Abjad Documentation

Release 4423

Víctor Adán, Trevor Bača

June 12, 2011

CONTENTS

Abjad helps composers build up complex pieces of music notation in an iterative and incremental way. You can use Abjad to create a symbolic representation of all the notes, rests, staves, nested rhythms, beams, slurs and other notational elements in any score.

Note: The Abjad documentation is still very much a work in progress.

Start here

CHAPTER

ONE

ABJAD?

Abjad is an interactive software system designed to help composers build up complex pieces of music notation in an iterative and incremental way. You can use Abjad to create a symbolic representation of all the notes, rests, staves, nested rhythms, beams, slurs and other notational elements in any score. Because Abjad wraps the powerful LilyPond music notation package, you can use Abjad to control extremely fine-grained typographic details of all elements of any score, like the color and thickness of noteheads, dots, slurs and brackets. And because Abjad extends the Python programming language, you can use Abjad to make powerful and systematic changes to any part of any score. The scores that you make in Abjad can range in size from small examples of only one or two notes to full pieces of orchestral score worked out against many dozes of staves.

1.1 Abjad extends python

Python is an object-oriented, dynamic programming language developed by Guido van Rossum in the 1990s. Python is now widely used for everything from straightforward scripting applications to the development and deployment of complex distributed systems. The language and interpreter features of Python are similar to Ruby, though the syntax of Python more closely resembles C, C++ and Java than most other languages. Much has been written about the benefits of Python and we are happy to add our voice to the chorus. We find Python to be an excellent all-purpose language that scales well, tests well, develops quickly, and keeps total lines of code to a minimum. For more on the benefits (and some limitations) of Python, see our page on *Why Python is right for Abjad*.

1.2 Abjad extends lilypond

LilyPond is an open source music notation package invented by Han-Wen Nienhuys and Jan Niewenhuizen in the 1990s and still under development today. LilyPond is a command-line driven music typography system that allows for the generation of music notation of extremely high quality. LilyPond differs from other music engraving programs in a number of important ways, some of which were critical in our choice of LilyPond as the notational powerhouse underneath Abjad. LilyPond separates musical content and page layout. LiyPond affords typographic control over almost everything. And, perhaps most importantly, LilyPond implements the rhythmic model of western music correctly: broken tuplets, nonbinary meters, and durations that cross measure and line boundaries all work correctly out of the box. For these and other details relating to our selection of LilyPond as the notational engine for Abjad, see our page on *Why LilyPond is right for Abjad*.

Examples

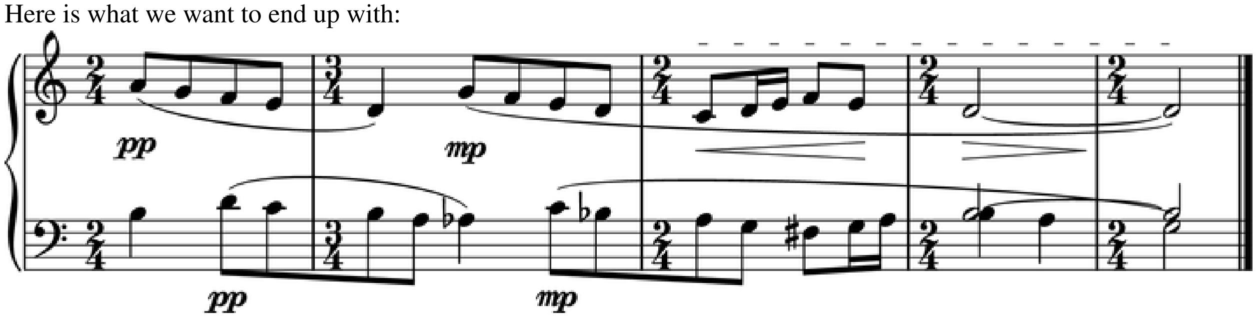
CHAPTER

TWO

BARTÓK: WANDERING

This example reconstructs the last five measures of Bartók's *Wandering* from Mikrokosmos vol. III. It demonstrates the use of many of the main classes in Abjad. The end result is just a few measures long, but the example covert most of the basic features you'll usually need.

Here is what we want to end up with:



2.1 The score

We will construct the fragment *top-down*, going from the high level containers to the details. We could have done it the other way around, but it will be easier to keep the big picture in mind this way. We encourage you to try rebuilding the example *bottom-down* as an exercise. First let's create the high level framework of the score:

```
abjad> piano = scoretools.PianoStaff([ ])
abjad> upper_staff = Staff([ ])
abjad> lower_staff = Staff([ ])
abjad> piano.append(upper_staff)
abjad> piano.append(lower_staff)
```

Here we created an empty piano staff and we've assigned it to the `piano` variable. Then we created two staves and assigned them to the `upper_staff` and `lower_staff` variables. Finally, we appended the staves to the piano staff.

2.2 The measures

Now let's add some measures to the framework:

```
abjad> m1 = Measure((2, 4), [ ])
abjad> m2 = Measure((3, 4), [ ])
abjad> m3 = Measure((2, 4), [ ])
```

```
abjad> m4 = Measure((2, 4), [ ])
abjad> m5 = Measure((2, 4), [ ])
abjad> upper_measures = [m1, m2, m3, m4, m5]
abjad> lower_measures = componenttools.clone_components_and_covered_spanners(upper_measures)
abjad> upper_staff.extend(upper_measures)
abjad> lower_staff.extend(lower_measures)
```

Notice that the *lower_measures* are simply copies of the *upper_measures*.

Note: The component tools house the different copying functions that Abjad provides for object duplication. The difference between them resides in the way each handles spanners attached to components during the duplication process.

Notice also that the measures are added to their corresponding staff via the `extend` method.

Note: Remember that `extend` is used for appending multiple objects that are grouped together in an iterable while `append` is used for single objects.

2.3 The notes

Now lets actually start adding some notes. Let's begin with the upper staff:

```
abjad> upper_measures[0].extend([Note(i, (1, 8)) for i in [9, 7, 5, 4]])
abjad> upper_measures[1].extend(notetools.make_notes([2,7,5,4,2], [(1, 4)]+[(1, 8)]*4))
abjad> notes = notetools.make_notes([0,2,4,5,4], [(1, 8), (1, 16), (1, 16), (1, 8), (1, 8)])
abjad> upper_measures[2].extend(notes)
abjad> upper_measures[3].append(Note(2, (1, 2)))
abjad> upper_measures[4].append(Note(2, (1, 2)))
```

Let's now create the notes for the lower staff. This will be a more intricate process that that needed for the upper staff. Notice that, for the upper staff, we simply added notes directly to the measures. This will not be possible for the lower staff due to the parallel *threads* (voices or melodic lines) found in the last two measures.

Note: The usual term for a melodic line is *voice*. Generally speaking, the language developed in Abjad uses the term *thread* to refer to this notion. A *thread* however, has a more concrete interpretation in Abjad. Please refer to the *Working with threads* section for a complete explanation of *threads*. The term Voice (with upper case) we use specifically for either the voice class or an instance of the class.

When two or more melodic lines are simultaneously present in the same staff, we need some way of grouping notes to disambiguate the paths of the possible melodic lines. We do this by creating explicit threads via the instantiation of the voice class. Here we create two threads: one called `main_voice` the other called `appendix_voice`. The threads are made explicit by instantiating voice classes and by naming them appropriately.

Let's move measure by measure in the construction of the lower staff:

```
abjad> main_voice_m1 = Voice(notetools.make_notes([-1, 2, 0], [(1, 4), (1, 8), (1, 8)]))
abjad> main_voice_m1.name = 'main_voice'
abjad> lower_measures[0].append(main_voice_m1)

abjad> main_voice_m2 = Voice(notetools.make_notes([-1, -3, -4, 0, -2], [(1, 8), (1, 8), (1, 4), (1, 8
main_voice_m2.name = 'main_voice'
abjad> lower_measures[1].append(main_voice_m2)
```

```
abjad> main_voice_m3 = Voice(notetools.make_notes([-3, -5, -6, -5, -3], [(1, 8), (1, 8), (1, 8), (1,
main_voice_m3.name = 'main_voice'
abjad> lower_measures[2].append(main_voice_m3)
```

Notice that every voice we create is equally named *main_voice* to guarantee the existence of a continuous thread. Many transformations and score traversal operations are possible across threads, so this is another reason why threads are important.

It is in the last two measures where we suddenly have two simultaneous voices in the lower staff. The new, second voice that seems to appear out of nowhere we will label *appendix_voice*:

```
abjad> appendix_voice_m4 = Voice([Note(-1, (1, 2))])
abjad> appendix_voice_m4.name = 'appendix_voice'
abjad> marktools.LilyPondCommandMark('voiceOne')(appendix_voice_m4)
abjad> main_voice_m4 = Voice([Note(-1, (1, 4)), Note(-3, (1, 4))])
abjad> main_voice_m4.name = 'main_voice'
abjad> marktools.LilyPondCommandMark('voiceTwo')(main_voice_m4)
abjad> p = Container([appendix_voice_m4, main_voice_m4])
abjad> p.is_parallel = True
abjad> lower_measures[3].append(p)
```

Note that the *number* property of the *appendix_voice* is set to `1`, and the *number* property of the *main_voice* is set to `2`. These determine the direction of the stem for each voice.

Note too that because both voices occur simultaneously in the score, we must put them in a parallel container to tell Abjad that they indeed run in parallel. Notice the setting of the boolean *parallel* property of the container. It is this container that is passed to the measure.

We now do a similar thing for the last measure:

```
abjad> appendix_voice_m5 = Voice([Note(-1, (1, 2))])
abjad> appendix_voice_m5.name = 'appendix_voice'
abjad> marktools.LilyPondCommandMark('voiceOne')(appendix_voice_m5)
abjad> main_voice_m5 =  Voice([Note(-5, (1, 2))])
abjad> main_voice_m5.name = 'main_voice'
abjad> marktools.LilyPondCommandMark('voiceTwo')(main_voice_m5)
abjad> p = Container([appendix_voice_m5, main_voice_m5])
abjad> p.is_parallel = True
abjad> lower_measures[4].append(p)
```

Let's see what we have up till now:

```
abjad> show(piano)
```

2.4 The details

Ok, let's add some detail. First, notice that the bottom staff has a treble clef, just like the top staff. Let's change that:

```
abjad> contexttools.ClefMark('bass')(lower_staff)
```

Now let's sprinkle some dynamic markings. For the top staff, we will add them to the first note of the first measure and the second note of the second measure. For the bottom staff, we will add dynamic markings to the second note of the first measure and the fourth note of the second measure. Note that because we created Voices inside the measures of the lo wer staff, we need to index those too:

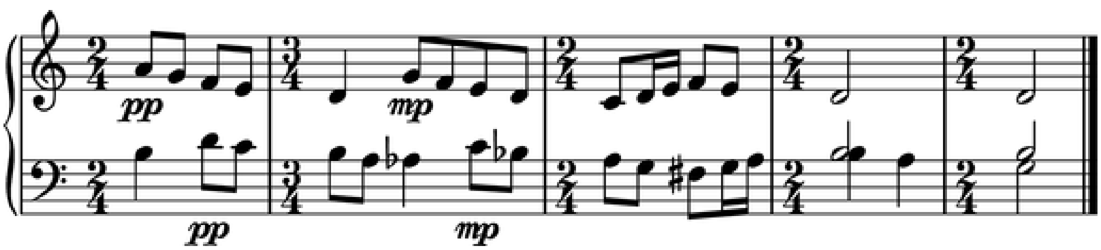
```
abjad> contexttools.DynamicMark('pp')(upper_measures[0][0])
abjad> contexttools.DynamicMark('mp')(upper_measures[1][1])
abjad> contexttools.DynamicMark('pp')(lower_measures[0][0][1])
abjad> contexttools.DynamicMark('mp')(lower_measures[1][0][3])
```

Let's also add a double bar line to the end of the piece:

```
abjad> marktools.LilyPondCommandMark('bar "|."', format_slot = 'closing')(lower_staff.leaves[-1])
```

Let's see how this is coming out:

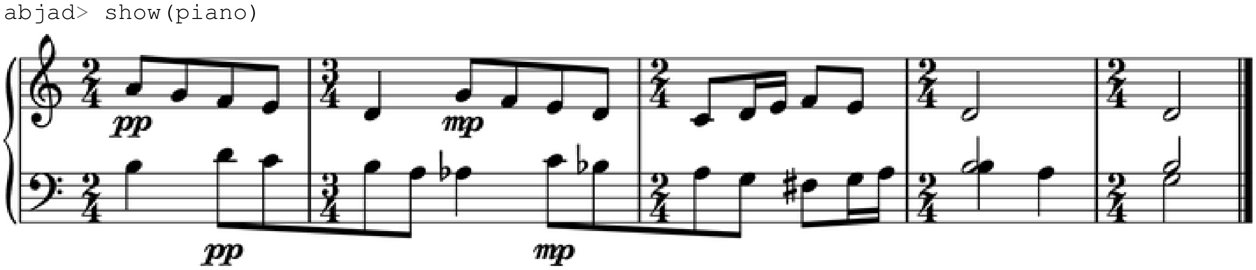
```
abjad> show(piano)
```



Notice that the beams of the eighth and sixteenth notes appear as you would usually expect: grouped by beat. We get this for free thanks to LilyPond's default rendering algorithm. This is not, however, the way Bartok notated his score. Let's set the beams as Bartok did, running some across the bar lines:

```
abjad> spannertools.BeamSpanner(upper_measures[0])
abjad> spannertools.BeamSpanner(lower_staff.leaves[1:5])
abjad> spannertools.BeamSpanner(lower_staff.leaves[6:10])
```

```
abjad> show(piano)
```



Now some slurs:

```
abjad> spannertools.SlurSpanner(upper_staff.leaves[0:5])
abjad> spannertools.SlurSpanner(upper_staff.leaves[5:])
abjad> spannertools.SlurSpanner(lower_staff.leaves[1:6])
abjad> slr = spannertools.SlurSpanner(lower_staff.leaves[6:13] + (main_voice_m4, main_voice_m5))
abjad> slr.position = 'down'
```

Notice that we store the last slur in the *slr* variable to change its `position` attribute to `'down'`. This does what you would expect!

Now hairpins:

```
abjad> spannertools.CrescendoSpanner(upper_staff.leaves[-7:-2])
abjad> spannertools.DecrescendoSpanner(upper_staff.leaves[-2:])
```

And a ritardando marking above the last seven notes of the upper staff:

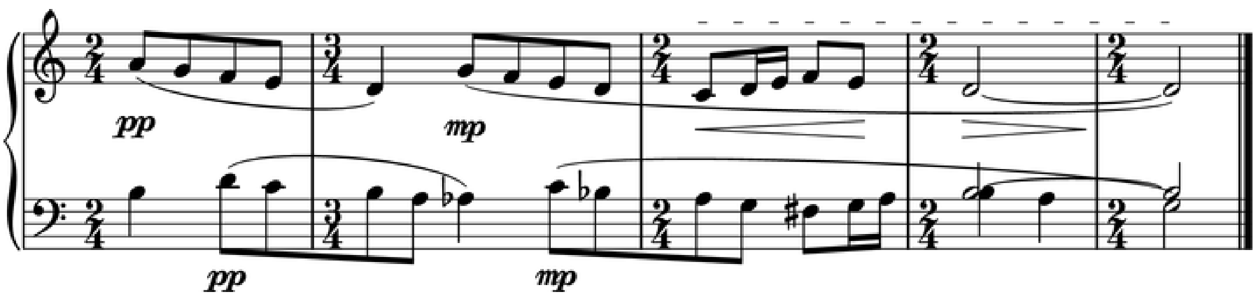
```
abjad> tx = spannertools.TextSpanner(upper_staff.leaves[-7:])
abjad> tx.bound_details__left__text = markuptools.Markup('ritard.')
```

And two ties connecting the last notes in the upper and lower staves:

```
abjad> tietools.TieSpanner(upper_staff[-2:])
abjad> tietools.TieSpanner([appendix_voice_m4[0], appendix_voice_m5[0]])
```

The final result:

```
abjad> show(piano)
```



CHAPTER

THREE

FERNEYHOUGH: UNSICHTBARE FARBEN

Mikhïal Malt analyzes the rhythmic materials of Ferneyhough's *Unsichtbare Farben* in *The OM Composer's Book 2*.

Malt details Ferneyhough's use of OpenMusic:

> The composer first created an exhaustive catalogue of rhythmic cells with two characteristics:
>
> 1. They are subdivided into two pulses, with proportions from `1/1` to `1/11`.
> 2. The second pulse is subdivided successively by `1`, `2`, `3`, `4`, `5` and `6`.

Here we recreate Malt's results in Abjad.

3.1 The proportions

First we define proportions:

```
abjad> proportions = [(1, n) for n in range(1, 11 + 1)]
abjad> proportions
[(1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (1, 7), (1, 8), (1, 9), (1, 10), (1, 11)]
```

3.2 The transforms

Then we make aliases:

```
abjad> make_tuplet = tuplettools.make_diminished_tuplet_from_duration_and_proportions_and_encourage_d
abjad> tie_chain_to_tuplet = tietools.tie_chain_to_diminished_tuplet_with_proportions_and_encourage_d
```

And create a helper:

```
def divide_tuplet(tuplet, n):
   last_tie_chain = tietools.get_tie_chain(tuplet[-1])
   proportions = n * [1]
   new = tie_chain_to_tuplet(last_tie_chain, proportions)
   return new
```

3.3 The rhythms

We set tuplet duration:

```
abjad> duration = Fraction(1, 4)
```

And make the rhythms:

```
for proportion in proportions:
   tuplets = [ ]
   for n in range(1, 6 + 1):
      tuplet = make_tuplet(duration, proportion)
      divide_tuplet(tuplet, n)
      tuplets.append(tuplet)
   staff.extend(tuplets)
```

3.4 The score

Finally we make the score:

```
abjad> staff = stafftools.RhythmicStaff(music)
abjad> score = Score([staff])
abjad> lily_file = lilyfiletools.make_basic_lily_file(score)
```

Configure containers:

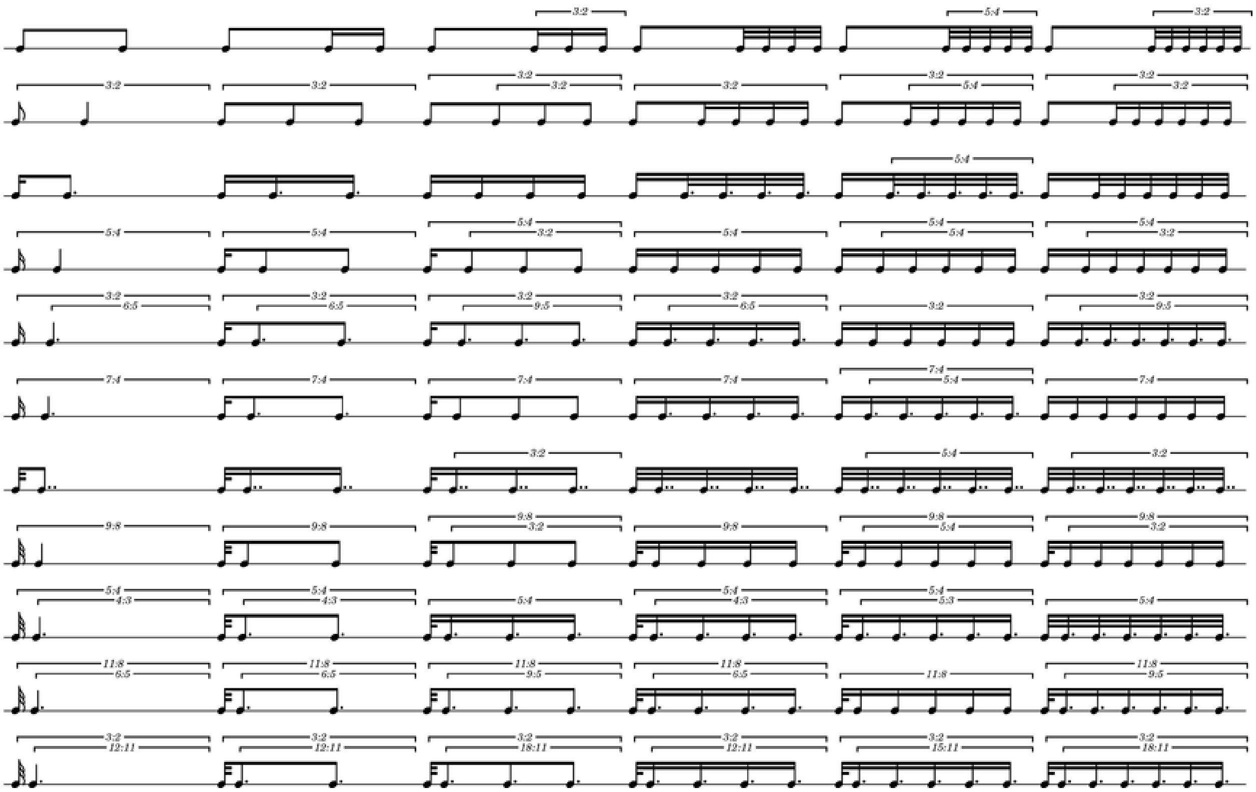
```
abjad> contexttools.TimeSignatureMark((1, 4))(staff)
abjad> score.override.bar_number.transparent = True
abjad> score.set.proportional_notation_duration = schemetools.SchemeMoment(1, 56)
abjad> score.set.tuplet_full_length = True
abjad> score.override.spacing_spanner.uniform_stretching = True
abjad> score.override.spacing_spanner.strict_note_spacing = True
abjad> score.override.tuplet_bracket.padding = 2
abjad> score.override.tuplet_bracket.staff_padding = 4
abjad> score.override.tuplet_number.text = schemetools.SchemeFunction('tuplet-number::calc-fraction-t
abjad> score.override.time_signature.stencil = False
abjad> score.override.bar_line.stencil = False
```

Configure the LilyPond file:

```
abjad> lily_file.default_paper_size = '11x17', 'portrait'
abjad> lily_file.global_staff_size = 12
abjad> lily_file.layout_block.indent = 0
abjad> lily_file.layout_block.ragged_right = True
abjad> lily_file.paper_block.ragged_bottom = True
abjad> space = schemetools.SchemePair('space', 18)
abjad> stretchability = schemetools.SchemePair('stretchability', 0)
abjad> vector = schemetools.SchemeVector(space, stretchability)
abjad> lily_file.paper_block.between_system_spacing = vector
```

And show the result:

```
abjad> show(lily_file)
```

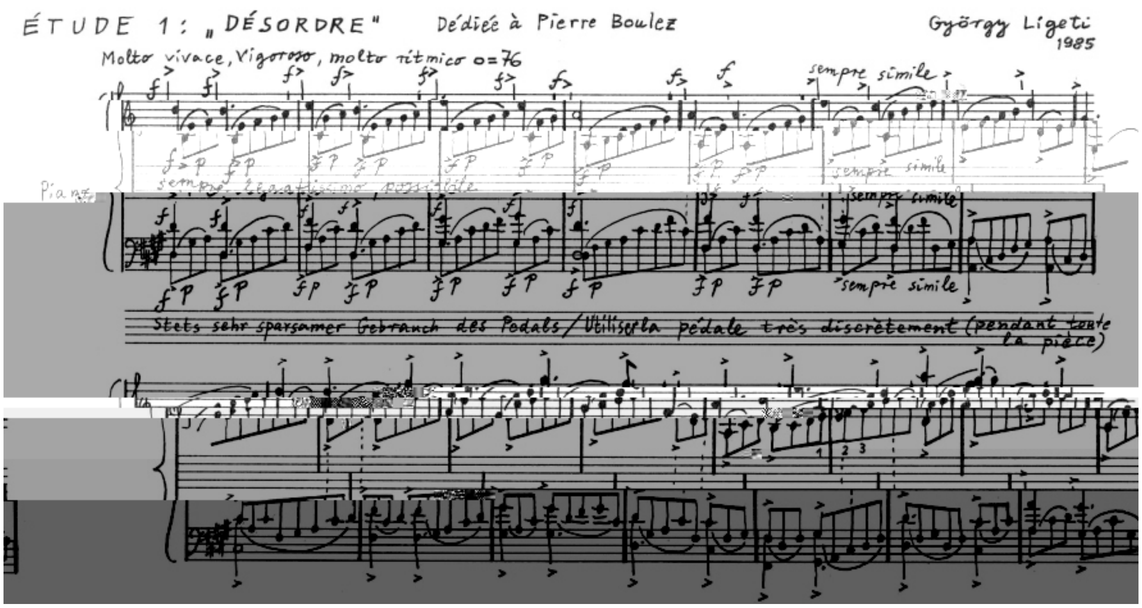


CHAPTER

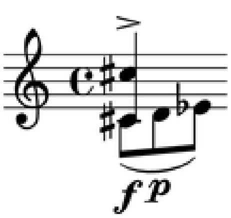
FOUR

LIGETI: DÉSORDRE

This example demonstrates the power of exploiting redundancy to model musical structure. The piece that concerns us here is Ligeti's *Désordre*: the first piano study from Book I. Specifically, we will focus on modeling the first section of the piece:



The redundancy is immediately evident in the repeating pattern found in both staves. The pattern is hierarchical. At the smallest level we have what we will here call a *cell*:



There are two of these cells per measure. Notice that the cells are strictly contained within the measure (i.e., there are no cells crossing a bar line). So, the next level in the hierarchy is the measure. Notice that the measure sizes (the meters) change and that these changes occur independently for each staff, so that each staff carries it's own sequence of measures. Thus, the staff is the next level in the hierarchy. Finally there's the piano staff, which is composed of the right hand and left hand staves.

In what follows we will model this structure in this order (*cell*, measure, staff, piano staff), from bottom to top.

4.1 The cell

Before plunging into the code, observe the following characteristic of the *cell*:

1. It is composed of two layers: the top one which is an octave "chord" and the bottom one which is a straight eighth note run.
2. The total duration of the *cell* can vary, and is always the sum of the eight note funs.
3. The eight note runs are always stem down while the octave "chord" is always stem up.
4. The eight note runs are always beamed together and slurred, and the first two notes always have the dynamic markings 'f' 'p'.

The two "layers" of the *cell* we will model with two Voices inside a parallel Container. The top Voice will hold the octave "chord" while the lower Voice will hold the eighth note run. First the eighth notes:

```
abjad> pitches = [1,2,3]
abjad> notes = notetools.make_notes(pitches, [(1, 8)])
abjad> spannertools.BeamSpanner(notes)
abjad> spannertools.SlurSpanner(notes)
abjad> contexttools.DynamicMark('f')(notes[0])
abjad> contexttools.DynamicMark('p')(notes[1])

abjad> voice_lower = Voice(notes)
abjad> voice_lower.name = 'rh_lower'
abjad> marktools.LilyPondCommandMark('voiceTwo')(voice_lower)
```

The notes belonging to the eighth note run are first beamed and slurred. Then we add the dynamic marks to the first two notes, and finally we put them inside a Voice. After naming the voice we number it `2` so that the stems of the notes point down.

Now we construct the octave:

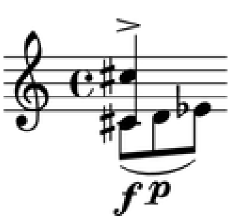
```
abjad> import math
abjad> n = int(math.ceil(len(pitches) / 2.))
abjad> chord = Chord([pitches[0], pitches[0] + 12], (n, 8))
abjad> marktools.Articulation('>')(chord)

abjad> voice_higher = Voice([chord])
abjad> voice_higher.name = 'rh_higher'
abjad> marktools.LilyPondCommandMark('voiceOne')(voice_higher)
```

The duration of the chord is half the duration of the running eighth notes if the duration of the running notes is divisible by two. Otherwise the duration of the chord is the next integer greater than this half. We add the articulation marking and finally ad the Chord to a Voice, to which we set the number to 1, forcing the stem to always point up.

Finally we combine the two voices in a parallel Container:

```
abjad> p = Container([voice_lower, voice_higher])
abjad> p.is_parallel = True
```

This results in the complete *Désordre cell*:



Because this *cell* appears over and over again, we want to reuse this code to generate any number of these *cells*. We here encapsulate it in a function that will take only a list of pitches:

```
def desordre_cell(pitches):
   '''The function constructs and returns a *Désordre cell*.
      - 'pitches' is a list of numbers or, more generally, pitch tokens.
   '''
   notes = [Note(p, (1, 8)) for p in pitches]
   spannertools.BeamSpanner(notes)
   spannertools.SlurSpanner(notes)
   contexttools.DynamicMark('f')(notes[0])
   contexttools.DynamicMark('p')(notes[1])
   v_lower = Voice(notes)
   v_lower.name = 'rh_lower'
   marktools.LilyPondCommandMark('voiceTwo')(v_lower)

   n = int(math.ceil(len(pitches) / 2.))
   chord = Chord([pitches[0], pitches[0] + 12], (n, 8))
   marktools.Articulation('>')(chord)
   v_higher = Voice([chord])
   v_higher.name = 'rh_higher'
   marktools.LilyPondCommandMark('voiceOne')(v_higher)
   p = Container([v_lower, v_higher])
   p.is_parallel = True
   ## make all 1/8 beats breakable
   for n in v_lower.leaves[:-1]:
      n.bar_line.kind = ''
   return p
```

Now we can call this function to create any number of *cells*. That was actually the hardest part of reconstructing the opening of Ligeti's *Désordre*. Because the repetition of patters occurs also at the level of measures and staves, we will now define functions to create these other higher level constructs.

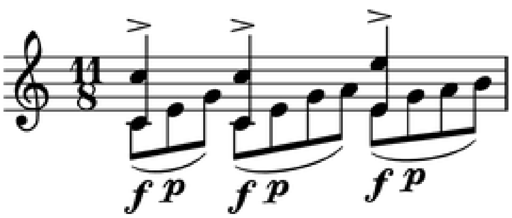
4.2 The measure

We define a function to create a measure from a list of lists of numbers:

```
def measure_build(pitches):
   '''Constructs a measure composed of *Désordre cells*.
      - 'pitches' is a list of lists of number (e.g., [[1,2,3], [2,3,4]])
   The function returns a DynamicMeasure.
   '''
   result = DynamicMeasure([ ])
   for seq in pitches:
      result.append(desordre_cell(seq))
```

The function is very simple. It simply creates a DynamicMeasure and then populates it with *cells* that are created internally with the function previously defined. The function takes a list *pitches* which is actually a list of lists of pitches (e.g., `[[1,2,3], [2,3,4]]`. The list of lists of pitches is iterated to create each of the *cells* to be appended to the DynamicMeasures. We could have defined the function to take ready made *cells* directly, but we are building the hierarchy of functions so that we can pass simple lists of lists of numbers to generate the full structure. To construct a Ligeti measure we would call the function like so:

```
abjad> measure = measure_build([[0,4,7], [0,4,7,9], [4,7,9,11]])
abjad> show(Staff([measure]))
```



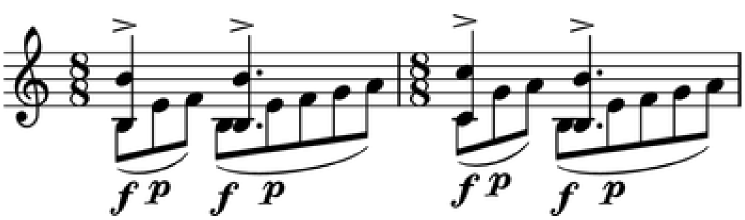
4.3 The staff

Now we move up to the next level, the staff:

```
def staff_build(pitches):
   '''Returns a Staff containing DynamicMeasures.'''
   result = Staff([ ])
   for seq in pitches:
      measure = measure_build(seq)
      result.append(measure)
   return result
```

The function again takes a plain list as argument. The list must be a list of lists (for measures) of lists (for cells) of pitches. The function simply constructs the Ligeti measures internally by calling our previously defined function and puts them inside a Staff. As with measures, we can now create full measure sequences with this new function:

```
abjad> pitches = [[[-1, 4, 5], [-1, 4, 5, 7, 9]], [[0, 7, 9], [-1, 4, 5, 7, 9]]]
abjad> staff = staff_build(pitches)
abjad> show(staff)
```



4.4 The score

Finally a function that will generate the whole opening section of the piece *Désordre*:

```
def desordre_build(pitches):
   '''Returns a complete PianoStaff with Ligeti music!'''
   assert len(pitches) == 2
   piano = PianoStaff([ ])
   ## build the music...
   for hand in pitches:
      seq = staff_build(hand)
      piano.append(seq)
   ## set clef and key signature to left hand staff...
   piano[1].clef.forced = stafftools.Clef('bass')
   piano[1].key_signature.forced = tonalitytools.KeySignature('b', 'major')
   return piano
```

The function creates a PianoStaff, constructs Staves with Ligeti music and appends these to the empty PianoStaff. Finally it sets the clef and key signature of the lower staff to match the original score. The argument of the function is a list of length 2, depth 3. The first element in the list corresponds to the upper staff, the second to the lower staff.

The final result:

```
abjad> top = [[[-1, 4, 5], [-1, 4, 5, 7, 9]], [[0, 7, 9], [-1, 4, 5, 7, 9]], [[2, 4, 5, 7, 9], [0, 5,
abjad> bottom = [[[-9, -4, -2], [-9, -4, -2, 1, 3]], [[-6, -2, 1], [-9, -4, -2, 1, 3]], [[-4, -2, 1,
abjad>
abjad> desordre = desordre_build([top, bottom])
abjad> show(desordre)
```

Now that we have the redundant aspect of the piece compactly expressed and encapsulated, we can play around with it by changing the sequence of pitches.

Note: In order for each staff to carry its own sequence of independent measure changes, LilyPond requires some special setting up prior to rendering. Specifically, one must move the *Timing_translator* from the score level to the level of staves. In this example we used the ‘tirnaveni’ template, which is configured to do just that. You may want to study this template (in the “templates” directory of the abjad distribution). Refer to the LilyPond documentation on Polymetric notation to learn all about how this works.

Reference manual — chords

CHAPTER

FIVE

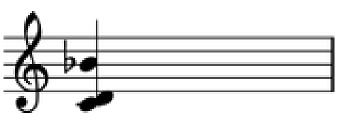
WORKING WITH CHORDS

5.1 Making chords from a LilyPond input string

You can make chords from a LilyPond input string:

```
abjad> chord = Chord("<c' d' bf'>4")
```

```
abjad> show(chord)
```

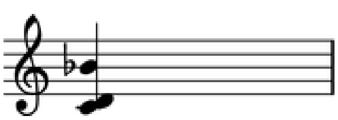


5.2 Making chords from chromatic pitch numbers and duration

You can also make chords from chromatic pitch numbers and duration:

```
abjad> chord = Chord([0, 2, 10], Duration(1, 4))
```

```
abjad> show(chord)
```



5.3 Getting all the written pitches of a chord at once

You can get all the written pitches of a chord at one time:

```
abjad> chord.pitches
(NamedChromaticPitch("c'"), NamedChromaticPitch("d'"), NamedChromaticPitch("bf'"))
```

Abjad returns a read-only tuple of named chromatic pitches.

5.4 Getting the written pitches of a chord one at a time

You can get the written pitches of a chord one at a time:

```
abjad> chord.pitches[0]
NamedChromaticPitch("c'")
```

Chords index the pitch they contain starting from `0` (just like tuples and lists).

5.5 Adding one pitch to a chord at a time

Use `append( )` to add one note to a chord.

You can add a pitch to a chord with a chromatic pitch number:

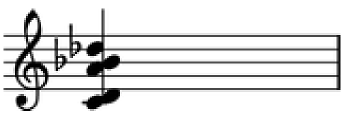
```
abjad> chord.append(9)
```

```
abjad> show(chord)
```

Or you can add a pitch to a chord with a chromatic pitch name:

```
abjad> chord.append("df''")
```

```
abjad> show(chord)
```



Chords sort their pitches every time you add a new one.

This means you can add pitches to your chord in any order.

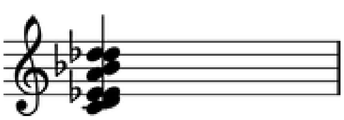
5.6 Adding many pitches to a chord at once

Use `extend( )` to add many pitches to a chord.

You can use chromatic pitch numbers:

```
abjad> chord.extend([3, 4, 14])
```

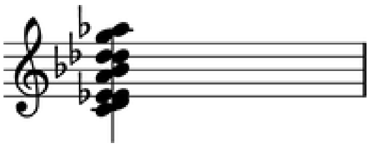
```
abjad> show(chord)
```



Or you can chromatic pitch names:

```
abjad> chord.extend(["g''", "af''"])
```

```
abjad> show(chord)
```

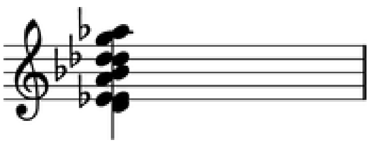


5.7 Deleting pitches from a chord

Delete pitches from a chord with `del( )`:

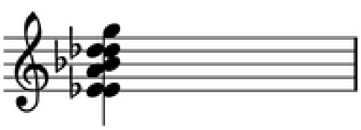
```
abjad> del(chord[0])
```

```
abjad> show(chord)
```



```
abjad> del(chord[0])
```

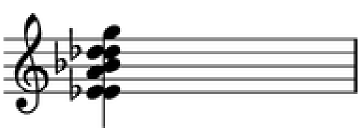
```
abjad> show(chord)
```



Negative indices work too:

```
abjad> del(chord[-1])
```

```
abjad> show(chord)
```



5.8 Formatting chords

Get the LilyPond input format of any Abjad object with `format`:

```
abjad> chord.format
<ef' e' a' bf' df'' d'' g''>4
```

Use `f( )` as a short-cut to print the LilyPond input format of any Abjad object:

```
abjad> f(chord)
<ef' e' a' bf' df'' d'' g''>4
```

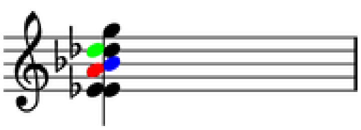
5.9 Working with note heads

Most of the time you will work with the pitches of a chord. But you can get the note heads of a chord, too:

```
abjad> chord.note_heads
(NoteHead("ef'"), NoteHead("e'"), NoteHead("a'"), NoteHead("bf'"), NoteHead("df''"), NoteHead("d''"),
```

This is useful when you want to apply LilyPond overrides to note heads in a chord one at a time:

```
abjad> chord[2].tweak.color = 'red'
abjad> chord[3].tweak.color = 'blue'
abjad> chord[4].tweak.color = 'green'

abjad> f(chord)
<
        ef'
        e'
        \tweak #'color #red
        a'
        \tweak #'color #blue
        bf'
        \tweak #'color #green
        df''
        d''
        g''
>4

abjad> show(chord)
```



5.10 Working with empty chords

Abjad allows empty chords:

```
abjad> chord = Chord([ ], Duration(1, 4))
Chord('<>4')
```

Abjad formats empty chords, too:

```
abjad> f(chord)
<>4
```

But if you pass empty chords to `show( )` LilyPond will complain because empty chords don't constitute valid LilyPond input.

When you are done working with an empty chord you can add pitches back into it chord in any of the ways described above:

```
abjad> chord.extend(["gf'", "df''", "g''"])

abjad> show(chord)
```

Reference manual — containers

CHAPTER

SIX

WORKING WITH CONTAINERS

6.1 Creating containers

Create a container with components:

```
abjad> container = Container([Note("ds'16"), Note("cs'16"), Note("e'16"), Note("c'16")])
```

```
abjad> show(container)
```

Or with a note-entry string:

```
abjad> container = Container("ds'16 cs'16 e'16 c'16 d'2 ~ d'8")
```

```
abjad> show(container)
```

6.2 Inspecting music

Return the components in a container with `music`:

```
abjad> container.music
(Note("ds'16"), Note("cs'16"), Note("e'16"), Note("c'16"), Note("d'2"), Note("d'8"))
```

Or with a special call to `__getslice__`:

```
abjad> container[:]
[Note("ds'16"), Note("cs'16"), Note("e'16"), Note("c'16"), Note("d'2"), Note("d'8")]
```

6.3 Inspecting length

Get the length of a container with `len( )`:

```
abjad> len(container)
6
```

6.4 Inspecting duration

Contents duration equals the sum of the duration of everything inside the container:

```
abjad> container.duration.contents
Duration(7, 8)
```

6.5 Adding one component to the end of a container

Add one component to the end of a container with `append`:

```
abjad> container.append(Note("af'32"))

abjad> show(container)
```

6.6 Adding many components to the end of a container

Add many components to the end of a container with `extend`:

```
abjad> container.extend([Note("c''32"), Note("a'32")])

abjad> show(container)
```

6.7 Finding the index of a component

Find the index of a component with `index`:

```
abjad> note = container[7]

abjad> container.index(note)
7
```

6.8 Inserting a component by index

Insert a component by index with `insert`:

```
abjad> container.insert(-3, Note("g'32"))
```

```
abjad> show(container)
```

6.9 Removing a component by index

Remove a component by index with `pop`:

```
abjad> container.pop(-1)
```

```
abjad> show(container)
```

6.10 Removing a component by reference

Remove a component by reference with `remove`:

```
abjad> container.remove(container[-1])
```

```
abjad> show(container)
```

Note: `__getslice__`, `__setslice__` and `__delslice__` remain to be documented.

6.11 Naming containers

You can name Abjad containers:

```
abjad> flute_staff = Staff(macros.scale(4))
abjad> flute_staff.name = 'Flute'
abjad> violin_staff = Staff(macros.scale(4))
abjad> violin_staff.name = 'Violin'
abjad> staff_group = scoretools.StaffGroup([flute_staff, violin_staff])
abjad> score = Score([staff_group])
```

Container names appear in LilyPond input:

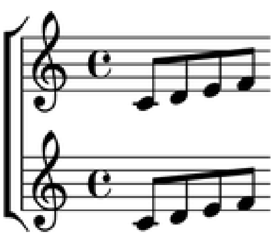
```
abjad> f(score)
\new Score <<
        \new StaffGroup <<
                \context Staff = "Flute" {
                        c'8
                        d'8
                        e'8
                        f'8
                }
                \context Staff = "Violin" {
                        c'8
                        d'8
                        e'8
                        f'8
                }
        >>
>>
```

And make it easy to retrieve containers later:

```
abjad> componenttools.get_first_component_in_expr_with_name(score, 'Flute')
Staff-"Flute"{4}
```

But container names do not appear in notational output:

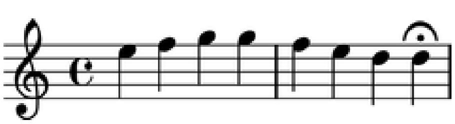
```
abjad> show(score)
```



6.12 Understanding { } and << >> in LilyPond

LilyPond uses curly { } braces to wrap a stream of musical events that are to be engraved one after the other:

```
\new Voice {
   e''4
   f''4
   g''4
   g''4
   f''4
   e''4
   d''4
   d''4 \fermata
}
```



LilyPond uses skeleton << >> braces to wrap two or more musical expressions that are to be played at the same time:

```
\new Staff <<
   \new Voice {
      \voiceOne
      e''4
      f''4
      g''4
      g''4
      f''4
      e''4
      d''4
      d''4 \fermata
   }
   \new Voice {
      \voiceTwo
      c''4
      c''4
      b'4
      c''4
      c''8
      b'8
      c''4
      b'4
      b'4 \fermata
   }
>>
```

The examples above are both LilyPond input.

The most common use of LilyPond `{ }` is to group a potentially long stream of notes and rests into a single expression.

The most common use of LilyPond `<< >>` is to group a relatively smaller number of note lists together polyphonically.

6.13 Understanding sequential and parallel containers

Abjad implements LilyPond `{ }` and `<< >>` in the container `is_parallel` attribute.

Some containers set `is_parallel` to false at initialization:

```
staff = Staff([ ])
staff.is_parallel
False
```

Other containers set `is_parallel` to true:

```
score = Score([ ])
score.is_parallel
True
```

6.14 Changing sequential and parallel containers

Set `is_parallel` by hand as necessary:

```
voice_1 = Voice(r"e''4 f''4 g''4 g''4 f''4 e''4 d''4 d''4  ermata")
voice_2 = Voice(r"c''4 c''4 b'4 c''4 c''8 b'8 c''4 b'4 b'4  ermata")
abjad> staff = Staff([voice_1, voice_2])
abjad> staff.is_parallel = True
abjad> marktools.LilyPondCommandMark('voiceOne')(voice_1)
abjad> marktools.LilyPondCommandMark('voiceTwo')(voice_2)
abjad> show(staff)
```

The staff in the example above is set to parallel after initialization to create a type of polyphonic staff:

```
abjad> f(staff)
\new Staff <<
        \new Voice {
                \voiceOne
                e''4
                f''4
                g''4
                g''4
                f''4
                e''4
                d''4
                d''4 -\fermata
        }
        \new Voice {
                \voiceTwo
                c''4
                c''4
                b'4
                c''4
                c''8
                b'8
                c''4
                b'4
                b'4 -\fermata
        }
>>
```

Reference manual — grobs

CHAPTER

SEVEN

UNDERSTANDING LILYPOND GROBS

LilyPond models music notation as a collection of graphic objects or grobs.

7.1 Grobs control typography

LilyPond grobs control the typographic details of the score:

```
\new Staff {
        c'4 (
        d'4 )
        e'4 (
        f'4 )
        g'4 (
        a'4 )
        g'2
}
```

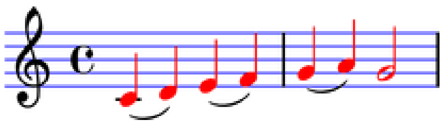
In the example above LilyPond creates a grob for every printed glyph. This includes the clef and time signature as well as the note heads, stems and slurs. If the example included beams, articulations or an explicit key signature then LilyPond would create grobs for those as well.

7.2 Grobs can be overridden

You can change the appearance of LilyPond grobs with grob overrides:

```
\new Staff \with {
        \override NoteHead #'color = #red
        \override StaffSymbol #'color = #blue
        \override Stem #'color = #red
} {
        c'4 (
        d'4 )
        e'4 (
        f'4 )
        g'4 (
        a'4 )
```

```
        g'2
}
```



7.3 Check the LilyPond docs

New grobs are added to LilyPond from time to time.

For a complete list of LilyPond grobs see the LilyPond documentation.

CHAPTER

EIGHT

UNDERSTANDING ABJAD OVERRIDES

8.1 Grob-override component plug-ins

All Abjad containers have a grob-override plug-in:

```
abjad> staff = Staff("c'4 d'4 e'4 f'4 g'4 a'4 g'2")

abjad> staff.override.staff_symbol.color = 'blue'

abjad> staff.override
LilyPondGrobOverrideComponentPlugIn(staff_symbol__color = 'blue')
```

All Abjad leaves have a grob-override plug-in, too:

```
abjad> leaf = staff[-1]

abjad> leaf.override.note_head.color = 'red'
abjad> leaf.override.stem.color = 'red'

abjad> leaf.override
LilyPondGrobOverrideComponentPlugIn(note_head__color = 'red', stem__color = 'red')
```

And so do Abjad spanners:

```
abjad> slur = spannertools.SlurSpanner(staff[:])

abjad> slur.override.slur.color = 'red'

abjad> slur.override
LilyPondGrobOverrideComponentPlugIn(slur__color = 'red')
```

8.2 Grob proxies

Grob-override plug-ins contain grob proxies:

```
abjad> leaf.override.note_head
LilyPondGrobProxy(color = 'red')

abjad> leaf.override.stem
LilyPondGrobProxy(color = 'red')
```

8.3 Dot-chained override syntax

The's dot-chained grob override syntax shown here results from the special way that the Abjad grob-override plug-in and grob proxy set and get their attributes.

CHAPTER

NINE

OVERRIDING CONTAINERS

You can override LilyPond grobs to change Abjad containers their contents.

9.1 Examining defaults

The symbols below are black with fixed thickness and predetermined spacing:

```
abjad> staff = Staff("c'4 d'4 e'4 f'4 g'4 a'4 g'2")
abjad> slur_1 = spannertools.SlurSpanner(staff[:2])
abjad> slur_2 = spannertools.SlurSpanner(staff[2:4])
abjad> slur_3 = spannertools.SlurSpanner(staff[4:6])

abjad> f(staff)
\new Staff {
        c'4 (
        d'4 )
        e'4 (
        f'4 )
        g'4 (
        a'4 )
        g'2
}

abjad> show(staff)
```

9.2 Overriding containers

You can override LilyPond grobs to change the look of Abjad containers:

```
abjad> staff.override.staff_symbol.color = 'blue'

abjad> f(staff)
\new Staff \with {
        \override StaffSymbol #'color = #blue
} {
        c'4 (
```

```
        d'4 )
        e'4 (
        f'4 )
        g'4 (
        a'4 )
        g'2
}
```

```
abjad> show(staff)
```

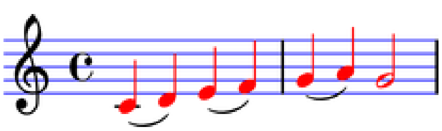
9.3 Overriding containers' contents

You can override LilyPond grobs to change the look of containers' contents, too:

```
abjad> staff.override.note_head.color = 'red'
abjad> staff.override.stem.color = 'red'
```

```
abjad> f(staff)
\new Staff \with {
        \override NoteHead #'color = #red
        \override StaffSymbol #'color = #blue
        \override Stem #'color = #red
} {
        c'4 (
        d'4 )
        e'4 (
        f'4 )
        g'4 (
        a'4 )
        g'2
}
```

```
abjad> show(staff)
```



9.4 Deleting overrides

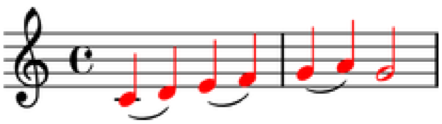
Delete grob overrides you no longer want:

```
abjad> del(staff.override.staff_symbol)
```

```
abjad> f(staff)
\new Staff \with {
        \override NoteHead #'color = #red
        \override Stem #'color = #red
```

```
} {
        c'4 (
        d'4 )
        e'4 (
        f'4 )
        g'4 (
        a'4 )
        g'2
}
```

```
abjad> show(staff)
```



CHAPTER

TEN

OVERRIDING LEAVES

You can override LilyPond grobs to change notes, rests and chords.

10.1 Examining defaults

The symbols below are black with fixed thickness and predetermined spacing:

```
abjad> staff = Staff("c'4 d'4 e'4 f'4 g'4 a'4 g'2")
abjad> slur_1 = spannertools.SlurSpanner(staff[:2])
abjad> slur_2 = spannertools.SlurSpanner(staff[2:4])
abjad> slur_3 = spannertools.SlurSpanner(staff[4:6])

abjad> f(staff)
\new Staff {
        c'4 (
        d'4 )
        e'4 (
        f'4 )
        g'4 (
        a'4 )
        g'2
}

abjad> show(staff)
```

10.2 Overriding leaves

You can override LilyPond grobs to change the look of notes, rests and chords:

```
abjad> staff[-1].override.note_head.color = 'red'
abjad> staff[-1].override.stem.color = 'red'

abjad> f(staff)
\new Staff {
        c'4 (
        d'4 )
```

```
        e'4 (
        f'4 )
        g'4 (
        a'4 )
        \once \override NoteHead #'color = #red
        \once \override Stem #'color = #red
        g'2
}
```

```
abjad> show(staff)
```

10.3 Deleting overrides

Delete grob overrides you no longer want:

```
abjad> del(staff[-1].override.stem)

abjad> f(staff)
\new Staff {
        c'4 (
        d'4 )
        e'4 (
        f'4 )
        g'4 (
        a'4 )
        \once \override NoteHead #'color = #red
        g'2
}
```

```
abjad> show(staff)
```

CHAPTER

ELEVEN

OVERRIDING SPANNERS

You can override LilyPond grobs to change Abjad spanners and their contents.

11.1 Examining defaults

The symbols below are black with fixed thickness and predetermined spacing:

```
abjad> staff = Staff("c'4 d'4 e'4 f'4 g'4 a'4 g'2")
abjad> slur_1 = spannertools.SlurSpanner(staff[:2])
abjad> slur_2 = spannertools.SlurSpanner(staff[2:4])
abjad> slur_3 = spannertools.SlurSpanner(staff[4:6])

abjad> f(staff)
\new Staff {
        c'4 (
        d'4 )
        e'4 (
        f'4 )
        g'4 (
        a'4 )
        g'2
}

abjad> show(staff)
```

11.2 Overriding spanners

You can override LilyPond grobs to change the look of spanners:

```
abjad> slur_1.override.slur.color = 'red'
abjad> slur_3.override.slur.color = 'red'

abjad> f(staff)
\new Staff {
        \override Slur #'color = #red
        c'4 (
```

```
        d'4 )
        \revert Slur #'color
        e'4 (
        f'4 )
        \override Slur #'color = #red
        g'4 (
        a'4 )
        \revert Slur #'color
        g'2
}
```

```
abjad> show(staff)
```

11.3 Overriding spanners' contents

You can override LilyPond grobs to change spanners' contents:

```
abjad> slur_2.override.slur.color = 'blue'
abjad> slur_2.override.note_head.color = 'blue'
abjad> slur_2.override.stem.color = 'blue'
```

```
abjad> f(staff)
\new Staff {
        \override Slur #'color = #red
        c'4 (
        d'4 )
        \revert Slur #'color
        \override NoteHead #'color = #blue
        \override Slur #'color = #blue
        \override Stem #'color = #blue
        e'4 (
        f'4 )
        \revert NoteHead #'color
        \revert Slur #'color
        \revert Stem #'color
        \override Slur #'color = #red
        g'4 (
        a'4 )
        \revert Slur #'color
        g'2
}
```

```
abjad> show(staff)
```

11.4 Deleting overrides

Delete grob overrides you no longer want:

```
abjad> del(slur_1.override.slur)
abjad> del(slur_3.override.slur)

abjad> f(staff)
\new Staff {
        c'4 (
        d'4 )
        \override NoteHead #'color = #blue
        \override Slur #'color = #blue
        \override Stem #'color = #blue
        e'4 (
        f'4 )
        \revert NoteHead #'color
        \revert Slur #'color
        \revert Stem #'color
        g'4 (
        a'4 )
        g'2
}

abjad> show(staff)
```

Reference manual — LilyPond files

CHAPTER

TWELVE

WORKING WITH LILYPOND FILES

12.1 Making LilyPond files

Make a basic LilyPond input file with the `lilyfiletools` package:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> lily_file = lilyfiletools.make_basic_lily_file(staff)

abjad> lily_file
LilyFile(Staff{4})
```

12.2 Inspecting file output

LilyPond input files that you create this way come equipped with many attributes that appear in file output:

```
abjad> f(lily_file)
% Abjad revision 4422
% 2011-06-12 17:02

\version "2.13.61"
\include "english.ly"
\include "/Users/trevorbaca/Documents/abjad/trunk/abjad/cfg/abjad.scm"

\score {
        \new Staff {
                c'8
                d'8
                e'8
                f'8
        }
}
```

12.3 Setting default paper size

Set default LilyPond paper size like this:

```
abjad> lily_file.default_paper_size = '11x17', 'landscape'
```

```
abjad> f(lily_file)
% Abjad revision 4422
% 2011-06-12 17:02

\version "2.13.61"
\include "english.ly"
\include "/Users/trevorbaca/Documents/abjad/trunk/abjad/cfg/abjad.scm"

#(set-default-paper-size "11x17" 'landscape)

\score {
        \new Staff {
                c'8
                d'8
                e'8
                f'8
        }
}
```

12.4 Setting global staff size

Set global staff size like this:

```
abjad> lily_file.global_staff_size = 16
```

```
abjad> f(lily_file)
% Abjad revision 4422
% 2011-06-12 17:02

\version "2.13.61"
\include "english.ly"
\include "/Users/trevorbaca/Documents/abjad/trunk/abjad/cfg/abjad.scm"

#(set-default-paper-size "11x17" 'landscape)
#(set-global-staff-size 16)

\score {
        \new Staff {
                c'8
                d'8
                e'8
                f'8
        }
}
```

Reference manual — marks

CHAPTER
THIRTEEN

WORKING WITH ANNOTATIONS

Annotate components with user-specific information for future use.

Annotations do not impact formatting.

13.1 Creating annotations

Use mark tools to create annotations:

```
abjad> annotation = marktools.Annotation('special pitch', pitchtools.NamedChromaticPitch('bs'))

abjad> annotation
Annotation('special pitch', NamedChromaticPitch('bs'))
```

13.2 Attaching annotations

Attach annotations by calling them:

```
abjad> note = Note("c'4")
abjad> annotation(note)

abjad> annotation
Annotation('special pitch', NamedChromaticPitch('bs'))(c'4)
```

13.3 Creating and attaching annotations in one step

Create and attach annotations in one step like this:

```
abjad> another_annotation = marktools.Annotation('special pitch', pitchtools.NamedChromaticPitch('bs'

abjad> another_annotation
Annotation('special pitch', NamedChromaticPitch('bs'))(c'4)
```

13.4 Getting annotations

Use mark tools to get annotations:

```
abjad> marktools.get_annotations_attached_to_component(note)
(Annotation('special pitch', NamedChromaticPitch('bs'))(c'4), Annotation('special pitch', NamedChroma
```

13.5 Detaching annotations by hand

Detach annotations by hand:

```
abjad> annotation.detach_mark( )

abjad> annotation
Annotation('special pitch', NamedChromaticPitch('bs'))
```

13.6 Detaching annotations automatically

Or use mark tools to detach all annotations at once:

```
abjad> print marktools.detach_annotations_attached_to_component(note)
(Annotation('special pitch', NamedChromaticPitch('bs')),)

abjad> marktools.get_annotations_attached_to_component(note)
()
```

13.7 Inspecting attachment

Use `start_component` to inspect attachment:

```
abjad> annotation(note)

abjad> annotation.start_component
Note("c'4")
```

13.8 Inspecting name

Use `name` to get the name of any annotation:

```
abjad> annotation.name
'special pitch'
```

13.9 Inspecting value

And use `value` to get the value of any annotation:

```
abjad> annotation.value
NamedChromaticPitch('bs')
```

CHAPTER

FOURTEEN

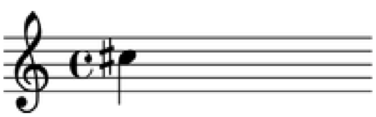
WORKING WITH COMMENTS

LilyPond comments begin with the `%` sign. Abjad models LilyPond comments as marks.

14.1 Adding comments

You can add comments before, after or to the right of any note, rest or chord:

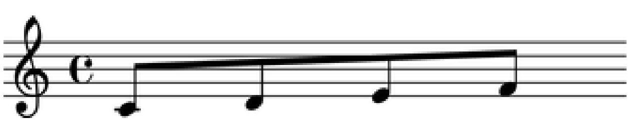
```
abjad> note = Note(13, (1, 4))
abjad> show(note)
```



```
abjad> marktools.Comment('This is a comment before the note.', 'before')(note)
abjad> marktools.Comment('This is a comment to the right of the note.', 'right')(note)

abjad> f(note)
% This is a comment before the note.
cs''4 % This is a comment to the right of the note.
```

You can add comments before, after, in the opening or in the closing of any container:

```
abjad> staff = Staff(macros.scale(4))
abjad> show(staff)
```



```
abjad> marktools.Comment('Here is a comment before the staff.', 'before')(staff)
abjad> marktools.Comment('Here is a comment in the staff opening.', 'opening')(staff)
abjad> marktools.Comment('Here is another comment in the staff opening.', 'opening')(staff)
abjad> marktools.Comment('Comment in the staff closing.', 'closing')(staff)
abjad> marktools.Comment('Comment after the staff.', 'after')(staff)

abjad> f(staff)
% Here is a comment before the staff.
\new Staff {
	% Here is a comment in the staff opening.
	% Here is another comment in the staff opening.
	c'8
```

```
        d'8
        e'8
        f'8
        % Comment in the staff closing.
}
% Comment after the staff.
```

14.2 Getting comments

Use mark tools to get comments:

```
abjad> marktools.get_comments_attached_to_component(note)
(Comment('This is a comment before the note.')(cs''4), Comment('This is a comment to the right of the
```

14.3 Detaching comments

Detach comments by hand:

```
abjad> comment_1, comment_2 = marktools.get_comments_attached_to_component(note)

abjad> comment_1.detach_mark( )
Comment('This is a comment before the note.')
abjad> comment_2.detach_mark( )
Comment('This is a comment to the right of the note.')

abjad> f(note)
cs''4

abjad> marktools.get_comments_attached_to_component(note)
()
```

Or use mark tools to detach comments automatically:

```
abjad> marktools.detach_comments_attached_to_component(staff)

abjad> f(staff)
\new Staff {
        c'8
        d'8
        e'8
        f'8
}

abjad> marktools.get_comments_attached_to_component(staff)
()
```

CHAPTER

FIFTEEN

WORKING WITH INSTRUMENT MARKS

Use context tools to add instrument marks:

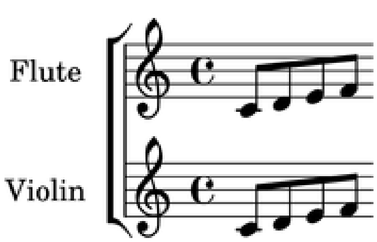
```
abjad> flute_staff = Staff(macros.scale(4))
abjad> violin_staff = Staff(macros.scale(4))
abjad> staff_group = scoretools.StaffGroup([flute_staff, violin_staff])
abjad> score = Score([staff_group])
abjad> contexttools.InstrumentMark('Flute ', 'Fl. ')(flute_staff)
abjad> contexttools.InstrumentMark('Violin ', 'Vn. ')(violin_staff)
```

Instrument marks appear as context settings in LilyPond input:

```
abjad> f(score)
\new Score <<
        \new StaffGroup <<
                \new Staff {
                        \set Staff.instrumentName = \markup { Flute  }
                        \set Staff.shortInstrumentName = \markup { Fl.  }
                        c'8
                        d'8
                        e'8
                        f'8
                }
                \new Staff {
                        \set Staff.instrumentName = \markup { Violin  }
                        \set Staff.shortInstrumentName = \markup { Vn.  }
                        c'8
                        d'8
                        e'8
                        f'8
                }
        >>
>>
```

Instrument marks appear as instrument names in notational output:

```
abjad> show(score)
```



Reference manual — measures

CHAPTER

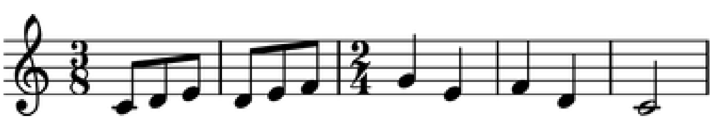
SIXTEEN

WORKING WITH MEASURES

16.1 Understanding measures in LilyPond

In LilyPond you specify time signatures by hand and LilyPond creates measures automatically:

```
\new Staff {
   \time 3/8
   c'8
   d'8
   e'8
   d'8
   e'8
   f'8
   \time 2/4
   g'4
   e'4
   f'4
   d'4
   c'2
}
```

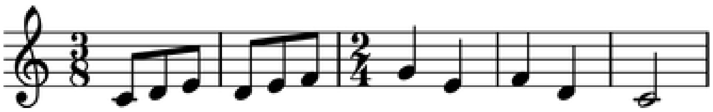


Here LilyPond creates five measures from two time signatures. This happens because behind-the-scenes LilyPond time-keeping tells the program when measures start and stop and how to draw the barlines that come between them.

16.2 Understanding measures in Abjad

Measures are optional in Abjad, too, and you may omit them in favor of time signatures:

```
abjad> staff = Staff("c'8 d'8 e'8 d'8 e'8 f'8 g'4 e'4 f'4 d'4 c'2")

abjad> contexttools.TimeSignatureMark((3, 8))(staff)
abjad> contexttools.TimeSignatureMark((2, 4))(staff[6])

abjad> show(staff)
```

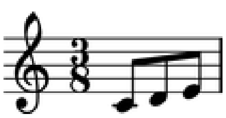


But you may also include explicit measures in the Abjad scores you build. The following sections explain how.

16.3 Creating measures

Create a measure with a meter and music:

```
abjad> measure = Measure((3, 8), macros.scale(3))

abjad> f(measure)
{
        \time 3/8
        c'8
        d'8
        e'8
}

abjad> show(measure)
```



CHAPTER

SEVENTEEN

WORKING WITH DYNAMIC MEASURES

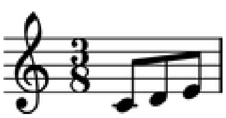
Dynamic measures adjust their time signatures on the fly as you add and remove music.

17.1 Creating dynamic measures

Create dynamic measures without a time signature:

```
abjad> measure = measuretools.DynamicMeasure("c'8 d'8 e'8")
```

```
abjad> show(measure)
```



17.2 Adding music to dynamic measures

Add music to dynamic measures the same as to all containers:

```
abjad> measure.extend([Note("fs'8"), Note("gs'8")])
```

```
abjad> show(measure)
```

17.3 Removing music from dynamic measures

Remove music from dynamic measures the same as with other containers:

```
abjad> del(measure[1:3])
```

```
abjad> show(measure)
```

17.4 Setting the denominator of dynamic measures

You can set the denominator of dynamic measures to any integer power of 2:

```
abjad> measure.denominator = 32
```

```
abjad> show(measure)
```

17.5 Suppressing the meter of dynamic measures

You can temporarily suppress the meter of dynamic measures:

```
abjad> measure.suppress_meter = True
```

```
abjad> f(measure)
{
        c'8
        fs'8
        gs'8
}
```

LilyPond will engrave the last active meter.

CHAPTER EIGHTEEN

WORKING WITH ANONYMOUS MEASURES

Anonymous determine their time signatures on the fly and then hide them at format time.

18.1 Creating anonymous measures

Create anonymous measures without a time signature:

```
abjad> measure = measuretools.AnonymousMeasure("c'8 d'8 e'8")
```

```
abjad> show(measure)
```

18.2 Adding music to anonymous measures

Add music to anonymous measures the same as to other containers:

```
abjad> measure.extend([Note("fs'8"), Note("gs'8")])
```

```
abjad> show(measure)
```

18.3 Removing music from anonymous measures

Remove music from anonymous measure the same as from other containers:

```
abjad> del(measure[1:3])
```

```
abjad> show(measure)
```

Reference manual — notes

CHAPTER

NINETEEN

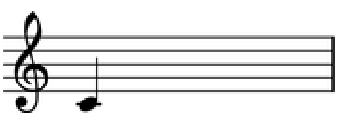
WORKING WITH NOTES

19.1 Making notes from a string

You can make notes from string:

```
abjad> note = Note("c'4")
```

```
abjad> show(note)
```

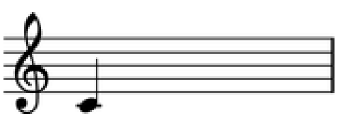


19.2 Making notes from chromatic pitch number and duration

You can also make notes from chromatic pitch number and duration:

```
abjad> note = Note(0, Duration(1, 4))
```

```
abjad> show(note)
```



(You even use `Note(0, (1, 4))` to create notes with numbers alone.)

19.3 Getting the written pitch of notes

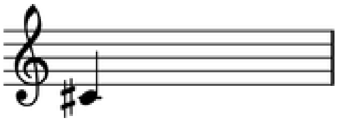
You can get the written pitch of notes:

```
abjad> note.pitch
NamedChromaticPitch("c'")
```

19.4 Changing the written pitch of notes

And you can change the written pitch of notes:

```
abjad> note.pitch = "cs'"
```



(You can use `note.pitch = 1` to change pitch with numbers, too.)

19.5 Getting the duration attributes of notes

Get the written duration of notes like this:

```
abjad> note.duration.written
Duration(1, 4)
```

Which is usually the same as preprolated duration:

```
abjad> note.duration.preprolated
Duration(1, 4)
```

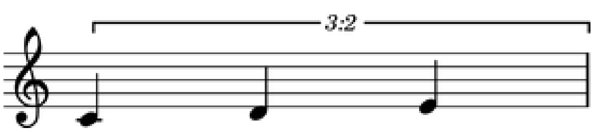
And prolated duration:

```
abjad> note.duration.prolated
Duration(1, 4)
```

Except for notes inside a tuplet:

```
abjad> tuplet = Tuplet((2, 3), [Note("c'4"), Note("d'4"), Note("e'4")])
```

```
abjad> show(tuplet)
```




```
abjad> note = tuplet[0]
```

Tupletted notes carry written duration:

```
abjad> note.duration.written
Duration(1, 4)
```

Prolation:

```
abjad> note.duration.prolation
Fraction(2, 3)
```

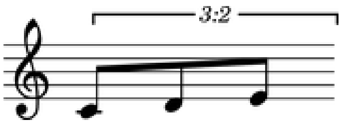
And prolated duration that is the product of the two:

```
abjad> note.duration.prolated
Duration(1, 6)
```

19.6 Changing the written duration of notes

You can change the written duration of notes:

```
abjad> tuplet[0].duration.written = Duration(1, 8)
abjad> tuplet[1].duration.written = Duration(1, 8)
abjad> tuplet[2].duration.written = Duration(1, 8)

abjad> show(tuplet)
```



Other duration attributes are read-only.

Reference manual — pitch

CHAPTER

TWENTY

WORKING WITH NAMED CHROMATIC PITCHES

Named chromatic pitches are the everyday pitches attached to notes and chords:

```
abjad> note = Note("cs''8")

abjad> note.pitch
NamedChromaticPitch("cs''")
```

20.1 Creation

Use pitch tools to create named chromatic pitches:

```
abjad> named_chromatic_pitch = pitchtools.NamedChromaticPitch("cs''")

abjad> named_chromatic_pitch
NamedChromaticPitch("cs''")
```

20.2 Name inspection

Use `str( )` to get the name of named chromatic pitches:

```
abjad> str(named_chromatic_pitch)
cs''
```

20.3 Octave inspection

Get the octave number of named chromatic pitches with `octave_number`:

```
abjad> named_chromatic_pitch.octave_number
5
```

20.4 Sorting

Named chromatic pitches sort by octave, diatonic pitch-class and accidental, in that order:

```
abjad> pitchtools.NamedChromaticPitch('es') < pitchtools.NamedChromaticPitch('ff')
True
```

20.5 Pitch comparison

Compare named chromatic pitches to each other:

```
abjad> named_chromatic_pitch_1 = pitchtools.NamedChromaticPitch("c''")
abjad> named_chromatic_pitch_2 = pitchtools.NamedChromaticPitch("d''")
```

```
abjad> named_chromatic_pitch_1 == named_chromatic_pitch_2
False
```

```
abjad> named_chromatic_pitch_1 != named_chromatic_pitch_2
True
```

```
abjad> named_chromatic_pitch_1 > named_chromatic_pitch_2
False
```

```
abjad> named_chromatic_pitch_1 < named_chromatic_pitch_2
True
```

```
abjad> named_chromatic_pitch_1 >= named_chromatic_pitch_2
False
```

```
abjad> named_chromatic_pitch_1 <= named_chromatic_pitch_2
True
```

20.6 Pitch conversion

Convert any named chromatic pitch to a named diatonic pitch:

```
abjad> named_chromatic_pitch.named_diatonic_pitch
NamedDiatonicPitch("c''")
```

To a numbered chromatic pitch:

```
abjad> named_chromatic_pitch.numbered_chromatic_pitch
NumberedChromaticPitch(13)
```

Or to a numbered diatonic pitch:

```
abjad> named_chromatic_pitch.numbered_diatonic_pitch
NumberedDiatonicPitch(7)
```

20.7 Pitch-class conversion

Convert any named chromatic pitch to a named chromatic pitch-class:

```
abjad> named_chromatic_pitch.named_chromatic_pitch_class
NamedChromaticPitchClass('cs')
```

To a named diatonic pitch-class:

```
abjad> named_chromatic_pitch.named_diatonic_pitch_class
NamedDiatonicPitchClass('c')
```

To a numbered chromatic pitch-class:

```
abjad> named_chromatic_pitch.numbered_chromatic_pitch_class
NumberedChromaticPitchClass(1)
```

Or to a numbered diatonic pitch-class:

```
abjad> named_chromatic_pitch.numbered_diatonic_pitch_class
NumberedDiatonicPitchClass(0)
```

20.8 Copying

Use `copy.copy( )` to copy named chromatic pitches:

```
abjad> import copy

abjad> copy.copy(named_chromatic_pitch)
NamedChromaticPitch("cs''")
```

Or use `copy.deepcopy( )` to do the same thing:

```
abjad> copy.deepcopy(named_chromatic_pitch)
NamedChromaticPitch("cs''")
```

Reference manual — rests

CHAPTER

TWENTYONE

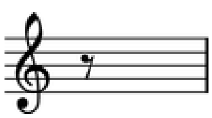
WORKING WITH RESTS

21.1 Making rests from strings

You can make rests from a string:

```
abjad> rest = Rest('r8')
```

```
abjad> show(rest)
```

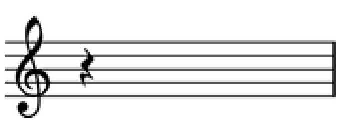


21.2 Making rests from durations

You can also make rests from a duration:

```
abjad> rest = Rest(Duration(1, 4))
```

```
abjad> show(rest)
```



(You can even use `Rest((1, 8))` to make rests from a duration pair.)

21.3 Getting the duration attributes of rests

Get the written duration of rests like this:

```
abjad> rest.duration.written
Duration(1, 4)
```

Which is usually the same as preprolated duration:

```
abjad> rest.duration.preprolated
Duration(1, 4)
```

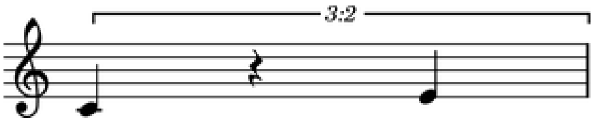
And prolated duration:

```
abjad> rest.duration.prolated
Duration(1, 4)
```

Except for rests inside a tuplet:

```
abjad> tuplet = Tuplet((2, 3), [Note("c'4"), Rest('r4'), Note("e'4")])
```

```
abjad> show(tuplet)
```




```
abjad> rest = tuplet[1]
```

Tupletted rests carry written duration:

```
abjad> rest.duration.written
Duration(1, 4)
```

Prolation:

```
abjad> rest.duration.prolation
Fraction(2, 3)
```

And prolated duration that is the product of the two:

```
abjad> rest.duration.prolated
Duration(1, 6)
```

21.4 Changing the written duration of rests

You can change the written duration of notes and rests:

```
abjad> tuplet[0].duration.written = Duration(1, 8)
abjad> tuplet[1].duration.written = Duration(1, 8)
abjad> tuplet[2].duration.written = Duration(1, 8)
```

```
abjad> show(tuplet)
```

Other duration attributes are read-only.

Reference manual — scores

CHAPTER

TWENTYTWO

WORKING WITH SCORES

22.1 Creating scores

Create a score like this:

```
abjad> treble_staff_1 = Staff("e'4 d'4 e'4 f'4 g'1")
abjad> treble_staff_2 = Staff("c'2. b8 a8 b1")

abjad> score = Score([treble_staff_1, treble_staff_2])

abjad> show(score)
```

22.2 Inspecting score music

Return score components with `music`:

```
abjad> score.music
(Staff{5}, Staff{4})
```

22.3 Inspecting score length

Get score length with `len( )`:

```
abjad> len(score)
2
```

22.4 Inspecting score duration

Score contents duration is equal to the duration of the longest component in score:

```
abjad> score.duration.contents
Duration(2, 1)
```

22.5 Adding one component to the bottom of a score

Add one component to the bottom of a score with `append`:

```
abjad> bass_staff = Staff("g4 f4 e4 d4 d1")
abjad> contexttools.ClefMark('bass')(bass_staff)

abjad> score.append(bass_staff)

abjad> show(score)
```

22.6 Finding the index of a score component

Find the index of a score component with `index`:

```
abjad> score.index(treble_staff_1)
0
```

22.7 Removing a score component by index

Use `pop` to remove a score component by index:

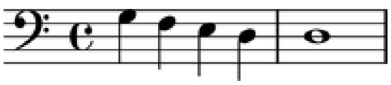
```
abjad> score.pop(1)

abjad> show(score)
```

22.8 Removing a score component by reference

Remove a score component by reference with `remove`:

```
abjad> score.remove(treble_staff_1)
```

```
abjad> show(score)
```



22.9 Testing score containment

Use `in` to find out whether a score contains a given component:

```
abjad> treble_staff_1 in score
False
```

```
abjad> treble_staff_2 in score
False
```

```
abjad> bass_staff in score
True
```

22.10 Naming scores

You can name Abjad scores:

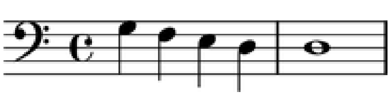
```
abjad> score.name = 'Example Score'
```

Score names appear in LilyPond input:

```
abjad> f(score)
\context Score = "Example Score" <<
        \new Staff {
                \clef "bass"
                g4
                f4
                e4
                d4
                d1
        }
>>
```

But do not appear in notational output:

```
abjad> show(score)
```



Reference manual — spanners

CHAPTER

TWENTYTHREE

WORKING WITH SPANNERS

Reference manual — staves

CHAPTER

TWENTYFOUR

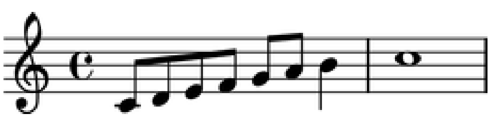
WORKING WITH STAVES

24.1 Creating staves

Create staves like this:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8 g'8 a'8 b'4 c''1")
```

```
abjad> show(staff)
```



24.2 Inspecting staff music

Return staff components with `music`:

```
abjad> staff.music
(Note("c'8"), Note("d'8"), Note("e'8"), Note("f'8"), Note("g'8"), Note("a'8"), Note("b'4"), Note("c''
```

24.3 Inspecting staff length

Get staff length with `len( )`:

```
abjad> len(staff)
8
```

24.4 Inspecting staff duration

Staff contents durations equals the sum of staff components' duration:

```
abjad> staff.duration.contents
Duration(2, 1)
```

24.5 Adding one component to the end of a staff

Add one component to the end of a staff with `append`:

```
abjad> staff.append(Note("d''2"))
```

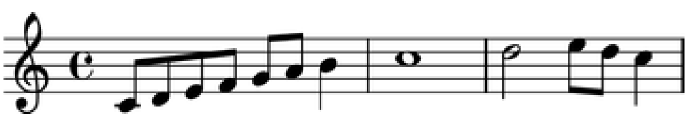
```
abjad> show(staff)
```

24.6 Adding many components to the end of a staff

Add many components to the end of a staff with `extend`:

```
abjad> notes = [Note("e''8"), Note("d''8"), Note("c''4")]
abjad> staff.extend(notes)
```

```
abjad> show(staff)
```



24.7 Finding the index of a staff component

Find staff component index with `index`:

```
abjad> notes[0]
Note("e''8")
```

```
abjad> staff.index(notes[0])
9
```

24.8 Removing a staff component by index

Use `pop` to remove a staff component by index:

```
abjad> staff[8]
Note("d''2")
```

```
abjad> staff.pop(8)
```

```
abjad> show(staff)
```

24.9 Removing a staff component by reference

Remove staff components by reference with `remove`:

```
abjad> staff.remove(staff[-1])
```

```
abjad> show(staff)
```

24.10 Naming staves

You can name Abjad staves:

```
abjad> staff.name = 'Example Staff'
```

Staff names appear in LilyPond input:

```
abjad> f(staff)
\context Staff = "Example Staff" {
        c'8
        d'8
        e'8
        f'8
        g'8
        a'8
        b'4
        c''1
        e''8
        d''8
}
```

But not in notational output:

```
abjad> show(staff)
```

24.11 Forcing context

Staff context equals `'Staff'` by default:

```
abjad> staff.context
'Staff'
```

You can force staff context:

```
abjad> staff.context = 'CustomUserStaff'
```

```
abjad> staff.context
'CustomUserStaff'

abjad> f(staff)
\context CustomUserStaff = "Example Staff" {
        c'8
        d'8
        e'8
        f'8
        g'8
        a'8
        b'4
        c''1
        e''8
        d''8
}
```

Force context when you have defined a new LilyPond context.

Reference manual — tree structure

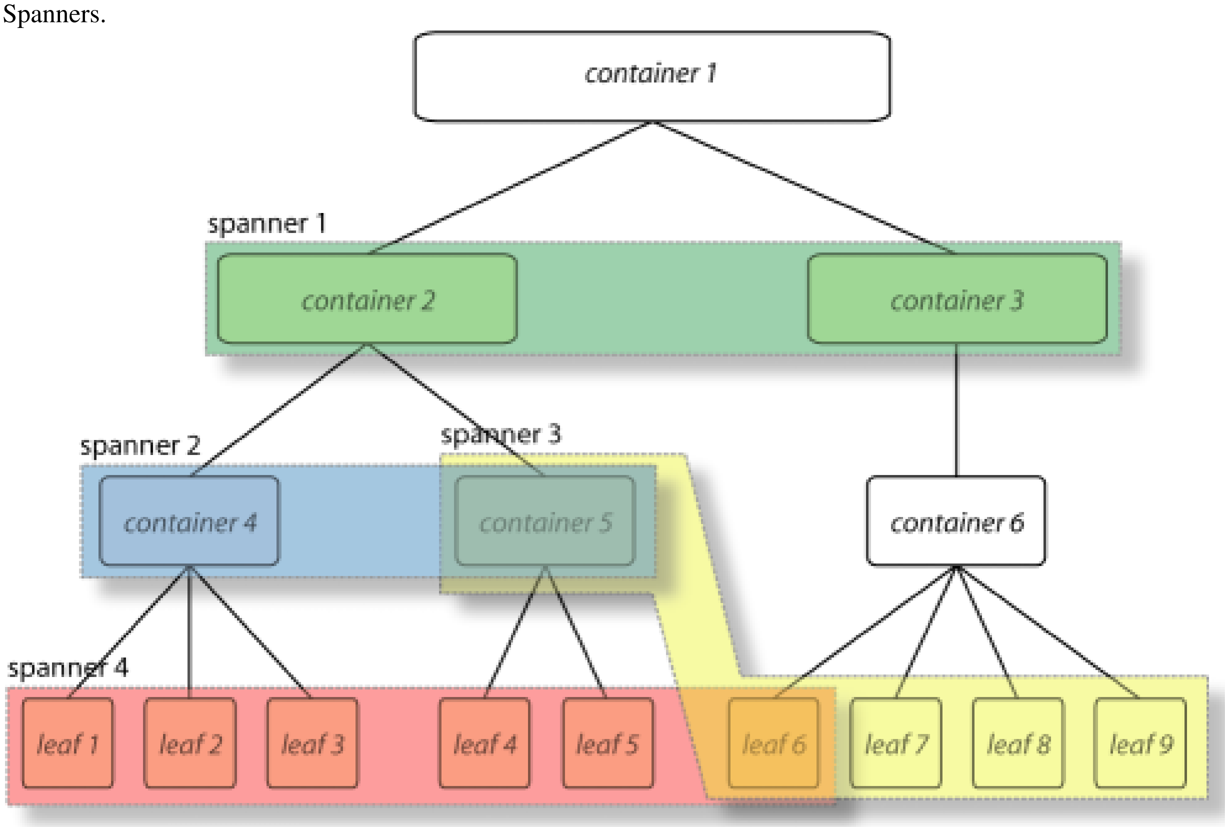
CHAPTER

TWENTYFIVE

LEAF, CONTAINER, SPANNER

At the heart of Abjad's Symbolic Score-Control lies a powerful model that we call the Leaf Container Spanner, or LCS, model of the musical score.

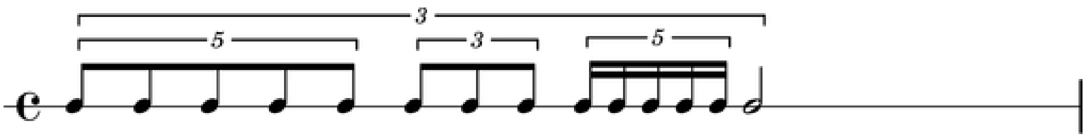
The LCS model can be schematically visualized as a superposition of two complementary and completely independent layers of structure: a *tree* that includes the Containers and the Leaves, and a layer of free floating *connectors* or Spanners.



There can be any number of Spanners, they may overlap, and they may connect to different levels of the tree hierarchy. The spanner attach to the elements of the tree, so a tree structure must exist for spanners to be made manifest.

25.1 Example 1

To understand the whys and hows of the LCS model implemented in Abjad, it is probably easier to base the discussion on concrete musical examples. Let's begin with a simple and rather abstract musical fragment: a measure with nested tuplets.



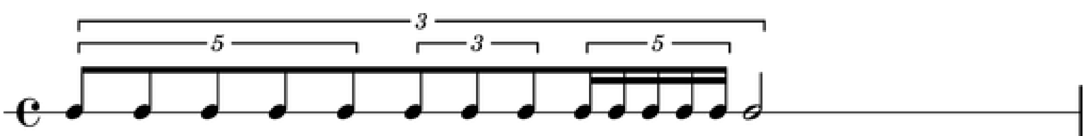
What we see in this little fragment is a measure with 4/4 meter, 14 notes and four tuplet brackets prolating the notes. The three bottom tuplets (with ratios 5:4, 3:2, 5:4) prolate all but the last note. The topmost tuplet prolates all the notes in the measure and combines with the bottom three tuplets to doubly prolate all but the last note. The topmost tuplet as thus prolates three tuplets, each of which in turn prolates a group of notes. We can think of a tuplet as *containing* notes or other tuplets or both. Thus, in our example, the topmost tuplet contains three tuplets and a half note. Each of the tuplets contained by the topmost tuplet in turn contains five, three, and five notes respectively. If we add the measure, then we have a measure that contains a tuplet that contains tuplets that contain notes. The structure of the measure with nested tuplets as we have just described it has two important properties:

1. It is a *hierarchical* structure.
2. It follows *exclusive membership*, meaning that each element in the hierarchy (a note, a tuplet or a measure) has one and only one *parent*. In other words a single note is not contained in more than one tuplet simultaneously, and no one tuplet is contained in more than one other tuplet at the same time.

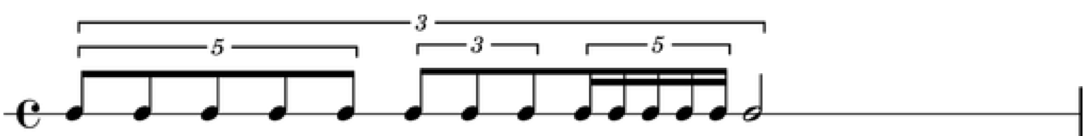
What we are describing here is a tree, and it is the structure of Abjad *containers*.

While this tree structure seem like the right way to represent the relationships between the elements of a score, it is not enough. Consider the tuplet example again with the following beaming alternatives:

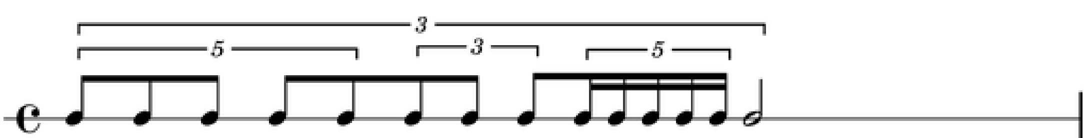
Beaming alternative 1:



Beaming alternative 2:



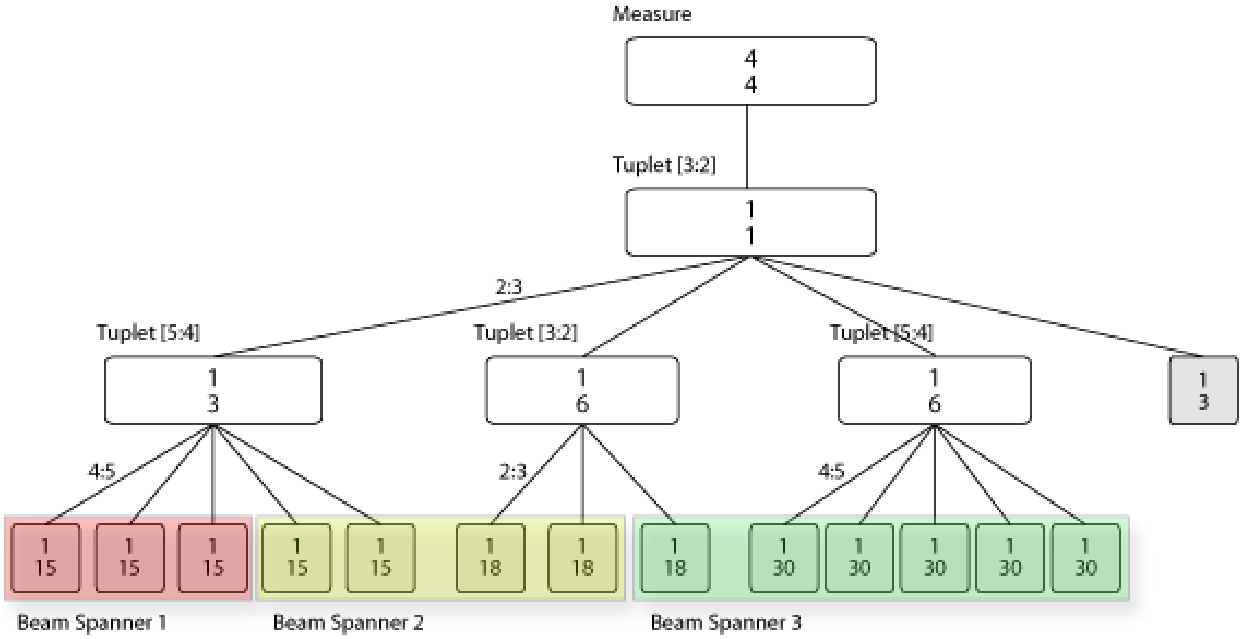
Beaming alternative 3:



Clearly the beaming of notes can be totally independent from the tuplet groupings. Beaming across tuplet groups implies beaming across nodes in the tree structure, which means that the beams do not adhere to the *exclusive (parenthood) membership* characteristic of the tree. Beams must then be modeled independently as a separate and complementary structure. These are the Abjad *spanners*.

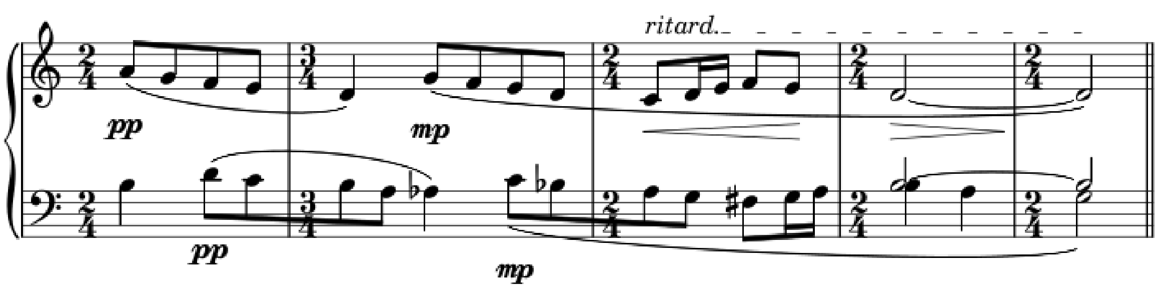
Below we have the score of our tuplet example with alternative beaming and its the Leaf-Container-Spanner graph. Notice that the colored blocks represent spanners.

Beaming alternative 3 (graph):



25.2 Example 2

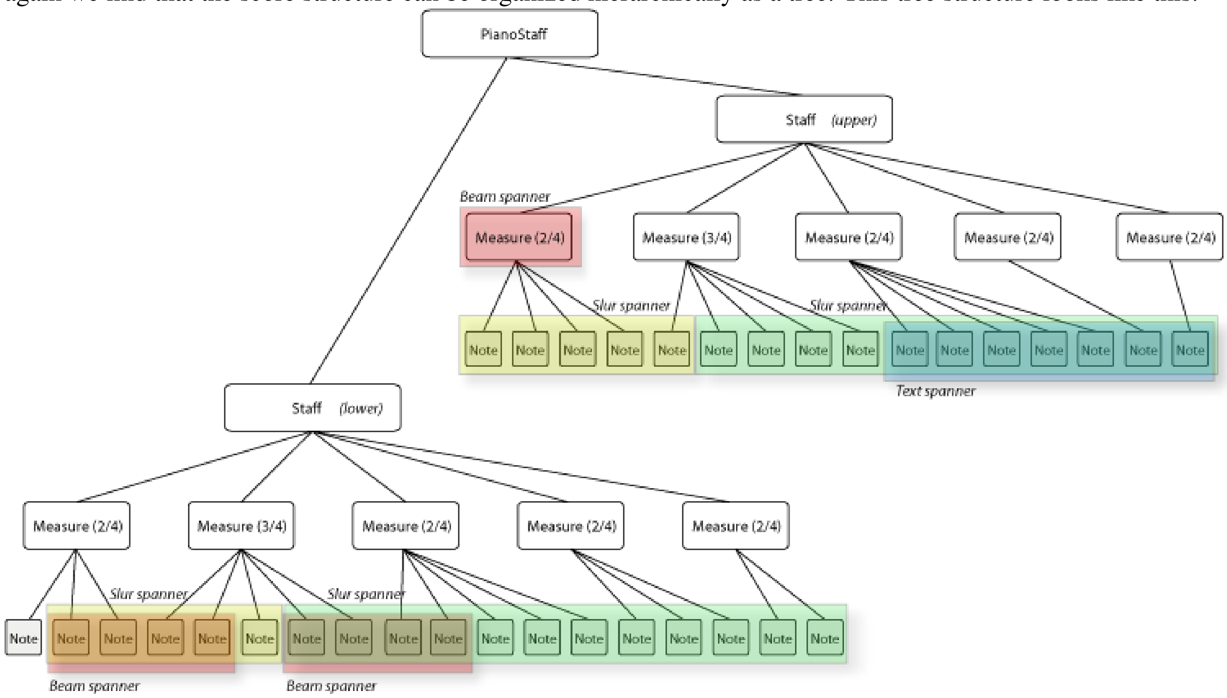
As a second example let's look at the last five measures of Bartók's *Wandering* from Mikrokosmos vol. III. As simple as it may seem, these five measures carry with them a lot of information pertaining to musical notation.



Note: Please refer to the *Bartok example* for a step by step construction of the musical fragment and its full Abjad code.

There are many musical signs of different types on the pages: notes, dynamic markings, clefs, staves, slurs, etc. These signs are structurally related to each other in different ways. Let's start by looking at the larger picture. The piano piece is written in two staves. As is customary, the staves are graphically grouped with a large curly brace attaching to them at the beginning or each system. Notice that each staff has a variety of signs associated with it. There are notes printed on the staff lines as well as meter indications and bar lines. Each note, for example, is in one and only one staff. A note is never in two staves at the same time. This is also true for measures. A measure in the top staff is not simultaneously drawn on the top staff and the bottom staff. It is better to think of each staff as having its own set of measures. Notice also that the notes in each staff fall within the region of one and only one measure, i.e. measures seem to contain notes. There is not one note that is at once in two measures (this is standard practice in musical notation, but it need not always be the case. See the *metric grid example* for a non-containment approach to meters).

As we continue describing the relationships between the musical signs in the page, we begin to discover a certain structure, or a convenient way of structuring the score for conceptualization and manipulation. All the music in a piano score seems to be written in what we might call a *staff group*. The staff group is *composed of* two staves. Each staff in turn appears to be composed of a series or measures, and each measure is composed of a series of notes. So again we find that the score structure can be organized hierarchically as a tree. This tree structure looks like this:



Notice again though that there are elements in the score that imply and require a different kind of grouping. The two four eighth-note runs in the lower staff are beamed together across the bar line and, based on our tree structure, across tree nodes. So do the slurs, the dynamics markings and the ritardando indication at the top of the score. As we have seen in the tuplets example, all these groups running across the tree structure can be defined with *spanners*.

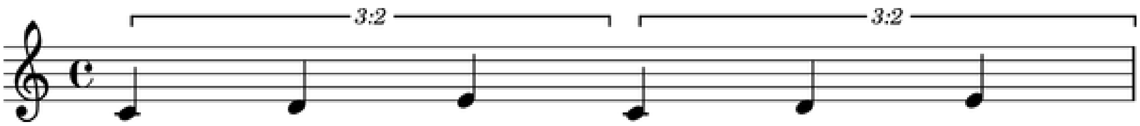
CHAPTER

TWENTYSIX

WORKING WITH COMPONENT PARENTAGE

Many score objects contain other score objects.

```
abjad> tuplet = Tuplet((2, 3), macros.scale(3, (1, 4)))
abjad> staff = Staff(2 * tuplet)
abjad> score = Score([staff])
abjad> show(score)
```



Abjad uses the idea of parentage to model the way objects contain each other.

26.1 Improper parentage

The improper parentage of the first note in score begins with the note itself:

```
abjad> note = score.leaves[0]
Note("c'4")

abjad> componenttools.get_improper_parentage_of_component(note)
(Note("c'4"), Tuplet(2/3, [c'4, d'4, e'4]), Staff{2}, Score<<1>>)
```

26.2 Proper parentage

The proper parentage of the note begins with only the immediate parent of the note:

```
abjad> componenttools.get_proper_parentage_of_component(note)
(Tuplet(2/3, [c'4, d'4, e'4]), Staff{2}, Score<<1>>)
```

Note: the length of the improper parentage of any component equals the length of the proper parentage of the component plus `1`.

26.3 Parentage attributes

Use component tools to find score depth:

```
abjad> componenttools.component_to_score_depth(note)
3
```

Or score root:

```
abjad> componenttools.component_to_score_root(note)
Score<<1>>
```

Or to find whether a component has no (proper) parentage at all:

```
abjad> componenttools.is_orphan_component(note)
False
```

CHAPTER TWENTYSEVEN

WORKING WITH THREADS

27.1 What is a thread?

A thread is a structural relationship binding a set of strictly sequential voice-level components.

Threads may be explicitly defined via voice instances:

```
abjad> v = Voice( )
```

Or they may exist implicitly in certain score constructs in the absence of voice containers:

```
abjad> staff = Staff(macros.scale(4))
```

Two contiguous voices must have the same name in order to be part of the same thread.

Here a thread does **not** exist between notes in different voices:

```
abjad> v_one = Voice(macros.scale(4, (1, 16)))
abjad> v_two = Voice(macros.scale(2))
abjad> staff = Staff([v_one, v_two])
abjad> f(staff)
\new Staff {
        \new Voice {
                c'16
                d'16
                e'16
                f'16
        }
        \new Voice {
                c'8
                d'8
        }
}
```

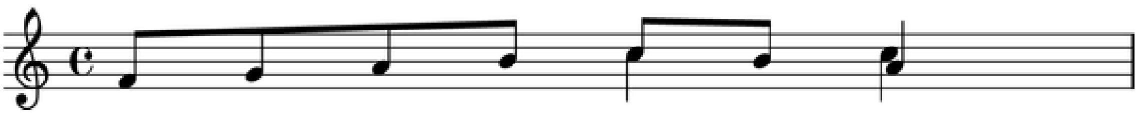
Here a thread does exist:

```
abjad> v_one.name = 'flute'
abjad> v_two.name = 'flute'
abjad> f(staff)
\new Staff {
        \context Voice = "flute" {
                c'16
                d'16
                e'16
                f'16
```

```
        }
        \context Voice = "flute" {
                c'8
                d'8
        }
}
```

27.2 What are threads for?

Consider the following situation:

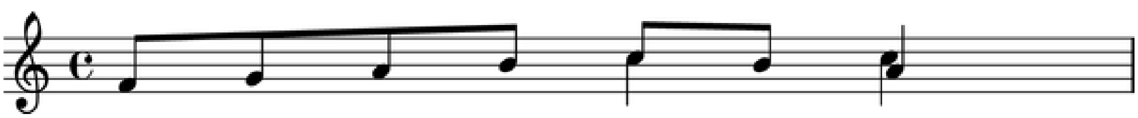


Are the two eighth notes in the second half of the measure the continuation of the ascending line in the first half, or is it the quarter note? Is the very last *C* the continuation of the top melodic line or is it the *A*? The stems might suggest an answer, but for Abjad, stem direction is not structural. What path should Abjad take to traverse this little score from the first note to the last *A*? This same problem appears when trying to apply spanners to parallel structures. Thus, threads are important in both score navigation and the application of spanners. In fact, threads are a requirement for spanner application.

In Abjad, the ambiguity is resolved through the explicit use of named voices.

The musical fragment above is constructed with the following code:

```
abjad> vA = Voice(notetools.make_notes([5, 7, 9, 11], [(1, 8)] * 4))
abjad> vB = Voice(notetools.make_notes([12, 11, 9], [(1, 8), (1, 8), (1, 4)]))
abjad> vC = Voice(Note(12, (1, 4)) * 2)
abjad> marktools.LilyPondCommandMark('voiceOne')(vA[0])
abjad> marktools.LilyPondCommandMark('voiceOne')(vB[0])
abjad> marktools.LilyPondCommandMark('voiceTwo')(vC[0])
abjad> p = Container([vB, vC])
abjad> p.is_parallel = True
abjad> staff = Staff([vA, p])
```



There's a staff that sequentially contains a voice and a parallel container. The container in turn holds two voices running simultaneously.

It is now clear from the code that the last *A* belongs with the two descending eighth notes. But there's still no indication about a relationship of continuity between the first voice in the sequence (*vA*) and any of the two following voices. Note that, while the LilyPond voice number commands setting may suggest that vA and vB belong together, this is not the case. The LilyPond voice number commands simply set the direction of stems in printed output.

To see this more clearly, suppose we want to add a slur spanner starting on the first note and ending on one of the last simultaneous notes. To attach the slur spanner to the voices we could try either:

```
abjad> spannertools.SlurSpanner([vA, vB])
```

or

```
abjad> spannertools.SlurSpanner([vA, vC])
```

But both raise a contiguity error. Abjad needs to see an explicit connection between either *vA* and *vB* or between *vA* and *vC*.

Observe the behavior of the `iterate_thread_forward_in_expr()` iterator on the *staff*:

::

> abjad> vA_thread_signature = threadtools.component_to_thread_signature(vA) abjad> notes = threadtools.iterate_thread_forward_in_expr(staff, Note, vA_thread_signature) abjad> print list(notes) [Note(“f‘8”), Note(“g‘8”), Note(“a‘8”), Note(“b‘8”)]

```
abjad> vB_thread_signature = threadtools.component_to_thread_signature(vB)
abjad> notes = threadtools.iterate_thread_forward_in_expr(staff, Note, vB_thread_signature)
abjad> print list(notes)
[Note("c''8"), Note("b'8"), Note("a'4")]

abjad> vC_thread_signature = threadtools.component_to_thread_signature(vC)
abjad> notes = threadtools.iterate_thread_forward_in_expr(staff, Note, vC_thread_signature)
abjad> print list(notes)
[Note("c''4"), Note("c''4")]
```

In each case we are passing a different **thread signature** to the `iterate_thread_forward_in_expr()` iterator, so each case returns a different list of notes.

We can see that the thread signature of each voice is indeed different by printing it:

```
abjad> vA_thread_signature = threadtools.component_to_thread_signature(vA)
abjad> vA_thread_signature
<       root: Staff-8186656 (8186656) *       score:  * staffgroup:  *       staff: Staff-8186656 *

abjad> vB_thread_signature = threadtools.component_to_thread_signature(vB)
abjad> vB_thread_signature
<       root: Staff-8186656 (8186656) *       score:  * staffgroup:  *       staff: Staff-8186656 *

abjad> vC_thread_signature = threadtools.component_to_thread_signature(vC)
abjad> vC_thread_signature
<       root: Staff-8186656 (8186656) *       score:  * staffgroup:  *       staff: Staff-8186656 *
```

And by comparing them with the binary equality operator:

```
abjad> vA_thread_signature == vB_thread_signature
False
abjad> vA_thread_signature == vC_thread_signature
False
abjad> vB_thread_signature == vC_thread_signature
False
```

To allow Abjad to treat the content of, say, voices *vA* and *vB* as belonging together, we explicitly define a thread between them. To do this all we need to do is give both voices the same name:

```
abjad> vA.name = 'piccolo'
abjad> vB.name = 'piccolo'
```

Now *vA* and *vB* and all their content belong to the same thread:

```
abjad> vA_thread_signature == vB_thread_signature
False
```

Note how the thread signatures have changed:

```
abjad> vA_thread_signature = threadtools.component_to_thread_signature(vA)
abjad> print vA_thread_signature
      root: Staff-8186656 (8186656)
     score:
staffgroup:
     staff: Staff-8186656
     voice: Voice-piccolo
      self: Voice-piccolo

abjad> vB_thread_signature = threadtools.component_to_thread_signature(vB)
abjad> print vB_thread_signature
      root: Staff-8186656 (8186656)
     score:
staffgroup:
     staff: Staff-8186656
     voice: Voice-piccolo
      self: Voice-piccolo

abjad> vC_thread_signature = threadtools.component_to_thread_signature(vC)
abjad> print vC_thread_signature
      root: Staff-8186656 (8186656)
     score:
staffgroup:
     staff: Staff-8186656
     voice: Voice-8186544
      self: Voice-8186544
```

And how the `threadtools.iterate_thread_forward_in_expr( )` function returns all the notes belonging to both *vA* and *vB* when passing it the full staff and the thread signature of *vA*:

```
abjad> notes = threadtools.iterate_thread_forward_in_expr(staff, Note, vA_thread_signature)
abjad> print list(notes)
[Note("f'8"), Note("g'8"), Note("a'8"), Note("b'8"), Note("c''8"), Note("b'8"), Note("a'4")]
```

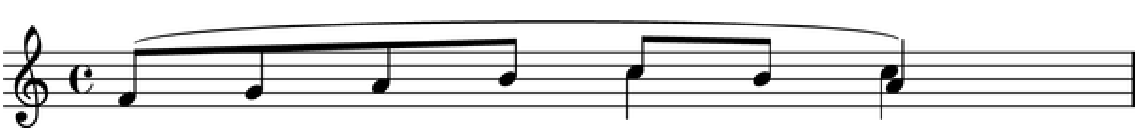
Now the slur spanner can be applied to voices *vA* and *vB*:

```
abjad> spannertools.SlurSpanner([vA, vB])
```

or directly to the notes returned by the `iterate_thread_forward_in_expr()` iteration tool, which are the notes belonging to both *vA* and *vB*:

```
abjad> notes = threadtools.iterate_thread_forward_in_expr(staff, Note, vA_thread_signature)
abjad> spannertools.SlurSpanner(list(notes))
```

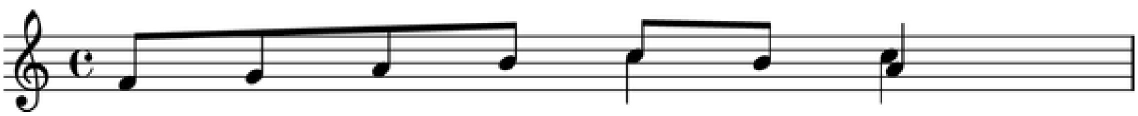
```
abjad> show(staff)
```



27.3 Coda

We could have constructed this score in a simpler way with only two voices, one of them starting with a LilyPond skip:

```
abjad> vX = Voice(notetools.make_notes([5, 7, 9, 11, 12, 11, 9], [(1, 8)] * 6 + [(1, 4)]))
abjad> vY = Voice([skiptools.Skip((2, 4))] + Note(12, (1, 4)) * 2)
abjad> marktools.LilyPondCommandMark('voiceOne')(vX[0])
abjad> marktools.LilyPondCommandMark('voiceTwo')(vY[0])
abjad> staff = Staff([vX, vY])
abjad> staff.is_parallel = True
```



Reference manual — tuplets

CHAPTER

TWENTYEIGHT

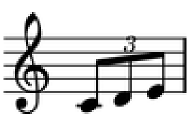
WORKING WITH TUPLETS

28.1 Making a tuplet from a LilyPond input string

You can make an Abjad tuplet from a multiplier and a LilyPond input string:

```
abjad> tuplet = Tuplet(Fraction(2, 3), "c'8 d'8 e'8")
```

```
abjad> show(tuplet)
```



28.2 Making a tuplet from a list of other Abjad components

You can also make a tuplet from a multiplier and a list of other Abjad components:

```
abjad> leaves = [Note("fs'8"), Note("g'8"), Rest('r8')]
```

```
abjad> tuplet = Tuplet(Fraction(2, 3), leaves)
```

```
abjad> show(tuplet)
```

28.3 Understanding the interpreter display of a tuplet

The interprer display of an Abjad tuplet contains three parts:

```
abjad> tuplet
Tuplet(2/3, [fs'8, g'8, r8])
```

`Tuplet` tells you the tuplet's class.

`2/3` tells you the tuplet's multiplier.

The list `[fs'8, g'8, r8]` shows the top-level components the tuplet contains.

28.4 Understanding the string representation of a tuplet

The string representation of a tuplet contains four parts:

```
abjad> print tuplet
{* 3:2 fs'8, g'8, r8 *}
```

Curly braces `{` and `}` indicate that the tuplet's music is interpreted sequentially instead of in parallel.

The asterisks `*` denote a fixed-multiplier tuplet.

`3:2` tells you the tuplet's ratio.

The remaining arguments show the top-level components of tuplet.

28.5 Inspecting the LilyPond format of a tuplet

Get the LilyPond input format of any Abjad object with `format`:

```
abjad> tuplet.format
"\\times 2/3 {\n\tfs'8\n\tg'8\n\tr8\n}"
```

Use `f( )` as a short-cut to print the LilyPond format of any Abjad object:

```
abjad> f(tuplet)
\times 2/3 {
        fs'8
        g'8
        r8
}
```

28.6 Inspecting the music in a tuplet

Get the music in any Abjad container with `music`:

```
abjad> tuplet.music
(Note("fs'8"), Note("g'8"), Rest('r8'))
```

Abjad returns a read-only tuple of components.

28.7 Inspecting a tuplet's leaves

Get the leaves in any Abjad container with `leaves`:

```
abjad> tuplet.leaves
(Note("fs'8"), Note("g'8"), Rest('r8'))
```

Abjad returns a read-only tuple of leaves.

28.8 Getting the length of a tuplet

Get the length of any Abjad container with `len( )`:

```
abjad> len(tuplet)
3
```

The length of every Abjad container is defined equal to the number of top-level components present in the container.

28.9 Getting the duration attributes of a tuplet

You set the multiplier of a tuplet at initialization:

```
abjad> tuplet.duration.multiplier
Fraction(2, 3)
```

The contents durations of a tuplet equals the sum of written durations of the components in the tuplet:

```
abjad> tuplet.duration.contents
Duration(3, 8)
```

The multiplied duration of a tuplet equals the product of the tuplet's multiplier and the tuplet's contents duration:

```
abjad> tuplet.duration.multiplied
Duration(1, 4)
```

28.10 Understanding rhythmic augmentation and diminution

A tuplet with a multiplier less than `1` constitutes a type of rhythmic diminution:

```
abjad> tuplet.duration.multiplier
Fraction(2, 3)

abjad> tuplet.duration.is_diminution
True
```

A tuplet with a multiplier greater than `1` is a type of rhythmic augmentation:

```
abjad> tuplet.duration.is_augmentation
False
```

28.11 Understanding binary and nonbinary tuplets

A tuplet is considered binary if the numerator of the tuplet multiplier is an integer power of `2`:

```
abjad> tuplet.duration.multiplier
Fraction(2, 3)

abjad> tuplet.duration.is_binary
True
```

Other tuplets are nonbinary:

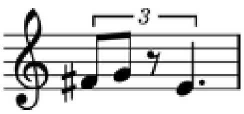
```
abjad> tuplet.duration.is_nonbinary
False
```

28.12 Adding one component to the end of a tuplet

Add one component to the end of a tuplet with `append`:

```
abjad> tuplet.append(Note("e'4."))
```

```
abjad> show(tuplet)
```

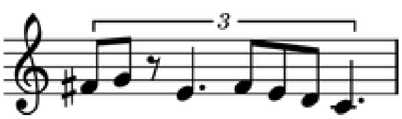


28.13 Adding many components to the end of a tuplet

Add many components to the end of a tuplet with `extend`:

```
abjad> notes = [Note("fs'8"), Note("e'8"), Note("d'8"), Note("c'4.")]
abjad> tuplet.extend(notes)
```

```
abjad> show(tuplet)
```



28.14 Finding the index of a component in a tuplet

Find the index of a component in a tuplet with `index()`:

```
abjad> notes[1]
Note("e'8")
```

```
abjad> tuplet.index(notes[1])
5
```

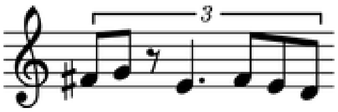
28.15 Removing a tuplet component by index

Use `pop()` to remove a tuplet component by index:

```
abjad> tuplet[7]
Note("c'4.")
```

```
abjad> tuplet.pop(7)
```

```
abjad> show(tuplet)
```

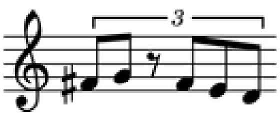


28.16 Removing a tuplet component by reference

Remove tuplet components by reference with `remove( )`:

```
abjad> tuplet.remove(tuplet[3])
```

```
abjad> show(tuplet)
```



28.17 Overriding attributes of the LilyPond tuplet number grob

Override attributes of the LilyPond tuplet number grob like this:

```
abjad> tuplet.override.tuplet_number.text = schemetools.SchemeFunction('tuplet-number::calc-fraction-
abjad> tuplet.override.tuplet_number.color = 'red'
```

```
abjad> f(tuplet)
\override TupletNumber #'color = #red
\override TupletNumber #'text = #tuplet-number::calc-fraction-text
\times 2/3 {
        fs'8
        g'8
        r8
        fs'8 [
        e'8
        d'8 ]
}
\revert TupletNumber #'color
\revert TupletNumber #'text
```

```
abjad> show(tuplet)
```

See the LilyPond docs for lists of grob attributes available.

28.18 Overriding attributes of the LilyPond tuplet bracket grob

Override attributes of the LilyPond tuplet bracket grob like this:

```
abjad> tuplet.override.tuplet_bracket.color = 'red'
```

```
abjad> f(tuplet)
\override TupletBracket #'color = #red
\override TupletNumber #'color = #red
\override TupletNumber #'text = #tuplet-number::calc-fraction-text
\times 2/3 {
        fs'8
        g'8
        r8
        fs'8 [
        e'8
        d'8 ]
}
\revert TupletBracket #'color
\revert TupletNumber #'color
\revert TupletNumber #'text
```

```
abjad> show(tuplet)
```

See the LilyPond docs for lists of grob attributes available.

Reference manual — voices

CHAPTER

TWENTYNINE

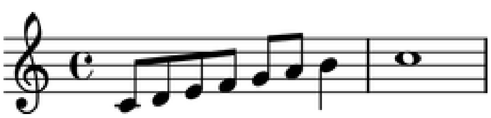
WORKING WITH VOICES

29.1 Making a voice from a LilyPond input string

You can make an Abjad voice from a LilyPond input string:

```
abjad> voice = Voice("c'8 d'8 e'8 f'8 g'8 a'8 b'4 c''1")
```

```
abjad> show(voice)
```



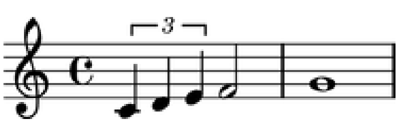
29.2 Making a voice from a list of other Abjad components

You can also make a voice from a list of other Abjad components:

```
abjad> components = [Tuplet(Fraction(2, 3), "c'4 d'4 e'4"), Note("f'2"), Note("g'1")]
```

```
abjad> voice = Voice(components)
```

```
abjad> show(voice)
```



29.3 Understanding the `repr` of a voice

The `repr` of an Abjad voice contains three parts:

```
abjad> voice
Voice{3}
```

`Voice` tells you the voice's class.

`3` tells you the voice's length (which is the number of top-level components the voice contains).

Curly braces `{` and `}` tell you that the music inside the voice is interpreted sequentially rather than in parallel.

29.4 Inspecting the LilyPond format of a voice

Get the LilyPond input format of any Abjad object with `format`:

```
abjad> voice.format
"\\new Voice {\n\t\\times 2/3 {\n\t\tc'4\n\t\td'4\n\t\te'4\n\t}\n\tf'2\n\tg'1\n}"
```

Use `f( )` as a short-cut to print the LilyPond format of any Abjad object:

```
abjad> f(voice)
\new Voice {
        \times 2/3 {
                c'4
                d'4
                e'4
        }
        f'2
        g'1
}
```

29.5 Inspecting the music in a voice

Get voice components with `music`:

```
abjad> voice.music
(Tuplet(2/3, [c'4, d'4, e'4]), Note("f'2"), Note("g'1"))
```

Abjad returns a read-only tuple of components.

29.6 Inspecting a voice's leaves

Get the leaves in a voice with `leaves`:

```
abjad> voice.leaves
(Note("c'4"), Note("d'4"), Note("e'4"), Note("f'2"), Note("g'1"))
```

Abjad returns a read-only tuple of leaves.

29.7 Getting the length of a voice

Get voice length with `len( )`:

```
abjad> len(voice)
3
```

The length of a voice is defined equal to the number of top-level components the voice contains.

29.8 Getting the duration attributes of a voice

The contents durations of a voice equals the sum of durations of the components in the voice:

```
abjad> voice.duration.contents
Duration(2, 1)
```

The preprolated duration of a voice is usually equal to the voice's contents duration:

```
abjad> voice.duration.preprolated
Duration(2, 1)
```

The prolated duration of a voice is usually equal to the voice's contents duration, too:

```
abjad> voice.duration.preprolated
Duration(2, 1)
```

Only when you nest a very small voice inside a tuplet will the prolated and preprolated duration of a voice differ.

Voices that are not nested inside a tuplet carry a prolation of `1`:

```
abjad> voice.duration.prolation
Fraction(1, 1)
```

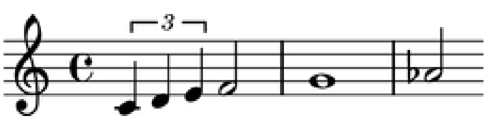
All voice duration attributes are read-only.

29.9 Adding one component to the end of a voice

Add one component to the end of a voice with `append`:

```
abjad> voice.append(Note("af'2"))
```

```
abjad> show(voice)
```



29.10 Adding many components to the end of a voice

Add many components to the end of a voice with `extend`:

```
abjad> notes = [Note("g'4"), Note("f'4")]
abjad> voice.extend(notes)
```

```
abjad> show(voice)
```

29.11 Finding the index of a component in a voice

Find the index of a component in a voice with `index( )`:

```
abjad> notes[0]
Note("g'4")
```

```
abjad> voice.index(notes[0])
4
```

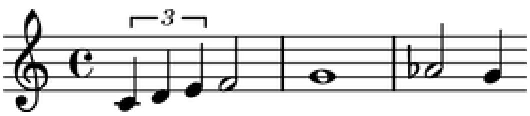
29.12 Removing a voice component by index

Use `pop( )` to remove a voice component by index:

```
abjad> voice[5]
Note("f'4")
```

```
abjad> voice.pop(5)
```

```
abjad> show(voice)
```

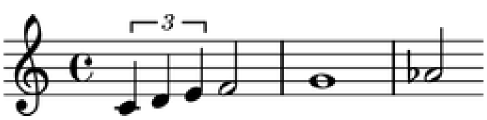


29.13 Removing a voice component by reference

Remove voice components by reference with `remove( )`:

```
abjad> voice.remove(voice[-1])
```

```
abjad> show(voice)
```



29.14 Naming voices

You can name Abjad voices:

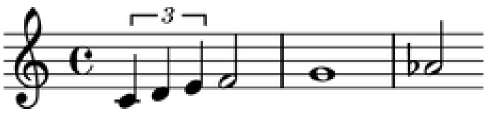
```
abjad> voice.name = 'Upper Voice'
```

Voice names appear in LilyPond input:

```
abjad> f(voice)
\context Voice = "Upper Voice" {
        \times 2/3 {
                c'4
                d'4
                e'4
        }
        f'2
        g'1
        af'2
}
```

But not in notational output:

```
abjad> show(voice)
```



29.15 Changing the context of a voice

The context of a voice is set to `'Voice'` by default:

```
abjad> voice.context
'Voice'
```

But you can change the context of a voice if you want:

```
abjad> voice.context = 'SpeciallyDefinedVoice'

abjad> voice.context
'SpeciallyDefinedVoice'

abjad> f(voice)
\context SpeciallyDefinedVoice = "Upper Voice" {
        \times 2/3 {
                c'4
                d'4
                e'4
        }
        f'2
        g'1
        af'2
}
```

Change the context of a voice when you have defined a new LilyPond context based on a LilyPond voice.

Advanced tutorials

CHAPTER

THIRTY

TIME SIGNATURE MARKS BY EXAMPLE

In this tutorial is to take a deeper look at what happens when we attach time signature marks to staves and other score components. To work through the tutorial, enter each of the examples into the Abjad interpreter and study what comes back. At the end of the tutorial you'll understand how time signature marks are created. You'll also understand how the states of different objects change when time signature marks are attached and detached.

First we start by creating a staff full of notes:

```
abjad> staff = Staff("c'4 d'4 e'4 f'4 g'2")
```

If we ask the Abjad interpreter about our staff reference Abjad will respond with the interpreter display of the object:

```
abjad> staff
Staff{5}
```

The `5` in `Staff{5}` shows that the staff contains 5 top-level components. The curly braces in `Staff{5}` show that the contents of the staff are to be read sequentially through time rather than in parallel.

Before we get to time signature marks let's take a moment and examine the state of the staff we've created. We can motivate this a bit by asking two questions:

1. what time signature is currently in effect for the staff we have just created?
2. what is the time signature currently in effect for the five notes contained within the staff we have just created?

The answer to both questions is the same: there is no time signature currently in effect for either our staff or for the five notes it contains.

We can see that this is the case with tools from the API:

```
abjad> contexttools.get_effective_time_signature(staff) is None
True
```

And:

```
abjad> for leaf in staff:
...   contexttools.get_effective_time_signature(leaf) is None
...
True
True
True
True
True
```

If we want, we can iterate both the staff and its leaves at one and the same time like this:

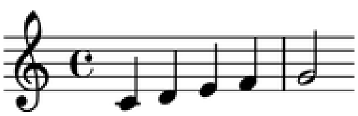
```
abjad> for component in componenttools.iterate_components_forward_in_expr(staff):
...   component, contexttools.get_effective_time_signature(component)
...
(Staff{5}, None)
(Note("c'4"), None)
(Note("d'4"), None)
(Note("e'4"), None)
(Note("f'4"), None)
(Note("g'2"), None)
```

This confirms the answer to our questions that there is not yet any time signature in effect for any component in our staff because we have not yet attached a time signature mark to any component in our staff.

So what happens if we format our staff and send it off to LilyPond to render as a PDF? Will LilyPond render the staff with a time signature? Without a time signature? Will LilyPond refuse to render the example at all?

We find out like this:

```
abjad> show(staff)
```



It turns out LilyPond defaults to a time signature of `4/4`.

What's important to note here is that because we have not yet attached a time signature mark any component in our staff Abjad says "no effective time signature here" while LilyPond says "OK, I'll default to `4/4` so we can get on with rendering your music."

We can further confirm that this is the case by asking Abjad for the LilyPond format of our staff:

```
abjad> f(staff)
\new Staff {
        c'4
        d'4
        e'4
        f'4
        g'2
}
```

The LilyPond format of our staff contains no LilyPond `\time` command. This is, again, because we have not yet attached a time signature mark to any component in our staff.

We can no practice attaching and detaching time signature marks to different components in our staff and study what happens as we do.

We'll start with `3/4`.

The easiest thing to do is to attach a time signature mark to the staff itself.

We'll do this in two separate steps and study each step to understand exactly what's going on.

First, we create a `3/4` time signature mark:

```
abjad> time_signature_mark = contexttools.TimeSignatureMark(3, 4)
```

If we ask the Abjad interpreter for the interpreter dispaly of our time signature mark we get the following:

```
abjad> time_signature_mark
TimeSignatureMark(3, 4)
```

All this tells us is that we have in fact created a 3/4 time signature mark. Nothing too exciting yet. At this point our 3/4 time signature is not yet attached to anything. We could say that the "state" of our time signature mark is "unattached." And we can see this like so:

```
abjad> time_signature_mark.start_component is None
True
```

What does it mean for a time signature mark to have 'start_component' equal to none? It means that the time signature isn't yet attached to any score component anywhere.

So now we attach our time signature mark to our staff:

```
abjad> time_signature_mark.attach_mark(staff)
TimeSignatureMark(3, 4)(Staff{5})
```

Abjad responds immediately by returning the time signature mark we have just attached.

Notice that our time signature mark's repr ha changed. The repr of our 3/4 time signature mark now includes the repr of the staff to which we have just attached the time signature mark. That is to say that the repr of our time signature mark is statal.

Our time signature mark has transitioned from an "unattached" state to an "attached" state. We can see this like so:

```
abjad> time_signature_mark.start_component
Staff{5}
```

And our staff has likewise transitioned from a state of having no effective time signature to a state of having an effective time signature:

```
abjad> contexttools.get_effective_time_signature(staff)
TimeSignatureMark(3, 4)(Staff{5})
```

And what about the leaves inside our staff? Do the leaves now "know" that they are governed by a 3/4 time signature?

Indeed they do:

```
abjad> for leaf in staff.leaves:
...     leaf, contexttools.get_effective_time_signature(leaf)
...
(Note("c'4"), TimeSignatureMark(3, 4)(Staff{5}))
(Note("d'4"), TimeSignatureMark(3, 4)(Staff{5}))
(Note("e'4"), TimeSignatureMark(3, 4)(Staff{5}))
(Note("f'4"), TimeSignatureMark(3, 4)(Staff{5}))
(Note("g'2"), TimeSignatureMark(3, 4)(Staff{5}))
```

So to briefly resume:

What we just did was to:

1. create a time signature mark
2. attach the time signature to a score component

This 2-step pattern is always the same when dealing with context marks: create then attach.

(We will find out later that there are short-cuts for different parts of this process. Right now we've chosen to create in a first step and attach in a second step so that we can examine the changing states of the objects involved.)

Before moving on let's look at the PDF corresponding to our staff:

```
abjad> show(staff)
```

And let's confirm what we see in the PDF in the staff's format:

```
abjad> f(staff)
\new Staff {
        \time 3/4
        c'4
        d'4
        e'4
        f'4
        g'2
}
```

The staff's format now contains a LilyPond `\time` command because we have attached an Abjad time signature mark to the staff.

What we've just been through above will cover over 80% of what you'll ever wind up doing with time signature marks: creating them and attaching them directly to staves. But what if we wanna get rid of a time signature mark? Or what if the time signature will be changing all over the place? We cover those cases next.

Detaching a time signature mark is easy:

```
abjad> time_signature_mark.detach_mark( )
TimeSignatureMark(3, 4)
```

The Abjad returns the mark we have just detached. And, observing the repr of the time signature mark, we see that the time signature mark has again changed state: the time signature mark has transitioned from attached to unattached. We confirm this like so:

```
abjad> time_signature_mark.start_component is None
True
```

And also like so:

```
abjad> contexttools.get_effective_time_signature(staff) is None
True
```

Yup: our time signature mark knows nothing about our staff. And vice versa. This is good.

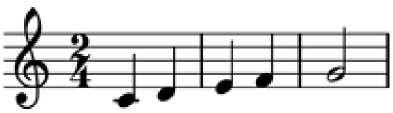
So now what if we want to set up a time signature of 2/4? That fits our music, too.

We have a couple of options.

We can simply create and attach a new time signature mark:

```
abjad> duple_time_signature_mark = contexttools.TimeSignatureMark(2, 4)
abjad> duple_time_signature_mark.attach_mark(staff)
TimeSignatureMark(2, 4)(Staff{5})

abjad> f(staff)
\new Staff {
        \time 2/4
        c'4
        d'4
        e'4
        f'4
        g'2
}
```

```
abjad> show(staff)
```



Yup. That works.

On the other hand, we could simply reuse our previous `3/4` time signature mark.

To do this we'll first detach our `2/4` time signature mark ...

```
abjad> duple_time_signature_mark.detach_mark( )
```

```
abjad> duple_time_signature_mark.detach_mark( )
TimeSignatureMark(2, 4)
```

... confirm that our staff is now time signatureless ...

```
abjad> contexttools.get_effective_time_signature(staff) is None
True
```

```
abjad> f(staff)
\new Staff {
        c'4
        d'4
        e'4
        f'4
        g'2
}
```

... reattach our previous `3/4` time signature ...

```
abjad> time_signature_mark.attach_mark(staff)
```

```
abjad> time_signature_mark.attach_mark(staff)
TimeSignatureMark(4, 4)(Staff{5})
```

... change the numerator of our time signature mark ...

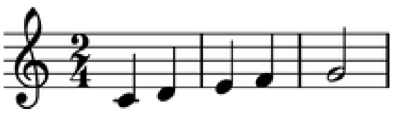
```
abjad> time_signature_mark.numerator = 2
```

... and check to make sure that everything is as it should be:

```
abjad> contexttools.get_effective_time_signature(staff)
TimeSignatureMark(2, 4)(Staff{5})
abjad> time_signature_mark.start_component
Staff{5}
```

```
abjad> f(staff)
\new Staff {
        \time 2/4
        c'4
        d'4
        e'4
        f'4
        g'2
}
```

```
abjad> show(staff)
```



And everything works as it should.

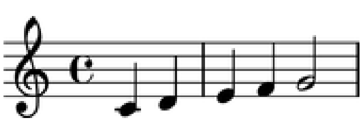
To change to, for example, `4/4` we change just change the time signature mark's numerator again:

```
abjad> time_signature_mark.numerator = 4

abjad> f(staff)
\new Staff {
        \time 4/4
        c'4
        d'4
        e'4
        f'4
        g'2
}
```

But what if our time signature has a `2/4` pick-up?

