# Exercise: Text Processing

Problems for exercise and homework 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/1740>

## Valid Usernames

Write a program which **reads usernames** on a **single** line (separated by **", "**) and **prints** all **valid usernames** on separate lines.

A valid username:

* has **length** between 3 and 16 characters inclusive
* **contains** only letters, numbers, hyphens, and underscores
* has **no redundant symbols** before, after or in between

username\_list = input().split(', ')  
  
valid\_usernames\_list = []  
is\_valid = False  
  
for x in username\_list:  
 if len(x) < 3 or len(x) > 16:  
 is\_valid = False  
 elif not x.isalnum() and '-' not in x and '\_' not in x:  
 is\_valid = False  
 else:  
 is\_valid = True  
 valid\_usernames\_list.append(x)  
  
print('\n'.join(valid\_usernames\_list))

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| sh, too\_long\_username, !lleg@l ch@rs, jeffbutt | jeffbutt |
| Jeff, john45, ab, cd, peter-ivanov, @smith | Jeff  John45  peter-ivanov |

## Character Multiplier

Create a program which receives **two strings** on a **single line** separated by a single **space** and prints the **sum** of their **multiplied character codes** as follows: multiply **str1[0]** with **str2[0]** and add the result to the total sum, then continue with the next two characters. If one of the strings is **longer** than the other, **add** the **remaining** character codes to the **total** **sum** without multiplication.

input\_string = input().split()  
  
str\_1 = input\_string[0]  
str\_2 = input\_string[1]  
  
min\_length = min(len(str\_1), len(str\_2))  
max\_length = max(len(str\_1), len(str\_2))  
current\_sum = 0  
total\_sum = 0  
  
for i in range(0, max\_length):  
  
 if i < len(str\_1) and i < len(str\_2):  
 current\_sum = ord(str\_1[i]) \* ord(str\_2[i])  
 total\_sum += current\_sum  
  
 elif i < len(str\_1) and i >= len(str\_2):  
 total\_sum += ord(str\_1[i])  
  
 elif i >= len(str\_1) and i < len(str\_2):  
 total\_sum += ord(str\_2[i])  
  
print(total\_sum)

Lecturer’s solution adopts a different logic - pastebin.com/aW9RR0zr :

input\_string = input().split()  
word\_1 = input\_string[0]  
word\_2 = input\_string[1]  
  
total\_sum = 0  
shorter\_word\_length = min(len(word\_1), len(word\_2))  
longer\_word\_length = max(len(word\_1), len(word\_2))  
  
# We process the shorter string first:  
for i in range(shorter\_word\_length):  
 total\_sum += ord(word\_1[i]) \* ord(word\_2[i])  
  
# If the strings have different lengths we have to process characters of the longer string  
for i in range(shorter\_word\_length, longer\_word\_length):  
 if len(word\_1) > len(word\_2):  
 curr\_word\_ch = word\_1[i]  
 else:  
 curr\_word\_ch = word\_2[i]  
  
 total\_sum += ord(curr\_word\_ch)  
  
print(total\_sum)

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| George Peter | 52114 |
| 123 522 | 7647 |
| a aaaa | 9700 |

## Extract File

Write a program which reads the path to a file and **subtracts** the **file name** and its **extension**.

file\_path\_splitted = input().split('\\')  
  
final\_file = file\_path\_splitted[-1]  
final\_file\_splitted = final\_file.split('.')  
file\_extension = final\_file\_splitted[1]  
file\_name = final\_file\_splitted[0]  
  
print(f"File name: {file\_name}")  
print(f"File extension: {file\_extension}")

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| C:\Internal\training-internal\Template.pptx | File name: Template  File extension: pptx |
| C:\Projects\Data-Structures\LinkedList.cs | File name: LinkedList  File extension: cs |

## Caesar Cipher

Write a program which returns an **encrypted version** of the same text. Encrypt the text by **replacing** **each character** whit the corresponding character **three** positions **forward in the ASCII table**. For example, **A** would be replaced with **D**, **B** would become **E**, and so on. Print the **encrypted** **text**.

sentence = input()  
  
reassigned\_ch = ''  
rearranged\_sentence = ''  
  
for x in sentence:  
 reassigned\_ch = ord(x) + 3  
 rearranged\_sentence += chr(reassigned\_ch)  
  
print(rearranged\_sentence)

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Programming is cool! | Surjudpplqj#lv#frro$ |
| One year has 365 days. | Rqh#|hdu#kdv#698#gd|v1 |

1. **Emoticon Finder**

Find all emoticons in the text. An emoticon **always starts with ":"** and is followed by a **symbol.**  
The input will be provided as a **single string.**

text = input()  
  
for i in range(len(text)):  
 if text[i] == ':':  
 print(f"{text[i]}{text[i+1]}")

**Example**

|  |  |
| --- | --- |
| **Input** | **Output** |
| There are so many emoticons nowadays :P. I have many ideas :O what input to place here :) | :P  :O  :) |

## Replace Repeating Chars

Write a program which reads a string from the console and **replaces** any **sequence of the same letters** with a **single** **corresponding letter**.

input\_string\_repeating\_chars = input()  
  
repeating\_char = ''  
non\_repeating\_char = ''  
string\_no\_repeating\_chars = []  
  
for i in range(0, len(input\_string\_repeating\_chars)-1):  
  
 if input\_string\_repeating\_chars[i] == input\_string\_repeating\_chars[i+1]:  
 repeating\_char = input\_string\_repeating\_chars[i]  
  
 else:  
 non\_repeating\_char = input\_string\_repeating\_chars[i]  
 string\_no\_repeating\_chars.append(input\_string\_repeating\_chars[i])  
  
  
string\_no\_repeating\_chars.append(input\_string\_repeating\_chars[-1])  
print(''.join(string\_no\_repeating\_chars))

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| aaaaabbbbbcdddeeeedssaa | abcdedsa |
| qqqwerqwecccwd | qwerqwecwd |

## String Explosion

Explosions are marked with **'**>**'**. Immediately after the mark, there will be an **integer** x, which signifies the **strength** of the explosion. You should **remove** x **characters**,starting **after** the punch **character** (**'**>**'**). If you find **another** explosion mark (**'**>**'**) while you are deleting characters, you should **add** the **strength** to your **previous** **explosion**. You should **not** delete the **explosion** character – **'**>**'**.

When all characters are processed, **print** the final string.

### Constraints

* You will **always** receive a **strength** for the punches
* The path will consist only of letters from the **Latin** **alphabet**, **integers** and the char **'**>**'**
* The strength of the punches will be in the interval [0…9]

input\_string = input()  
  
rearranged\_string = ''  
explosion\_power = 0  
  
for i in range(len(input\_string)):  
 if input\_string[i] == '>':  
 rearranged\_string += '>'  
 explosion\_power += int(input\_string[i+1])  
 else:  
 if explosion\_power > 0:  
 explosion\_power -= 1  
 else:  
 rearranged\_string += input\_string[i]  
  
print(rearranged\_string)

A second solutions, partly by RS finalised after a few hours of struggling:

string\_input = input()  
  
rearranged\_str = ''  
explosion = 0  
  
for i in range(len(string\_input)):  
 if string\_input[i] != '>':  
 if explosion == 0:  
 rearranged\_str += string\_input[i]  
 else:  
 explosion -= 1  
  
 elif string\_input[i] == '>':  
 explosion += int(string\_input[i + 1])  
 rearranged\_str += '>'  
  
print(rearranged\_str)

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| abv>1>1>2>2asdasd | abv>>>>dasd | 1st explosion is at index **3** and it is with **strength** of **1**. We delete **only** the **digit** **after** the explosion character. The string will look like this: abv>>1>2>2asdasd  2nd explosion is with strength **one** and the string transforms to this: abv>>>2>2asdasd  3rd explosion is now with strength of 2. We delete the digit and we find **another** explosion. At this point the string looks like this: abv>>>>2asdasd.  4th explosion is with strength **2**. We have **1** strength **left** from the previous explosion, we **add** the strength of the **current** explosion to what is **left** and that adds up to a **total** strength of **3**. We **delete** the next **three** **characters** and we **receive** the **string** abv>>>>dasd  We do **not** have **any more explosions** and we print the result: abv>>>>dasd |
| pesho>2sis>1a>2akarate>4hexmaster | pesho>is>a>karate>master |  |

## \*Letters Change Numbers

Nakov likes Math. But he also likes the English alphabet a lot. He invented a game with numbers and letters from the **English** alphabet. The game was simple. You get a string consisting of a **number between two letters**. Depending on whether the letter was in front of the number or after it you would perform different mathematical operations on the number to achieve the result.

**First** you start with the letter **before** the number.

* If it's **uppercase** you **divide** the number by the letter's **position** in the alphabet.
* If it's **lowercase** you **multiply** the number with the letter's **position** in the alphabet.

**Then** you move to the **letter after** the number.

* If it's **uppercase** you **subtract** its position from the resulted number.
* If it's **lowercase** you **add** its position to the resulted number.

But the game became too easy for Nakov really quick. He decided to complicate it a bit by doing the same but with **multiple** strings keeping track of only the **total sum** of all results. Once he started to solve this with more strings and bigger numbers it became quite hard to do it only in his mind. So he kindly asks you to write a program that **calculates the sum of all numbers after the operations on each number have been done**.

