Sator Documentation

Release .01

Caleb Smith

March 13, 2012

CONTENTS

A python module for atonal music analysis.

CHAPTER

ONE

FEATURES

- Create tone rows, pitch class sets and pitch sets.
- **Each pitch or pitch class set can have its own properties including:**
    - Ordered vs. Unordered
    - Multiset vs. Unique element sets
    - A configurable Modulus
    - Definable canonical operators - (TTO's used to determine SC membership)
- Easily construct matrices, find prime forms and interval class vectors.
- Compare pitch or pitch class sets with various similarity relations
- Explore non-standard pitch class spaces.

CHAPTER

TWO

REQUIREMENTS

- There are no requirements for sator.

CHAPTER

THREE

INSTALLATION

Sator is available on PyPI, so the easiest way to install it is to use pip:

```
pip install sator
```

CHAPTER

FOUR

TESTING

- Use runtests.py to run the test suite

CHAPTER

FIVE

AUTHORS

Caleb Smith

CHAPTER

SIX

DOCUMENTATION

The documentation is hosted on Read the Docs and is available here

6.1 Contents

6.1.1 Overview

In this overview, we will discuss the most essential parts of sator, but it is assumed that the reader is well versed in atonal theory.

Constructing Tone Rows and Pitch/Pitch Class Sets

The core module in sator contains the three essential classes for instantiating and manipulating tone rows, pitch sets, and pitch class sets.

To construct rows and sets, import the ToneRow, PCSet, and PSet classes from the core module as follows:

```
from sator.core import ToneRow, PCSet, PSet
```

To instantiate an empty pitch set, or pitch class set, use:

```
a = PSet()
b = PCSet()
```

- ToneRow objects are excluded from the above example because by definition, they cannot be empty.

The classes' constructors take an optional number of positional arguments as pitches or pc's. These arguments can be integers, lists, tuples, sets of integers, or another ToneRow, PCSet, or PSet object. Any of the following are both valid and equivalent:

```
a = PSet([0, 2, 4, 6, 8])
a = PSet(0, 2, 4, 6, 8)
a = PSet(0, [2, 4], 6, 8)
b = PSet(a)
```

The constructors also take several optional keyword arguments. For further details, refer to *Constructor Options*

For more information about how pitch/pitch class data is stored and retrieved and how to instantiate objects from objects of other classes refer to *Data Inspection* Casting objects from PSet, PCSet, or ToneRow to another works as expected, but it is not recommened to cast in both directions arbitrarily as data may be lost. Refer to "data inspection" above for more details.

While PSet objects contain pitch data, it is neccessary and desirable to have upper and lower limits. Rahn numbers are assumed, and pitches may be within 10 octaves of C4 (e.g. Rahn number 0). A PSet with pitches outside of this range will have the octave of the offending pitches reduced or raised 10 octaves appropriately.

Operating on Tone Row and Pitch/Pitch Class objects

Each of these objects can be iterated over, has a length, and with the exception of tone rows, can have additional pitches or pitch classes added or removed from them. Each class also supports an insert and copy method. The following example introduces an overview of the supported operators:

```
a = PCSet([0, 3, 9], ordered=True)
a = a + 11
a = a - 3
print a
Out: [0, 9, 11]
for pc in a:
    print pc
Out: 0
Out: 9
Out: 11
print len(a)
Out: 3
a.insert(0, 10)
print a
Out: [10, 0, 9, 11]
b = a.copy()
print b
Out: [10, 0, 9, 11]
```

Refer to *Operators* for more specifics on these operators.

TTO's

At the heart of atonal music are the twelve tone operators, or TTO's. Each of sator's classes have methods for these, which are detailed at *TTO's*

Below is a simple example of using each to modify a PCSet in place:

```
a = PCSet(0, 1, 3)
a.i()
print a
Out: [0, 9, 11]
a.t(6)
print a
Out: [3, 5, 6]
a.m()
Out: [1, 3, 6]
a.t_m(6, 7)
print a
Out: [0, 1, 3]
```

Attributes, Generators, and Properties

Sator core class objects have various boolean attributes such as ordered and multiset. They also have several property and generator methods. For more information on each topic, refer to the relevant links below:

Attributes

Generators

Properties and Static Methods

Tone Rows

ToneRow objects share many of the same methods as PCSet and PSet methods, but sometimes these methods have different or limited meaning. ToneRow objects also have many unique methods such as: P, R, I, RI, M, MI, RM, RMI, and swap.

