Open labs assignment

Python programming

Question 1

What is pop in Python?

Python list pop() is an inbuilt function in Python **that removes and returns the last value from the List or the given index value**. ... index (optional) – The value at index is popped out and removed. If the index is not given, then the last element is popped out and removed.

Question 2

## **Ten list method and how use it**



# **What is a list and what is a list method?**

In Python, a list is a collection of pieces of data. A list is surrounded by square brackets [ ] with each item separated by a comma ( , ), and can contain anywhere from zero to infinity items (or however many your computer will allow). Strings, numbers, booleans, even other lists can be items in a list. Lists are ordered and mutable (changeable), meaning each item is assigned to a specific index and can be sorted, and has the ability to be altered. Once you have a list, you are able to manipulate that list using what’s known as a ‘method.’ Like string methods, to use a list method, you simply write the list followed by .[method](). For example, to run the append() method on the list [‘pepperoni’, ‘sausage’, ‘mushroom’], you would just write [‘pepperoni’, ‘sausage’, ‘mushroom’].append(‘onion’); if that list were set to a variable, you would do that variable.append(‘onion’). As you see with the append() example, some methods accept what are known as ‘arguments,’ which go in the parenthesis and further define what the method will do. The Python language has a lot of built-in methods, like append(), that allow you to easily alter lists.

## **append() and extend()**

The **append()** method allows you to add another item to the end of your list. The method takes one required argument, which is the item you wish to add to your list.

Syntax: list.append(item)**toppings = ['pepperoni', 'sausage', 'mushroom']**toppings.append('onion')   
--> ['pepperoni', 'sausage', 'mushroom', 'onion']

The **extend()** method is similar to append() in that it allows you to add onto your list; however, the extend() method allows you to add all of the items from another iterable (list, tuple, set, etc.) to the end of your list as separate items instead of one item. The method takes one required argument, the iterable.

Syntax: list.extend(iterable)**toppings = ['pepperoni', 'sausage', 'mushroom']  
more\_toppings = ['onion', 'bacon']**toppings.**extend**(more\_toppings)  
--> ['pepperoni', 'sausage', 'mushroom', 'onion', 'bacon']To contrast...toppings.**append**(more\_toppings)--> ['pepperoni', 'sausage', 'mushroom', ['onion', 'bacon']]

As you can see above, when the extend() method is used with an iterable, each item in the iterable is added to the list as separate items no longer bounded. On the contrary, when the append() method is used with an iterable as the argument, the entire iterable is added to the list as one item. It’s always important to pay close attention to the commas and brackets present in a list.

## **pop() and remove()**

The **pop()** method allows you to remove an element from your list at a specified index value. The method can take one optional argument, the integer value of the index you wish to remove — by default, pop() will remove the last item in the list, as the default value is -1.

Syntax: list.pop(index)**toppings = ['pepperoni', 'sausage', 'mushroom']**toppings.pop(1)  
--> ['pepperoni', 'mushroom']toppings.pop()  
--> ['pepperoni', 'sausage']  
If you wanted to retrieve the removed item...extra = toppings.pop(1)  
extra --> 'sausage'

Like the pop() method, the **remove()** method allows you to remove an item from your list. The remove() method, though, removes the first occurrence of a specified value in a list. The method takes one required argument, the item you wish to remove.

Syntax: list.remove(item)**toppings = ['pepperoni', 'sausage', 'mushroom']**toppings.remove('sausage')  
--> ['pepperoni', 'mushroom']

For both pop() and remove(), if the argument is out of range or does not exist in the list, respectively, you will get an error.

## **sort() and reverse()**

The **sort()**method sorts a list by certain criteria. The method can take two optional arguments. The first argument is setting either reverse=True or reverse=False. By default, this argument is set to reverse=False, which will result in alphabetical order if the list consists of only strings, or ascending order if the list consists of only numbers. The second argument allows you to set a key= to a function that you can use to specify how exactly you want your list sorted if it’s more complex than the default ordering that the sort() method does. This could be a built-in Python function, a function you defined elsewhere in your program, or an in-line [lambda function](https://www.youtube.com/watch?v=goPKxNtuxWo) that you write.

