
CMSC 201 Spring 2016

Lab 10 – Tuples and Dictionaries

Assignment: Lab 10 – Tuples and Dictionaries

Due Date: During discussion, April 18th through April 21st

Value: 10 points

Part 1: Data Types

So far this semester, we have been learning about the different data types that we can use. We have discussed numbers (both integers and floats), Booleans, strings, and lists. Interestingly, strings and lists share a lot of characteristics and functionalities, such as indexing and slicing operations. We have a name for these types of structures: they are called sequence types. In this lab we are going to practicing using two more data types that are also sequence structures: tuples and dictionaries.

Part 2: Tuple Creation

Tuples look a lot like lists. They contain information in a “list” form and they act pretty similarly. The two main differences between tuples and lists are:

1. The tuples cannot be changed (they are immutable), unlike lists
2. Tuples use parentheses, whereas lists use square brackets

So when should you use a tuple and when should you use a list? Generally, we use a tuple when the number of items in the structure is known in advance, and won't ever need to be changed. Tuples are faster than lists, so if we won't be changing the values in the structure, we should use a tuple.

Creating a tuple is as simple as assigning comma-separated values to a variable between parentheses. Technically, the parentheses are optional – however, for clarity, we will always include them.

Here are some examples of tuple creation:

```
tup1 = ('physics', 'chemistry', 1997, 2000)
tup2 = (1, 2, 3, 4, 5)
tup3 = ("a", "b", "c", "d")
```

Part 3: Updating and Deleting Tuples

As we have discussed, tuples are immutable, which means they cannot be changed (*i.e.*, updated in place). For this reason, we cannot directly add or remove individual items of a tuple after it has been created. However, we can use pieces of existing tuples to create new ones. For example:

```
tup1 = ("CMSC", "201")
tup2 = ("Rules", "Hooray")

tup3 = tup1 + tup2
```

The `tup3` variable now contains the following information:

```
('CMSC', '201', 'Rules', 'Hooray')
```

Although we cannot change a tuple, it is possible to completely delete one. To do this, we use the function `del`, which is just short for “delete.” To remove the variable `tup3`, we would use the following command:

```
del (tup3)
```

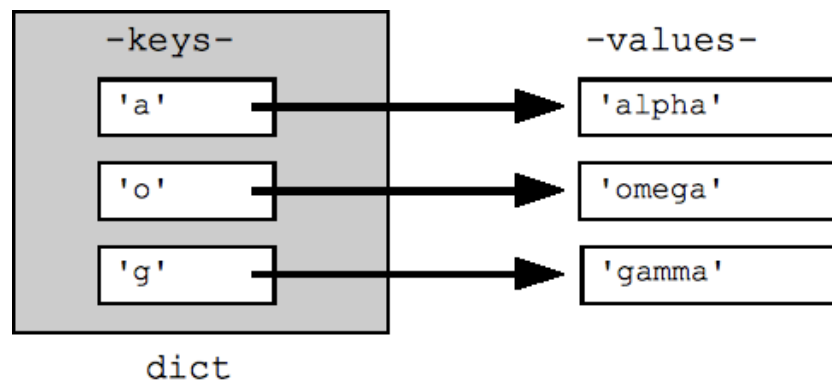
After using the `del` function, the variable is completely gone! If we try to reference it, Python will tell us that there is a “NameError” and that the variable named “`tup3`” is not defined.

As tuples are similar to strings, almost all the functions we use on strings are usable on tuples including:

- `len()` length
- `+` concatenation
- `*` repetition
- `3 in (1, 2, 3)` membership
- `for x in tupleName:` iteration

Part 4: Dictionaries

Another useful data type built into Python is the **dictionary**. Dictionaries are sometimes found in other languages as “associative memories” or “associative arrays.” Dictionaries basically map a key to a value. So, in the example below, we have a dictionary that maps the key ‘a’ to the value ‘alpha’; the key ‘o’ to the value ‘omega’; and the key ‘g’ to the value ‘gamma’.



We can create this dictionary with this line of code:

```
greek = {'a': 'alpha', 'o': 'omega', 'g': 'gamma'}
```

Dictionaries may look a lot like lists, but there are a few key differences:

1. A dictionary uses curly braces instead of square brackets
2. A dictionary is made up of (key, value) pairs
3. The key and value are separated by a colon (:)
4. The (key, value) pairs are separated by a comma (,)
5. The keys must be unique (just like the indexes of a list are unique)

Lists are indexed by **order**, which we see as a range of numbers. Dictionaries are indexed by **association**, or their key values. Keys can be any immutable type, and every key in a dictionary must be unique. Strings, floats, and integers are common choices for a key; tuples can also be used as a key.

Part 5: Dictionary Functions

We can start by looking at how we could create a simple dictionary. Let's create a new dictionary called `example`.

```
example = {'fname' : 'Santa', 'lname' : 'Claus',
           'occupation' : 'Salesman'}
```

In this dictionary, we have mapped `fname` to `Santa`, `lname` to `Claus`, and `occupation` to `Salesman`. Using this dictionary, we can perform a number of operations.