Refer to *Tone Rows* for more information

Similarity Relations

Similarity relations are imported from sator.sim and are used to make various kinds of comparisons between pitch or pitch class sets. For example:

```
from sator.core import PCSet
from sator.sim import m, c, z
a = PCSet(0, 1, 2, 4, 7, 9)
b = PCSet(0, 1, 3, 5, 6, 8)
print c(a, b)
Out: True
print z(a, b)
Out: True
print m(a, b)
Out: False
```

Refer to *Similarity Relations* for more information.

6.1.2 Constructor Options

The PCSet and PSet classes take the following keyword arguments:

Key	Type	Description	Example	Default
mod	int	Set the object's modulus	mod=7	12
ordered	boolean	Is the set ordered or unordered?	ordered=True	False
multiset	boolean	Is the set a multiset?	multiset=True	False

- Tone rows are ordered by definition, and can not be multisets, these kwargs have no affect when constructing ToneRow objects.
- Constructing a tone row with fewer pitch classes than its modulus is by definition a pitch class set, and not a tone row. As a result, you must use the mod= kwarg when constructing a tone row with a modulus less than 12.
- The modulus must be greater than 0 and less than 32. Other values will raise an InvalidModulus exception.

6.1.3 Data Inspection

ToneRow, PCSet and PSet instances store all of their data in the pitches field, while other data such as pitch classes are handled as properties, which dervive from pitches. For example, a PCSet may have the pitches [0, 13, -1] and it's pcs property would output [0, 1, 11].

The following table shows the available properties/fields along with a description for each.

Property	Description
pitches	An ordered list of pitches
pcs	An ordered list of pitch classes
uo_pitches	An unordered list of pitches
uo_pcs	An unordered list of pitch classes
ppc Best representation of the object as described below.	

The following rules are used to determine what the ppc property should output:

- Output pitches for PSets, pitch classes for everthing else
- Output the pitches or pitch classes in order if the ordered attribute is set to True
- Output the pitches or pitch classes in ascending order if the ordered attribute is set to False (Default)

Each class uses str(ppc) for its __repr__, so ppc is used to output an object's basic representation. Unless another property is used explicitly, an object's pitches field is used for copying or instantiating another object. For example:

```
a = PSet(0, -1, 18)
b = PCSet(a)
c = PSet(b)
print c
Out: [0, -1, 18)
print c == a
Out: True
```

However, if a pitch collection contains different pitches of the same pitch class, data can be lost in conversion as in the following example:

```
a = PSet(0, 6, 18)
b = PCSet(a)
c = PSet(b)
print c
Out: [0, 6]
print c == a
Out: False
```

Because preservation of pitch data is not guaranteed casting objects from PSet to PCSet and back to PSet, is not recommended. PCSet objects have pitch class only methods such as the Mm TTO, while PSet objects contain pitch set only methods, such as finding the root.

ToneRows can also be instantiated from PSet or PCSet instances, but they must have each possible pitch class given the modulus of the object. When instantiating a new ToneRow with a modulus other than 12, it must be specified as a kwarg as in the following example:

```
a = PSet(0, 2, 5, 3, 4, 1, 6)
b = ToneRow(a, mod=7)
```

The following are important considerations when instantiating and working with tone rows:

- ToneRows must have each possible pitch class. If instantiating with fewer, an IncompleteToneRow exception is raised.
- ToneRows are ordered by definition and can not have their ordered field set to False
- ToneRows can not be multisets.

6.1.4 Operators

Iteration and Length

ToneRow, PCSet, and PSet objects can be iterated over, which returns each pitch or pitch class represented by the object's .ppc property. The set's ordered property is respected by the iterator, which returns each element in the order given for ordered sets, and in ascending order for unordered sets. For example:

```
a = PCSet([0, 3, 5, 6, 9], ordered=False)
for pc in a:
    print pc
```

Returns 0, 3, ... 9 However:

```
a = PCSet([9, 5, 3, 2], ordered=True)
for pc in a:
    print pc
```

Returns 9, 5, ... 2

Similarly the len() property returns the length of the object's .ppc property. The length of the set is not the same as the set's cardinality. Refer to `cardinality()` to see the distinction between `len()` and `cardinality()`. This is best illustrated by the following example, as a contains 3 pitches, and only 1 pitch class:

```
a = PSet(11, -1, 23)
print a.cardinality
Out: 1
print len(a)
Out: 3
```

Evaluation

ToneRow, PCSet and PSet objects can be compared to each other for equality and inequality using the == and != operators. Use the following list to see the criteria for equality amongst these objects:

- If one object is ordered, the pitches or pitch classes must be in the same order.
- One object can be a list, tuple, or set rather than a ToneRow, PCSet, or PSet object.
- A ToneRow is equal to another object if they contain the same pitch classes in the same order.
- Two PSets are equal if they contain the same pitches.
- Two PCSets are equal if they contain the same pitch classes.
- A PSet and PCSet are equal if they contain the same pitch classes.

Addition and Subtraction

Pitches or pitch classes can be added or removed from an existing set with the += or = + idioms. The addition and subtraction operators each return a new object, so it can also be used to instantiate a new object. Integers, lists, tuples, sets and instances of sator core objects can all be added or subtracted. For examle:

```
a += [3, 9]
b = b + [0]
c = a + b
```

Addition and subtraction can also be used for evaluation such as:

```
a = PSet(0, 1, 11)
a + [3, 9] == [0, 1, 3, 9, 11]
Out: True
a - 1 == [0, 11]
Out: True
```

- When subtracting from a multiset, each instance of the pitch or pitch class will be removed.

When adding or subtracting a pitch or pitch class is not possible, because it is already present or not in the instance, no errors are raised. For example:

```
a = PSet(0, 1, 11)
print a - 3
Out: [0, 1, 11]
```

Insert

Similarly, pitch and pitch sets, but not tone rows, have an insert method. This method is meaningful only for sets which have their ordered field set to True but still adds the pitch or pitch class nonetheless. Insert takes an index and a new pitch/pitch class as it's arguments. This method can be used as follows:

```
a = PCSet([0, 4, 8], ordered=True)
a.insert(1, 2)
print a
Out: [0, 2, 4, 8]
```

Copy

Objects are mutable in Python, which may lead to unexpected behavior. For example:

```
a = PSet(0, 3, 6, ordered=True)
b = a
b += 8
print a
Out: [0, 3, 6, 8]
```

To instantiate a new ToneRow, PCSet or PSet from another use the copy method as shown below:

```
a = PSet(0, 3, 6, ordered=True)
b = a.copy()
b += 8
print a
Out: [0, 3, 6]
print b
Out: [0, 3, 6, 8]
print b.ordered
out: True
```

Clear

To remove all pitches or pitch classes from a PCSet or PSet use the clear method as shown here:

```
a = PSet(0, 3, 6)
a.clear()
print a
Out: []
```

- Clear is not a method of ToneRow, because tone rows can never be empty

6.1.5 TTO's

TTO is an acronym for "Twelve Tone Operators."

Using TTO's in place

To modify a ToneRow, PCSet, or PSet by TTO in place, use the t, i, m, mi, and t_m methods. The table below shows each method's associated TTO, arguments and defaults. Arguments enclosed in brackets are optional, and use the default if not provided.

Name	TTO	Arguments	Defaults
t	Tn	n	NA
i	TnI	[n]	n=0
m	TnM	[n]	n=0
mi	TnMI	[n]	n=0
t_m	TnMm	n, m	NA

- m, mi, and t_m are not possible for pitch sets. Therefore, these methods are only available for ToneRow and PCSet instances.

Below are some examples:

```
a = PCSet(0, 4, 9)
a.t(1)
print a
Out: [1, 5, 10]
a.i()
print a
Out: [2, 7, 11]
a.m()
print a
Out: [7, 10, 11]
a.mi()
print a
Out: [1, 5, 10]
a.t_m(1, 11)
print a
Out: [0, 3, 8]
```

Returning new instances via a TTO

To return new set or row instances modified by a TTO, import and use the following functions:

```
>>> from sator.core import transpose, invert, multiply, transpose_multiply
```

The table below shows each function's assocatied TTO, arguments and defaults. Arguments enclosed in brackets are optional, and use the default if not provided.

Name	TTO	Arguments	Defaults
transpose	Tn	object, n	NA
invert	TnI	object, [n]	n=0
multiply	T0Mm	object, [m]	m=5
transpose_multiply	TnMm	object, n, m	NA

The following are some examples of each:

```
a = PCSet(0, 4, 9)
b = transpose(a, 1)
print b
Out: [1, 5, 10]
c = invert(a)
print c
Out: [0, 3, 8]
d = transpose_multiply(a, 3, 7)
print d
Out: [3, 6, 7]
```

- Multiply and transpose_multiply will raise an InvalidTTO exception if they are called with a PSet

6.1.6 Attributes

PCSet and PSet instances have attributes which can be set via the methods listed below. ToneRow instances have some but not all of these methods and exceptions are noted after the method's description.

Mod

mod([modulus])

Sets the modulus of the object to modulus. Without an argument, the current modulus is returned. The modulus must be greater than 0 and less than 32. Other values will raise an InvalidModulus exception.