Syntax: list.sort(reverse=True|False, key=function)**toppings = ['pepperoni', 'sausage', 'mushroom']**toppings.sort()  
--> ['mushroom', 'pepperoni', 'sausage']toppings.sort(reverse=True)  
--> ['sausage', 'pepperoni', 'mushroom']toppings.sort(reverse=True, key=lambda x: len(x))  
--> ['pepperoni', 'mushroom', 'sausage']  
\* Sorted in reverse order by length of the topping name  
**prices = [1.50, 2.00, 0.50]**prices.sort(reverse=False)  
--> [0.50, 1.50, 2.00]prices.sort(reverse=True)  
--> [2.00, 1.50, 0.50]  
 **pies = [['bacon', 'ranch'], ['sausage', 'peppers']]**pies.sort(reverse=True)  
--> [['sausage', 'peppers'], ['bacon', 'ranch']]  
\* Sorts iterators by their first value

The **reverse()** method simply reverses the order of the items in the list. The method takes no arguments.

Syntax: list.reverse()**toppings = ['pepperoni', 'sausage', 'mushroom']**toppings.reverse()  
--> ['mushroom', 'sausage', 'pepperoni'] **prices = [1.50, 2.00, 0.50]**prices.reverse()  
--> [0.50, 2.00, 1.50]

## **count()**

The **count()**method returns the number of occurrences of a specified item in a list. The method takes one required argument, which is the item you wish to find the count of. This method can be useful if you wish to find out what items appear more than once in a list.

Syntax: list.count(item)**toppings = ['pepperoni', 'sausage', 'mushroom', 'sausage']**toppings.count('sausage')  
--> 2toppings.count('pepperoni')  
--> 1toppings.count('bacon')  
--> 0

## **index()**

The **index()** method returns the index of the first occurrence of the specified item. The method takes one required argument, which is the item whose index you wish to find. If the item does not exist in the list, you will get an error.

Syntax: list.index(item)**toppings = ['pepperoni', 'sausage', 'mushroom', 'sausage']**toppings.index('mushroom')  
--> 2toppings.index('pepperoni')  
--> 0

## **insert()**

The **insert()** method inserts a specified item into a list at a specified index. The method takes two required arguments — the integer index you wish to insert the value at and the item you’d like to insert.

Syntax: list.insert(index, index)**toppings = ['pepperoni', 'sausage', 'mushroom']**toppings.insert(1, 'onion')  
--> ['pepperoni', 'onion', 'sausage', 'mushroom']

## **copy()**

The **copy()** method simply returns a copy of your list. The method takes no arguments.

Syntax: list.copy()**toppings = ['pepperoni', 'sausage', 'mushroom']**toppings**2** = toppings.copy()  
toppings**2** --> ['pepperoni', 'sausage', 'mushroom']

## **clear()**

The **clear()** method simply removes all items from a list, leaving an empty list. The method takes no arguments.

Syntax: list.clear()**toppings = ['pepperoni', 'sausage', 'mushroom']**toppings.clear()



Question 3

**10 most useful Python Dictionary Methods**

Dictionary is used in python to store multiple data with key-value pairs. It works like an associative array of other programming languages. The curly ({}) brackets are used to define a dictionary and the key-value is defined by the colon(:). The content of the key and value can be numeric or string. Python has many built-in methods to do different types of tasks on the dictionary data such as add, update, delete, search, count, etc. 10 most useful dictionary methods of python are explained in this article.

**Use of items() method**

**items()** method is used to return a list with the tuple pairs of all keys and values of a dictionary.

**Syntax:**

dictionary.items()

This method does not take any argument.

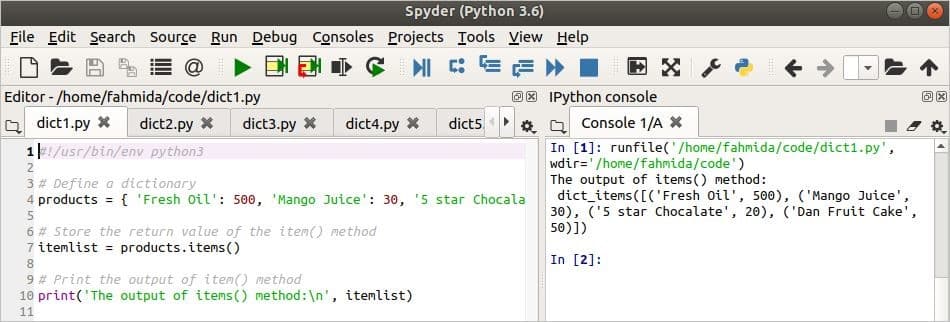
**Example:**

A dictionary variable named **products** is declared in the script. The keys contain the product name and the value contains the price of the product. Next, **items()** method is used for the dictionary and stored into another variable named **itemlist** and it is printed later.

*#!/usr/bin/env python3*  
  
*# Define a dictionary*  
products = { 'Fresh Oil': 500, 'Mango Juice': 30, '5 star Chocalate': 20,  
'Dan Fruit Cake':50 }  
  
*# Store the return value of the item() method*  
itemlist = products.items()  
  
*# Print the output of item() method*  
print('The output of items() method:\n', itemlist)

**Output:**

The following output will appear after running the script from spyder.

[](https://linuxhint.com/wp-content/uploads/2020/07/1-9.jpg)

**Use of keys() method**

**keys()** method is used to return a list of all keys of a dictionary.

**Syntax:**

dictionary.keys()

This method does not take any argument.

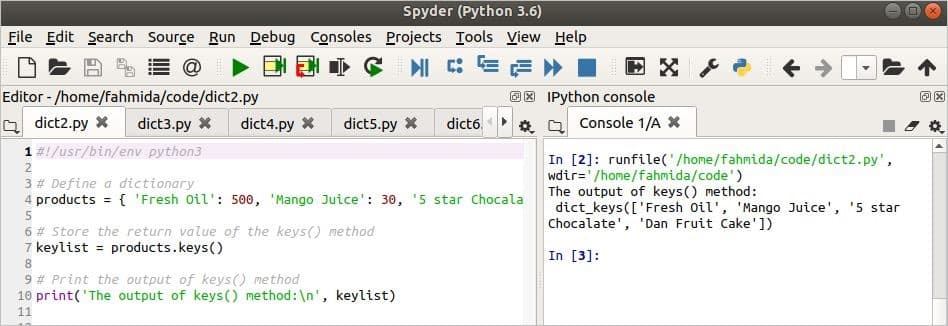
**Example:**

A dictionary variable named **products** is declared in the script. The keys contain the product name and the value contains the price of the product like the previous example. **keys()**method is applied in the dictionary and the return values are stored in the variable named keylist that is printed later. If you want the list of all values of the dictionary then you have to use **values()**method.

*#!/usr/bin/env python3*  
  
*# Define a dictionary*  
products = { 'Fresh Oil': 500, 'Mango Juice': 30, '5 star Chocalate': 20,  
'Dan Fruit Cake':50 }  
  
*# Store the return value of the keys() method*  
keylist = products.keys()  
  
*# Print the output of keys() method*  
print('The output of keys() method:\n', keylist)

**Output:**

The following output will appear after running the script from spyder.

[](https://linuxhint.com/wp-content/uploads/2020/07/2-9.jpg)

**Use of setdefault() method**

**setdefault()**method is used to get the value of any particular key from a dictionary if the key exists. This method can be used to set a default value if the specified key does not exist in the dictionary.

**Syntax:**

dictionary.setdefault(key\_value [, default\_value])

This method has two arguments. The first argument is mandatory and it is used to take the key value that will be searched in the dictionary. The second argument is optional and it is used to set a default value if the key used in the first argument does not exist in the dictionary. If the key does not exist in the dictionary and the default value is not defined then this method will return ‘**none**’.

