General

Convert letters to their ordinal value

In Python, each character has a corresponding ordinal value (`int`). For example, the question mark (`?`) has ordinal value 63. The value can be easily obtained using the built-in function `ord`.

```
>>> ord('?')
63
```

The interesting thing about these ordinal values, is that successive letters in the alphabet have successive ordinal values. This holds for both uppercase and lowercase letters. If we know that the letter `a` has ordinal value 97, it immediately follows that the letter `b` must have ordinal value 98. Using this knowledge, we can easily determine the position of a letter in the alphabet.

```
>>> letter = 'd'
>>> ord(letter)
100
>>> ord('a')
97
>>> ord(letter) - ord('a') + 1   # d is the 4th letter of the alphabet
4
>>> ord('z') - ord('a') + 1      # z is the 26th letter of the alphabet
26
```

Suskewiet

Remove leading and/or trailing whitespace

In Python, you can use the string method `strip` to remove leading and trailing whitespace (spaces, tabs and newlines). In case you only want to remove leading whitespace, you can use the string method `lstrip`. In case you only want to remove trailing whitespace, you can use the string method `rstrip`. You can also pass an argument to these string methods, that indicates which leading and/or trailing characters have to be removed. For more details about these string methods, we refer to The Python Standard Library.

```
>>> text = '  This is a text  '
>>> text.strip()
'This is a text'
>>> text.lstrip()
'This is a text  '
>>> text.rstrip()
'  This is a text'
>>> text2 = '===This is a text==='
>>> text2.lstrip('=')
'This is a text==='
```

String repetition

To repeat a given string a fixed number of times, you can multiply that string with an integer. As with the multiplication of numbers, this uses the operator `*`. The order of the string and the integer does not matter in the multiplication

```
>>> 'a' * 3
'aaa'
>>> 5 * 'ab'
'abababababab'
```

This is very handy in case the number of repetitions of the string is not known beforehand, but for example can be retrieved from a variable.

```
>>> repetitions = 4
>>> repetitions * 'abc'
'abcabcabcabc'
```

String methods `islower()` and `isupper()`

The string method `islower` checks if *all* characters of a string are lowercase letters. The string method `isupper` checks if *all* characters of a string are uppercase letters.

```
>>> 'ABCD'.islower()
False
>>> 'ABCD'.isupper()
True
>>> 'abcd'.islower()
True
>>> 'abcd'.isupper()
False
>>> 'AbCd'.islower()
False
>>> 'AbCd'.isupper()
False
```

Specific information

When solving this exercise, it is important to break down the problem into smaller problems that can be solved separately (divide-and-conquer principle). For each string that has to be processed, you need to perform the following steps:

- remove all characters that are no letters
- check if the string ends with `suskewiet`
- check if the first part only consists of the letters `ktreu`

At first, you can just count the number of correct songs using a counter variable. Afterwards you can generate the output in the expected format based on the value assigned to this variable.

Word evolutions

The string method `index`

The string method `index` may be used to determine the position of the leftmost occurrence of a character in a strings.

```
>>> 'mississippi'.index('i')
```

This code fragment finds the leftmost occurrence of the letter `i` at position 1 in the string `mississippi`. Note that Python indexes the positions of the characters in a string starting from 0, not starting from 1.

You can also pass a second argument to the string methode `index` that indicates the position where left-to-right scanning of the string starts when searching for the next occurrence of the first argument. For example, if you already know that the first occurrence of the letter `i` in the string `mississippi` is at position 1, you can find the second occurrence of the letter `i` by starting to search from position 2 onwards:

```
>>> 'mississippi'.index('i', 2)
4
```