Default_m

default_m([m])

Sets the default m that is used for Mm (and MI) for TTO's and for determining the prime form if Mm is canonical (Refer to canon below for more). Without an argument, the current default_m is returned

Multiset

multiset([boolean])

If boolean is True, the object is now a multiset and will accept new pitches or pitch classes that are duplicates of existing set members. If boolean is False, the object is not a multiset and will not accept new pitches or pitch classes that are duplicates of existing members. Without an argument, the current multiset status is returned.

- ToneRow instances have multiset set to False and it can not be multisets.

Ordered

ordered([boolean])

If boolean is True, the object will be considered an ordered set, and its __repr__ will show the pitches or pitch classes in the order they were added. Equivalence tests will consider the order of the set. If boolean is False, the object will be considered an unordered set, and its __repr__ will show the pitches or pitch classes in ascending numerical order regardless of the order in which they were added. Equivalence tests will not consider the order of the set, only its members. Unordered sets maintain order internally, so they do not loose the order of pitches/pitch classes when ordered is set to True.

- ToneRow instances have ordered set to True and can not be unordered.

Canon

canon(t, i, m) where t, i, and m are booleans.

Set the status of Tn, TnI, and TnM as canonical operators. These operators are used for determining set-class membership. The default is to use Tn/TnI, which would result from calling .canon(True, True, False). As an example, to use only Tn, as some theorists propose, use canon(True, False, False). To use Tn/TnI/TnM use .canon(True, True, True)

- Set-class membership is irrelevant for tone rows since they are all members of the aggregate. ToneRow instances do not have a canon method.

get_canon()

Returns a three tuple with the current status of canonical operators. If the canonical operators are Tn/TnM, the return would be (True, False, True)

- ToneRow instances do not have this method.

6.1.7 Generators

All of the generator methods are prefaced by each, sub or super. Below is a brief description of all of the instance generator methods:

- each_n() - Yields each possible n for the object's modulus. (0 - 11 for mod 12)
- each_tto() - Yields a two tuple in the form of (n, m) for every possible TTO that can be performed on an object.
- each_set() - Yields each possible unordered set for the object's modulus
- each_card() - Yields each unordered set with the same cardinality as the object, for the object's modulus
- each_prime() - Yields each unique set-class for the object's modulus
- each_permutation() - Yields every possible ordered set or tone row of the given set or tone row.
- subsets() - Yields each subset of the given object (depth first)
- subprimes() - Yields the unique set-classes of the subsets of the given object
- supersets() - Yields each superset of the given object (depth first)
- superprimes() - Yields the unique set-classes of the supersets of the given object

The each methods do not take any arguments, while the super and sub methods optionally take one argument. If given, the subsets or supersets will terminate recursion after reaching the cardinality specified.

The following generators are class methods, take the arguments listed, and yield results similar to their instance method counterparts.

- each_n_in_mod(modulus) - Yields each possible n where n is 0 <= n < modulus
- each_set_in_mod(modulus) - Yields each possible unordered set for the given modulus
- each_prime_in_mod(modulus) - Yields each unique set-class for the given modulus
- each_card_in_mod(cardinality, modulus) - Yields each unordered set with the given cardinality in the given modulus.
- each_prime_in_card_mod(cardinality, modulus) - Yields each unique set-class with the given cardinality in the given modulus.

6.1.8 Properties and Static Methods

The properties and methods are grouped into categories below.

Static Methods

fortename(string) Returns a new unordered PCSet instance whose Forte name is equivalent to the first argument, which is a string.

For example:: a = PCSet.forte_name('3-11') print a Out: [0, 3, 7]

fromint(integer, [modulus=12]) Returns a new unordered PCSet instance whose integer representation is equal to the first argument. Takes an optional keyword argument modulus, that defaults to 12.

For example:: a = PCSet.fromint(343) print a Out: [0, 1, 2, 4, 6, 8] print a.setint Out: 343

The static or class methods that are generators are described under *Generators*

Properties for TTO rotations

Rotations

- t_rotations - A list of objects representing each possible transposition of the given object
- i_rotations - A list of objects representing each possible transpostiion of the given object after inversion
- m_rotations - A list of objects representing each possible transpostiion of the given object after Mm, where m is the default_m of the given object
- mi_rotations - A list of objects representing each possible transpostiion of the given object after MI, where MI is the inverse operator of Mm, and m is the default_m of the given object.
- all_rotations - A list of objects representing each possible transformation of the given object under a TTO

Set methods

These property methods have a limited meaning for ToneRow objects and are only available to PSet and PCSet objects.

