Exercises: Files, Directories and Exceptions

Problems for exercises and homework for the "Python-Fundamentals-Jan-2018" course @ SoftUni.

This exercise does **NOT** have a **Judge Contest**. You will have to **test** every problem **locally**.

1. Odd Lines

Write a program that reads a text file and writes its every **odd** line in another file. Line numbers starts from 0.

Examples

Input.txt	Output.txt
Two house holds, both a like in dignity, In fair Verona, where we lay our scene, From ancient grudge break to new mutiny, Where civil blood makes civil hands unclean. From forth the fatal loins of these two foes A pair of star-cross'd lovers take their life; Whose miss adventured piteous overthrows Do with their death bury their parent's strife.	In fair Verona, where we lay our scene, Where civil blood makes civil hands unclean. A pair of star-cross'd lovers take their life; Do with their death bury their parent's strife.

2. Line Numbers

Write a program that reads a text file and inserts line numbers in front of each of its lines. The result should be written to another text file.

Examples

Input.txt	Output.txt
Two house holds, both a like in dignity, In fair Verona, where we lay our scene, From ancient grudge break to new mutiny, Where civil blood makes civil hands unclean. From forth the fatal loins of these two foes A pair of star-cross'd lovers take their life; Whose miss adventured piteous overthrows Do with their death bury their parent's strife.	1. Two house holds, both a like in dignity, 2. In fair Verona, where we lay our scene, 3. From ancient grudge break to new mutiny, 4. Where civil blood makes civil hands unclean. 5. From forth the fatal loins of these two foes 6. A pair of star-cross'd lovers take their life; 7. Whose miss adventured piteous overthrows 8. Do with their death bury their parent's strife.

3. Merge Files

Write a program that reads the contents of two text files and merges them together into a third one.

Examples

Input1.txt	Input2.txt	Output.txt
1 3 5	2 4 6	1 2 3 4 5 6

4. Filter Extensions

You will receive a **folder** called **input**, with various files with custom extensions. You may add or remove the files as you wish, but they are the only way to test your code.

Write a program which accepts a single input line from the Console, holding an extension … like: "txt", "bmp", "rar" etc.

Print the **NAMES AND EXTENSIONS** of all files, which **have** the **given extension**.

Examples

Input	Input Folder	Output

`txt`	input
	controller
	coverphoto
	file.file
	filed-recordings.file
	profilepic
	script
	something
	test.000.001.in
	test.000.001.out
	test.000.002.in
	test.000.002.out
		`test.000.001.in.txt`
		`test.000.001.out.txt`
		`test.000.002.in.txt`
		`test.000.002.out.txt`

5. HTML Contents

You have been tasked to create a program which represents a Console interface for creating **HTML files**.

Every HTML file naturally holds the following elements:

```
"<!DOCTYPE html>
 <html>
 <head>
 </head>
 <body>
 </body>
 </html>"
```

You will need to add them at the end in order to form the file.

You will start receiving input lines in the following format:

```
{tag} {content}
```

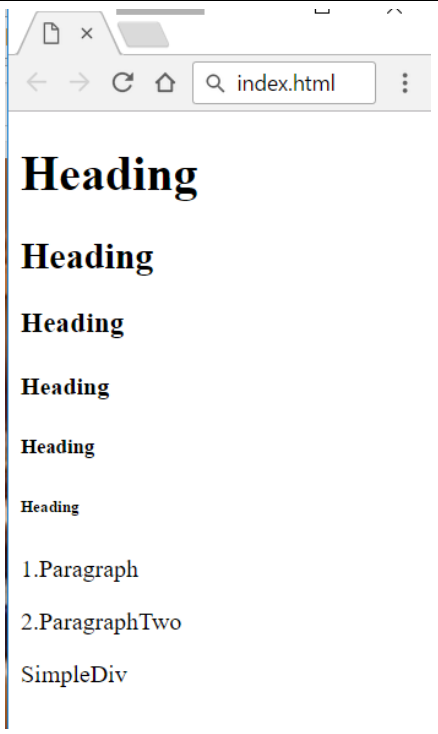
You should **generate** a **string** from **every input line** – like this: **`<{tag}>{content}</{tag}>`** …

If the tag is "**title**" you should add the **generated string** between the **`<head>`** and **`</head>`** tags with a **tabulation** ("**`\t`**") **before** it.
If you receive the "**title**" tag **MORE** than **ONCE**, you should **CHANGE** its **value**.
In **any other case** you should **APPEND** the **generated string** between the **`<body>`** and **`</body>`** tags with a **tabulation** ("**`\t`**") **before** it.

