λόγος (ver. 3.0.0)

Short Description

The programm is a tool for conversion of a text into tone rows. It outputs lists of notes and/or Music XML files, which can be read by a notation software. In this project Finale is used.

Requierments

- Python3 Download the latest stable version at: https://www.python.org/downloads/
- Libraries with key signatures for Finale (provided with the script)
- Font "Accidentals.ttf"

Detailed Description

The programm takes a text file and translates it to a tone row. A rule for the process of tranlsation should be given in a seprate JSON-File. In the JSON-file the correspondance between the letters of the text and pitches should be established.

Note Syntax

The programm uses following syntax for notes: `<notename><octave><ct>`

`<notename>`
c cis d dis e f fis g gis a ais h (german system).

`<octave>`
-great+22vb – sub sub contra octave
-great+15vb – sub contra octave
-great+8vb – contra octave
-great – great octave
-small – small octave
' – 1 Line (middle c) octave
'' – 2 Line octave
''' – 3 Line octave
'''' – 4 Line octave
''''+8va – 5 Line octave
''''+15va – 6 Line octave
''''+22va – 7 Line octave

`<ct>`
Additional transpositon in cents.

Examples:
ais''+15va
cis'
d-great+8vb
a-small
dis-great+8vb+25ct

The Note Object (Developer's infomation)

The script works with lists of Note objects. The declaration of the class you can find in the module note.py. A Note object can be initialized either with a MIDI-Pitch (also float) or with a note name, according to the described syntax for notes. Also a duration, articulation and a comment can be passed.

For example:

```
my_note = Note(60.25, 'q', 'a', 'I am a Note')
```

This line will initialize a Note object with the MIDI-Pitch 60.25, duration 'q', articulation 'a' and the comment 'I am a Note'.

To initialize a Note object with a note name use:

```
my_note = Note('c\'+25ct', 'q', 'a', 'I am a Note')
```

Regardless of the type of initialization (with note name or with MIDI-Pitch) the object calculates and stores the both representations of the pitch.

To access the MIDI-Pitch: my_note.midi
```
>> 60.25
```

To access the note name: my_note.note
```
>> c'+25ct
```

Text Conversion

Main Algorithm

For every single letter, which is to be translated into a pitch a rule should be given. A letter is not just attached to a fixed pitch. Every single time the letter appears in the text, the pitch which is given in the JSON-file with the rule will be transposed according to a pattern.

A rule for the letter "a" could look as follows:

```
    "a": {
          "main_note": "cis'",
          "pattern": [1, 1, 1, -3],
          "units": "1/8",
          "alternative": "á",
```

```
        "modifier": "a"
    }
```

This means that every time, when the letter "a" occurs in the text, the pitch cis' will be generated. Additionally, this fundamental pitch will be transposed depending on how many times this letter occured so far. These transpositions happen according to the `pattern` -key of the dictionary. The pattern key sets the relative transposition in `units` : 1/2 for half tones, 1/4 for quarter tones, 1/8 for eight tones.

In the example above the following pitches will be generated:

1. "a" occurs for the first time Output: cis' (No transposition fo the very first time the letter occurs in the text.)
2. "a" occurs for the second time Output: cis'+1*(1/8 tone) = cis' + 25ct (Transposition = +1 Step (1st element in pattern) = +1/8-tone.)
3. "a" occurs for the third time Output: cis'+25ct (prev pitch) + 1*(1/8 tone) = cis' + 50ct (Transposition = +1 Step (2nd element in pattern) =
   +1/8-tone.)
4. "a" occurs for the fourth time Output: cis'+50ct (prev pitch) + 1*(1/8 tone) = cis' + 75ct (Transposition = +1 Step (3rd element in pattern) =
   +1/8-tone.)
5. "a" occurs for the fifth time Output: cis'+75ct (prev pitch) - 3*(1/8 tone) = cis' (Transposition = -3 Steps (4th element in pattern) =
   -3/8-tone.)
6. "a" occurs for the 6th time Output: cis' (prev pitch) + 1*(1/8 tone) = cis' + 25ct

So the pattern goes in cycle.

Alternatives / Accents

As in the example above it is possible to work with alternatives of letters. If an alternative is given and occurs in the text the note gets a modifier: doubled duration or accent on the note. For double duration use `dd` for modifier, for accent over the note use `a` . The alternatives are subjected to the same logic of transposition and from this perspective not considered as different letters. For the process of the cyclic transposition the letter and its alternative are indistinguishable.

Example for a JSON-File with translation rules

See rules_example.json:

```
  {
"a": {
  "main_note": "cis'",
  "pattern": [1, 1, 1, -3],
  "units": "1/8",
  "alternative": "á",
  "modifier": "a"
},
"e": {
  "main_note": "d'",
  "pattern": [1, 1, -2],
```

```
    "units": "1/4",
    "alternative": "é",
    "modifier": "dd"
  },
  "o": {
    "main_note": "h'",
    "pattern": [1, 1, -2],
    "units": "1/4",
    "alternative": "ó",
    "modifier": "dd"
  }
}
```

Main Menu

To start the script open your terminal and type following line: python /your_path_to_the_script/logos.py

/your_path_to_the_script is your path to the directory, where logos.py is located. You can also just type python in terminal and drag&drop the file logos.py.

