# More Exercises: Lists Basics

Additional exercises for the [Python Fundamentals Course @SoftUni](https://softuni.bg/trainings/3368/python-fundamentals-may-2021).

Submit your solutions in the SoftUni judge system at <https://judge.softuni.bg/Contests/1726>

## Zeros to Back

Write a program that receives a **single string** (**integers** **separated** by a comma and space **", ")**, finds all the **zeros** and moves them **to the back** without messing up the other elements. Print the resulting **integer list**.

### Example

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1, 0, 1, 2, 0, 1, 3 | [1, 1, 2, 1, 3, 0, 0] |

## Big Numbers Lover

*You really like big numbers, so you always find a way to form the largest one from the numbers given to you.*

You will receive a single line containing **numbers** separated by a **single space**. Your task to **rearrange the given numbers to form the largest number possible.**

### Example

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3 3 0 34 5 9 | 9534330 |
| 1 2 3 | 321 |

## Messaging

On the **first line** you will receive a **sequence of numbers**, separated by a single space. On the **second line** you will receive a **string with chars**.

Your task is to write a program which sends a message only using chars from the given string. **Each** **char** the program adds to the message should be found by **its index**. The index you are looking for is the **sum of a number's digits from the sequence**. If the index is **greater than the length of the text**, continue counting **from the beginning** (so that you always have a valid index). When you find a char, you should **add it** to the message and **remove it** from the string. It means that for the next index, the text will be with one character less.

**Print** **the final message**.

### Example

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

## Car Race

Write a program which finds 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** **line** and **print the winner with his total time** (the **racer with less time**). If you have a **zero in the list**, you should **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}".**

The time should be **formatted** to the **first decimal point.**

### 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. | |

## Josephus Permutation

*This problem takes its name by arguably the most important event in the life of the ancient historian Josephus: according to his tale, he and his 40 soldiers were trapped in a cave by the Romans during a siege. Refusing to surrender to the enemy, they instead opted for mass suicide, with a twist: they formed a circle and proceeded to kill one man every three, until one last man was left (and that it was supposed to kill himself to end the act). Well, Josephus and another man were the last two and, as we now know every detail of the story, you may have correctly guessed that they did not exactly follow through the original idea.*

You are now to create a program that prints a **Josephus permutation**, receiving **two lines** of code (the list itself **(string** with elements separated **by a single space)** and a number **k**) as if they were in a circle and **counted out every k** places until none remained.

### Example

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comment** |
| 1 2 3 4 5 6 7  3 | [3,6,2,7,5,1,4] | [1,2,3,4,5,6,7] - initial sequence  [1,2,4,5,6,7] => 3 is counted out and goes into the result [3]  [1,2,4,5,7] => 6 is counted out and goes into the result [3,6]  [1,4,5,7] => 2 is counted out and goes into the result [3,6,2]  [1,4,5] => 7 is counted out and goes into the result [3,6,2,7]  [1,4] => 5 is counted out and goes into the result [3,6,2,7,5]  [4] => 1 is counted out and goes into the result [3,6,2,7,5,1]  [] => 4 is counted out and goes into the result [3,6,2,7,5,1,4] |

## Tic-Tac-Toe

You will receive a field of a tic-tac-toe game in **three lines** containing **numbers, separated by a single space**.

***Legend***:

* 0 - **empty** space
* 1 - **first** player move
* 2 - **second** player move

Find out who the **winner** is. If the **first** player **wins**,print **"First player won"**. If the **second** player **wins**, print **"Second player won"**. Otherwise, print **"Draw!"**.

### Example

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2 0 1  0 1 0  1 0 2 | First player won |
| 0 1 0  2 2 2  1 0 0 | Second player won |
| 1 0 2  0 1 2  1 2 0 | Draw! |