- cardinality - Returns the cardinality of the set.
- setint - Returns the set's integer representation. An unordered PCSet of the set can be derived from this integer and the fromint static method.
- pcint - Returns the integer representation of the set's prime form.
- invariance_vector - Returns a list of (n, m) pairs in which each is a TnMm operation for which the set is invariant.

These methods take one positional argument, which can be any of PSet, PCSet, list, tuple, or set and return a PCSet or boolean as appropriate. They mimic the Python built-in set methods of the same name. In the description, A is used to denote the current object, and B is the object that is passed in as an argument.

- union(other) - The union of the two objects. The resulting object has all of the elements of both. (A or B)
- intersection(other) - The intersection of the two objects. The resulting object has only the elements that are in both. (A and B)
- difference(other) - The difference of the two objects. The resulting object has the elements of A, excepting those in B.
- symmetric_difference(other) - The symmetric difference of the two objects. The resulting object has all of the elements of A, excepting those in B, as well as all of the elements of B, excepting those in A.

- issuperset(other) - Returns True if the current object is a superset of the argument object, otherwise False
- issubset(other) - Returns True if the current object is a subset of the argument object, otherwise False
- isdisjoint(other) - Returns True if the current object and argument object are disjoin, otherwise False

It is worthwhile to note that the behavior of some of these methods are duplicated elsewhere and that they are included here for ease of use with other set methods. The following methods have the same behavior as the method they are listed with:

- union - The + operator
- difference - The - operator

Set-Class

These properties are related to the object's set-class and are therefore not available to ToneRow objects, since all tone rows have the same set-class, which is the aggregate of the given modulus.

- prime - The set in prime form. (Use the canon method to change the canonical operators used.)
- prime_operation - Returns a two tuple in the form of (n, m) which would transform the set into its prime form under TnMm using .t_m(n, m)
- forte - Returns the Forte name of the set.
- icv - Returns the interval class vector of the set. N.B. - The first integer represents the number of occurences of IC 0, which some texts omit.
- ds - Returns the degrees of symmetry of the set. (The number of Tn/TnI operations for which the set is invariant)
- mpartner - Returns an unordered PCSet instance, which is the M-partner of the current set, which is a PSet or PCSet.
- zpartner - Returns an unordered PCSet instance, which is the Z-partner of the current set, which is a PSet or PCSet.
- literal_compliment - Returns an unordered PCSet, which represents the literal compliment of the set
- abstract_compliment - Returns an unordered PCSet, which represents the abstract compliment of the set. This is the same as the literal compliment in prime form.

Non-TTO transformations

The following non-TTO transformations are available for PCSet objects only

- c() - Change the given object in place to its literal compliment
- z() - Change the given object in place to its z-partner

Pitch Set Only Properties and Methods

The following properties are only available for PSet objects.

- root - Determine the root(s) of an ordered pitch set using Paul Hindemith's method. Returns a list with one root, or multiple roots if the root is indeterminate. If the set is unordered, ascending order is the assumed voicing.

Neo-Riemannian Transformations

Neo-Riemannian transformations and their related methods are only available for PSet objects. The following methods return new PSet instances, modified by the appropriate Neo-Riemannian Transformation. If the root, third, and fifth can not be found, or there are more than one, the given set is returned unmodified. These transformations preserve order and change only one pitch by one or two semitones (depending on the transformation) N.B. - While all major/minor triads are supported for all transformations, other trichords or sets with other cardinalities may give unexpected results. The only requirements are that the set has a modulus of 12, and a determinate root, third and fifth.

- P() - Parallel (C major becomes C minor)
- R() - Relative (C major becomes A minor)
- L() - Leading-Tone or "Leittonwechsel" (C major becomes E minor)

Composite transformations (i.e. transformations created by performing P, L, or R)

- S() - Slide (C major becomes Db minor) - equivalent to L().P().R()
- N() - Nebenverwandt (C major becomes F minor) - equivalent to R().L().P()
- H() - Hexatonic Pole - from Richard Cohn (C major becomes Ab minor) - equivalent to L().P().L() or P().L().P()

All Neo-Riemannian Transformations are involutions and are equivalent to a TnI operation, though order is preserved. The following methods are not transformations but are available for working with Neo-Riemannian transformations: When the argument takes a string as input, the string is not case-sensitive and characters other than p, l, r, n, h, and s are ignored

- transform(string) - Returns a set equal to the given set after each transformation in the string is performed successively.
- neo(string) - A generator that takes string input and yields the resulting set after each transformation.
- cycle(string) - A generator that performs neo successively until the original set is reached. In this way .cycle("plr") would generate the PLR cycle.
- paths(object) - Takes another PSet as an argument, and returns a list of strings in which each string lists the transformations that can be applied to the original set to reach the input set. Only P, L, and R are used for the search.