**For example**, you are given the sequence **"**A12b s17G**"**:

We have two strings – "A12b" and "s17G". We do the operations on each and sum them. We start with the letter before the number on the first string. **A is Uppercase** and its position in the alphabet is **1**. So we divide the number 12 with the position 1 **(12/1 = 12)**. Then we move to the letter after the number. **b is lowercase** and its position is 2. So we add 2 to the resulted number **(12+2=14)**. Similarly for the second string **s is lowercase** and its position is 19 so we multiply it with the number **(17\*19 = 323)**. Then we have Uppercase G with position 7, so we subtract it from the resulted number **(323 – 7 = 316)**. Finally, we sum the 2 results and we get **14 + 316=330**.

### Input

The input comes from the console as a **single line, holding the sequence of strings**. Strings are separated by **one or more white spaces**.

The input data will always be valid and in the format described. There is no need to check it explicitly.

### Output

Print at the console a single number: the **total sum of all processed numbers** rounded up to **two digits** after the decimal separator.

### Constraints

The **count** of the strings will be in the range [1 … 10]**.**

* The numbers between the letters will be integers in range [1 … 2 147 483 647]**.**
* Time limit: 0.3 sec. Memory limit: 16 MB.

#RS solution Letters Change Numbers, Text Processing Exercise, Jul 2021  
input\_list = input().split()  
  
result = 0  
  
for x in input\_list:  
 number = int(x[1:-1])  
 if x[0].isupper():  
 # If uppercase, divide the number by the letter's position.  
 result += number / (ord(x[0])-64)  
 elif x[0].islower():  
 # If lowercase, multiply the number with the letter's position.  
 result += number \* (ord(x[0]) - 96)  
  
 if x[-1].isupper():  
 # If uppercase, subtract its position from the resulted number.  
 result -= (ord(x[-1]) - 64)  
 elif x[-1].islower():  
 # If lowercase, add its position to the resulted number.  
 result += (ord(x[-1]) - 96)  
  
print(f"{result:.2f}")

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| A12b s17G | 330.00 | 12/1=12, 12+2=14, 17\*19=323, 323–7=316, **14+316=330** |
| P34562Z q2576f H456z | 46015.12 |  |
| a1A | 0.00 |  |

## \*Rage Quit

Every gamer knows what rage-quitting means. It’s basically when you’re just not good enough and you blame everybody else for losing a game. You press the CAPS LOCK key on the keyboard and flood the chat with gibberish to show your frustration.

Chochko is a gamer, and a bad one at that. He asks for your help; he wants to be the most annoying kid in his team, so when he rage-quits he wants something truly spectacular. He’ll give you **a series of strings followed by non-negative numbers**, e.g. "a3"; you need to print on the console **each string repeated N times**; **convert the letters to uppercase beforehand**. In the example, you need to write back "AAA".

On the output, print first a statistic of the **number of unique symbols** used (the casing of letters is irrelevant, meaning that 'a' and 'A' are the same); the format shoud be "Unique symbols used {0}". Then, **print the rage message** itself.

The **strings and numbers will not be separated by anything**. The input will always start with a string and for each string there will be a corresponding number. The entire input will be given on a **single line**; Chochko is too lazy to make your job easier.

### Input

* The input data should be read from the console.
* It consists of a single line holding a series of **string-number sequences**.
* The input data will always be valid and in the format described. There is no need to check it explicitly.

### Output

* The output should be printed on the console. It should consist of **exactly two lines**.
* On the first line, print the **number of unique symbols used** in the message.
* On the second line, print the **resulting rage message** itself.

### Constraints

* The count of **string-number pairs** will be in the range [1 … 20 000].
* Each string will contain any character **except digits**. The **length** of each string will be in the range [1 … 20].
* The **repeat count** for each string will be an integer in the range [0 … 20].
* Allowed working time for your program: 0.3 seconds. Allowed memory: 64 MB.

# RS solution Rage Quit, Text processing exercise, Jul 2021

import re  
  
input\_string = input().upper()  
  
splitted\_input = re.split(r"(\d+)", input\_string)  
  
result\_list = []  
  
for i in range(0, len(splitted\_input)-1, 2):  
 string\_to\_repeat = splitted\_input[i]  
 repeat\_times = int(splitted\_input[i+1])  
 result\_list.append(string\_to\_repeat \* repeat\_times)  
  
result\_to\_print = ''.join(result\_list)  
unique\_characters = len(set(result\_to\_print))  
  
print(f"Unique symbols used: {unique\_characters}")  
print(result\_to\_print)

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| a3 | Unique symbols used: 1  AAA | We have just one string-number pair. The symbol is 'a', convert it to uppercase and repeat 3 times: AAA.  Only one symbol is used ('A'). |
| aSd2&5s@1 | Unique symbols used: 5  ASDASD&&&&&S@ | "aSd" is converted to "ASD" and repeated twice; "&" is repeated 5 times; "s@" is converted to "S@" and repeated once.  5 symbols are used: 'A', 'S', 'D', '&' and '@'. |

## \*Winning Ticket

Lottery is exciting. What is not, is checking a million tickets for winnings only by hand. So, you are given the task to create a program which automatically checks if a ticket is a winner.

