Hash64

Quick Background

The information in this section is not important to solve the exercise. It is merely provided for context.

SHA384 is a *secure hash algorithm*. It takes input and then "hashes" this into a single number. This number is 384 bits long, so it is quite large, which is why Python represents the hash value as an array of 48 bytes.

Base64 is a way of representing numbers. You are used to decimal notation, which uses 10 digits, meaning it is base 10. Binary has two digits, so it is base 2. Base64 simply uses 64 different digits.

Your Task

For each integer I going from 0 to 10000 (included), compute the SHA384 value and encode the hash in base64. Write your results in a file where each line contains the integer I followed by a space, followed by its hash value:

I HASH

Details

* The Python standard library contains functions that compute the SHA384 and base64 encoding for you.
* In order to compute the SHA384 of an integer I:
  + Convert I to a string.
  + Convert this string to raw bytes using encode. Use ascii as encoding.
  + Give these raw bytes to the SHA384 function.
* The base64 function returns raw bytes. In order to print this, you need to convert these to a regular string. Rely on the decode method. Again use ascii as encoding.

Example

We consider here I = 0.

* We convert it to a string, which gives us '0'.
* We convert this to raw bytes using ascii encoding, which, which Python will show as b'0'.
* We hash this using SHA384, which gives us the following byte array (48 long):

b'\_\x91U\x0e\xdb\x03\xf0\xbb\x89\x17\xdaW\xf0\xf8\x81\x89v\xf5\xda\x97\x13\x07\xb7\xeeH\x86\xbb\x95\x1cH\x91\xa1\xf1o\x84\r\xae\x8feZ\xa5\xdfq\x88\x84\xeb\xc1['

* We convert this to base64, yielding another byte array:

b'X5FVDtsD8LuJF9pX8PiBiXb12pcTB7fuSIa7lRxIkaHxb4QNro9lWqXfcYiE68Fb'

* Finally, transforming this into a string using ascii gives

'X5FVDtsD8LuJF9pX8PiBiXb12pcTB7fuSIa7lRxIkaHxb4QNro9lWqXfcYiE68Fb'

In other words, the first line of output.txt should be

0 X5FVDtsD8LuJF9pX8PiBiXb12pcTB7fuSIa7lRxIkaHxb4QNro9lWqXfcYiE68Fb

To help you check your work, here are the first five lines of output.txt:

0 X5FVDtsD8LuJF9pX8PiBiXb12pcTB7fuSIa7lRxIkaHxb4QNro9lWqXfcYiE68Fb

1 R/BdNnsMMuQ4+2Pmz0pfNcKqL5DcdUP4pBoPlc6KQKMTq1zzYTSiBoxMlpy1Dbd2

2 0GNFdwXWbW8BbkzddH2zr41w6/02ut1j3myMpKnYv7XYdOf711CqgE3K3a5+7vUe

3 avEcg1hoIsPHS7PM73KLrlz+5nytgt10AnEeUwvseC/AKv8nNWnSLd/7OxRfNDdo

4 vPbg68jFGFNyyNjw9Elye9UsUoSGDxxG6lPclD+r9Ax1BnYpgT8SvZlNdaOfRIQ9

# Md2adoc

You are probably familiar with HTML: it is a markup language, which allows you to define the structure of a text. HTML, however, is quite verbose and not very human-friendly. Markdown and AsciiDoc are markup languages that are more lightweight and easier to use than HTML.

Your task will be to convert a Markdown file into an equivalent AsciiDoc file. A full translation would take a lot of time, so we focus on just two things.

## Section Headings

HTML uses <h1>...</h1>, <h2>...<h2>, etc. to denote headings of different levels. Markdown uses the more lightweight syntax # ... and ## ...: the number of # symbols indicates how "deep" the heading is. AsciiDoc works exactly the same, except it uses = symbols instead of #.

The first part of your job is to look for all lines starting with on or more # symbols and replace these by =. Note that the # has to be placed at the very beginning of the line, otherwise it needs to be ignored.