The LilyPond command for pick-ups is `\partial`. Abjad time signature marks implement this as a read / write attribute:

```
abjad> time_signature_mark.partial = Duration(2, 4)

abjad> f(staff)
\new Staff {
        \partial 2
        \time 4/4
        c'4
        d'4
        e'4
        f'4
        g'2
}

abjad> show(staff)
```



And what if time signature changes all over the place?

We'll use the trivial example of a measure in `4/4` followed by a measure in `2/4`.

To do this we will need two time signature marks.

We've already got a `4/4` time signature mark attached to our staff:

```
abjad> f(staff)
\new Staff {
        \partial 2
        \time 4/4
        c'4
        d'4
```

```
        e'4
        f'4
        g'2
}
```

Let's get rid of the pick-up:

```
abjad> time_signature_mark.partial = None

abjad> f(staff)
\new Staff {
        \time 4/4
        c'4
        d'4
        e'4
        f'4
        g'2
}
```

Now what about the `2/4` time signature mark?

We create it in the usual way:

```
abjad> duple_time_signature_mark = contexttools.TimeSignatureMark(2, 4)
abjad> duple_time_signature_mark
TimeSignatureMark(2, 4)
```

But should we attach it? We can't attach our `2/4` time signature to our staff because we've already attached our `4/4` time signature to our staff. And it only makes sense to attach one time signature to any given score component.

Observe that we've built our score in a very straightforward way: we have a single staff that contains a (flat) sequence of notes. This means that we have only one choice for where to attach the new `2/4` time signature mark. And that is one the `g'2` that comes on the downbeat of the second measure. We do that like this:

```
abjad> duple_time_signature_mark.attach_mark(staff[4])

abjad> duple_time_signature_mark.attach_mark(staff[4])
TimeSignatureMark(2, 4)(g'2)

abjad> f(staff)
\new Staff {
        \time 4/4
        c'4
        d'4
        e'4
        f'4
        \time 2/4
        g'2
}

abjad> show(staff)
```

And everything works as we would like.

Incidentally, `staff[4]` means the component sitting at index `4` inside our staff. Using the interpreter we can verify that this is `g'2`:

```
abjad> staff[4]
Note("g'2")
```

Depending on how we had chosen to build our staff we would have had more options for where to attach our `2/4` time signature mark. If, for example, we had chosen to populate our staff with a series of measures then it's possible we could have attached our `2/4` time signature to a measure instead of a note.

That covers the vast majority of things you'll do with time signature marks.

But before we stop we should mention another useful API function and then talk about some short-cuts.

First an API function to detach ALL context marks attaching to a component:

We call the function a first time:

```
abjad> contexttools.detach_all_context_marks_attached_to_component(staff)
(TimeSignatureMark(4, 4),)

abjad> f(staff)
\new Staff {
        c'4
        d'4
        e'4
        f'4
        \time 2/4
        g'2
}
```

And then a second time:

:: abjad> contexttools.detach_all_context_marks_attached_to_component(staff[4]) (TimeSignatureMark(2, 4),)

```
abjad> f(staff)
\new Staff {
        c'4
        d'4
        e'4
        f'4
        g'2
}
```

Now there are now context marks of any sort attached to our staff or to the notes in our staff.

Be careful with this function, though: it removes *all* context marks. So even though we just used the function to remove time signature marks, it also would have removed any clef marks or tempo marks if we had had those attached to our score, too.

And now for the short-cuts:

Our staff currently has no time signature marks attached:

```
abjad> f(staff)
\new Staff {
        c'4
        d'4
        e'4
        f'4
        g'2
}
```

So to recreate our `3/4` time signature we can do this ...

```
abjad> time_signature_mark = contexttools.TimeSignatureMark(3, 4)
```

... and then use a short-cut to avoid calling `time_signature_mark.attach_mark( )` like this:

```
abjad> time_signature_mark(staff)
TimeSignatureMark(3, 4)(Staff{5})

abjad> f(staff)
\new Staff {
        \time 3/4
        c'4
        d'4
        e'4
        f'4
        g'2
}
```

What's going on here is that all context marks implement the special `__call__( )` method as a short-cut for `attach_mark( )`. What is the special `__call__( )` method? The `__call__( )` method is what makes a function, class or any other Python object callable. The statement `time_signature_mark(staff)` has has parentheses in it because the time signature mark is callable; and the time signature mark is callable because all context marks implement the special `__call__( )` method.

Note too that all context marks understand an *empty call* as a short-cut for `detach_mark( )`. Like this:

```
abjad> time_signature_mark( )
TimeSignatureMark(3, 4)

abjad> f(staff)
\new Staff {
        c'4
        d'4
        e'4
        f'4
        g'2
}
```

The empty call made against the time signature mark causes the time signature mark to detach from its start component.

The fact that context marks implement the special `__call__( )` method as a short-cut for attach_mark( ) means that context marks can be created and attached in a single line:

```
abjad> contexttools.TimeSignatureMark(2, 4)(staff)
TimeSignatureMark(2, 4)(Staff{5})

abjad> f(staff)
\new Staff {
        \time 2/4
        c'4
        d'4
        e'4
        f'4
        g'2
}
```

What's going on here?

What's going on is that `contexttools.TimeSignatureMark(2, 4)` creates a time signature mark in the usual way and that – immediately after this – the newly created time signature mark is available for us to call it against our staff.

This last short-cut form of ...

```
abjad> contexttools.TimeSignatureMark(2, 4)(staff)
```

... is the usual way that you will see context marks of all sorts presented in the docs.

Developer documentation

CHAPTER

THIRTYONE

CODEBASE

31.1 How the Abjad codebase is laid out

The Abjad codebase comprises several dozen different Python packages that together implement a formal model of the musical score. Abjad r2330 includes 73 top-level packages.

```
abjad$ ls

__init__.py   cluster       hairpin      octavation   spanner
__init__.pyc  comments      harmonic     offset       staff
accidental    component     instrument   override     staffgroup
articulations container     interfaces   parentage    stem
barline       context       layout       pianopedal   templates
barnumber     core          leaf         pitch        tempo
beam          debug         lily         rational     text
book          demos         markup       receipt      thread
bracket       directives    measure      rest         tie
brackets      documentation meter        scm          tools
breaks        dots          metricgrid   score        tremolo
cfg           dynamics      navigator    scr          trill
checks        exceptions    note         skip         tuplet
chord         glissando     notehead     slur         update
clef          grace         numbering    spacing      voice
```

The remaining sections of this chapter cover the topics necessary to familiarize developers coming to the project for the first time.

31.2 Installing the development version

If you'd like to be at the cutting edge of the Abjad development you should install Subversion on your local machine, check out from Google Code, and then tell Python and your operating system about Abjad.

1. Install Subversion.

   You can check to see if Subversion is already installed on your machine first.

   ```
   svn help
   ```

   If Subversion responds then it is already installed. Otherwise visit the Subversion website.

2. Check out the Abjad codebase.

```
svn checkout http://abjad.googlecode.com/svn/abjad/trunk abjad-trunk
```

3. Make the Python interpreter aware of Abjad. Symlink your Python `site-packages/` directory to the `abjad-trunk/` directory. [1]

   ```
   ln -s /path/to/abjad-trunk SITE-PACKAGES-DIR/abjad
   ```

4. Alternatively, you can include the `abjad-trunk` directory in your `PYTHONPATH` environment variable.

   ```
   export PYTHONPATH="/path/to/abjad-trunk:"$PYTHONPATH
   ```

5. Finally, add the `abjad-trunk/scr/` directory to your `PATH`.

   ```
   export PATH="/path/to/abjad-trunk/scr:"$PATH
   ```

   You will then be able to run Abjad directly with the `abj` command.

Notes

[1] `SITE-PACKAGES-DIR` should be the Python `site-packages/` directory. The Linux `site-packages/` directory is usually `/usr/lib/python2.x/site-packages`.

CHAPTER

THIRTYTWO

DOCS

The Abjad documentation is included in its entirety when you check out the Abjad codebase. You may add to and edit the docs as soon as you download and install Abjad. However, to build HTML or PDF versions of the docs and see the results of your changes you will first need to download and install Sphinx, the automated documentation build and management system used by Abjad and a number of other Python projects, including www.python.org.

This remaining sections of this chapter describe how to find and edit the Abjad docs, and how to build the the docs with Sphinx.

32.1 How the docs are laid out

The Abjad documentation source files are included in the `documentation` directory of every Abjad download.

```
abjad$ ls -d d*

debug         directives    dots
demos         documentation dynamics
```

The `documentation` directory contains everything required to build HTML, LaTeX and PDF versions of the Abjad docs, including the page that you're reading now. List the contents of the `documentation` directory and take a look around.

```
abjad$ ls documentation

Makefile    _templates chapters    index.rst  scr
_static     _themes    conf.py     make.bat
```

The core content of the Abjad docs lives in `documentation/chapters`.

```
abjad$ ls documentation/chapters/

api          background   fundamentals tutorial
appendices   developers   introduction
```

The `documentation/chapters` subdirectories mirror the main sections on the front page of the Abjad docs.

What you'll find as you inspect the chapters directories, or as you consider adding a new chapter directory, are a collection of `.rst` files organized into directories. The `.rst` extension identifies files written in restructured text, or reST, described more fully below. [1]

[1] Restructured text is abbreviated REST or REST and should not be confused with the REST and SOAP protocols in use in other development projects on the Web.

```
abjad$ ls documentation/chapters/appendices/glossary

index.rst
```

32.2 Running `make clean`

After you have downloaded and installed Sphinx, change to the Abjad `documentation` directory and use the Sphinx makefile to remove any existing `documentation/_build` directory prior to making a new build of the docs.

```
abjad$ cd documentation
documentation$ make clean

rm -rf _build/*
```

This removes the `documentation/_build` directory and its contents. After `make clean` feel free to build new HTML or other versions of of the docs as described in the following sections.

32.3 Autogenerating the Abjad API

The `documentation/scr` directory includes a script to autogenerate the Abjad API. Run this script before building the main part of the docs for the first time.

```
documentation$ scr/make-abjad-api
Run script in interactive mode? [Y/n]: n

Writing file /Users/trevorbaca/Documents/abjad/trunk/abjad/documentation/chapter
s/api/accidental/accidental.rst ...
Writing file /Users/trevorbaca/Documents/abjad/trunk/abjad/documentation/chapter
s/api/accidental/interface.rst ...
Writing file /Users/trevorbaca/Documents/abjad/trunk/abjad/documentation/chapter
s/api/articulations/articulation.rst ...

... (many lines omitted) ...

Writing file /Users/trevorbaca/Documents/abjad/trunk/abjad/documentation/chapter
s/api/voice/voice.rst ...
Writing file /Users/trevorbaca/Documents/abjad/trunk/abjad/documentation/chapter
s/api/voice/interface/interface.rst ...
Writing file /Users/trevorbaca/Documents/abjad/trunk/abjad/documentation/chapter
s/api/index.rst ...
```

Rerun `make-abjad-api` any time you add or remove a public class, method or function from the codebase.

32.4 Building the HTML docs

Change to the Abjad `documentation` directory and run `make html`.

```
abjad$ cd documentation
documentation$ make html

Making output directory...
Running Sphinx v0.6.1
```

```
loading pickled environment... not found
building [html]: targets for 568 source files that are out of date
updating environment: 568 added, 0 changed, 0 removed
reading sources... [ 13%] chapters/api/debug/debugghandlertoregatorsg
reading sources... [ 37%] chapters/api/tools/clonewp/by_leaf_counts_with_parenta
reading sources... [ 38%] chapters/api/tools/clonewp/by_leaf_range_with_parentag
reading sources... [ 38%] chapters/api/tools/componenttools/get_duration_crosser
reading sources... [ 38%] chapters/api/tools/componenttools/get_duration_preprol
reading sources... [ 39%] chapters/api/tools/componenttools/get_le_duration_prol

... (many more lines omitted) ...

writing output... [ 85%] chapters/api/tools/spannertools/give_attached_to_childr
writing output... [ 95%] chapters/fundamentals/duration/interfaces_compared/inde
writing output... [100%] index                  /indexdexexexng/indexxdexindex
writing additional files... genindex modindex search
copying images... done
copying static files... done
dumping search index... done
dumping object inventory... done
build succeeded.

Build finished. The HTML pages are in _build/html.
```

You will then find the complete HTML version of the docs in `documentation/_build/html`.

```
documentation$ ls _build/
doctress html
```

The output from Sphinx is verbose the first time you build the docs. On sequent builds, Sphinx reports changes only.

```
documentation$ make html

sphinx-build -b html -d _build/doctrees   . _build/html
Running Sphinx v0.6.1
loading pickled environment... done
building [html]: targets for 1 source files that are out of date
updating environment: 0 added, 1 changed, 0 removed
reading sources... [100%] chapters/devel/documentation/index
looking for now-outdated files... none found
pickling environment... done
checking consistency... done
preparing documents... done
writing output... [100%] index                  ation/index
writing additional files... genindex modindex search
copying static files... done
dumping search index... done
dumping object inventory... done
build succeeded.

Build finished. The HTML pages are in _build/html.
```

32.5 Building a coverage report

Change to the Abjad `documentation` directory and call `sphinx-build` explicitly with the coverage builder, source directory and target directory.

```
documentation$ sphinx-build -b coverage . _build/coverage
Making output directory...
Running Sphinx v0.6.1
loading pickled environment... not found
building [coverage]: coverage overview
updating environment: 568 added, 0 changed, 0 removed
reading sources... [ 37%] chapters/api/tools/clonewp/by_leaf_counts_with_parenta
reading sources... [ 38%] chapters/api/tools/clonewp/by_leaf_range_with_parentag
reading sources... [ 38%] chapters/api/tools/componenttools/get_duration_crosser

... (many lines omitted) ...

reading sources... [ 85%] chapters/api/tools/spannertools/withdraw_from_containe
reading sources... [ 95%] chapters/fundamentals/duration/interfaces_compared/ind
reading sources... [100%] index                t/indexdexexexng/indexxdexindex
looking for now-outdated files... none found
pickling environment... done
checking consistency... done
build succeeded.
```

The coverage report is now available in the `documentation/_build/coverage` directory.

```
documentation$ ls _build/
coverage doctrees html
```

32.6 Building other versions of the docs

Examine the Sphinx makefile in the Abjad `documentation/` directory or change to the `documentation/` directory and type `make` with no arguments to see a list of the other versions of the Abjad docs that are available to build.

```
documentation$ make

Please use 'make <target>' where <target> is one of
  html      to make standalone HTML files
  dirhtml   to make HTML files named index.html in directories
  pickle    to make pickle files
  json      to make JSON files
  htmlhelp  to make HTML files and a HTML help project
  qthelp    to make HTML files and a qthelp project
  latex     to make LaTeX files, you can set PAPER=a4 or PAPER=letter
  changes   to make an overview of all changed/added/deprecated items
  linkcheck to check all external links for integrity
  doctest   to run all doctests embedded in the documentation (if enabled)
```

32.7 Inserting images with `abjad-book`

Use *abjad-book* to insert snippets of notation in the docs you write in ReST.

Embed Abjad code between open and close <abjad> </abjad> tags in your `.rst.raw` sourcefile and then call `abjad-book` to create a pure `.rst` file.

```
abjad-book foo.rst.raw foo.rst
```

```
Parsing file ...
Rendering "example-1.ly" ...
Rendering "example-2.ly" ...
```

You will need to build the HTML docs again to see your work.

```
make html
```

CHAPTER

THIRTYTHREE

TESTS

Abjad includes an extensive battery of tests. Abjad is in a state of rapid development and extension. Major refactoring efforts are common every six to eight months, and are likely to remain so at least 2012. And yet Abjad continues to allow the creation of complex pieces of fully notated score in the midst of these changes. We believe this is due to the extensive coverage provided by the automated regression battery described in the following sections. [1]

33.1 Automated regression?

A battery is any collection of tests. Regression tests differ from other types of test in that they are designed to be run again and again during many different stages of the development process. Regression tests help ensure that the system continues to function correctly as we make changes to it. An automated regression battery is one that can be run automatically by some sort of driver with minimal manual intervention.

Several different test drivers are now in use in the Python community. Of these, Abjad uses py.test. The `py.test` distribution is not included in the Python standard library, so one of the first thing new contributors to Abjad should do is download and install `py.test`, and then run the existing battery.

33.2 Running the battery

Change to the directory where you have Abjad installed. Then run `py.test`.

```
abjad$ py.test
============================== test process starts ===============================
executable:   /Library/Frameworks/Python.framework/Versions/2.5/Resources/Python
.app/Contents/MacOS/Python  (2.5.0-final-0)
using py lib: /Library/Frameworks/Python.framework/Versions/2.5/lib/python2.5/sit
e-packages/py <rev unknown>

accidental/test/test_accidental_compare.py[3] ...
accidental/test/test_accidental_eq.py[3] ...
accidental/test/test_accidental_init.py[2] ..
accidental/test/test_accidental_interface_grob_handling.py[2] ..
accidental/test/test_accidental_interface_style.py[2] ..

... (many lines omitted) ...

tuplet/test/test_tuplet_number_grob_handling.py[3] ...
update/test/test_update_interface.py[10] ..........
```

[1] Abjad r2371 includes 2165 tests.

```
voice/interface/test/test_voice_interface_explicit.py[2] ..
voice/interface/test/test_voice_interface_number.py[5] .....
voice/test/test_voice_len.py[2] ..

================ tests finished: 2165 passed in 232.53 seconds =================
```

33.3 Reading test output

`py.test` crawls the entire directory structure from which you call it, running tests in alphabetical order. `py.test` prints the total number of tests per file in square brackets and prints test results as a single `.` dot for success or else an `F` for failure.

33.4 Writing tests

Project check-in standards ask that tests accompany all code committed to the Abjad repository. If you add a new function, class or method to Abjad, you should add a new test file for that function, class or method. If you fix or extend an existing function, class or method, you should find the existing test file that covers that code and then either add a completely new test to the test file or else update an existing test already present in the test file.

33.5 Test files start with `test_`

When `py.test` first starts up it crawls the entire directory structure from which you call it prior to running a single test. As `py.test` executes this preflight work, it looks for any files beginning or ending with the string `test` and then collects and alphabetizes these. Only after making such a catalog of tests does `py.test` begin execution. This collect-and-cache behavior leads to the important point about naming, below.

33.6 Avoiding name conflicts

Note: The names of **test functions** must be absolutely unique across the entire directory structure on which you call `py.test`. You must never share names between test functions. For example, you must not have two tests named `test_grob_handling_01( )` **even if both tests live in different test files**. That is, a test named `test_grob_handling_01( )` living in the file `test_accidental_grob_handling.py` and a second test named `test_grob_handling_01( )` living in the file `test_notehead_grob_handling.py` will conflict with the each other when `py.test` runs. And, unfortunately, **‘‘py.test is silent about such conflicts when it runs**. That is, should you run `py.test` with the duplicate naming situation described here, what will happen is that `py.test` will correctly run and report results for the **first** such test it finds. However, when `py.test` encounters the second like-named test, `py.test` will incorrectly report cached results for the **first** test rather than the second. The take-away is to include some sort of namespacing indicators in every test name and not to be afraid of long test names. The `test_grob_handling_01( )` example given here fixes easily when the two tests rename to `test_accidental_grob_handling_01( )` and `test_notehead_grob_handling_01( )`.

CHAPTER THIRTYFOUR

SCRIPTS

The `abjad/scr/devel` directory contains scripts for Abjad developers. Add `abjad/scr/devel` to your `PATH` to use the scripts described below.

```
abjad$ ls scr/devel
abj-grep           abj-rmpycs         count-source-lines
abj-grp            abj-update         replace-in-files
```

34.1 Searching the Abjad codebase with `abj-grep`

Abjad provides a wrapper around UNIX `grep` in the form of `abj-grep`. Use this script to recursively search the entire Abjad codebase, leaving out non-human-readable files, files located in special `.svn` Subversion subdirectories, and all files in the `abjad/documentation` directories. You can run `abj-grep` from any directory on your system; you needn't be in the Abjad source directories when you call `abj-grep`.

```
$ abj-grep 'is_assignable('

leaf/duration.py:111:         if not durtools.is_assignable(rational):
tempo/indication.py:67:         assert durtools.is_assignable(arg)
tools/check/are_scalable.py:12:         if not durtools.is_assignable(candidate_duration):
tools/durtools/is_assignable.py:5:def is_assignable(duration):
tools/durtools/prolated_to_written.py:2:from abjad.tools.durtools.is_assignable import is_assignable
tools/durtools/prolated_to_written.py:15:   if is_assignable(prolated_duration):
tools/tietools/duration_change.py:28:   if durtools.is_assignable(new_written_duration):
tools/tuplettools/contents_scale.py:30:   if durtools.is_assignable(multiplier):
```

34.2 Removing old `*.pyc` files with `abj-rmpycs`

See the section on `abj-update` below for the reasons that it is a good idea to periodically remove the byte-compiled `*.pyc` files that Python generates for its own use behind the scenes. Abjad supplies `abj-rmpycs` to delete all the `*.pyc` in the Abjad codebase, leaving other `*.pyc` on your system untouched.

34.3 Updating your development copy of Abjad with `abj-update`

The normal way of updating your working copy of a Subversion repository is with the `svn update` or `svn up` command. You can update your working copy of Abjad in the usual way with `svn up`. But Abjad supplies an `abj-update` script as a wrapper around the usual Subversion update commands. In addition to updating your working copy of Abjad, `abj-update` populates the `abjad/.version` file with the most recent revision number of the system, and then removes all `*.pyc` files from your Abjad install. The benefits here are twofold. First, Abjad adds the most recent revision number of the system to all `.ly` files that you generate when working with Abjad. If you do not update the Abjad version file on a regular basis, the headers in your Abjad-generated `.ly` files will list the wrong version of the system. Second, as is the case in working with any substantial Python codebase, it is a good idea to periodically remove the byte-compiled `*.pyc` files that Python creates for its own use. The reason for this is inadvertant name aliasing. That is, if there was previously a module named `foo.py` somewhere in the system and if Python had at some point imported the module and created `foo.pyc` as a byprodct, this `.pyc` file will remain on the filesystem even if you later decide to remove, or rename, the source `foo.py` module. This lead to confusion because days or weeks after `foo.py` has been removed, Python will still find `foo.pyc` and seem to make the contents of `foo.py` available from beyond the grave. Updating with `abj-update` takes care of these two situations.

34.4 Counting lines of code with `count-source-lines`

Run `count-source-lines` for a count of lines of count divided between source and test files.

```
abjad$ count-source-lines

source_modules: 713
test_modules: 580

source_lines: 25899
test_lines: 46111

total lines: 72010
test-to-source ratio is 1.8 : 1
```

The script is directory-dependent so you can run it any the entire Abjad codebase or any subdirectory of the codebase.

34.5 Global search-and-replace with `replace-in-files`

You probably won't need to use `replace-in-files` very often. But if you are making changes to Abjad that will cause some name, such as `FooBar`, to be globally changed everywhere in the Abjad codebase to, say to `foo_bar`, then you can use `replace-in-files` to save lots of time.

```
$ replace-in-files --help

     Usage:

     replace-in-files DIR OLD_TEXT NEW_TEXT [CONFIRM=true/false]

     Crawl directory DIR and read every file in it recursively.
     Replace OLD_TEXT with NEW_TEXT in each file.

     Set CONFIRM to `false` to replace without prompting.
```

34.6 Adding new development scripts

If you write and then find yourself using a certain script over and over again when you're developing new code for Abjad, consider contributing back to the project so we can include your script in the next public release of Abjad. Scripts in the the Abjad script directories end with no file extension and try to be as OS-portable as possible, which usually means writing the script in Python, rather than your operating system's shell, and relying heavily on Python's `os` module.

CHAPTER

THIRTYFIVE

TIMING CODE

You can time code with Python's built-in `timeit` module:

```
from abjad import *
import timetime

timer = timeit.Timer('Note(0, (1, 4))', 'from __main__ import Note')
print timer.timeit(1000)
```

```
3.97960996628
```

These results show that 1000 notes take 4 seconds to create.

Other Python timing modules are available for download on the public Internet.

CHAPTER

THIRTYSIX

PROFILING CODE

Profile code with `profile_expr( )` in the `iotools` package:

```
abjad> iotools.profile_expr('Note(0, (1, 4))')

Sat Aug 14 13:03:16 2010    _tmp_abj_profile

         2214 function calls (2187 primitive calls) in 0.010 CPU seconds

   Ordered by: cumulative time
   List reduced from 157 to 12 due to restriction <12>

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.010    0.010 <string>:1(<module>)
        1    0.000    0.000    0.010    0.010 Note.py:9(__init__)
        1    0.000    0.000    0.010    0.010 _NoteInitializer.py:8(__init__)
        1    0.000    0.000    0.009    0.009 _Leaf.py:19(__init__)
        3    0.000    0.000    0.008    0.003 _Component.py:80(__init__)
        1    0.000    0.000    0.007    0.007 GraceInterface.py:6(__init__)
        2    0.000    0.000    0.007    0.003 Grace.py:8(__init__)
        2    0.000    0.000    0.006    0.003 Container.py:12(__init__)
        3    0.003    0.001    0.003    0.001 MeterInterface.py:16(__init__)
       79    0.000    0.000    0.002    0.000 _GrobHandler.py:13(__init__)
  412/393    0.001    0.000    0.002    0.000 _GrobHandler.py:27(__setattr__)
       90    0.000    0.000    0.001    0.000 _FormatContributor.py:6(__init__)
```

These results show 2214 function calls to create a note.

The `profile_expr( )` function wraps the Python `cProfile` and `pstats` modules.

CHAPTER THIRTYSEVEN

MEMORY CONSUMPTION

You can examine memory consumption with tools included in the `guppy` module:

```
from guppy import hpy
hp = hpy( )
hp.setrelheap( )
notes = [Note(0, (1, 4)) for x in range(1000)]
h = hp.heap( )
print h
```

```
Partition of a set of 544106 objects. Total size = 62090200 bytes.
 Index  Count   %     Size   % Cumulative  % Kind (class / dict of class)
     0  79000  15 11060000  18   11060000  18 dict (no owner)
     1   2000   0  3352000   5   14412000  23 dict of abjad.components.Grace.Grace.Grace
     2  49001   9  2376132   4   16788132  27 list
     3   1000   0  1676000   3   18464132  30 dict of abjad.components.Note.Note.Note
     4  51004   9  1644200   3   20108332  32 tuple
     5   3000   1  1572000   3   21680332  35 dict of
                                              abjad.interfaces.BeamInterface.BeamInterface.BeamInterf
                                              ace
     6   3000   1  1572000   3   23252332  37 dict of
                                              abjad.interfaces.BreaksInterface.BreaksInterface.Breaks
                                              Interface
     7   3000   1  1572000   3   24824332  40 dict of
                                              abjad.interfaces.ClefInterface.ClefInterface.ClefInterf
                                              ace
     8   3000   1  1572000   3   26396332  43 dict of
                                              abjad.interfaces.DirectivesInterface.DirectivesInterfac
                                              e.DirectivesInterface
     9   3000   1  1572000   3   27968332  45 dict of
                                              abjad.interfaces.InstrumentInterface.InstrumentInterfac
                                              e.InstrumentInterface
<138 more rows. Type e.g. '_.more' to view.>
```

These results show 62.1M for 1000 notes.

You must download `guppy` from the public Internet because the module is not included in the Python standard library.

CHAPTER

THIRTYEIGHT

CLASS ATTRIBUTES

Consider the definition of this class:

```
class FooWithInstanceAttribute(object):

   def __init__(self):
      self.constants = (
         'red', 'orange', 'yellow', 'green',
         'blue', 'indigo', 'violet',
         )
```

1000 objects consume 176k:

```
from guppy import hpy
hp = hpy( )
hp.setrelheap( )
objects = [FooWithInstanceAttribute( ) for x in range(1000)]
h = hp.heap( )
print h

Partition of a set of 2004 objects. Total size = 176536 bytes.
 Index  Count   %     Size   % Cumulative  % Kind (class / dict of class)
     0   1000  50   140000  79     140000  79 dict of __main__.FooWithInstanceAttribute
     1   1000  50    32000  18     172000  97 __main__.FooWithInstanceAttribute
     2      1   0     4132   2     176132 100 list
     3      1   0      348   0     176480 100 types.FrameType
     4      1   0       44   0     176524 100 __builtin__.weakref
     5      1   0       12   0     176536 100 int
```

But consider the definition of this class:

```
class FooWithSharedClassAttribute(object):

   def __init__(self):
      pass

   self.constants = (
      'red', 'orange', 'yellow', 'green',
      'blue', 'indigo', 'violet',
      )
```

1000 objects consume only 36k:

```
from guppy import hpy
hp = hpy( )
hp.setrelheap( )
```

```
objects = [FooWithClassAttribute( ) for x in range(1000)]
h = hp.heap( )
print h

Partition of a set of 1004 objects. Total size = 36536 bytes.
 Index  Count   %     Size   % Cumulative  % Kind (class / dict of class)
     0   1000 100    32000  88     32000  88 __main__.FooWithClassAttribute
     1      1   0     4132  11     36132  99 list
     2      1   0      348   1     36480 100 types.FrameType
     3      1   0       44   0     36524 100 __builtin__.weakref
     4      1   0       12   0     36536 100 int
```

Objects that share class attributes between them can consume less memory than objects that don't. But consider the usual provisions between class attributes and instance attributes when implementing custom classes. Class attributes make sense when objects will never modify the attribute in question. Class attributes also make sense when objects will modify the attribute in question and will desire to change the attribute in question for all other like objects at the same time. Probably best to use instance attributes in most other cases.

CHAPTER

THIRTYNINE

SLOTS

Consider the definition of this class:

```
class Foo(object)

   def __init__(self, a, b, c):
      self.a = a
      self.b = b
      self.c = c
```

1000 objects consume 176k:

```
from guppy import hpy
hp = hpy( )
hp.setrelheap( )
objects = [Foo(1, 2, 3) for x in range(1000)]
h = hp.heap( )
print h

Partition of a set of 2004 objects. Total size = 176536 bytes.
 Index  Count   %     Size   % Cumulative  % Kind (class / dict of class)
     0   1000  50   140000  79    140000  79 dict of __main__.FooWithInstanceAttribute
     1   1000  50    32000  18    172000  97 __main__.FooWithInstanceAttribute
     2      1   0     4132   2    176132 100 list
     3      1   0      348   0    176480 100 types.FrameType
     4      1   0       44   0    176524 100 __builtin__.weakref
     5      1   0       12   0    176536 100 int
```

But consider the definition of this class:

```
class FooWithSlots(object):

   __slots__ = ('a', 'b', 'c')
   def __init__(self, a, b, c):
      self.a = a
      self.b = b
      self.c = c
```

1000 objects consume only 40k:

```
from guppy import hpy
hp = hpy( )
hp.setrelheap( )
objects = [FooWithSlots(1, 2, 3) for x in range(1000)]
h = hp.heap( )
print h
```

```
Partition of a set of 1004 objects. Total size = 40536 bytes.
 Index  Count   %     Size   % Cumulative  % Kind (class / dict of class)
     0   1000 100    36000  89     36000  89 __main__.Bar
     1      1   0     4132  10     40132  99 list
     2      1   0      348   1     40480 100 types.FrameType
     3      1   0       44   0     40524 100 __builtin__.weakref
     4      1   0       12   0     40536 100 int
```

The example here confirms the Python Reference Manual 3.4.2.4: "By default, instances of both old and new-style classes have a dictionary for attribute storage. This wastes space for objects having very few instance variables. The space consumption can become acute when creating large numbers of instances."

CHAPTER

FORTY

TO-DO

Once you're comfortable making changes to Abjad, check out the open to-do items listed below:

Todo

this function should (but does not) copy marks that attach to *components* and to the immediate parent of the first component; extend function to do so.

(The *original entry* is located in chapters/api/tools/componenttools/clone_components_and_immediate_parent_of_first_component.rst, line 68.)

Todo

Add usage examples.

(The *original entry* is located in chapters/api/tools/componenttools/iterate_components_depth_first.rst, line 9.)

Todo

optimize to avoid behind-the-scenes full-score traversal.

(The *original entry* is located in chapters/api/tools/componenttools/iterate_timeline_backward_from_component.rst, line 38.)

Todo

optimize to avoid behind-the-scenes full-score traversal.

(The *original entry* is located in chapters/api/tools/componenttools/iterate_timeline_backward_in_expr.rst, line 40.)

Todo

optimize to avoid behind-the-scenes full-score traversal.

(The *original entry* is located in chapters/api/tools/componenttools/iterate_timeline_forward_from_component.rst, line 36.)

Todo

optimize to avoid behind-the-scenes full-score traversal.

(The *original entry* is located in chapters/api/tools/componenttools/iterate_timeline_forward_in_expr.rst, line 40.)

Todo

implement `componenttools.list_leftmost_components_with_prolated_duration_at_least()`.

(The *original entry* is located in chapters/api/tools/componenttools/list_leftmost_components_with_prolated_duration_at_most.rst, line 20.)

Todo

implement `componenttools.list_rightmost_components_with_prolated_duration_at_most()`.

(The *original entry* is located in chapters/api/tools/componenttools/list_leftmost_components_with_prolated_duration_at_most.rst, line 23.)

Todo

implement `componenttools.list_rightmost_components_with_prolated_duration_at_least()`.

(The *original entry* is located in chapters/api/tools/componenttools/list_leftmost_components_with_prolated_duration_at_most.rst, line 26.)

Todo

add `n = 1` keyword to generalize flipped distance.

(The *original entry* is located in chapters/api/tools/componenttools/move_component_subtree_to_right_in_immediate_parent_of_compo line 35.)

Todo

make `componenttools.move_component_subtree_to_right_in_immediate_parent_of_component()` work when spanners attach to children of component:

(The *original entry* is located in chapters/api/tools/componenttools/move_component_subtree_to_right_in_immediate_parent_of_compo line 37.)

Todo

regularize return value of function.

(The *original entry* is located in chapters/api/tools/componenttools/remove_component_subtree_from_score_and_spanners.rst, line 95.)

Todo

implement `measuretools.iterate_measures_forward_in_expr(expr, i = 0, j = None)` as a companion to this function.

(The *original entry* is located in chapters/api/tools/measuretools/get_nth_measure_in_expr.rst, line 48.)

Todo

implement `measuretools.change_nonbinary_measure_to_binary( )`.

(The *original entry* is located in chapters/api/tools/measuretools/scale_measure_denominator_and_adjust_measure_contents.rst, line 48.)

Todo

implement `measuretools.set_measure_denominator_and_adjust_contents( )`.

(The *original entry* is located in chapters/api/tools/measuretools/set_measure_denominator_and_adjust_numerator.rst, line 38.)

Todo

Implement an optional *wrap* keyword to specify whether this function should wrap around the ened of *sequence* whenever `len(sequence) < start + length` or not.

(The *original entry* is located in chapters/api/tools/seqtools/repeat_runs_in_sequence_to_count.rst, line 53.)

Todo

Reimplement this function to return a generator.

(The *original entry* is located in chapters/api/tools/seqtools/repeat_runs_in_sequence_to_count.rst, line 55.)

Todo

Return (immutable) tuple instead of (mutable) list.

(The *original entry* is located in chapters/api/tools/spannertools/Spanner/Spanner.rst, line 244.)

Todo

write `tietools.get_preprolated_tie_chain_duration( )` tests.

(The *original entry* is located in chapters/api/tools/tietools/get_preprolated_tie_chain_duration.rst, line 7.)

Todo

Write tietools.get_prolated_tie_chain_duration( ) tests.

(The *original entry* is located in chapters/api/tools/tietools/get_prolated_tie_chain_duration.rst, line 7.)

Todo

Write tietools.get_tie_chain_duration_in_seconds( ) tests.

(The *original entry* is located in chapters/api/tools/tietools/get_tie_chain_duration_in_seconds.rst, line 7.)

Todo

Implement `diatonic_interval_class_set_to_chord_quality_string( )`.

(The *original entry* is located in chapters/api/tools/tonalitytools/diatonic_interval_class_segment_to_chord_quality_string.rst, line 19.)

Todo

make work with nested tuplets.

(The *original entry* is located in chapters/api/tools/tuplettools/change_augmented_tuplets_in_expr_to_diminished.rst, line 16.)

Todo

make work with nested tuplets.

(The *original entry* is located in chapters/api/tools/tuplettools/change_diminished_tuplets_in_expr_to_augmented.rst, line 16.)

Todo

optimize without full-component traversal.

(The *original entry* is located in chapters/api/tools/verticalitytools/get_vertical_moment_at_prolated_offset_in_expr.rst, line 47.)

Todo

optimize without full-component traversal.

(The *original entry* is located in chapters/api/tools/verticalitytools/get_vertical_moment_starting_with_component.rst, line 51.)

Todo

optimize without multiple full-component traversal.

(The *original entry* is located in chapters/api/tools/verticalitytools/iterate_vertical_moments_backward_in_expr.rst, line 58.)

Todo

optimize without multiple full-component traversal.

(The *original entry* is located in chapters/api/tools/verticalitytools/iterate_vertical_moments_forward_in_expr.rst, line 58.)

Todo

Add release dates.

(The *original entry* is located in chapters/appendices/versions/index.rst, line 99.)

Appendices

CHAPTER

FORTYONE

FROM TREVOR AND VÍCTOR

We are composers Trevor Bača and Víctor Adán, creators of Abjad, and our earliest collaborative work dates back to shared undergraduate years in Austin. It was the mid- to late-90s and we found ourselves interested in ways of building up ever larger sets of musical materials in our scores, with ever greater amounts of musical information.

Our work then began with pitch formalization, creating materials in C and then writing the results as MIDI to hear what we'd created. Turns out that this is a fairly common gateway into materials generation for many composers, and so it was for us. Probably this was, and is, due to the ever present availability of MIDI and, to a lesser extent, CSound. But even back then it was clear to us to finding ways to embody other aspects of the musical score – from nested rhythms to the different approaches to the musical measure to the arbitrarily complex structures possible with overlapping musical voices – would require a wholly different level of consideration, and different development techniques as well.

As an example, consider flat lists of floating-point values. This basic data structure, together with the constant need some type of quantification or rounding, feeds much of most composers' work with CSound, pd and the like. It is a good thing, therefore, that essentially all modern programming languages include tools for manipulating flat lists of floats out of the box, or in the standard library. But what happens when you want to think of pitch as something much more than integers for core values with, perhaps, floats for microtones? What if you want to work with pitches as fully-fledged objects? Objects capable of carrying arbitrarily large sets of attributes and values? Objects that might group together, first into sets, and then into larger assemblages, and then into still larger complexes of pitch information loaded, or even overloaded, with cross-relationships or textural implications? Carrying this surplus of information about pitch, or the potential uses of pitch, in data structures limited to, or centered around, the list-of-floats paradigm then becomes a burden.

And what of working with rhythms not only as offset values, as implied by the list-of-floats approach, but as arbitrarily nested, stretched, compressed and stacked sets of values, as allowed by the tupleting and measure structures of conventional score? A different approach is needed.

There was, and still is, no reason to believe that general purpose programming languages and development tools should come readily supplied with the objects and methods most suitable for composerly applications. And this means that the attributes of a domain-specific language that will best meet the needs of composes interested in working formally with the full complement of capabilities in traditional score remains an open question.

We continued our work in score formalization independently until 2005, Trevor in a system that would come to be called Lascaux, and Víctor in a system dubbed Cuepatlahto. We experimented with C, Mathematica and Matlab as the core programming languages driving our systems before settling independently on Python, Víctor out of experiece at MIT, where he was working on his masters at the Media Lab with Berry Vercoe, and Trevor out of the working necessities of a professional developer and engineer.

We passed through indepedent experiences using Finale, Sibelius, Leland Smith's SCORE, and even Adobe Illustrator as the notational rendering engines for Lascaux and Cuepatlahto. Through all of this, both systems were designed to tackle a shared set of problems. These included:

1. The difficulty involved in transcribing larger scale and highly parameterized gestures and textures into traditional Western notation.

2. The general inflexbility of closed, commercial music notation software packages.
3. The relative inability of objects on the printed page in conventional score to point to each other — or, indeed, to other objects or ideas outside the printed page — in ways rich enough to help capture, model and develop long-range, nonlocal relationships throughout our scores.

Afer collaborating on a joint paper describing the two systems, and after discussing collaborative design and implementation at length, both online and in weekends' long review of our respective codebases, we decided to combine our efforts into a single, unified project. That project is now Abjad.

In our work on Abjad we strive to develop a powerful and flexible symbolic system. We picked the phrase 'formalized score control', or FSC, as a nod to Xenakis, who was so far ahead in so many ways, and also to highlight our primary project goal: to bring the full power of modern programming languages, and tools in mathematics, text processing, pattern recognition, and modular, iterative and incremental development to bear on all parts of the compositional process.

CHAPTER

FORTYTWO

WHY LILYPOND IS RIGHT FOR ABJAD

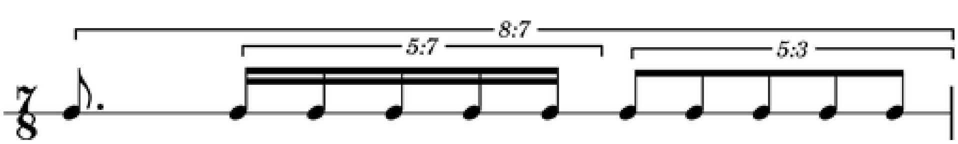
Early versions of Abjad wrote MIDI files for input to Finale and Sibelius. Later versions of Abjad wrote `.pbx` files for input into Leland Smith's SCORE. Over time we found LilyPond superior to Finale, Sibelius and SCORE.

42.1 Nested tuplets works out of the box

LilyPond uses a single construct to nest tuplets arbitrarily:

```
\new stafftools.RhythmicStaff {
   \time 7/8
   \times 7/8 {
      c8.
      \times 7/5 { c16 c16 c16 c16 c16 }
      \times 3/5 { c8 c8 c8 c8 c8 }
   }
}
```

```
abjad> staff = stafftools.RhythmicStaff([Measure((7, 8), [ ])])
abjad> measure = staff[0]
abjad> measure.append(Note('c8.'))
abjad> measure.append(Tuplet((7, 5), 5 * Note('c16')))
abjad> spannertools.BeamSpanner(measure[-1])
abjad> measure.append(Tuplet((3, 5), 5 * Note('c8')))
abjad> spannertools.BeamSpanner(measure[-1])
abjad> Tuplet((7, 8), measure.music)
abjad> staff.override.tuplet_bracket.bracket_visibility = True
abjad> staff.override.tuplet_bracket.padding = 1.6
abjad> show(staff)
```



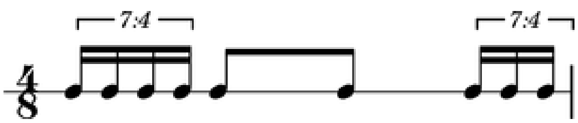
LilyPond's tuplet input syntax works the same as any other recursive construct.

42.2 Broken tuplets work out of the box

LilyPond engraves tupletted notes interrupted by nontupletted notes correctly:

```
\new Staff {
   \times 4/7 { c’16 c’16 c’16 c’16 }
   c’8 c’8
   \times 4/7 { c’16 c’16 c’16 }
}
```

```
abjad> t = Tuplet((4, 7), Note(0, (1, 16)) * 4)
abjad> notes = Note(0, (1, 8)) * 2
abjad> u = Tuplet((4, 7), Note(0, (1, 16)) * 3)
abjad> spannertools.BeamSpanner(t)
abjad> spannertools.BeamSpanner(notes)
abjad> spannertools.BeamSpanner(u)
abjad> measure = Measure((4, 8), [t] + notes + [u])
abjad> staff = stafftools.RhythmicStaff([measure])
abjad> show(staff)
```

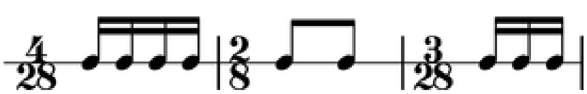


42.3 Nonbinary meters work out of the box

The rhythm above rewrites with time signatures in place of tuplets:

```
\new Staff {
   \time 4/28 c’16 c’16 c’16 c’16 |
   \time 2/8  c’8  c’8 |
   \time 3/28 c’16 c’16 c’16 |
}
```

```
abjad> t = Measure((4, 28), Note(0, (1, 16)) * 4)
abjad> u = Measure((2, 8), Note(0, (1, 8)) * 2)
abjad> v = Measure((3, 28), Note(0, (1, 16)) * 3)
abjad> spannertools.BeamSpanner(t)
abjad> spannertools.BeamSpanner(u)
abjad> spannertools.BeamSpanner(v)
abjad> staff = stafftools.RhythmicStaff([t, u, v])
abjad> show(staff)
```



The time signatures `4/28` and `3/28` here have a denominator not equal to `4`, `8`, `16` or any other nonnegative integer power of two. Abjad calls such time signatures **nonbinary meters** and LilyPond engraves them correctly.

42.4 Lilypond models the musical measure correctly

Most engraving packages make the concept of the measure out to be more important than it should. We see evidence of this wherever an engraving package makes it difficult for either a long note or the notes of a tuplet to cross a barline. These difficulties come from working the idea of measure-as-container deep into object model of the package.

There is a competing way to model the musical measure that we might call the measure-as-background way of thinking about things. Western notation pratice started absent any concept of the barline, introduced the idea gradually, and has since retreated from the necessity of the convention. Engraving packages that pick out an understanding of the barline from the 18th or 19th centuries subscribe to the measure-as-container view of things and oversimplify the problem. One result of this is to render certain barline-crossing rhythmic figures either an inelegant hack or an outright impossibility. LilyPond eschews the measure-as-container model in favor of the measure-as-background model better able to handle both earlier and later notation practice.

CHAPTER

FORTYTHREE

WHY PYTHON IS RIGHT FOR ABJAD

Abjad is an interactive software system designed to help composers build up complex pieces of music notation in an iterative and incremental way. Straightforward procedural syntax, the availability of basic functional constructs, and the interactive interpreter make Python the natural choice for Abjad.

CHAPTER

FORTYFOUR

WHY MIDI IS NOT ENOUGH

Given that Abjad models written musical score, it might seem odd for MIDI to be even mentioned in this manual. Yet, until fairly recently, MIDI has played a role (sometimes tangential, other times fundamental) in a variety of software tools related to music notation and engraving.

44.1 A very brief overview of midi

MIDI (Musical Instrument Digital Interface) was first introduced in 1981 by Dave Smith, the founder of Sequential Circuits. The original purpose of MIDI was to allow the communication between different electronic musical instruments; more specifically, to allow one device to send **control** data to another device. Typical messages might be "note On" (play a *note*) "note Off" (turn off a *note*). A MIDI "note" message, for example, is composed of three bytes: the first byte (the Status byte) tells the device what kind of message this is (e.g. a Note On message). The second byte encodes key number (which key was pressed) and the third byte, velocity (how hard the key was pressed). It should be clear that a *Note* in this context means something very different than *Note* in the context of a traditional printed score. While the bias towards keyboard interfaces is clear in the definition of the MIDI Note control message, one can still give the MIDI note a more general use by reinterpreting "key number" as pitch and "velocity" as loudness, the usual perceptual correlates of these control changes as well as the most meaningful musical parameters in western music.

With the subsequent proliferation of music production software, the SMF (Standard Midi File) was introduced to allow the recording and storage of the control data from a MIDI stream. The SMF required a time stamp to keep track of when control messages took place. These are called "delta-times" in the SMF specification.

"The MTrk chunk type is where actual song data is stored. It is simply a stream of MIDI events (and non-MIDI events), preceded by delta-time values."

In combination with the MIDI Note message, the addition of duration now allowed one to have a minimal but sufficient **machine** representation–a machine score–of music requiring only these parameters: duration, pitch and loudness. Such is the case of most piano music.

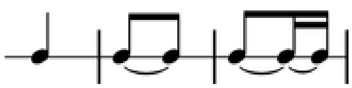
44.2 Limitations of midi from the point of view of score modeling

But, alas, there is much more information in a printed score that can not be practically encoded in a SMF. Common musical notions such as meter, clef, key signature, articulation, to name only a few, are ignored. A desire to include some of these concepts in MIDI is evident in the inclusion of some so called *meta-events*. From the SMF specification: " specifies non-MIDI information useful to this format or to sequencers." Examples of *meta-events* are *Time Signature* and *Key Signature*. In addition to the semantic elements just mentioned, there are also the typographical elements (such as line thickness, spacing, color, fonts, etc.) that all printed scores carry. This extra layer of information is completely absent in a SMF. However, from the point of view of encoding a printed score, the main limitation of MIDI is not the lack musical features or the absence of typographical data, but the assumption that musical durations, pitches and loudnesses can be each fully and efficiently encoded with integers or even fractions. In a printed score, this is not the case for any of them. MIDI encodes only *magnitudes*: time interval magnitudes, pitch interval magnitudes, velocity magnitudes. While these may be sufficient attributes for an automated piano performance, they are not all the attributes of notes in a printed score.

44.3 Written note durations vs. midi delta-times

Assume a fixed tempo has been set. Assume that all magnitudes are represented with (and limited to) rational numbers. A time interval magnitude d = 1/4 has an infinity of equivalent representations in terms of magnitude: d = 1/4 = 1/8 * 2 = 1/8 + 1/16 * 2 ... etc. So, for example, while equivalent in magnitude, these are not the same notated durations:

```
abjad> m1 = measuretools.AnonymousMeasure([Note(0, (1, 4))])
abjad> m2 = measuretools.AnonymousMeasure(Note(0, (1, 8)) * 2)
abjad> tietools.TieSpanner(m2)
abjad> m3 = measuretools.AnonymousMeasure([Note(0, (1, 8))] + Note(0, (1, 16)) * 2)
abjad> tietools.TieSpanner(m3)
abjad> r = stafftools.RhythmicStaff([m1, m2, m3])
abjad> iotools.write_expr_to_ly(r, 'example1')
```



44.4 Written note pitch vs. midi note-on

A similar thing happens with pitches. In MIDI, key (pitch) number 61 is a half tone above middle C. But how is this pitch to be notated? As a C sharp or a B flat?

```
abjad> m1 = measuretools.AnonymousMeasure([Note(1, (1, 4))])
abjad> m2 = measuretools.AnonymousMeasure([Note(('df', 4), (1, 4))])
abjad> r = Staff([m1, m2])
abjad> iotools.write_expr_to_ly(r, 'example2')
```

44.5 Conclusion

MIDI was not designed for score representation. MIDI is a simple communication protocol intended for real-time control. As such, it naturally lacks the adequate model to represent the full range of information found in printed scores.

CHAPTER

FORTYFIVE

CONFIGURATION

When first run, Abjad creates an `.abjad` directory in your own `$HOME` directory. In `$HOME/.abjad` you will find the Abjad configuration file: `config.py`. Here you can tell Abjad about your preferred PDF file viewer, MIDI player, your preferred LilyPond language, etc. All relevant variables have defaults that you can change to suit your needs. In Linux, for example, you might want to set your `pdfviewer` to `evince` and your `midiplayer` to `timidity`.

`config.py` is a regular Python file, so you should make sure the file follows Python syntax.

CHAPTER

FORTYSIX

RECALLING OUTPUT

46.1 Reopening Abjad PDFs

After you build a piece of notation and open with `show()` you will usually close the resulting PDF and continue working, changing your output notation in an iterative and incremental way.

```
abjad> staff = Staff(construct.scale(8))
abjad> show(staff)
```

But what if you need to go back and open the resulting PDF again? Abjad provides `pdf()` for precisely this purpose. Type the following at the Abjad prompt to open the most recent PDF written by Abjad.

```
abjad> pdf( )
```

If you want to open not the next-to-most recent PDF generated by Abjad, pass in a `-1`. And for the next-to-next-to-most recent, pass in a `-2`, and so on.

46.2 Looking at LilyPond output

Abjad generates a LilyPond `.ly` file for every Abjad expression that you build and `show()`. To look at these LilyPond `.ly` files that Abjad builds behind the scenes, use `ly()`.

```
abjad> ly( )

% Abjad revision 2362
% 2009-06-25 10:30

\version "2.12.2"
\include "english.ly"
\include "/Users/trevorbaca/Documents/abjad/trunk/abjad/scm/abjad.scm"

\new Staff {
   c'8
   d'8
   e'8
   f'8
   g'8
   a'8
   b'8
   c''8
}
```

Abjad opens the LilyPond `.ly` file in your favorite text editor.

These LilyPond `.ly` files that Abjad generates all have the same basic structure. The current version of Abjad and the date appear first, followed by the mandatory LilyPond version string and LilyPond directives for English note names and the default Abjad `.scm` file. The remainder of the file is reserved for the LilyPond input code corresponding to the expression you just built in Abjad.

When you are done looking at the LilyPond `.ly` file quit your text editor to return to the Abjad interpreter.

46.3 Looking at the LilyPond log

If things go wrong when you call `show()` or one of the other Abjad functions that call LilyPond behind the scenes, if may be helpful to examine the output that LilyPond writes to the LilyPond log.

```
abjad> log( )

GNU LilyPond 2.12.2
Processing `1420.ly'
Parsing...
Interpreting music...
Preprocessing graphical objects...
Finding the ideal number of pages...
Fitting music on 1 page...
Drawing systems...
Layout output to `1420.ps'...
Converting to `./1420.pdf'...
```

This is the normal output that LilyPond generates every time you call the program behind. When you are done looking at the LilyPond log, quit your text editor to return to the Abjad interpreter.

CHAPTER

FORTYSEVEN

WORKING WITH LILYPOND MULTIPLIERS

The LilyPond * operator allows the creation of duration multipliers against notes, rests, chords and skips.

You can assign LilyPond multipliers in Abjad:

```
abjad> note = Note(0, (1, 4))
abjad> note.duration.multiplier = Fraction(1, 6)
```

LilyPond multipliers change the multiplied duration of notes, rests, chords and skips:

```
abjad> note.duration.multiplied
Duration(1, 24)
```

LilyPond multipliers leave written duration unchanged:

```
abjad> note.duration.written
Duration(1, 4)
```

CHAPTER

FORTYEIGHT

LILYPOND EQUIVALENCIES IN ABJAD

48.1 Turning on proportional notation

Turn on proportional notation like this:

```
abjad> score = Score([ ])
abjad> score.set.proportional_notation_duration = schemetools.SchemeMoment(1, 24)
abjad> score.override.spacing_spanner.uniform_stretching = True
abjad> score.override.spacing_spanner.strict_note_spacing = True
```

To produce LilyPond input that looks like this:

```
abjad> f(score)
\new Score \with {
        \override SpacingSpanner #'strict-note-spacing = ##t
        \override SpacingSpanner #'uniform-stretching = ##t
        proportionalNotationDuration = #(ly:make-moment 1 24)
} <<
>>
```

CHAPTER

FORTYNINE

CODING STANDARDS

Indent with spaces, not with tabs. Use three spaces at a time:

```
def foo(x, y):
   return x + y
```

Introduce comments with two pound signs and a single space:

```
## comment before foo
def foo(x, y):
   return x + y
```

Favor early imports at the head of each module. Only one `import` per line:

```
from foo import x
from foo import y
from foo import z
```

Include two blank lines after `import` statements before the rest of the module:

```
from foo import x
from foo import y
from foo import z


class Foo(object):
   ...
   ...
```

Wrap docstrings with triple apostrophes and align like this:

```
def foo(x, y):
   '''This is the first line of the foo docstring.
   This is the second line of the foo docstring.
   And this is the last line of the foo docstring.'''
```

Use paired apostrophes to delimit strings:

```
s = 'foo'
```

Use paired quotation marks to delimit strings within a string:

```
s = 'foo and "bar"'
```

Name classes in upper camelcase:

```
def FooBar(object):
    ...
    ...
```

Name bound methods in underscore-delimited lowercase:

```
def Foo(object):

    def bar_blah(self):
        ...

    def bar_baz(self):
        ...
```

Name module-level functions in underscore-delimited lowercase:

```
def foo_bar( ):
    ...

def foo_blah( ):
    ...
```

Separate bound method definitions with a single empty line:

```
class FooBar(object):

    def __init__(self, x, y):
        ...

    def bar_blah(self):
        ...

    def bar_baz(self):
        ...
```

Organize the definitions of core classes into the five following major sections plus initialization:

```
class FooBar(object):

    def __init__(self, x, y):
        ...

    ## OVERLOADS ##

    def __repr__(self):
        ...

    def __str__(self):
        ...

    ## PRIVATE ATTRIBUTES ##

    @property
    def _foo(self):
        ...

    ## PUBLIC ATTRIBUTES ##

    @property
    def bar(self):
```

```
        ...

    ## PRIVATE METHODS ##

    def _blah(self, x, y):
        ...

    ## PUBLIC METHODS ##

    def baz(self, z):
        ...
```

Preceed private class attributes with a single underscore:

```
class FooBar(object):

    ## PRIVATE ATTRIBUTES ##

    @property
    def _foo(self):
        ...

    ## PRIVATE METHODS ##

    def _blah(self, x, y):
        ...
```

Include a single space in between empty parentheses:

```
def foo( ):
    ...
    ...
```

Use < less-than signs in preference to greater-than signs:

```
if x < y < z:
    ...
```

Limit lines to 80 characters and use \ to break lines where necessary.

Eliminate trivial slice indices. Use `s[:4]` instead of `s[0:4]`.

Do not abbreviate variable names.

Name variables that represent a list or other collection of objects in the plural.

Implement only one class per module.

Implement only one function per module.

Author one `py.test` test file for every module-level function.

Author one `py.test` test file for every bound method in the public interface of a class.

CHAPTER

FIFTY

WORKING WITH LISTS OF NUMBERS

Python provides a built-in `list` class that you can use to carry around almost anything. The examples here show how to create a list of numbers and then do things with the numbers in the list.

Create a list with square brackets.

```
abjad> my_list = [23, 7, 10, 18, 13, 20, 3, 2, 18, 9, 14, 3]
abjad> my_list
[23, 7, 10, 18, 13, 20, 3, 2, 18, 9, 14, 3]
```

Use `len( )` to find the number of elements in any list.

```
abjad> len(my_list)
12
```

Use `append( )` to add one element to a list.

```
abjad> my_list.append(5)
abjad> my_list
[23, 7, 10, 18, 13, 20, 3, 2, 18, 9, 14, 3, 5]
```

Use `extend( )` to extend one list with the contents of another.

```
abjad> my_other_list = [19, 11, 4, 10, 12]
abjad> my_list.extend(my_other_list)
abjad> my_list
[23, 7, 10, 18, 13, 20, 3, 2, 18, 9, 14, 3, 5, 19, 11, 4, 10, 12]
```

Use `reverse( )` to reverse the elements in a list.

```
abjad> my_list.reverse( )
abjad> my_list
[12, 10, 4, 11, 19, 5, 3, 14, 9, 18, 2, 3, 20, 13, 18, 10, 7, 23]
```

You can return a single value from a list with a numeric index.

```
abjad> my_list[0]
12
abjad> my_list[1]
10
abjad> my_list[2]
4
```

You can return many values from a list with slice notation.

```
abjad> my_list[:4]
[12, 10, 4, 11]
```

More information on these and all other operations defined on the built-in Python `list` is available in the Python tutorial.

CHAPTER

FIFTYONE

PITCH CONVENTIONS

51.1 Accidental abbreviations

Abjad abbreviates accidentals according to the LilyPond `english.ly` module:

accidental name	abbreviation
quarter sharp	‘qs’
quarter flat	‘qf’
sharp	‘s’
flat	‘f’
three-quarters sharp	‘tqs’
three-quarters flat	‘tqf’
double sharp	‘ss’
double flat	‘ff’

51.2 Chromatic pitch numbers

Abjad numbers chromatic pitches by semitone with middle C set equal to 0:

The code to generate this table is as follows:

```
score, treble_staff, bass_staff = scoretools.make_empty_piano_score( )
duration = Fraction(1, 32)

treble = measuretools.AnonymousMeasure([ ])
bass = measuretools.AnonymousMeasure([ ])

treble_staff.append(treble)
bass_staff.append(bass)

pitches = range(-12, 12 + 1)

cfgtools.set_default_accidental_spelling('sharps')
```

```
for i in pitches:
   note = Note(i, duration)
   rest = Rest(duration)
   clef = pitchtools.suggest_clef_for_named_chromatic_pitches([note.pitch])
   if clef == contexttools.ClefMark('treble'):
      treble.append(note)
      bass.append(rest)
   else:
      treble.append(rest)
      bass.append(note)
   diatonic_pitch_number = str(note.pitch.numbered_chromatic_pitch)
   markuptools.Markup(diatonic_pitch_number, 'down')(bass[-1])

score.override.rest.transparent = True
score.override.stem.stencil = False

show(score, 'paris.ly')
```

51.3 Diatonic pitch numbers

Abjad numbers diatonic pitches by staff space with middle C set equal to 0:

The code to generate this table is as follows:

```
score, treble_staff, bass_staff = scoretools.make_empty_piano_score( )
duration = Fraction(1, 32)

treble = measuretools.AnonymousMeasure([ ])
bass = measuretools.AnonymousMeasure([ ])

treble_staff.append(treble)
bass_staff.append(bass)

pitches =[ ]
diatonic_pitches = [0, 2, 4, 5, 7, 9, 11]

pitches.extend([-24 + x for x in diatonic_pitches])
pitches.extend([-12 + x for x in diatonic_pitches])
pitches.extend([0 + x for x in diatonic_pitches])
pitches.extend([12 + x for x in diatonic_pitches])
pitches.append(24)
cfgtools.set_default_accidental_spelling('sharps')

for i in pitches:
   note = Note(i, duration)
   rest = Rest(duration)
   clef = pitchtools.suggest_clef_for_named_chromatic_pitches([note.pitch])
   if clef == contexttools.ClefMark('treble'):
```

```
        treble.append(note)
        bass.append(rest)
    else:
        treble.append(rest)
        bass.append(note)
    diatonic_pitch_number = abs(note.pitch.numbered_diatonic_pitch)
    markuptools.Markup(diatonic_pitch_number, 'down')(bass[-1])

score.override.rest.transparent = True
score.override.stem.stencil = False

show(score, 'paris.ly')
```

51.4 Octave designation

Abjad designates octaves with both numbers and ticks:

Octave notation	Tick notation
C7	c’‘’‘
C6	c’‘’
C5	c’‘
C4	c’
C3	c
C2	c,
C1	c,,

51.5 Accidental spelling

Abjad chooses between enharmonic spellings at pitch-initialization according to the following table:

Chromatic pitch-class number	Chromatic pitch-class name (default)
0	C
1	C#
2	D
3	Eb
4	E
5	F
6	F#
7	G
8	Gb
9	A
10	Bb
11	B

```
abjad> staff = Staff([Note(n, (1, 8)) for n in range(12)])
abjad> show(staff)
```

Use pitch tools to respell with sharps:

```
abjad> pitchtools.respell_named_chromatic_pitches_in_expr_with_sharps(staff)
abjad> show(staff)
```

Or flats:

```
abjad> pitchtools.respell_named_chromatic_pitches_in_expr_with_flats(staff)
abjad> show(staff)
```

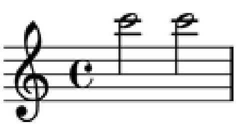
CHAPTER FIFTYTWO

SETTING PITCH DEVIATION

Use deviation to model the fact that two pitches differ by a fraction of a semitone:

```
abjad> note_1 = Note(24, (1, 2))
abjad> note_2 = Note(24, (1, 2))
abjad> staff = Staff([note_1, note_2])
```

```
abjad> show(staff)
```



```
abjad> note_2.pitch = pitchtools.NamedChromaticPitch(24, deviation = -31)
```

The pitch of the the first note is greater than the pitch of the second:

```
abjad> note_1.pitch > note_2.pitch
True
```

Use markup to include indications of pitch deviation in your score:

```
abjad> markuptools.Markup(note_2.pitch.deviation_in_cents, 'up')(note_2)
```

CHAPTER

FIFTYTHREE

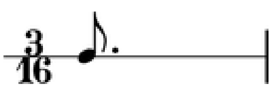
DURATION CONVENTIONS

53.1 Introduction

Abjad publishes information about many durated score objects.

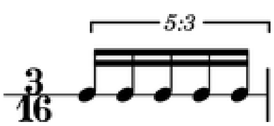
Notes, rests, chords and skips carry some duration attributes:

```
abjad> note = Note(0, (3, 16))
abjad> measure = Measure((3, 16), [note])
abjad> staff = stafftools.RhythmicStaff([measure])
abjad> note.duration.written
Duration(3, 16)
```



Tuplets, measures, voices, staves and the other containers carry duration attributes, too:

```
abjad> tuplet = tuplettools.FixedDurationTuplet((3, 16), Note(0, (1, 16)) * 5)
abjad> measure = Measure((3, 16), [tuplet])
abjad> staff = stafftools.RhythmicStaff([measure])
abjad> tuplet.duration.multiplier
Duration(3, 5)
```



The next chapters document core duration concepts in Abjad.

53.2 Assignability

Western notation readily admits rational values like `1/4`. But values like `1/5` notate only with tuplet brackets or special time signatures. Abjad formalizes the difference between rationals like `1/4` and `1/5` in the definition of rational assignability.

Rational values `n/d` are assignable when and only when numerator `n` is of the form `k(2**u-j)` and denominator `d` is of the form `2**v`. In this definition `u` and `v` must be nonnegative integers, `k` must be a positive integer, and `j` must be either `0` or `1`.

Abjad initializes notes, rests and chords with assignable durations only.

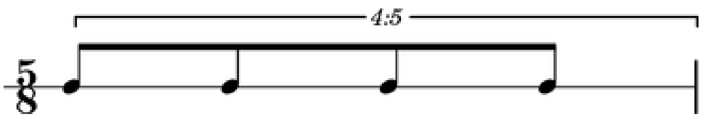
53.3 Prolation

Abjad uses **prolation** as a cover term for rhythmic augmentation and diminution. Augmentation increases the duration of notes, rests and chords. Diminution does the opposite. Western notation employs tuplet brackets and special types of time signature to effect prolation.

53.3.1 Tuplet prolation

Tuplets prolate their contents:

```
abjad> tuplet = Tuplet((5, 4), 4 * Note("c'8"))
abjad> staff = stafftools.RhythmicStaff([Measure((5, 8), [tuplet])])
abjad> spannertools.BeamSpanner(tuplet)
abjad> show(staff)
```



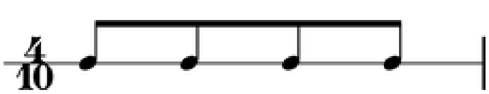

```
abjad> note = tuplet[0]
abjad> note.duration.written
Duration(1, 8)

abjad> note.duration.prolation
Fraction(5, 4)

abjad> note.duration.prolated
Duration(5, 32)
```

Notes here with written duration `1/8` carry prolation factor `5/4` and prolated duration `5/32`.

53.3.2 Meter prolation

Time signatures in western notation usually carry a denominator equal to a nonnegative integer power of `2`. Abjad calls these conventional meters **binary meters**. Denominators equal to integers other than integer powers of `2` are also possible. Such **nonbinary meters** rhythmically diminish the contents of the measures they govern:

```
abjad> measure = Measure((4, 10), Note(0, (1, 8)) * 4)
abjad> spannertools.BeamSpanner(measure)
abjad> staff = stafftools.RhythmicStaff([measure])
```



```
abjad> note = staff.leaves[0]
abjad> note.duration.prolation
Fraction(4, 5)

abjad> note.duration.prolated
Duration(1, 8)

abjad> note.duration.prolation
Fraction(4, 5)
```

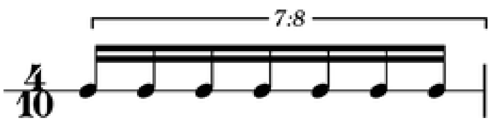
```
abjad> note.duration.prolated
Duration(1, 10)
```

Notes here with written duration `1/8` carry prolation factor `4/5` and prolated duration `1/10`.

53.3.3 The prolation chain

Tuplets nest and combine freely with different types of meter. When two or more **prolation donors** conspire, the prolation factor they collectively bestow on leaf-level music equals the cumulative product of all prolation factors in the **prolation chain**. All durated components carry a prolation chain:

```
abjad> tuplet = tuplettools.FixedDurationTuplet((4, 8), Note(0, (1, 16)) * 7)
abjad> spannertools.BeamSpanner(tuplet)
abjad> measure = Measure((4, 10), [tuplet])
abjad> staff = stafftools.RhythmicStaff([measure])
```




```
abjad> measure.duration.multiplier
Fraction(4, 5)

abjad> note = measure.leaves[0]
abjad> note.duration.prolation
Duration(32, 35)

abjad> note.duration.prolated
Duration(2, 35)
```

Notes here with written duration `1/16` carry prolated duration `2/35`.

Note: Western notation does not recognize tuplet brackets carrying one-to-one ratios. Such **trivial tuplets** may, however, be useful during different stages of composition, and Abjad allows them for that reason. Trivial tuplets carry **zero prolation**. Zero-prolated tuplets neither augment nor diminish the music they contain.

Note: Abjad implements one of two competing nonbinary **meter-interpretation schemes**. The first, **implicit meter-interpretation** given here, follows, for example, Ferneyhough, in that nonbinary meters prolate the contents of the measures they govern implicitly, ie, without recourse to tuplet brackets. The second, **explicit meter-interpretation**, which we find in, for example, Sciarrino, insists instead on the presence of some tuplet bracket, usually engraved in some broken or incomplete way. The implicit meter-interpretation that Abjad implements differs from the explicit meter-interpretation native to LilyPond. Abjad will eventually implement both implicit and explicit meter-interpretation, settable on a container-by-container basis.

Note: Nonbinary meter `n/d` rhythmically diminishes the contents of the measure it governs by a factor `j/k`, with `k=d`, and with `j` equal to the greatest integer power of `2` less than `d`. That is, `j=2**int(log2(d))`.

53.4 Duration types

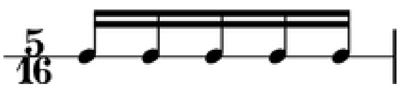
Abjad publishes duration information about all score components.

53.4.1 Written duration

Abjad uses **written duration** to refer to the face value of notes, rests and chords prior to prolation. Abjad written duration corresponds to the informal names most frequently used when talking about note duration.

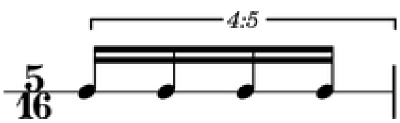
These sixteenth notes are worth a sixteenth of a whole note:

```
abjad> measure = Measure((5, 16), Note(0, (1, 16)) * 5)
abjad> spannertools.BeamSpanner(measure)
abjad> staff = stafftools.RhythmicStaff([measure])
abjad> note = measure[0]
abjad> note.duration.written
Duration(1, 16)
```



These sixteenth notes are worth more than a sixteenth of a whole note:

```
abjad> tuplet = tuplettools.FixedDurationTuplet((5, 16), Note(0, (1, 16)) * 4)
abjad> spannertools.BeamSpanner(tuplet)
abjad> measure = Measure((5, 16), [tuplet])
abjad> staff = stafftools.RhythmicStaff([measure])
abjad> note = tuplet[0]
abjad> note.duration.written
Duration(1, 16)
```



The notes in these examples are 'sixteenth notes' that carry different prolated durations. Abjad written duration captures the fact that the note heads and flag counts of the two examples match.

Written duration is a user-assignable rational number. Users can assign and reassign the written duration of notes, rests and chords at initialization and at any time during the life of the note, rest or chord. Written durations must be assignable; see the chapter on *assignability* for details. Note that Abjad containers do not carry written duration.

53.4.2 Prolated duration

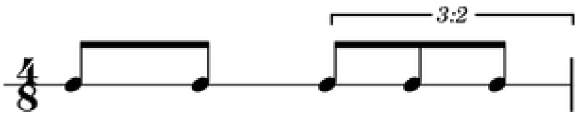
Prolation refers to the duration-scaling effects of tuplets and special types of time signature. Prolation is a way of thinking about the contribution that musical structure makes to the duration of score objects. All durated Abjad objects carry a prolated duration. Prolated duration is an emergent property of notes, tuplets and other durated objects. The prolated duration of notes, rests and chords equals the product of the written duration and prolation of those objects. The prolated duration of tuplets, measures and other containers equals the the container's duration interface multiplied by the container's prolation.

53.4.3 Contents duration

Abjad defines the **contents duration** of tuplets, measures, voices, staves and other containers equal to the sum of the **preprolated duration** of each of the elements in the container.

The measure here contains two eighth notes and tuplet. These elements carry preprolated durations equal to 1/8, 1/8 and 2/8, respectively:

```
abjad> notes = Note(0, (1, 8)) * 2
abjad> spannertools.BeamSpanner(notes)
abjad> tuplet = tuplettools.FixedDurationTuplet((2, 8), Note(0, (1, 8)) * 3)
abjad> spannertools.BeamSpanner(tuplet)
measure = Measure((4, 8), notes + [tuplet])
abjad> staff = stafftools.RhythmicStaff([measure])
abjad> measure.duration.contents
Duration(1, 2)
```



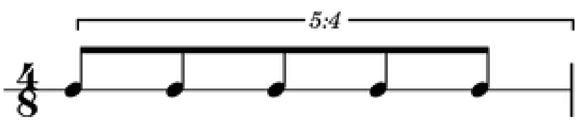
The contents duration of the measure here equals 1/8 + 1/8 + 2/8 = 4/8.

53.4.4 Target duration

Abjad defines the target duration of fixed-duration tuplets equal to composer-settable duration to which the tuplet prolates its contents.

This fixed-duration tuplet carries a target duration equal to 4/8:

```
abjad> tuplet = tuplettools.FixedDurationTuplet((4, 8), Note(0, (1, 8)) * 5)
abjad> spannertools.BeamSpanner(tuplet)
measure = Measure((4, 8), [tuplet])
abjad> staff = stafftools.RhythmicStaff([measure])
abjad> print tuplet.duration.contents
5/8
abjad> tuplet.duration.target
Duration(1, 2)
```



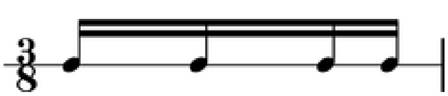
The tuplet contents sum to 5/8. But tuplet target duration always equals 4/8.

53.4.5 Multiplied duration

Abjad defines the multiplied duration of notes, rests and chords equal to the product of written duration and leaf multiplier.

The first two notes below carry leaf mulitipliers equal to 2/1:

```
abjad> notes = Note(0, (1, 16)) * 4
abjad> notes[0].duration.multiplier = Fraction(2, 1)
abjad> notes[1].duration.multiplier = Fraction(2, 1)
measure = Measure((3, 8), notes)
abjad> spannertools.BeamSpanner(measure)
abjad> staff = stafftools.RhythmicStaff([measure])
abjad> note = measure[0]
abjad> note.duration.written
Duration(1, 16)
```



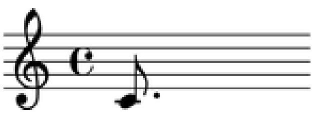
```
abjad> note.duration.multiplier
Fraction(2, 1)

abjad> note.duration.written * note.duration.multiplier
Duration(1, 8)
abjad> note.duration.multiplied
Duration(1, 8)
```

The written duration of these first two notes equals 1/16 and so the multiplied duration of these first two notes equals 1/16 * 2/1 = 1/8.

53.5 Duration initialization

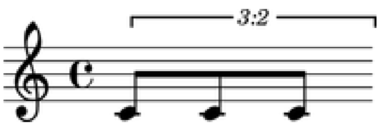
Durated Abjad classes initialize duration from arguments in the form *(n, d)* with numerator *n* and denominator *d*.

```
abjad> note = Note(0, (3, 16))
```



Durated classes include notes, rests, chords, skips, tuplets and measures.

```
abjad> tuplet = tuplettools.FixedDurationTuplet((2, 8), Note(0, (1, 8)) * 3)
abjad> spannertools.BeamSpanner(tuplet)
```

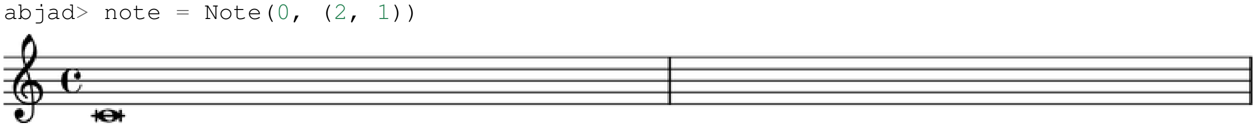


Abjad restricts notes, rests, chords and skips to durations like 3/16 that can be written with dots, beams and flags without ties or brackets. Abjad allows arbitrary positive durations like 5/8 for tuplets and measures.

```
abjad> tuplet = tuplettools.FixedDurationTuplet((5, 8), Note(0, (1, 8)) * 4)
abjad> spannertools.BeamSpanner(tuplet)
```

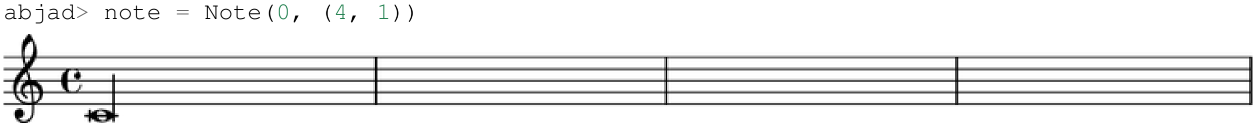
Abjad supports breves.

```
abjad> note = Note(0, (2, 1))
```



And longas.

```
abjad> note = Note(0, (4, 1))
```



Note: The restriction that the written durations of notes, rests, chords and skips be expressible with some combination of dots, flags and beams without recourse to ties and brackets generalizes to the condition of note_head assignability. Values *(n, d)* are note_head-assignable when and only when (1) d is a nonnegative integer power of 2; (2) n is either a nonnegative integer power of 2 or is a nonnegative integer power of 2, minus 1; and (3) n/d is less than or equal to 8. Condition (3) captures the fact that LilyPond provides no glyph with greater duration than the maxima (equal to eight whole notes).

Note: Integer forms like *4* as a substitute for *(4, 1)* in *Note(0, (4, 1))* are undocumented but allowed.

Note: Abjad allows maxima note_heads as in *Note(0, (8, 1))*. LilyPond implements a *maxima* command but does not supply a corresponding glyph for the note_head.

53.6 LilyPond multipliers

LilyPond provides an asterisk * operator to scale the durations of notes, rests and chords by arbitrarily positive rational values. LilyPond multipliers are inivisible and generate no typographic output of their own. However, while independent from the typographic output, LilyPond multipliers do factor in in calculations of duration and time.

Abjad implements LilyPond multpliers as the settable *duration.multiplier* attribute of notes, rests and chords.

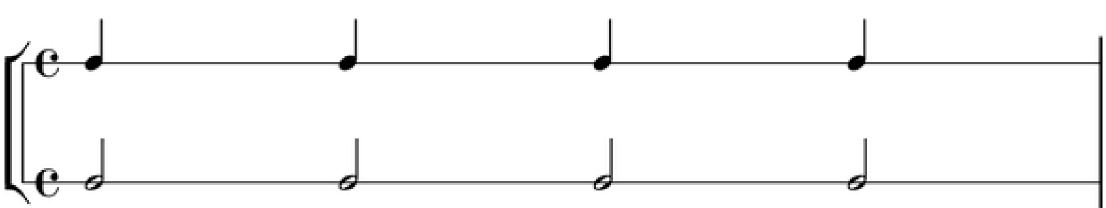
```
abjad> note = Note(0, (1, 4))
abjad> note.duration.multiplier = Fraction(1, 2)
abjad> note.duration.multiplier
Fraction(1, 2)

abjad> f(note)
c'4 * 1/2
```

Abjad also implements a *duration.multiplied* attribute to examine the duration of a note, rest or chord as affected by the multiplier.

```
abjad> note.duration.multiplied
Duration(1, 8)
```

LilyPond multipliers give the half notes here multiplied durations equal to a quarter note.

```
abjad> notes = Note(0, (1, 4)) * 4
abjad> multiplied_note = Note(0, (1, 2))
abjad> multiplied_note.duration.multiplier = Fraction(1, 2)
abjad> multiplied_notes = multiplied_note * 4
abjad> top = stafftools.RhythmicStaff(notes)
abjad> bottom = stafftools.RhythmicStaff(multiplied_notes)
abjad> staves = scoretools.StaffGroup([top, bottom])
```



Note: Abjad models multiplication fundamentally differently than prolation . See the chapter on *Prolation* for more information.

Note: The LilyPond multiplication * operator differs from the Abjad multiplication * operator. LilyPond multiplication scales duration of LilyPond notes, rests and chords. Abjad multiplication copies Abjad containers and leaves.

53.7 Duration interfaces compared

type	core	leaf	container	measure	tuplet	fd tuplet	fm tuplet
contents	–	–	R	R	R	R	R
multiplied	–	R	–	–	–	R	R
multiplier	–	RW	–	R	R	R	RW
preprolated	R	R	R	R	R	R	R
prolated	R	R	R	R	R	R	R
prolation	R	R	R	R	R	R	R
target	–	–	–	–	–	RW	–
written	–	RW	–	–	–	–	–

The table contains a total of only four settable duration attributes, divided among only three classes. Durated Abjad classes offer up many read-only duration attributes but very few read-write duration attributes.

All classes carry all three prolation-related attributes because all classes can nest inside containers. It is possible, for example, to nest an entire voice within a fixed-duration tuplet.

Note: Leaf multipliers and tuplet multipliers differ.

Note: `_MeasureDurationInterface` implements *nonbinary* attributes not shown above.

CHAPTER

FIFTYFOUR

TEMPLATE GALLERY

Abjad provides a number of score templates in the `abjad/templates` directory:

```
abjad> cfgtools.list_abjad_templates( )
('coventry.ly', 'lagos.ly', 'oedo.ly', 'paris.ly', 'tangiers.ly', 'thebes.ly', 'tirnaveni.ly')
```

Templates provide header, layout, paper and grob settings for different types of score.

54.1 Default LilyPond layout

```
abjad> import random
abjad> pitches = [random.randrange(0, 25) for x in range(32)]
abjad> staff_1 = Staff([ ])
abjad> staff_2 = Staff([ ])
abjad> score = Score([staff_1, staff_2])
abjad> staff_1.extend([Note(x, (1, 8)) for x in pitches[:16]])
abjad> staff_2.extend([Note(x, (1, 8)) for x in pitches[16:]])
abjad> show(score)
```

54.2 lagos.ly

```
abjad> pitches = [random.randrange(0, 25) for x in range(32)]
abjad> staff_1 = Staff([ ])
abjad> staff_2 = Staff([ ])
abjad> score = Score([staff_1, staff_2])
abjad> staff_1.extend([Note(x, (1, 8)) for x in pitches[:16]])
abjad> staff_2.extend([Note(x, (1, 8)) for x in pitches[16:]])
abjad> show(score, template = 'lagos')
```

54.3 oedo.ly

```
abjad> pitches = [random.randrange(0, 25) for x in range(32)]
abjad> staff_1 = Staff([ ])
abjad> staff_2 = Staff([ ])
abjad> score = Score([staff_1, staff_2])
abjad> staff_1.extend([Note(x, (1, 8)) for x in pitches[:16]])
abjad> staff_2.extend([Note(x, (1, 8)) for x in pitches[16:]])
abjad> show(score, template = 'oedo')
```

54.4 tangiers.ly

```
abjad> pitches = [random.randrange(0, 25) for x in range(32)]
abjad> staff_1 = Staff([ ])
abjad> staff_2 = Staff([ ])
abjad> score = Score([staff_1, staff_2])
abjad> staff_1.extend([Note(x, (1, 8)) for x in pitches[:16]])
abjad> staff_2.extend([Note(x, (1, 8)) for x in pitches[16:]])
abjad> show(score, template = 'tangiers')
```

54.5 `tirnaveni.ly`

```
abjad> pitches = [random.randrange(0, 25) for x in range(32)]
abjad> staff_1 = Staff([ ])
abjad> staff_2 = Staff([ ])
abjad> score = Score([staff_1, staff_2])
abjad> staff_1.extend([Note(x, (1, 8)) for x in pitches[:16]])
abjad> staff_2.extend([Note(x, (1, 8)) for x in pitches[16:]])
abjad> show(score, template = 'tirnaveni')
```

CHAPTER

FIFTYFIVE

TEXT ALIGNMENT

LilyPond provides many ways to position text.

55.1 Default alignment

LilyPond left-aligns markup relative to the left edge of note head by default.

```
abjad> notes = notetools.make_repeated_notes(1, Fraction(1, 4))
abjad> staff = stafftools.RhythmicStaff(notes)
abjad> leaves = staff.leaves
abjad> markuptools.Markup('XX', 'up')(leaves[0])
abjad> show(staff, 'thebes')
```

55.2 TextScript #'self-alignment-X

Use `#'self-alignment-X` to left-, center- or right-align markup relative to the left edge of note head.

Note: changes to `#'self-alignment-X` do not change the fact that markup positioning is by default relative to the LEFT edge of note head.

```
abjad> notes = notetools.make_repeated_notes(3, Fraction(1, 4))
abjad> staff = stafftools.RhythmicStaff(notes)
abjad> leaves = staff.leaves
abjad> markuptools.Markup('XX', 'up')(leaves[0])
abjad> leaves[0].override.text_script.self_alignment_X = 'left'
abjad> markuptools.Markup('XX', 'up')(leaves[1])
abjad> leaves[1].override.text_script.self_alignment_X = 'center'
abjad> markuptools.Markup('XX', 'up')(leaves[2])
abjad> leaves[2].override.text_script.self_alignment_X = 'right'
abjad> show(staff, 'thebes')
```

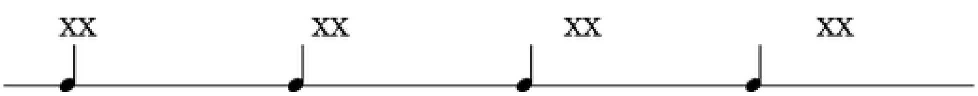
55.3 TextScript #'X-offset

Use #'X-offset to offset markup by some number of magic units in the horizontal direction.

Note: Specify #'X-offset arguments as numbers like #2.5. Do not specify #'X-offset arguments as direction contstants like #right.

Note: changes to #'X-offset do not change the fact that markup positioning is by default relative to the LEFT edge of note head.

```
abjad> notes = notetools.make_repeated_notes(4, Fraction(1, 4))
abjad> staff = stafftools.RhythmicStaff(notes)
abjad> leaves = staff.leaves
abjad> markuptools.Markup('XX', 'up')(leaves[0])
abjad> leaves[0].override.text_script.X_offset = 0
abjad> markuptools.Markup('XX', 'up')(leaves[1])
abjad> leaves[1].override.text_script.X_offset = 2
abjad> markuptools.Markup('XX', 'up')(leaves[2])
abjad> leaves[2].override.text_script.X_offset = 4
abjad> markuptools.Markup('XX', 'up')(leaves[3])
abjad> leaves[3].override.text_script.X_offset = 6
abjad> show(staff, 'thebes')
```



CHAPTER

FIFTYSIX

ABJAD-BOOK

`abjad-book` is an independent application included in every installation of Abjad. `abjad-book` allows you to write Abjad code in the middle of documents written in HTML, LaTeX or ReST. We created `abjad-book` to help us document Abjad. Our work on `abjad-book` was inspired by `lilypond-book`, which does for LilyPond much what `abjad-book` does for Abjad.

56.1 HTML with embedded Abjad

To see `abjad-book` in action, open a file and write some HTML by hand. Add some Abjad code to your HTML between open and close <abjad> </abjad> tags.

```
<html>

<p>This is an <b>HTML</b> document.</p>

<p>The code is standard hypertext mark-up.</p>

<p>Here is some music notation generated automatically by Abjad:</p>

<abjad>
v = Voice(construct.scale(8))
Beam(v)
write_ly(v, 'example-1') <hide
show(v)
</abjad>

<p>And here is more ordinary <b>HTML</b>.</p>

</html>
```

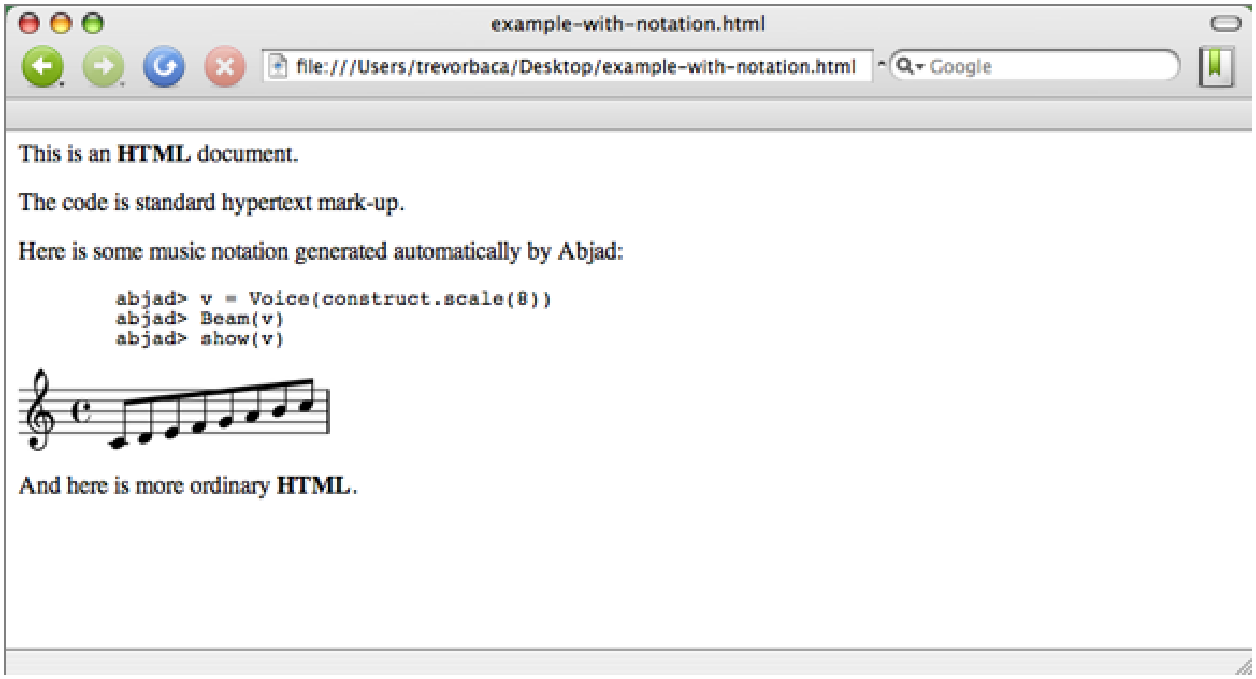
Save your the file with the name `example.html.raw`. You now have an HTML file with embedded Abjad code.

In the terminal, call `abjad-book` on `example.html.raw`.

```
$ abjad-book example.html.raw example.html

Parsing file...
Rendering "example-1.ly"...
```

The application opens `example.html.raw`, finds all Abjad code between <abjad> </abjad> tags, executes it, and then creates and inserts image files of music notation accordingly.

Open `example.html` with your browser.



That's all there is to it. `abjad-book` lets you open a file and type HTML by hand with Abjad sandwiched between the special <abjad> </abjad> tags described here. Run `abjad-book` on such a hybrid file to create pure HTML with images of music notation created by Abjad.

Note: `abjad-book` makes use of ImageMagick's convert application to crop and scale PNG images generated for HTML and ReST documents. For LaTeX documents, `abjad-book` uses `pdfcrop` for cropping PDFs.

56.2 LaTeX with embedded Abjad

You can use `abjad-book` to insert Abjad code and score excerpts into any LaTeX you create. Type the sample code below into a file.

```
\documentclass{article}
\usepackage{graphicx}
\usepackage{listings}
\begin{document}

This is a standard LaTeX document with embedded Abjad.

The code below creates an Abjad measure and then prints the measure
format string.

<abjad>
measure = RigidMeasure((5, 8), construct.scale(5))
print measure.format
</abjad>

This next bit of code knows about the measure we defined earlier.
This code renders the measure as a PDF using a template suitable
for inclusion in LaTeX documents.
```

```
<abjad>
write_ly(measure, 'example-1', 'oedo') <hide
</abjad>

And this is the end of the our sample LaTeX document.

\end{document}
```

Save your file with the name `example.tex.raw`. You now have a LaTeX file with embedded Abjad code.

In the terminal, call `abjad-book` on `example.tex.raw`.

```
$ abjad-book example.tex.raw example.tex

Processing 'example.tex.raw'. Will write output to 'example.tex'...
Parsing file...
Rendering "example-1.ly"...
```

The application open `example.tex.raw`, finds all code between Abjad tags, executes it, and then creates and inserts Abjad interpreter output and PDF files of music notation. You can view the contents of the next LaTeX file `abjad-book` has created.

```
\documentclass{article}
\usepackage{graphicx}
\usepackage{listings}
\begin{document}

This is a standard LaTeX document with embedded Abjad.

The code below creates an Abjad measure and then prints the measure
format string.

\begin{lstlisting}[basicstyle=\footnotesize, tabsize=4, showtabs=false, showspaces=false]
   abjad> measure = RigidMeasure((5, 8), construct.scale(5))
   abjad> print measure.format
   {
      \time 5/8
      c'8
      d'8
      e'8
      f'8
      g'8
   }
\end{lstlisting}

This next bit of code knows about the measure we defined earlier.
This code renders the measure as a PDF using a template suitable
for inclusion in LaTeX documents.

\includegraphics{images/example-1.pdf}

And this is the end of the our sample LaTeX document.

\end{document}
```

You can now process the file `example.tex` just like any other LaTeX file, using `pdflatex` or TexShop or whatever LaTeX compilation program you normally use on your computer.

```
$ pdflatex example.tex

This is pdfTeXk, Version 3.141592-1.40.3 (Web2C 7.5.6)
 %&-line parsing enabled.
entering extended mode
...
```

And then open the resulting PDF.

56.3 Using `abjad-book` on ReST documents

You can call `abjad-book` on ReST documents, too. Follow the examples given here for HTML and LaTeX documents and modify accordingly.

56.4 Using [hide = True]

You can add `[hide = True]` to any `abjad-book` example to show only music notation.

```
<abjad>[hide = True]
staff = Staff(construct.scale(8))
write_ly(staff, 'staff-example', 'oedo')
</abjad>
```

CHAPTER

FIFTYSEVEN

X11 COLOR NAMES

Abjad supports the X11 color names available in LilyPond

CHAPTER

FIFTYEIGHT

PARALLEL PROCESSING

Generating and acting upon score objects, especially large ones, can be very time consuming. However, you can speed up your score generation greatly if you can find ways to parallelize it!

Python provides a number of packages to handle parallel processing, using both threads and processes. Unfortunately, due to the Global Interpreter Lock (GIL), you won't see much performance improvement by multithreading your score generation. Luckily, the `multiprocessing` package gives us high level control over processes in a very similar manner to how one might manage threads.

`multiprocessing` provides a class, `Pool`, which acts as a pool of POSIX processes (just like the common thread-pool pattern). `Pool`, in turn, implements a parallelized `map` method, which works *basically* the same as Python's builtin `map` function. If you don't provide `Pool` with and arguments, it will create as many worker-processes as you have cores.

```
from multiprocessing import Pool
from abjad import *

def proc(notes_to_make):
   con = Container([])
   con.extend(leaftools.make_repeated_notes(notes_to_make))
   return con

def make(parallel = True):
   notes_per_fragment = range(1, 4)
   if parallel:
      pool = Pool()
      result = pool.map_async(proc, notes_per_fragment)
      pool.close() ## prevent the pool from accepting new work
      pool.join() ## wait for all child processes to return
      return result.get()
   else:
      return map(proc, notes_per_fragment)
```

```
abjad> make(parallel = True)
[{c'8}, {c'8, c'8}, {c'8, c'8, c'8}]

abjad> make(parallel = False)
[{c'8}, {c'8, c'8}, {c'8, c'8, c'8}]
```

A few words of caution about the above code fragment:

One, it's very useful to be able to turn the parallelization on and off, for debugging purposes, as errors encountered during processing may not appear (especially if one process fails, while another continues, and then the entire `map_async` simply hangs after the final process exits). Just as annoying, when errors do appear, the offending line in your code won't!

Two, do not use nested function definitions in your parallel procedure. The code above will fail if you redefine `proc` inside `make`. Similarly, if you pass a list of class instances to `map_async` which define another class inside themselves, it will also fail. This is a quirk of how `multiprocessing` passes information around.

Three, if you're computing very large fragments in parallel, expect a wait after your fragment generating procedures complete while the results are returned to the main python process. If the function never returns, then one of your processes failed, and you'll have to go find it.

CHAPTER

FIFTYNINE

GLOSSARY

assignability Attribute used of rational numbers that can be written as the duration of notes and rests without recourse to ties. The numbers `1/8` and `3/16` are assignable while the numbers `5/16` and `9/16` are not.

coverage The percentage of public classes, methods and functions currently documented in the system (doc coverage). Also the percentage of code exercised when the regression tests run (test coverage).

driver Used in reference to the testing process the term refers to the application chosen to execute a collection of tests before, during or after making changes to the system. Abjad uses py.test to execute the regression battery automatically.

grob LilyPond contraction of 'graphic object'. LilyPond grobs are either 'printing' or 'nonprinting'.

parentage The containment profile of any Abjad component. Consider a note contained within a tuplet contained within a staff. The 'improper' parentage of that note lists the note itself, the containing tuplet and the containing staff, all in that order. The 'proper' parentage of that note lists only the containing tuplet and the containing staff.

render To format an Abjad object as a PDF. Same as calling `show()`.

thread Time-sequential components within a voice. See the chapter on *Working with threads* for a detailed discussion.

CHAPTER

SIXTY

BIBLIOGRAPHY

CHAPTER

SIXTYONE

VERSION HISTORY

61.1 Abjad 1.1

61.1.1 Abjad 1.1.1

Abjad 1.1.1.tar.gz

- More complete and cleaner documentation!
- The configuration file `config` changed to pure python `config.py`. The file now supports more settings previously read as environment variables. All user setings are now found in this file. Users no longer need to set environment variables.
- Some new classes:
  - `_HistoryInterface`. Use the _HistoryInterface to apply attributes to any component in score that will be completely ignored by Abjad. Think of the _HistoryInterface as a private user namespace.
  - `_NoteColumnInterface` to handle the LilyPond NoteColumn grob.
  - `_SpanBarInterface`. See API for details.
  - `InvisibleStaff( )` staff.
  - `Moment` utility class to model the *Abjad* representation of the *LilyPond* moment.
- New Spanners:
  - `TempoProportional` spanner.
- More than a dozen new tools added.

61.1.2 Abjad 1.1.0

Abjad 1.1.tar.gz | Documentation

- Many structure transform tools added. See the *abjad.tools.** in the *Abjad API* package.
- Construction, transformation, manipulation and all other tools now grouped cleanly into packages.
- New `abjad-book` application available. Use `abjad-book` to interpret Abjad code blocks embedded in HTML, LaTex and reST documents.

61.2 Abjad 1.0.1055

Abjad 1.0.1055dev.tar.gz

Changes to the public interface:

- Abjad now models ties exclusively with the Tie spanner. The old `_TieInterface._set` attribute is now deprecated.
- You can no longer say `t.tie = True` or `t.tie = False`, for leaf `t`. You must structurally span `t` as `Tie(t)` instead.
- New public properties in `_SpannerReceptor`: `chain, parented, count`.
- New public helpers:
  - `construct.notes_curve()`
  - `durtools.rationalize()`
  - `iterate.tie_chains()`
  - `list_helpers()`
  - `mathtools.interpolate_divide()`
  - `measuretools.concentrate()`
  - `measuretools.scale_and_remeter()`
  - `measuretools.spin()`
  - `play()`
- Grace note `append()` and `extend()` no longer throw errors.

61.3 Abjad 1.0.1022

Abjad 1.0.1012dev.tar.gz

- First public release of Abjad.

Todo

Add release dates.

CHAPTER

SIXTYTWO

WHAT NEXT?

The most powerful features of Abjad are the set of interlocking objects that structure the system. Find out how Abjad models pitch, duration, leaves, containers and spanners in the chapters on Abjad fundamentals. These chapters explain how to work with the basic Abjad components.

Read some of the chapters concerning materials generation to figure out how to create starting materials.

And then read about structure traversal and manipulation to learn how to move around in large pieces of notation and change them while you go.

When you get stuck, check out the public interface in the *Abjad API*.

When you start to extend Abjad with custom code that you write for your own scores, read the chapters on developing with Abjad. These chapters describe how the codebase is laid out, how to add documentation and tests to the system, and how to contribute code that you write back to the public release of Abjad. We love contributions from composers working in many different ways. So get in touch and consider contributing to the project when the time feels right.

62.1 Get in touch!

Please join our two new mailing lists:

Questions or comments? Join the abjad-user list.

Want to contribute? Join the abjad-devel list.

CHAPTER

SIXTYTHREE

ABJAD API

63.1 Abjad API

63.1.1 Abjad score components

Chord

class `abjad.`**`Chord`**(**args*, ***kwargs*)

Bases: `abjad.components._Leaf._Leaf._Leaf`

Abjad model of a chord:

```
abjad> Chord([4, 13, 17], (1, 4))
Chord("<e' cs'' f''>4")
```

Return chord instance.

`append`(*note_head_token*)

Append *note_head_token* to chord:

```
abjad> chord = Chord([4, 13, 17], (1, 4))
abjad> chord
Chord("<e' cs'' f''>4")

abjad> chord.append(19)
abjad> chord
Chord("<e' cs'' f'' g''>4")
```

Sort chord note heads automatically after append and return none.

`extend`(*note_head_tokens*)

Extend chord with *note_head_tokens*:

```
abjad> chord = Chord([4, 13, 17], (1, 4))
abjad> chord
Chord("<e' cs'' f''>4")

abjad> chord.extend([2, 12, 18])
abjad> chord
Chord("<d' e' c'' cs'' f'' fs''>4")
```

Sort chord note heads automatically after extend and return none.

fingered_pitches

Read-only fingered pitches:

```
abjad> staff = Staff("<c''' e'''>4 <d''' fs'''>4")
abjad> glockenspiel = instrumenttools.Glockenspiel( )(staff)
abjad> instrumenttools.transpose_notes_and_chords_in_expr_from_sounding_pitch_to_fingered_pi

abjad> f(staff)
\new Staff {
   \set Staff.instrumentName = \markup { Glockenspiel }
   \set Staff.shortInstrumentName = \markup { Gkspl. }
   <c' e'>4
   <d' fs'>4
}

abjad> staff[0].fingered_pitches
(NamedChromaticPitch("c'"), NamedChromaticPitch("e'"))
```

Return tuple of named chromatic pitches.

note_heads

Get read-only tuple of note heads in chord:

```
abjad> chord = Chord([7, 12, 16], (1, 4))
abjad> chord.note_heads
(NoteHead("g'"), NoteHead("c''"), NoteHead("e''"))
```

Set chord note heads from any iterable:

```
abjad> chord = Chord([7, 12, 16], (1, 4))
abjad> chord.note_heads = [0, 2, 6]
abjad> chord
Chord("<c' d' fs'>4")
```

pitches

Get read-only tuple of pitches in chord:

```
abjad> chord = Chord([7, 12, 16], (1, 4))
abjad> chord.pitches
(NamedChromaticPitch("g'"), NamedChromaticPitch("c''"), NamedChromaticPitch("e''"))
```

Set chord pitches from any iterable:

```
abjad> chord = Chord([7, 12, 16], (1, 4))
abjad> chord.pitches = [0, 2, 6]
abjad> chord
Chord("<c' d' fs'>4")
```

pop (*i=-1*)

Remove note head at index *i* in chord:

```
abjad> chord = Chord([4, 13, 17], (1, 4))
abjad> chord
Chord("<e' cs'' f''>4")

abjad> chord.pop(1)
NoteHead("cs''")

abjad> chord
Chord("<e' f''>4")
```

Return note head.

remove (*note_head*)

Remove *note_head* from chord:

```
abjad> chord = Chord([4, 13, 17], (1, 4))
abjad> chord
Chord("<e' cs'' f''>4")

abjad> chord.remove(chord[1])
abjad> chord
Chord("<e' f''>4")
```

Return none.

sounding_pitches

Read-only sounding pitches:

```
abjad> staff = Staff("<c''' e'''>4 <d''' fs'''>4")
abjad> glockenspiel = instrumenttools.Glockenspiel( )(staff)
abjad> instrumenttools.transpose_notes_and_chords_in_expr_from_sounding_pitch_to_fingered_pi

abjad> f(staff)
\new Staff {
   \set Staff.instrumentName = \markup { Glockenspiel }
   \set Staff.shortInstrumentName = \markup { Gkspl. }
   <c' e'>4
   <d' fs'>4
}

abjad> staff[0].sounding_pitches
(NamedChromaticPitch("c'''"), NamedChromaticPitch("e'''"))
```

Return tuple of named chromatic pitches.

Container

class abjad.**Container** (*music=None*, ***kwargs*)

Bases: abjad.components._Component._Component._Component

Abjad model of a music container:

```
abjad> container = Container(macros.scale(4))
abjad> f(container)
{
   c'8
   d'8
   e'8
   f'8
}
```

Return container object.

append (*component*)

Append *component* to container:

```
abjad> container = Container("c'8 d'8 e'8")
abjad> beam = spannertools.BeamSpanner(container.music)
```

```
abjad> f(container)
{
   c'8 [
   d'8
   e'8 ]
}

abjad> container.append(Note("f'8"))

abjad> f(container)
{
   c'8 [
   d'8
   e'8 ]
   f'8
}
```

Return none.

extend (*expr*)

Extend *expr* against container:

```
abjad> container = Container("c'8 d'8 e'8")
abjad> beam = spannertools.BeamSpanner(container.music)

abjad> f(container)
{
   c'8 [
   d'8
   e'8 ]
}

abjad> container.extend([Note("cs'8"), Note("ds'8"), Note("es'8")])

abjad> f(container)
{
   c'8 [
   d'8
   e'8 ]
   cs'8
   ds'8
   es'8
}
```

Return none.

index (*component*)

Index *component* in container:

```
abjad> container = Container("c'8 d'8 e'8")

abjad> note = container[-1]
abjad> note
Note("e'8")

abjad> container.index(note)
2
```

Return nonnegative integer.

insert (*i*, *component*)

Insert *component* in container at index *i*:

```
abjad> container = Container("c'8 d'8 e'8")
abjad> beam = spannertools.BeamSpanner(container.music)

abjad> f(container)
{
   c'8 [
   d'8
   e'8 ]
}

abjad> container.insert(1, Note("cs'8"))

abjad> f(container)
{
   c'8 [
   cs'8
   d'8
   e'8 ]
}
```

Return none.

is_parallel

Get parallel container:

```
abjad> container = Container([Voice("c'8 d'8 e'8"), Voice('g4.')])

abjad> f(container)
{
   \new Voice {
      c'8
      d'8
      e'8
   }
   \new Voice {
      g4.
   }
}

abjad> container.is_parallel
False
```

Return boolean.

Set parallel container:

```
abjad> container.is_parallel = True

abjad> f(container)
<<
   \new Voice {
      c'8
      d'8
      e'8
   }
   \new Voice {
      g4.
```

```
    }
>>
```

Return none.

leaves
Read-only tuple of leaves in container:

```
abjad> container = Container("c'8 d'8 e'8")

abjad> container.leaves
(Note("c'8"), Note("d'8"), Note("e'8"))
```

Return tuple of zero or more leaves.

music
Read-only tuple of components in container:

```
abjad> container = Container("c'8 d'8 e'8")

abjad> container.music
(Note("c'8"), Note("d'8"), Note("e'8"))
```

Return tuple or zero or more components.

pop(*i=-1*)
Pop component at index *i* from container:

```
abjad> container = Container("c'8 d'8 e'8")
abjad> beam = spannertools.BeamSpanner(container.music)

abjad> f(container)
{
   c'8 [
   d'8
   e'8 ]
}

abjad> container.pop(-1)
Note("e'8")

abjad> f(container)
{
   c'8 [
   d'8 ]
}
```

Return component.

remove(*component*)
Remove *component* from container:

```
abjad> container = Container("c'8 d'8 e'8")
abjad> beam = spannertools.BeamSpanner(container.music)

abjad> f(container)
{
   c'8 [
   d'8
   e'8 ]
}
```

```
abjad> note = container[-1]
abjad> note
Note("e'8")

abjad> container.remove(note)

abjad> f(container)
{
   c'8 [
   d'8 ]
}
```

Return none.

Measure

class abjad.**Measure** (*meter*, *music=None*, ***kwargs*)

Bases: `abjad.components.Container.Container.Container`

Abjad model of a measure:

```
abjad> measure = Measure((4, 8), macros.scale(4))
abjad> f(measure)
{
   \time 4/8
   c'8
   d'8
   e'8
   f'8
}
```

Return measure object.

is_full

True if preprolated duration matches effective meter duration.

Note

class abjad.**Note** (**args*, ***kwargs*)

Bases: `abjad.components._Leaf._Leaf._Leaf`

Abjad model of a note:

```
abjad> Note(13, (3, 16))
Note("cs''8.")
```

fingered_pitch

Read-only fingered pitch of note:

```
abjad> staff = Staff("d''8 e''8 f''8 g''8")
abjad> piccolo = instrumenttools.Piccolo( )(staff)
abjad> instrumenttools.transpose_notes_and_chords_in_expr_from_sounding_pitch_to_fingered_pi

abjad> f(staff)
\new Staff {
   \set Staff.instrumentName = \markup { Piccolo }
   \set Staff.shortInstrumentName = \markup { Picc. }
```

```
    d'8
    e'8
    f'8
    g'8
}

abjad> staff[0].fingered_pitch
NamedChromaticPitch("d'")
```

Return named chromatic pitch.

note_head

Get note head of note:

```
abjad> note = Note(13, (3, 16))
abjad> note.note_head
NoteHead("cs''")
```

Set note head of note:

```
abjad> note = Note(13, (3, 16))
abjad> note.note_head = 14
abjad> note
Note("d''8.")
```

pitch

Get named pitch of note:

```
abjad> note = Note(13, (3, 16))
abjad> note.pitch
NamedChromaticPitch("cs''")
```

Set named pitch of note:

```
abjad> note = Note(13, (3, 16))
abjad> note.pitch = 14
abjad> note
Note("d''8.")
```

sounding_pitch

Read-only sounding pitch of note:

```
abjad> staff = Staff("d''8 e''8 f''8 g''8")
abjad> piccolo = instrumenttools.Piccolo( )(staff)

abjad> instrumenttools.transpose_notes_and_chords_in_expr_from_sounding_pitch_to_fingered_pi

abjad> f(staff)
\new Staff {
    \set Staff.instrumentName = \markup { Piccolo }
    \set Staff.shortInstrumentName = \markup { Picc. }
    d'8
    e'8
    f'8
    g'8
}
abjad> staff[0].sounding_pitch
NamedChromaticPitch("d''")
```

Return named chromatic pitch.

Rest

class `abjad.Rest` (**args*, ***kwargs*)

Bases: `abjad.components._Leaf._Leaf._Leaf`

Abjad model of a rest:

```
abjad> Rest((3, 16))
Rest('r8.')
```

Score

class `abjad.Score` (*music=None*, ***kwargs*)

Bases: `abjad.components._Context._Context._Context`

Abjad model of a score:

```
abjad> staff_1 = Staff(macros.scale(4))
abjad> staff_2 = Staff(macros.scale(4))
abjad> score = Score([staff_1, staff_2])
abjad> f(score)
\new Score <<
   \new Staff {
      c'8
      d'8
      e'8
      f'8
   }
   \new Staff {
      c'8
      d'8
      e'8
      f'8
   }
>>
```

Return score object.

Staff

class `abjad.Staff` (*music=None*, ***kwargs*)

Bases: `abjad.components._Context._Context._Context`

Abjad model of a staff:

```
abjad> staff = Staff(macros.scale(4))
abjad> f(staff)
\new Staff {
   c'8
   d'8
   e'8
   f'8
}
```

Return staff object.

Tuplet

class `abjad.Tuplet` (*multiplier*, *music=None*, ***kwargs*)

Bases: `abjad.components.Container.Container.Container`

Abjad model of a tuplet:

```
abjad> tuplet = Tuplet((2, 3), macros.scale(3))
abjad> f(tuplet)
\times 2/3 {
   c'8
   d'8
   e'8
}
```

Return tuplet object.

duration
Tuplet duration interface.

force_fraction
Read / write boolean to force `n:m` fraction.

is_invisible
Read / write boolean to render tuplet invisible.

is_trivial
True when tuplet multiplier is one, otherwise False.

ratio
Tuplet multiplier formatted with colon as ratio.

Voice

class `abjad.Voice` (*music=None*, ***kwargs*)

Bases: `abjad.components._Context._Context._Context`

Abjad model of a voice:

```
abjad> voice = Voice(macros.scale(4))
abjad> f(voice)
\new Voice {
   c'8
   d'8
   e'8
   f'8
}
```

Return voice object.

63.1.2 Abjad composition packages

cfgtools

cfgtools.get_abjad_revision_string

`abjad.tools.cfgtools.get_abjad_revision_string()`

New in version 1.1.2. Get Abjad revision string:

```
abjad> cfgtools.get_abjad_revision_string( ) # doctest: +SKIP
'4392'
```

Return string.

cfgtools.get_abjad_version_string

`abjad.tools.cfgtools.get_abjad_version_string()`
New in version 1.1.2. Get Abjad version string:

```
abjad> cfgtools.get_abjad_version_string( )
'1.1.2'
```

Return string.

cfgtools.get_lilypond_version_string

`abjad.tools.cfgtools.get_lilypond_version_string()`
New in version 1.1.2. Get LilyPond version string:

```
abjad> cfgtools.get_lilypond_version_string( ) # doctest: +SKIP
'2.13.61'
```

Return string.

cfgtools.get_python_version_string

`abjad.tools.cfgtools.get_python_version_string()`
New in version 1.1.2. Get Python version string:

```
abjad> cfgtools.get_python_version_string( )
'2.6.1'
```

Return string.

cfgtools.list_abjad_environment_variables

`abjad.tools.cfgtools.list_abjad_environment_variables()`
New in version 1.1.1. List Abjad environment variables.

Return tuple of zero or more environment variable / setting pairs.

Abjad environment variables are defined in `abjad/cfg/cfg.py`. Changed in version 1.1.2: renamed `cfgtools.list_settings( )` to `cfgtools.list_abjad_environment_variables( )`.

cfgtools.list_abjad_templates

`abjad.tools.cfgtools.list_abjad_templates()`
New in version 1.1.2. List Abjad templates:

```
abjad> cfgtools.list_abjad_templates( )
('coventry.ly', 'lagos.ly', 'oedo.ly', 'paris.ly', 'tangiers.ly', 'thebes.ly', 'tirnaveni.ly')
```

Return tuple of zero or more strings.

Abjad templates are housed in `abjad/templates`.

cfgtools.set_default_accidental_spelling

`abjad.tools.cfgtools.`**`set_default_accidental_spelling`**`(`*spelling='mixed'*`)`

New in version 1.1.1. Set default accidental spelling to sharps:

```
abjad> cfgtools.set_default_accidental_spelling('sharps')

abjad> [Note(13, (1, 4)), Note(15, (1, 4))]
[Note("cs''4"), Note("ds''4")]
```

Set default accidental spelling to flats:

```
abjad> cfgtools.set_default_accidental_spelling('flats')

abjad> [Note(13, (1, 4)), Note(15, (1, 4))]
[Note("df''4"), Note("ef''4")]
```

Set default accidental spelling to mixed:

```
abjad> cfgtools.set_default_accidental_spelling( )

abjad> [Note(13, (1, 4)), Note(15, (1, 4))]
[Note("cs''4"), Note("ef''4")]
```

Mixed is system default.

Mixed test case must appear last here for doc tests to check correctly.

Return none. Changed in version 1.1.2: renamed `pitchtools.change_default_accidental_spelling( )` to `cfgtools.set_default_accidental_spelling( )`.

chordtools

chordtools.Cluster

class `abjad.tools.chordtools.`**`Cluster`**`(`*music=None*, ***kwargs*`)`

Bases: `abjad.components.Container.Container.Container` New in version 1.1.1. Abjad model of a tone cluster container:

```
abjad> cluster = chordtools.Cluster("c'8 d'8 b'8")

abjad> cluster
Cluster(c'8, d'8, b'8)

abjad> f(cluster)
\makeClusters {
   c'8
   d'8
   b'8
}
```

Return cluster object.

chordtools.arpeggiate_chord

abjad.tools.chordtools.**arpeggiate_chord**(*chord*)

New in version 1.1.1. Arpeggiate *chord*:

```
abjad> chord = Chord("<c' d'' ef''>8")

abjad> chordtools.arpeggiate_chord(chord)
[Note("c'8"), Note("d''8"), Note("ef''8")]
```

Arpeggiated notes inherit *chord* written duration.

Arpeggiated notes do not inherit other *chord* attributes.

Return list of newly constructed notes. Changed in version 1.1.2: renamed `chordtools.arpeggiate( )` to `chordtools.arpeggiate_chord( )`.

chordtools.change_defective_chord_to_note_or_rest

abjad.tools.chordtools.**change_defective_chord_to_note_or_rest**(*chord*)

New in version 1.1.1. Change zero-length *chord* to rest:

```
abjad> chord = Chord([ ], (3, 16))

abjad> chord
Chord('<>8.')

abjad> chordtools.change_defective_chord_to_note_or_rest(chord)
Rest('r8.')
```

Change length-one chord to note:

```
abjad> chord = Chord("<cs''>8.")

abjad> chord
Chord("<cs''>8.")

abjad> chordtools.change_defective_chord_to_note_or_rest(chord)
Note("cs''8.")
```

Return chords with length greater than one unchanged:

```
abjad> chord = Chord("<c' c'' cs''>8.")

abjad> chord
Chord("<c' c'' cs''>8.")

abjad> chordtools.change_defective_chord_to_note_or_rest(chord)
Chord("<c' c'' cs''>8.")
```

Return notes unchanged:

```
abjad> note = Note("c'4")

abjad> note
Note("c'4")
```

```
abjad> chordtools.change_defective_chord_to_note_or_rest(note)
Note("c'4")
```

Return rests unchanged:

```
abjad> rest = Rest('r4')

abjad> rest
Rest('r4')

abjad> chordtools.change_defective_chord_to_note_or_rest(rest)
Rest('r4')
```

Return note, rest, chord or none. Changed in version 1.1.2: renamed `chordtools.cast_defective( )` to `chordtools.change_defective_chord_to_note_or_rest( )`.

chordtools.color_chord_note_heads_by_pitch_class_color_map

`abjad.tools.chordtools.`**`color_chord_note_heads_by_pitch_class_color_map`**`(`*chord*, *color_map*`)`

New in version 1.1.2. Color *chord* note heads by pitch-class *color_map*:

```
abjad> chord = Chord([12, 14, 18, 21, 23], (1, 4))

abjad> pitches = [[-12, -10, 4], [-2, 8, 11, 17], [19, 27, 30, 33, 37]]
abjad> colors = ['red', 'blue', 'green']
abjad> color_map = pitchtools.NumberedChromaticPitchClassColorMap(pitches, colors)

abjad> chordtools.color_chord_note_heads_by_pitch_class_color_map(chord, color_map)
Chord("<c'' d'' fs'' a'' b''>4")

abjad> f(chord)
<
        \tweak #'color #red
        c''
        \tweak #'color #red
        d''
        \tweak #'color #green
        fs''
        \tweak #'color #green
        a''
        \tweak #'color #blue
        b''
>4
```

Also works on notes:

```
abjad> note = Note(0, (1, 4))

abjad> chordtools.color_chord_note_heads_by_pitch_class_color_map(note, color_map)
Note("c'4")

abjad> f(note)
\once \override NoteHead #'color = #red
c'4
```

When *chord* is neither a chord nor note return *chord* unchanged:

```
abjad> staff = Staff([ ])

abjad> chordtools.color_chord_note_heads_by_pitch_class_color_map(staff, color_map)
Staff{ }
```

Return *chord*. Changed in version 1.1.2: renamed `chordtools.color_note_heads_by_pc( )` to `chordtools.color_chord_note_heads_by_pitch_class_color_map( )`.

chordtools.divide_chord_by_chromatic_pitch_number

`abjad.tools.chordtools.`**`divide_chord_by_chromatic_pitch_number`**`(`*chord, pitch=NamedChromaticPitch('b')*`)`

New in version 1.1.1. Divide *chord* by chromatic *pitch* number:

```
abjad> chord = Chord(range(12), Duration(1, 4))

abjad> chord
Chord("<c' cs' d' ef' e' f' fs' g' af' a' bf' b'>4")

abjad> chordtools.divide_chord_by_chromatic_pitch_number(chord, pitchtools.NamedChromaticPitch(6
(Chord("<fs' g' af' a' bf' b'>4"), Chord("<c' cs' d' ef' e' f'>4"))
```

Input *chord* may be a note, rest or chord but not a skip.

Zero-length parts return rests, length-one parts return notes and other parts return chords.

Return pair of newly constructed leaves. Changed in version 1.1.2: renamed `chordtools.split_by_pitch_number( )` to `chordtools.divide_chord_by_chromatic_pitch_number( )`.

chordtools.divide_chord_by_diatonic_pitch_number

`abjad.tools.chordtools.`**`divide_chord_by_diatonic_pitch_number`**`(`*chord, pitch=NamedChromaticPitch('b')*`)`

New in version 1.1.1. Divide *chord* by diatonic *pitch* number:

```
abjad> chord = Chord(range(12), Duration(1, 4))

abjad> chord
Chord("<c' cs' d' ef' e' f' fs' g' af' a' bf' b'>4")

abjad> chordtools.divide_chord_by_diatonic_pitch_number(chord, pitchtools.NamedChromaticPitch(6)
(Chord("<f' fs' g' af' a' bf' b'>4"), Chord("<c' cs' d' ef' e'>4"))
```

Input *chord* may be a note, rest or chord but not a skip.

Zero-length parts return as rests, length-one parts return as notes and other parts return as chords.

Return pair of newly constructed leaves. Changed in version 1.1.2: renamed `chordtools.split_by_altitude( )` to `chordtools.divide_chord_by_diatonic_pitch_number( )`.

chordtools.get_arithmetic_mean_of_chord

`abjad.tools.chordtools.`**`get_arithmetic_mean_of_chord`**`(`*chord*`)`

New in version 1.1.2. Get arithmetic mean of chromatic pitch number of pitches in *chord*:

```
abjad> chord = Chord("<g' c'' e''>4")

abjad> chordtools.get_arithmetic_mean_of_chord(chord)
11.666666666666666
```

Return none when *chord* is empty:

```
abjad> chord = Chord("< >4")

abjad> chordtools.get_arithmetic_mean_of_chord(chord) is None
True
```

Return number or none.

chordtools.get_note_head_from_chord_by_pitch

`abjad.tools.chordtools.get_note_head_from_chord_by_pitch`(*chord*, *pitch*)

New in version 1.1.2. Get note head from *chord* by *pitch*:

```
abjad> chord = Chord("<c'' d'' b''>4")

abjad> chordtools.get_note_head_from_chord_by_pitch(chord, 14)
NoteHead("d''")
```

Raise missing note head error when *chord* contains no note head with pitch equal to *pitch*.

Raise extra note head error when *chord* contains more than one note head with pitch equal to *pitch*. Changed in version 1.1.2: renamed `chordtools.get_note_head( )` to `chordtools.get_note_head_from_chord_by_pitch( )`.

chordtools.iterate_chords_backward_in_expr

`abjad.tools.chordtools.iterate_chords_backward_in_expr`(*expr*, *start=0*, *stop=None*)

New in version 1.1.2. Iterate chords backward in *expr*:

```
abjad> staff = Staff("<e' g' c''>8 a'8 r8 <d' f' b'>8 r2")

abjad> f(staff)
\new Staff {
   <e' g' c''>8
   a'8
   r8
   <d' f' b'>8
   r2
}

abjad> for chord in chordtools.iterate_chords_backward_in_expr(staff):
...   chord
Chord("<d' f' b'>8")
Chord("<e' g' c''>8")
```

Ignore threads.

Return generator.

chordtools.iterate_chords_forward_in_expr

`abjad.tools.chordtools.`**`iterate_chords_forward_in_expr`**(*expr*, *start=0*, *stop=None*)

New in version 1.1.2. Iterate chords forward in *expr*:

```
abjad> staff = Staff("<e' g' c''>8 a'8 r8 <d' f' b'>8 r2")

abjad> f(staff)
\new Staff {
   <e' g' c''>8
   a'8
   r8
   <d' f' b'>8
   r2
}

abjad> for chord in chordtools.iterate_chords_forward_in_expr(staff):
...   chord
Chord("<e' g' c''>8")
Chord("<d' f' b'>8")
```

Ignore threads.

Return generator.

chordtools.yield_all_subchords_of_chord

`abjad.tools.chordtools.`**`yield_all_subchords_of_chord`**(*chord*)

New in version 1.1.2. Yield all subchords of *chord* in binary string order:

```
abjad> chord = Chord("<c' d' af' a'>4")

abjad> for subchord in chordtools.yield_all_subchords_of_chord(chord):
...     subchord
...
Rest('r4')
Note("c'4")
Note("d'4")
Chord("<c' d'>4")
Note("af'4")
Chord("<c' af'>4")
Chord("<d' af'>4")
Chord("<c' d' af'>4")
Note("a'4")
Chord("<c' a'>4")
Chord("<d' a'>4")
Chord("<c' d' a'>4")
Chord("<af' a'>4")
Chord("<c' af' a'>4")
Chord("<d' af' a'>4")
Chord("<c' d' af' a'>4")
```

Include empty chord as rest.

Return generator of newly constructed leaves. Changed in version 1.1.2: renamed `chordtools.subchords( )` to `chordtools.yield_all_subchords_of_chord( )`.

chordtools.yield_groups_of_chords_in_sequence

abjad.tools.chordtools.**yield_groups_of_chords_in_sequence**(*sequence*)

New in version 1.1.2. Yield groups of chords in *sequence*:

```
abjad> staff = Staff("c'8 d'8 r8 r8 <e' g'>8 <f' a'>8 g'8 a'8 r8 r8 <b' d''>8 <c'' e''>8")

abjad> f(staff)
\new Staff {
   c'8
   d'8
   r8
   r8
   <e' g'>8
   <f' a'>8
   g'8
   a'8
   r8
   r8
   <b' d''>8
   <c'' e''>8
}

abjad> for chord in chordtools.yield_groups_of_chords_in_sequence(staff):
...     chord
...
(Chord("<e' g'>8"), Chord("<f' a'>8"))
(Chord("<b' d''>8"), Chord("<c'' e''>8"))
```

Return generator.

componenttools

componenttools.all_are_components

abjad.tools.componenttools.**all_are_components**(*expr*, *klasses=None*)

New in version 1.1.1. True when elements in *expr* are all components:

```
abjad> componenttools.all_are_components(3 * Note("c'4"))
True
```

Otherwise false:

```
abjad> componenttools.all_are_components(['foo', 'bar'])
False
```

True when elements in *expr* are all *klasses*:

```
abjad> componenttools.all_are_components(3 * Note("c'4"), klasses = Note)
True
```

Otherwise false:

```
abjad> componenttools.all_are_components(['foo', 'bar'], klasses = Note)
False
```

Return boolean.

componenttools.all_are_components_in_same_parent

`abjad.tools.componenttools.all_are_components_in_same_parent(expr, klasses=None, allow_orphans=True)`

New in version 1.1.1. True when elements in *expr* are all components in same parent. Otherwise false:

```
abjad> staff = Staff(notetools.make_notes([12, 14, 16], [(1, 8)]))
abjad> componenttools.all_are_components_in_same_parent(staff.leaves)
True
```

True when elements in *expr* are all *klasses* in same parent. Otherwise false:

```
abjad> staff = Staff(notetools.make_notes([12, 14, 16], [(1, 8)]))
abjad> componenttools.all_are_components_in_same_parent(staff.leaves, klasses = (Note, ))
True
```

Return boolean.

componenttools.all_are_components_in_same_score

`abjad.tools.componenttools.all_are_components_in_same_score(expr, klasses=None, allow_orphans=True)`

New in version 1.1.1. True when elements in *expr* are all components in same score. Otherwise false:

```
abjad> score = Score([Staff(macros.scale(3))])
abjad> componenttools.all_are_components_in_same_score(score.leaves)
True
```

True when elements in *expr* are all *klasses* in same score. Otherwise false:

```
abjad> score = Score([Staff(macros.scale(3))])
abjad> componenttools.all_are_components_in_same_score(score.leaves, klasses = (Note, ))
True
```

Return boolean.

componenttools.all_are_components_in_same_thread

`abjad.tools.componenttools.all_are_components_in_same_thread(expr, klasses=None, allow_orphans=True)`

New in version 1.1.1. True when elements in *expr* are all components in same thread. Otherwise false:

```
abjad> voice = Voice(macros.scale(3))
abjad> componenttools.all_are_components_in_same_thread(voice.leaves)
True
```

True when elements in *expr* are all *klasses* in same thread. Otherwise false:

```
abjad> voice = Voice(macros.scale(3))
abjad> componenttools.all_are_components_in_same_thread(voice.leaves, klasses = Note)
True
```

Return boolean.

componenttools.all_are_components_scalable_by_multiplier

`abjad.tools.componenttools.all_are_components_scalable_by_multiplier(components, multiplier)`

New in version 1.1.1. True when *components* are all scalable by *multiplier*:

```
abjad> components = [Note(0, (1, 8))]
abjad> componenttools.all_are_components_scalable_by_multiplier(components, Duration(3, 2))
True
```

Otherwise false:

```
abjad> components = [Note(0, (1, 8))]
abjad> componenttools.all_are_components_scalable_by_multiplier(components, Duration(2, 3))
False
```

Return boolean. Changed in version 1.1.2: renamed `durtools.are_scalable( )` to `componenttools.all_are_components_scalable_by_multiplier( )`.

componenttools.all_are_contiguous_components

`abjad.tools.componenttools.all_are_contiguous_components(expr, klasses=None, allow_orphans=True)`

New in version 1.1.1. True when elements in *expr* are all contiguous components. Otherwise false:

```
abjad> staff = Staff(macros.scale(3))
abjad> componenttools.all_are_contiguous_components(staff.leaves)
True
```

True when elements in *expr* are all contiguous *klasses*. Otherwise false:

```
abjad> staff = Staff(macros.scale(3))
abjad> componenttools.all_are_contiguous_components(staff.leaves, klasses = Note)
True
```

Return boolean.

componenttools.all_are_contiguous_components_in_same_parent

`abjad.tools.componenttools.all_are_contiguous_components_in_same_parent(expr, klasses=None, allow_orphans=True)`

New in version 1.1.1. True when elements in *expr* are all contiguous components in same parent. Otherwise false:

```
abjad> staff = Staff(macros.scale(3))
abjad> componenttools.all_are_contiguous_components_in_same_parent(staff.leaves)
True
```

True when elements in *expr* are all contiguous *klasses* in same parent. Otherwise false:

```
abjad> staff = Staff(macros.scale(3))
abjad> componenttools.all_are_contiguous_components_in_same_parent(staff.leaves, klasses = Note)
True
```

Return boolean.

componenttools.all_are_contiguous_components_in_same_score

`abjad.tools.componenttools.`**`all_are_contiguous_components_in_same_score`**`(`*expr*, *klasses=None*, *allow_orphans=True*`)`

New in version 1.1.1. True when elements in *expr* are all contiguous components in same score. Otherwise false:

```
abjad> score = Score([Staff(macros.scale(3))])
abjad> componenttools.all_are_contiguous_components_in_same_score(score.leaves)
True
```

True when elements in *expr* are all contiguous *klasses* in same score. Otherwise false:

```
abjad> score = Score([Staff(macros.scale(3))])
abjad> componenttools.all_are_contiguous_components_in_same_score(score.leaves, klasses = Note)
True
```

Return boolean.

componenttools.all_are_contiguous_components_in_same_thread

`abjad.tools.componenttools.`**`all_are_contiguous_components_in_same_thread`**`(`*expr*, *klasses=None*, *allow_orphans=True*`)`

New in version 1.1.1. True when elements in *expr* are all contiguous components in same thread. Otherwise false:

```
abjad> staff = Staff(macros.scale(3))
abjad> componenttools.all_are_contiguous_components_in_same_thread(staff.leaves)
True
```

True when elements in *expr* are all contiguous *klasses* in same thread. Otherwise false:

```
abjad> staff = Staff(macros.scale(3))
abjad> componenttools.all_are_contiguous_components_in_same_thread(staff.leaves, klasses = Note)
True
```

Return boolean.

componenttools.all_are_orphan_components

`abjad.tools.componenttools.`**`all_are_orphan_components`**`(`*expr*`)`

New in version 1.1.2. True when *expr* is an iterable of zero or more orphan components.

Othewise false.

componenttools.all_are_thread_contiguous_components

`abjad.tools.componenttools.`**`all_are_thread_contiguous_components`**`(`*expr*, *klasses=None*, *allow_orphans=True*`)`

New in version 1.1.1. True when elements in *expr* are all thread-contiguous components:

```
t = Voice(notetools.make_repeated_notes(4))
t.insert(2, Voice(notetools.make_repeated_notes(2)))
Container(t[:2])
Container(t[-2:])
macros.diatonicize(t)

\new Voice {
   {
      c'8
      d'8
   }
   \new Voice {
      e'8
      f'8
   }
   {
      g'8
      a'8
   }
}

assert _are_thread_contiguous_components(t[0:1] + t[-1:])
assert _are_thread_contiguous_components(t[0][:] + t[-1:])
assert _are_thread_contiguous_components(t[0:1] + t[-1][:])
assert _are_thread_contiguous_components(t[0][:] + t[-1][:])
```

Return boolean.

Thread-contiguous components are, by definition, spannable.

componenttools.clone_and_partition_governed_component_subtree_by_leaf_counts

abjad.tools.componenttools.**clone_and_partition_governed_component_subtree_by_leaf_counts**(*co*
le

New in version 1.1.1. Clone *container* and partition clone according to *leaf_counts*:

```
abjad> voice = Voice(tuplettools.FixedDurationTuplet((2, 8), notetools.make_repeated_notes(3)) *
abjad> spannertools.BeamSpanner(voice[0].leaves)
BeamSpanner(c'8, c'8, c'8)
abjad> spannertools.BeamSpanner(voice[1].leaves)
BeamSpanner(c'8, c'8, c'8)
abjad> macros.diatonicize(voice)
abjad> f(voice)
\new Voice {
  \times 2/3 {
          c'8 [
          d'8
          e'8 ]
  }
  \times 2/3 {
          f'8 [
          g'8
          a'8 ]
  }
}

abjad> first, second, third = componenttools.clone_and_partition_governed_component_subtree_by_l
```

```
abjad> f(first)
\new Voice {
  \times 2/3 {
          c'8 [ ]
  }
}

abjad> f(second)
\new Voice {
  \times 2/3 {
          d'8 [
          e'8 ]
  }
}

abjad> f(third)
\new Voice {
  \times 2/3 {
          f'8 [
          g'8
          a'8 ]
  }
}
```

Set *leaf_counts* to an iterable of zero or more positive integers.

Return a list of parts equal in length to that of *leaf_counts*. Changed in version 1.1.2: renamed `clonewp.by_leaf_counts_with_parentage( )` to `componenttools.clone_and_partition_governed_component_subtree_by_leaf_counts( )`.

componenttools.clone_components_and_covered_spanners

`abjad.tools.componenttools.`**`clone_components_and_covered_spanners`**`(`*components*, *n=1*`)`

New in version 1.1.1. Clone *components* and covered spanners.

The *components* must be thread-contiguous.

Covered spanners are those spanners that cover *components*.

The steps taken in this function are as follows. Withdraw *components* from crossing spanners. Preserve spanners that *components* cover. Deep copy *components*. Reapply crossing spanners to source *components*. Return copied components with covered spanners.

```
abjad> voice = Voice(Measure((2, 8), notetools.make_repeated_notes(2)) * 3)
abjad> macros.diatonicize(voice)
abjad> beam = spannertools.BeamSpanner(voice.leaves[:4])
abjad> f(voice)
\new Voice {
        {
                \time 2/8
                c'8 [
                d'8
        }
        {
                \time 2/8
                e'8
```

```
                f'8 ]
        }
        {
                \time 2/8
                g'8
                a'8
        }
}
```

```
abjad> result = componenttools.clone_components_and_covered_spanners(voice.leaves)
abjad> result
(Note("c'8"), Note("d'8"), Note("e'8"), Note("f'8"), Note("g'8"), Note("a'8"))
```

```
abjad> new_voice = Voice(result)
abjad> f(new_voice)
\new Voice {
        c'8 [
        d'8
        e'8
        f'8 ]
        g'8
        a'8
}
```

```
abjad> voice.leaves[0] is new_voice.leaves[0]
False
```

Clone *components* a total of *n* times.

```
abjad> result = componenttools.clone_components_and_covered_spanners(voice.leaves[:2], n = 3)
abjad> result
(Note("c'8"), Note("d'8"), Note("c'8"), Note("d'8"), Note("c'8"), Note("d'8"))
```

```
abjad> new_voice = Voice(result)
abjad> f(new_voice)
\new Voice {
        c'8
        d'8
        c'8
        d'8
        c'8
        d'8
}
```

Changed in version 1.1.2: renamed `clone.covered( )` to `componenttools.clone_components_and_covered_s` `)`.

componenttools.clone_components_and_fracture_crossing_spanners

abjad.tools.componenttools.**clone_components_and_fracture_crossing_spanners**(*components*, *n=1*)

New in version 1.1.1. Clone *components* and fracture crossing spanners.

The *components* must be thread-contiguous.

The steps this function takes are as follows. Deep copy *components*. Deep copy spanners that attach to any component in *components*. Fracture spanners that attach to components not in *components*. Return Python list of copied components.

```
abjad> voice = Voice(Measure((2, 8), notetools.make_repeated_notes(2)) * 3)
abjad> macros.diatonicize(voice)
abjad> beam = spannertools.BeamSpanner(voice.leaves[:4])
abjad> f(voice)
\new Voice {
        {
                \time 2/8
                c'8 [
                d'8
        }
        {
                \time 2/8
                e'8
                f'8 ]
        }
        {
                \time 2/8
                g'8
                a'8
        }
}
```

```
abjad> result = componenttools.clone_components_and_fracture_crossing_spanners(voice.leaves[2:4]
abjad> result
(Note("e'8"), Note("f'8"))
```

```
abjad> new_voice = Voice(result)
abjad> f(new_voice)
\new Voice {
        e'8 [
        f'8 ]
}
```

```
abjad> voice.leaves[2] is new_voice.leaves[0]
False
```

Clone *components* a total of *n* times.

```
abjad> result = componenttools.clone_components_and_fracture_crossing_spanners(voice.leaves[2:4]
abjad> result
(Note("e'8"), Note("f'8"), Note("e'8"), Note("f'8"), Note("e'8"), Note("f'8"))
```

```
abjad> new_voice = Voice(result)
abjad> f(new_voice)
\new Voice {
        e'8 [
        f'8 ]
        e'8 [
        f'8 ]
        e'8 [
        f'8 ]
}
```

Changed in version 1.1.2: renamed `clone.fracture( )` to `componenttools.clone_components_and_fracture`
`)`.

componenttools.clone_components_and_immediate_parent_of_first_component

abjad.tools.componenttools.**clone_components_and_immediate_parent_of_first_component**(*componen*

New in version 1.1.1. Clone *components* and immediate parent of first component.

The *components* must be thread-contiguous.

Return in newly created container equal to type of first element in *copmonents*.

If the parent of the first element in *components* is a tuplet then insure that the tuplet multiplier of the function output equals the tuplet multiplier of the parent of the first element in *components*.

```
abjad> voice = Voice(tuplettools.FixedDurationTuplet((2, 8), notetools.make_repeated_notes(3)) *
abjad> macros.diatonicize(voice)
abjad> beam = spannertools.BeamSpanner(voice.leaves[:4])
abjad> f(voice)
\new Voice {
        \times 2/3 {
                c'8 [
                d'8
                e'8
        }
        \times 2/3 {
                f'8 ]
                g'8
                a'8
        }
        \times 2/3 {
                b'8
                c''8
                d''8
        }
}
abjad> new_tuplet = componenttools.clone_components_and_immediate_parent_of_first_component(voic
abjad> new_tuplet
FixedDurationTuplet(1/6, [c'8, d'8])
abjad> f(new_tuplet)
\times 2/3 {
        c'8 [
        d'8 ]
}
```

Parent-contiguity is not required. Thread-contiguous *components* suffice.

```
abjad> new_tuplet = componenttools.clone_components_and_immediate_parent_of_first_component(voic
abjad> new_tuplet
FixedDurationTuplet(5/12, [c'8, d'8, e'8, f'8, g'8])
abjad> f(new_tuplet)
\times 2/3 {
        c'8 [
        d'8
        e'8
        f'8 ]
        g'8
}
```

Note: this function copies only the *immediate parent* of the first element in *components*. This function ignores any further parentage of *components* above the immediate parent of *components*.

Todo

this function should (but does not) copy marks that attach to *components* and to the immediate parent of the first component; extend function to do so.

Changed in version 1.1.2: renamed `clonewp.with_parent()` to `componenttools.clone_components_and_immediate_parent_of_first_component()`.

componenttools.clone_components_and_remove_all_spanners

`abjad.tools.componenttools.`**`clone_components_and_remove_all_spanners`**(*components*, *n=1*)

New in version 1.1.1. Clone *components* and remove all spanners.

The *components* must be thread-contiguous.

The steps taken by this function are as follows. Withdraw all components at any level in *components* from spanners. Deep copy unspanned components in *components*. Reapply spanners to all components at any level in *components*.

```
abjad> voice = Voice(Measure((2, 8), notetools.make_repeated_notes(2)) * 3)
abjad> macros.diatonicize(voice)
abjad> beam = spannertools.BeamSpanner(voice.leaves[:4])
abjad> f(voice)
\new Voice {
        {
                \time 2/8
                c'8 [
                d'8
        }
        {
                \time 2/8
                e'8
                f'8 ]
        }
        {
                \time 2/8
                g'8
                a'8
        }
}
```

```
abjad> result = componenttools.clone_components_and_remove_all_spanners(voice.leaves[2:4])
abjad> result
(Note("e'8"), Note("f'8"))
```

```
abjad> new_voice = Voice(result)
abjad> f(new_voice)
\new Voice {
        e'8
        f'8
}
```

```
abjad> voice.leaves[2] is new_voice.leaves[0]
False
```

Clone *components* a total of *n* times.

```
abjad> result = componenttools.clone_components_and_remove_all_spanners(voice.leaves[2:4], n = 3
abjad> result
(Note("e'8"), Note("f'8"), Note("e'8"), Note("f'8"), Note("e'8"), Note("f'8"))

abjad> new_voice = Voice(result)
abjad> f(new_voice)
\new Voice {
        e'8
        f'8
        e'8
        f'8
        e'8
        f'8
}
```

Changed in version 1.1.2: renamed `clone.unspan( )` to `componenttools.clone_components_and_remove_all` `).`

componenttools.clone_governed_component_subtree_by_leaf_range

`abjad.tools.componenttools.`**`clone_governed_component_subtree_by_leaf_range`**(*component*, *start=0*, *stop=None*)

New in version 1.1.1. Clone governed *component* subtree by leaf range.

Governed subtree means *component* together with children of *component*.

Leaf range refers to the sequential parentage of *component* from *start* leaf index to *stop* leaf index:

```
abjad> t = Staff([Voice(tuplettools.FixedDurationTuplet((2, 8), notetools.make_repeated_notes(3)
abjad> macros.diatonicize(t)
abjad> f(t)
\new Staff {
   \new Voice {
      \times 2/3 {
         c'8
         d'8
         e'8
      }
      \times 2/3 {
         f'8
         g'8
         a'8
      }
   }
}

abjad> u = componenttools.clone_governed_component_subtree_by_leaf_range(t, 1, 5)
abjad> f(u)
\new Staff {
   \new Voice {
      \times 2/3 {
         d'8
         e'8
      }
      \times 2/3 {
         f'8
```

```
            g'8
        }
    }
}
```

Clone sequential containers in leaves' parentage up to the first parallel container in leaves' parentage.

Trim and shrink cloned containers as necessary.

When *stop* is none copy all leaves from *start* forward. Changed in version 1.1.2: renamed `clonewp.by_leaf_range_with_parentage( )` to `componenttools.clone_governed_component_subtree_by_leaf_range( )`.

componenttools.clone_governed_component_subtree_from_prolated_duration_to

abjad.tools.componenttools.**clone_governed_component_subtree_from_prolated_duration_to**(*compo*
start=
stop=

New in version 1.1.1. Clone governed *component* subtree from *start* prolated duration to *stop* prolated duration.

Governed subtree refers to *component* together with the children of *component*:

```
abjad> voice = Voice(notetools.make_repeated_notes(2))
abjad> voice.append(tuplettools.FixedDurationTuplet((2, 8), notetools.make_repeated_notes(3)))
abjad> macros.diatonicize(voice)
abjad> f(voice)
\new Voice {
  c'8
  d'8
  \times 2/3 {
          e'8
          f'8
          g'8
  }
}

abjad> new = componenttools.clone_governed_component_subtree_from_prolated_duration_to(voice, (0
abjad> f(new)
\new Voice {
  c'8
  d'8
  \times 2/3 {
          e'8
          f'16
  }
}
```

Raise contiguity error if asked to slice a parallel container.

```
abjad> staff = Staff(Voice(macros.scale(2)) * 2)
abjad> staff.is_parallel = True
abjad> f(staff)
\new Staff <<
  \new Voice {
          c'8
          d'8
  }
  \new Voice {
          c'8
```

```
            d'8
    }
>>
```

Raise contiguity error when attempting to copy fleaves from parallel container.

But note that cases with `0 = start` work correctly:

```
abjad> new = componenttools.clone_governed_component_subtree_from_prolated_duration_to(voice, (0
abjad> f(new)
\new Voice {
  c'8
}
```

Cases with `0 < start` do not work correctly:

```
abjad> new = componenttools.clone_governed_component_subtree_from_prolated_duration_to(voice, (1
abjad> f(new)
\new Voice {
  c'8
  d'8
}
```

Create ad hoc tuplets as required:

```
abjad> voice = Voice([Note(0, (1, 4))])
abjad> new = componenttools.clone_governed_component_subtree_from_prolated_duration_to(voice, 0,
abjad> f(new)
\new Voice {
  \times 2/3 {
          c'8
  }
}
```

Function does NOT clone parentage of *component* when *component* is a leaf:

```
abjad> voice = Voice([Note(0, (1, 4))])
abjad> new_leaf = componenttools.clone_governed_component_subtree_from_prolated_duration_to(voic
abjad> f(new_leaf)
c'8
abjad> new_leaf._parentage.parent is None
True
```

Return (untrimmed_copy, first_dif, second_dif).

componenttools.component_to_parentage_signature

`abjad.tools.componenttools.`**`component_to_parentage_signature`**`(`*component*`)`

New in version 1.1.1. Change *component* to parentage signature:

```
abjad> tuplet = tuplettools.FixedDurationTuplet((2, 8), macros.scale(3))
abjad> staff = Staff([tuplet])
abjad> note = staff.leaves[0]
abjad> print componenttools.component_to_parentage_signature(note)
      root: Staff-... (...)
     score:
staffgroup:
     staff: Staff-...
```

```
voice:
 self: Note-...
```

Return parentage signature.

componenttools.component_to_pitch_and_rhythm_skeleton

abjad.tools.componenttools.**component_to_pitch_and_rhythm_skeleton**(*component*)
New in version 1.1.2. Change *component* to pitch and rhythm skeleton:

```
abjad> tuplet = Tuplet((3, 4), macros.scale(4))
abjad> measure = Measure((6, 16), [tuplet])
abjad> staff = Staff([measure])
abjad> score = Score(staff * 2)
abjad> macros.diatonicize(score)

abjad> skeleton = componenttools.component_to_pitch_and_rhythm_skeleton(score)
abjad> print skeleton
Score([
   Staff([
      Measure((6, 16), [
         Tuplet(Duration(3, 4), [
            Note(('c', 4), Duration(1, 8)),
            Note(('d', 4), Duration(1, 8)),
            Note(('e', 4), Duration(1, 8)),
            Note(('f', 4), Duration(1, 8))
         ])
      ])
   ]),
   Staff([
      Measure((6, 16), [
         Tuplet(Duration(3, 4), [
            Note(('g', 4), Duration(1, 8)),
            Note(('a', 4), Duration(1, 8)),
            Note(('b', 4), Duration(1, 8)),
            Note(('c', 5), Duration(1, 8))
         ])
      ])
   ])
])

abjad> new = eval(skeleton)
abjad> new
Score<<2>>

abjad> f(new)
\new Score <<
   \new Staff {
      {
         \time 6/16
         \fraction \times 3/4 {
            c'8
            d'8
            e'8
            f'8
         }
      }
```

```
        }
        \new Staff {
            {
                \time 6/16
                \fraction \times 3/4 {
                    g'8
                    a'8
                    b'8
                    c''8
                }
            }
        }
    >>
```

Return string.

componenttools.component_to_pitch_and_rhythm_skeleton_with_interface_attributes

`abjad.tools.componenttools.component_to_pitch_and_rhythm_skeleton_with_interface_attributes`

New in version 1.1.2. Change *component* to pitch and rhythm skeleton with interface attributes.

Return string.

Note: function currently not working.

componenttools.component_to_score_depth

`abjad.tools.componenttools.component_to_score_depth(component)`

New in version 1.1.1. Change *component* to score depth:

```
abjad> tuplet = tuplettools.FixedDurationTuplet((2, 8), macros.scale(3))
abjad> staff = Staff([tuplet])
abjad> componenttools.component_to_score_depth(staff.leaves[0])
2
```

Return nonnegative integer.

componenttools.component_to_score_index

`abjad.tools.componenttools.component_to_score_index(component)`

New in version 1.1.2. Change *component* to score index:

```
abjad> staff_1 = Staff(tuplettools.FixedDurationTuplet((2, 8), notetools.make_repeated_notes(3))
abjad> staff_2 = Staff([tuplettools.FixedDurationTuplet((2, 8), notetools.make_repeated_notes(3)
abjad> score = Score([staff_1, staff_2])
abjad> macros.diatonicize(score)
abjad> f(score)
\new Score <<
        \new Staff {
                \times 2/3 {
                        c'8
                        d'8
                        e'8
                }
```

```
                \times 2/3 {
                        f'8
                        g'8
                        a'8
                }
        }
        \new Staff {
                \times 2/3 {
                        b'8
                        c''8
                        d''8
                }
        }
>>
```

```
abjad> for leaf in score.leaves:
...     leaf, componenttools.component_to_score_index(leaf)
...
(Note("c'8"), (0, 0, 0))
(Note("d'8"), (0, 0, 1))
(Note("e'8"), (0, 0, 2))
(Note("f'8"), (0, 1, 0))
(Note("g'8"), (0, 1, 1))
(Note("a'8"), (0, 1, 2))
(Note("b'8"), (1, 0, 0))
(Note("c''8"), (1, 0, 1))
(Note("d''8"), (1, 0, 2))
```

Return tuple of zero or more nonnegative integers.

componenttools.component_to_score_root

abjad.tools.componenttools.**component_to_score_root**(*component*)

New in version 1.1.1. Change *component* to score root:

```
abjad> tuplet = tuplettools.FixedDurationTuplet((2, 8), macros.scale(3))
abjad> staff = Staff([tuplet])
abjad> note = staff.leaves[0]
abjad> componenttools.component_to_score_root(note)
Staff{1}
```

Return score root.

componenttools.component_to_tuplet_depth

abjad.tools.componenttools.**component_to_tuplet_depth**(*component*)

New in version 1.1.1. Change *component* to tuplet depth:

```
abjad> tuplet = tuplettools.FixedDurationTuplet((2, 8), macros.scale(3))
abjad> staff = Staff([tuplet])
abjad> note = staff.leaves[0]

abjad> componenttools.component_to_tuplet_depth(note)
1
```

```
abjad> componenttools.component_to_tuplet_depth(tuplet)
0

abjad> componenttools.component_to_tuplet_depth(staff)
0
```

Return nonnegative integer.

componenttools.cut_component_at_prolated_duration

abjad.tools.componenttools.**cut_component_at_prolated_duration**(*component*, *prolated_duration*)

New in version 1.1.2. Cut *component* at dotted *prolated_duration*:

```
abjad> staff = Staff(macros.scale(4))
abjad> spannertools.BeamSpanner(staff.leaves)
BeamSpanner(c'8, d'8, e'8, f'8)
abjad> componenttools.cut_component_at_prolated_duration(staff, Duration(1, 32))
abjad> f(staff)
\new Staff {
   c'16. [
   d'8
   e'8
   f'8 ]
}
```

Cut *component* at tied *prolated_duration*:

```
abjad> staff = Staff(macros.scale(4))
abjad> spannertools.BeamSpanner(staff.leaves)
BeamSpanner(c'8, d'8, e'8, f'8)
abjad> componenttools.cut_component_at_prolated_duration(staff, Duration(3, 64))
abjad> f(staff)
\new Staff {
   c'16 [ ~
   c'64
   d'8
   e'8
   f'8 ]
}
```

Cut *component* at nonbinary *prolated_duration*:

```
abjad> staff = Staff(macros.scale(4))
abjad> spannertools.BeamSpanner(staff.leaves)
BeamSpanner(c'8, d'8, e'8, f'8)
abjad> componenttools.cut_component_at_prolated_duration(staff, Duration(1, 24))
abjad> f(staff)
\new Staff {
   \times 2/3 {
      c'8 [
   }
   d'8
   e'8
   f'8 ]
}
```

Return none.

componenttools.extend_in_parent_of_component_and_do_not_grow_spanners

abjad.tools.componenttools.**extend_in_parent_of_component_and_do_not_grow_spanners**(*component*, *components*)

New in version 1.1.1. Extend *components* in parent of *component* and do not grow spanners:

```
abjad> t = Voice(macros.scale(3))
abjad> spannertools.BeamSpanner(t[:])
BeamSpanner(c'8, d'8, e'8)
abjad> componenttools.extend_in_parent_of_component_and_do_not_grow_spanners(t[-1], macros.scale
[Note("e'8"), Note("c'8"), Note("d'8"), Note("e'8")]

abjad> print t.format
\new Voice {
   c'8 [
   d'8
   e'8 ]
   c'8
   d'8
   e'8
}
```

Return list of *component* and *components*. Changed in version 1.1.2: renamed `extend_in_parent( )` to `extend_in_parent_of_component_and_do_not_grow_spanners( )`.

componenttools.extend_in_parent_of_component_and_grow_spanners

abjad.tools.componenttools.**extend_in_parent_of_component_and_grow_spanners**(*component*, *new_components*)

New in version 1.1.2. Extend *new_components* in parent of *component* and grow spanners:

```
abjad> voice = Voice(macros.scale(3))
abjad> spannertools.BeamSpanner(voice[:])
BeamSpanner(c'8, d'8, e'8)

abjad> f(voice)
\new Voice {
   c'8 [
   d'8
   e'8 ]
}

abjad> new_components = macros.scale(3)
abjad> componenttools.extend_in_parent_of_component_and_grow_spanners(voice.leaves[-1], new_comp
[Note("e'8"), Note("c'8"), Note("d'8"), Note("e'8")]

abjad> f(voice)
\new Voice {
   c'8 [
   d'8
   e'8
   c'8
   d'8
   e'8 ]
}
```

Return *component* and *new_components* together in list.

componenttools.extend_left_in_parent_of_component_and_do_not_grow_spanners

abjad.tools.componenttools.**extend_left_in_parent_of_component_and_do_not_grow_spanners**(*comp com- po- nents*

New in version 1.1.1. Extend *components* left in parent of *component* and do not grow spanners:

```
abjad> t = Voice(macros.scale(3))
abjad> spannertools.BeamSpanner(t[:])
BeamSpanner(c'8, d'8, e'8)
abjad> componenttools.extend_left_in_parent_of_component_and_do_not_grow_spanners(t[0], macros.s
[Note("c'8"), Note("d'8"), Note("e'8"), Note("c'8")]

abjad> print t.format
\new Voice {
   c'8
   d'8
   e'8
   c'8 [
   d'8
   e'8 ]
}
```

Return *components* and *component* together in newly created list. Changed in version 1.1.2: renamed `extend_left_in_parent( )` to `extend_left_in_parent_of_component_and_do_not_grow_spanners( )`.

componenttools.extend_left_in_parent_of_component_and_grow_spanners

abjad.tools.componenttools.**extend_left_in_parent_of_component_and_grow_spanners**(*component*, *new_componen*

New in version 1.1.2. Extend *new_components* left in parent of *component* and grow spanners:

```
abjad> voice = Voice(macros.scale(3))
abjad> spannertools.BeamSpanner(voice[:])
BeamSpanner(c'8, d'8, e'8)

abjad> f(voice)
\new Voice {
   c'8 [
   d'8
   e'8 ]
}

abjad> new_components = 3 * Note(0, (1, 16))
abjad> componenttools.extend_left_in_parent_of_component_and_grow_spanners(voice[0], new_compone
[Note("c'16"), Note("c'16"), Note("c'16"), Note("c'8")]

abjad> f(voice)
\new Voice {
   c'16 [
   c'16
   c'16
   c'8
```

```
      d'8
      e'8 ]
   }
```

Return *new_components* and *component* together in newly created list. Changed in version 1.1.2: renamed `splice_left( )` to `componenttools.extend_left_in_parent_of_component_and_grow_spanners( )`.

componenttools.get_component_start_offset

`abjad.tools.componenttools.`**`get_component_start_offset`**`(`*component*`)`

New in version 1.1.1. Get *component* start offset:

```
abjad> staff = Staff(macros.scale(4))
abjad> f(staff)
\new Staff {
   c'8
   d'8
   e'8
   f'8
}

abjad> componenttools.get_component_start_offset(staff[1])
Duration(1, 8)
```

Return nonnegative fraction.

componenttools.get_component_start_offset_in_seconds

`abjad.tools.componenttools.`**`get_component_start_offset_in_seconds`**`(`*component*`)`

New in version 1.1.1. Get *component* start offset in seconds:

```
abjad> staff = Staff(macros.scale(4))
abjad> score = Score([staff])
abjad> contexttools.TempoMark(Duration(1, 4), 52)(score)
TempoMark(4, 52)(Score<<1>>)
abjad> f(score) # doctest: +SKIP
\new Score <<
   \new Staff {
      \tempo 4=52
      c'8
      d'8
      e'8
      f'8
   }
>>

abjad> componenttools.get_component_start_offset_in_seconds(score.leaves[1])
Duration(15, 26)
```

Return nonnegative fraction.

componenttools.get_component_stop_offset

`abjad.tools.componenttools.`**`get_component_stop_offset`**`(`*component*`)`

New in version 1.1.1. Get *component* stop offset:

```
abjad> staff = Staff(macros.scale(4))
abjad> f(staff)
\new Staff {
   c'8
   d'8
   e'8
   f'8
}

abjad> componenttools.get_component_stop_offset(staff[1])
Duration(1, 4)
```

Return positive fraction.

componenttools.get_component_stop_offset_in_seconds

abjad.tools.componenttools.**get_component_stop_offset_in_seconds**(*component*)

New in version 1.1.1. Get *component* stop offset in seconds:

```
abjad> staff = Staff(macros.scale(4))
abjad> score = Score([staff])
abjad> contexttools.TempoMark(Duration(1, 4), 52)(score)
TempoMark(4, 52)(Score<<1>>)
abjad> f(score) # doctest: +SKIP
\new Score <<
   \new Staff {
      \tempo 4=52
      c'8
      d'8
      e'8
      f'8
   }
>>

abjad> componenttools.get_component_stop_offset_in_seconds(score.leaves[1])
Duration(15, 13)
```

Return positive fraction.

componenttools.get_first_component_in_expr_with_name

abjad.tools.componenttools.**get_first_component_in_expr_with_name**(*expr*, *name*)

New in version 1.1.1. Get first component in *expr* with *name*:

```
abjad> flute_staff = Staff(macros.scale(4))
abjad> flute_staff.name = 'Flute'
abjad> violin_staff = Staff(macros.scale(4))
abjad> violin_staff.name = 'Violin'
abjad> staff_group = scoretools.StaffGroup([flute_staff, violin_staff])
abjad> score = Score([staff_group])

abjad> componenttools.get_first_component_in_expr_with_name(score, 'Violin')
Staff-"Violin"{4}
```

Changed in version 1.1.2: Function returns first component found. Function previously returned tuple of all components found.Changed in version 1.1.2: renamed `scoretools.find( )` to componenttools.get_first_component_in_expr_with_name( ).Changed in version 1.1.2: Removed *klass* and *context* keywords. Function operates only on component name.

componenttools.get_first_component_with_name_in_improper_parentage_of_component

abjad.tools.componenttools.**get_first_component_with_name_in_improper_parentage_of_component**

New in version 1.1.2. Get first component with *name* in improper parentage of *component*:

```
abjad> score = Score([Staff("c'4 d'4 e'4 f'4")])
abjad> score.name = 'The Score'

abjad> f(score)
\context Score = "The Score" <<
   \new Staff {
      c'4
      d'4
      e'4
      f'4
   }
>>

abjad> leaf = score.leaves[0]

abjad> componenttools.get_first_component_with_name_in_improper_parentage_of_component(leaf, 'Th
Score-"The Score"<<1>>

abjad> componenttools.get_first_component_with_name_in_improper_parentage_of_component(leaf, 'fo
True
```

Return component or none.

componenttools.get_first_component_with_name_in_proper_parentage_of_component

abjad.tools.componenttools.**get_first_component_with_name_in_proper_parentage_of_component**(

New in version 1.1.2. Get first component with *name* in proper parentage of *component*:

```
abjad> score = Score([Staff("c'4 d'4 e'4 f'4")])
abjad> score.name = 'The Score'

abjad> f(score)
\context Score = "The Score" <<
   \new Staff {
      c'4
      d'4
      e'4
      f'4
   }
>>

abjad> leaf = score.leaves[0]

abjad> componenttools.get_first_component_with_name_in_proper_parentage_of_component(leaf, 'The
Score-"The Score"<<1>>
```

```
abjad> componenttools.get_first_component_with_name_in_proper_parentage_of_component(leaf, 'foo'
True
```

Return component or none.

componenttools.get_first_instance_of_klass_in_improper_parentage_of_component

abjad.tools.componenttools.**get_first_instance_of_klass_in_improper_parentage_of_component**(*c*
k

New in version 1.1.2. Get first instance of *klass* in improper parentage of *component*:

```
abjad> staff = Staff(macros.scale(4))
abjad> componenttools.get_first_instance_of_klass_in_improper_parentage_of_component(staff[0], N
Note("c'8")
```

Return component or none.

componenttools.get_first_instance_of_klass_in_proper_parentage_of_component

abjad.tools.componenttools.**get_first_instance_of_klass_in_proper_parentage_of_component**(*com*
klas

New in version 1.1.1. Get first instance of *klass* in proper parentage of *component*:

```
abjad> staff = Staff(macros.scale(4))
abjad> componenttools.get_first_instance_of_klass_in_proper_parentage_of_component(staff[0], Sta
Staff{4}
```

Return component or none. Changed in version 1.1.2: renamed `componenttools.get_first( )` to `componenttools.get_first_instance_of_klass_in_proper_parentage_of_component( )`.

componenttools.get_improper_parentage_of_component

abjad.tools.componenttools.**get_improper_parentage_of_component**(*component*)

New in version 1.1.1. Get improper parentage of *component*:

```
abjad> tuplet = Tuplet((2, 3), macros.scale(3))
abjad> staff = Staff([tuplet])
abjad> note = staff.leaves[0]

abjad> componenttools.get_improper_parentage_of_component(note)
(Note("c'8"), Tuplet(2/3, [c'8, d'8, e'8]), Staff{1})
```

Return tuple of zero or more components.

componenttools.get_likely_multiplier_of_components

abjad.tools.componenttools.**get_likely_multiplier_of_components**(*components*)

New in version 1.1.2. Get likely multiplier of *components*:

```
abjad> staff = Staff(macros.scale(4, (7, 32)))
abjad> f(staff)
\new Staff {
   c'8..
   d'8..
```

```
    e'8..
    f'8..
}
abjad> componenttools.get_likely_multiplier_of_components(staff[:])
Duration(7, 4)
```

Return 1 when no multiplier is likely:

```
abjad> staff = Staff(macros.scale(4))
abjad> f(staff)
\new Staff {
    c'8
    d'8
    e'8
    f'8
}
abjad> componenttools.get_likely_multiplier_of_components(staff[:])
Duration(1, 1)
```

Return none when more than one multiplier is likely:

```
abjad> staff = Staff(notetools.make_notes([0, 2, 4, 5], [(3, 16), (7, 32)]))
abjad> f(staff)
\new Staff {
    c'8.
    d'8..
    e'8.
    f'8..
}
abjad> componenttools.get_likely_multiplier_of_components(staff[:]) is None
True
```

Return fraction or none.

componenttools.get_nth_component_in_expr

abjad.tools.componenttools.**get_nth_component_in_expr**(*expr*, *klasses*, *n=0*)

New in version 1.1.1. Get component *n* in the *klasses* of *expr*:

```
abjad> staff = Staff([ ])
abjad> durations = [Duration(n, 16) for n in range(1, 5)]
abjad> notes = notetools.make_notes([0, 2, 4, 5], durations)
abjad> rests = resttools.make_rests(durations)
abjad> leaves = seqtools.interlace_sequences(notes, rests)
abjad> staff.extend(leaves)

abjad> print staff.format
\new Staff {
        c'16
        r16
        d'8
        r8
        e'8.
        r8.
        f'4
        r4
}
```

```
abjad> for n in range(4):
...      componenttools.get_nth_component_in_expr(staff, Note, n)
...
Note("c'16")
Note("d'8")
Note("e'8.")
Note("f'4")

abjad> for n in range(4):
...      componenttools.get_nth_component_in_expr(staff, Rest, n)
...
Rest('r16')
Rest('r8')
Rest('r8.')
Rest('r4')

abjad> componenttools.get_nth_component_in_expr(staff, Staff)
Staff{8}
```

Read right-to-left for negative values of *n*:

```
abjad> for n in range(3, -1, -1):
...      componenttools.get_nth_component_in_expr(staff, Rest, n)
...
Rest('r4')
Rest('r8.')
Rest('r8')
Rest('r16')
```

Return component or none. Changed in version 1.1.2: renamed `iterate.get_nth( )` to `componenttools.get_nth_component_in_expr( )`.

componenttools.get_nth_namesake_from_component

`abjad.tools.componenttools.`**`get_nth_namesake_from_component`**`(`*component*, *n*`)`

New in version 1.1.2. For positive *n*, return namesake to the right of *component*:

```
abjad> t = Staff(macros.scale(4))
abjad> componenttools.get_nth_namesake_from_component(t[1], 1)
Note("e'8")
```

For negative *n*, return namesake to the left of *component*:

```
abjad> t = Staff(macros.scale(4))
abjad> componenttools.get_nth_namesake_from_component(t[1], -1)
Note("c'8")
```

Return *component* when *n* is zero:

```
abjad> t = Staff(macros.scale(4))
abjad> componenttools.get_nth_namesake_from_component(t[1], 0)
Note("d'8")
```

Return component or none.

componenttools.get_parent_and_start_stop_indices_of_components

`abjad.tools.componenttools.`**`get_parent_and_start_stop_indices_of_components`**`(`*components*`)`

New in version 1.1.1. Get parent and start / stop indices of *components*:

```
abjad> t = Staff(macros.scale(6))
abjad> print t.format
\new Staff {
   c'8
   d'8
   e'8
   f'8
   g'8
   a'8
}

abjad> leaves = t[-2:]
abjad> leaves
[Note("g'8"), Note("a'8")]
abjad> componenttools.get_parent_and_start_stop_indices_of_components(leaves)
(Staff{6}, 4, 5)
```

Return parent / start index / stop index triple. Return parent as component or none. Return nonnegative integer start index and nonnegative index stop index. Changed in version 1.1.2: renamed `componenttools.get_with_indices( )` to `componenttools.get_parent_and_start_stop_indices_of_components( )`.

componenttools.get_proper_parentage_of_component

`abjad.tools.componenttools.`**`get_proper_parentage_of_component`**`(`*component*`)`

New in version 1.1.1. Get proper parentage of *component*:

```
abjad> tuplet = tuplettools.FixedDurationTuplet((2, 8), macros.scale(3))
abjad> staff = Staff([tuplet])
abjad> note = staff.leaves[0]
abjad> componenttools.get_proper_parentage_of_component(note)
(FixedDurationTuplet(1/4, [c'8, d'8, e'8]), Staff{1})
```

Return tuple of zero or more components.

componenttools.is_beamable_component

`abjad.tools.componenttools.`**`is_beamable_component`**`(`*expr*`)`

New in version 1.1.1. True when *expr* is a beamable component. Otherwise false:

```
abjad> componenttools.is_beamable_component(Note(13, (1, 16)))
True
```

Return boolean.

componenttools.is_orphan_component

`abjad.tools.componenttools.`**`is_orphan_component`**`(`*component*`)`

New in version 1.1.1. True when *component* has no parent. Otherwise false:

```
abjad> note = Note(0, (1, 4))
abjad> componenttools.is_orphan_component(note)
True
```

Return boolean. Changed in version 1.1.2: renamed `componenttools.component_is_orphan( )` to `componenttools.is_orphan_component( )`.

componenttools.is_well_formed_component

`abjad.tools.componenttools.`**`is_well_formed_component`**`(`*expr*, *allow_empty_containers=True*`)`

New in version 1.1.1. True when *component* is well formed:

```
abjad> staff = Staff(macros.scale(4))
abjad> spannertools.BeamSpanner(staff[:])
BeamSpanner(c'8, d'8, e'8, f'8)
abjad> componenttools.is_well_formed_component(staff)
True
```

Otherwise false:

```
abjad> staff = Staff(macros.scale(4))
abjad> staff[1].duration.written = Duration(1, 4)
abjad> spannertools.BeamSpanner(staff[:])
BeamSpanner(c'8, d'4, e'8, f'8)
abjad> componenttools.is_well_formed_component(staff)
False
```

Beamed quarter notes are not well formed.

Return boolean.

componenttools.iterate_components_backward_in_expr

`abjad.tools.componenttools.`**`iterate_components_backward_in_expr`**`(`*expr*, *klass=<class 'abjad.components._Component._Componen*, *start=0*, *stop=None*`)`

New in version 1.1.1. Iterate components backward in *expr*:

```
abjad> staff = Staff(tuplettools.FixedDurationTuplet((2, 8), notetools.make_repeated_notes(3)) *
abjad> macros.diatonicize(staff)
abjad> f(staff)
\new Staff {
        \times 2/3 {
                c'8
                d'8
                e'8
        }
        \times 2/3 {
                f'8
                g'8
                a'8
        }
}
```

```
abjad> for x in componenttools.iterate_components_backward_in_expr(staff, Note):
...     x
...
Note("a'8")
Note("g'8")
Note("f'8")
Note("e'8")
Note("d'8")
Note("c'8")
```

New in version 1.1.2: optional *start* and *stop* keyword parameters.

```
abjad> for x in componenttools.iterate_components_backward_in_expr(staff, Note, start = 0, stop
...     x
...
Note("a'8")
Note("g'8")
Note("f'8")
Note("e'8")

abjad> for x in componenttools.iterate_components_backward_in_expr(staff, Note, start = 4):
...     x
...
Note("d'8")
Note("c'8")

abjad> for x in componenttools.iterate_components_backward_in_expr(staff, Note, start = 4, stop
...     x
...
Note("d'8")
Note("c'8")
```

This function is thread-agnostic. Changed in version 1.1.2: renamed `iterate.backwards( )` to `componenttools.iterate_components_backward_in_expr( )`.

componenttools.iterate_components_depth_first

`abjad.tools.componenttools.`**`iterate_components_depth_first`**(*component*, *capped=True*, *unique=True*, *forbid=None*, *direction='left'*)

New in version 1.1.1. Iterate components depth-first from *component*.

Todo

Add usage examples.

Changed in version 1.1.2: renamed `iterate.depth_first( )` to `componenttools.iterate_components_depth_first( )`.

componenttools.iterate_components_forward_in_expr

`abjad.tools.componenttools.`**`iterate_components_forward_in_expr`**(*expr*, *klass=<class 'abjad.components._Component._Component*, *start=0*, *stop=None*)

New in version 1.1.1. Iterate components forward in *expr*:

```
abjad> container = Container(Voice(notetools.make_repeated_notes(2)) * 2)
abjad> container.is_parallel = True
abjad> container[0].name = 'voice 1'
abjad> container[1].name = 'vocie 2'
abjad> staff = Staff(container * 2)
abjad> macros.diatonicize(staff)
abjad> f(staff)
\new Staff {
        <<
                \context Voice = "voice 1" {
                        c'8
                        d'8
                }
                \context Voice = "vocie 2" {
                        e'8
                        f'8
                }
        >>
        <<
                \context Voice = "voice 1" {
                        g'8
                        a'8
                }
                \context Voice = "vocie 2" {
                        b'8
                        c''8
                }
        >>
}
abjad> for x in componenttools.iterate_components_forward_in_expr(staff, Note):
...     x
...
Note("c'8")
Note("d'8")
Note("e'8")
Note("f'8")
Note("g'8")
Note("a'8")
Note("b'8")
Note("c''8")
```

New in version 1.1.2: optional *start* and *stop* keyword parameters.

```
abjad> for x in componenttools.iterate_components_forward_in_expr(staff, Note, start = 0, stop =
...     x
...
Note("c'8")
Note("d'8")
Note("e'8")
Note("f'8")
```

```
abjad> for x in componenttools.iterate_components_forward_in_expr(staff, Note, start = 4):
...     x
...
Note("g'8")
Note("a'8")
Note("b'8")
Note("c''8")

abjad> for x in componenttools.iterate_components_forward_in_expr(staff, Note, start = 4, stop =
...     x
...
Note("g'8")
Note("a'8")
```

This function is thread-agnostic. Changed in version 1.1.2: renamed `iterate.naive( )` to `componenttools.iterate_components_forward_in_expr( )`.Changed in version 1.1.2: *klass* now defaults to `_Component`.

componenttools.iterate_namesakes_backward_from_component

abjad.tools.componenttools.**iterate_namesakes_backward_from_component**(*component*, *start=0*, *stop=None*)

New in version 1.1.2. Iterate namesakes backward from *component*:

```
abjad> container = Container(Staff(notetools.make_repeated_notes(2)) * 2)
abjad> container.is_parallel = True
abjad> container[0].name = 'staff 1'
abjad> container[1].name = 'staff 2'
abjad> score = Score([ ])
abjad> score.is_parallel = False
abjad> score.extend(container * 2)
abjad> macros.diatonicize(score)
abjad> print score.format
\new Score {
        <<
                \context Staff = "staff 1" {
                        c'8
                        d'8
                }
                \context Staff = "staff 2" {
                        e'8
                        f'8
                }
        >>
        <<
                \context Staff = "staff 1" {
                        g'8
                        a'8
                }
                \context Staff = "staff 2" {
                        b'8
                        c''8
                }
        >>
}
```

```
abjad> for staff in componenttools.iterate_namesakes_backward_from_component(score[-1][0]):
...     print staff.format
...
\context Staff = "staff 1" {
        g'8
        a'8
}
\context Staff = "staff 1" {
        c'8
        d'8
}
```

Return generator.

componenttools.iterate_namesakes_forward_from_component

abjad.tools.componenttools.**iterate_namesakes_forward_from_component**(*component*, *start=0*, *stop=None*)

New in version 1.1.1. Iterate namesakes forward from *component*:

```
abjad> container = Container(Staff(notetools.make_repeated_notes(2)) * 2)
abjad> container.is_parallel = True
abjad> container[0].name = 'staff 1'
abjad> container[1].name = 'staff 2'
abjad> score = Score([ ])
abjad> score.is_parallel = False
abjad> score.extend(container * 2)
abjad> macros.diatonicize(score)
abjad> print score.format
\new Score {
        <<
                \context Staff = "staff 1" {
                        c'8
                        d'8
                }
                \context Staff = "staff 2" {
                        e'8
                        f'8
                }
        >>
        <<
                \context Staff = "staff 1" {
                        g'8
                        a'8
                }
                \context Staff = "staff 2" {
                        b'8
                        c''8
                }
        >>
}
```

```
abjad> for staff in componenttools.iterate_namesakes_forward_from_component(score[0][0]):
...     print staff.format
...
\context Staff = "staff 1" {
```

```
        c'8
        d'8
}
\context Staff = "staff 1" {
        g'8
        a'8
}
```

Return generator.

componenttools.iterate_timeline_backward_from_component

`abjad.tools.componenttools.`**`iterate_timeline_backward_from_component`**`(`*expr*, *klass=<class 'abjad.components._Leaf._Leaf._Lea*

New in version 1.1.2. Iterate timeline backward from *component*:

```
abjad> score = Score([ ])
abjad> score.append(Staff(notetools.make_repeated_notes(4, Duration(1, 4))))
abjad> score.append(Staff(notetools.make_repeated_notes(4)))
abjad> macros.diatonicize(score)
abjad> f(score)
\new Score <<
        \new Staff {
                c'4
                d'4
                e'4
                f'4
        }
        \new Staff {
                g'8
                a'8
                b'8
                c''8
        }
>>
abjad> for leaf in componenttools.iterate_timeline_backward_from_component(score[1][2]):
...     leaf
...
Note("b'8")
Note("c'4")
Note("a'8")
Note("g'8")
```

Yield components sorted backward by score offset stop time.

Todo

optimize to avoid behind-the-scenes full-score traversal.

---

componenttools.iterate_timeline_backward_in_expr

abjad.tools.componenttools.**iterate_timeline_backward_in_expr**(*expr*, *klass=<class 'abjad.components._Leaf._Leaf._Leaf'>*)

New in version 1.1.2. Iterate timeline backward in *expr*:

```
abjad> score = Score([ ])
abjad> score.append(Staff(notetools.make_repeated_notes(4, Duration(1, 4))))
abjad> score.append(Staff(notetools.make_repeated_notes(4)))
abjad> macros.diatonicize(score)
abjad> f(score)
\new Score <<
        \new Staff {
                c'4
                d'4
                e'4
                f'4
        }
        \new Staff {
                g'8
                a'8
                b'8
                c''8
        }
>>
abjad> for leaf in componenttools.iterate_timeline_backward_in_expr(score):
...     leaf
...
Note("f'4")
Note("e'4")
Note("d'4")
Note("c''8")
Note("b'8")
Note("c'4")
Note("a'8")
Note("g'8")
```

Todo

optimize to avoid behind-the-scenes full-score traversal.

componenttools.iterate_timeline_forward_from_component

abjad.tools.componenttools.**iterate_timeline_forward_from_component**(*expr*, *klass=<class 'abjad.components._Leaf._Leaf._Leaf*

New in version 1.1.2. Iterate timeline forward from *component*:

```
abjad> score = Score([ ])
abjad> score.append(Staff(notetools.make_repeated_notes(4, Duration(1, 4))))
abjad> score.append(Staff(notetools.make_repeated_notes(4)))
abjad> macros.diatonicize(score)
abjad> f(score)
\new Score <<
```

```
        \new Staff {
                c'4
                d'4
                e'4
                f'4
        }
        \new Staff {
                g'8
                a'8
                b'8
                c''8
        }
>>
abjad> for leaf in componenttools.iterate_timeline_forward_from_component(score[1][2]):
...     leaf
...
Note("b'8")
Note("c''8")
Note("e'4")
Note("f'4")
```

Todo

optimize to avoid behind-the-scenes full-score traversal.

---

componenttools.iterate_timeline_forward_in_expr

`abjad.tools.componenttools.`**`iterate_timeline_forward_in_expr`**(*expr*, *klass=<class 'abjad.components._Leaf._Leaf._Leaf'>*)

New in version 1.1.2. Iterate timeline forward in *expr*:

```
abjad> score = Score([ ])
abjad> score.append(Staff(notetools.make_repeated_notes(4, Duration(1, 4))))
abjad> score.append(Staff(notetools.make_repeated_notes(4)))
abjad> macros.diatonicize(score)
abjad> f(score)
\new Score <<
        \new Staff {
                c'4
                d'4
                e'4
                f'4
        }
        \new Staff {
                g'8
                a'8
                b'8
                c''8
        }
>>
abjad> for leaf in componenttools.iterate_timeline_forward_in_expr(score):
...     leaf
...
Note("c'4")
Note("g'8")
```

```
Note("a'8")
Note("d'4")
Note("b'8")
Note("c''8")
Note("e'4")
Note("f'4")
```

Todo

optimize to avoid behind-the-scenes full-score traversal.

---

componenttools.list_badly_formed_components_in_expr

abjad.tools.componenttools.**list_badly_formed_components_in_expr**(*expr*, *allow_empty_containers=True*)

New in version 1.1.1. List badly formed components in *expr*:

```
abjad> staff = Staff(macros.scale(4))
abjad> staff[1].duration.written = Duration(1, 4)
abjad> spannertools.BeamSpanner(staff[:])
BeamSpanner(c'8, d'4, e'8, f'8)
abjad> f(staff)
\new Staff {
        c'8 [
        d'4
        e'8
        f'8 ]
}
abjad> componenttools.list_badly_formed_components_in_expr(staff)
[Note("d'4")]
```

Beamed quarter notes are not well formed.

Return newly created list of zero or more components.

componenttools.list_improper_contents_of_component_that_cross_prolated_offset

abjad.tools.componenttools.**list_improper_contents_of_component_that_cross_prolated_offset**(c

New in version 1.1.2. List improper contents of *component* that cross *prolated_offset*:

```
abjad> staff = Staff(Measure((2, 8), notetools.make_repeated_notes(2)) * 2)
abjad> macros.diatonicize(staff)
abjad> f(staff)
\new Staff {
   {
      \time 2/8
      c'8
      d'8
   }
   {
      \time 2/8
      e'8
      f'8
```

```
    }
}
```

Examples refer to the score above.

No components cross prolated offset `0`:

```
abjad> componenttools.list_improper_contents_of_component_that_cross_prolated_offset(staff, 0)
[]
```

Staff, measure and leaf cross prolated offset `1/16`:

```
abjad> componenttools.list_improper_contents_of_component_that_cross_prolated_offset(staff, Dura
[Staff{2}, Measure(2/8, [c'8, d'8]), Note("c'8")]
```

Staff and measure cross prolated offset `1/8`:

```
abjad> componenttools.list_improper_contents_of_component_that_cross_prolated_offset(staff, Dura
[Staff{2}, Measure(2/8, [c'8, d'8])]
```

Staff crosses prolated offset `1/4`:

```
abjad> componenttools.list_improper_contents_of_component_that_cross_prolated_offset(staff, Dura
[Staff{2}]
```

No components cross prolated offset `99`:

```
abjad> componenttools.list_improper_contents_of_component_that_cross_prolated_offset(staff, 99)
[]
```

Return list.

componenttools.list_leftmost_components_with_prolated_duration_at_most

`abjad.tools.componenttools.`**`list_leftmost_components_with_prolated_duration_at_most`**`(`*component pro- lated_dura*

New in version 1.1.2. List leftmost components in *component* with prolated duration at most *prolated_duration*.

Return tuple of `components[:i]` together with the prolated duration of `components[:i]`:

```
abjad> voice = Voice(macros.scale(4))
abjad> componenttools.list_leftmost_components_with_prolated_duration_at_most(voice[:], Duration
([Note("c'8"), Note("d'8")], Duration(1, 4))
```

Maximize `i` such that the prolated duration of `components[:i]` is no greater than *prolated_duration*.

Input *components* must be thread-contiguous.

Todo

implement `componenttools.list_leftmost_components_with_prolated_duration_at_least()`.

Todo

implement `componenttools.list_rightmost_components_with_prolated_duration_at_most()`.

Todo

implement `componenttools.list_rightmost_components_with_prolated_duration_at_least( )`.

Changed in version 1.1.2: renamed `componenttools.get_le_duration_prolated( )` to `componenttools.list_leftmost_components_with_prolated_duration_at_most( )`.

componenttools.move_component_subtree_to_right_in_immediate_parent_of_component

abjad.tools.componenttools.**move_component_subtree_to_right_in_immediate_parent_of_component**

New in version 1.1.2. Move *component* subtree to right in immediate parent of *component*:

```
abjad> t = Voice(macros.scale(4))
abjad> spannertools.BeamSpanner(t[:2])
BeamSpanner(c'8, d'8)
abjad> spannertools.BeamSpanner(t[2:])
BeamSpanner(e'8, f'8)
abjad> f(t)
\new Voice {
   c'8 [
   d'8 ]
   e'8 [
   f'8 ]
}

abjad> componenttools.move_component_subtree_to_right_in_immediate_parent_of_component(t[1])
abjad> f(t)
\new Voice {
   c'8 [
   e'8 ]
   d'8 [
   f'8 ]
}
```

Return none.

Todo

add `n = 1` keyword to generalize flipped distance.

Todo

make `componenttools.move_component_subtree_to_right_in_immediate_parent_of_component( )` work when spanners attach to children of component:

```
abjad> voice = Voice(tuplettools.FixedDurationTuplet((2, 8), notetools.make_repeated_notes(3)) *
abjad> spannertools.BeamSpanner(voice.leaves[:4])
BeamSpanner(c'8, c'8, c'8, c'8)
abjad> macros.diatonicize(voice)
abjad> componenttools.move_component_subtree_to_right_in_immediate_parent_of_component(voice[0])
abjad> f(voice)
\new Voice {
   \times 2/3 {
```

```
        f'8 ]
        g'8
        a'8
    }
    \times 2/3 {
        c'8 [
        d'8
        e'8
    }
}
abjad> componenttools.is_well_formed_component(voice)
False
```

Preserve spanners. Changed in version 1.1.2: renamed `componenttools.flip( )` to `componenttools.move_component_subtree_to_right_in_immediate_parent_of_component( )`.

componenttools.move_parentage_and_spanners_from_components_to_components

`abjad.tools.componenttools.`**`move_parentage_and_spanners_from_components_to_components`**`(`*donors*, *recipients*`)`

New in version 1.1.1. Move parentage and spanners from *donors* to *recipients*.

Give everything from donors to recipients. Almost exactly the same as container setitem logic. This helper works with orphan donors. Container setitem logic can not work with orphan donors. Return donors. Changed in version 1.1.2: renamed `scoretools.bequeath( )` to `componenttools.move_parentage_and_spanners_from_components_to_components( )`.

componenttools.number_is_between_prolated_start_and_stop_offsets_of_component

`abjad.tools.componenttools.`**`number_is_between_prolated_start_and_stop_offsets_of_component`**`(`

New in version 1.1.2. True when *timepoint* is within the prolated duration of *component*:

```
abjad> staff = Staff(macros.scale(4))
abjad> leaf = staff.leaves[0]
abjad> componenttools.number_is_between_prolated_start_and_stop_offsets_of_component(Duration(1,
True
abjad> componenttools.number_is_between_prolated_start_and_stop_offsets_of_component(Duration(1,
True
```

Otherwise false:

```
abjad> componenttools.number_is_between_prolated_start_and_stop_offsets_of_component(Duration(1,
False
```

Return boolean.

componenttools.number_is_between_start_and_stop_offsets_of_component_in_seconds

abjad.tools.componenttools.**number_is_between_start_and_stop_offsets_of_component_in_seconds**

New in version 1.1.2. True when *timepoint* is within the duration of *component* in seconds:

```
abjad> staff = Staff(macros.scale(4))
abjad> contexttools.TempoMark(Duration(1, 2), 60, target_context = Staff)(staff)
TempoMark(2, 60)(Staff{4})

abjad> leaf = staff.leaves[0]
abjad> componenttools.number_is_between_start_and_stop_offsets_of_component_in_seconds(0.1, leaf
True
abjad> componenttools.number_is_between_start_and_stop_offsets_of_component_in_seconds(0.333, le
True
```

Otherwise false:

```
abjad> componenttools.number_is_between_start_and_stop_offsets_of_component_in_seconds(0.5, staf
False
```

Return boolean.

componenttools.partition_components_cyclically_by_durations_in_seconds_exactly_with_overhang

abjad.tools.componenttools.**partition_components_cyclically_by_durations_in_seconds_exactly_**

New in version 1.1.1. Partition *components* cyclically by *durations_in_seconds* exactly with overhang.

componenttools.partition_components_cyclically_by_durations_in_seconds_exactly_without_overhang

abjad.tools.componenttools.**partition_components_cyclically_by_durations_in_seconds_exactly_**

New in version 1.1.1. Partition *components* cyclically by *durations_in_seconds* exactly without overhang.

componenttools.partition_components_cyclically_by_durations_in_seconds_ge_with_overhang

abjad.tools.componenttools.**partition_components_cyclically_by_durations_in_seconds_ge_with_**

New in version 1.1.1. Partition *components* cyclically by durations in seconds greater than or equal to *durations_in_seconds*, with overhang.

componenttools.partition_components_cyclically_by_durations_in_seconds_ge_without_overhang

abjad.tools.componenttools.**partition_components_cyclically_by_durations_in_seconds_ge_witho**

New in version 1.1.1. Partition *components* cyclically by durations in seconds that are equal to or just greater than *durations_in_seconds*, without overhang.

componenttools.partition_components_cyclically_by_durations_in_seconds_le_with_overhang

abjad.tools.componenttools.**partition_components_cyclically_by_durations_in_seconds_le_with_**

New in version 1.1.1. Partition *components* cyclically by durations in seconds equal to or just less than *durations_in_seconds*, with overhang.

componenttools.partition_components_cyclically_by_durations_in_seconds_le_without_overhang

abjad.tools.componenttools.**partition_components_cyclically_by_durations_in_seconds_le_witho**

New in version 1.1.1. Partition *components* cyclically by durations in seconds that equal or are just less than *durations_in_seconds*, without overhang

componenttools.partition_components_cyclically_by_prolated_durations_exactly_with_overhang

abjad.tools.componenttools.**partition_components_cyclically_by_prolated_durations_exactly_wi**

New in version 1.1.1. Partition *components* cyclically by *prolated_durations* exactly, with overhang.

componenttools.partition_components_cyclically_by_prolated_durations_exactly_without_overhang

abjad.tools.componenttools.**partition_components_cyclically_by_prolated_durations_exactly_wi**

New in version 1.1.1. Partition *components* cyclically by *prolated_durations* exactly, without overhang.

componenttools.partition_components_cyclically_by_prolated_durations_ge_with_overhang

abjad.tools.componenttools.**partition_components_cyclically_by_prolated_durations_ge_with_ov**

New in version 1.1.1. Partition *components* cyclically by *prolated_durations* greater than or equal, with overhang:

```
abjad> staff = Staff(Measure((2, 8), notetools.make_repeated_notes(2)) * 4)
abjad> macros.diatonicize(staff)
abjad> f(staff)
```

```
\new Staff {
   {
      \time 2/8
      c'8
      d'8
   }
   {
      \time 2/8
      e'8
      f'8
   }
   {
      \time 2/8
      g'8
      a'8
   }
   {
      \time 2/8
      b'8
      c''8
   }
}
```

```
abjad> groups = componenttools.partition_components_cyclically_by_prolated_durations_ge_with_ove

abjad> for group in groups:
...     group
...
[Note("c'8"), Note("d'8")]
[Note("e'8")]
[Note("f'8"), Note("g'8")]
[Note("a'8")]
[Note("b'8"), Note("c''8")]
```

Return list of lists.

Note: function works not just on components but on any durated objects including spanners.

componenttools.partition_components_cyclically_by_prolated_durations_ge_without_overhang

abjad.tools.componenttools.**partition_components_cyclically_by_prolated_durations_ge_withou**

New in version 1.1.1. Partition *components* cyclically by prolated durations that equal or are just greater than *prolated_durations*, without overhang.

componenttools.partition_components_cyclically_by_prolated_durations_le_with_overhang

abjad.tools.componenttools.**partition_components_cyclically_by_prolated_durations_le_with_ov**

New in version 1.1.1. Partition *components* cyclically by prolated duration that equal or are just less than *prolated_durations*, with overhang.

componenttools.partition_components_cyclically_by_prolated_durations_le_without_overhang

abjad.tools.componenttools.**partition_components_cyclically_by_prolated_durations_le_without**

New in version 1.1.1. Partition *components* cyclically by prolated durations that equal or are just less than *prolated_durations*, without overhang.

componenttools.partition_components_once_by_durations_in_seconds_exactly_with_overhang

abjad.tools.componenttools.**partition_components_once_by_durations_in_seconds_exactly_with_o**

New in version 1.1.1. Partition *components* once by *durations_in_seconds* exactly, with overhang.

componenttools.partition_components_once_by_durations_in_seconds_exactly_without_overhang

abjad.tools.componenttools.**partition_components_once_by_durations_in_seconds_exactly_withou**

New in version 1.1.1. Partition *components* cyclically by *durations_in_seconds* exactly, without overhang.

componenttools.partition_components_once_by_durations_in_seconds_ge_with_overhang

abjad.tools.componenttools.**partition_components_once_by_durations_in_seconds_ge_with_overha**

New in version 1.1.1. Partition *components* once by durations in seconds that equal or are just greater than *durations_in_seconds*, with overhang.

componenttools.partition_components_once_by_durations_in_seconds_ge_without_overhang

abjad.tools.componenttools.**partition_components_once_by_durations_in_seconds_ge_without_ove**

New in version 1.1.1. Partition *components* once by durations in seconds that equal or are just greater than *durations_in_seconds*, without overhang.

componenttools.partition_components_once_by_durations_in_seconds_le_with_overhang

abjad.tools.componenttools.**partition_components_once_by_durations_in_seconds_le_with_overha**

New in version 1.1.1. Partition *components* once by durations in seconds that equal or are just less than *durations_in_seconds*, with overhang.

componenttools.partition_components_once_by_durations_in_seconds_le_without_overhang

abjad.tools.componenttools.**partition_components_once_by_durations_in_seconds_le_without_ove**

New in version 1.1.1. Partition *components* once by durations in seconds that equal or are just less than *durations_in_seconds*, without overhang.

componenttools.partition_components_once_by_prolated_durations_exactly_with_overhang

abjad.tools.componenttools.**partition_components_once_by_prolated_durations_exactly_with_ove**

New in version 1.1.1. Partition *components* once by *prolated_durations* exactly, with overhang.

componenttools.partition_components_once_by_prolated_durations_exactly_without_overhang

abjad.tools.componenttools.**partition_components_once_by_prolated_durations_exactly_without_**

New in version 1.1.1. Partition *components* once by *prolated_durations* exactly, without overhang.

componenttools.partition_components_once_by_prolated_durations_ge_with_overhang

abjad.tools.componenttools.**partition_components_once_by_prolated_durations_ge_with_overhang**

New in version 1.1.1. Partition *components* cyclically by prolated durations that equal or are just greater than *prolated_durations*, with overhang.

componenttools.partition_components_once_by_prolated_durations_ge_without_overhang

abjad.tools.componenttools.**partition_components_once_by_prolated_durations_ge_without_overh**

New in version 1.1.1. Partition *components* cyclically by prolated durations that equal or are just greater than *prolated_durations*, without overhang.

componenttools.partition_components_once_by_prolated_durations_le_with_overhang

abjad.tools.componenttools.**partition_components_once_by_prolated_durations_le_with_overhang**

New in version 1.1.1. Partition *components* once by prolated durations that equal or are just less than *prolated_durations*, with overhang.

componenttools.partition_components_once_by_prolated_durations_le_without_overhang

abjad.tools.componenttools.**partition_components_once_by_prolated_durations_le_without_overl**

New in version 1.1.1. Partition *components* once by prolated durations that equal or are just less than *prolated_durations*, without overhang.

componenttools.remove_component_subtree_from_score_and_spanners

abjad.tools.componenttools.**remove_component_subtree_from_score_and_spanners**(*components*)

New in version 1.1.1. Remove arbitrary *components* and children of *components* from score and spanners:

```
abjad> score = Voice(notetools.make_repeated_notes(2))
abjad> score.insert(1, Container(notetools.make_repeated_notes(2)))
abjad> macros.diatonicize(score)
abjad> spannertools.BeamSpanner(score.leaves)
BeamSpanner(c'8, d'8, e'8, f'8)
abjad> spannertools.GlissandoSpanner(score.leaves)
GlissandoSpanner(c'8, d'8, e'8, f'8)

abjad> f(score)
\new Voice {
   c'8 [ \glissando
   {
      d'8 \glissando
      e'8 \glissando
   }
   f'8 ]
}
```

Examples refer to the score above.

Remove one leaf from score:

```
abjad> componenttools.remove_component_subtree_from_score_and_spanners(score.leaves[1:2]) # doct
(Note(d', 8),)

abjad> f(score) # doctest: +SKIP
\new Voice {
   c'8 [ \glissando
   {
      e'8 \glissando
   }
   f'8 ]
}
```

Remove contiguous leaves from score:

```
abjad> result = componenttools.remove_component_subtree_from_score_and_spanners(score.leaves[:2]
(Note(c', 8), Note(d', 8))

abjad> f(score) # doctest: +SKIP
\new Voice {
   {
      e'8 [ \glissando
   }
```

```
   f'8 ]
}
```

Remove noncontiguous leaves from score:

```
abjad> componenttools.remove_component_subtree_from_score_and_spanners([score.leaves[0], score.l
[Note(c', 8), Note(e', 8)]

abjad> f(score) # doctest: +SKIP
\new Voice {
   {
      d'8 [ \glissando
   }
   f'8 ]
}
```

Remove container from score:

```
abjad> result = componenttools.remove_component_subtree_from_score_and_spanners(score[1:2])
abjad> result # doctest: +SKIP
[{d'8, e'8}]

abjad> f(score) # doctest: +SKIP
\new Voice {
   c'8 [ \glissando
   f'8 ]
}
```

Withdraw *components* and children of *components* from spanners.

Return either tuple or list of *components* and children of *components*.

Todo

regularize return value of function.

Note: rename to `componenttools.remove_components_from_score_deep( )`.

Changed in version 1.1.2: renamed `componenttools.detach( )` to `componenttools.remove_component_subtree_from_score_and_spanners( )`.

componenttools.replace_components_with_children_of_components

`abjad.tools.componenttools.`**`replace_components_with_children_of_components`**(*components*)

New in version 1.1.1. Remove arbitrary *components* from score but retain children of *components* in score:

```
abjad> staff = Staff(Container(notetools.make_repeated_notes(2)) * 2)
abjad> macros.diatonicize(staff)
abjad> spannertools.SlurSpanner(staff[:])
SlurSpanner({c'8, d'8}, {e'8, f'8})
abjad> spannertools.BeamSpanner(staff.leaves)
BeamSpanner(c'8, d'8, e'8, f'8)

abjad> f(staff)
\new Staff {
   {
```

```
      c'8 [ (
      d'8
   }
   {
      e'8
      f'8 ] )
   }
}
```

```
abjad> componenttools.replace_components_with_children_of_components(staff[0:1])
[{ }]

abjad> f(staff)
\new Staff {
   c'8 [ (
   d'8
   {
      e'8
      f'8 ] )
   }
}
```

Return *components*.

Note: should be renamed to `componenttools.remove_components_from_score_shallow( )`

Changed in version 1.1.2: renamed `componenttools.slip( )` to `componenttools.replace_components_with_children_of_components( )`.

componenttools.report_component_format_contributions_as_string

`abjad.tools.componenttools.`**`report_component_format_contributions_as_string`**`(`*component, verbose=False*`)`

New in version 1.1.1. Report *component* format contributions as string.

Set *verbose* to True or False.

componenttools.report_component_format_contributions_to_screen

`abjad.tools.componenttools.`**`report_component_format_contributions_to_screen`**`(`*component, verbose=False*`)`

New in version 1.1.1. Report *component* format contributions to screen.

Set *verbose* to True or False.

componenttools.split_component_at_prolated_duration_and_do_not_fracture_crossing_spanners

`abjad.tools.componenttools.`**`split_component_at_prolated_duration_and_do_not_fracture_crossi`**

New in version 1.1.1. Split *component* at *prolated_duration* and do not fracture crossing spanners.

Leave spanners untouched.

Return split parts:

```
abjad> t = Staff(Measure((2, 8), notetools.make_repeated_notes(2)) * 2)
abjad> macros.diatonicize(t)
abjad> spannertools.BeamSpanner(t[0])
BeamSpanner(|2/8(2)|)
abjad> spannertools.BeamSpanner(t[1])
BeamSpanner(|2/8(2)|)
abjad> spannertools.SlurSpanner(t.leaves)
SlurSpanner(c'8, d'8, e'8, f'8)
abjad> f(t)
\new Staff {
   {
      \time 2/8
      c'8 [ (
      d'8 ]
   }
   {
      \time 2/8
      e'8 [
      f'8 ] )
   }
}

abjad> halves = componenttools.split_component_at_prolated_duration_and_do_not_fracture_crossing
abjad> f(t)
\new Staff {
   {
      \time 2/8
      c'32 [ (
      c'16.
      d'8 ]
   }
   {
      \time 2/8
      e'8 [
      f'8 ] )
   }
}
```

Works on both leaves and containers. Changed in version 1.1.2: renamed `split.unfractured_at_duration( )` to `componenttools.split_component_at_prolated_duration_a` `)`.

componenttools.split_component_at_prolated_duration_and_fracture_crossing_spanners

abjad.tools.componenttools.**split_component_at_prolated_duration_and_fracture_crossing_spann**

New in version 1.1.1. Split *component* at *prolated_duration* and fracture crossing spanners.

Return split parts:

```
abjad> t = Staff(Measure((2, 8), notetools.make_repeated_notes(2)) * 2)
abjad> macros.diatonicize(t)
```

```
abjad> spannertools.BeamSpanner(t[0])
BeamSpanner(|2/8(2)|)
abjad> spannertools.BeamSpanner(t[1])
BeamSpanner(|2/8(2)|)
abjad> spannertools.SlurSpanner(t.leaves)
SlurSpanner(c'8, d'8, e'8, f'8)
abjad> f(t)
\new Staff {
   {
      \time 2/8
      c'8 [ (
      d'8 ]
   }
   {
      \time 2/8
      e'8 [
      f'8 ] )
   }
}
```

```
halves = componenttools.split_component_at_prolated_duration_and_fracture_crossing_spanners(t.le
\new Staff {
   {
      \time 2/8
      c'32 ( ) [
      c'16. (
      d'8 ]
   }
   {
      \time 2/8
      e'8 [
      f'8 ] )
   }
}
```

Function works on both leaves and containers. Changed in version 1.1.2: renamed `split.fractured_at_duration( )` to `componenttools.split_component_at_prolated_duration_and`
`)`.

componenttools.split_components_cyclically_by_prolated_durations_and_do_not_fracture_crossing_spanners

`abjad.tools.componenttools.`**`split_components_cyclically_by_prolated_durations_and_do_not_fra`**

New in version 1.1.1. Partition *components* cyclically by prolated *durations* and do not fracture spanners:

```
abjad> staff = Staff(Measure((2, 8), notetools.make_repeated_notes(2)) * 2)
abjad> macros.diatonicize(staff)
abjad> spannertools.BeamSpanner(staff[0])
BeamSpanner(|2/8(2)|)
abjad> spannertools.BeamSpanner(staff[1])
BeamSpanner(|2/8(2)|)
abjad> spannertools.SlurSpanner(staff.leaves)
SlurSpanner(c'8, d'8, e'8, f'8)
abjad> f(staff)
```

```
\new Staff {
   {
      \time 2/8
      c'8 [ (
      d'8 ]
   }
   {
      \time 2/8
      e'8 [
      f'8 ] )
   }
}
```

```
abjad> durations = [Duration(3, 32)]
abjad> componenttools.split_components_cyclically_by_prolated_durations_and_do_not_fracture_cros
[[Note("c'16.")], [Note("c'32"), Note("d'16")],
[Note("d'16"), Note("e'32")], [Note("e'16.")], [Note("f'16.")], [Note("f'32")]]
```

```
abjad> f(staff)
\new Staff {
   {
      \time 2/8
      c'16. [ (
      c'32
      d'16
      d'16 ]
   }
   {
      \time 2/8
      e'32 [
      e'16.
      f'16.
      f'32 ] )
   }
}
```

Return list of partitioned components. Changed in version 1.1.2: renamed `partition.cyclic_unfractured_by_durations( )` to `componenttools.split_components_cyclically_by_prolated_durations_and_do_not_fracture_c` `)`.

componenttools.split_components_cyclically_by_prolated_durations_and_fracture_crossing_spanners

`abjad.tools.componenttools.`**`split_components_cyclically_by_prolated_durations_and_fracture_c`**

New in version 1.1.1. Partition *components* cyclically by prolated *durations* and fracture spanners:

```
abjad> staff = Staff(Measure((2, 8), notetools.make_repeated_notes(2)) * 2)
abjad> macros.diatonicize(staff)
abjad> spannertools.BeamSpanner(staff[0])
BeamSpanner(|2/8(2)|)
abjad> spannertools.BeamSpanner(staff[1])
BeamSpanner(|2/8(2)|)
abjad> spannertools.SlurSpanner(staff.leaves)
```

```
SlurSpanner(c'8, d'8, e'8, f'8)
abjad> f(staff)
\new Staff {
   {
      \time 2/8
      c'8 [ (
      d'8 ]
   }
   {
      \time 2/8
      e'8 [
      f'8 ] )
   }
}

abjad> durations = [Duration(3, 32)]
abjad> componenttools.split_components_cyclically_by_prolated_durations_and_fracture_crossing_sp
[[Note("c'16.")], [Note("c'32"), Note("d'16")], [Note("d'16"), Note("e'32")],
[Note("e'16.")], [Note("f'16.")], [Note("f'32")]]

abjad> f(staff)
\new Staff {
   {
      \time 2/8
      c'16. ( ) [
      c'32 (
      d'16 )
      d'16 ] (
   }
   {
      \time 2/8
      e'32 ) [
      e'16. (
      f'16. )
      f'32 ] ( )
   }
}
```

Return list of partitioned components. Changed in version 1.1.2: renamed `partition.cyclic_fractured_by_durations( )` to `componenttools.split_components_cyclically_`
`).`

componenttools.split_components_once_by_prolated_durations_and_do_not_fracture_crossing_spanners

`abjad.tools.componenttools.`**`split_components_once_by_prolated_durations_and_do_not_fracture_`**

New in version 1.1.1. Split *components* once by prolated *durations* and do not fracture crossing spanners:

```
abjad> t = Staff(Container(notetools.make_repeated_notes(2)) * 2)
abjad> macros.diatonicize(t)
abjad> spannertools.BeamSpanner(t[0])
BeamSpanner({c'8, d'8})
abjad> spannertools.BeamSpanner(t[1])
BeamSpanner({e'8, f'8})
```

```
abjad> spannertools.SlurSpanner(t.leaves)
SlurSpanner(c'8, d'8, e'8, f'8)
abjad> f(t)
\new Staff {
   {
      c'8 [ (
      d'8 ]
   }
   {
      e'8 [
      f'8 ] )
   }
}

abjad> durations = [Duration(1, 32), Duration(3, 32), Duration(5, 32)]
abjad> parts = componenttools.split_components_once_by_prolated_durations_and_do_not_fracture_cr

abjad> f(t)
\new Staff {
   {
      c'32 [ (
   }
   {
      c'16.
   }
   {
      d'8 ]
   }
   {
      e'8 [
      f'8 ] )
   }
}
```

Changed in version 1.1.2: renamed `partition.unfractured_by_durations( )` to `componenttools.split_components_once_by_prolated_durations_and_do_not_fracture_crossin` `)`.

componenttools.split_components_once_by_prolated_durations_and_fracture_crossing_spanners

`abjad.tools.componenttools.`**`split_components_once_by_prolated_durations_and_fracture_crossin`**

New in version 1.1.1. Split *components* once by prolated *durations* and fracture crossing spanners:

```
abjad> t = Staff(Measure((2, 8), notetools.make_repeated_notes(2)) * 2)
abjad> macros.diatonicize(t)
abjad> spannertools.BeamSpanner(t[0])
BeamSpanner(|2/8(2)|)
abjad> spannertools.BeamSpanner(t[1])
BeamSpanner(|2/8(2)|)
abjad> spannertools.SlurSpanner(t.leaves)
SlurSpanner(c'8, d'8, e'8, f'8)
abjad> f(t)
\new Staff {
```

```
        {
                \time 2/8
                c'8 [ (
                d'8 ]
        }
        {
                \time 2/8
                e'8 [
                f'8 ] )
        }
}
```

```
abjad> durations = [Duration(1, 32), Duration(3, 32), Duration(5, 32)]
abjad> parts = componenttools.split_components_once_by_prolated_durations_and_fracture_crossing_
abjad> f(t)
\new Staff {
        {
                \time 1/32
                c'32 [ ] ( )
        }
        {
                \time 3/32
                c'16. [ ] ( )
        }
        {
                \time 4/32
                d'8 [ ] (
        }
        {
                \time 2/8
                e'8 [
                f'8 ] )
        }
}
```

Changed in version 1.1.2: renamed `partition.fractured_by_durations( )` to `componenttools.split_components_once_by_prolated_durations_and_fracture_crossing_spanne` `)`.

componenttools.sum_duration_of_components_in_seconds

`abjad.tools.componenttools.`**`sum_duration_of_components_in_seconds`**(*components*)

New in version 1.1.1. Sum duration of *components* in seconds:

```
abjad> tuplet = tuplettools.FixedDurationTuplet((2, 8), macros.scale(3))
abjad> score = Score([Staff([tuplet])])
abjad> contexttools.TempoMark(Duration(1, 4), 48)(score)
TempoMark(4, 48)(Score<<1>>)
abjad> f(score) # doctest: +SKIP
\new Score <<
   \new Staff {
      \times 2/3 {
         \tempo 4=48
         c'8
         d'8
         e'8
      }
```

```
    }
>>

abjad> componenttools.sum_duration_of_components_in_seconds(tuplet[:])
Duration(5, 4)
```

Changed in version 1.1.2: renamed `durtools.sum_seconds( )` to `componenttools.sum_duration_of_components_in_seconds( )`.

componenttools.sum_preprolated_duration_of_components

`abjad.tools.componenttools.`**`sum_preprolated_duration_of_components`**`(`*components*`)`

New in version 1.1.1. Sum preprolated duration of *components*:

```
abjad> tuplet = tuplettools.FixedDurationTuplet((2, 8), macros.scale(3))
abjad> componenttools.sum_preprolated_duration_of_components(tuplet[:])
Duration(3, 8)
```

Return zero on empty iterable:

```
abjad> componenttools.sum_preprolated_duration_of_components([ ])
0
```

Raise contiguity error on nonparent-contiguous *components*:

```
abjad> t = Voice(tuplettools.FixedDurationTuplet((2, 8), notetools.make_repeated_notes(3)) * 2)
abjad> macros.diatonicize(t)
abjad> f(t)
\new Voice {
   \times 2/3 {
      c'8
      d'8
      e'8
   }
   \times 2/3 {
      f'8
      g'8
      a'8
   }
}
abjad> componenttools.sum_preprolated_duration_of_components(t.leaves)
Duration(3, 4)
```

Changed in version 1.1.2: renamed `componenttools.get_duration_preprolated( )` to `componenttools.sum_preprolated_duration_of_components( )`.

componenttools.sum_prolated_duration_of_components

`abjad.tools.componenttools.`**`sum_prolated_duration_of_components`**`(`*components*`)`

New in version 1.1.1. Sum prolated duration of *components*:

```
abjad> tuplet = tuplettools.FixedDurationTuplet((2, 8), macros.scale(3))
abjad> f(tuplet)
\times 2/3 {
   c'8
   d'8
   e'8
```

```
}
abjad> componenttools.sum_prolated_duration_of_components(tuplet[:])
Duration(1, 4)
```

Changed in version 1.1.2: renamed `durtools.sum_prolated( )` to `componenttools.sum_prolated_duration_of_components( )`.

componenttools.tabulate_well_formedness_violations_in_expr

abjad.tools.componenttools.**tabulate_well_formedness_violations_in_expr**(*expr*, *allow_empty_containers=True*)

New in version 1.1.1. Tabulate well-formedness violations in *expr*:

```
abjad> staff = Staff(macros.scale(4))
abjad> staff[1].duration.written = Duration(1, 4)
abjad> spannertools.BeamSpanner(staff[:])
BeamSpanner(c'8, d'4, e'8, f'8)
abjad> f(staff)
\new Staff {
        c'8 [
        d'4
        e'8
        f'8 ]
}
```

```
abjad> componenttools.tabulate_well_formedness_violations_in_expr(staff)
 1 /    4 beamed quarter note
 0 /    1 discontiguous spanner
 0 /    5 duplicate i d
 0 /    1 empty container
 0 /    0 intermarked hairpin
 0 /    0 misdurated measure
 0 /    0 misfilled measure
 0 /    4 mispitched tie
 0 /    4 misrepresented flag
 0 /    5 missing parent
 0 /    0 nested measure
 0 /    0 overlapping beam
 0 /    0 overlapping glissando
 0 /    0 overlapping octavation
 0 /    0 short hairpin
```

Beamed quarter notes are not well formed.

componenttools.yield_components_grouped_by_preprolated_duration

abjad.tools.componenttools.**yield_components_grouped_by_preprolated_duration**(*components*)

New in version 1.1.2. Yield components grouped by preprolated duration:

```
abjad> notes = notetools.make_notes([0], [(1, 4), (1, 4), (1, 8), (1, 16), (1, 16), (1, 16)])
abjad> for x in componenttools.yield_components_grouped_by_preprolated_duration(notes):
...     x
...
(Note("c'4"), Note("c'4"))
```

```
(Note("c'8"),)
(Note("c'16"), Note("c'16"), Note("c'16"))
```

Return generator.

componenttools.yield_components_grouped_by_prolated_duration

abjad.tools.componenttools.**yield_components_grouped_by_prolated_duration**(*components*)

New in version 1.1.2. Yield *component* grouped by prolated duration:

```
abjad> notes = notetools.make_notes([0], [(1, 4), (1, 4), (1, 8), (1, 16), (1, 16), (1, 16)])
abjad> for x in componenttools.yield_components_grouped_by_prolated_duration(notes):
...     x
...
(Note("c'4"), Note("c'4"))
(Note("c'8"),)
(Note("c'16"), Note("c'16"), Note("c'16"))
```

Return generator.

componenttools.yield_groups_of_mixed_klasses_in_sequence

abjad.tools.componenttools.**yield_groups_of_mixed_klasses_in_sequence**(*sequence*, *klasses*)

New in version 1.1.2. Yield groups of mixed *klasses* in *sequence*:

```
abjad> staff = Staff("c'8 d'8 r8 r8 <e' g'>8 <f' a'>8 g'8 a'8 r8 r8 <b' d''>8 <c'' e''>8")

abjad> f(staff)
\new Staff {
   c'8
   d'8
   r8
   r8
   <e' g'>8
   <f' a'>8
   g'8
   a'8
   r8
   r8
   <b' d''>8
   <c'' e''>8
}

abjad> for group in componenttools.yield_groups_of_mixed_klasses_in_sequence(staff, (Note, Chord
...   group
(Note("c'8"), Note("d'8"))
(Chord("<e' g'>8"), Chord("<f' a'>8"), Note("g'8"), Note("a'8"))
(Chord("<b' d''>8"), Chord("<c'' e''>8"))
```

Return generator.

componenttools.yield_topmost_components_grouped_by_type

abjad.tools.componenttools.**yield_topmost_components_grouped_by_type**(*expr*)

New in version 1.1.2. Yield topmost components in *expr* grouped by type:

```
abjad> staff = Staff(leaftools.make_leaves([0, 2, 4, None, None, 5, 7], [(1, 8)]))
abjad> for x in componenttools.yield_topmost_components_grouped_by_type(staff):
...     x
...
(Note("c'8"), Note("d'8"), Note("e'8"))
(Rest('r8'), Rest('r8'))
(Note("f'8"), Note("g'8"))
```

Return generator.

componenttools.yield_topmost_components_of_klass_grouped_by_type

abjad.tools.componenttools.**yield_topmost_components_of_klass_grouped_by_type**(*expr*, *klass*)

New in version 1.1.2. Yield topmost components of *klass* in *expr* grouped by type:

```
abjad> staff = Staff(leaftools.make_leaves([0, 2, 4, None, None, 5, 7], [(1, 8)]))
abjad> for x in componenttools.yield_topmost_components_of_klass_grouped_by_type(staff, Note):
...     x
...
(Note("c'8"), Note("d'8"), Note("e'8"))
(Note("f'8"), Note("g'8"))
```

Return generator.

containertools

containertools.color_contents_of_container

abjad.tools.containertools.**color_contents_of_container**(*container*, *color*)

New in version 1.1.2. Color contents of *container*:

```
abjad> measure = Measure((2, 8), "c'8 d'8")

abjad> containertools.color_contents_of_container(measure, 'red')
Measure(2/8, [c'8, d'8])

abjad> f(measure)
{
        \override Accidental #'color = #red
        \override Beam #'color = #red
        \override Dots #'color = #red
        \override NoteHead #'color = #red
        \override Rest #'color = #red
        \override Stem #'color = #red
        \override TupletBracket #'color = #red
        \override TupletNumber #'color = #red
        \time 2/8
        c'8
        d'8
        \revert Accidental #'color
        \revert Beam #'color
        \revert Dots #'color
        \revert NoteHead #'color
        \revert Rest #'color
        \revert Stem #'color
        \revert TupletBracket #'color
```

```
        \revert TupletNumber #'color
}
```

Return none. Changed in version 1.1.2: renamed `containertools.contents_color( )` to `containertools.color_contents_of_container( )`.

containertools.delete_contents_of_container

`abjad.tools.containertools.`**`delete_contents_of_container`**`(`*container*`)`

Delete contents of *container*:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> spannertools.BeamSpanner(staff.leaves)
BeamSpanner(c'8, d'8, e'8, f'8)

abjad> f(staff)
\new Staff {
   c'8 [
   d'8
   e'8
   f'8 ]
}

abjad> containertools.delete_contents_of_container(staff)
[Note("c'8"), Note("d'8"), Note("e'8"), Note("f'8")]

abjad> f(staff)
\new Staff {
}
```

Return *container* contents. Changed in version 1.1.2: renamed `containertools.contents_delete( )` to `containertools.delete_contents_of_container( )`.

containertools.delete_contents_of_container_starting_at_or_after_prolated_offset

`abjad.tools.containertools.`**`delete_contents_of_container_starting_at_or_after_prolated_offse`**

New in version 1.1.2. Delete contents of *container* starting at or after *prolated_offset*:

```
abjad> staff = Staff(macros.scale(4))
abjad> spannertools.BeamSpanner(staff.leaves)
BeamSpanner(c'8, d'8, e'8, f'8)

abjad> f(staff)
\new Staff {
   c'8 [
   d'8
   e'8
   f'8 ]
}

abjad> containertools.delete_contents_of_container_starting_at_or_after_prolated_offset(staff, D
Staff{1}
```

```
abjad> f(staff)
\new Staff {
   c'8 [ ]
}
```

Return *container*. Changed in version 1.1.2: renamed `containertools.contents_delete_starting_not_before_` `)` to `containertools.delete_contents_of_container_starting_at_or_after_prolated_offset(` `).`

containertools.delete_contents_of_container_starting_before_or_at_prolated_offset

`abjad.tools.containertools.`**`delete_contents_of_container_starting_before_or_at_prolated_offs`**

New in version 1.1.2. Delete contents of *container* starting before or at *prolated_offset*:

```
abjad> staff = Staff(macros.scale(4))
abjad> spannertools.BeamSpanner(staff.leaves)
BeamSpanner(c'8, d'8, e'8, f'8)

abjad> f(staff)
\new Staff {
   c'8 [
   d'8
   e'8
   f'8 ]
}

abjad> containertools.delete_contents_of_container_starting_before_or_at_prolated_offset(staff,
Staff{2}

abjad> f(staff)
\new Staff {
   e'8 [
   f'8 ]
}
```

Return *container*. Changed in version 1.1.2: renamed `containertools.contents_delete_starting_not_after_p` `)` to `containertools.delete_contents_of_container_starting_before_or_at_prolated_offset(` `).`

containertools.delete_contents_of_container_starting_strictly_after_prolated_offset

`abjad.tools.containertools.`**`delete_contents_of_container_starting_strictly_after_prolated_of`**

New in version 1.1.2. Delete contents of *container* starting strictly after *prolated_offset*:

```
abjad> staff = Staff(macros.scale(4))
abjad> spannertools.BeamSpanner(staff.leaves)
BeamSpanner(c'8, d'8, e'8, f'8)

abjad> f(staff)
\new Staff {
   c'8 [
   d'8
```

```
    e'8
    f'8 ]
}

abjad> containertools.delete_contents_of_container_starting_strictly_after_prolated_offset(staff
Staff{2}

abjad> f(staff)
\new Staff {
    c'8 [
    d'8 ]
}
```

Return *container*. Changed in version 1.1.2: renamed `containertools.contents_delete_starting_after_prola` `)` to `containertools.delete_contents_of_container_starting_strictly_after_prolated_offset` `)`.

containertools.delete_contents_of_container_starting_strictly_before_prolated_offset

`abjad.tools.containertools.`**`delete_contents_of_container_starting_strictly_before_prolated_o`**

New in version 1.1.2. Delete contents of *container* contents starting strictly before *prolated_offset*:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> spannertools.BeamSpanner(staff.leaves)
BeamSpanner(c'8, d'8, e'8, f'8)

abjad> f(staff)
\new Staff {
    c'8 [
    d'8
    e'8
    f'8 ]
}

abjad> containertools.delete_contents_of_container_starting_strictly_before_prolated_offset(staf
Staff{3}

abjad> f(staff)
\new Staff {
    d'8 [
    e'8
    f'8 ]
}
```

Return *container*. Changed in version 1.1.2: renamed `containertools.contents_delete_starting_before_prol` `)` to `containertools.delete_contents_of_container_starting_strictly_before_prolated_offse` `)`.

containertools.fuse_like_named_contiguous_containers_in_expr

`abjad.tools.containertools.`**`fuse_like_named_contiguous_containers_in_expr`**`(`*expr*`)`

Fuse like-named contiguous containers in *expr*:

```
abjad> staff = Staff(Voice("c'8 c'8") * 2)
abjad> macros.diatonicize(staff.leaves)
abjad> staff[0].name = 'soprano'
abjad> staff[1].name = 'soprano'

abjad> f(staff)
\new Staff {
   \context Voice = "soprano" {
      c'8
      d'8
   }
   \context Voice = "soprano" {
      e'8
      f'8
   }
}

abjad> containertools.fuse_like_named_contiguous_containers_in_expr(staff)
Staff{1}

abjad> f(staff)
\new Staff {
   \context Voice = "soprano" {
      c'8
      d'8
      e'8
      f'8
   }
}
```

Return *expr*. Changed in version 1.1.2: renamed `fuse.containers_by_reference( )` to `containertools.fuse_like_named_contiguous_containers_in_expr( )`.

containertools.get_element_starting_at_exactly_prolated_offset

`abjad.tools.containertools.`**`get_element_starting_at_exactly_prolated_offset`**(*container*, *prolated_offset*)

New in version 1.1.2. Get *container* element starting at exactly *prolated_offset*:

```
abjad> voice = Voice("c'8 d'8 e'8 f'8 g'8 a'8 b'8 c''8")

abjad> containertools.get_element_starting_at_exactly_prolated_offset(voice, Duration(6, 8))
Note("b'8")
```

Raise missing component error when no *container* element starts at exactly *prolated_offset*. Changed in version 1.1.2: renamed `containertools.get_element_starting_at_prolated_offset( )` to `containertools.get_element_starting_at_exactly_prolated_offset( )`.

containertools.get_first_container_in_improper_parentage_of_component

`abjad.tools.containertools.`**`get_first_container_in_improper_parentage_of_component`**(*component*)

New in version 1.1.2. Get first container in improper parentage of *component*:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
```

```
abjad> f(staff)
\new Staff {
   c'8
   d'8
   e'8
   f'8
}

abjad> containertools.get_first_container_in_improper_parentage_of_component(staff[1])
Staff{4}
```

Return container or none.

containertools.get_first_container_in_proper_parentage_of_component

abjad.tools.containertools.**get_first_container_in_proper_parentage_of_component**(*component*)
New in version 1.1.2. Get first container in proper parentage of *component*:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")

abjad> f(staff)
\new Staff {
   c'8
   d'8
   e'8
   f'8
}

abjad> containertools.get_first_container_in_proper_parentage_of_component(staff[1])
Staff{4}
```

Return container or none.

containertools.get_first_element_starting_at_or_after_prolated_offset

abjad.tools.containertools.**get_first_element_starting_at_or_after_prolated_offset**(*container*, *prolated_offset*)
New in version 1.1.2. Get first *container* element starting at or after *prolated_offset*:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")

abjad> containertools.get_first_element_starting_at_or_after_prolated_offset(staff, Duration(1,
Note("d'8")
```

Return component.

Return none when no *container* element starts at or after *prolated_offset*. Changed in version 1.1.2: renamed `containertools.get_leftmost_element_starting_not_before_prolated_offset( )` to `containertools.get_first_element_starting_at_or_after_prolated_offset( )`.

containertools.get_first_element_starting_before_or_at_prolated_offset

abjad.tools.containertools.**get_first_element_starting_before_or_at_prolated_offset**(*container*, *pro-lated_offse*

New in version 1.1.2. Get first *container* element starting before or at *prolated_offset*:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
```

```
abjad> containertools.get_first_element_starting_before_or_at_prolated_offset(staff, Duration(1,
Note("d'8")
```

Return component.

Return none when no *container* element starts before or at *prolated_offset*. Changed in version 1.1.2: renamed `containertools.get_rightmost_element_starting_not_after_prolated_offset( )` to `containertools.get_first_element_starting_before_or_at_prolated_offset( )`.

containertools.get_first_element_starting_strictly_after_prolated_offset

abjad.tools.containertools.**get_first_element_starting_strictly_after_prolated_offset**(*containe* *pro-lated_of*

New in version 1.1.2. Get first *container* element starting strictly after *prolated_offset*:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
```

```
abjad> containertools.get_first_element_starting_strictly_after_prolated_offset(staff, Duration(
Note("e'8")
```

Return component.

Return none when no *container* element starts strictly after *prolated_offset*. Changed in version 1.1.2: renamed `containertools.get_leftmost_element_starting_after_prolated_offset( )` to `containertools.get_first_element_starting_strictly_after_prolated_offset( )`.

containertools.get_first_element_starting_strictly_before_prolated_offset

abjad.tools.containertools.**get_first_element_starting_strictly_before_prolated_offset**(*contai* *pro-lated_*

New in version 1.1.2. Get first *container* element starting strictly before *prolated_offset*:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
```

```
abjad> containertools.get_first_element_starting_strictly_before_prolated_offset(staff, Duration
Note("c'8")
```

Return component.

Return none when *container* element starts stirctly before *prolated_offset*. Changed in version 1.1.2: renamed `containertools.get_rightmost_element_starting_before_prolated_offset( )` to `containertools.get_first_element_starting_strictly_before_prolated_offset( )`.

containertools.insert_component_and_do_not_fracture_crossing_spanners

`abjad.tools.containertools.`**`insert_component_and_do_not_fracture_crossing_spanners`**`(`*container*, *i*, *component*`)`

New in version 1.1.2. Insert *component* into *container* at index *i* and do not fracture crossing spanners:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> spannertools.BeamSpanner(staff.leaves)
BeamSpanner(c'8, d'8, e'8, f'8)

abjad> f(staff)
\new Staff {
   c'8 [
   d'8
   e'8
   f'8 ]
}

abjad> containertools.insert_component_and_do_not_fracture_crossing_spanners(staff, 1, Note("cs'
Staff{5}

abjad> f(staff)
\new Staff {
   c'8 [
   cs'8
   d'8
   e'8
   f'8 ]
}
```

Return *container*. Changed in version 1.1.2: renamed `containertools.insert_and_do_not_fracture( )` to `containertools.insert_component_and_do_not_fracture_crossing_spanners( )`.

containertools.insert_component_and_fracture_crossing_spanners

`abjad.tools.containertools.`**`insert_component_and_fracture_crossing_spanners`**`(`*container*, *i*, *component*`)`

Insert *component* into *container* at index *i* and fracture spanners:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> spannertools.BeamSpanner(staff.leaves)
BeamSpanner(c'8, d'8, e'8, f'8)

abjad> f(staff)
\new Staff {
   c'8 [
   d'8
   e'8
   f'8 ]
}
```

```
abjad> containertools.insert_component_and_fracture_crossing_spanners(staff, 1, Rest((1, 8)))
[(BeamSpanner(c'8, d'8, e'8, f'8), BeamSpanner(c'8), BeamSpanner(d'8, e'8, f'8)), (BeamSpanner(d

abjad> f(staff)
\new Staff {
   c'8 [ ]
   r8
   d'8 [
   e'8
   f'8 ]
}
```

Return list of fractured spanners. Changed in version 1.1.2: renamed `containertools.insert_and_fracture( )` to `containertools.insert_component_and_fracture_cro` `).`

containertools.iterate_containers_backward_in_expr

`abjad.tools.containertools.`**`iterate_containers_backward_in_expr`**`(`*expr*, *start=0*, *stop=None*`)`

New in version 1.1.2. Iterate containers backward in *expr*:

```
abjad> staff = Staff([Voice("c'8 d'8"), Voice("e'8 f'8 g'8")])
abjad> Tuplet((2, 3), staff[1][:])
Tuplet(2/3, [e'8, f'8, g'8])
abjad> staff.is_parallel = True

abjad> f(staff)
\new Staff <<
   \new Voice {
      c'8
      d'8
   }
   \new Voice {
      \times 2/3 {
         e'8
         f'8
         g'8
      }
   }
>>

abjad> for x in containertools.iterate_containers_backward_in_expr(staff):
...    x
Staff<<2>>
Voice{1}
Tuplet(2/3, [e'8, f'8, g'8])
Voice{2}
```

Ignore threads.

Return generator.

containertools.iterate_containers_forward_in_expr

abjad.tools.containertools.**iterate_containers_forward_in_expr**(*expr*, *start=0*, *stop=None*)

New in version 1.1.2. Iterate containers forward in *expr*:

```
abjad> staff = Staff([Voice("c'8 d'8"), Voice("e'8 f'8 g'8")])
abjad> Tuplet((2, 3), staff[1][:])
Tuplet(2/3, [e'8, f'8, g'8])
abjad> staff.is_parallel = True

abjad> f(staff)
\new Staff <<
   \new Voice {
      c'8
      d'8
   }
   \new Voice {
      \times 2/3 {
         e'8
         f'8
         g'8
      }
   }
>>

abjad> for x in containertools.iterate_containers_forward_in_expr(staff):
...    x
Staff<<2>>
Voice{2}
Voice{1}
Tuplet(2/3, [e'8, f'8, g'8])
```

Ignore threads.

Return generator.

containertools.move_parentage_children_and_spanners_from_components_to_empty_container

abjad.tools.containertools.**move_parentage_children_and_spanners_from_components_to_empty_co**

Move parentage, children and spanners from *components* to empty *container*:

```
abjad> voice = Voice(Container("c'8 c'8") * 3)
abjad> macros.diatonicize(voice)
abjad> spannertools.BeamSpanner(voice.leaves)
BeamSpanner(c'8, d'8, e'8, f'8, g'8, a'8)

abjad> f(voice)
\new Voice {
   {
      c'8 [
      d'8
   }
   {
```

```
      e'8
      f'8
   }
   {
      g'8
      a'8 ]
   }
}
```

```
abjad> tuplet = Tuplet((3, 4), [ ])
abjad> containertools.move_parentage_children_and_spanners_from_components_to_empty_container(vo
```

```
abjad> f(voice)
\new Voice {
   \fraction \times 3/4 {
      c'8 [
      d'8
      e'8
      f'8
   }
   {
      g'8
      a'8 ]
   }
}
```

Return none. Changed in version 1.1.2: renamed `scoretools.donate( )` to `containertools.move_parentage_children_and_spanners_from_components_to_empty_container( )`.

containertools.remove_empty_containers_in_expr

`abjad.tools.containertools.`**`remove_empty_containers_in_expr`**`(`*expr*`)`

Remove empty containers in *expr*:

```
abjad> staff = Staff(Container(notetools.make_repeated_notes(2)) * 4)
abjad> macros.diatonicize(staff.leaves)
abjad> spannertools.BeamSpanner(staff[:])
BeamSpanner({c'8, d'8}, {e'8, f'8}, {g'8, a'8}, {b'8, c''8})
abjad> containertools.delete_contents_of_container(staff[1])
[Note("e'8"), Note("f'8")]
abjad> containertools.delete_contents_of_container(staff[-1])
[Note("b'8"), Note("c''8")]
```

```
abjad> f(staff)
\new Staff {
   {
      c'8 [
      d'8
   }
   {
   }
   {
      g'8
      a'8 ]
   }
   {
```

```
      }
   }

   abjad> containertools.remove_empty_containers_in_expr(staff)

   abjad> f(staff)
   \new Staff {
      {
         c'8 [
         d'8
      }
      {
         g'8
         a'8 ]
      }
   }
```

Return none. Changed in version 1.1.2: renamed `containertools.remove_empty( )` to `containertools.remove_empty_containers_in_expr( )`.

containertools.repeat_contents_of_container

`abjad.tools.containertools.`**`repeat_contents_of_container`**`(`*container*, *total=2*`)`

New in version 1.1.1. Repeat contents of *container*:

```
abjad> staff = Staff("c'8 d'8")
abjad> spannertools.BeamSpanner(staff.leaves)
BeamSpanner(c'8, d'8)

abjad> f(staff)
\new Staff {
   c'8 [
   d'8 ]
}

abjad> containertools.repeat_contents_of_container(staff, 3)
Staff{6}

abjad> f(staff)
\new Staff {
   c'8 [
   d'8 ]
   c'8 [
   d'8 ]
   c'8 [
   d'8 ]
}
```

Leave *container* unchanged when *total* is `1`.

Empty *container* when *total* is `0`.

Return *container*. Changed in version 1.1.2: renamed `containertools.contents_multiply( )` to `containertools.repeat_contents_of_container( )`.

containertools.repeat_last_n_elements_of_container

abjad.tools.containertools.**repeat_last_n_elements_of_container**(*container*, *n=1*, *total=2*)

New in version 1.1.1. Repeat last *n* elements of *container*:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> spannertools.BeamSpanner(staff.leaves)
BeamSpanner(c'8, d'8, e'8, f'8)

abjad> f(staff)
\new Staff {
   c'8 [
   d'8
   e'8
   f'8 ]
}

abjad> containertools.repeat_last_n_elements_of_container(staff, n = 2, total = 3)
Staff{8}

abjad> f(staff)
\new Staff {
   c'8 [
   d'8
   e'8
   f'8 ]
   e'8 [
   f'8 ]
   e'8 [
   f'8 ]
}
```

Return *container*. Changed in version 1.1.2: renamed `containertools.extend_cyclic( )` to `containertools.repeat_last_n_elements_of_container( )`.

containertools.replace_contents_of_target_container_with_contents_of_source_container

abjad.tools.containertools.**replace_contents_of_target_container_with_contents_of_source_co**

New in version 1.1.2. Replace contents of *target_container* with contents of *source_container*:

```
abjad> staff = Staff(Tuplet((2, 3), "c'8 d'8 e'8") * 3)
abjad> macros.diatonicize(staff)
abjad> spannertools.BeamSpanner(staff.leaves)
BeamSpanner(c'8, d'8, ... [5] ..., c''8, d''8)

abjad> f(staff)
\new Staff {
   \times 2/3 {
      c'8 [
      d'8
      e'8
   }
   \times 2/3 {
      f'8
      g'8
      a'8
```

```
    }
    \times 2/3 {
       b'8
       c''8
       d''8 ]
    }
}

abjad> container = Container(macros.scale(3))
abjad> spannertools.SlurSpanner(container.leaves)
SlurSpanner(c'8, d'8, e'8)

abjad> f(container)
{
   c'8 (
   d'8
   e'8 )
}

abjad> containertools.replace_contents_of_target_container_with_contents_of_source_container(sta
Tuplet(2/3, [c'8, d'8, e'8])

abjad> f(staff)
\new Staff {
   \times 2/3 {
      c'8 [
      d'8
      e'8
   }
   \times 2/3 {
      c'8 (
      d'8
      e'8 )
   }
   \times 2/3 {
      b'8
      c''8
      d''8 ]
   }
}
```

Leave *source_container* empty:

```
abjad> container
{ }
```

Return *target_container*.

containertools.replace_larger_left_half_of_elements_in_container_with_big_endian_rests

abjad.tools.containertools.**replace_larger_left_half_of_elements_in_container_with_big_endia**

New in version 1.1.2. Replace larger left half of elements in *container* with big-endian rests:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8 g'8 a'8 b'8 c''8 d''8 e''8")

abjad> f(staff)
\new Staff {
```

```
      c'8
      d'8
      e'8
      f'8
      g'8
      a'8
      b'8
      c''8
      d''8
      e''8
   }

abjad> containertools.replace_larger_left_half_of_elements_in_container_with_big_endian_rests(st
Staff{7}

abjad> f(staff)
\new Staff {
      r2
      r8
      a'8
      b'8
      c''8
      d''8
      e''8
   }
```

Return *container*.

containertools.replace_larger_left_half_of_elements_in_container_with_little_endian_rests

abjad.tools.containertools.**replace_larger_left_half_of_elements_in_container_with_little_er**

New in version 1.1.2. Replace larger left half of elements in *container* with little-endian rests:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8 g'8 a'8 b'8 c''8 d''8 e''8")

abjad> f(staff)
\new Staff {
      c'8
      d'8
      e'8
      f'8
      g'8
      a'8
      b'8
      c''8
      d''8
      e''8
   }

abjad> containertools.replace_larger_left_half_of_elements_in_container_with_little_endian_rests
Staff{7}

abjad> f(staff)
\new Staff {
      r8
      r2
      a'8
```

```
      b'8
      c''8
      d''8
      e''8
   }
```

Return *container*.

containertools.replace_larger_right_half_of_elements_in_container_with_big_endian_rests

abjad.tools.containertools.**replace_larger_right_half_of_elements_in_container_with_big_endi**

New in version 1.1.2. Replace larger right half of elements in *container* with big-endian rests:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8 g'8 a'8 b'8 c''8 d''8 e''8")

abjad> f(staff)
\new Staff {
   c'8
   d'8
   e'8
   f'8
   g'8
   a'8
   b'8
   c''8
   d''8
   e''8
}

abjad> containertools.replace_larger_right_half_of_elements_in_container_with_big_endian_rests(s
Staff{7}

abjad> f(staff)
\new Staff {
   c'8
   d'8
   e'8
   f'8
   g'8
   r2
   r8
}
```

Return *container*.

containertools.replace_larger_right_half_of_elements_in_container_with_little_endian_rests

abjad.tools.containertools.**replace_larger_right_half_of_elements_in_container_with_little_e**

New in version 1.1.2. Replace larger right half of elements in *container* with little-endian rests:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8 g'8 a'8 b'8 c''8 d''8 e''8")

abjad> f(staff)
\new Staff {
   c'8
   d'8
```

```
      e'8
      f'8
      g'8
      a'8
      b'8
      c''8
      d''8
      e''8
   }

   abjad> containertools.replace_larger_right_half_of_elements_in_container_with_little_endian_rest
   Staff{7}

   abjad> f(staff)
   \new Staff {
      c'8
      d'8
      e'8
      f'8
      g'8
      r8
      r2
   }
```

Return *container*.

containertools.replace_n_edge_elements_in_container_with_big_endian_rests

abjad.tools.containertools.**replace_n_edge_elements_in_container_with_big_endian_rests**(*contai n*)

New in version 1.1.2. Replace *n* edge elements in *container* with big-endian rests:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8 g'8 a'8")

abjad> f(staff)
\new Staff {
   c'8
   d'8
   e'8
   f'8
   g'8
   a'8
}

abjad> containertools.replace_n_edge_elements_in_container_with_big_endian_rests(staff, -5)
Staff{3}

abjad> f(staff)
\new Staff {
   c'8
   r2
   r8
}
```

Return *container*. Changed in version 1.1.2: renamed `containertools.replace_first_n_elements_in_containe )` to `containertools.replace_n_edge_elements_in_container_with_big_endian_rests( )`.

containertools.replace_n_edge_elements_in_container_with_little_endian_rests

abjad.tools.containertools.**replace_n_edge_elements_in_container_with_little_endian_rests**(*co*
n)

New in version 1.1.2. Replace *n* edge elements in *container* with little-endian rests:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8 g'8 a'8")

abjad> f(staff)
\new Staff {
   c'8
   d'8
   e'8
   f'8
   g'8
   a'8
}

abjad> containertools.replace_n_edge_elements_in_container_with_little_endian_rests(staff, -5)
Staff{3}

abjad> f(staff)
\new Staff {
   c'8
   r8
   r2
}
```

Return *container*. Changed in version 1.1.2: renamed `containertools.replace_first_n_elements_in_containe` `)` to `containertools.replace_n_edge_elements_in_container_with_little_endian_rests(` `).`

containertools.replace_n_edge_elements_in_container_with_rests

abjad.tools.containertools.**replace_n_edge_elements_in_container_with_rests**(*container*, *n*)

New in version 1.1.2. Replace first *n* elements in *container* with big-endian rests:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8 g'8 a'8")

abjad> f(staff)
\new Staff {
   c'8
   d'8
   e'8
   f'8
   g'8
   a'8
}

abjad> containertools.replace_n_edge_elements_in_container_with_rests(staff, 5)
Staff{3}

abjad> f(staff)
\new Staff {
   r2
   r8
```

```
   a'8
}
```

Replace last *n* elements in *container* with little-endian rests:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8 g'8 a'8")

abjad> f(staff)
\new Staff {
   c'8
   d'8
   e'8
   f'8
   g'8
   a'8
}

abjad> containertools.replace_n_edge_elements_in_container_with_rests(staff, -5)
Staff{3}

abjad> f(staff)
\new Staff {
   c'8
   r8
   r2
}
```

Return *container*. Changed in version 1.1.2: renamed `containertools.replace_first_n_elements_in_containe` `)` to `containertools.replace_n_edge_elements_in_container_with_rests( )`.

containertools.replace_smaller_left_half_of_elements_in_container_with_big_endian_rests

`abjad.tools.containertools.`**`replace_smaller_left_half_of_elements_in_container_with_big_endi`**

New in version 1.1.2. Replace smaller left half of elements in *container* with big-endian rests:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8 g'8 a'8 b'8 c''8 d''8 e''8")

abjad> f(staff)
\new Staff {
   c'8
   d'8
   e'8
   f'8
   g'8
   a'8
   b'8
   c''8
   d''8
   e''8
}

abjad> containertools.replace_smaller_left_half_of_elements_in_container_with_big_endian_rests(s
Staff{7}

abjad> f(staff)
\new Staff {
   r2
```

```
      r8
      a'8
      b'8
      c''8
      d''8
      e''8
   }
```

Return *container*.

containertools.replace_smaller_left_half_of_elements_in_container_with_little_endian_rests

abjad.tools.containertools.**replace_smaller_left_half_of_elements_in_container_with_little_e**

New in version 1.1.2. Replace smaller left half of elements in *container* with little-endian rests:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8 g'8 a'8 b'8 c''8 d''8 e''8")

abjad> f(staff)
\new Staff {
   c'8
   d'8
   e'8
   f'8
   g'8
   a'8
   b'8
   c''8
   d''8
   e''8
}

abjad> containertools.replace_smaller_left_half_of_elements_in_container_with_little_endian_rest
Staff{7}

abjad> f(staff)
\new Staff {
   r8
   r2
   a'8
   b'8
   c''8
   d''8
   e''8
}
```

Return *container*.

containertools.replace_smaller_right_half_of_elements_in_container_with_big_endian_rests

abjad.tools.containertools.**replace_smaller_right_half_of_elements_in_container_with_big_enc**

New in version 1.1.2. Relace smaller right half of elements in *container* with big-endian rests:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8 g'8 a'8 b'8 c''8 d''8 e''8")

abjad> f(staff)
\new Staff {
```

```
      c'8
      d'8
      e'8
      f'8
      g'8
      a'8
      b'8
      c''8
      d''8
      e''8
   }

   abjad> containertools.replace_smaller_right_half_of_elements_in_container_with_big_endian_rests(
   Staff{7}

   abjad> f(staff)
   \new Staff {
      c'8
      d'8
      e'8
      f'8
      g'8
      r2
      r8
   }
```

Return *container*.

containertools.replace_smaller_right_half_of_elements_in_container_with_little_endian_rests

abjad.tools.containertools.**replace_smaller_right_half_of_elements_in_container_with_little_**

New in version 1.1.2. Replace smaller right half of elements in *container* with little-endian rests:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8 g'8 a'8 b'8 c''8 d''8 e''8")

abjad> f(staff)
\new Staff {
   c'8
   d'8
   e'8
   f'8
   g'8
   a'8
   b'8
   c''8
   d''8
   e''8
}

abjad> containertools.replace_smaller_right_half_of_elements_in_container_with_little_endian_res
Staff{7}

abjad> f(staff)
\new Staff {
   c'8
   d'8
   e'8
```

```
      f'8
      g'8
      r8
      r2
   }
```

Return *container*.

containertools.report_container_modifications_as_string

abjad.tools.containertools.**report_container_modifications_as_string**(*container*)
Report *container* modifications as string:

```
abjad> container = Container("c'8 d'8 e'8 f'8")
abjad> container.override.note_head.color = 'red'
abjad> container.override.note_head.style = 'harmonic'

abjad> f(container)
{
   \override NoteHead #'color = #red
   \override NoteHead #'style = #'harmonic
   c'8
   d'8
   e'8
   f'8
   \revert NoteHead #'color
   \revert NoteHead #'style
}

abjad> string = containertools.report_container_modifications_as_string(container)

abjad> print string # doctest: +SKIP
{
   \override NoteHead #'color = #red
   \override NoteHead #'style = #'harmonic

   %%% 4 components omitted %%%

   \revert NoteHead #'color
   \revert NoteHead #'style
}
```

Return string.

containertools.report_container_modifications_to_screen

abjad.tools.containertools.**report_container_modifications_to_screen**(*container*)
Report *container* modifications to screen:

```
abjad> container = Container("c'8 d'8 e'8 f'8")
abjad> container.override.note_head.color = 'red'
abjad> container.override.note_head.style = 'harmonic'

abjad> f(container)
{
   \override NoteHead #'color = #red
```

```
    \override NoteHead #'style = #'harmonic
    c'8
    d'8
    e'8
    f'8
    \revert NoteHead #'color
    \revert NoteHead #'style
}

abjad> containertools.report_container_modifications_to_screen(container) # doctest: +SKIP
{
    \override NoteHead #'color = #red
    \override NoteHead #'style = #'harmonic

    %%% 4 components omitted %%%

    \revert NoteHead #'color
    \revert NoteHead #'style
}
```

Return none.

containertools.reverse_contents_of_container

`abjad.tools.containertools.`**`reverse_contents_of_container`**`(`*container*`)`

New in version 1.1.1. Reverse contents of *container*:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> spannertools.BeamSpanner(staff.leaves[:2])
BeamSpanner(c'8, d'8)
abjad> spannertools.SlurSpanner(staff.leaves[2:])
SlurSpanner(e'8, f'8)

abjad> f(staff)
\new Staff {
    c'8 [
    d'8 ]
    e'8 (
    f'8 )
}

abjad> containertools.reverse_contents_of_container(staff)
Staff{4}

abjad> f(staff) # doctest: +SKIP
\new Staff {
    f'8 (
    e'8 )
    d'8 [
    c'8 ]
}
```

Return *container*. Changed in version 1.1.2: renamed `containertools.contents_reverse( )` to `containertools.reverse_contents_of_container( )`.

containertools.scale_contents_of_container

abjad.tools.containertools.**scale_contents_of_container**(*container*, *multiplier*)

New in version 1.1.1. Scale contents of *container* by dot *multiplier*:

```
abjad> staff = Staff("c'8 d'8")
abjad> spannertools.BeamSpanner(staff.leaves)
BeamSpanner(c'8, d'8)

abjad> f(staff)
\new Staff {
   c'8 [
   d'8 ]
}

abjad> containertools.scale_contents_of_container(staff, Duration(3, 2))
Staff{2}

abjad> f(staff)
\new Staff {
   c'8. [
   d'8. ]
}
```

Scale contents of *container* by tie *multiplier*:

```
abjad> staff = Staff("c'8 d'8")
abjad> spannertools.BeamSpanner(staff.leaves)
BeamSpanner(c'8, d'8)

abjad> f(staff)
\new Staff {
   c'8 [
   d'8 ]
}

abjad> containertools.scale_contents_of_container(staff, Duration(5, 4))
Staff{4}

abjad> f(staff)
\new Staff {
   c'8 [ ~
   c'32
   d'8 ~
   d'32 ]
}
```

Scale contents of *container* by nonbinary *multiplier*:

```
abjad> staff = Staff("c'8 d'8")
abjad> spannertools.BeamSpanner(staff.leaves)
BeamSpanner(c'8, d'8)

abjad> f(staff)
\new Staff {
   c'8 [
   d'8 ]
}
```

```
abjad> containertools.scale_contents_of_container(staff, Duration(4, 3))
Staff{2}

abjad> f(staff)
\new Staff {
   \times 2/3 {
      c'4 [
   }
   \times 2/3 {
      d'4 ]
   }
}
```

Return *container*. Changed in version 1.1.2: renamed `containertools.contents_scale( )` to `containertools.scale_contents_of_container( )`.

containertools.set_container_multiplier

`abjad.tools.containertools.`**`set_container_multiplier`**(*container*, *multiplier*)

Set *container multiplier*:

```
abjad> tuplet = tuplettools.FixedDurationTuplet((2, 8), "c'8 d'8 e'8")

abjad> f(tuplet)
\times 2/3 {
   c'8
   d'8
   e'8
}

abjad> containertools.set_container_multiplier(tuplet, Duration(3, 4))

abjad> f(tuplet)
\fraction \times 3/4 {
   c'8
   d'8
   e'8
}
```

Return none. Changed in version 1.1.2: renamed `containertools.multiplier_set( )` to `containertools.set_container_multiplier( )`.

containertools.split_container_at_index_and_do_not_fracture_crossing_spanners

`abjad.tools.containertools.`**`split_container_at_index_and_do_not_fracture_crossing_spanners`**(

Split *container* at *index* and do not fracture crossing spanners:

```
abjad> voice = Voice(Measure((3, 8), "c'8 c'8 c'8") * 2)
abjad> macros.diatonicize(voice)
abjad> beam = spannertools.BeamSpanner(voice[:])

abjad> f(voice)
\new Voice {
   {
```

```
         \time 3/8
         c'8 [
         d'8
         e'8
      }
      {
         \time 3/8
         f'8
         g'8
         a'8 ]
      }
   }
```

```
abjad> containertools.split_container_at_index_and_do_not_fracture_crossing_spanners(voice[1], 1
(Measure(1/8, [f'8]), Measure(2/8, [g'8, a'8]))
```

```
abjad> f(voice)
\new Voice {
   {
      \time 3/8
      c'8 [
      d'8
      e'8
   }
   {
      \time 1/8
      f'8
   }
   {
      \time 2/8
      g'8
      a'8 ]
   }
}
```

Leave spanners and leaves untouched.

Resize resizable containers.

Preserve container multiplier.

Preserve meter denominator.

Return split parts. Changed in version 1.1.2: renamed `split.unfractured_at_index( )` to `containertools.split_container_at_index_and_do_not_fracture_crossing_spanners( )`.

containertools.split_container_at_index_and_fracture_crossing_spanners

`abjad.tools.containertools.`**`split_container_at_index_and_fracture_crossing_spanners`**(*container*, *index*)

Split *container* at *index* and fracture crossing spanners:

```
abjad> voice = Voice(tuplettools.FixedDurationTuplet((2, 8), "c'8 c'8 c'8") * 2)
abjad> tuplet = voice[1]
abjad> macros.diatonicize(voice)
abjad> beam = spannertools.BeamSpanner(voice[:])
```

```
abjad> f(voice)
\new Voice {
        \times 2/3 {
                c'8 [
                d'8
                e'8
        }
        \times 2/3 {
                f'8
                g'8
                a'8 ]
        }
}

abjad> left, right = containertools.split_container_at_index_and_fracture_crossing_spanners(tupl

abjad> f(voice)
\new Voice {
        \times 2/3 {
                c'8 [
                d'8
                e'8
        }
        \times 2/3 {
                f'8 ]
        }
        \times 2/3 {
                g'8 [
                a'8 ]
        }
}
```

Leave leaves untouched.

Create two new copies of *container*.

Empty *container* of original contents.

Return split parts. Changed in version 1.1.2: renamed `split.fractured_at_index( )` to `containertools.split_container_at_index_and_fracture_crossing_spanners( )`.

containertools.split_container_cyclically_by_counts_and_do_not_fracture_crossing_spanners

abjad.tools.containertools.**split_container_cyclically_by_counts_and_do_not_fracture_crossi**

Split *container* cyclically by *counts* and do not fracture crossing spanners:

```
abjad> container = Container("c'8 d'8 e'8 f'8 g'8 a'8 b'8 c''8")
abjad> voice = Voice([container])
abjad> beam = spannertools.BeamSpanner(voice)
abjad> slur = spannertools.SlurSpanner(container)

abjad> f(voice)
\new Voice {
   {
      c'8 [ (
      d'8
      e'8
```

```
            f'8
            g'8
            a'8
            b'8
            c''8 ] )
        }
    }

abjad> containertools.split_container_cyclically_by_counts_and_do_not_fracture_crossing_spanners
[[{c'8}], [{d'8, e'8, f'8}], [{g'8}], [{a'8, b'8, c''8}]]

abjad> f(voice)
\new Voice {
    {
        c'8 [ (
    }
    {
        d'8
        e'8
        f'8
    }
    {
        g'8
    }
    {
        a'8
        b'8
        c''8 ] )
    }
}
```

Return list of list-wrapped container pieces. Changed in version 1.1.2: renamed `partition.cyclic_unfractured_by_counts( )` to `containertools.split_container_cyclically_by_counts_and_do_not_fracture_crossing_spann` `).`

containertools.split_container_cyclically_by_counts_and_fracture_crossing_spanners

abjad.tools.containertools.**split_container_cyclically_by_counts_and_fracture_crossing_spann**

Split *container* cyclically by *counts* and fracture crossing spanners:

```
abjad> container = Container("c'8 d'8 e'8 f'8 g'8 a'8 b'8 c''8")
abjad> voice = Voice([container])
abjad> beam = spannertools.BeamSpanner(voice)
abjad> slur = spannertools.SlurSpanner(container)

abjad> f(voice)
\new Voice {
    {
        c'8 [ (
        d'8
        e'8
        f'8
        g'8
        a'8
        b'8
```

```
        c''8 ] )
    }
}

abjad> containertools.split_container_cyclically_by_counts_and_fracture_crossing_spanners(contai
[[{c'8}], [{d'8, e'8, f'8}], [{g'8}], [{a'8, b'8, c''8}]]

abjad> f(voice)
\new Voice {
    {
        c'8 ( ) [
    }
    {
        d'8 (
        e'8
        f'8 )
    }
    {
        g'8 ( )
    }
    {
        a'8 (
        b'8
        c''8 ] )
    }
}
```

Return list of list-wrapped container pieces. Changed in version 1.1.2: renamed `partition.cyclic_fractured_by_counts( )` to `containertools.split_container_cyclically_by_counts_and_fracture_crossing_spanners( )`.

containertools.split_container_once_by_counts_and_do_not_fracture_crossing_spanners

`abjad.tools.containertools.`**`split_container_once_by_counts_and_do_not_fracture_crossing_span`**

Split *container* once by *counts* and do no fracture crossing spanners:

```
abjad> container = Container("c'8 d'8 e'8 f'8 g'8 a'8 b'8 c''8")
abjad> voice = Voice([container])
abjad> beam = spannertools.BeamSpanner(voice)
abjad> slur = spannertools.SlurSpanner(container)

abjad> f(voice)
\new Voice {
    {
        c'8 [ (
        d'8
        e'8
        f'8
        g'8
        a'8
        b'8
        c''8 ] )
    }
}
```

```
abjad> containertools.split_container_once_by_counts_and_do_not_fracture_crossing_spanners(conta
[[{c'8}], [{d'8, e'8, f'8}], [{g'8, a'8, b'8, c''8}]]

abjad> f(voice)
\new Voice {
   {
      c'8 [ (
   }
   {
      d'8
      e'8
      f'8
   }
   {
      g'8
      a'8
      b'8
      c''8 ] )
   }
}
```

Return list of list-wrapped container pieces. Changed in version 1.1.2: renamed `partition.unfractured_by_counts( )` to `containertools.split_container_once_by_counts_and_c` `)`.

containertools.split_container_once_by_counts_and_fracture_crossing_spanners

`abjad.tools.containertools.`**`split_container_once_by_counts_and_fracture_crossing_spanners`**`(`*co*
co

Split *container* once by *counts* and fracture crossing spanners:

```
abjad> container = Container("c'8 d'8 e'8 f'8 g'8 a'8 b'8 c''8")
abjad> voice = Voice([container])
abjad> beam = spannertools.BeamSpanner(voice)
abjad> slur = spannertools.SlurSpanner(container)

abjad> f(voice)
\new Voice {
   {
      c'8 [ (
      d'8
      e'8
      f'8
      g'8
      a'8
      b'8
      c''8 ] )
   }
}

abjad> containertools.split_container_once_by_counts_and_fracture_crossing_spanners(container, [
[[{c'8}], [{d'8, e'8, f'8}], [{g'8, a'8, b'8, c''8}]]

abjad> f(voice)
\new Voice {
   {
      c'8 ( ) [
```

```
   }
   {
      d'8 (
      e'8
      f'8 )
   }
   {
      g'8 (
      a'8
      b'8
      c''8 ] )
   }
}
```

Return list of list-wrapped container pieces. Changed in version 1.1.2: renamed `partition.fractured_by_counts( )` to `containertools.split_container_once_by_counts_and_fra` `)`.

contexttools

contexttools.ClefMark

class `abjad.tools.contexttools.`**`ClefMark`**(*arg*, *target_context=None*)

Bases: `abjad.tools.contexttools.ContextMark.ContextMark.ContextMark` New in version 1.1.2. Abjad model of a clef:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")

abjad> contexttools.ClefMark('treble')(staff)
ClefMark('treble')(Staff{4})

abjad> f(staff)
\new Staff {
   \clef "treble"
   c'8
   d'8
   e'8
   f'8
}
```

Clef marks target the staff context by default.

`clef_name_string`

Get clef name string:

```
abjad> clef = contexttools.ClefMark('treble')
abjad> clef.clef_name_string
'treble'
```

Set clef name string:

```
abjad> clef.clef_name_string = 'alto'
abjad> clef.clef_name_string
'alto'
```

Return string.

`format`

Read-only LilyPond format of clef:

```
abjad> clef = contexttools.ClefMark('treble')
abjad> clef.format
'\\clef "treble"'
```

Return string.

middle_c_position

Read-only middle-C position of clef:

```
abjad> clef = contexttools.ClefMark('treble')
abjad> clef.middle_c_position
-6
```

Return integer number of stafflines.

contexttools.ContextMark

class `abjad.tools.contexttools.`**ContextMark**(*target_context=None*)

Bases: `abjad.tools.marktools.Mark.Mark.Mark` New in version 1.1.2. Abstract class from which concrete context marks inherit:

```
abjad> note = Note("c'4")

abjad> contexttools.ContextMark( )(note)
ContextMark( )(c'4)
```

Context marks override `__call__` to attach to Abjad components.

Context marks implement `__slots__`.

detach_mark()

Detach mark:

```
abjad> note = Note("c'4")
abjad> context_mark = contexttools.ContextMark( )(note)

abjad> context_mark.start_component
Note("c'4")

abjad> context_mark.detach_mark( )
ContextMark( )

abjad> context_mark.start_component is None
True
```

Return context mark.

effective_context

Read-only reference to effective context of context mark:

```
abjad> note = Note("c'4")
abjad> context_mark = contexttools.ContextMark( )(note)

abjad> context_mark.effective_context is None
True
```

Return context mark or none.

target_context

Read-only reference to target context of context mark:

```
abjad> note = Note("c'4")
abjad> context_mark = contexttools.ContextMark( )(note)

abjad> context_mark.target_context is None
True
```

Return context mark or none.

contexttools.DynamicMark

class `abjad.tools.contexttools.`**`DynamicMark`**(*dynamic_name_string*, *target_context=None*)

Bases: `abjad.tools.contexttools.ContextMark.ContextMark.ContextMark` New in version 1.1.2. Abjad model of a dynamic mark:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")

abjad> contexttools.DynamicMark('f')(staff[0])
DynamicMark('f')(c'8)

abjad> f(staff)
\new Staff {
   c'8 \f
   d'8
   e'8
   f'8
}
```

Dynamic marks target the staff context by default.

static **`composite_dynamic_name_to_steady_state_dynamic_name`**(*dynamic_name*)

Change composite *dynamic_name* to steady state dynamic name:

```
abjad> contexttools.DynamicMark.composite_dynamic_name_to_steady_state_dynamic_name('sfp')
'p'
```

Return string.

