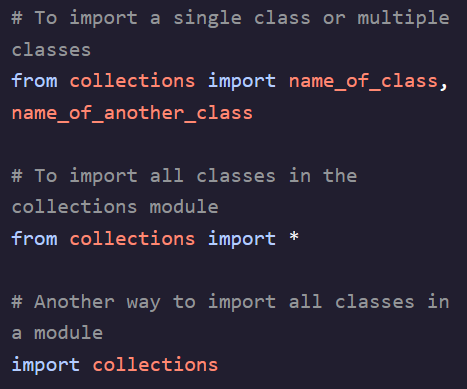
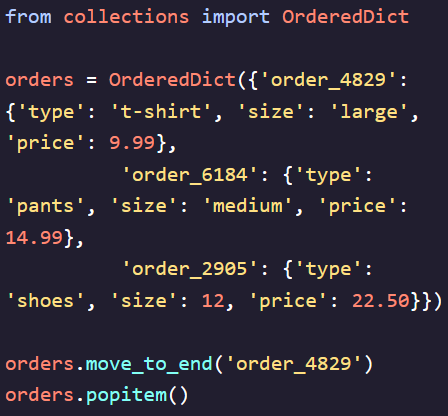
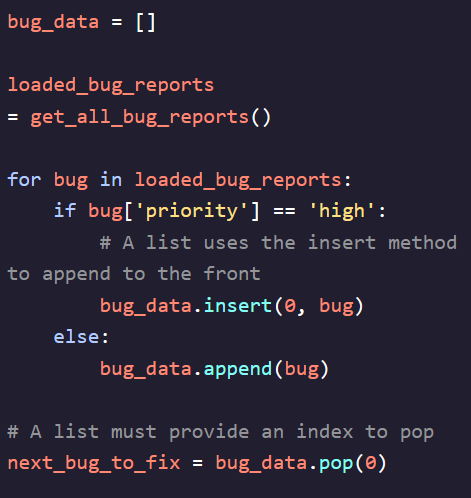
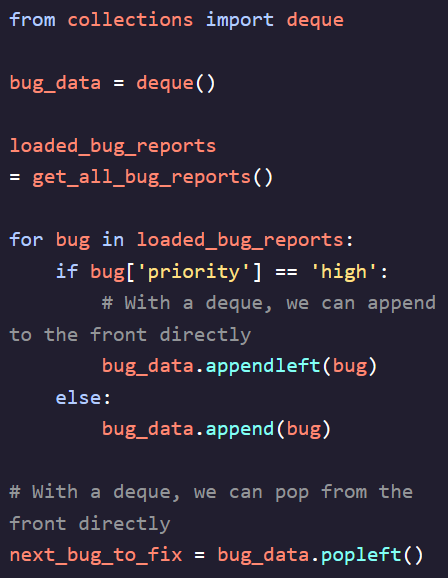
**Containers in Python:**

- Any object that stores and organizes data (elements in a list, key-value pairs in a dictionary, or accessing data within tuples)  
- Many specialized containers exist within Python for specific jobs and can be imported into code from modules or even custom-made  
- ***Lists*** - An ordered group of elements. Elements can be added, removed, accessed, and modified  
- ***Tuples*** - Immutable objects which group multiple elements together. Similar to lists, except that they cannot be modified once created  
- ***Dictionaries*** - Unordered groups of key-value pairs  
- ***Sets*** - Unordered groups of elements that cannot contain duplicates, elements cannot be modified.

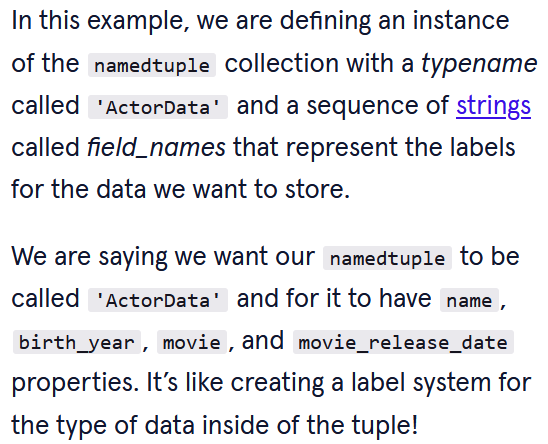
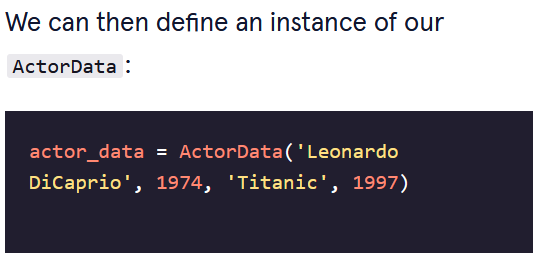
**Specialized Containers:**

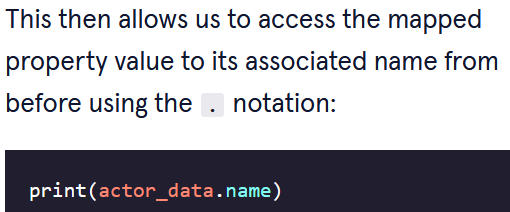
- Classes using the ***collections*** module are very similar to the built-in containers we have already used  
- Each of these specialized containers focuses on a certain improvement to its built-in counterpart such as optimizing performance, better organization, fewer steps for performing tasks, and more!  
- Need to import ***collections*** module before you can use it  
 

**Deque:**

- Lists are not optimized for appending and popping large amounts of data but they are great at accessing data at any index provided  
- ***Deque*** – Similar to lists but are optimized for appending and popping to the front and back rather than having optimized accessing.  
- Great for working with data where elements in the middle don’t need to be accessed very often or at all  
 

**Named Tuple:**

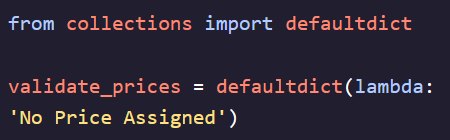
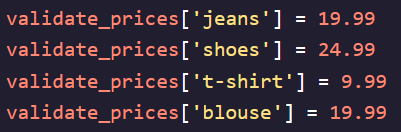
- With a regular tuple we run into issues when hosting various data or even nested data  
- When slicing using indices it is hard to know what exactly that piece of data is and we need to create a variable for each separate property   
- ***namedtuple*** – Allows us to have an immutable tuple object, but every element becomes self-documented  
   



- ***namedtuples*** use CapWords convention because they actually return a subclass and need to follow the conventions used for classes  
- The key advantages over dictionaries are  
1. They are immutable and maintain their order while dictionaries do not  
2. They are more lightweight than dictionaries and take up no more memory than a regular tuple

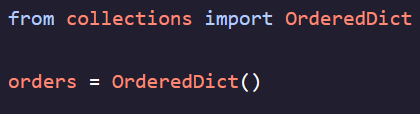
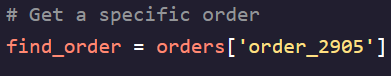
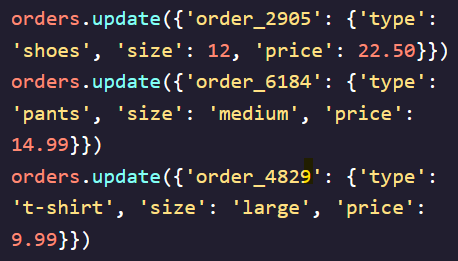
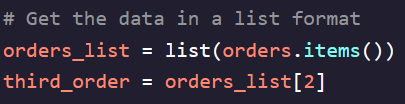
**DefaultDict:**

