# Pair exercise, Introduction to Dictionaries

HD Sheets, July 2024 updated 11/13/2024

For DSE5002

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 [23]:

```
# creating a dictionary

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

In [24]:

```
#retrieve values using the key

dictionary_emp1["last"]
```

Out[24]:

```
'Smith'
```

In [25]:

```
# what do we have for Member functions

dir(dictionary_emp1)
```

Out[25]:
```
['__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 [26]:
```python
# list of all keys

dictionary_emp1.keys()
```

Out[26]:
```
dict_keys(['first', 'middle', 'last'])
```

In [27]:
```python
#getting all the items in a dictionary
```

```
dictionary_emp1.items()
```

Out[27]:

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

In [28]:

```
#adding one dictionary to another

address1={"street":"156 Broadway","town":"Milwaukee","state":"Wisconson","zip":"340

#add the address1 dictionary to dictionary_emp1

dictionary_emp1.update(address1)

dictionary_emp1
```

Out[28]:

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

In [29]:

```
a=dictionary_emp1.pop('zip')
print(a)
print(dictionary_emp1)
```

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

# The In operator and dictionaries

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

In [30]:

```
'first' in dictionary_emp1
```

Out[30]:

```
True
```

In [31]:

```
'biscuit' in dictionary_emp1
```

Out[31]:

```
False
```

## Mutability

We can change a dictionary once created

In [32]:

```
dictionary_emp1['first']="Robert"
dictionary_emp1
```

```
Out[32]: {'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 [33]: #interating on key


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

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

```
In [34]: #iteratign on the values

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

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

```
In [35]: #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 [36]: # a comprehension using both key and value

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

```
Out[36]: ['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 [37]: desktop_dict = {}
```

```
In [38]: desktop_dict['Chrome']='yellow'
         desktop_dict['Edge']='blue'
         desktop_dict['File Explorer']='orange'
         desktop_dict['pqAdmin4']='white'
         desktop_dict['Solidworks']='red'
```

```
In [39]: b=[key+"-"+value for key,value in desktop_dict.items()]
         b
```

```
Out[39]: ['Chrome-yellow',
          'Edge-blue',
          'File Explorer-orange',
          'pqAdmin4-white',
          'Solidworks-red']
```

# Default Dictionary

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

```
In [40]: 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 [41]:

```
filename = 'data/dr_jeckyl-1.txt'
```

In [42]:

```
# 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[42]:

```
28739
```

In [43]:

```
# set up dictionary

word_count={}

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 [44]:

```
word_count['hyde']
```

Out[44]:

```
53
```

In [45]:

```
word_count['doctor']
```

Out[45]:

```
13
```

# 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 [46]:

```
# 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[46]:

```
6441
```

In [47]:

```
forward['hyde']
```

Out[47]:

```
12
```

In [48]:

```
forward['a']
```

Out[48]:

```
136
```

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 [49]:

```
# just do a list comprehension using the forward items and reverse the key:value pa
# a dictionary where we can look up the words based on their codes

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

In [50]:

```
reverse[12]
```

Out[50]:

```
{12: 'hyde'}
```

In [51]:

```
reverse[11]
```

Out[51]:

```
{11: 'mr.'}
```

In [ ]: