# More Exercises: Lists

Problems for exercise and homework for the ["C# Fundamentals" course @ SoftUni](https://softuni.bg/modules/57/tech-module-4-0)  
You can check your solutions here: [Judge](https://judge.softuni.bg/Contests/1300)

## 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).

After **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  This(4) is(2) some(4) message(7) for(3) you(3) |

## Car Race

Write a program to calculate the **winner of a car race**.

You will receive an **array of numbers**.

Each element of the array 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 array is the finish line**.

The **number of elements** in the array **will always be odd**.

Calculate **the total time for each racer to reach the finish**, which is the **middle of the array**, and **print the winner with his total time** (the **racer with less time**).

If you have a **zero in the array**, 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  21 0 14 82 12 0 29 13 9 0 13 | 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 | |

## 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 |

## \*Mixed up Lists

Write a program that **mixes up** **two lists** by some rules. You will receive **two lines of input**, each one being a **list of numbers**. The **rules** for mixing are:

* Start from the **beginning of the first** list and from the **ending of the second.**
* **Add** element **from the first** and element **from the second.**
* At the end there will always be a list, in which there are **2 elements remaining.**
* These elements will be the **range of the elements you need to print.**
* **Loop through the result list** and take **only the elements that fulfill the condition.**
* Print the elements **ordered in ascending** order and **separated by a space.**

### Example

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1 5 23 64 2 3 34 54 12  43 23 12 31 54 51 92 | 23 23 31 34 43 51 |
| **Comment** | |
| After looping through the two of the arrays we get:  1 92 5 51 23 54 64 31 2 12 3 23 34 43  The constrains are 54 and 12 (so we take only the numbers between them): 51 23 31 23 34 43  We print the result sorted | |

## \*Drum Set

Gabsy is Orgolt's Final Revenge charming drummer.

She has a drum set but the different drums have different origins – some she bought, some are gifts, so they are all with **different quality**.

Every day she practices on each of them, so she does damage and reduces the drum`s quality. Sometimes a drum brakes, so she needs to buy new one. Help her keep her drum set organized.

You will receive Gabsy's **savings**, the money she can spend on new drums. Next you will receive a **sequence** **of integers,** which represents the **initial quality** of each drumin Gabsy's drum set.

Until you receive the command **"Hit it again, Gabsy!"**, you will be receiving an integer: the **hit power** Gabsy applies **on each drum,** while practicing. When the power is applied, you should **decrease** the value of the drum's quality with the **current** **power**.

When a certain drum **reaches 0 quality**, it breaks.

Then Gabsy should buy a replacement.

She needs to buy the exact same model.

Therefore, its quality will be **the same as the initial quality** of the broken drum.

The price is calculated by the formula: {initialQuality} \* 3.

Gabsy will always replace her broken drums **until the moment she can no longer afford it**.

If she doesn't have enough money for a replacement, the broken drum is **removed** from the drum set.

When you receive the command **"Hit it again, Gabsy!"**,the program ends and you should print the current state of the drum set. On the second line you should print the **remaining money** in Gabsy's savings account.

### Input

* On the **first line** you will receive the **savings** – a floating-point number.
* On the **second line** you will recieve the **drum set**: a **sequence** of **integers**, **separated** by **spaces**.
* Until you receive the command **"Hit it again, Gabsy!",** you will be receiving **integers** – the hit power Gabsy applies on each drum.

### Output

* On the first line you should print **each drum** in the drum set, **separated** by **space**.
* Then you must print the **money** that are left on the **second line** in the format "Gabsy has {money left}lv.", formatted with two digits after the decimal point.

### Constraints

* The **savings – a floating-point number in the range [0.00, 10000.00]**
* The **quality of each drum in the drum set** – an integer in the range **[1, 1000]**.
* The **hit** **power** will be in the **range [0, 1000]**
* Allowed working **time** / **memory**: **100ms** / **16MB**.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comment** |
| 1000.00  58 65 33  11  12  18  10  Hit it again, Gabsy! | 7 14 23  Gabsy has 901.00lv. | DrumSet – 58 65 33.  Day 1: hit power applied = 11 => 47 54 22;  Day 2: hit power applied = 12 => 35 42 10;  Day 3: hit power applied = 18 => 17 24 -8;  The third drum breaks. But Gabsy has enough savings, so she replaces it => 17 24 33;  Day 4: hit power applied = 10 => 7 14 23;  We print the current state of the drum set and what's left in Gabsy's bank account. |
| 154.00  55 111 3 5 8 50  2  50  8  23  1  Hit it again, Gabsy! | 27 2 4 7  Gabsy has 10.00lv. | DrumSet – 55 111 3 5 8 50.  Day 1: hit power applied = 2 => 53 109 1 3 6 48;  Day 2: hit power applied = 50 => 3 59 0 0 0 0;  The third drum breaks. But Gabsy has enough savings, so she replaces it => 3 59 3 0 0 0;  The fourd drum breaks. But Gabsy has enough savings, so she replaces it => 3 59 3 5 8 50;  Day 3: hit power applied = 18 => 17 24 -8;  The third drum breaks. But Gabsy has enough savings, so she replaces it => 17 24 33;  Day 4: hit power applied = 10 => 7 14 23; |