Press Enter.

You will see following menu:

1: Text → Notes
2: Convert a list with Note objects into a Music-XML-File
3: Print variables
4: Delete variables
5: Export variables
6: Import variables
q: Quit

For choosing a certain menu option just enter the corresponding number and press enter.
To quit the programm enter q.

1: Text → Notes

Converts a text into a list of Note objects.
Each list is saved as a variable.

After choosing the menu option 1 you will see following dialogue.

```
▶▶▶ Enter the name of the file with the text.
b: Back ‖ m: Main Menu ‖ q: Quit
>>
```

Here you can enter the filename with the text you wish to convert to notes,
go one step back in the menu structure (by entering b), go to the main menu (by entering m),
or quitting the program (by entering q). The file with the text should be in the same directory as the logos.py.

Entering the filename with the converting rule

```
▶▶▶ Enter the name of the JSON-File with the convertion rule.
b: Back ‖ m: Main Menu ‖ q: Quit
>>
```
Enter the name of the JSON-file containing the rule for the conversion.

Entering the variable name for the result

```
▶▶▶ Enter the variable name for the result.
b: Back ‖ m: Main Menu ‖ q: Quit
>>
```
Enter the name of the variable for storing the list with Note objects.

The programm will generate a list with Note objects stored in the given variable.
All the Note objects in this list have their MIDI-Pitch as a float number,
the note name according to the described syntax, articulation and duration if given
and the letter itself from the text as comment of the Note object.

2: Convert a list with Note objects into a Music-XML-File

This option creates a Music-XML file from a list of Note objects stored previously
in a variable.

Entering the variable name

```
▶▶▶ Enter the variable names for the lists with note objects separated by spaces.
▶▶▶ The lists will be rendered on separate staves in a Music-XML-File.
b: Back ‖ m: Main Menu ‖ q: Quit
>>
```
Here you can enter multiple variables, and they will be rendered on separate staves
in the Music XML file. If a modifier was given, the note will be rendered either with accent or with doubled duration, depending on the modifier. The corresponding letters will be rendered as lyrics in Music XML file.

Entering the number of notes per staff, regular duration and pitch resolution

```
▶▶▶ Enter the number of notes per a staff, durations
▶▶▶ and the pitch resolution, separated by spaces
b: Back ‖ m: Main Menu ‖ q: Quit
```
Additionally syntax information will be also printed out in the terminal.

Enter the three parameters in one line separated by spaces: number of notes per staff, regular duration and pitch resolution. Regular duration is the duration all notes will be rendered with, if no duration modifier is given in the rule. If the modifier `dd` is given (double duration) the note will be rendered with doubled duration relatively to the regular duration.

Syntax for regular durations:
```
2: for half notes
p4: for dotted quarter notes
4: for quarter notes
p8: for dotted eigth notes
8: for eigth notes
p16: for dotted 16th
```

```
16: for 16th
32: for 32th
```

The pitch resolution sets the accuracy of roundinng of the pitches when rendering to Music XML. Keep in mind, that if working with Finale, you need to have Accidentals.ttf intalled on your system. You also have to import the corresponding library with key signatures to your Finale project (the libraries for 1/4 and 1/8-tones are provided with this distribution).

Syntax for pitch resolution:

```
1/16: round to 16th tones
1/8: round to 8th tones
1/6: round to 6th tones
1/4: round to quarter tones
1/3: round to third tones
1/2: round to half tones
```

For example:
`10 p4 1/8` means 10 dotted quarters per staff in the Music XML file. The pitches will be rounded to 1/8th-tones.

Entering the filename for Music XML

```
▶▶▶ Enter the name for the Music-XML-Files without filename extension
▶▶▶ The extension will be generated automatically for two Music-XML-Files:
▶▶▶ for playback and notatioin.
b: Back ‖ m: Main Menu ‖ q: Quit
```

Two files will be generated: one for notation only and one for playback in Finale. In the playback file every note has an expression attached to it, so it is possible to programm microtonal transpositions using for example pitch bend in Finale.

3: Print variables and 4: Delete variables

Using main menu options 3 or 4 you can print out or delete the variables you created previously.

5: Export variables

Here you can export the variables you created to a JSON-file. So you can save your project to continue working on it later.

```
▶▶▶ Enter the name for the JSON-File.
b: Back ‖ m: Main Menu ‖ q: Quit
```

Here you shoud provide the name for the JSON-file with the .json extesion, in which all your variables will be stored. The file will be created in you working directory.

6: Import variables

By choosing this option of the main menu you can import previously stored project from a JSON-File.

Importing Music XML to Finale

To import the generated Music XML files to Finale do following steps:

1. Open Finale
2. Click on "Import Music XML"
3. Import the key signature libraries
    i. Go to Document Menu
    ii. Go to Document Options
    iii. Click "Load library"
4. Choose a rpvided library: key_1_4.lib (for maximal pitch resolution 1/4 tone) or key_1_8.lib (for maximal pitch resolution 1/8 tone).
5. Change the key signature on the staves:
    i. Activate the key signature tool
    ii. Double click on a staff
    iii. Choose Non-Standard
    iv. Click twice on Next till "Linear key format 2" appears.