When you receive the command "**exit**" the input ends. The **content** you have **generated** should be **stored** in a file called "**`index.html`**" (**.html extension**).

Examples

Input	Output	index.html

`h1 Heading`
`h2 Heading`
`h3 Heading`
`h4 Heading`
`h5 Heading`
`h6 Heading`
`title Test`
`p 1.Paragraph`
`p 2.ParagraphTwo`
`div SimpleDiv`
`title HTML-Contents`
`exit`
	`<!DOCTYPE html>`
	`<html>`
	`<head>`
	`    <title>HTML-Contents</title>`
	`</head>`
	`<body>`
	`    <h1>Heading</h1>`
	`    <h2>Heading</h2>`
	`    <h3>Heading</h3>`
	`    <h4>Heading</h4>`
	`    <h5>Heading</h5>`
	`    <h6>Heading</h6>`
	`    <p>1.Paragraph</p>`
	`    <p>2.ParagraphTwo</p>`
	`    <div>SimpleDiv</div>`
	`</body>`
	`</html>`
		
		index.html
		Heading
		Heading
		Heading
		Heading
		Heading
		Heading
		1.Paragraph
		2.ParagraphTwo
		SimpleDiv

6. User Database

You have been tasked to create a database for several users, using … Text files.

The client will give you several input commands. There are two main commands:

- **`register {username} {password} {confirmPassword}`**
- **`login {username} {password}`**
- **`logout`**

If you receive the **register command**, you must **store** the **user** in your **database** of **users**, with the **given password**.

- If there is already a **user** with the **given username**, you must print "**`The given username already exists.`**", and **ignore** the command.
- If the **password** and **`confirmPassword`**, do **NOT** match, print on the console "**`The two passwords must match.`**", and **ignore** the command.

If you receive the **login command**, you must **log in** the **user** with the **given username** and **password**.

- If there is already a logged in user, you must print "**`There is already a logged in user.`**", and **ignore** the command.
- If there is **NO user**, with the **given username** you must print "**`There is no user with the given username.`**", and **ignore** the command.
- If the **password** is does **NOT match** the **one** with which the **user** was **registered**, you must print "**`The password you entered is incorrect.`**", and **ignore** the command.

If you receive the **logout command**, you must **logout** the, **currently logged in**, **user**.

- If there is **NO currently logged in user**, you must print "`There is no currently logged in user.`", and **ignore** the command.

When you receive the command "**exit**", the input sequence ends. You must **store** the **current database** of **REGISTERED** users, in a **file** called "**users.txt**". The way you store them is up to you. You must load it, every time the program is **ran**.

Examples

Input	Output

`register Simo 123 123`
`register Ivo 123 132`
`login Simo 132`
`login Simo 123`
`logout`
`register pesho pesho pesho`
`login Ivo 123`
`login pesho pesho`
`exit`
	`The two passwords must match.`
	`The password you entered is incorrect.`
	`There is no user with the given username.`

The **second example test**, **DEPENDS** on the **first one**. Run the first one, **save** the **resulting database** from it, and then run the **second one**, in **order** to get **correct results**.

Input	Output

`register Merry 123456 123456`
`register pesho pesho pesho`
`logout`
`login Simo 123`
`logout`
`exit`
	`The given username already exists.`
	`There is no currently logged in user.`

7. Folder Size

You are given a folder named "TestFolder". Get the size of all files in the folder, which are **NOT directories.** The result should be written to another text file in **Megabytes**.

