# Musenot

*Release 0.0.1*

**Marco de Freitas**

**Jul 20, 2021**

# CONTENTS:

CHAPTER

# ONE

# MODULE CONTENTS

## 1.1 musenot.default module

This module provides a bunch of default values for music concepts

The cycle of note names NAMES_WHEEL=(“a”,”b”,”c”,”d”,”e”,”f”,”g”)

The default scale. Builded by mapping note names cycle to default scale structure in semi-tones. Blank spaces generated by whole steps have None as name. SCALE=(“a”,None,”b”,”c”,None,”d”,None,”e”,”f”,None,”g”,None)

The default degrees of the major scale mesued in semitones. DEGREES=(0,2,4,5,7,9,11,12)

Generic names of the major scale intervals. INTERVALS=(“first”,”second”,”third”,”forth”,”fifth”,”sixth”,”seventh”,”eighth”)

Default accidentals string with their lower/higer semi-tone value ACCIDENTALS={‘#’:’1’, ‘b’:’-1’}

## 1.2 musenot.notes module

**class** musenot.notes.**NoteName**(*name*)

Bases: object

This class represents note names according to single letter convention: a,b,c,d,e,f,g.

**Parameters** **string** (*string*) – Note name

**Variables**

- **label** – The note name itself, usual string representation.
- **index** – The note name index in Names Wheel. See musenot.default.
- **indexInRefSclae** – The note name index in reference scale. See musenot.default.

NoteName is initialised by a single letter string.

**Example**

```
>>> note=NoteName('c')
>>> print(note.label)
c
>>> print(note.indexInNamesWheel)
2
>>> print(note.indexInRefScale)
3
```

**NAMES_WHEEL = ('a', 'b', 'c', 'd', 'e', 'f', 'g')**

**__init__**(*name*)

**__str__**()
Return str(self).

**class** musenot.notes.**AbsNote**(*abs_note_name*)
Bases: musenot.notes.NoteName

Class represents abstract notes

Abstract notes, with no determined pitch or octave, are used in music theory to discuss general relations.

**Variables** **name** – NoteName object

**__init__**(*abs_note_name*)

**Parameters** **string** (*String*) – Qualified note name

```
>>> x = AbsNote('c')
>>> x.label
'c'
>>> x.indexInNamesWheel
2
>>> x.indexInRefScale
3
```

**__str__**()

```
>>> a =AbsNote('c')
>>> print(a)
c
```

**is_enharmonic**(*other*)
Test if AbsNote instance is enharmonic with other note represented in short hand notation or as Absterval instance

>> > a = AbsNote('cb') >> > a.is_enharmonic(AbsNote('b')) True >> > a.is_enharmonic('a##') True

**regex_process_alteration**(*expr: str*) → int
!!expr start without the note name !! :param expr: :return:

## 1.3 musenot.intervals module

**class** musenot.intervals.**AbsInterval**(*cypher*, *size=None*, *name=None*, *descending=False*)
Bases: musenot.intervals.IntervalName

Class for abstract intervals

**Parameters** **string** – interval analytic description

```
>>> a=AbsInterval('5')
>>> print(a)
Perfect fifth
```

```
>>> b=AbsInterval('b3')
>>> print(b)
minor third
```

```
>>> print(a+b)
minor seventh
```

```
>>> print(AbsInterval('-3'))
descending Major third
```

```
>>> thirdmajor=AbsInterval('3')
>>> thirdminor=AbsInterval('b3')
>>> print(thirdmajor+thirdminor)
Perfect fifth
```

```
>>> c1=AbsInterval.from_description('third',4)
>>> print(c1)
Major third
```

```
>>> c1=AbsInterval.from_description('third',4,True)
>>> print(c1)
descending Major third
```

**add_absnote**(*other:* musenot.notes.AbsNote) → *musenot.notes.AbsNote*

**Parameters** **other** –

**Returns**

**add_absterval**(*other: musenot.intervals.AbsInterval*) → musenot.intervals.AbsInterval

**Parameters** **other** –

**Returns**

**add_half_step**(*other: int*)

**Parameters** **other** –

**Returns**

**static from_description**(*name*, *size*, *descending=False*)

**static from_notes**(*qualified_name1*, *qualified_name2*, *descending=False*, *octave=0*)

```
>>> AbsInterval.from_notes('c','d')
Absterval('2')
>>> AbsInterval.from_notes('c','eb')
Absterval('b3')
>>> AbsInterval.from_notes('c','g',descending=True)
Absterval('4')
>>> AbsInterval.from_notes('c','g',octave=1)
```

(continues on next page)

(continued from previous page)

```
Absterval('12')
>>> AbsInterval.from_notes('c','g',True,1)
Absterval('11')
```

**init_from_cypher**(*interval_cypher: str*)

**invert**()

```
>>> a=AbsInterval('3')
>>> a.invert()
Absterval('b6')
>>> print(a.invert())
minor sixth
```

```
>>> a+a.invert()
Absterval('8')
```

**property short_hand**

**class** musenot.intervals.**IntervalName**(*name: str*)

Bases: object

This class represents IntervalNames. Can be initialized either by a interval number or an explicite name. See examples below.

**Parameters**

- **number** (*String*) – numeric representation of the interval captured by regex.
- **name** – explicit name initialization.

**Example**

```
>>> IntervalName('1')
unison
```

```
>>> IntervalName('2')
second
```

```
>>> a=IntervalName('3')
>>> print(a.index)
2
>>> print(a.type)
Major
```

```
>>> IntervalName('','unison')
unison
```

```
>>> IntervalName('','second')
second
```

```
>>> a=IntervalName('','third')
>>> print(a.index)
```

(continues on next page)

(continued from previous page)

```
2
>>> print(a.label)
third
```

**ACCIDENTALS = {'': 0, '#': 1, 'b': -1}**

**ADJECTIVS_majmin = ('diminished', 'minor', 'Major', 'augmented')**

**ADJECTIVS_perf = ('diminished', 'Perfect', 'augmented')**

**DEGREES_semitones = (0, 2, 4, 5, 7, 9, 11, 12)**

**INTERVALS = ('unison', 'second', 'third', 'fourth', 'fifth', 'sixth', 'seventh', 'octave', 'ninth', 'tenth', 'eleventh', 'twelfth', 'thirteenth', 'fourteenth', 'fifteenth', 'sixteenth')**

**INTERVALS_types = ('Perfect', 'Major', 'Major', 'Perfect', 'Perfect', 'Major', 'Major', 'Perfect', 'Perfect', 'Major', 'Major', 'Perfect', 'Perfect', 'Major', 'Major', 'Perfect')**

## 1.4 musenot.scales module

**class** musenot.scales.**Abscale**(*code*, *penta=None*, *hexa=None*)

Bases: object

This is a class representation of abstract scales, i.e a structure of intervals with no determined note names.

**Variables**

- **code** – Successiv scale intervals length mesured in half-steps, string type.
- **steps** – Corresponding list of seconds intervals, represented by a list of musenot.intervals.Absterval.
- **degrees** – Step interval from root, represented by a list of musenot.intervals.Absterval.

Creation of an abstract scale is done by providing scale steps measured in half-steps >>> major=Abscale('2212221') >>> minor=Abscale('2122122')

Scale degrees is a scale representation where each note is represented as an interval from the root >>> print(major.degrees) [Absterval('1'), Absterval('2'), Absterval('3'), Absterval('4'), Absterval('5'), Absterval('6'), Absterval('7'), Absterval('8')] >>> print(minor.degrees) [Absterval('1'), Absterval('2'), Absterval('b3'), Absterval('4'), Absterval('5'), Absterval('b6'), Absterval('b7'), Absterval('8')]

Scale structure can be represented as a list of successive intervals called steps >>> print(major.steps) [Absterval('2'), Absterval('2'), Absterval('b2'), Absterval('2'), Absterval('2'), Absterval('2'), Absterval('b2')] >>> print(minor.steps) [Absterval('2'), Absterval('b2'), Absterval('2'), Absterval('2'), Absterval('b2'), Absterval('2'), Absterval('2')]