## Bullet Point Lists

In Markdown, bullet lists are written

\* Level 1 item

\* Second level 1 item

\* Subitem (level 2)

\* Subsubitem (level 3)

\* Another subitem (level 2)

As you can see, bullet point lists can be recognized by their \* symbols. Sublists are indented using exactly two spaces per level, i.e., a subsubitem will start with 4 spaces, then an asterisk, then the title.

In AsciiDoc, the same list is written

\* Level 1 item

\* Second level 1 item

\*\* Subitem (level 2)

\*\*\* Subsubitem (level 3)

\*\* Another subitem (level 2)

Instead of using indentation, AsciiDoc simply increases the number of \*s to denote the depth.

## Your Task

Translate the given input.md file to output.txt:

* Translate all section headings.
* Translate all bullet lists.

All other content must be kept unmodified.

## Example

Consider the following Markdown file:

# Title

Some paragraph text

## Subtitle 1

Text.

More text.

## Subtitle 2

### Subsubtitle

Let's make a nice list.

\* First item

\* Second item

\* Third item

\* Subitem

\* Another subitem

\* Fourth item

\* Deeper we go

\* Even deeper

\* Very deep

## Something else entirely

Last line.

It must be translated to

= Title

Some paragraph text

== Subtitle 1

Text.

More text.

== Subtitle 2

=== Subsubtitle

Let's make a nice list.

\* First item

\* Second item

\* Third item

\*\* Subitem

\*\* Another subitem

\* Fourth item

\*\* Deeper we go

\*\*\* Even deeper

\*\*\*\* Very deep

== Something else entirely

Last line.

# rank

Take a peek at input.json. It contains a list of people's scores. We ask of you to create a ranking: order the people by decreasing score. In other words, the person with the highest score is ranked #1, the person with the second highest score is ranked #2, etc.

Write your results to a file named output.txt. Each line in this file must be formatted as

RANK NAME

The lines must be ordered by increasing rank.

Note: you can assume that each person in the list has a different score. You don't need to worry about how to order people with equal scores.

## Example

Say input.json has the following contents:

[

{

"name": "Laura Michael",

"score": 1482

},

{

"name": "Joshua Park",

"score": 9106

},

{

"name": "Kristina Jackson",

"score": 9663

}

]

The corresponding output.txt must then contain:

1 Kristina Jackson

2 Joshua Park

3 Laura Michael

# toc

You are given as AsciiDoc file for which you need to create a table of contents.

First, you will need to find all section headings. A heading is a line that starts with one or more = symbols, followed by a space, followed by a title. The number of = symbols represents the level of the section. On the line before the heading, its id is defined using the syntax [#id].

[#main]

= Title

Some text

[#sub]

== Subtitle

Some more text.

Lorem ipsum and stuff.

[#subsub]

=== Subsubtitle

Text.

In the example above, there are three headings:

* The first one has title Title and id main. It has level 1.
* The second one has title Subtitle and id sub. It has level 2.
* The third one has title Subsubtitle and id subsub. It has level 3.

From this, you need to create a table of contents as a bullet point list with links to the corresponding heading. The structure of the bullet point list must mirror that of the entire document: a level N heading must be represented by a bullet point with "depth" N.

\* <<main,Title>>

\*\* <<sub,Subtitle>>

\*\*\* <<subsub,Subsubtitle>>

Note the following:

* A bullet point of depth K starts with K \* symbols.
* A link is written <<id, title>>.

Write this table of contents to a file named output.txt.

# Verify-zips

The directory data contains a number of zip files. Each of these zip archives is supposed to contain three files:

* file-a
* file-b
* file-c

However, some will be missing one or more of these files, or contain other unwanted files. Your job is to identify this faulty zips and list them. Write a script that produces a file output.txt that lists all these zip files in alphabetical order.

## Example

Say we have these zip files with contents

* sub000.zip
  + file-a
  + file-b
  + file-c
* sub001.zip
  + file-a
  + file-b
  + file-c
  + extra
* sub002.zip
  + file-a
  + file-c

The archive sub000.zip is correct: is contains the three expected files and only those files. sub001.zip is incorrect: the file extra should not be there. sub002.zip is also incorrect: it misses file-b.

In this situation, output.txt should contain

sub001.zip

sub002.zip