TASK

Beginner Data Structures — Lists and Dictionaries

Visit our website

Introduction

Welcome to The Beginner Data Structures — Lists and Dictionaries Task!

This Task aims to ensure that you have a concrete understanding of strings and list manipulation and also to give you a little introduction to dictionaries. In **example.py**, you will see examples that deal with operations that can be applied to elements in lists as well as dictionaries. This Task also touches on functions and how they can be used to compute certain values on list elements as well as dictionaries (otherwise known as hash maps).

Get in touch

Connect for support

Remember that with our courses, you're not alone! You can contact your mentor to get support on any aspect of your course.

The best way to get help is to login to **www.hyperiondev.com/portal** to start a chat with your mentor. You can also schedule a call or get support via email.

Your mentor is happy to offer you support that is tailored to your individual career or education needs. Do not hesitate to ask a question or for additional support!

Python List Methods

There are many useful built-in list methods available for you to use. We have already looked at the *append()* method.

Some other List methods can be found below:

● *extend()*	- Adds all elements of a list to the another list
● *insert()*	- Inserts an item at the defined index
● *remove()*	- Removes an item from the list
● *pop()*	- Removes and returns an element at the given index
● *index()*	- Returns the index of the first matched item
● *count()*	- Returns the count of number of items passed as an argument
● *sort()*	- Sorts items in a list in ascending order
● *reverse()*	- Reverses the order of items in the list

Copying Lists

There are several ways to make a copy of a list. For example, you could use the **slice operator**. The slice operator always creates a new list by making a copy of a portion of another list. Slice a whole list to make a copy of that list. See below for an example of this :

```
>>> a = [1, 2, 3]
>>> b = a[:]
>>> b[1] = 10
>>> print a
[1, 2, 3]
>>> print b
[1, 10, 3]
```

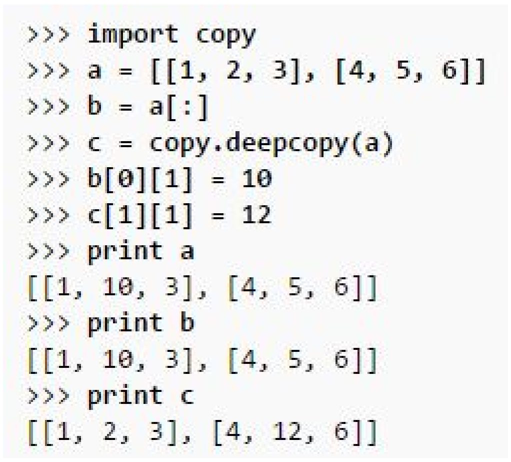
Taking the slice **[:]** creates a new copy of the list. However, it only copies the outer list. Any sublist inside is still a reference to the sublist in the original list. This is called a *shallow copy*.

Alternatively, you could use the **copy() method** of the *copy* module. Using the *copy()* method ensures that if you modify the copied list (list B), the original list (list A) remains the same. However, if list A contains other lists as items, those inner lists can still be modified if the corresponding inner lists in list B are modified. The *copy.copy()* method makes a shallow copy in the same way that slicing a list does.

However, the copy module also contains a function called *deepcopy()*. This makes a copy of the list and any lists contained in it.

To use the *deepcopy()* and *copy()* methods you must import the *copy* module.

You use the **deepcopy** function of the *copy* module, as shown below:



```
>>> import copy
>>> a = [[1, 2, 3], [4, 5, 6]]
>>> b = a[:]
>>> c = copy.deepcopy(a)
>>> b[0][1] = 10
>>> c[1][1] = 12
>>> print a
[[1, 10, 3], [4, 5, 6]]
>>> print b
[[1, 10, 3], [4, 5, 6]]
>>> print c
[[1, 2, 3], [4, 12, 6]]
```

For more information about deepcopy, see **here**.

In summary, the two main methods for copying a list are using the *slice* operator or using the *copy* module. Using the *copy* module allows one to make use of the *deepcopy()* method, which is the best method to use if a list contains other lists.

List Comprehension

List comprehension can be used to construct lists elegantly and concisely. It is a powerful tool that will apply some operation to every element in a list and then put the element into a new list. List comprehension consists of an expression followed by a *for statement* inside square brackets.

For Example:

```
num_list = ['1', '5', '8', '14', '25', '31']
new_num_list_ints = [int(element) for element in num_list]
```

For each element in *num_list*, we are casting it to an Integer and putting it into a new list called *new_num_list_ints*.

