DSC 510

WEEK 7 DICTIONARIES, TUPLES, AND JSON

Tuple

A **tuple** is part of the standard language. This is a data structure very similar to the **list** data structure. The main difference being that tuple manipulation are faster than list because tuples are immutable which means that it can not be modified.

Like a list, a tuple is typically created in an assignment statement:

<tupleVariable> = (<element1>, <element2>, ... <elementN>)

An example of creating a Tuple of friends is shown below:

```
>>> friendsTuple = ('Joe', 'Martha', 'John', 'Susan')
>>> print(friendsTuple)
('Joe', 'Martha', 'John', 'Susan')
>>>
```

Tuple vs List

Notice the syntax for a Tuple is slightly different from that of a list.

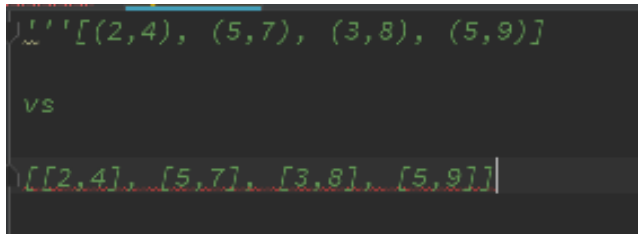
```
list_num = [1,2,3,4]
tup_num = (1,2,3,4)

print(list_num)
print(tup_num)
```

Output

```
[1, 2, 3, 4]
(1, 2, 3, 4)
```

Why Use Tuples or Lists

- Tuples take up less memory than lists.
- Tuples can be modified whereas lists can not (they're immutable).
- Lists have more built in methods than tuples. Depending on what you're trying to accomplish this might be an issue.
- Using a tuple instead of a list can give the programmer and the interpreter a hint that the data should not be changed.
- Tuples are commonly used as the equivalent of a dictionary without keys to store data.
- Lists of tuples are easier to read than lists of lists.



Dictionary

A dictionary is similar to a list in that it allows you to refer to a collection of data by a single variable name. However, it differs from a list in one fundamental way. In a list, order is important and the order of the elements in a list never changes (unless you explicitly do so). Because the order of elements in a list is important, you refer to each element in a list using its index (its position within the list).

In a dictionary, the data is represented in what are called ***key/value*** pairs. The keys within a dictionary must be unique. The syntax of a dictionary looks like this:

```
{<key>:<value>, <key>:<value>, ..., <key>:<value>}
```

Dictionaries are indexed by *keys*, which can be any immutable type; strings and numbers can always be keys. Tuples can be used as keys if they contain only strings, numbers, or tuples; if a tuple contains any mutable object either directly or indirectly, it cannot be used as a key. You can't use lists as keys, since lists can be modified in place

MLB Team Dictionary

```
MLB_team = {
    'Colorado' : 'Rockies',
    'Boston'   : 'Red Sox',
    'Minnesota': 'Twins',
    'Milwaukee': 'Brewers',
    'Seattle'  : 'Mariners',
    'St Louis': 'Cardinals',
    'New York': 'Mets',
    'Kansas City': 'Royals',
}
```

Dictionary Usage

```
#Create a new dictionary called “telephone”
>>> telephone = {'mike': '402-555-1212', 'barb': '314-231-8973', 'andy': '515-643-9087'}

#adds a new element to the dictionary
>>> telephone['kelly'] = '402-330-9870’

#Print the dictionary
>>> telephone
{'kelly': '402-330-9870', 'mike': '402-555-1212', 'barb': '314-231-8973', 'andy': '515-643-9087'}
>>> print(telephone)
{'kelly': '402-330-9870', 'mike': '402-555-1212', 'barb': '314-231-8973', 'andy': '515-643-9087'}

#Access the value of the "mike" key’
>>> telephone['mike']
402-555-1212’

#Assign a new value for an existing key in a dictionary
>>> telephone[’mike'] = ’402-396-9078’ #change value of an existing key
```

Find Dictionary Keys and Values

There are two additional operations (functions) that you can use on a dictionary. If you want, you can find all the keys defined in a dictionary with a call to <dictionary>.keys(). You can find all the values with a call to <dictionary>.values(). Both calls return a list. Here is an example using our previously defined dictionary:

```
>>> print(telephone.keys())
dict_keys(['kelly', 'mike', 'barb', 'andy'])

>>> print(telephone.values())
dict_values(['402-330-9870', '402-555-1212', '314-231-8973', '515-643-9087'])
```

In Keyword

To ensure that we are using a valid key, we can use the in operator before attempting to use a key in a dictionary. The in operator is used like this:

```
<key> in <dictionary>
```

Example:

```
'mike' in telephone
```

You can also use a similar syntax to see if a value is in the keys or the values of a dictionary.

```
>>> 'mike' in telephone.keys()
True
>>> 'mike' in telephone.values()
False
>>>
```

Checking Values with “If” Statements

In any code where we think that a key might not be found, it's a good idea to add some defensive coding to check and ensure that the key is in the dictionary before we attempt to use it on the dictionary. Typically, we build this type of check using an if statement:

```
if myKey in myDict:
    # OK, we can now successfully use myDict[myKey]
else:
    # The key was not found, print some error message or take some other action
```

Sometimes, it may make more logical sense to code the reverse test. We can use not in to test for the key not being in the dictionary:

```
if myKey not in myDict:
    # The key was not found, do whatever you need to do
```