`dynamic_name_string`

Get dynamic name string:

```
abjad> dynamic = contexttools.DynamicMark('f')
abjad> dynamic.dynamic_name_string
'f'
```

Set dynamic name string:

```
abjad> dynamic.dynamic_name_string = 'p'
abjad> dynamic.dynamic_name_string
'p'
```

Return string.

static **`dynamic_name_to_dynamic_ordinal`**(*dynamic_name*)

Change *dynamic_name* to dynamic ordinal:

```
abjad> contexttools.DynamicMark.dynamic_name_to_dynamic_ordinal('fff')
4
```

Return integer.

static dynamic_ordinal_to_dynamic_name (*dynamic_ordinal*)

Change *dynamic_ordinal* to dynamic name:

```
abjad> contexttools.DynamicMark.dynamic_ordinal_to_dynamic_name(-5)
'pppp'
```

Return string.

format

Read-only LilyPond input format of dynamic mark:

```
abjad> dynamic_mark = contexttools.DynamicMark('f')
abjad> dynamic_mark.format
'\f'
```

Return string.

static is_dynamic_name (*arg*)

True when *arg* is dynamic name. False otherwise:

```
abjad> contexttools.DynamicMark.is_dynamic_name('f')
True
```

Return boolean.

contexttools.InstrumentMark

class abjad.tools.contexttools.**InstrumentMark** (*instrument_name*, *short_instrument_name*, *target_context=None*)

Bases: abjad.tools.contexttools.ContextMark.ContextMark.ContextMark New in version 1.1.2. Abjad model of an instrument change:

```
abjad> staff = Staff(macros.scale(4))

abjad> contexttools.InstrumentMark('Flute', 'Fl.')(staff) # doctest: +SKIP
InstrumentMark('Flute', 'Fl.')(Staff{4})

abjad> f(staff) # doctest: +SKIP
\new Staff {
   \set Staff.instrumentName = \markup { Flute }
   \set Staff.shortInstrumentName = \markup { Fl. }
   c'8
   d'8
   e'8
   f'8
}
```

Instrument marks target staff context by default.

format

Read-only LilyPond input format of instrument mark:

```
abjad> instrument = contexttools.InstrumentMark('Flute', 'Fl.')
abjad> instrument.format
['\set Staff.instrumentName = \markup { Flute }', '\set Staff.shortInstrumentName = \markup
```

Return list.

instrument_name

Get instrument name:

```
abjad> instrument = contexttools.InstrumentMark('Flute', 'Fl.')
abjad> instrument.instrument_name
Markup('Flute')
```

Set instrument name:

```
abjad> instrument.instrument_name = 'Alto Flute'
abjad> instrument.instrument_name
Markup('Alto Flute')
```

Return markup.

short_instrument_name
Get short instrument name:

```
abjad> instrument = contexttools.InstrumentMark('Flute', 'Fl.')
abjad> instrument.short_instrument_name
Markup('Fl.')
```

Set short instrument name:

```
abjad> instrument.short_instrument_name = 'Alto Fl.'
abjad> instrument.short_instrument_name
Markup('Alto Fl.')
```

Return markup.

contexttools.KeySignatureMark

class abjad.tools.contexttools.**KeySignatureMark**(*tonic*, *mode*, *target_context=None*)
Bases: abjad.tools.contexttools.ContextMark.ContextMark.ContextMark New in version 1.1.2. Abjad model of a key signature setting or key signature change:

```
abjad> staff = Staff("e'8 fs'8 gs'8 a'8")

abjad> contexttools.KeySignatureMark('e', 'major')(staff)
KeySignatureMark(NamedChromaticPitchClass('e'), Mode(major))(Staff{4})

abjad> f(staff)
\new Staff {
   \key e \major
   e'8
   fs'8
   gs'8
   a'8
}
```

Key signature marks target staff context by default.

format
Read-only LilyPond format of key signature mark:

```
abjad> key_signature = contexttools.KeySignatureMark('e', 'major')
abjad> key_signature.format
'\\key e \\major'
```

Return string.

mode
Get mode of key signature:

```
abjad> key_signature = contexttools.KeySignatureMark('e', 'major')
abjad> key_signature.mode
Mode(major)
```

Set mode of key signature:

```
abjad> key_signature.mode = 'minor'
abjad> key_signature.mode
Mode(minor)
```

Return mode.

name
Read-only name of key signature:

```
abjad> key_signature = contexttools.KeySignatureMark('e', 'major')
abjad> key_signature.name
'E major'
```

Return string.

tonic
Get tonic of key signature:

```
abjad> key_signature = contexttools.KeySignatureMark('e', 'major')
abjad> key_signature.tonic
NamedChromaticPitchClass('e')
```

Set tonic of key signature:

```
abjad> key_signature.tonic = 'd'
abjad> key_signature.tonic
NamedChromaticPitchClass('d')
```

Return named chromatic pitch.

contexttools.StaffChangeMark

class `abjad.tools.contexttools.`**`StaffChangeMark`**(*staff*, *target_context=None*)
Bases: `abjad.tools.contexttools.ContextMark.ContextMark.ContextMark` New in version 1.1.2. Abjad model of a staff change:

```
abjad> piano_staff = scoretools.PianoStaff([ ])
abjad> rh_staff = Staff("c'8 d'8 e'8 f'8")
abjad> rh_staff.name = 'RHStaff'
abjad> lh_staff = Staff("s2")
abjad> lh_staff.name = 'LHStaff'
abjad> piano_staff.extend([rh_staff, lh_staff])

abjad> f(piano_staff)
\new PianoStaff <<
   \context Staff = "RHStaff" {
      c'8
      d'8
      e'8
      f'8
   }
```

```
    \context Staff = "LHStaff" {
        s2
    }
>>

abjad> contexttools.StaffChangeMark(lh_staff)(rh_staff[2])
StaffChangeMark(Staff-"LHStaff"{1})(e'8)

abjad> f(piano_staff) # doctest: +SKIP
\new PianoStaff <<
    \context Staff = "RHStaff" {
        c'8
        d'8
        \change Staff = LHStaff
        e'8
        f'8
    }
    \context Staff = "LHStaff" {
        s2
    }
>>
```

Staff change marks target staff context by default.

format

Read-only LilyPond format of staff change mark:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> staff.name = 'RHStaff'
abjad> staff_change = contexttools.StaffChangeMark(staff)
abjad> staff_change.format
'\\change Staff = RHStaff'
```

Return string.

staff

Get staff of staff change mark:

```
abjad> rh_staff = Staff("c'8 d'8 e'8 f'8")
abjad> rh_staff.name = 'RHStaff'
abjad> staff_change = contexttools.StaffChangeMark(rh_staff)
abjad> staff_change.staff
Staff-"RHStaff"{4}
```

Set staff of staff change mark:

```
abjad> lh_staff = Staff("s2")
abjad> lh_staff.name = 'LHStaff'
abjad> staff_change.staff = lh_staff
abjad> staff_change.staff
Staff-"LHStaff"{1}
```

Return staff.

contexttools.TempoMark

class `abjad.tools.contexttools.`**`TempoMark`**`(`**args*, ***kwargs*`)`

Bases: `abjad.tools.contexttools.ContextMark.ContextMark.ContextMark` New in version 1.1.2. Abjad model of a tempo indication:

```
abjad> score = Score([ ])
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> score.append(staff)

abjad> contexttools.TempoMark(Duration(1, 8), 52)(staff[0])
TempoMark(8, 52)(c'8)

abjad> f(score)
\new Score <<
   \tempo 8=52
   \new Staff {
      c'8
      d'8
      e'8
      f'8
   }
>>
```

Tempo marks target **score** context by default.

duration

Get duration of tempo mark:

```
abjad> tempo = contexttools.TempoMark(Duration(1, 8), 52)
abjad> tempo.duration
Duration(1, 8)
```

Set duration of tempo mark:

```
abjad> tempo.duration = Duration(1, 4)
abjad> tempo.duration
Duration(1, 4)
```

Return duration.

format

Read-only LilyPond format of tempo mark:

```
abjad> tempo = contexttools.TempoMark(Duration(1, 8), 52)
abjad> tempo.format
'\\tempo 8=52'
```

Return string.

quarters_per_minute

Read-only quarters per minute of tempo mark:

```
abjad> tempo = contexttools.TempoMark(Duration(1, 8), 52)
abjad> tempo.quarters_per_minute
Duration(104, 1)
```

Return fraction.

units_per_minute

Get units per minute of tempo mark:

```
abjad> tempo = contexttools.TempoMark(Duration(1, 8), 52)
abjad> tempo.units_per_minute
52
```

Set units per minute of tempo mark:

```
abjad> tempo.units_per_minute = 56
abjad> tempo.units_per_minute
56
```

Return number.

contexttools.TimeSignatureMark

class `abjad.tools.contexttools.`**`TimeSignatureMark`**(**args*, ***kwargs*)

Bases: `abjad.tools.contexttools.ContextMark.ContextMark.ContextMark` New in version 1.1.2. Abjad model of a time signature:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
```

```
abjad> contexttools.TimeSignatureMark((4, 8))(staff[0])
TimeSignatureMark(4, 8)(c'8)
```

```
abjad> f(staff)
\new Staff {
   \time 4/8
   c'8
   d'8
   e'8
   f'8
}
```

Abjad time signature marks target **staff context** by default.

Initialize time signature marks to **score context** like this:

```
abjad> contexttools.TimeSignatureMark((4, 8), target_context = Score)
TimeSignatureMark(4, 8, target_context = Score)
```

Time signatures are immutable.

denominator

Get denominator of time signature mark:

```
abjad> meter = contexttools.TimeSignatureMark(3, 8)
abjad> meter
TimeSignatureMark(3, 8)
abjad> meter.denominator
8
```

Set denominator of time signature mark:

```
abjad> meter.denominator = 16
abjad> meter.denominator
16
```

Return integer.

duration

Read-only duration of time signature mark:

```
abjad> meter = contexttools.TimeSignatureMark(3, 8)
abjad> meter.duration
Duration(3, 8)
```

Return fraction.

format

Read-only LilyPond format of time signature mark:

```
abjad> meter = contexttools.TimeSignatureMark(3, 8)
abjad> meter.format
'\\time 3/8'
```

Return string.

is_nonbinary

Read-only indicator true when time siganture mark is nonbinary:

```
abjad> meter = contexttools.TimeSignatureMark(3, 8)
abjad> meter.is_nonbinary
False
```

Return boolean.

multiplier

Read-only multiplier of time signature mark:

```
abjad> meter = contexttools.TimeSignatureMark(3, 8)
abjad> meter.multiplier
Duration(1, 1)
```

Return fraction.

numerator

Get numerator of time signature mark:

```
abjad> meter = contexttools.TimeSignatureMark(3, 8)
abjad> meter.numerator
3
```

Set numerator of time signature mark:

```
abjad> meter.numerator = 4
abjad> meter.numerator
4
```

Set integer.

partial

Get partial measure pick-up of time signature mark:

```
abjad> meter = contexttools.TimeSignatureMark(3, 8, partial = Duration(1, 8))
abjad> meter.partial
Duration(1, 8)
```

Set partial measure pick-up of time signature mark:

```
abjad> meter.partial = Duration(1, 4)
abjad> meter.partial
Duration(1, 4)
```

Set fraction or none.

contexttools.detach_all_context_marks_attached_to_component

abjad.tools.contexttools.**detach_all_context_marks_attached_to_component**(*start_component*, *klasses=(<class 'abjad.tools.contexttools.Conte* *))*

New in version 1.1.2. Detach context marks attached to *start_component*:

```
abjad> staff = Staff(macros.scale(4))
abjad> clef_mark = contexttools.ClefMark('treble')(staff)
abjad> dynamic_mark = contexttools.DynamicMark('p')(staff[0])
abjad> f(staff)
\new Staff {
   \clef "treble"
   c'8 \p
   d'8
   e'8
   f'8
}
```

```
abjad> contexttools.detach_all_context_marks_attached_to_component(staff[0])
(DynamicMark('p'),)
```

```
abjad> f(staff)
\new Staff {
   \clef "treble"
   c'8
   d'8
   e'8
   f'8
}
```

Return tuple of zero or marks. Changed in version 1.1.2: renamed `contexttools.detach_context_marks_attached_to_start_component()` to `contexttools.detach_all_context_marks_attached_to_component()`.

contexttools.get_context_marks_attached_to_any_improper_parent_of_component

abjad.tools.contexttools.**get_context_marks_attached_to_any_improper_parent_of_component**(*com*

New in version 1.1.2. Get all context marks attached to any improper parent of *component*:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> contexttools.ClefMark('treble')(staff)
ClefMark('treble')(Staff{4})
abjad> contexttools.DynamicMark('f')(staff[0])
DynamicMark('f')(c'8)
```

```
abjad> f(staff)
\new Staff {
   \clef "treble"
   c'8 \f
   d'8
   e'8
   f'8
}
```

```
abjad> contexttools.get_context_marks_attached_to_any_improper_parent_of_component(staff[0])
set([DynamicMark('f')(c'8), ClefMark('treble')(Staff{4})])
```

Return unordered set of zero or more context marks. Changed in version 1.1.2: renamed `contexttools.get_all_context_marks_attached_to_any_improper_parent_of_component()` to `contexttools.get_context_marks_attached_to_any_improper_parent_of_component()`.

contexttools.get_context_marks_attached_to_component

`abjad.tools.contexttools.`**`get_context_marks_attached_to_component`**(*start_component*, *klasses=(<class 'abjad.tools.contexttools.ContextMark.Co*
))

New in version 1.1.2. Get context marks attached to *start_component*:

```
abjad> staff = Staff(macros.scale(4))
abjad> clef_mark = contexttools.ClefMark('treble')(staff)
abjad> dynamic_mark = contexttools.DynamicMark('p')(staff[0])

abjad> f(staff)
\new Staff {
   \clef "treble"
   c'8 \p
   d'8
   e'8
   f'8
}

abjad> contexttools.get_context_marks_attached_to_component(staff[0])
(DynamicMark('p')(c'8),)
```

Return tuple of zero or more context marks. Changed in version 1.1.2: renamed `contexttools.get_context_marks_attached_to_start_component()` to `contexttools.get_context_marks_attached_to_component()`.

contexttools.get_dynamic_marks_attached_to_component

`abjad.tools.contexttools.`**`get_dynamic_marks_attached_to_component`**(*component*)

New in version 1.1.2. Get dynamic marks attached to *component*:

```
abjad> staff = Staff(macros.scale(4))
abjad> clef_mark = contexttools.ClefMark('treble')(staff)
abjad> dynamic_mark = contexttools.DynamicMark('p')(staff[0])
abjad> f(staff)
\new Staff {
   \clef "treble"
   c'8 \p
   d'8
   e'8
   f'8
}
```

```
abjad> contexttools.get_dynamic_marks_attached_to_component(staff[0])
(DynamicMark('p')(c'8),)
```

Return tuple of zero or more dynamic marks.

contexttools.get_effective_clef

`abjad.tools.contexttools.`**`get_effective_clef`**`(`*component*`)`

New in version 1.1.2. Get effective clef of *component*:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> contexttools.ClefMark('treble')(staff)
ClefMark('treble')(Staff{4})

abjad> f(staff)
\new Staff {
   \clef "treble"
   c'8
   d'8
   e'8
   f'8
}

abjad> for note in staff:
...     print note, contexttools.get_effective_clef(note)
...
c'8 ClefMark('treble')(Staff{4})
d'8 ClefMark('treble')(Staff{4})
e'8 ClefMark('treble')(Staff{4})
f'8 ClefMark('treble')(Staff{4})
```

Return clef mark or none.

contexttools.get_effective_context_mark

`abjad.tools.contexttools.`**`get_effective_context_mark`**`(`*component*, *klass*`)`

New in version 1.1.2. Get effective context mark of *klass* from *component*:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> contexttools.TimeSignatureMark(4, 8)(staff)
TimeSignatureMark(4, 8)(Staff{4})

abjad> f(staff)
\new Staff {
   \time 4/8
   c'8
   d'8
   e'8
   f'8
}

abjad> contexttools.get_effective_context_mark(staff[0], contexttools.TimeSignatureMark)
TimeSignatureMark(4, 8)(Staff{4})
```

Return context mark or none.

contexttools.get_effective_dynamic

abjad.tools.contexttools.**get_effective_dynamic**(*component*)

New in version 1.1.2. Get effective dynamic of *component*:

```
abjad> staff = Staff(macros.scale(4))
abjad> contexttools.DynamicMark('f')(staff[0])
DynamicMark('f')(c'8)

abjad> f(staff)
\new Staff {
   c'8 \f
   d'8
   e'8
   f'8
}

abjad> for note in staff:
...     print note, contexttools.get_effective_dynamic(note)
...
c'8 DynamicMark('f')(c'8)
d'8 DynamicMark('f')(c'8)
e'8 DynamicMark('f')(c'8)
f'8 DynamicMark('f')(c'8)
```

Return dynamic mark or none.

contexttools.get_effective_instrument

abjad.tools.contexttools.**get_effective_instrument**(*component*)

New in version 1.1.2. Get effective instrument of *component*:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> contexttools.InstrumentMark('Flute', 'Fl.')(staff)
InstrumentMark('Flute', 'Fl.')

abjad> f(staff)
\new Staff {
   \set Staff.instrumentName = \markup { Flute }
   \set Staff.shortInstrumentName = \markup { Fl. }
   c'8
   d'8
   e'8
   f'8
}

abjad> for note in staff:
...     print note, contexttools.get_effective_instrument(note)
...
c'8 InstrumentMark('Flute', 'Fl.')
d'8 InstrumentMark('Flute', 'Fl.')
e'8 InstrumentMark('Flute', 'Fl.')
f'8 InstrumentMark('Flute', 'Fl.')
```

Return instrument mark or none.

contexttools.get_effective_key_signature

`abjad.tools.contexttools.`**`get_effective_key_signature`**`(`*component*`)`

New in version 1.1.2. Get effective key signature of *component*:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> contexttools.KeySignatureMark('c', 'major')(staff)
KeySignatureMark(NamedChromaticPitchClass('c'), Mode(major))(Staff{4})

abjad> f(staff)
\new Staff {
   \key c \major
   c'8
   d'8
   e'8
   f'8
}

abjad> for note in staff:
...     note, contexttools.get_effective_key_signature(note)
...
(Note("c'8"), KeySignatureMark(NamedChromaticPitchClass('c'), Mode(major))(Staff{4}))
(Note("d'8"), KeySignatureMark(NamedChromaticPitchClass('c'), Mode(major))(Staff{4}))
(Note("e'8"), KeySignatureMark(NamedChromaticPitchClass('c'), Mode(major))(Staff{4}))
(Note("f'8"), KeySignatureMark(NamedChromaticPitchClass('c'), Mode(major))(Staff{4}))
```

Return key signature mark or none.

contexttools.get_effective_staff

`abjad.tools.contexttools.`**`get_effective_staff`**`(`*component*`)`

New in version 1.1.2. Get effective staff of *component*:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> staff.name = 'First Staff'

abjad> f(staff)
\context Staff = "First Staff" {
   c'8
   d'8
   e'8
   f'8
}

abjad> for note in staff:
...     print note, contexttools.get_effective_staff(note)
...
c'8 Staff-"First Staff"{4}
d'8 Staff-"First Staff"{4}
e'8 Staff-"First Staff"{4}
f'8 Staff-"First Staff"{4}
```

Return staff or none.

contexttools.get_effective_tempo

`abjad.tools.contexttools.`**`get_effective_tempo`**(*component*)

New in version 1.1.2. Get effective tempo of *component*:

```
abjad> score = Score([ ])
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> score.append(staff)
abjad> contexttools.TempoMark(Duration(1, 8), 52)(staff[0])
TempoMark(8, 52)(c'8)
```

```
abjad> f(score)
\new Score <<
   \tempo 8=52
   \new Staff {
      c'8
      d'8
      e'8
      f'8
   }
>>
```

```
abjad> for note in staff:
...     print note, contexttools.get_effective_tempo(note)
...
c'8 TempoMark(8, 52)(c'8)
d'8 TempoMark(8, 52)(c'8)
e'8 TempoMark(8, 52)(c'8)
f'8 TempoMark(8, 52)(c'8)
```

Return tempo mark or none.

contexttools.get_effective_time_signature

`abjad.tools.contexttools.`**`get_effective_time_signature`**(*component*)

New in version 1.1.2. Get effective time signature of *component*:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> contexttools.TimeSignatureMark(4, 8)(staff)
TimeSignatureMark(4, 8)(Staff{4})
```

```
abjad> f(staff)
\new Staff {
   \time 4/8
   c'8
   d'8
   e'8
   f'8
}
```

```
abjad> for note in staff:
...     note, contexttools.get_effective_time_signature(note)
...
(Note("c'8"), TimeSignatureMark(4, 8)(Staff{4}))
(Note("d'8"), TimeSignatureMark(4, 8)(Staff{4}))
(Note("e'8"), TimeSignatureMark(4, 8)(Staff{4}))
(Note("f'8"), TimeSignatureMark(4, 8)(Staff{4}))
```

Return time signature mark or none.

contexttools.iterate_contexts_backward_in_expr

`abjad.tools.contexttools.`**`iterate_contexts_backward_in_expr`**(*expr*, *start=0*, *stop=None*)

New in version 1.1.2. Iterate contexts backward in *expr*:

```
abjad> staff = Staff([Voice("c'8 d'8"), Voice("e'8 f'8 g'8")])
abjad> Tuplet((2, 3), staff[1][:])
Tuplet(2/3, [e'8, f'8, g'8])
abjad> staff.is_parallel = True

abjad> f(staff)
\new Staff <<
   \new Voice {
      c'8
      d'8
   }
   \new Voice {
      \times 2/3 {
         e'8
         f'8
         g'8
      }
   }
>>

abjad> for x in contexttools.iterate_contexts_backward_in_expr(staff):
...   x
Staff<<2>>
Voice{1}
Voice{2}
```

Ignore threads.

Return generator.

contexttools.iterate_contexts_forward_in_expr

`abjad.tools.contexttools.`**`iterate_contexts_forward_in_expr`**(*expr*, *start=0*, *stop=None*)

New in version 1.1.2. Iterate contexts forward in *expr*:

```
abjad> staff = Staff([Voice("c'8 d'8"), Voice("e'8 f'8 g'8")])
abjad> Tuplet((2, 3), staff[1][:])
Tuplet(2/3, [e'8, f'8, g'8])
abjad> staff.is_parallel = True

abjad> f(staff)
\new Staff <<
   \new Voice {
      c'8
      d'8
   }
   \new Voice {
      \times 2/3 {
```

```
            e'8
            f'8
            g'8
        }
    }
>>
```

```
abjad> for x in contexttools.iterate_contexts_forward_in_expr(staff):
...     x
Staff<<2>>
Voice{2}
Voice{1}
```

Ignore threads.

Return generator.

contexttools.set_accidental_style_on_sequential_contexts_in_expr

abjad.tools.contexttools.**set_accidental_style_on_sequential_contexts_in_expr**(*expr*, *accidental_style*)

New in version 1.1.2. Set *accidental_style* for sequential semantic contexts in *expr*:

```
abjad> score = Score(Staff(macros.scale(2)) * 2)
abjad> contexttools.set_accidental_style_on_sequential_contexts_in_expr(score, 'forget')
```

```
abjad> f(score)
\new Score <<
        \new Staff {
                #(set-accidental-style 'forget)
                c'8
                d'8
        }
        \new Staff {
                #(set-accidental-style 'forget)
                c'8
                d'8
        }
>>
```

Skip nonsemantic contexts.

Function looks like a hack but isn't. LilyPond uses the dedicated command shown here to set accidental style. This means that it is not possible to set accidental style on a top-level context like score with a single override.

durtools

durtools.Duration

class abjad.tools.durtools.**Duration**

Bases: `fractions.Fraction` New in version 1.1.2. Abjad model of musical duration:

```
abjad> Duration(15, 16)
Duration(15, 16)
```

Durations inherit from built-in `Fraction`.

durtools.assignable_rational_to_dot_count

abjad.tools.durtools.**assignable_rational_to_dot_count**(*rational*)

New in version 1.1.2. Change assignable *rational* to dot count:

```
abjad> for n in range(1, 9):
...     try:
...             rational = Fraction(n, 16)
...             dot_count = durtools.assignable_rational_to_dot_count(rational)
...             print '%s\t%s' % (rational, dot_count)
...     except AssignabilityError:
...             pass
...
1/16    0
1/8     0
3/16    1
1/4     0
3/8     1
7/16    2
1/2     0
```

Raise assignability error when *rational* not assignable.

Return nonnegative integer.

durtools.assignable_rational_to_lilypond_duration_string

abjad.tools.durtools.**assignable_rational_to_lilypond_duration_string**(*rational*)

New in version 1.1.2. Change assignable *rational* to LilyPond duration string:

```
abjad> durtools.assignable_rational_to_lilypond_duration_string(Fraction(3, 16))
'8.'
```

Raise assignability error when *rational* not assignable.

Return string.

durtools.duration_pair_to_prolation_string

abjad.tools.durtools.**duration_pair_to_prolation_string**(*pair*)

New in version 1.1.2. Change positive integer duration *pair* to colon-separated prolation string:

```
abjad> durtools.duration_pair_to_prolation_string((2, 3))
'3:2'
```

Return string.

durtools.duration_token_to_big_endian_list_of_assignable_duration_pairs

abjad.tools.durtools.**duration_token_to_big_endian_list_of_assignable_duration_pairs**(*duration_*

New in version 1.1.1. Change *duration_token* to big-endian tuple of assignable duration pairs:

```
abjad> duration_tokens = [(n, 16) for n in range(10, 20)]
abjad> for duration_token in duration_tokens:
...     print duration_token, durtools.duration_token_to_big_endian_list_of_assignable_duration_
...
(10, 16) ((8, 16), (2, 16))
(11, 16) ((8, 16), (3, 16))
(12, 16) ((12, 16),)
(13, 16) ((12, 16), (1, 16))
(14, 16) ((14, 16),)
(15, 16) ((15, 16),)
(16, 16) ((16, 16),)
(17, 16) ((16, 16), (1, 16))
(18, 16) ((16, 16), (2, 16))
(19, 16) ((16, 16), (3, 16))
```

Return tuple of integer pairs. Changed in version 1.1.2: renamed `durtools.token_decompose( )` to `durtools.duration_token_to_big_endian_list_of_assignable_duration_pairs( )`.

durtools.duration_token_to_duration_pair

abjad.tools.durtools.**duration_token_to_duration_pair**(*duration_token*)

New in version 1.1.1. Change *duration_token* to duration pair:

```
abjad> durtools.duration_token_to_duration_pair(Fraction(2, 4))
(1, 2)
```

New in version 1.1.2: Change LilyPond duration string to duration pair:

```
abjad> durtools.duration_token_to_duration_pair('8.')
(3, 16)
```

Return pair. Changed in version 1.1.2: renamed `durtools.token_unpack( )` to `durtools.duration_token_to_duration_pair( )`.

durtools.duration_token_to_rational

abjad.tools.durtools.**duration_token_to_rational**(*duration_token*)

New in version 1.1.2. Change *duration_token* to rational:

```
abjad> durtools.duration_token_to_rational((4, 16))
Fraction(1, 4)

abjad> durtools.duration_token_to_rational('4.')
Fraction(3, 8)
```

Return fraction.

durtools.duration_tokens_to_duration_pairs

abjad.tools.durtools.**duration_tokens_to_duration_pairs**(*duration_tokens*)

New in version 1.1.2. Change *duration_tokens* to duration pairs:

```
abjad> durtools.duration_tokens_to_duration_pairs([Fraction(2, 4), 3, '8.', (5, 16)])
[(1, 2), (3, 1), (3, 16), (5, 16)]
```

Return new object of *duration_tokens* type.

durtools.duration_tokens_to_duration_pairs_with_least_common_denominator

`abjad.tools.durtools.duration_tokens_to_duration_pairs_with_least_common_denominator(duratio`

New in version 1.1.2. Change *duration_tokens* to duration pairs with least common denominator:

```
abjad> durtools.duration_tokens_to_duration_pairs_with_least_common_denominator([Fraction(2, 4),
[(8, 16), (48, 16), (3, 16), (5, 16)]
```

Return new object of *duration_tokens* type.

durtools.duration_tokens_to_least_common_denominator

`abjad.tools.durtools.duration_tokens_to_least_common_denominator(duration_tokens)`

New in version 1.1.2. Change *duration_tokens* to least common denominator:

```
abjad> durtools.duration_tokens_to_least_common_denominator([Fraction(2, 4), 3, '8.', (5, 16)])
16
```

Return positive integer.

durtools.duration_tokens_to_rationals

`abjad.tools.durtools.duration_tokens_to_rationals(duration_tokens)`

New in version 1.1.2. Change *duration_tokens* to rationals:

```
abjad> durtools.duration_tokens_to_rationals([Fraction(2, 4), 3, '8.', (5, 16)])
[Fraction(1, 2), Fraction(3, 1), Fraction(3, 16), Fraction(5, 16)]
```

Return new object of *duration_tokens* type.

durtools.group_duration_tokens_by_implied_prolation

`abjad.tools.durtools.group_duration_tokens_by_implied_prolation(durations)`

New in version 1.1.1. Group *durations* by implied prolation:

```
abjad> durtools.group_duration_tokens_by_implied_prolation([(1, 4), (1, 8), (1, 3), (1, 6), (1,
[[(1, 4), (1, 8)], [(1, 3), (1, 6)], [(1, 4)]]
```

Return list of integer pair lists. Changed in version 1.1.2: renamed `durtools.agglomerate_by_prolation( )` to `durtools.group_duration_tokens_by_implied_prolat` `)`.

durtools.is_assignable_rational

`abjad.tools.durtools.is_assignable_rational(expr)`

New in version 1.1.1. True when *expr* is assignable rational. Otherwise false:

```
abjad> for numerator in range(0, 16 + 1):
...     duration = Fraction(numerator, 16)
...     print '%s\t%s' % (duration, durtools.is_assignable_rational(duration))
...
0     False
```

```
1/16  True
1/8   True
3/16  True
1/4   True
5/16  False
3/8   True
7/16  True
1/2   True
9/16  False
5/8   False
11/16 False
3/4   True
13/16 False
7/8   True
15/16 True
1     True
```

Return boolean. Changed in version 1.1.2: renamed `durtools.is_assignable( )` to `durtools.is_assignable_rational( )`.

durtools.is_binary_rational

abjad.tools.durtools.**is_binary_rational**(*rational*)

New in version 1.1.1. True when *rational* is of the form `1/2**n`. Otherwise false:

```
abjad> for n in range(1, 17): # doctest: +SKIP
...     rational = Fraction(1, n)
...     print '%s\t%s' % (rational, durtools.is_binary_rational(rational))
...
1       True
1/2     True
1/3     False
1/4     True
1/5     False
1/6     False
1/7     False
1/8     True
1/9     False
1/10    False
1/11    False
1/12    False
1/13    False
1/14    False
1/15    False
1/16    True
```

Return boolean.

durtools.is_duration_pair

abjad.tools.durtools.**is_duration_pair**(*arg*)

New in version 1.1.1. True when *arg* has the form of a pair of integers that initialize a positive rational:

```
abjad> durtools.is_duration_pair((5, 16))
True
```

Otherwise false:

```
abjad> durtools.is_duration_pair((-5, 16))
False
```

Return boolean. Changed in version 1.1.2: renamed `durtools.is_pair( )` to `durtools.is_duration_pair( )`.

durtools.is_duration_token

`abjad.tools.durtools.`**`is_duration_token`**`(`*expr*`)`

New in version 1.1.2. True when *expr* has the form of an Abjad duration pair:

```
abjad> durtools.is_duration_token('8.')
True
```

Otherwise false:

```
abjad> durtools.is_duration_token('foo')
False
```

Return boolean.

durtools.is_lilypond_duration_name

`abjad.tools.durtools.`**`is_lilypond_duration_name`**`(`*expr*`)`

New in version 1.1.2. True when *expr* is a LilyPond duartion name:

```
abjad> durtools.is_lilypond_duration_name('\\breve')
True
```

Otherwise false:

```
abjad> durtools.is_lilypond_duration_name('foo')
False
```

The regex `^(\\breve|\\longa|\\maxima)$` underlies this predicate.

Return boolean.

durtools.is_lilypond_duration_string

`abjad.tools.durtools.`**`is_lilypond_duration_string`**`(`*expr*`)`

New in version 1.1.2. True when *expr* is a LilyPond duration string:

```
abjad> durtools.is_lilypond_duration_string('4.. * 1/2')
True
```

Otherwise false:

```
abjad> durtools.is_lilypond_duration_string('foo')
False
```

The regex `^(1|2|4|8|16|32|64|128|\breve|\longa|\maxima)\s*(\.*)\s*(\*\s*(\d+(/\d+)?))?$` underlies this predicate.

Return boolean.

durtools.lilypond_duration_string_to_rational

abjad.tools.durtools.**lilypond_duration_string_to_rational**(*duration_string*)

New in version 1.1.2. Change LilyPond *duration_string* to rational:

```
abjad> durtools.lilypond_duration_string_to_rational('8.')
Fraction(3, 16)
```

Return fraction.

durtools.lilypond_duration_string_to_rational_list

abjad.tools.durtools.**lilypond_duration_string_to_rational_list**(*duration_string*)

New in version 1.1.2. Change LilyPond *duration_string* to rational list:

```
abjad> durtools.lilypond_duration_string_to_rational_list('8.. 32 8.. 32')
[Fraction(7, 32), Fraction(1, 32), Fraction(7, 32), Fraction(1, 32)]
```

Return list of fractions.

durtools.multiply_duration_pair

abjad.tools.durtools.**multiply_duration_pair**(*pair*, *multiplier*)

New in version 1.1.1. Multiply duration *pair* by rational *multiplier*:

```
abjad> durtools.multiply_duration_pair((4, 8), Fraction(4, 5))
(16, 40)
```

Naive multiplication with no simplification of anything intended for certain types of meter multiplication.

Return integer pair. Changed in version 1.1.2: renamed `durtools.pair_multiply_naive( )` to `durtools.multiply_duration_pair( )`.

durtools.multiply_duration_pair_and_reduce_factors

abjad.tools.durtools.**multiply_duration_pair_and_reduce_factors**(*pair*, *multiplier*)

New in version 1.1.1. Multiply *pair* by rational *multiplier* and reduce factors:

```
abjad> durtools.multiply_duration_pair_and_reduce_factors((4, 8), Fraction(2, 3))
(4, 12)
```

Intended for certain types of meter multiplication.

Return integer pair. Changed in version 1.1.2: renamed `durtools.pair_multiply_reduce_factors( )` to `durtools.multiply_duration_pair_and_reduce_factors( )`.

durtools.multiply_duration_pair_and_try_to_preserve_numerator

abjad.tools.durtools.**multiply_duration_pair_and_try_to_preserve_numerator**(*pair*, *multiplier*)

New in version 1.1.1. Multiply duration *pair* by rational *multiplier* and try to preserve numerator:

```
abjad> durtools.multiply_duration_pair_and_try_to_preserve_numerator((9, 16), Fraction(2, 3))
(9, 24)
```

Intended for certain types of meter multiplication.

Return integer pair. Changed in version 1.1.2: renamed `durtools.pair_multiply_constant_numerator( )` to `durtools.multiply_duration_pair_and_try_to_preserve_numerator( )`.

durtools.numeric_seconds_to_clock_string

`abjad.tools.durtools.`**`numeric_seconds_to_clock_string`**`(`*seconds*`)`

New in version 1.1.2. Change numeric *seconds* to clock string:

```
abjad> durtools.numeric_seconds_to_clock_string(117)
'1\'57"'
```

Return string.

durtools.numeric_seconds_to_escaped_clock_string

`abjad.tools.durtools.`**`numeric_seconds_to_escaped_clock_string`**`(`*seconds*`)`

New in version 1.1.2. Change numeric *seconds* to escaped clock string:

```
abjad> note = Note(0, (1, 4))
abjad> clock_string = durtools.numeric_seconds_to_escaped_clock_string(117)
abjad> markuptools.Markup('"%s"' % clock_string, 'up')(note)
Markup('"1\'57\\""', 'up')

abjad> f(note)
c'4 ^ \markup { "1'57\"" }
```

Escape seconds indicator for output as LilyPond markup.

Return string.

durtools.positive_integer_to_implied_prolation_multipler

`abjad.tools.durtools.`**`positive_integer_to_implied_prolation_multipler`**`(`*n*`)`

New in version 1.1.1. Change positive integer *n* to implied porlation multiplier:

```
abjad> for denominator in range(1, 17): # doctest: +SKIP
...     multiplier = durtools.positive_integer_to_implied_prolation_multipler(denominator)
...     print '%s\t%s' % (denominator, multiplier)
...
1       1
2       1
3       2/3
4       1
5       4/5
6       2/3
7       4/7
8       1
9       8/9
10      4/5
11      8/11
12      2/3
```

```
13      8/13
14      4/7
15      8/15
16      1
```

Return positive fraction less than or equal to `1`. Changed in version 1.1.2: renamed `durtools.denominator_to_multiplier( )` to `durtools.positive_integer_to_implied_prolation_multipler( )`.

durtools.rational_to_duration_pair_with_multiple_of_specified_integer_denominator

`abjad.tools.durtools.`**`rational_to_duration_pair_with_multiple_of_specified_integer_denominat`**

Change *duration* to duration pair with multiple of specified *integer_denominator*:

```
abjad> durtools.rational_to_duration_pair_with_multiple_of_specified_integer_denominator(Fractic
(1, 2)
abjad> durtools.rational_to_duration_pair_with_multiple_of_specified_integer_denominator(Fractic
(2, 4)
abjad> durtools.rational_to_duration_pair_with_multiple_of_specified_integer_denominator(Fractic
(4, 8)
abjad> durtools.rational_to_duration_pair_with_multiple_of_specified_integer_denominator(Fractic
(8, 16)

abjad> durtools.rational_to_duration_pair_with_multiple_of_specified_integer_denominator(Fractic
(3, 6)
abjad> durtools.rational_to_duration_pair_with_multiple_of_specified_integer_denominator(Fractic
(3, 6)
abjad> durtools.rational_to_duration_pair_with_multiple_of_specified_integer_denominator(Fractic
(6, 12)
abjad> durtools.rational_to_duration_pair_with_multiple_of_specified_integer_denominator(Fractic
(12, 24)

abjad> durtools.rational_to_duration_pair_with_multiple_of_specified_integer_denominator(Fractic
(5, 10)
abjad> durtools.rational_to_duration_pair_with_multiple_of_specified_integer_denominator(Fractic
(5, 10)
abjad> durtools.rational_to_duration_pair_with_multiple_of_specified_integer_denominator(Fractic
(10, 20)
abjad> durtools.rational_to_duration_pair_with_multiple_of_specified_integer_denominator(Fractic
(20, 40)
```

Return integer pair. Changed in version 1.1.2: renamed `durtools.in_terms_of_binary_multiple( )` to `durtools.rational_to_duration_pair_with_multiple_of_specified_integer_denominator( )`.

durtools.rational_to_duration_pair_with_specified_integer_denominator

`abjad.tools.durtools.`**`rational_to_duration_pair_with_specified_integer_denominator`**(*duration*, *integer_denomin*

New in version 1.1.1. Change *duration* to duraiton pair with specified *integer_denominator*:

```
abjad> for n in range(1, 17):
...     rational = Fraction(n, 16)
...     pair = durtools.rational_to_duration_pair_with_specified_integer_denominator(rational, 1
...     print '%s\t%s' % (rational, pair)
...
1/16    (1, 16)
1/8     (2, 16)
3/16    (3, 16)
1/4     (4, 16)
5/16    (5, 16)
3/8     (6, 16)
7/16    (7, 16)
1/2     (8, 16)
9/16    (9, 16)
5/8     (10, 16)
11/16   (11, 16)
3/4     (12, 16)
13/16   (13, 16)
7/8     (14, 16)
15/16   (15, 16)
1       (16, 16)
```

Return integer pair. Changed in version 1.1.2: renamed `durtools.in_terms_of( )` to `durtools.rational_to_duration_pair_with_specified_integer_denominator( )`.

durtools.rational_to_equal_or_greater_assignable_rational

abjad.tools.durtools.**rational_to_equal_or_greater_assignable_rational**(*rational*)

New in version 1.1.1. Change *rational* to equal or greater assignable rational:

```
abjad> for n in range(1, 17): # doctest: +SKIP
...     prolated = Fraction(n, 16)
...     written = durtools.rational_to_equal_or_greater_assignable_rational(prolated)
...     print '%s/16\t%s' % (n, written)
...
1/16    1/16
2/16    1/8
3/16    3/16
4/16    1/4
5/16    3/8
6/16    3/8
7/16    7/16
8/16    1/2
9/16    3/4
10/16   3/4
11/16   3/4
12/16   3/4
13/16   7/8
14/16   7/8
15/16   15/16
16/16   1
```

Return fraction.

Function returns dotted and double dotted durations where possible. Changed in version 1.1.2: Fixed to produce monotonically increasing output in response to monotonically increasing input.Changed in version 1.1.2: renamed `durtools.prolated_to_written_not_less_than( )` to `durtools.rational_to_equal_or_greater_assignable_rational( )`.

durtools.rational_to_equal_or_greater_binary_rational

abjad.tools.durtools.**rational_to_equal_or_greater_binary_rational**(*rational*)

New in version 1.1.1. Change *rational* to equal to greater binary rational:

```
abjad> for n in range(1, 17): # doctest: +SKIP
...     rational = Fraction(n, 16)
...     written_duration = durtools.rational_to_equal_or_greater_binary_rational(rational)
...     print '%s/16\t%s' % (n, written_duration)
...
1/16    1/16
2/16    1/8
3/16    1/4
4/16    1/4
5/16    1/2
6/16    1/2
7/16    1/2
8/16    1/2
9/16    1
10/16   1
11/16   1
12/16   1
13/16   1
14/16   1
15/16   1
16/16   1
```

```
abjad> durtools.rational_to_equal_or_greater_binary_rational(Fraction(1, 80))
Fraction(1, 64)
```

```
abjad> durtools.rational_to_equal_or_greater_binary_rational(Fraction(17, 16))
Fraction(2, 1)
```

Use to find written duration of tupletted leaves.

Return fraction. Changed in version 1.1.2: renamed `durtools.naive_prolated_to_written_not_less_than( )` to `durtools.rational_to_equal_or_greater_binary_rational( )`.

durtools.rational_to_equal_or_lesser_assignable_rational

abjad.tools.durtools.**rational_to_equal_or_lesser_assignable_rational**(*rational*)

New in version 1.1.1. Change *rational* to equal or lesser assignable rational:

```
abjad> for n in range(1, 17): # doctest: +SKIP
...     rational = Fraction(n, 16)
...     written = durtools.rational_to_equal_or_lesser_assignable_rational(rational)
...     print '%s/16\t%s' % (n, written)
...
1/16    1/16
2/16    1/8
3/16    3/16
4/16    1/4
5/16    1/4
6/16    3/8
7/16    7/16
8/16    1/2
9/16    1/2
10/16   1/2
```

```
11/16   1/2
12/16   3/4
13/16   3/4
14/16   7/8
15/16   15/16
16/16   1
```

Return fraction.

Function returns dotted and double dotted durations where possible. Changed in version 1.1.2: Fixed to produce monotonically increasing output in response to monotonically increasing input.Changed in version 1.1.2: renamed `durtools.prolated_to_written_not_greater_than( )` to `durtools.rational_to_equal_or_lesser_assignable_rational( )`.

durtools.rational_to_equal_or_lesser_binary_rational

`abjad.tools.durtools.`**`rational_to_equal_or_lesser_binary_rational`**(*rational*)

New in version 1.1.1. Change *rational* to equal or lesser binary rational:

```
abjad> for n in range(1, 17): # doctest: +SKIP
...     rational = Fraction(n, 16)
...     written_duration = durtools.rational_to_equal_or_lesser_binary_rational(rational)
...     print '%s/16\t%s' % (n, written_duration)
...
1/16    1/16
2/16    1/8
3/16    1/8
4/16    1/4
5/16    1/4
6/16    1/4
7/16    1/4
8/16    1/2
9/16    1/2
10/16   1/2
11/16   1/2
12/16   1/2
13/16   1/2
14/16   1/2
15/16   1/2
16/16   1
```

```
abjad> durtools.rational_to_equal_or_lesser_binary_rational(Fraction(1, 80))
Fraction(1, 128)
```

Return fraction.

Function intended to find written duration of notes inside tuplet. Changed in version 1.1.2: renamed `durtools.naive_prolated_to_written_not_greater_than( )` to `durtools.rational_to_equal_or_lesser_binary_rational( )`.

durtools.rational_to_flag_count

`abjad.tools.durtools.`**`rational_to_flag_count`**(*rational*)

New in version 1.1.2. Change *rational* to number of flags required to notate:

```
abjad> durtools.rational_to_flag_count(Fraction(1, 32))
3
```

Return nonnegative integer.

durtools.rational_to_fraction_string

abjad.tools.durtools.**rational_to_fraction_string**(*rational*)

New in version 1.1.1. Change *rational* to fraction string:

```
abjad> durtools.rational_to_fraction_string(Fraction(2, 4))
'1/2'
```

Return string.

durtools.rational_to_prolation_string

abjad.tools.durtools.**rational_to_prolation_string**(*rational*)

New in version 1.1.2. Change *rational* to prolation string:

```
abjad> generator = durtools.yield_all_positive_rationals_in_cantor_diagonalized_order_uniquely(
abjad> for n in range(16): # doctest: +SKIP
...     rational = generator.next( )
...     prolation_string = durtools.rational_to_prolation_string(rational)
...     print '%s\\t%s' % (rational, prolation_string)
...
1       1:1
2       1:2
1/2     2:1
1/3     3:1
3       1:3
4       1:4
3/2     2:3
2/3     3:2
1/4     4:1
1/5     5:1
5       1:5
6       1:6
5/2     2:5
4/3     3:4
3/4     4:3
2/5     5:2
```

Return string.

durtools.rational_to_proper_fraction

abjad.tools.durtools.**rational_to_proper_fraction**(*rational*)

New in version 1.1.2. Change *rational* to proper fraction:

```
abjad> durtools.rational_to_proper_fraction(Fraction(116, 8))
(14, Fraction(1, 2))
```

Return pair.

durtools.rewrite_rational_under_new_tempo

abjad.tools.durtools.**rewrite_rational_under_new_tempo**(*prolated_duration_1*, *tempo_mark_1*, *tempo_mark_2*)

New in version 1.1.2. Given *prolated_duration_1* governed by *tempo_mark_1*, return *prolated_duration_2* governed by *tempo_mark_2* such that *prolated_duration_1* and *prolated_duration_2* consume exactly the same amount of time in seconds.

Consider the two tempo indications below.

```
abjad> tempo_mark_1 = contexttools.TempoMark(Fraction(1, 4), 60)
abjad> tempo_mark_2 = contexttools.TempoMark(Fraction(1, 4), 90)
```

The first tempo indication specifies quarter = 60 MM. The second tempo indication specifies quarter = 90 MM.

The second tempo is 1 1/2 times as fast as the first.

```
abjad> tempo_mark_2 / tempo_mark_1
Fraction(3, 2)
```

An triplet eighth note at tempo 1 equals a regular eighth note at tempo 2.

```
abjad> durtools.rewrite_rational_under_new_tempo(Fraction(1, 12), tempo_mark_1, tempo_mark_2)
Fraction(1, 8)
```

Conversely, a regular eighth not at tempo 1 equals a dotted sixteenth at tempo 2.

```
abjad> durtools.rewrite_rational_under_new_tempo(Fraction(1, 8), tempo_mark_1, tempo_mark_2)
Fraction(3, 16)
```

Return fraction.

durtools.yield_all_assignable_rationals_in_cantor_diagonalized_order

abjad.tools.durtools.**yield_all_assignable_rationals_in_cantor_diagonalized_order**()

New in version 1.1.2. Yield all assignable rationals in Cantor diagonalized order:

```
abjad> generator = durtools.yield_all_assignable_rationals_in_cantor_diagonalized_order( )
abjad> for n in range(16):
...     generator.next( )
...
Fraction(1, 1)
Fraction(2, 1)
Fraction(1, 2)
Fraction(3, 1)
Fraction(4, 1)
Fraction(3, 2)
Fraction(1, 4)
Fraction(6, 1)
Fraction(3, 4)
Fraction(7, 1)
Fraction(8, 1)
Fraction(7, 2)
Fraction(1, 8)
Fraction(7, 4)
Fraction(3, 8)
Fraction(12, 1)
```

Return fraction generator.

durtools.yield_all_positive_integer_pairs_in_cantor_diagonalized_order

abjad.tools.durtools.**yield_all_positive_integer_pairs_in_cantor_diagonalized_order**()

New in version 1.1.2. Yield all positive integer pairs in Cantor diagonalized order:

```
abjad> generator = durtools.yield_all_positive_integer_pairs_in_cantor_diagonalized_order( )
abjad> for n in range(16):
...     generator.next( )
...
(1, 1)
(2, 1)
(1, 2)
(1, 3)
(2, 2)
(3, 1)
(4, 1)
(3, 2)
(2, 3)
(1, 4)
(1, 5)
(2, 4)
(3, 3)
(4, 2)
(5, 1)
(6, 1)
```

Return pair generator.

durtools.yield_all_positive_rationals_in_cantor_diagonalized_order

abjad.tools.durtools.**yield_all_positive_rationals_in_cantor_diagonalized_order**()

New in version 1.1.2. Yield all positive rationals in Cantor diagonalized order:

```
abjad> generator = durtools.yield_all_positive_rationals_in_cantor_diagonalized_order( )
abjad> for n in range(16):
...     generator.next( )
...
Fraction(1, 1)
Fraction(2, 1)
Fraction(1, 2)
Fraction(1, 3)
Fraction(1, 1)
Fraction(3, 1)
Fraction(4, 1)
Fraction(3, 2)
Fraction(2, 3)
Fraction(1, 4)
Fraction(1, 5)
Fraction(1, 2)
Fraction(1, 1)
Fraction(2, 1)
Fraction(5, 1)
Fraction(6, 1)
```

Return fraction generator.

durtools.yield_all_positive_rationals_in_cantor_diagonalized_order_uniquely

abjad.tools.durtools.**yield_all_positive_rationals_in_cantor_diagonalized_order_uniquely**()

New in version 1.1.2. Yield all positive rationals in Cantor diagonalized order uniquely:

```
abjad> generator = durtools.yield_all_positive_rationals_in_cantor_diagonalized_order_uniquely(
abjad> for n in range(16):
...     generator.next( )
...
Fraction(1, 1)
Fraction(2, 1)
Fraction(1, 2)
Fraction(1, 3)
Fraction(3, 1)
Fraction(4, 1)
Fraction(3, 2)
Fraction(2, 3)
Fraction(1, 4)
Fraction(1, 5)
Fraction(5, 1)
Fraction(6, 1)
Fraction(5, 2)
Fraction(4, 3)
Fraction(3, 4)
Fraction(2, 5)
```

Return fraction generator.

durtools.yield_all_prolation_rewrite_pairs_of_rational_in_cantor_diagonalized_order

abjad.tools.durtools.**yield_all_prolation_rewrite_pairs_of_rational_in_cantor_diagonalized_o**

New in version 1.1.2. Yield all prolation rewrite pairs of *prolated_duration* in Cantor diagonalized order.

Ensure written duration never less than *minimum_written_duration*.

The different ways to notate a prolated duration of `1/8`:

```
abjad> pairs = durtools.yield_all_prolation_rewrite_pairs_of_rational_in_cantor_diagonalized_ord
abjad> for pair in pairs: pair
...
(Fraction(1, 1), Fraction(1, 8))
(Fraction(2, 3), Fraction(3, 16))
(Fraction(4, 3), Fraction(3, 32))
(Fraction(4, 7), Fraction(7, 32))
(Fraction(8, 7), Fraction(7, 64))
(Fraction(8, 15), Fraction(15, 64))
(Fraction(16, 15), Fraction(15, 128))
(Fraction(16, 31), Fraction(31, 128))
```

The different ways to notate a prolated duration of `1/12`.

```
abjad> pairs = durtools.yield_all_prolation_rewrite_pairs_of_rational_in_cantor_diagonalized_ord
abjad> for pair in pairs: pair
...
(Fraction(2, 3), Fraction(1, 8))
```

```
(Fraction(4, 3), Fraction(1, 16))
(Fraction(8, 9), Fraction(3, 32))
(Fraction(16, 9), Fraction(3, 64))
(Fraction(16, 21), Fraction(7, 64))
(Fraction(32, 21), Fraction(7, 128))
(Fraction(32, 45), Fraction(15, 128))
```

The different ways to notate a prolated duration of `5/48`.

```
abjad> pairs = durtools.yield_all_prolation_rewrite_pairs_of_rational_in_cantor_diagonalized_ord
abjad> for pair in pairs: pair
...
(Fraction(5, 6), Fraction(1, 8))
(Fraction(5, 3), Fraction(1, 16))
(Fraction(5, 9), Fraction(3, 16))
(Fraction(10, 9), Fraction(3, 32))
(Fraction(20, 21), Fraction(7, 64))
(Fraction(40, 21), Fraction(7, 128))
(Fraction(8, 9), Fraction(15, 128))
```

Return generator of paired fractions.

gracetools

gracetools.Grace

class `abjad.tools.gracetools.`**`Grace`**(*music=None*, *kind='grace'*, ***kwargs*)

Bases: `abjad.components.Container.Container.Container`

Abjad model of grace music:

```
abjad> voice = Voice("c'8 d'8 e'8 f'8")
abjad> spannertools.BeamSpanner(voice[:])
BeamSpanner(c'8, d'8, e'8, f'8)

abjad> f(voice)
\new Voice {
   c'8 [
   d'8
   e'8
   f'8 ]
}

abjad> grace_notes = [Note("c'16"), Note("d'16")]
abjad> gracetools.Grace(grace_notes, kind = 'grace')(voice[1])
Note("d'8")

abjad> f(voice)
\new Voice {
   c'8 [
   \grace {
      c'16
      d'16
   }
   d'8
   e'8
   f'8 ]
}
```

```
abjad> after_grace_notes = [Note("e'16"), Note("f'16")]
abjad> gracetools.Grace(after_grace_notes, kind = 'after')(voice[1])
Note("d'8")

abjad> f(voice)
\new Voice {
        c'8 [
        \grace {
                c'16
                d'16
        }
        \afterGrace
        d'8
        {
                e'16
                f'16
        }
        e'8
        f'8 ]
}
```

Grace objects are containers you can fill with notes, rests and chords.

Grace containers override the special `__call__` method.

Use `Grace( )` to attach grace containers to nongrace notes, rests and chords.

detach()

Detach grace container from leaf:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> grace_container = gracetools.Grace([Note("cs'16")], kind = 'grace')
abjad> grace_container(staff[1])
abjad> f(staff)
\new Staff {
   c'8
   \grace {
      cs'16
   }
   d'8
   e'8
   f'8
}

abjad> grace_container.detach( )
Grace( )
abjad> f(staff)
\new Staff {
   c'8
   d'8
   e'8
   f'8
}
```

Return grace container.

kind

Get *kind* of grace container:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> gracetools.Grace([Note("cs'16")], kind = 'grace')(staff[1])
abjad> grace_container = staff[1].grace
abjad> grace_container.kind
'grace'
```

Return string.

Set *kind* of grace container:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> gracetools.Grace([Note("cs'16")], kind = 'grace')(staff[1])
abjad> grace_container = staff[1].grace
abjad> grace_container.kind = 'acciaccatura'
abjad> grace_container.kinda
'acciaccatura'
```

Set string.

Valid options include `'after'`, `'grace'`, `'acciaccatura'`, `'appoggiatura'`.

gracetools.detach_grace_containers_attached_to_leaf

abjad.tools.gracetools.**detach_grace_containers_attached_to_leaf**(*leaf*)

New in version 1.1.2. Detach grace containers attached to *leaf*:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> grace_container = gracetools.Grace([Note("cs'16")], kind = 'grace')
abjad> grace_container(staff[1])

abjad> f(staff)
\new Staff {
   c'8
   \grace {
      cs'16
   }
   d'8
   e'8
   f'8
}

abjad> gracetools.get_grace_containers_attached_to_leaf(staff[1])
(Grace(cs'16),)

abjad> gracetools.detach_grace_containers_attached_to_leaf(staff[1])
(Grace( ),)

abjad> f(staff)
\new Staff {
   c'8
   d'8
   e'8
   f'8
}

abjad> gracetools.get_grace_containers_attached_to_leaf(staff[1])
()
```

Return tuple.

gracetools.get_grace_containers_attached_to_leaf

abjad.tools.gracetools.**get_grace_containers_attached_to_leaf**(*leaf*)

New in version 1.1.2. Get grace containers attached to leaf:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> gracetools.Grace([Note("cs'16")], kind = 'grace')(staff[1])
abjad> gracetools.Grace([Note("ds'16")], kind = 'after')(staff[1])

abjad> f(staff)
\new Staff {
   c'8
   \grace {
      cs'16
   }
   \afterGrace
   d'8
   {
      ds'16
   }
   e'8
   f'8
}

abjad> gracetools.get_grace_containers_attached_to_leaf(staff[1])
(Grace(cs'16), Grace(ds'16))
```

Return tuple.

gracetools.iterate_components_and_grace_containers_forward_in_expr

abjad.tools.gracetools.**iterate_components_and_grace_containers_forward_in_expr**(*expr*, *klass*)

Iterate components of *klass* forward in *expr*:

```
abjad> voice = Voice("c'8 d'8 e'8 f'8")
abjad> spannertools.BeamSpanner(voice[:])
BeamSpanner(c'8, d'8, e'8, f'8)

abjad> grace_notes = [Note("c'16"), Note("d'16")]
abjad> gracetools.Grace(grace_notes, kind = 'grace')(voice[1])
Note("d'8")

abjad> after_grace_notes = [Note("e'16"), Note("f'16")]
abjad> gracetools.Grace(after_grace_notes, kind = 'after')(voice[1])
Note("d'8")

abjad> f(voice)
\new Voice {
        c'8 [
        \grace {
                c'16
                d'16
        }
        \afterGrace
        d'8
        {
                e'16
```

```
                f'16
            }
            e'8
            f'8 ]
    }

abjad> for note in gracetools.iterate_components_and_grace_containers_forward_in_expr(voice, Not
...     note
...
Note("c'8")
Note("c'16")
Note("d'16")
Note("d'8")
Note("e'16")
Note("f'16")
Note("e'8")
Note("f'8")
```

Include grace leaves before main leaves.

Include grace leaves after main leaves. Changed in version 1.1.2: renamed `iterate.grace( )` to `componenttools.iterate_components_and_grace_containers_forward_in_expr( )`.

instrumenttools

instrumenttools.Accordion

class `abjad.tools.instrumenttools.`**`Accordion`**(*instrument_name='Accordion', short_instrument_name='Acc.', target_context=None*)

Bases: `abjad.tools.instrumenttools._KeyboardInstrument._KeyboardInstrument._KeyboardInst` `abjad.tools.instrumenttools._ReedInstrument._ReedInstrument._ReedInstrument`

Abjad model of the accordion:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")

abjad> instrumenttools.Accordion(target_context = Staff)(staff)
Accordion('Accordion', 'Acc.')

abjad> f(staff)
\new Staff {
   \set Staff.instrumentName = \markup { Accordion }
   \set Staff.shortInstrumentName = \markup { Acc. }
   c'8
   d'8
   e'8
   f'8
}
```

The accordion targets piano staff context by default.

instrumenttools.AltoFlute

class `abjad.tools.instrumenttools.`**`AltoFlute`**(*instrument_name='Alto Flute', short_instrument_name='Alt. Fl.', target_context=None*)

Bases: `abjad.tools.instrumenttools.Flute.Flute.Flute`

Abjad model of the alto flute:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")

abjad> instrumenttools.AltoFlute( )(staff)
AltoFlute('Alto Flute', 'Alt. Fl.')

abjad> f(staff)
\new Staff {
   \set Staff.instrumentName = \markup { Alto Flute }
   \set Staff.shortInstrumentName = \markup { Alt. Fl. }
   c'8
   d'8
   e'8
   f'8
}
```

The alto flute targets staff context by default.

instrumenttools.BassClarinet

class `abjad.tools.instrumenttools.`**`BassClarinet`**(*instrument_name='Bass Clarinet', short_instrument_name='Bass Cl.', target_context=None*)

Bases: `abjad.tools.instrumenttools.Clarinet.Clarinet.Clarinet` New in version 1.1.2. Abjad model of the bass clarinet:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")

abjad> instrumenttools.BassClarinet( )(staff)
BassClarinet('Bass Clarinet', 'Bass Cl.')

abjad> f(staff)
\new Staff {
   \set Staff.instrumentName = \markup { Bass Clarinet }
   \set Staff.shortInstrumentName = \markup { Bass Cl. }
   c'8
   d'8
   e'8
   f'8
}
```

The bass clarinet targets staff context by default.

instrumenttools.BassFlute

class `abjad.tools.instrumenttools.`**`BassFlute`**(*instrument_name='Bass Flute', short_instrument_name='Bass Fl.', target_context=None*)

Bases: `abjad.tools.instrumenttools.Flute.Flute.Flute` New in version 1.1.2. Abjad model of the bass flute:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")

abjad> instrumenttools.BassFlute( )(staff)
BassFlute('Bass Flute', 'Bass Fl.')
```

```
abjad> f(staff)
\new Staff {
   \set Staff.instrumentName = \markup { Bass Flute }
   \set Staff.shortInstrumentName = \markup { Bass Fl. }
   c'8
   d'8
   e'8
   f'8
}
```

The bass flute targets staff context by default.

instrumenttools.Bassoon

class `abjad.tools.instrumenttools.`**`Bassoon`**(*instrument_name='Bassoon'*, *short_instrument_name='Bsn.'*, *target_context=None*)

Bases: `abjad.tools.instrumenttools._DoubleReedInstrument._DoubleReedInstrument._DoubleRee`

New in version 1.1.2. Abjad model of the bassoon:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> contexttools.ClefMark('bass')(staff)
ClefMark('bass')(Staff{4})

abjad> instrumenttools.Bassoon( )(staff)
Bassoon('Bassoon', 'Bsn.')

abjad> f(staff)
\new Staff {
   \clef "bass"
   \set Staff.instrumentName = \markup { Bassoon }
   \set Staff.shortInstrumentName = \markup { Bsn. }
   c'8
   d'8
   e'8
   f'8
}
```

The bassoon targets staff context by default.

instrumenttools.Cello

class `abjad.tools.instrumenttools.`**`Cello`**(*instrument_name='Cello'*, *short_instrument_name='Vc.'*, *target_context=None*)

Bases: `abjad.tools.instrumenttools._StringInstrument._StringInstrument._StringInstrument`

New in version 1.1.2. Abjad model of the cello:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> contexttools.ClefMark('bass')(staff)
ClefMark('bass')(Staff{4})

abjad> instrumenttools.Cello( )(staff)
Cello('Cello', 'Vc.')

abjad> f(staff)
\new Staff {
```

```
    \clef "bass"
    \set Staff.instrumentName = \markup { Cello }
    \set Staff.shortInstrumentName = \markup { Vc. }
    c'8
    d'8
    e'8
    f'8
}
```

The cello targets staff context by default.

instrumenttools.Clarinet

class `abjad.tools.instrumenttools.`**`Clarinet`**(*instrument_name='Clarinet'*, *short_instrument_name='Cl.'*, *target_context=None*)

Bases: `abjad.tools.instrumenttools._SingleReedInstrument._SingleReedInstrument._SingleRee`

New in version 1.1.2. Abjad model of the B-flat clarinet:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")

abjad> instrumenttools.Clarinet( )(staff)
Clarinet('Clarinet', 'Cl.')

abjad> f(staff)
\new Staff {
    \set Staff.instrumentName = \markup { Clarinet }
    \set Staff.shortInstrumentName = \markup { Cl. }
    c'8
    d'8
    e'8
    f'8
}
```

The clarinet targets staff context by default.

instrumenttools.Contrabass

class `abjad.tools.instrumenttools.`**`Contrabass`**(*instrument_name='Contrabass'*, *short_instrument_name='Vb.'*, *target_context=None*)

Bases: `abjad.tools.instrumenttools._StringInstrument._StringInstrument._StringInstrument`

New in version 1.1.2. Abjad model of the contrabass:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> contexttools.ClefMark('bass')(staff)
ClefMark('bass')(Staff{4})

abjad> instrumenttools.Contrabass( )(staff)
Contrabass('Contrabass', 'Vb.')

abjad> f(staff)
\new Staff {
    \clef "bass"
    \set Staff.instrumentName = \markup { Contrabass }
    \set Staff.shortInstrumentName = \markup { Vb. }
```

```
    c'8
    d'8
    e'8
    f'8
}
```

The contrabass targets staff context by default.

instrumenttools.ContrabassFlute

class `abjad.tools.instrumenttools.`**`ContrabassFlute`**(*instrument_name='Contrabass Flute'*, *short_instrument_name='Cbass Fl.'*, *target_context=None*)

Bases: `abjad.tools.instrumenttools.Flute.Flute.Flute` New in version 1.1.2. Abjad model of the contrabass flute:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")

abjad> instrumenttools.ContrabassFlute( )(staff)
ContrabassFlute('Contrabass Flute', 'Cbass Fl.')

abjad> f(staff)
\new Staff {
    \set Staff.instrumentName = \markup { Contrabass Flute }
    \set Staff.shortInstrumentName = \markup { Cbass Fl. }
    c'8
    d'8
    e'8
    f'8
}
```

The contrabass flute targets staff context by default.

instrumenttools.Contrabassoon

class `abjad.tools.instrumenttools.`**`Contrabassoon`**(*instrument_name='Contrabassoon'*, *short_instrument_name='Contrabsn.'*, *target_context=None*)

Bases: `abjad.tools.instrumenttools.Bassoon.Bassoon.Bassoon` New in version 1.1.2. Abjad model of the contrabassoon:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> contexttools.ClefMark('bass')(staff)
ClefMark('bass')(Staff{4})

abjad> instrumenttools.Contrabassoon( )(staff)
Contrabassoon('Contrabassoon', 'Contrabsn.')

abjad> f(staff)
\new Staff {
    \clef "bass"
    \set Staff.instrumentName = \markup { Contrabassoon }
    \set Staff.shortInstrumentName = \markup { Contrabsn. }
    c'8
    d'8
    e'8
```

```
    f'8
}
```

The contrabassoon targets staff context by default.

instrumenttools.EFlatClarinet

class `abjad.tools.instrumenttools.`**`EFlatClarinet`**(*instrument_name='Clarinet in E-flat', short_instrument_name='Cl. E-flat', target_context=None*)

Bases: `abjad.tools.instrumenttools.Clarinet.Clarinet.Clarinet` New in version 1.1.2. Abjad model of the E-flat clarinet:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")

abjad> instrumenttools.EFlatClarinet( )(staff)
EFlatClarinet('Clarinet in E-flat', 'Cl. E-flat')

abjad> f(staff)
\new Staff {
    \set Staff.instrumentName = \markup { Clarinet in E-flat }
    \set Staff.shortInstrumentName = \markup { Cl. E-flat }
    c'8
    d'8
    e'8
    f'8
}
```

The E-flat clarinet targets staff context by default.

instrumenttools.EnglishHorn

class `abjad.tools.instrumenttools.`**`EnglishHorn`**(*instrument_name='English Horn', short_instrument_name='Eng. hn.', target_context=None*)

Bases: `abjad.tools.instrumenttools.Oboe.Oboe.Oboe` New in version 1.1.2. Abjad model of the English horn:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")

abjad> instrumenttools.EnglishHorn( )(staff)
EnglishHorn('English Horn', 'Eng. hn.')

abjad> f(staff)
\new Staff {
    \set Staff.instrumentName = \markup { English Horn }
    \set Staff.shortInstrumentName = \markup { Eng. hn. }
    c'8
    d'8
    e'8
    f'8
}
```

The English horn targets staff context by default.

instrumenttools.Flute

class `abjad.tools.instrumenttools.`**`Flute`**(*instrument_name='Flute', short_instrument_name='Fl.', target_context=None*)

Bases: `abjad.tools.instrumenttools._WindInstrument._WindInstrument._WindInstrument`

New in version 1.1.2. Abjad model of the flute:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")

abjad> instrumenttools.Flute( )(staff)
Flute('Flute', 'Fl.')

abjad> f(staff)
\new Staff {
   \set Staff.instrumentName = \markup { Flute }
   \set Staff.shortInstrumentName = \markup { Fl. }
   c'8
   d'8
   e'8
   f'8
}
```

The flute targets staff context by default.

instrumenttools.FrenchHorn

class `abjad.tools.instrumenttools.`**`FrenchHorn`**(*instrument_name='French Horn', short_instrument_name='Fr. hn.', target_context=None*)

Bases: `abjad.tools.instrumenttools._BrassInstrument._BrassInstrument._BrassInstrument`, `abjad.tools.instrumenttools._WindInstrument._WindInstrument._WindInstrument`

New in version 1.1.2. Abjad model of the French horn:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")

abjad> instrumenttools.FrenchHorn( )(staff)
FrenchHorn('French Horn', 'Fr. hn.')

abjad> f(staff)
\new Staff {
   \set Staff.instrumentName = \markup { French Horn }
   \set Staff.shortInstrumentName = \markup { Fr. hn. }
   c'8
   d'8
   e'8
   f'8
}
```

The French horn targets staff context by default.

instrumenttools.Glockenspiel

class `abjad.tools.instrumenttools.`**`Glockenspiel`**(*instrument_name='Glockenspiel', short_instrument_name='Gkspl.', target_context=None*)

Bases: `abjad.tools.instrumenttools._PercussionInstrument._PercussionInstrument._Percussio`

New in version 1.1.2. Abjad model of the glockenspiel:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")

abjad> instrumenttools.Glockenspiel( )(staff)
Glockenspiel('Glockenspiel', 'Gkspl.')

abjad> f(staff)
\new Staff {
   \set Staff.instrumentName = \markup { Glockenspiel }
   \set Staff.shortInstrumentName = \markup { Gkspl. }
   c'8
   d'8
   e'8
   f'8
}
```

The glockenspiel targets staff context by default.

instrumenttools.Guitar

class `abjad.tools.instrumenttools.`**`Guitar`**(*instrument_name='Guitar'*, *short_instrument_name='Gt.'*, *target_context=None*)

Bases: `abjad.tools.instrumenttools._StringInstrument._StringInstrument._StringInstrument`

New in version 1.1.2. Abjad model of the guitar:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")

abjad> instrumenttools.Guitar( )(staff)
Guitar('Guitar', 'Gt.')

abjad> f(staff)
\new Staff {
   \set Staff.instrumentName = \markup { Guitar }
   \set Staff.shortInstrumentName = \markup { Gt. }
   c'8
   d'8
   e'8
   f'8
}
```

The guitar targets staff context by default.

instrumenttools.Harp

class `abjad.tools.instrumenttools.`**`Harp`**(*instrument_name='Harp'*, *short_instrument_name='Hp.'*, *target_context=None*)

Bases: `abjad.tools.instrumenttools._StringInstrument._StringInstrument._StringInstrument`

New in version 1.1.2. Abjad model of the harp:

```
abjad> piano_staff = scoretools.PianoStaff([Staff("c'8 d'8 e'8 f'8"), Staff("c'4 b4")])

abjad> instrumenttools.Harp( )(piano_staff)
Harp('Harp', 'Hp.')

abjad> f(piano_staff)
\new PianoStaff <<
```

```
    \set PianoStaff.instrumentName = \markup { Harp }
    \set PianoStaff.shortInstrumentName = \markup { Hp. }
    \new Staff {
        c'8
        d'8
        e'8
        f'8
    }
    \new Staff {
        c'4
        b4
    }
>>
```

The harp targets piano staff context by default.

instrumenttools.Marimba

class `abjad.tools.instrumenttools.`**`Marimba`**(*instrument_name='Marimba'*, *short_instrument_name='Mb.'*, *target_context=None*)

Bases: `abjad.tools.instrumenttools._PercussionInstrument._PercussionInstrument._Percussic`

New in version 1.1.2. Abjad model of the marimba:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")

abjad> instrumenttools.Marimba( )(staff)
Marimba('Marimba', 'Mb.')

abjad> f(staff)
\new Staff {
    \set Staff.instrumentName = \markup { Marimba }
    \set Staff.shortInstrumentName = \markup { Mb. }
    c'8
    d'8
    e'8
    f'8
}
```

The marimba targets staff context by default.

instrumenttools.Oboe

class `abjad.tools.instrumenttools.`**`Oboe`**(*instrument_name='Oboe'*, *short_instrument_name='Ob.'*, *target_context=None*)

Bases: `abjad.tools.instrumenttools._DoubleReedInstrument._DoubleReedInstrument._DoubleRee`

New in version 1.1.2. Abjad model of the oboe:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")

abjad> instrumenttools.Oboe( )(staff)
Oboe('Oboe', 'Ob.')

abjad> f(staff)
\new Staff {
    \set Staff.instrumentName = \markup { Oboe }
```

```
    \set Staff.shortInstrumentName = \markup { Ob. }
    c'8
    d'8
    e'8
    f'8
}
```

The oboe targets staff context by default.

instrumenttools.Piano

class `abjad.tools.instrumenttools.`**`Piano`**(*instrument_name='Piano'*, *short_instrument_name='Pf.'*, *target_context=None*)

Bases: `abjad.tools.instrumenttools._KeyboardInstrument._KeyboardInstrument._KeyboardInst`

New in version 1.1.2. Abjad model of the piano:

```
abjad> piano_staff = scoretools.PianoStaff([Staff("c'8 d'8 e'8 f'8"), Staff("c'4 b4")])

abjad> instrumenttools.Piano( )(piano_staff)
Piano('Piano', 'Pf.')

abjad> f(piano_staff)
\new PianoStaff <<
    \set PianoStaff.instrumentName = \markup { Piano }
    \set PianoStaff.shortInstrumentName = \markup { Pf. }
    \new Staff {
        c'8
        d'8
        e'8
        f'8
    }
    \new Staff {
        c'4
        b4
    }
>>
```

The piano target piano staff context by default.

instrumenttools.Piccolo

class `abjad.tools.instrumenttools.`**`Piccolo`**(*instrument_name='Piccolo'*, *short_instrument_name='Picc.'*, *target_context=None*)

Bases: `abjad.tools.instrumenttools.Flute.Flute.Flute` New in version 1.1.2. Abjad model of the piccolo:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")

abjad> instrumenttools.Piccolo( )(staff)
Piccolo('Piccolo', 'Picc.')

abjad> f(staff)
\new Staff {
    \set Staff.instrumentName = \markup { Piccolo }
    \set Staff.shortInstrumentName = \markup { Picc. }
```

```
    c'8
    d'8
    e'8
    f'8
}
```

The piccolo targets staff context by default.

instrumenttools.Trombone

class `abjad.tools.instrumenttools.`**`Trombone`**(*instrument_name='Trombone'*, *short_instrument_name='Trb.'*, *target_context=None*)

Bases: `abjad.tools.instrumenttools._BrassInstrument._BrassInstrument._BrassInstrument`

New in version 1.1.2. Abjad model of the trombone:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> contexttools.ClefMark('bass')(staff)
ClefMark('bass')(Staff{4})

abjad> instrumenttools.Trombone( )(staff)
Trombone('Trombone', 'Trb.')

abjad> f(staff)
\new Staff {
    \clef "bass"
    \set Staff.instrumentName = \markup { Trombone }
    \set Staff.shortInstrumentName = \markup { Trb. }
    c'8
    d'8
    e'8
    f'8
}
```

The trombone targets staff context by default.

instrumenttools.Trumpet

class `abjad.tools.instrumenttools.`**`Trumpet`**(*instrument_name='Trumpet'*, *short_instrument_name='Tp.'*, *target_context=None*)

Bases: `abjad.tools.instrumenttools._BrassInstrument._BrassInstrument._BrassInstrument`

New in version 1.1.2. Abjad model of the trumpet:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")

abjad> instrumenttools.Trumpet( )(staff)
Trumpet('Trumpet', 'Tp.')

abjad> f(staff)
\new Staff {
    \set Staff.instrumentName = \markup { Trumpet }
    \set Staff.shortInstrumentName = \markup { Tp. }
    c'8
    d'8
    e'8
```

```
    f'8
}
```

The trumpet targets staff context by default.

instrumenttools.Tuba

class `abjad.tools.instrumenttools.`**`Tuba`**(*instrument_name='Tuba', short_instrument_name='Tb.', target_context=None*)

Bases: `abjad.tools.instrumenttools._BrassInstrument._BrassInstrument._BrassInstrument`

New in version 1.1.2. Abjad model of the tuba:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> contexttools.ClefMark('bass')(staff)
ClefMark('bass')(Staff{4})

abjad> instrumenttools.Tuba( )(staff)
Tuba('Tuba', 'Tb.')

abjad> f(staff)
\new Staff {
    \clef "bass"
    \set Staff.instrumentName = \markup { Tuba }
    \set Staff.shortInstrumentName = \markup { Tb. }
    c'8
    d'8
    e'8
    f'8
}
```

The tuba targets staff context by default.

instrumenttools.UntunedPercussion

class `abjad.tools.instrumenttools.`**`UntunedPercussion`**(*instrument_name='Percussion', short_instrument_name='Perc.', target_context=None*)

Bases: `abjad.tools.instrumenttools._PercussionInstrument._PercussionInstrument._Percussio`

New in version 1.1.2. Abjad model of untuned percussion:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")

abjad> instrumenttools.UntunedPercussion( )(staff)
UntunedPercussion('Percussion', 'Perc.')

abjad> f(staff)
\new Staff {
    \set Staff.instrumentName = \markup { Percussion }
    \set Staff.shortInstrumentName = \markup { Perc. }
    c'8
    d'8
    e'8
    f'8
}
```

Untuned percussion targets the staff context by default.

instrumenttools.Vibraphone

class `abjad.tools.instrumenttools.`**`Vibraphone`**(*instrument_name='Vibraphone'*, *short_instrument_name='Vibr.'*, *target_context=None*)

Bases: `abjad.tools.instrumenttools._PercussionInstrument._PercussionInstrument._Percussio`

New in version 1.1.2. Abjad model of the vibraphone:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")

abjad> instrumenttools.Vibraphone( )(staff)
Vibraphone('Vibraphone', 'Vibr.')

abjad> f(staff)
\new Staff {
   \set Staff.instrumentName = \markup { Vibraphone }
   \set Staff.shortInstrumentName = \markup { Vibr. }
   c'8
   d'8
   e'8
   f'8
}
```

The vibraphone targets staff context by default.

instrumenttools.Viola

class `abjad.tools.instrumenttools.`**`Viola`**(*instrument_name='Viola'*, *short_instrument_name='Va.'*, *target_context=None*)

Bases: `abjad.tools.instrumenttools._StringInstrument._StringInstrument._StringInstrument`

New in version 1.1.2. Abjad model of the viola:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> contexttools.ClefMark('alto')(staff)
ClefMark('alto')(Staff{4})

abjad> instrumenttools.Viola( )(staff)
Viola('Viola', 'Va.')

abjad> f(staff)
\new Staff {
   \clef "alto"
   \set Staff.instrumentName = \markup { Viola }
   \set Staff.shortInstrumentName = \markup { Va. }
   c'8
   d'8
   e'8
   f'8
}
```

The viola targets staff context by default.

instrumenttools.Violin

class `abjad.tools.instrumenttools.`**`Violin`**(*instrument_name='Violin'*, *short_instrument_name='Vn.'*, *target_context=None*)

Bases: `abjad.tools.instrumenttools._StringInstrument._StringInstrument._StringInstrument`

New in version 1.1.2. Abjad model of the violin:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")

abjad> instrumenttools.Violin( )(staff)
Violin('Violin', 'Vn.')

abjad> f(staff)
\new Staff {
   \set Staff.instrumentName = \markup { Violin }
   \set Staff.shortInstrumentName = \markup { Vn. }
   c'8
   d'8
   e'8
   f'8
}
```

The violin targets staff context by default.

instrumenttools.Xylophone

class `abjad.tools.instrumenttools.`**`Xylophone`**(*instrument_name='Xylophone'*, *short_instrument_name='Xyl.'*, *target_context=None*)

Bases: `abjad.tools.instrumenttools._PercussionInstrument._PercussionInstrument._Percussio`

New in version 1.1.2. Abjad model of the xylphone:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")

abjad> instrumenttools.Xylophone( )(staff)
Xylophone('Xylophone', 'Xyl.')

abjad> f(staff)
\new Staff {
   \set Staff.instrumentName = \markup { Xylophone }
   \set Staff.shortInstrumentName = \markup { Xyl. }
   c'8
   d'8
   e'8
   f'8
}
```

The xylophone targets staff context by default.

instrumenttools.get_effective_instrument

`abjad.tools.instrumenttools.`**`get_effective_instrument`**(*component*)

New in version 1.1.2. Get effective instrument from *component*:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> instrumenttools.Flute( )(staff)
Flute('Flute', 'Fl.')

abjad> f(staff)
\new Staff {
   \set Staff.instrumentName = \markup { Flute }
   \set Staff.shortInstrumentName = \markup { Fl. }
   c'8
   d'8
   e'8
   f'8
}

abjad> instrumenttools.get_effective_instrument(staff[0])
Flute('Flute', 'Fl.')
```

Return instrument or none.

instrumenttools.iterate_notes_and_chords_in_expr_outside_traditional_instrument_ranges

abjad.tools.instrumenttools.**iterate_notes_and_chords_in_expr_outside_traditional_instrument**
New in version 1.1.2. Iterate notes and chords in *expr* outside traditional instrument ranges:

```
abjad> staff = Staff("c'8 r8 <d fs>8 r8")
abjad> instrumenttools.Violin( )(staff)
Violin('Violin', 'Vn.')

abjad> for note_or_chord in instrumenttools.iterate_notes_and_chords_in_expr_outside_traditional
...   note_or_chord
Chord('<d fs>8')
```

Return generator.

instrumenttools.notes_and_chords_in_expr_are_on_expected_clefs

abjad.tools.instrumenttools.**notes_and_chords_in_expr_are_on_expected_clefs**(*expr*, *percussion_clef_is_allowed=*
New in version 1.1.2. True when notes and chords in *expr* are on expected clefs:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> contexttools.ClefMark('treble')(staff)
ClefMark('treble')(Staff{4})
abjad> instrumenttools.Violin( )(staff)
Violin('Violin', 'Vn.')

abjad> instrumenttools.notes_and_chords_in_expr_are_on_expected_clefs(staff)
True
```

False otherwise:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> contexttools.ClefMark('alto')(staff)
ClefMark('alto')(Staff{4})
```

```
abjad> instrumenttools.Violin( )(staff)
Violin('Violin', 'Vn.')

abjad> instrumenttools.notes_and_chords_in_expr_are_on_expected_clefs(staff)
False
```

Allow percussion clef when *percussion_clef_is_allowed* is true:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> contexttools.ClefMark('percussion')(staff)
ClefMark('percussion')(Staff{4})
abjad> instrumenttools.Violin( )(staff)
Violin('Violin', 'Vn.')

abjad> f(staff)
\new Staff {
   \clef "percussion"
   \set Staff.instrumentName = \markup { Violin }
   \set Staff.shortInstrumentName = \markup { Vn. }
   c'8
   d'8
   e'8
   f'8
}

abjad> instrumenttools.notes_and_chords_in_expr_are_on_expected_clefs(staff, percussion_clef_is_
True
```

Disallow percussion clef when *percussion_clef_is_allowed* is false:

```
abjad> instrumenttools.notes_and_chords_in_expr_are_on_expected_clefs(staff, percussion_clef_is_
False
```

Return boolean.

instrumenttools.notes_and_chords_in_expr_are_within_traditional_instrument_ranges

abjad.tools.instrumenttools.**notes_and_chords_in_expr_are_within_traditional_instrument_rang**

New in version 1.1.2. True when notes and chords in *expr* are within traditional instrument ranges:

```
abjad> staff = Staff("c'8 r8 <d' fs'>8 r8")
abjad> instrumenttools.Violin( )(staff)
Violin('Violin', 'Vn.')

abjad> instrumenttools.notes_and_chords_in_expr_are_within_traditional_instrument_ranges(staff)
True
```

False otherwise:

```
abjad> staff = Staff("c'8 r8 <d fs>8 r8")
abjad> instrumenttools.Violin( )(staff)
Violin('Violin', 'Vn.')

abjad> instrumenttools.notes_and_chords_in_expr_are_within_traditional_instrument_ranges(staff)
False
```

Return boolean.

instrumenttools.transpose_notes_and_chords_in_expr_from_fingered_pitch_to_sounding_pitch

abjad.tools.instrumenttools.**transpose_notes_and_chords_in_expr_from_fingered_pitch_to_sounc**

New in version 1.1.2. Transpose notes and chords in *expr* from sounding pitch to fingered pitch:

```
abjad> staff = Staff("<c' e' g'>4 d'4 r4 e'4")
abjad> instrumenttools.Clarinet( )(staff)
Clarinet('Clarinet', 'Cl.')

abjad> f(staff)
\new Staff {
   \set Staff.instrumentName = \markup { Clarinet }
   \set Staff.shortInstrumentName = \markup { Cl. }
   <c' e' g'>4
   d'4
   r4
   e'4
}

abjad> for leaf in staff.leaves:
...    leaf.written_pitch_indication_is_at_sounding_pitch = False

abjad> instrumenttools.transpose_notes_and_chords_in_expr_from_fingered_pitch_to_sounding_pitch(

abjad> f(staff)
\new Staff {
   \set Staff.instrumentName = \markup { Clarinet }
   \set Staff.shortInstrumentName = \markup { Cl. }
   <bf d' f'>4
   c'4
   r4
   d'4
}
```

Return none.

instrumenttools.transpose_notes_and_chords_in_expr_from_sounding_pitch_to_fingered_pitch

abjad.tools.instrumenttools.**transpose_notes_and_chords_in_expr_from_sounding_pitch_to_finge**

New in version 1.1.2. Transpose notes and chords in *expr* from sounding pitch to fingered pitch:

```
abjad> staff = Staff("<c' e' g'>4 d'4 r4 e'4")
abjad> instrumenttools.Clarinet( )(staff)
Clarinet('Clarinet', 'Cl.')

abjad> f(staff)
\new Staff {
   \set Staff.instrumentName = \markup { Clarinet }
   \set Staff.shortInstrumentName = \markup { Cl. }
   <c' e' g'>4
   d'4
   r4
   e'4
}

abjad> instrumenttools.transpose_notes_and_chords_in_expr_from_sounding_pitch_to_fingered_pitch(
```

```
abjad> f(staff)
\new Staff {
   \set Staff.instrumentName = \markup { Clarinet }
   \set Staff.shortInstrumentName = \markup { Cl. }
   <d' fs' a'>4
   e'4
   r4
   fs'4
}
```

Return none.

iotools

iotools.clear_terminal

`abjad.tools.iotools.`**`clear_terminal`**`()`

New in version 1.1.2. Run `clear` if OS is POSIX-compliant (UNIX / Linux / MacOS).

Run `cls` if OS is not POSIX-compliant (Windows):

```
abjad> iotools.clear_terminal( )
```

Return none.

iotools.f

`abjad.tools.iotools.`**`f`**`(`*expr*`)`

Format *expr* and print to standard out:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")

abjad> f(staff)
\new Staff {
        c'8
        d'8
        e'8
        f'8
}
```

Return none.

iotools.format_input_lines_as_doc_string

`abjad.tools.iotools.`**`format_input_lines_as_doc_string`**`(`*input_lines*, *tab_width=3*`)`

New in version 1.1.2. Format *input_lines* as doc string.

Format expressions intelligently.

Treat blank lines intelligently.

Capture hash-suffixed line output.

Use when writing docstrings.

Example skipped because docstring goes crazy on example input.

iotools.format_input_lines_as_regression_test

`abjad.tools.iotools.`**`format_input_lines_as_regression_test`**`(`*input_lines*, *tab_width=3*`)`

New in version 1.1.2. Format *input_lines* as regression test:

```
abjad> input_lines = '''
... staff = Staff(macros.scale(4))
... spannertools.BeamSpanner(staff.leaves)
... f(staff)
...
... tuplettools.FixedDurationTuplet((2, 8), staff[:3])
... f(staff)
... '''
abjad> iotools.format_input_lines_as_regression_test(input_lines) # doctest: +SKIP
```

```
staff = Staff(macros.scale(4))
spannertools.BeamSpanner(staff.leaves)

r'''
\new Staff {
     c'8 [
     d'8
     e'8
     f'8 ]
}
'''

tuplettools.FixedDurationTuplet((2, 8), staff[:3])

r'''
\new Staff {
     \times 2/3 {
             c'8 [
             d'8
             e'8
     }
     f'8 ]
}

assert componenttools.is_well_formed_component(staff)
assert staff.format == "\\new Staff {\n\t\\times 2/3 {\n\t\tc'8 [\n\t\td'8\n\t\te'8\n\t}\n\tf
'''
```

Format expressions intelligently.

Treat blank lines intelligently.

Remove line-final hash characters.

Used when writing tests.

iotools.get_last_output_file_name

`abjad.tools.iotools.`**`get_last_output_file_name`**`()`

Get last output file name like `6222.ly`.

Return string.

iotools.get_next_output_file_name

`abjad.tools.iotools.get_next_output_file_name()`

Get next output file name like `6223.ly`.

Return string.

iotools.log

`abjad.tools.iotools.log()`

Open the LilyPond log file in the `vi`:

```
abjad> iotools.log( ) ## doctest: +SKIP

GNU LilyPond 2.12.2
Processing ‘0440.ly’
Parsing...
Interpreting music...
Preprocessing graphical objects...
Finding the ideal number of pages...
Fitting music on 1 page...
Drawing systems...
Layout output to ‘0440.ps’...
Converting to ‘./0440.pdf’...
```

Exit `vi` in the usual way with `:q` or equivalent to return to the Abjad interpreter.

Return none.

iotools.ly

`abjad.tools.iotools.ly(target=-1)`

Open the last LilyPond output file in `vi`:

```
abjad> iotools.ly( ) # doctest: +SKIP

% Abjad revision 2162
% 2009-05-31 14:29

\version "2.12.2"
\include "english.ly"
\include "/Path/to/abjad/trunk/abjad/cfg/abjad.scm"

{
   c’4
}
```

Open the next-to-last LilyPond output file in `vi`:

```
abjad> iotools.ly(-2) # doctest: +SKIP
```

Exit `vi` in the usual way with `:q` or equivalent.

Return none.

iotools.parse_lilypond_input_string

abjad.tools.iotools.**parse_lilypond_input_string**(*note_entry_string*)

New in version 1.1.2. Parse LilyPond *note_entry_string*:

```
abjad> note_entry_string = "g'2 a'2 g'4. fs'8 e'4 d'4"
abjad> iotools.parse_lilypond_input_string(note_entry_string)
{g'2, a'2, g'4., fs'8, e'4, d'4}
```

Return container of note, rest and chord instances.

Handle simple beaming, slurs and articulations.

Do not parse tuplets, measures or other complex LilyPond input.

iotools.pdf

abjad.tools.iotools.**pdf**(*target=-1*)

Open the last PDF generated by Abjad with `iotools.pdf( )`.

Open the next-to-last PDF generated by Abjad with `iotools.pdf(-2)`.

Return none.

Abjad writes PDFs to the `~/.abjad/output` directory by default.

You may change this by setting the `abjad_output` variable in the `config.py` file.

iotools.play

abjad.tools.iotools.**play**(*expr*)

Play *expr*:

```
abjad> note = Note(0, (1, 4))
abjad> iotools.play(note) # doctest: +SKIP
```

This input renders and then opens a one-note MIDI file.

Abjad outputs MIDI files of the format `filename.mid` under Windows.

Abjad outputs MIDI files of the format `filename.midi` under other operating systems.

iotools.profile_expr

abjad.tools.iotools.**profile_expr**(*expr*, *sort_by='cum'*, *num_lines=12*, *strip_dirs=True*)

Profile *expr*:

```
abjad> iotools.profile_expr('Staff(notetools.make_repeated_notes(8))') # doctest: +SKIP
Tue Apr  5 20:32:40 2011    _tmp_abj_profile

         2852 function calls (2829 primitive calls) in 0.006 CPU seconds

   Ordered by: cumulative time
   List reduced from 118 to 12 due to restriction <12>

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.006    0.006 <string>:1(<module>)
        1    0.000    0.000    0.003    0.003 make_repeated_notes.py:5(make_repeated_notes)
```

```
       1    0.001    0.001    0.003    0.003 make_notes.py:12(make_notes)
       1    0.000    0.000    0.003    0.003 Staff.py:21(__init__)
       1    0.000    0.000    0.003    0.003 _Context.py:11(__init__)
       1    0.000    0.000    0.003    0.003 Container.py:23(__init__)
       1    0.000    0.000    0.003    0.003 Container.py:271(_initialize_music)
       2    0.000    0.000    0.002    0.001 all_are_thread_contiguous_components.py:9(all_are_
      52    0.001    0.000    0.002    0.000 component_to_thread_signature.py:5(component_to_th
       1    0.000    0.000    0.002    0.002 _construct_unprolated_notes.py:4(_construct_unprol
       8    0.000    0.000    0.002    0.000 _construct_tied_note.py:5(_construct_tied_note)
       8    0.000    0.000    0.002    0.000 _construct_tied_leaf.py:5(_construct_tied_leaf)
```

Function wraps the built-in Python `cProfile` module.

Set *expr* to any string of Abjad input.

Set *sort_by* to *'cum'*, *'time'* or *'calls'*.

Set *num_lines* to any positive integer.

Set *strip_dirs* to `True` to strip directory names from output lines.

Note: This function fails on some Linux distros. Some Linux distributions do not include the Python `pstats` module.

Note: This function creates the file `_tmp_abj_profile` in the directory from which it is run.

Note: For information on reading the output of the different Python profilers, see the Python docs.

Changed in version 1.1.2: renamed `check.profile( )` to `iotools.profile_expr( )`.

iotools.redo

`abjad.tools.iotools.`**`redo`**`(`*target=-1*, *lily_time=10*`)`

Rerender the last `.ly` file created in Abjad and then show the resulting PDF:

```
abjad> iotools.redo( ) # doctest: +SKIP
```

Rerender the next-to-last `.ly` file created in Abjad and then show the resulting PDF:

```
abjad> iotools.redo(-2) # doctest: +SKIP
```

Return none.

iotools.remove_abjad_pyc_files

`abjad.tools.iotools.`**`remove_abjad_pyc_files`**`()`

Remove `.pyc` files from Abjad source tree:

```
abjad> iotools.remove_abjad_pyc_files( ) # doctest: +SKIP
```

Return none.

iotools.save_last_ly_as

abjad.tools.iotools.**save_last_ly_as**(*file_name*)
New in version 1.1.2. Save last ly file as *file_name*:

```
abjad> iotools.save_last_ly_as('/project/output/example-1.ly') # doctest: +SKIP
```

Return none.

iotools.save_last_pdf_as

abjad.tools.iotools.**save_last_pdf_as**(*file_name*)
New in version 1.1.2. Save last PDF as *file_name*:

```
abjad> iotools.save_last_pdf_as('/project/output/example-1.pdf') # doctest: +SKIP
```

Return none.

iotools.show

abjad.tools.iotools.**show**(*expr*, *template=None*, *return_timing=False*, *suppress_pdf=False*)
Show *expr*:

```
abjad> note = Note(0, (1, 4))
abjad> show(note) # doctest: +SKIP
```

Show *expr* with *template*:

```
abjad> note = Note(0, (1, 4))
abjad> show(note, template = 'tangiers') # doctest: +SKIP
```

Show *expr* and return both Abjad and LilyPond processing time in seconds:

```
abjad> staff = Staff(Note(0, (1, 4)) * 200)
abjad> show(note, return_timing = True) # doctest: +SKIP
(0, 3)
```

Return none or timing tuple.

Abjad writes LilyPond input files to the `~/.abjad/output` directory by default.

You may change this by setting the `abjad_output` variable in the `config.py` file.

iotools.underscore_delimited_lowercase_to_lowercamelcase

abjad.tools.iotools.**underscore_delimited_lowercase_to_lowercamelcase**(*string*)
New in version 1.1.2. Change underscore-delimited lowercase *string* to lowercamelcase:

```
abjad> string = 'bass_figure_alignment_positioning'
abjad> iotools.underscore_delimited_lowercase_to_lowercamelcase(string)
'bassFigureAlignmentPositioning'
```

Changed in version 1.1.2: renamed `stringtools.underscore_delimited_lowercase_to_lowercamelcase( )` to `iotools.underscore_delimited_lowercase_to_lowercamelcase( )`.

iotools.underscore_delimited_lowercase_to_uppercamelcase

`abjad.tools.iotools.`**`underscore_delimited_lowercase_to_uppercamelcase`**`(`*string*`)`

New in version 1.1.2. Change underscore-delimited lowercase *string* to uppercamelcase:

```
abjad> string = 'bass_figure_alignment_positioning'
abjad> iotools.underscore_delimited_lowercase_to_uppercamelcase(string)
'BassFigureAlignmentPositioning'
```

Changed in version 1.1.2: renamed `stringtools.underscore_delimited_lowercase_to_uppercamelcase( )` to `iotools.underscore_delimited_lowercase_to_uppercamelcase( )`.

iotools.write_expr_to_ly

`abjad.tools.iotools.`**`write_expr_to_ly`**`(`*expr*, *file_name*, *template=None*`)`

Write *expr* to *file_name*:

```
abjad> note = Note(0, (1, 4))
abjad> iotools.write_expr_to_ly(note, '/home/user/foo.ly') # doctest: +SKIP
```

Write *expr* to *file_name* with *template*:

```
abjad> note = Note(0, (1, 4))
abjad> iotools.write_expr_to_ly(note, '/home/user/foo.ly', 'paris') # doctest: +SKIP
```

Retur none. Changed in version 1.1.2: renamed `io.write_ly( )` to `io.write_expr_to_ly( )`.

iotools.write_expr_to_ly_and_to_pdf_and_show

`abjad.tools.iotools.`**`write_expr_to_ly_and_to_pdf_and_show`**`(`*expr*, *name*, *template=None*, *write=True*`)`

Write *expr* to named `.ly` and to PDF and then open the resulting PDF:

```
abjad> iotools.write_expr_to_ly_and_to_pdf_and_show(Note("c'8"), 'file_name_stem') # doctest: +S
```

Write *expr* to temporary `.ly` and to PDF and then open the resulting PDF:

```
abjad> iotools.write_expr_to_ly_and_to_pdf_and_show(Note("c'8"), 'file_name_stem', write = False
```

Return none.

The purpose of this function is to save named `.ly` and PDF output. Changed in version 1.1.2: renamed `io.write_and_show( )` to `io.write_expr_to_ly_and_to_pdf_and_show( )`.

iotools.write_expr_to_pdf

`abjad.tools.iotools.`**`write_expr_to_pdf`**`(`*expr*, *file_name*, *template=None*`)`

Write *expr* to pdf *file_name*:

```
abjad> note = Note(0, (1, 4))
abjad> iotools.write_expr_to_pdf(note, 'one_note.pdf') # doctest: +SKIP
```

Write *expr* to pdf *file_name* with *template*:

```
abjad> note = Note(0, (1, 4))
abjad> iotools.write_expr_to_pdf(note, 'one_note.pdf', 'paris') # doctest: +SKIP
```

Return none.

layouttools

layouttools.FixedStaffPositioning

class `abjad.tools.layouttools.FixedStaffPositioning`(*system_y_offsets*, *staff_alignment_offsets=None*)

Bases: `abjad.core._StrictComparator._StrictComparator._StrictComparator`, `abjad.core._Immutable._Immutable._Immutable`

Indicator object to model fixed-systems layout across an entire score. Instantiate a `FixedStaffPositioning` object with numeric indication of fixed distances between systems. Then pass to `apply_fixed_staff_positioning()`.

```
abjad> t = Staff(Measure((2, 8), notetools.make_repeated_notes(2)) * 4)
abjad> macros.diatonicize(t)
abjad> layouttools.set_line_breaks_cyclically_by_line_duration_ge(t, Duration(4, 8))
```

```
\new Staff {
                \time 2/8
                c'8
                d'8
                \time 2/8
                e'8
                f'8
                \break
                \time 2/8
                g'8
                a'8
                \time 2/8
                b'8
                c''8
                \break
}
```

```
abjad> systems = layouttools.SystemYOffsets(40, 5)
abjad> staves = layouttools.StaffAlignmentDistances(0, 15)
abjad> positioning = layouttools.FixedStaffPositioning(systems, staves)
abjad> layouttools.apply_fixed_staff_positioning(t, positioning)
```

```
\new Staff {
                \overrideProperty #"Score.NonMusicalPaperColumn"
                #'line-break-system-details
                #'((Y-offset . 20))
                \time 2/8
                c'8
                d'8
                \time 2/8
                e'8
                f'8
                \break
                \pageBreak
                \overrideProperty #"Score.NonMusicalPaperColumn"
                #'line-break-system-details
                #'((Y-offset . 20))
                \time 2/8
                g'8
```

```
                    a'8
                    \time 2/8
                    b'8
                    c''8
                    \break
}
```

Note: Staff alignment offsets and staff alignment distances are both allowed.

staff_alignment_offsets

system_y_offsets

layouttools.LayoutSchema

class abjad.tools.layouttools.**LayoutSchema**(*line_break_duration*, *system_y_offsets_tuple*, *staff_alignment_offsets_tuple*, *in_seconds=False*)

Bases: abjad.core._StrictComparator._StrictComparator._StrictComparator, abjad.core._Immutable._Immutable._Immutable

Indicator to line-break an arbitrary score and then position staves and systems regularly throughout.

Short-cut to avoid instanting SystemYOffsets and StaffAlignmentDistances by hand.

layouttools.SpacingIndication

class abjad.tools.layouttools.**SpacingIndication**(*tempo_indication*, *proportional_notation_duration*)

Bases: abjad.core._StrictComparator._StrictComparator._StrictComparator, abjad.core._Immutable._Immutable._Immutable

Spacing indication token.

LilyPond `Score.proportionalNotationDuration` will equal `proportional_notation_duration` when tempo equals `tempo_indication`.

```
abjad> tempo = contexttools.TempoMark(Duration(1, 8), 44)
abjad> spacing_indication = layouttools.SpacingIndication(tempo, Duration(1, 68))
abjad> spacing_indication
SpacingIndication(TempoMark(8, 44), 1/68)
```

Spacing indications are immutable.

normalized_spacing_duration
Read-only proportional notation duration at 60 MM.

proportional_notation_duration
LilyPond proportional notation duration context setting.

tempo_indication
Abjad tempo indication object.

layouttools.StaffAlignmentDistances

class `abjad.tools.layouttools.`**`StaffAlignmentDistances`**(**args*)

Bases: `abjad.core._StrictComparator._StrictComparator._StrictComparator`, `abjad.core._Immutable._Immutable._Immutable`

Class to model distances between staves in a system. Specify distances by hand when initializing the class. Distances may be even or uneven.

```
abjad> staves = layouttools.StaffAlignmentDistances(18, 18, 18)
```

Pass instances of this class as the second argument to `FixedStaffPositioning`.

layouttools.SystemYOffsets

class `abjad.tools.layouttools.`**`SystemYOffsets`**(*interval*, *systems_per_page*, *skip_systems_on_first_page=1*)

Bases: `abjad.core._StrictComparator._StrictComparator._StrictComparator`, `abjad.core._Immutable._Immutable._Immutable`

Used to specify systems starting at even intervals running down every page.

Set *interval* to a positive number. Set *systems_per_page* to a positive number. Set *skip_systems_on_first_page* to a positive integer less than *systems_per_page*, defaulting to `1`.

```
abjad> specification = layouttools.SystemYOffsets(38, 5) # doctest: +SKIP
SystemYOffsets([0], 44, 88, 132, 176 | 0, 44, 88, 132, 176 | ...)
```

Pass instances of this class to other layout functions.

layouttools.apply_fixed_staff_positioning

`abjad.tools.layouttools.`**`apply_fixed_staff_positioning`**(*expr*, *positioning*, *klass=<class 'abjad.components.Measure.Measure.Measure'>*)

Apply *positioning* to *expr*. Music *expr* must already be marked with line breaks.

```
abjad> t = Staff(Measure((2, 8), notetools.make_repeated_notes(2)) * 4)
abjad> macros.diatonicize(t)
abjad> layouttools.set_line_breaks_cyclically_by_line_duration_ge(t, Duration(4, 8))
abjad> print t.format # doctest: +SKIP
\new Staff {
                \time 2/8
                c'8
                d'8
                \time 2/8
                e'8
                f'8
                \break
                \time 2/8
                g'8
                a'8
                \time 2/8
                b'8
                c''8
                \break
}
```

```
abjad> systems = layouttools.SystemYOffsets(40, 5)
abjad> staves = layouttools.StaffAlignmentDistances(15)
abjad> positioning = layouttools.FixedStaffPositioning(systems, staves)
abjad> layouttools.apply_fixed_staff_positioning(t, positioning)
abjad> print t.format # doctest: +SKIP
\new Staff {
        {
                \overrideProperty #"Score.NonMusicalPaperColumn"
                #'line-break-system-details
                #'((Y-offset . 40) (alignment-offsets . (0 -15)))
                \time 2/8
                c'8
                d'8
        }
        {
                \time 2/8
                e'8
                f'8
                \break
                \noPageBreak
        }
        {
                \overrideProperty #"Score.NonMusicalPaperColumn"
                #'line-break-system-details
                #'((Y-offset . 80) (alignment-offsets . (0 -15)))
                \time 2/8
                g'8
                a'8
        }
        {
                \time 2/8
                b'8
                c''8
                \break
        }
}
```

Return none.

layouttools.apply_layout_schema

abjad.tools.layouttools.**apply_layout_schema**(*expr*, *layout_schema*, *klass=<class 'abjad.components.Measure.Measure.Measure'>*, *adjust_eol=False*, *add_empty_bars=False*)

New in version 1.1.2. Apply *layout_schema* to *expr*.

The following example line breaks every 4 eighth notes, lays out 5 systems per page, spaces systems 40 vertical spaces apart, leaves empty vertical space equivalent to a single system at the top of the first page, sets the first staff in each system to alignment distance 0 and sets the second staff in each system to alignment distance 15.

```
abjad> score = Score(2 * Staff(Measure((2, 8), notetools.make_repeated_notes(2)) * 4))
abjad> macros.diatonicize(score)
abjad> schema = layouttools.LayoutSchema(Duration(4, 8), (40, 5, 1), (15, ))
abjad> layouttools.apply_layout_schema(score[0], schema)
abjad> f(score)
\new Score <<
   \new Staff {
      \overrideProperty #"Score.NonMusicalPaperColumn"
```

```
        #'line-break-system-details
        #'((Y-offset . 40) (alignment-distances . (15)))
        {
            \time 2/8
            c'8
            d'8
        }
        {
            \time 2/8
            e'8
            f'8
        }
        \noPageBreak
        \overrideProperty #"Score.NonMusicalPaperColumn"
        #'line-break-system-details
        #'((Y-offset . 80) (alignment-distances . (15)))
        {
            \time 2/8
            g'8
            a'8
        }
        {
            \time 2/8
            b'8
            c''8
        }
    }
    \new Staff {
        {
            \time 2/8
            d''8
            e''8
        }
        {
            \time 2/8
            f''8
            g''8
        }
        {
            \time 2/8
            a''8
            b''8
        }
        {
            \time 2/8
            c'''8
            d'''8
        }
    }
>>
```

Return none.

layouttools.make_spacing_vector

`abjad.tools.layouttools.`**`make_spacing_vector`**(*basic_distance*, *minimum_distance*, *padding*, *stretchability*)

New in version 1.1.2. Make spacing vector:

```
abjad> layouttools.make_spacing_vector(0, 0, 12, 0)
SchemeVector((basic_distance . 0), (minimum_distance . 0), (padding . 12), (stretchability . 0))
```

Use to set paper block spacing attributes:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> lily_file = lilyfiletools.make_basic_lily_file(staff)
abjad> lily_file.paper_block.system_system_spacing = layouttools.make_spacing_vector(0, 0, 12, 0

abjad> f(lily_file) # doctest: +SKIP
% Abjad revision 4229
% 2011-04-07 15:19

\version "2.13.44"
\include "english.ly"
\include "/abjad/trunk/abjad/cfg/abjad.scm"

\paper {
   system-system-spacing = #'((basic_distance . 0) (minimum_distance . 0) (padding . 12) (stretc
}

\score {
   \new Staff {
      c'8
      d'8
      e'8
      f'8
   }
}
```

Return scheme vector.

layouttools.set_line_breaks_cyclically_by_line_duration_ge

`abjad.tools.layouttools.`**`set_line_breaks_cyclically_by_line_duration_ge`**(*expr*, *line_duration*, *klass=<class 'abjad.components.Measure.Me*, *adjust_eol=False*, *add_empty_bars=False*)

Iterate *klass* instances in *expr* and accumulate prolated duration. Add line break after every total less than or equal to *line_duration*.

```
abjad> t = Staff(Measure((2, 8), notetools.make_repeated_notes(2)) * 4)
abjad> macros.diatonicize(t)
abjad> print t.format
\new Staff {
   {
      \time 2/8
```

```
        c'8
        d'8
    }
    {
        \time 2/8
        e'8
        f'8
    }
    {
        \time 2/8
        g'8
        a'8
    }
    {
        \time 2/8
        b'8
        c''8
    }
}
```

```
abjad> layouttools.set_line_breaks_cyclically_by_line_duration_ge(t, Duration(4, 8))
abjad> print t.format # doctest: +SKIP
\new Staff {
                \time 2/8
                c'8
                d'8
                \time 2/8
                e'8
                f'8
                \break
                \time 2/8
                g'8
                a'8
                \time 2/8
                b'8
                c''8
                \break
}
```

Set *adjust_eol* to `True` to include a magic Scheme incantation to move end-of-line LilyPond TimeSignature and BarLine grobs to the right. Changed in version 1.1.2: renamed `layout.line_break_every_prolated( )` to `layout.set_line_breaks_cyclically_by_line_duration_ge( )`.

layouttools.set_line_breaks_cyclically_by_line_duration_in_seconds_ge

`abjad.tools.layouttools.`**`set_line_breaks_cyclically_by_line_duration_in_seconds_ge`**(*expr, line_duratio klass=<clas 'ab-jad.compone ad-just_eol=Fa add_empty_*

Iterate *klass* instances in *expr* and accumulate duration in seconds. Add line break after every total less than or equal to *line_duration*.

```
abjad> t = Staff(Measure((2, 8), notetools.make_repeated_notes(2)) * 4)
abjad> macros.diatonicize(t)
abjad> tempo_mark = contexttools.TempoMark(Duration(1, 8), 44, target_context = Staff)(t)
abjad> print t.format # doctest: +SKIP
\new Staff {
                \time 2/8
                \tempo 8=44
                c'8
                d'8
                \time 2/8
                e'8
                f'8
                \time 2/8
                g'8
                a'8
                \time 2/8
                b'8
                c''8
}
```

```
abjad> layouttools.set_line_breaks_cyclically_by_line_duration_in_seconds_ge(t, Duration(6))
abjad> print t.format # doctest: +SKIP
\new Staff {
                \time 2/8
                \tempo 8=44
                c'8
                d'8
                \time 2/8
                e'8
                f'8
                \break
                \time 2/8
                g'8
                a'8
                \time 2/8
                b'8
                c''8
}
```

Set `adjust_eol = True` to include a magic Scheme incantation to move end-of-line LilyPond TimeSignature and BarLine grobs to the right. Changed in version 1.1.2: renamed `layout.line_break_every_seconds( )` to `layout.set_line_breaks_cyclically_by_line_duration_in_seconds_ge( )`.

leaftools

leaftools.change_written_leaf_duration_and_preserve_preprolated_leaf_duration

abjad.tools.leaftools.**change_written_leaf_duration_and_preserve_preprolated_leaf_duration**(*l w t*

New in version 1.1.1. Change *leaf* written duration to *written_duration* and preserve preprolated *leaf* duration:

```
abjad> note = Note(0, (1, 4))
abjad> note.duration.written
Duration(1, 4)
abjad> note.duration.preprolated
Duration(1, 4)
```

```
abjad> leaftools.change_written_leaf_duration_and_preserve_preprolated_leaf_duration(note, Durat
Note("c'8. * 4/3")

abjad> note.duration.written
Duration(3, 16)
abjad> note.duration.preprolated
Duration(1, 4)
```

Add LilyPond multiplier where necessary.

Return *leaf*. Changed in version 1.1.2: Renamed from `leaftools.duration_rewrite( )`. `leaftools.change_written_leaf_duration_and_preserve_preprolated_leaf_duration( )`.

leaftools.color_leaf

`abjad.tools.leaftools.`**`color_leaf`**(*leaf*, *color*)

New in version 1.1.2. Color note:

```
abjad> note = Note("c'4")

abjad> leaftools.color_leaf(note, 'red')
Note("c'4")

abjad> f(note)
\once \override Accidental #'color = #red
\once \override Dots #'color = #red
\once \override NoteHead #'color = #red
c'4
```

Color rest:

```
abjad> rest = Rest('r4')

abjad> leaftools.color_leaf(rest, 'red')
Rest('r4')

abjad> f(rest)
\once \override Dots #'color = #red
\once \override Rest #'color = #red
r4
```

Color chord:

```
abjad> chord = Chord("<c' e' bf'>4")

abjad> leaftools.color_leaf(chord, 'red')
Chord("<c' e' bf'>4")

abjad> f(chord)
\once \override Accidental #'color = #red
\once \override Dots #'color = #red
\once \override NoteHead #'color = #red
<c' e' bf'>4
```

Return *leaf*.

leaftools.color_leaves_in_expr

abjad.tools.leaftools.**color_leaves_in_expr**(*expr*, *color*)

New in version 1.1.2. Color leaves in *expr*:

```
abjad> staff = Staff([Note(1, (3, 16)), Rest((3, 16)), skiptools.Skip((3, 16)), Chord([0, 1, 9],
abjad> spannertools.BeamSpanner(staff.leaves)
BeamSpanner(cs'8., r8., s8., <c' cs' a'>8.)
abjad> f(staff)
\new Staff {
   cs'8. [
   r8.
   s8.
   <c' cs' a'>8. ]
}

abjad> leaftools.color_leaves_in_expr(staff, 'red')

abjad> f(staff)
\new Staff {
   \once \override Accidental #'color = #red
   \once \override Dots #'color = #red
   \once \override NoteHead #'color = #red
   cs'8. [
   \once \override Dots #'color = #red
   \once \override Rest #'color = #red
   r8.
   s8.
   \once \override Accidental #'color = #red
   \once \override Dots #'color = #red
   \once \override NoteHead #'color = #red
   <c' cs' a'>8. ]
}
```

Return none.

leaftools.copy_written_duration_and_multiplier_from_leaf_to_leaf

abjad.tools.leaftools.**copy_written_duration_and_multiplier_from_leaf_to_leaf**(*source_leaf*, *target_leaf*)

New in version 1.1.2. Copy written duration and multiplier from *source_leaf* to *target_leaf*:

```
abjad> note = Note(0, (1, 4))
abjad> note.duration.multiplier = Duration(1, 2)
abjad> rest = Rest((1, 64))
abjad> leaftools.copy_written_duration_and_multiplier_from_leaf_to_leaf(note, rest)
Rest('r4 * 1/2')
```

Return *target_leaf*.

leaftools.divide_leaf_meiotically

abjad.tools.leaftools.**divide_leaf_meiotically**(*leaf*, *n=2*)

New in version 1.1.1. Divide *leaf* meiotically *n* times:

```
abjad> staff = Staff(macros.scale(4))
abjad> spannertools.BeamSpanner(staff.leaves)
BeamSpanner(c'8, d'8, e'8, f'8)
abjad> f(staff)
\new Staff {
   c'8 [
   d'8
   e'8
   f'8 ]
}

abjad> leaftools.divide_leaf_meiotically(staff[0], n = 4)

abjad> f(staff)
\new Staff {
   c'32 [
   c'32
   c'32
   c'32
   d'8
   e'8
   f'8 ]
}
```

Replace *leaf* with *n* new leaves.

Preserve parentage and spanners.

Allow divisions into only `1, 2, 4, 8, 16, ...` and other nonnegative integer powers of `2`.

Produce only leaves and never tuplets or other containers.

Return none.

leaftools.divide_leaves_in_expr_meiotically

`abjad.tools.leaftools.`**`divide_leaves_in_expr_meiotically`**(*expr*, *n=2*)

New in version 1.1.1. Divide leaves meiotically in *expr* *n* times:

```
abjad> staff = Staff(macros.scale(4))
abjad> spannertools.BeamSpanner(staff.leaves)
BeamSpanner(c'8, d'8, e'8, f'8)
abjad> f(staff)
\new Staff {
   c'8 [
   d'8
   e'8
   f'8 ]
}

abjad> leaftools.divide_leaves_in_expr_meiotically(staff[2:], n = 4)

abjad> f(staff)
\new Staff {
   c'8 [
   d'8
   e'32
   e'32
   e'32
```

```
      e'32
      f'32
      f'32
      f'32
      f'32 ]
   }
```

Replace every leaf in *expr* with *n* new leaves.

Preserve parentage and spanners.

Allow divisions into only `1, 2, 4, 8, 16, ...` and other nonnegative integer powers of `2`.

Produce only leaves and never tuplets or other containers.

Return none. Changed in version 1.1.2: renamed `leaftools.meiose( )` to `leaftools.divide_leaves_in_expr_meiotically( )`.

leaftools.expr_has_leaf_with_dotted_written_duration

`abjad.tools.leaftools.`**`expr_has_leaf_with_dotted_written_duration`**`(`*expr*`)`

New in version 1.1.2. True when *expr* has at least one leaf with dotted writtern duration:

```
abjad> notes = notetools.make_notes([0], [(1, 16), (2, 16), (3, 16)])
abjad> leaftools.expr_has_leaf_with_dotted_written_duration(notes)
True
```

False otherwise:

```
abjad> notes = notetools.make_notes([0], [(1, 16), (2, 16), (4, 16)])
abjad> leaftools.expr_has_leaf_with_dotted_written_duration(notes)
False
```

Return boolean.

leaftools.fuse_leaves_big_endian

`abjad.tools.leaftools.`**`fuse_leaves_big_endian`**`(`*leaves*`)`

New in version 1.1.1. Fuse thread-contiguous *leaves*:

```
abjad> staff = Staff(macros.scale(4))
abjad> leaftools.fuse_leaves_big_endian(staff[1:])
[Note("d'4.")]
abjad> f(staff)
\new Staff {
   c'8
   d'4.
}
```

Rewrite duration of first leaf in *leaves*.

Detach all leaves in *leaves* other than first leaf from score.

Return list of first leaf in *leaves*. Changed in version 1.1.2: renamed `fuse.leaves_by_reference( )` to `leaftools.fuse_leaves_big_endian( )`.

leaftools.fuse_leaves_in_container_once_by_counts_into_big_endian_notes

abjad.tools.leaftools.**fuse_leaves_in_container_once_by_counts_into_big_endian_notes**(*container*, *counts*)

New in version 1.1.1. Fuse leaves in *container* once by *counts* into big-endian notes.

leaftools.fuse_leaves_in_container_once_by_counts_into_big_endian_rests

abjad.tools.leaftools.**fuse_leaves_in_container_once_by_counts_into_big_endian_rests**(*container*, *counts*)

New in version 1.1.1. Fuse leaves in *container* once by *counts* into big-endian rests.

leaftools.fuse_leaves_in_container_once_by_counts_into_little_endian_notes

abjad.tools.leaftools.**fuse_leaves_in_container_once_by_counts_into_little_endian_notes**(*conta* *coun*

New in version 1.1.1. Fuse leaves in *container* once by *counts* into little-endian notes.

leaftools.fuse_leaves_in_container_once_by_counts_into_little_endian_rests

abjad.tools.leaftools.**fuse_leaves_in_container_once_by_counts_into_little_endian_rests**(*conta* *coun*

New in version 1.1.1. Fuse leaves in *container* once by *counts* into little-endian rests.

leaftools.fuse_leaves_in_tie_chain_by_immediate_parent_big_endian

abjad.tools.leaftools.**fuse_leaves_in_tie_chain_by_immediate_parent_big_endian**(*tie_chain*)

New in version 1.1.1. Fuse leaves in *tie_chain* by immediate parent:

```
abjad> staff = Staff(Measure((2, 8), notetools.make_repeated_notes(2)) * 2)
abjad> tietools.TieSpanner(staff.leaves)
TieSpanner(c'8, c'8, c'8, c'8)
abjad> f(staff)
\new Staff {
   {
      \time 2/8
      c'8 ~
      c'8 ~
   }
   {
      \time 2/8
      c'8 ~
      c'8
   }
}

abjad> tie_chain = tietools.get_tie_chain(staff.leaves[0])
abjad> leaftools.fuse_leaves_in_tie_chain_by_immediate_parent_big_endian(tie_chain)
[[Note("c'4")], [Note("c'4")]]

abjad> f(staff)
\new Staff {
   {
      \time 2/8
```

```
        c'4 ~
    }
    {
        \time 2/8
        c'4
    }
}
```

Return list of fused notes by parent. Changed in version 1.1.2: renamed `fuse.leaves_in_tie_chain( )` to `leaftools.fuse_leaves_in_tie_chain_by_immediate_parent_big_endian( )`.

leaftools.fuse_tied_leaves_in_components_once_by_prolated_durations_without_overhang

`abjad.tools.leaftools.`**`fuse_tied_leaves_in_components_once_by_prolated_durations_without_ove`**

New in version 1.1.1. Fuse tied leaves in *components* once by *prolated_durations* without overhang:

```
abjad> staff = Staff(notetools.make_repeated_notes(8))
abjad> tietools.TieSpanner(staff.leaves)
TieSpanner(c'8, c'8, c'8, c'8, c'8, c'8, c'8, c'8)
abjad> f(staff)
\new Staff {
   c'8 ~
   c'8 ~
   c'8 ~
   c'8 ~
   c'8 ~
   c'8 ~
   c'8 ~
   c'8
}
```

```
abjad> leaftools.fuse_tied_leaves_in_components_once_by_prolated_durations_without_overhang(staf
```

```
abjad> f(staff)
\new Staff {
   c'4. ~
   c'4. ~
   c'8 ~
   c'8
}
```

Return none. Changed in version 1.1.2: renamed `fuse.tied_leaves_by_prolated_durations( )` to `leaftools.fuse_tied_leaves_in_components_once_by_prolated_durations_without_overhang( )`.

leaftools.get_composite_offset_difference_series_from_leaves_in_expr

`abjad.tools.leaftools.`**`get_composite_offset_difference_series_from_leaves_in_expr`**`(`*expr*`)`

New in version 1.1.2. Get composite offset difference series from leaves in *expr*:

```
abjad> staff_1 = Staff([tuplettools.FixedDurationTuplet((4, 8), notetools.make_repeated_notes(3)
abjad> staff_2 = Staff(notetools.make_repeated_notes(4))
abjad> score = Score([staff_1, staff_2])
abjad> macros.diatonicize(score)
```

```
abjad> f(score)
   \new Score <<
           \new Staff {
                   \fraction \times 4/3 {
                           c'8
                           d'8
                           e'8
                   }
           }
           \new Staff {
                   f'8
                   g'8
                   a'8
                   b'8
           }
   >>
abjad> leaftools.get_composite_offset_difference_series_from_leaves_in_expr(score)
[Duration(1, 8), Duration(1, 24), Duration(1, 12), Duration(1, 12), Duration(1, 24), Duration(1,
```

Composite offset difference series defined equal to time intervals between unique start and stop offsets of leaves in *expr*.

Return list of fractions.

leaftools.get_composite_offset_series_from_leaves_in_expr

abjad.tools.leaftools.**get_composite_offset_series_from_leaves_in_expr**(*expr*)

New in version 1.1.2. Get composite offset series from leaves in *expr*:

```
abjad> staff_1 = Staff([tuplettools.FixedDurationTuplet((4, 8), notetools.make_repeated_notes(3)
abjad> staff_2 = Staff(notetools.make_repeated_notes(4))
abjad> score = Score([staff_1, staff_2])
abjad> macros.diatonicize(score)
abjad> f(score)
   \new Score <<
           \new Staff {
                   \fraction \times 4/3 {
                           c'8
                           d'8
                           e'8
                   }
           }
           \new Staff {
                   f'8
                   g'8
                   a'8
                   b'8
           }
   >>
abjad> leaftools.get_composite_offset_series_from_leaves_in_expr(score)
[Duration(0, 1), Duration(1, 8), Duration(1, 6), Duration(1, 4), Duration(1, 3), Duration(3, 8),
```

Equal to list of unique start and stop offsets of leaves in *expr*.

Return list of fractions.

leaftools.get_leaf_at_index_in_measure_number_in_expr

`abjad.tools.leaftools.`**`get_leaf_at_index_in_measure_number_in_expr`**(*expr*, *measure_number*, *leaf_index*)

New in version 1.1.2. Get leaf at *leaf_index* in *measure_number* in *expr*:

```
abjad> t = Staff(Measure((2, 8), notetools.make_repeated_notes(2)) * 3)
abjad> macros.diatonicize(t)
abjad> f(t)
\new Staff {
   {
      \time 2/8
      c'8
      d'8
   }
   {
      \time 2/8
      e'8
      f'8
   }
   {
      \time 2/8
      g'8
      a'8
   }
}
```

```
abjad> leaftools.get_leaf_at_index_in_measure_number_in_expr(t, 2, 0)
Note("e'8")
```

Return leaf or none.

leaftools.get_nth_leaf_in_expr

`abjad.tools.leaftools.`**`get_nth_leaf_in_expr`**(*expr*, *n=0*)

New in version 1.1.2. Get *n* th leaf in *expr*:

```
abjad> staff = Staff(Measure((2, 8), notetools.make_repeated_notes(2)) * 3)
abjad> macros.diatonicize(staff)
abjad> f(staff)
\new Staff {
        {
                \time 2/8
                c'8
                d'8
        }
        {
                \time 2/8
                e'8
                f'8
        }
        {
                \time 2/8
                g'8
                a'8
        }
}
```

```
abjad> for n in range(6):
...     leaftools.get_nth_leaf_in_expr(staff, n)
...
Note("c'8")
Note("d'8")
Note("e'8")
Note("f'8")
Note("g'8")
Note("a'8")
```

Read backwards for negative values of *n*.