The following examples show the Neo-Riemannian transformation methods in action:

```
a = PSet(0, 4, 7, ordered=True)
b = a.H()
print a.P().L().P() == a.H() == b
Out: True
print a.paths(a)
Out: ['PP', 'RR', 'LL']
print a.paths(b)
Out: ['LPL', 'PLP']
c = a.transform('prl')
for each in a.neo('prl'):
    print each
Out: [0, 3, 7]
     [-2, 3, 7]
     [-2, 2, 7]
print c
Out: [-2, 2, 7]
for each in a.cycle('pl'):
    print each
Out: [0, 3, 7]
     [0, 3, 8]
```

```
[-1, 3, 8]
[-1, 4, 8]
[-1, 4, 7]
[0, 4, 7]
```

N.B. - The output for .cycle(‘pl’) is a hexatonic system (Richard Cohn). In the example above, this was the “Northern” cycle.

6.1.9 Tone Rows

Tone rows can be thought of as a very restrictive type of ordered pitch class collection in which all possible elements are present. As a result, there are many methods for sets that are meaningless, and therefore withheld from the ToneRow class, such as .ordered() and the methods for adding/removing pitch classes.

Inherited Methods and Methods with Different Meanings

The following are some methods which ToneRow objects inherit from a parent class common to sets (SetRowBase), but are in no way meaningful to tone rows: * pitches, pcs, uo_pitches, uo_pcs - Tone rows are always ordered pitch class collections, so these methods are irrelavent to the API and only exist for consistency internally. * multiset, ordered - These are overridden in ToneRow to do nothing

The following are some methods that have a different meaning or use with ToneRow objects: * Wheras the x_rotations (where x is t, i, m, or mi) methods are thought of as rotations when used with sets, here they constitue the T, I, M, and MI matrices * default_m - The prime form is not meaningful in tone rows (they are always the aggregate) but the default_m is used to determine the m used by the M and MI matrices, as well as any TTO operations containing M/MI on the row.

ToneRow Methods

The following are methods that are unique to ToneRow objects:

- swap(a, b) - Given indices a and b, swap the PC’s in the tone row that are in these positions.
- P - Prime form of the row
- I - Inversion of the row
- R - Retrograde of the row
- RI - Retrograde inversion of the row
- M - Mm of the row (m is supplied by default_m)
- MI - MmI of the row
- RM - Retrograde of the Mm of the row
- RMI - Retrograde of the MmI of the row

6.1.10 Similarity Relations

Similarity relations take two sator core objects as input and are imported from the sator.sim module. The following is a list of general similarity functions and their descriptions:

- m(a, b) - Returns True if the sets are M-partners, otherwise False
- c(a, b) - Returns True if the sets are abstract compliments, otherwise False

- z(a, b) - Returns True if the sets are Z-partners, otherwise False
- zc(a, b) - Returns True if the sets are abstract compliments and Z-partners, otherwise False

Each of these return a boolean.

The following is a list of similarity functions invented by Robert Morris:

- iv(a, b) - Returns a list of totals of each ordered pitch interval that can be expressed from a pc in a to a pc in b. -Morris' IV(a, b)
- sim(a, b) - Returns the sum of absolute value differences between the icv's of a and b (excluding icv0) -Morris' SIM(a, b)
- asim(a, b) - Returns the sim(a, b) divided by the total of possible differences in the icv's of a and b. Takes a boolean kwarg rational, which changes the return to a two tuple representing a rational number -Morris' ASIM(a, b)

6.1.11 Core module Index

class `sator.setrowbase.`**`PCBase`**

Base class for Tone rows and PC sets

`m`(*sub_n=0*)

Perform M on the object in place. If an argument is provided, also transpose the object in place by that amount.

`mi`(*sub_n=0*)

Perform M and I on the object in place. If an argument is provided, also transpose the object in play by that amount.

`t_m`(*sub_n*, *sub_m*)

Perform TnMm on the object in place, where n and m are positional arguments. If n is not provided, it defaults to 0. If m is not provided it defaults to the default_m of the object.

class `sator.setrowbase.`**`SetRowBase`**(**args*, ***kwargs*)

Base class for PC/pitch sets and tone rows

`all_rotations`

Return a flat list of objects for each possible TTO of the given object

`copy`(*pitches=None*, ***kwargs*)

Use to copy a ToneRow/PSet/PCSet with all data attributes.

