# More Exercises: Lists Advanced

Additional exercises for the [Python Fundamentals Course @SoftUni](https://softuni.bg/trainings/2442/python-fundamentals-september-2019). Submit your solutions in the SoftUni judge system at <https://judge.softuni.bg/Contests/1732>

***Note: All the exercises are excluded from your homework!***

## Messaging

You will be given a **list of numbers** and a **string**. For each element of the list you have to **calculate the sum of its digits** and take the **element, corresponding to that index from the text**. If the index is **greater than the length of the text**, start counting **from the beginning** (so that you always have a valid index). Then **you get that** **element** from the text, you must **remove the character** you have taken from it (so for the next index, the text will be with one character less).

### Example

|  |  |
| --- | --- |
| **Input** | **Output** |
| 9992 562 8933  This is some message for you | hey |

numbers=[x **for** x **in** input().split()]  
string=input()  
output=**''**indexes=[]  
**for** element **in** numbers:  
 sum=0  
 **for** i **in** element:  
 sum+=int(i)  
 indexes.append(sum)  
  
**for** index **in** indexes:  
 **while** index>=len(string):  
 index=index//len(string)  
 output+=string[index]  
 string2=string[:index]+string[index+1:]  
 string=string2  
  
print(output)

## Car Race

Write a program to calculate the **winner of a car race**. You will receive a **list of numbers**. Each element of the list represents the **time needed to pass through that step** (the index). There are going to be **two cars**. **One** of them **starts** from the **left side** and the **other one starts from the right** **side**. **The middle index of the list is the finish line**. The **number of elements** in the list **will always be odd**. Calculate **the total time for each racer to reach the finish**, which is the **middle of the list**, and **print the winner with his total time** (the **racer with less time**). If you have a **zero in the list**, you have to **reduce the time of the racer that reached it by 20%** (**from his current time**).

Print the result in the following format **"The winner is {left/right} with total time: {total time}".**

### Example

|  |  |
| --- | --- |
| **Input** | **Output** |
| 29 13 9 0 13 0 21 0 14 82 12 | The winner is left with total time: 53.8 |
| **Comment** | |
| The time of the left racer is (29 + 13 + 9) \* 0.8 (because of the zero) + 13 = 53.8  The time of the right racer is (82 + 12 + 14) \* 0.8 + 21 = 107.4  The winner is the left racer, so we print it | |

numbers = [int(x) **for** x **in** input().split()]  
left\_racer = 0  
right\_racer = 0  
**for** i **in** range(len(numbers) // 2):  
 **if** numbers[i] != 0:  
 left\_racer += numbers[i]  
 **else**:  
 left\_racer = left\_racer \* 8 / 10  
 **if** numbers[len(numbers) - i - 1] != 0:  
 right\_racer += numbers[len(numbers) - i - 1]  
 **else**:  
 right\_racer = right\_racer \* 8 / 10  
  
**if** left\_racer < right\_racer:  
 print(**f'The winner is left with total time: {**left\_racer**:.1f}'**)  
**else**:  
 print(**f'The winner is right with total time: {**right\_racer**:.1f}'**)

## Take/Skip Rope

Write a program, which reads a **string** and **skips** through it, extracting a **hidden message**. The algorithm you have to implement is as follows:

Let’s take the string “skipTest\_String044170” as an example.

Take every **digit** from the string and **store it** somewhere. After that, **remove** all the digits from the string. After this operation, you should have **two lists of items**: the **numbers list** and the **non-numbers list**:

* Numbers list: [0, 4, 4, 1, 7, 0]
* Non-numbers: [s, k, i, p, T, e, s, t, \_, S, t, r, i, n, g]

After that, take every digit in the **numbers list** and split it up into a **take list** and a **skip list**, depending on whether the digit is in an **even** or an **odd** index:

* Numbers list: [0, 4, 4, 1, 7, 0]
* Take list: [0, 4, 7]
* Skip list: [4, 1, 0]

Afterwards, **iterate** over both of the lists and **skip** {skipCount} characters from the **non-numbers list**, then **take** {takeCount} characters and store it in a **result string**. Note that the skipped characters are **summed up** as they go. The process would look like this on the aforementioned **non-numbers list**:

1. Take **0** characters  Taken: "", skip **4** characters (total **0**)  Skipped: "**skipTest\_String**" Result: ""
2. Take **4** characters Taken: "**Test"**, skip **1** characters (total **4**)  Skipped: "**skip**"  Result: "**Test**"
3. Take **7** characters Taken: "**String**", skip **0** characters (total **9**) Skipped: ""  Result: "**TestString**"

After that, just print the **result string** on the console.