```
abjad> leaftools.get_nth_leaf_in_expr(staff, -1)
Note("a'8")
```

Note: Because this function returns as soon as it finds instance *n* of *klasses*, it is more efficient to call `leaftools.get_nth_leaf_in_expr(expr, 0)` than `expr.leaves[0]`. It is likewise more efficient to call `leaftools.get_nth_leaf_in_expr(expr, -1)` than `expr.leaves[-1]`.

Return leaf of none.

leaftools.get_nth_leaf_in_thread_from_leaf

abjad.tools.leaftools.**get_nth_leaf_in_thread_from_leaf**(*leaf*, *n=0*)

New in version 1.1.2. Get *n* th leaf in thread from *leaf*:

```
abjad> staff = Staff(2 * Voice(macros.scale(4)))
abjad> macros.diatonicize(staff)
abjad> f(staff)
\new Staff {
   \new Voice {
      c'8
      d'8
      e'8
      f'8
   }
   \new Voice {
      g'8
      a'8
      b'8
      c''8
   }
}

abjad> for n in range(8):
...     print n, leaftools.get_nth_leaf_in_thread_from_leaf(staff[0][0], n)
...
0 c'8
1 d'8
2 e'8
3 f'8
4 None
5 None
6 None
7 None
```

Return leaf or none.

leaftools.is_bar_line_crossing_leaf

abjad.tools.leaftools.**is_bar_line_crossing_leaf**(*leaf*)

New in version 1.1.2. True when *leaf* crosses bar line:

```
abjad> t = Staff(macros.scale(4))
abjad> t[2].duration.written *= 2
abjad> contexttools.TimeSignatureMark(2, 8, partial = Duration(1, 8))(t[2])
TimeSignatureMark(2, 8, partial = Duration(1, 8))(e'4)
abjad> f(t)
\new Staff {
        c'8
        d'8
        \partial 8
        \time 2/8
        e'4
        f'8
}
abjad> leaftools.is_bar_line_crossing_leaf(t.leaves[2])
True
```

Otherwise false:

```
abjad> leaftools.is_bar_line_crossing_leaf(t.leaves[3])
False
```

Return boolean.

leaftools.iterate_leaf_pairs_forward_in_expr

abjad.tools.leaftools.**iterate_leaf_pairs_forward_in_expr**(*expr*)

New in version 1.1.2. Iterate leaf pairs forward in *expr*:

```
abjad> score = Score([ ])
abjad> notes = macros.scale(4) + [Note(7, (1, 4))]
abjad> score.append(Staff(notes))
abjad> notes = [Note(x, (1, 4)) for x in [-12, -15, -17]]
abjad> score.append(Staff(notes))
abjad> contexttools.ClefMark('bass')(score[1])
ClefMark('bass')(Staff{3})

abjad> f(score)
\new Score <<
        \new Staff {
                c'8
                d'8
                e'8
                f'8
                g'4
        }
        \new Staff {
                \clef "bass"
                c4
                a,4
                g,4
        }
>>
```

```
abjad> for pair in leaftools.iterate_leaf_pairs_forward_in_expr(score):
...      pair
(Note("c'8"), Note('c4'))
(Note("c'8"), Note("d'8"))
(Note('c4'), Note("d'8"))
(Note("d'8"), Note("e'8"))
(Note("d'8"), Note('a,4'))
(Note('c4'), Note("e'8"))
(Note('c4'), Note('a,4'))
(Note("e'8"), Note('a,4'))
(Note("e'8"), Note("f'8"))
(Note('a,4'), Note("f'8"))
(Note("f'8"), Note("g'4"))
(Note("f'8"), Note('g,4'))
(Note('a,4'), Note("g'4"))
(Note('a,4'), Note('g,4'))
(Note("g'4"), Note('g,4'))
```

Iterate leaf pairs left-to-right and top-to-bottom.

Return generator.

leaftools.iterate_leaves_backward_in_expr

`abjad.tools.leaftools.`**`iterate_leaves_backward_in_expr`**(*expr*, *start=0*, *stop=None*)

New in version 1.1.2. Iterate leaves backward in *expr*:

```
abjad> staff = Staff(Measure((2, 8), notetools.make_repeated_notes(2)) * 3)
abjad> macros.diatonicize(staff)
abjad> f(staff)
\new Staff {
        {
                \time 2/8
                c'8
                d'8
        }
        {
                \time 2/8
                e'8
                f'8
        }
        {
                \time 2/8
                g'8
                a'8
        }
}
```

```
abjad> for leaf in leaftools.iterate_leaves_backward_in_expr(staff):
...     leaf
...
Note("a'8")
Note("g'8")
Note("f'8")
Note("e'8")
Note("d'8")
Note("c'8")
```

Use the optional *start* and *stop* keyword parameters to control the indices of iteration.

```
abjad> for leaf in leaftools.iterate_leaves_backward_in_expr(staff, start = 3):
...     leaf
...
Note("e'8")
Note("d'8")
Note("c'8")

abjad> for leaf in leaftools.iterate_leaves_backward_in_expr(staff, start = 0, stop = 3):
...     leaf
...
Note("a'8")
Note("g'8")
Note("f'8")

abjad> for leaf in leaftools.iterate_leaves_backward_in_expr(staff, start = 2, stop = 4):
...     leaf
...
Note("f'8")
Note("e'8")
```

Ignore threads.

Return generator.

leaftools.iterate_leaves_forward_in_expr

abjad.tools.leaftools.**iterate_leaves_forward_in_expr**(*expr*, *start=0*, *stop=None*)

New in version 1.1.2. Iterate leaves forward in *expr*:

```
abjad> staff = Staff(Measure((2, 8), notetools.make_repeated_notes(2)) * 3)
abjad> macros.diatonicize(staff)
abjad> f(staff)
\new Staff {
        {
                \time 2/8
                c'8
                d'8
        }
        {
                \time 2/8
                e'8
                f'8
        }
        {
                \time 2/8
                g'8
                a'8
        }
}

abjad> for leaf in leaftools.iterate_leaves_forward_in_expr(staff):
...     leaf
...
Note("c'8")
Note("d'8")
Note("e'8")
```

```
Note("f'8")
Note("g'8")
Note("a'8")
```

Use the optional *start* and *stop* keyword parameters to control the start and stop indices of iteration.

```
abjad> for leaf in leaftools.iterate_leaves_forward_in_expr(staff, start = 3):
...     leaf
...
Note("f'8")
Note("g'8")
Note("a'8")

abjad> for leaf in leaftools.iterate_leaves_forward_in_expr(staff, start = 0, stop = 3):
...     leaf
...
Note("c'8")
Note("d'8")
Note("e'8")

abjad> for leaf in leaftools.iterate_leaves_forward_in_expr(staff, start = 2, stop = 4):
...     leaf
...
Note("e'8")
Note("f'8")
```

Ignore threads.

Return generator.

leaftools.iterate_notes_and_chords_backward_in_expr

abjad.tools.leaftools.**iterate_notes_and_chords_backward_in_expr**(*expr*, *start=0*, *stop=None*)

New in version 1.1.2. Iterate notes and chords backward in *expr*:

```
abjad> staff = Staff("<e' g' c''>8 a'8 r8 <d' f' b'>8 r2")

abjad> f(staff)
\new Staff {
   <e' g' c''>8
   a'8
   r8
   <d' f' b'>8
   r2
}

abjad> for leaf in leaftools.iterate_notes_and_chords_backward_in_expr(staff):
...   leaf
Chord("<d' f' b'>8")
Note("a'8")
Chord("<e' g' c''>8")
```

Ignore threads.

Return generator. Changed in version 1.1.2: renamed `pitchtools.iterate_notes_and_chords_backward_in_expr( )` to `leaftools.iterate_notes_and_chords_backward_in_expr( )`.

leaftools.iterate_notes_and_chords_forward_in_expr

abjad.tools.leaftools.**iterate_notes_and_chords_forward_in_expr**(*expr*, *start=0*, *stop=None*)

New in version 1.1.2. Iterate notes and chords forward in *expr*:

```
abjad> staff = Staff("<e' g' c''>8 a'8 r8 <d' f' b'>8 r2")

abjad> f(staff)
\new Staff {
   <e' g' c''>8
   a'8
   r8
   <d' f' b'>8
   r2
}

abjad> for leaf in leaftools.iterate_notes_and_chords_forward_in_expr(staff):
...    leaf
Chord("<e' g' c''>8")
Note("a'8")
Chord("<d' f' b'>8")
```

Ignore threads.

Return generator. Changed in version 1.1.2: renamed `pitchtools.iterate_notes_and_chords_forward_in_expr` `)` to `leaftools.iterate_notes_and_chords_forward_in_expr( )`.

leaftools.label_leaves_in_expr_with_inversion_equivalent_chromatic_interval_classes

abjad.tools.leaftools.**label_leaves_in_expr_with_inversion_equivalent_chromatic_interval_cla**

New in version 1.1.2. Label leaves in *expr* with inversion-equivalent chromatic interval classes:

```
abjad> staff = Staff(notetools.make_notes([0, 25, 11, -4, -14, -13, 9, 10, 6, 5], [Duration(1, 8
abjad> leaftools.label_leaves_in_expr_with_inversion_equivalent_chromatic_interval_classes(staff
abjad> f(staff)
\new Staff {
   c'8 ^ \markup { 1 }
   cs'''8 ^ \markup { 2 }
   b'8 ^ \markup { 2 }
   af8 ^ \markup { 2 }
   bf,8 ^ \markup { 1 }
   b,8 ^ \markup { 2 }
   a'8 ^ \markup { 1 }
   bf'8 ^ \markup { 4 }
   fs'8 ^ \markup { 1 }
   f'8
}
```

Return none.

leaftools.label_leaves_in_expr_with_leaf_depth

abjad.tools.leaftools.**label_leaves_in_expr_with_leaf_depth**(*expr*, *markup_direction='down'*)

New in version 1.1.1. Label leaves in *expr* with leaf depth:

```
abjad> staff = Staff(macros.scale(5))
abjad> tuplettools.FixedDurationTuplet((2, 8), staff[-3:])
FixedDurationTuplet(1/4, [e'8, f'8, g'8])
abjad> leaftools.label_leaves_in_expr_with_leaf_depth(staff)
abjad> f(staff)
\new Staff {
        c'8 _ \markup { \small 1 }
        d'8 _ \markup { \small 1 }
        \times 2/3 {
                e'8 _ \markup { \small 2 }
                f'8 _ \markup { \small 2 }
                g'8 _ \markup { \small 2 }
        }
}
```

Changed in version 1.1.2: renamed `label.leaf_depth( )` to `leaftools.label_leaves_in_expr_with_leaf_d` `)`. Return none.

leaftools.label_leaves_in_expr_with_leaf_durations

`abjad.tools.leaftools.`**`label_leaves_in_expr_with_leaf_durations`**`(`*expr*, *markup_direction='down'*`)`

New in version 1.1.1. Label leaves in *expr* with leaf durations:

```
abjad> tuplet = tuplettools.FixedDurationTuplet((1, 4), macros.scale(3))
abjad> leaftools.label_leaves_in_expr_with_leaf_durations(tuplet)
abjad> f(tuplet)
\times 2/3 {
        c'8 _ \markup { \column { \small 1/8 \small 1/12 } }
        d'8 _ \markup { \column { \small 1/8 \small 1/12 } }
        e'8 _ \markup { \column { \small 1/8 \small 1/12 } }
}
```

Label both written duration and prolated duration.

Return none.

leaftools.label_leaves_in_expr_with_leaf_indices

`abjad.tools.leaftools.`**`label_leaves_in_expr_with_leaf_indices`**`(`*expr*, *markup_direction='down'*`)`

New in version 1.1.2. Label leaves in *expr* with leaf indices:

```
abjad> staff = Staff(macros.scale(4))
abjad> leaftools.label_leaves_in_expr_with_leaf_indices(staff)
abjad> f(staff)
\new Staff {
        c'8 _ \markup { \small 0 }
        d'8 _ \markup { \small 1 }
        e'8 _ \markup { \small 2 }
        f'8 _ \markup { \small 3 }
}
```

Return none.

leaftools.label_leaves_in_expr_with_leaf_numbers

`abjad.tools.leaftools.`**`label_leaves_in_expr_with_leaf_numbers`**`(`*expr*, *markup_direction='down'*`)`

New in version 1.1.1. Label leaves in *expr* with leaf numbers:

```
abjad> staff = Staff(macros.scale(4))
abjad> leaftools.label_leaves_in_expr_with_leaf_numbers(staff)
abjad> f(staff)
\new Staff {
        c'8 _ \markup { \small 1 }
        d'8 _ \markup { \small 2 }
        e'8 _ \markup { \small 3 }
        f'8 _ \markup { \small 4 }
}
```

Number leaves starting from `1`. Changed in version 1.1.2: renamed `label.leaf_numbers( )` to `leaftools.label_leaves_in_expr_with_leaf_numbers( )`. Return none.

leaftools.label_leaves_in_expr_with_melodic_chromatic_interval_classes

`abjad.tools.leaftools.`**`label_leaves_in_expr_with_melodic_chromatic_interval_classes`**`(`*expr*, *markup_di*

New in version 1.1.2. Label leaves in *expr* with melodic chromatic interval classes:

```
abjad> staff = Staff(notetools.make_notes([0, 25, 11, -4, -14, -13, 9, 10, 6, 5], [Duration(1, 8
abjad> leaftools.label_leaves_in_expr_with_melodic_chromatic_interval_classes(staff)
abjad> f(staff)
\new Staff {
        c'8 ^ \markup { +1 }
        cs'''8 ^ \markup { -2 }
        b'8 ^ \markup { -2 }
        af8 ^ \markup { -10 }
        bf,8 ^ \markup { +1 }
        b,8 ^ \markup { +10 }
        a'8 ^ \markup { +1 }
        bf'8 ^ \markup { -4 }
        fs'8 ^ \markup { -1 }
        f'8
}
```

Return none.

leaftools.label_leaves_in_expr_with_melodic_chromatic_intervals

`abjad.tools.leaftools.`**`label_leaves_in_expr_with_melodic_chromatic_intervals`**`(`*expr*, *markup_direction='u*

New in version 1.1.2. Label leaves in *expr* with melodic chromatic intervals:

```
abjad> staff = Staff(notetools.make_notes([0, 25, 11, -4, -14, -13, 9, 10, 6, 5], [Duration(1, 8
abjad> leaftools.label_leaves_in_expr_with_melodic_chromatic_intervals(staff)
abjad> f(staff)
\new Staff {
        c'8 ^ \markup { +25 }
        cs'''8 ^ \markup { -14 }
        b'8 ^ \markup { -15 }
        af8 ^ \markup { -10 }
```

```
        bf,8 ^ \markup { +1 }
        b,8 ^ \markup { +22 }
        a'8 ^ \markup { +1 }
        bf'8 ^ \markup { -4 }
        fs'8 ^ \markup { -1 }
        f'8
}
```

Return none.

leaftools.label_leaves_in_expr_with_melodic_counterpoint_interval_classes

abjad.tools.leaftools.**label_leaves_in_expr_with_melodic_counterpoint_interval_classes**(*expr*, *marku*

New in version 1.1.2. Label leaves in *expr* with melodic counterpoint interval classes:

```
abjad> staff = Staff(notetools.make_notes([0, 25, 11, -4, -14, -13, 9, 10, 6, 5], [Duration(1, 8
abjad> leaftools.label_leaves_in_expr_with_melodic_counterpoint_interval_classes(staff)
abjad> f(staff)
\new Staff {
        c'8 ^ \markup { +8 }
        cs'''8 ^ \markup { -2 }
        b'8 ^ \markup { -2 }
        af8 ^ \markup { -7 }
        bf,8 ^ \markup { +1 }
        b,8 ^ \markup { +7 }
        a'8 ^ \markup { +2 }
        bf'8 ^ \markup { -4 }
        fs'8 ^ \markup { +1 }
        f'8
}
```

Return none.

leaftools.label_leaves_in_expr_with_melodic_counterpoint_intervals

abjad.tools.leaftools.**label_leaves_in_expr_with_melodic_counterpoint_intervals**(*expr*, *markup_directio*

New in version 1.1.2. Label leaves in *expr* with melodic counterpoint intervals:

```
abjad> staff = Staff(notetools.make_notes([0, 25, 11, -4, -14, -13, 9, 10, 6, 5], [Duration(1, 8
abjad> leaftools.label_leaves_in_expr_with_melodic_counterpoint_intervals(staff)
abjad> f(staff)
\new Staff {
        c'8 ^ \markup { +15 }
        cs'''8 ^ \markup { -9 }
        b'8 ^ \markup { -9 }
        af8 ^ \markup { -7 }
        bf,8 ^ \markup { 1 }
        b,8 ^ \markup { +14 }
        a'8 ^ \markup { +2 }
        bf'8 ^ \markup { -4 }
        fs'8 ^ \markup { 1 }
        f'8
}
```

Return none.

leaftools.label_leaves_in_expr_with_melodic_diatonic_interval_classes

`abjad.tools.leaftools.`**`label_leaves_in_expr_with_melodic_diatonic_interval_classes`**(*expr*, *markup_dire*

New in version 1.1.2. Label leaves in *expr* with melodic diatonic interval classes:

```
abjad> staff = Staff(notetools.make_notes([0, 25, 11, -4, -14, -13, 9, 10, 6, 5], [Duration(1, 8
abjad> leaftools.label_leaves_in_expr_with_melodic_diatonic_interval_classes(staff)
abjad> f(staff)
\new Staff {
        c'8 ^ \markup { +aug8 }
        cs'''8 ^ \markup { -M2 }
        b'8 ^ \markup { -aug2 }
        af8 ^ \markup { -m7 }
        bf,8 ^ \markup { aug1 }
        b,8 ^ \markup { +m7 }
        a'8 ^ \markup { +m2 }
        bf'8 ^ \markup { -dim4 }
        fs'8 ^ \markup { aug1 }
        f'8
}
```

Return none.

leaftools.label_leaves_in_expr_with_melodic_diatonic_intervals

`abjad.tools.leaftools.`**`label_leaves_in_expr_with_melodic_diatonic_intervals`**(*expr*, *markup_direction='up*

New in version 1.1.2. Label leaves in *expr* with melodic diatonic intervals:

```
abjad> staff = Staff(notetools.make_notes([0, 25, 11, -4, -14, -13, 9, 10, 6, 5], [Duration(1, 8
abjad> leaftools.label_leaves_in_expr_with_melodic_diatonic_intervals(staff)
abjad> f(staff)
\new Staff {
        c'8 ^ \markup { +aug15 }
        cs'''8 ^ \markup { -M9 }
        b'8 ^ \markup { -aug9 }
        af8 ^ \markup { -m7 }
        bf,8 ^ \markup { +aug1 }
        b,8 ^ \markup { +m14 }
        a'8 ^ \markup { +m2 }
        bf'8 ^ \markup { -dim4 }
        fs'8 ^ \markup { -aug1 }
        f'8
}
```

Return none.

leaftools.label_leaves_in_expr_with_pitch_class_numbers

`abjad.tools.leaftools.`**`label_leaves_in_expr_with_pitch_class_numbers`**(*expr*, *number=True*, *color=False*, *markup_direction='down'*)

New in version 1.1.1. Label leaves in *expr* with pitch-class numbers:

```
abjad> t = Staff(macros.scale(4))
abjad> leaftools.label_leaves_in_expr_with_pitch_class_numbers(t)
abjad> print t.format
\new Staff {
   c'8 _ \markup { \small 0 }
   d'8 _ \markup { \small 2 }
   e'8 _ \markup { \small 4 }
   f'8 _ \markup { \small 5 }
}
```

When `color = True` call `color_note_head_by_numbered_chromatic_pitch_class_color_map()`.

```
abjad> t = Staff(macros.scale(4))
abjad> leaftools.label_leaves_in_expr_with_pitch_class_numbers(t, color = True, number = False)
abjad> print t.format
\new Staff {
   \once \override NoteHead #'color = #(x11-color 'red)
   c'8
   \once \override NoteHead #'color = #(x11-color 'orange)
   d'8
   \once \override NoteHead #'color = #(x11-color 'ForestGreen)
   e'8
   \once \override NoteHead #'color = #(x11-color 'MediumOrchid)
   f'8
}
```

You can set *number* and *color* at the same time. Changed in version 1.1.2: renamed `label.leaf_pcs( )` to `leaftools.label_leaves_in_expr_with_pitch_class_numbers( )`. Return none.

leaftools.label_leaves_in_expr_with_pitch_numbers

`abjad.tools.leaftools.`**`label_leaves_in_expr_with_pitch_numbers`**(*expr*, *markup_direction='down'*)

New in version 1.1.1. Label leaves in *expr* with pitch numbers:

```
abjad> staff = Staff(leaftools.make_leaves([None, 12, [13, 14, 15], None], [(1, 4)]))
abjad> leaftools.label_leaves_in_expr_with_pitch_numbers(staff)
abjad> f(staff)
\new Staff {
        r4
        c''4 _ \markup { \small 12 }
        <cs'' d'' ef''>4 _ \markup { \column { \small 15 \small 14 \small 13 } }
        r4
}
```

Return none. Changed in version 1.1.2: renamed `label.leaf_pitch_numbers( )` to `leaftools.label_leaves_in_expr_with_pitch_numbers( )`.

leaftools.label_leaves_in_expr_with_prolated_leaf_duration

`abjad.tools.leaftools.`**`label_leaves_in_expr_with_prolated_leaf_duration`**(*expr*, *markup_direction='down'*)

New in version 1.1.1. Label leaves in *expr* with prolated leaf duration:

```
abjad> tuplet = tuplettools.FixedDurationTuplet((1, 4), macros.scale(3))
abjad> leaftools.label_leaves_in_expr_with_prolated_leaf_duration(tuplet)
abjad> f(tuplet)
```

```
\times 2/3 {
        c'8 _ \markup { \small 1/12 }
        d'8 _ \markup { \small 1/12 }
        e'8 _ \markup { \small 1/12 }
}
```

Return none.

leaftools.label_leaves_in_expr_with_tuplet_depth

abjad.tools.leaftools.**label_leaves_in_expr_with_tuplet_depth**(*expr*, *markup_direction='down'*)

New in version 1.1.1. Label leaves in *expr* with tuplet depth:

```
abjad> staff = Staff(macros.scale(5))
abjad> tuplettools.FixedDurationTuplet((2, 8), staff[-3:])
FixedDurationTuplet(1/4, [e'8, f'8, g'8])
abjad> leaftools.label_leaves_in_expr_with_tuplet_depth(staff)
abjad> f(staff)
\new Staff {
        c'8 _ \markup { \small 0 }
        d'8 _ \markup { \small 0 }
        \times 2/3 {
                e'8 _ \markup { \small 1 }
                f'8 _ \markup { \small 1 }
                g'8 _ \markup { \small 1 }
        }
}
```

Return none. Changed in version 1.1.2: renamed `label.leaf_depth_tuplet( )` to `leaftools.label_leaves_in_expr_with_tuplet_depth( )`.

leaftools.label_leaves_in_expr_with_written_leaf_duration

abjad.tools.leaftools.**label_leaves_in_expr_with_written_leaf_duration**(*expr*, *markup_direction='down'*)

New in version 1.1.1. Label leaves in *expr* with writen leaf duration:

```
abjad> tuplet = tuplettools.FixedDurationTuplet((1, 4), macros.scale(3))
abjad> leaftools.label_leaves_in_expr_with_leaf_durations(tuplet)
abjad> f(tuplet)
\times 2/3 {
   c'8 _ \markup { \column { \small 1/8 \small 1/12 } }
   d'8 _ \markup { \column { \small 1/8 \small 1/12 } }
   e'8 _ \markup { \column { \small 1/8 \small 1/12 } }
}
```

Return none.

leaftools.leaf_to_augmented_tuplet_with_n_notes_of_equal_written_duration

abjad.tools.leaftools.**leaf_to_augmented_tuplet_with_n_notes_of_equal_written_duration**(*leaf*, *n*)

New in version 1.1.2. Change *leaf* to augmented tuplet with *n* notes of equal written duration:

```
abjad> for n in range(1, 11):
...     note = Note(0, (3, 16))
...     tuplet = leaftools.leaf_to_augmented_tuplet_with_n_notes_of_equal_written_duration(note,
...     print tuplet
...
{@ 1:1 c'8. @}
{@ 1:1 c'16., c'16. @}
{@ 1:1 c'16, c'16, c'16 @}
{@ 1:1 c'32., c'32., c'32., c'32. @}
{@ 5:8 c'64., c'64., c'64., c'64., c'64. @}
{@ 1:1 c'32, c'32, c'32, c'32, c'32, c'32 @}
{@ 7:8 c'64., c'64., c'64., c'64., c'64., c'64., c'64. @}
{@ 1:1 c'64., c'64., c'64., c'64., c'64., c'64., c'64., c'64. @}
{@ 3:4 c'64, c'64, c'64, c'64, c'64, c'64, c'64, c'64, c'64 @}
{@ 5:8 c'128., c'128., c'128., c'128., c'128., c'128., c'128., c'128., c'128., c'128. @}
```

Return augmented fixed-duration tuplet.

leaftools.leaf_to_augmented_tuplet_with_proportions

abjad.tools.leaftools.**leaf_to_augmented_tuplet_with_proportions**(*leaf*, *proportions*)

New in version 1.1.2. Change *leaf* to augmented tuplet with *proportions*:

```
abjad> note = Note(0, (3, 16))
abjad> print leaftools.leaf_to_augmented_tuplet_with_proportions(note, [1])
{@ 1:1 c'8. @}
abjad> print leaftools.leaf_to_augmented_tuplet_with_proportions(note, [1, 2])
{@ 1:1 c'16, c'8 @}
abjad> print leaftools.leaf_to_augmented_tuplet_with_proportions(note, [1, 2, 2])
{@ 5:8 c'64., c'32., c'32. @}
abjad> print leaftools.leaf_to_augmented_tuplet_with_proportions(note, [1, 2, 2, 3])
{@ 2:3 c'64, c'32, c'32, c'32. @}
abjad> print leaftools.leaf_to_augmented_tuplet_with_proportions(note, [1, 2, 2, 3, 3])
{@ 11:12 c'64, c'32, c'32, c'32., c'32. @}
abjad> print leaftools.leaf_to_augmented_tuplet_with_proportions(note, [1, 2, 2, 3, 3, 4])
{@ 5:8 c'128, c'64, c'64, c'64., c'64., c'32 @}
```

Return augmented fixed-duration tuplet.

leaftools.leaf_to_diminished_tuplet_with_n_notes_of_equal_written_duration

abjad.tools.leaftools.**leaf_to_diminished_tuplet_with_n_notes_of_equal_written_duration**(*leaf*, *n*)

New in version 1.1.2. Change *leaf* to diminished tuplet with *n* notes of equal written duration:

```
abjad> for n in range(1, 11):
...     note = Note(0, (3, 16))
...     tuplet = leaftools.leaf_to_diminished_tuplet_with_n_notes_of_equal_written_duration(note
...     print tuplet
...
{@ 1:1 c'8. @}
{@ 1:1 c'16., c'16. @}
{@ 1:1 c'16, c'16, c'16 @}
{@ 1:1 c'32., c'32., c'32., c'32. @}
{@ 5:4 c'32., c'32., c'32., c'32., c'32. @}
{@ 1:1 c'32, c'32, c'32, c'32, c'32, c'32 @}
```

```
{@ 7:4 c'32., c'32., c'32., c'32., c'32., c'32., c'32. @}
{@ 1:1 c'64., c'64., c'64., c'64., c'64., c'64., c'64., c'64. @}
{@ 3:2 c'32, c'32, c'32, c'32, c'32, c'32, c'32, c'32, c'32 @}
{@ 5:4 c'64., c'64., c'64., c'64., c'64., c'64., c'64., c'64., c'64., c'64. @}
```

Return diminished fixed-duration tuplet.

leaftools.leaf_to_diminished_tuplet_with_proportions

abjad.tools.leaftools.**leaf_to_diminished_tuplet_with_proportions**(*leaf*, *proportions*)

New in version 1.1.2. Change *leaf* to diminished tuplet with *proportions*:

```
abjad> note = Note(0, (3, 16))
abjad> print leaftools.leaf_to_diminished_tuplet_with_proportions(note, [1])
{@ 1:1 c'8. @}
abjad> print leaftools.leaf_to_diminished_tuplet_with_proportions(note, [1, 2])
{@ 1:1 c'16, c'8 @}
abjad> print leaftools.leaf_to_diminished_tuplet_with_proportions(note, [1, 2, 2])
{@ 5:4 c'32., c'16., c'16. @}
abjad> print leaftools.leaf_to_diminished_tuplet_with_proportions(note, [1, 2, 2, 3])
{@ 4:3 c'32, c'16, c'16, c'16. @}
abjad> print leaftools.leaf_to_diminished_tuplet_with_proportions(note, [1, 2, 2, 3, 3])
{@ 11:6 c'32, c'16, c'16, c'16., c'16. @}
abjad> print leaftools.leaf_to_diminished_tuplet_with_proportions(note, [1, 2, 2, 3, 3, 4])
{@ 5:4 c'64, c'32, c'32, c'32., c'32., c'16 @}
```

Return diminshed fixed-duration tuplet.

leaftools.list_prolated_durations_of_leaves_in_expr

abjad.tools.leaftools.**list_prolated_durations_of_leaves_in_expr**(*expr*)

New in version 1.1.2. List prolated durations of leaves in *expr*:

```
abjad> staff = Staff(tuplettools.FixedDurationTuplet((2, 8), macros.scale(3)) * 2)
abjad> leaftools.list_prolated_durations_of_leaves_in_expr(staff)
[Duration(1, 12), Duration(1, 12), Duration(1, 12), Duration(1, 12), Duration(1, 12), Duration(1
```

Return list of fractions.

leaftools.list_written_durations_of_leaves_in_expr

abjad.tools.leaftools.**list_written_durations_of_leaves_in_expr**(*expr*)

New in version 1.1.2. List the written durations of leaves in *expr*:

```
abjad> staff = Staff(tuplettools.FixedDurationTuplet((2, 8), macros.scale(3)) * 2)
abjad> leaftools.list_written_durations_of_leaves_in_expr(staff)
[Duration(1, 8), Duration(1, 8), Duration(1, 8), Duration(1, 8), Duration(1, 8), Duration(1, 8)]
```

Return list of fractions.

leaftools.make_leaves

abjad.tools.leaftools.**make_leaves**(*pitches*, *durations*, *direction='big-endian'*, *tied_rests=False*)

New in version 1.1.1. Construct a list of notes, rests or chords.

Set *pitches* is a single pitch, or a list of pitches, or a tuple of pitches.

Integer pitches create notes.

```
abjad> leaftools.make_leaves([2, 4, 19], [(1, 4)])
[Note("d'4"), Note("e'4"), Note("g''4")]
```

Tuple pitches create chords.

```
abjad> leaftools.make_leaves([(0, 1, 2), (3, 4, 5), (6, 7, 8)], [(1, 4)])
[Chord("<c' cs' d'>4"), Chord("<ef' e' f'>4"), Chord("<fs' g' af'>4")]
```

Set *pitches* to a list of none to create rests.

```
abjad> leaftools.make_leaves([None, None, None, None], [(1, 8)])
[Rest('r8'), Rest('r8'), Rest('r8'), Rest('r8')]
```

You can mix and match pitch values.

```
abjad> leaftools.make_leaves([12, (1, 2, 3), None, 12], [(1, 4)])
[Note("c''4"), Chord("<cs' d' ef'>4"), Rest('r4'), Note("c''4")]
```

If the length of *pitches* is less than the length of *durations*, the function reads *durations* cyclically.

```
abjad> leaftools.make_leaves([13], [(1, 8), (1, 8), (1, 4), (1, 4)])
[Note("cs''8"), Note("cs''8"), Note("cs''4"), Note("cs''4")]
```

Set *durations* to a single duration, a list of duration, or a tuple of durations.

If the length of *durations* is less than the length of *pitches*, the function reads *pitches* cyclically.

```
abjad> leaftools.make_leaves([13, 14, 15, 16], [(1, 8)])
[Note("cs''8"), Note("d''8"), Note("ef''8"), Note("e''8")]
```

Duration values not of the form `m / 2 ** n` return leaves nested inside a fixed-multiplier tuplet.

```
abjad> leaftools.make_leaves([14], [(1, 12), (1, 12), (1, 12)])
[Tuplet(2/3, [d''8, d''8, d''8])]
```

Set *direction* to `'little-endian'` to return tied leaf durations from least to greatest.

```
abjad> staff = Staff(leaftools.make_leaves([15], [(13, 16)], direction = 'little-endian'))
abjad> f(staff)
\new Staff {
	ef''16 ~
	ef''2.
}
```

Set *tied_rests* to true to return tied rests for durations like `5/16` and `9/16`.

```
abjad> staff = Staff(leaftools.make_leaves([None], [(5, 16)], tied_rests = True))
abjad> f(staff)
\new Staff {
	r4 ~
	r16
}
```

Return list of leaves. Changed in version 1.1.2: renamed `construct.leaves( )` to `leaftools.make_leaves( )`.

leaftools.make_leaves_from_note_value_signal

`abjad.tools.leaftools.`**`make_leaves_from_note_value_signal`**`(`*note_value_signal*, *denominator_of_signal*, *tied_rests=False*`)`

New in version 1.1.2. Make leaves from *note_value_signal* and *denominator_of_signal*:

```
abjad> leaves = leaftools.make_leaves_from_note_value_signal([3, -3, 5, -5], 8)
abjad> staff = Staff(leaves)

abjad> f(staff)
\new Staff {
   c'4.
   r4.
   c'2 ~
   c'8
   r2
   r8
}
```

Interpret positive elements in *note_value_signal* as notes.

Interpret negative elements in *note_value_signal* as rests.

Set the pitch of all notes to middle C.

Return list of notes and / or rests.

leaftools.remove_initial_rests_from_sequence

`abjad.tools.leaftools.`**`remove_initial_rests_from_sequence`**`(`*sequence*`)`

New in version 1.1.2. Remove initial rests from *sequence*:

```
abjad> staff = Staff("r8 r8 c'8 d'8 r4 r4")

abjad> f(staff)
\new Staff {
   r8
   r8
   c'8
   d'8
   r4
   r4
}

abjad> leaftools.remove_initial_rests_from_sequence(staff)
[Note("c'8"), Note("d'8"), Rest('r4'), Rest('r4')]

abjad> f(staff)
\new Staff {
   r8
   r8
   c'8
   d'8
   r4
```

```
      r4
   }
```

Return list.

leaftools.remove_leaf_and_shrink_durated_parent_containers

abjad.tools.leaftools.**remove_leaf_and_shrink_durated_parent_containers**(*leaf*)

New in version 1.1.1. Remove *leaf* and shrink durated parent containers:

```
abjad> measure = Measure((4, 8), tuplettools.FixedDurationTuplet((2, 8), notetools.make_repeatec
abjad> macros.diatonicize(measure)
abjad> spannertools.BeamSpanner(measure.leaves)
BeamSpanner(c'8, d'8, e'8, f'8, g'8, a'8)
abjad> f(measure)
{
   \time 4/8
   \times 2/3 {
      c'8 [
      d'8
      e'8
   }
   \times 2/3 {
      f'8
      g'8
      a'8 ]
   }
}
```

```
abjad> leaftools.remove_leaf_and_shrink_durated_parent_containers(measure.leaves[0])
```

```
abjad> f(measure)
{
   \time 5/12
   \scaleDurations #'(2 . 3) {
      {
         d'8 [
         e'8
      }
      {
         f'8
         g'8
         a'8 ]
      }
   }
}
```

Return none.

leaftools.remove_outer_rests_from_sequence

abjad.tools.leaftools.**remove_outer_rests_from_sequence**(*sequence*)

New in version 1.1.2. Remove outer rests from *sequence*:

```
abjad> staff = Staff("r8 r8 c'8 d'8 r4 r4")
```

```
abjad> f(staff)
\new Staff {
   r8
   r8
   c'8
   d'8
   r4
   r4
}

abjad> leaftools.remove_outer_rests_from_sequence(staff)
[Note("c'8"), Note("d'8")]

abjad> f(staff)
\new Staff {
   r8
   r8
   c'8
   d'8
   r4
   r4
}
```

Return list.

leaftools.remove_terminal_rests_from_sequence

abjad.tools.leaftools.**remove_terminal_rests_from_sequence**(*sequence*)

New in version 1.1.2. Remove terminal rests from *sequence*:

```
abjad> staff = Staff("r8 r8 c'8 d'8 r4 r4")

abjad> f(staff)
\new Staff {
   r8
   r8
   c'8
   d'8
   r4
   r4
}

abjad> leaftools.remove_terminal_rests_from_sequence(staff)
[Rest('r8'), Rest('r8'), Note("c'8"), Note("d'8")]

abjad> f(staff)
\new Staff {
   r8
   r8
   c'8
   d'8
   r4
   r4
}
```

Return list.

leaftools.repeat_leaf_and_extend_spanners

`abjad.tools.leaftools.`**`repeat_leaf_and_extend_spanners`**`(`*leaf*, *total=1*`)`

New in version 1.1.1. Repeat *leaf* and extend spanners:

```
abjad> staff = Staff(macros.scale(4))
abjad> spannertools.BeamSpanner(staff.leaves)
BeamSpanner(c'8, d'8, e'8, f'8)
abjad> f(staff)
\new Staff {
   c'8 [
   d'8
   e'8
   f'8 ]
}

abjad> leaftools.repeat_leaf_and_extend_spanners(staff[0], total = 3)

abjad> f(staff)
\new Staff {
   c'8 [
   c'8
   c'8
   d'8
   e'8
   f'8 ]
}
```

Preserve *leaf* written duration.

Preserve parentage and spanners.

Return none. Changed in version 1.1.2: renamed `leaftools.clone_and_splice_leaf( )` to `leaftools.repeat_leaf_and_extend_spanners( )`.

leaftools.repeat_leaves_in_expr_and_extend_spanners

`abjad.tools.leaftools.`**`repeat_leaves_in_expr_and_extend_spanners`**`(`*expr*, *total=1*`)`

New in version 1.1.1. Repeat leaves in *expr* and extend spanners:

```
abjad> staff = Staff(macros.scale(4))
abjad> spannertools.BeamSpanner(staff.leaves)
BeamSpanner(c'8, d'8, e'8, f'8)
abjad> f(staff)
\new Staff {
   c'8 [
   d'8
   e'8
   f'8 ]
}

abjad> result = leaftools.repeat_leaves_in_expr_and_extend_spanners(staff[2:], total = 3)

abjad> f(staff)
\new Staff {
   c'8 [
   d'8
   e'8
```

```
    e'8
    e'8
    f'8
    f'8
    f'8 ]
}
```

Preserve leaf written durations.

Preserve parentage and spanners.

Return none. Changed in version 1.1.2: renamed `leaftools.multiply( )` to `leaftools.repeat_leaves_in_expr_and_extend_spanners( )`.

leaftools.scale_preprolated_leaf_duration

abjad.tools.leaftools.**scale_preprolated_leaf_duration**(*leaf*, *multiplier*)

New in version 1.1.1. Scale preprolated *leaf* leaf duration by dotted *multiplier*:

```
abjad> staff = Staff(macros.scale(4))
abjad> spannertools.BeamSpanner(staff.leaves)
BeamSpanner(c'8, d'8, e'8, f'8)
abjad> leaftools.scale_preprolated_leaf_duration(staff[1], Duration(3, 2))
[Note("d'8.")]
abjad> f(staff)
\new Staff {
    c'8 [
    d'8.
    e'8
    f'8 ]
}
```

Scale preprolated *leaf* duration by tied *multiplier*:

```
abjad> staff = Staff(macros.scale(4))
abjad> spannertools.BeamSpanner(staff.leaves)
BeamSpanner(c'8, d'8, e'8, f'8)
abjad> leaftools.scale_preprolated_leaf_duration(staff[1], Duration(5, 4))
[Note("d'8"), Note("d'32")]
abjad> f(staff)
\new Staff {
    c'8 [
    d'8 ~
    d'32
    e'8
    f'8 ]
}
```

Scale preprolated *leaf* duration by nonbinary *multiplier*:

```
abjad> staff = Staff(macros.scale(4))
abjad> spannertools.BeamSpanner(staff.leaves)
BeamSpanner(c'8, d'8, e'8, f'8)
abjad> leaftools.scale_preprolated_leaf_duration(staff[1], Duration(2, 3))
[Note("d'8")]
abjad> f(staff)
\new Staff {
    c'8 [
    \times 2/3 {
```

```
        d'8
    }
    e'8
    f'8 ]
}
```

Scale preprolated *leaf* duration by tied nonbinary *multiplier*:

```
abjad> staff = Staff(macros.scale(4))
abjad> spannertools.BeamSpanner(staff.leaves)
BeamSpanner(c'8, d'8, e'8, f'8)
abjad> leaftools.scale_preprolated_leaf_duration(staff[1], Duration(5, 6))
[Note("d'8"), Note("d'32")]
abjad> f(staff)
\new Staff {
    c'8 [
    \times 2/3 {
        d'8 ~
        d'32
    }
    e'8
    f'8 ]
}
```

Return *leaf*. Changed in version 1.1.2: renamed from `leaftools.duration_scale( )`. `leaftools.scale_preprolated_leaf_duration( )`.

leaftools.set_preprolated_leaf_duration

`abjad.tools.leaftools.`**`set_preprolated_leaf_duration`**(*leaf*, *new_preprolated_duration*)

New in version 1.1.1. Set preprolated *leaf* duration:

```
abjad> staff = Staff(macros.scale(4))
abjad> spannertools.BeamSpanner(staff.leaves)
BeamSpanner(c'8, d'8, e'8, f'8)
abjad> leaftools.set_preprolated_leaf_duration(staff[1], Duration(3, 16))
[Note("d'8.")]
abjad> f(staff)
\new Staff {
    c'8 [
    d'8.
    e'8
    f'8 ]
}
```

Set tied preprolated *leaf* duration:

```
abjad> staff = Staff(macros.scale(4))
abjad> spannertools.BeamSpanner(staff.leaves)
BeamSpanner(c'8, d'8, e'8, f'8)
abjad> leaftools.set_preprolated_leaf_duration(staff[1], Duration(5, 32))
[Note("d'8"), Note("d'32")]
abjad> f(staff)
\new Staff {
    c'8 [
    d'8 ~
    d'32
    e'8
```

```
      f'8 ]
}
```

Set nonbinary preprolated *leaf* duration:

```
abjad> staff = Staff(macros.scale(4))
abjad> spannertools.BeamSpanner(staff.leaves)
BeamSpanner(c'8, d'8, e'8, f'8)
abjad> leaftools.set_preprolated_leaf_duration(staff[1], Duration(1, 12))
[Note("d'8")]
abjad> f(staff)
\new Staff {
   c'8 [
   \times 2/3 {
      d'8
   }
   e'8
   f'8 ]
}
```

Set tied nonbinary preprolated *leaf* duration:

```
abjad> staff = Staff(macros.scale(4))
abjad> spannertools.BeamSpanner(staff.leaves)
BeamSpanner(c'8, d'8, e'8, f'8)
abjad> leaftools.set_preprolated_leaf_duration(staff[1], Duration(5, 48))
[Note("d'8"), Note("d'32")]
abjad> f(staff)
\new Staff {
   c'8 [
   \times 2/3 {
      d'8 ~
      d'32
   }
   e'8
   f'8 ]
}
```

Set preprolated *leaf* duration with LilyPond multiplier:

```
abjad> note = Note(0, (1, 8))
abjad> note.duration.multiplier = Duration(1, 2)
abjad> leaftools.set_preprolated_leaf_duration(note, Duration(5, 48))
[Note("c'8 * 5/6")]
abjad> f(note)
c'8 * 5/6
```

Return list of *leaf* and leaves newly tied to *leaf*. Changed in version 1.1.2: renamed `leaftools.change_leaf_preprolated_duration( )` to `leaftools.set_preprolated_leaf_duration( )`.

leaftools.show_leaves

`abjad.tools.leaftools.`**`show_leaves`**(*leaves*, *template=None*, *suppress_pdf=False*)

New in version 1.1.2. Show *leaves* in temporary piano staff score:

```
abjad> leaves = leaftools.make_leaves([None, 1, (-24, -22, 7, 21), None], (1, 4))
abjad> score = leaftools.show_leaves(leaves) # doctest: +SKIP
```

```
\new Score <<
        \new PianoStaff <<
                \context Staff = "treble" {
                        \clef "treble"
                        r4
                        cs'4
                        <g' a''>4
                        r4
                }
                \context Staff = "bass" {
                        \clef "bass"
                        r4
                        r4
                        <c, d,>4
                        r4
                }
        >>
>>
```

Useful when working with notes, rests, chords not yet added to score.

Return temporary piano staff score.

leaftools.split_leaf_at_prolated_duration_and_rest_right_half

abjad.tools.leaftools.**split_leaf_at_prolated_duration_and_rest_right_half**(*leaf*, *prolated_duration*)

New in version 1.1.1. Split *leaf* at *prolated_duration* and rest right half:

```
abjad> t = Staff(macros.scale(4))
abjad> spannertools.SlurSpanner(t[:])
SlurSpanner(c'8, d'8, e'8, f'8)
abjad> f(t)
\new Staff {
   c'8 (
   d'8
   e'8
   f'8 )
}

abjad> leaftools.split_leaf_at_prolated_duration_and_rest_right_half(t.leaves[1], (1, 32))
([Note("d'32")], [Note("d'16.")])

abjad> f(t)
\new Staff {
   c'8 (
   d'32
   r16.
   e'8
   f'8 )
}
```

Return list of leaves to left of *prolated_duration* together with list of leaves to right of *prolated_duration*. Changed in version 1.1.2: renamed `leaftools.shorten( )` to `leaftools.split_leaf_at_prolated_duration_and_rest_right_half( )`.

leaftools.yield_groups_of_mixed_notes_and_chords_in_sequence

abjad.tools.leaftools.**yield_groups_of_mixed_notes_and_chords_in_sequence**(*sequence*)

New in version 1.1.2. Yield groups of mixed notes and chords in *sequence*:

```
abjad> staff = Staff("c'8 d'8 r8 r8 <e' g'>8 <f' a'>8 g'8 a'8 r8 r8 <b' d''>8 <c'' e''>8")

abjad> f(staff)
\new Staff {
   c'8
   d'8
   r8
   r8
   <e' g'>8
   <f' a'>8
   g'8
   a'8
   r8
   r8
   <b' d''>8
   <c'' e''>8
}

abjad> for group in leaftools.yield_groups_of_mixed_notes_and_chords_in_sequence(staff):
...     group
...
(Note("c'8"), Note("d'8"))
(Chord("<e' g'>8"), Chord("<f' a'>8"), Note("g'8"), Note("a'8"))
(Chord("<b' d''>8"), Chord("<c'' e''>8"))
```

Return generator.

lilyfiletools

lilyfiletools.AbjadRevisionToken

class abjad.tools.lilyfiletools.**AbjadRevisionToken**

Bases: `abjad.core._Immutable._Immutable._Immutable` New in version 1.1.2. Abjad version token:

```
abjad> lilyfiletools.AbjadRevisionToken( )
AbjadRevisionToken(Abjad revision ...)
```

Return Abjad version token.

format

Format contribution of Abjad version token:

```
abjad> lilyfiletools.AbjadRevisionToken( ).format
'Abjad revision ...'
```

Return string.

lilyfiletools.BookBlock

class abjad.tools.lilyfiletools.**BookBlock**

Bases: `abjad.tools.lilyfiletools._BlockNonattributed._BlockNonattributed._BlockNonattribu`

New in version 1.1.2. Abjad model of LilyPond input file book block.

lilyfiletools.BookpartBlock

class abjad.tools.lilyfiletools.**BookpartBlock**

Bases: abjad.tools.lilyfiletools._BlockNonattributed._BlockNonattributed._BlockNonattribu

New in version 1.1.2. Abjad model of LilyPond input file bookpart block.

lilyfiletools.DateTimeToken

class abjad.tools.lilyfiletools.**DateTimeToken**

Bases: abjad.core._Immutable._Immutable._Immutable New in version 1.1.2. Date time token:

```
abjad> lilyfiletools.DateTimeToken( )
DateTimeToken(...)
```

Return date / time token.

format

Format contribution of date time token:

```
abjad> lilyfiletools.DateTimeToken( ).format
'...'
```

Return string.

lilyfiletools.HeaderBlock

class abjad.tools.lilyfiletools.**HeaderBlock**

Bases: abjad.tools.lilyfiletools._BlockAttributed._BlockAttributed._BlockAttributed

New in version 1.1.2. Abjad model of LilyPond input file header block:

```
abjad> header_block = lilyfiletools.HeaderBlock( )
abjad> header_block.composer = markuptools.Markup('Josquin')
abjad> header_block.title = markuptools.Markup('Missa sexti tonus')

abjad> f(header_block)
\header {
   composer = \markup { Josquin }
   title = \markup { Missa sexti tonus }
}
```

Return header block.

lilyfiletools.LayoutBlock

class abjad.tools.lilyfiletools.**LayoutBlock**

Bases: abjad.tools.lilyfiletools._BlockAttributed._BlockAttributed._BlockAttributed

New in version 1.1.2. Abjad model of LilyPond input file layout block.

contexts

lilyfiletools.LilyFile

class abjad.tools.lilyfiletools.**LilyFile**

Bases: list New in version 1.1.2. Abjad model of LilyPond input file:

```
abjad> staff = Staff(macros.scale(4))
abjad> lily_file = lilyfiletools.make_basic_lily_file(staff)
abjad> lily_file.file_initial_user_comments.append('File construct as an example.')
abjad> lily_file.file_initial_user_comments.append('Parts shown here for positioning.')
abjad> lily_file.file_initial_user_includes.append('external-settings-file-1.ly')
abjad> lily_file.file_initial_user_includes.append('external-settings-file-2.ly')
abjad> lily_file.default_paper_size = 'letter', 'portrait'
abjad> lily_file.global_staff_size = 16
abjad> lily_file.header_block.composer = markuptools.Markup('Josquin')
abjad> lily_file.header_block.title = markuptools.Markup('Missa sexti tonus')
abjad> lily_file.layout_block.indent = 0
abjad> lily_file.layout_block.left_margin = 15
abjad> lily_file.paper_block.oddFooterMarkup = markuptools.Markup('The odd-page footer')
abjad> lily_file.paper_block.evenFooterMarkup = markuptools.Markup('The even-page footer')
```

```
abjad> f(lily_file) # doctest: +SKIP
% Abjad revision 3719
% 2010-09-24 09:01

% File construct as an example.
% Parts shown here for positioning.

\version "2.13.32"
\include "english.ly"
\include "/Users/trevorbaca/Documents/abjad/trunk/abjad/cfg/abjad.scm"

\include "external-settings-file-1.ly"
\include "external-settings-file-2.ly"

#(set-default-paper-size "letter" 'portrait)
#(set-global-staff-size 16)

\header {
   composer = \markup { Josquin }
   title = \markup { Missa sexti tonus }
}

\layout {
   indent = #0
   left-margin = #15
}

\paper {
   evenFooterMarkup = \markup { The even-page footer }
   oddFooterMarkup = \markup { The odd-page footer }
}

\new Staff {
   c'8
   d'8
   e'8
   f'8
}
```

default_paper_size
LilyPond default paper size.

file_initial_system_comments
Read-only list of file-initial system comments.

file_initial_system_includes
List of file-initial system include commands.

file_initial_user_comments
Read-only list of file-initial user comments.

file_initial_user_includes
List of file-initial user include commands.

format
Format-time contribution of LilyPond file.

global_staff_size
LilyPond global staff size.

lilyfiletools.LilyPondLanguageToken

class `abjad.tools.lilyfiletools.`**`LilyPondLanguageToken`**
Bases: `abjad.core._Immutable._Immutable._Immutable` New in version 1.1.2. LilyPond language token:

```
abjad> lilyfiletools.LilyPondLanguageToken( )
LilyPondLanguageToken(\include "english.ly")
```

Return LilyPond language token.

format
Format contribution of LilyPond language token:

```
abjad> lilyfiletools.LilyPondLanguageToken( ).format
'\\include "english.ly"'
```

Return string.

lilyfiletools.LilyPondVersionToken

class `abjad.tools.lilyfiletools.`**`LilyPondVersionToken`**
Bases: `abjad.core._Immutable._Immutable._Immutable` New in version 1.1.2. LilyPond version token:

```
abjad> lilyfiletools.LilyPondVersionToken( )
LilyPondVersionToken(\version "...")
```

Return LilyPond version token.

format
Format contribution of LilyPond version token:

```
abjad> lilyfiletools.LilyPondVersionToken( ).format
'\\version "..."'
```

Return string.

lilyfiletools.MidiBlock

class `abjad.tools.lilyfiletools.`**`MidiBlock`**
Bases: `abjad.tools.lilyfiletools._BlockAttributed._BlockAttributed._BlockAttributed`
New in version 1.1.2. Abjad model of LilyPond input file midi block.

lilyfiletools.PaperBlock

class abjad.tools.lilyfiletools.**PaperBlock**

Bases: abjad.tools.lilyfiletools._BlockAttributed._BlockAttributed._BlockAttributed

New in version 1.1.2. Abjad model of LilyPond input file paper block.

minimal_page_breaking

lilyfiletools.ScoreBlock

class abjad.tools.lilyfiletools.**ScoreBlock**

Bases: abjad.tools.lilyfiletools._BlockNonattributed._BlockNonattributed._BlockNonattribu

New in version 1.1.2. Abjad model of LilyPond input file score block.

lilyfiletools.make_basic_lily_file

abjad.tools.lilyfiletools.**make_basic_lily_file**(*music=None*)

New in version 1.1.2. Make basic LilyPond file with *music*:

```
abjad> score = Score([Staff(macros.scale(4))])
abjad> lily_file = lilyfiletools.make_basic_lily_file(score)
abjad> lily_file.header_block.composer = markuptools.Markup('Josquin')
abjad> lily_file.layout_block.indent = 0
abjad> lily_file.paper_block.top_margin = 15
abjad> lily_file.paper_block.left_margin = 15
```

```
abjad> f(lily_file) # doctest: +SKIP
\header {
        composer = \markup { Josquin }
}

\layout {
        indent = #0
}

\paper {
        left-margin = #15
        top-margin = #15
}

\new Score <<
        \new Staff {
                c'8
                d'8
                e'8
                f'8
        }
>>
```

Equip LilyPond file with header, layout and paper blocks.

Return LilyPond file.

marktools

marktools.Annotation

class `abjad.tools.marktools.`**`Annotation`**`(`*name*, *value=None*`)`

Bases: `abjad.tools.marktools.Mark.Mark.Mark` New in version 1.1.2. User-defined annotation:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")

abjad> f(staff)
\new Staff {
   c'8
   d'8
   e'8
   f'8
}

abjad> marktools.Annotation('special pitch', pitchtools.NamedChromaticPitch('ds'))(staff[0])
Annotation('special pitch', NamedChromaticPitch('ds'))(c'8)

abjad> f(staff)
\new Staff {
   c'8
   d'8
   e'8
   f'8
}
```

Annotations contribute no formatting.

Annotations implement `__slots__`.

`name`

Get name of annotation:

```
abjad> annotation = marktools.Annotation('special_pitch', pitchtools.NamedChromaticPitch('ds
abjad> annotation.name
'special_pitch'
```

Set name of annotation:

```
abjad> annotation.name = 'revised special pitch'
abjad> annotation.name
'revised special pitch'
```

Set string.

`value`

Get value of annotation:

```
abjad> annotation = marktools.Annotation('special_pitch', pitchtools.NamedChromaticPitch('ds
abjad> annotation.value
NamedChromaticPitch('ds')
```

Set value of annotation:

```
abjad> annotation.value = pitchtools.NamedChromaticPitch('e')
abjad> annotation.value
NamedChromaticPitch('e')
```

Set arbitrary object.

marktools.Articulation

class `abjad.tools.marktools.`**`Articulation`**`(`**args*`)`
Bases: `abjad.tools.marktools.Mark.Mark.Mark`

Abjad model of musical articulation:

```
abjad> note = Note("c'4")

abjad> marktools.Articulation('staccato')(note)
Articulation('staccato', '-')(c'4)

abjad> f(note)
c'4 -\staccato
```

Articulations implement `__slots__`.

`direction_string`
Get direction string of articulation:

```
abjad> articulation = marktools.Articulation('staccato')
abjad> articulation.direction_string
'-'
```

Set direction string of articulation:

```
abjad> articulation.direction_string = '^'
abjad> articulation.direction_string
'^'
```

Set string.

`format`
Read-only LilyPond format string of articulation:

```
abjad> articulation = marktools.Articulation('staccato', 'up')
abjad> articulation.format
'^\staccato'
```

Return string.

`name_string`
Get name string of articulation:

```
abjad> articulation = marktools.Articulation('staccato', 'up')
abjad> articulation.name_string
'staccato'
```

Set name string of articulation:

```
abjad> articulation.name_string = 'marcato'
abjad> articulation.name_string
'marcato'
```

Set string.

marktools.Comment

class `abjad.tools.marktools.`**`Comment`**`(`*comment_name_string*, *format_slot='opening'*`)`
Bases: `abjad.tools.marktools.Mark.Mark.Mark` New in version 1.1.2. User-defined comment:

```
abjad> note = Note("c'4")

abjad> marktools.Comment('this is a comment')(note)
Comment('this is a comment')(c'4)

abjad> f(note)
% this is a comment
c'4
```

Comments implement `__slots__`.

contents_string

Get contents string of comment:

```
abjad> comment = marktools.Comment('comment contents string')
abjad> comment.contents_string
'comment contents string'
```

Set contents string of comment:

```
abjad> comment.contents_string = 'new comment contents string'
abjad> comment.contents_string
'new comment contents string'
```

Set string.

format

Read-only LilyPond input format of comment:

```
abjad> comment = marktools.Comment('this is a comment.')
abjad> comment.format
'% this is a comment.'
```

Return string.

marktools.LilyPondCommandMark

class `abjad.tools.marktools.`**`LilyPondCommandMark`**(*command_name_string*, *format_slot='opening'*)

Bases: `abjad.tools.marktools.Mark.Mark.Mark` New in version 1.1.2. LilyPond command mark:

```
abjad> staff = Staff(macros.scale(4))
abjad> slur = spannertools.SlurSpanner(staff.leaves)

abjad> lilypond_command = marktools.LilyPondCommandMark('slurDotted')(staff[0])

abjad> f(staff)
\new Staff {
   \slurDotted
   c'8 (
   d'8
   e'8
   f'8 )
}
```

LilyPond command marks implement `__slots__`.

command_name_string

Get command name string of LilyPond command mark:

```
abjad> lilypond_command = marktools.LilyPondCommandMark('slurDotted')
abjad> lilypond_command.command_name_string
'slurDotted'
```

Set command name string of LilyPond command mark:

```
abjad> lilypond_command.command_name_string = 'slurDashed'
abjad> lilypond_command.command_name_string
'slurDashed'
```

Set string.

format
Read-only LilyPond input format of LilyPond command mark:

```
abjad> note = Note(0, (1, 4))
abjad> lilypond_command = marktools.LilyPondCommandMark('slurDotted')(note)
abjad> lilypond_command.format
'\slurDotted'
```

Return string.

marktools.Mark

class `abjad.tools.marktools.`**Mark**
Bases: `object` New in version 1.1.2. Abstract class from which concrete marks inherit:

```
abjad> note = Note("c'4")

abjad> marktools.Mark( )(note)
Mark( )(c'4)
```

Marks override `__call__` to attach to a note, rest or chord.

Marks implement `__slots__`.

attach_mark(*start_component*)
Attach mark to *start_component*:

```
abjad> note = Note("c'4")
abjad> mark = marktools.Mark( )

abjad> mark.attach_mark(note)
Mark( )(c'4)

abjad> mark.start_component
Note("c'4")
```

Return mark.

detach_mark()
Detach mark:

```
abjad> note = Note("c'4")
abjad> mark = marktools.Mark( )(note)

abjad> mark.start_component
Note("c'4")
```

```
abjad> mark.detach_mark( )
Mark( )

abjad> mark.start_component is None
True
```

Return mark.

start_component

Read-only reference to mark start component:

```
abjad> note = Note("c'4")
abjad> mark = marktools.Mark( )(note)

abjad> mark.start_component
Note("c'4")
```

Return component or none.

marktools.StemTremolo

class `abjad.tools.marktools.`**StemTremolo**(*tremolo_flags*)

Bases: `abjad.tools.marktools.Mark.Mark.Mark` New in version 1.1.2. Abjad model of stem tremolo:

```
abjad> note = Note("c'4")

abjad> marktools.StemTremolo(16)(note)
StemTremolo(16)(c'4)

abjad> f(note)
c'4 :16
```

Stem tremolos implement `__slots__`.

format

Read-only LilyPond format string:

```
abjad> stem_tremolo = marktools.StemTremolo(16)
abjad> stem_tremolo.format
':16'
```

Return string.

tremolo_flags

Get tremolo flags:

```
abjad> stem_tremolo = marktools.StemTremolo(16)
abjad> stem_tremolo.tremolo_flags
16
```

Set tremolo flags:

```
abjad> stem_tremolo.tremolo_flags = 32
abjad> stem_tremolo.tremolo_flags
32
```

Set integer.

marktools.apply_articulations_to_notes_and_chords_in_expr

abjad.tools.marktools.**apply_articulations_to_notes_and_chords_in_expr**(*expr*, *articulations*)

New in version 1.1.2. Apply *articulations* to notes and chords in *expr*:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> marktools.apply_articulations_to_notes_and_chords_in_expr(staff, list('^.'))

abjad> f(staff)
\new Staff {
   c'8 -\marcato -\staccato
   d'8 -\marcato -\staccato
   e'8 -\marcato -\staccato
   f'8 -\marcato -\staccato
}
```

Return none.

marktools.detach_annotations_attached_to_component

abjad.tools.marktools.**detach_annotations_attached_to_component**(*component*)

New in version 1.1.2. Detach annotations attached to *component*:

```
abjad> staff = Staff(macros.scale(4))
abjad> slur = spannertools.SlurSpanner(staff.leaves)
abjad> marktools.Annotation('annotation 1')(staff[0])
Annotation('annotation 1')(c'8)
abjad> marktools.Annotation('annotation 2')(staff[0])
Annotation('annotation 2')(c'8)

abjad> f(staff)
\new Staff {
   c'8 (
   d'8
   e'8
   f'8 )
}

abjad> marktools.get_annotations_attached_to_component(staff[0])
(Annotation('annotation 1')(c'8), Annotation('annotation 2')(c'8))

abjad> marktools.detach_annotations_attached_to_component(staff[0])
(Annotation('annotation 1'), Annotation('annotation 2'))

abjad> marktools.get_annotations_attached_to_component(staff[0])
()
```

Return tuple or zero or more annotations detached.

marktools.detach_articulations_attached_to_component

abjad.tools.marktools.**detach_articulations_attached_to_component**(*component*)

New in version 1.1.2. Detach articulations attached to *component*:

```
abjad> staff = Staff(macros.scale(4))
abjad> slur = spannertools.SlurSpanner(staff.leaves)
abjad> marktools.Articulation('^')(staff[0])
Articulation('^', '-')(c'8)
abjad> marktools.Articulation('.')(staff[0])
Articulation('.', '-')(c'8)

abjad> f(staff)
\new Staff {
   c'8 -\marcato -\staccato (
   d'8
   e'8
   f'8 )
}

abjad> marktools.get_articulations_attached_to_component(staff[0])
(Articulation('^', '-')(c'8), Articulation('.', '-')(c'8))

abjad> marktools.detach_articulations_attached_to_component(staff[0])
(Articulation('^', '-'), Articulation('.', '-'))

abjad> marktools.get_articulations_attached_to_component(staff[0])
()
```

Return tuple or zero or more articulations detached.

marktools.detach_comments_attached_to_component

abjad.tools.marktools.**detach_comments_attached_to_component**(*component*)

New in version 1.1.2. Detach comments attached to *component*:

```
abjad> staff = Staff(macros.scale(4))
abjad> slur = spannertools.SlurSpanner(staff.leaves)
abjad> marktools.Comment('comment 1')(staff[0])
Comment('comment 1')(c'8)
abjad> marktools.Comment('comment 2')(staff[0])
Comment('comment 2')(c'8)

abjad> f(staff)
\new Staff {
   % comment 1
   % comment 2
   c'8 (
   d'8
   e'8
   f'8 )
}

abjad> marktools.detach_comments_attached_to_component(staff[0])
(Comment('comment 1'), Comment('comment 2'))

abjad> f(staff)
\new Staff {
   c'8 (
   d'8
   e'8
   f'8 )
}
```

```
abjad> marktools.get_comments_attached_to_component(staff[0])
()
```

Return tuple or zero or more comments.

marktools.detach_lilypond_command_marks_attached_to_component

abjad.tools.marktools.**detach_lilypond_command_marks_attached_to_component**(*component*, *command_name_string=None*)

New in version 1.1.2. Detach LilyPond command marks attached to *component*:

```
abjad> staff = Staff(macros.scale(4))
abjad> slur = spannertools.SlurSpanner(staff.leaves)
abjad> marktools.LilyPondCommandMark('slurDotted')(staff[0])
LilyPondCommandMark('slurDotted')(c'8)
abjad> marktools.LilyPondCommandMark('slurUp')(staff[0])
LilyPondCommandMark('slurUp')(c'8)
```

```
abjad> f(staff)
\new Staff {
   \slurDotted
   \slurUp
   c'8 (
   d'8
   e'8
   f'8 )
}
```

```
abjad> marktools.detach_lilypond_command_marks_attached_to_component(staff[0])
(LilyPondCommandMark('slurDotted'), LilyPondCommandMark('slurUp'))
```

```
abjad> f(staff)
\new Staff {
   c'8 (
   d'8
   e'8
   f'8 )
}
```

Return tuple of zero or more marks detached.

marktools.detach_marks_attached_to_component

abjad.tools.marktools.**detach_marks_attached_to_component**(*component*)

New in version 1.1.2. Detach marks attached to *component*:

```
abjad> staff = Staff(macros.scale(4))
abjad> slur = spannertools.SlurSpanner(staff.leaves)
abjad> marktools.Articulation('^')(staff[0])
Articulation('^', '-')(c'8)
abjad> marktools.Comment('comment 1')(staff[0])
Comment('comment 1')(c'8)
abjad> marktools.LilyPondCommandMark('slurUp')(staff[0])
LilyPondCommandMark('slurUp')(c'8)
```

```
abjad> f(staff)
\new Staff {
   % comment 1
   \slurUp
   c'8 -\marcato (
   d'8
   e'8
   f'8 )
}

abjad> marktools.get_marks_attached_to_component(staff[0])
(Articulation('^', '-')(c'8), Comment('comment 1')(c'8), LilyPondCommandMark('slurUp')(c'8))

abjad> marktools.detach_marks_attached_to_component(staff[0])
(Articulation('^', '-'), Comment('comment 1'), LilyPondCommandMark('slurUp'))

abjad> marktools.get_marks_attached_to_component(staff[0])
()
```

Return tuple or zero or more marks detached.

marktools.detach_stem_tremolos_attached_to_component

abjad.tools.marktools.**detach_stem_tremolos_attached_to_component**(*component*)

New in version 1.1.2. Detach stem tremolos attached to *component*:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> marktools.StemTremolo(16)(staff[0])
StemTremolo(16)(c'8)

abjad> f(staff)
\new Staff {
   c'8 :16
   d'8
   e'8
   f'8
}

abjad> marktools.get_stem_tremolos_attached_to_component(staff[0])
(StemTremolo(16)(c'8),)

abjad> marktools.detach_stem_tremolos_attached_to_component(staff[0])
(StemTremolo(16),)

abjad> marktools.get_stem_tremolos_attached_to_component(staff[0])
()
```

Return tuple or zero or more stem tremolos detached.

marktools.get_annotation_attached_to_component

abjad.tools.marktools.**get_annotation_attached_to_component**(*component*)

New in version 1.1.2. Get exactly one annotation attached to *component*:

```
abjad> staff = Staff(macros.scale(4))
abjad> marktools.Annotation('special information')(staff[0])
Annotation('special information')(c'8)
```

```
abjad> f(staff)
\new Staff {
   c'8
   d'8
   e'8
   f'8
}

abjad> marktools.get_annotation_attached_to_component(staff[0])
Annotation('special information')(c'8)
```

Return one annotation.

Raise missing mark error when no annotation is attached.

Raise extra mark error when more than one annotation is attached.

marktools.get_annotations_attached_to_component

abjad.tools.marktools.**get_annotations_attached_to_component**(*component*)
New in version 1.1.2. Get annotations attached to *component*:

```
abjad> staff = Staff(macros.scale(4))
abjad> marktools.Annotation('annotation 1')(staff[0])
Annotation('annotation 1')(c'8)
abjad> marktools.Annotation('annotation 2')(staff[0])
Annotation('annotation 2')(c'8)

abjad> f(staff)
\new Staff {
   c'8
   d'8
   e'8
   f'8
}

abjad> marktools.get_annotations_attached_to_component(staff[0])
(Annotation('annotation 1')(c'8), Annotation('annotation 2')(c'8))
```

Return tuple of zero or more annotations.

marktools.get_articulations_attached_to_component

abjad.tools.marktools.**get_articulations_attached_to_component**(*component*)
New in version 1.1.2. Get articulations attached to *component*:

```
abjad> staff = Staff(macros.scale(4))
abjad> marktools.Articulation('staccato')(staff[0])
Articulation('staccato', '-')(c'8)
abjad> marktools.Articulation('marcato')(staff[0])
Articulation('marcato', '-')(c'8)

abjad> f(staff)
\new Staff {
   c'8 -\marcato -\staccato
   d'8
   e'8
```

```
    f'8
}

abjad> marktools.get_articulations_attached_to_component(staff[0])
(Articulation('staccato', '-')(c'8), Articulation('marcato', '-')(c'8))
```

Return tuple of zero or more articulations.

marktools.get_comments_attached_to_component

abjad.tools.marktools.**get_comments_attached_to_component**(*component*)

New in version 1.1.2. Get comments attached to *component*:

```
abjad> staff = Staff(macros.scale(4))
abjad> slur = spannertools.SlurSpanner(staff.leaves)
abjad> marktools.Comment('comment 1')(staff[0])
Comment('comment 1')(c'8)
abjad> marktools.Comment('comment 2')(staff[0])
Comment('comment 2')(c'8)

abjad> f(staff)
\new Staff {
    % comment 1
    % comment 2
    c'8 (
    d'8
    e'8
    f'8 )
}

abjad> marktools.get_comments_attached_to_component(staff[0])
(Comment('comment 1')(c'8), Comment('comment 2')(c'8))
```

Return tuple of zero or more comments.

marktools.get_lilypond_command_marks_attached_to_component

abjad.tools.marktools.**get_lilypond_command_marks_attached_to_component**(*component*, *command_name_string=None*)

New in version 1.1.2. Get LilyPond command marks attached to *component*:

```
abjad> staff = Staff(macros.scale(4))
abjad> slur = spannertools.SlurSpanner(staff.leaves)
abjad> marktools.LilyPondCommandMark('slurDotted')(staff[0])
LilyPondCommandMark('slurDotted')(c'8)
abjad> marktools.LilyPondCommandMark('slurUp')(staff[0])
LilyPondCommandMark('slurUp')(c'8)

abjad> f(staff)
\new Staff {
    \slurDotted
    \slurUp
    c'8 (
    d'8
    e'8
```

```
   f'8 )
}

abjad> marktools.get_lilypond_command_marks_attached_to_component(staff[0])
(LilyPondCommandMark('slurDotted')(c'8), LilyPondCommandMark('slurUp')(c'8))
```

Return tuple of zero or more marks.

marktools.get_marks_attached_to_component

abjad.tools.marktools.**get_marks_attached_to_component**(*component*)

New in version 1.1.2. Get all marks attached to *component*':

```
abjad> staff = Staff(macros.scale(4))
abjad> slur = spannertools.SlurSpanner(staff.leaves)
abjad> comment_mark = marktools.Comment('beginning of note content')(staff[0])
abjad> marktools.LilyPondCommandMark('slurDotted')(staff[0])
LilyPondCommandMark('slurDotted')(c'8)

abjad> f(staff)
\new Staff {
   % beginning of note content
   \slurDotted
   c'8 (
   d'8
   e'8
   f'8 )
}

abjad> marktools.get_marks_attached_to_component(staff[0])
(Comment('beginning of note content')(c'8), LilyPondCommandMark('slurDotted')(c'8))
```

Return tuple of zero or more marks. Changed in version 1.1.2: renamed `marktools.get_all_marks_attached_to_component()` to `marktools.get_marks_attached_to_component()`.

marktools.get_stem_tremolos_attached_to_component

abjad.tools.marktools.**get_stem_tremolos_attached_to_component**(*component*)

New in version 1.1.2. Get stem tremolos attached to *component*:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> marktools.StemTremolo(16)(staff[0])
StemTremolo(16)(c'8)

abjad> f(staff)
\new Staff {
   c'8 :16
   d'8
   e'8
   f'8
}

abjad> marktools.get_stem_tremolos_attached_to_component(staff[0])
(StemTremolo(16)(c'8),)
```

Return tuple of zero or more stem tremolos.

marktools.get_value_of_annotation_attached_to_component

abjad.tools.marktools.**get_value_of_annotation_attached_to_component**(*component*, *name*, *default_value=None*)

New in version 1.1.2. Get value of annotation with *name* attached to *component*:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> marktools.Annotation('special dictionary', { })(staff[0])
Annotation('special dictionary', {})(c'8)

abjad> f(staff)
\new Staff {
   c'8
   d'8
   e'8
   f'8
}

abjad> marktools.get_value_of_annotation_attached_to_component(staff[0], 'special dictionary')
{}
```

Return arbitrary value of annotation.

Return *default_value* when no annotation with *name* is attached.

Raise extra mark error when more than one annotation with *name* is attached.

marktools.is_component_with_lilypond_command_mark_attached

abjad.tools.marktools.**is_component_with_lilypond_command_mark_attached**(*expr*, *command_name_string=None*)

True when *expr* is component with LilyPond command mark attached:

```
abjad> note = Note(0, (1, 4))
abjad> marktools.LilyPondCommandMark('stemUp')(note)
LilyPondCommandMark('stemUp')(c'4)

abjad> marktools.is_component_with_lilypond_command_mark_attached(note)
True
```

False otherwise:

```
abjad> note = Note(0, (1, 4))

abjad> marktools.is_component_with_lilypond_command_mark_attached(note)
False
```

Return boolean.

markuptools

markuptools.Markup

class abjad.tools.markuptools.**Markup**(*arg*, *direction_string=None*, *style_string='backslash'*)

Bases: abjad.tools.contexttools.ContextMark.ContextMark.ContextMark

Abjad model of backslash-style LilyPond markup or Scheme-style LilyPond markup.

Initialize backslash-style markup from string:

```
abjad> markup = markuptools.Markup(r'\bold { "This is markup text." }')

abjad> markup
Markup('\\bold { "This is markup text." }')

abjad> f(markup)
\markup { \bold { "This is markup text." } }
```

Initialize Scheme-style markup from string:

```
abjad> markup = markuptools.Markup("(markup #:draw-line '(0 . -1))", style_string = 'scheme')

abjad> markup
Markup("(markup #:draw-line '(0 . -1))")

abjad> f(markup)
#(markup #:draw-line '(0 . -1))
```

Initialize any markup from existing markup:

```
abjad> markup_1 = markuptools.Markup('foo', direction_string = 'up')
abjad> markup_2 = markuptools.Markup(markup_1, direction_string = 'down')

abjad> f(markup_1)
^ \markup { foo }

abjad> f(markup_2)
_ \markup { foo }
```

Attach markup to score components like this:

```
abjad> note = Note("c'4")

abjad> markup = markuptools.Markup(r'\bold { "This is markup text." }')

abjad> markup(note)
Markup('\\bold { "This is markup text." }')

abjad> f(note)
c'4 \markup { \bold { "This is markup text." } }
```

Set *direction_string* to `'up'`, `'down'`, `'neutral'` or none.

Set *style_string* to `'backslash'` or `'scheme'`.

Markup objects are immutable.

format

Read-only LilyPond format of markup:

```
abjad> markup = markuptools.Markup(r'\bold { "This is markup text." }')
abjad> markup.format
'\\markup { \\bold { "This is markup text." } }'
```

Return string.

markuptools.MarkupCommand

class `abjad.tools.markuptools.`**`MarkupCommand`**(*command*, *args*, *markup*, *is_braced=True*)

Bases: `abjad.core._Immutable._Immutable._Immutable`

Abjad model of a LilyPond markup command:

```
abjad> circle = markuptools.MarkupCommand('draw-circle', ['#2.5', '#0.1', '##f'], None)
abjad> square = markuptools.MarkupCommand('rounded-box', None, ['hello?'])
abjad> line = markuptools.MarkupCommand('line', None, [square, 'wow!'])
abjad> rotate = markuptools.MarkupCommand('rotate', ['#60'], [line])
abjad> combine = markuptools.MarkupCommand('combine', None, [rotate, circle], is_braced = False)

abjad> print combine
\combine \rotate #60 \line { \rounded-box hello? wow! } \draw-circle #2.5 #0.1 ##f
```

Insert markup command in markup to attach to score components:

```
abjad> note = Note("c'4")

abjad> markup = markuptools.Markup(combine)

abjad> markup(note)
Markup('\\combine \\rotate #60 \\line { \\rounded-box hello? wow! } \\draw-circle #2.5 #0.1 ##f'

abjad> f(note)
c'4 \markup { \combine \rotate #60 \line { \rounded-box hello? wow! } \draw-circle #2.5 #0.1 ##f
```

Markup commands are immutable.

args
Read-only tuple of markup command arguments.

command
Read-only string of markup command command-name.

format
Read-only format of markup command:

```
abjad> markup_command = markuptools.MarkupCommand('draw-circle', ['#2.5', '#0.1', '##f'], No
abjad> markup_command.format
'\\draw-circle #2.5 #0.1 ##f'
```

Return list of strings.

is_braced
Read-only boolean of markup command bracing.

markup
Read-only tuple of markup command's child markup.

report(*output='screen'*)
Report, in an indented human-readable format, the structure of a formatted MarkupCommand.

markuptools.get_markup_attached_to_component

`abjad.tools.markuptools.`**`get_markup_attached_to_component`**(*component*)

New in version 1.1.2. Get markup attached to *component*:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> slur = spannertools.SlurSpanner(staff[:])
abjad> markuptools.Markup('foo')(staff[0])
Markup('foo')
abjad> markuptools.Markup('bar')(staff[0])
Markup('bar')
```

```
abjad> f(staff)
\new Staff {
   c'8 - \markup { \column { foo bar } } (
   d'8
   e'8
   f'8 )
}
```

```
abjad> markuptools.get_markup_attached_to_component(staff[0])
(Markup('foo'), Markup('bar'))
```

Return tuple of zero or more markup objects.

markuptools.make_big_centered_page_number_markup

`abjad.tools.markuptools.`**`make_big_centered_page_number_markup`**`(`*text=None*`)`

New in version 1.1.1. Make big centered page number markup:

```
abjad> markup = markuptools.make_big_centered_page_number_markup( )
```

```
abjad> f(markup)
\markup {
   \fill-line {
   \bold \fontsize #3 \concat {
   \on-the-fly #print-page-number-check-first
   \fromproperty #'page:page-number-string } } }
```

Return markup. Changed in version 1.1.2: renamed `markuptools.big_centered_page_number( )` to `markuptools.make_big_centered_page_number_markup( )`.

markuptools.remove_markup_attached_to_component

`abjad.tools.markuptools.`**`remove_markup_attached_to_component`**`(`*component*`)`

New in version 1.1.2. Remove markup attached to *component*:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> slur = spannertools.SlurSpanner(staff[:])
abjad> markuptools.Markup('foo')(staff[0])
Markup('foo')
abjad> markuptools.Markup('bar')(staff[0])
Markup('bar')
```

```
abjad> f(staff)
\new Staff {
   c'8 - \markup { \column { foo bar } } (
   d'8
   e'8
   f'8 )
}
```

```
abjad> markuptools.remove_markup_attached_to_component(staff[0])
(Markup('foo'), Markup('bar'))

abjad> f(staff)
\new Staff {
   c'8 (
   d'8
   e'8
   f'8 )
}
```

Return tuple of zero or more markup objects.

markuptools.remove_markup_from_leaves_in_expr

`abjad.tools.markuptools.`**`remove_markup_from_leaves_in_expr`**`(`*expr*`)`

New in version 1.1.1. Remove markup from leaves in *expr*:

```
abjad> staff = Staff(macros.scale(4))
abjad> leaftools.label_leaves_in_expr_with_pitch_class_numbers(staff)
abjad> f(staff)
\new Staff {
        c'8 _ \markup { \small 0 }
        d'8 _ \markup { \small 2 }
        e'8 _ \markup { \small 4 }
        f'8 _ \markup { \small 5 }
}

abjad> markuptools.remove_markup_from_leaves_in_expr(staff)
abjad> f(staff)
\new Staff {
        c'8
        d'8
        e'8
        f'8
}
```

Return none. Changed in version 1.1.2: renamed `label.clear_leaves( )` to `markuptools.remove_markup_from_leaves_in_expr( )`.

mathtools

mathtools.arithmetic_mean

`abjad.tools.mathtools.`**`arithmetic_mean`**`(`*sequence*`)`

New in version 1.1.1. Arithmetic means of *sequence* as an exact integer:

```
abjad> mathtools.arithmetic_mean([1, 2, 2, 20, 30])
11
```

As a rational:

```
abjad> mathtools.arithmetic_mean([1, 2, 20])
Fraction(23, 3)
```

As a float:

```
abjad> mathtools.arithmetic_mean([2, 2, 20.0])
8.0
```

Return number. Changed in version 1.1.2: renamed `seqtools.arithmetic_mean( )` to `mathtools.arithmetic_mean( )`.

mathtools.binomial_coefficient

`abjad.tools.mathtools.`**`binomial_coefficient`**`(`*n*, *k*`)`

New in version 1.1.2. Binomial coefficient of *n* choose *k*:

```
abjad> for k in range(8):
...     print k, '\t', mathtools.binomial_coefficient(8, k)
...
0   1
1   8
2   28
3   56
4   70
5   56
6   28
7   8
```

Return positive integer.

mathtools.cumulative_products

`abjad.tools.mathtools.`**`cumulative_products`**`(`*sequence*`)`

Cumulative products of *sequence*:

```
abjad> mathtools.cumulative_products([1, 2, 3, 4, 5, 6, 7, 8])
[1, 2, 6, 24, 120, 720, 5040, 40320]
```

```
abjad> mathtools.cumulative_products([1, -2, 3, -4, 5, -6, 7, -8])
[1, -2, -6, 24, 120, -720, -5040, 40320]
```

Raise type error when *sequence* is neither list nor tuple.

Raise value error on empty *sequence*.

Return list. Changed in version 1.1.2: renamed `seqtools.cumulative_products( )` to `mathtools.cumulative_products( )`.

mathtools.cumulative_signed_weights

`abjad.tools.mathtools.`**`cumulative_signed_weights`**`(`*sequence*`)`

Cumulative signed weights of *sequence*:

```
abjad> l = [1, -2, -3, 4, -5, -6, 7, -8, -9, 10]
abjad> mathtools.cumulative_signed_weights(l)
[1, -3, -6, 10, -15, -21, 28, -36, -45, 55]
```

Raise type error when *sequence* is not a list.

For cumulative (unsigned) weights use `mathtools.cumulative_sums([abs(x) for x in l])`.

Return list. Changed in version 1.1.2: renamed `seqtools.cumulative_weights_signed( )` to `mathtools.cumulative_signed_weights( )`.

mathtools.cumulative_sums

`abjad.tools.mathtools.`**`cumulative_sums`**`(`*sequence*`)`

Cumulative sums of *sequence*:

```
abjad> mathtools.cumulative_sums([1, 2, 3, 4, 5, 6, 7, 8])
[1, 3, 6, 10, 15, 21, 28, 36]
```

Raise type error when *sequence* is neither list nor tuple.

Raise value error on empty *sequence*.

Return list. Changed in version 1.1.2: renamed `seqtools.cumulative_sums( )` to `mathtools.cumulative_sums( )`.

mathtools.cumulative_sums_zero

`abjad.tools.mathtools.`**`cumulative_sums_zero`**`(`*sequence*`)`

Cumulative sums of *sequence* starting from `0`:

```
abjad> mathtools.cumulative_sums_zero([1, 2, 3, 4, 5, 6, 7, 8])
[0, 1, 3, 6, 10, 15, 21, 28, 36]
```

Return `[0]` on empty *sequence*:

```
abjad> mathtools.cumulative_sums_zero([ ])
[0]
```

Return list. Changed in version 1.1.2: renamed `mathtools.cumulative_sums_zero( )` to `mathtools.cumulative_sums_zero( )`.

mathtools.cumulative_sums_zero_pairwise

`abjad.tools.mathtools.`**`cumulative_sums_zero_pairwise`**`(`*sequence*`)`

List pairwise cumulative sums of *sequence* from `0`:

```
abjad> mathtools.cumulative_sums_zero_pairwise([1, 2, 3, 4, 5, 6])
[(0, 1), (1, 3), (3, 6), (6, 10), (10, 15), (15, 21)]
```

Return list of pairs. Changed in version 1.1.2: renamed `seqtools.pairwise_cumulative_sums_zero( )` to `mathtools.cumulative_sums_zero_pairwise( )`.

mathtools.difference_series

`abjad.tools.mathtools.`**`difference_series`**`(`*sequence*`)`

Difference series of *sequence*:

```
abjad> mathtools.difference_series([1, 1, 2, 3, 5, 5, 6])
[0, 1, 1, 2, 0, 1]
```

Return list. Changed in version 1.1.2: renamed `seqtools.difference_series( )` to `mathtools.difference_series( )`.

mathtools.divide_number_by_ratio

`abjad.tools.mathtools.`**`divide_number_by_ratio`**`(`*number*, *ratio*`)`

Divide integer by *ratio*:

```
abjad> mathtools.divide_number_by_ratio(1, [1, 1, 3])
[Fraction(1, 5), Fraction(1, 5), Fraction(3, 5)]
```

Divide fraction by *ratio*:

```
abjad> mathtools.divide_number_by_ratio(Fraction(1), [1, 1, 3])
[Fraction(1, 5), Fraction(1, 5), Fraction(3, 5)]
```

Divide float by ratio:

```
abjad> mathtools.divide_number_by_ratio(1.0, [1, 1, 3])
[0.20000000000000001, 0.20000000000000001, 0.60000000000000009]
```

Raise type error on nonnumeric *number*.

Raise type error on noninteger in *ratio*.

Return list of fractions or list of floats. Changed in version 1.1.2: renamed `mathtools.divide_number_by_ratio( )` to `mathtools.divide_number_by_ratio( )`.

mathtools.divisors

`abjad.tools.mathtools.`**`divisors`**`(`*n*`)`

Positive divisors of integer *n* in increasing order:

```
abjad> mathtools.divisors(84)
[1, 2, 3, 4, 6, 7, 12, 14, 21, 28, 42, 84]

abjad> for x in range(10, 20):
...     print x, mathtools.divisors(x)
...
10 [1, 2, 5, 10]
11 [1, 11]
12 [1, 2, 3, 4, 6, 12]
13 [1, 13]
14 [1, 2, 7, 14]
15 [1, 3, 5, 15]
16 [1, 2, 4, 8, 16]
17 [1, 17]
18 [1, 2, 3, 6, 9, 18]
19 [1, 19]
```

Allow nonpositive *n*:

```
abjad> mathtools.divisors(-27)
[1, 3, 9, 27]
```

Raise type error on noninteger *n*.

Raise not implemented error on `0`.

Return list of positive integers.

mathtools.factors

abjad.tools.mathtools.**factors**(*n*)

Integer factors of positive integer *n* in increasing order:

```
abjad> mathtools.factors(84)
[1, 2, 2, 3, 7]

abjad> for n in range(10, 20):
...   print n, mathtools.factors(n)
...
10 [1, 2, 5]
11 [1, 11]
12 [1, 2, 2, 3]
13 [1, 13]
14 [1, 2, 7]
15 [1, 3, 5]
16 [1, 2, 2, 2, 2]
17 [1, 17]
18 [1, 2, 3, 3]
19 [1, 19]
```

Raise type error on noninteger *n*.

Raise value error on nonpositive *n*.

Return list of one or more positive integers.

mathtools.get_shared_numeric_sign

abjad.tools.mathtools.**get_shared_numeric_sign**(*sequence*)

Return `1` when all *sequence* elements are positive:

```
abjad> mathtools.get_shared_numeric_sign([1, 2, 3])
1
```

Return `-1` when all *sequence* elements are negative:

```
abjad> mathtools.get_shared_numeric_sign([-1, -2, -3])
-1
```

Return `0` on empty *sequence*:

```
abjad> mathtools.get_shared_numeric_sign([ ])
0
```

Otherwise return none:

```
abjad> mathtools.get_shared_numeric_sign([1, 2, -3]) is None
True
```

Return `1`, `-1`, `0` or none. Changed in version 1.1.2: renamed `seqtools.sign( )` to `mathtools.get_shared_numeric_sign( )`.

mathtools.greatest_common_divisor

abjad.tools.mathtools.**greatest_common_divisor**(**integers*)

New in version 1.1.2. Greatest common divisor of *integers*:

```
abjad> mathtools.greatest_common_divisor(84, -94, -144)
2
```

Allow nonpositive *integers*.

Raise type error on noninteger *integers*.

Raise not implemented error when `0` in *integers*.

Return positive integer.

mathtools.greatest_multiple_less_equal

`abjad.tools.mathtools.`**`greatest_multiple_less_equal`**`(`*m*, *n*`)`

Greatest integer multiple of *m* less than or equal to *n*:

```
abjad> mathtools.greatest_multiple_less_equal(10, 47)
40

abjad> for m in range(1, 10):
...     print m, mathtools.greatest_multiple_less_equal(m, 47)
...
1 47
2 46
3 45
4 44
5 45
6 42
7 42
8 40
9 45

abjad> for n in range(10, 100, 10):
...     print mathtools.greatest_multiple_less_equal(7, n), n
...
7 10
14 20
28 30
35 40
49 50
56 60
70 70
77 80
84 90
```

Raise type error on nonnumeric *m*.

Raise type error on nonnumeric *n*.

Return nonnegative integer.

mathtools.greatest_power_of_two_less_equal

`abjad.tools.mathtools.`**`greatest_power_of_two_less_equal`**`(`*n*, *i=0*`)`

Greatest integer power of two less than or equal to positive *n*:

```
abjad> for n in range(10, 20):
...     print '\t%s\t%s' % (n, mathtools.greatest_power_of_two_less_equal(n))
```

```
...
   10 8
   11 8
   12 8
   13 8
   14 8
   15 8
   16 16
   17 16
   18 16
   19 16
```

Greatest-but-`i` integer power of `2` less than or equal to positive *n*:

```
abjad> for n in range(10, 20):
...     print '\t%s\t%s' % (n, mathtools.greatest_power_of_two_less_equal(n, i = 1))
...
   10 4
   11 4
   12 4
   13 4
   14 4
   15 4
   16 8
   17 8
   18 8
   19 8
```

Raise type error on nonnumeric *n*.

Raise value error on nonpositive *n*.

Return positive integer.

mathtools.integer_equivalent_number_to_integer

`abjad.tools.mathtools.`**`integer_equivalent_number_to_integer`**`(`*number*`)`

New in version 1.1.2. Integer-equivalent *number* to integer:

```
abjad> mathtools.integer_equivalent_number_to_integer(17.0)
17
```

Return noninteger-equivalent number unchanged:

```
abjad> mathtools.integer_equivalent_number_to_integer(17.5)
17.5
```

Raise type error on nonnumber input.

Return number.

mathtools.integer_to_base_k_tuple

`abjad.tools.mathtools.`**`integer_to_base_k_tuple`**`(`*n*, *k*`)`

New in version 1.1.2. Nonnegative integer *n* to base-*k* tuple:

```
abjad> mathtools.integer_to_base_k_tuple(1066, 10)
(1, 0, 6, 6)
```

Return tuple of one or more positive integers.

mathtools.integer_to_binary_string

abjad.tools.mathtools.**integer_to_binary_string**(*n*)

Positive integer *n* to binary string:

```
abjad> mathtools.integer_to_binary_string(5)
'101'

abjad> for n in range(1, 17):
...     print '\t%s\t%s' % (n, mathtools.integer_to_binary_string(n))
...
   1  1
   2  10
   3  11
   4  100
   5  101
   6  110
   7  111
   8  1000
   9  1001
   10 1010
   11 1011
   12 1100
   13 1101
   14 1110
   15 1111
   16 10000
```

Return string. Changed in version 1.1.2: renamed `mathtools.binary_string( )` to `mathtools.integer_to_binary_string( )`.

mathtools.interpolate_cosine

abjad.tools.mathtools.**interpolate_cosine**(*y1*, *y2*, *mu*)

Cosine interpolate *y1* and *y2* with *mu* normalized `[0, 1]`:

```
abjad> mathtools.interpolate_cosine(0, 1, 0.5)
0.49999999999999994
```

Return float. Changed in version 1.1.2: renamed `interpolate.cosine( )` to `mathtools.interpolate_cosine( )`.

mathtools.interpolate_divide

abjad.tools.mathtools.**interpolate_divide**(*total*, *start_frac*, *stop_frac*, *exp='cosine'*)

Divide *total* into segments of sizes computed from interpolating between *start_frac* and *stop_frac*:

```
abjad> mathtools.interpolate_divide(10, 1, 1, exp=1)
[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
abjad> sum(_)
10.0
```

```
abjad> mathtools.interpolate_divide(10, 5, 1)
[4.7986734489043181, 2.8792040693425909, 1.3263207210948171,
0.99580176065827419]
abjad> sum(_)
10.0
```

Set `exp='cosine'` for cosine interpolation.

Set *exp* to a numeric value for exponential interpolation with *exp* as the exponent.

Scale resulting segments so that their sum equals exactly *total*.

Return a list of floats. Changed in version 1.1.2: renamed `interpolate.divide( )` to `mathtools.interpolate_divide( )`.

mathtools.interpolate_divide_multiple

`abjad.tools.mathtools.`**`interpolate_divide_multiple`**`(`*totals*, *key_values*, *exp='cosine'*`)`

New in version 1.1.2. Interpolate *key_values* such that the sum of the resulting interpolated values equals the given *totals*:

```
abjad> mathtools.interpolate_divide_multiple([100, 50], [20, 10, 20]) # doctest: +SKIP
[19.4487, 18.5201, 16.2270, 13.7156, 11.7488, 10.4879,
9.8515, 9.5130, 10.4213, 13.0736, 16.9918]
```

The operation is the same as `mathtools.interpolate_divide( )`. But this function takes multiple *totals* and *key_values* at once.

Precondition: `len(totals) == len(key_values) - 1`.

Set *totals* equal to a list or tuple of the total sum of interpolated values.

Set *key_values* equal a list or tuple of key values to interpolate.

Set *exp* to *consine* for consine interpolation.

Set *exp* to a number for exponential interpolation.

Returns a list of floats. Changed in version 1.1.2: renamed `interpolate.divide_multiple( )` to `mathtools.interpolate_divide_multiple( )`.

mathtools.interpolate_exponential

`abjad.tools.mathtools.`**`interpolate_exponential`**`(`*y1*, *y2*, *mu*, *exp=1*`)`

Exponential interpolate *y1* and *y2* with *mu* normalized `[0, 1]`:

```
abjad> mathtools.interpolate_exponential(0, 1, 0.5, 4)
0.0625
```

Set *exp* equal to the exponent of interpolation.

Return float. Changed in version 1.1.2: renamed `interpolate.exponential( )` to `mathtools.interpolate_exponential( )`.

mathtools.interpolate_linear

`abjad.tools.mathtools.`**`interpolate_linear`**`(`*y1*, *y2*, *mu*`)`

Linear interpolate *y1* and *y2* with *mu* normalized `[0, 1]`:

```
abjad> mathtools.interpolate_linear(0, 1, 0.5)
0.5
```

Return float. Changed in version 1.1.2: renamed `interpolate.linear( )` to `mathtools.interpolate_linear( )`.

mathtools.is_assignable_integer

`abjad.tools.mathtools.`**`is_assignable_integer`**`(`*expr*`)`

New in version 1.1.2. True when *expr* is equivalent to an integer and can be written without recourse to ties:

```
abjad> for n in range(0, 16 + 1):
...     print '%s\t%s' % (n, mathtools.is_assignable_integer(n))
...
0  False
1  True
2  True
3  True
4  True
5  False
6  True
7  True
8  True
9  False
10 False
11 False
12 True
13 False
14 True
15 True
16 True
```

Otherwise false.

Return boolean. Changed in version 1.1.2: renamed `mathtools.is_assignable( )` to `mathtools.is_assignable_integer( )`.

mathtools.is_dotted_integer

`abjad.tools.mathtools.`**`is_dotted_integer`**`(`*expr*`)`

New in version 1.1.2. True when *expr* is equivalent to a positive integer and can be written with zero or more dots:

```
abjad> for expr in range(16):
...     print '%s %s' % (expr, mathtools.is_dotted_integer(expr))
...
0       False
1       False
2       False
3       True
4       False
5       False
6       True
7       True
8       False
9       False
10      False
```

```
11      False
12      True
13      False
14      True
15      True
```

Otherwise false.

Return boolean.

Integer *n* qualifies as dotted when `abs(n)` is of the form `2**j * (2**k - 1)` with integers `0 <= j`, `2 < k`.

mathtools.is_integer_equivalent_number

`abjad.tools.mathtools.`**`is_integer_equivalent_number`**`(`*expr*`)`

New in version 1.1.2. True *expr* is a number and *expr* is equivalent to an integer:

```
abjad> mathtools.is_integer_equivalent_number(12.0)
True
```

Otherwise false:

```
abjad> mathtools.is_integer_equivalent_number(Duration(1, 2))
False
```

Return boolean.

mathtools.is_negative_integer

`abjad.tools.mathtools.`**`is_negative_integer`**`(`*expr*`)`

New in version 1.1.2. True when *expr* equals a negative integer:

```
abjad> mathtools.is_negative_integer(-1)
True
```

Otherwise false:

```
abjad> mathtools.is_negative_integer(0)
False
```

```
abjad> mathtools.is_negative_integer(99)
False
```

Return boolean.

mathtools.is_nonnegative_integer

`abjad.tools.mathtools.`**`is_nonnegative_integer`**`(`*expr*`)`

New in version 1.1.2. True when *expr* equals a nonnegative integer:

```
abjad> mathtools.is_nonnegative_integer(99)
True
```

```
abjad> mathtools.is_nonnegative_integer(0)
True
```

Otherwise false:

```
abjad> mathtools.is_nonnegative_integer(-1)
False
```

Return boolean.

mathtools.is_nonnegative_integer_equivalent_number

abjad.tools.mathtools.**is_nonnegative_integer_equivalent_number**(*expr*)
New in version 1.1.2. True when *expr* is a nonnegative integer-equivalent number. Otherwise false:

```
abjad> mathtools.is_nonnegative_integer_equivalent_number(Duration(4, 2))
True
```

Return boolean.

mathtools.is_nonnegative_integer_power_of_two

abjad.tools.mathtools.**is_nonnegative_integer_power_of_two**(*expr*)
True when *expr* is a nonnegative integer power of `2`:

```
abjad> for n in range(10):
...     print n, mathtools.is_nonnegative_integer_power_of_two(n)
...
0 True
1 True
2 True
3 False
4 True
5 False
6 False
7 False
8 True
9 False
```

Otherwise false.

Return boolean. Changed in version 1.1.2: renamed `mathtools.is_power_of_two( )` to `mathtools.is_nonnegative_integer_power_of_two( )`.

mathtools.is_positive_integer

abjad.tools.mathtools.**is_positive_integer**(*expr*)
New in version 1.1.2. True when *expr* equals a positive integer:

```
abjad> mathtools.is_positive_integer(99)
True
```

Otherwise false:

```
abjad> mathtools.is_positive_integer(0)
False
```

```
abjad> mathtools.is_positive_integer(-1)
False
```

Return boolean.

mathtools.is_positive_integer_equivalent_number

`abjad.tools.mathtools.`**`is_positive_integer_equivalent_number`**(*expr*)

New in version 1.1.2. True when *expr* is a positive integer-equivalent number. Otherwise false:

```
abjad> mathtools.is_positive_integer_equivalent_number(Duration(4, 2))
True
```

Return boolean.

mathtools.least_common_multiple

`abjad.tools.mathtools.`**`least_common_multiple`**(**integers*)

Least common multiple of positive *integers*:

```
abjad> mathtools.least_common_multiple(2, 4, 5, 10, 20)
20
```

Return positive integer.

mathtools.least_multiple_greater_equal

`abjad.tools.mathtools.`**`least_multiple_greater_equal`**(*m*, *n*)

Return the least integer multiple of *m* greater than or equal to *n*.

```
abjad> mathtools.least_multiple_greater_equal(10, 47)
50
```

```
abjad> for m in range(1, 10):
...     print m, mathtools.least_multiple_greater_equal(m, 47)
...
1 47
2 48
3 48
4 48
5 50
6 48
7 49
8 48
9 54
```

```
abjad> for n in range(10, 100, 10):
...     print mathtools.least_multiple_greater_equal(7, n), n
...
14 10
21 20
35 30
42 40
56 50
63 60
70 70
84 80
91 90
```

Return integer.

mathtools.least_power_of_two_greater_equal

abjad.tools.mathtools.**least_power_of_two_greater_equal**(*n*, *i=0*)

Return least integer power of two greater than or equal to positive *n*:

```
abjad> for n in range(10, 20):
...     print '\t%s\t%s' % (n, mathtools.least_power_of_two_greater_equal(n))
...
   10 16
   11 16
   12 16
   13 16
   14 16
   15 16
   16 16
   17 32
   18 32
   19 32
```

When `i = 1`, return the first integer power of `2` greater than the least integer power of `2` greater than or equal to *n*.

```
abjad> for n in range(10, 20):
...     print '\t%s\t%s' % (n, mathtools.least_power_of_two_greater_equal(n, i = 1))
...
   10 32
   11 32
   12 32
   13 32
   14 32
   15 32
   16 32
   17 64
   18 64
   19 64
```

When `i = 2`, return the second integer power of `2` greater than the least integer power of `2` greater than or equal to *n*, and, in general, return the `i` th integer power of `2` greater than the least integer power of `2` greater than or equal to *n*.

Raise type error on nonnumeric *n*.

Raise value error on nonpositive *n*.

Return integer.

mathtools.next_integer_partition

abjad.tools.mathtools.**next_integer_partition**(*integer_partition*)

New in version 1.1.2. Next integer partition following *integer_partition* in descending lex order:

```
abjad> mathtools.next_integer_partition((8, 3))
(8, 2, 1)

abjad> mathtools.next_integer_partition((8, 2, 1))
(8, 1, 1, 1)

abjad> mathtools.next_integer_partition((8, 1, 1, 1))
(7, 4)
```

Input *integer_partition* must be sequence of positive integers.

Return integer partition as tuple of positive integers.

mathtools.partition_integer_by_ratio

abjad.tools.mathtools.**partition_integer_by_ratio**(*n*, *ratio*)

Partition positive integer-equivalent *n* by *ratio*:

```
abjad> mathtools.partition_integer_by_ratio(10, [1, 2])
[3, 7]
```

Partition positive integer-equivalent *n* by *ratio* with negative parts:

```
abjad> mathtools.partition_integer_by_ratio(10, [1, -2])
[3, -7]
```

Partition negative integer-equivalent *n* by *ratio*:

```
abjad> mathtools.partition_integer_by_ratio(-10, [1, 2])
[-3, -7]
```

Partition negative integer-equivalent *n* by *ratio* with negative parts:

```
abjad> mathtools.partition_integer_by_ratio(-10, [1, -2])
[-3, 7]
```

Return result with weight equal to absolute value of *n*.

Raise type error on noninteger *n*.

Return list of integers.

mathtools.partition_integer_into_canonic_parts

abjad.tools.mathtools.**partition_integer_into_canonic_parts**(*n*, *direction='big-endian'*)

Partition integer *n* into big-endian or small-endian parts.

Return all parts positive on positive *n*:

```
abjad> for n in range(1, 11):
...     print n, mathtools.partition_integer_into_canonic_parts(n)
...
1 (1,)
2 (2,)
3 (3,)
4 (4,)
5 (4, 1)
6 (6,)
7 (7,)
8 (8,)
9 (8, 1)
10 (8, 2)
```

Return all parts negative on negative *n*:

```
abjad> for n in reversed(range(-20, -10)):
...     print n, mathtools.partition_integer_into_canonic_parts(n)
...
-11 (-8, -3)
-12 (-12,)
-13 (-12, -1)
-14 (-14,)
-15 (-15,)
-16 (-16,)
-17 (-16, -1)
-18 (-16, -2)
-19 (-16, -3)
-20 (-16, -4)
```

Return little-endian tuple When `direction = 'little-endian'`:

```
abjad> for n in range(11, 21):
...     print n, mathtools.partition_integer_into_canonic_parts(n, direction = 'little-endian')
...
11 (3, 8)
12 (12,)
13 (1, 12)
14 (14,)
15 (15,)
16 (16,)
17 (1, 16)
18 (2, 16)
19 (3, 16)
20 (4, 16)
```

Return big-endian tuple `t = (t_0, ..., t_j)` such that

- `sum(t) == n`
- `t_i` can be written without recourse to ties, and
- `t_(i + 1) < t_i` for every `t_i` in `t`.

Raise type error on noninteger *n*.

Return tuple of one or more integers.

mathtools.partition_integer_into_halves

`abjad.tools.mathtools.`**`partition_integer_into_halves`**(*n*, *bigger='left'*, *even='allowed'*)

Write positive integer *n* as the pair `t = (left, right)` such that `n == left + right`.

When *n* is odd the greater part of `t` corresponds to the value of *bigger*:

```
abjad> mathtools.partition_integer_into_halves(7, bigger = 'left')
(4, 3)
abjad> mathtools.partition_integer_into_halves(7, bigger = 'right')
(3, 4)
```

Likewise when *n* is even and `even = 'disallowed'`:

```
abjad> mathtools.partition_integer_into_halves(8, bigger = 'left', even = 'disallowed')
(5, 3)
```

```
abjad> mathtools.partition_integer_into_halves(8, bigger = 'right', even = 'disallowed')
(3, 5)
```

But when *n* is even and `even = 'allowed'` then `left == right` and *bigger* is ignored:

```
abjad> mathtools.partition_integer_into_halves(8)
(4, 4)
abjad> mathtools.partition_integer_into_halves(8, bigger = 'left')
(4, 4)
abjad> mathtools.partition_integer_into_halves(8, bigger = 'right')
(4, 4)
```

When *n* is `0` return `(0, 0)`:

```
abjad> mathtools.partition_integer_into_halves(0)
(0, 0)
```

When *n* is `0` and `even = 'disallowed'` raise partition error.

Raise type error on noninteger *n*.

Raise value error on negative *n*.

Return pair of positive integers.

mathtools.partition_integer_into_thirds

`abjad.tools.mathtools.`**`partition_integer_into_thirds`**`(`*n*, *smallest='middle'*, *biggest='middle'*`)`

Partition positive integer *n* into `left, middle, right` parts.

When `n % 3 == 0`, `left == middle == right`:

```
abjad> mathtools.partition_integer_into_thirds(9)
(3, 3, 3)
```

When `n % 3 == 1`, set biggest part to *biggest*:

```
abjad> mathtools.partition_integer_into_thirds(10, biggest = 'left')
(4, 3, 3)
abjad> mathtools.partition_integer_into_thirds(10, biggest = 'middle')
(3, 4, 3)
abjad> mathtools.partition_integer_into_thirds(10, biggest = 'right')
(3, 3, 4)
```

When `n % 3 == 2`, set smallest part to *smallest*:

```
abjad> mathtools.partition_integer_into_thirds(11, smallest = 'left')
(3, 4, 4)
abjad> mathtools.partition_integer_into_thirds(11, smallest = 'middle')
(4, 3, 4)
abjad> mathtools.partition_integer_into_thirds(11, smallest = 'right')
(4, 4, 3)
```

Raise type error on noninteger *n*.

Raise value error on nonpositive *n*.

Return triple of positive integers.

mathtools.partition_integer_into_units

`abjad.tools.mathtools.`**`partition_integer_into_units`**`(n)`

Partition positive integer into units:

```
abjad> mathtools.partition_integer_into_units(6)
[1, 1, 1, 1, 1, 1]
```

Partition negative integer into units:

```
abjad> mathtools.partition_integer_into_units(-5)
[-1, -1, -1, -1, -1]
```

Partition `0` into units:

```
abjad> mathtools.partition_integer_into_units(0)
[]
```

Return list of zero or more parts with absolute value equal to `1`.

mathtools.remove_powers_of_two

`abjad.tools.mathtools.`**`remove_powers_of_two`**`(n)`

Remove powers of `2` from the factors of positive integer *n*:

```
abjad> for n in range(10, 100, 10):
...     print '\t%s\t%s' % (n, mathtools.remove_powers_of_two(n))
...
   10 5
   20 5
   30 15
   40 5
   50 25
   60 15
   70 35
   80 5
   90 45
```

Raise type error on noninteger *n*.

Raise value error on nonpositive *n*.

Return positive integer.

mathtools.sign

`abjad.tools.mathtools.`**`sign`**`(n)`

Return `-1` on negative *n*:

```
abjad> mathtools.sign(-96.2)
-1
```

Return `0` when *n* is `0`:

```
abjad> mathtools.sign(0)
0
```

Return `1` on positive *n*:

```
abjad> mathtools.sign(Duration(9, 8))
1
```

Return `-1`, `0` or `1`.

mathtools.weight

`abjad.tools.mathtools.`**`weight`**`(`*sequence*, *start=0*`)`

Sum of the absolute value of the elements in *sequence*:

```
abjad> mathtools.weight([-1, -2, 3, 4, 5])
15
```

Absolute value of *start*:

```
abjad> mathtools.weight([ ])
0
```

Return nonnegative integer. Changed in version 1.1.2: renamed `seqtools.weight( )` to `mathtools.weight( )`.

mathtools.yield_all_compositions_of_integer

`abjad.tools.mathtools.`**`yield_all_compositions_of_integer`**`(`*n*`)`

New in version 1.1.2. Yield all compositions of positive integer *n* in descending lex order:

```
abjad> for integer_composition in mathtools.yield_all_compositions_of_integer(5):
...     integer_composition
...
(5,)
(4, 1)
(3, 2)
(3, 1, 1)
(2, 3)
(2, 2, 1)
(2, 1, 2)
(2, 1, 1, 1)
(1, 4)
(1, 3, 1)
(1, 2, 2)
(1, 2, 1, 1)
(1, 1, 3)
(1, 1, 2, 1)
(1, 1, 1, 2)
(1, 1, 1, 1, 1)
```

Integer compositions are ordered integer partitions.

Return generator of positive integer tuples of length at least `1`. Changed in version 1.1.2: renamed `mathtools.integer_compositions( )` to `mathtools.yield_all_compositions_of_integer( )`.

mathtools.yield_all_partitions_of_integer

`abjad.tools.mathtools.`**`yield_all_partitions_of_integer`**`(`*n*`)`

New in version 1.1.2. Yield all partitions of positive integer *n* in descending lex order:

```
abjad> for partition in mathtools.yield_all_partitions_of_integer(7):
...     partition
...
(7,)
(6, 1)
(5, 2)
(5, 1, 1)
(4, 3)
(4, 2, 1)
(4, 1, 1, 1)
(3, 3, 1)
(3, 2, 2)
(3, 2, 1, 1)
(3, 1, 1, 1, 1)
(2, 2, 2, 1)
(2, 2, 1, 1, 1)
(2, 1, 1, 1, 1, 1)
(1, 1, 1, 1, 1, 1, 1)
```

Return generator of positive integer tuples of length at least `1`. Changed in version 1.1.2: renamed `mathtools.integer_partitions( )` to `mathtools.yield_all_partitions_of_integer( )`.

measuretools

measuretools.AnonymousMeasure

class `abjad.tools.measuretools.`**`AnonymousMeasure`**(*music=None*, ***kwargs*)

Bases: `abjad.tools.measuretools.DynamicMeasure.DynamicMeasure.DynamicMeasure`

Dynamic measure with no time signature:

```
abjad> measure = measuretools.AnonymousMeasure(macros.scale(4))
abjad> f(measure)
{
   \override Staff.TimeSignature #'stencil = ##f
   \time 1/2
   c'8
   d'8
   e'8
   f'8
   \revert Staff.TimeSignature #'stencil
}
```

```
abjad> measure.extend(macros.scale(2))
abjad> f(measure)
{
   \override Staff.TimeSignature #'stencil = ##f
   \time 3/4
   c'8
   d'8
   e'8
   f'8
   c'8
   d'8
   \revert Staff.TimeSignature #'stencil
}
```

Return anonymous measure.

measuretools.DynamicMeasure

class abjad.tools.measuretools.**DynamicMeasure**(*music=None*, ***kwargs*)

Bases: abjad.components.Measure.Measure.Measure

denominator

extend(*expr*)

suppress_meter

measuretools.append_spacer_skip_to_underfull_measure

abjad.tools.measuretools.**append_spacer_skip_to_underfull_measure**(*rigid_measure*)

New in version 1.1.1. Append spacer skip to underfull *measure*:

```
abjad> measure = Measure((4, 12), macros.scale(4))
abjad> contexttools.TimeSignatureMark(5, 12)(measure)
TimeSignatureMark(5, 12)(|5/12, c'8, d'8, e'8, f'8|)
abjad> measure.duration.is_underfull
True

abjad> measuretools.append_spacer_skip_to_underfull_measure(measure)
Measure(5/12, [c'8, d'8, e'8, f'8, s1 * 1/8])

abjad> f(measure)
{
   \time 5/12
   \scaleDurations #'(2 . 3) {
      c'8
      d'8
      e'8
      f'8
      s1 * 1/8
   }
}
```

Append nothing to nonunderfull *measure*.

Return *measure*. Changed in version 1.1.2: renamed measuretools.make_measures_with_full_measure_spacer_
) to measuretools.append_spacer_skip_to_underfull_measure( ).

measuretools.append_spacer_skips_to_underfull_measures_in_expr

abjad.tools.measuretools.**append_spacer_skips_to_underfull_measures_in_expr**(*expr*)

New in version 1.1.1. Append spacer skips to underfull measures in *expr*:

```
abjad> staff = Staff(Measure((3, 8), macros.scale(3)) * 3)
abjad> contexttools.TimeSignatureMark(4, 8)(staff[1])
TimeSignatureMark(4, 8)(|4/8, c'8, d'8, e'8|)
abjad> contexttools.TimeSignatureMark(5, 8)(staff[2])
TimeSignatureMark(5, 8)(|5/8, c'8, d'8, e'8|)
abjad> staff[1].duration.is_underfull
True
```

```
abjad> staff[2].duration.is_underfull
True

abjad> measuretools.append_spacer_skips_to_underfull_measures_in_expr(staff)
[Measure(4/8, [c'8, d'8, e'8, s1 * 1/8]), Measure(5/8, [c'8, d'8, e'8, s1 * 1/4])]

abjad> f(staff)
\new Staff {
   {
      \time 3/8
      c'8
      d'8
      e'8
   }
   {
      \time 4/8
      c'8
      d'8
      e'8
      s1 * 1/8
   }
   {
      \time 5/8
      c'8
      d'8
      e'8
      s1 * 1/4
   }
}
```

Return measures treated. Changed in version 1.1.2: renamed `measuretools.remedy_underfull_measures( )` to `measuretools.append_spacer_skips_to_underfull_measures_in_expr( )`.Changed in version 1.1.2: renamed `measuretools.append_spacer_skips_to_underfull_measures_in( )` to `measuretools.append_spacer_skips_to_underfull_measures_in_expr( )`.

measuretools.apply_beam_spanner_to_measure

`abjad.tools.measuretools.`**`apply_beam_spanner_to_measure`**`(`*measure*`)`

New in version 1.1.2. Apply beam spanner to *measure*:

```
abjad> measure = Measure((2, 8), macros.scale(2))
abjad> f(measure)
{
   \time 2/8
   c'8
   d'8
}

abjad> measuretools.apply_beam_spanner_to_measure(measure)
BeamSpanner(|2/8(2)|)

abjad> f(measure)
{
   \time 2/8
   c'8 [
   d'8 ]
}
```

Return beam spanner.

measuretools.apply_beam_spanners_to_measures_in_expr

abjad.tools.measuretools.**apply_beam_spanners_to_measures_in_expr**(*expr*)

New in version 1.1.1. Apply beam spanners to measures in *expr*:

```
abjad> staff = Staff(Measure((2, 8), notetools.make_repeated_notes(2)) * 2)
abjad> macros.diatonicize(staff)
abjad> f(staff)
\new Staff {
   {
      \time 2/8
      c'8
      d'8
   }
   {
      \time 2/8
      e'8
      f'8
   }
}
```

```
abjad> measuretools.apply_beam_spanners_to_measures_in_expr(staff)
[BeamSpanner(|2/8(2)|), BeamSpanner(|2/8(2)|)]
```

```
abjad> f(staff)
\new Staff {
   {
      \time 2/8
      c'8 [
      d'8 ]
   }
   {
      \time 2/8
      e'8 [
      f'8 ]
   }
}
```

Return list of beams created. Changed in version 1.1.2: renamed `measuretools.beam( )` to `measuretools.apply_beam_spanners_to_measures_in_expr( )`.Changed in version 1.1.2: renamed `measuretools.apply_beam_spanners_to_measures_in( )` to `measuretools.apply_beam_spanners_to_measures_in_expr( )`.

measuretools.apply_complex_beam_spanner_to_measure

abjad.tools.measuretools.**apply_complex_beam_spanner_to_measure**(*measure*)

New in version 1.1.2. Apply complex beam spanner to *measure*:

```
abjad> measure = Measure((2, 8), macros.scale(2))
abjad> f(measure)
{
   \time 2/8
   c'8
   d'8
}
```

```
abjad> measuretools.apply_complex_beam_spanner_to_measure(measure)
DuratedComplexBeamSpanner(|2/8(2)|)

abjad> f(measure)
{
   \time 2/8
   \set stemLeftBeamCount = #0
   \set stemRightBeamCount = #1
   c'8 [
   \set stemLeftBeamCount = #1
   \set stemRightBeamCount = #0
   d'8 ]
}
```

Return complex beam spanner.

measuretools.apply_complex_beam_spanners_to_measures_in_expr

abjad.tools.measuretools.**apply_complex_beam_spanners_to_measures_in_expr**(*expr*)
New in version 1.1.2. Apply complex beam spanners to measures in *expr*:

```
abjad> staff = Staff(Measure((2, 8), notetools.make_repeated_notes(2)) * 2)
abjad> macros.diatonicize(staff)
abjad> f(staff)
\new Staff {
   {
      \time 2/8
      c'8
      d'8
   }
   {
      \time 2/8
      e'8
      f'8
   }
}

abjad> measuretools.apply_complex_beam_spanners_to_measures_in_expr(staff)
[DuratedComplexBeamSpanner(|2/8(2)|), DuratedComplexBeamSpanner(|2/8(2)|)]

abjad> f(staff)
\new Staff {
   {
      \time 2/8
      \set stemLeftBeamCount = #0
      \set stemRightBeamCount = #1
      c'8 [
      \set stemLeftBeamCount = #1
      \set stemRightBeamCount = #0
      d'8 ]
   }
   {
      \time 2/8
      \set stemLeftBeamCount = #0
      \set stemRightBeamCount = #1
      e'8 [
      \set stemLeftBeamCount = #1
```

```
        \set stemRightBeamCount = #0
        f'8 ]
    }
}
```

Return list of beams created. Changed in version 1.1.2: renamed `measuretools.apply_complex_beam_spanners_to_measures_in( )` to `measuretools.apply_complex_beam_spanners_to_measures_in_expr( )`.

measuretools.apply_durated_complex_beam_spanner_to_measures

`abjad.tools.measuretools.`**`apply_durated_complex_beam_spanner_to_measures`**`(`*measures*`)`

New in version 1.1.1. Apply durated complex beam spanner to *measures*:

```
abjad> staff = Staff(Measure((2, 8), notetools.make_repeated_notes(2)) * 2)
abjad> macros.diatonicize(staff)
abjad> f(staff)
\new Staff {
    {
        \time 2/8
        c'8
        d'8
    }
    {
        \time 2/8
        e'8
        f'8
    }
}
```

```
abjad> measures = staff[:]
abjad> measuretools.apply_durated_complex_beam_spanner_to_measures(measures)
DuratedComplexBeamSpanner(|2/8(2)|, |2/8(2)|)
```

```
abjad> f(staff)
\new Staff {
    {
        \time 2/8
        \set stemLeftBeamCount = #0
        \set stemRightBeamCount = #1
        c'8 [
        \set stemLeftBeamCount = #1
        \set stemRightBeamCount = #1
        d'8
    }
    {
        \time 2/8
        \set stemLeftBeamCount = #1
        \set stemRightBeamCount = #1
        e'8
        \set stemLeftBeamCount = #1
        \set stemRightBeamCount = #0
        f'8 ]
    }
}
```

Set beam spanner durations to preprolated measure durations.

Return beam spanner created. Changed in version 1.1.2: renamed `measuretools.beam_together( )`.

measuretools.apply_full_measure_tuplets_to_contents_of_measures_in_expr

`abjad.tools.measuretools.`**`apply_full_measure_tuplets_to_contents_of_measures_in_expr`**(*expr*)

Apply full-measure tuplets to contents of measures in *expr*:

```
abjad> staff = Staff([Measure((2, 8), "c'8 d'8"), Measure((3, 8), "e'8 f'8 g'8")])

abjad> f(staff)
\new Staff {
   {
      \time 2/8
      c'8
      d'8
   }
   {
      \time 3/8
      e'8
      f'8
      g'8
   }
}

abjad> measuretools.apply_full_measure_tuplets_to_contents_of_measures_in_expr(staff)

abjad> f(staff)
\new Staff {
   {
      \time 2/8
      {
         c'8
         d'8
      }
   }
   {
      \time 3/8
      {
         e'8
         f'8
         g'8
      }
   }
}
```

Return none.

measuretools.color_measure

`abjad.tools.measuretools.`**`color_measure`**(*measure*, *color='red'*)

New in version 1.1.2. Color *measure* with *color*:

```
abjad> measure = Measure((2, 8), macros.scale(2))
abjad> f(measure)
{
   \time 2/8
   c'8
```

```
    d'8
}

abjad> measuretools.color_measure(measure, 'red')
Measure(2/8, [c'8, d'8])

abjad> f(measure)
{
  \override Beam #'color = #red
  \override Dots #'color = #red
  \override NoteHead #'color = #red
  \override Staff.TimeSignature #'color = #red
  \override Stem #'color = #red
  \time 2/8
  c'8
  d'8
  \revert Beam #'color
  \revert Dots #'color
  \revert NoteHead #'color
  \revert Staff.TimeSignature #'color
  \revert Stem #'color
}
```

Return colored *measure*.

Color names appear in LilyPond Learning Manual appendix B.5.

measuretools.color_nonbinary_measures_in_expr

abjad.tools.measuretools.**color_nonbinary_measures_in_expr**(*expr*, *color='red'*)

New in version 1.1.2. Color nonbinary measures in *expr* with *color*:

```
abjad> staff = Staff(Measure((2, 8), macros.scale(2)) * 2)
abjad> measuretools.scale_measure_denominator_and_adjust_measure_contents(staff[1], 3)
Measure(3/12, [c'8., d'8.])
abjad> f(staff)
\new Staff {
   {
      \time 2/8
      c'8
      d'8
   }
   {
      \time 3/12
      \scaleDurations #'(2 . 3) {
         c'8.
         d'8.
      }
   }
}

abjad> measuretools.color_nonbinary_measures_in_expr(staff, 'red')
[Measure(3/12, [c'8., d'8.])]

abjad> f(staff)
\new Staff {
  {
     \time 2/8
```

```
      c'8
      d'8
   }
   {
      \override Beam #'color = #red
      \override Dots #'color = #red
      \override NoteHead #'color = #red
      \override Staff.TimeSignature #'color = #red
      \override Stem #'color = #red
      \time 3/12
      \scaleDurations #'(2 . 3) {
         c'8.
         d'8.
      }
      \revert Beam #'color
      \revert Dots #'color
      \revert NoteHead #'color
      \revert Staff.TimeSignature #'color
      \revert Stem #'color
   }
}
```

Return list of measures colored.

Color names appear in LilyPond Learning Manual appendix B.5. Changed in version 1.1.2: renamed `measuretools.color_nonbinary_measures_in( )` to `measuretools.color_nonbinary_measures_in_expr( )`.

measuretools.comment_measures_in_container_with_measure_numbers

`abjad.tools.measuretools.`**`comment_measures_in_container_with_measure_numbers`**(*container*, *style='comment'*)

Label measure numbers in *container* according to *style*.

Note: functionality current not implemented.

Turn measure number labels on with `style = 'comment'`.

```
abjad> staff = Staff(Measure((2, 8), notetools.make_repeated_notes(2)) * 3)
abjad> macros.diatonicize(staff)
abjad> measuretools.comment_measures_in_container_with_measure_numbers(staff, style = 'comment')
abjad> f(staff) # doctest: +SKIP
\new Staff {
        % start measure 1
        {
                \time 2/8
                c'8
                d'8
        }
        % stop measure 1
        % start measure 2
        {
                \time 2/8
                e'8
                f'8
        }
        % stop measure 2
```

```
                % start measure 3
                {
                        \time 2/8
                        g'8
                        a'8
                }
                % stop measure 3
        }
```

Turn measure number labels off with `style = None`.

```
abjad> measuretools.comment_measures_in_container_with_measure_numbers(staff, style = None) # dc
abjad> f(staff) # doctest: +SKIP
\new Staff {
        {
                \time 2/8
                c'8
                d'8
        }
        {
                \time 2/8
                e'8
                f'8
        }
        {
                \time 2/8
                g'8
                a'8
        }
}
```

Changed in version 1.1.2: renamed `label.measure_numbers( )` to `measuretools.comment_measures_in_container_with_measure_numbers( )`.

measuretools.extend_measures_in_expr_and_apply_full_measure_tuplets_to_measure_contents

`abjad.tools.measuretools.`**`extend_measures_in_expr_and_apply_full_measure_tuplets_to_measure_`**

Extend measures in *expr* with *supplement* and apply full-measure tuplets to contents of measures:

```
abjad> staff = Staff([Measure((2, 8), "c'8 d'8"), Measure((3, 8), "e'8 f'8 g'8")])

abjad> f(staff)
\new Staff {
   {
      \time 2/8
      c'8
      d'8
   }
   {
      \time 3/8
      e'8
      f'8
      g'8
   }
}
```

```
abjad> supplement = [Rest((1, 16))]
abjad> measuretools.extend_measures_in_expr_and_apply_full_measure_tuplets_to_measure_contents(s

abjad> f(staff)
\new Staff {
   {
      \time 2/8
      \times 4/5 {
         c'8
         d'8
         r16
      }
   }
   {
      \time 3/8
      \fraction \times 6/7 {
         e'8
         f'8
         g'8
         r16
      }
   }
}
```

Return none.

measuretools.fill_measures_in_expr_with_big_endian_notes

`abjad.tools.measuretools.`**`fill_measures_in_expr_with_big_endian_notes`**(*expr*, *iterctrl=None*)

Fill measures in *expr* with big-endian notes.

measuretools.fill_measures_in_expr_with_full_measure_spacer_skips

`abjad.tools.measuretools.`**`fill_measures_in_expr_with_full_measure_spacer_skips`**(*expr*, *iterctrl=None*)

Fill measures in *expr* with full-measure spacer skips.

measuretools.fill_measures_in_expr_with_little_endian_notes

`abjad.tools.measuretools.`**`fill_measures_in_expr_with_little_endian_notes`**(*expr*, *iterctrl=None*)

Fill measures in *expr* with little-endian notes.

measuretools.fill_measures_in_expr_with_meter_denominator_notes

abjad.tools.measuretools.**fill_measures_in_expr_with_meter_denominator_notes**(*expr*, *iterctrl=None*)

Fill measures in *expr* with meter denominator notes:

```
abjad> staff = Staff([Measure((3, 4), [ ]), Measure((3, 16), [ ]), Measure((3, 8), [ ])])
abjad> measuretools.fill_measures_in_expr_with_meter_denominator_notes(staff)
```

```
\new Staff {
   {
      \time 3/4
      c'4
      c'4
      c'4
   }
   {
      \time 3/16
      c'16
      c'16
      c'16
   }
   {
      \time 3/8
      c'8
      c'8
      c'8
   }
}
```

Delete existing contents of measures in *expr*.

Return none.

measuretools.fill_measures_in_expr_with_repeated_notes

abjad.tools.measuretools.**fill_measures_in_expr_with_repeated_notes**(*expr*, *written_duration*, *iterctrl=None*)

Fill measures in *expr* with repeated notes.

measuretools.fuse_contiguous_measures_in_container_cyclically_by_counts

abjad.tools.measuretools.**fuse_contiguous_measures_in_container_cyclically_by_counts**(*container*, *counts*, *mark=Fa*

Fuse *container* measures cyclically by *counts*:

```
abjad> staff = Staff(Measure((2, 8), notetools.make_repeated_notes(2)) * 5)
abjad> macros.diatonicize(staff)
abjad> f(staff)
\new Staff {
   {
      \time 2/8
```

```
      c'8
      d'8
   }
   {
      \time 2/8
      e'8
      f'8
   }
   {
      \time 2/8
      g'8
      a'8
   }
   {
      \time 2/8
      b'8
      c''8
   }
   {
      \time 2/8
      d''8
      e''8
   }
}
```

```
abjad> counts = (2, 1)
abjad> measuretools.fuse_contiguous_measures_in_container_cyclically_by_counts(staff, counts) #

abjad> f(staff) # doctest: +SKIP
\new Staff {
   {
      \time 4/8
      c'8
      d'8
      e'8
      f'8
   }
   {
      \time 2/8
      g'8
      a'8
   }
   {
      \time 4/8
      b'8
      c''8
      d''8
      e''8
   }
}
```

Return none.