`default_m`(*new_m=None*)

Takes one argument as the new default argument for M operations. (The default for Mod 12 is 5) Without an argument, returns the current default m.

`each_n`()

Yields a number for each possible member in the object considering its modulus. (An object with a modulus of 12 would return [0, 1, 2...11])

classmethod **`each_n_in_mod`**(*mod*)

Same as the instance method but takes one positional arg as the modulus

`each_permutation`()

A generator that yields ordered objects that represent each permutation of the given object.

`each_tto`()

Yields an (n, m) pair for each TTO that can be performed on the given object

i (*sub_n=0*)
Invert the object in place. If an argument is provided, also transpose the object in place by that amount.

i_rotations
Returns a list of objects for each possible transposition of the given object after inversion.

m_rotations
Returns a list of objects for each possible transposition of the given object after M.

mi_rotations
Returns a list of objects for each possible transposition of the given object after MI.

mod (*new_mod=None*)
Takes one argument as the new modulus of the system. Without an argument, returns the current modulus.

multiset (*value=None*)
Takes one boolean argument and determines if the object is a multiset. (The default for all objects is False. ToneRows cannot be multisets) Without an argument, returns the current setting.

ordered (*value=None*)
Takes one boolean argument and determines if the object is ordered. (The default for PCSets is False. The default for PSets is True.) Without an argument, returns the current setting.

pcs
Returns the pitch classes of the current set/row

ppc
Returns the pitches or pcs of a ToneRow, PCSet, or PSet taking into account the ordered and multiset settings.

t (*sub_n*)
Transpose the object in place by the argument provided.

t_rotations
Returns a list of objects for each possible transposition of the given object.

uo_pcs
Returns unordered pitch classes in ascending order

uo_pitches
Returns the unordered pitches in ascending order

class sator.setbase.**SetBase** (**args*, ***kwargs*)
Base class for PCSet and PSet

exception OnlySetableMethod
Exception to raise if the argument used can not be made into a set

SetBase.**abstract_compliment**
Returns a PCSet of the abstract compliment of the given object.

SetBase.**args_are_sets** (*f*, **args*, ***kwargs*)
Decorator to help with set methods. Ensures that args are sets

SetBase.**canon** (*t*, *i*, *m*)
Takes arguments in the form of (T, I, M) where each is a boolean. These arguments determine which TTO's are canonical. These TTO's are used to determine an object's set-class. (The default canonical operators are T and I, hence the common name Tn/TnI type). Ex:

> a.canon(True, False, False)
>
> a.prime would now give the Tn-type, and ignore inversion as an operation for determining set-class membership.

`SetBase.`**`cardinality`**
Returns the cardinality of the given object.

`SetBase.`**`clear`**`()`
Remove all pitches/pitch classes from the object.

`SetBase.`**`difference`**`(`**args*, ***kwargs*`)`
Return an instance that represents the difference of the current PSet or PCSet and another as the first and only positional argument.

`SetBase.`**`ds`**
Degrees of symmetry (number of Tn/TnI operations for which this set is invariant)

`SetBase.`**`each_card`**`()`
Yields every set with the same cardinality as the given object, taking into account the object's modulus.

classmethod `SetBase.`**`each_card_in_mod`**`(`*card*, *mod*`)`
Same as the instance method but takes two args for cardinality and modulus respectively

`SetBase.`**`each_prime`**`()`
Yields each unique set-class in the modulus of the given object.

classmethod `SetBase.`**`each_prime_in_card_mod`**`(`*card*, *mod*`)`
Yields every unique prime form with a given cardinality in the given modulus

`SetBase.`**`each_set`**`()`
Yields every possible set in the modulus of the given object.

`SetBase.`**`forte`**
Returns the Forte name for the given object.

static `SetBase.`**`forte_name`**`(`*fname*`)`
A static method that returns a PCSet object with the fort-name provided as a string argument. Returns an empty PCSet if the argument is not a string with a valid Forte name.

static `SetBase.`**`fromint`**`(`*integer*, *modulus=12*`)`
Static method that returns a PCSet object with pc's generated from their integer representation.

Ex: 0 = [], 1 = [0], 2 = [1], 3 = [0, 1], 4 = [2], 5 = [0, 2] PCSet.fromint(5) returns PCSet([0, 2])

`SetBase.`**`get_canon`**
Returns a three tuple showing which TTO's are canonical for the given object. These are in the order (T, I, M). Refer to canon() for details on how these settings are used.

`SetBase.`**`icv`**
Returns the interval class vector of the given object.