Examples

Output.txt
5.161738395690918

8. Re-Directory

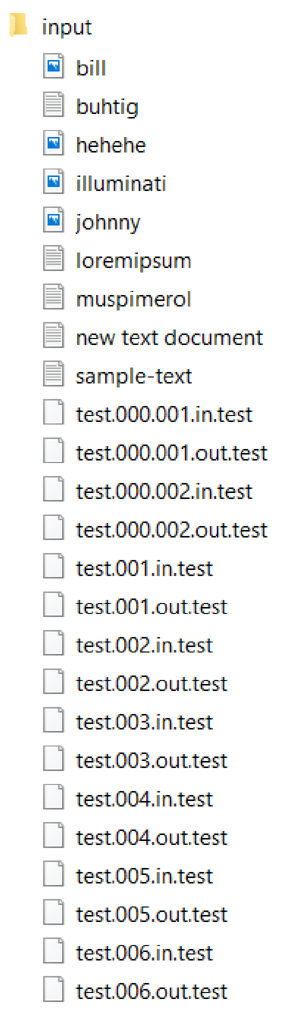
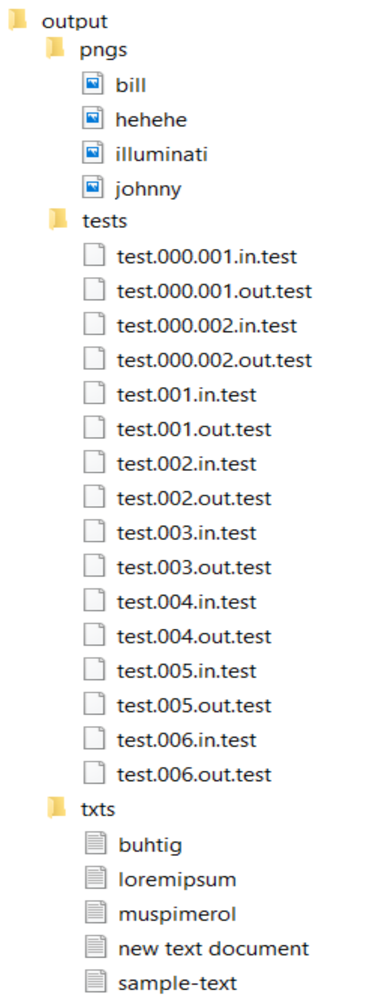
You have been tasked to distribute a directory (folder) of files with various extensions to different folders. The files should be distributed by their file extension.

You need to **group all the files**, which have the **same extension**, into a **folder named**: "**`{extension} + s`**"

In other words: all "**`.txt`**" files must be put in a folder called "**`txts`**".

The resulting folders should be put in a folder "**`output`**".

Examples

Input Folder	Output Folder
 input bill buhtig hehehe illuminati johnny loremipsum muspimerol new text document sample-text test.000.001.in.test test.000.001.out.test test.000.002.in.test test.000.002.out.test test.001.in.test test.001.out.test test.002.in.test test.002.out.test test.003.in.test test.003.out.test test.004.in.test test.004.out.test test.005.in.test test.005.out.test test.006.in.test test.006.out.test	 output pngs bill hehehe illuminati johnny tests test.000.001.in.test test.000.001.out.test test.000.002.in.test test.000.002.out.test test.001.in.test test.001.out.test test.002.in.test test.002.out.test test.003.in.test test.003.out.test test.004.in.test test.004.out.test test.005.in.test test.005.out.test test.006.in.test test.006.out.test txts buhtig loremipsum muspimerol new text document sample-text

9. Products

You have been tasked to create a **File Database** for several **stocked products** at a universal shop.

A product has a **Type** (**string**), **Name** (**string**), **Price** (**decimal**) and **Quantity** (**integer**).

The **type** of the product can be – "**Food**", "**Electronics**", "**Domestics**".
The **name** of the product may consist of **any ASCII character**, except **space**.
The **price** of the product will be a **floating-point number** with up to **20 digits** after the **decimal point**.
The **quantity** of the product will be an **integer** in **range [0, 1000]**.

The software program you must build should be a **Console interface**. You will receive **several input lines**, containing **information** about **products**, in the following format:

```
{name} {type} {price} {quantity}
```

You should **store every product**, with its **respective properties**.

If you receive a **product NAME**, which already **exists AND** has the **SAME TYPE**, you should **REPLACE** its **price** and **quantity**, with the **given ones**.

The products are stored **virtually**, in your program's memory – they are called **ACTIVE products**.

When you receive the **command** "**stock**", in the **input**, you must **stock all products**, you have, in a **file**.

When you receive the **command** "**analyze**", in the **input**, you must **print all STOCKED** products, in **alphabetical order**, by their **TYPE**, each printed in the following format:

```
"{type}, Product: {name}
Price: ${price}, Amount Left: {quantity}"
```

In case there are **NO products** print "**No products stocked**".