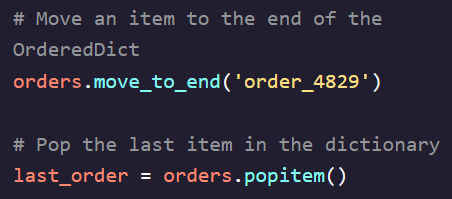
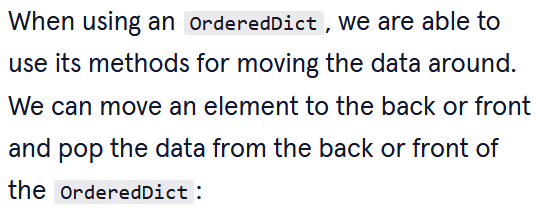
- When we try to access a key-value pair in a dictionary but the key does not exist, it will normally throw a KeyError  
- ***defaultdict*** – allows us to have a default missing value in the dictionary

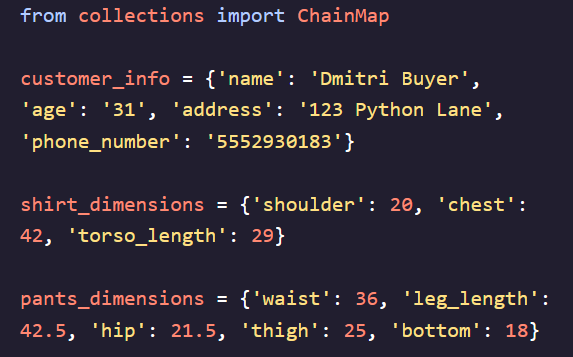

**OrderedDict:**

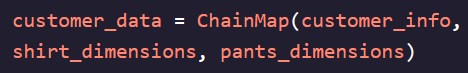
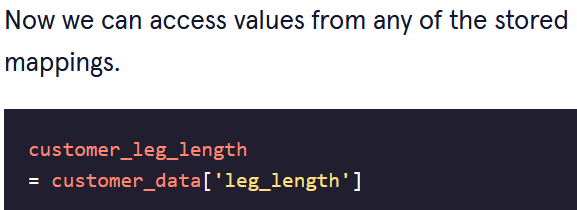
- Allows us to access values using keys but also preserves the order of the elements inside of it. Useful for keeping track of multiple dictionaries, and accessing their contents

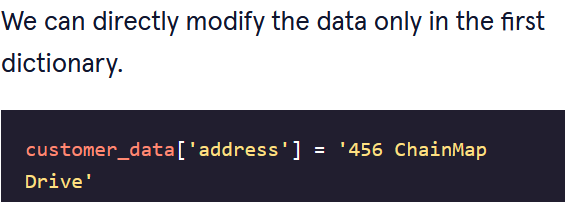
   
 

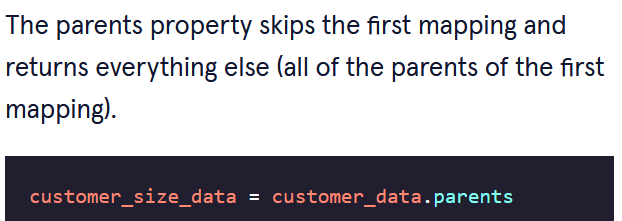
 

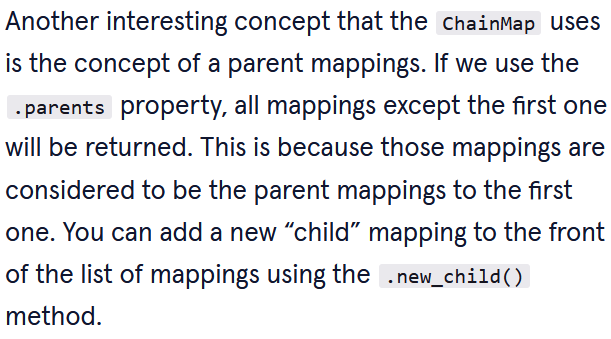
**ChainMap:**

- Allows us to store many mappings in an ordered group but lookups (accessing the value using a key) are repeated for every mapping inside of ***ChainMap*** until something is found or the end is reached  
- If we try to modify data, only the first mapping receives the changes (think of it as a giant dictionary which stores all of the other dictionaries, and if there are repeated ‘keys’ then the first found result is returned)  
- To modify data in deeper dictionaries, you need to iterate through them using a loop  


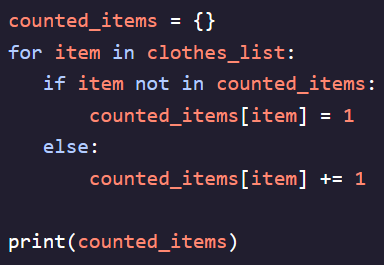
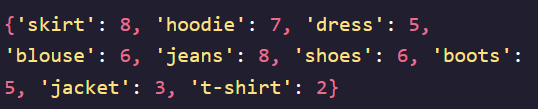
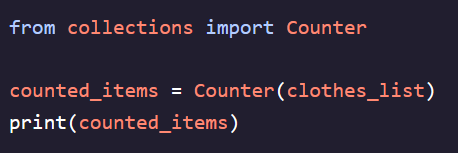
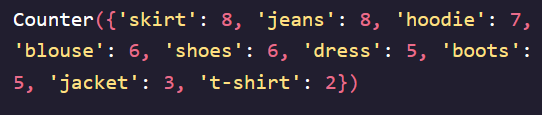
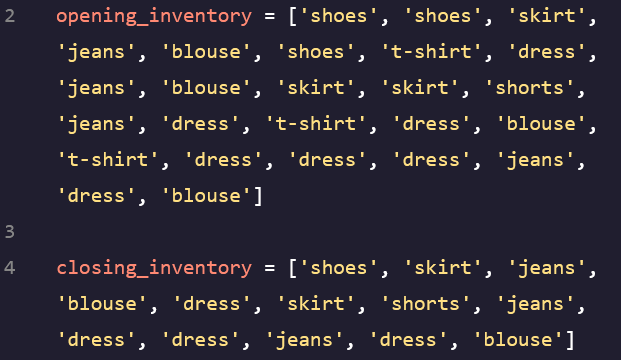
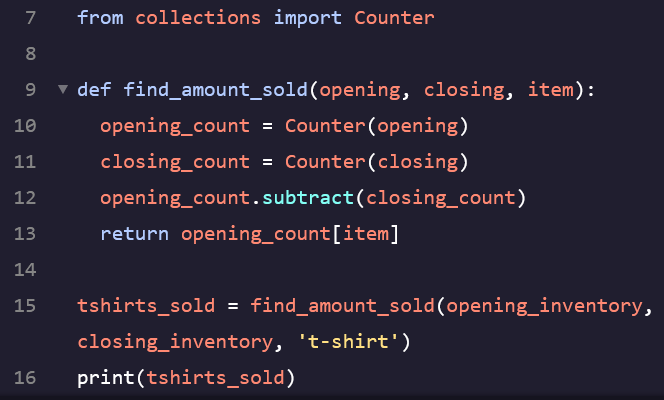
  




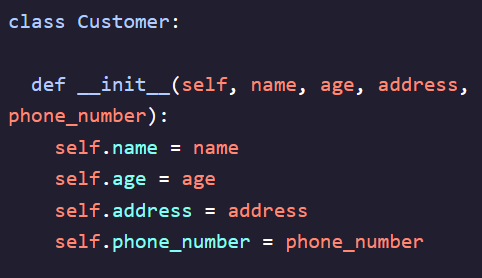
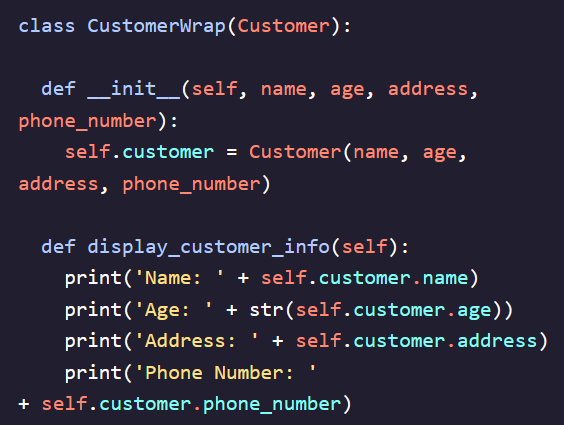
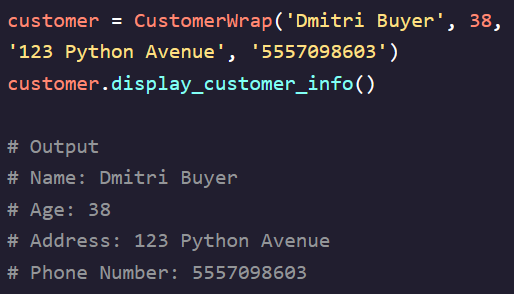