Set *mark* to true to mark fused measures for later reference. Changed in version 1.1.2: renamed `fuse.measures_by_counts_cyclic( )` to `measuretools.fuse_contiguous_measures_in_container_cyclically_by_counts( )`.

measuretools.fuse_measures

`abjad.tools.measuretools.`**`fuse_measures`**`(`*measures*`)`

Fuse *measures*:

```
abjad> staff = Staff(measuretools.make_measures_with_full_measure_spacer_skips([(1, 8), (2, 16)]
abjad> measuretools.fill_measures_in_expr_with_repeated_notes(staff, Duration(1, 16))
abjad> macros.diatonicize(staff)
abjad> spannertools.BeamSpanner(staff.leaves)
BeamSpanner(c'16, d'16, e'16, f'16)
abjad> f(staff)
\new Staff {
   {
      \time 1/8
      c'16 [
      d'16
   }
   {
      \time 2/16
      e'16
      f'16 ]
   }
}

abjad> measuretools.fuse_measures(staff[:])
Measure(2/8, [c'16, d'16, e'16, f'16])

abjad> f(staff)
\new Staff {
   {
      \time 2/8
      c'16 [
      d'16
      e'16
      f'16 ]
   }
}
```

Return new measure.

Allow parent-contiguous *measures*.

Allow outside-of-score *measures*.

Do not define measure fusion across intervening container boundaries.

Calculate best new time signature.

Instantiate new measure.

Give *measures* contents to new measure.

Give *measures* dominant spanners to new measure.

Give *measures* parentage to new measure.

Leave *measures* empty, unspanned and outside-of-score. Changed in version 1.1.2: renamed `fuse.measures_by_reference( )` to `measuretools.fuse_measures( )`.

measuretools.get_first_measure_in_improper_parentage_of_component

abjad.tools.measuretools.**get_first_measure_in_improper_parentage_of_component**(*component*)

New in version 1.1.2. Get first measure in improper parentage of *component*:

```
abjad> measure = Measure((2, 4), "c'8 d'8 e'8 f'8")
abjad> staff = Staff([measure])

abjad> f(staff)
\new Staff {
   {
      \time 2/4
      c'8
      d'8
      e'8
      f'8
   }
}

abjad> measuretools.get_first_measure_in_improper_parentage_of_component(staff.leaves[0])
Measure(2/4, [c'8, d'8, e'8, f'8])
```

Return measure or none.

measuretools.get_first_measure_in_proper_parentage_of_component

abjad.tools.measuretools.**get_first_measure_in_proper_parentage_of_component**(*component*)

New in version 1.1.2. Get first measure in proper parentage of *component*:

```
abjad> measure = Measure((2, 4), "c'8 d'8 e'8 f'8")
abjad> staff = Staff([measure])

abjad> f(staff)
\new Staff {
   {
      \time 2/4
      c'8
      d'8
      e'8
      f'8
   }
}

abjad> measuretools.get_first_measure_in_proper_parentage_of_component(staff.leaves[0])
Measure(2/4, [c'8, d'8, e'8, f'8])
```

Return measure or none.

measuretools.get_next_measure_from_component

abjad.tools.measuretools.**get_next_measure_from_component**(*component*)

New in version 1.1.1. When *component* is voice, staff or other sequential context, and when *component* contains a measure, return first measure in *component*. This starts the process of forwards measure iteration.

```
abjad> staff = Staff(Measure((2, 8), notetools.make_repeated_notes(2)) * 2)
abjad> macros.diatonicize(staff)
```

```
abjad> measuretools.get_next_measure_from_component(staff)
Measure(2/8, [c'8, d'8])
```

When *component* is voice, staff or other sequential context, and when *component* contains no measure, raise missing measure error.

When *component* is a measure and there is a measure immediately following *component*, return measure immediately following component.

```
abjad> staff = Staff(Measure((2, 8), notetools.make_repeated_notes(2)) * 2)
abjad> macros.diatonicize(staff)
abjad> measuretools.get_prev_measure_from_component(staff[0]) is None
True
```

When *component* is a measure and there is no measure immediately following *component*, return `None`.

```
abjad> staff = Staff(Measure((2, 8), notetools.make_repeated_notes(2)) * 2)
abjad> macros.diatonicize(staff)
abjad> measuretools.get_prev_measure_from_component(staff[-1])
Measure(2/8, [c'8, d'8])
```

When *component* is a leaf and there is a measure in the parentage of *component*, return the measure in the parentage of *component*.

```
abjad> staff = Staff(Measure((2, 8), notetools.make_repeated_notes(2)) * 2)
abjad> macros.diatonicize(staff)
abjad> measuretools.get_prev_measure_from_component(staff.leaves[0])
Measure(2/8, [c'8, d'8])
```

When *component* is a leaf and there is no measure in the parentage of *component*, raise missing measure error. Changed in version 1.1.2: renamed `iterate.measure_next( )` to `measuretools.get_next_measure_from_component( )`.

measuretools.get_nth_measure_in_expr

`abjad.tools.measuretools.`**`get_nth_measure_in_expr`**(*expr*, *n=0*)

New in version 1.1.2. Return measure *n* in *expr*.

```
abjad> staff = Staff(Measure((2, 8), notetools.make_repeated_notes(2)) * 3)
abjad> macros.diatonicize(staff)
abjad> f(staff)
\new Staff {
        {
                \time 2/8
                c'8
                d'8
        }
        {
                \time 2/8
                e'8
                f'8
        }
        {
                \time 2/8
                g'8
                a'8
        }
}
```

Read forward for positive values of *n*.

```
abjad> for n in range(3):
...     measuretools.get_nth_measure_in_expr(staff, n)
...
Measure(2/8, [c'8, d'8])
Measure(2/8, [e'8, f'8])
Measure(2/8, [g'8, a'8])
```

Read backward for negative values of *n*.

```
abjad> for n in range(3, -1, -1):
...     measuretools.get_nth_measure_in_expr(staff, n)
...
Measure(2/8, [g'8, a'8])
Measure(2/8, [e'8, f'8])
Measure(2/8, [c'8, d'8])
```

Todo

implement `measuretools.iterate_measures_forward_in_expr(expr, i = 0, j = None)` as a companion to this function.

Changed in version 1.1.2: renamed `iterate.get_nth_measure( )` to `measuretools.get_nth_measure_in_expr( )`.Changed in version 1.1.2: renamed `iterate.get_nth_measure_in_expr( )` to `measuretools.get_nth_measure_in_expr( )`.

measuretools.get_one_indexed_measure_number_in_expr

`abjad.tools.measuretools.`**`get_one_indexed_measure_number_in_expr`**`(`*expr*, *measure_number*`)`

New in version 1.1.2. Return *measure_number* in *expr*.

```
abjad> t = Staff(Measure((2, 8), notetools.make_repeated_notes(2)) * 3)
abjad> macros.diatonicize(t)
abjad> f(t)
\new Staff {
   {
      \time 2/8
      c'8
      d'8
   }
   {
      \time 2/8
      e'8
      f'8
   }
   {
      \time 2/8
      g'8
      a'8
   }
}
abjad> measuretools.get_one_indexed_measure_number_in_expr(t, 3)
Measure(2/8, [g'8, a'8])
```

Note: measures number from 1.

Changed in version 1.1.2: renamed `iterate.get_measure_number( )` to `measuretools.get_one_indexed_measure_number_in_expr( )`.Changed in version 1.1.2: renamed `iterate.get_measure_number_in_expr( )` to `measuretools.get_one_indexed_measure_number_in_expr( )`.

measuretools.get_prev_measure_from_component

`abjad.tools.measuretools.`**`get_prev_measure_from_component`**`(`*component*`)`

New in version 1.1.1. When *component* is voice, staff or other sequential context, and when *component* contains a measure, return last measure in *component*. This starts the process of backwards measure iteration.

```
abjad> staff = Staff(Measure((2, 8), notetools.make_repeated_notes(2)) * 2)
abjad> macros.diatonicize(staff)
abjad> measuretools.get_prev_measure_from_component(staff)
Measure(2/8, [e'8, f'8])
```

When *component* is voice, staff or other sequential context, and when *component* contains no measure, raise missing measure error.

When *component* is a measure and there is a measure immediately preceeding *component*, return measure immediately preceeding component.

```
abjad> staff = Staff(Measure((2, 8), notetools.make_repeated_notes(2)) * 2)
abjad> macros.diatonicize(staff)
abjad> measuretools.get_prev_measure_from_component(staff[-1])
Measure(2/8, [c'8, d'8])
```

When *component* is a measure and there is no measure immediately preceeding *component*, return `None`.

```
abjad> staff = Staff(Measure((2, 8), notetools.make_repeated_notes(2)) * 2)
abjad> macros.diatonicize(staff)
abjad> measuretools.get_prev_measure_from_component(staff[0]) is None
True
```

When *component* is a leaf and there is a measure in the parentage of *component*, return the measure in the parentage of *component*.

```
abjad> staff = Staff(Measure((2, 8), notetools.make_repeated_notes(2)) * 2)
abjad> macros.diatonicize(staff)
abjad> measuretools.get_prev_measure_from_component(staff.leaves[0])
Measure(2/8, [c'8, d'8])
```

When *component* is a leaf and there is no measure in the parentage of *component*, raise missing measure error. Changed in version 1.1.2: renamed `iterate.measure_prev( )` to `measuretools.get_prev_measure_from_component( )`.

measuretools.iterate_measures_backward_in_expr

`abjad.tools.measuretools.`**`iterate_measures_backward_in_expr`**`(`*expr*, *start=0*, *stop=None*`)`

New in version 1.1.2. Yield right-to-left measures in *expr*.

```
abjad> staff = Staff(Measure((2, 8), notetools.make_repeated_notes(2)) * 3)
abjad> macros.diatonicize(staff)
abjad> f(staff)
\new Staff {
        {
                \time 2/8
                c'8
                d'8
        }
        {
                \time 2/8
                e'8
                f'8
        }
        {
                \time 2/8
                g'8
                a'8
        }
}
```

```
abjad> for measure in measuretools.iterate_measures_backward_in_expr(staff):
...     measure
...
Measure(2/8, [g'8, a'8])
Measure(2/8, [e'8, f'8])
Measure(2/8, [c'8, d'8])
```

Use the optional *start* and *stop* keyword parameters to control indices of iteration.

```
abjad> for measure in measuretools.iterate_measures_backward_in_expr(staff, start = 1):
...     measure
...
Measure(2/8, [e'8, f'8])
Measure(2/8, [c'8, d'8])
```

```
abjad> for measure in measuretools.iterate_measures_backward_in_expr(staff, start = 0, stop = 2)
...     measure
...
Measure(2/8, [g'8, a'8])
Measure(2/8, [e'8, f'8])
```

Note: naive iteration ignores threads.

Changed in version 1.1.2: renamed `iterate.measures_backward_in()` to `measuretools.iterate_measures_backward_in_expr()`.Changed in version 1.1.2: renamed `iterate.measures_backward_in_expr()` to `measuretools.iterate_measures_backward_in_expr()`.

measuretools.iterate_measures_forward_in_expr

`abjad.tools.measuretools.`**`iterate_measures_forward_in_expr`**(*expr*, *start=0*, *stop=None*)

New in version 1.1.2. Yield left-to-right measures in *expr*.

```
abjad> staff = Staff(Measure((2, 8), notetools.make_repeated_notes(2)) * 3)
abjad> macros.diatonicize(staff)
abjad> f(staff)
\new Staff {
        {
                \time 2/8
                c'8
                d'8
        }
        {
                \time 2/8
                e'8
                f'8
        }
        {
                \time 2/8
                g'8
                a'8
        }
}
```

```
abjad> for measure in measuretools.iterate_measures_forward_in_expr(staff):
...     measure
...
Measure(2/8, [c'8, d'8])
Measure(2/8, [e'8, f'8])
Measure(2/8, [g'8, a'8])
```

Use the optional *start* and *stop* keyword parameters to control the start and stop indices of iteration.

```
abjad> for measure in measuretools.iterate_measures_forward_in_expr(staff, start = 1):
...     measure
...
Measure(2/8, [e'8, f'8])
Measure(2/8, [g'8, a'8])
```

```
abjad> for measure in measuretools.iterate_measures_forward_in_expr(staff, start = 0, stop = 2):
...     measure
...
Measure(2/8, [c'8, d'8])
Measure(2/8, [e'8, f'8])
```

Note: naive iteration ignores threads.

Changed in version 1.1.2: renamed `iterate.measures_forward_in()` to `measuretools.iterate_measures_forward_in_expr()`.Changed in version 1.1.2: renamed `iterate.measures_forward_in_expr()` to `measuretools.iterate_measures_forward_in_expr()`.

measuretools.make_measures_with_full_measure_spacer_skips

abjad.tools.measuretools.**make_measures_with_full_measure_spacer_skips**(*meters*)

Make rigid measures with full-measure spacer skips from *meters*:

```
abjad> measures = measuretools.make_measures_with_full_measure_spacer_skips([(1, 8), (5, 16), (5
```

```
abjad> staff = Staff(measures)
abjad> f(staff)
\new Staff {
        {
                \time 1/8
                s1 * 1/8
        }
        {
                \time 5/16
                s1 * 5/16
        }
        {
                \time 5/16
                s1 * 5/16
        }
}
```

Return list of rigid measures. Changed in version 1.1.2: renamed `measuretools.make( )` to `measuretools.make_measures_with_full_measure_spacer_skips( )`.Changed in version 1.1.2: renamed `measuretools.make_rigid_measures_with_full_measure_spacer_skips( )` to `measuretools.make_measures_with_full_measure_spacer_skips( )`.

measuretools.move_measure_prolation_to_full_measure_tuplet

`abjad.tools.measuretools.`**`move_measure_prolation_to_full_measure_tuplet`**(*expr*)

Turn nonbinary measures into binary measures containing a single fixed-duration tuplet.

This is the inverse of measuretools.move_prolation_of_full_measure_tuplet_to_meter_of_measure( ).

Note that not all nonbinary measures can be made binary.

Returns None because processes potentially many measures. Changed in version 1.1.2: renamed `measuretools.project( )` to `measuretools.move_measure_prolation_to_full_measure_tuplet( )`.

measuretools.move_prolation_of_full_measure_tuplet_to_meter_of_measure

`abjad.tools.measuretools.`**`move_prolation_of_full_measure_tuplet_to_meter_of_measure`**(*expr*)

Subsume all measures in expr containing only top-level tuplet. Measures usually become nonbinary as as result of subsumption.

Return none.

Example:

```
abjad> t = Measure((2, 8), [tuplettools.FixedDurationTuplet((2, 8), macros.scale(3))])
abjad> measuretools.move_prolation_of_full_measure_tuplet_to_meter_of_measure(t)
abjad> f(t)
{
   \time 3/12
   \scaleDurations #'(2 . 3) {
      c'8
      d'8
      e'8
   }
}
```

Changed in version 1.1.2: renamed `measuretools.subsume()` to `measuretools.move_prolation_of_full_measure_tuplet_to_meter_of_measure()`.

measuretools.multiply_contents_of_measures_in_expr

`abjad.tools.measuretools.`**`multiply_contents_of_measures_in_expr`**`(`*expr*, *n*`)`

Multiply contents `n - 1` times and adjust meter of every measure in *expr*:

```
abjad> measure = Measure((3, 8), macros.scale(3))
abjad> spannertools.BeamSpanner(measure.leaves)
BeamSpanner(c'8, d'8, e'8)
abjad> f(measure)
{
   \time 3/8
   c'8 [
   d'8
   e'8 ]
}
```

```
abjad> measuretools.multiply_contents_of_measures_in_expr(measure, 3)
```

```
abjad> f(measure)
{
   \time 9/8
   c'8 [
   d'8
   e'8 ]
   c'8 [
   d'8
   e'8 ]
   c'8 [
   d'8
   e'8 ]
}
```

Changed in version 1.1.2: renamed `measuretools.spin()` to `measuretools.multiply_contents_of_measures_in_expr()`.Changed in version 1.1.2: renamed `measuretools.multiply_measure_contents_in()` to `measuretools.multiply_contents_of_measures_in_expr()`.

measuretools.multiply_contents_of_measures_in_expr_and_scale_meter_denominators

`abjad.tools.measuretools.`**`multiply_contents_of_measures_in_expr_and_scale_meter_denominators`**

Expr may be any Abjad expression. Concentration_pairs a Python list of pairs, each of the form (spin_count, scalar_denominator). Both spin_count and scalar_denominator must be positive integers.

Iterate expr. For every measure in expr, spin measure by the spin_count element in concentration_pair and scale measure by 1/scalar_denominator element in concentration_pair.

Return Python list of transformed measures.

Example:

```
abjad> t = Measure((3, 16), notetools.make_repeated_notes(3, Duration(1, 16)))
abjad> print(measuretools.multiply_contents_of_measures_in_expr_and_scale_meter_denominators(t,
|9/48, c'32, c'32, c'32, c'32, c'32, c'32, c'32, c'32, c'32|
```

Example:

```
abjad> t = Measure((3, 16), notetools.make_repeated_notes(3, Duration(1, 16)))
abjad> print(measuretools.multiply_contents_of_measures_in_expr_and_scale_meter_denominators(t,
|9/32, c'32, c'32, c'32, c'32, c'32, c'32, c'32, c'32, c'32|
```

Example:

```
abjad> t = Measure((3, 16), notetools.make_repeated_notes(3, Duration(1, 16)))
abjad> print(measuretools.multiply_contents_of_measures_in_expr_and_scale_meter_denominators(t,
|9/16, c'16, c'16, c'16, c'16, c'16, c'16, c'16, c'16, c'16|
```

Changed in version 1.1.2: renamed `measuretools.concentrate( )` to `measuretools.multiply_contents_of_measures_in_expr_and_scale_meter_denominators( )`.Changed in version 1.1.2: renamed `measuretools.multiply_measure_contents_and_scale_meter_denomi` `)` to `measuretools.multiply_contents_of_measures_in_expr_and_scale_meter_denominators( )`.

measuretools.pad_measures_in_expr_with_rests

`abjad.tools.measuretools.`**`pad_measures_in_expr_with_rests`**`(`*expr*, *front*, *back*, *splice=False*`)`

New in version 1.1.1. Iterate all measures in *expr*. Insert rest with duration equal to *front* at beginning of each measure. Insert rest with duation aqual to *back* at end of each measure.

Set *front* to a positive rational or none. Set *back* to a positive rational or none. Return none.

Note: This function is designed to help create regularly spaced charts and tables of musical materials. This function makes most sense when used on `AnonymousMeasure` and `DynamicMeasure` instances.

```
abjad> t = Staff(measuretools.AnonymousMeasure(macros.scale(2)) * 2)
abjad> front, back = Duration(1, 32), Duration(1, 64)
abjad> measuretools.pad_measures_in_expr_with_rests(t, front, back) # doctest: +SKIP
abjad> f(t) # doctest: +SKIP
\new Staff {
                \override Staff.TimeSignature #'stencil = ##f
                \time 19/64
                r32
                c'8
                d'8
                r64
                \revert Staff.TimeSignature #'stencil
                \override Staff.TimeSignature #'stencil = ##f
                \time 19/64
                r32
                c'8
                d'8
                r64
                \revert Staff.TimeSignature #'stencil
}
```

Works when measures contain stacked voices:

```
abjad> measure = measuretools.DynamicMeasure(Voice(notetools.make_repeated_notes(2)) * 2)
abjad> measure.is_parallel = True
abjad> t = Staff(measure * 2)
abjad> macros.diatonicize(t)
abjad> measuretools.pad_measures_in_expr_with_rests(t, Duration(1, 32), Duration(1, 64)) # docte
```

```
abjad> f(t) # doctest: +SKIP
\new Staff {
      \time 19/64
      \new Voice {
         r32
         c'8
         d'8
         r64
      }
      \new Voice {
         r32
         e'8
         f'8
         r64
      }
      \time 19/64
      \new Voice {
         r32
         g'8
         a'8
         r64
      }
      \new Voice {
         r32
         b'8
         c''8
         r64
      }
}
```

Set the optional *splice* keyword to `True` to extend edge spanners over newly inserted rests.

```
abjad> t = measuretools.DynamicMeasure(macros.scale(2))
abjad> spannertools.BeamSpanner(t[:])
BeamSpanner(c'8, d'8)
abjad> t.formatter.number.self = 'comment' # doctest: +SKIP
abjad> measuretools.pad_measures_in_expr_with_rests(t, Duration(1, 32), Duration(1, 64), splice
```

```
abjad> f(t) # doctest: +SKIP
% start measure 1
   \time 19/64
   r32 [
   c'8
   d'8
   r64 ]
% stop measure 1
```

Raise value when *front* is neither a positive rational nor none.

Raise value when *back* is neither a positive rational nor none. Changed in version 1.1.2: renamed `layout.insert_measure_padding_rest( )` to `measuretools.pad_measures_in_expr_with_rests( )`.

measuretools.pad_measures_in_expr_with_skips

`abjad.tools.measuretools.`**`pad_measures_in_expr_with_skips`**`(`*expr*, *front*, *back*, *splice=False*`)`

New in version 1.1.2. Iterate all measures in *expr*. Insert skip with duration equal to *front* at beginning of each measure. Insert skip with duation aqual to *back* at end of each measure.

Set *front* to a positive rational or none. Set *back* to a positive rational or none. Return none.

Note: This function is designed to help create regularly spaced charts and tables of musical materials. This function makes most sense when used on `AnonymousMeasure` and `DynamicMeasure` instances.

```
abjad> t = Staff(measuretools.AnonymousMeasure(macros.scale(2)) * 2)
abjad> front, back = Duration(1, 32), Duration(1, 64)
abjad> measuretools.pad_measures_in_expr_with_skips(t, front, back) # doctest: +SKIP
abjad> f(t) # doctest: +SKIP
\new Staff {
                \override Staff.TimeSignature #'stencil = ##f
                \time 19/64
                s32
                c'8
                d'8
                s64
                \revert Staff.TimeSignature #'stencil
                \override Staff.TimeSignature #'stencil = ##f
                \time 19/64
                s32
                c'8
                d'8
                s64
                \revert Staff.TimeSignature #'stencil
}
```

Works when measures contain stacked voices.

```
abjad> measure = measuretools.DynamicMeasure(Voice(notetools.make_repeated_notes(2)) * 2)
abjad> measure.is_parallel = True
abjad> t = Staff(measure * 2)
abjad> macros.diatonicize(t)
abjad> measuretools.pad_measures_in_expr_with_skips(t, Duration(1, 32), Duration(1, 64)) # docte

abjad> f(t) # doctest: +SKIP
\new Staff {
      \time 19/64
      \new Voice {
         s32
         c'8
         d'8
         s64
      }
      \new Voice {
         s32
         e'8
         f'8
         s64
      }
      \time 19/64
      \new Voice {
```

```
            s32
            g'8
            a'8
            s64
         }
         \new Voice {
            s32
            b'8
            c''8
            s64
         }
}
```

Set the optional *splice* keyword to `True` to extend edge spanners over newly inserted skips.

```
abjad> t = measuretools.DynamicMeasure(macros.scale(2))
abjad> spannertools.BeamSpanner(t[:])
BeamSpanner(c'8, d'8)
abjad> t.formatter.number.self = 'comment' # doctest: +SKIP
abjad> measuretools.pad_measures_in_expr_with_skips(t, Duration(1, 32), Duration(1, 64), splice

abjad> f(t) # doctest: +SKIP
% start measure 1
   \time 19/64
   s32 [
   c'8
   d'8
   s64 ]
% stop measure 1
```

Raise value error when *front* is neither a positive rational nor none.

Raise value error when *back* is neither a positive rational nor none. Changed in version 1.1.2: renamed `layout.insert_measure_padding_skip( )` to `measuretools.pad_measures_in_expr_with_skips( )`.

measuretools.pitch_array_row_to_measure

`abjad.tools.measuretools.`**`pitch_array_row_to_measure`**(*pitch_array_row*, *cell_duration_denominator=8*)

New in version 1.1.2. Change *pitch_array_row* to measure with meter *pitch_array_row.width* over *cell_duration_denominator*.

```
abjad> from abjad.tools import pitcharraytools
abjad> array = pitcharraytools.PitchArray([
...     [1, (2, 1), ([-2, -1.5], 2)],
...     [(7, 2), (6, 1), 1]])
abjad> print array
[  ] [d'] [bf bqf    ]
[g'     ] [fs'   ] [ ]

abjad> measure = measuretools.pitch_array_row_to_measure(array.rows[0])
abjad> f(measure)
{
        \time 4/8
        r8
        d'8
```

```
        <bf bqf>4
}
```

Return measure.

measuretools.pitch_array_to_measures

abjad.tools.measuretools.**pitch_array_to_measures**(*pitch_array*, *cell_duration_denominator=8*)

New in version 1.1.2. Change *pitch_array* to measures with meters *row.width* over *cell_duration_denominator* for each row in *pitch_array*.

```
abjad> from abjad.tools import pitcharraytools
abjad> array = pitcharraytools.PitchArray([
...     [1, (2, 1), ([-2, -1.5], 2)],
...     [(7, 2), (6, 1), 1]])
abjad> print array
[  ] [d'] [bf bqf    ]
[g'     ] [fs'   ] [ ]

abjad> measuretools.pitch_array_to_measures(array)
[Measure(4/8, [r8, d'8, <bf bqf>4]), Measure(4/8, [g'4, fs'8, r8])]
abjad> for measure in _:
...     f(measure)
...
{
        \time 4/8
        r8
        d'8
        <bf bqf>4
}
{
        \time 4/8
        g'4
        fs'8
        r8
}
```

Return list of measures.

measuretools.replace_contents_of_measures_in_expr

abjad.tools.measuretools.**replace_contents_of_measures_in_expr**(*expr*, *new_contents*)

New in version 1.1.1. Replace contents of measures in *expr* with *new_contents*:

```
abjad> staff = Staff(measuretools.make_measures_with_full_measure_spacer_skips([(1, 8), (3, 16)]
abjad> f(staff)
\new Staff {
   {
      \time 1/8
      s1 * 1/8
   }
   {
      \time 3/16
      s1 * 3/16
```

```
        }
    }

    abjad> notes = macros.scale(4, Duration(1, 16))
    abjad> measuretools.replace_contents_of_measures_in_expr(staff, notes)
    [Measure(1/8, [c'16, d'16]), Measure(3/16, [e'16, f'16, s1 * 1/16])]

    abjad> f(staff)
    \new Staff {
        {
            \time 1/8
            c'16
            d'16
        }
        {
            \time 3/16
            e'16
            f'16
            s1 * 1/16
        }
    }
```

Preserve duration of all measures.

Skip measures that are too small.

Pad extra space at end of measures with spacer skip.

If not enough measures raise stop iteration.

Return measures iterated. Changed in version 1.1.2: renamed `measuretools.overwrite_contents( )` to `measuretools.replace_contents_of_measures_in_expr( )`.

measuretools.report_meter_distribution_as_string

`abjad.tools.measuretools.`**`report_meter_distribution_as_string`**`(`*expr*`)`

Report meter distribution of *expr* as string.

```
abjad> measuretools.report_meter_distribution_as_string(t) # doctest: +SKIP
'\t3/80\t2\n\t2/16\t73\n\t7/40\t1\n\t3/16\t20\n\t16/80\t1\n\t17/80\t1\n
\t19/80\t1\n\t4/16\t73\n\t5/16\t62\n\t13/40\t1\n\t27/80\t1\n\t6/16\t12\
n\t7/16\t16\n\t8/16\t13\n\t9/16\t15\n\t10/16\t4\n'
```

Return string.

measuretools.report_meter_distribution_to_screen

`abjad.tools.measuretools.`**`report_meter_distribution_to_screen`**`(`*expr*`)`

Report meter distribution of *expr* to screen.

```
abjad> measuretools.report_meter_distribution_to_screen(t) # doctest: +SKIP
  2/16    62
  3/16    14
  4/16    66
  5/16    57
  6/16    17
  7/16    20
  8/16    16
```

```
9/16    19
10/16   4
```

Return none.

measuretools.scale_contents_of_measures_in_expr

`abjad.tools.measuretools.`**`scale_contents_of_measures_in_expr`**`(`*expr*, *multiplier=1*`)`

Iterate expr. For every measure in expr:

1. multiply measure's meter by multiplier
2. scale measure's contents to fit new meter

Extends containertools.scale_contents_of_container( ). Returns None because iterates possibly many measures.

This might best be a bound method on Measure. Changed in version 1.1.2: renamed `measuretools.scale( )` to `measuretools.scale_contents_of_measures_in_expr( )`.Changed in version 1.1.2: renamed `measuretools.scale_measure_contents_in( )` to `measuretools.scale_contents_of_measures_in_expr( )`.

measuretools.scale_measure_by_multiplier_and_adjust_meter

`abjad.tools.measuretools.`**`scale_measure_by_multiplier_and_adjust_meter`**`(`*measure*, *multiplier=1*`)`

Multiply the duration of every element in measure by multiplier. Then rewrite the meter of measure as appropriate.

Return treated measure.

Like magic.

Example:

```
abjad> t = Measure((3, 8), macros.scale(3))
abjad> measuretools.scale_measure_by_multiplier_and_adjust_meter(t, Duration(2, 3))
Measure(3/12, [c'8, d'8, e'8])
abjad> f(t)
{
     \time 3/12
     \scaleDurations #'(2 . 3) {
             c'8
             d'8
             e'8
     }
}
```

Changed in version 1.1.2: renamed `measuretools.scale_and_remeter( )` to `measuretools.scale_measure_by_multiplier_and_adjust_meter( )`.

measuretools.scale_measure_denominator_and_adjust_measure_contents

`abjad.tools.measuretools.`**`scale_measure_denominator_and_adjust_measure_contents`**`(`*measure*, *new_denominato*

New in version 1.1.1. Change binary *measure* to nonbinary measure with *new_denominator_factor*:

```
abjad> measure = Measure((2, 8), macros.scale(2))
abjad> spannertools.BeamSpanner(measure.leaves)
BeamSpanner(c'8, d'8)
abjad> f(measure)
{
   \time 2/8
   c'8 [
   d'8 ]
}

abjad> measuretools.scale_measure_denominator_and_adjust_measure_contents(measure, 3)
Measure(3/12, [c'8., d'8.])

abjad> f(measure)
{
   \time 3/12
   \scaleDurations #'(2 . 3) {
      c'8. [
      d'8. ]
   }
}
```

Treat *new_denominator_factor* like clever form of `1`: `3/3` or `5/5` or `7/7`, etc.

Preserve *measure* prolated duration.

Derive new *measure* multiplier.

Scale *measure* contents.

Pick best new meter.

Todo

implement `measuretools.change_nonbinary_measure_to_binary( )`.

Changed in version 1.1.2: renamed `measuretools.change_binary_measure_to_nonbinary( )` to `measuretools.scale_measure_denominator_and_adjust_measure_contents( )`.

measuretools.set_measure_denominator_and_adjust_numerator

`abjad.tools.measuretools.`**`set_measure_denominator_and_adjust_numerator`**(*measure*, *denominator*)

Set *measure* meter *denominator* and multiply meter numerator accordingly:

```
abjad> measure = Measure((3, 8), macros.scale(3))
abjad> spannertools.BeamSpanner(measure.leaves)
BeamSpanner(c'8, d'8, e'8)
abjad> f(measure)
{
   \time 3/8
   c'8 [
   d'8
   e'8 ]
}
```

```
abjad> measuretools.set_measure_denominator_and_adjust_numerator(measure, 16)
Measure(6/16, [c'8, d'8, e'8])

abjad> f(measure)
{
   \time 6/16
   c'8 [
   d'8
   e'8 ]
}
```

Leave *measure* contents unchanged.

Return *measure*.

Todo

implement `measuretools.set_measure_denominator_and_adjust_contents( )`.

Changed in version 1.1.2: renamed `measuretools.set_measure_denominator_and_multiply_numerator( )` to `measuretools.set_measure_denominator_and_adjust_numerator( )`.

metertools

metertools.Meter

class `abjad.tools.metertools.`**`Meter`**`(*args, **kwargs)`

Bases: `abjad.core._StrictComparator._StrictComparator._StrictComparator`, `abjad.core._Immutable._Immutable._Immutable`

DEPRECATED.

Use `TimeSignatureMark` instead.

Abjad model of time signature:

```
abjad> metertools.Meter((5, 32))
Meter(5, 32)
```

return meter.

denominator
Integer denominator of meter.

duration
Duration duration of meter.

format
LilyPond input format of meter.

is_nonbinary
Boolean indicator of nonbinary meter.

multiplier
Duration prolation multiplier of meter.

numerator
Integer numerator of meter.

partial
Duration partial-measure pickup prior to meter.

metertools.duration_and_possible_denominators_to_meter

abjad.tools.metertools.**duration_and_possible_denominators_to_meter**(*duration, denominators=None, factor=None*)

Make new meter equal to *duration*:

```
abjad> metertools.duration_and_possible_denominators_to_meter(Duration(3, 2))
Meter(3, 2)
```

Make new meter equal to *duration* with denominator equal to the first possible element in *denominators*:

```
abjad> metertools.duration_and_possible_denominators_to_meter(Duration(3, 2), denominators = [5,
Meter(9, 6)
```

Make new meter equal to *duration* with denominator divisible by *factor*:

```
abjad> metertools.duration_and_possible_denominators_to_meter(Duration(3, 2), factor = 5)
Meter(15, 10)
```

Return new meter. Changed in version 1.1.2: renamed `metertools.make_best( )` to `metertools.duration_and_possible_denominators_to_meter( )`.

metertools.get_nonbinary_factor_from_meter_denominator

abjad.tools.metertools.**get_nonbinary_factor_from_meter_denominator**(*meter*)

Get nonbinary factor from nonbinary *meter* denominator:

```
abjad> metertools.get_nonbinary_factor_from_meter_denominator(metertools.Meter(3, 12))
3

abjad> metertools.get_nonbinary_factor_from_meter_denominator(metertools.Meter(3, 13))
13

abjad> metertools.get_nonbinary_factor_from_meter_denominator(metertools.Meter(3, 14))
7

abjad> metertools.get_nonbinary_factor_from_meter_denominator(metertools.Meter(3, 15))
15
```

Get `1` from binary *meter* denominator:

```
abjad> metertools.get_nonbinary_factor_from_meter_denominator(metertools.Meter(3, 16))
1
```

Return nonnegative integer.

metertools.is_meter_token

abjad.tools.metertools.**is_meter_token**(*expr*)

True when *expr* has the form of an Abjad meter token:

```
abjad> metertools.is_meter_token(metertools.Meter(3, 8))
True

abjad> metertools.is_meter_token(Duration(3, 8))
True

abjad> metertools.is_meter_token((3, 8))
True
```

Otherwise false:

```
abjad> metertools.is_meter_token('text')
False
```

Return boolean.

metertools.is_meter_with_equivalent_binary_representation

abjad.tools.metertools.**is_meter_with_equivalent_binary_representation**(*expr*)

True when *expr* is a meter with binary-valued duration:

```
abjad> metertools.is_meter_with_equivalent_binary_representation(metertools.Meter(3, 12))
True
```

Otherwise false:

```
abjad> metertools.is_meter_with_equivalent_binary_representation(metertools.Meter(4, 12))
False

abjad> metertools.is_meter_with_equivalent_binary_representation('text')
False
```

Return boolean.

metertools.list_meters_of_measures_in_expr

abjad.tools.metertools.**list_meters_of_measures_in_expr**(*components*)

List meters of measures in *expr*:

```
abjad> staff = Staff([Measure((2, 8), "c8 d8"), Measure((3, 8), "c8 d8 e8"), Measure((4, 8), "c8

abjad> f(staff)
\new Staff {
   {
      \time 2/8
      c8
      d8
   }
   {
      \time 3/8
      c8
      d8
      e8
   }
   {
      \time 4/8
      c8
```

```
        d8
        e8
        f8
    }
}
```

```
abjad> metertools.list_meters_of_measures_in_expr(staff)
[TimeSignatureMark(2, 8)(|2/8, c8, d8|), TimeSignatureMark(3, 8)(|3/8, c8, d8, e8|), TimeSignatu
```

Return list of zero or more time signatures. Changed in version 1.1.2: renamed `metertools.extract_meter_list( )` to `metertools.list_meters_of_measures_in_expr( )`.Changed in version 1.1.2: now returns list of meters instead of list of integer pairs.

metertools.meter_to_binary_meter

`abjad.tools.metertools.`**`meter_to_binary_meter`**(*nonbinary_meter*, *contents_multiplier=Fraction(1, 1)*)

Change nonbinary *meter* to binary meter:

```
abjad> metertools.meter_to_binary_meter(metertools.Meter(3, 12))
Meter(2, 8)
```

Preserve binary *meter*:

```
abjad> metertools.meter_to_binary_meter(metertools.Meter(2, 8))
Meter(2, 8)
```

Return newly constructed meter. Changed in version 1.1.2: renamed `metertools.make_binary( )` to `metertools.meter_to_binary_meter( )`.

notetools

notetools.NaturalHarmonic

class `abjad.tools.notetools.`**`NaturalHarmonic`**(**args*)

Bases: `abjad.components.Note.Note.Note`, `abjad.tools.notetools._Harmonic._Harmonic._Harmoni`

Abjad model of natural harmonic.

Initialize natural harmonic by hand:

```
abjad> notetools.NaturalHarmonic("cs'8.")
NaturalHarmonic(cs', 8.)
```

Initialize natural harmonic from note:

```
abjad> note = Note("cs'8.")
```

```
abjad> notetools.NaturalHarmonic(note)
NaturalHarmonic(cs', 8.)
```

Natural harmonics are immutable.

notetools.NoteHead

class `abjad.tools.notetools.`**`NoteHead`**(**args*)

Bases: `abjad.core._UnaryComparator._UnaryComparator._UnaryComparator`

Abjad model of a note head:

```
abjad> notetools.NoteHead(13)
NoteHead("cs''")
```

Note heads are immutable.

`format`

Read-only LilyPond input format of note head:

```
abjad> note_head = notetools.NoteHead("cs''")
abjad> note_head.format
"cs''"
```

Return string.

`named_chromatic_pitch`

Read-only named chromatic pitch equal to note head:

```
abjad> note_head = notetools.NoteHead("cs''")
abjad> note_head.named_chromatic_pitch
NamedChromaticPitch("cs''")
```

Return named chromatic pitch.

`pitch`

Get named pitch of note head:

```
abjad> note_head = notetools.NoteHead("cs''")
abjad> note_head.pitch
NamedChromaticPitch("cs''")
```

Set named pitch of note head:

```
abjad> note_head = notetools.NoteHead("cs''")
abjad> note_head.pitch = "d''"
abjad> note_head.pitch
NamedChromaticPitch("d''")
```

Set pitch token.

`tweak`

Read-only LilyPond tweak reservoir:

```
abjad> note_head = notetools.NoteHead("cs''")
abjad> note_head.tweak
LilyPondTweakReservoir( )
```

Return LilyPond tweak reservoir.

notetools.add_artificial_harmonic_to_note

`abjad.tools.notetools.`**`add_artificial_harmonic_to_note`**(*note, melodic_diatonic_interval=MelodicDiatonicInterval('*

Add artifical harmonic to *note* at *melodic_diatonic_interval*:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> spannertools.BeamSpanner(staff[:])
BeamSpanner(c'8, d'8, e'8, f'8)

abjad> f(staff)
\new Staff {
   c'8 [
   d'8
   e'8
   f'8 ]
}

abjad> notetools.add_artificial_harmonic_to_note(staff[0])
Chord("<c' f'>8")

abjad> f(staff)
\new Staff {
   <
      c'
      \tweak #'style #'harmonic
      f'
   >8 [
   d'8
   e'8
   f'8 ]
}
```

Create new artificial harmonic chord from *note*.

Move parentage and spanners from *note* to artificial harmonic chord.

Return artificial harmonic chord. Changed in version 1.1.2: renamed `harmonictools.add_artificial( )` to `notetools.add_artificial_harmonic_to_note( )`.

notetools.color_note_head_by_numbered_chromatic_pitch_class_color_map

abjad.tools.notetools.**color_note_head_by_numbered_chromatic_pitch_class_color_map**(*pitch_carrie*
Color *pitch_carrier* note head:

```
abjad> note = Note("c'4")

abjad> notetools.color_note_head_by_numbered_chromatic_pitch_class_color_map(note)
Note("c'4")

abjad> f(note)
\once \override NoteHead #'color = #(x11-color 'red)
c'4
```

Numbered chromatic pitch-class color map:

```
0: red
1: MediumBlue
2: orange
3: LightSlateBlue
4: ForestGreen
5: MediumOrchid
6: firebrick
```

```
7: DeepPink
8: DarkOrange
9: IndianRed
10: CadetBlue
11: SeaGreen
12: LimeGreen
```

Numbered chromatic pitch-class color map can not be changed.

Raise type error when *pitch_carrier* is not a pitch carrier.

Raise extra pitch error when *pitch_carrier* carries more than 1 note head.

Raise missing pitch error when *pitch_carrier* carries no note head.

Return *pitch_carrier*. Changed in version 1.1.2: renamed `pitchtools.color_by_pc( )` to `notetools.color_note_head_by_numbered_chromatic_pitch_class_color_map( )`.Changed in version 1.1.2: renamed `notetools.color_note_head_by_numeric_chromatic_pitch_class_col` `)` to `notetools.color_note_head_by_numbered_chromatic_pitch_class_color_map( )`.

notetools.iterate_notes_backward_in_expr

`abjad.tools.notetools.`**`iterate_notes_backward_in_expr`**`(`*expr*, *start=0*, *stop=None*`)`

New in version 1.1.2. Yield right-to-left notes in *expr*:

```
abjad> staff = Staff(Measure((2, 8), notetools.make_repeated_notes(2)) * 3)
abjad> macros.diatonicize(staff)
abjad> f(staff)
\new Staff {
        {
                \time 2/8
                c'8
                d'8
        }
        {
                \time 2/8
                e'8
                f'8
        }
        {
                \time 2/8
                g'8
                a'8
        }
}
```

```
abjad> for leaf in notetools.iterate_notes_backward_in_expr(staff):
...     leaf
...
Note("a'8")
Note("g'8")
Note("f'8")
Note("e'8")
Note("d'8")
Note("c'8")
```

Use optional *start* and *stop* keyword parameters to control indices of iteration:

```
abjad> for leaf in notetools.iterate_notes_backward_in_expr(staff, start = 3):
...     leaf
...
Note("e'8")
Note("d'8")
Note("c'8")

abjad> for leaf in notetools.iterate_notes_backward_in_expr(staff, start = 0, stop = 3):
...     leaf
...
Note("a'8")
Note("g'8")
Note("f'8")

abjad> for leaf in notetools.iterate_notes_backward_in_expr(staff, start = 2, stop = 4):
...     leaf
...
Note("f'8")
Note("e'8")
```

Return note generator. Changed in version 1.1.2: renamed `iterate.notes_backward_in( )` to `notetools.iterate_notes_backward_in_expr( )`.

notetools.iterate_notes_forward_in_expr

abjad.tools.notetools.**iterate_notes_forward_in_expr**(*expr*, *start=0*, *stop=None*)

New in version 1.1.2. Yield left-to-right notes in *expr*:

```
abjad> staff = Staff(Measure((2, 8), notetools.make_repeated_notes(2)) * 3)
abjad> macros.diatonicize(staff)
abjad> f(staff)
\new Staff {
        {
                \time 2/8
                c'8
                d'8
        }
        {
                \time 2/8
                e'8
                f'8
        }
        {
                \time 2/8
                g'8
                a'8
        }
}

abjad> for leaf in notetools.iterate_notes_forward_in_expr(staff):
...     leaf
...
Note("c'8")
Note("d'8")
Note("e'8")
Note("f'8")
```

```
Note("g'8")
Note("a'8")
```

Use optional *start* and *stop* keyword parameters to control start and stop indices of iteration:

```
abjad> for leaf in notetools.iterate_notes_forward_in_expr(staff, start = 3):
...     leaf
...
Note("f'8")
Note("g'8")
Note("a'8")

abjad> for leaf in notetools.iterate_notes_forward_in_expr(staff, start = 0, stop = 3):
...     leaf
...
Note("c'8")
Note("d'8")
Note("e'8")

abjad> for leaf in notetools.iterate_notes_forward_in_expr(staff, start = 2, stop = 4):
...     leaf
...
Note("e'8")
Note("f'8")
```

Return generator. Changed in version 1.1.2: renamed `iterate.notes_forward_in( )` to `notetools.iterate_notes_forward_in_expr( )`.

notetools.label_notes_in_expr_with_note_indices

`abjad.tools.notetools.`**`label_notes_in_expr_with_note_indices`**(*expr*, *markup_direction='down'*)

New in version 1.1.2. Label notes in *expr* with note indices:

```
abjad> staff = Staff("c'8 d'8 r8 r8 g'8 a'8 r8 c''8")

abjad> notetools.label_notes_in_expr_with_note_indices(staff)

abjad> f(staff)
\new Staff {
   c'8 _ \markup { \small 0 }
   d'8 _ \markup { \small 1 }
   r8
   r8
   g'8 _ \markup { \small 2 }
   a'8 _ \markup { \small 3 }
   r8
   c''8 _ \markup { \small 4 }
}
```

Return none.

notetools.make_accelerating_notes_with_lilypond_multipliers

abjad.tools.notetools.**make_accelerating_notes_with_lilypond_multipliers**(*pitches*, *total*, *start*, *stop*, *exp='cosine'*, *written=Duration(1, 8)*)

Make accelerating notes with LilyPond multipliers:

```
abjad> notetools.make_accelerating_notes_with_lilypond_multipliers([1,2], (1, 2), (1, 4), (1, 8)
[Note("cs'8 * 113/64"), Note("d'8 * 169/128"), Note("cs'8 * 117/128")]
```

```
abjad> voice = Voice(_)
abjad> voice.duration.prolated
Duration(1, 2)
```

Set note pitches cyclically from *pitches*.

Return as many interpolation values as necessary to fill the *total* duration requested.

Interpolate durations from *start* to *stop*.

Set note durations to *written* duration times computed interpolated multipliers.

Return list of notes. Changed in version 1.1.2: renamed `construct.notes_curve( )` to `notetools.make_accelerating_notes_with_lilypond_multipliers( )`.

notetools.make_notes

abjad.tools.notetools.**make_notes**(*pitches*, *durations*, *direction='big-endian'*)

Make notes according to *pitches* and *durations*.

Cycle through *pitches* when the length of *pitches* is less than the length of *durations*:

```
abjad> notetools.make_notes([0], [(1, 16), (1, 8), (1, 8)])
[Note("c'16"), Note("c'8"), Note("c'8")]
```

Cycle through *durations* when the length of *durations* is less than the length of *pitches*:

```
abjad> notetools.make_notes([0, 2, 4, 5, 7], [(1, 16), (1, 8), (1, 8)])
[Note("c'16"), Note("d'8"), Note("e'8"), Note("f'16"), Note("g'8")]
```

Create ad hoc tuplets for nonassignable durations:

```
abjad> notetools.make_notes([0], [(1, 16), (1, 12), (1, 8)])
[Note("c'16"), Tuplet(2/3, [c'8]), Note("c'8")]
```

Set *direction* to `'big-endian'` to express tied values in decreasing duration:

```
abjad> notetools.make_notes([0], [(13, 16)], direction = 'big-endian')
[Note("c'2."), Note("c'16")]
```

Set *direction* to `'little-endian'` to express tied values in increasing duration:

```
abjad> notetools.make_notes([0], [(13, 16)], direction = 'little-endian')
[Note("c'16"), Note("c'2.")]
```

Set *pitches* to a single pitch or a sequence of pitches.

Set *durations* to a single duration or a list of durations.

Return list of newly constructed notes. Changed in version 1.1.2: renamed `construct.notes( )` to `notetools.make_notes( )`.

notetools.make_notes_with_multiplied_durations

`abjad.tools.notetools.`**`make_notes_with_multiplied_durations`**(*pitch*, *written_duration*, *multiplied_durations*)

New in version 1.1.2. Make *written_duration* notes with *pitch* and *multiplied_durations*:

```
abjad> notetools.make_notes_with_multiplied_durations(0, Duration(1, 4), [(1, 2), (1, 3), (1, 4)
[Note("c'4 * 2"), Note("c'4 * 4/3"), Note("c'4 * 1"), Note("c'4 * 4/5")]
```

Useful for making spatially positioned notes.

Return list of notes.

notetools.make_percussion_note

`abjad.tools.notetools.`**`make_percussion_note`**(*pitch*, *total_duration*, *max_note_duration=(1, 8)*)

Make percussion note:

```
abjad> notetools.make_percussion_note(2, (1, 4), (1, 8))
[Note("d'8"), Rest('r8')]

abjad> notetools.make_percussion_note(2, (1, 64), (1, 8))
[Note("d'64")]

abjad> notetools.make_percussion_note(2, (5, 64), (1, 8))
[Note("d'16"), Rest('r64')]

abjad> notetools.make_percussion_note(2, (5, 4), (1, 8))
[Note("d'8"), Rest('r1'), Rest('r8')]
```

Return list of newly constructed note followed by zero or more newly constructed rests.

Durations of note and rests returned will sum to *total_duration*.

Duration of note returned will be no greater than *max_note_duration*.

Duration of rests returned will sum to note duration taken from *total_duration*.

Useful for percussion music where attack duration is negligible and tied notes undesirable. Changed in version 1.1.2: renamed `construct.percussion_note( )` to `notetools.make_percussion_note( )`.

notetools.make_quarter_notes_with_lilypond_multipliers

`abjad.tools.notetools.`**`make_quarter_notes_with_lilypond_multipliers`**(*pitches*, *multiplied_durations*)

New in version 1.1.2. Make quarter notes with *pitches* and *multiplied_durations*:

```
abjad> notetools.make_quarter_notes_with_lilypond_multipliers([0, 2, 4, 5], [(1, 4), (1, 5), (1,
[Note("c'4 * 1"), Note("d'4 * 4/5"), Note("e'4 * 2/3"), Note("f'4 * 4/7")]
```

Read *pitches* cyclically where the length of *pitches* is less than the length of *multiplied_durations*:

```
abjad> notetools.make_quarter_notes_with_lilypond_multipliers([0], [(1, 4), (1, 5), (1, 6), (1,
[Note("c'4 * 1"), Note("c'4 * 4/5"), Note("c'4 * 2/3"), Note("c'4 * 4/7")]
```

Read *multiplied_durations* cyclically where the length of *multiplied_durations* is less than the length of *pitches*:

```
abjad> notetools.make_quarter_notes_with_lilypond_multipliers([0, 2, 4, 5], [(1, 5)])
[Note("c'4 * 4/5"), Note("d'4 * 4/5"), Note("e'4 * 4/5"), Note("f'4 * 4/5")]
```

Return list of zero or more newly constructed notes. Changed in version 1.1.2: renamed `construct.quarter_notes_with_multipliers( )` to `notetools.make_quarter_notes_with_lilypond_multipliers( )`.

notetools.make_repeated_notes

`abjad.tools.notetools.`**`make_repeated_notes`**(*count*, *duration=Duration(1, 8)*)

Make *count* repeated notes with note head-assignable *duration*:

```
abjad> notetools.make_repeated_notes(4)
[Note("c'8"), Note("c'8"), Note("c'8"), Note("c'8")]
```

Make *count* repeated tie chains with tied *duration*:

```
abjad> notes = notetools.make_repeated_notes(2, (5, 16))
abjad> voice = Voice(notes)

abjad> f(voice)
\new Voice {
   c'4 ~
   c'16
   c'4 ~
   c'16
}
```

Make ad hoc tuplet holding *count* repeated notes with nonbinary *duration*:

```
abjad> notetools.make_repeated_notes(3, (1, 12))
[Tuplet(2/3, [c'8, c'8, c'8])]
```

Set pitch of all notes created to middle C.

Return list of zero or more newly constructed notes or list of one newly constructed tuplet. Changed in version 1.1.2: renamed `construct.run( )` to `notetools.make_repeated_notes( )`.

notetools.make_repeated_notes_from_time_signature

`abjad.tools.notetools.`**`make_repeated_notes_from_time_signature`**(*time_signature*, *pitch="c"'*)

New in version 1.1.2. Make repeated notes from *time_signature*:

```
abjad> notetools.make_repeated_notes_from_time_signature((5, 32))
[Note("c'32"), Note("c'32"), Note("c'32"), Note("c'32"), Note("c'32")]
```

Make repeated notes with *pitch* from *time_signature*:

```
abjad> notetools.make_repeated_notes_from_time_signature((5, 32), pitch = "d''")
[Note("d''32"), Note("d''32"), Note("d''32"), Note("d''32"), Note("d''32")]
```

Return list of notes.

notetools.make_repeated_notes_from_time_signatures

abjad.tools.notetools.**make_repeated_notes_from_time_signatures**(*time_signatures*, *pitch="c"'*)

Make repated notes from *time_signatures*:

```
notetools.make_repeated_notes_from_time_signatures([(2, 8), (3, 32)])
[[Note("c'8"), Note("c'8")], [Note("c'32"), Note("c'32"), Note("c'32")]]
```

Make repeated notes with *pitch* from *time_signatures*:

```
abjad> notetools.make_repeated_notes_from_time_signatures([(2, 8), (3, 32)], pitch = "d''")
[[Note("d''8"), Note("d''8")], [Note("d''32"), Note("d''32"), Note("d''32")]]
```

Return two-dimensional list of note lists.

Use `seqtools.flatten_sequence( )` to flatten output if required.

notetools.make_repeated_notes_with_shorter_notes_at_end

abjad.tools.notetools.**make_repeated_notes_with_shorter_notes_at_end**(*pitch*, *written_duration*, *total_duration*, *prolation=Duration(1, 1)*)

Make repeated notes with *pitch* and *written_duration* summing to *total_duration* under *prolation*:

```
abjad> voice = Voice(notetools.make_repeated_notes_with_shorter_notes_at_end(0, Duration(1, 16),

abjad> f(voice)
\new Voice {
   c'16
   c'16
   c'16
   c'16
}
```

Fill binary remaining duration with binary notes of lesser written duration:

```
abjad> voice = Voice(notetools.make_repeated_notes_with_shorter_notes_at_end(0, Duration(1, 16),

abjad> f(voice)
\new Voice {
   c'16
   c'16
   c'16
   c'16
   c'32
}
```

Fill nonbinary remaining duration with ad hoc tuplet:

```
abjad> voice = Voice(notetools.make_repeated_notes_with_shorter_notes_at_end(0, Duration(1, 16),

abjad> f(voice)
\new Voice {
   c'16
   c'16
   c'16
   c'16
   c'16
   c'16
   \times 4/5 {
      c'32
   }
}
```

Set *prolation* when constructing notes in a nonbinary measure.

Return list of newly constructed components. Changed in version 1.1.2: renamed `construct.note_train( )` to `notetools.make_repeated_notes_with_shorter_notes_at_end( )`.

notetools.yield_groups_of_notes_in_sequence

`abjad.tools.notetools.`**`yield_groups_of_notes_in_sequence`**`(`*sequence*`)`

New in version 1.1.2. Yield groups of notes in *sequence*:

```
abjad> staff = Staff("c'8 d'8 r8 r8 <e' g'>8 <f' a'>8 g'8 a'8 r8 r8 <b' d''>8 <c'' e''>8")

abjad> f(staff)
\new Staff {
   c'8
   d'8
   r8
   r8
   <e' g'>8
   <f' a'>8
   g'8
   a'8
   r8
   r8
   <b' d''>8
   <c'' e''>8
}

abjad> for note in notetools.yield_groups_of_notes_in_sequence(staff):
...     note
...
(Note("c'8"), Note("d'8"))
(Note("g'8"), Note("a'8"))
```

Return generator.

pitchtools

pitchtools.Accidental

class abjad.tools.pitchtools.**Accidental**

Bases: abjad.core._StrictComparator._StrictComparator._StrictComparator, abjad.core._Immutable._Immutable._Immutable New in version 1.1.2. Abjad model of the accidental:

```
abjad> pitchtools.Accidental('s')
Accidental('s')
```

Accidentals are immutable.

alphabetic_string

Read-only alphabetic string:

```
abjad> accidental = pitchtools.Accidental('s')
abjad> accidental.alphabetic_string
's'
```

Return string.

format

Read-only LilyPond input format of accidental:

```
abjad> accidental = pitchtools.Accidental('s')
abjad> accidental.format
's'
```

Return string.

is_adjusted

True for all accidentals equal to a nonzero number of semitones. False otherwise:

```
abjad> accidental = pitchtools.Accidental('s')
abjad> accidental.is_adjusted
True
```

Return boolean.

name_string

Read-only name string of accidental:

```
abjad> accidental = pitchtools.Accidental('s')
abjad> accidental.name_string
'sharp'
```

Return string.

semitones

Read-only semitones of accidental:

```
abjad> accidental = pitchtools.Accidental('s')
abjad> accidental.semitones
1
```

Return number.

symbolic_string

Read-only symbolic string of accidental:

```
abjad> accidental = pitchtools.Accidental('s')
abjad> accidental.symbolic_string
'#'
```

Return string.

pitchtools.HarmonicChromaticInterval

class `abjad.tools.pitchtools.`**`HarmonicChromaticInterval`**

Bases: `abjad.tools.pitchtools._ChromaticInterval._ChromaticInterval._ChromaticInterval`, `abjad.tools.pitchtools._HarmonicInterval._HarmonicInterval._HarmonicInterval`

New in version 1.1.2. Abjad model of harmonic chromatic interval:

```
abjad> pitchtools.HarmonicChromaticInterval(-14)
HarmonicChromaticInterval(14)
```

Harmonic chromatic intervals are immutable.

`harmonic_chromatic_interval_class`

Read-only harmonic chromatic interval-class:

```
abjad> harmonic_chromatic_interval = pitchtools.HarmonicChromaticInterval(14)
abjad> harmonic_chromatic_interval.harmonic_chromatic_interval_class
HarmonicChromaticIntervalClass(2)
```

Return harmonic chromatic interval-class.

pitchtools.HarmonicChromaticIntervalClass

class `abjad.tools.pitchtools.`**`HarmonicChromaticIntervalClass`**

Bases: `abjad.tools.pitchtools._ChromaticIntervalClass._ChromaticIntervalClass._Chromatic`
`abjad.tools.pitchtools._HarmonicIntervalClass._HarmonicIntervalClass._HarmonicInterval`

New in version 1.1.2. Abjad model of harmonic chromatic interval-class:

```
abjad> pitchtools.HarmonicChromaticIntervalClass(-14)
HarmonicChromaticIntervalClass(2)
```

Harmonic chromatic interval-classes are immutable.

pitchtools.HarmonicChromaticIntervalClassVector

class `abjad.tools.pitchtools.`**`HarmonicChromaticIntervalClassVector`**(*expr*)

Bases: `abjad.tools.pitchtools._Vector._Vector._Vector` New in version 1.1.2. Abjad model of harmonic chromatic interval-class vector:

```
abjad> staff = Staff(macros.scale(5))
abjad> hcicv = pitchtools.HarmonicChromaticIntervalClassVector(staff)
abjad> print hcicv
0 1 3 2 1 2 0 1 0 0 0 0
```

Harmonic chromatic interval-class vector is quartertone-aware:

```
abjad> staff.append(Note(1.5, (1, 4)))
abjad> hcicv = pitchtools.HarmonicChromaticIntervalClassVector(staff)
abjad> print hcicv
0 1 3 2 1 2 0 1 0 0 0 0
1 1 1 1 0 1 0 0 0 0 0 0
```

Harmonic chromatic interval-class vectors are immutable.

has_none_of(*chromatic_interval_numbers*)

True when harmonic chromatic interval-class vector contains none of *chromatic_interval_numbers*. Otherwise false:

```
abjad> hcicv = pitchtools.HarmonicChromaticIntervalClassVector(Staff(macros.scale(5)))
abjad> hcicv.has_none_of([9, 10, 11])
True
```

Return boolean.

pitchtools.HarmonicChromaticIntervalSegment

class abjad.tools.pitchtools.**HarmonicChromaticIntervalSegment**

Bases: abjad.tools.pitchtools._IntervalSegment._IntervalSegment._IntervalSegment

New in version 1.1.2. Abjad model of harmonic chromatic interval segment:

```
abjad> pitchtools.HarmonicChromaticIntervalSegment([10, -12, -13, -13.5])
HarmonicChromaticIntervalSegment(10, 12, 13, 13.5)
```

Harmonic chromatic interval segments are immutable.

pitchtools.HarmonicChromaticIntervalSet

class abjad.tools.pitchtools.**HarmonicChromaticIntervalSet**

Bases: abjad.tools.pitchtools._IntervalSet._IntervalSet._IntervalSet New in version 1.1.2. Abjad model of harmonic chromatic interval set:

```
abjad> pitchtools.HarmonicChromaticIntervalSet([10, -12, -13, -13, -13.5])
HarmonicChromaticIntervalSet(10, 12, 13, 13.5)
```

Harmonic chromatic interval sets are immutable.

harmonic_chromatic_interval_numbers

harmonic_chromatic_intervals

pitchtools.HarmonicCounterpointInterval

class abjad.tools.pitchtools.**HarmonicCounterpointInterval**

Bases: abjad.tools.pitchtools._CounterpointInterval._CounterpointInterval._CounterpointI abjad.tools.pitchtools._HarmonicInterval._HarmonicInterval._HarmonicInterval

New in version 1.1.2. Abjad model of harmonic counterpoint interval:

```
abjad> pitchtools.HarmonicCounterpointInterval(-9)
HarmonicCounterpointInterval(9)
```

Harmonic counterpoint intervals are immutable.

harmonic_counterpoint_interval_class

pitchtools.HarmonicCounterpointIntervalClass

class abjad.tools.pitchtools.**HarmonicCounterpointIntervalClass**

Bases: abjad.tools.pitchtools._CounterpointIntervalClass._CounterpointIntervalClass._Cour abjad.tools.pitchtools._HarmonicIntervalClass._HarmonicIntervalClass._HarmonicInterval

New in version 1.1.2. Abjad model of harmonic counterpoint interval-class:

```
abjad> pitchtools.HarmonicCounterpointIntervalClass(-9)
HarmonicCounterpointIntervalClass(2)
```

Harmonic counterpoint interval-classes are immutable.

pitchtools.HarmonicDiatonicInterval

class `abjad.tools.pitchtools.`**`HarmonicDiatonicInterval`**

Bases: `abjad.tools.pitchtools._DiatonicInterval._DiatonicInterval._DiatonicInterval`, `abjad.tools.pitchtools._HarmonicInterval._HarmonicInterval._HarmonicInterval`

New in version 1.1.2. Abjad model harmonic diatonic interval:

```
abjad> pitchtools.HarmonicDiatonicInterval('M9')
HarmonicDiatonicInterval('M9')
```

Harmonic diatonic intervals are immutable.

`harmonic_counterpoint_interval`

`harmonic_diatonic_interval_class`

`melodic_diatonic_interval_ascending`

`melodic_diatonic_interval_descending`

`semitones`

`staff_spaces`

pitchtools.HarmonicDiatonicIntervalClass

class `abjad.tools.pitchtools.`**`HarmonicDiatonicIntervalClass`**

Bases: `abjad.tools.pitchtools._DiatonicIntervalClass._DiatonicIntervalClass._DiatonicInte`
`abjad.tools.pitchtools._HarmonicIntervalClass._HarmonicIntervalClass._HarmonicInterval`

New in version 1.1.2. Abjad model harmonic diatonic interval-class:

```
abjad> pitchtools.HarmonicDiatonicIntervalClass('-M9')
HarmonicDiatonicIntervalClass('M2')
```

Harmonic diatonic interval-classes are immutable.

`invert``()`

Read-only inversion of harmonic diatonic interval-class:

```
abjad> hdic = pitchtools.HarmonicDiatonicIntervalClass('major', -9)
abjad> hdic.invert( )
HarmonicDiatonicIntervalClass('m7')
```

Return harmonic diatonic interval-class.

pitchtools.HarmonicDiatonicIntervalClassSet

class `abjad.tools.pitchtools.`**`HarmonicDiatonicIntervalClassSet`**

Bases: `abjad.tools.pitchtools._IntervalClassSet._IntervalClassSet._IntervalClassSet`

New in version 1.1.2. Abjad model of harmonic diatonic interval-class set:

```
abjad> pitchtools.HarmonicDiatonicIntervalClassSet('m2 M2 m3 M3') # doctest: +SKIP
HarmonicDiatonicIntervalClassSet('m2 M2 m3 M3')
```

Harmonic diatonic interval-class sets are immutable.

harmonic_diatonic_interval_classes

pitchtools.HarmonicDiatonicIntervalSegment

class abjad.tools.pitchtools.**HarmonicDiatonicIntervalSegment**

Bases: abjad.tools.pitchtools._IntervalSegment._IntervalSegment._IntervalSegment

New in version 1.1.2. Abjad model of harmonic diatonic interval segment:

```
abjad> pitchtools.HarmonicDiatonicIntervalSegment('m2 M9 m3 M3')
HarmonicDiatonicIntervalSegment('m2 M9 m3 M3')
```

Harmonic diatonic interval segments are immutable.

harmonic_chromatic_interval_segment

melodic_chromatic_interval_segment

melodic_diatonic_interval_segment

pitchtools.HarmonicDiatonicIntervalSet

class abjad.tools.pitchtools.**HarmonicDiatonicIntervalSet**

Bases: abjad.tools.pitchtools._IntervalSet._IntervalSet._IntervalSet New in version 1.1.2. Abjad model of harmonic diatonic interval set:

```
abjad> pitchtools.HarmonicDiatonicIntervalSet('m2 m2 M2 M9')
HarmonicDiatonicIntervalSet('m2 M2 M9')
```

Harmonic diatonic interval sets are immutable.

harmonic_chromatic_interval_set

harmonic_diatonic_interval_numbers

harmonic_diatonic_intervals

pitchtools.InversionEquivalentChromaticIntervalClass

class abjad.tools.pitchtools.**InversionEquivalentChromaticIntervalClass**

Bases: abjad.tools.pitchtools._IntervalClass._IntervalClass._IntervalClass

New in version 1.1.2. Abjad model of inversion-equivalent chromatic interval-class:

```
abjad> pitchtools.InversionEquivalentChromaticIntervalClass(1)
InversionEquivalentChromaticIntervalClass(1)
```

Inversion-equivalent chromatic interval-classes are immutable.

inversion_equivalent_chromatic_interval_number

pitchtools.InversionEquivalentChromaticIntervalClassSegment

class abjad.tools.pitchtools.**InversionEquivalentChromaticIntervalClassSegment**

Bases: abjad.tools.pitchtools._IntervalClassSegment._IntervalClassSegment._IntervalClassS

New in version 1.1.2. Abjad model of inversion-equivalent chromatic interval-class segment:

```
abjad> pitchtools.InversionEquivalentChromaticIntervalClassSegment([2, 1, 0, 5.5, 6])
InversionEquivalentChromaticIntervalClassSegment(2, 1, 0, 5.5, 6)
```

Inversion-equivalent chromatic interval-class segments are immutable.

pitchtools.InversionEquivalentChromaticIntervalClassSet

class abjad.tools.pitchtools.**InversionEquivalentChromaticIntervalClassSet**

Bases: abjad.tools.pitchtools._IntervalClassSet._IntervalClassSet._IntervalClassSet

New in version 1.1.2. Abjad model of inversion-equivalent chromatic interval-class set:

```
abjad> pitchtools.InversionEquivalentChromaticIntervalClassSet([1, 1, 6, 2, 2])
InversionEquivalentChromaticIntervalClassSet(1, 2, 6)
```

Inversion-equivalent chromatic interval-class sets are immutable.

inversion_equivalent_chromatic_interval_class_numbers

inversion_equivalent_chromatic_interval_classes

pitchtools.InversionEquivalentChromaticIntervalClassVector

class abjad.tools.pitchtools.**InversionEquivalentChromaticIntervalClassVector**(**args*, ***kwargs*)

Bases: abjad.tools.pitchtools._Vector._Vector._Vector New in version 1.1.2. Abjad model of inversion-equivalent chromatic interval-class vector:

```
abjad> pitchtools.InversionEquivalentChromaticIntervalClassVector([1, 1, 6, 2, 2, 2])
InversionEquivalentChromaticIntervalClassVector(0 | 2 3 0 0 0 1)
```

Initialize by inversion-equivalent chromatic interval-class counts:

```
abjad> pitchtools.InversionEquivalentChromaticIntervalClassVector(counts = [2, 3, 0, 0, 0, 1])
InversionEquivalentChromaticIntervalClassVector(0 | 2 3 0 0 0 1)
```

Inversion-equivalent chromatic interval-class vectors are immutable.

pitchtools.InversionEquivalentDiatonicIntervalClass

class abjad.tools.pitchtools.**InversionEquivalentDiatonicIntervalClass**

Bases: abjad.tools.pitchtools._DiatonicIntervalClass._DiatonicIntervalClass._DiatonicInte

New in version 1.1.2. Abjad model of inversion-equivalent diatonic interval-class:

```
abjad> pitchtools.InversionEquivalentDiatonicIntervalClass('-m14')
InversionEquivalentDiatonicIntervalClass('M2')
```

Inversion-equivalent diatonic interval-classes are immutable.

pitchtools.InversionEquivalentDiatonicIntervalClassSegment

class abjad.tools.pitchtools.**InversionEquivalentDiatonicIntervalClassSegment**

Bases: abjad.tools.pitchtools._IntervalSegment._IntervalSegment._IntervalSegment

New in version 1.1.2. Abjad model of inversion-equivalent diatonic interval-class segment:

```
abjad> pitchtools.InversionEquivalentDiatonicIntervalClassSegment([('major', 2), ('major', 9), (
InversionEquivalentDiatonicIntervalClassSegment(M2, M2, m2, m2)
```

Inversion-equivalent diatonic interval-class segments are immutable.

is_tertian

True when all diatonic interval-classes in segment are tertian. Otherwise false:

```
abjad> dics = pitchtools.InversionEquivalentDiatonicIntervalClassSegment([('major', 3), ('mi
abjad> dics.is_tertian
True
```

Return boolean.

pitchtools.InversionEquivalentDiatonicIntervalClassVector

class abjad.tools.pitchtools.**InversionEquivalentDiatonicIntervalClassVector**(*expr*)

Bases: abjad.tools.pitchtools._Vector._Vector._Vector New in version 1.1.2. Abjad model of inversion-equivalent diatonic interval-class vector:

```
abjad> staff = Staff(macros.scale(5))
abjad> pitchtools.InversionEquivalentDiatonicIntervalClassVector(staff)
InversionEquivalentDiatonicIntervalClassVector(P1: 0, aug1: 0, m2: 1, M2: 3, aug2: 0, dim3: 0, m
```

Inversion-equivalent diatonic interval-class vector are not quatertone-aware.

Inversion-equivalent diatonic interval-class vectors are immutable.

pitchtools.MelodicChromaticInterval

class abjad.tools.pitchtools.**MelodicChromaticInterval**

Bases: abjad.tools.pitchtools._ChromaticInterval._ChromaticInterval._ChromaticInterval, abjad.tools.pitchtools._MelodicInterval._MelodicInterval._MelodicInterval

New in version 1.1.2. Abjad model of melodic chromatic interval:

```
abjad> pitchtools.MelodicChromaticInterval(-14)
MelodicChromaticInterval(-14)
```

Melodic chromatic intervals are immutable.

chromatic_interval_number

Read-only chromatic interval number:

```
abjad> pitchtools.MelodicChromaticInterval(-14).chromatic_interval_number
-14
```

Return integer or float.

direction_number

Read-only numeric sign:

```
abjad> pitchtools.MelodicChromaticInterval(-14).direction_number
-1
```

Return integer.

harmonic_chromatic_interval

Read-only harmonic chromatic interval:

```
abjad> pitchtools.MelodicChromaticInterval(-14).harmonic_chromatic_interval
HarmonicChromaticInterval(14)
```

Return harmonic chromatic interval.

melodic_chromatic_interval_class
Read-only melodic chromatic interval-class:

```
abjad> pitchtools.MelodicChromaticInterval(-14).melodic_chromatic_interval_class
MelodicChromaticIntervalClass(-2)
```

Return melodic chromatic interval-class.

pitchtools.MelodicChromaticIntervalClass

class `abjad.tools.pitchtools.`**`MelodicChromaticIntervalClass`**
Bases: `abjad.tools.pitchtools._ChromaticIntervalClass._ChromaticIntervalClass._Chromatic` `abjad.tools.pitchtools._MelodicIntervalClass._MelodicIntervalClass._MelodicIntervalCla`
New in version 1.1.2. Abjad model of melodic chromatic interval-class:

```
abjad> pitchtools.MelodicChromaticIntervalClass(-14)
MelodicChromaticIntervalClass(-2)
```

Melodic chromatic interval-classes are immutable.

pitchtools.MelodicChromaticIntervalClassSegment

class `abjad.tools.pitchtools.`**`MelodicChromaticIntervalClassSegment`**
Bases: `abjad.tools.pitchtools._IntervalClassSegment._IntervalClassSegment._IntervalClassS`
New in version 1.1.2. Abjad model of melodic chromatic interval-class segment:

```
abjad> pitchtools.MelodicChromaticIntervalClassSegment([-2, -14, 3, 5.5, 6.5])
MelodicChromaticIntervalClassSegment(-2, -2, +3, +5.5, +6.5)
```

Melodic chromatic interval-class segments are immutable.

pitchtools.MelodicChromaticIntervalClassVector

class `abjad.tools.pitchtools.`**`MelodicChromaticIntervalClassVector`**(*mcic_tokens*)
Bases: `abjad.tools.pitchtools._Vector._Vector._Vector` New in version 1.1.2. Abjad model of melodic chromatic interval-class vector:

```
abjad> print pitchtools.MelodicChromaticIntervalClassVector([-2, -14, 3, 5.5, 6.5])
  .   |   .   .   1   .   .   .   |   .   .   .   .   .   .
      |   .   2   .   .   .   .   |   .   .   .   .   .   .
      |   .   .   .   .   .   1   |   1   .   .   .   .   .
      |   .   .   .   .   .   .   |   .   .   .   .   .   .
```

Melodic chromatic interval-class vectors are immutable.

pitchtools.MelodicChromaticIntervalSegment

class `abjad.tools.pitchtools.`**`MelodicChromaticIntervalSegment`**
Bases: `abjad.tools.pitchtools._IntervalSegment._IntervalSegment._IntervalSegment`
New in version 1.1.2. Abjad model of melodic chromatic interval segment:

```
abjad> pitchtools.MelodicChromaticIntervalSegment([11, 13, 13.5, -2, 2.5])
MelodicChromaticIntervalSegment(+11, +13, +13.5, -2, +2.5)
```

Melodic chromatic interval segments are immutable.

harmonic_chromatic_interval_segment

melodic_chromatic_interval_class_segment

melodic_chromatic_interval_class_vector

melodic_chromatic_interval_numbers

slope

The slope of a melodic interval segment is the sum of its intervals divided by its length:

```
abjad> pitchtools.MelodicChromaticIntervalSegment([1, 2]).slope
Fraction(3, 2)
```

Return fraction.

spread

The maximum harmonic interval spanned by any combination of the intervals within a harmonic chromatic interval segment:

```
abjad> pitchtools.MelodicChromaticIntervalSegment([1, 2, -3, 1, -2, 1]).spread
HarmonicChromaticInterval(4)
abjad> pitchtools.MelodicChromaticIntervalSegment([1, 1, 1, 2, -3, -2]).spread
HarmonicChromaticInterval(5)
```

Return harmonic chromatic interval.

pitchtools.MelodicChromaticIntervalSet

class `abjad.tools.pitchtools.`**MelodicChromaticIntervalSet**

Bases: `abjad.tools.pitchtools._IntervalSet._IntervalSet._IntervalSet` New in version 1.1.2. Abjad model of melodic chromatic interval set:

```
abjad> pitchtools.MelodicChromaticIntervalSet([11, 11, 13.5, 13.5])
MelodicChromaticIntervalSet(+11, +13.5)
```

Melodic chromatic interval sets are immutable.

harmonic_chromatic_interval_set

melodic_chromatic_interval_numbers

melodic_chromatic_intervals

pitchtools.MelodicCounterpointInterval

class `abjad.tools.pitchtools.`**MelodicCounterpointInterval**

Bases: `abjad.tools.pitchtools._CounterpointInterval._CounterpointInterval._CounterpointIr` `abjad.tools.pitchtools._MelodicInterval._MelodicInterval._MelodicInterval`
New in version 1.1.2. Abjad model of melodic counterpoint interval:

```
abjad> pitchtools.MelodicCounterpointInterval(-9)
MelodicCounterpointInterval(-9)
```

Melodic counterpoint intervals are immutable.

direction_number

melodic_counterpoint_interval_class

pitchtools.MelodicCounterpointIntervalClass

class `abjad.tools.pitchtools.`**`MelodicCounterpointIntervalClass`**

Bases: `abjad.tools.pitchtools._CounterpointIntervalClass._CounterpointIntervalClass._Cour` `abjad.tools.pitchtools._MelodicIntervalClass._MelodicIntervalClass._MelodicIntervalCla`

New in version 1.1.2. Abjad model of melodic counterpoint interval-class:

```
abjad> pitchtools.MelodicCounterpointIntervalClass(-9)
MelodicCounterpointIntervalClass(-2)
```

Melodic counterpoint interval-classes are immutable.

pitchtools.MelodicDiatonicInterval

class `abjad.tools.pitchtools.`**`MelodicDiatonicInterval`**

Bases: `abjad.tools.pitchtools._DiatonicInterval._DiatonicInterval._DiatonicInterval`, `abjad.tools.pitchtools._MelodicInterval._MelodicInterval._MelodicInterval`

New in version 1.1.2. Abjad model of melodic diatonic interval:

```
abjad> pitchtools.MelodicDiatonicInterval('+M9')
MelodicDiatonicInterval('+M9')
```

Melodic diatonic intervals are immutable.

direction_number

direction_string

harmonic_chromatic_interval

harmonic_counterpoint_interval

harmonic_diatonic_interval

inversion_equivalent_chromatic_interval_class

melodic_chromatic_interval

melodic_counterpoint_interval

melodic_diatonic_interval_class

semitones

staff_spaces

pitchtools.MelodicDiatonicIntervalClass

class `abjad.tools.pitchtools.`**`MelodicDiatonicIntervalClass`**

Bases: `abjad.tools.pitchtools._DiatonicIntervalClass._DiatonicIntervalClass._DiatonicInte` `abjad.tools.pitchtools._MelodicIntervalClass._MelodicIntervalClass._MelodicIntervalCla`

New in version 1.1.2. Abjad model of melodic diatonic interval-class:

```
abjad> pitchtools.MelodicDiatonicIntervalClass('-M9')
MelodicDiatonicIntervalClass('-M2')
```

Melodic diatonic interval-classes are immutable.

direction_number

direction_symbol

direction_word

pitchtools.MelodicDiatonicIntervalSegment

class abjad.tools.pitchtools.**MelodicDiatonicIntervalSegment**

Bases: abjad.tools.pitchtools._IntervalSegment._IntervalSegment._IntervalSegment

New in version 1.1.2. Abjad model of melodic diatonic interval segment:

```
abjad> pitchtools.MelodicDiatonicIntervalSegment('M2 M9 -m3 -P4')
MelodicDiatonicIntervalSegment('+M2 +M9 -m3 -P4')
```

Melodic diatonic interval segments are immutable.

harmonic_chromatic_interval_segment

harmonic_diatonic_interval_segment

melodic_chromatic_interval_segment

pitchtools.MelodicDiatonicIntervalSet

class abjad.tools.pitchtools.**MelodicDiatonicIntervalSet**

Bases: abjad.tools.pitchtools._IntervalSet._IntervalSet._IntervalSet New in version 1.1.2. Abjad model of melodic diatonic interval set:

```
abjad> pitchtools.MelodicDiatonicIntervalSet('M2 M2 -m3 -P4')
MelodicDiatonicIntervalSet('-P4 -m3 +M2')
```

Melodic diatonic interval sets are immutable.

harmonic_chromatic_interval_set

harmonic_diatonic_interval_set

melodic_chromatic_interval_set

melodic_diatonic_interval_numbers

melodic_diatonic_intervals

pitchtools.NamedChromaticPitch

class abjad.tools.pitchtools.**NamedChromaticPitch**

Bases: abjad.tools.pitchtools._Pitch._Pitch._Pitch New in version 1.1.1. Abjad model of named chromatic pitch:

```
abjad> pitchtools.NamedChromaticPitch("cs''")
NamedChromaticPitch("cs''")
```

Named chromatic pitches are immutable.

chromatic_pitch_class_name

Read-only chromatic pitch-class name:

```
abjad> named_chromatic_pitch = pitchtools.NamedChromaticPitch("cs''")
abjad> named_chromatic_pitch.chromatic_pitch_class_name
'cs'
```

Return string.

chromatic_pitch_class_number
Read-only chromatic pitch-class number:

```
abjad> named_chromatic_pitch = pitchtools.NamedChromaticPitch("cs''")
abjad> named_chromatic_pitch.chromatic_pitch_class_number
1
```

Return integer or float.

chromatic_pitch_name
Read-only chromatic pitch name:

```
abjad> named_chromatic_pitch = pitchtools.NamedChromaticPitch("cs''")
abjad> named_chromatic_pitch.chromatic_pitch_name
"cs''"
```

Return string.

chromatic_pitch_number
Read-only chromatic pitch-class number:

```
abjad> named_chromatic_pitch = pitchtools.NamedChromaticPitch("cs''")
abjad> named_chromatic_pitch.chromatic_pitch_number
13
```

Return integer or float.

deviation_in_cents
Read-only deviation of named chromatic pitch in cents:

```
abjad> named_chromatic_pitch = pitchtools.NamedChromaticPitch("cs''")
abjad> named_chromatic_pitch.deviation_in_cents is None
True
```

Return integer or none.

diatonic_pitch_class_name
Read-only diatonic pitch-class name:

```
abjad> named_diatonic_pitch = pitchtools.NamedChromaticPitch("cs''")
abjad> named_diatonic_pitch.diatonic_pitch_class_name
'c'
```

Return string.

diatonic_pitch_class_number
Read-only diatonic pitch-class number:

```
abjad> named_diatonic_pitch = pitchtools.NamedChromaticPitch("cs''")
abjad> named_diatonic_pitch.diatonic_pitch_class_number
0
```

Return integer.

diatonic_pitch_name
Read-only diatonic pitch name:

```
abjad> named_diatonic_pitch = pitchtools.NamedChromaticPitch("cs''")
abjad> named_diatonic_pitch.diatonic_pitch_name
"c''"
```

Return string.

diatonic_pitch_number

Read-only diatonic pitch number:

```
abjad> named_diatonic_pitch = pitchtools.NamedChromaticPitch("cs''")
abjad> named_diatonic_pitch.diatonic_pitch_number
7
```

Return integer.

format

Read-only LilyPond input format of named chromatic pitch:

```
abjad> named_chromatic_pitch = pitchtools.NamedChromaticPitch("cs''")
abjad> named_chromatic_pitch.format
"cs''"
```

Return string.

named_chromatic_pitch_class

Read-only named pitch-class:

```
abjad> named_chromatic_pitch = pitchtools.NamedChromaticPitch("cs''")
abjad> named_chromatic_pitch.named_chromatic_pitch_class
NamedChromaticPitchClass('cs')
```

Return named chromatic pitch-class.

named_diatonic_pitch

Read-only named diatonic pitch:

```
abjad> named_chromatic_pitch = pitchtools.NamedChromaticPitch("cs''")
abjad> named_chromatic_pitch.named_diatonic_pitch
NamedDiatonicPitch("c''")
```

Return named diatonic pitch.

named_diatonic_pitch_class

Read-only named diatonic pitch-class:

```
abjad> named_chromatic_pitch = pitchtools.NamedChromaticPitch("cs''")
abjad> named_chromatic_pitch.named_diatonic_pitch_class
NamedDiatonicPitchClass('c')
```

Return named diatonic pitch-class.

numbered_chromatic_pitch

Read-only numbered chromatic pitch from named chromatic pitch:

```
abjad> named_chromatic_pitch = pitchtools.NamedChromaticPitch("cs''")
abjad> named_chromatic_pitch.numbered_chromatic_pitch_class
NumberedChromaticPitchClass(1)
```

Return numbered chromatic pitch-class.

numbered_chromatic_pitch_class

Read-only numbered pitch-class:

```
abjad> named_chromatic_pitch = pitchtools.NamedChromaticPitch("cs''")
abjad> named_chromatic_pitch.numbered_chromatic_pitch_class
NumberedChromaticPitchClass(1)
```

Return numbered chromatic pitch-class.

numbered_diatonic_pitch

Read-only numbered diatonic pitch:

```
abjad> named_chromatic_pitch = pitchtools.NamedChromaticPitch("cs''")
abjad> named_chromatic_pitch.numbered_diatonic_pitch
NumberedDiatonicPitch(7)
```

Return numbered diatonic pitch.

numbered_diatonic_pitch_class

Read-only numbered diatonic pitch:

```
abjad> named_chromatic_pitch = pitchtools.NamedChromaticPitch("cs''")
abjad> named_chromatic_pitch.numbered_diatonic_pitch_class
NumberedDiatonicPitchClass(0)
```

Return numbered diatonic pitch-class.

octave_number

Read-only integer octave number:

```
abjad> named_chromatic_pitch = pitchtools.NamedChromaticPitch("cs''")
abjad> named_chromatic_pitch.octave_number
5
```

Return integer.

pitchtools.NamedChromaticPitchClass

class abjad.tools.pitchtools.**NamedChromaticPitchClass**

Bases: abjad.tools.pitchtools._PitchClass._PitchClass._PitchClass New in version 1.1.2. Abjad model of named chromatic pitch-class:

```
abjad> pitchtools.NamedChromaticPitchClass('cs')
NamedChromaticPitchClass('cs')
```

Named chromatic pitch-classes are immutable.

apply_accidental(*accidental*)

Apply *accidental*:

```
abjad> named_chromatic_pitch_class = pitchtools.NamedChromaticPitchClass('cs')
abjad> named_chromatic_pitch_class.apply_accidental('qs')
NamedChromaticPitchClass('ctqs')
```

Return named chromatic pitch-class.

numbered_chromatic_pitch_class

Read-only numbered chromatic pitch-class:

```
abjad> named_chromatic_pitch_class = pitchtools.NamedChromaticPitchClass('cs')
abjad> named_chromatic_pitch_class.numbered_chromatic_pitch_class
NumberedChromaticPitchClass(1)
```

Return numbered chromatic pitch-class.

transpose (*melodic_diatonic_interval*)

Transpose named chromatic pitch-class by *melodic_diatonic_interval*:

```
abjad> named_chromatic_pitch_class = pitchtools.NamedChromaticPitchClass('cs')
abjad> named_chromatic_pitch_class.transpose(pitchtools.MelodicDiatonicInterval('major', 2))
NamedChromaticPitchClass('ds')
```

Return named chromatic pitch-class.

pitchtools.NamedChromaticPitchClassSegment

class abjad.tools.pitchtools.**NamedChromaticPitchClassSegment**

Bases: abjad.tools.pitchtools._PitchClassSegment._PitchClassSegment._PitchClassSegment

New in version 1.1.2. Abjad model of named chromatic pitch-class segment:

```
abjad> pitchtools.NamedChromaticPitchClassSegment(['gs', 'a', 'as', 'c', 'cs'])
NamedChromaticPitchClassSegment(['gs', 'a', 'as', 'c', 'cs'])
```

Named chromatic pitch-class segments are immutable.

inversion_equivalent_diatonic_interval_class_segment

is_equivalent_under_transposition (*arg*)

named_chromatic_pitch_class_set

named_chromatic_pitch_classes

numbered_chromatic_pitch_class_segment

numbered_chromatic_pitch_class_set

numbered_chromatic_pitch_classes

retrograde ()

rotate (*n*)

transpose (*melodic_diatonic_interval*)

pitchtools.NamedChromaticPitchClassSet

class abjad.tools.pitchtools.**NamedChromaticPitchClassSet**

Bases: abjad.tools.pitchtools._PitchClassSet._PitchClassSet._PitchClassSet

New in version 1.1.2. Abjad model of a named chromatic pitch-class set:

```
abjad> named_chromatic_pitch_class_set = pitchtools.NamedChromaticPitchClassSet(['gs', 'g', 'as'

abjad> named_chromatic_pitch_class_set
NamedChromaticPitchClassSet(['as', 'c', 'cs', 'g', 'gs'])

abjad> print named_chromatic_pitch_class_set
{as, c, cs, g, gs}
```

Named chromatic pitch-class sets are immutable.

inversion_equivalent_diatonic_interval_class_vector

named_chromatic_pitch_classes

Read-only named chromatic pitch-classes:

```
abjad> named_chromatic_pitch_class_set = pitchtools.NamedChromaticPitchClassSet(['gs', 'g',
abjad> named_chromatic_pitch_class_set.named_chromatic_pitch_classes # doctest: +SKIP
(NamedChromaticPitchClass('c'), NamedChromaticPitchClass('cs'), NamedChromaticPitchClass('g'
```

Return tuple.

numbered_chromatic_pitch_class_set

order_by (*npc_seg*)

transpose (*melodic_diatonic_interval*)
Transpose all npcs in self by melodic diatonic interval.

pitchtools.NamedChromaticPitchSegment

class abjad.tools.pitchtools.**NamedChromaticPitchSegment**
Bases: abjad.tools.pitchtools._PitchSegment._PitchSegment._PitchSegment New in version 1.1.2. Abjad model of a named chromatic pitch segment:

```
abjad> pitchtools.NamedChromaticPitchSegment(['bf', 'bqf', "fs'", "g'", 'bqf', "g'"])
NamedChromaticPitchSegment("bf bqf fs' g' bqf g'")
```

Named chromtic pitch segments are immutable.

chromatic_pitch_numbers

harmonic_chromatic_interval_class_segment

harmonic_chromatic_interval_segment

harmonic_diatonic_interval_class_segment

harmonic_diatonic_interval_segment

inflection_point_count

inversion_equivalent_chromatic_interval_class_segment

inversion_equivalent_chromatic_interval_class_set

inversion_equivalent_chromatic_interval_class_vector

local_maxima

local_minima

melodic_chromatic_interval_class_segment

melodic_chromatic_interval_segment

melodic_diatonic_interval_class_segment

melodic_diatonic_interval_segment

named_chromatic_pitch_class_vector

named_chromatic_pitch_set

named_chromatic_pitch_vector

named_chromatic_pitches

numbered_chromatic_pitch_class_segment

numbered_chromatic_pitch_class_set

transpose (*melodic_interval*)
Transpose pitches in pitch segment by melodic interval and emit new pitch segment.

pitchtools.NamedChromaticPitchSet

class abjad.tools.pitchtools.**NamedChromaticPitchSet**
Bases: abjad.tools.pitchtools._PitchSet._PitchSet._PitchSet New in version 1.1.2. Abjad model of a named chromatic pitch set:

```
abjad> pitchtools.NamedChromaticPitchSet(['bf', 'bqf', "fs'", "g'", 'bqf', "g'"])
NamedChromaticPitchSet(['bf', 'bqf', "fs'", "g'"])
```

Named chromatic pitch sets are immutable.

chromatic_pitch_numbers

duplicate_pitch_classes

is_pitch_class_unique

named_chromatic_pitches

numbered_chromatic_pitch_class_set

numbered_chromatic_pitch_classes

transpose (*n*)
Transpose all pcs in self by n.

pitchtools.NamedChromaticPitchVector

class abjad.tools.pitchtools.**NamedChromaticPitchVector** (*pitch_tokens*)
Bases: abjad.tools.pitchtools._Vector._Vector._Vector New in version 1.1.2. Abjad model of named chromatic pitch vector:

```
abjad> named_chromatic_pitch_vector = pitchtools.NamedChromaticPitchVector(["c''", "c''", "cs''"

abjad> named_chromatic_pitch_vector
NamedChromaticPitchVector(c'': 2, cs'': 3)

abjad> print named_chromatic_pitch_vector
NamedChromaticPitchVector(c'': 2, cs'': 3)
```

Named chromatic pitch vectors are immutable.

chromatic_pitch_numbers

named_chromatic_pitches

pitchtools.NamedDiatonicPitch

class abjad.tools.pitchtools.**NamedDiatonicPitch**
Bases: abjad.tools.pitchtools._DiatonicPitch._DiatonicPitch._DiatonicPitch
New in version 1.1.2. Abjad model of a named diatonic pitch:

```
abjad> named_diatonic_pitch = pitchtools.NamedDiatonicPitch("c''")
```

```
abjad> named_diatonic_pitch
NamedDiatonicPitch("c''")

abjad> print named_diatonic_pitch
c''
```

Named diatonic pitches are immutable.

chromatic_pitch_class_name

Read-only chromatic pitch-class name:

```
abjad> pitchtools.NamedDiatonicPitch("c''").chromatic_pitch_class_name
'c'
```

Return string.

chromatic_pitch_class_number

Read-only chromatic pitch-class number:

```
abjad> pitchtools.NamedDiatonicPitch("c''").chromatic_pitch_class_number
0
```

Return integer.

chromatic_pitch_name

Read-only chromatic pitch name:

```
abjad> pitchtools.NamedDiatonicPitch("c''").chromatic_pitch_name
"c''"
```

Return string.

chromatic_pitch_number

Read-only chromatic pitch number:

```
abjad> pitchtools.NamedDiatonicPitch("c''").chromatic_pitch_number
12
```

Return integer.

diatonic_pitch_class_name

Read-only diatonic pitch-class name:

```
abjad> pitchtools.NamedDiatonicPitch("c''").diatonic_pitch_class_name
'c'
```

Return string.

diatonic_pitch_class_number

Read-only diatonic pitch-class number:

```
abjad> pitchtools.NamedDiatonicPitch("c''").diatonic_pitch_class_number
0
```

Return integer.

diatonic_pitch_name

Read-only diatonic pitch name:

```
abjad> pitchtools.NamedDiatonicPitch("c''").diatonic_pitch_name
"c''"
```

Return string.

diatonic_pitch_number

Read-only diatonic pitch number:

```
abjad> pitchtools.NamedDiatonicPitch("c''").diatonic_pitch_number
7
```

Return integer.

format

Read-only LilyPond input format of named diatonic pitch:

```
abjad> pitchtools.NamedDiatonicPitch("c''").format
"c''"
```

Return string.

named_chromatic_pitch

Read-only named chromatic pitch:

```
abjad> pitchtools.NamedDiatonicPitch("c''").named_chromatic_pitch
NamedChromaticPitch("c''")
```

Return named chromatic pitch.

named_chromatic_pitch_class

Read-only named chromatic pitch-class:

```
abjad> pitchtools.NamedDiatonicPitch("c''").named_chromatic_pitch_class
NamedChromaticPitchClass('c')
```

Return named chromatic pitch-class.

named_diatonic_pitch_class

Read-only named diatonic pitch-class:

```
abjad> pitchtools.NamedDiatonicPitch("c''").named_diatonic_pitch_class
NamedDiatonicPitchClass('c')
```

Return named diatonic pitch-class.

numbered_chromatic_pitch

Read-only numbered chromatic pitch:

```
abjad> pitchtools.NamedDiatonicPitch("c''").numbered_chromatic_pitch
NumberedChromaticPitch(12)
```

Return numbered chromatic pitch.

numbered_chromatic_pitch_class

Read-only numbered chromatic pitch-class:

```
abjad> pitchtools.NamedDiatonicPitch("c''").numbered_chromatic_pitch_class
NumberedChromaticPitchClass(0)
```

Return numbered chromatic pitch-class.

numbered_diatonic_pitch

Read-only numbered diatonic pitch:

```
abjad> pitchtools.NamedDiatonicPitch("c''").numbered_diatonic_pitch
NumberedDiatonicPitch(7)
```

Return numbered diatonic pitch.

numbered_diatonic_pitch_class
Read-only numbered diatonic pitch-class:

```
abjad> pitchtools.NamedDiatonicPitch("c''").numbered_diatonic_pitch_class
NumberedDiatonicPitchClass(0)
```

Return numbered diatonic pitch-class.

pitchtools.NamedDiatonicPitchClass

class abjad.tools.pitchtools.**NamedDiatonicPitchClass**
Bases: abjad.tools.pitchtools._DiatonicPitchClass._DiatonicPitchClass._DiatonicPitchClass
New in version 1.1.2. Abjad model of a named diatonic pitch-class:

```
abjad> pitchtools.NamedDiatonicPitchClass('c')
NamedDiatonicPitchClass('c')
```

Named diatonic pitch-classes are immutable.

numbered_diatonic_pitch_class
Read-only numbered diatonic pitch-class from named diatonic pitch-class:

```
abjad> named_diatonic_pitch_class = pitchtools.NamedDiatonicPitchClass('c')
abjad> named_diatonic_pitch_class.numbered_diatonic_pitch_class
NumberedDiatonicPitchClass(0)
```

Return numbered diatonic pitch-class.

pitchtools.NumberedChromaticPitch

class abjad.tools.pitchtools.**NumberedChromaticPitch**
Bases: abjad.tools.pitchtools._ChromaticPitch._ChromaticPitch._ChromaticPitch, abjad.tools.pitchtools._NumberedPitch._NumberedPitch._NumberedPitch New in version 1.1.2. Abjad model of a numbered chromatic pitch:

```
abjad> pitchtools.NumberedChromaticPitch(13)
NumberedChromaticPitch(13)
```

Numbered chromatic pitches are immutable.

apply_accidental (*accidental=None*)
Apply *accidental*:

```
abjad> pitchtools.NumberedChromaticPitch(13).apply_accidental('flat')
NumberedChromaticPitch(12)
```

Return numbered chromatic pitch.

chromatic_pitch_number
Read-only chromatic pitch-class number:

```
abjad> pitchtools.NumberedChromaticPitch(13).chromatic_pitch_number
13
```

Return integer or float.

diatonic_pitch_class_number
Read-only diatonic pitch-class number:

```
abjad> pitchtools.NumberedChromaticPitch(13).diatonic_pitch_class_number
0
```

Return integer.

diatonic_pitch_number
Read-only diatonic pitch-class number:

```
abjad> pitchtools.NumberedChromaticPitch(13).diatonic_pitch_number
7
```

Return integer.

transpose (*n=0*)
Tranpose by *n* semitones:

```
abjad> pitchtools.NumberedChromaticPitch(13).transpose(1)
NumberedChromaticPitch(14)
```

Return numbered chromatic pitch.

pitchtools.NumberedChromaticPitchClass

class abjad.tools.pitchtools.**NumberedChromaticPitchClass**
Bases: abjad.tools.pitchtools._PitchClass._PitchClass._PitchClass New in version 1.1.2. Abjad model of a numbered chromatic pitch-class:

```
abjad> pitchtools.NumberedChromaticPitchClass(13)
NumberedChromaticPitchClass(1)
```

Numbered chromatic pitch-classes are immutable.

apply_accidental (*accidental=None*)
Emit new numbered chromatic pitch-class as sum of self and accidental.

invert ()
Invert pitch-class.

multiply (*n*)
Multiply pitch-class by n.

transpose (*n*)
Transpose pitch-class by n.

pitchtools.NumberedChromaticPitchClassColorMap

class abjad.tools.pitchtools.**NumberedChromaticPitchClassColorMap**
Bases: abjad.core._Immutable._Immutable._Immutable New in version 1.1.2. Abjad model of a numbered chromatic pitch-class color map:

```
abjad> chromatic_pitch_class_numbers = [[-8, 2, 10, 21], [0, 11, 32, 41], [15, 25, 42, 43]]
abjad> colors = ['red', 'green', 'blue']
abjad> pitchtools.NumberedChromaticPitchClassColorMap(chromatic_pitch_class_numbers, colors)
NumberedChromaticPitchClassColorMap([[-8, 2, 10, 21], [0, 11, 32, 41], [15, 25, 42, 43]], ['red'
```

Numbered chromatic pitch-class color maps are immutable.

colors

get (*key, alternative=None*)

pairs

pitch_iterables

twelve_tone_complete

twenty_four_tone_complete

pitchtools.NumberedChromaticPitchClassSegment

class abjad.tools.pitchtools.**NumberedChromaticPitchClassSegment**

Bases: abjad.tools.pitchtools._PitchClassSegment._PitchClassSegment._PitchClassSegment

New in version 1.1.2. Abjad model of a numbered chromatic pitch-class segment:

```
abjad> pitchtools.NumberedChromaticPitchClassSegment([-2, -1.5, 6, 7, -1.5, 7])
NumberedChromaticPitchClassSegment([10, 10.5, 6, 7, 10.5, 7])
```

Numbered chromatic pitch-class segments are immutable.

alpha()

Morris alpha transform of numbered chromatic pitch-class segment:

```
numbered_chromatic_pitch_class_segment = pitchtools.NumberedChromaticPitchClassSegment([10,
numbered_chromatic_pitch_class_segment.alpha( )
NumberedChromaticPitchClassSegment([11, 11.5, 7, 6, 11.5, 6])
```

Return numbered chromatic pitch-class segment.

inversion_equivalent_chromatic_interval_class_segment

Read-only inversion-equivalent chromatic interval-class segment:

```
numbered_chromatic_pitch_class_segment = pitchtools.NumberedChromaticPitchClassSegment([10,
numbered_chromatic_pitch_class_segment.inversion_equivalent_chromatic_interval_class_segment
InversionEquivalentChromaticIntervalClassSegment(0.5, 4.5, 1, 3.5, 3.5)
```

Return inversion-equivalent chromatic interval-class segment.

invert()

Invert numbered chromatic pitch-class segment:

```
numbered_chromatic_pitch_class_segment = pitchtools.NumberedChromaticPitchClassSegment([10,
numbered_chromatic_pitch_class_segment.invert( )
NumberedChromaticPitchClassSegment([2, 1.5, 6, 5, 1.5, 5])
```

Return numbered chromatic pitch-class segment.

multiply(*n*)

Multiply numbered chromatic pitch-class segment by *n*:

```
numbered_chromatic_pitch_class_segment = pitchtools.NumberedChromaticPitchClassSegment([10,
numbered_chromatic_pitch_class_segment.multiply(5)
NumberedChromaticPitchClassSegment([2, 4.5, 6, 11, 4.5, 11])
```

Return numbered chromatic pitch-class segment.

numbered_chromatic_pitch_class_set

Read-only numbered chromatic pitch-class set from numbered chromatic pitch-class segment:

```
numbered_chromatic_pitch_class_segment = pitchtools.NumberedChromaticPitchClassSegment([10,
numbered_chromatic_pitch_class_segment.numbered_chromatic_pitch_class_set
NumberedChromaticPitchClassSet([6, 7, 10, 10.5])
```

Return numbered chromatic pitch-class set.

retrograde()

Retrograde of numbered chromatic pitch-class segment:

```
numbered_chromatic_pitch_class_segment = pitchtools.NumberedChromaticPitchClassSegment([10,
numbered_chromatic_pitch_class_segment.retrograde( )
NumberedChromaticPitchClassSegment([7, 10.5, 7, 6, 10.5, 10])
```

Return numbered chromatic pitch-class segment.

rotate(*n*)

Rotate numbered chromatic pitch-class segment:

```
numbered_chromatic_pitch_class_segment = pitchtools.NumberedChromaticPitchClassSegment([10,
numbered_chromatic_pitch_class_segment.rotate(1)
NumberedChromaticPitchClassSegment([7, 10, 10.5, 6, 7, 10.5])
```

Return numbered chromatic pitch-class segment.

transpose(*n*)

Transpose numbered chromatic pitch-class segment:

```
numbered_chromatic_pitch_class_segment = pitchtools.NumberedChromaticPitchClassSegment([10,
numbered_chromatic_pitch_class_segment.transpose(10)
NumberedChromaticPitchClassSegment([8, 8.5, 4, 5, 8.5, 5])
```

Return numbered chromatic pitch-class segment.

pitchtools.NumberedChromaticPitchClassSet

class abjad.tools.pitchtools.**NumberedChromaticPitchClassSet**

Bases: abjad.tools.pitchtools._PitchClassSet._PitchClassSet._PitchClassSet

New in version 1.1.2. Abjad model of a numbered chromatic pitch-class set:

```
abjad> numbered_chromatic_pitch_class_set = pitchtools.NumberedChromaticPitchClassSet([-2, -1.5,

abjad> numbered_chromatic_pitch_class_set
NumberedChromaticPitchClassSet([6, 7, 10, 10.5])

abjad> print numbered_chromatic_pitch_class_set
{6, 7, 10, 10.5}
```

Numbered chromatic pitch-class sets are immutable.

inversion_equivalent_chromatic_interval_class_set

Read-only inversion-equivalent chromatic interval-class set:

```
abjad> numbered_chromatic_pitch_class_set = pitchtools.NumberedChromaticPitchClassSet([-2, -
abjad> numbered_chromatic_pitch_class_set.inversion_equivalent_chromatic_interval_class_set
InversionEquivalentChromaticIntervalClassSet(0.5, 1, 3, 3.5, 4, 4.5)
```

Return inversion-equivalent chromatic interval-class set.

inversion_equivalent_chromatic_interval_class_vector

Read-only inversion-equivalent chromatic interval-class vector:

```
abjad> numbered_chromatic_pitch_class_set = pitchtools.NumberedChromaticPitchClassSet([-2, -
abjad> numbered_chromatic_pitch_class_set.inversion_equivalent_chromatic_interval_class_vect
InversionEquivalentChromaticIntervalClassVector(0 | 1 0 1 1 0 0 1 0 0 1 1 0)
```

Return inversion-equivalent chromatic interval-class vector.

invert()
Invert numbered chromatic pitch-class set:

```
abjad> numbered_chromatic_pitch_class_set = pitchtools.NumberedChromaticPitchClassSet([-2, -
abjad> numbered_chromatic_pitch_class_set.invert( )
NumberedChromaticPitchClassSet([1.5, 2, 5, 6])
```

Return numbered chromatic pitch-class set.

is_transposed_subset(*pcset*)
True when self is transposed subset of *pcset*. False otherwise:

```
abjad> pcset_1 = pitchtools.NumberedChromaticPitchClassSet([-2, -1.5, 6, 7, -1.5, 7])
abjad> pcset_2 = pitchtools.NumberedChromaticPitchClassSet([-2, -1.5, 6, 7, -1.5, 7, 7.5, 8]

abjad> pcset_1.is_transposed_subset(pcset_2)
True
```

Return boolean.

is_transposed_superset(*pcset*)
True when self is transposed superset of *pcset*. False otherwise:

```
abjad> pcset_1 = pitchtools.NumberedChromaticPitchClassSet([-2, -1.5, 6, 7, -1.5, 7])
abjad> pcset_2 = pitchtools.NumberedChromaticPitchClassSet([-2, -1.5, 6, 7, -1.5, 7, 7.5, 8]

abjad> pcset_2.is_transposed_superset(pcset_1)
True
```

Return boolean.

multiply(*n*)
Multiply numbered chromatic pitch-class set by *n*:

```
abjad> numbered_chromatic_pitch_class_set = pitchtools.NumberedChromaticPitchClassSet([-2, -
abjad> numbered_chromatic_pitch_class_set.multiply(5)
NumberedChromaticPitchClassSet([2, 4.5, 6, 11])
```

Return numbered chromatic pitch-class set.

numbered_chromatic_pitch_classes
Read-only numbered chromatic pitch-classes:

```
abjad> numbered_chromatic_pitch_class_set = pitchtools.NumberedChromaticPitchClassSet([-2, -
abjad> numbered_chromatic_pitch_class_set.numbered_chromatic_pitch_classes
(NumberedChromaticPitchClass(6), NumberedChromaticPitchClass(7), NumberedChromaticPitchClass
```

Return tuple.

prime_form
To be implemented.

transpose(*n*)
Transpose numbered chromatic pitch-class set by *n*:

```
abjad> numbered_chromatic_pitch_class_set = pitchtools.NumberedChromaticPitchClassSet([-2, -
abjad> numbered_chromatic_pitch_class_set.multiply(5)
NumberedChromaticPitchClassSet([2, 4.5, 6, 11])
```

Return numbered chromatic pitch-class set.

pitchtools.NumberedChromaticPitchClassVector

class abjad.tools.pitchtools.**NumberedChromaticPitchClassVector**(*pitch_class_tokens*)

Bases: abjad.tools.pitchtools._Vector._Vector._Vector New in version 1.1.2. Abjad model of numbered chromatic pitch-class vector:

```
abjad> numbered_chromatic_pitch_class_vector = pitchtools.NumberedChromaticPitchClassVector([13,

abjad> numbered_chromatic_pitch_class_vector
NumberedChromaticPitchClassVector(0 2 0 0 0 0 | 3 0 0 0 0 0 || 0 0 3 0 0 0 | 0 0 0 0 0 0)

abjad> print numbered_chromatic_pitch_class_vector
0 2 0 0 0 0 | 3 0 0 0 0 0
0 0 3 0 0 0 | 0 0 0 0 0 0
```

Numbered chromatic pitch-class vectors are immutable.

chromatic_pitch_class_numbers

Read-only chromatic pitch-class numbers from numbered chromatic pitch-class vector:

```
abjad> numbered_chromatic_pitch_class_vector = pitchtools.NumberedChromaticPitchClassVector(
abjad> numbered_chromatic_pitch_class_vector.chromatic_pitch_class_numbers
[1, 2.5, 6]
```

Return list.

numbered_chromatic_pitch_classes

Read-only numbered chromatic pitch-classes from numbered chromatic pitch-class vector:

```
abjad> numbered_chromatic_pitch_class_vector = pitchtools.NumberedChromaticPitchClassVector(
abjad> numbered_chromatic_pitch_class_vector.numbered_chromatic_pitch_classes
[NumberedChromaticPitchClass(2.5), NumberedChromaticPitchClass(1), NumberedChromaticPitchCla
```

Return list.

pitchtools.NumberedDiatonicPitch

class abjad.tools.pitchtools.**NumberedDiatonicPitch**

Bases: abjad.tools.pitchtools._DiatonicPitch._DiatonicPitch._DiatonicPitch, abjad.tools.pitchtools._NumberedPitch._NumberedPitch._NumberedPitch New in version 1.1.2. Abjad model of a numbered diatonic pitch:

```
abjad> pitchtools.NumberedDiatonicPitch(7)
NumberedDiatonicPitch(7)
```

Numbered diatonic pitches are immutable.

chromatic_pitch_number

Read-only chromatic pitch number:

```
abjad> pitchtools.NumberedDiatonicPitch(7).chromatic_pitch_number
12
```

Return integer.

diatonic_pitch_number

Read-only diatonic pitch number:

```
abjad> pitchtools.NumberedDiatonicPitch(7).diatonic_pitch_number
7
```

Return integer.

named_diatonic_pitch

Read-only named diatonic pitch:

```
abjad> pitchtools.NumberedDiatonicPitch(7).named_diatonic_pitch
NamedDiatonicPitch("c''")
```

Return named diatonic pitch.

named_diatonic_pitch_class

Read-only named diatonic pitch-class:

```
abjad> pitchtools.NumberedDiatonicPitch(7).named_diatonic_pitch_class
NamedDiatonicPitchClass('c')
```

Return named diatonic pitch-class.

numbered_diatonic_pitch_class

Read-only numbered diatonic pitch-class:

```
abjad> pitchtools.NumberedDiatonicPitch(7).numbered_diatonic_pitch_class
NumberedDiatonicPitchClass(0)
```

Return numbered diatonic pitch-class.

pitchtools.NumberedDiatonicPitchClass

class `abjad.tools.pitchtools.`**`NumberedDiatonicPitchClass`**

Bases: `abjad.tools.pitchtools._NumberedPitchClass._NumberedPitchClass._NumberedPitchClass`, `abjad.tools.pitchtools._DiatonicPitchClass._DiatonicPitchClass._DiatonicPitchClass`

New in version 1.1.2. Abjad model of a numbered diatonic pitch-class:

```
abjad> pitchtools.NumberedDiatonicPitchClass(0)
NumberedDiatonicPitchClass(0)
```

Numbered diatonic pitch-classes are immutable.

named_diatonic_pitch_class

Read-only named diatonic pitch-class from numbered diatonic pitch-class:

```
abjad> numbered_diatonic_pitch_class = pitchtools.NumberedDiatonicPitchClass(0)
abjad> numbered_diatonic_pitch_class.named_diatonic_pitch_class
NamedDiatonicPitchClass('c')
```

Return named diatonic pitch-class.

pitchtools.PitchRange

class `abjad.tools.pitchtools.`**`PitchRange`**(**args*)

Bases: `abjad.core._Immutable._Immutable._Immutable` New in version 1.1.2. Abjad model of pitch range:

```
abjad> pitchtools.PitchRange(-12, 36)
PitchRange((NamedChromaticPitch('c'), 'inclusive'), (NamedChromaticPitch("c''''"), 'inclusive'))
```

Init from pitch numbers, pitch instances or other pitch range objects.

Pitch ranges implement all six Python rich comparators.

Pitch ranges are immutable.

start_pitch

Read-only start pitch of range:

```
abjad> pitch_range = pitchtools.PitchRange(-12, 36)
abjad> pitch_range.start_pitch
NamedChromaticPitch('c')
```

Return pitch.

start_pitch_is_included_in_range

True when start pitch is included in range. Otherwise false:

```
abjad> pitch_range = pitchtools.PitchRange(-12, 36)
abjad> pitch_range.start_pitch_is_included_in_range
True
```

Return boolean.

stop_pitch

Read-only stop pitch of range:

```
abjad> pitch_range = pitchtools.PitchRange(-12, 36)
abjad> pitch_range.stop_pitch
NamedChromaticPitch("c''''")
```

Return pitch.

stop_pitch_is_included_in_range

True when stop pitch is included in range. Otherwise false:

```
abjad> pitch_range = pitchtools.PitchRange(-12, 36)
abjad> pitch_range.stop_pitch_is_included_in_range
True
```

Return boolean.

pitchtools.TwelveToneRow

class abjad.tools.pitchtools.**TwelveToneRow**

Bases: abjad.tools.pitchtools.NumberedChromaticPitchClassSegment.NumberedChromaticPitchCl

New in version 1.1.2. Abjad model of twelve-tone row:

```
abjad> pitchtools.TwelveToneRow([0, 1, 11, 9, 3, 6, 7, 5, 4, 10, 2, 8])
TwelveToneRow([0, 1, 11, 9, 3, 6, 7, 5, 4, 10, 2, 8])
```

Twelve-tone rows validate pitch-classes at initialization.

Twelve-tone rows inherit canonical operators from numbered chromatic pitch-class segment.

Twelve-tone rows return numbered chromatic pitch-class segments on calls to getslice.

Twelve-tone rows are immutable.

pitchtools.all_are_chromatic_pitch_class_name_octave_number_pairs

`abjad.tools.pitchtools.`**`all_are_chromatic_pitch_class_name_octave_number_pairs`**`(`*expr*`)`

New in version 1.1.1. True when all elements of *expr* are pitch tokens. Otherwise false:

```
abjad> pitchtools.all_are_chromatic_pitch_class_name_octave_number_pairs([('c', 4), ('d', 4), pi
True
```

Return boolean. Changed in version 1.1.2: renamed `pitchtools.is_pitch_token_collection( )` to `pitchtools.all_are_chromatic_pitch_class_name_octave_number_pairs( )`.

pitchtools.apply_accidental_to_named_chromatic_pitch

`abjad.tools.pitchtools.`**`apply_accidental_to_named_chromatic_pitch`**`(`*named_chromatic_pitch*, *acciden-tal=None*`)`

New in version 1.1.2. Apply *accidental* to *named_chromatic_pitch*:

```
abjad> pitch = pitchtools.NamedChromaticPitch("cs''")
abjad> pitchtools.apply_accidental_to_named_chromatic_pitch(pitch, 'f')
NamedChromaticPitch("c''")
```

Return new named pitch.

pitchtools.apply_octavation_spanner_to_pitched_components

`abjad.tools.pitchtools.`**`apply_octavation_spanner_to_pitched_components`**`(`*expr*, *ot-tava_numbered_diatonic_pitch quin-de-cisima_numbered_diatonic_pi*

New in version 1.1.1. Apply octavation spanner to pitched components in *expr*:

```
abjad> t = Measure((4, 8), notetools.make_notes([24, 26, 27, 29], [(1, 8)]))
abjad> pitchtools.apply_octavation_spanner_to_pitched_components(t, ottava_numbered_diatonic_pit
OctavationSpanner(|4/8(4)|)

abjad> print t.format
   {
      \time 4/8
      \ottava #1
      c'''8
      d'''8
      ef'''8
      f'''8
      \ottava #0
   }
```

Apply octavation spanner according to the diatonic pitch number of the maximum pitch in *expr*.

Return octavation spanner.

pitchtools.calculate_harmonic_chromatic_interval_class_from_pitch_carrier_to_pitch_carrier

abjad.tools.pitchtools.**calculate_harmonic_chromatic_interval_class_from_pitch_carrier_to_pi**

New in version 1.1.2. Calculate harmonic chromatic interval-class from *pitch_carrier_1* to *pitch_carrier_2*:

```
abjad> pitchtools.calculate_harmonic_chromatic_interval_class_from_pitch_carrier_to_pitch_carrie
HarmonicChromaticIntervalClass(2)
```

Return harmonic chromatic interval-class.

pitchtools.calculate_harmonic_chromatic_interval_from_pitch_carrier_to_pitch_carrier

abjad.tools.pitchtools.**calculate_harmonic_chromatic_interval_from_pitch_carrier_to_pitch_ca**

New in version 1.1.2. Calculate harmonic chromatic interval from *pitch_carrier_1* to *pitch_carrier_2*:

```
abjad> pitchtools.calculate_harmonic_chromatic_interval_from_pitch_carrier_to_pitch_carrier(pitc
HarmonicChromaticInterval(14)
```

Return harmonic chromatic interval.

pitchtools.calculate_harmonic_counterpoint_interval_class_from_named_chromatic_pitch_to_named_chromatic_

abjad.tools.pitchtools.**calculate_harmonic_counterpoint_interval_class_from_named_chromatic_**

New in version 1.1.2. Calculate harmonic counterpoint interval-class from *pitch_carrier_1* to *pitch_carrier_2*:

```
abjad> pitchtools.calculate_harmonic_counterpoint_interval_class_from_named_chromatic_pitch_to_n
HarmonicCounterpointIntervalClass(2)
```

Return harmonic counterpoint interval-class. Changed in version 1.1.2: renamed `pitchtools.calculate_harmonic_counterpoint_interval_class_from_named_pchromatic_pitch_t` `)` to `pitchtools.calculate_harmonic_counterpoint_interval_class_from_named_chromatic_pitch` `).`

pitchtools.calculate_harmonic_counterpoint_interval_from_named_chromatic_pitch_to_named_chromatic_pitch

abjad.tools.pitchtools.**calculate_harmonic_counterpoint_interval_from_named_chromatic_pitch_**

New in version 1.1.2. Calculate harmonic counterpoint interval *pitch_carrier_1* to *pitch_carrier_2*:

```
abjad> pitchtools.calculate_harmonic_counterpoint_interval_from_named_chromatic_pitch_to_named_c
HarmonicCounterpointInterval(9)
```

Return harmonic counterpoint interval-class.

pitchtools.calculate_harmonic_diatonic_interval_class_from_named_chromatic_pitch_to_named_chromatic_pitch

abjad.tools.pitchtools.**calculate_harmonic_diatonic_interval_class_from_named_chromatic_pitc**

New in version 1.1.2. Calculate harmonic diatonic interval-class from *pitch_carrier_1* to *pitch_carrier_2*:

```
abjad> pitchtools.calculate_harmonic_diatonic_interval_class_from_named_chromatic_pitch_to_named
HarmonicDiatonicIntervalClass('M2')
```

Return harmonic diatonic interval-class.

pitchtools.calculate_harmonic_diatonic_interval_from_named_chromatic_pitch_to_named_chromatic_pitch

abjad.tools.pitchtools.**calculate_harmonic_diatonic_interval_from_named_chromatic_pitch_to_n**

New in version 1.1.2. Calculate harmonic diatonic interval from *pitch_carrier_1* to *pitch_carrier_2*:

```
abjad> pitchtools.calculate_harmonic_diatonic_interval_from_named_chromatic_pitch_to_named_chrom
HarmonicDiatonicInterval('M9')
```

Return harmonic diatonic interval.

pitchtools.calculate_melodic_chromatic_interval_class_from_pitch_carrier_to_pitch_carrier

abjad.tools.pitchtools.**calculate_melodic_chromatic_interval_class_from_pitch_carrier_to_pit**

New in version 1.1.2. Calculate melodic chromatic interval-class from *pitch_carrier_1* to *pitch_carrier_2*:

```
abjad> pitchtools.calculate_melodic_chromatic_interval_class_from_pitch_carrier_to_pitch_carrier
MelodicChromaticIntervalClass(+2)
```

Return melodic chromatic interval-class.

pitchtools.calculate_melodic_chromatic_interval_from_pitch_carrier_to_pitch_carrier

abjad.tools.pitchtools.**calculate_melodic_chromatic_interval_from_pitch_carrier_to_pitch_ca**

New in version 1.1.2. Calculate melodic chromatic interval from *pitch_carrier_1* to *pitch_carrier_2*:

```
abjad> pitchtools.calculate_melodic_chromatic_interval_from_pitch_carrier_to_pitch_carrier(pitch
MelodicChromaticInterval(+14)
```

Return melodic chromatic interval.

pitchtools.calculate_melodic_counterpoint_interval_class_from_named_chromatic_pitch_to_named_chromatic_pi

abjad.tools.pitchtools.**calculate_melodic_counterpoint_interval_class_from_named_chromatic_p**

New in version 1.1.2. Calculate melodic counterpoint interval-class from *pitch_carrier_1* to *pitch_carrier_2*:

```
abjad> pitchtools.calculate_melodic_counterpoint_interval_class_from_named_chromatic_pitch_to_na
MelodicCounterpointIntervalClass(+2)
```

Return melodic counterpoint interval-class.

pitchtools.calculate_melodic_counterpoint_interval_from_named_chromatic_pitch_to_named_chromatic_pitch

abjad.tools.pitchtools.**calculate_melodic_counterpoint_interval_from_named_chromatic_pitch_t**

New in version 1.1.2. Calculate melodic counterpoint interval *pitch_carrier_1* to *pitch_carrier_2*:

```
abjad> pitchtools.calculate_melodic_counterpoint_interval_from_named_chromatic_pitch_to_named_ch
MelodicCounterpointInterval(+9)
```

Return melodic counterpoint interval.

pitchtools.calculate_melodic_diatonic_interval_class_from_named_chromatic_pitch_to_named_chromatic_pitch

`abjad.tools.pitchtools.`**`calculate_melodic_diatonic_interval_class_from_named_chromatic_pitc`**

New in version 1.1.2. Calculate melodic diatonic interval-class from *pitch_carrier_1* to *pitch_carrier_2*:

```
abjad> pitchtools.calculate_melodic_diatonic_interval_class_from_named_chromatic_pitch_to_named_
MelodicDiatonicIntervalClass('+M2')
```

Return melodic diatonic interval-class.

pitchtools.calculate_melodic_diatonic_interval_from_named_chromatic_pitch_to_named_chromatic_pitch

`abjad.tools.pitchtools.`**`calculate_melodic_diatonic_interval_from_named_chromatic_pitch_to_na`**

New in version 1.1.2. Calculate melodic diatonic interval from *pitch_carrier_1* to *pitch_carrier_2*:

```
abjad> pitchtools.calculate_melodic_diatonic_interval_from_named_chromatic_pitch_to_named_chroma
MelodicDiatonicInterval('+M9')
```

Return melodic diatonic interval.

pitchtools.chromatic_pitch_class_name_to_chromatic_pitch_class_number

`abjad.tools.pitchtools.`**`chromatic_pitch_class_name_to_chromatic_pitch_class_number`**(*chromatic_p*

New in version 1.1.2. Change *chromatic_pitch_class_name* to chromatic pitch-class number:

```
abjad> pitchtools.chromatic_pitch_class_name_to_chromatic_pitch_class_number('cs')
1
```

Return chromatic pitch-class number.

pitchtools.chromatic_pitch_class_name_to_diatonic_pitch_class_name

`abjad.tools.pitchtools.`**`chromatic_pitch_class_name_to_diatonic_pitch_class_name`**(*chromatic_pitch_*

New in version 1.1.2. Change *chromatic_pitch_class_name* to diatonic pitch-class name:

```
abjad> pitchtools.chromatic_pitch_class_name_to_diatonic_pitch_class_name('cs')
'c'
```

Return string.

pitchtools.chromatic_pitch_class_name_to_diatonic_pitch_class_name_alphabetic_accidental_abbreviation_pair

`abjad.tools.pitchtools.`**`chromatic_pitch_class_name_to_diatonic_pitch_class_name_alphabetic_a`**

New in version 1.1.1. Change *chromatic_pitch_class_name* to diatonic pitch-class name / alphabetic accidental abbreviation pair:

```
abjad> pitchtools.chromatic_pitch_class_name_to_diatonic_pitch_class_name_alphabetic_accidental_
('c', 's')
```

Return pair of strings. Changed in version 1.1.2: renamed `pitchtools.name_to_letter_accidental( )` to `pitchtools.chromatic_pitch_class_name_to_diatonic_pitch_class_name_alphabetic_accide`
`).`

pitchtools.chromatic_pitch_class_number_to_chromatic_pitch_class_name

`abjad.tools.pitchtools.`**`chromatic_pitch_class_number_to_chromatic_pitch_class_name`**(*chromatic_p*
New in version 1.1.1. Change *chromatic_pitch_class_number* to chromatic pitch-class name:

```
abjad> for n in range(0, 13):
...     pc = n / 2.0
...     pitch_name_string = pitchtools.chromatic_pitch_class_number_to_chromatic_pitch_class_nam
...     print '%s   %s' % (pc, pitch_name_string)
...
0.0   c
0.5   cqs
1.0   cs
1.5   dqf
2.0   d
2.5   dqs
3.0   ef
3.5   eqf
4.0   e
4.5   eqs
5.0   f
5.5   fqs
6.0   fs
```

Return string. Changed in version 1.1.2: renamed `pitchtools.pc_to_pitch_name( )` to `pitchtools.chromatic_pitch_class_number_to_chromatic_pitch_class_name( )`.

pitchtools.chromatic_pitch_class_number_to_chromatic_pitch_class_name_with_flats

`abjad.tools.pitchtools.`**`chromatic_pitch_class_number_to_chromatic_pitch_class_name_with_flat`**
New in version 1.1.1. Change chromatic pitch-class number to chromatic pitch-class name with flats:

```
abjad> for n in range(13):
...     pc = n / 2.0
...     name = pitchtools.chromatic_pitch_class_number_to_chromatic_pitch_class_name_with_flats(
...     print '%s   %s' % (pc, name)
...
0.0   c
0.5   dtqf
1.0   df
1.5   dqf
2.0   d
2.5   etqf
3.0   ef
3.5   eqf
4.0   e
4.5   fqf
5.0   f
5.5   gtqf
6.0   gf
```

Return string. Changed in version 1.1.2: renamed `pitchtools.pc_to_pitch_name_flats( )` to `pitchtools.chromatic_pitch_class_number_to_chromatic_pitch_class_name_with_flats(`

).

pitchtools.chromatic_pitch_class_number_to_chromatic_pitch_class_name_with_sharps

abjad.tools.pitchtools.**chromatic_pitch_class_number_to_chromatic_pitch_class_name_with_shar**
New in version 1.1.1. Change *chromatic_pitch_class_number* to chromatic pitch-class name with sharps:

```
abjad> for n in range(13):
...     pc = n / 2.0
...     name = pitchtools.chromatic_pitch_class_number_to_chromatic_pitch_class_name_with_sharps
...     print '%s   %s' % (pc, name)
...
0.0   c
0.5   cqs
1.0   cs
1.5   ctqs
2.0   d
2.5   dqs
3.0   ds
3.5   dtqs
4.0   e
4.5   eqs
5.0   f
5.5   fqs
6.0   fs
```

Return string. Changed in version 1.1.2: renamed `pitchtools.pc_to_pitch_name_sharps( )` to `pitchtools.chromatic_pitch_class_number_to_chromatic_pitch_class_name_with_sharps( )`.

pitchtools.chromatic_pitch_class_number_to_diatonic_pitch_class_number

abjad.tools.pitchtools.**chromatic_pitch_class_number_to_diatonic_pitch_class_number**(*chromatic_*
New in version 1.1.2. Change *chromatic_pitch_class_number* to diatonic pitch-class number:

```
abjad> pitchtools.chromatic_pitch_class_number_to_diatonic_pitch_class_number(1)
0
```

Return integer.

pitchtools.chromatic_pitch_name_to_chromatic_pitch_class_name

abjad.tools.pitchtools.**chromatic_pitch_name_to_chromatic_pitch_class_name**(*chromatic_pitch_name*)
New in version 1.1.2. Change *chromatic_pitch_name* to chromatic pitch-class name:

```
abjad> pitchtools.chromatic_pitch_name_to_chromatic_pitch_class_name("cs''")
'cs'
```

Return string.

pitchtools.chromatic_pitch_name_to_chromatic_pitch_class_number

abjad.tools.pitchtools.**chromatic_pitch_name_to_chromatic_pitch_class_number**(*chromatic_pitch_nam*
New in version 1.1.2. Change *chromatic_class_name* to chromatic pitch-class-number:

```
abjad> pitchtools.chromatic_pitch_name_to_chromatic_pitch_class_number("cs''")
1
```

Return integer or float.

pitchtools.chromatic_pitch_name_to_chromatic_pitch_number

abjad.tools.pitchtools.**chromatic_pitch_name_to_chromatic_pitch_number**(*chromatic_pitch_name*)
New in version 1.1.2. Change *chromatic_pitch_name* to chromatic pitch number:

```
abjad> pitchtools.chromatic_pitch_name_to_chromatic_pitch_number("cs''")
13
```

Return integer or float.

pitchtools.chromatic_pitch_name_to_diatonic_pitch_class_name

abjad.tools.pitchtools.**chromatic_pitch_name_to_diatonic_pitch_class_name**(*chromatic_pitch_name*)
New in version 1.1.2. Change *chromatic_pitch_name* to diatonic pitch name:

```
abjad> pitchtools.chromatic_pitch_name_to_diatonic_pitch_class_name("cs''")
'c'
```

Return string.

pitchtools.chromatic_pitch_name_to_diatonic_pitch_class_number

abjad.tools.pitchtools.**chromatic_pitch_name_to_diatonic_pitch_class_number**(*chromatic_pitch_name*
New in version 1.1.2. Change *chromatic_pitch_name* to diatonic pitch-class number:

```
abjad> pitchtools.chromatic_pitch_name_to_diatonic_pitch_class_number("cs''")
0
```

Return integer.

pitchtools.chromatic_pitch_name_to_diatonic_pitch_name

abjad.tools.pitchtools.**chromatic_pitch_name_to_diatonic_pitch_name**(*chromatic_pitch_name*)
New in version 1.1.2. Change *chromatic_pitch_name* to diatonic pitch name:

```
abjad> pitchtools.chromatic_pitch_name_to_diatonic_pitch_name("cs''")
"c''"
```

Return string.

pitchtools.chromatic_pitch_name_to_diatonic_pitch_number

abjad.tools.pitchtools.**chromatic_pitch_name_to_diatonic_pitch_number**(*chromatic_pitch_name*)
New in version 1.1.2. Change *chromatic_pitch_name* to diatonic pitch number:

```
abjad> pitchtools.chromatic_pitch_name_to_diatonic_pitch_number("cs''")
7
```

Return integer.

pitchtools.chromatic_pitch_name_to_octave_number

abjad.tools.pitchtools.**chromatic_pitch_name_to_octave_number**(*chromatic_pitch_name*)

New in version 1.1.2. Change *chromatic_pitch_name* to octave number:

```
abjad> pitchtools.chromatic_pitch_name_to_octave_number('cs')
3
```

Return integer.

pitchtools.chromatic_pitch_names_string_to_named_chromatic_pitch_list

abjad.tools.pitchtools.**chromatic_pitch_names_string_to_named_chromatic_pitch_list**(*chromatic_p*

New in version 1.1.2. Change *chromatic_pitch_names_string* to named chromatic pitch list:

```
abjad> pitchtools.chromatic_pitch_names_string_to_named_chromatic_pitch_list("cs, cs cs' cs''")
[NamedChromaticPitch('cs,'), NamedChromaticPitch('cs'), NamedChromaticPitch("cs'"), NamedChromat
```

Return list of named chromatic pitches.

pitchtools.chromatic_pitch_number_and_accidental_semitones_to_octave_number

abjad.tools.pitchtools.**chromatic_pitch_number_and_accidental_semitones_to_octave_number**(*chr*
ac-
ci-
den
*tal_*

New in version 1.1.1. Change *chromatic_pitch_number* and *accidental_semitones* to octave number:

```
abjad> pitchtools.chromatic_pitch_number_and_accidental_semitones_to_octave_number(12, -2)
5
```

Return integer. Changed in version 1.1.2: renamed `pitchtools.pitch_number_and_accidental_semitones_to_o` `)` to `pitchtools.chromatic_pitch_number_and_accidental_semitones_to_octave_number(` `)`.

pitchtools.chromatic_pitch_number_diatonic_pitch_class_name_to_alphabetic_accidental_abbreviation_octave_n

abjad.tools.pitchtools.**chromatic_pitch_number_diatonic_pitch_class_name_to_alphabetic_accid**

New in version 1.1.1. Change *chromatic_pitch_number* and *diatonic_pitch_class_name* to alphabetic accidental abbreviation / octave number pair:

```
abjad> pitchtools.chromatic_pitch_number_diatonic_pitch_class_name_to_alphabetic_accidental_abbr
('ss', 5)
```

Return pair. Changed in version 1.1.2: renamed `pitchtools.number_letter_to_accidental_octave(` `)` to `pitchtools.chromatic_pitch_number_diatonic_pitch_class_name_to_alphabetic_accidenta` `)`.

pitchtools.chromatic_pitch_number_to_chromatic_pitch_class_number

`abjad.tools.pitchtools.`**`chromatic_pitch_number_to_chromatic_pitch_class_number`**(*chromatic_pitch_n*

New in version 1.1.2. Change *chromatic_pitch_number* to chromatic pitch-class number:

```
abjad> pitchtools.chromatic_pitch_number_to_chromatic_pitch_class_number(13)
1
```

Return integer or float.

pitchtools.chromatic_pitch_number_to_chromatic_pitch_name

`abjad.tools.pitchtools.`**`chromatic_pitch_number_to_chromatic_pitch_name`**(*chromatic_pitch_number, accidental_spelling='mixed'*)

New in version 1.1.2. Change *chromatic_pitch_number* to chromatic pitch name:

```
abjad> pitchtools.chromatic_pitch_number_to_chromatic_pitch_name(13)
"cs''"
```

Return string.

pitchtools.chromatic_pitch_number_to_diatonic_pitch_class_name_alphabetic_accidental_abbreviation_octave_n

`abjad.tools.pitchtools.`**`chromatic_pitch_number_to_diatonic_pitch_class_name_alphabetic_acci`**

Change *chromatic_pitch_number* to diatonic pitch-class name / alphabetic accidental abbreviation / octave number triple:

```
abjad> pitchtools.chromatic_pitch_number_to_diatonic_pitch_class_name_alphabetic_accidental_abbr
('c', 's', 5)
```

Return tuple. Changed in version 1.1.2: renamed `pitchtools.number_to_letter_accidental_octave( )` to `pitchtools.chromatic_pitch_number_to_diatonic_pitch_class_name_alphabetic_accidenta )`.

pitchtools.chromatic_pitch_number_to_diatonic_pitch_class_number

`abjad.tools.pitchtools.`**`chromatic_pitch_number_to_diatonic_pitch_class_number`**(*chromatic_pitch_nu*

New in version 1.1.2. Change *chromatic_pitch_number* to diatonic pitch-class number:

```
abjad> pitchtools.chromatic_pitch_number_to_diatonic_pitch_class_number(13)
0
```

Return integer.

pitchtools.chromatic_pitch_number_to_diatonic_pitch_number

`abjad.tools.pitchtools.`**`chromatic_pitch_number_to_diatonic_pitch_number`**(*chromatic_pitch_number*)

New in version 1.1.2. Change *chromatic_pitch_number* to diatonic pitch number:

```
abjad> pitchtools.chromatic_pitch_number_to_diatonic_pitch_number(13)
7
```

Return integer.

pitchtools.chromatic_pitch_number_to_octave_number

abjad.tools.pitchtools.**chromatic_pitch_number_to_octave_number**(*chromatic_pitch_number*)

New in version 1.1.1. Change *chromatic_pitch_number* to octave number:

```
abjad> pitchtools.chromatic_pitch_number_to_octave_number(13)
5
```

Return integer. Changed in version 1.1.2: renamed `pitchtools.pitch_number_to_octave( )` to `pitchtools.chromatic_pitch_number_to_octave_number( )`.

pitchtools.clef_and_staff_position_number_to_named_chromatic_pitch

abjad.tools.pitchtools.**clef_and_staff_position_number_to_named_chromatic_pitch**(*clef*, *staff_position_nu*

New in version 1.1.2. Change *clef* and *staff_position_number* to named chromatic pitch:

```
abjad> clef = contexttools.ClefMark('treble')
abjad> for n in range(-6, 6):
...   pitch = pitchtools.clef_and_staff_position_number_to_named_chromatic_pitch(clef, n)
...   print '%s\t%s\t%s' % (clef.clef_name_string, n, pitch)
treble   -6 c'
treble   -5 d'
treble   -4 e'
treble   -3 f'
treble   -2 g'
treble   -1 a'
treble   0  b'
treble   1  c''
treble   2  d''
treble   3  e''
treble   4  f''
treble   5  g''
```

Return named chromatic pitch.

pitchtools.diatonic_interval_number_and_chromatic_interval_number_to_melodic_diatonic_interval

abjad.tools.pitchtools.**diatonic_interval_number_and_chromatic_interval_number_to_melodic_di**

New in version 1.1.2. Change *diatonic_interval_number* and *chromatic_interval_number* to melodic diatonic interval:

```
abjad> pitchtools.diatonic_interval_number_and_chromatic_interval_number_to_melodic_diatonic_int
MelodicDiatonicInterval('+m2')
```

Return melodic diatonic interval.

pitchtools.diatonic_pitch_class_name_to_chromatic_pitch_class_number

abjad.tools.pitchtools.**diatonic_pitch_class_name_to_chromatic_pitch_class_number**(*diatonic_pitch*
New in version 1.1.1. Change *diatonic_pitch_class_name* to chromatic pitch-class number:

```
abjad> pitchtools.diatonic_pitch_class_name_to_chromatic_pitch_class_number('f')
5
```

Return integer.

pitchtools.diatonic_pitch_class_name_to_diatonic_pitch_class_number

abjad.tools.pitchtools.**diatonic_pitch_class_name_to_diatonic_pitch_class_number**(*diatonic_pitch_*
New in version 1.1.2. Change *diatonic_pitch_class_name* to diatonic pitch-class number:

```
abjad> pitchtools.diatonic_pitch_class_name_to_diatonic_pitch_class_number('c')
0
```

Return integer.

pitchtools.diatonic_pitch_class_number_to_chromatic_pitch_class_number

abjad.tools.pitchtools.**diatonic_pitch_class_number_to_chromatic_pitch_class_number**(*diatonic_pi*
New in version 1.1.2. Change *diatonic_pitch_class_number* to chromatic pitch-class number:

```
abjad> pitchtools.diatonic_pitch_class_number_to_chromatic_pitch_class_number(6)
11
```

Return nonnegative integer.

pitchtools.diatonic_pitch_class_number_to_diatonic_pitch_class_name

abjad.tools.pitchtools.**diatonic_pitch_class_number_to_diatonic_pitch_class_name**(*diatonic_pitch_*
New in version 1.1.2. Change *diatonic_pitch_class_number* to diatonic pitch-class name:

```
abjad> pitchtools.diatonic_pitch_class_number_to_diatonic_pitch_class_name(0)
'c'
```

Return string.

pitchtools.diatonic_pitch_name_to_chromatic_pitch_class_name

abjad.tools.pitchtools.**diatonic_pitch_name_to_chromatic_pitch_class_name**(*diatonic_pitch_name*)
New in version 1.1.2. Change *diatonic_pitch_name* to chromatic pitch-class name:

```
abjad> pitchtools.diatonic_pitch_name_to_chromatic_pitch_class_name("c''")
'c'
```

Return string.

pitchtools.diatonic_pitch_name_to_chromatic_pitch_class_number

abjad.tools.pitchtools.**diatonic_pitch_name_to_chromatic_pitch_class_number**(*diatonic_pitch_name*)
New in version 1.1.2. Change *diatonic_pitch_name* to chromatic pitch-class number:

```
abjad> pitchtools.diatonic_pitch_name_to_chromatic_pitch_class_number("c''")
0
```

Return integer.

pitchtools.diatonic_pitch_name_to_chromatic_pitch_name

abjad.tools.pitchtools.**diatonic_pitch_name_to_chromatic_pitch_name**(*diatonic_pitch_name*)
New in version 1.1.2. Change *diatonic_pitch_name* to chromatic pitch name:

```
abjad> pitchtools.diatonic_pitch_name_to_chromatic_pitch_name("c''")
"c''"
```

Return string.

pitchtools.diatonic_pitch_name_to_chromatic_pitch_number

abjad.tools.pitchtools.**diatonic_pitch_name_to_chromatic_pitch_number**(*diatonic_pitch_name*)
New in version 1.1.2. Change *diatonic_pitch_name* to chromatic pitch number:

```
abjad> pitchtools.diatonic_pitch_name_to_chromatic_pitch_number("c''")
12
```

Return integer.

pitchtools.diatonic_pitch_name_to_diatonic_pitch_class_name

abjad.tools.pitchtools.**diatonic_pitch_name_to_diatonic_pitch_class_name**(*diatonic_pitch_name*)
New in version 1.1.2. Change *diatonic_pitch_name* to diatonic pitch-class name:

```
abjad> pitchtools.diatonic_pitch_name_to_diatonic_pitch_class_name("c''")
'c'
```

Return string.

pitchtools.diatonic_pitch_name_to_diatonic_pitch_class_number

abjad.tools.pitchtools.**diatonic_pitch_name_to_diatonic_pitch_class_number**(*diatonic_pitch_name*)
New in version 1.1.2. Change *diatonic_pitch_name* to diatonic pitch-class number:

```
abjad> pitchtools.diatonic_pitch_name_to_diatonic_pitch_class_number("c''")
0
```

Return integer.

pitchtools.diatonic_pitch_name_to_diatonic_pitch_number

abjad.tools.pitchtools.**diatonic_pitch_name_to_diatonic_pitch_number**(*diatonic_pitch_name*)
New in version 1.1.2. Change *diatonic_pitch_name* to diatonic pitch number:

```
abjad> pitchtools.diatonic_pitch_name_to_diatonic_pitch_number("c''")
7
```

Return integer.

pitchtools.diatonic_pitch_number_to_chromatic_pitch_number

abjad.tools.pitchtools.**diatonic_pitch_number_to_chromatic_pitch_number**(*diatonic_pitch_number*)

New in version 1.1.2. Change *diatonic_pitch_number* to chromatic pitch number:

```
abjad> pitchtools.diatonic_pitch_number_to_chromatic_pitch_number(7)
12
```

Return integer.

pitchtools.diatonic_pitch_number_to_diatonic_pitch_class_name

abjad.tools.pitchtools.**diatonic_pitch_number_to_diatonic_pitch_class_name**(*diatonic_pitch_number*)

New in version 1.1.2. Change *diatonic_pitch_number* to diatonic pitch-class name:

```
abjad> pitchtools.diatonic_pitch_number_to_diatonic_pitch_class_name(7)
'c'
```

Return string.

pitchtools.diatonic_pitch_number_to_diatonic_pitch_class_number

abjad.tools.pitchtools.**diatonic_pitch_number_to_diatonic_pitch_class_number**(*diatonic_pitch_numbe*

New in version 1.1.2. Change *diatonic_pitch_number* to diatonic pitch-class number:

```
abjad> pitchtools.diatonic_pitch_number_to_diatonic_pitch_class_number(7)
0
```

Return nonnegative integer.

pitchtools.diatonic_pitch_number_to_diatonic_pitch_name

abjad.tools.pitchtools.**diatonic_pitch_number_to_diatonic_pitch_name**(*diatonic_pitch_number*)

New in version 1.1.2. Change *diatonic_pitch_number* to diatonic pitch name:

```
abjad> pitchtools.diatonic_pitch_number_to_diatonic_pitch_name(7)
"c''"
```

Return string.

pitchtools.expr_has_duplicate_named_chromatic_pitch

abjad.tools.pitchtools.**expr_has_duplicate_named_chromatic_pitch**(*expr*)

New in version 1.1.2. True when *expr* has duplicate named chromatic pitch. Otherwise false:

```
abjad> chord = Chord([13, 13, 14], (1, 4))
abjad> pitchtools.expr_has_duplicate_named_chromatic_pitch(chord)
True
```

Return boolean.

pitchtools.expr_has_duplicate_numbered_chromatic_pitch_class

abjad.tools.pitchtools.**expr_has_duplicate_numbered_chromatic_pitch_class**(*expr*)

New in version 1.1.2. True when *expr* has duplicate numbered chromatic pitch-class. Otherwise false:

```
abjad> chord = Chord([1, 13, 14], (1, 4))
abjad> pitchtools.expr_has_duplicate_numbered_chromatic_pitch_class(chord)
True
```

Return boolean. Changed in version 1.1.2: renamed `pitchtools.expr_has_duplicate_numeric_chromatic_pitc` `)` to `pitchtools.expr_has_duplicate_numbered_chromatic_pitch_class( )`.

pitchtools.expr_to_melodic_chromatic_interval_segment

abjad.tools.pitchtools.**expr_to_melodic_chromatic_interval_segment**(*expr*)

New in version 1.1.2. Change *expr* to melodic chromatic interval segment:

```
abjad> staff = Staff(macros.scale(8))
abjad> pitchtools.expr_to_melodic_chromatic_interval_segment(staff)
MelodicChromaticIntervalSegment(+2, +2, +1, +2, +2, +2, +1)
```

Return melodic chromatic interval segment.

pitchtools.get_named_chromatic_pitch_from_pitch_carrier

abjad.tools.pitchtools.**get_named_chromatic_pitch_from_pitch_carrier**(*pitch_carrier*)

New in version 1.1.1. Get named chromatic pitch from *pitch_carrier*:

```
abjad> pitch = pitchtools.NamedChromaticPitch('df', 5)
abjad> pitch
NamedChromaticPitch("df''")
abjad> pitchtools.get_named_chromatic_pitch_from_pitch_carrier(pitch)
NamedChromaticPitch("df''")

abjad> note = Note(('df', 5), (1, 4))
abjad> note
Note("df''4")
abjad> pitchtools.get_named_chromatic_pitch_from_pitch_carrier(note)
NamedChromaticPitch("df''")

abjad> note = Note(('df', 5), (1, 4))
abjad> note.note_head
NoteHead("df''")
abjad> pitchtools.get_named_chromatic_pitch_from_pitch_carrier(note.note_head)
NamedChromaticPitch("df''")

abjad> chord = Chord([('df', 5)], (1, 4))
abjad> chord
Chord("<df''>4")
abjad> pitchtools.get_named_chromatic_pitch_from_pitch_carrier(chord)
NamedChromaticPitch("df''")

abjad> pitchtools.get_named_chromatic_pitch_from_pitch_carrier(13)
NamedChromaticPitch("cs''")
```

Raise missing pitch error when *pitch_carrier* carries no pitch.

Raise extra pitch error when *pitch_carrier* carries more than one pitch.

Return named chromatic pitch. Changed in version 1.1.2: renamed `pitchtools.get_pitch( )` to `pitchtools.get_named_chromatic_pitch_from_pitch_carrier( )`.

pitchtools.get_numbered_chromatic_pitch_class_from_pitch_carrier

abjad.tools.pitchtools.**get_numbered_chromatic_pitch_class_from_pitch_carrier**(*pitch_carrier*)

New in version 1.1.2. Get numbered chromatic pitch-class from *pitch_carrier*:

```
abjad> note = Note("cs'4")
abjad> pitchtools.get_numbered_chromatic_pitch_class_from_pitch_carrier(note)
NumberedChromaticPitchClass(1)
```

Raise missing pitch error on empty chords.

Raise extra pitch error on many-note chords.

Return numbered chromatic pitch-class. Changed in version 1.1.2: renamed `pitchtools.get_numeric_chromatic_pitch_class_from_pitch_carrier( )` to `pitchtools.get_numbered_chromatic_pitch_class_from_pitch_carrier( )`.

pitchtools.insert_and_transpose_nested_subruns_in_chromatic_pitch_class_number_list

abjad.tools.pitchtools.**insert_and_transpose_nested_subruns_in_chromatic_pitch_class_number_

New in version 1.1.1. Insert and transpose nested subruns in *chromatic_pitch_class_number_list* according to *subrun_indicators*:

```
abjad> notes = [Note(p, (1, 4)) for p in [0, 2, 7, 9, 5, 11, 4]]
abjad> subrun_indicators = [(0, [2, 4]), (4, [3, 1])]
abjad> pitchtools.insert_and_transpose_nested_subruns_in_chromatic_pitch_class_number_list(notes

abjad> t = [ ]
abjad> for x in notes:
...   try:
...      t.append(x.pitch.chromatic_pitch_number)
...   except AttributeError:
...      t.append([y.pitch.chromatic_pitch_number for y in x])

abjad> t
[0, [5, 7], 2, [4, 0, 6, 11], 7, 9, 5, [10, 6, 8], 11, [7], 4]
```

Set *subrun_indicators* to a list of zero or more `(index, length_list)` pairs.

For each `(index, length_list)` pair in *subrun_indicators* the function will read *index* mod `len(notes)` and insert a subrun of length `length_list[0]` immediately after `notes[index]`, a subrun of length `length_list[1]` immediately after `notes[index+1]`, and, in general, a subrun of `length_list[i]` immediately after `notes[index+i]`, for `i < length(length_list)`.

New subruns are wrapped with lists. These wrapper lists are designed to allow inspection of the structural changes to *notes* immediately after the function returns. For this reason most calls to this function will be followed by `notes = seqtools.flatten_sequence(notes)`:

```
abjad> notes = seqtools.flatten_sequence(notes)
abjad> notes
[Note("c'4"), Note("f'4"), Note("g'4"), Note("d'4"), Note("e'4"), Note("c'4"), Note("fs'4"), Not
```

This function is designed to work on a built-in Python list of notes. This function is **not** designed to work on Abjad voices, staves or other containers because the function currently implements no spanner-handling. That is, this function is designed to be used during precomposition when other, similar abstract pitch transforms may be common.

Return list of integers and / or floats. Changed in version 1.1.2: renamed `pitchtools.insert_transposed_pc_subruns( )` to `pitchtools.insert_and_transpose_nested_sub` `)`.

pitchtools.instantiate_pitch_and_interval_test_collection

`abjad.tools.pitchtools.`**`instantiate_pitch_and_interval_test_collection`**`()`

New in version 1.1.2. Instantiate pitch and interval test collection:

```
abjad> for x in pitchtools.instantiate_pitch_and_interval_test_collection( ): x
...
HarmonicChromaticInterval(1)
HarmonicChromaticIntervalClass(1)
HarmonicCounterpointInterval(1)
HarmonicCounterpointIntervalClass(1)
HarmonicDiatonicInterval('M2')
HarmonicDiatonicIntervalClass('M2')
InversionEquivalentChromaticIntervalClass(1)
InversionEquivalentDiatonicIntervalClass('M2')
MelodicChromaticInterval(+1)
MelodicChromaticIntervalClass(+1)
MelodicCounterpointInterval(1)
MelodicCounterpointIntervalClass(+1)
MelodicDiatonicInterval('+M2')
MelodicDiatonicIntervalClass('+M2')
NamedChromaticPitch('c')
NamedChromaticPitchClass('c')
NamedDiatonicPitch('c')
NamedDiatonicPitchClass('c')
NumberedChromaticPitch(1)
NumberedChromaticPitchClass(1)
NumberedDiatonicPitch(1)
NumberedDiatonicPitchClass(1)
```

Use to test pitch and interval interface consistency.

Return list.

pitchtools.inventory_aggregate_subsets

`abjad.tools.pitchtools.`**`inventory_aggregate_subsets`**`()`

New in version 1.1.2. Inventory aggregate subsets:

```
abjad> U_star = pitchtools.inventory_aggregate_subsets( )
abjad> len(U_star)
4096
abjad> for pcset in U_star[:20]:
...   pcset
NumberedChromaticPitchClassSet([])
NumberedChromaticPitchClassSet([0])
NumberedChromaticPitchClassSet([1])
NumberedChromaticPitchClassSet([0, 1])
```

```
NumberedChromaticPitchClassSet([2])
NumberedChromaticPitchClassSet([0, 2])
NumberedChromaticPitchClassSet([1, 2])
NumberedChromaticPitchClassSet([0, 1, 2])
NumberedChromaticPitchClassSet([3])
NumberedChromaticPitchClassSet([0, 3])
NumberedChromaticPitchClassSet([1, 3])
NumberedChromaticPitchClassSet([0, 1, 3])
NumberedChromaticPitchClassSet([2, 3])
NumberedChromaticPitchClassSet([0, 2, 3])
NumberedChromaticPitchClassSet([1, 2, 3])
NumberedChromaticPitchClassSet([0, 1, 2, 3])
NumberedChromaticPitchClassSet([4])
NumberedChromaticPitchClassSet([0, 4])
NumberedChromaticPitchClassSet([1, 4])
NumberedChromaticPitchClassSet([0, 1, 4])
```

There are 4096 subsets of the aggregate.

This is `U*` in [Morris 1987].

Return list of numbered chromatic pitch-class sets.

pitchtools.inventory_inversion_equivalent_diatonic_interval_classes

abjad.tools.pitchtools.**inventory_inversion_equivalent_diatonic_interval_classes**()

New in version 1.1.2. Inventory inversion-equivalent diatonic interval-classes:

```
abjad> for dic in pitchtools.inventory_inversion_equivalent_diatonic_interval_classes( ):
...     dic
...
InversionEquivalentDiatonicIntervalClass('P1')
InversionEquivalentDiatonicIntervalClass('aug1')
InversionEquivalentDiatonicIntervalClass('m2')
InversionEquivalentDiatonicIntervalClass('M2')
InversionEquivalentDiatonicIntervalClass('aug2')
InversionEquivalentDiatonicIntervalClass('dim3')
InversionEquivalentDiatonicIntervalClass('m3')
InversionEquivalentDiatonicIntervalClass('M3')
InversionEquivalentDiatonicIntervalClass('dim4')
InversionEquivalentDiatonicIntervalClass('P4')
InversionEquivalentDiatonicIntervalClass('aug4')
```

There are 11 inversion-equivalent diatonic interval-classes.

It is an open question as to whether octaves should be included.

Return list of inversion-equivalent diatonic interval-classes.

pitchtools.is_alphabetic_accidental_abbreviation

abjad.tools.pitchtools.**is_alphabetic_accidental_abbreviation**(*expr*)

New in version 1.1.2. True when *expr* is an alphabetic accidental abbrevation. Otherwise false:

```
abjad> pitchtools.is_alphabetic_accidental_abbreviation('tqs')
True
```

The regex `^([s]{1,2}|[f]{1,2}|t?q?[fs])!?$` underlies this predicate.

Return boolean.

pitchtools.is_chromatic_pitch_class_name

`abjad.tools.pitchtools.`**`is_chromatic_pitch_class_name`**`(`*expr*`)`

New in version 1.1.2. True when *expr* is a chromatic pitch-class name. Otherwise false:

```
abjad> pitchtools.is_chromatic_pitch_class_name('fs')
True
```

The regex `^([a-g,A-G])(([s]{1,2}|[f]{1,2}|t?q?[fs]|)!?)$` underlies this predicate.

Return boolean.

pitchtools.is_chromatic_pitch_class_name_octave_number_pair

`abjad.tools.pitchtools.`**`is_chromatic_pitch_class_name_octave_number_pair`**`(`*expr*`)`

New in version 1.1.1. True when *arg* has the form of a chromatic pitch-class / octave number pair. Otherwise false:

```
abjad> pitchtools.is_chromatic_pitch_class_name_octave_number_pair(('cs', 5))
True
```

Return boolean. Changed in version 1.1.2: renamed `pitchtools.is_pair( )` to `pitchtools.is_chromatic_pitch_class_name_octave_number_pair( )`.

pitchtools.is_chromatic_pitch_class_number

`abjad.tools.pitchtools.`**`is_chromatic_pitch_class_number`**`(`*expr*`)`

New in version 1.1.2. True *expr* is a chromatic pitch-class number. Otherwise false:

```
abjad> pitchtools.is_chromatic_pitch_class_number(1)
True
```

The chromatic pitch-class numbers are equal to the set `[0, 0.5, ..., 11, 11.5]`.

Return boolean.

pitchtools.is_chromatic_pitch_name

`abjad.tools.pitchtools.`**`is_chromatic_pitch_name`**`(`*expr*`)`

New in version 1.1.2. True *expr* is a chromatic pitch name. Otherwise false:

```
abjad> pitchtools.is_chromatic_pitch_name('c,')
True
```

The regex `^([a-g,A-G])(([s]{1,2}|[f]{1,2}|t?q?[f,s]|)!?)(,+|'+|)$` underlies this predicate.

Return boolean.

pitchtools.is_chromatic_pitch_number

`abjad.tools.pitchtools.`**`is_chromatic_pitch_number`**`(`*expr*`)`

New in version 1.1.2. True *expr* is a chromatic pitch number. Otherwise false:

```
abjad> pitchtools.is_chromatic_pitch_number(13)
True
```

The chromatic pitch numbers are equal to the set of all integers in union with the set of all integers plus of minus `0.5`.

Return boolean.

pitchtools.is_diatonic_pitch_class_name

`abjad.tools.pitchtools.`**`is_diatonic_pitch_class_name`**`(`*expr*`)`

New in version 1.1.2. True when *expr* is a diatonic pitch-class name. Otherwise false:

```
abjad> pitchtools.is_diatonic_pitch_class_name('c')
True
```

The regex `^[a-g,A-G]$` underlies this predicate.

Return boolean.

pitchtools.is_diatonic_pitch_class_number

`abjad.tools.pitchtools.`**`is_diatonic_pitch_class_number`**`(`*expr*`)`

New in version 1.1.2. True when *expr* is a diatonic pitch-class number. Otherwise false:

```
abjad> pitchtools.is_diatonic_pitch_class_number(0)
True
```

The diatonic pitch-class numbers are equal to the set `[0, 1, 2, 3, 4, 5, 6]`.

Return boolean.

pitchtools.is_diatonic_pitch_name

`abjad.tools.pitchtools.`**`is_diatonic_pitch_name`**`(`*expr*`)`

New in version 1.1.2. True when *expr* is a diatonic pitch name. Otherwise false:

```
abjad> pitchtools.is_diatonic_pitch_name("c''")
True
```

The regex `(^[a-g,A-G])(,+|'+|)$` underlies this predicate.

Return boolean.

pitchtools.is_diatonic_pitch_number

`abjad.tools.pitchtools.`**`is_diatonic_pitch_number`**`(`*expr*`)`

New in version 1.1.2. True when *expr* is a diatonic pitch number. Otherwise false:

```
abjad> pitchtools.is_diatonic_pitch_number(7)
True
```

The diatonic pitch numbers are equal to the set of integers.

Return boolean.

pitchtools.is_diatonic_quality_abbreviation

`abjad.tools.pitchtools.is_diatonic_quality_abbreviation(expr)`

New in version 1.1.2. True when *expr* is a diatonic quality abbreviation. Otherwise false:

```
abjad> pitchtools.is_diatonic_quality_abbreviation('aug')
True
```

The regex `^M|m|P|aug|dim$` underlies this predicate.

Return boolean.

pitchtools.is_harmonic_diatonic_interval_abbreviation

`abjad.tools.pitchtools.is_harmonic_diatonic_interval_abbreviation(expr)`

New in version 1.1.2. True when *expr* is a harmonic diatonic interval abbreviation. Otherwise false:

```
abjad> pitchtools.is_harmonic_diatonic_interval_abbreviation('M9')
True
```

The regex `^(M|m|P|aug|dim)(\d+)$` underlies this predicate.

Return boolean.

pitchtools.is_melodic_diatonic_interval_abbreviation

`abjad.tools.pitchtools.is_melodic_diatonic_interval_abbreviation(expr)`

New in version 1.1.2. True when *expr* is a melodic diatonic interval abbreviation. Otherwise false:

```
abjad> pitchtools.is_melodic_diatonic_interval_abbreviation('+M9')
True
```

The regex `^([+,-]?)(M|m|P|aug|dim)(\d+)$` underlies this predicate.

Return boolean.

pitchtools.is_named_chromatic_pitch_token

`abjad.tools.pitchtools.is_named_chromatic_pitch_token(pitch_token)`

New in version 1.1.1. True when *pitch_token* has the form of an Abjad pitch token. Otherwise false:

```
abjad> pitchtools.is_named_chromatic_pitch_token(('c', 4))
True
```

Return boolean. Changed in version 1.1.2: renamed `pitchtools.is_pitch_token( )` to `pitchtools.is_named_chromatic_pitch_token( )`.

pitchtools.is_octave_tick_string

`abjad.tools.pitchtools.is_octave_tick_string(expr)`

New in version 1.1.2. True when *expr* is an octave tick string. Otherwise false:

```
abjad> pitchtools.is_octave_tick_string(',,,')
True
```

The regex `^,+|'+|$` underlies this predicate.

Return boolean.

pitchtools.is_pitch_carrier

`abjad.tools.pitchtools.`**`is_pitch_carrier`**`(`*expr*`)`

New in version 1.1.1. True when *expr* is an Abjad pitch, note, note-head of chord instance. Otherwise false:

```
abjad> note = Note(0, (1, 4))
abjad> pitchtools.is_pitch_carrier(note)
True
```

Return boolean. Changed in version 1.1.2: renamed `pitchtools.is_carrier( )` to `pitchtools.is_pitch_carrier( )`.

pitchtools.iterate_named_chromatic_pitch_pairs_forward_in_expr

`abjad.tools.pitchtools.`**`iterate_named_chromatic_pitch_pairs_forward_in_expr`**`(`*expr*`)`

New in version 1.1.2. Iterate left-to-right, top-to-bottom named chromatic pitch pairs in *expr*:

```
abjad> score = Score([ ])
abjad> notes = macros.scale(4) + [Note(7, (1, 4))]
abjad> score.append(Staff(notes))
abjad> notes = [Note(x, (1, 4)) for x in [-12, -15, -17]]
abjad> score.append(Staff(notes))
abjad> contexttools.ClefMark('bass')(score[1])
ClefMark('bass')(Staff{3})

abjad> f(score)
\new Score <<
        \new Staff {
                c'8
                d'8
                e'8
                f'8
                g'4
        }
        \new Staff {
                \clef "bass"
                c4
                a,4
                g,4
        }
>>

abjad> for pair in pitchtools.iterate_named_chromatic_pitch_pairs_forward_in_expr(score):
...     pair
...
(NamedChromaticPitch("c'"), NamedChromaticPitch('c'))
(NamedChromaticPitch("c'"), NamedChromaticPitch("d'"))
(NamedChromaticPitch('c'), NamedChromaticPitch("d'"))
(NamedChromaticPitch("d'"), NamedChromaticPitch("e'"))
(NamedChromaticPitch("d'"), NamedChromaticPitch('a,'))
```

```
(NamedChromaticPitch('c'), NamedChromaticPitch("e'"))
(NamedChromaticPitch('c'), NamedChromaticPitch('a,'))
(NamedChromaticPitch("e'"), NamedChromaticPitch('a,'))
(NamedChromaticPitch("e'"), NamedChromaticPitch("f'"))
(NamedChromaticPitch('a,'), NamedChromaticPitch("f'"))
(NamedChromaticPitch("f'"), NamedChromaticPitch("g'"))
(NamedChromaticPitch("f'"), NamedChromaticPitch('g,'))
(NamedChromaticPitch('a,'), NamedChromaticPitch("g'"))
(NamedChromaticPitch('a,'), NamedChromaticPitch('g,'))
(NamedChromaticPitch("g'"), NamedChromaticPitch('g,'))
```

Chords are handled correctly.

```
abjad> chord_1 = Chord([0, 2, 4], (1, 4))
abjad> chord_2 = Chord([17, 19], (1, 4))
abjad> staff = Staff([chord_1, chord_2])

abjad> f(staff)
\new Staff {
        <c' d' e'>4
        <f'' g''>4
}

abjad> for pair in pitchtools.iterate_named_chromatic_pitch_pairs_forward_in_expr(staff):
...   print pair
(NamedChromaticPitch("c'"), NamedChromaticPitch("d'"))
(NamedChromaticPitch("c'"), NamedChromaticPitch("e'"))
(NamedChromaticPitch("d'"), NamedChromaticPitch("e'"))
(NamedChromaticPitch("c'"), NamedChromaticPitch("f''"))
(NamedChromaticPitch("c'"), NamedChromaticPitch("g''"))
(NamedChromaticPitch("d'"), NamedChromaticPitch("f''"))
(NamedChromaticPitch("d'"), NamedChromaticPitch("g''"))
(NamedChromaticPitch("e'"), NamedChromaticPitch("f''"))
(NamedChromaticPitch("e'"), NamedChromaticPitch("g''"))
(NamedChromaticPitch("f''"), NamedChromaticPitch("g''"))
```

Return generator.

pitchtools.list_chromatic_pitch_numbers_in_expr

abjad.tools.pitchtools.**list_chromatic_pitch_numbers_in_expr**(*expr*)
New in version 1.1.2. List chromatic pitch numbers in *expr*:

```
abjad> tuplet = tuplettools.FixedDurationTuplet((2, 8), macros.scale(3))
abjad> pitchtools.list_chromatic_pitch_numbers_in_expr(tuplet)
(0, 2, 4)
```

Return tuple of zero or more numbers.

pitchtools.list_harmonic_chromatic_intervals_in_expr

abjad.tools.pitchtools.**list_harmonic_chromatic_intervals_in_expr**(*expr*)
New in version 1.1.2. List harmonic chromatic intervals in *expr*:

```
abjad> staff = Staff(macros.scale(4))
abjad> for interval in sorted(pitchtools.list_harmonic_chromatic_intervals_in_expr(staff)):
...     interval
```

```
...
HarmonicChromaticInterval(1)
HarmonicChromaticInterval(2)
HarmonicChromaticInterval(2)
HarmonicChromaticInterval(3)
HarmonicChromaticInterval(4)
HarmonicChromaticInterval(5)
```

Return unordered set.

pitchtools.list_harmonic_diatonic_intervals_in_expr

abjad.tools.pitchtools.**list_harmonic_diatonic_intervals_in_expr**(*expr*)

New in version 1.1.2. List harmonic diatonic intervals in *expr*:

```
abjad> staff = Staff(macros.scale(4))
abjad> for interval in sorted(pitchtools.list_harmonic_diatonic_intervals_in_expr(staff)):
...     interval
...
HarmonicDiatonicInterval('m2')
HarmonicDiatonicInterval('M2')
HarmonicDiatonicInterval('M2')
HarmonicDiatonicInterval('m3')
HarmonicDiatonicInterval('M3')
HarmonicDiatonicInterval('P4')
```

Return unordered set.

pitchtools.list_inversion_equivalent_chromatic_interval_classes_pairwise_between_pitch_carriers

abjad.tools.pitchtools.**list_inversion_equivalent_chromatic_interval_classes_pairwise_betwee**

New in version 1.1.2. List inversion-equivalent chromatic interval-classes pairwise between *pitch_carriers*:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8 g'8 a'8 b'8 c''8")

abjad> f(staff)
\new Staff {
        c'8
        d'8
        e'8
        f'8
        g'8
        a'8
        b'8
        c''8
}

abjad> pitchtools.list_inversion_equivalent_chromatic_interval_classes_pairwise_between_pitch_ca
[InversionEquivalentChromaticIntervalClass(2), InversionEquivalentChromaticIntervalClass(2), Inv
InversionEquivalentChromaticIntervalClass(2), InversionEquivalentChromaticIntervalClass(2), Inve
InversionEquivalentChromaticIntervalClass(1)]

abjad> pitchtools.list_inversion_equivalent_chromatic_interval_classes_pairwise_between_pitch_ca
[InversionEquivalentChromaticIntervalClass(2), InversionEquivalentChromaticIntervalClass(2), Inv
InversionEquivalentChromaticIntervalClass(2), InversionEquivalentChromaticIntervalClass(2), Inve
InversionEquivalentChromaticIntervalClass(1), InversionEquivalentChromaticIntervalClass(0)]
```

```
abjad> notes = macros.scale(8)
abjad> notes.reverse( )
abjad> notes
[Note("c''8"), Note("b'8"), Note("a'8"), Note("g'8"), Note("f'8"), Note("e'8"), Note("d'8"), Not

abjad> pitchtools.list_inversion_equivalent_chromatic_interval_classes_pairwise_between_pitch_ca
[InversionEquivalentChromaticIntervalClass(1), InversionEquivalentChromaticIntervalClass(2), Inv
InversionEquivalentChromaticIntervalClass(2), InversionEquivalentChromaticIntervalClass(1), Inve
InversionEquivalentChromaticIntervalClass(2)]

abjad> pitchtools.list_inversion_equivalent_chromatic_interval_classes_pairwise_between_pitch_ca
[InversionEquivalentChromaticIntervalClass(1), InversionEquivalentChromaticIntervalClass(2), Inv
InversionEquivalentChromaticIntervalClass(2), InversionEquivalentChromaticIntervalClass(1), Inve
InversionEquivalentChromaticIntervalClass(2), InversionEquivalentChromaticIntervalClass(0)]
```

When `wrap = False` do not return `pitch_carriers[-1] - pitch_carriers[0]` as last in series.

When `wrap = True` do return `pitch_carriers[-1] - pitch_carriers[0]` as last in series.

Return list.

pitchtools.list_melodic_chromatic_interval_numbers_pairwise_between_pitch_carriers

abjad.tools.pitchtools.**list_melodic_chromatic_interval_numbers_pairwise_between_pitch_carr**

New in version 1.1.1. List melodic chromatic interval numbers pairwise between *pitch_carriers*:

```
abjad> staff = Staff(macros.scale(8))
abjad> print staff.format
\new Staff {
        c'8
        d'8
        e'8
        f'8
        g'8
        a'8
        b'8
        c''8
}

abjad> pitchtools.list_melodic_chromatic_interval_numbers_pairwise_between_pitch_carriers(staff)
[2, 2, 1, 2, 2, 2, 1]

abjad> pitchtools.list_melodic_chromatic_interval_numbers_pairwise_between_pitch_carriers(staff,
[2, 2, 1, 2, 2, 2, 1, -12]

abjad> notes = macros.scale(8)
abjad> notes.reverse( )
abjad> notes
[Note("c''8"), Note("b'8"), Note("a'8"), Note("g'8"), Note("f'8"), Note("e'8"), Note("d'8"), Not

abjad> pitchtools.list_melodic_chromatic_interval_numbers_pairwise_between_pitch_carriers(notes)
[-1, -2, -2, -2, -1, -2, -2]

abjad> pitchtools.list_melodic_chromatic_interval_numbers_pairwise_between_pitch_carriers(notes,
[-1, -2, -2, -2, -1, -2, -2, 12]
```

When `wrap = False` do not return `pitch_carriers[-1] - pitch_carriers[0]` as last in series.

When `wrap = True` do return `pitch_carriers[-1] - pitch_carriers[0]` as last in series.

Return list. Changed in version 1.1.2: renamed `pitchtools.get_signed_interval_series( )` to `pitchtools.list_melodic_chromatic_interval_numbers_pairwise_between_pitch_carriers( )`.

pitchtools.list_named_chromatic_pitch_carriers_in_expr_sorted_by_numbered_chromatic_pitch_class

`abjad.tools.pitchtools.`**`list_named_chromatic_pitch_carriers_in_expr_sorted_by_numbered_chron`**

New in version 1.1.2. List named chromatic pitch carriers in *expr* sorted by numbered chromatic pitch-class:

```
abjad> chord = Chord([9, 11, 12, 14, 16], (1, 4))
abjad> notes = chordtools.arpeggiate_chord(chord)
abjad> pitchtools.list_named_chromatic_pitch_carriers_in_expr_sorted_by_numbered_chromatic_pitch
[Note("c''4"), Note("d''4"), Note("e''4"), Note("a'4"), Note("b'4")]
```

The elements in *pitch_carriers* are not changed in any way.

Return list. Changed in version 1.1.2: renamed `pitchtools.list_named_chromatic_pitch_carriers_in_expr_` `)` to `pitchtools.list_named_chromatic_pitch_carriers_in_expr_sorted_by_numbered_chromatic_` `)`.

pitchtools.list_named_chromatic_pitches_in_expr

`abjad.tools.pitchtools.`**`list_named_chromatic_pitches_in_expr`**(*expr*)

New in version 1.1.2. List named chromatic pitches in *expr*:

```
abjad> t = Staff("c'4 d'4 e'4 f'4")
abjad> beam = spannertools.BeamSpanner(t[:])
abjad> pitchtools.list_named_chromatic_pitches_in_expr(beam)
(NamedChromaticPitch("c'"), NamedChromaticPitch("d'"), NamedChromaticPitch("e'"), NamedChromatic
```

Return tuple.

pitchtools.list_numbered_chromatic_pitch_classes_in_expr

`abjad.tools.pitchtools.`**`list_numbered_chromatic_pitch_classes_in_expr`**(*expr*)

New in version 1.1.2. List numbered chromatic pitch-classes in *expr*:

```
abjad> chord = Chord([13, 14, 15], (1, 4))
abjad> pitchtools.list_numbered_chromatic_pitch_classes_in_expr(chord)
(NumberedChromaticPitchClass(1), NumberedChromaticPitchClass(2), NumberedChromaticPitchClass(3))
```

Works with notes, chords, defective chords.

Return tuple or zero or more numbered chromatic pitch-classes. Changed in version 1.1.2: renamed `pitchtools.list_numeric_chromatic_pitch_classes_in_expr( )` to `pitchtools.list_numbered_chromatic_pitch_classes_in_expr( )`.

pitchtools.list_octave_transpositions_of_pitch_carrier_within_pitch_range

`abjad.tools.pitchtools.`**`list_octave_transpositions_of_pitch_carrier_within_pitch_range`**(*pitches*
pitch_r

New in version 1.1.1. List octave transpositions of *pitches* in *pitch_range*:

```
abjad> chord = Chord([0, 2, 4], (1, 4))
abjad> pitch_range = pitchtools.PitchRange(0, 48)
abjad> pitchtools.list_octave_transpositions_of_pitch_carrier_within_pitch_range(chord, pitch_ra
[Chord(c' d' e', 4), Chord(c'' d'' e'', 4), Chord(c''' d''' e''', 4), Chord(c'''' d'''' e'''', 4
```

Return list.

pitchtools.list_ordered_named_chromatic_pitch_pairs_from_expr_1_to_expr_2

abjad.tools.pitchtools.**list_ordered_named_chromatic_pitch_pairs_from_expr_1_to_expr_2**(*expr_1*, *expr_2*)

New in version 1.1.2. List ordered named chromatic pitch pairs from *expr_1* to *expr_2*:

```
abjad> chord_1 = Chord([0, 1, 2], (1, 4))
abjad> chord_2 = Chord([3, 4], (1, 4))
abjad> for pair in pitchtools.list_ordered_named_chromatic_pitch_pairs_from_expr_1_to_expr_2(cho
...      pair
(NamedChromaticPitch("c'"), NamedChromaticPitch("ef'"))
(NamedChromaticPitch("c'"), NamedChromaticPitch("e'"))
(NamedChromaticPitch("cs'"), NamedChromaticPitch("ef'"))
(NamedChromaticPitch("cs'"), NamedChromaticPitch("e'"))
(NamedChromaticPitch("d'"), NamedChromaticPitch("ef'"))
(NamedChromaticPitch("d'"), NamedChromaticPitch("e'"))
```

Return generator.

pitchtools.list_unordered_named_chromatic_pitch_pairs_in_expr

abjad.tools.pitchtools.**list_unordered_named_chromatic_pitch_pairs_in_expr**(*expr*)

New in version 1.1.2. List unordered named chromatic pitch pairs in *expr*:

```
abjad> for pair in pitchtools.list_unordered_named_chromatic_pitch_pairs_in_expr(Chord([0, 1, 2,
...     pair
...
(NamedChromaticPitch("c'"), NamedChromaticPitch("cs'"))
(NamedChromaticPitch("c'"), NamedChromaticPitch("d'"))
(NamedChromaticPitch("c'"), NamedChromaticPitch("ef'"))
(NamedChromaticPitch("cs'"), NamedChromaticPitch("d'"))
(NamedChromaticPitch("cs'"), NamedChromaticPitch("ef'"))
(NamedChromaticPitch("d'"), NamedChromaticPitch("ef'"))
```

Return generator.

pitchtools.make_n_middle_c_centered_pitches

abjad.tools.pitchtools.**make_n_middle_c_centered_pitches**(*n*)

New in version 1.1.2. Make *n* middle-c centered pitches, where $0 < n$:

```
abjad> for p in pitchtools.make_n_middle_c_centered_pitches(5): p
NamedChromaticPitch('f')
NamedChromaticPitch('a')
NamedChromaticPitch("c'")
NamedChromaticPitch("e'")
NamedChromaticPitch("g'")
```

```
abjad> for p in pitchtools.make_n_middle_c_centered_pitches(4): p
NamedChromaticPitch('g')
NamedChromaticPitch('b')
NamedChromaticPitch("d'")
NamedChromaticPitch("f'")
```

Return list of zero or more named chromatic pitches.

pitchtools.named_chromatic_pitch_and_clef_to_staff_position_number

abjad.tools.pitchtools.**named_chromatic_pitch_and_clef_to_staff_position_number**(*pitch*, *clef*)

New in version 1.1.2. Change named chromatic *pitch* and *clef* to staff position number:

```
abjad> staff = Staff(macros.scale(8))
abjad> clef = contexttools.ClefMark('treble')
abjad> for note in staff:
...     pitch = note.pitch
...     number = pitchtools.named_chromatic_pitch_and_clef_to_staff_position_number(pitch, clef)
...     print '%s\t%s' % (pitch, number)
c'      -6
d'      -5
e'      -4
f'      -3
g'      -2
a'      -1
b'      0
c''     1
```

Return integer.

pitchtools.named_chromatic_pitch_tokens_to_named_chromatic_pitches

abjad.tools.pitchtools.**named_chromatic_pitch_tokens_to_named_chromatic_pitches**(*pitch_tokens*)

New in version 1.1.2. Change named chromatic *pitch_tokens* to named chromatic pitches:

```
abjad> pitchtools.named_chromatic_pitch_tokens_to_named_chromatic_pitches([0, 2, ('ef', 4)])
[NamedChromaticPitch("c'"), NamedChromaticPitch("d'"), NamedChromaticPitch("ef'")]
```

Return list of zero or more named chromatic pitches.

pitchtools.named_chromatic_pitches_to_harmonic_chromatic_interval_class_number_dictionary

abjad.tools.pitchtools.**named_chromatic_pitches_to_harmonic_chromatic_interval_class_number_**

New in version 1.1.1. Change named chromatic pitches to harmonic chromatic interval-class number dictionary:

```
abjad> chord = Chord([0, 2, 11], (1, 4))
abjad> vector = pitchtools.named_chromatic_pitches_to_harmonic_chromatic_interval_class_number_d
abjad> vector
{0: 0, 1: 0, 2: 1, 3: 0, 4: 0, 5: 0, 6: 0, 7: 0, 8: 0, 9: 1, 10: 0, 11: 1}
```

Return dictionary. Changed in version 1.1.2: renamed `pitchtools.get_interval_vector( )` to `pitchtools.named_chromatic_pitches_to_harmonic_chromatic_interval_class_number_diction` `)`.

pitchtools.named_chromatic_pitches_to_inversion_equivalent_chromatic_interval_class_number_dictionary

abjad.tools.pitchtools.**named_chromatic_pitches_to_inversion_equivalent_chromatic_interval_c**

New in version 1.1.1. Change named chromatic *pitches* to inversion-equivalent chromatic interval-class number dictionary:

```
abjad> chord = Chord([0, 2, 11], (1, 4))
abjad> vector = pitchtools.named_chromatic_pitches_to_inversion_equivalent_chromatic_interval_cl
abjad> for i in range(7):
...     print '\t%s\t%s' % (i, vector[i])
...
   0   0
   1   1
   2   1
   3   1
   4   0
   5   0
   6   0
```

Changed in version 1.1.2: works with quartertones. Return dictionary. Changed in version 1.1.2: renamed `pitchtools.get_interval_class_vector( )` to `pitchtools.named_chromatic_pitches_to_inversion_equivalent_chromatic_interval_class_nu` `)`.

pitchtools.octave_number_to_octave_tick_string

abjad.tools.pitchtools.**octave_number_to_octave_tick_string**(*octave_number*)

New in version 1.1.2. Change *octave_number* to octave tick string:

```
abjad> for octave_number in range(-1, 9):
...     print "%s\t%s" % (octave_number, pitchtools.octave_number_to_octave_tick_string(octave_n
...
-1 ,,,,
0  ,,,
1  ,,
2  ,
3
4  '
5  ''
6  '''
7  ''''
8  '''''
```

Raise type error on noninteger input.

Return string.

pitchtools.octave_tick_string_to_octave_number

abjad.tools.pitchtools.**octave_tick_string_to_octave_number**(*tick_string*)

New in version 1.1.2. Change *tick_string* to octave number:

```
abjad> pitchtools.octave_tick_string_to_octave_number("'")
4
```

Raise type error on nonstring input.

Raise value error on input not of tick string format.

Return integer.

pitchtools.ordered_chromatic_pitch_class_numbers_are_within_ordered_chromatic_pitch_numbers

abjad.tools.pitchtools.**ordered_chromatic_pitch_class_numbers_are_within_ordered_chromatic_p**

New in version 1.1.1. True if ordered *chromatic_pitch_class_numbers'are within ordered 'chromatic_pitch_numbers*:

```
abjad> pcs = [2, 7, 10]
abjad> pitches = [6, 9, 12, 13, 14, 19, 22, 27, 28, 29, 32, 35]
abjad> pitchtools.ordered_chromatic_pitch_class_numbers_are_within_ordered_chromatic_pitch_numbe
True
```

Return boolean. Changed in version 1.1.2: renamed `pitchtools.are_in_octave_order( )` to `pitchtools.ordered_chromatic_pitch_class_numbers_are_within_ordered_chromatic_pitch_nu )`.

pitchtools.pentatonic_pitch_number_to_chromatic_pitch_number

abjad.tools.pitchtools.**pentatonic_pitch_number_to_chromatic_pitch_number**(*pentatonic_scale_degree*, *transpose=1*, *phase=0*)

New in version 1.1.1. Changed *pentatonic_scale_degree* number to chromatic pitch number:

```
abjad> for pentatonic_scale_degree in range(9): # doctest: +SKIP
...     chromatic_pitch_number = pitchtools.pentatonic_pitch_number_to_chromatic_pitch_number(pe
...     print '%s\t%s' % (pentatonic_scale_degree, chromatic_pitch_number)
...
0   1
1   3
2   6
3   8
4   10
5   13
6   15
7   18
8   20
```

Pentatonic scale degrees may be negative:

```
abjad> for pentatonic_scale_degree in range(-1, -9, -1): # doctest: +SKIP
...     chromatic_pitch_number = pitchtools.pentatonic_pitch_number_to_chromatic_pitch_number(pe
...     print '%s\t%s' % (pentatonic_scale_degree, chromatic_pitch_number)
...
-1 -2
-2 -4
-3 -6
-4 -9
-5 -11
-6 -14
-7 -16
-8 -18
```

Return integer. Changed in version 1.1.2: renamed `pitchtools.pentatonic_to_chromatic( )` to `pitchtools.pentatonic_pitch_number_to_chromatic_pitch_number( )`.

pitchtools.permute_named_chromatic_pitch_carrier_list_by_twelve_tone_row

`abjad.tools.pitchtools.`**`permute_named_chromatic_pitch_carrier_list_by_twelve_tone_row`**(*pitches*, *row*)

New in version 1.1.2. Permute named chromatic pitch carrier list by twelve-tone *row*:

```
abjad> notes = notetools.make_notes([17, -10, -2, 11], [Duration(1, 4)])
abjad> row = pitchtools.TwelveToneRow([10, 0, 2, 6, 8, 7, 5, 3, 1, 9, 4, 11])
abjad> pitchtools.permute_named_chromatic_pitch_carrier_list_by_twelve_tone_row(notes, row)
[Note('bf4'), Note('d4'), Note("f''4"), Note("b'4")]
```

Function works by reference only. No objects are cloned.

Return list.

pitchtools.register_chromatic_pitch_class_numbers_by_chromatic_pitch_number_aggregate

`abjad.tools.pitchtools.`**`register_chromatic_pitch_class_numbers_by_chromatic_pitch_number_agg`**

New in version 1.1.1. Register chromatic *pitch_class_numbers* by chromatic pitch-number *aggregate*:

```
abjad> pitchtools.register_chromatic_pitch_class_numbers_by_chromatic_pitch_number_aggregate(
...     [10, 0, 2, 6, 8, 7, 5, 3, 1, 9, 4, 11],
...     [10, 19, 20, 23, 24, 26, 27, 29, 30, 33, 37, 40])
[10, 24, 26, 30, 20, 19, 29, 27, 37, 33, 40, 23]
```

Return list of zero or more chromatic pitch numbers. Changed in version 1.1.2: renamed `pitchtools.registrate( )` to `pitchtools.register_chromatic_pitch_class_numbers_by_chromat`
`)`.

pitchtools.respell_named_chromatic_pitches_in_expr_with_flats

`abjad.tools.pitchtools.`**`respell_named_chromatic_pitches_in_expr_with_flats`**(*expr*)

New in version 1.1.1. Respell named chromatic pitches in *expr* with flats:

```
abjad> staff = Staff(notetools.make_repeated_notes(6))
abjad> macros.chromaticize(staff)

abjad> f(staff)
\new Staff {
   c'8
   cs'8
   d'8
   ef'8
   e'8
   f'8
}

abjad> pitchtools.respell_named_chromatic_pitches_in_expr_with_flats(staff)
```

```
abjad> f(staff)
\new Staff {
   c'8
   df'8
   d'8
   ef'8
   e'8
   f'8
}
```

Return none. Changed in version 1.1.2: renamed `pitchtools.make_flat( )` to `pitchtools.respell_named_chromatic_pitches_in_expr_with_flats( )`.

pitchtools.respell_named_chromatic_pitches_in_expr_with_sharps

abjad.tools.pitchtools.**respell_named_chromatic_pitches_in_expr_with_sharps**(*expr*)

New in version 1.1.1. Respell named chromatic pitches in *expr* with sharps:

```
abjad> staff = Staff(notetools.make_repeated_notes(6))
abjad> macros.chromaticize(staff)

abjad> f(staff)
\new Staff {
   c'8
   cs'8
   d'8
   ef'8
   e'8
   f'8
}

abjad> pitchtools.respell_named_chromatic_pitches_in_expr_with_sharps(staff)

abjad> f(staff)
\new Staff {
   c'8
   cs'8
   d'8
   ds'8
   e'8
   f'8
}
```

Return none. Changed in version 1.1.2: renamed `pitchtools.make_sharp( )` to `pitchtools.respell_named_chromatic_pitches_in_expr_with_sharps( )`.

pitchtools.set_ascending_named_chromatic_pitches_on_nontied_pitched_components_in_expr

abjad.tools.pitchtools.**set_ascending_named_chromatic_pitches_on_nontied_pitched_components_**

New in version 1.1.1. Set ascending named chromatic pitches on nontied pitched components in *expr*:

```
abjad> staff = Voice(notetools.make_notes(0, [(5, 32)] * 4))
abjad> macros.chromaticize(staff)

abjad> f(staff)
\new Voice {
```

```
        c'8 ~
        c'32
        cs'8 ~
        cs'32
        d'8 ~
        d'32
        ef'8 ~
        ef'32
}
```

Used primarily in generating test file examples.

Return none. Changed in version 1.1.2: renamed `pitchtools.chromaticize( )` to `pitchtools.set_ascending_named_chromatic_pitches_on_nontied_pitched_components_in_expr( )`.

pitchtools.set_ascending_named_diatonic_pitches_on_nontied_pitched_components_in_expr

abjad.tools.pitchtools.**set_ascending_named_diatonic_pitches_on_nontied_pitched_components_i**

New in version 1.1.1. Set ascending named diatonic pitches on nontied pitched components in *expr*:

```
abjad> staff = Staff(notetools.make_notes(0, [(5, 32)] * 4))
abjad> macros.diatonicize(staff)

abjad> f(staff)
\new Staff {
   c'8 ~
   c'32
   d'8 ~
   d'32
   e'8 ~
   e'32
   f'8 ~
   f'32
}
```

Used primarily in generating test file examples. New in version 1.1.2: Optional *key_signature* keyword argument. Return none. Changed in version 1.1.2: renamed `pitchtools.diatonicize( )` to `pitchtools.set_ascending_named_diatonic_pitches_on_nontied_pitched_components_in_expr( )`.

pitchtools.suggest_clef_for_named_chromatic_pitches

abjad.tools.pitchtools.**suggest_clef_for_named_chromatic_pitches**(*pitches, clefs=['treble', 'bass']*)

New in version 1.1.1. Suggest clef for named chromatic *pitches*:

```
abjad> staff = Staff(notetools.make_notes(range(-12, -6), [(1, 4)]))
abjad> pitchtools.suggest_clef_for_named_chromatic_pitches(staff)
ClefMark('bass')
```

Suggest clef based on minimal number of ledger lines.

Return clef mark. Changed in version 1.1.2: renamed `pitchtools.suggest_clef( )` to `pitchtools.suggest_clef_for_named_chromatic_pitches( )`.

pitchtools.transpose_chromatic_pitch_by_melodic_chromatic_interval_segment

abjad.tools.pitchtools.**transpose_chromatic_pitch_by_melodic_chromatic_interval_segment**(*pitch seg- ment*

New in version 1.1.2. Transpose chromatic *pitch* by melodic chromatic interval *segment*:

```
abjad> ncp = pitchtools.NumberedChromaticPitch(0)
abjad> mcis = pitchtools.MelodicChromaticIntervalSegment([0, -1, 2])
abjad> pitchtools.transpose_chromatic_pitch_by_melodic_chromatic_interval_segment(ncp, mcis)
[NumberedChromaticPitch(0), NumberedChromaticPitch(-1), NumberedChromaticPitch(1)]
```

Transpose by each interval in *segment* such that each tranposition transposes the resulting pitch of the previous transposition.

Return list of numbered chromatic pitches.

pitchtools.transpose_chromatic_pitch_class_number_by_octaves_to_nearest_neighbor_of_chromatic_pitch_numl

abjad.tools.pitchtools.**transpose_chromatic_pitch_class_number_by_octaves_to_nearest_neighbo**

New in version 1.1.1. Transpose *chromatic_pitch_class_number* by octaves to nearest neighbor of *chromatic_pitch_number*:

```
abjad> pitchtools.transpose_chromatic_pitch_class_number_by_octaves_to_nearest_neighbor_of_chrom
16
```

Resulting chromatic pitch number must be within one tritone of *pitch_number*.

Return integer or float. Changed in version 1.1.2: renamed `pitchtools.nearest_neighbor( )` to `pitchtools.transpose_chromatic_pitch_class_number_by_octaves_to_nearest_neighbor_of_ch )`.

pitchtools.transpose_chromatic_pitch_number_by_octave_transposition_mapping

abjad.tools.pitchtools.**transpose_chromatic_pitch_number_by_octave_transposition_mapping**(*chr map ping*

New in version 1.1.1. Transpose *chromatic_pitch_number* by the some number of octaves up or down. Derive correct number of octaves from *mapping* where *mapping* is a list of `(range_spec, octave)` pairs and `range_spec` is, in turn, a `(start, stop)` pair suitable to pass to the built-in Python `range( )` function:

```
abjad> mapping = [((-39, -13), 0), ((-12, 23), 12), ((24, 48), 24)]
```

The mapping given here comprises three `(range_spec, octave)` pairs. The first such pair is `((-39, -13), 0)` and can be read as follows: “any pitches between `-39` and `-13` should be transposed into the octave rooted at pitch `0`.” The octave rooted at pitch `0` equals the twelve pitches `range(0, 0 + 12)` or `[0, 1, ..., 10, 11]`.

The second `(range_spec, octave)` pair is `((-12, 23), 12)` and can be read as “any pitches between `-12` and `23` should be transposed into the octave rooted at pitch `12`,” with the octave rooted at pitch `12` equal to the twelve pitches `range(12, 12 + 12)` or `[12, 13, ..., 22, 23]`.

The third and last `(range_spec, octave)` pair is `((24, 48), 24)` and can be read as “any pitches between `24` and `48` should be transposed to the octave rooted at `24`,” with the octave rooted at `24` equal to the twelve pitches `range(24, 24, + 12)` or `[24, 25, ..., 34, 35]`.

The mapping given here divides the compass of the piano, from `-39` to `48`, into three disjunct subranges and then explains how to transpose pitches found in any of those three disjunct subranges. This means that, for example, all the f-sharps within the range of the piano now undergo a known transposition under *mapping* as defined here:

```
abjad> pitchtools.transpose_chromatic_pitch_number_by_octave_transposition_mapping(-30, mapping)
6
```

We verify that pitch `-30` should map to pitch `6` by noticing that pitch `-30` falls in the first of the three subranges defined by *mapping* from `-39` to `-13` and then noting that *mapping* sends pitches with that subrange to the octave rooted at pitch `0`. The octave transposition of `-30` that falls within the octave rooted at `0` is `6`:

```
abjad> pitchtools.transpose_chromatic_pitch_number_by_octave_transposition_mapping(-18, mapping)
6
```

Likewise, *mapping* sends pitch `-18` to pitch `6` because pitch `-18` falls in the same subrange from `-39` to `-13` as did pitch `-39` and so undergoes the same transposition to the octave rooted at `0`.

In this way we can map all f-sharps from `-39` to `48` according to *mapping*:

```
abjad> pitch_numbers = [-30, -18, -6, 6, 18, 30, 42]
abjad> for n in pitch_numbers:
...   n, pitchtools.transpose_chromatic_pitch_number_by_octave_transposition_mapping(n, mapping)
(-30, 6)
(-18, 6)
(-6, 18)
(6, 18)
(18, 18)
(30, 30)
(42, 30)
```

And so on.

Return chromatic pitch number. Changed in version 1.1.2: renamed `pitchtools.send_pitch_number_to_octave( )` to `pitchtools.transpose_chromatic_pitch_number_` `)`.

pitchtools.transpose_named_chromatic_pitch_by_melodic_chromatic_interval_and_respell

`abjad.tools.pitchtools.`**`transpose_named_chromatic_pitch_by_melodic_chromatic_interval_and_re`**

New in version 1.1.1. Transpose named chromatic pitch by *melodic_chromatic_interval* and respell *staff_spaces* above or below:

```
abjad> pitch = pitchtools.NamedChromaticPitch(0)
abjad> pitchtools.transpose_named_chromatic_pitch_by_melodic_chromatic_interval_and_respell(pitc
NamedChromaticPitch("dtqf'")
```

Return new named chromatic pitch. Changed in version 1.1.2: renamed `pitchtools.staff_space_transpose( )` to `pitchtools.transpose_named_chromatic_pitch_by_melo` `)`.

pitchtools.transpose_pitch_carrier_by_melodic_interval

`abjad.tools.pitchtools.`**`transpose_pitch_carrier_by_melodic_interval`**`(`*pitch_carrier*, *melodic_interval*`)`

New in version 1.1.2. Transpose *pitch_carrier* by diatonic *melodic_interval*:

```
abjad> chord = Chord("<c' e' g'>4")

abjad> pitchtools.transpose_pitch_carrier_by_melodic_interval(chord, '+m2')
Chord("<df' f' af'>4")
```

Transpose *pitch_carrier* by chromatic *melodic_interval*:

```
abjad> chord = Chord("<c' e' g'>4")

abjad> pitchtools.transpose_pitch_carrier_by_melodic_interval(chord, 1)
Chord("<cs' f' af'>4")
```

Return non-pitch-carrying input unchaged:

```
abjad> rest = Rest('r4')

abjad> pitchtools.transpose_pitch_carrier_by_melodic_interval(rest, 1)
Rest('r4')
```

Return *pitch_carrier*.

pitchtools.transpose_pitch_expr_into_pitch_range

abjad.tools.pitchtools.**transpose_pitch_expr_into_pitch_range**(*pitch_expr*, *pitch_range*)

New in version 1.1.2. Transpose *pitch_expr* into *pitch_range*:

```
abjad> pitchtools.transpose_pitch_expr_into_pitch_range([-2, -1, 13, 14], pitchtools.PitchRange(
[10, 11, 1, 2]
```

Return new *pitch_expr* object.

resttools

resttools.MultiMeasureRest

class abjad.tools.resttools.**MultiMeasureRest**(**args*, ***kwargs*)

Bases: `abjad.components.Rest.Rest.Rest` New in version 1.1.2. Abjad model of a multi-measure rest:

```
abjad> resttools.MultiMeasureRest((1, 4))
MultiMeasureRest('R4')
```

Multi-measure rests are immutable.

resttools.is_lilypond_rest_string

abjad.tools.resttools.**is_lilypond_rest_string**(*expr*)

New in version 1.1.2. True when *expr* is a LilyPond rest string:

```
abjad> resttools.is_lilypond_rest_string('r4.. * 1/2')
True
```

Otherwise false:

```
abjad> resttools.is_lilypond_rest_string('text')
False
```

The regex `^(r|R)\s*(1|2|4|8|16|32|64|128|\breve|\longa|\maxima)\s*(\.*)\s*(\*\s*(\d+(/\d+`
underlies this predicate.

Return boolean.

resttools.iterate_rests_backward_in_expr

`abjad.tools.resttools.`**`iterate_rests_backward_in_expr`**(*expr*, *start=0*, *stop=None*)

New in version 1.1.2. Iterate rests backward in *expr*:

```
abjad> staff = Staff("<e' g' c''>8 a'8 r8 <d' f' b'>8 r2")

abjad> f(staff)
\new Staff {
   <e' g' c''>8
   a'8
   r8
   <d' f' b'>8
   r2
}

abjad> for rest in resttools.iterate_rests_backward_in_expr(staff):
...   rest
Rest('r2')
Rest('r8')
```

Ignore threads.

Return generator.

resttools.iterate_rests_forward_in_expr

`abjad.tools.resttools.`**`iterate_rests_forward_in_expr`**(*expr*, *start=0*, *stop=None*)

New in version 1.1.2. Iterate rests forward in *expr*:

```
abjad> staff = Staff("<e' g' c''>8 a'8 r8 <d' f' b'>8 r2")

abjad> f(staff)
\new Staff {
   <e' g' c''>8
   a'8
   r8
   <d' f' b'>8
   r2
}

abjad> for rest in resttools.iterate_rests_forward_in_expr(staff):
...   rest
Rest('r8')
Rest('r2')
```

Ignore threads.

Return generator.

resttools.make_multi_measure_rests

`abjad.tools.resttools.make_multi_measure_rests(duration_tokens)`

New in version 1.1.2. Make multi-measure rests from *duration_tokens*:

```
abjad> resttools.make_multi_measure_rests([(4, 4), (7, 4)])
[MultiMeasureRest('R1'), MultiMeasureRest('R1..')]
```

Return list.

resttools.make_repeated_rests_from_time_signature

`abjad.tools.resttools.make_repeated_rests_from_time_signature(time_signature)`

New in version 1.1.2. Make repeated rests from *time_signature*:

```
abjad> resttools.make_repeated_rests_from_time_signature((5, 32))
[Rest('r32'), Rest('r32'), Rest('r32'), Rest('r32'), Rest('r32')]
```

Return list of newly constructed rests.

resttools.make_repeated_rests_from_time_signatures

`abjad.tools.resttools.make_repeated_rests_from_time_signatures(time_signatures)`

Make repated rests from *time_signatures*:

```
resttools.make_repeated_rests_from_time_signatures([(2, 8), (3, 32)])
[[Rest('r8'), Rest('r8')], [Rest('r32'), Rest('r32'), Rest('r32')]]
```

Return two-dimensional list of newly constructed rest lists.

Use `seqtools.flatten_sequence( )` to flatten output if required.

resttools.make_rests

`abjad.tools.resttools.make_rests(duration_tokens, direction='big-endian', tied=False)`

New in version 1.1.1. Make rests.

Make big-endian rests:

```
abjad> resttools.make_rests([(5, 16), (9, 16)], direction = 'big-endian')
[Rest('r4'), Rest('r16'), Rest('r2'), Rest('r16')]
```

Make little-endian rests:

```
abjad> resttools.make_rests([(5, 16), (9, 16)], direction = 'little-endian')
[Rest('r16'), Rest('r4'), Rest('r16'), Rest('r2')]
```

Make tied rests:

```
abjad> voice = Voice(resttools.make_rests([(5, 16), (9, 16)], tied = True))

abjad> f(voice)
\new Voice {
   r4 ~
   r16
   r2 ~
   r16
}
```

Return list of rests. Changed in version 1.1.2: renamed `construct.rests( )` to `resttools.make_rests( )`.

resttools.set_vertical_positioning_pitch_on_rest

`abjad.tools.resttools.`**`set_vertical_positioning_pitch_on_rest`**`(`*rest*, *pitch*`)`

New in version 1.1.2. Set vertical positioning *pitch* on *rest*:

```
abjad> rest = Rest((1, 4))

abjad> resttools.set_vertical_positioning_pitch_on_rest(rest, "d''")
Rest('r4')

abjad> f(rest)
d''4 \rest
```

Raise type error when *rest* is not a rest.

Return *rest*.

resttools.yield_groups_of_rests_in_sequence

`abjad.tools.resttools.`**`yield_groups_of_rests_in_sequence`**`(`*sequence*`)`

New in version 1.1.2. Yield groups of rests in *sequence*:

```
abjad> staff = Staff("c'8 d'8 r8 r8 <e' g'>8 <f' a'>8 g'8 a'8 r8 r8 <b' d''>8 <c'' e''>8")

abjad> f(staff)
\new Staff {
   c'8
   d'8
   r8
   r8
   <e' g'>8
   <f' a'>8
   g'8
   a'8
   r8
   r8
   <b' d''>8
   <c'' e''>8
}

abjad> for rest in resttools.yield_groups_of_rests_in_sequence(staff):
...      rest
...
(Rest('r8'), Rest('r8'))
(Rest('r8'), Rest('r8'))
```

Return generator.

schemetools

schemetools.SchemeAssociativeList

class `abjad.tools.schemetools.`**`SchemeAssociativeList`**

Bases: `tuple`, `abjad.core._Immutable._Immutable._Immutable` New in version 1.1.2. Abjad model of Scheme associative list:

```
abjad> schemetools.SchemeAssociativeList(('space', 2), ('padding', 0.5))
SchemeAssociativeList(SchemePair('space', 2), SchemePair('padding', 0.5))
```

Scheme associative lists are immutable.

format

LilyPond input format of Scheme associative list:

```
abjad> scheme_associative_list = schemetools.SchemeAssociativeList(('space', 2), ('padding',
abjad> scheme_associative_list.format
"#'((space . 2) (padding . 0.5))"
```

Return string.

schemetools.SchemeBoolean

class abjad.tools.schemetools.**SchemeBoolean**

Bases: abjad.core._Immutable._Immutable._Immutable

Abjad model of Scheme boolean:

```
abjad> schemetools.SchemeBoolean(True)
SchemeBoolean(True)
```

Scheme variables are immutable.

arg

format

LilyPond input format of Scheme boolean:

```
abjad> scheme_boolean = schemetools.SchemeBoolean(True)
abjad> scheme_boolean.format
'##t'
```

Return string.

schemetools.SchemeColor

class abjad.tools.schemetools.**SchemeColor**

Bases: abjad.core._StrictComparator._StrictComparator._StrictComparator, abjad.core._Immutable._Immutable._Immutable

Abjad model of Scheme color:

```
abjad> schemetools.SchemeColor('ForestGreen')
SchemeColor('ForestGreen')
```

Scheme colors are immutable.

format

LilyPond input format of Scheme color:

```
abjad> scheme_color = schemetools.SchemeColor('ForestGreen')
abjad> scheme_color.format
"#(x11-color 'ForestGreen)"
```

Return string.

schemetools.SchemeFunction

class abjad.tools.schemetools.**SchemeFunction**

Bases: abjad.core._StrictComparator._StrictComparator._StrictComparator, abjad.core._Immutable._Immutable._Immutable

Abjad model of Scheme function:

```
abjad> schemetools.SchemeFunction('magstep', -3)
SchemeFunction('magstep', -3)
```

Scheme functions are immutable.

format

LilyPond input format of Scheme function:

```
abjad> scheme_function = schemetools.SchemeFunction('magstep', -3)
abjad> scheme_function.format
'#(magstep -3)'
```

Return string.

schemetools.SchemeMoment

class abjad.tools.schemetools.**SchemeMoment**

Bases: abjad.core._StrictComparator._StrictComparator._StrictComparator, abjad.core._Immutable._Immutable._Immutable

Abjad model of LilyPond moment:

```
abjad> schemetools.SchemeMoment(1, 68)
SchemeMoment(1, 68)
```

Initialize scheme moments with a single fraction, two integers or another scheme moment.

Scheme moments are immutable.

duration

Duration of scheme moment:

```
abjad> scheme_moment = schemetools.SchemeMoment(1, 68)
abjad> scheme_moment.duration
Fraction(1, 68)
```

Return duration.

format

LilyPond input format of scheme moment:

```
abjad> scheme_moment = schemetools.SchemeMoment(1, 68)
abjad> scheme_moment.format
'#(ly:make-moment 1 68)'
```

Return string.

schemetools.SchemeNumber

class abjad.tools.schemetools.**SchemeNumber**

Bases: abjad.core._Immutable._Immutable._Immutable

Abjad model of Scheme number:

```
abjad> schemetools.SchemeNumber(1.1)
SchemeNumber(1.1...)
```

Scheme numbers are immutable.

format

LilyPond input format of Scheme number:

```
abjad> scheme_number = schemetools.SchemeNumber(1.1)
abjad> scheme_number.format
'#1.1'
```

Return string.

number

schemetools.SchemePair

class `abjad.tools.schemetools.`**`SchemePair`**

Bases: `tuple`, `abjad.core._Immutable._Immutable._Immutable`

Abjad model of Scheme pair:

```
abjad> schemetools.SchemePair('spacing', 4)
SchemePair('spacing', 4)
```

Initialize Scheme pairs with a tuple, two separate values or another Scheme pair.

Scheme pairs are immutable.

format

LilyPond input format of Scheme pair:

```
abjad> scheme_pair = schemetools.SchemePair('spacing', 4)
abjad> scheme_pair.format
"#'(spacing . 4)"
```

Return string.

schemetools.SchemeString

class `abjad.tools.schemetools.`**`SchemeString`**

Bases: `abjad.core._StrictComparator._StrictComparator._StrictComparator`, `abjad.core._Immutable._Immutable._Immutable`

Abjad model of Scheme string:

```
abjad> schemetools.SchemeString('grace')
SchemeString('grace')
```

Scheme strings are immutable.

format

LilyPond input format of Scheme string:

```
abjad> scheme_string = schemetools.SchemeString('grace')
abjad> scheme_string.format
'#"grace"'
```

Return string.

schemetools.SchemeVariable

class `abjad.tools.schemetools.SchemeVariable`

Bases: `abjad.core._StrictComparator._StrictComparator._StrictComparator`, `abjad.core._Immutable._Immutable._Immutable`

Abjad model of Scheme variable:

```
abjad> schemetools.SchemeVariable('grace')
SchemeVariable('grace')
```

Scheme variables are immutable.

format

LilyPond input format of Scheme variable:

abjad> scheme_variable = schemetools.SchemeVariable('UP') abjad> scheme_variable.format '#UP'

Return string.

schemetools.SchemeVector

class `abjad.tools.schemetools.SchemeVector`

Bases: `tuple`, `abjad.core._Immutable._Immutable._Immutable` New in version 1.1.2. Abjad model of Scheme vector:

```
abjad> schemetools.SchemeVector(True, True, False)
SchemeVector(True, True, False)
```

Scheme vectors and Scheme vector constants differ in only their LilyPond input format.

Scheme vectors are immutable.

format

LilyPond input format of Scheme vector:

```
abjad> scheme_vector = schemetools.SchemeVector(True, True, False)
abjad> scheme_vector.format
"#'(#t #t #f)"
```

Return string.

schemetools.SchemeVectorConstant

class `abjad.tools.schemetools.SchemeVectorConstant`

Bases: `tuple`, `abjad.core._Immutable._Immutable._Immutable` New in version 1.1.2. Abjad model of Scheme vector constant:

```
abjad> schemetools.SchemeVectorConstant(True, True, False)
SchemeVectorConstant(True, True, False)
```

Scheme vectors and Scheme vector constants differ in only their LilyPond input format.

Scheme vector constants are immutable.

format

LilyPond input format of scheme vector constant:

```
abjad> scheme_vector_constant = schemetools.SchemeVectorConstant(True, True, False)
abjad> scheme_vector_constant.format
"#'#(#t #t #f)"
```

Return string.

scoretools

scoretools.GrandStaff

class `abjad.tools.scoretools.`**`GrandStaff`**(*music*)

Bases: `abjad.tools.scoretools.StaffGroup.StaffGroup.StaffGroup`

Abjad model of grand staff:

```
abjad> staff_1 = Staff("c'4 d'4 e'4 f'4 g'1")
abjad> staff_2 = Staff("g2 f2 e1")

abjad> grand_staff = scoretools.GrandStaff([staff_1, staff_2])

abjad> f(grand_staff)
\new GrandStaff <<
   \new Staff {
      c'4
      d'4
      e'4
      f'4
      g'1
   }
   \new Staff {
      g2
      f2
      e1
   }
>>
```

Return grand staff.

scoretools.PianoStaff

class `abjad.tools.scoretools.`**`PianoStaff`**(*music*)

Bases: `abjad.tools.scoretools.StaffGroup.StaffGroup.StaffGroup`

Abjad model of piano staff:

```
abjad> staff_1 = Staff("c'4 d'4 e'4 f'4 g'1")
abjad> staff_2 = Staff("g2 f2 e1")

abjad> piano_staff = scoretools.PianoStaff([staff_1, staff_2])

abjad> f(piano_staff)
\new PianoStaff <<
   \new Staff {
      c'4
      d'4
```

```
         e'4
         f'4
         g'1
      }
      \new Staff {
         g2
         f2
         e1
      }
   >>
```

Return piano staff.

scoretools.StaffGroup

class `abjad.tools.scoretools.`**`StaffGroup`**(*music=[ ]*, ***kwargs*)

Bases: `abjad.components._Context._Context._Context`

Abjad model of staff group:

```
abjad> staff_1 = Staff("c'4 d'4 e'4 f'4 g'1")
abjad> staff_2 = Staff("g2 f2 e1")

abjad> staff_group = scoretools.StaffGroup([staff_1, staff_2])

abjad> f(staff_group)
\new StaffGroup <<
   \new Staff {
      c'4
      d'4
      e'4
      f'4
      g'1
   }
   \new Staff {
      g2
      f2
      e1
   }
>>
```

Return staff group.

scoretools.add_double_bar_to_end_of_score

`abjad.tools.scoretools.`**`add_double_bar_to_end_of_score`**(*score*)

New in version 1.1.2. Add double bar to end of *score*:

```
abjad> staff = Staff("c'4 d'4 e'4 f'4")

abjad> scoretools.add_double_bar_to_end_of_score(staff)
LilyPondCommandMark('bar "|."')(f'4)

abjad> f(staff)
\new Staff {
   c'4
```

```
   d'4
   e'4
   f'4
   \bar "|."
}
```

Return double bar.

scoretools.add_markup_to_end_of_score

abjad.tools.scoretools.**add_markup_to_end_of_score**(*score*, *markup*, *extra_offset=None*)

New in version 1.1.2. Add *markup* to end of *score*:

```
abjad> staff = Staff("c'4 d'4 e'4 f'4")
abjad> markup = r'\italic \right-column { "Bremen - Boston - Los Angeles." "Jul 2010 - May 2011.
abjad> markup = markuptools.Markup(markup, 'down')
abjad> scoretools.add_markup_to_end_of_score(staff, markup, (4, -2))
Markup('\\italic \\right-column { "Bremen - Boston - Los Angeles." "Jul 2010 - May 2011." }', 'd
```

```
abjad> f(staff)
\new Staff {
   c'4
   d'4
   e'4
   \once \override TextScript #'extra-offset = #'(4 . -2)
   f'4 _ \markup { \italic \right-column { "Bremen - Boston - Los Angeles." "Jul 2010 - May 2011
}
```

Return *markup*.

scoretools.get_first_score_in_improper_parentage_of_component

abjad.tools.scoretools.**get_first_score_in_improper_parentage_of_component**(*component*)

New in version 1.1.2. Get first score in improper parentage of *component*:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> score = Score([staff])
```

```
abjad> f(score)
\new Score <<
   \new Staff {
      c'8
      d'8
      e'8
      f'8
   }
>>
```

```
abjad> scoretools.get_first_score_in_improper_parentage_of_component(score.leaves[0])
Score<<1>>
```

Return score or none.

scoretools.get_first_score_in_proper_parentage_of_component

`abjad.tools.scoretools.get_first_score_in_proper_parentage_of_component(component)`

New in version 1.1.2. Get first score in proper parentage of *component*:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> score = Score([staff])

abjad> f(score)
\new Score <<
   \new Staff {
      c'8
      d'8
      e'8
      f'8
   }
>>

abjad> scoretools.get_first_score_in_proper_parentage_of_component(score.leaves[0])
Score<<1>>
```

Return score or none.

scoretools.iterate_scores_backward_in_expr

`abjad.tools.scoretools.iterate_scores_backward_in_expr(expr, start=0, stop=None)`

New in version 1.1.2. Iterate scores backward in *expr*:

```
abjad> score_1 = Score([Staff("c'8 d'8 e'8 f'8")])
abjad> score_2 = Score([Staff("c'1"), Staff("g'1")])
abjad> scores = [score_1, score_2]

abjad> for score in scoretools.iterate_scores_backward_in_expr(scores):
...   score
Score<<2>>
Score<<1>>
```

Ignore threads.

Return generator.

scoretools.iterate_scores_forward_in_expr

`abjad.tools.scoretools.iterate_scores_forward_in_expr(expr, start=0, stop=None)`

New in version 1.1.2. Iterate scores forward in *expr*:

```
abjad> score_1 = Score([Staff("c'8 d'8 e'8 f'8")])
abjad> score_2 = Score([Staff("c'1"), Staff("g'1")])
abjad> scores = [score_1, score_2]

abjad> for score in scoretools.iterate_scores_forward_in_expr(scores):
...   score
Score<<1>>
Score<<2>>
```

Ignore threads.

Return generator.

scoretools.make_empty_piano_score

`abjad.tools.scoretools.`**`make_empty_piano_score`**`()`

New in version 1.1.1. Make empty piano score:

```
abjad> score, treble, bass = scoretools.make_empty_piano_score( )

abjad> f(score)
\new Score <<
  \new PianoStaff <<
     \context Staff = "treble" {
        \clef "treble"
     }
     \context Staff = "bass" {
        \clef "bass"
     }
  >>
>>
```

Return score, treble staff, bass staff. Changed in version 1.1.2: renamed `scoretools.make_piano_staff( )` to `scoretools.make_empty_piano_score( )`.

scoretools.make_piano_score_from_leaves

`abjad.tools.scoretools.`**`make_piano_score_from_leaves`**`(`*leaves*`)`

New in version 1.1.2. Make piano score from *leaves*:

```
abjad> notes = [Note(x, (1, 4)) for x in [-12, 37, -10, 2, 4, 17]]
abjad> score, treble_staff, bass_staff = scoretools.make_piano_score_from_leaves(notes)

abjad> f(score)
\new Score <<
  \new PianoStaff <<
     \context Staff = "treble" {
        \clef "treble"
        r4
        cs''''4
        r4
        d'4
        e'4
        f''4
     }
     \context Staff = "bass" {
        \clef "bass"
        c4
        r4
        d4
        r4
        r4
        r4
     }
  >>
>>
```

Return score, treble staff, bass staff.

scoretools.make_piano_sketch_score_from_leaves

abjad.tools.scoretools.**make_piano_sketch_score_from_leaves**(*leaves*)

New in version 1.1.2. Make piano sketch score from *leaves*:

```
abjad> notes = notetools.make_notes([-12, -10, -8, -7, -5, 0, 2, 4, 5, 7], [(1, 4)])
abjad> score, treble_staff, bass_staff = scoretools.make_piano_sketch_score_from_leaves(notes)

abjad> f(score)
\new Score \with {
   \override BarLine #'stencil = ##f
   \override BarNumber #'transparent = ##t
   \override SpanBar #'stencil = ##f
   \override TimeSignature #'transparent = ##t
} <<
   \new PianoStaff <<
      \context Staff = "treble" {
         \clef "treble"
         #(set-accidental-style 'forget)
         r4
         r4
         r4
         r4
         r4
         c'4
         d'4
         e'4
         f'4
         g'4
      }
      \context Staff = "bass" {
         \clef "bass"
         #(set-accidental-style 'forget)
         c4
         d4
         e4
         f4
         g4
         r4
         r4
         r4
         r4
         r4
      }
   >>
>>
```

Make time signatures and bar numbers transparent.

Do not print bar lines or span bars.

Set all staff accidental styles to forget.

Return score, treble staff, bass staff.

scoretools.make_pitch_array_score_from_pitch_arrays

abjad.tools.scoretools.**make_pitch_array_score_from_pitch_arrays**(*pitch_arrays*)

New in version 1.1.2. Make pitch-array score from *pitch_arrays*:

```
abjad> from abjad.tools import pitcharraytools

abjad> array_1 = pitcharraytools.PitchArray([
...    [1, (2, 1), ([-2, -1.5], 2)],
...    [(7, 2), (6, 1), 1]])

abjad> array_2 = pitcharraytools.PitchArray([
...    [1, 1, 1],
...    [1, 1, 1]])

abjad> score = scoretools.make_pitch_array_score_from_pitch_arrays([array_1, array_2])

abjad> f(score)
\new Score <<
        \new StaffGroup <<
                \new Staff {
                        {
                                \time 4/8
                                r8
                                d'8
                                <bf bqf>4
                        }
                        {
                                \time 3/8
                                r8
                                r8
                                r8
                        }
                }
                \new Staff {
                        {
                                \time 4/8
                                g'4
                                fs'8
                                r8
                        }
                        {
                                \time 3/8
                                r8
                                r8
                                r8
                        }
                }
        >>
>>
```

Create one staff per pitch-array row.

Return score.

seqtools

seqtools.CyclicList

class `abjad.tools.seqtools.`**`CyclicList`**

Bases: `list` New in version 1.1.2. Abjad model of cyclic list:

```
abjad> cyclic_list = seqtools.CyclicList('abcd')

abjad> cyclic_list
['a', 'b', 'c', 'd']

abjad> for x in range(8):
...     print x, cyclic_list[x]
...
0 a
1 b
2 c
3 d
4 a
5 b
6 c
7 d
```

Cyclic lists overload the item-getting method of built-in lists.

Cyclic lists return a value for any integer index.

Cyclic lists otherwise behave exactly like built-in lists.

seqtools.CyclicMatrix

class `abjad.tools.seqtools.`**`CyclicMatrix`**`(*args, **kwargs)`

Bases: `abjad.tools.seqtools.Matrix.Matrix.Matrix` New in version 1.1.2. Abjad model of cyclic matrix.

Initialize from rows:

```
abjad> cyclic_matrix = seqtools.CyclicMatrix([[0, 1, 2, 3], [10, 11, 12, 13], [20, 21, 22, 23]])

abjad> cyclic_matrix
CyclicMatrix(3x4)

abjad> cyclic_matrix[2]
(20, 21, 22, 23)

abjad> cyclic_matrix[2][2]
22

abjad> cyclic_matrix[99]
(0, 1, 2, 3)

abjad> cyclic_matrix[99][99]
3
```

Initialize from columns:

```
abjad> cyclic_matrix = seqtools.CyclicMatrix(columns = [[0, 10, 20], [1, 11, 21], [2, 12, 22], [

abjad> cyclic_matrix
CyclicMatrix(3x4)

abjad> cyclic_matrix[2]
(20, 21, 22, 23)
```

```
abjad> cyclic_matrix[2][2]
22

abjad> cyclic_matrix[99]
(0, 1, 2, 3)

abjad> cyclic_matrix[99][99]
3
```

CyclicMatrix implements only item retrieval in this revision.

Concatenation and division remain to be implemented.

Standard transforms of linear algebra remain to be implemented.

columns

Read-only columns:

```
abjad> cyclic_matrix = seqtools.CyclicMatrix([[0, 1, 2, 3], [10, 11, 12, 13], [20, 21, 22, 2

abjad> cyclic_matrix.columns
((0, 10, 20), (1, 11, 21), (2, 12, 22), (3, 13, 23))
```

Return cyclic tuple.

rows

Read-only rows:

```
abjad> cyclic_matrix = seqtools.CyclicMatrix([[0, 1, 2, 3], [10, 11, 12, 13], [20, 21, 22, 2

abjad> cyclic_matrix.rows
((0, 1, 2, 3), (10, 11, 12, 13), (20, 21, 22, 23))
```

Return cyclic tuple.

seqtools.CyclicTuple

class `abjad.tools.seqtools.`**`CyclicTuple`**

Bases: `tuple` New in version 1.1.2. Abjad model of cyclic tuple:

```
abjad> cyclic_tuple = seqtools.CyclicTuple('abcd')

abjad> cyclic_tuple
('a', 'b', 'c', 'd')

abjad> for x in range(8):
...     print x, cyclic_tuple[x]
...
0 a
1 b
2 c
3 d
4 a
5 b
6 c
7 d
```

Cyclic tuples overload the item-getting method of built-in tuples.

Cyclic tuples return a value for any integer index.

Cyclic tuples otherwise behave exactly like built-in tuples.

seqtools.Matrix

class `abjad.tools.seqtools.`**`Matrix`**`(`**args*, ***kwargs*`)`

Bases: `object` New in version 1.1.2. Abjad model of matrix.

Initialize from rows:

```
abjad> matrix = seqtools.Matrix([[0, 1, 2, 3], [10, 11, 12, 13], [20, 21, 22, 23]])

abjad> matrix
Matrix(3x4)

abjad> matrix[:]
((0, 1, 2, 3), (10, 11, 12, 13), (20, 21, 22, 23))

abjad> matrix[2]
(20, 21, 22, 23)

abjad> matrix[2][0]
20
```

Initialize from columns:

```
abjad> matrix = seqtools.Matrix(columns = [[0, 10, 20], [1, 11, 21], [2, 12, 22], [3, 13, 23]])

abjad> matrix
Matrix(3x4)

abjad> matrix[:]
((0, 1, 2, 3), (10, 11, 12, 13), (20, 21, 22, 23))

abjad> matrix[2]
(20, 21, 22, 23)

abjad> matrix[2][0]
20
```

Matrix implements only item retrieval in this revision.

Concatenation and division remain to be implemented.

Standard transforms of linear algebra remain to be implemented.

`columns`

Read-only columns:

```
abjad> matrix = seqtools.Matrix([[0, 1, 2, 3], [10, 11, 12, 13], [20, 21, 22, 23]])

abjad> matrix.columns
((0, 10, 20), (1, 11, 21), (2, 12, 22), (3, 13, 23))
```

Return tuple.

`rows`

Read-only rows:

```
abjad> matrix = seqtools.Matrix([[0, 1, 2, 3], [10, 11, 12, 13], [20, 21, 22, 23]])
```

```
abjad> matrix.rows
((0, 1, 2, 3), (10, 11, 12, 13), (20, 21, 22, 23))
```

Return tuple.

seqtools.all_are_assignable_integers

`abjad.tools.seqtools.all_are_assignable_integers(expr)`

New in version 1.1.2. True when *expr* is a sequence and all elements in *expr* are notehead-assignable integers:

```
abjad> seqtools.all_are_assignable_integers([1, 2, 3, 4, 6, 7, 8, 12, 14, 15, 16])
True
```

True when *expr* is an empty sequence:

```
abjad> seqtools.all_are_assignable_integers([ ])
True
```

False otherwise:

```
abjad> seqtools.all_are_assignable_integers('foo')
False
```

Return boolean.

seqtools.all_are_equal

`abjad.tools.seqtools.all_are_equal(expr)`

New in version 1.1.2. True when *expr* is a sequence and all elements in *expr* are equal:

```
abjad> seqtools.all_are_equal([99, 99, 99, 99, 99, 99])
True
```

True when *expr* is an empty sequence:

```
abjad> seqtools.all_are_equal([ ])
True
```

False otherwise:

```
abjad> seqtools.all_are_equal(17)
False
```

Return boolean.

seqtools.all_are_integer_equivalent_numbers

`abjad.tools.seqtools.all_are_integer_equivalent_numbers(expr)`

New in version 1.1.2. True when *expr* is a sequence and all elements in *expr* are integer-equivalent numbers:

```
abjad> seqtools.all_are_integer_equivalent_numbers([1, 2, 3.0, Fraction(4, 1)])
True
```

Otherwise false:

```
abjad> seqtools.all_are_integer_equivalent_numbers([1, 2, 3.5, 4])
False
```

Return boolean.

seqtools.all_are_nonnegative_integer_equivalent_numbers

`abjad.tools.seqtools.`**`all_are_nonnegative_integer_equivalent_numbers`**`(`*expr*`)`

New in version 1.1.2. True *expr* is a sequence and when all elements in *expr* are nonnegative integer-equivalent numbers. Otherwise false:

```
abjad> seqtools.all_are_nonnegative_integer_equivalent_numbers([0, 0.0, Fraction(0), 2, 2.0, Fra
True
```

Return boolean.

seqtools.all_are_nonnegative_integer_powers_of_two

`abjad.tools.seqtools.`**`all_are_nonnegative_integer_powers_of_two`**`(`*expr*`)`

New in version 1.1.2. True when *expr* is a sequence and all elements in *expr* are nonnegative integer powers of two:

```
abjad> seqtools.all_are_nonnegative_integer_powers_of_two([0, 1, 1, 1, 2, 4, 32, 32])
True
```

True when *expr* is an empty sequence:

```
abjad> seqtools.all_are_nonnegative_integer_powers_of_two([ ])
True
```

False otherwise:

```
abjad> seqtools.all_are_nonnegative_integer_powers_of_two(17)
False
```

Return boolean.

seqtools.all_are_nonnegative_integers

`abjad.tools.seqtools.`**`all_are_nonnegative_integers`**`(`*expr*`)`

New in version 1.1.2. True when *expr* is a sequence and all elements in *expr* are nonnegative integers:

```
abjad> seqtools.all_are_nonnegative_integers([0, 1, 2, 99])
True
```

Otherwise false:

```
abjad> seqtools.all_are_nonnegative_integers([0, 1, 2, -99])
False
```

Return boolean.

seqtools.all_are_numbers

`abjad.tools.seqtools.`**`all_are_numbers`**`(`*expr*`)`

New in version 1.1.1. True when *expr* is a sequence and all elements in *expr* are numbers:

```
abjad> seqtools.all_are_numbers([1, 2, 3.0, Fraction(13, 8)])
True
```

True when *expr* is an empty sequence:

```
abjad> seqtools.all_are_numbers([ ])
True
```

False otherwise:

```
abjad> seqtools.all_are_numbers(17)
False
```

Return boolean. Changed in version 1.1.2: renamed `seqtools.is_numeric( )` to `seqtools.all_are_numbers( )`.

seqtools.all_are_positive_integer_equivalent_numbers

`abjad.tools.seqtools.`**`all_are_positive_integer_equivalent_numbers`**`(`*expr*`)`

New in version 1.1.2. True when *expr* is a sequence and all elements in *expr* are positive integer-equivalent numbers. Otherwise false:

```
abjad> seqtools.all_are_positive_integer_equivalent_numbers([Fraction(4, 2), 2.0, 2])
True
```

Return boolean.

seqtools.all_are_positive_integers

`abjad.tools.seqtools.`**`all_are_positive_integers`**`(`*expr*`)`

New in version 1.1.2. True when *expr* is a sequence and all elements in *expr* are positive integers:

```
abjad> seqtools.all_are_positive_integers([1, 2, 3, 99])
True
```

Otherwise false:

```
abjad> seqtools.all_are_positive_integers(17)
False
```

Return boolean.

seqtools.all_are_unequal

`abjad.tools.seqtools.`**`all_are_unequal`**`(`*expr*`)`

New in version 1.1.1. True when *expr* is a sequence all elements in *expr* are unequal:

```
abjad> seqtools.all_are_unequal([1, 2, 3, 4, 9])
True
```

True when *expr* is an empty sequence:

```
abjad> seqtools.all_are_unequal([ ])
True
```

False otherwise:

```
abjad> seqtools.all_are_unequal(17)
False
```

Return boolean. Changed in version 1.1.2: renamed `seqtools.is_unique( )` to `seqtools.all_are_unequal( )`.

seqtools.count_length_two_runs_in_sequence

abjad.tools.seqtools.**count_length_two_runs_in_sequence**(*sequence*)

New in version 1.1.1. Count length-2 runs in *sequence*:

```
abjad> seqtools.count_length_two_runs_in_sequence([0, 0, 1, 1, 1, 2, 3, 4, 5])
3
```

Return nonnegative integer. Changed in version 1.1.2: renamed `seqtools.count_repetitions( )` to `seqtools.count_length_two_runs_in_sequence( )`.

seqtools.divide_sequence_elements_by_greatest_common_divisor

abjad.tools.seqtools.**divide_sequence_elements_by_greatest_common_divisor**(*sequence*)

New in version 1.1.2. Divide *sequence* elements by greatest common divisor:

```
abjad> seqtools.divide_sequence_elements_by_greatest_common_divisor([2, 2, -8, -16])
[1, 1, -4, -8]
```

Allow negative *sequence* elements.

Raise type error on noninteger *sequence* elements.

Raise not implemented error when `0` in *sequence*.

Return new *sequence* object.

seqtools.flatten_sequence

abjad.tools.seqtools.**flatten_sequence**(*sequence*, *klasses=None*, *depth=-1*)

New in version 1.1.1. Flatten *sequence*:

```
abjad> seqtools.flatten_sequence([1, [2, 3, [4]], 5, [6, 7, [8]]])
[1, 2, 3, 4, 5, 6, 7, 8]
```

Flatten *sequence* to depth `1`:

```
abjad> seqtools.flatten_sequence([1, [2, 3, [4]], 5, [6, 7, [8]]], depth = 1)
[1, 2, 3, [4], 5, 6, 7, [8]]
```

Flatten *sequence* to depth `2`:

```
abjad> seqtools.flatten_sequence([1, [2, 3, [4]], 5, [6, 7, [8]]], depth = 2)
[1, 2, 3, 4, 5, 6, 7, 8]
```

Leave *sequence* unchanged.

Return newly constructed *sequence* object. Changed in version 1.1.2: renamed `listtools.flatten( )` to `seqtools.flatten_sequence( )`.

seqtools.flatten_sequence_at_indices

abjad.tools.seqtools.**flatten_sequence_at_indices**(*sequence*, *indices*, *klasses=None*, *depth=-1*)

New in version 1.1.2. Flatten *sequence* at *indices*:

```
abjad> seqtools.flatten_sequence_at_indices([0, 1, [2, 3, 4], [5, 6, 7]], [3])
[0, 1, [2, 3, 4], 5, 6, 7]
```

Flatten *sequence* at negative *indices*:

```
abjad> seqtools.flatten_sequence_at_indices([0, 1, [2, 3, 4], [5, 6, 7]], [-1])
[0, 1, [2, 3, 4], 5, 6, 7]
```

Leave *sequence* unchanged.

Return newly constructed *sequence* object.

seqtools.get_indices_of_sequence_elements_equal_to_true

abjad.tools.seqtools.**get_indices_of_sequence_elements_equal_to_true**(*sequence*)

New in version 1.1.1. Get indices of *sequence* elements equal to true:

```
abjad> seqtools.get_indices_of_sequence_elements_equal_to_true([0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1,
(3, 4, 5, 9, 10, 11, 12)
```

Return newly constructed tuple of zero or more nonnegative integers. Changed in version 1.1.2: renamed `listtools.true_indices( )` to `seqtools.get_indices_of_sequence_elements_equal_to_true( )`.

seqtools.get_sequence_degree_of_rotational_symmetry

abjad.tools.seqtools.**get_sequence_degree_of_rotational_symmetry**(*sequence*)

New in version 1.1.2. Change *sequence* to degree of rotational symmetry:

```
abjad> seqtools.get_sequence_degree_of_rotational_symmetry([1, 2, 3, 4, 5, 6])
1

abjad> seqtools.get_sequence_degree_of_rotational_symmetry([1, 2, 3, 1, 2, 3])
2

abjad> seqtools.get_sequence_degree_of_rotational_symmetry([1, 2, 1, 2, 1, 2])
3

abjad> seqtools.get_sequence_degree_of_rotational_symmetry([1, 1, 1, 1, 1, 1])
6
```

Return positive integer.

seqtools.get_sequence_element_at_cyclic_index

abjad.tools.seqtools.**get_sequence_element_at_cyclic_index**(*sequence*, *index*)

New in version 1.1.2. Get *sequence* element at nonnegative cyclic *index*:

```
abjad> for index in range(10):
...     print '%s\t%s' % (index, seqtools.get_sequence_element_at_cyclic_index('string', index))
...
0   s
1   t
2   r
3   i
4   n
5   g
6   s
7   t
```

```
8   r
9   i
```

Get *sequence* element at negative cyclic *index*:

```
abjad> for index in range(1, 11):
...      print '%s\t%s' % (-index, seqtools.get_sequence_element_at_cyclic_index('string', -index
...
-1     g
-2     n
-3     i
-4     r
-5     t
-6     s
-7     g
-8     n
-9     i
-10    r
```

Return reference to *sequence* element.

seqtools.get_sequence_elements_at_indices

abjad.tools.seqtools.**get_sequence_elements_at_indices**(*sequence*, *indices*)
New in version 1.1.2. Get *sequence* elements at *indices*:

```
abjad> seqtools.get_sequence_elements_at_indices('string of text', (2, 3, 10, 12))
('r', 'i', 't', 'x')
```

Return newly constructed tuple of references to *sequence* elements.

seqtools.get_sequence_elements_frequency_distribution

abjad.tools.seqtools.**get_sequence_elements_frequency_distribution**(*sequence*)
New in version 1.1.2. Get *sequence* elements frequency distribution:

```
abjad> seqtools.get_sequence_elements_frequency_distribution([1, 3, 3, 3, 2, 1, 1, 2, 3, 3, 1, 2
[(1, 4), (2, 3), (3, 5)]
```

Return list of element / count pairs.

seqtools.get_sequence_period_of_rotation

abjad.tools.seqtools.**get_sequence_period_of_rotation**(*sequence*, *n*)
New in version 1.1.2. Change *sequence* to period of rotation:

```
abjad> seqtools.get_sequence_period_of_rotation([1, 2, 3, 1, 2, 3], 1)
3

abjad> seqtools.get_sequence_period_of_rotation([1, 2, 3, 1, 2, 3], 2)
3

abjad> seqtools.get_sequence_period_of_rotation([1, 2, 3, 1, 2, 3], 3)
1
```

Return positive integer.

seqtools.increase_sequence_elements_at_indices_by_addenda

`abjad.tools.seqtools.`**`increase_sequence_elements_at_indices_by_addenda`**`(`*sequence, addenda, indices*`)`

New in version 1.1.1. Increase *sequence* by *addenda* at *indices*:

```
abjad> sequence = [1, 1, 2, 3, 5, 5, 1, 2, 5, 5, 6]
abjad> seqtools.increase_sequence_elements_at_indices_by_addenda(sequence, [0.5, 0.5], [0, 4, 8]
[1.5, 1.5, 2, 3, 5.5, 5.5, 1, 2, 5.5, 5.5, 6]
```

Return list. Changed in version 1.1.2: renamed `seqtools.increase_at_indices( )` to `seqtools.increase_sequence_elements_at_indices_by_addenda( )`.

seqtools.increase_sequence_elements_cyclically_by_addenda

`abjad.tools.seqtools.`**`increase_sequence_elements_cyclically_by_addenda`**`(`*sequence, addenda, shield=True, trim=True*`)`

New in version 1.1.1.. Increase *sequence* cyclically by *addenda*:

```
abjad> seqtools.increase_sequence_elements_cyclically_by_addenda(range(10), [10, -10], shield =
[10, -9, 12, -7, 14, -5, 16, -3, 18, -1]
```

Increase *sequence* cyclically by addenda and map nonpositive values to `1`:

```
abjad> seqtools.increase_sequence_elements_cyclically_by_addenda(range(10), [10, -10], shield =
[10, 1, 12, 1, 14, 1, 16, 1, 18, 1]
```

Return list. Changed in version 1.1.2: renamed `seqtools.increase_cyclic( )` to `seqtools.increase_sequence_elements_cyclically_by_addenda( )`.

seqtools.interlace_sequences

`abjad.tools.seqtools.`**`interlace_sequences`**`(`**sequences*`)`

New in version 1.1.1. Interlace *sequences*:

```
k = range(100, 103)
l = range(200, 201)
m = range(300, 303)
n = range(400, 408)
t = seqtools.interlace_sequences(k, l, m, n)
[100, 200, 300, 400, 101, 301, 401, 102, 302, 402, 403, 404, 405, 406, 407]
```

Return list. Changed in version 1.1.2: renamed `seqtools.interlace( )` to `seqtools.interlace_sequences( )`.

seqtools.is_monotonically_decreasing_sequence

`abjad.tools.seqtools.`**`is_monotonically_decreasing_sequence`**`(`*expr*`)`

New in version 1.1.2. True when *expr* is a sequence and the elements in *expr* decrease monotonically:

```
abjad> expr = [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
abjad> seqtools.is_monotonically_decreasing_sequence(expr)
True

abjad> expr = [3, 3, 3, 3, 3, 3, 3, 2, 1, 0]
abjad> seqtools.is_monotonically_decreasing_sequence(expr)
True

abjad> expr = [3, 3, 3, 3, 3, 3, 3, 3, 3, 3]
abjad> seqtools.is_monotonically_decreasing_sequence(expr)
True
```

False when *expr* is a sequence and the elements in *expr* do not decrease monotonically:

```
abjad> expr = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
abjad> seqtools.is_monotonically_decreasing_sequence(expr)
False

abjad> expr = [0, 1, 2, 3, 3, 3, 3, 3, 3, 3]
abjad> seqtools.is_monotonically_decreasing_sequence(expr)
False
```

True when *expr* is a sequence and *expr* is empty:

```
abjad> expr = [ ]
abjad> seqtools.is_monotonically_decreasing_sequence(expr)
True
```

False when *expr* is not a sequence:

```
abjad> seqtools.is_monotonically_decreasing_sequence(17)
False
```

Return boolean.

seqtools.is_monotonically_increasing_sequence

abjad.tools.seqtools.**is_monotonically_increasing_sequence**(*expr*)

New in version 1.1.2. True when *expr* is a sequence and the elements in *expr* increase monotonically:

```
abjad> expr = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
abjad> seqtools.is_monotonically_increasing_sequence(expr)
True

abjad> expr = [0, 1, 2, 3, 3, 3, 3, 3, 3, 3]
abjad> seqtools.is_monotonically_increasing_sequence(expr)
True

abjad> expr = [3, 3, 3, 3, 3, 3, 3, 3, 3, 3]
abjad> seqtools.is_monotonically_increasing_sequence(expr)
True
```

False when *expr* is a sequence and the elements in *expr* do not increase monotonically:

```
abjad> expr = [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
abjad> seqtools.is_monotonically_increasing_sequence(expr)
False
```

```
abjad> expr = [3, 3, 3, 3, 3, 3, 3, 2, 1, 0]
abjad> seqtools.is_monotonically_increasing_sequence(expr)
False
```

True when *expr* is a sequence and *expr* is empty:

```
abjad> expr = [ ]
abjad> seqtools.is_monotonically_increasing_sequence(expr)
True
```

False when *expr* is not a sequence:

```
abjad> seqtools.is_monotonically_increasing_sequence(17)
False
```

Return boolean.

seqtools.is_permutation

`abjad.tools.seqtools.is_permutation(expr, length=None)`

New in version 1.1.2. True when *expr* is a permutation:

```
abjad> seqtools.is_permutation([4, 5, 0, 3, 2, 1])
True
```

Otherwise false:

```
abjad> seqtools.is_permutation([1, 1, 5, 3, 2, 1])
False
```

True when *expr* is a permutation of first *length* nonnegative integers:

```
abjad> seqtools.is_permutation([4, 5, 0, 3, 2, 1], length = 6)
True
```

Otherwise false:

```
abjad> seqtools.is_permutation([4, 0, 3, 2, 1], length = 6)
False
```

Return boolean.

seqtools.is_repetition_free_sequence

`abjad.tools.seqtools.is_repetition_free_sequence(expr)`

New in version 1.1.2. True when *expr* is a sequence and *expr* is repetition free:

```
abjad> seqtools.is_repetition_free_sequence([0, 1, 2, 6, 7, 8])
True
```

False when *expr* is a sequence and *expr* is not repetition free:

```
abjad> seqtools.is_repetition_free_sequence([0, 1, 2, 2, 7, 8])
False
```

True when *expr* is an empty sequence:

```
abjad> seqtools.is_repetition_free_sequence([ ])
True
```

False *expr* is not a sequence:

```
abjad> seqtools.is_repetition_free_sequence(17)
False
```

Return boolean.

seqtools.is_restricted_growth_function

abjad.tools.seqtools.**is_restricted_growth_function**(*expr*)

New in version 1.1.2. True when *expr* is a sequence and *expr* meets the criteria for a restricted growth function:

```
abjad> seqtools.is_restricted_growth_function([1, 1, 1, 1])
True
```

```
abjad> seqtools.is_restricted_growth_function([1, 1, 1, 2])
True
```

```
abjad> seqtools.is_restricted_growth_function([1, 1, 2, 1])
True
```

```
abjad> seqtools.is_restricted_growth_function([1, 1, 2, 2])
True
```

Otherwise false:

```
abjad> seqtools.is_restricted_growth_function([1, 1, 1, 3])
False
```

```
abjad> seqtools.is_restricted_growth_function(17)
False
```

A restricted growth function is a sequence `l` such that `l[0] == 1` and such that `l[i] <= max(l[:i]) + 1` for `1 <= i <= len(l)`.

Return boolean.

seqtools.is_strictly_decreasing_sequence

abjad.tools.seqtools.**is_strictly_decreasing_sequence**(*expr*)

New in version 1.1.2. True when *expr* is a sequence and the elements in *expr* decrease strictly:

```
abjad> expr = [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
abjad> seqtools.is_strictly_decreasing_sequence(expr)
True
```

False when *expr* is a sequence and the elements in *expr* do not decrease strictly:

```
abjad> expr = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
abjad> seqtools.is_strictly_decreasing_sequence(expr)
False
```

```
abjad> expr = [0, 1, 2, 3, 3, 3, 3, 3, 3, 3]
abjad> seqtools.is_strictly_decreasing_sequence(expr)
False
```

```
abjad> expr = [3, 3, 3, 3, 3, 3, 3, 3, 3, 3]
abjad> seqtools.is_strictly_decreasing_sequence(expr)
False
```

```
abjad> expr = [3, 3, 3, 3, 3, 3, 3, 2, 1, 0]
abjad> seqtools.is_strictly_decreasing_sequence(expr)
False
```

True when *expr* is an empty sequence:

```
abjad> seqtools.is_strictly_decreasing_sequence([ ])
True
```

False *expr* is not a sequence:

```
abjad> seqtools.is_strictly_decreasing_sequence(17)
False
```

Return boolean.

seqtools.is_strictly_increasing_sequence

abjad.tools.seqtools.**is_strictly_increasing_sequence**(*expr*)

New in version 1.1.2. True when *expr* is a sequence and the elements in *expr* increase strictly:

```
abjad> expr = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
abjad> seqtools.is_strictly_increasing_sequence(expr)
True
```

False when *expr* is a sequence and the elements in *expr* do not increase strictly:

```
abjad> expr = [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
abjad> seqtools.is_strictly_increasing_sequence(expr)
False

abjad> expr = [3, 3, 3, 3, 3, 3, 3, 2, 1, 0]
abjad> seqtools.is_strictly_increasing_sequence(expr)
False

abjad> expr = [3, 3, 3, 3, 3, 3, 3, 3, 3, 3]
abjad> seqtools.is_strictly_increasing_sequence(expr)
False

abjad> expr = [0, 1, 2, 3, 3, 3, 3, 3, 3, 3]
abjad> seqtools.is_strictly_increasing_sequence(expr)
False
```

True when *expr* is an empty sequence:

```
abjad> seqtools.is_strictly_increasing_sequence([ ])
True
```

False when *expr* is not a sequence:

```
abjad> seqtools.is_strictly_increasing_sequence(17)
False
```

Return boolean.

seqtools.iterate_sequence_cyclically

abjad.tools.seqtools.**iterate_sequence_cyclically**(*sequence*, *step=1*, *start=0*, *length='inf'*)

New in version 1.1.1. Iterate *sequence* cyclically according to *step*, *start* and *length*:

```
abjad> sequence = [1, 2, 3, 4, 5, 6, 7]

abjad> list(seqtools.iterate_sequence_cyclically(sequence, length = 20))
[1, 2, 3, 4, 5, 6, 7, 1, 2, 3, 4, 5, 6, 7, 1, 2, 3, 4, 5, 6]

abjad> list(seqtools.iterate_sequence_cyclically(sequence, 2, length = 20))
[1, 3, 5, 7, 2, 4, 6, 1, 3, 5, 7, 2, 4, 6, 1, 3, 5, 7, 2, 4]

abjad> list(seqtools.iterate_sequence_cyclically(sequence, 2, 3, length = 20))
[4, 6, 1, 3, 5, 7, 2, 4, 6, 1, 3, 5, 7, 2, 4, 6, 1, 3, 5, 7]

abjad> list(seqtools.iterate_sequence_cyclically(sequence, -2, 5, length = 20))
[6, 4, 2, 7, 5, 3, 1, 6, 4, 2, 7, 5, 3, 1, 6, 4, 2, 7, 5, 3]
```

Changed in version 1.1.2: allows generator input.

```
abjad> list(seqtools.iterate_sequence_cyclically(xrange(1, 8), -2, 5, length = 20))
[6, 4, 2, 7, 5, 3, 1, 6, 4, 2, 7, 5, 3, 1, 6, 4, 2, 7, 5, 3]
```

Set *step* to jump size and direction across sequence.

Set *start* to the index of *sequence* where the function begins iterating.

Set *length* to number of elements to return. Set to `'inf'` to return infinitely.