When you receive the command "**sales**", in the **input**, you must **print all types** of **ACTIVE products**, and the **income**, **earned** if **all products** and their **quantities** from that **TYPE** are **sold**. In other words, you need to calculate for **every product** from the **respective type**, its **quantity * price**. You must then **sum all sums**, from the products – that's the **INCOME**.

The output should be formatted like this:

```
"{firstType}: ${income}
{secondType}: ${income}
{thirdType}: ${income}"
```

The **types** must be **ordered** in **descending order**, by their **total income**. If one of the types, has **NO products**, **DO NOT PRINT IT**.

ALL PRICES, must be **FORMATTED** to the **second digit**, after the **decimal point**.

The input ends when you receive the command "**exit**". You do **NOT print anything**, you do **NOT store anything on files**. . .
You **just exit the program**.

Note

You **only STOCK** products in the **external FILE**, when you receive the command "**stock**". Do **NOT** stock products at the **end** of the **program execution**.

When you start the program, you should check if you have any stocked products, and if you do, you should **load** them into your **database**.

Examples

Input	Output
SamsungSmartTV Electronics 4000.50 10 Banana Food 1.50 10000 IPhone7 Electronics 1000 100 Apple Food 1 100000 Microwave Electronics 149.99 2500 Toster Electronics 20.00 15730 sales Mopper Domestics 10.05 10000 ToiletPaper Domestics 5.50 100000 analyze sales stock exit	Electronics: $829580.00 Food: $115000.00 No products stocked Electronics: $829580.00 Domestics: $650500.00 Food: $115000.00

The **second example test**, **DEPENDS** on the **first one**. Run the first one and then run the **second one**, in **order** to get **correct results**.

Input	Output

`analyze`
`sales`
`Banana Electronics 1000 50`
`Banana Food 2.09 1000000`
`ToshibaLaptop Electronics 1500 10`
`LenovoLaptop Electronics 1999.99 100`
`AcerLaptop Electronics 1394.49 1000`
`sales`
`stock`
`analyze`
`exit`
	`Domestics, Product: Mopper`
	`Price: $10.05, Amount Left: 10000`
	`Domestics, Product: ToiletPaper`
	`Price: $5.50, Amount Left: 100000`
	`Electronics, Product: SamsungSmartTV`
	`Price: $4000.50, Amount Left: 10`
	`Electronics, Product: IPhone7`
	`Price: $1000, Amount Left: 100`
	`Electronics, Product: Microwave`
	`Price: $149.99, Amount Left: 2500`
	`Electronics, Product: Toster`
	`Price: $20.00, Amount Left: 15730`
	`Food, Product: Banana`
	`Price: $1.50, Amount Left: 10000`
	`Food, Product: Apple`
	`Price: $1, Amount Left: 100000`
	`Electronics: $829580.00`
	`Domestics: $650500.00`
	`Food: $115000.00`
	`Electronics: $2489069.00`
	`Food: $2190000.00`
	`Domestics: $650500.00`
	`Domestics, Product: Mopper`
	`Price: $10.05, Amount Left: 10000`
	`Domestics, Product: ToiletPaper`
	`Price: $5.50, Amount Left: 100000`
	`Electronics, Product: SamsungSmartTV`
	`Price: $4000.50, Amount Left: 10`
	`Electronics, Product: IPhone7`
	`Price: $1000, Amount Left: 100`
	`Electronics, Product: Microwave`
	`Price: $149.99, Amount Left: 2500`
	`Electronics, Product: Toster`
	`Price: $20.00, Amount Left: 15730`
	`Electronics, Product: Banana`
	`Price: $1000, Amount Left: 50`
	`Electronics, Product: ToshibaLaptop`
	`Price: $1500, Amount Left: 10`
	`Electronics, Product: LenovoLaptop`
	`Price: $1999.99, Amount Left: 100`
	`Electronics, Product: AcerLaptop`
	`Price: $1394.49, Amount Left: 1000`
	`Food, Product: Banana`
	`Price: $2.09, Amount Left: 1000000`
	`Food, Product: Apple`
	`Price: $1, Amount Left: 100000`

Note

Use diffchecker.com, to **test** your **output** and the **correct output** of the **tests**, since they are **quite big**.