 15}
```

L04: Practice

1. Create a new Python 3 Notebook
2. Add a comment at the top of the cell with your name and the date
3. Solve this problem: Given a list of words, create a dictionary mapping lengths of words to alphabetically sorted lists of words of that length.
4. Download the Notebook (.ipynb) file and submit to Canvas.

E.g., if the word list is `['one', 'two', 'three', 'six']`, then the new dictionary should be `{ 3: ['one', 'six', 'two'], 5: ['three'] }`

In [21]:

```
#Name:
#Date:
#In Class Python Practice: L04

words = ['apple', 'plum', 'pear', 'peach', 'orange', 'cherry', 'quince']

d = {}

# YOUR CODE HERE


```

In [ ]:

```

```