

## **CS506 Programming for Computing**

### **H0S02 Lists, Tuples, Dictionary, Set**

10/10/2020 Developed by Kim Nguyen

3/28/2022 Revised by Ken Ling

07/06/2023 Reviewed by Maram Elwakeel

10/04/2023 Reviewed by Maram Elwakeel

School of Technology & Computing (STC) @ City University of Seattle (CityU)



#### **Before You Start**

- Version numbers may not match with the most current version at the time of writing. If given the option to choose between the stable release (long-term support) or the most recent, please select the stable release rather than the beta-testing version.
- There might be subtle discrepancies along with the steps. Please use your best judgment while going through this cookbook-style tutorial to complete each step.
- For your working directory, use your course number. This tutorial may use a different course number as an example.
- All the steps and concepts in this tutorial are from the textbook, so if you encounter problems in this tutorial, please try to read and compare the textbook to solve the problem. If you still can't solve the problem, please feel free to contact your course TA.
- Avoid copy-pasting code from the book or the GitHub repository. Instead, type out the code yourself. Resort to copy-pasting only when you are stuck and find that things are not working as expected.

#### **Learning Outcomes**

Students will be able to:

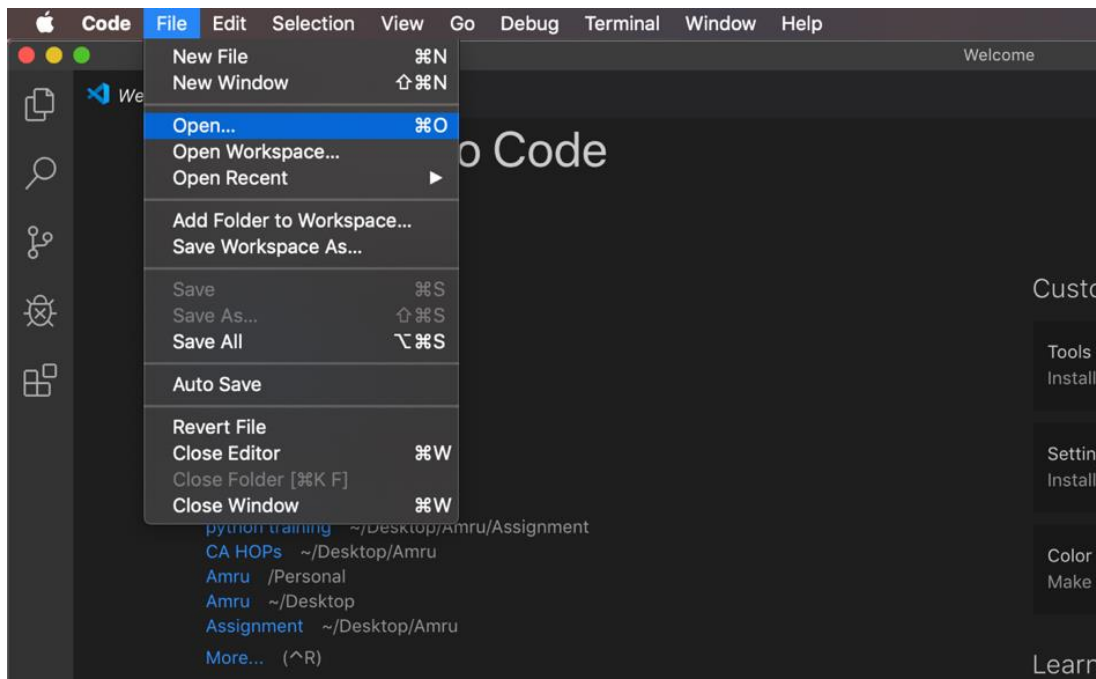
- Understand the List Data Type and Tuples in Python
- Write a Python program with functions that can be performed in Lists and Tuples

## Resources

Python crash course: a hands-on, project-based introduction to programming: Matthes, E. (2019): [Available online link](#)

## Section 1. Preparation

- 1) In Visual Studio Code, open the private repository generated when you accepted the HOS02 assignment  
(If you cannot find that repository in your machine, you might have not cloned the repo, if so, please do before proceeding).



Open the terminal from the VSCode by hitting the control + ~ key, navigate into Module 2 folder using the following command:

```
>>> cd Module 2
```

## Section 2. Lists

The list is the most versatile datatype available in Python which can be written as a list of comma-separated values (items) between square brackets. The important thing about a list is that items in a list need not be of the same type.

Creating a list is as simple as putting different comma-separated values between square brackets.

### Creating, accessing and updating values

- 1) Under Module2 create a file called **List\_basic.py** and type the following code

```
list1 = ['physics', 'chemistry', 1997, 2000]
list2 = [1, 2, 3, 4, 5, 6, 7 ]
#accessing
print ("list1[0]: ", list1[0])
print ("list2[1:5]: ", list2[1:5])
#updating
print ("Value available at index 2 : ")
print (list1[2])
list1[2] = 2001
print ("New value available at index 2 : ")
print (list1[2])
```

To access values in lists, use the square brackets for slicing along with the index or indices to obtain value available at that index. Python considers the first item in a list to be at position 0, not position 1.

- 2) Type the following in the terminal to check the output of the above code

```
>>> python3 List_basic.py
```

```
list1[0]: physics
list2[1:5]: [2, 3, 4, 5]
Value available at index 2 :
1997
New value available at index 2 :
2001
```

**Challenge 1: Explain the code above by commenting each line of code**

### Section 3. Adding elements

#### Append

The simplest way to add a new element to a list is to *append* the item to the list. When you append an item to a list, the new element is added to the end of the list. The `append()` method makes it easy to build lists dynamically.

1) In the same file **List\_basic.py** add the below code

```
print (list1[2])
#Adding
list1.append(2020)
print("New List:" ,list1)
```

2) Type the following in the terminal to check the output of the above code

```
>>> python3 List_basic.py
```

```
New List: ['physics', 'chemistry', 2001, 2000, 2020]
```

#### Section 4. Inserting elements

You can add a new element at any position in your list by using the `insert()` method. You do this by specifying the index of the new element and the value of the new item.

1) In the same file **List\_basic.py** add the below code

```
#Insert  
list1.insert(0, 'Python')  
print("After inserting: ",list1)
```

2) Type the following in the terminal to check the output of the above code

```
>>> python3 List_basic.py
```

```
After inserting: ['Python', 'physics', 'chemistry', 2001, 2000, 2020]
```

#### Section 5. Removing element

There are 3 ways to remove an element from a list.

- i) If you know the position of the item you want to remove from a list, you can use the `del` statement.
- ii) The `pop()` method removes the last item in a list, but it lets you work with that item after removing it. The term *pop* comes from thinking of a list as a stack of items and popping one item off the top of the stack. In this analogy, the top of a stack corresponds to the end of a list. You can use `pop()` to remove an item from any position in a list by including the index of the item you want to remove in parentheses.
- iii) If you only know the value of the item you want to remove, you can use the `remove()` method.

1) Under Module2 create a file **List\_remove.py** and type the following code.

```
#del
motorcycles = ['honda', 'yamaha', 'suzuki']
del motorcycles[1]
print(motorcycles)
```

2) Type the following in the terminal to check the output of the above code

```
>>> python3 List_remove.py
```

```
['honda', 'suzuki']
```

The `del` used at index 1 deletes the value Yamaha. You can no longer access the value that was removed from the list after the `del` statement is used.

3) Add the below code in the same file

```
#pop
motorcycles = ['honda', 'yamaha', 'suzuki']
popped_motorcycle = motorcycles.pop()
print(motorcycles)
print(popped_motorcycle)
first_owned = motorcycles.pop(0)
print("The first motorcycle I owned was a" , first_owned)
```

4) Type the following in the terminal to check the output of the above code

```
>>>python3 List_remove.py
```

```
['honda', 'yamaha']
suzuki
The first motorcycle I owned was a honda
```

At pop(), a value from the list and store that value in the variable popped\_motorcycle. We print the list to show that a value has been removed from the list. Then we print the popped value to prove that we still have access to the value that was removed.

The output shows that the value 'suzuki' was removed from the end of the list and is now assigned to the variable popped\_motorcycle.

5) Add the below code in the same file

```
#remove
motorcycles = ['honda', 'yamaha', 'suzuki', 'ducati']
motorcycles.remove('ducati')
print(motorcycles)
```

- 6) Type the following in the terminal to check the output of the above code

```
>>>python3 List_remove.py
```

```
['honda', 'yamaha', 'suzuki']
```

The code removes “ducati” from the list. You can use the `remove()` method to work with a value that's being removed from a list.

## Section 6. Using Loop in lists

- 1) Let's use a for loop to print out all the elements of a list. Create a file **replaceNegative.py** and enter the following code:

```
1 original = [8,20,-10,55,-777]
2
3 for i in range(len(original)):
4     print(original[i])
5
```

The `range()` function causes Python to start counting at 0 and it stops when it reaches the length of the list. The `len()` function calculates the length of the list. The `i` consists of the range index and it



is used for navigating throughout the list. If the value in the list is less than 0, that is a negative value then that value is converted to absolute value and returned to list.

Run the code:

```
>>> python3 replaceNegative.py
```

```
8
20
-10
55
-777
```

**Challenge 2: Edit the program to replace any negative numbers in the list with positive ones.**

**Expected result:**

```
[8, 20, 10, 0, 55, 777]
```

*Hint: you can use `abs()` function to turn negative numbers to positive*

## Section 7. Tuples

A tuple is a sequence of immutable Python objects. The tuple data type is like a list, except tuples are typed with parentheses and cannot be modified, appended, or removed. The following is how tuples look like.

```
tup1 = ('physics', 'chemistry', 1997, 2000);
```

- 1) Create a **tuple.py** file and define a tuple with multiple data types and try to modify the grade by assigning 4.0 value to item at index 1 which is 2.0

```
1  courses = ('CS101', 2.0, 3)
2  courses[1] = 4.0
```

Note that you will get error message like because Tuples can't be modified, **unlike** List

- 2) Type the following in the terminal to check the output of the above code

```
>>> python3 tuple.py
```

```
Traceback (most recent call last):
  File "tuple.py", line 2, in <module>
    courses[1] = 4.0
TypeError: 'tuple' object does not support item assignment
```

- 3) Create a **sortTuple.py** with the following code to sort tuples by its first value.

```
1  def first(n):
2      return n[0]
3
4
5  def sort_list_first(tuples):
6      return sorted(tuples, key=first)
7
8
9  print(sort_list_first([(5, 2), (2, 1), (4, 4), (3, 2), (1, 2)]))
```

- 4) Type the following in the terminal to check the output of the above code

```
>>> python3 sortTuple.py
```

Output will be sorted.

```
[(1, 2), (2, 1), (3, 2), (4, 4), (5, 2)]
```

“def” is the function and it will be explained in the later chapters. the sort function takes in a keyword argument called key. What key does is it provides a way to specify a function that returns what you would like your items sorted by. The function gets an "invisible" argument passed to it that represents an item in the list and returns a value that you would like to be the item's "key" for sorting.

## Section 8. Dictionary Data Type

A dictionary is a collection of many values. Unlike List, indexes for dictionaries can use many different data types, it's called keys, and a key with its associated value is called a key-value pair. Each key is separated from its value by a colon (:), the items are separated by commas and the whole thing is enclosed in curly braces. Keys are unique within a dictionary while values may not be. The values of a dictionary can be of any type, but the keys must be of an immutable data type such as strings, numbers, or tuples.

In the Dictionary functions like accessing values, updating the dictionary, adding new pairs and deleting the elements can be performed.

- 1) Under Module2 create a file **dictionary.py** and type the below code. This program describes how to access, update, add and delete in a dictionary.

```

#Accessing
dict = {'Name': 'abc', 'Age': 7 }
print ("Name : ", dict['Name'] ,"\n" , "Age :", dict['Age'])

#updating
dict['Age']= 20
print("Updated Age :", dict['Age'])

#Adding
dict['Phone_no']= 123456789
print("After adding the new pair :", dict)

#Deleting
del dict['Phone_no']
print("After deleting phone_no :", dict)

```

2) In the terminal type the following to check the output for the above code

```
>>> python3 dictionary.py
```

```

Name : abc
Age : 7
Updated Age : 20
After adding the new pair : {'Name': 'abc', 'Age': 20, 'Phone_no': 123456789}
After deleting : {'Name': 'abc', 'Age': 20}

```

*Note: If the key you ask for doesn't exist, you'll get an error. More than one entry per key not allowed. Which means no duplicate key is allowed. When duplicate keys encountered during assignment, the last assignment wins.*

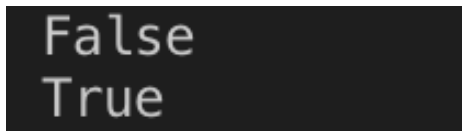
Dictionary is unordered. Thus, if you compare two dictionaries with the same content but not the same order, it will return true.

