Routing of telephone calls

BY

Saaish Bhonagiri

(saaishbhonagiri33@gmail.com)

GIT REPO: https://github.com/saaishb/Telephone-Operator

alaTest (ICSS) Challenge 2018

1.Introduction:

Some telephone operators have submitted their price lists including price per minute for different phone number prefixes. The price lists look like this:

Operator A:

1	0.9
268	5.1
46	0.17
4620	0.0
468	0.15
4631	0.15
4673	0.9
46732	1.1

Operator B:

1	0.92
44	0.5
46	0.2
467	1.0
48	1.2

And so on ….

The left column represents the telephone prefix (country + area code) and the right column represents the operator's price per minute for a number starting with that prefix. When several prefixes match the same number, the longest one should be used. If a price list does not include a certain prefix you cannot use that operator to dial numbers starting with that prefix.

2.AIM:

The goal with this exercise is to write a program that can handle ***any*** number of price lists (operators) and then can calculate which operator that is cheapest for a certain number. Each price list can have thousands of entries, but they will all fit together in memory. Telephone numbers should be inputted in the same format as in price lists. The challenge is to find the cheapest operator for that number.

3.Software Requirements

SNO	Requirement	Description	Test case
R1	Single Operator with single prices and single prefix	The script should be able to handle the single operator with single price list entry.	T1
R2	Multiple operators with single price and single prefix	The script should be able to handle multiple operators with single price list entry in all the operators	T2
R3	Single operator with multiple prices and prefixes	The script should be able to handle the single operator with multiple pricelist entries in the operator	T3

R4	Multiple operators with multiple prices and prefixes	The script should be able to handle the multiple operators with multiple pricelist entries.	T4
R5	Prefix doesn't match in the available operators	The script should be able to handle if the provided mobile number's prefix doesn't matches the prefixes in the available operators and returns a message in output.	T5
R6	Selecting the longest prefix matching multiple prefixes	The script should be able to handle if the provided mobile number's prefix matches the multiple prefixes in the price list, then the longest prefix that matches is selected	T6
R7	Obtaining the cheapest	The script should be able handle all the requirements and to be able to obtain the cheapest operator from a series of multiple operators with multiple number of entries to the price list.	T7

4.User Documentation:

The package consists of executable file 'tel_operator.py' which is the script used to run this application. The script itself has the function that handles all the requirements and the script also has the unit tests which will have seven test cases that are performed with various input parameters to the function and results will be a passing test message and 'OK' for all those seven test cases.

The use case of this project follows the series of steps as shown below.

> **Note**: The script requires **python3** package.

For Linux users:

- Extract the 'Telephone_Operator.tar.gz' file with 'tar xvzf'command.
- Go to the repository 'Telephone_operator' with 'cd' command.
- Then check for the file 'tel_operator.py' and run it using the command 'python3 tel_operator.py'

For Windows users:

- Extract the 'Telephone_Operator.zip' and go to the project folder and run it using 'python3 tel_operator.py'

Workflow of the application:

The script after successful compilation asks the user to enter the mobile number for which the user wants to calculate the cheapest operator.

The script checks if the user has entered the valid number else it will ask the user to enter the number again. This occurs recursively until the user enters the valid input number. The following format examples of input telephone numbers are valid.

- +4612346799
- +46-1-2-1-2
- 46738829
- 46-76-8468-78

The operators can be manipulated inside this file ‘tel_operator.py’ following the format:

operators = {'Operator A':{'prefix' : 'price', 'prefix2':'price2'.....},'Opearor B':{'prefix' : 'price', 'prefix2':'price2'.....}.....}

Then the output will be replied in the following manner:

- Your mobile number is: #input number is deisplayed
- Current operators are: #current operators are displayed
- The output is displayed based on the input number and available operators
- Then unit tests are performed and ‘OK’ is displayed (optional for user and do not depend on your input)

Example1 :

```
Enter the phone number:+467

Your mobile number is : 467

current operators are:

('Operator A', {'463': '1.3', '376': '2.2', '46732': '1.1'})

No operator has the matching price with your mobile number

#### UNIT TESTS #####

.......

----------------------------------------------------------------------
-
Ran 7 tests in 0.001s
```

```
OK
```

Example2:

```
Enter the phone number:+467983


Your mobile number is : 467983


current operators are:


('Operator A', {'467': '1.3', '4673': '333', '467325': '5300', '376': '
2.2', '46732': '1.1'})

('Operator B', {'46': '0.9', '4673': '1.0'})


('Operator C', {'4673258': '0.02', '463': '1.1', '467325': '0.01'})


The cheapest operator for the provided input number: 467983 is: Operato
r B with price: 0.9

#### UNIT TESTS #####


.......


----------------------------------------------------------------------
-

Ran 7 tests in 0.001s




OK
```

5.Developer Documentation:

The script has the two parts: The first part handles the functions that execute the logic and the second part has unit tests.