Scale steps structure can be divided in two tetracords with an intermediate interval >>> print(major.tetracord1) ['2', '2', 'b2'] >>> print(major.tetracord2) ['2', '2', 'b2'] >>> print(major.inter_tetracord) ['2']

Scale steps and scale degrees are Absterval objects, Absterval methods can be called on Abscale.steps and Abscale.degrees >>> major.steps[0]+major.steps[1]+major.steps[2] Absterval('4') >>> print(major.steps[0]+major.steps[1]+major.steps[2]) Perfect fourth >>> major.degrees[1]-major.degrees[0] Absterval('2') >>> print(major.degrees[1]-major.degrees[0]) Major second

**alter**(*key*, *value*)

```
>>> major=Abscale('2212131')
>>> major.alter(2,-1)
>>> print(major)
['1', '2', 'b3', '4', '5', 'b6', '7', '8']
```

**harmonization**(*degree*, *size=3*)

```
>>> major=Abscale('2212221')
>>> chord=major.harmonization(1,3)
>>> print(chord.degrees)
[Absterval('1'), Absterval('3'), Absterval('5')]
>>> print(chord.code)
['1', '3', '5']
```

**property inter_tetracord**

**property tetracord1**

**property tetracord2**

**class** musenot.scales.**Scale**(*code*, *root*)

Bases: musenot.scales.Abscale

```
>>> a=Scale('2212221', 'c')
>>> print(a.root)
c
```

**harmonization**(*degree*, *size=3*)

```
>>> a=Scale('2212221', 'c')
>>> b=a.harmonization(1,3)
>>> print(b.notes)
[Absnote('c'), Absnote('e'), Absnote('g')]
```

**transpose**(*interval*)

```
>>> a=Scale('2212221', 'c')
>>> a.transpose('b3')
>>> print(a)
eb,f,g,ab,bb,c,d,eb
>>> a.transpose('-b3')
>>> print(a)
c,d,e,f,g,a,b,c
>>> a.transpose('3')
>>> print(a)
e,f#,g#,a,b,c#,d#,e
```

musenot.scales.**scaleify**(*notes*)

```
>>> a=scaleify(('a','b','c','d','e','f','g','a'))
>>> print(a.notes)
[Absnote('a'), Absnote('b'), Absnote('c'), Absnote('d'), Absnote('e'), Absnote('f'),
↪ Absnote('g'), Absnote('a')]
>>> print(a.degrees)
[Absterval('1'), Absterval('2'), Absterval('b3'), Absterval('4'), Absterval('5'),␣
↪Absterval('b6'), Absterval('b7'), Absterval('8')]
```

## 1.5 musenot.chords module

**class** musenot.chords.**AbsChord**(*code*)

Bases: object

```
>>> degrees = ['1', 'b3', '5']
>>> a=AbsChord(degrees)
>>> print(a.degrees)
[Absterval('1'), Absterval('b3'), Absterval('5')]
>>> a.invert(0)
[Absterval('1'), Absterval('b3'), Absterval('5')]
>>> a.invert(1)
[Absterval('b3'), Absterval('5'), Absterval('1')]
>>> a.invert(2)
[Absterval('5'), Absterval('1'), Absterval('b3')]
>>> len(a)
3
>>> a[1]+=1
>>> print(a.degrees)
[Absterval('1'), Absterval('3'), Absterval('5')]
```

**invert**(*index*)

**class** musenot.chords.**Chord**(*notes*)

Bases: musenot.chords.AbsChord

```
>>> notes=[ 'c', 'e', 'g']
>>> myChord = Chord(notes)
>>> print(myChord.notes)
[AbsNote('c'), AbsNote('e'), AbsNote('g')]
>>> myChord.root
AbsNote('c')
>>> myChord.degrees
[Absterval('1'), Absterval('3'), Absterval('5')]
>>> len(myChord)
3
```

**property root**

## 1.6 musenot.dictionary module

CHAPTER

TWO

# INDICES AND TABLES

- genindex
- modindex
- search

# INDEX

## Symbols



## A



## C



## D



## F



## H



## I



## M



## N



## R



## S

## T