Table talk

Specific information

Use the string method `isalpha` to check whether a character is a letter. The method recognizes both uppercase and lowercase letters.

```
>>> 'f'.isalpha()
True
>>> 'Y'.isalpha()
True
>>> '3'.isalpha()
False
>>> '.'.isalpha()
False
```

Stop codons

Iterate both the elements and their positions of sequence types

The built-in function `enumerate` can be used to request an iterator for a given sequence type (strings, lists, tuples, files, ...) that both returns the position and the value at that position for the next element of the sequence type. The example below illustrates how this can be used to simultaneously iterate the positions of a string and the characters on those position.

```
>>> for index, character in enumerate('abc'):
...     print(f'index: {index}')
...     print(f'character: {character}')
...
index: 0
character: a
index: 1
character: b
index: 2
character: c
```

You can also use this to simultaneously iterate over the characters of two strings.

```
>>> first = 'abc'
>>> second = 'def'
>>> for index, character in enumerate(first):
...    print(f'{character}-{second[index]}')
...
a-d
b-e
c-f
```

However, in this case it is better to use the built-in function `zip`, which is especially equipped to iterate over multiple iterable objects at once.

```
>>> first = 'abc'
>>> second = 'def'
>>> for character1, character2 in zip(first, second):
...     print(f'{character1}-{character2}')
...
a-d
b-e
c-f
```

Pangrammatic window

Specific information

The easiest strategy to implement the function `window` is to iterate over all possible substrings of the given text. For each of these substrings you then determine whether it is a pangram, and you remember which one of the pangrams is the shortest one.

To iterate over all substrings of a given string you can iterate over all possible start positions of substrings, and then iterate over all possible stop positions for each possible start position. This can be implemented using a nested `for`-loop. The following code snippet, for example, outputs all substrings of the string `'abc'`, where we only take into account substrings having at least length 1.

```
>>> s = 'abc'
>>> for start in range(len(s) - 1):
...     for stop in range(start, len(s)):
...         print(s[start:stop + 1])
...

a
ab
abc
b
bc
```

If you think twice, there's even no need to iterate over all possible substrings to find the shortest pangram. If you want to avoid spurious computations, you can give it a thought how you can avoid that Python should do unnecessary work.

Rövarspråket

Specific information

The solution to this assignment comes with realizing that processing a group of successive consonants needs to be postponed until a character is encountered that is not a consonants (which completes the group of successive consonants). This is illustrates by means of the following example, which translates the string `abcdefg` into Rövarspråket.

The idea is to traverse the string that needs to be encoded character per character. When a consonant is encountered, it is not immediately appended to the encoded string, but instead it is appended to the current group of consecutive consonants. When a non-consonant is encountered, it closes the current group of consecutive consonants. In case such a group has been formed, it is appended to the encoded string, followed

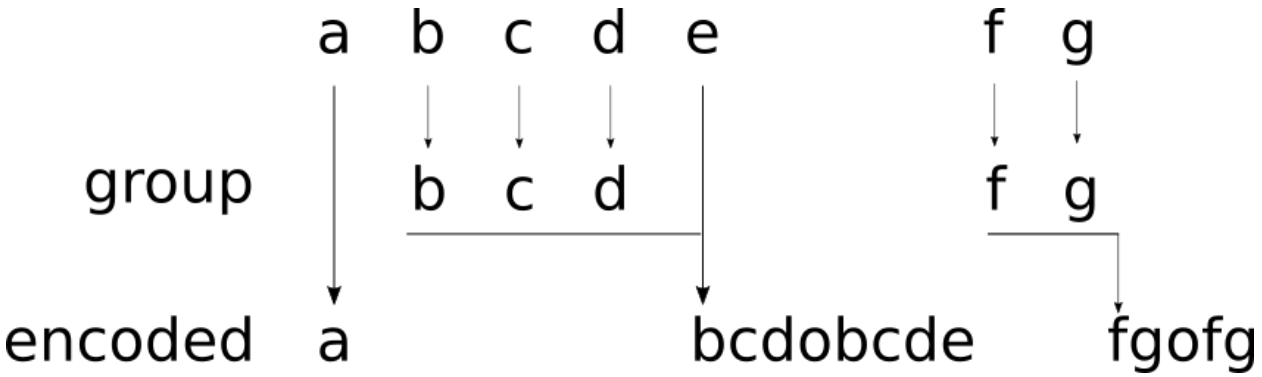

Figure 1: Rövarspråket

by the letter `o` and another repetition of the group of consecutive consonants. Only after this has been done, the non-consonant is appended to the encoded string, and a new group of consonants is started.

After all character of the string that needs to be encoded have been processed, there might still be a group of consecutive consonants that has not been processed. This is the case if the string ends in a consonant. In the example given above, this is the case for the group `fg`. If you end up in this situation, you still need to append the current group of consecutive consonants to the encoded string, followed by the letter `o` and another repetition of the group of consecutive consonants.