A. **Returning** something from the dictionary:

```
print(example['fname'], example['lname'])
```

B. **Adding** something to the dictionary:

```
example['salary'] = 1250595
```

C. **Updating** something in the dictionary:

```
example['occupation'] = 'Pilot'
```

D. **Deleting** something from the dictionary:

We can remove a single entry by referencing the key itself:

```
del example['occupation']
```

We can remove all entries in the dictionary by using `clear`

```
example.clear()
```

We can delete the entire dictionary by using `del`

```
del example
```

E. **Checking if a key is present** in the dictionary:

```
'occupation' in example
```

(This will return a Boolean (True or False) that indicates whether the key is in the dictionary.)

- F. Dictionaries also have methods that enable some additional functionality. In addition to the commands and examples above, here are some of the more helpful methods we can use:
- a. **dict.items()**
 - i. Returns a list of `dict`'s (key, value) tuple pairs
 - b. **dict.values()**
 - i. Returns a list of dictionary `dict`'s values
 - c. **dict.keys()**
 - i. Returns a list of dictionary `dict`'s keys

Part 6: State Abbreviator

After logging into GL, navigate to the `Labs` folder inside your `201` folder. Create a folder there called `lab10`, and go inside the newly created `lab10` directory.

```
linux2[1]% cd 201
linux2[2]% cd Labs
linux2[3]% pwd
/afs/umbc.edu/users/k/k/k38/home/201/Labs
linux2[4]% mkdir lab10
linux2[5]% cd lab10
linux2[6]% pwd
/afs/umbc.edu/users/k/k/k38/home/201/Labs/lab10
linux2[7]% █
```

To open the file for editing, type
`emacs lab10.py`
and hit enter.

The first thing you should do in your new file is create and fill out the comment header block at the top of your file. Here is a template:

```
# File:          lab10.py
# Author:        YOUR NAME
# Date:          TODAY'S DATE
# Section:       YOUR SECTION NUMBER
# E-mail:        USERNAME@umbc.edu
# Description:   YOUR DESCRIPTION GOES HERE AND HERE
#               YOUR DESCRIPTION CONTINUED SOME MORE
```

For Lab 10, you will be implementing an application that abbreviates a state name. First things first, download the file containing the states and their abbreviations by running this command inside your lab10 folder:

```
cp /afs/umbc.edu/users/k/k/k38/pub/cs201/states.txt .
```

The file contains the states and their respective abbreviations.

In order to complete your lab, you will need to implement a program that does the following tasks:

1. Read in the `states.txt` file (you can hardcode the filename). (*HINT: You will need to take a look at how the file is formatted to be able to extract the parts you need!*)
2. Write a function to store the data from the file in a dictionary, where the state names are the key, and the abbreviation is the value. (*HINT: Your function should return the dictionary. The `split()` function should prove useful in getting the data out of the file.*)
3. Ask the user to input a state – check if the word appears in the dictionary’s keys. (*HINT: Use the `in` keyword discussed earlier.*)
 - a. If the word doesn’t appear in the dictionary, tell the user that.
 - b. If the word does appear in the dictionary, return the translation.
4. Allow the user to keep looking up words as long as they like; if they type “`exit`”, the program should finish.
5. If the user types “list” it lists each of the state names (*HINT: Use the `keys()` method to list the keys*)

You can find a sample run of the program on the next page.

Here is a sample run of the program, with the user input in blue:

```
-bash-4.1$ python lab10.py
Welcome to the State Abbreviator
Please enter the state to abbreviate (list to get list and
exit to exit): Pennsylvania
The abbreviation of the state: Pennsylvania is PA

Please enter the state to abbreviate (list to get list and
exit to exit): Maryland
The abbreviation of the state: Maryland is MD

Please enter the state to abbreviate (list to get list and
exit to exit): maryland
Sorry. That is not a state.
Please enter the state to abbreviate (list to get list and
exit to exit): exit
Thank you for using the state abbreviator!
```

HINT: If you are having a tough time figuring out where to start, check out this code framework. This is only one way of writing the code for this lab.

```
# convertToDict takes in the file and converts to a dict
# Input: a file object
# Output: a dictionary containing the file contents
def convertToDict(fileContents):
    # function definition goes here

def main():
    states = open("states.txt")
    # A function call to convertToDict goes here
    print("Welcome to the State Abbreviator")
    word = ""
    # This should be a while loop for input
    # The while loop needs to check for exit and list,
    # otherwise it prints the state and it's abbreviation
    # The rest of your code goes inside the loop
    print("Thank you for using the state abbreviator!")
main()
```

Part 7: Completing Your Lab

To test your program, first enable Python 3, then run `lab10.py`. Try asking it to abbreviate different states than the one shown in the sample run shown above.

Since this is an in-person lab, you do not need to use the `submit` command to complete your lab. Instead, raise your hand to let your TA know that you are finished.

They will come over and check your work – they may ask you to run your program for them, and they may also want to see your code. Once they've checked your work, they'll give you a score for the lab, and you are free to leave.

IMPORTANT: If you leave the lab without the TA checking your work, you will receive a **zero** for this week's lab. Make sure you have been given a grade before you leave!

References:

Python Software Foundation. (2015). "Dictionaries". Retrieved from <https://docs.python.org/2/tutorial/datastructures.html>

Tutorials Point (2015). "Python Dictionary". Retrieved from http://www.tutorialspoint.com/python/python_dictionary.htm

Tutorials Point (2015). "Python Tuples". Retrieved from http://www.tutorialspoint.com/python/python_tuples.htm