Assignment 6

As usual, the parts of this assignment are worth equal amounts. This assignment brings together Python and *sets* that were covered in the Math. You will probably be helped by drawing some of the sets

# 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.

**ANS:** The reason why that “used lists but order is irrelevant” is an issue is because that we should not use python “List here”, for now we are only having 3 students is the list, we could arrange them into different possibilities of party without changing their orders, but when where are more than 3 students, we need to change their order in order to arrange them into all different possibilities. Changing the order will make the list different.

# 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]*

**ANS:** The postcondition is fulfilled because of for *(Including [a\_new\_element]):* it will return the partition that includes all the partitions of S union and SET *{a\_new\_element}* that contain LIST *[a\_new\_element]* which is [[0, 11], [22]] and [[0], [11], [22]]]

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

**ANS:** because of *copy()* are not copying everything its just sharing one elements with the two collectors, but when we are using *deepcopy()* it will construct a new compound object and then inserts copies into from the copies of the original objects, so that you could modify the *deepcopy* individually.

**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

# Build 2: Tests

Here is a test for *extend\_partitions*(). It prints an application of *extend\_partitions*() to particular parameters, and then prints what the output should be. The user can compare these to be sure that the function is operating as specified by its postconditions.

*# TESTS =========================================  
  
# OF extend\_partitions() ============*print(extend\_partitions([[[0],[11]]], 22))  
print(**"[[[0, 22], [11]], [[0], [11, 22]], [[0], [11], [22]]]\n"**)

Provide two additional tests for *extend\_partitions*() in the same form as this one.

**ANS:**

print(extend\_partitions([[[0],[11]]], [[[37],[89]]]))

print("**[[[0, [[[37], [89]]]], [11]], [[0], [11, [[[37], [89]]]]], [[0], [11], [[[[37], [89]]]]]]\**n")

print(extend\_partitions([[[0],[11],[5],[2]]], [[[37],[89]]]))

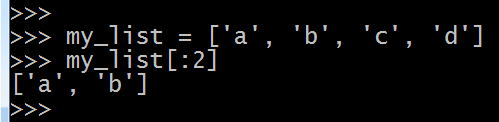
print**("[[[0, [[[37], [89]]]], [11], [5], [2]], [[0], [11, [[[37], [89]]]], [5], [2]], [[0], [11], [5, [[[37], [89]]]], [2]], [[0], [11], [5], [2, [[[37], [89]]]]], [[0], [11], [5], [2], [[[[37], [89]]]]]]**\n")

# Build 3: Writing your code

Use *extend\_partitions*() (cut and paste it) to write the following function.

**def** all\_partitions\_of(a\_list):  
 *'''  
 Precondition: a\_list is any list  
  
 Returns returned\_partitions = a list of all partitions of a\_list}  
  
 Example: for a\_list = [0, 11, 22], this function returns  
 [[[0, 11, 22]], [[0, 22], [11]], [[0], [11, 22]], [[0, 11], [22]],  
 [[0], [11], [22]]]  
 '''*

You may need the following kind of operation:



Here is the algorithm breakdown that I used.

*# (Trivial?): EITHER a\_list has >1 element*

*# OR a\_list has <2 elements and this returned  
# and returned\_partitions = the list of all partitions of a\_list*

… 2 lines of code …

Here is the next objective I fulfilled, complete with code:

*# (i): 0 < i <= len(a\_list) AND  
# returned\_partitions = the list of all partitions of a\_list[:i]*i, returned\_partitions = 1, [[[a\_list[0]]]]

Here is the last objective:

*# i >= len(a\_list)*… code …

My code is a *while* loop with two lines within it, and it uses the helper function extend\_partitions().

Conclude the code with the following line:

**return** returned\_partitions

Here is why the stated objectives are sufficient: if the execution makes it to the end, you can conclude …

*a\_list has >1 element*

AND

*0 < i <= len(a\_list)*

AND *returned\_partitions = the list of all partitions of a\_list[:i]*AND

*# i >= len(a\_list)*

so …

*returned\_partitions = the list of all partitions of a\_list*

Here are tests that you can use to validate your code:

*# OF all\_partitions\_of() ===========*print(**"[[[]]]<-->"** + str(all\_partitions\_of([])) + **'\n'**) *# one list--consisting of []*print(**"[[[333]]]<-->"** + str(all\_partitions\_of([333])) + **'\n'**)  
  
print(**'all\_partitions\_of([0,22]):'**)  
print(all\_partitions\_of([0,22]))  
print(**"[[[0, 22]], [[0], [22]]]\n"**)  
  
print(**'all\_partitions\_of([0,11,22]):'**)  
print(all\_partitions\_of([0,11,22]))  
print(**"[[[0, 22], [11]], [[0], [11, 22]], [[0, 11, 22]], [[0], [11], [22]], [[0, 11], [22]]]\n"**)  
  
print(**'all\_partitions\_of([0,11,22,33]):'**)  
print(all\_partitions\_of([0,11,22,33]))  
  
print(**'all\_partitions\_of([7,8,1,3,5]):'**)  
print(all\_partitions\_of([7,8,1,3,5]))

The last of these tests produces the following:

all\_partitions\_of([7,8,1,3,5]):

[[[7, 8, 1, 3, 5]], [[7, 1, 3, 5], [8]], [[7, 1, 3], [8, 5]], [[7, 1, 5], [8, 3]], [[7, 1], [8, 3, 5]], [[7, 3, 5], [8, 1]], [[7, 3], [8, 1, 5]], [[7, 5], [8, 1, 3]], [[7], [8, 1, 3, 5]], [[7, 8, 3, 5], [1]], [[7, 8, 3], [1, 5]], [[7, 8, 5], [1, 3]], [[7, 8], [1, 3, 5]], [[7, 3, 5], [8], [1]], [[7, 3], [8, 5], [1]], [[7, 3], [8], [1, 5]], [[7, 5], [8, 3], [1]], [[7], [8, 3, 5], [1]], [[7], [8, 3], [1, 5]], [[7, 5], [8], [1, 3]], [[7], [8, 5], [1, 3]], [[7], [8], [1, 3, 5]], [[7, 8, 1, 5], [3]], [[7, 8, 1], [3, 5]], [[7, 1, 5], [8], [3]], [[7, 1], [8, 5], [3]], [[7, 1], [8], [3, 5]], [[7, 5], [8, 1], [3]], [[7], [8, 1, 5], [3]], [[7], [8, 1], [3, 5]], [[7, 8, 5], [1], [3]], [[7, 8], [1, 5], [3]], [[7, 8], [1], [3, 5]], [[7, 5], [8], [1], [3]], [[7], [8, 5], [1], [3]], [[7], [8], [1, 5], [3]], [[7], [8], [1], [3, 5]], [[7, 8, 1, 3], [5]], [[7, 1, 3], [8], [5]], [[7, 1], [8, 3], [5]], [[7, 3], [8, 1], [5]], [[7], [8, 1, 3], [5]], [[7, 8, 3], [1], [5]], [[7, 8], [1, 3], [5]], [[7, 3], [8], [1], [5]], [[7], [8, 3], [1], [5]], [[7], [8], [1, 3], [5]], [[7, 8, 1], [3], [5]], [[7, 1], [8], [3], [5]], [[7], [8, 1], [3], [5]], [[7, 8], [1], [3], [5]], [[7], [8], [1], [3], [5]]]