Strings Processing- Exercises

1. Reverse Strings

You will be given a series of strings until you receive an "**end**" command. Write a program that reverses strings and prints each pair on a separate line in the format "**{word} = {reversed word}**".

Input	Output

helLo
bottle
end
	helLo = oLleh
	bottle = elttob

Dog
caT
chAir
end
	Dog = goD
	caT = Tac
	chAir = riAhc

2. Repeat Strings

Write a Program That Reads an Array of Strings. Each String is Repeated N Times, Where N is the length of the String. Print the Concatenated String.

Input	Output
hi abc add	hihiabcabcabcaddaddadd
work	workworkworkwork
ball	ballballballball

3. Substring

On the first line, you will receive a string. On the second line, you will receive a second string. Write a program that removes all the occurrences of the first String in the second until there is no match. At the end, print the remaining String.

Examples

Input	Output	Comment
ice oicepiciceek	opk	We remove ice once, and we get "opiciceek" We match "ice" one more time, and we get "opicek" There is one more match. The result is "opk"
e fixture	fixtur	

4. Text Filter

Write a program that takes a **text** and a **string of banned words**. All words included in the ban list should be replaced with **asterisks** "*****", equal to the word's length. The entries in the ban list will be separated by a **comma** and **space** "**, **".

The ban list should be entered on the first input line and the text on the second input line.

Examples

Input	Output
Linux, Windows	It is not *****, it is GNU/*****. ***** is merely the kernel, while GNU adds the

It is not Linux, it is GNU/Linux. Linux is merely the kernel, while GNU adds the functionality. Therefore, we owe it to them by calling the OS GNU/Linux! Sincerely, a Windows client	functionality. Therefore, we owe it to them by calling the OS GNU/*****! Sincerely, a ******* client
computer, programming, set In computer programming, an application programming interface (API) is a set of subroutine definitions, communication protocols, and tools for building software.	In ******** ***********, an application *********** interface (API) is a *** of subroutine definitions, communication protocols, and tools for building software.

5. Digits, Letters and Other

Write a program that receives a single string and on the first line prints all the digits, on the second – all the letters, and on the third – all the other characters. There will always be at least one digit, one letter, and another character.

Input	Output
Agd#53Dfg^&4F53	53453 AgdDfgF #^&
a1!	1 a !

- Read the input.
- Use a loop to iterate through all characters in the text. If the char is a digit, print it, otherwise, ignore it.
- Do the same for the letters and other chars.
  - Find something like **isDigit** method for the letters.

6. Valid Usernames

Write a program that reads usernames on a single line (joined by "**,** ") and prints all valid usernames.

A valid username is:

- Has a **length** of between 3 and 16 characters.
- **It contains** only letters, numbers, hyphens, and underscores.

Input	Output

sh, too_long_username, !lleg@l ch@rs, jefferson	jefferson
Jeff, john45, ab, cd, peter-ivanov, @smith	Jeff John45 peter-ivanov

7. Extract File

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

Input	**Output**
C:\home\academy\presentation.pptx	File name: presentation File extension: pptx
C:\Projects\website\some.file.jar	File name: some.file File extension: jar

8. Caesar Cipher

Write a program that returns an encrypted version of the same text. Encrypt the text by shifting each character with three positions forward. For example, A would be replaced by D, B would become E, and so on. Print the encrypted text.

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

9. Multiply Big Number

You are given two lines – the first one can be a really big number (0 to 10^{50}). The second one will be a single-digit number (0 to 9). You must display the product of these numbers.

Note: do not use the **BigInteger** class.

Input	**Outpu t**		**Input**	**Output**		**Input**	**Output**
23 2	46		9999 9	89991		923847238931983192462832102 4	3695388955727932769851328408

10. Replace Repeating Chars

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

Input	Output
aaaaabbbbbcdddeeeedssaa	abcdedsa
qqqwerqwecccwd	qwerqwecwd

11. Extract Person Information

Write a program that reads **lines** of strings and extracts the **name** and **age** of a given person. The person's name will be **between** "**@**" and "**|**". The person's **age** will be **between** "**#**" and "*****".

Example: "**Hello my name is @Peter| and I am #20* years old.**"

For each found name and age, **print** a line in the following format **"{name} is {age} years old."**

Input	Output
2 **Here is a name @George| and an age #18*** **Another name @Billy| #35* is his age**	George is 18 years old. Billy is 35 years old.
3 **random name @lilly| random digits #5* age** **@Marry| with age #19*** **here Comes @Garry| he is #48* years old**	lilly is 5 years old. Marry is 19 years old. Garry is 48 years old.

12. Ascii Sumator

Write a program that prints a **sum of all characters between two given characters** (their **ASCII code**). In the **first line,** you will get a **character**. In the **second line,** you get **another character**. On the **last line,** you get a **random string**. Find all the characters **between the two given** and **print their ASCII sum**.

Input	Output
.	363

@ **dsg12gr5653feee5**	
? **E** **@ABCEF**	262

13. *Morse Code Translator

Write a program that translates messages from **Morse code to English** (**capital letters). Use** this page to help you (**without the numbers**). The words will be separated by a **space (' ')**. There will be a "**|**" character which you should **replace with ' '** (space).