`SetBase.`**`insert`**`(`*place*, *pitch*`)`
Given arguments (place, pitch) insert the pitch at the place position. Take care to inspect the object's pitches attribute rather than it's __repr__, which uses the ppc attribute and may truncate duplicates. If the position is too great, the pitch will be appended at the end.

`SetBase.`**`intersection`**`(`**args*, ***kwargs*`)`
Return an instance that represents the intersection of the current PSet or PCSet and another as the first and only positional argument.

`SetBase.`**`invariance_vector`**
A property that returns the list of (n, m) pairs that produce an invariant set via TnMm

`SetBase.`**`isdisjoint`**`(`**args*, ***kwargs*`)`
Return True if the current PSet or PCSet is disjoint with another object taken as the first and only positional argument, otherwise False

`SetBase.`**`issubset`**`(`**args*, ***kwargs*`)`
: Return True if the current PSet or PCSet is a subset of another object taken as the first and only positional argument, otherwise False.

`SetBase.`**`issuperset`**`(`**args*, ***kwargs*`)`
: Return True if the current PSet or PCSet is a superset of another object taken as the first and only positional argument, otherwise False

`SetBase.`**`literal_compliment`**
: Returns a PCSet of the literal compliment of the given object.

`SetBase.`**`m_vector`**`(`*m*`)`
: Find David Lewin's M-vector. (Also described in Composition with Pitch Classes - Robert Morris) Finds the number of each set-class with cardinality m which are subsets of a given pitch class. The ICV is equivalent to the m-vector of a pitch class when m is 2.

`SetBase.`**`mpartner`**
: Return a PCSet for the M-partner of the given object.

`SetBase.`**`pcint`**
: Returns the integer representation of a given object in prime form.

`SetBase.`**`prime`**
: Return a PCSet that represents the given object in prime form, taking into account its canonical TTO's (set these with .canon(T, I, M)).

`SetBase.`**`prime_operation`**
: A property that returns (n, m) to perform on the given object via TnMm in order to obtain its prime form.

`SetBase.`**`setint`**
: Returns the integer representation for the unique PC's in a given object

`SetBase.`**`subprimes`**`(`*limit=0*`)`
: Yields the subsets of the given object which have a unique set-class. Takes an optional limit argument with the same behavior as subsets().

`SetBase.`**`subsets`**`(`*limit=0*`)`
: Yields the subsets of the given object. Takes an optional argument, which limits the subsets to those with a cardinality >= the limit. With no argument, returns all subsets.

`SetBase.`**`superprimes`**`(`*limit=0*`)`
: Yields the supersets of the given object which have a unique set-class. Takes an optional limit argument with the same behavior as supersets()

`SetBase.`**`supersets`**`(`*limit=0*`)`
: Yields the supersets of the given object. Takes an optional argument, which limits the supersets to those with a cardinality <= the limit. With no argument, returns all supersets.

`SetBase.`**`symmetric_difference`**`(`**args*, ***kwargs*`)`
: Return an instance that represents the symmetric_difference of the current PSet or PCSet and another as the first and only positional argument.

`SetBase.`**`union`**`(`**args*, ***kwargs*`)`
: Return an instance that represents the union of the current PSet or PCSet and another as the first and only positional argument.

`SetBase.`**`zpartner`**
: Property that returns the Z-partner of the given object if it exists, otherwise returns None.

class `sator.pset.`**`PSet`**`(`**args*, ***kwargs*`)`
: A class for pitch sets, which adds pitch set only methods.

H()
Hexatonic Pole (Cohn)

N()
Nebenverwandt

exception NotNeoR
Can not be transformed by a Neo-Riemannian operator

PSet.**S**()
Slide

PSet.**cycle**(*ts*)
Cycle through a list of transformations until the original set is reached.

PSet.**paths**(*other*)
A breadth first tree search to find the shortest path(s) from the given object to another. Takes one argument as the goal set, returns a list with one or more strings indicating the transformations between the given set and the goal set.

PSet.**root**
Find the root(s) of an ordered pitch set, using Paul Hindemith's method

PSet.**transform**(*ts*)
Returns the given object after performing the list of transformations given as a string argument. If the string is empty, the given object is returned.

class sator.pcset.**PCSet**(**args*, ***kwargs*)
A Class for pitch class sets which adds pitch class only methods

c()
Change the given object in place to its literal compliment.

z()
Change the given object in place to its Z-partner if possible. Otherwise leave the object unchanged.

sator.core.**transpose**(*a*, *n*)

sator.core.**invert**(*a*, *n=0*)

sator.core.**multiply**(*a*, *m=5*)

sator.core.**transpose_multiply**(*a*, *n*, *m=5*)

CHAPTER

SEVEN

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX