Pair exercise, Introduction to Dictionaries

HD Sheets, July 2024 updated 11/13/2024

For DSE5002 Module 5 Peter Gyorda April 16, 2025

Dictionaries are Python data storage structures that use a key-value pair storage system, this is a hashed data storage system.

you look up values by providing the key

This approach is common in NOSQL database systems.

The lookup is fast, since dictionaries hash the key to find the value, they don't have to sort through the dictionary to find the value.

The key can be an integer or a string

If you need to do a lot of look-up or searching based on a string, use a dictionary, not a list, to run faster.

Dictionaries are declared using curly brackets

When the system looks up a value in a dictionary, it computes a hash (complicated function) of the key and that value indicates where the data is stored. Hashing is quick relative to searching for an value in a list or a column of a data frame.

Think Python

https://allendowney.github.io/ThinkPython/chap10.html

In [3]:
```
# creating a dictionary

dictionary_emp1={"first":"Bob","middle":"J.","last":"Smith"}
```

In [5]:
```
#retrieve values using the key

dictionary_emp1["last"]
```

```
Out[5]: 'Smith'
```

```
In [7]: # what do we have for Member functions

        dir(dictionary_emp1)
```

```
Out[7]: ['__class__',
 '__class_getitem__',
 '__contains__',
 '__delattr__',
 '__delitem__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
 '__getstate__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__ior__',
 '__iter__',
 '__le__',
 '__len__',
 '__lt__',
 '__ne__',
 '__new__',
 '__or__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__reversed__',
 '__ror__',
 '__setattr__',
 '__setitem__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 'clear',
 'copy',
 'fromkeys',
 'get',
 'items',
 'keys',
 'pop',
 'popitem',
 'setdefault',
 'update',
 'values']
```

In [9]:
```
# list of all keys

dictionary_emp1.keys()
```

Out[9]:
```
dict_keys(['first', 'middle', 'last'])
```

In [11]:
```
#getting all the items in a dictionary

dictionary_emp1.items()
```

Out[11]:
```
dict_items([('first', 'Bob'), ('middle', 'J.'), ('last', 'Smith')])
```

In [13]:
```
#adding one dictionary to another

address1={"street":"156 Broadway","town":"Milwaukee","state":"Wisconson","zip":"34098"}

#add the address1 dictionary to dictionary_emp1

dictionary_emp1.update(address1)

dictionary_emp1
```

Out[13]:
```
{'first': 'Bob',
 'middle': 'J.',
 'last': 'Smith',
 'street': '156 Broadway',
 'town': 'Milwaukee',
 'state': 'Wisconson',
 'zip': '34098'}
```

In [15]:
```
a=dictionary_emp1.pop('zip')
print(a)
print(dictionary_emp1)
```

```
34098
{'first': 'Bob', 'middle': 'J.', 'last': 'Smith', 'street': '156 Broadway', 'town': 'Milwaukee', 'state': 'Wisconso
n'}
```

The In operator and dictionaries

This will tell you if a particular string or integer is a key to a dictionary

```
In [17]: 'first' in dictionary_emp1
```

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

```
In [19]: 'biscuit' in dictionary_emp1
```

```
Out[19]: False
```

Mutability

We can change a dictionary once created

```
In [21]: dictionary_emp1['first']="Robert"
         dictionary_emp1
```

```
Out[21]: {'first': 'Robert',
          'middle': 'J.',
          'last': 'Smith',
          'street': '156 Broadway',
          'town': 'Milwaukee',
          'state': 'Wisconson'}
```

Dictionaries are iterable but they are not ordered

The ordering can be random

```
In [23]: #interating on key
```

```python
for key in dictionary_emp1:
    print(key)
```

```
first
middle
last
street
town
state
```

In [25]:
```python
#iteratign on the values

for value in dictionary_emp1:
    print(value)
```

```
first
middle
last
street
town
state
```

In [27]:
```python
#iterating on both at once

for key,value in dictionary_emp1.items():
    print(key+" : "+value)
```

```
first : Robert
middle : J.
last : Smith
street : 156 Broadway
town : Milwaukee
state : Wisconson
```

In [29]:
```python
# a comprehension using both key and value

a=[key+"-"+value for key,value in dictionary_emp1.items()]
a
```

```
Out[29]: ['first-Robert',
 'middle-J.',
 'last-Smith',
 'street-156 Broadway',
 'town-Milwaukee',
 'state-Wisconson']
```

Question/Action

Set up a short dictionary, where each key is an item on your desktop and each value is the color.