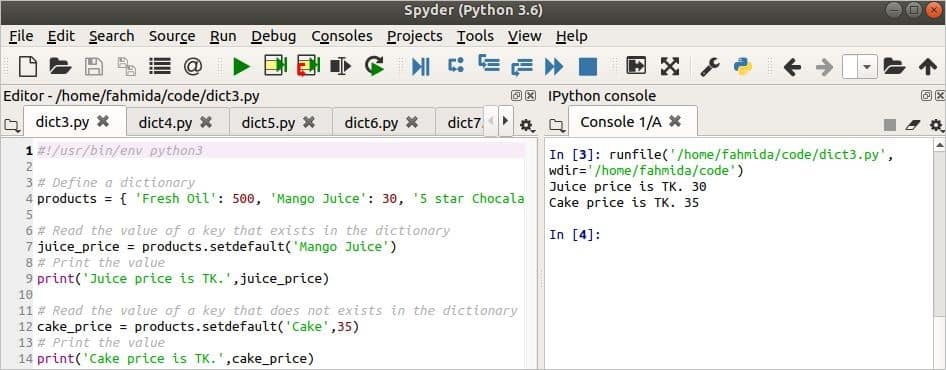
**Example:**

The following script shows the use of the **setdefault()**method in the dictionary. The dictionary named products is declared in the script like previous examples. Here, the **setdefault()** method is used with one argument for the first time and stored into the variable named  **juice\_price**. The method is used with two arguments for the second time and stored into the variable named **cake\_price**. Both variables are printed later.

*#!/usr/bin/env python3*  
  
*# Define a dictionary*  
products = { 'Fresh Oil': 500, 'Mango Juice': 30, '5 star Chocolate': 20,  
'Dan Fruit Cake':50 }  
  
*# Read the value of a key that exists in the dictionary*  
juice\_price = products.setdefault('Mango Juice')  
*# Print the value*  
print('Juice price is TK.',juice\_price)  
  
*# Read the value of a key that does not exists in the dictionary*  
cake\_price = products.setdefault('Cake',35)  
*# Print the value*  
print('Cake price is TK.',cake\_price)

**Output:**

The following output will appear after running the script from spyder. The key, ‘Mango Juice’ exists in the dictionary and the value of this key is 30 that is printed. The key, ‘Cake’ does not exist in the dictionary. So, the default value of the setdefault() method, 35 is printed.

[](https://linuxhint.com/wp-content/uploads/2020/07/3-11.jpg)

**Use of get() method**

get() method works similar to setdefault() method but there is one difference between these methods. Two arguments are mandatory in the get() method and the second argument is optional in the setdefault() method.

**Syntax:**

dictionary.get(key\_value , default\_value)

It will return the corresponding value from the dictionary which key matches the first argument of this method otherwise it will return the default value that is assigned in the second argument.

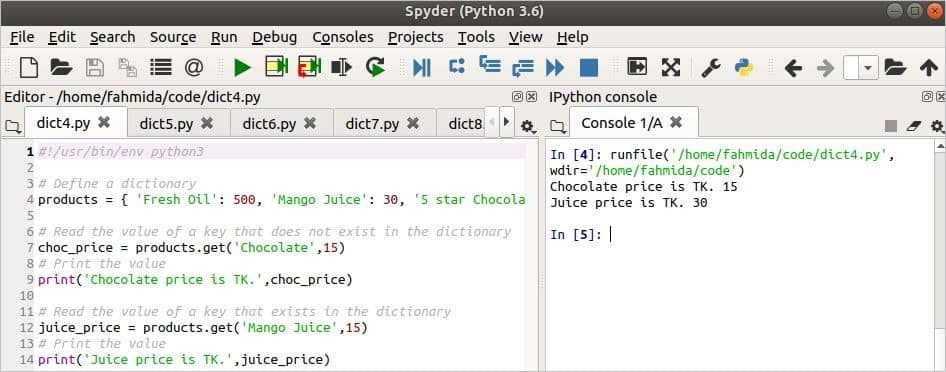
**Example:**

The same dictionary variable of the previous example is used in the following script. Here, get() method is used for two times with two different key values. The return values of this method are printed later.