Iterate Through a Dictionary

There are several ways to iterate through a dictionary using a for loop. In the example below we use the for loop to iterate through the keys in the dictionary.

```
>>> statesDict = {\ 'California':38802000, 'Texas':26956000, 'Florida':19893000,
 'New York':19746000,\ 'Illinois': 12880000, 'Pennsylvania': 12787000,
 'Ohio':11594000, 'Georgia': 10097000,\ 'North Carolina': 9943964,
 'Michigan':9909000, 'New Jersey': 8938000}

>>> for state in statesDict:
          population = statesDict[state]
          print state, population
```

Iterate through Dictionaries using items()

When looping through dictionaries, the key and corresponding value can also be retrieved at the same time using the items() method. Python automatically assigns the first variable as the name of a key in that dictionary, and the second variable as the corresponding value for that key.

```
>>> knights = {'gallahad': 'the pure', 'robin': 'the brave'}
>>> for key, value in knights.items():
    print("{}: {}".format(key, val))

gallahad: the pure
Robin: the brave
```

Create Dictionary from List Example

```
# Count words in a string program

#Create a string which stores the phrase
speech = "to be or not to be"
#Split the speech string into a list where each word is an item in the list
speech_list = speech.split()
#Create a dictionary called word_count_dict
word_count_dict = {}

#We are going to use the following logic to create the list
#We are going to use a for loop to iterate through the list.
#If the word (from the list) doesn't exist in the dictionary we are going to create a key value pair for the word
#The Key value pair will have the word (from the list) as the key and the number of times the word appears in the list as the value
#If the word does exist we will create a key in the dictionary and increment the value by 1.

#iterate through each word in the list
for word in speech_list:
    # if the word is in the dictionary then we need to increment the value associated with the key (i.e. the word from the string)
    if word in word_count_dict:
        word_count_dict[word] += 1
    # if the word doesn't exist in the dictionary then we need to add 1 to the value associated with the key
    else:
        word_count_dict[word] = 1
#Print out dictionary
print(word_count_dict)
```

Dictionary from List Output

- When we run the previous example we will get a dictionary. The dictionary contains key value pairs with the words from the list as a key and the number of times the word appears in the list as the value. Original list is “To Be Or Not To Be”

```
{'to': 2, 'be': 2, 'or': 1, 'not': 1}
```

List of Dictionaries

It is common to combine lists and dictionaries. In the example below we have a list of cars called carsList. Each car dictionary has a make, model, year, doors, and mileage key with an associated value. Below is an example of the list as well as an iteration example. The overall list is defined by the [ ] and each dictionary within the list is defined by the { }

```
carsList = [{'make':'Toyota', 'model':'Prius', 'year': 2006, 'doors':4, 'mileage': 65000},
            {'make':'Honda', 'model':'Civic', 'year': 2010, 'doors':2, 'mileage': 54321},
            {'make':'Ford', 'model':'Fusion', 'year': 2012, 'doors':4, 'mileage': 24680},
            {'make':'Chevy', 'model':'Volt', 'year': 2015, 'doors':4, 'mileage': 7890}]

for carDict in carsList:
    if (carDict['doors'] == 4) and (carDict['mileage'] < 50000):
        print(carDict['make'], carDict['model'], carDict['license'])
```

Popular Dictionary Methods

- **items()**: all the key-value pairs as a list of tuples
- **keys()**: all the keys as a list
- **values()**: all the values as a list

JSON Data

JSON (JavaScript Object Notation) is a common format of data that is often used by applications to send data to other applications. It is common for an application that uses a 3rd party API to receive data back from that system in JSON format. The interesting thing about JSON data is that it resembles Python Dictionaries which makes it extremely easy to work with.

Python Example to obtain JSON data

In this example we are accessing an API and requesting information in return. Specifically we're asking for data regarding breaches from an open API. We'll cover these concepts more in future weeks but for now I want to show the power of Python to obtain JSON data and what that data really looks like.

```
import requests
import json

url = "https://haveibeenpwned.com/api/v2/breaches?domain=adobe.com"

payload = ""
headers = {
    'cache-control': "no-cache",
    'Postman-Token': "abd8a247-803a-45bd-9996-7c51b462c3d6"
    }

response = requests.request("GET", url, data=payload, headers=headers)
parsed = json.loads(response.text)
print(json.dumps(parsed, indent=4, sort_keys=True))
```

JSON Response

This is the data we get back from the previous Python example. Notice that this data looks a lot like a dictionary. In fact it's actually a list of a dictionaries as shown on previous slides. Notice the [] which define the list and the {} which define the dictionary. If we received more than one dictionary it would be included in one big list. This is typical for JSON responses.

https://docs.python.org/3.6/library/functions.html#input

```
C:\Users\mikee\PycharmProjects\DSC510\test.py\venv\Scripts\python.exe C:/Users/mikee/.PyCharm2018.2/config/scratches/J
[
    {
        "AddedDate": "2013-12-04T00:00:00Z",
        "BreachDate": "2013-10-04",
        "DataClasses": [
            "Email addresses",
            "Password hints",
            "Passwords",
            "Usernames"
        ],
        "Description": "In October 2013, 153 million Adobe accounts were breached with each containing an internal ID,
        "Domain": "adobe.com",
        "IsFabricated": false,
        "IsRetired": false,
        "IsSensitive": false,
        "IsSpamList": false,
        "IsVerified": true,
        "LogoPath": "https://haveibeenpwned.com/Content/Images/PwnedLogos/Adobe.png",
        "ModifiedDate": "2013-12-04T00:00:00Z",
        "Name": "Adobe",
        "PwnCount": 152445165,
        "Title": "Adobe"
    }
]

Process finished with exit code 0
```