Python Lesson – Dictionaries

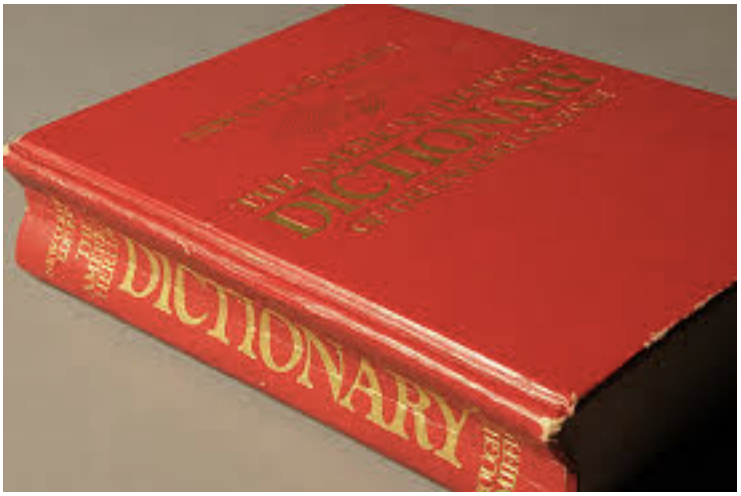

Image Source: https://encrypted-tbn3.gstatic.com/images?q=tbn:ANd9GcSVkMx0ulre7iqizJ6FHgrwy2fjXrsuORsm5R9gq-caIe3pJHXa

Lesson Description
In this lesson, we will be learning about a new data structure—the dictionary. The dictionary (also known as a hashtable or hashmap in other languages) is one of the most powerful data structures Python has available to use. Luckily, since it's built in to the Python language, we don't have to implement it ourselves. We will also be learning about tuples. We have briefly introduced them previously, but will now do so more explicitly and make use of them along with the Dictionary.

What you will learn:
- How to use a Dictionary Data Structure
- How to use Tuples

Getting Setup

In this lesson, we are going to be working in the Python Idle editor as well as the terminal. We will learn about the dictionary in the Idle editor, and then we are going to graduate to creating our own files and working with data.

Our Second Major Data Structure – The Dictionary
The list has been the focus of most of our discussion previously, but now it is time to use another powerful data structure. The Dictionary is a powerful data structure that has a 'key' and a 'value'. Each key is unique in the dictionary, and it has an associated value. The associated value however, does not need to be unique.

Examples of dictionaries in real life include:
- A phone book:
  - Key – The phone number
  - Value – The persons name
- A physical dictionary (hence where the name of this data structure comes from)
  - Key – The word
  - Value – The description of the word (i.e. the definition).
- A Student identification number at a university
  - Key – the number
  - Value – The persons name
- Your Subway Loyalty Card:
  - Key – Your card number
  - Value – Your points you've amassed!

The key in the dictionary can be a string or a number. In fact, it can be any data type! The takeaway though is that the key must be unique!

Here's a look at some sample tables of the above examples to again illustrate keys and values.

Key (Phone Number)	Value (Persons Name)
7323245	'Joe'

9822912	‘Sue’
6323421	‘Moe’

Note that the phone number is represented as an integer and each of the values as a string.

Key (Word)	Value (Definition)
‘Cat’	‘A small domesticated carnivore’
‘Python’	‘Any of several boa constrictors in the subfamily Pythoninae’
‘Byte’	‘Adjacent bits, usually eight, processed by a computer as a unit’

Note that the key is a string and the value also a string in this example.

Key (Student ID Number)	Value (Persons Name)
127323	‘Mike’
187428	‘Tomoki’
493209	‘Raoul’

Key (Reward Card Number)	Value (Points)
1209482104812	47
2098520935820	434
3248098324093	434

Note that in this example, the points might change, and dictionaries allow us to modify values. Values also can be duplicated, but remember the keys cannot (the older key will be overridden if there is a duplicate!).

Note: On Learning to Program: A dictionary is one of many data structures available to us that is built into Python. In fact, there are numerous other data structures and algorithms available to us that we always have at our disposal. It may even become intimidating or overwhelming! However, what I urge you to do is to work through this exercise (and future exercises) at least once through even if you don’t understand all of the details. Then revisit the details. The best way to learn is often to complete a project, and then when you revisit it you will have a better idea of what problem you are trying to solve, and what is important to learn and understand in intimate detail.