Examples

Input	Output
.. | -- .- -.. . | -.-- --- ..- | .-- .-. .. - . | .- | .-.. --- -. --. | -.-. --- -.. .	I MADE YOU WRITE A LONG CODE
.. | .... --- .--. . | -.-- --- ..- | .- .-. . | -. --- - | -- .- -..	I HOPE YOU ARE NOT MAD

14. HTML

You will receive **3 lines** of input. On the **first line,** you will receive a **title of an article**. On the **next line,** you will receive the **content of that article**. On the next **n** lines, until you receive "**end of comments**", you will get the **comments about the article**. Print the **whole information in HTML format**. The **title** should be in "**h1**" **tag (<h1></h1>);** the **content** in **article tag (<article></article>);** each **comment** should be in **div tag (<div></div>).** For more clarification, see the example below

Input	Output
Article Some content of the article some comment more comment last comment end of comments	<h1> Article </h1> <article> Some content of the article </article> <div>

	some comment </div> <div> more comment </div> <div> last comment </div>
The Reckoning John Grisham’s The Reckoning is the master storytellers most powerful, surprising, and accomplished novel yet. some comment1 more comment2 last comment3 end of comments	<h1> The Reckoning </h1> <article> John Grisham’s The Reckoning is the master storytellers most powerful, surprising, and accomplished novel yet. </article> <div> some comment1 </div> <div> more comment2 </div> <div> last comment3 </div>

15. Letter

You will receive an **array**, which holds the **string** and **another array**.

The string is a letter which has a few **holes**, you must fill with **strings from the array** you receive at the second index.

If the **length** of the hole is **4** you must **replace** it with **string** with the **same length** and so on…

Input
'Hi, grandma! I\'m so _____ to write to you. _______ the winter vacation, so many ________ things happened. My dad bought me a sled. Mom started a new job as a ___________. My brother\'s ankle is _________, and now it bothers me even more. Every night Mom cooks ____ on your recipe because it is the most delicious. I hope this year Santa will ______ me a robot.', ['pie', 'bring', 'glad', 'During', 'amazing', 'pharmacist', 'sprained']
Output
Hi, grandma! I'm so glad to write to you. During the winter vacation, so many amazing things happened. My dad bought me a sled. Mom started a new job as a pharmacist. My brother's ankle is sprained, and now it bothers me even more. Every night Mom cooks pie on your recipe because it is the most delicious. I hope this year Santa will bring me a robot.

16. Match Full Name

Write a Java Program to **match full names** from a list of names and **print** them on the console.

Writing the Regular Expression

First, write a regular expression to match a valid full name, according to these conditions:

- A valid full name has the following characteristics:
    - It consists of **two words**.
    - Each word **starts** with a **capital letter**.
    - After the first letter, it **only contains lowercase letters afterward**.
    - **Each** of the **two words** should be **at least two letters long**.
    - The **two words** are **separated** by a **single space**.

To help you out, we've outlined several steps:

1. Use an online regex tester like https://regex101.com/
2. Check out how to use **character sets** (denoted with square brackets - '**[]**')
3. Specify that you want **two words** with a space between them (the **space character ' '**, and **not** any whitespace symbol)
4. For each word, specify that it should begin with an uppercase letter using a **character set**. The desired characters are in a range – **from** '**A**' **to** '**Z**'.

5. For each word, specify that what follows the first letter are only **lowercase letters**, one or more – use another character set and the correct **quantifier**.

6. To prevent letters' capture across new lines, put "**\b**" at the beginning and the end of your regex. This will ensure that what precedes and what follows the match is a word boundary (like a new line).

Input
Ivan Ivanov, Ivan ivanov, ivan Ivanov, IVan Ivanov, Georgi Georgiev
Output
Ivan Ivanov Georgi Georgiev
Input
peter georgiev, peter Georgiev, Peter GeoRgiev, PEter GEorgiev, Peter Georgiev, Anna Petrova
Output
Peter Georgiev Anna Petrova

17. Match Phone Number

Write a regular expression to match a **valid phone number** from **Sofia**. After you find all **valid phones**, **print** them on the console, separated by a **comma and a space ", "**.

Compose the Regular Expression

A valid number has the following characteristics:

- It starts with "**+359**".
- Then, it is followed by the area code (always **2**).
- After that, it's followed by the **number** itself:
    - The number consists of **7 digits** (separated into **two groups** of **3** and **4 digits,** respectively).
- The different **parts** are **separated** by **either a space or a hyphen** ('**-**').

You can use the following RegEx properties to **help** with the matching:

- Use **quantifiers** to match a **specific number** of **digits.**
- Use a **capturing group** to ensure the delimiter is **only one of the allowed characters (space or hyphen)** and **not** a **combination** of both (e.g., **+359 2-111 111** has **mixed delimiters**, it is **invalid**). Use a **group back reference** to achieve this.
- Add a **word boundary** at the **end** of the match to avoid **partial matches** (the last example is on the right-hand side).