In the first part, the operators can be declared in the nested dictionaries with multiple price list entries for multiple operators. Then the input from the user is taken and stored in a variable. The number is first sent to 'check' function which validates the number and executes recursively until the user enters the correct format of mobile number. When the correct number is validated, the number and operators dictionary are sent into the 'cheap' function. The function iterates the nested way, for each individual operator it checks which one is the best suitable prefix and price value by comparing repetitively from the previous stored values. Then for all the suitable prefixes and price values from all the individual operators I the previous step are compared with each other if any for the available operators. They are compared in a similar way as in the nested for loop by utilizing the previous stored values and the cheapest operator for the provided input number is obtained.

The second part of the code consists of seven unit tests. These unit tests require 'unittests' python module. So, the module is imported. Each unit test tests takes various possible ways of input and the 'cheap' function is called and the output is verified. The individual test cases are explained more in Testing chapter

6.Testing

Test T1:

Purpose	To test Single Operator with single prices and single prefix
Requirement	R1

Operation	The single operator and single price list entry is declared, and the output is verified with a mobile number. operator_t1 = {'Operator A' : {'4678':'3.4'}} num = '467845687'
Expected Result	('Operator A','3.4')

Test T2:

Purpose	To test Multiple operators with single price and single prefix
Requirement	R2
Operation	The multiple operators and single price list entries for each is declared, and the output is verified with a mobile number. operator_t2 = {'Operator A' : {'46732': '1.1'}, 'Operator B':{'4673':'1.0'}} num = '46732456'
Expected Result	('Operator B','1.0')

Test T3:

Purpose	To test Single operator with multiple prices and prefixes

Requirement	R3
Operation	The single and multiple price list entries for it is declared, and the output is verified with a mobile number. operator_t3 = {'Operator A' : {'46732': '1.1','463':'1.3','376':'2.2'}} num = '46332456'
Expected Result	('Operator A','1.3')

Test T4:

Purpose	To test Multiple operators with multiple prices and prefixes
Requirement	R4
Operation	The multiple operators and multiple price list entries for each of them are declared, and the output is verified with a mobile number. operator_t4 = {'Operator A’: {'46732': '1.1','463':'1.3','376':'2.2'}, 'Operator B’: {'4673':'1.0','46':1.9} } num = '46332456'
Expected Result	('Operator A','1.3')

Test T5:

Purpose	To test if Prefix doesn’t match in the available operators
Requirement	R5
Operation	Enter a number with its prefix not present in the prefixes declared in the available operators and very the output. operator_t5 = {'Operator A' : {'46732': '1.1','463':'1.3','376':'2.2'}} num = '676332456'
Expected Result	('','') The output message will be printed as "No operator has the matching price with your mobile number"

Test T6:

Purpose	To test Selecting the longest prefix matching multiple prefixes
Requirement	R6
Operation	Provide an operator which has multiple prefixes for the provided input number, then the output is verified if the longest one is selected.

	operator_t6 = {'Operator A' : {'46732': '1.1','467':'1.3','376':'2.2','467325':'5300','4673':'333'}} num = '46732514443'
Expected Result	('Operator A','5300')

Test T7:

Purpose	To test for Obtaining the cheapest
Requirement	R7
Operation	Provide dictionaries with all the possible requirements ad the output is verified if the cheapest is being selected. operator_t7 = {'Operator A' : {'46732': '1.1','467':'1.3','376':'2.2','467325':'5300','4673':'333'},'Operator B':{'4673':'1.0','46':'0.9'},'Operator C':{'463':'1.1','467325':'0.01','4673258':'0.02'}} num = '467314443'
Expected Result	('Operator B','1.0')