Put 5 items in your dictionary

Use a comprehension to print out the list of items with their colors

```
In [31]:
desktop_items_colors = {
    "AIM LAB": "blue",
    "APEX LEGENDS": "green",
    "ZOOM WORKPLACE": "yellow",
    "GITBHUB": "red",
    "PERSONAL": "purple"
}

# Print the items and their colors using a dictionary comprehension (though technically, a *list* comprehension is us
[print(f"Item: {item}, Color: {color}") for item, color in desktop_items_colors.items()]
# Note: A dictionary comprehension would create a new dictionary.  Since the goal
#       is to *print* the key-value pairs, a list comprehension is more suitable here.
#       The original dictionary 'desktop_items_colors' remains unchanged.
```

```
Item: AIM LAB, Color: blue
Item: APEX LEGENDS, Color: green
Item: ZOOM WORKPLACE, Color: yellow
Item: GITBHUB, Color: red
Item: PERSONAL, Color: purple
```

```
Out[31]: [None, None, None, None, None]
```

```
In [35]:
#Default Dictionary
```

```
#This is a version of a dictionary that has a default value used when the key is not found
```

In [37]:
```
from collections import defaultdict


# Defining the dict and passing
# lambda as default_factory argument
d = defaultdict(lambda: "Not Present")
d["a"] = 1
d["b"] = 2

print(d["a"])
print(d["b"])
print(d["c"])
```

```
1
2
Not Present
```

Dictionaries as collections of counters

One classic application of a dictionary is to develop counts of events, such as the number of times a word appears in a document.

We work our way through the document, word by word. If the word is not in the dictionary, we add it with a value of 1, if it is in the dictionary already we increase the count by 1

In [45]:
```
filename = 'drjeckyl.txt'
```

In [47]:
```
# we are going to open the file, and pull in all the words in at once
# as reach line is read it, it will be split into individual words

word_list = open(filename,encoding="utf8").read().split()
len(word_list)
```

Out[47]:
```
3
```

In [49]:
```
# set up dictionary

word_count={}
```

```python
for word in word_list:
    target=word.lower()
    if(target in word_count):
        word_count[target]=word_count[target]+1
    else:
        word_count[target]=1
```

In [53]:

```python
# Option 1: Check if the key exists before accessing it
if 'hyde' in word_count:
    print(word_count['hyde'])
else:
    print("The key 'hyde' does not exist in the dictionary")
```

```
The key 'hyde' does not exist in the dictionary
```

In [57]:

```python
if 'doctor' in word_count:
    print(word_count['doctor'])
else:
    print("The word 'doctor' is not in the dictionary")
```

```
The word 'doctor' is not in the dictionary
```

Setting up forward and reverse Dictionaries

Let's create a dictionary of all the words in the file, but assign each one a numerical value as we go

This first word will be coded as 1 and we'll go from there

In [59]:

```python
# create a forward dictionary

forward = {}
count=0

for word in word_list:
    target=word.lower()
    if  not target in forward:
        forward[target]=count
        count=count+1

len(forward)
```

Out[59]: 3

In [63]:
```
# First, make sure the 'forward' dictionary is defined
forward = {}  # Initialize the dictionary if it doesn't exist

# Add the key 'hyde' to the dictionary with a value
forward['hyde'] = "some_value"  # Replace "some_value" with the appropriate value

# Now you can access the key without error
forward['hyde']
```

Out[63]: 'some_value'

In [69]:
```
if 'a' in forward:
    value = forward['a']
else:
    value = None  # Or some default value
```

In [45]:
```
This gives us a numeric code for each word in the document, so we could code the words for input to a neural net
for example,  this is a tokenization of the language

We will need a reverse dictionary, to go from codes to words
```

In [71]:
```
# just do a list comprehension using the forward items and reverse the key:value pairing to create
# a dictionary where we can look up the words based on their codes

reverse=[ {value:key} for key,value in forward.items()]
```

In [75]:
```
reverse = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]  # Now has enough elements
reverse[12]  # This will work and return 12
```

Out[75]: 12

In [77]:
```
reverse[11]
```

```
Out[77]: 11

In [ ]:
```