Dictionaries

Dictionaries are used to store data and are very similar to lists. However, lists are ordered sets of elements, whereas dictionaries are unordered sets. Also, elements in dictionaries are accessed via *keys* and not via their index positions the way lists are. When the key is known, you can use it to retrieve the value associated with it.

Creating a Dictionary

To create a dictionary, place the items inside curly braces ({ }) and separate them by commas (,). An item has a *key* and a *value*, which is expressed as a pair (key: value). Items in a dictionary can have a value of any data type. However, the *key* must be either a string or number and must be unique.

For example:

```
int_key_dict = {1: 'apple',
                2: 'banana',
                3: 'orange'
                }
```

Accessing Elements from a Dictionary

While you might use indexing to access elements in a list, dictionaries use keys. Keys can be used to access values either by placing them inside square brackets ([]), such as with indices in lists, or with the *get()* method. However, if you use the *get()* method, it will return 'None' instead of 'KeyError', if the key is not found.

For Example:

```
profile_dict = {'name': 'Chris',
                'surname': 'Smith',
                'age': 28,
                'cell': '083 233 3242'
               }

print (profile_dict['surname'])    # prints out 'Smith'
print (profile_dict.get('cell'))   # prints out '083 233 3242'
```

Changing Elements in a Dictionary

We can add new items or change items using the assignment operator (=). If there is already a key present, the value gets updated. Otherwise, if there is no key, a new key: value pair is added.

Dictionary Membership Test

You can test if a key is in a dictionary by using the keyword *in*. Enter the key you want to test for membership, followed by the *in* keyword and, lastly, the name of the dictionary. This will return either *True* or *False*, depending on whether the dictionary contains the key or not. The membership test is for keys only, not for values.

```
doubles = { 1: 2,
            2: 4,
            3: 6,
            4: 8,
            5: 10
          }

print(1 in doubles)
# prints out True
```

Instructions

First, read ‘example.py’. Open it using IDLE (Right-click the file and select ‘Edit with IDLE’).

- **example.py** should help you understand some simple Python. Every task will have example code to help you get started. Make sure you read all of **example.py** and try your best to understand.

- You may run **example.py** to see the output. Feel free to write and run your own example code before doing the Task to become more comfortable with Python.

Compulsory Task 1

Follow these steps:

- Create a new Python file in this folder called **cafe.py**.
- Create a list called menu, which should contain at least 4 items in the cafe.
- Next, create a dictionary called stock, which should contain the stock value for each item on your menu.
- Create another dictionary called price, which should contain the prices for each item on your menu.
- Next, calculate the total stock worth in the cafe. You will need to remember to loop through the appropriate dictionaries and lists to do this.
- Finally, print out the result of your calculation.

Compulsory Task 2

Follow these steps:

- Create a new Python file in this folder called **hash.py**

- Create a dictionary called *countryMap*, where the KEYS are the name of a country (i.e. a string), and the VALUE for each key is the name of that country's capital city.
    - For example:

```
countryMap = {
   'UnitedKingdom': 'London',
   'Sweden': 'Stockholm',
   'Canada': 'Ottawa',
}
```

- What does *print(countryMap['Sweden'])* return?

Compulsory Task 3

Follow these steps:

- Write a Python program called **counting.py** to count the number of times a character occurs (character frequency) in a string.

- Store each letter followed by the number of occurrences in a list and print it out.

- Sample String : 'google.com'

- Expected Result : {'o': 3, 'g': 2, '.': 1, 'e': 1, 'l': 1, 'm': 1, 'c': 1}

Optional Bonus Task

Follow these steps:

- Create a new Python file in this folder called **optional_task.py**
- Write a program that will give you the meaning of a given abbreviation.
- Create a dictionary that contains some abbreviations and their meanings.
- Let the abbreviation be the key and the meaning of the abbreviation be the value (e.g. ADSL: Asymmetric Digital Subscriber Line).

- Make sure that your dictionary has at least 4 abbreviations and their meanings. If you need ideas on some abbreviations, go to https://blog.hyperiondev.com/index.php/2017/11/17/8-software-development-acronyms-you-need-to-know/
- After you have created your dictionary, add 2 more abbreviations and their meanings to it.
- Now ask the user to enter an abbreviation and check if that abbreviation is in your dictionary.
- If it is, print out the abbreviation and its meaning.
- If it is not in the dictionary, print out "Abbreviation not found"

Rate us

Share your thoughts

Hyperion strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think the content of this task, or this course as a whole, can be improved or think we've done a good job?

Click here to share your thoughts anonymously.