### Input

* First line: The **encrypted** message as a **string**

### Output

* First line: The **decrypted** message as a **string**

### Constraints

* The count of digits in the input string will **always be even**.
* The encrypted message will contain any printable ASCII character.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| T2exs15ti23ng1\_3cT1h3e0\_Roppe | TestingTheRope |
| O{1ne1T2021wf312o13Th111xreve!!@! | OneTwoThree!!! |
| this forbidden mess of an age rating 0127504740 | hidden message |

data=input()  
number\_list=[]  
non\_numbers=[]  
**for** i **in** data:  
 **if** i.isdigit():  
 number\_list.append(int(i))  
 **else**:  
 non\_numbers.append(i)  
non\_numbers=**''**.join(non\_numbers)  
  
take\_list=[]  
skip\_list=[]  
**for** i **in** range(0,len(number\_list)-1,2):  
 take\_list.append(number\_list[i])  
 skip\_list.append(number\_list[i+1])  
  
result=**''**start\_pos=0  
**for** i **in** range(len(take\_list)):  
 *#take* result+=non\_numbers[start\_pos:start\_pos+take\_list[i]:]  
 *#skip* start\_pos+=take\_list[i]+skip\_list[i]  
  
print(result)

## Social Distribution

*A core idea of several left-wing ideologies is that the wealthiest should support the poorest, no matter what and that is exactly what you are called to do for this problem.*

On the first line you will be given the **population** (numbers separated by comma and space **", "**). On the second line you will be given the **minimum wealth**. You have to **distribute** the wealth, so that there is no part of the population that has **less than the minimum** wealth. To do that, you should always take wealth from the **wealthiest part of the population**. There will be cases, where the distribution will **not be possible**. In that case, print **"No equal distribution possible"**

### Example

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2, 3, 5, 15, 75  5 | [5, 5, 5, 15, 70] |
| 2, 3, 5, 15, 75  20 | [20, 20, 20, 20, 20] |
| 2, 3, 5, 45, 45  30 | No equal distribution possible |

population=list(map(**lambda** x:int(x),input().split(**', '**)))  
min\_wealth=int(input())  
  
**def** rich\_to\_poor(lista,mw):  
 i=0  
 **while** lista[i]<mw:  
 giving=mw-lista[i]  
 lista[i]=mw  
 lista[lista.index(max(lista))]-=giving  
 i+=1  
  
**if** len(population)\*min\_wealth>sum(population):  
 print(**'No equal distribution possible'**)  
**else**:  
 rich\_to\_poor(population,min\_wealth)  
 print(population)

## Kate's Way Out

*Kate is stuck into a maze, you have to help her to find her way out*

On the **first line** you will be given how many **rows** there are in the maze. On the **next n lines** you will be given the **maze itself**. Here is a legend for the maze:

* **"#"** - means a **wall**; Kate cannot go through there
* **" "** - means **empty** space; Kate can go through there
* **"k"** - the initial **position of Kate**; start looking for a way out from there

There are two options: Kate either gets out or not. If Kate **can get** out print the following:   
**"Kate got out in {number\_of\_moves} moves"**. Otherwise print: **"Kate cannot get out"**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 4  ######  ## k#  ## ###  ## ### | Kate got out in 5 moves |
| 5  ######  ## k#  ## ###  ######  ## ### | Kate cannot get out |
| 6  ######  # #  # ##  ## ###  #k ##  ###### |  |
| 6  # ####  # # k#  # ##  ## # #  ## ##  ### ## |  |
| 6  ###### #  # # #  # ## # #  # # # #  #k## #  ######## |  |
| 5  ######  # # k#  # ##  ## # #  ## ### |  |