*#!/usr/bin/env python3*  
  
*# Define a dictionary*  
products = { 'Fresh Oil': 500, 'Mango Juice': 30, '5 star Chocolate': 20,  
'Dan Fruit Cake':50 }  
  
*# Read the value of a key that does not exist in the dictionary*  
choc\_price = products.get('Chocolate',15)  
*# Print the value*  
print('Chocolate price is TK.',choc\_price)  
  
*# Read the value of a key that exists in the dictionary*  
juice\_price = products.get('Mango Juice',15)  
*# Print the value*  
print('Juice price is TK.',juice\_price)

**Output:**

The following output will appear after running the script from spyder. The first key used in get() method is ‘**Chocolate**’ that does not exist in the dictionary. So the default value is returned and printed. The second key value used in the get() method is ‘**Mango Juice**’ that exists in the dictionary and the corresponding value of that key is returned from the dictionary and printed.

[](https://linuxhint.com/wp-content/uploads/2020/07/4-9.jpg)

**Use of len() method**

**len()**method is used to count the total number of elements in the dictionary.

**Syntax:**

len(dictionary)

It takes a dictionary variable as an argument and returns the total numbers of elements of that dictionary.

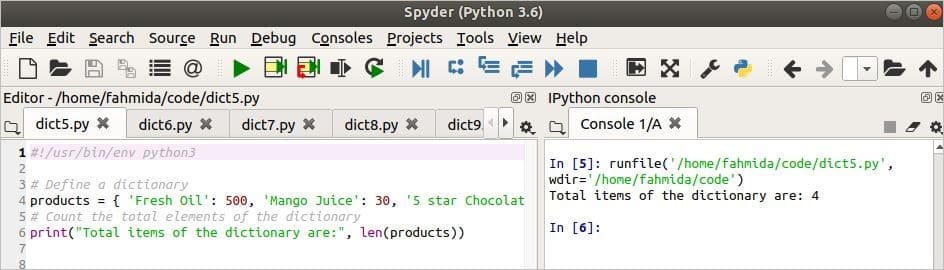
**Example:**

The following script counts the total elements of the dictionary named **products** and the returned value is printed.

*#!/usr/bin/env python3*  
  
*# Define a dictionary*  
products = { 'Fresh Oil': 500, 'Mango Juice': 30, '5 star Chocolate': 20,  
'Dan Fruit Cake':50 }  
*# Count the total elements of the dictionary*  
print("Total items of the dictionary are:", len(products))

**Output:**

The following output will appear after running the script from spyder. 4 elements in the dictionary are printed in the output.

[](https://linuxhint.com/wp-content/uploads/2020/07/5-4.jpg)

**Use of pop() method**

The**pop()** method is used to retrieve the particular value and remove the element from a dictionary based on the key value.

**Syntax:**

dictionary.pop(key [, value])

This method has two arguments. The first argument is mandatory which is used to take the key value. The second argument is optional and it is used to set the default value that will be returned if the key assigned in the first argument does not exist in the dictionary.

**Example:**

A dictionary variable named dictvar is declared in the following script that contains four elements. The **pop()** method is used for two times in the script with two different key values. In the first **pop()**method, ‘Phone’ is used as key and no optional value is used. In the second pop() method, the same key value is used with an optional value. The two returned values of this method are printed later.

*#!/usr/bin/env python3*  
  
*# Define a dictionary*  
dictvar = { 'Name': 'Sakib Al Hasan', 'Profession': 'Cricketer','Phone':'01866564234',  
'Salary':300000 } print("\nThe content of the dictionary:\n",dictvar)  
  
*# Read and delete a value from the dictionary if exists*  
print("\nThe phone no is:", dictvar.pop('Phone'))  
  
*# Print the dictionary after pop*  
print("\nThe content of the dictionary after pop:\n",dictvar)  
  
*# Read a key of the dictionary that does not exist*  
print("\nThe phone no is:", dictvar.pop('Phone','01766345234'))

**Output:**

QUESTION 4

Image result for 10 tuple method and how to use them

Python Tuple Functions

len() Like a list, a Python tuple is of a certain length. ...

max() It returns the item from the tuple with the highest value. ...

min() Like the max() function, the min() returns the item with the lowest values. ...

sum() ...

any() ...

all() ...

sorted() ...

tuple()