Lab 6

Assignment 6 brings together Python and *sets* that were covered in the Math. You will probably be helped by drawing some of the sets. This lab is in two parts—which are the first two parts of Assignment 6.

# Build 1

# Build 1 Part A: Known issues

Here is the introduction to the program:

*'''*

*INTENT: Produce all partitions of a set  
  
EXAMPLE: A teacher wants to partition her (small) class consisting of Adam, Carlene, and Frank,  
to create study groups. Find the possible ways to do this, each partition being a list of lists.*

*The output would be as follows (though not necessarily on separate lines):  
[[Adam, Carlene, Frank]] -- a list consisting of just one list  
[[Adam], [Carlene, Frank]]  
[[Adam, Carlene], [Frank]]  
[[Adam, Frank], [Carline]]   
[[Adam], [Carlene], [Frank]] -- a list consisting of three simple lists  
  
KNOWN ISSUES: Used lists but order is irrelevant.*

*'''*

In software development, a “known issue” is one that the developer is aware of, could not attend to, and notes for subsequent work as a professional obligation. They are not considered serious at this stage, and they often refer to clear improvements. In 1 or 2 sentences, explain why and how the one above is indeed an issue.

***Answer:***

* List is caring about order and allow repeated. Although, Sets are no order and no duplication. So, the partition in the given problem appears to be a set much more than a list.
* Because list keep the order, so [[Adam, Carlene], [Frank]] is different of [[Frank], [Adam, Carlene]], but they are equivalent in this situation because order does not matter.

# Build 1 Part B: Program decomposition +

Consider the (helper) function *extend\_partitions()* given below. You are not being asked to code for this part of the assignment, just answer questions about the function.

1. Using the documentation supplied as comments in the code, explain why the postcondition is fulfilled. You can assume that each code block fulfils its stated objectives.

For example, below is the objective (the desired outcome) of the first code block. The part in parentheses *(Excluding [a\_new\_element]):* is a label for the objective so you can get an idea of what it’s about before reading the details. Notice that there is a difference between simply *an element* such as ‘hi’ and *the list consisting of the element* such as [‘hi’].

*# (Excluding [a\_new\_element]): returned\_partition includes all partitions  
 # of (S union {a\_new\_element}) that don't contain the list [a\_new\_element]*

***Answer:***

The postcondition is fulfilled because:

* When added a\_new\_element to the S set, a\_new\_element is added to all of set elements (some\_partitions = [[[0, 11]], [[0], [11]]]), so the set contains all list elements and a\_new\_element, but doesn’t contain the list [a\_new\_element].
* Each object in the sublist (\_partition) of some\_partition, the new element is attached. When a\_new\_element is added to the sublist (\_partition) of the nested list (the original list (some\_partitions))
* With the first block code:
  + The for loop loops through all of the nested list (some\_partition) **“*for \_partition in some\_partitions*”,**
  + and in each iteration, the internal for loop continue loop through the sublist (\_partition) by the length of the sublist **“*for i in range(len(\_partition)):”.***
  + Each iteration of internal for loop we create a new\_partition by deepcopy() sublist (\_partition in the block code). “***new\_partition = copy.deepcopy(\_partition)”***
  + A\_new\_element is added to the new\_partition list, “***new\_partition[i].append(a\_new\_element)***”
  + New\_partition list is appended to the return list (returned\_partitions) which will be returned by extend\_partitions function. ***“returned\_partitions.append(new\_partition)”***
* With the second block of code:
  + The for loop loops through all of the nested list (some\_partition) **“*for \_partition in some\_partitions*”,**
  + And in each iteration, we created appended\_partition by the deepcopy() sublist ***“appended\_partition = copy.deepcopy(\_partition)”***
  + Appending a\_new\_element to appended\_partition ***“appended\_partition.append([a\_new\_element])”***
  + Appened\_partition list is appended to the return list (returned\_partitions) which will be returned by extend\_partitions function. ***“returned\_partitions.append(appended\_partition)***

1. Explain why *deepcopy*() was used rather than just *copy*(). You may need to do a bit of research.

***Answer:***

It is better to use deepcopy() because:

* With copy() functions, it creates a new list but when we do any changes to this new copies list, it have an effect on the original list if the list is 2 dimension. It only copies the first layer of the list. Which is mean when we have nested list (2 dimension list), it still has references from the original list. Ex: if we make any changes in the sublist of the original list it will also make changes in the new copy list. Because the original list should not be changed, so this copy() function is not appropriate in this situation.
* With deepcopy(), it allows to copy whole new list (list or 2 dimension list) and then, recursively, insert copies into it of the objects found in the original. Deepcopy() creates independent copy of original object and all its nested objects. So if we make changes to original list, we will see no changes to the new list. Because we are not modifying the original list in this situation, the deepcopy() is appropriate in this situation.

*Reference :*

https://docs.python.org/3/library/copy.html

**import** copy  
  
  
**def** extend\_partitions(some\_partitions, a\_new\_element):  
 *'''  
 Preconditions:  
 1. some\_partitions consists of partition of a set S (which need  
 not be specified!) in the form of a list of lists of the set's elements.  
 Example: S = {0, 11}, some\_partitions = [[[0, 11]], [[0], [11]]]  
 2. a\_new\_element does not occur in some\_partitions  
  
 Returns: returned\_partitions = all partitions of (S union {a\_new\_element})  
  
 Example: for S = {0, 11} and a\_new\_element = 22, this returns the following  
 list (containing 5 elements): [[[0, 11, 22]], [[0, 22], [11]],  
 [[0], [11, 22]], [[0, 11], [22]], [[0], [11], [22]]]  
 '''* returned\_partitions = []  
  
 *# (Excluding [a\_new\_element]): returned\_partition includes all partitions  
 # of (S union {a\_new\_element}) that don't contain the list [a\_new\_element]  
  
 # Example: For S = {0, 11} and a\_new\_element = 22, returned\_partitions would include  
 # [[0, 11, 22]], [[0, 22], [11]], [[0], [11, 22]], and [[[0], [11]], 22]]]  
 # (notice that none of these partitions contains [22])* **for** \_partition **in** some\_partitions: *# e.g., \_partition = [[0], [11]]* **for** i **in** range(len(\_partition)): *# e.g., i points to [0]* new\_partition = copy.deepcopy(\_partition)  
 new\_partition[i].append(a\_new\_element) *# e.g., get [[0, 22], [11]]* returned\_partitions.append(new\_partition)  
  
 *# (Including [a\_new\_element]): returned\_partition includes all partitions  
 # of S union {a\_new\_element} that contain [a\_new\_element]  
  
 # e.g., For the example above, returned\_partition includes  
 # [[0, 11], [22]] and [[0], [11], [22]]]* **for** \_partition **in** some\_partitions: *# e.g., [[0, 11]]* appended\_partition = copy.deepcopy(\_partition)  
 appended\_partition.append([a\_new\_element])  
 returned\_partitions.append(appended\_partition)  
 *# e.g., append [[0, 11], [22]] in the example* **return** returned\_partitions