Ex1: Armstrong Numbers

An Armstrong number of n digits is an integer such that the sum of its digits each one to the power of n is equal to the number itself.

For example, 9,474 is an Armstrong number as $9^4 + 4^4 + 7^4 + 4^4 = 9474$.

All 1-digit numbers are obviously Armstrong numbers.

Write program that reads a sequence of numbers, one number per line, from the file numbers.txt; and copies the Armstrong numbers to the new file armstrong.txt, one number per line, in the same order.

For example, given this numbers.txt:

```
42
7
1634
1743
2
45656
565
371
407
8208
153
6
4646
370
9474
9475
```

The resulting armstrong.txt should be:

```
7
1634
2
371
407
8208
153
6
370
9474
```

Ex2: Computing statistics in a ASCII file

An ASCII art company is trying to compute some statistics on theirs pictures. For doing so, the company is needing a Python program able to compute the statistics in the use of the different characters on their pictures.

You are asked to write a Python program that opens a text file named landscape.txt, that contains a picture converted in ASCII characters. The size of the picture is not known a priori.

Then, the program asks the user where to compute the statistics, for doing so, the program must ask for the position of a square in the picture. The square position is given by asking a coordinate (X,Y) and the square size N, the coordinate indicates the upper left corner of the square. Assuming that in the original image, the upper left corner has the coordinate (0,0), the program must check if the provided square is placed inside the picture or not. In the negative case, the program terminates showing the user an error.

Once a useful square is read, the program must count in the mentioned area the occurrences of each character and provide this information as a percentage.

The program must be structured using functions, and errors regarding the file manipulation must be handled.

Example: if the file landscape.txt is:

```
^^^^^^^~~^~^~^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^!YJYY7^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^!PPPPJ~^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^!5PPPJ!~^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^!5PPPPPJ!^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^YPPPPP7~^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^JPPGP57^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^!YPGP5PJ!~^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^~J5555555!~^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^!!J5555555!^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^!!7YYYYYYY!^^~~~~~~~~~~~~~~~~~~~!!!!!!!!~
^^^~7!!777777777???JJJ?JJJJYYYYYYYYYYYYY5557
YY5555???77J?77J5YYJJJYY555555PPP5PP5555555?
GPGG5PPPP55PYYJ?JYYY?!!!7???JJJ?????????????7
JYYJ?755?J5GP55???7?!^^:~7Y5J77?JJYY5JJYY?77
JP55Y~7777?JJ?7????777777J55PJ!!7J55??7???JY
PBGGP??77??7???J???????77777???7?JPGPPPYJJY5
J5PG5!!777?P5??7?J??7!7??777J5775GGGGGGGGGPP
JJ?Y5J?!!!?5YJJJYPJ7????7!7!7??7?YYYJJJ??J?7
```

And the user introduces the following information

```
Please, enter the coordinates (x,y):
6,1
Please, enter the square size:
10
```

So the part of the figure to analyze is:

```
^!YJYY7^^^
^!PPPPJ~^^
^!5PPPJ!~^
!5PPPPPJ!^
^YPPPPP7~^
^JPPGP57^^
!YPGP5PJ!~
J5555555!~
J5555555!^
7YYYYYYY!^
```

so, the program must print:

```
P-> 23.0%
5-> 18.0%
^-> 17.0%
Y-> 12.0%
!-> 11.0%
J->  8.0%
~->  5.0%
7->  4.0%
G->  2.0%
```

On the contrary, considering the same file,

```
Please, enter the coordinates (x,y):
26,13
Please, enter the square size:
10
```

The program must print:

```
ERROR!! the square to analyze is out of limits.
```

Exercise 3: MURPHY'S LAWS

Consider a text file, named Murphy_reads.txt, which contains some of the maxims derived from the famous * Murphy's law *.

In the text file, each maxim is made up of two or more lines of text: on the first line there is the ** title ** of the maxim, on the following lines there is the ** statement **. The different maxims are separated from each other by a blank line.

