ACE Task Submission – Programming

Hard Task

Introduction

I'm Kunal Kumar, from BCA Shift-1 Section-B. My CET Rank was 53. I have submitted the hard task which was:

Given a string containing digits from 2-9 inclusive, return all possible letter combinations that the number could represent. Return the answer in any order. A mapping of digits to letters (just like on the telephone buttons) is given below. Note that 1 does not map to any letters.

Here is the link to the github repo: [ACE-Task-Submission-Programming]

Code

Approach

I decided to do the problem in python because its the language, I'm most comfortable in and the one I have worked with for the longest time.

First thing that crossed my mind as soon as I read the question was to map the digits in a dictionary making the digits as keys/index for the string of letters they represent. For Ex- 2 is mapped to 'abc' and therefore in the mapping dictionary its represented as { 2:'abc' }. Doing this from 2-9, I get a dictionary that maps all letters of the alphabet to a single integer like keypad phone.

Then, the second thing that came to thought were the input cases, what if the user inputs 1 which does not map to any letter or a character that is not a number. This could be simply solved by encasing the main mapping code into a function and only executing the function when certain conditions are met which are that the input string does not contain any alphabets or other symbols and it does not contain the number 1. A simple if statement to before calling the function solves this.

Now, to the main part of the function the mapping of letters to their respective numbers and if there are more than one number then creating combinations of their mapped letters. So here we have two cases. Either only one number is input, which makes the job lot easier of just separating the string of letters into a list of single character string which will be our mapped list. This can easily be done by the `list()` function in python.

And if the input string has multiple numbers then after creating the mapped list with the first number we then need to map the letters from the other number to the already mapped letters and create combinations of the first letter of the mapped list with all the letters of the second number, then do the same with the second letter and so on. Then move to the third number in the input string and do this process all over again.

My first thought on the mapping was to make these cases defined first and then map the problem, so I checked the length of the input string if it was 1 then the mapped list would be created and that's all. And if the the length was more than 1 then I would do the something else. But while writing this code I realized that I was doing the same process twice with two cases that initially did the same thing.

So instead I decided to redo it and came up with the idea to first create an index variable which would hold the current index of the iteration being run and if the index was equal to the of the input string the loop for mapping would stop and our mapped list would be created. This was better than adding an extra block of code for just one number. I could just check the index.

Now that the cases are defined all I have to do is to map the digits and create combinations. For the first number its quite easy just take the string of digits and convert it into a list using the `list()` function and that's my mapped list.

Then for the numbers after the first number, I would first create a mapping list, which would contain the letters mapped to the digit as another list separate from the mapped list and first loop through the list using a for loop which would go into a nested for loop that loops through the mapping list, so with each iteration of the topmost for loop, the lower loop adds that letter in the mapped list to all the letters of the mapping list one by one separately and appends them to a result list. After the loop stops then the mapped list is set equal to the result list.

This process has to repeat for all the digits in the valid input string so we encase the whole mapping code in a while loop with condition that the index value is lesser than the length of the input string.

And at the end of the mapping function the mapped list is printed.

Explanation

```
keypad_map = {2:"abc", 3:"def", 4:"ghi", 5:"jkl", 6:"mno", 7:"pqrs", 8:"tuv", 9:"wxyz"}

digits = input("Enter Digits: ")
```

This is the declaration of the global variables for the program, the keypad_map as its name states is the map in the form of a dictionary that has the digits of the keypad as keys and the letters they represent as values. The digits variable is just the input string.

```
def keypad_mapper():

    i = 0

    mapped_lst = []
```

The function keypad_mapper contains the code that maps the digits to a list. The two variables defined are i the index variable and mapped_lst, this as the name suggest will be the final list containing and mapped combinations of the letters.

```
while i < len(digits):
    if i == 0:
        digit = int(digits[i])
        charset = keypad_map[digit]
        mapped_lst = list(charset)
```

We start with a while loop that checks if the index exceeds the length of digits. Then if the index is 0 meaning it is the first digit being mapped then first it the string get converted to a integer and stored in the variable digit, which is then used to create a string called charset that contains the string mapped by the digit. Then this charset is converted to a list using the `list()` function. The mapped list is set equal to this list. And the loop continues.

```
elif i ⩾ 0:
    digit = int(digits[i])
    charset = keypad_map[digit]
    mapping_lst = list(charset)
    temp_lst = []

    for j in mapped_lst:
        for k in mapping_lst:
            temp_lst.append(j+k)
    mapped_lst = temp_lst
```

If the index is greater than 0 meaning its not the first number but the second or third or any other number then first a mapping list is created from the same process as the mapping of the first number and another result list is created. Then we use a for loop to loop over the previously mapped list and take one element from the mapped list each time and append the characters of the mapping list to it one by one separately creating new strings and then appending them to the result list. At last the mapped list is set equal to the result list.

```
    i += 1
print(mapped_lst)
```

The index of the loop is increased and when the loop stops then the final mapped list is printed.

```
if digits.isdigit() and "1" not in digits:
    keypad_mapper()
else:
    print("Invalid: Only Digits 2-9 Should Be Given As Input!")
```

The keypad_mapper function is called only when all the characters in the input string are integers or digits, which is checked through the `.isdigit()` string function, and if the ‘1’ is not in the input string, which is checked through the not in operator. If these conditions are not met an appropriate error message is displayed.

Here is the look at the complete code:

```
# ACE Task Submission: Keypad Mapping

keypad_map = {2:"abc", 3:"def", 4:"ghi", 5:"jkl", 6:"mno", 7:"pqrs", 8:"tuv", 9:"wxyz"}

digits = input("Enter Digits: ")


def keypad_mapper():

    i = 0

    mapped_lst = []


    while i < len(digits):

        if i == 0:

            digit = int(digits[i])

            charset = keypad_map[digit]

            mapped_lst = list(charset)

        elif i ⩾ 0:

            digit = int(digits[i])

            charset = keypad_map[digit]

            mapping_lst = list(charset)

            temp_lst = []


            for j in mapped_lst:

                for k in mapping_lst:

                    temp_lst.append(j+k)

            mapped_lst = temp_lst

        i += 1

    print(mapped_lst)


if digits.isdigit() and "1" not in digits:

    keypad_mapper()

else:

    print("Invalid: Only Digits 2-9 Should Be Given As Input!")
```