1. You want to help your mom buy groceries within a budget. She needs to buy many things, but you want to tell her up to which items are within her budget.

Write a function BazaarList that takes **at least 1 parameter**. The first parameter will always be the budget, and the next parameters will all be tuples in this form: (item, price). **You can pass any number of tuples you want**. Your job is to make a dictionary that has the items as keys, and prices as values UP TILL the item which **does not cross the budget**. So if adding an item to the dictionary crosses the budget, then you will not add it. Return this dictionary.

[CO1,CO3, CO4, CO6]

[8 marks]

**Function call 1:**

print(BazaarList(50))

**Sample Output 1:**

{}

**Explanation:** As only a budget was given, there was nothing to add to the dictionary.

**Function call 2:**

print(BazaarList(600, ('eggs', 50), ('Chicken', 500), ('beef', 800)))

**Sample Output 2:**

{'eggs': 50, 'Chicken': 500}

**Explanation:** Budget given is 600, so the total cost of items that can be added to the dictionary is 600. When eggs are added, price becomes 50. When Chicken is added, price becomes 50 + 500 = 550. When Beef is added, price becomes 550 + 800 = 1430 which is more than 600, therefore beef cannot be added to the dictionary.

**Function call 3:**

Result = BazaarList(7500, ('Apples', 250), ('Oranges', 5200))

print(Result)

**Sample Output 2:**

{'Apples': 250, 'Oranges': 5200}

**Explanation:** The sum of both the items is 5450 which is less than the budget 7500, therefore both items were added to the dictionary.

1. Write a function that asks the user for an input, then counts the frequency of the characters that exist at prime indexes. Take user input inside the function. Store the frequencies in a list in this form: (character, frequency) and return this list. **You should not save duplicate tuples, so if a character and its frequency is already added, do not add it again (Keep in mind that the combination of (character,frequency) will be the same for each character, so you can use the in keyword to check this).** You are allowed to use the count function.

**Hint: Prime numbers are the numbers that are ONLY divisible by 1 and themself and 0 and 1 are not prime numbers. You need to find the possible prime indexes first based on the length of the string, then find the characters in those indexes. Final step will be calculating the count of each character in the original string.**

[CO1,CO3, CO4, CO6]

[5 marks]

**Function call : [ function call is same each time ]**

print(Function\_name())

**Sample Input 1:**

abcccddeeccff

**Sample Output 1:**

[('c', 5), ('d', 2), ('e', 2), ('f', 2)]

**Explanation:** The length of the input is 13, so the possible prime indexes are 2,3,5, 7, 11. The character at index 2 is c, and count of c is 5, so (c, 5) is added to the list. The character at index 3 is also c, and the count of c is obviously 5, so (c,5) already exists in the list and will not be added again. The characters at index 5, 7 and 11 are d, e and f, which have the counts 2, ,2 and 2 respectively, so (d, 2), , ('e', 2) and ('f', 2) are added to the list

**Sample Input 2:**

abcdefghij

**Sample Output 2:**

[('c', 1), ('d', 1), ('f', 1), ('h', 1)]

**Explanation:** The length of the input is 10, so the possible prime indexes are 2,3,5, 7. The character at index 2 is c, and count of c is 1, so (c, 1) is added to the list. The characters at index 3, 5, 7 are d, f and h, which have the counts 1,1, 1 respectively, so ('d', 1), ('f', 1), ('h', 1) are added to the list

1. Trace the output of the following code:

| **1** | **L1 = [0,1]** |
| --- | --- |
| **2** | **d1 = {'a': 2, 'c':5, 'd':73, 'e': 11, 'f':3, 'l':15, 'k':10}** |
| **3** | **L1 = d1** |
| **4** | **for i in L1:** |
| **5** | **if L1[i] % 3 == 0:** |
| **6** | **continue** |
| **7** | **else:** |
| **8** | **x = l1[i] \*7 - int(l1[i]/6) \* 5 // 2** |
| **9** | **L1[i] = x - 5\*\*3** |
| **10** | **print(l1[i])** |

[7 Marks]

|  |
| --- |
|  |
|  |
|  |
|  |