- 3) Let's find out what will happen when we compare two lists and dictionary with similar values but different order. Create a file **compare.py** and type the code below.

```
# compare two list
first = ['cats', 'dogs', 55]
second = ['dogs', 55, 'cats']
print(first==second)

# compare two dictionary, order doesnot matter
first_dict = {'name': 'aaa', 'species': 'human', 'age': 20}
second_dict = {'species': 'human', 'age': 20, 'name': 'aaa'}
print(first_dict==second_dict)
```

- 4) In the terminal type **python3 compare.py** to see the output for the above code.



```
False
True
```

The output on the terminal should be **False** and **True** respectively, since order matters in List but not in Dictionary as long as it contains the same values.

## Section 9. Looping in Dictionary

A single Python dictionary can contain just a few key-value pairs or millions of pairs. Python lets you loop through a dictionary. You can loop through all of a dictionary's key-value pairs, through its keys, or through its values.

- 1) Create a file **dict\_for.py** and type the code below. This program explains how multiple dictionaries are combined and used inside loop for accessing the key and value.

```
alien_0 = {'color': 'green', 'points': 5}
alien_1 = {'color': 'yellow', 'points': 10}

aliens = [alien_0,alien_1]

#Accessing key,value
for i in aliens:
    for key,value in i.items():
        print("KEY: ", key, "\t" , "VALUE :", value)
```

Type **python3 dict\_for.py** in the terminal for checking the output.

```
KEY:  color      VALUE : green
KEY:  points     VALUE : 5
KEY:  color      VALUE : yellow
KEY:  points     VALUE : 10
```

Note: For accessing only keys, use `.keys()` and for values use `.values()` in the loop.

- 2)Let's create a program to find the richest man from the dictionary and save it as **richest.py**

```

income = {'Alice': 90000,
          'Bob': 100000,
          'Jeff': 200000,
          'Apiwat': 999998,
          'Stark': 999999}

highest = max(income, key=income.get)
print("The richest man on earth:", end=' ')
print(highest + ' with $' + str(income[highest]))

```

Type “python3 dict\_for.py” in the terminal. The result would show Stark is the richest man on earth.

```
The richest man on earth: Stark with $999999
```

**Challenge 3: Add to the code above to show the person with the lowest income.**

## Section 10. Set

A set is a collection which is unordered and unindexed. In Python sets are written with curly brackets.

Example:

```
thisset = {"apple", "banana", "cherry"}
```

**Accessing items:** You cannot access items in a set by referring to an index, since sets are unordered, the items have no index. But you can loop through the set items using a for loop, or ask if a specified value is present in a set, by using the in keyword.

**Modify items:** Once a set is created, you cannot change its items, but you can add new items.

**Add items:** To add one item to a set use the `add()` method. To add more than one item to a set use the `update()` method.

**Remove items:** To remove an item in a set, use the `remove()`, or the `discard()` method.

**Joining two sets:** There are several ways to join two or more sets in Python. You can use the `union()` method that returns a new set containing all items from both sets, or the `update()` method that inserts all the items from one set into another.

**Constructor:** It is also possible to use the `set()` constructor to make a set.

- 1) Create a file **Set.py** and type the following program for understanding how joining of two sets work and constructor.

```
# joining two sets
set1 = {"a", "b", "c"}
set2 = {1,2,3}
set3 = set1.union(set2)
print(set3)
set1.update(set2)
print(set1)

#constructor
thisset = set(("apple", "banana", "cherry")) # note the double round-brackets
print(thisset)
```

Type **python3 Set.py** in the terminal for the output.

```
{1, 'a', 2, 'b', 3, 'c'}
{1, 'a', 2, 'b', 3, 'c'}
{'apple', 'cherry', 'banana'}
```



*Note: Both union() and update() will exclude any duplicate items.*

## **Push your work to GitHub**

Push your work to GitHub Follow the instructions on the CityU STC TA Center Github.io  
[Submit your work page](#).