Dictionary Examples

Lets first create a dictionary to model a phonebook. The first thing we need to do is decide whether our key will be a name or a value. As we saw in our previous example, we can use a phone number. Phone numbers themselves are unique, so they are a good candidate for a key.

However, we can also use a name as a key, because it is typically easier for us to remember a persons name.

```
phoneBook = {'Mike': 55555555} # 'Mike' is the key
```

In this example, we can only have one friend named Mike. If we want more Mike's, we'd have to store them as Mike01, Mike02, Mike03, etc.

We can print out a specific entry from our phonebook using the following:

```
print phoneBook['Mike']
```

If we want to output the entire contents of the phonebook, we can simply use the print command.

```
# Print phoneBook
print phoneBook
```

Over time, we will want to grow and modify our dictionary, so we can add entries like the following.

```
# Add a new item
phoneBook['Michelle'] = 43255322
```

```
# Lets confirm our entry went in.
print phoneBook
```

```
# Delete Mike from our phonebook, we'll never need to call him!
del phoneBook['Mike']
```

```
# If Michelle changes her number, we can update it by accessing her
entry with her key ('Michelle') and then simply re-assigning a new
value.
phoneBook['Michelle'] = 3252352
```

```
# Confirm our changes have been made.
print phoneBook
```

```
# Sometimes we are not sure who is in our phonebook, so we have to
# iterate over all of # the keys. When we know what keys are available,
# we can then use those keys to quickly index into our phone book and
# retrieve the value (a phone number in this case).
# print keys
for x in phoneBook:
      print x

# Alternatively, if we just need the numbers, we can print out all of
the values.
# This might be a nice thing to do if you want to call everyone and
wish them a happy
# new year.
```

```
# This might be an evil thing to do if you want to call everyone and
try to scam them!
# Look out!
# print values
for x in phoneBook:
        print phoneBook[x]
```

ANOTHER LOOK AT THE DICTIONARY

The dictionary is what is known as an ‘associative data structure’. This means that a value is associated with a key. Great, that makes perfect sense! This should be intuitive, because this is often how our brain works. We generally do not think of a number and say, oh that is Mike’s number. We generally think of a name (e.g. ‘Mike’), and then recall what his number is. Our brains work very well by associating one thing with the next, so it should not be a surprise we can do something similar with computers.

THE TUPLE

The Tuple is a way in Python to group information together. It is like a list, except that we cannot modify it once we have created a tuple.

Lets go ahead and create a tuple that groups together information about an individual.

Lets create a Person Tuple that will take a str, and an int as the two types of data we want to store. The first value is a string storing a name, and the second is an integer value for how many miles they ran this month.

```
Person = ('Mike',100)
>>> Person
('Mike', 100)
```

So the tuple itself is just a collection of values held together. We can actually store tuples in a list if we want. Lets create some more tuples with names and each persons favorite number as a value in a second field.

```
>>> Mike = ('Mike',1)
>>> Willie = ('Willie',2)
>>> Tomoki = ('Tomoki', 17)
>>> Raoul = ('Raoul',14)
>>> BestFriends = [Willie,Mike,Tomoki,Raoul]
>>> BestFriends
[('Willie', 2), ('Mike', 1), ('Tomoki', 17), ('Raoul', 14)]
```

Now when the BestFriends list is output, we see that each entry (separated by a comma) is a tuple that we created.

We can then access elements in our list by using their position in the list as we have previously done. Our list will now return a tuple. And if we check the type of the entry, we see that it is indeed a tuple.

```
>>> BestFriends[0]
('Willie', 2)
>>> type(BestFriends[0])
<type 'tuple'>
```

If we want individual fields of the tuple within our list, we can then index those after we've retrieved them from our list. So the following below says, get the first entry in BestFriends with the first field of the tuple. Then we get the first entry in BestFriends with the second field of the tuple (Again, remembering we always index from 0 to get the first item in any data structure in Python).

```
>>> BestFriends[0][0]
'Willie'
>>> BestFriends[0][1]
2
```

This might take some getting use to, as we're using two levels of indirection to access the individual element we would like. With some practice however, you will be able to do this with no problem.

Lets now combine what we have learned with dictionaries and tuples. Lets make the….NBA Superstar Basketball Dictionary!

Challenge Problem

Take in the below NBA player data, and create tuples for them. Here is the format:

The first value in the tuple is the number of championships won, and the second is the number of season the player has played.