Eg:

```
Principio degli elementi persi
Il raggio di caduta dal banco di lavoro di piccoli elementi varia inversamente alle
loro dimensioni e
direttamente alla loro importanza per il completamento del lavoro intrapreso.

Legge di Perussel
Non c'è lavoro tanto semplice che non possa essere fatto male.

Regola di Ray sulla precisione
Misura con un micrometro.
Segna con un gessetto.
Taglia con un'ascia.

Legge dei semafori
Se è verde non hai fretta.

Legge di Pudder
Chi ben comincia, finisce male. Chi comincia male, finisce peggio.

Seconda Legge di Horowitz
In qualunque momento tu accenda la radio, sentirai le ultime note della tua canzone
preferita.

Legge di Vile sul valore
Più un oggetto costa, più lontano bisognerà spedirlo per farlo riparare.
```

A second text file, named arguments.txt contains a series of words (one per line, without spaces or punctuation).

```
male
fretta
lavoro
```

Write a Python program to identify Murphy's laws that address certain topics.

The program must identify all maxims in which at least one of these words appears ** in the statement ** (it doesn't matter whether it appears in the title or not). The searched word must appear as a "complete" word in the text, partial matches are not allowed. For example, if the word were "with", a maxim containing "accounts" should not be selected. The comparison must be done by ignoring the difference between upper and lower case, and removing the punctuation characters.

If a maxim contains more than one word to search for, it must still be printed only once. For the maximum corresponding to the criterion, print the title (complete) and the beginning (first 50 characters) of the statement. If the statement were longer than 50 characters, indicate with '...' the fact that it was truncated.

In the case of files with the contents indicated in the examples, the program must produce the following output:

```
Principio degli elementi persi - Il raggio di caduta dal banco di lavoro di
piccoli...
Legge di Perussel - Non c'è lavoro tanto semplice che non possa essere...
Legge dei semafori - Se è verde non hai fretta.
Legge di Pudder - Chi ben comincia, finisce male. Chi comincia male,...
```

Exercise 4: GLUCOMETER

The new generation of wearable devices allow patients with diabetes mellitus to be continuously monitored and non-invasive blood sugar. The values of the individual devices are sent on a daily basis to a data center which assembles a single file containing the data of all patients connected to the monitoring service. You want create a program capable of extracting some statistics. Specifically, a check on the exceeding the maximum level of **200mg/dL.**

Technical specifications follow.

The program has to manage a glucometers.txt input file containing the data of a whole day of measurement. This file must be understood as an aggregate of the data collected and therefore containing, in no particular order, the data of all patients.

Data for a single patient is in chronological order and missing measurements may occur (intervals in which the patient is not wearing the device).

All measurements are made on a regular basis: one sample every 5 minutes.

Each line of the file contains 5 different data separated by a single space and in the following order:

1. patient identification code consisting of 4 hexadecimal characters (PPPP format)
2. acquisition time (hh:mm format)
3. blood glucose value (mg/dL)
4. body temperature (°C)
5. heart rate (bpm).

The program, after storing the data in an appropriate data structure, will have to print the list of all the patients who have recorded at least one exceedance and, for each exceedance, the time and blood glucose level corresponding. The list must appear in order of criticality, starting with the patient with the most exceedances.

GLUCOMETER.TXT

```
1BF0 17:00 160 37.0 68
1BF0 17:05 168 37.0 68
1BF0 21:00 180 37.3 66
1BF0 21:05 210 37.1 67
0AE1 21:10 187 37.3 69
0AE1 21:15 192 37.3 70
0AE1 21:20 195 37.4 70
0AE1 21:25 201 37.4 75
BBB3 22:30 108 37.5 73
BBB3 22:35 200 37.5 73
0AE1 23:05 203 37.4 73
0AE1 23:10 210 37.5 71
1BF0 21:10 213 37.2 68
```

OUTPUT:

```
0AE1 21:25 201
0AE1 23:05 203
0AE1 23:10 210

1BF0 21:05 210
1BF0 21:10 213

BBB3 22:35 200
```