Example: Understanding Permutations

I understand that permutations can be difficult to understand, so we are going to walk through the sample code with the simple list [“4”,”5”,”6”].

Starting with the driver program on Line 33:

data = list(‘456’)

Then we go into a FOR loop:

for p in permutation(data):

    print (p)

What this code is saying is that we will have to go through the permutation function to print out the actual permutation, so let’s head back to Line 2, where the function begins.

def permutation(lst):

I marked this run of the function in **RED** (to help make things easier to understand).

So, the first comment says – if lst is empty, then there are no permutations. That is NOT the case because there are values in lst.

Skip down to the comment on Line 8, which says – If there is only one element in lst then only one permutation is possible.

This means that if there is only ONE value in the list, then the permutation is the list itself. However, there is more than one value in the list, so we must continue through the code.

So now we are down to the comment on Line 13, which says – Find the permutations for lst if there are more than one character(s).

That is the case we are in.

Create a new empty list that will store the current permutation:

l = [ ]

Next, look at the list data. We have a new FOR loop to move through.

```
for i in range(len(lst)):
        m = lst[i]
```

What this means is from zero (because Python lists begin with zero) all the way to the length of the list (we have 3 characters in the list), we have to go through Lines 20 through 29, until we have handled everything in the list.

List item 0, looking at our data, will be 4. So m = "4".

Next, we have to determine the remaining list values. The remaining list, or remLst, will be the value of the current list from the beginning of the list to the value of I, PLUS the value of the current list from i+1 to the end of the list.

What does that look like in code?

remLst = lst[:i] + lst[i+1:] ➔ remLst = lst[:0] + lst[0+1:]

Looking at our data, the remaining list remLst will be **[“5”,”6”]**.

At Line 28, we have another FOR loop to go through, which will run the permutation of the remaining list remLst:

```
for p in permutation(remLst):
        l.append([m] + p)
```

Looking at our data, this means we will append m (which is 4) to our list, then go back through the permutation function to determine the rest of the list.

Going back up to Line 2 with the remLst value [“5”, “6”] – and I will mark this run of the permutation function in **BLUE** so you can see what will happen:

Line 5 is still FALSE, because the length of the list above is NOT zero.

Line 10 is still FALSE, because the length of the list above is NOT one.

We now have a new empty list l for this run of the FOR loop in Line 28.

l = [ ]

You would think that Line 19 would be the next number in the iteration of the FOR loop, but remember, WE ARE STARTING OVER with the permutation of the remaining list.

So Line 19 is looking at the NEW list that has the values **[“5”, “6”]** and the value of i starts over again at zero through the length of the NEW list.

The new value of lst[0] is now “5”, so this becomes the new value of m.

Then we have to determine the NEW remaining list remLst.

remLst = lst[:i] + lst[i+1:] ➔ remLst = lst[:0] + [0+1:]

Looking at the data, our remLst will be [“6”].

At Line 28, we have another FOR loop to go through, which will run the permutation of the remaining list remLst:

```
for p in permutation(remLst):
        l.append([m] + p)
```

We will append m (which is now 5) to our list, then go back through the permutation function to determine the rest of the list.

Going back up to Line 2 with the remLst value [“6”], and changing colors to **ORANGE** to see what is happening:

Line 5 is still FALSE, because the length of the list above is NOT zero.

BUT look at this - Line 10 is now TRUE, because the length of the list above is one. So that means we return [lst] (in our case, we return **[“6”]**) to the place where it was called.

It was called from the FOR loop in Line 28

Our list l now has the following values:

l = [“5”, “6”]

The “5” is the value of m, and the “6” is the appended value of p.

We are not done yet. Not yet.

We have to go back up to Line 19 to complete that FOR loop also. What are we using for that lst, though?

Remember when our remLst value was **[“5”, “6”]** the second time we went through the permutation function? We went through the FOR loop the first time where i = 0. NOW we have to execute the FOR loop where i = 1.

Looking at remLst = **[“5”, “6”]**:

m = lst[1] = **“6”**

Then we extract the NEW remaining list in Line 24:

remLst = lst[:i] + lst[i+1:] ➔ remLst = lst[:1] + lst[1+1:]

Our NEW remaining list remLst is now [“5”].

Moving to Line 28, we will be heading back to the permutation function with the value of remLst, which is [“5”], and we will change colors to PLUM to see what is going on in this iteration.

Line 5 is still FALSE, because the length of the list above is NOT zero.

BUT look at this - Line 10 is now TRUE, because the length of the list above is one. So that means we return [lst] (in our case, we return **[“5”]**) to the place where it was called.

It was called from the FOR loop in Line 28:

Our list l now has the following values:

l = [“6”, “5”]

The “6” is the value of m, and the “5” is the appended value of p.

So, are we done NOW?

Sorry, we are not done. These two lists have to be sent back to where they were called. In this case, they were called in Line 28, where m = “4”.

At this point, we have two lists in l:

[“4”, “5”, “6”]

[“4”, “6”, “5”]

We have to go back up to Line 19 to complete that FOR loop also.

Hold on, STOP. Didn’t we just get through that FOR loop?

We got through the FOR loop only for the smaller, two value remLst. We now have to continue this FOR loop on Line 19 for the original data list. Remember the RED marked at the top of this guide?

```
for i in range(len(lst)):
        m = lst[i]
```

We ONLY completed this FOR loop where m=lst[i] = lst[0] = “4”. We now have to execute this entire FOR loop (Lines 20 through 29) where i is now equal to 1.

This is where we stand:

- We have completed all the permutations of the original list where “4” is the first value of the list.
- We still have to complete all the permutations of the original list where “5” is the first list value, and then we have to complete all the permutations of the original list where “6” is the first list value.

Now, going back to Line 19:

for i in range(len(lst)):

m = lst[i]

Now the value of i is equal to 1.

List item 1, looking at our original data [“4”, “5”, “6”], will be 5. So m = “5”.

From this point, we have to determine the remaining list values. The remaining list, or remLst, will be the value of the current list from the beginning of the list to the value of I, PLUS the value of the current list from i+1 to the end of the list.

What does that look like in code?

remLst = lst[:i] + lst[i+1:] ➔ remLst = lst[:1] + lst[1+1:]

Looking at our data, the remaining list remLst will be **[“4”,”6”]**.

How in the world did we get the values 4 and 6 from the code remLst = lst[:i] + lst[i+1:]???

- Well, the first part of the remaining list will be the slice of the list from the beginning to value 1. Looking at the original list, that would be “4”.
- The second part of the remaining list will be the slice of the list from i+1 to the end of the list. Looking at the original list, that would be “6”.

At Line 28, we have another FOR loop to go through, which will run the permutation of the remaining list remLst:

for p in permutation(remLst):

    l.append([m] + p)

Looking at our data, this means we will append m (which is 5) to our list, then go back through the permutation function to determine the rest of the list.

Going back up to Line 2 with the remLst value [“4”, “6”] – and I will mark this run of the permutation function in **GREEN** so you can see what will happen:

Line 5 is still FALSE, because the length of the list above is NOT zero.

Line 10 is still FALSE, because the length of the list above is NOT one.

We now have a new empty list l for this run of the FOR loop in Line 28.

l = [ ]

You would think that Line 19 would be the next number in the iteration of the FOR loop, but remember, WE ARE STARTING OVER with the permutation of the remaining list.

So Line 19 is looking at the NEW list that has the values **[“4”, “6”]** and the value of i starts over again at zero through the length of the NEW list.

The new value of lst[0] is now “4”, so this becomes the new value of m.

Then we have to determine the NEW remaining list remLst.

remLst = lst[:i] + lst[i+1:] ➔ remLst = lst[:0] + [0+1:]

Looking at the data, our remLst will be [“6”].

At Line 28, we have another FOR loop to go through, which will run the permutation of the remaining list remLst:

```
for p in permutation(remLst):
        l.append([m] + p)
```

We will append m (which is now 4) to our list, then go back through the permutation function to determine the rest of the list.

Going back up to Line 2 with the remLst value [“6”], and changing colors to **NAVY** to see what is happening:

Line 5 is still FALSE, because the length of the list above is NOT zero.

Line 10 has changed to TRUE, because the length of the list above is one. So that means we return [lst] (in our case, we return **[“6”]**) to the place where it was called.

It was called from the FOR loop in Line 28

Our list l now has the following values:

l = [“4”, “6”]

The “4” is the value of m, and the “6” is the appended value of p.

We are not done yet. Not yet.

We have to go back up to Line 19 to complete that FOR loop also. What are we using for that lst, though?

Remember when our remLst value was **[“4”, “6”]** the second time we went through the permutation function? We went through the FOR loop the first time where i = 0. NOW we have to execute the FOR loop where i = 1.

Looking at remLst = **[“4”, “6”]**:

m = lst[1] = **“6”**

Then we extract the NEW remaining list in Line 24:

remLst = lst[:i] + lst[i+1:] ➔ remLst = lst[:1] + lst[1+1:]

Our NEW remaining list remLst is now [“4”].

Moving to Line 28, we will be heading back to the permutation function with the value of remLst, which is [“4”], and we will change colors to PINK to see what is going on in this iteration.

Line 5 is still FALSE, because the length of the list above is NOT zero.

This iteration, Line 10 changed to be TRUE, because the length of the list above is one. So that means we return [lst] (in our case, we return **[“4”]**) to the place where it was called.

It was called from the FOR loop in Line 28:

Our list l now has the following values:

l = [“6”, “4”]

The “6” is the value of m, and the “4” is the appended value of p.

So, are we done NOW?

We are STILL not done. These two lists have to be sent back to where they were called. In this case, they were called in Line 28, where m = “5”.

At this point, we have two MORE lists added to list l:

[“4”, “5”, “6”]

[“4”, “6”, “5”]

- **[“5”, “4”, “6”]**
- **[“5”, “6”, “4”]**

Based on what we have completed, we see that we have ONE MORE FOR LOOP ITERATION TO GO in Line 19:

The value of i is now equal to 2, and the value of m = lst[i] = lst[2] = 6.

You probably already can see the last two values of our list l, can’t you?

You are correct; the values are:

[“6”, “4”, “5”]

[“6”, “5”, “4”]

The **GRAY** and the **GOLD** above are used to emphasize the executions of the permutation function where i=2 in the FOR loop on Line 19.

Once all of this has been done, we can move to Line 30, where it says

return l

Where are we returning this list?

We are sending it back to the driver program, to where it says

print p

This will print out the complete list of permutations for the original list that was given.