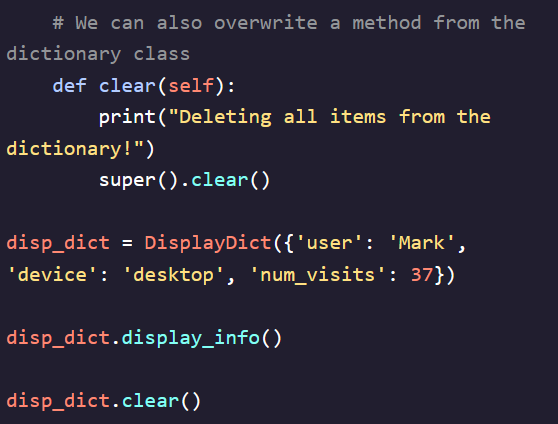
**Counter:**

- A common task performed in Python is counting the instances of an element in a collection  
- Instantly counts elements for any hashable object and stores the data as a dictionary where the keys are the elements and the values are the number of occurrences  
- Also includes methods such as mathematical operations for subtracting one count dictionary from another, finding most common elements, generating a new list based of number of occurrences of an element and etc…  
    
   
 

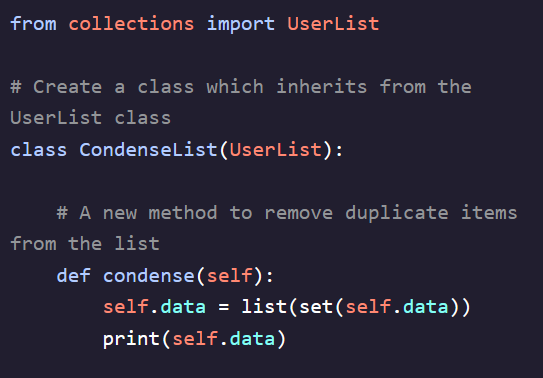
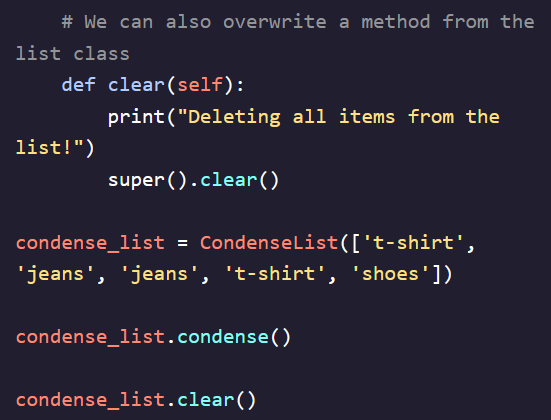
**Wrapper Containers:**

- Wrappers are modifications to functions or classes which change the behavior in some way   
- Allow us to create different variations of classes with different purposes while avoiding duplicate code  
   
 

**UserDict:**

- Allows us to create our own version of a dictionary that has all the functionality of a normal dictionary but elements can be accessed directly through the ***data***property  
 

**UserList:**

- Allows us to create our own version of a list that has all the functionality of a normal dictionary but elements can be accessed directly through the ***data***property  
 

**UserString:**

- Allows us to create our own version of a string that has all the functionality of a normal dictionary but elements can be accessed directly through the ***data***property  