You are given a **collection of tickets separated by commas and spaces**. You need to check every one of them if it has a winning combination of symbols.

**A valid ticket should have exactly 20 characters**. The winning symbols are '**@**', '**#**', '**$**' and '**^**'. But in order for a ticket to be a winner the symbol should uninterruptedly repeat for at least **6 times** in both the **tickets left half** and the **tickets right half**.

For example, a valid winning ticket should be something like this:

"Cash$$$$$$Ca$$$$$$sh"

The left half "Cash$$$$$$" contains "$$$$$$", which is also contained in the tickets right half "Ca$$$$$$sh". A winning ticket should contain symbols repeating up to 10 times in both halves, which is considered a Jackpot (for example: "$$$$$$$$$$$$$$$$$$$$").

**Input**

The input will be read from the console. The input consists of a **single line** containing all tickets **separated by commas and one or more white spaces** in the format:

* "{ticket}, {ticket}, … {ticket}"

**Output**

Print the result for every ticket in the order of their appearance, each on a separate line in the format:

* **Invalid ticket -** "invalid ticket"
* **No match -** "ticket "{ticket}" - no match"
* **Match with length 6 to 9 -** "ticket "{ticket}" - {match length}{match symbol}"
* **Match with length 10 -** "ticket "{ticket}" - {match length}{match symbol} Jackpot!"

**Constrains**

* Number of tickets will be in range [0 … 100]

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| Cash$$$$$$Ca$$$$$$sh | ticket "Cash$$$$$$Ca$$$$$$sh" - 6$ |
| $$$$$$$$$$$$$$$$$$$$, aabb , th@@@@@@eemo@@@@@@ey | ticket "$$$$$$$$$$$$$$$$$$$$" - 10$ Jackpot!  invalid ticket  ticket "th@@@@@@eemo@@@@@@ey" - 6@ |
| validticketnomatch:( | ticket "validticketnomatch:(" - no match |

Pastebin.com/44wKJp7Y

# Function that returns longest subsequence of string  
def longest\_symbol\_subsequence(text: str, symbol: str):  
 longest\_subsequence = 0  
  
 for i in range(len(text)):  
 curr\_ch = text[i]  
 current\_subsequence = 0  
  
 if curr\_ch == symbol:  
 current\_subsequence += 1  
  
 for j in range(i + 1, len(text)):  
 next\_ch = text[j]  
  
 if next\_ch == symbol:  
 current\_subsequence += 1  
 else:  
 if current\_subsequence > longest\_subsequence:  
 longest\_subsequence = current\_subsequence  
 break  
  
 return longest\_subsequence  
  
  
# Demo  
# print(longest\_symbol\_subsequence('@@@asd@@@@ad@@', '@'))  
  
tickets = [t.strip() for t in input().split(', ') if not t.isspace()]  
  
# 01. @@@@@@  
winning\_combo\_at = '@' \* 6  
# 02. ######  
winning\_combo\_hash = '#' \* 6  
# 03. $$$$$$  
winning\_combo\_dollar = '$' \* 6  
# 04. ^^^^^^  
winning\_combo\_circumflex = '^' \* 6  
  
for ticket in tickets:  
 if len(ticket) != 20:  
 # This way I insure that the ticket will always be 20 characters length  
 print('invalid ticket')  
 continue  
  
 left\_half = ticket[:10]  
 right\_half = ticket[10:]  
  
 match\_symbol = ''  
  
 # There is at least 6 symbols combo  
 if winning\_combo\_at in left\_half and winning\_combo\_at in right\_half:  
 match\_symbol = '@'  
 elif winning\_combo\_hash in left\_half and winning\_combo\_hash in right\_half:  
 match\_symbol = '#'  
 elif winning\_combo\_dollar in left\_half and winning\_combo\_dollar in right\_half:  
 match\_symbol = '$'  
 elif winning\_combo\_circumflex in left\_half and winning\_combo\_circumflex in right\_half:  
 match\_symbol = '^'  
 else:  
 print(f'ticket "{ticket}" - no match')  
 continue  
  
 # Here we have a matching ticket  
 left\_matches = left\_half.count(match\_symbol)  
 right\_matches = right\_half.count(match\_symbol)  
  
 # In case we have different matches length in left and right half, we take the min  
 min\_matches = min(left\_matches, right\_matches)  
  
 if min\_matches == 10:  
 # 10 matches we have - There is a Jackpot!!!  
 print(f'ticket "{ticket}" - {min\_matches}{match\_symbol} Jackpot!')  
 else:  
 print(f'ticket "{ticket}" - {min\_matches}{match\_symbol}')