Key	Value
Bill Russell	(11,13)
Sam Jones	(10,12)
Robert Horry	(7,16)
Michael Jordan	(6,15)

Shaquille O'Neil	(4,19)
Mike Shah	(0,0)

Example:
Creating a tuple.

```
samJones = (10,12)
```

This makes the variable 'samJones' hold the tuple (10,12).

We then want to put all of these players (or rather, their tuples) into a dictionary, and then output all of the NBA players who have won at least 1 championship.

Here is an example of creating an empty dictionary, and adding our player to it.

```
# Now lets create an empty dictionary
nbaDictionary = {}

# Add a key of 'Sam Jones' and a
# value of samJones (which is a tuple)
nbaDictionary['Sam Jones'] = samJones
```

Goals we want to achieve:

- Output all NBA players in our dictionary who have won at least one championship
- Output all of the NBA players in our dictionary who have won a championship in greater in 50% of the seasons they have played.

Hints and gotchas:

- print out your dictionary after you input your entries to visually see the data.
- Can we divide by zero in mathematics?
- Use an 'and' statement to check two conditions.
    - e.g. if value > 5 and value != 0:
- When dividing two integers, we get an integer back. If we want a float (i.e. decimal number) as a result, we have to divide to floats.
    - Example:

```
>>> 7/5
1
>>> float(7)/float(5)
1.4
```

THE CORRECT OUTPUT

The correct output is below and the solution follows in the next section.

```
==== NBA Superstar Dictionary ====
{'Michael Jordan': (6, 15), 'Sam Jones': (10, 12), 'Shaquille O Neil': (4, 19)
, 'billRussell': (11, 13), 'Robert Horry': (7, 16), 'Mike Shah': (0, 0)}

Michael Jordan has won 6 championships!
Sam Jones has won 10 championships!
Shaquille O Neil has won 4 championships!
billRussell has won 11 championships!
Robert Horry has won 7 championships!

Sam Jones has a good winning percentage
billRussell has a good winning percentage
```

THE SOLUTION

```
# First we'll create a tuple that will
# hold information about that Basketball Star
# The information we will hold is the
# - Number of Championships Won
# - How many seasons they played professional Basketball
Information = (int,int,int)

# Now lets create variables for each NBA Superstars
billRussell = (11,13)
samJones = (10,12)
robertHorry = (7,16)
michaelJordan = (6,15)
shaquilleONeil = (4,19)
mikeShah = (0,0)

# Now lets create an empty dictionary
nbaDictionary = {}

# Now lets populate the dictionary with our entries
# The Key: The players name
# The Value; The players information
nbaDictionary['billRussell'] = billRussell
nbaDictionary['Michael Jordan'] = michaelJordan
nbaDictionary['Sam Jones'] = samJones
nbaDictionary['Robert Horry'] = robertHorry
nbaDictionary['Shaquille O Neil'] = shaquilleONeil
nbaDictionary['Mike Shah'] = mikeShah

# Finally, lets print out the dictionary to see that our
# entries have been added
print "==== NBA Superstar Dictionary ===="
print nbaDictionary

print
# Now lets query for nba stars in our dictionary who have won
# more than 1 championship
for x in nbaDictionary:
    # Now this is a little tricky.
    # Remember, x is our key, so it will return something
    # like 'Michael Jordan'
    # We then need to access that key, which is stored in
    # nbaDictionary, and then access the [0]th value, which
    # is our tuple, which represents championships.
    if nbaDictionary[x][0] > 1:
        print x+' has won '+str(nbaDictionary[x][0]) +' championships!'

print
# Print out the players who have a greater than 50% championship record
for x in nbaDictionary:
    # Access the tuple at championships / seasons played
    if nbaDictionary[x][1] != 0 and float(nbaDictionary[x][0]) /
float(nbaDictionary[x][1]) > 0.5 :
        print x+' has a good winning percentage'
```

If you want to take this problem even further you can add the following items.

GOING FURTHER

- Create your own query on the dictionary.
  - Add all of the players to a list who have won championships in at least half of the seasons that they have played.
  - Make a function that adds a players name and value into the dictionary (this will save many lines of code).
    - Make up some more players to add to the list.
    - Create an interactive prompt that allows you to choose 5 players from the dictionary, and add them to a list, to assemble your all-star NBA team!