- Ensure that before the **'+'** sign, there is either a **space** or the **beginning of the string**.

Input
+359 2 222 2222,359-2-222-2222, +359/2/222/2222, +359-2 222 2222 +359 2-222-2222, +359-2-222-222, +359-2-222-22222 +359-2-222-2222
Output
+359 2 222 2222, +359-2-222-2222
Input
+359 2 222 2222,359-2-222-2222, +359/2/222/2222, +359-2 222 2222 +359 2-222-2222, +359-2-222-222, +359-2-222-22222 +359-2-222-2222
Output
+359 2 222 2222, +359-2-222-2222

18. Pascal-Case Splitter

You will receive a **single string**.

This string is written in **PascalCase** format. Your task here is to split this string by **every word** in it.

Print them joined by **comma** and **space.**

Input	**Output**
'SplitMeIfYouCan'	Split, Me, If, You, Can
'HoldTheDoor'	Hold, The, Door
'ThisIsSoAnnoyingToDo'	This, Is, So, Annoying, To, Do

19. Match Dates

Write a program that matches a date in the format "**dd{separator}MMM{separator}yyyy**". Use **named capturing groups** in your regular expression.

Compose the Regular Expression

Every valid date has the following characteristics:

- Always starts with **two digits**, followed by a **separator.**
- After that, it has **one uppercase** and **two lowercase** letters (e.g., **Jan**, **Mar**).
- After that, it has a **separator** and **exactly 4 digits** (for the year).

- The separator could be either of three things: a period ("**.**"), a hyphen ("**-**") or a forward-slash ("**/**").
- The separator needs to be **the same** for the whole date (e.g., 13**.**03**.**2016 is valid, 13**.**03**/**2016 is **NOT**). Use a **group back reference** to check for this.

Input
13/Jul/1928, 10-Nov-1934, , 01/Jan-1951,25.Dec.1937, 23#09#1973, 1/Feb/2016
Output
Day: 13, Month: Jul, Year: 1928 Day: 10, Month: Nov, Year: 1934 Day: 25, Month: Dec, Year: 1937
Input
01/Jan-1951 29/Feb/2024 1/Jan-1951 27-Feb-2007 1/Jan-1951 1/Mar/2016 23/october/197
Output
Day: 29, Month: Feb, Year: 2024 Day: 27, Month: Feb, Year: 2007

20. Star Battles Enigma

The war is at its peak, but you, young Padawan, can turn the tides with your programming skills. You are tasked to create a program to **decrypt** the messages of The Order and prevent the death of hundreds of lives.

You will receive several messages, which are **encrypted** using the legendary star enigma. You should **decrypt the messages** following these rules:

To properly decrypt a message, you should **count all the letters [s, t, a, r]** – **case insensitive** and **remove** the count from the **current ASCII value of each symbol** of the encrypted message.

After decryption:

Each message should have a **planet name, population, attack type ('A', as an attack or 'D', as destruction), and soldier count.**

The planet's name **starts after '@'** and contains **only letters from the Latin alphabet**.

The planet population **starts after ':'** and is an **Integer**;

The attack type may be **"A"(attack) or "D"(destruction)** and must be **surrounded by "!"** (Exclamation mark).

The **soldier count** starts after **"->"** and should be an Integer.

The order in the message should be: **planet name -> planet population -> attack type -> soldier count.** Each part can be separated from the others by **any character except: '@', '-', '!', ':' and '>'.**

Input / Constraints

- The **first line holds n** – the number of **messages** – **integer in the range [1…100].**
- On the next **n** lines, you will be receiving encrypted messages.

Output

After decrypting all messages, you should print the decrypted information in the following format:

First print the attacked planets, then the destroyed planets.
"**Attacked planets: {attackedPlanetsCount}**"
"**-> {planetName}**"
"**Destroyed planets: {destroyedPlanetsCount}**"
"**-> {planetName}**"

Input	Output	Comments
2 **STCDoghudd4=63333$D$0A53333** **EHfsytsnhf?8555&I&2C9555SR**	Attacked planets: 1 -> Alderaa Destroyed planets: 1 -> Cantonica	We receive two messages, and to decrypt them, we calculate the key: The first message has decryption key 3. So we subtract from each character's code 3. **PQ@Alderaa1:30000!A!->20000** The second message has key 5. **@Cantonica:3000!D!->4000NM** **Both messages are valid and** contain planet, population, attack type, and soldier count.
3 **tt(''DGsvywgerx>5444444444%H%1B9444** **GQhrr|A977777(H(TTTT** **EHfsytsnhf?8555&I&2C9555SR**	Attacked planets: 0 Destroyed planets: 2 -> Coruscant -> Cantonica	We receive three messages. Message one is decrypted with key 4: **pp$##@Coruscant:1000000000!D!->5000** Message two is decrypted with key 7: **@Jakku:200000!A!MMMM** This is the **invalid message**, missing soldier count, so we continue.

		The third message has key 5. **@Cantonica:3000!D!->4000NM**

"It's a trap!" – Admiral Ackbar