Return generator. Changed in version 1.1.2: renamed `seqtools.phasor( )` to `seqtools.iterate_sequence_cyclically( )`.

seqtools.iterate_sequence_cyclically_from_start_to_stop

abjad.tools.seqtools.**iterate_sequence_cyclically_from_start_to_stop**(*sequence*, *start*, *stop*)

New in version 1.1.1. Iterate *sequence* cyclically from *start* to *stop*:

```
abjad> list(seqtools.iterate_sequence_cyclically_from_start_to_stop(range(20), 18, 10))
[18, 19, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Return generator of references to *sequence* elements. Changed in version 1.1.2: renamed `seqtools.get_cyclic( )` to `seqtools.iterate_sequence_cyclically_from_start_to_stop( )`.

seqtools.iterate_sequence_forward_and_backward_nonoverlapping

abjad.tools.seqtools.**iterate_sequence_forward_and_backward_nonoverlapping**(*sequence*)

New in version 1.1.2. Iterate *sequence* first forward and then backward, with first and last elements repeated:

```
abjad> list(seqtools.iterate_sequence_forward_and_backward_nonoverlapping([1, 2, 3, 4, 5]))
[1, 2, 3, 4, 5, 5, 4, 3, 2, 1]
```

Return generator.

seqtools.iterate_sequence_forward_and_backward_overlapping

abjad.tools.seqtools.**iterate_sequence_forward_and_backward_overlapping**(*sequence*)

New in version 1.1.2. Iterate *sequence* first forward and then backward, with first and last elements appearing only once:

```
abjad> list(seqtools.iterate_sequence_forward_and_backward_overlapping([1, 2, 3, 4, 5]))
[1, 2, 3, 4, 5, 4, 3, 2]
```

Return generator.

seqtools.iterate_sequence_nwise_cyclic

abjad.tools.seqtools.**iterate_sequence_nwise_cyclic**(*sequence*, *n*)

New in version 1.1.2. Iterate elements in *sequence* cyclically *n* at a time:

```
abjad> g = seqtools.iterate_sequence_nwise_cyclic(range(6), 3)
abjad> for n in range(10):
...   print g.next( )
(0, 1, 2)
(1, 2, 3)
(2, 3, 4)
(3, 4, 5)
(4, 5, 0)
(5, 0, 1)
(0, 1, 2)
(1, 2, 3)
(2, 3, 4)
(3, 4, 5)
```

Return generator.

seqtools.iterate_sequence_nwise_strict

abjad.tools.seqtools.**iterate_sequence_nwise_strict**(*sequence*, *n*)

New in version 1.1.2. Iterate elements in *sequence* *n* at a time:

```
abjad> list(seqtools.iterate_sequence_nwise_strict(range(10), 4))
[(0, 1, 2, 3), (1, 2, 3, 4), (2, 3, 4, 5), (3, 4, 5, 6), (4, 5, 6, 7), (5, 6, 7, 8), (6, 7, 8, 9
```

Return generator.

seqtools.iterate_sequence_nwise_wrapped

abjad.tools.seqtools.**iterate_sequence_nwise_wrapped**(*sequence*, *n*)

New in version 1.1.2. Iterate elements in *sequence* *n* at a time wrapped to beginning:

```
abjad> list(seqtools.iterate_sequence_nwise_wrapped(range(6), 3))
[(0, 1, 2), (1, 2, 3), (2, 3, 4), (3, 4, 5), (4, 5, 0), (5, 0, 1)]
```

Return generator.

seqtools.iterate_sequence_pairwise_cyclic

abjad.tools.seqtools.**iterate_sequence_pairwise_cyclic**(*sequence*)

New in version 1.1.1. Iterate *sequence* pairwise cyclic:

```
abjad> generator = seqtools.iterate_sequence_pairwise_cyclic(range(6))
```

```
abjad> generator.next( )
(0, 1)
abjad> generator.next( )
(1, 2)
abjad> generator.next( )
(2, 3)
abjad> generator.next( )
(3, 4)
abjad> generator.next( )
(4, 5)
abjad> generator.next( )
(5, 0)
abjad> generator.next( )
(0, 1)
abjad> generator.next( )
(1, 2)
```

Return pair generator.

seqtools.iterate_sequence_pairwise_strict

`abjad.tools.seqtools.`**`iterate_sequence_pairwise_strict`**`(`*sequence*`)`

New in version 1.1.1. Iterate *sequence* pairwise strict:

```
abjad> list(seqtools.iterate_sequence_pairwise_strict(range(6)))
[(0, 1), (1, 2), (2, 3), (3, 4), (4, 5)]
```

Return pair generator.

seqtools.iterate_sequence_pairwise_wrapped

`abjad.tools.seqtools.`**`iterate_sequence_pairwise_wrapped`**`(`*sequence*`)`

New in version 1.1.1. Iterate *sequence* pairwise wrapped:

```
abjad> list(seqtools.iterate_sequence_pairwise_wrapped(range(6)))
[(0, 1), (1, 2), (2, 3), (3, 4), (4, 5), (5, 0)]
```

Return pair generator.

seqtools.join_subsequences_by_sign_of_subsequence_elements

`abjad.tools.seqtools.`**`join_subsequences_by_sign_of_subsequence_elements`**`(`*sequence*`)`

New in version 1.1.1. Join subsequences in *sequence* by sign:

```
abjad> sequence = [[1, 2], [3, 4], [-5, -6, -7], [-8, -9, -10], [11, 12]]
abjad> seqtools.join_subsequences_by_sign_of_subsequence_elements(sequence)
[[1, 2, 3, 4], [-5, -6, -7, -8, -9, -10], [11, 12]]

abjad> sequence = [[1, 2], [ ], [ ], [3, 4, 5], [6, 7]]
abjad> seqtools.join_subsequences_by_sign_of_subsequence_elements(sequence)
[[1, 2], [], [3, 4, 5, 6, 7]]
```

Return newly constructed list. Changed in version 1.1.2: renamed `seqtools.join_sublists_by_sign( )` to `seqtools.join_subsequences_by_sign_of_subsequence_e` `)`.

seqtools.map_sequence_elements_to_canonic_tuples

abjad.tools.seqtools.**map_sequence_elements_to_canonic_tuples**(*sequence, direction='big-endian'*)

New in version 1.1.1. Partition *sequence* elements into canonic big-endian parts:

```
abjad> seqtools.map_sequence_elements_to_canonic_tuples(range(10))
[(0,), (1,), (2,), (3,), (4,), (4, 1), (6,), (7,), (8,), (8, 1)]
```

Partition *sequence* elements into canonic little-endian parts:

```
abjad> seqtools.map_sequence_elements_to_canonic_tuples(range(10), direction = 'little-endian')
[(0,), (1,), (2,), (3,), (4,), (1, 4), (6,), (7,), (8,), (1, 8)]
```

Raise type error when *sequence* is not a list.

Raise value error on noninteger elements in *sequence*.

Return list of tuples. Changed in version 1.1.2: renamed `seqtools.partition_elements_into_canonic_parts( )` to `seqtools.map_sequence_elements_to_canonic_tuples( )`.

seqtools.map_sequence_elements_to_numbered_sublists

abjad.tools.seqtools.**map_sequence_elements_to_numbered_sublists**(*sequence*)

New in version 1.1.1. Map *sequence* elements to numbered sublists:

```
abjad> seqtools.map_sequence_elements_to_numbered_sublists([1, 2, -3, -4, 5])
[[1], [2, 3], [-4, -5, -6], [-7, -8, -9, -10], [11, 12, 13, 14, 15]]

abjad> seqtools.map_sequence_elements_to_numbered_sublists([1, 0, -3, -4, 5])
[[1], [], [-2, -3, -4], [-5, -6, -7, -8], [9, 10, 11, 12, 13]]
```

Note that numbering starts at `1`.

Return newly constructed list of lists. Changed in version 1.1.2: renamed `seqtools.lengths_to_counts( )` to `seqtools.map_sequence_elements_to_numbered_sublists( )`.

seqtools.negate_absolute_value_of_sequence_elements_at_indices

abjad.tools.seqtools.**negate_absolute_value_of_sequence_elements_at_indices**(*sequence, indices*)

New in version 1.1.1. Negate the absolute value of *sequence* elements at *indices*:

```
abjad> sequence = [1, 2, 3, 4, 5, -6, -7, -8, -9, -10]

abjad> seqtools.negate_sequence_elements_at_indices(sequence, [0, 1, 2])
[-1, -2, -3, 4, 5, -6, -7, -8, -9, -10]
```

Return newly constructed list. Changed in version 1.1.2: renamed `seqtools.negate_elements_at_indices_absolutely( )` to `seqtools.negate_absolute_value_of_s`
`).`

seqtools.negate_absolute_value_of_sequence_elements_cyclically

abjad.tools.seqtools.**negate_absolute_value_of_sequence_elements_cyclically**(*sequence*, *indices*, *period*)

New in version 1.1.2. Negate the absolute value of *sequence* elements at *indices* cyclically according to *period*:

```
abjad> sequence = [1, 2, 3, 4, 5, -6, -7, -8, -9, -10]

abjad> seqtools.negate_absolute_value_of_sequence_elements_cyclically(sequence, [0, 1, 2], 5)
[-1, -2, -3, 4, 5, -6, -7, -8, -9, -10]
```

Return newly constructed list.

seqtools.negate_sequence_elements_at_indices

abjad.tools.seqtools.**negate_sequence_elements_at_indices**(*sequence*, *indices*)

New in version 1.1.1. Negate *sequence* elements at *indices*:

```
abjad> sequence = [1, 2, 3, 4, 5, -6, -7, -8, -9, -10]

abjad> seqtools.negate_sequence_elements_at_indices(sequence, [0, 1, 2])
[-1, -2, -3, 4, 5, -6, -7, -8, -9, -10]
```

Return newly constructed list. Changed in version 1.1.2: renamed `seqtools.negate_elements_at_indices( )` to `seqtools.negate_sequence_elements_at_indices( )`.

seqtools.negate_sequence_elements_cyclically

abjad.tools.seqtools.**negate_sequence_elements_cyclically**(*sequence*, *indices*, *period*)

New in version 1.1.2. Negate *sequence* elements at *indices* cyclically according to *period*:

```
abjad> sequence = [1, 2, 3, 4, 5, -6, -7, -8, -9, -10]

abjad> seqtools.negate_sequence_elements_cyclically(sequence, [0, 1, 2], 5)
[-1, -2, -3, 4, 5, 6, 7, 8, -9, -10]
```

Return newly constructed list.

seqtools.overwrite_sequence_elements_at_indices

abjad.tools.seqtools.**overwrite_sequence_elements_at_indices**(*sequence*, *pairs*)

New in version 1.1.1. Overwrite *sequence* elements at indices according to *pairs*:

```
seqtools.overwrite_sequence_elements_at_indices(range(10), [(0, 3), (5, 3)])
[0, 0, 0, 3, 4, 5, 5, 5, 8, 9]
```

Set *pairs* to a list of `(anchor_index, length)` pairs.

Return new list. Changed in version 1.1.2: renamed `seqtools.overwrite_slices_at( )` to `seqtools.overwrite_sequence_elements_at_indices( )`.

seqtools.partition_sequence_by_ratio_of_lengths

abjad.tools.seqtools.**partition_sequence_by_ratio_of_lengths**(*sequence*, *lengths*)

New in version 1.1.2. Partition *sequence* by ratio of *lengths*:

```
abjad> seqtools.partition_sequence_by_ratio_of_lengths(tuple(range(10)), [1, 1, 2])
[(0, 1, 2), (3, 4), (5, 6, 7, 8, 9)]
```

Use rounding magic to avoid fractional part lengths.

Return list of *sequence* objects.

seqtools.partition_sequence_by_ratio_of_weights

abjad.tools.seqtools.**partition_sequence_by_ratio_of_weights**(*sequence*, *weights*)

New in version 1.1.2. Partition *sequence* by ratio of *weights*:

```
abjad> seqtools.partition_sequence_by_ratio_of_weights([1] * 10, [1, 1, 1])
[[1, 1, 1], [1, 1, 1, 1], [1, 1, 1]]

abjad> seqtools.partition_sequence_by_ratio_of_weights([1] * 10, [1, 1, 1, 1])
[[1, 1, 1], [1, 1], [1, 1, 1], [1, 1]]

abjad> seqtools.partition_sequence_by_ratio_of_weights([1] * 10, [2, 2, 3])
[[1, 1, 1], [1, 1, 1], [1, 1, 1, 1]]

abjad> seqtools.partition_sequence_by_ratio_of_weights([1] * 10, [3, 2, 2])
[[1, 1, 1, 1], [1, 1, 1], [1, 1, 1]]

abjad> seqtools.partition_sequence_by_ratio_of_weights([1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2], [1,
[[1, 1, 1, 1, 1, 1, 2, 2], [2, 2, 2, 2]]

abjad> seqtools.partition_sequence_by_ratio_of_weights([1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2], [1,
[[1, 1, 1, 1, 1, 1], [2, 2, 2], [2, 2, 2]]
```

Weights of parts of returned list equal *weights_ratio* proportions with some rounding magic.

Return list of lists.

seqtools.partition_sequence_by_restricted_growth_function

abjad.tools.seqtools.**partition_sequence_by_restricted_growth_function**(*sequence*, *restricted_growth_function*)

New in version 1.1.2. Partition *sequence* by *restricted_growth_function*:

```
abjad> l = range(10)
abjad> rgf = [1, 1, 2, 2, 1, 2, 3, 3, 2, 4]
abjad> seqtools.partition_sequence_by_restricted_growth_function(l, rgf)
[[0, 1, 4], [2, 3, 5, 8], [6, 7], [9]]
```

Raise value error when *sequence* length does not equal *restricted_growth_function* length.

Return list of lists.

seqtools.partition_sequence_by_sign_of_elements

`abjad.tools.seqtools.`**`partition_sequence_by_sign_of_elements`**`(`*sequence, sign=[-1, 0, 1]*`)`

New in version 1.1.1. Partition *sequence* elements by sign:

```
abjad> sequence = [0, 0, -1, -1, 2, 3, -5, 1, 2, 5, -5, -6]

abjad> list(seqtools.partition_sequence_by_sign_of_elements(sequence))
[[0, 0], [-1, -1], [2, 3], [-5], [1, 2, 5], [-5, -6]]

abjad> list(seqtools.partition_sequence_by_sign_of_elements(sequence, sign = [-1]))
[0, 0, [-1, -1], 2, 3, [-5], 1, 2, 5, [-5, -6]]

abjad> list(seqtools.partition_sequence_by_sign_of_elements(sequence, sign = [0]))
[[0, 0], -1, -1, 2, 3, -5, 1, 2, 5, -5, -6]

abjad> list(seqtools.partition_sequence_by_sign_of_elements(sequence, sign = [1]))
[0, 0, -1, -1, [2, 3], -5, [1, 2, 5], -5, -6]

abjad> list(seqtools.partition_sequence_by_sign_of_elements(sequence, sign = [-1, 0]))
[[0, 0], [-1, -1], 2, 3, [-5], 1, 2, 5, [-5, -6]]

abjad> list(seqtools.partition_sequence_by_sign_of_elements(sequence, sign = [-1, 1]))
[0, 0, [-1, -1], [2, 3], [-5], [1, 2, 5], [-5, -6]]

abjad> list(seqtools.partition_sequence_by_sign_of_elements(sequence, sign = [0, 1]))
[[0, 0], -1, -1, [2, 3], -5, [1, 2, 5], -5, -6]

abjad> list(seqtools.partition_sequence_by_sign_of_elements(sequence, sign = [-1, 0, 1]))
[[0, 0], [-1, -1], [2, 3], [-5], [1, 2, 5], [-5, -6]]
```

When `-1` in `sign`, group negative elements.

When `0` in `sign`, group `0` elements.

When `1` in `sign`, group positive elements.

Return list of tuples of *sequence* element references. Changed in version 1.1.2: renamed `listtools.group_by_sign( )` to `seqtools.partition_sequence_by_sign_of_elements( )`.

seqtools.partition_sequence_by_value_of_elements

`abjad.tools.seqtools.`**`partition_sequence_by_value_of_elements`**`(`*sequence*`)`

New in version 1.1.1. Group *sequence* elements by equality:

```
abjad> seqtools.partition_sequence_by_value_of_elements([0, 0, -1, -1, 2, 3, -5, 1, 1, 5, -5])
[(0, 0), (-1, -1), (2,), (3,), (-5,), (1, 1), (5,), (-5,)]
```

Return list of tuples of *sequence* element references. Changed in version 1.1.2: renamed `seqtools.group_by_equality( )` to `seqtools.partition_sequence_by_value_of_elements( )`.

seqtools.partition_sequence_cyclically_by_counts_with_overhang

`abjad.tools.seqtools.`**`partition_sequence_cyclically_by_counts_with_overhang`**(*sequence*, *counts*)

New in version 1.1.1. Partition *sequence* cyclically by *counts* with overhang:

```
abjad> seqtools.partition_sequence_cyclically_by_counts_with_overhang(range(16), [4, 6])
[[0, 1, 2, 3], [4, 5, 6, 7, 8, 9], [10, 11, 12, 13], [14, 15]]
```

Return list of *sequence* objects. Changed in version 1.1.2: renamed `listtools.partition_sequence_cyclically_by_counts_with_overhang( )` to `seqtools.partition_sequence_cyclically_by_counts_with_overhang( )`.

seqtools.partition_sequence_cyclically_by_counts_without_overhang

`abjad.tools.seqtools.`**`partition_sequence_cyclically_by_counts_without_overhang`**(*sequence*, *counts*)

New in version 1.1.1. Partition *sequence* cyclically by *counts* without overhang:

```
abjad> seqtools.partition_sequence_cyclically_by_counts_without_overhang(range(16), [4, 6])
[[0, 1, 2, 3], [4, 5, 6, 7, 8, 9], [10, 11, 12, 13]]
```

Return list of *sequence* objects Changed in version 1.1.2: renamed `listtools.partition_sequence_cyclically_by_counts_without_overhang( )` to `seqtools.partition_sequence_cyclically_by_counts_without_overhang( )`.

seqtools.partition_sequence_cyclically_by_weights_at_least_with_overhang

`abjad.tools.seqtools.`**`partition_sequence_cyclically_by_weights_at_least_with_overhang`**(*sequenc* *weights*

New in version 1.1.1. Partition *sequence* elements cyclically by *weights* at least with overhang:

```
abjad> sequence = [3, 3, 3, 3, 4, 4, 4, 4, 5, 5]
abjad> seqtools.partition_sequence_cyclically_by_weights_at_least_with_overhang(sequence, [10, 4
[[3, 3, 3, 3], [4], [4, 4, 4], [5], [5]]
```

Return list sequence element reference lists. Changed in version 1.1.2: renamed `seqtools.group_sequence_elements_cyclically_by_weights_at_least_with_overhang( )` to `seqtools.partition_sequence_cyclically_by_weights_at_least_with_overhang( )`.

seqtools.partition_sequence_cyclically_by_weights_at_least_without_overhang

`abjad.tools.seqtools.`**`partition_sequence_cyclically_by_weights_at_least_without_overhang`**(*seq* *wei*

New in version 1.1.1. Partition *sequence* elements cyclically by *weights* at least without overhang:

```
abjad> sequence = [3, 3, 3, 3, 4, 4, 4, 4, 5, 5]
abjad> seqtools.partition_sequence_cyclically_by_weights_at_least_without_overhang(sequence, [10
[[3, 3, 3, 3], [4], [4, 4, 4], [5]]
```

Return list sequence element reference lists. Changed in version 1.1.2: renamed `seqtools.group_sequence_elements_cyclically_by_weights_at_least_without_overhang( )` to `seqtools.partition_sequence_cyclically_by_weights_at_least_without_overhang( )`.

seqtools.partition_sequence_cyclically_by_weights_at_most_with_overhang

`abjad.tools.seqtools.`**`partition_sequence_cyclically_by_weights_at_most_with_overhang`**`(`*sequence, weights*`)`

New in version 1.1.1. Partition *sequence* elements cyclically by *weights* at most with overhang:

```
abjad> sequence = [3, 3, 3, 3, 4, 4, 4, 4, 5, 5]
abjad> seqtools.partition_sequence_cyclically_by_weights_at_most_with_overhang(sequence, [10, 5]
[[3, 3, 3], [3], [4, 4], [4], [4, 5], [5]]
```

Return list sequence element reference lists. Changed in version 1.1.2: renamed `seqtools.group_sequence_elements_cyclically_by_weights_at_most_with_overhang()` to `seqtools.partition_sequence_cyclically_by_weights_at_most_with_overhang()`.

seqtools.partition_sequence_cyclically_by_weights_at_most_without_overhang

`abjad.tools.seqtools.`**`partition_sequence_cyclically_by_weights_at_most_without_overhang`**`(`*seque weig*

New in version 1.1.1. Partition *sequence* elements cyclically by *weights* at most without overhang:

```
abjad> sequence = [3, 3, 3, 3, 4, 4, 4, 4, 5]
abjad> seqtools.partition_sequence_cyclically_by_weights_at_most_without_overhang(sequence, [10,
[[3, 3, 3], [3], [4, 4], [4]]
```

Return list sequence element reference lists. Changed in version 1.1.2: renamed `seqtools.group_sequence_elements_cyclically_by_weights_at_most_without_overhang()` to `seqtools.partition_sequence_cyclically_by_weights_at_most_without_overhang()`.

seqtools.partition_sequence_cyclically_by_weights_exactly_with_overhang

`abjad.tools.seqtools.`**`partition_sequence_cyclically_by_weights_exactly_with_overhang`**`(`*sequence, weights*`)`

New in version 1.1.1. Partition *sequence* elements cyclically by *weights* exactly with overhang:

```
abjad> sequence = [3, 3, 3, 3, 4, 4, 4, 4, 5]
abjad> seqtools.partition_sequence_cyclically_by_weights_exactly_with_overhang(sequence, [12])
[[3, 3, 3, 3], [4, 4, 4], [4, 5]]
```

Return list of sequence element reference lists. Changed in version 1.1.2: renamed `seqtools.group_sequence_elements_cyclically_by_weights_exactly_with_overhang()` to `seqtools.partition_sequence_cyclically_by_weights_exactly_with_overhang()`.

seqtools.partition_sequence_cyclically_by_weights_exactly_without_overhang

`abjad.tools.seqtools.`**`partition_sequence_cyclically_by_weights_exactly_without_overhang`**`(`*seque weig*

New in version 1.1.1. Partition *sequence* elements cyclically by *weights* exactly without overhang:

```
abjad> sequence = [3, 3, 3, 3, 4, 4, 4, 4, 5]
abjad> seqtools.partition_sequence_cyclically_by_weights_exactly_without_overhang(sequence, [12]
[[3, 3, 3, 3], [4, 4, 4]]
```

Return list of sequence element reference lists. Changed in version 1.1.2: renamed `seqtools.group_sequence_elements_cyclically_by_weights_exactly_without_overhang()` to `seqtools.partition_sequence_cyclically_by_weights_exactly_without_overhang()`.

seqtools.partition_sequence_extended_to_counts_with_overhang

`abjad.tools.seqtools.`**`partition_sequence_extended_to_counts_with_overhang`**(*sequence*, *counts*)

New in version 1.1.2. Partition *sequence* extended to *counts* with overhang:

```
abjad> seqtools.partition_sequence_extended_to_counts_with_overhang([1, 2, 3, 4], [6, 6, 6])
[[1, 2, 3, 4, 1, 2], [3, 4, 1, 2, 3, 4], [1, 2, 3, 4, 1, 2], [3, 4]]
```

Return new object of *sequence* type.

seqtools.partition_sequence_extended_to_counts_without_overhang

`abjad.tools.seqtools.`**`partition_sequence_extended_to_counts_without_overhang`**(*sequence*, *counts*)

New in version 1.1.2. Partition *sequence* extended to *counts* without overhang:

```
abjad> seqtools.partition_sequence_extended_to_counts_without_overhang([1, 2, 3, 4], [6, 6, 6])
[[1, 2, 3, 4, 1, 2], [3, 4, 1, 2, 3, 4], [1, 2, 3, 4, 1, 2]]
```

Return new object of *sequence* type.

seqtools.partition_sequence_once_by_counts_with_overhang

`abjad.tools.seqtools.`**`partition_sequence_once_by_counts_with_overhang`**(*sequence*, *counts*)

New in version 1.1.1. Partition *sequence* once by *counts* with overhang:

```
abjad> seqtools.partition_sequence_once_by_counts_with_overhang(range(16), [4, 6])
[[0, 1, 2, 3], [4, 5, 6, 7, 8, 9], [10, 11, 12, 13, 14, 15]]
```

Return list of *sequence* objects. Changed in version 1.1.2: renamed `listtools.partition_sequence_once_by_counts_with_overhang()` to `seqtools.partition_sequence_once_by_counts_with_overhang()`.

seqtools.partition_sequence_once_by_counts_without_overhang

`abjad.tools.seqtools.`**`partition_sequence_once_by_counts_without_overhang`**(*sequence*, *counts*)

New in version 1.1.1. Partition *sequence* once by *counts* without overhang:

```
abjad> seqtools.partition_sequence_once_by_counts_without_overhang(range(16), [4, 6])
[[0, 1, 2, 3], [4, 5, 6, 7, 8, 9]]
```

Return list of *sequence* objects. Changed in version 1.1.2: renamed `listtools.partition_sequence_once_by_counts_without_overhang()` to `seqtools.partition_sequence_once_by_counts_without_overhang()`.

seqtools.partition_sequence_once_by_weights_at_least_with_overhang

`abjad.tools.seqtools.`**`partition_sequence_once_by_weights_at_least_with_overhang`**(*sequence*, *weights*)

New in version 1.1.1. Partition *sequence* elements once by *weights* at least with overhang:

```
abjad> sequence = [3, 3, 3, 3, 4, 4, 4, 4, 5, 5]
abjad> seqtools.partition_sequence_once_by_weights_at_least_with_overhang(sequence, [10, 4])
[[3, 3, 3, 3], [4], [4, 4, 4, 5, 5]]
```

Return list sequence element reference lists. Changed in version 1.1.2: renamed `seqtools.group_sequence_elements_once_by_weights_at_least_with_overhang( )` to `seqtools.partition_sequence_once_by_weights_at_least_with_overhang( )`.

seqtools.partition_sequence_once_by_weights_at_least_without_overhang

`abjad.tools.seqtools.`**`partition_sequence_once_by_weights_at_least_without_overhang`**(*sequence*, *weights*)

New in version 1.1.1. Partition *sequence* elements once by *weights* at least without overhang:

```
abjad> sequence = [3, 3, 3, 3, 4, 4, 4, 4, 5, 5]
abjad> seqtools.partition_sequence_once_by_weights_at_least_without_overhang(sequence, [10, 4])
[[3, 3, 3, 3], [4]]
```

Return list sequence element reference lists. Changed in version 1.1.2: renamed `seqtools.group_sequence_elements_once_by_weights_at_least_without_overhang( )` to `seqtools.partition_sequence_once_by_weights_at_least_without_overhang( )`.

seqtools.partition_sequence_once_by_weights_at_most_with_overhang

`abjad.tools.seqtools.`**`partition_sequence_once_by_weights_at_most_with_overhang`**(*sequence*, *weights*)

New in version 1.1.1. Partition *sequence* elements once by *weights* at most with overhang:

```
abjad> sequence = [3, 3, 3, 3, 4, 4, 4, 4, 5, 5]
abjad> seqtools.partition_sequence_once_by_weights_at_most_with_overhang(sequence, [10, 4])
[[3, 3, 3], [3], [4, 4, 4, 4, 5, 5]]
```

Return list sequence element reference lists. Changed in version 1.1.2: renamed `seqtools.group_sequence_elements_once_by_weights_at_most_with_overhang( )` to `seqtools.partition_sequence_once_by_weights_at_most_with_overhang( )`.

seqtools.partition_sequence_once_by_weights_at_most_without_overhang

`abjad.tools.seqtools.`**`partition_sequence_once_by_weights_at_most_without_overhang`**(*sequence*, *weights*)

New in version 1.1.1. Partition *sequence* elements once by *weights* at most without overhang:

```
abjad> sequence = [3, 3, 3, 3, 4, 4, 4, 4, 5, 5]
abjad> seqtools.partition_sequence_once_by_weights_at_most_without_overhang(sequence, [10, 4])
[[3, 3, 3], [3]]
```

Return list sequence element reference lists. Changed in version 1.1.2: renamed `seqtools.group_sequence_elements_once_by_weights_at_most_without_overhang(`

) to `seqtools.partition_sequence_once_by_weights_at_most_without_overhang( )`.

seqtools.partition_sequence_once_by_weights_exactly_with_overhang

`abjad.tools.seqtools.`**`partition_sequence_once_by_weights_exactly_with_overhang`**(*sequence*, *weights*)

New in version 1.1.1. Partition *sequence* elements once by *weights* exactly with overhang:

```
abjad> sequence = [3, 3, 3, 3, 4, 4, 4, 4, 5, 5]
abjad> seqtools.partition_sequence_once_by_weights_exactly_with_overhang(sequence, [3, 9])
[[3], [3, 3, 3], [4, 4, 4, 4, 5, 5]]
```

Return list sequence element reference lists. Changed in version 1.1.2: renamed `seqtools.group_sequence_elements_once_by_weights_exactly_with_overhang( )` to `seqtools.partition_sequence_once_by_weights_exactly_with_overhang( )`.

seqtools.partition_sequence_once_by_weights_exactly_without_overhang

`abjad.tools.seqtools.`**`partition_sequence_once_by_weights_exactly_without_overhang`**(*sequence*, *weights*)

New in version 1.1.1. Partition *sequence* elements once by *weights* exactly without overhang:

```
abjad> sequence = [3, 3, 3, 3, 4, 4, 4, 4, 5, 5]
abjad> seqtools.partition_sequence_once_by_weights_exactly_without_overhang(sequence, [3, 9])
[[3], [3, 3, 3]]
```

Return list sequence element reference lists. Changed in version 1.1.2: renamed `seqtools.group_sequence_elements_once_by_weights_exactly_without_overhang( )` to `seqtools.partition_sequence_once_by_weights_exactly_without_overhang( )`.

seqtools.permute_sequence

`abjad.tools.seqtools.`**`permute_sequence`**(*sequence*, *permutation*)

New in version 1.1.2. Permute *sequence* by *permutation*:

```
abjad> seqtools.permute_sequence([10, 11, 12, 13, 14, 15], [5, 4, 0, 1, 2, 3])
[15, 14, 10, 11, 12, 13]
```

Return newly constructed *sequence* object.

seqtools.remove_sequence_elements_at_indices

`abjad.tools.seqtools.`**`remove_sequence_elements_at_indices`**(*sequence*, *indices*)

New in version 1.1.2. Remove *sequence* elements at *indices*:

```
abjad> seqtools.remove_sequence_elements_at_indices(range(20), [1, 16, 17, 18])
[0, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 19]
```

Ignore negative indices.

Return list.

seqtools.remove_sequence_elements_at_indices_cyclically

`abjad.tools.seqtools.`**`remove_sequence_elements_at_indices_cyclically`**(*sequence*, *indices*, *period*, *offset=0*)

New in version 1.1.2. Remove *sequence* elements at *indices* mod *period* plus *offset*:

```
abjad> seqtools.remove_sequence_elements_at_indices_cyclically(range(20), [0, 1], 5, 3)
[0, 1, 2, 5, 6, 7, 10, 11, 12, 15, 16, 17]
```

Ignore negative indices.

Return list.

seqtools.remove_subsequence_of_weight_at_index

`abjad.tools.seqtools.`**`remove_subsequence_of_weight_at_index`**(*sequence*, *weight*, *index*)

New in version 1.1.1. Remove subsequence of *weight* at *index*:

```
abjad> seqtools.remove_subsequence_of_weight_at_index((1, 1, 2, 3, 5, 5, 1, 2, 5, 5, 6), 13, 4)
(1, 1, 2, 3, 5, 5, 6)
```

Return newly constructed *sequence* object. Changed in version 1.1.2: renamed `listtools.remove_weighted_subrun_at( )` to `seqtools.remove_subsequence_of_weight_at_index( )`.

seqtools.repeat_runs_in_sequence_to_count

`abjad.tools.seqtools.`**`repeat_runs_in_sequence_to_count`**(*sequence*, *indicators*)

New in version 1.1.1. Repeat subruns in *sequence* according to *indicators*. The *indicators* input parameter must be a list of zero or more `(start, length, count)` triples. For every `(start, length, count)` indicator in *indicators*, the function copies `sequence[start:start+length]` and inserts `count` new copies of `sequence[start:start+length]` immediately after `sequence[start:start+length]` in *sequence*.

Note: The function reads the value of `count` in every `(start, length, count)` triple not as the total number of occurrences of `sequence[start:start+length]` to appear in *sequence* after execution, but rather as the number of new occurrences of `sequence[start:start+length]` to appear in *sequence* after execution.

Note: The function wraps newly created subruns in tuples. That is, this function returns output with one more level of nesting than given in input.

To insert `10` count of `sequence[:2]` at `sequence[2:2]`:

```
abjad> seqtools.repeat_runs_in_sequence_to_count(range(20), [(0, 2, 10)])
[0, 1, (0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1),
2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

To insert `5` count of `sequence[10:12]` at `sequence[12:12]` and then insert `5` count of `sequence[:2]` at `sequence[2:2]`:

```
abjad> sequence = range(20)

abjad> seqtools.repeat_runs_in_sequence_to_count(sequence, [(0, 2, 5), (10, 2, 5)])
[0, 1, (0, 1, 0, 1, 0, 1, 0, 1, 0, 1), 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, (10, 11, 10, 11, 10, 11,
```

Note: This function wraps around the end of *sequence* whenever `len(sequence) < start + length`.

To insert 2 count of `[18, 19, 0, 1]` at `sequence[2:2]`:

```
abjad> seqtools.repeat_runs_in_sequence_to_count(sequence, [(18, 4, 2)])
[0, 1, (18, 19, 0, 1, 18, 19, 0, 1), 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18,
```

To insert 2 count of `[18, 19, 0, 1, 2, 3, 4]` at `sequence[4:4]`:

```
abjad> seqtools.repeat_runs_in_sequence_to_count(sequence, [(18, 8, 2)])
[0, 1, 2, 3, 4, 5, (18, 19, 0, 1, 2, 3, 4, 5, 18, 19, 0, 1, 2, 3, 4, 5), 6, 7, 8, 9, 10, 11, 12,
```

Todo

Implement an optional *wrap* keyword to specify whether this function should wrap around the ened of *sequence* whenever `len(sequence) < start + length` or not.

Todo

Reimplement this function to return a generator.

Generalizations of this function would include functions to repeat subruns in *sequence* to not only a certain count, as implemented here, but to a certain length, weight or sum. That is, `seqtools.repeat_subruns_to_length( )`, `seqtools.repeat_subruns_to_weight( )` and `seqtools.repeat_subruns_to_sum( )`. Changed in version 1.1.2: renamed `seqtools.repeat_subruns_to_count( )` to `seqtools.repeat_runs_in_sequence_to_count( )`.

seqtools.repeat_sequence_elements_at_indices

`abjad.tools.seqtools.`**`repeat_sequence_elements_at_indices`**`(`*sequence*, *indices*, *total*`)`

New in version 1.1.2. Repeat *sequence* elements at *indices* to *total* length:

```
abjad> seqtools.repeat_sequence_elements_at_indices(range(10), [6, 7, 8], 3)
[0, 1, 2, 3, 4, 5, [6, 6, 6], [7, 7, 7], [8, 8, 8], 9]
```

Return list.

seqtools.repeat_sequence_elements_at_indices_cyclically

`abjad.tools.seqtools.`**`repeat_sequence_elements_at_indices_cyclically`**`(`*sequence*, *cycle_token*, *total*`)`

New in version 1.1.2. Repeat *sequence* elements at indices specified by *cycle_token* to *total* length:

```
abjad> seqtools.repeat_sequence_elements_at_indices_cyclically(range(10), (5, [1, 2]), 3)
[0, [1, 1, 1], [2, 2, 2], 3, 4, 5, [6, 6, 6], [7, 7, 7], 8, 9]
```

The *cycle_token* may be a sieve:

```
abjad> from abjad.tools import sievetools

abjad> sieve = sievetools.cycle_tokens_to_sieve((5, [1, 2]))
abjad> seqtools.repeat_sequence_elements_at_indices_cyclically(range(10), sieve, 3)
[0, [1, 1, 1], [2, 2, 2], 3, 4, 5, [6, 6, 6], [7, 7, 7], 8, 9]
```

Return list.

seqtools.repeat_sequence_elements_n_times_each

abjad.tools.seqtools.**repeat_sequence_elements_n_times_each**(*sequence*, *n*)

New in version 1.1.1. Repeat *sequence* elements *n* times each:

```
abjad> seqtools.repeat_sequence_elements_n_times_each((1, -1, 2, -3, 5, -5, 6), 2)
(1, 1, -1, -1, 2, 2, -3, -3, 5, 5, -5, -5, 6, 6)
```

Return newly constructed *sequence* object with copied *sequence* elements. Changed in version 1.1.2: renamed `listtools.repeat_elements_to_count( )` to `seqtools.repeat_sequence_elements_n_times_each( )`.

seqtools.repeat_sequence_n_times

abjad.tools.seqtools.**repeat_sequence_n_times**(*sequence*, *n*)

New in version 1.1.2. Repeat *sequence* *n* times:

```
abjad> seqtools.repeat_sequence_n_times((1, 2, 3, 4, 5), 3)
(1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5)
```

Repeat *sequence* `0` times:

```
abjad> seqtools.repeat_sequence_n_times((1, 2, 3, 4, 5), 0)
()
```

Return newly constructed *sequence* object of copied *sequence* elements.

seqtools.repeat_sequence_to_length

abjad.tools.seqtools.**repeat_sequence_to_length**(*sequence*, *length*, *start=0*)

New in version 1.1.1. Repeat *sequence* to nonnegative integer *length*:

```
abjad> seqtools.repeat_sequence_to_length(range(5), 11)
[0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0]
```

Repeat *sequence* to nonnegative integer *length* from *start*:

```
abjad> seqtools.repeat_sequence_to_length(range(5), 11, start = 2)
[2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2]
```

Return newly constructed *sequence* object. Changed in version 1.1.2: renamed `listtools.repeat_list_to_length( )` to `seqtools.repeat_sequence_to_length( )`.

seqtools.repeat_sequence_to_weight_at_least

abjad.tools.seqtools.**repeat_sequence_to_weight_at_least**(*sequence*, *weight*)

New in version 1.1.1. Repeat *sequence* to *weight* at least:

```
abjad> seqtools.repeat_sequence_to_weight_at_least((5, -5, -5), 23)
(5, -5, -5, 5, -5)
```

Return newly constructed *sequence* object.

seqtools.repeat_sequence_to_weight_at_most

abjad.tools.seqtools.**repeat_sequence_to_weight_at_most**(*sequence*, *weight*)

New in version 1.1.1. Repeat *sequence* to *weight* at most:

```
abjad> seqtools.repeat_sequence_to_weight_at_most((5, -5, -5), 23)
(5, -5, -5, 5)
```

Return newly constructed *sequence* object.

seqtools.repeat_sequence_to_weight_exactly

abjad.tools.seqtools.**repeat_sequence_to_weight_exactly**(*sequence*, *weight*)

New in version 1.1.1. Repeat *sequence* to *weight* exactly:

```
abjad> seqtools.repeat_sequence_to_weight_exactly((5, -5, -5), 23)
(5, -5, -5, 5, -3)
```

Return newly constructed *sequence* object.

seqtools.replace_sequence_elements_cyclically_with_new_material

abjad.tools.seqtools.**replace_sequence_elements_cyclically_with_new_material**(*sequence*, *indices*, *new_material*)

New in version 1.1.1. Replace *sequence* elements cyclically at *indices* with *new_material*:

```
abjad> seqtools.replace_sequence_elements_cyclically_with_new_material(range(20), ([0], 2), (['A
['A', 1, 'B', 3, 4, 5, 'A', 7, 'B', 9, 10, 11, 'A', 13, 'B', 15, 16, 17, 'A', 19]

abjad> seqtools.replace_sequence_elements_cyclically_with_new_material(range(20), ([0], 2), (['*
['*', 1, '*', 3, '*', 5, '*', 7, '*', 9, '*', 11, '*', 13, '*', 15, '*', 17, '*', 19]

abjad> seqtools.replace_sequence_elements_cyclically_with_new_material(range(20), ([0], 2), (['A
['A', 1, 'B', 3, 'C', 5, 'D', 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]

abjad> seqtools.replace_sequence_elements_cyclically_with_new_material(range(20), ([0, 1, 8, 13]
['A', 'B', 2, 3, 4, 5, 6, 7, 'C', 9, 10, 11, 12, 'D', 14, 15, 16, 17, 18, 19]
```

Raise type error when *sequence* not a list.

Return newly constructed list. Changed in version 1.1.2: renamed `seqtools.replace_elements_cyclic( )` to `seqtools.replace_sequence_elements_cyclically_wit` `)`.

seqtools.retain_sequence_elements_at_indices

`abjad.tools.seqtools.`**`retain_sequence_elements_at_indices`**`(`*sequence*, *indices*`)`

New in version 1.1.2. Retain *sequence* elements at *indices*:

```
abjad> seqtools.retain_sequence_elements_at_indices(range(20), [1, 16, 17, 18])
[1, 16, 17, 18]
```

Ignore negative indices.

Return list.

seqtools.retain_sequence_elements_at_indices_cyclically

`abjad.tools.seqtools.`**`retain_sequence_elements_at_indices_cyclically`**`(`*sequence*, *indices*, *period*, *offset=0*`)`

New in version 1.1.2. Retain *sequence* elements at *indices* mod *period* plus *offset*:

```
abjad> seqtools.retain_sequence_elements_at_indices_cyclically(range(20), [0, 1], 5, 3)
[3, 4, 8, 9, 13, 14, 18, 19]
```

Ignore negative values in *indices*.

Return list.

seqtools.reverse_sequence

`abjad.tools.seqtools.`**`reverse_sequence`**`(`*sequence*`)`

New in version 1.1.2. Reverse *sequence*:

```
abjad> seqtools.reverse_sequence((1, 2, 3, 4, 5))
(5, 4, 3, 2, 1)
```

Return new *sequence* object.

seqtools.reverse_sequence_elements

`abjad.tools.seqtools.`**`reverse_sequence_elements`**`(`*sequence*`)`

New in version 1.1.2. Reverse *sequence* elements:

```
abjad> seqtools.reverse_sequence_elements([1, (2, 3, 4), 5, (6, 7)])
[1, (4, 3, 2), 5, (7, 6)]
```

Return new *sequence* object.

seqtools.rotate_sequence

`abjad.tools.seqtools.`**`rotate_sequence`**`(`*sequence*, *n*`)`

New in version 1.1.1. Rotate *sequence* to the right:

```
abjad> seqtools.rotate_sequence(range(10), 4)
[6, 7, 8, 9, 0, 1, 2, 3, 4, 5]
```

Rotate *sequence* to the left:

```
abjad> seqtools.rotate_sequence(range(10), -3)
[3, 4, 5, 6, 7, 8, 9, 0, 1, 2]
```

Rotate *sequence* neither to the right nor the left:

```
abjad> seqtools.rotate_sequence(range(10), 0)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Return newly created *sequence* object. Changed in version 1.1.2: renamed `seqtools.rotate( )` to `seqtools.rotate_sequence( )`.

seqtools.splice_new_elements_between_sequence_elements

`abjad.tools.seqtools.`**`splice_new_elements_between_sequence_elements`**(*sequence*, *new_elements*, *overhang=(0, 0)*)

New in version 1.1.1. Splice copies of *new_elements* between each of the elements of *sequence*:

```
abjad> sequence = [0, 1, 2, 3, 4]
abjad> new_elements = ['A', 'B']

abjad> seqtools.splice_new_elements_between_sequence_elements(sequence, new_elements)
[0, 'A', 'B', 1, 'A', 'B', 2, 'A', 'B', 3, 'A', 'B', 4]
```

Splice copies of *new_elements* between each of the elements of *sequence* and after the last element of *sequence*:

```
abjad> seqtools.splice_new_elements_between_sequence_elements(sequence, new_elements, overhang =
[0, 'A', 'B', 1, 'A', 'B', 2, 'A', 'B', 3, 'A', 'B', 4, 'A', 'B']
```

Splice copies of *new_elements* before the first element of *sequence* and between each of the other elements of *sequence*:

```
abjad> seqtools.splice_new_elements_between_sequence_elements(sequence, new_elements, overhang =
['A', 'B', 0, 'A', 'B', 1, 'A', 'B', 2, 'A', 'B', 3, 'A', 'B', 4]
```

Splice copies of *new_elements* before the first element of *sequence*, after the last element of *sequence* and between each of the other elements of *sequence*:

```
abjad> seqtools.splice_new_elements_between_sequence_elements(sequence, new_elements, overhang =
['A', 'B', 0, 'A', 'B', 1, 'A', 'B', 2, 'A', 'B', 3, 'A', 'B', 4, 'A', 'B']
```

Return newly constructed list. Changed in version 1.1.2: renamed `seqtools.insert_slice_cyclic( )` to `seqtools.splice_new_elements_between_sequence_elements( )`.

seqtools.split_sequence_cyclically_by_weights_with_overhang

`abjad.tools.seqtools.`**`split_sequence_cyclically_by_weights_with_overhang`**(*sequence*, *weights*)

New in version 1.1.2. Split *sequence* cyclically by *weights* with overhang:

```
abjad> seqtools.split_sequence_cyclically_by_weights_with_overhang((10, -10, 10, -10), [3, 15, 3
[(3,), (7, -8), (-2, 1), (3,), (6, -9), (-1,)]
```

Return list of *sequence* objects.

seqtools.split_sequence_cyclically_by_weights_without_overhang

`abjad.tools.seqtools.`**`split_sequence_cyclically_by_weights_without_overhang`**(*sequence*, *weights*)

New in version 1.1.2. Split *sequence* cyclically by *weights* without overhang:

```
abjad> seqtools.split_sequence_cyclically_by_weights_without_overhang((10, -10, 10, -10), [3, 15
[(3,), (7, -8), (-2, 1), (3,), (6, -9)]
```

Return list of *sequence* objects.

seqtools.split_sequence_extended_to_weights_with_overhang

`abjad.tools.seqtools.`**`split_sequence_extended_to_weights_with_overhang`**(*sequence*, *weights*)

New in version 1.1.2. Split *sequence* extended to *weights* with overhang:

```
abjad> seqtools.split_sequence_extended_to_weights_with_overhang([1, 2, 3, 4, 5], [7, 7, 7])
[[1, 2, 3, 1], [3, 4], [1, 1, 2, 3], [4, 5]]
```

Return new object of *sequence* type.

seqtools.split_sequence_extended_to_weights_without_overhang

`abjad.tools.seqtools.`**`split_sequence_extended_to_weights_without_overhang`**(*sequence*, *weights*)

New in version 1.1.2. Split *sequence* extended to *weights* without overhang:

```
abjad> seqtools.split_sequence_extended_to_weights_without_overhang([1, 2, 3, 4, 5], [7, 7, 7])
[[1, 2, 3, 1], [3, 4], [1, 1, 2, 3]]
```

Return new object of *sequence* type.

seqtools.split_sequence_once_by_weights_with_overhang

`abjad.tools.seqtools.`**`split_sequence_once_by_weights_with_overhang`**(*sequence*, *weights*)

New in version 1.1.2. Split *sequence* once by *weights* with overhang:

```
abjad> seqtools.split_sequence_once_by_weights_with_overhang((10, -10, 10, -10), [3, 15, 3])
[(3,), (7, -8), (-2, 1), (9, -10)]
```

Return list of *sequence* objects.

seqtools.split_sequence_once_by_weights_without_overhang

`abjad.tools.seqtools.`**`split_sequence_once_by_weights_without_overhang`**(*sequence*, *weights*)

New in version 1.1.2. Split *sequence* once by *weights* without overhang:

```
abjad> seqtools.split_sequence_once_by_weights_without_overhang((10, -10, 10, -10), [3, 15, 3])
[(3,), (7, -8), (-2, 1)]
```

Return list of *sequence* objects.

seqtools.sum_consecutive_sequence_elements_by_sign

abjad.tools.seqtools.**sum_consecutive_sequence_elements_by_sign**(*sequence, sign=[-1, 0, 1]*)

New in version 1.1.1. Sum consecutive *sequence* elements by *sign*:

```
abjad> sequence = [0, 0, -1, -1, 2, 3, -5, 1, 2, 5, -5, -6]

abjad> seqtools.sum_consecutive_sequence_elements_by_sign(sequence)
[0, -2, 5, -5, 8, -11]

abjad> seqtools.sum_consecutive_sequence_elements_by_sign(sequence, sign = [-1])
[0, 0, -2, 2, 3, -5, 1, 2, 5, -11]

abjad> seqtools.sum_consecutive_sequence_elements_by_sign(sequence, sign = [0])
[0, -1, -1, 2, 3, -5, 1, 2, 5, -5, -6]

abjad> seqtools.sum_consecutive_sequence_elements_by_sign(sequence, sign = [1])
[0, 0, -1, -1, 5, -5, 8, -5, -6]

abjad> seqtools.sum_consecutive_sequence_elements_by_sign(sequence, sign = [-1, 0])
[0, -2, 2, 3, -5, 1, 2, 5, -11]

abjad> seqtools.sum_consecutive_sequence_elements_by_sign(sequence, sign = [-1, 1])
[0, 0, -2, 5, -5, 8, -11]

abjad> seqtools.sum_consecutive_sequence_elements_by_sign(sequence, sign = [0, 1])
[0, -1, -1, 5, -5, 8, -5, -6]

abjad> seqtools.sum_consecutive_sequence_elements_by_sign(sequence, sign = [-1, 0, 1])
[0, -2, 5, -5, 8, -11]
```

When `-1` in *sign*, sum consecutive negative elements.

When `0` in *sign*, sum consecutive `0` elements.

When `1` in *sign*, sum consecutive positive elements.

Return list. Changed in version 1.1.2: renamed `seqtools.sum_by_sign( )` to `seqtools.sum_consecutive_sequence_elements_by_sign( )`.

seqtools.sum_sequence_elements_at_indices

abjad.tools.seqtools.**sum_sequence_elements_at_indices**(*sequence, pairs, period=None, overhang=True*)

New in version 1.1.1. Sum *sequence* elements at indices according to *pairs*:

```
abjad> seqtools.sum_sequence_elements_at_indices(range(10), [(0, 3)])
[3, 3, 4, 5, 6, 7, 8, 9]
```

Sum *sequence* elements cyclically at indices according to *pairs* and *period*:

```
abjad> seqtools.sum_sequence_elements_at_indices(range(10), [(0, 3)], period = 4)
[3, 3, 15, 7, 17]
```

Sum *sequence* elements cyclically at indices according to *pairs* and *period* and do not return incomplete final sum:

```
abjad> seqtools.sum_sequence_elements_at_indices(range(10), [(0, 3)], period = 4, overhang = Fal
[3, 3, 15, 7]
```

Replace `sequence[i:i+count]` with `sum(sequence[i:i+count])` for each `(i, count)` in *pairs*.

Indices in *pairs* must be less than *period* when *period* is not none.

Return new list. Changed in version 1.1.2: renamed `seqtools.sum_slices_at( )` to `seqtools.sum_sequence_elements_at_indices( )`.

seqtools.truncate_runs_in_sequence

`abjad.tools.seqtools.`**`truncate_runs_in_sequence`**`(`*sequence*`)`

New in version 1.1.1. Truncate subruns of like elements in *sequence* to length `1`:

```
abjad> seqtools.truncate_runs_in_sequence([1, 1, 2, 3, 3, 3, 9, 4, 4, 4])
[1, 2, 3, 9, 4]
```

Return empty list when *sequence* is empty:

```
abjad> seqtools.truncate_runs_in_sequence([ ])
[]
```

Raise type error when *sequence* is not a list.

Return new list. Changed in version 1.1.2: renamed `seqtools.truncate_subruns( )` to `seqtools.truncate_runs_in_sequence( )`.

seqtools.truncate_sequence_to_sum

`abjad.tools.seqtools.`**`truncate_sequence_to_sum`**`(`*sequence*, *sum*`)`

New in version 1.1.1. Truncate *sequence* to *sum*:

```
abjad> for n in range(10):
...     print n, seqtools.truncate_sequence_to_sum([-1, 2, -3, 4, -5, 6, -7, 8, -9, 10], n)
...
0 []
1 [-1, 2]
2 [-1, 2, -3, 4]
3 [-1, 2, -3, 4, -5, 6]
4 [-1, 2, -3, 4, -5, 6, -7, 8]
5 [-1, 2, -3, 4, -5, 6, -7, 8, -9, 10]
6 [-1, 2, -3, 4, -5, 6, -7, 8, -9, 10]
7 [-1, 2, -3, 4, -5, 6, -7, 8, -9, 10]
8 [-1, 2, -3, 4, -5, 6, -7, 8, -9, 10]
9 [-1, 2, -3, 4, -5, 6, -7, 8, -9, 10]
```

Return empty list when *sum* is `0`:

```
abjad> seqtools.truncate_sequence_to_sum([1, 2, 3, 4, 5], 0)
[]
```

Raise type error when *sequence* is not a list.

Raise value error on negative *sum*.

Return new list. Changed in version 1.1.2: renamed `seqtools.truncate_to_sum( )` to `seqtools.truncate_sequence_to_sum( )`.

seqtools.truncate_sequence_to_weight

abjad.tools.seqtools.**truncate_sequence_to_weight**(*sequence*, *weight*)

New in version 1.1.1. Truncate *sequence* to *weight*:

```
abjad> l = [-1, 2, -3, 4, -5, 6, -7, 8, -9, 10]
abjad> for x in range(10):
...     print x, seqtools.truncate_sequence_to_weight(l, x)
...
0 []
1 [-1]
2 [-1, 1]
3 [-1, 2]
4 [-1, 2, -1]
5 [-1, 2, -2]
6 [-1, 2, -3]
7 [-1, 2, -3, 1]
8 [-1, 2, -3, 2]
9 [-1, 2, -3, 3]
```

Return empty list when *weight* is 0:

```
abjad> seqtools.truncate_sequence_to_weight([1, 2, 3, 4, 5], 0)
[]
```

Raise type error when *sequence* is not a list.

Raise value error on negative *weight*.

Return new list. Changed in version 1.1.2: renamed `seqtools.truncate_to_weight( )` to `seqtools.truncate_sequence_to_weight( )`.

seqtools.yield_all_combinations_of_sequence_elements

abjad.tools.seqtools.**yield_all_combinations_of_sequence_elements**(*sequence*, *min_length=None*, *max_length=None*)

New in version 1.1.2. Yield all combinations of *sequence* in binary string order:

```
abjad> list(seqtools.yield_all_combinations_of_sequence_elements([1, 2, 3, 4]))
[[], [1], [2], [1, 2], [3], [1, 3], [2, 3], [1, 2, 3], [4], [1, 4],
[2, 4], [1, 2, 4], [3, 4], [1, 3, 4], [2, 3, 4], [1, 2, 3, 4]]
```

Yield all combinations of *sequence* greater than or equal to *min_length* in binary string order:

```
abjad> list(seqtools.yield_all_combinations_of_sequence_elements([1, 2, 3, 4], min_length = 3))
[[1, 2, 3], [1, 2, 4], [1, 3, 4], [2, 3, 4], [1, 2, 3, 4]]
```

Yield all combinations of *sequence* less than or equal to *max_length* in binary string order:

```
abjad> list(seqtools.yield_all_combinations_of_sequence_elements([1, 2, 3, 4], max_length = 2))
[[], [1], [2], [1, 2], [3], [1, 3], [2, 3], [4], [1, 4], [2, 4], [3, 4]]
```

Yield all combiantions of *sequence* greater than or equal to *min_length* and less than or equal to *max_length* in lex order:

```
abjad> list(seqtools.yield_all_combinations_of_sequence_elements([1, 2, 3, 4], min_length = 2, m
[[1, 2], [1, 3], [2, 3], [1, 4], [2, 4], [3, 4]]
```

Return generator of newly created *sequence* objects. Changed in version 1.1.2: renamed `seqtools.sublists( )` to `seqtools.yield_all_combinations_of_sequence_elements( )`.

seqtools.yield_all_k_ary_sequences_of_length

`abjad.tools.seqtools.`**`yield_all_k_ary_sequences_of_length`**`(`*k*, *length*`)`

New in version 1.1.2. Generate all *k*-ary sequences of *length*:

```
abjad> for sequence in seqtools.yield_all_k_ary_sequences_of_length(2, 3):
...     sequence
...
(0, 0, 0)
(0, 0, 1)
(0, 1, 0)
(0, 1, 1)
(1, 0, 0)
(1, 0, 1)
(1, 1, 0)
(1, 1, 1)
```

Return generator of tuples.

seqtools.yield_all_pairs_between_sequences

`abjad.tools.seqtools.`**`yield_all_pairs_between_sequences`**`(`*l*, *m*`)`

New in version 1.1.2. Yield all pairs between sequences *l* and *m*:

```
abjad> for pair in seqtools.yield_all_pairs_between_sequences([1, 2, 3], [4, 5]):
...     pair
...
(1, 4)
(1, 5)
(2, 4)
(2, 5)
(3, 4)
(3, 5)
```

Return pair generator.

seqtools.yield_all_partitions_of_sequence

`abjad.tools.seqtools.`**`yield_all_partitions_of_sequence`**`(`*sequence*`)`

New in version 1.1.2. Yield all partitions of *sequence*:

```
abjad> for partition in seqtools.yield_all_partitions_of_sequence([0, 1, 2, 3]):
...     partition
...
[[0, 1, 2, 3]]
[[0, 1, 2], [3]]
[[0, 1], [2, 3]]
[[0, 1], [2], [3]]
[[0], [1, 2, 3]]
[[0], [1, 2], [3]]
[[0], [1], [2, 3]]
[[0], [1], [2], [3]]
```

Return generator of newly created lists.

seqtools.yield_all_permutations_of_sequence

`abjad.tools.seqtools.`**`yield_all_permutations_of_sequence`**`(`*sequence*`)`

New in version 1.1.1. Yield all permutations of *sequence* in lex order:

```
abjad> list(seqtools.yield_all_permutations_of_sequence((1, 2, 3)))
[(1, 2, 3), (1, 3, 2), (2, 1, 3), (2, 3, 1), (3, 1, 2), (3, 2, 1)]
```

Return generator of *sequence* objects. Changed in version 1.1.2: renamed `listtools.permutations( )` to `seqtools.yield_all_permutations_of_sequence( )`.

seqtools.yield_all_permutations_of_sequence_in_orbit

`abjad.tools.seqtools.`**`yield_all_permutations_of_sequence_in_orbit`**`(`*sequence*, *permutation*`)`

New in version 1.1.2. Yield all permutations of *sequence* in orbit of *permutation* in lex order:

```
abjad> list(seqtools.yield_all_permutations_of_sequence_in_orbit((1, 2, 3, 4), [1, 2, 3, 0]))
[(1, 2, 3, 4), (2, 3, 4, 1), (3, 4, 1, 2), (4, 1, 2, 3)]
```

Return generator of *sequence* objects.

seqtools.yield_all_restricted_growth_functions_of_length

`abjad.tools.seqtools.`**`yield_all_restricted_growth_functions_of_length`**`(`*length*`)`

New in version 1.1.2. Generate all restricted growth functions of *length* in lex order:

```
abjad> for rgf in seqtools.yield_all_restricted_growth_functions_of_length(4):
...     rgf
...
(1, 1, 1, 1)
(1, 1, 1, 2)
(1, 1, 2, 1)
(1, 1, 2, 2)
(1, 1, 2, 3)
(1, 2, 1, 1)
(1, 2, 1, 2)
(1, 2, 1, 3)
(1, 2, 2, 1)
(1, 2, 2, 2)
(1, 2, 2, 3)
(1, 2, 3, 1)
(1, 2, 3, 2)
(1, 2, 3, 3)
(1, 2, 3, 4)
```

Return generator of tuples.

seqtools.yield_all_rotations_of_sequence

`abjad.tools.seqtools.`**`yield_all_rotations_of_sequence`**`(`*sequence*, *n=1*`)`

New in version 1.1.2. Yield all *n*-rotations of *sequence* up to identity:

```
abjad> list(seqtools.yield_all_rotations_of_sequence([1, 2, 3, 4], -1))
[[1, 2, 3, 4], [2, 3, 4, 1], [3, 4, 1, 2], [4, 1, 2, 3]]
```

Return generator of *sequence* objects.

seqtools.yield_all_set_partitions_of_sequence

abjad.tools.seqtools.**yield_all_set_partitions_of_sequence**(*sequence*)

New in version 1.1.2. Yield all set partitions of *sequence* in restricted growth function order:

```
abjad> for set_partition in seqtools.yield_all_set_partitions_of_sequence([21, 22, 23, 24]):
...     set_partition
...
[[21, 22, 23, 24]]
[[21, 22, 23], [24]]
[[21, 22, 24], [23]]
[[21, 22], [23, 24]]
[[21, 22], [23], [24]]
[[21, 23, 24], [22]]
[[21, 23], [22, 24]]
[[21, 23], [22], [24]]
[[21, 24], [22, 23]]
[[21], [22, 23, 24]]
[[21], [22, 23], [24]]
[[21, 24], [22], [23]]
[[21], [22, 24], [23]]
[[21], [22], [23, 24]]
[[21], [22], [23], [24]]
```

Return generator of list of lists.

seqtools.yield_all_subsequences_of_sequence

abjad.tools.seqtools.**yield_all_subsequences_of_sequence**(*sequence*, *min_length=0*, *max_length=None*)

New in version 1.1.2. Yield all subsequences of *sequence* in lex order:

```
abjad> list(seqtools.yield_all_subsequences_of_sequence([0, 1, 2]))
[[], [0], [0, 1], [0, 1, 2], [1], [1, 2], [2]]
```

Yield all subsequences of *sequence* greater than or equal to *min_length* in lex order:

```
abjad> list(seqtools.yield_all_subsequences_of_sequence([0, 1, 2, 3, 4], min_length = 3))
[[0, 1, 2], [0, 1, 2, 3], [0, 1, 2, 3, 4], [1, 2, 3], [1, 2, 3, 4], [2, 3, 4]]
```

Yield all subsequences of *sequence* less than or equal to *max_length* in lex order:

```
abjad> list(seqtools.yield_all_subsequences_of_sequence([0, 1, 2, 3, 4], max_length = 3))
[[], [0], [0, 1], [0, 1, 2], [1], [1, 2], [1, 2, 3], [2], [2, 3], [2, 3, 4], [3], [3, 4], [4]]
```

Yield all subsequences of *sequence* greater than or equal to *min_length* and less than or equal to *max_length* in lex order:

```
abjad> list(seqtools.yield_all_subsequences_of_sequence([0, 1, 2, 3, 4], min_length = 3, max_len
[[0, 1, 2], [1, 2, 3], [2, 3, 4]]
```

Return generator of newly created *sequence* slices.

seqtools.yield_all_unordered_pairs_of_sequence

abjad.tools.seqtools.**yield_all_unordered_pairs_of_sequence**(*sequence*)

New in version 1.1.2. Yield all unordered pairs of *sequence*:

```
abjad> list(seqtools.yield_all_unordered_pairs_of_sequence([1, 2, 3, 4]))
[(1, 2), (1, 3), (1, 4), (2, 3), (2, 4), (3, 4)]
```

Yield all unordered pairs of length-1 *sequence*:

```
abjad> list(seqtools.yield_all_unordered_pairs_of_sequence([1]))
[]
```

Yield all unordered pairs of empty *sequence*:

```
abjad> list(seqtools.yield_all_unordered_pairs_of_sequence([ ]))
[]
```

Yield all unordered pairs of *sequence* with duplicate elements:

```
abjad> list(seqtools.yield_all_unordered_pairs_of_sequence([1, 1, 1]))
[(1, 1), (1, 1), (1, 1)]
```

Pairs are tuples instead of sets to accommodate duplicate *sequence* elements.

Return generator.

seqtools.yield_outer_product_of_sequences

abjad.tools.seqtools.**yield_outer_product_of_sequences**(*sequences*)

New in version 1.1.1. Yield outer product of *sequences*:

```
abjad> list(seqtools.yield_outer_product_of_sequences([[1, 2, 3], ['a', 'b']]))
[[1, 'a'], [1, 'b'], [2, 'a'], [2, 'b'], [3, 'a'], [3, 'b']]

abjad> list(seqtools.yield_outer_product_of_sequences([[1, 2, 3], ['a', 'b'], ['X', 'Y']]))
[[1, 'a', 'X'], [1, 'a', 'Y'], [1, 'b', 'X'], [1, 'b', 'Y'],
[2, 'a', 'X'], [2, 'a', 'Y'], [2, 'b', 'X'], [2, 'b', 'Y'],
[3, 'a', 'X'], [3, 'a', 'Y'], [3, 'b', 'X'], [3, 'b', 'Y']]

abjad> list(seqtools.yield_outer_product_of_sequences([[1, 2, 3], [4, 5], [6, 7, 8]]))
[[1, 4, 6], [1, 4, 7], [1, 4, 8], [1, 5, 6], [1, 5, 7], [1, 5, 8],
[2, 4, 6], [2, 4, 7], [2, 4, 8], [2, 5, 6], [2, 5, 7], [2, 5, 8],
[3, 4, 6], [3, 4, 7], [3, 4, 8], [3, 5, 6], [3, 5, 7], [3, 5, 8]]
```

Return generator. Changed in version 1.1.2: renamed `seqtools.outer_product( )` to `seqtools.yield_outer_product_of_sequences( )`.

seqtools.zip_sequences_cyclically

abjad.tools.seqtools.**zip_sequences_cyclically**(**sequences*)

New in version 1.1.1. Zip *sequences* cyclically:

```
abjad> seqtools.zip_sequences_cyclically([1, 2, 3], ['a', 'b'])
[(1, 'a'), (2, 'b'), (3, 'a')]
```

New in version 1.1.1: Arbitrary number of input sequences now allowed.

```
abjad> seqtools.zip_sequences_cyclically([10, 11, 12], [20, 21], [30, 31, 32, 33])
[(10, 20, 30), (11, 21, 31), (12, 20, 32), (10, 21, 33)]
```

Cycle over the elements of the sequences of shorter length.

Return list of length equal to sequence of greatest length in *sequences*. Changed in version 1.1.2: renamed `seqtools.zip_cyclic( )` to `seqtools.zip_sequences_cyclically( )`.

seqtools.zip_sequences_without_truncation

`abjad.tools.seqtools.`**`zip_sequences_without_truncation`**`(*sequences)`

New in version 1.1.1. Zip *sequences* nontruncating:

```
abjad> seqtools.zip_sequences_without_truncation([1, 2, 3, 4], [11, 12, 13], [21, 22, 23])
[(1, 11, 21), (2, 12, 22), (3, 13, 23), (4,)]
```

Lengths of the tuples returned may differ but will always be greater than or equal to `1`.

Return list of tuples. Changed in version 1.1.2: renamed `seqtools.zip_nontruncating( )` to `seqtools.zip_sequences_without_truncation( )`.

skiptools

skiptools.Skip

class `abjad.tools.skiptools.`**`Skip`**`(*args, **kwargs)`

Bases: `abjad.components._Leaf._Leaf._Leaf`

Abjad model of a LilyPond skip:

```
abjad> skiptools.Skip((3, 16))
Skip('s8.')
```

Return skip.

skiptools.iterate_skips_backward_in_expr

`abjad.tools.skiptools.`**`iterate_skips_backward_in_expr`**`(expr, start=0, stop=None)`

New in version 1.1.2. Iterate skips backward in *expr*:

```
abjad> staff = Staff("<e' g' c''>8 a'8 s8 <d' f' b'>8 s2")

abjad> f(staff)
\new Staff {
   <e' g' c''>8
   a'8
   s8
   <d' f' b'>8
   s2
}

abjad> for skip in skiptools.iterate_skips_backward_in_expr(staff):
...   skip
Skip('s2')
Skip('s8')
```

Ignore threads.

Return generator.

skiptools.iterate_skips_forward_in_expr

abjad.tools.skiptools.**iterate_skips_forward_in_expr**(*expr*, *start=0*, *stop=None*)
New in version 1.1.2. Iterate skips forward in *expr*:

```
abjad> staff = Staff("<e' g' c''>8 a'8 s8 <d' f' b'>8 s2")

abjad> f(staff)
\new Staff {
   <e' g' c''>8
   a'8
   s8
   <d' f' b'>8
   s2
}

abjad> for skip in skiptools.iterate_skips_forward_in_expr(staff):
...   skip
Skip('s8')
Skip('s2')
```

Ignore threads.

Return generator.

skiptools.make_repeated_skips_from_time_signature

abjad.tools.skiptools.**make_repeated_skips_from_time_signature**(*time_signature*)
New in version 1.1.2. Make repeated skips from *time_signature*:

```
abjad> skiptools.make_repeated_skips_from_time_signature((5, 32))
[Skip('s32'), Skip('s32'), Skip('s32'), Skip('s32'), Skip('s32')]
```

Return list of skips.

skiptools.make_repeated_skips_from_time_signatures

abjad.tools.skiptools.**make_repeated_skips_from_time_signatures**(*time_signatures*)
Make repated skips from *time_signatures*:

```
skiptools.make_repeated_skips_from_time_signatures([(2, 8), (3, 32)])
[[Skip('s8'), Skip('s8')], [Skip('s32'), Skip('s32'), Skip('s32')]]
```

Return list of skip lists.

skiptools.make_skips_with_multiplied_durations

abjad.tools.skiptools.**make_skips_with_multiplied_durations**(*written_duration*, *multiplied_durations*)
New in version 1.1.2. Make *written_duration* skips with *multiplied_durations*:

```
abjad> skiptools.make_skips_with_multiplied_durations(Duration(1, 4), [(1, 2), (1, 3), (1, 4), (
[Skip('s4 * 2'), Skip('s4 * 4/3'), Skip('s4 * 1'), Skip('s4 * 4/5')]
```

Useful for making invisible layout voices.

Return list of skips. Changed in version 1.1.2: renamed `construct.skips_with_multipliers( )` to `skiptools.make_skips_with_multiplied_durations( )`.

skiptools.replace_leaves_in_expr_with_skips

`abjad.tools.skiptools.`**`replace_leaves_in_expr_with_skips`**`(`*expr*`)`

New in version 1.1.1. Replace leaves in *expr* with skips:

```
abjad> staff = Staff(Measure((2, 8), macros.scale(2)) * 2)
abjad> skiptools.replace_leaves_in_expr_with_skips(staff[0])
abjad> print staff.format
\new Staff {
  {
     \time 2/8
     s8
     s8
  }
  {
     \time 2/8
     c'8
     d'8
  }
}
```

Return none. Changed in version 1.1.2: renamed `leaftools.replace_leaves_with_skips_in( )` to `skiptools.replace_leaves_in_expr_with_skips( )`.

skiptools.yield_groups_of_skips_in_sequence

`abjad.tools.skiptools.`**`yield_groups_of_skips_in_sequence`**`(`*sequence*`)`

New in version 1.1.2. Yield groups of skips in *sequence*:

```
abjad> staff = Staff("c'8 d'8 s8 s8 <e' g'>8 <f' a'>8 g'8 a'8 s8 s8 <b' d''>8 <c'' e''>8")

abjad> f(staff)
\new Staff {
   c'8
   d'8
   s8
   s8
   <e' g'>8
   <f' a'>8
   g'8
   a'8
   s8
   s8
   <b' d''>8
   <c'' e''>8
}
```

```
abjad> for skip in skiptools.yield_groups_of_skips_in_sequence(staff):
...     skip
...
(Skip('s8'), Skip('s8'))
(Skip('s8'), Skip('s8'))
```

Return generator.

spannertools

spannertools.BeamSpanner

class `abjad.tools.spannertools.`**`BeamSpanner`**(*components=None*)
Bases: `abjad.tools.spannertools.Spanner.Spanner.Spanner`

Abjad beam spanner:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8 g'2")

abjad> f(staff)
\new Staff {
   c'8
   d'8
   e'8
   f'8
   g'2
}

abjad> spannertools.BeamSpanner(staff[:4])
BeamSpanner(c'8, d'8, e'8, f'8)

abjad> f(staff)
\new Staff {
   c'8 [
   d'8
   e'8
   f'8 ]
   g'2
}
```

Return beam spanner.

spannertools.BracketSpanner

class `abjad.tools.spannertools.`**`BracketSpanner`**(*components=None*)
Bases: `abjad.tools.spannertools.TextSpanner.TextSpanner.TextSpanner`

Abjad bracket spanner:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")

abjad> spannertools.BracketSpanner(staff[:])
BracketSpanner(c'8, d'8, e'8, f'8)

abjad> f(staff)
\new Staff {
   \override TextSpanner #'bound-details #'left #'text = #(markup #:draw-line '(0 . -1))
```

```
    \override TextSpanner #'bound-details #'left-broken #'text = ##f
    \override TextSpanner #'bound-details #'right #'text = #(markup #:draw-line '(0 . -1))
    \override TextSpanner #'bound-details #'right-broken #'text = ##f
    \override TextSpanner #'color = #red
    \override TextSpanner #'dash-fraction = #1
    \override TextSpanner #'staff-padding = #2
    \override TextSpanner #'thickness = #1.5
    c'8 \startTextSpan
    d'8
    e'8
    f'8 \stopTextSpan
    \revert TextSpanner #'bound-details #'left #'text
    \revert TextSpanner #'bound-details #'left-broken #'text
    \revert TextSpanner #'bound-details #'right #'text
    \revert TextSpanner #'bound-details #'right-broken #'text
    \revert TextSpanner #'color
    \revert TextSpanner #'dash-fraction
    \revert TextSpanner #'staff-padding
    \revert TextSpanner #'thickness
}
```

Render 1.5-unit thick solid red spanner.

Draw nibs at beginning and end of spanner.

Do not draw nibs at line breaks.

Return bracket spanner.

spannertools.ComplexBeamSpanner

class `abjad.tools.spannertools.`**`ComplexBeamSpanner`**(*components=None*, *lone=False*)

Bases: `abjad.tools.spannertools.BeamSpanner.BeamSpanner.BeamSpanner`

Abjad complex beam spanner:

```
abjad> staff = Staff("c'16 e'16 r16 f'16 g'2")

abjad> f(staff)
\new Staff {
    c'16
    e'16
    r16
    f'16
    g'2
}

abjad> spannertools.ComplexBeamSpanner(staff[:4])
ComplexBeamSpanner(c'16, e'16, r16, f'16)

abjad> f(staff)
\new Staff {
    \set stemLeftBeamCount = #0
    \set stemRightBeamCount = #2
    c'16 [
    \set stemLeftBeamCount = #2
    \set stemRightBeamCount = #2
    e'16 ]
    r16
```

```
    \set stemLeftBeamCount = #2
    \set stemRightBeamCount = #0
    f'16 [ ]
    g'2
}
```

Return complex beam spanner.

lone

Beam lone leaf and force beam nibs to left:

```
abjad> note = Note("c'16")

abjad> beam = spannertools.ComplexBeamSpanner([note], lone = 'left')

abjad> f(note)
\set stemLeftBeamCount = #2
\set stemRightBeamCount = #0
c'16 [ ]
```

Beam lone leaf and force beam nibs to right:

```
abjad> note = Note("c'16")

abjad> beam = spannertools.ComplexBeamSpanner([note], lone = 'right')

abjad> f(note)
\set stemLeftBeamCount = #0
\set stemRightBeamCount = #2
c'16 [ ]
```

Beam lone leaf and force beam nibs to both left and right:

```
abjad> note = Note("c'16")

abjad> beam = spannertools.ComplexBeamSpanner([note], lone = 'both')

abjad> f(note)
\set stemLeftBeamCount = #2
\set stemRightBeamCount = #2
c'16 [ ]
```

Beam lone leaf and accept LilyPond default nibs at both left and right:

```
abjad> note = Note("c'16")

abjad> beam = spannertools.ComplexBeamSpanner([note], lone = True)

abjad> f(note)
\set stemLeftBeamCount = #2
\set stemRightBeamCount = #2
c'16 [ ]
```

Do not beam lone leaf:

```
abjad> note = Note("c'16")

abjad> beam = spannertools.ComplexBeamSpanner([note], lone = False)
```

```
abjad> f(note)
c'16
```

Set to 'left', 'right', 'both', true or false as shown above.

Ignore this setting when spanner contains more than one leaf.

spannertools.CrescendoSpanner

class abjad.tools.spannertools.**CrescendoSpanner**(*components=None*, *include_rests=True*)

Bases: abjad.tools.spannertools.HairpinSpanner.HairpinSpanner.HairpinSpanner

Abjad crescendo spanner that includes rests:

```
abjad> staff = Staff("r4 c'8 d'8 e'8 f'8 r4")

abjad> f(staff)
\new Staff {
   r4
   c'8
   d'8
   e'8
   f'8
   r4
}

abjad> spannertools.CrescendoSpanner(staff[:], include_rests = True)
CrescendoSpanner(r4, c'8, d'8, e'8, f'8, r4)

abjad> f(staff)
\new Staff {
   r4 \<
   c'8
   d'8
   e'8
   f'8
   r4 \!
}
```

Abjad crescendo spanner that does not include rests:

```
abjad> staff = Staff("r4 c'8 d'8 e'8 f'8 r4")

abjad> f(staff)
\new Staff {
   r4
   c'8
   d'8
   e'8
   f'8
   r4
}

abjad> spannertools.CrescendoSpanner(staff[:], include_rests = False)
CrescendoSpanner(r4, c'8, d'8, e'8, f'8, r4)

abjad> f(staff)
\new Staff {
```

```
        r4
        c'8 \<
        d'8
        e'8
        f'8 \!
        r4
    }
```

Return crescendo spanner.

spannertools.DecrescendoSpanner

class `abjad.tools.spannertools.`**`DecrescendoSpanner`**(*components=None*, *include_rests=True*)

Bases: `abjad.tools.spannertools.HairpinSpanner.HairpinSpanner.HairpinSpanner`

Abjad decrescendo spanner that includes rests:

```
abjad> staff = Staff("r4 c'8 d'8 e'8 f'8 r4")

abjad> f(staff)
\new Staff {
    r4
    c'8
    d'8
    e'8
    f'8
    r4
}

abjad> spannertools.DecrescendoSpanner(staff[:], include_rests = True)
DecrescendoSpanner(r4, c'8, d'8, e'8, f'8, r4)

abjad> f(staff)
\new Staff {
    r4 \>
    c'8
    d'8
    e'8
    f'8
    r4 \!
}
```

Abjad decrescendo spanner that does not include rests:

```
abjad> staff = Staff("r4 c'8 d'8 e'8 f'8 r4")

abjad> f(staff)
\new Staff {
    r4
    c'8
    d'8
    e'8
    f'8
    r4
}
```

```
abjad> spannertools.DecrescendoSpanner(staff[:], include_rests = False)
DecrescendoSpanner(r4, c'8, d'8, e'8, f'8, r4)

abjad> f(staff)
\new Staff {
   r4
   c'8 \>
   d'8
   e'8
   f'8 \!
   r4
}
```

Return decrescendo spanner.

spannertools.DuratedComplexBeamSpanner

class `abjad.tools.spannertools.`**`DuratedComplexBeamSpanner`**(*components=None*, *durations=None*, *span=1*, *lone=False*)

Bases: `abjad.tools.spannertools.ComplexBeamSpanner.ComplexBeamSpanner.ComplexBeamSpanner`

Abjad durated complex beam spanner:

```
staff = Staff("c'16 d'16 e'16 f'16")

durations = [Duration(1, 8), Duration(1, 8)]
beam = spannertools.DuratedComplexBeamSpanner(staff[:], durations, 1)

f(staff)
\new Staff {
   \set stemLeftBeamCount = #0
   \set stemRightBeamCount = #2
   c'16 [
   \set stemLeftBeamCount = #2
   \set stemRightBeamCount = #1
   d'16
   \set stemLeftBeamCount = #1
   \set stemRightBeamCount = #2
   e'16
   \set stemLeftBeamCount = #2
   \set stemRightBeamCount = #0
   f'16 ]
}
```

Beam all beamable leaves in spanner explicitly.

Group leaves in spanner according to *durations*.

Span leaves between duration groups according to *span*.

Return durated complex beam spanner.

`durations`

Get spanner leaf group durations:

```
abjad> staff = Staff("c'16 d'16 e'16 f'16")
abjad> durations = [Duration(1, 8), Duration(1, 8)]
abjad> beam = spannertools.DuratedComplexBeamSpanner(staff[:], durations)
```

```
abjad> beam.durations
[Duration(1, 8), Duration(1, 8)]
```

Set spanner leaf group durations:

```
abjad> staff = Staff("c'16 d'16 e'16 f'16")
abjad> durations = [Duration(1, 8), Duration(1, 8)]
abjad> beam = spannertools.DuratedComplexBeamSpanner(staff[:], durations)
abjad> beam.durations = [Duration(1, 4)]
abjad> beam.durations
[Duration(1, 4)]
```

Set iterable.

span

Get top-level beam count:

```
abjad> staff = Staff("c'16 d'16 e'16 f'16")
abjad> durations = [Duration(1, 8), Duration(1, 8)]
abjad> beam = spannertools.DuratedComplexBeamSpanner(staff[:], durations, 1)
abjad> beam.span
1
```

Set top-level beam count:

```
abjad> staff = Staff("c'16 d'16 e'16 f'16")
abjad> durations = [Duration(1, 8), Duration(1, 8)]
abjad> beam = spannertools.DuratedComplexBeamSpanner(staff[:], durations, 1)
abjad> beam.span = 2
abjad> beam.span
2
```

Set nonnegative integer.

spannertools.DynamicTextSpanner

class `abjad.tools.spannertools.`**`DynamicTextSpanner`**(*components=None*, *mark=''*)

Bases: `abjad.tools.spannertools.Spanner.Spanner.Spanner`

Abjad dynamic text spanner:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")

abjad> spannertools.DynamicTextSpanner(staff[:], 'f')
DynamicTextSpanner(c'8, d'8, e'8, f'8)

abjad> f(staff)
\new Staff {
   c'8 \f
   d'8
   e'8
   f'8
}
```

Format dynamic *mark* at first leaf in spanner.

Return dynamic text spanner.

mark

Get dynamic string:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> dynamic_text_spanner = spannertools.DynamicTextSpanner(staff[:], 'f')
abjad> dynamic_text_spanner.mark
'f'
```

Set dynamic string:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> dynamic_text_spanner = spannertools.DynamicTextSpanner(staff[:], 'f')
abjad> dynamic_text_spanner.mark = 'p'
abjad> dynamic_text_spanner.mark
'p'
```

Set string.

spannertools.GlissandoSpanner

class `abjad.tools.spannertools.`**`GlissandoSpanner`**(*components=None*)

Bases: `abjad.tools.spannertools.Spanner.Spanner.Spanner`

Abjad glissando spanner:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")

abjad> spannertools.GlissandoSpanner(staff[:])
GlissandoSpanner(c'8, d'8, e'8, f'8)

abjad> f(staff)
\new Staff {
   c'8 \glissando
   d'8 \glissando
   e'8 \glissando
   f'8
}
```

Format nonlast leaves in spanner with LilyPond glissando command.

Return glissando spanner.

spannertools.HairpinSpanner

class `abjad.tools.spannertools.`**`HairpinSpanner`**(*components=None, descriptor='<', include_rests=True*)

Bases: `abjad.tools.spannertools.Spanner.Spanner.Spanner`

Abjad hairpin spanner that includes rests:

```
abjad> staff = Staff("r4 c'8 d'8 e'8 f'8 r4")

abjad> f(staff)
\new Staff {
   r4
   c'8
   d'8
   e'8
   f'8
   r4
}
```

```
abjad> spannertools.HairpinSpanner(staff[:], 'p < f', include_rests = True)
HairpinSpanner(r4, c'8, d'8, e'8, f'8, r4)

abjad> f(staff)
\new Staff {
   r4 \< \p
   c'8
   d'8
   e'8
   f'8
   r4 \f
}
```

Abjad hairpin spanner that does not include rests:

```
abjad> staff = Staff("r4 c'8 d'8 e'8 f'8 r4")

abjad> f(staff)
\new Staff {
   r4
   c'8
   d'8
   e'8
   f'8
   r4
}

abjad> spannertools.HairpinSpanner(staff[:], 'p < f', include_rests = False)
HairpinSpanner(r4, c'8, d'8, e'8, f'8, r4)

abjad> f(staff)
\new Staff {
   r4
   c'8 \< \p
   d'8
   e'8
   f'8 \f
   r4
}
```

Return hairpin spanner.

include_rests

Get boolean hairpin rests setting:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> hairpin = spannertools.HairpinSpanner(staff[:], 'p < f', include_rests = True)
abjad> hairpin.include_rests
True
```

Set boolean hairpin rests setting:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> hairpin = spannertools.HairpinSpanner(staff[:], 'p < f', include_rests = True)
abjad> hairpin.include_rests = False
abjad> hairpin.include_rests
False
```

Set boolean.

static is_hairpin_shape_string (*arg*)

True when *arg* is a hairpin shape string. Otherwise false:

```
abjad> spannertools.HairpinSpanner.is_hairpin_shape_string('<')
True
```

Return boolean.

shape_string

Get hairpin shape string:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> hairpin = spannertools.HairpinSpanner(staff[:], 'p < f')
abjad> hairpin.shape_string
'<'
```

Set hairpin shape string:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> hairpin = spannertools.HairpinSpanner(staff[:], 'p < f')
abjad> hairpin.shape_string = '>'
abjad> hairpin.shape_string
'>'
```

Set string.

start_dynamic_string

Get hairpin start dynamic string:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> hairpin = spannertools.HairpinSpanner(staff[:], 'p < f')
abjad> hairpin.start_dynamic_string
'p'
```

Set hairpin start dynamic string:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> hairpin = spannertools.HairpinSpanner(staff[:], 'p < f')
abjad> hairpin.start_dynamic_string = 'mf'
abjad> hairpin.start_dynamic_string
'mf'
```

Set string.

stop_dynamic_string

Get hairpin stop dynamic string:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> hairpin = spannertools.HairpinSpanner(staff[:], 'p < f')
abjad> hairpin.stop_dynamic_string
'f'
```

Set hairpin stop dynamic string:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> hairpin = spannertools.HairpinSpanner(staff[:], 'p < f')
abjad> hairpin.stop_dynamic_string = 'mf'
abjad> hairpin.stop_dynamic_string
'mf'
```

Set string.

spannertools.HiddenStaffSpanner

class `abjad.tools.spannertools.`**`HiddenStaffSpanner`**`(`*components=None*`)`

Bases: `abjad.tools.spannertools.Spanner.Spanner.Spanner`

Abjad hidden staff spanner:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")

abjad> spannertools.HiddenStaffSpanner(staff[:2])
HiddenStaffSpanner(c'8, d'8)

abjad> f(staff)
\new Staff {
   \stopStaff
   c'8
   d'8
   \startStaff
   e'8
   f'8
}
```

Hide staff behind leaves in spanner.

Return hidden staff spanner.

spannertools.MeasuredComplexBeamSpanner

class `abjad.tools.spannertools.`**`MeasuredComplexBeamSpanner`**`(`*components=None, lone=False, span=1*`)`

Bases: `abjad.tools.spannertools.ComplexBeamSpanner.ComplexBeamSpanner.ComplexBeamSpanner`

Abjad measured complex beam spanner:

```
abjad> staff = Staff([Measure((2, 16), "c'16 d'16"), Measure((2, 16), "e'16 f'16")])

abjad> spannertools.MeasuredComplexBeamSpanner(staff.leaves)
MeasuredComplexBeamSpanner(c'16, d'16, e'16, f'16)

abjad> f(staff)
\new Staff {
   {
      \time 2/16
      \set stemLeftBeamCount = #0
      \set stemRightBeamCount = #2
      c'16 [
      \set stemLeftBeamCount = #2
      \set stemRightBeamCount = #1
      d'16
   }
   {
      \time 2/16
      \set stemLeftBeamCount = #1
      \set stemRightBeamCount = #2
      e'16
      \set stemLeftBeamCount = #2
      \set stemRightBeamCount = #0
      f'16 ]
```

```
    }
}
```

Beam leaves in spanner explicitly.

Group leaves by measures.

Format top-level *span* beam between measures.

Return measured complex beam spanner.

span

Get top-level beam count:

```
abjad> staff = Staff([Measure((2, 16), "c'16 d'16"), Measure((2, 16), "e'16 f'16")])
abjad> beam = spannertools.MeasuredComplexBeamSpanner(staff.leaves)
abjad> beam.span
1
```

Set top-level beam count:

```
abjad> staff = Staff([Measure((2, 16), "c'16 d'16"), Measure((2, 16), "e'16 f'16")])
abjad> beam = spannertools.MeasuredComplexBeamSpanner(staff.leaves)
abjad> beam.span = 2
abjad> beam.span
2
```

Set nonnegative integer.

spannertools.MetricGridSpanner

class `abjad.tools.spannertools.`**`MetricGridSpanner`**(*components=None*, *meters=None*)

Bases: `abjad.tools.spannertools.Spanner.Spanner.Spanner`

Abjad metric grid spanner:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8 g'8 a'8 b'8 c'8")

abjad> spannertools.MetricGridSpanner(staff.leaves, meters = [(1, 8), (1, 4)])
MetricGridSpanner(c'8, d'8, e'8, f'8, g'8, a'8, b'8, c'8)

abjad> f(staff)
\new Staff {
   \time 1/8
   c'8
   \time 1/4
   d'8
   e'8
   \time 1/8
   f'8
   \time 1/4
   g'8
   a'8
   \time 1/8
   b'8
   \time 1/4
   c'8
}
```

Format leaves in spanner cyclically with *meters*.

Return metric grid spanner.

meters

Get metric grid meters:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8 g'8 a'8 b'8 c'8")
abjad> metric_grid_spanner = spannertools.MetricGridSpanner(staff.leaves, meters = [(1, 8),
abjad> list(metric_grid_spanner.meters)
[(Meter(1, 8), 0, False), (Meter(1, 4), Duration(1, 8), False), (Meter(1, 8), Duration(3, 8)
 (Meter(1, 4), Duration(1, 2), False), (Meter(1, 8), Duration(3, 4), False), (Meter(1, 4), D
```

Set metric grid meters:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8 g'8 a'8 b'8 c'8")
abjad> metric_grid_spanner = spannertools.MetricGridSpanner(staff.leaves, meters = [(1, 8),
abjad> metric_grid_spanner.meters = [(1, 4)]
abjad> list(metric_grid_spanner.meters)
[(Meter(1, 4), 0, False), (Meter(1, 4), Duration(1, 4), True), (Meter(1, 4), Duration(1, 2),
```

Set iterable.

split_on_bar()

Temporarily unavailable.

splitting_condition(*leaf*)

User-definable boolean function to determine whether leaf should be split.

Function defaults to return true.

spannertools.MultipartBeamSpanner

class `abjad.tools.spannertools.`**`MultipartBeamSpanner`**(*components=None*)

Bases: `abjad.tools.spannertools.BeamSpanner.BeamSpanner.BeamSpanner` New in version 1.1.2. Abjad multipart beam spanner:

```
abjad> staff = Staff("c'8 d'8 e'4 f'8 g'8 r4")

abjad> spannertools.MultipartBeamSpanner(staff[:])
MultipartBeamSpanner(c'8, d'8, e'4, f'8, g'8, r4)

abjad> f(staff)
\new Staff {
   c'8 [
   d'8 ]
   e'4
   f'8 [
   g'8 ]
   r4
}
```

Avoid rests.

Avoid large-duration notes.

Return multipart beam spanner.

spannertools.OctavationSpanner

class `abjad.tools.spannertools.OctavationSpanner`(*components=None*, *start=0*, *stop=0*)

Bases: `abjad.tools.spannertools.Spanner.Spanner.Spanner`

Abjad octavation spanner:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")

abjad> spanner = spannertools.OctavationSpanner(staff[:], start = 1)

abjad> f(staff)
\new Staff {
   \ottava #1
   c'8
   d'8
   e'8
   f'8
   \ottava #0
}
```

Return octavation spanner.

start

Get octavation start:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> octavation = spannertools.OctavationSpanner(staff[:], start = 1)
abjad> octavation.start
1
```

Set octavation start:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> octavation = spannertools.OctavationSpanner(staff[:], start = 1)
abjad> octavation.start
1
```

Set integer.

stop

Get octavation stop:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> octavation = spannertools.OctavationSpanner(staff[:], start = 2, stop = 1)
abjad> octavation.stop
1
```

Set octavation stop:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> octavation = spannertools.OctavationSpanner(staff[:], start = 2, stop = 1)
abjad> octavation.stop = 0
abjad> octavation.stop
0
```

Set integer.

spannertools.PhrasingSlurSpanner

class `abjad.tools.spannertools.`**`PhrasingSlurSpanner`**(*components=None*)
Bases: `abjad.tools.spannertools.Spanner.Spanner.Spanner`

Abjad phrasing slur spanner:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")

abjad> spannertools.PhrasingSlurSpanner(staff[:])
PhrasingSlurSpanner(c'8, d'8, e'8, f'8)

abjad> f(staff)
\new Staff {
   c'8 \(
   d'8
   e'8
   f'8 \)
}
```

Return phrasing slur spanner.

spannertools.PianoPedalSpanner

class `abjad.tools.spannertools.`**`PianoPedalSpanner`**(*components=None*)
Bases: `abjad.tools.spannertools.Spanner.Spanner.Spanner`

Abjad piano pedal spanner:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")

abjad> spannertools.PianoPedalSpanner(staff[:])
PianoPedalSpanner(c'8, d'8, e'8, f'8)

abjad> f(staff)
\new Staff {
   \set Staff.pedalSustainStyle = #'mixed
   c'8 \sustainOn
   d'8
   e'8
   f'8 \sustainOff
}
```

Return piano pedal spanner.

`kind`

Get piano pedal spanner kind:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> spanner = spannertools.PianoPedalSpanner(staff[:])
abjad> spanner.kind
'sustain'
```

Set piano pedal spanner kind:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> spanner = spannertools.PianoPedalSpanner(staff[:])
abjad> spanner.kind = 'sostenuto'
abjad> spanner.kind
'sostenuto'
```

Acceptable values 'sustain', 'sostenuto', 'corda'.

style

Get piano pedal spanner style:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> spanner = spannertools.PianoPedalSpanner(staff[:])
abjad> spanner.style
'mixed'
```

Set piano pedal spanner style:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> spanner = spannertools.PianoPedalSpanner(staff[:])
abjad> spanner.style = 'bracket'
abjad> spanner.style
'bracket'
```

Acceptable values 'mixed', 'bracket', 'text'.

spannertools.SlurSpanner

class abjad.tools.spannertools.**SlurSpanner**(*components=None*)

Bases: abjad.tools.spannertools.Spanner.Spanner.Spanner

Abjad slur spanner:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")

abjad> spannertools.SlurSpanner(staff[:])
SlurSpanner(c'8, d'8, e'8, f'8)

abjad> f(staff)
\new Staff {
   c'8 (
   d'8
   e'8
   f'8 )
}
```

Return slur spanner.

spannertools.Spanner

class abjad.tools.spannertools.**Spanner**(*components=None*)

Bases: abjad.core._StrictComparator._StrictComparator._StrictComparator

Any type of notation object that stretches horizontally and encompasses some number of notes, rest, chords, tuplets, measures, voices or other Abjad components.

Beams, slurs, hairpins, trills, glissandi and piano pedal brackets all stretch horizontally on the page to encompass multiple notes and all implement as Abjad spanners. That is, these spanner all have an obvious graphic reality with definite start-, stop- and midpoints.

Abjad also implements a number of spanners of a different type, such as tempo and instrument spanners, which mark a group of notes, rests, chords or measues as carrying a certain tempo or being played by a certain instrument.

The spanner class described here abstracts the functionality that all such spanners, both graphic and nongraphics, share. This shared functionality includes methods to add, remove, inspect and test components governed by the spanner, as well as basic formatting properties. The other spanner classes, such as beam and glissando, all inherit from this class and receive the functionality implemented here.

append (*component*)

Add *component* to right of spanner.

```
abjad> voice = Voice(macros.scale(4))
abjad> spanner = spannertools.Spanner(voice[:2])
abjad> spanner
Spanner(c'8, d'8)

abjad> spanner.append(voice[2])
abjad> spanner
Spanner(c'8, d'8, e'8)
```

Return none.

append_left (*component*)

Add *component* to left of spanner.

```
abjad> voice = Voice(macros.scale(4))
abjad> spanner = spannertools.Spanner(voice[2:])
abjad> spanner
Spanner(e'8, f'8)

abjad> spanner.append_left(voice[1])
abjad> spanner
Spanner(d'8, e'8, f'8)
```

Return none.

clear ()

Remove all components from spanner:

```
abjad> voice = Voice(macros.scale(4))
abjad> spanner = spannertools.Spanner(voice[:])
abjad> spanner
Spanner(c'8, d'8, e'8, f'8)

abjad> spanner.clear( )
abjad> spanner
Spanner( )
```

Return none.

components

Return read-only tuple of components in spanner.

```
abjad> voice = Voice(macros.scale(4))
abjad> spanner = spannertools.Spanner(voice[:2])
abjad> spanner.components
(Note("c'8"), Note("d'8"))
```

Changed in version 1.1.1: Now returns an (immutable) tuple instead of a (mutable) list.

duration

Return read-only reference to spanner duration interface.

Spanner duration interface implements `written`, `preprolated` and `prolated` attributes.

```
abjad> voice = Voice(macros.scale(4))
abjad> spanner = spannertools.Spanner(voice[:2])
abjad> spanner
Spanner(c'8, d'8)

abjad> spanner.duration.written
Duration(1, 4)

abjad> spanner.duration.preprolated
Duration(1, 4)

abjad> spanner.duration.prolated
Duration(1, 4)
```

Spanner duration interface also implements `seconds` attribute.

extend (*components*)

Add iterable *components* to right of spanner:

```
abjad> voice = Voice(macros.scale(4))
abjad> spanner = spannertools.Spanner(voice[:2])
abjad> spanner
Spanner(c'8, d'8)

abjad> spanner.extend(voice[2:])
abjad> spanner
Spanner(c'8, d'8, e'8, f'8)
```

Return none.

extend_left (*components*)

Add iterable *components* to left of spanner:

```
abjad> voice = Voice(macros.scale(4))
abjad> spanner = spannertools.Spanner(voice[2:])
abjad> spanner
Spanner(e'8, f'8)

abjad> spanner.extend_left(voice[:2])
abjad> spanner
Spanner(c'8, d'8, e'8, f'8)
```

Return none.

fracture (*i*, *direction='both'*)

Fracture spanner at *direction* of component at index *i*.

Valid values for *direction* are `'left'`, `'right'` and `'both'`.

Return original, left and right spanners.

```
abjad> voice = Voice(macros.scale(4))
abjad> beam = spannertools.BeamSpanner(voice[:])
abjad> beam
BeamSpanner(c'8, d'8, e'8, f'8)

abjad> beam.fracture(1, direction = 'left')
(BeamSpanner(c'8, d'8, e'8, f'8), BeamSpanner(c'8), BeamSpanner(d'8, e'8, f'8))
```

```
abjad> print voice.format
\new Voice {
        c'8 [ ]
        d'8 [
        e'8
        f'8 ]
}
```

Return tuple.

fuse (*spanner*)

Fuse contiguous spanners.

Return new spanner.

```
abjad> voice = Voice(macros.scale(4))
abjad> left_beam = spannertools.BeamSpanner(voice[:2])
abjad> right_beam = spannertools.BeamSpanner(voice[2:])

abjad> print voice.format
\new Voice {
        c'8 [
        d'8 ]
        e'8 [
        f'8 ]
}

abjad> left_beam.fuse(right_beam)
[(BeamSpanner(c'8, d'8), BeamSpanner(e'8, f'8), BeamSpanner(c'8, d'8, e'8, f'8))]

abjad> print voice.format
\new Voice {
        c'8 [
        d'8
        e'8
        f'8 ]
}
```

Todo

Return (immutable) tuple instead of (mutable) list.

---

index (*component*)

Return nonnegative integer index of *component* in spanner.

```
abjad> voice = Voice(macros.scale(4))
abjad> spanner = spannertools.Spanner(voice[2:])
abjad> spanner
Spanner(e'8, f'8)

abjad> spanner.index(voice[-2])
0
```

Return nonnegative integer.

leaves

Return read-only tuple of leaves in spanner.

```
abjad> voice = Voice(macros.scale(4))
abjad> spanner = spannertools.Spanner(voice[:2])
abjad> spanner.leaves
(Note("c'8"), Note("d'8"))
```

Changed in version 1.1.1: Now returns an (immutable) tuple instead of a (mutable) list.

Note: When dealing with large, complex scores accessing this attribute can take some time. Best to make a local copy with leaves = spanner.leaves first. Or use spanner- specific iteration tools.

---

offset
New in version 1.1.1. Return read-only reference to spanner offset interface.

Spanner offset interface implements `start` and `stop` attributes.

```
abjad> voice = Voice(macros.scale(4))
abjad> spanner = spannertools.Spanner(voice[2:])
abjad> spanner
Spanner(e'8, f'8)

abjad> spanner._offset.start
Duration(1, 4)

abjad> spanner._offset.stop
Duration(1, 2)
```

Return duration.

override
LilyPond grob override component plug-in.

pop()
Remove and return rightmost component in spanner.

```
abjad> voice = Voice(macros.scale(4))
abjad> spanner = spannertools.Spanner(voice[:])
abjad> spanner
Spanner(c'8, d'8, e'8, f'8)

abjad> spanner.pop( )
Note("f'8")

abjad> spanner
Spanner(c'8, d'8, e'8)
```

Return component.

pop_left()
Remove and return leftmost component in spanner.

```
abjad> voice = Voice(macros.scale(4))
abjad> spanner = spannertools.Spanner(voice[:])
abjad> spanner
Spanner(c'8, d'8, e'8, f'8)

abjad> spanner.pop_left( )
Note("c'8")
```

```
abjad> spanner
Spanner(d'8, e'8, f'8)
```

Return component.

set
LilyPond context setting component plug-in.

spannertools.StaffLinesSpanner

class `abjad.tools.spannertools.`**`StaffLinesSpanner`**`(`*components=None*, *arg=5*`)`
Bases: `abjad.tools.spannertools.Spanner.Spanner.Spanner`

Abjad staff lines spanner:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")

abjad> spannertools.StaffLinesSpanner(staff[:2], 1)
StaffLinesSpanner(c'8, d'8)

abjad> f(staff)
\new Staff {
   \stopStaff
   \override Staff.StaffSymbol #'line-count = #1
   \startStaff
   c'8
   d'8
   \stopStaff
   \revert Staff.StaffSymbol #'line-count
   \startStaff
   e'8
   f'8
}
```

Staff lines spanner handles changing either the line-count or the line-positions property of the StaffSymbol grob, as well as automatically stopping and restarting the staff so that the change may take place.

Return staff lines spanner.

lines
Get staff lines spanner line count:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> spanner = spannertools.StaffLinesSpanner(staff[:2], 1)
abjad> spanner.lines
1
```

Set staff lines spanner line count:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> spanner = spannertools.StaffLinesSpanner(staff[:2], 1)
abjad> spanner.lines = 2
abjad> spanner.lines
2
```

Set integer.

spannertools.TextScriptSpanner

class `abjad.tools.spannertools.`**`TextScriptSpanner`**(*components=None*)

Bases: `abjad.tools.spannertools.Spanner.Spanner.Spanner` New in version 1.1.2. Abjad text script spanner:

```
abjad> staff = Staff(macros.scale(4))

abjad> spanner = spannertools.TextScriptSpanner(staff[:])
abjad> spanner.override.text_script.color = 'red'
abjad> markuptools.Markup(r'\italic { espressivo }', 'up')(staff[1])
Markup('\\italic { espressivo }', 'up')

abjad> f(staff)
\new Staff {
        \override TextScript #'color = #red
        c'8
        d'8 ^ \markup { \italic { espressivo } }
        e'8
        f'8
        \revert TextScript #'color
}
```

Override LilyPond TextScript grob.

Return text script spanner.

spannertools.TextSpanner

class `abjad.tools.spannertools.`**`TextSpanner`**(*components=None*)

Bases: `abjad.tools.spannertools.Spanner.Spanner.Spanner` New in version 1.1.2. Abjad text spanner:

```
abjad> staff = Staff(macros.scale(4))
abjad> text_spanner = spannertools.TextSpanner(staff[:])

abjad> markup = markuptools.Markup('(markup #:bold #:italic "foo")', style_string = 'scheme')
abjad> text_spanner.override.text_spanner.bound_details__left__text = markup
abjad> markup = markuptools.Markup("(markup #:draw-line '(0 . -1))", style_string = 'scheme')
abjad> text_spanner.override.text_spanner.bound_details__right__text = markup
abjad> text_spanner.override.text_spanner.dash_fraction = 1

abjad> f(staff)
\new Staff {
   \override TextSpanner #'bound-details #'left #'text = #(markup #:bold #:italic "foo")
   \override TextSpanner #'bound-details #'right #'text = #(markup #:draw-line '(0 . -1))
   \override TextSpanner #'dash-fraction = #1
   c'8 \startTextSpan
   d'8
   e'8
   f'8 \stopTextSpan
   \revert TextSpanner #'bound-details #'left #'text
   \revert TextSpanner #'bound-details #'right #'text
   \revert TextSpanner #'dash-fraction
}
```

Override LilyPond TextSpanner grob.

Return text spanner.

spannertools.TrillSpanner

class abjad.tools.spannertools.**TrillSpanner**(*components=None*)

Bases: abjad.tools.spannertools.Spanner.Spanner.Spanner

Abjad trill spanner:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")

abjad> spannertools.TrillSpanner(staff[:])
TrillSpanner(c'8, d'8, e'8, f'8)

abjad> f(staff)
\new Staff {
   c'8 \startTrillSpan
   d'8
   e'8
   f'8 \stopTrillSpan
}
```

Override LilyPond TrillSpanner grob.

Return trill spanner.

pitch

Optional read / write pitch for pitched trills.

```
abjad> t = Staff(macros.scale(4))
abjad> trill = spannertools.TrillSpanner(t[:2])
abjad> trill.pitch = pitchtools.NamedChromaticPitch('cs', 4)


abjad> f(t)
\new Staff {
   \pitchedTrill c'8 \startTrillSpan cs'
   d'8 \stopTrillSpan
   e'8
   f'8
}
```

Set pitch.

spannertools.destroy_all_spanners_attached_to_component

abjad.tools.spannertools.**destroy_all_spanners_attached_to_component**(*component*, *klass=None*)

New in version 1.1.1. Destroy all spanners attached to *component*:

```
abjad> staff = Staff(macros.scale(4))
abjad> beam = spannertools.BeamSpanner(staff.leaves)
abjad> slur = spannertools.SlurSpanner(staff.leaves)
abjad> trill = spannertools.TrillSpanner(staff)
abjad> f(staff)
\new Staff {
   c'8 [ ( \startTrillSpan
   d'8
   e'8
   f'8 ] ) \stopTrillSpan
}
```

```
abjad> spannertools.destroy_all_spanners_attached_to_component(staff[0])
abjad> f(staff)
\new Staff {
   c'8 \startTrillSpan
   d'8
   e'8
   f'8 \stopTrillSpan
}
```

Return none.

spannertools.find_index_of_spanner_component_at_score_offset

abjad.tools.spannertools.**find_index_of_spanner_component_at_score_offset**(*spanner*, *score_offset*)

Return index of component in 'spanner' that begins at exactly 'score_offset':

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> beam = spannertools.BeamSpanner(staff.leaves)

abjad> f(staff)
\new Staff {
   c'8 [
   d'8
   e'8
   f'8 ]
}

abjad> spannertools.find_index_of_spanner_component_at_score_offset(beam, Duration(3, 8))
3
```

Raise spanner population error when no component in *spanner* begins at exactly *score_offset*. Changed in version 1.1.2: renamed `spannertools.find_index_at_score_offset()` to `spannertools.find_index_of_spanner_component_at_score_offset()`.

spannertools.find_spanner_component_starting_at_exactly_score_offset

abjad.tools.spannertools.**find_spanner_component_starting_at_exactly_score_offset**(*spanner*, *score_offset*)

Find *spanner* component starting at exactly *score_offset*:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> beam = spannertools.BeamSpanner(staff.leaves)

abjad> f(staff)
\new Staff {
   c'8 [
   d'8
   e'8
   f'8 ]
}

abjad> spannertools.find_spanner_component_starting_at_exactly_score_offset(beam, Duration(3, 8)
Note("f'8")
```

When no *spanner* component starts at exactly *score_offset* return none.

Return *spanner* component or none. Changed in version 1.1.2: renamed `spannertools.find_component_at_score_offset( )` to `spannertools.find_spanner_component_starting_at_exactly_score_offset( )`.

spannertools.fracture_all_spanners_attached_to_component

`abjad.tools.spannertools.`**`fracture_all_spanners_attached_to_component`**(*component*, *direction='both'*, *klass=None*)

New in version 1.1.1. Fracture all spanners attached to *component* according to *direction*:

```
abjad> staff = Staff(macros.scale(4))
abjad> beam = spannertools.BeamSpanner(staff.leaves)
abjad> slur = spannertools.SlurSpanner(staff.leaves)
abjad> trill = spannertools.TrillSpanner(staff)
abjad> f(staff)
\new Staff {
   c'8 [ ( \startTrillSpan
   d'8
   e'8
   f'8 ] ) \stopTrillSpan
}

abjad> spannertools.fracture_all_spanners_attached_to_component(staff[1], 'right')
[(BeamSpanner(c'8, d'8, e'8, f'8), BeamSpanner(c'8, d'8), BeamSpanner(e'8, f'8)), (SlurSpanner(c
abjad> f(staff)
\new Staff {
   c'8 [ ( \startTrillSpan
   d'8 ] )
   e'8 [ (
   f'8 ] ) \stopTrillSpan
}
```

Set *direction* to left, right or both.

spannertools.fracture_spanners_that_cross_components

`abjad.tools.spannertools.`**`fracture_spanners_that_cross_components`**(*components*)

Fracture to the left of the leftmost component. Fracture to the right of the rightmost component. Do not fracture spanners of any components at higher levels of score. Do not fracture spanners of any components at lower levels of score. Return components.

Components must be thread-contiguous. Some spanners may copy during fracture. This helper is public-safe.

Example:

```
t = Staff(Container(notetools.make_repeated_notes(2)) * 3)
macros.diatonicize(t)
spannertools.CrescendoSpanner(t)
spannertools.BeamSpanner(t[:])
spannertools.TrillSpanner(t.leaves)

\new Staff {
   {
      c'8 [ \< \startTrillSpan
      d'8
```

```
    }
    {
       e'8
       f'8
    }
    {
       g'8
       a'8 ] \! \stopTrillSpan
    }   }
```

```
spannertools.fracture_spanners_that_cross_components(t[1:2])
```

```
\new Staff {
   {
      c'8 [ \< \startTrillSpan
      d'8 ]
   }
   {
      e'8 [
      f'8 ]
   }
   {
      g'8 [
      a'8 ] \! \stopTrillSpan
   }
}
```

Changed in version 1.1.2: renamed `spannertools.fracture_crossing( )` to `spannertools.fracture_spanners_that_cross_components( )`.

spannertools.get_beam_spanner_attached_to_component

`abjad.tools.spannertools.`**`get_beam_spanner_attached_to_component`**`(`*component*`)`

New in version 1.1.2. Get the only beam spanner attached to *component*:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> beam = spannertools.BeamSpanner(staff.leaves)
```

```
abjad> f(staff)
\new Staff {
   c'8 [
   d'8
   e'8
   f'8 ]
}
```

```
abjad> spannertools.get_beam_spanner_attached_to_component(staff[0])
BeamSpanner(c'8, d'8, e'8, f'8)
```

```
abjad> _ is beam
True
```

Return beam spanner.

Raise missing spanner error when no beam spanner attached to *component*.

Raise extra spanner error when more than one beam spanner attached to *component*. Changed in version 1.1.2: renamed `beamtools.get_beam_spanner( )` to spannertools.get_beam_spanner_attached_to_component( ).Changed in version 1.1.2: renamed beamtools.get_beam_spanner_attached_to_component( ) to spannertools.get_beam_spanner_attached_to_component( ).

spannertools.get_nth_leaf_in_spanner

abjad.tools.spannertools.**get_nth_leaf_in_spanner**(*spanner*, *idx*)

Get nth leaf in spanner, no matter how complicated the nesting situation. Changed in version 1.1.2: renamed spannertools.get_nth_leaf( ) to spannertools.get_nth_leaf_in_spanner( ).

spannertools.get_spanners_attached_to_any_improper_child_of_component

abjad.tools.spannertools.**get_spanners_attached_to_any_improper_child_of_component**(*component*, *klass=None*)

New in version 1.1.2. Get all spanners attached to any improper children of *component*:

```
abjad> staff = Staff(macros.scale(4))
abjad> beam = spannertools.BeamSpanner(staff.leaves)
abjad> first_slur = spannertools.SlurSpanner(staff.leaves[:2])
abjad> second_slur = spannertools.SlurSpanner(staff.leaves[2:])
abjad> trill = spannertools.TrillSpanner(staff)

abjad> f(staff)
\new Staff {
   c'8 [ ( \startTrillSpan
   d'8 )
   e'8 (
   f'8 ] ) \stopTrillSpan
}

abjad> len(spannertools.get_spanners_attached_to_any_improper_child_of_component(staff)) == 4
True
```

Get all spanners of *klass* attached to any proper children of *component*:

```
abjad> spanner_klass = spannertools.SlurSpanner
abjad> spannertools.get_spanners_attached_to_any_proper_child_of_component(staff, spanner_klass)
set([SlurSpanner(c'8, d'8), SlurSpanner(e'8, f'8)])
```

Get all spanners of any *klass* attached to any proper children of *component*:

```
abjad> spanner_klasses = (spannertools.SlurSpanner, spannertools.BeamSpanner)
abjad> spannertools.get_spanners_attached_to_any_proper_child_of_component(staff, spanner_klasse
set([BeamSpanner(c'8, d'8, e'8, f'8), SlurSpanner(c'8, d'8), SlurSpanner(e'8, f'8)])
```

Return unordered set of zero or more spanners. Changed in version 1.1.2: renamed spannertools.get_all_spanners_attached_to_any_improper_children_of_component( ) to spannertools.get_spanners_attached_to_any_improper_child_of_component( ).Changed in version 1.1.2: renamed spannertools.get_all_spanners_attached_to_any_improper_child_o) to spannertools.get_spanners_attached_to_any_improper_child_of_component( ).

spannertools.get_spanners_attached_to_any_improper_parent_of_component

abjad.tools.spannertools.**get_spanners_attached_to_any_improper_parent_of_component**(*component*, *klass=None*)

New in version 1.1.1. Get all spanners attached to improper parentage of *component*:

```
abjad> staff = Staff(macros.scale(4))
abjad> beam = spannertools.BeamSpanner(staff.leaves)
abjad> slur = spannertools.SlurSpanner(staff.leaves)
abjad> trill = spannertools.TrillSpanner(staff)
abjad> f(staff)
\new Staff {
   c'8 [ ( \startTrillSpan
   d'8
   e'8
   f'8 ] ) \stopTrillSpan
}
```

```
abjad> spannertools.get_spanners_attached_to_any_improper_parent_of_component(staff[0])
set([BeamSpanner(c'8, d'8, e'8, f'8), SlurSpanner(c'8, d'8, e'8, f'8), TrillSpanner({c'8, d'8, e
```

Return unordered set of zero or more spanners. Changed in version 1.1.2: renamed `spannertools.get_all_spanners_attached_to_improper_parentage_of_component()` to `spannertools.get_spanners_attached_to_any_improper_parent_of_component()`.Changed in version 1.1.2: renamed `spannertools.get_all_spanners_attached_to_any_improper_parent_` `)` to `spannertools.get_spanners_attached_to_any_improper_parent_of_component()`.

spannertools.get_spanners_attached_to_any_proper_child_of_component

abjad.tools.spannertools.**get_spanners_attached_to_any_proper_child_of_component**(*component*, *klass=None*)

New in version 1.1.2. Get all spanners attached to any proper children of *component*:

```
abjad> staff = Staff(macros.scale(4))
abjad> beam = spannertools.BeamSpanner(staff.leaves)
abjad> first_slur = spannertools.SlurSpanner(staff.leaves[:2])
abjad> second_slur = spannertools.SlurSpanner(staff.leaves[2:])
abjad> trill = spannertools.TrillSpanner(staff)
```

```
abjad> f(staff)
\new Staff {
   c'8 [ ( \startTrillSpan
   d'8 )
   e'8 (
   f'8 ] ) \stopTrillSpan
}
```

```
abjad> len(spannertools.get_spanners_attached_to_any_proper_child_of_component(staff)) == 3
True
```

Get all spanners of *klass* attached to any proper children of *component*:

```
abjad> spanner_klass = spannertools.SlurSpanner
abjad> spannertools.get_spanners_attached_to_any_proper_child_of_component(staff, spanner_klass)
set([SlurSpanner(c'8, d'8), SlurSpanner(e'8, f'8)])
```

Get all spanners of any *klass* attached to any proper children of *component*:

```
abjad> spanner_klasses = (spannertools.SlurSpanner, spannertools.BeamSpanner)
abjad> spannertools.get_spanners_attached_to_any_proper_child_of_component(staff, spanner_klasse
set([BeamSpanner(c'8, d'8, e'8, f'8), SlurSpanner(c'8, d'8), SlurSpanner(e'8, f'8)])
```

Return unordered set of zero or more spanners. Changed in version 1.1.2: renamed `spannertools.get_all_spanners_attached_to_any_proper_children_of_component( )` to `spannertools.get_spanners_attached_to_any_proper_child_of_component( )`.Changed in version 1.1.2: renamed `spannertools.get_all_spanners_attached_to_any_proper_child_of_( )` to `spannertools.get_spanners_attached_to_any_proper_child_of_component( )`.

spannertools.get_spanners_attached_to_any_proper_parent_of_component

`abjad.tools.spannertools.`**`get_spanners_attached_to_any_proper_parent_of_component`**`(`*component*, *klass=None*`)`

New in version 1.1.2. Get all spanners attached to any proper parent of *component*:

```
abjad> staff = Staff(macros.scale(4))
abjad> beam = spannertools.BeamSpanner(staff.leaves)
abjad> slur = spannertools.SlurSpanner(staff.leaves)
abjad> trill = spannertools.TrillSpanner(staff)
abjad> f(staff)
\new Staff {
   c'8 [ ( \startTrillSpan
   d'8
   e'8
   f'8 ] ) \stopTrillSpan
}

abjad> spannertools.get_spanners_attached_to_any_proper_parent_of_component(staff[0])
set([TrillSpanner({c'8, d'8, e'8, f'8})])
```

Return unordered set of zero or more spanners. Changed in version 1.1.2: renamed `spannertools.get_all_spanners_attached_to_any_proper_parent_of_component( )` to `spannertools.get_spanners_attached_to_any_proper_parent_of_component( )`.

spannertools.get_spanners_attached_to_component

`abjad.tools.spannertools.`**`get_spanners_attached_to_component`**`(`*component*, *klass=None*`)`

New in version 1.1.2. Get all spanners attached to *component*:

```
abjad> staff = Staff(macros.scale(4))
abjad> beam = spannertools.BeamSpanner(staff.leaves)
abjad> first_slur = spannertools.SlurSpanner(staff.leaves[:2])
abjad> second_slur = spannertools.SlurSpanner(staff.leaves[2:])
abjad> crescendo = spannertools.CrescendoSpanner(staff.leaves)

abjad> f(staff)
\new Staff {
   c'8 [ \< (
   d'8 )
   e'8 (
   f'8 ] \! )
}
```

```
abjad> spannertools.get_spanners_attached_to_component(staff.leaves[0])
set([BeamSpanner(c'8, d'8, e'8, f'8), SlurSpanner(c'8, d'8), CrescendoSpanner(c'8, d'8, e'8, f'8
```

Get spanners of *klass* attached to *component*:

```
abjad> klass = spannertools.BeamSpanner
abjad> spannertools.get_spanners_attached_to_component(staff.leaves[0], klass)
set([BeamSpanner(c'8, d'8, e'8, f'8)])
```

Get spanners of any *klass* attached to *component*:

```
abjad> klasses = (spannertools.BeamSpanner, spannertools.SlurSpanner)
abjad> spannertools.get_spanners_attached_to_component(staff.leaves[0], klasses)
set([BeamSpanner(c'8, d'8, e'8, f'8), SlurSpanner(c'8, d'8)])
```

Return unordered set of zero or more spanners. Changed in version 1.1.2: renamed `spannertools.get_all_spanners_attached_to_component( )` to `spannertools.get_spanners_attached_to_component( )`.

spannertools.get_spanners_contained_by_components

`abjad.tools.spannertools.`**`get_spanners_contained_by_components`**`(`*components*`)`

Return unordered set of spanners contained within any component in list of thread-contiguous components. Getter for t.spanners.contained across thread-contiguous components.

Changed in version 1.1.2: renamed `spannertools.get_contained( )` to `spannertools.get_spanners_contained_by_components( )`.

spannertools.get_spanners_covered_by_components

`abjad.tools.spannertools.`**`get_spanners_covered_by_components`**`(`*components*`)`

Return unordered set of spanners completely contained within the time bounds of thread-contiguous components.

Compare 'covered' spanners with 'contained' spanners. Compare 'covered' spanners with 'dominant' spanners.

Changed in version 1.1.2: renamed `spannertools.get_covered( )` to `spannertools.get_spanners_covered_by_components( )`.

spannertools.get_spanners_on_components_or_component_children

`abjad.tools.spannertools.`**`get_spanners_on_components_or_component_children`**`(`*components*`)`

Return unordered set of all spanners attaching to any component in *components* or attaching to any of the children of any of the components in *components*. Changed in version 1.1.2: renamed `spannertools.get_attached( )` to `spannertools.get_spanners_on_components_or_component_children( )`.

spannertools.get_spanners_that_cross_components

`abjad.tools.spannertools.`**`get_spanners_that_cross_components`**`(`*components*`)`

Assert thread-contiguous components. Collect spanners that attach to any component in 'components'. Return unordered set of crossing spanners. A spanner P crosses a list of thread-contiguous components C when P and

C share at least one component and when it is the case that NOT ALL of the components in P are also in C. In other words, there is some intersection – but not total intersection – between the components of P and C.

Compare 'crossing' spanners with 'covered' spanners. Compare 'crossing' spanners with 'dominant' spanners. Compare 'crossing' spanners with 'contained' spanners. Compare 'crossing' spanners with 'attached' spanners. Changed in version 1.1.2: renamed `spannertools.get_crossing( )` to `spannertools.get_spanners_that_cross_components( )`.

spannertools.get_spanners_that_dominate_component_pair

`abjad.tools.spannertools.`**`get_spanners_that_dominate_component_pair`**`(`*left*, *right*`)`

Return Python list of (spanner, index) pairs. 'left' must be either an Abjad component or None. 'right' must be either an Abjad component or None.

If both 'left' and 'right' are components, then 'left' and 'right' must be thread-contiguous.

This is a special version of spannertools.get_spanners_that_dominate_components( ). This version is useful for finding spanners that dominant a zero-length 'crack' between components, as in t[2:2]. Changed in version 1.1.2: renamed `spannertools.get_dominant_between( )` to `spannertools.get_spanners_that_dominate_component_pair( )`.

spannertools.get_spanners_that_dominate_components

`abjad.tools.spannertools.`**`get_spanners_that_dominate_components`**`(`*components*`)`

Return Python list of (spanner, index) pairs. Each (spanner, index) pair gives a spanner which dominates all components in 'components' together with the start-index at which spanner first encounters 'components'.

Use this helper to 'lift' any and all spanners temporarily from 'components', perform some action to the underlying score tree, and then reattach all spanners to new score components.

This operation always leaves all expressions in tact. Changed in version 1.1.2: renamed `spannertools.get_dominant( )` to `spannertools.get_spanners_that_dominate_components( )`.

spannertools.get_spanners_that_dominate_container_components_from_to

`abjad.tools.spannertools.`**`get_spanners_that_dominate_container_components_from_to`**`(`*container*, *start*, *stop*`)`

Return Python list of (spanner, index) pairs. Each spanner dominates the components specified by slice with start index 'start' and stop index 'stop'. Generalization of dominant spanner-finding functions for slices. This exists for slices like t[2:2] that are empty lists.

Changed in version 1.1.2: renamed `spannertools.get_dominant_slice( )` to `spannertools.get_spanners_that_dominate_container_components_from_to( )`.

spannertools.get_the_only_spanner_attached_to_any_improper_parent_of_component

`abjad.tools.spannertools.`**`get_the_only_spanner_attached_to_any_improper_parent_of_component`**

New in version 1.1.1. Get the only spanner attached to any improper parent *component*:

```
abjad> staff = Staff(macros.scale(4))
abjad> beam = spannertools.BeamSpanner(staff.leaves)
abjad> slur = spannertools.SlurSpanner(staff.leaves)
abjad> trill = spannertools.TrillSpanner(staff)
abjad> f(staff)
\new Staff {
   c'8 [ ( \startTrillSpan
   d'8
   e'8
   f'8 ] ) \stopTrillSpan
}

abjad> print spannertools.get_the_only_spanner_attached_to_component(staff)
TrillSpanner({c'8, d'8, e'8, f'8})
```

Raise missing spanner error when no spanner attached to *component*.

Raise extra spanner error when more than one spanner attached to *component*.

Return a single spanner.

Note: function will usually be called with *klass* specifier set.

spannertools.get_the_only_spanner_attached_to_component

abjad.tools.spannertools.**get_the_only_spanner_attached_to_component**(*component*, *klass=None*)

New in version 1.1.1. Get the only spanner attached to *component*:

```
abjad> staff = Staff(macros.scale(4))
abjad> beam = spannertools.BeamSpanner(staff.leaves)
abjad> slur = spannertools.SlurSpanner(staff.leaves)
abjad> trill = spannertools.TrillSpanner(staff)
abjad> f(staff)
\new Staff {
   c'8 [ ( \startTrillSpan
   d'8
   e'8
   f'8 ] ) \stopTrillSpan
}

abjad> print spannertools.get_the_only_spanner_attached_to_component(staff)
TrillSpanner({c'8, d'8, e'8, f'8})
```

Raise missing spanner error when no spanner attached to *component*.

Raise extra spanner error when more than one spanner attached to *component*.

Return a single spanner.

Note: function will usually be called with *klass* specifier set.

spannertools.is_component_with_beam_spanner_attached

abjad.tools.spannertools.**is_component_with_beam_spanner_attached**(*expr*)

New in version 1.1.2. True when *expr* is component with beam spanner attached:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> beam = spannertools.BeamSpanner(staff.leaves)

abjad> spannertools.is_component_with_beam_spanner_attached(staff[0])
True
```

Otherwise false:

```
abjad> note = Note("c'8")

abjad> spannertools.is_component_with_beam_spanner_attached(note)
False
```

Return boolean. Changed in version 1.1.2: renamed `beamtools.is_component_with_beam_spanner_attached( )` to `spannertools.is_component_with_beam_spanner_attached( )`.

spannertools.is_component_with_spanner_attached

abjad.tools.spannertools.**is_component_with_spanner_attached**(*expr*, *klass=None*)

New in version 1.1.2. True when *expr* is a component with spanner attached:

```
abjad> staff = Staff(macros.scale(4))
abjad> beam = spannertools.BeamSpanner(staff.leaves)
abjad> f(staff)
\new Staff {
   c'8 [
   d'8
   e'8
   f'8 ]
}

abjad> spannertools.is_component_with_spanner_attached(staff[0])
True
```

Otherwise false:

```
abjad> spannertools.is_component_with_spanner_attached(staff)
False
```

When *klass* is not none then true when *expr* is a component with a spanner of *klass* attached.

Return true or false.

spannertools.iterate_components_backward_in_spanner

abjad.tools.spannertools.**iterate_components_backward_in_spanner**(*spanner*, *klass=<class 'abjad.components._Component._Compon*

New in version 1.1.2. Yield components in *spanner* one at a time from left to right.

```
abjad> t = Staff(macros.scale(4))
abjad> p = spannertools.BeamSpanner(t[2:])
abjad> notes = spannertools.iterate_components_backward_in_spanner(p, klass = Note)
abjad> for note in notes:
...   note
Note("f'8")
Note("e'8")
```

Changed in version 1.1.2: renamed `spannertools.iterate_components_backward( )` to `spannertools.iterate_components_backward_in_spanner( )`.

spannertools.iterate_components_forward_in_spanner

`abjad.tools.spannertools.`**`iterate_components_forward_in_spanner`**(*spanner*, *klass=<class 'abjad.components._Component._Component'>*)

New in version 1.1.2. Yield components in *spanner* one at a time from left to right.

```
abjad> t = Staff(macros.scale(4))
abjad> p = spannertools.BeamSpanner(t[2:])
abjad> notes = spannertools.iterate_components_forward_in_spanner(p, klass = Note)
abjad> for note in notes:
...   note
Note("e'8")
Note("f'8")
```

Changed in version 1.1.2: renamed `spannertools.iterate_components_forward( )` to `spannertools.iterate_components_forward_in_spanner( )`.

spannertools.make_dynamic_spanner_below_with_nib_at_right

`abjad.tools.spannertools.`**`make_dynamic_spanner_below_with_nib_at_right`**(*dynamic_text*, *components=None*)

New in version 1.1.2. Span *components* with text spanner. Position spanner below staff and configure with *dynamic_text*, solid line and upward-pointing nib at right.

```
abjad> t = Staff(macros.scale(4))
abjad> spannertools.make_dynamic_spanner_below_with_nib_at_right('mp', t[:])
TextSpanner(c'8, d'8, e'8, f'8)
abjad> f(t)
\new Staff {
  \override TextSpanner #'bound-details #'left #'text = \markup { \dynamic { mp } }
  \override TextSpanner #'bound-details #'right #'text = #(markup #:draw-line '(0 . 1))
  \override TextSpanner #'bound-details #'right-broken #'text = ##f
  \override TextSpanner #'dash-fraction = #1
  \override TextSpanner #'direction = #down
  c'8 \startTextSpan
  d'8
  e'8
  f'8 \stopTextSpan
  \revert TextSpanner #'bound-details #'left #'text
  \revert TextSpanner #'bound-details #'right #'text
  \revert TextSpanner #'bound-details #'right-broken #'text
```

```
    \revert TextSpanner #'dash-fraction
    \revert TextSpanner #'direction
}
```

Changed in version 1.1.2: renamed `spanners.dynamic_spanner_below_with_nib_at_right( )` to `spannertools.make_dynamic_spanner_below_with_nib_at_right( )`.

spannertools.make_solid_text_spanner_above_with_nib_at_right

`abjad.tools.spannertools.`**`make_solid_text_spanner_above_with_nib_at_right`**`(`*left_text, components=None*`)`

New in version 1.1.2. Span *components* with text spanner. Position spanner above staff and configure with *left_text*, solid line and downward-pointing nib at right.

```
abjad> t = Staff(macros.scale(4))
abjad> spannertools.make_solid_text_spanner_above_with_nib_at_right('foo', t[:])
TextSpanner(c'8, d'8, e'8, f'8)
abjad> f(t)
\new Staff {
    \override TextSpanner #'bound-details #'left #'text = \markup { foo }
    \override TextSpanner #'bound-details #'right #'text = #(markup #:draw-line '(0 . -1))
    \override TextSpanner #'bound-details #'right-broken #'text = ##f
    \override TextSpanner #'dash-fraction = #1
    \override TextSpanner #'direction = #up
    c'8 \startTextSpan
    d'8
    e'8
    f'8 \stopTextSpan
    \revert TextSpanner #'bound-details #'left #'text
    \revert TextSpanner #'bound-details #'right #'text
    \revert TextSpanner #'bound-details #'right-broken #'text
    \revert TextSpanner #'dash-fraction
    \revert TextSpanner #'direction
}
```

Changed in version 1.1.2: renamed `spanners.solid_text_spanner_above_with_nib_at_right( )` to `spannertools.make_solid_text_spanner_above_with_nib_at_right( )`.

spannertools.make_solid_text_spanner_below_with_nib_at_right

`abjad.tools.spannertools.`**`make_solid_text_spanner_below_with_nib_at_right`**`(`*left_text, components=None*`)`

New in version 1.1.2. Span *components* with text spanner. Position spanner below staff and configure with *left_text*, solid line and upward-pointing nib at right.

```
abjad> t = Staff(macros.scale(4))
abjad> spannertools.make_solid_text_spanner_below_with_nib_at_right('foo', t[:])
TextSpanner(c'8, d'8, e'8, f'8)
abjad> f(t)
\new Staff {
    \override TextSpanner #'bound-details #'left #'text = \markup { foo }
    \override TextSpanner #'bound-details #'right #'text = #(markup #:draw-line '(0 . 1))
```

```
    \override TextSpanner #'bound-details #'right-broken #'text = ##f
    \override TextSpanner #'dash-fraction = #1
    \override TextSpanner #'direction = #down
    c'8 \startTextSpan
    d'8
    e'8
    f'8 \stopTextSpan
    \revert TextSpanner #'bound-details #'left #'text
    \revert TextSpanner #'bound-details #'right #'text
    \revert TextSpanner #'bound-details #'right-broken #'text
    \revert TextSpanner #'dash-fraction
    \revert TextSpanner #'direction
}
```

Changed in version 1.1.2: renamed `spanners.solid_text_spanner_below_with_nib_at_right( )` to `spannertools.make_solid_text_spanner_below_with_nib_at_right( )`.

spannertools.move_spanners_from_component_to_children_of_component

`abjad.tools.spannertools.`**`move_spanners_from_component_to_children_of_component`**`(`*donor*`)`

Give spanners attaching directly to donor to recipients. Usual use is to give attached spanners from parent to children, which is a composer-safe operation. Changed in version 1.1.2: renamed `spannertools.give_attached_to_children( )` to `spannertools.move_spanners_from_component_to_children_of_component( )`.

spannertools.report_as_string_format_contributions_of_all_spanners_attached_to_component

`abjad.tools.spannertools.`**`report_as_string_format_contributions_of_all_spanners_attached_to_`**

New in version 1.1.1. Report as string format contributions of all spanners attached to *component*:

```
abjad> staff = Staff(macros.scale(4))
abjad> beam = spannertools.BeamSpanner(staff.leaves)
abjad> slur = spannertools.SlurSpanner(staff.leaves)
abjad> trill = spannertools.TrillSpanner(staff)
abjad> f(staff)
\new Staff {
   c'8 [ ( \startTrillSpan
   d'8
   e'8
   f'8 ] ) \stopTrillSpan
}
```

```
abjad> spannertools.report_as_string_format_contributions_of_all_spanners_attached_to_component(
'BeamSpanner\n\t_right\n\t\t[\nSlurSpanner\n\t_right\n\t\t(\n'
```

Return string.

spannertools.report_as_string_format_contributions_of_all_spanners_attached_to_improper_parentage_of_comp

`abjad.tools.spannertools.`**`report_as_string_format_contributions_of_all_spanners_attached_to_`**

New in version 1.1.1. Report as string format contributions of all spanners attached to improper parentage of *component*:

```
abjad> staff = Staff(macros.scale(4))
abjad> beam = spannertools.BeamSpanner(staff.leaves)
abjad> slur = spannertools.SlurSpanner(staff.leaves)
abjad> trill = spannertools.TrillSpanner(staff)
abjad> f(staff)
\new Staff {
   c'8 [ ( \startTrillSpan
   d'8
   e'8
   f'8 ] ) \stopTrillSpan
}

abjad> spannertools.report_as_string_format_contributions_of_all_spanners_attached_to_component(
'BeamSpanner\n\t_right\n\t\t[\nSlurSpanner\n\t_right\n\t\t(\n'
```

Return string.

spannertools.report_to_screen_format_contributions_of_all_spanners_attached_to_component

abjad.tools.spannertools.**report_to_screen_format_contributions_of_all_spanners_attached_to_**

New in version 1.1.1. Report to screen format contributions of all spanners attached to *component*:

```
abjad> staff = Staff(macros.scale(4))
abjad> beam = spannertools.BeamSpanner(staff.leaves)
abjad> slur = spannertools.SlurSpanner(staff.leaves)
abjad> trill = spannertools.TrillSpanner(staff)
abjad> f(staff)
\new Staff {
   c'8 [ ( \startTrillSpan
   d'8
   e'8
   f'8 ] ) \stopTrillSpan
}

abjad> spannertools.report_to_screen_format_contributions_of_all_spanners_attached_to_component(
BeamSpanner
   _right
      [
SlurSpanner
   _right
      (
```

Return none.

spannertools.report_to_screen_format_contributions_of_all_spanners_attached_to_improper_parentage_of_comp

abjad.tools.spannertools.**report_to_screen_format_contributions_of_all_spanners_attached_to_**

New in version 1.1.1. Report to screen format contributions of all spanners attached to improper parentage of *component*:

```
abjad> staff = Staff(macros.scale(4))
abjad> beam = spannertools.BeamSpanner(staff.leaves)
abjad> slur = spannertools.SlurSpanner(staff.leaves)
abjad> trill = spannertools.TrillSpanner(staff)
abjad> f(staff)
```

```
\new Staff {
   c'8 [ ( \startTrillSpan
   d'8
   e'8
   f'8 ] ) \stopTrillSpan
}

abjad> spannertools.report_to_screen_format_contributions_of_all_spanners_attached_to_improper_p
BeamSpanner
   _right
      [
SlurSpanner
   _right
      (
TrillSpanner
   _right
      \startTrillSpan
```

Return none.

spannertools.withdraw_components_from_spanners_covered_by_components

`abjad.tools.spannertools.`**`withdraw_components_from_spanners_covered_by_components`**`(`*components*`)`

Find every spanner covered by 'components'. Withdraw all components in 'components' from covered spanners. Return 'components'. The operation always leaves all score trees in tact.

Changed in version 1.1.2: renamed `spannertools.withdraw_from_covered( )` to `spannertools.withdraw_components_from_spanners_covered_by_components( )`.

stafftools

stafftools.RhythmicStaff

class `abjad.tools.stafftools.`**`RhythmicStaff`**`(`*music*=[ ], ***kwargs*`)`

Bases: `abjad.components.Staff.Staff.Staff`

Abjad model of a rhythmic staff.

stafftools.get_first_staff_in_improper_parentage_of_component

`abjad.tools.stafftools.`**`get_first_staff_in_improper_parentage_of_component`**`(`*component*`)`

New in version 1.1.2. Get first staff in improper parentage of *component*:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")

abjad> f(staff)
\new Staff {
   c'8
   d'8
   e'8
   f'8
}

abjad> stafftools.get_first_staff_in_improper_parentage_of_component(staff[1])
Staff{4}
```

Return staff or none.

stafftools.get_first_staff_in_proper_parentage_of_component

abjad.tools.stafftools.**get_first_staff_in_proper_parentage_of_component**(*component*)
New in version 1.1.2. Get first staff in proper parentage of *component*:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")

abjad> f(staff)
\new Staff {
   c'8
   d'8
   e'8
   f'8
}

abjad> stafftools.get_first_staff_in_proper_parentage_of_component(staff[1])
Staff{4}
```

Return staff or none.

stafftools.iterate_staves_backward_in_expr

abjad.tools.stafftools.**iterate_staves_backward_in_expr**(*expr*, *start=0*, *stop=None*)
New in version 1.1.2. Iterate staves backward in *expr*:

```
abjad> score = Score(4 * Staff([ ]))

abjad> f(score)
\new Score <<
   \new Staff {
   }
   \new Staff {
   }
   \new Staff {
   }
   \new Staff {
   }
>>

abjad> for staff in stafftools.iterate_staves_backward_in_expr(score):
...     staff
...
Staff{ }
Staff{ }
Staff{ }
Staff{ }
```

Return generator.

stafftools.iterate_staves_forward_in_expr

abjad.tools.stafftools.**iterate_staves_forward_in_expr**(*expr*, *start=0*, *stop=None*)
New in version 1.1.2. Iterate staves forward in *expr*:

```
abjad> score = Score(4 * Staff([ ]))

abjad> f(score)
\new Score <<
   \new Staff {
   }
   \new Staff {
   }
   \new Staff {
   }
   \new Staff {
   }
>>

abjad> for staff in stafftools.iterate_staves_forward_in_expr(score):
...     staff
...
Staff{ }
Staff{ }
Staff{ }
Staff{ }
```

Return generator.

stafftools.make_invisible_staff

`abjad.tools.stafftools.`**`make_invisible_staff`**`(`*music*`)`

Staff constructor that hides meter, bar line and staff lines. Changed in version 1.1.2: Invisible staff class changed to invisible staff function.

stafftools.make_rhythmic_sketch_staff

`abjad.tools.stafftools.`**`make_rhythmic_sketch_staff`**`(`*music*`)`

Make rhythmic staff with transparent meter and transparent bar lines.

tempotools

tempotools.integer_tempo_to_multiplier_tempo_pairs

`abjad.tools.tempotools.`**`integer_tempo_to_multiplier_tempo_pairs`**`(`*integer_tempo, maximum_numerator=None, maximum_denominator=None*`)`

New in version 1.1.2. Return all multiplier, tempo pairs possible from *integer_tempo*.

Tempi must be no less than `integer_tempo / 2` and not greater than `2 * integer_tempo`.

```
abjad> pairs = tempotools.integer_tempo_to_multiplier_tempo_pairs(58, 8, 8)
abjad> for pair in pairs:
...     pair
...
(Fraction(1, 2), Fraction(29, 1))
(Fraction(1, 1), Fraction(58, 1))
(Fraction(3, 2), Fraction(87, 1))
(Fraction(2, 1), Fraction(116, 1))
```

Return list.

tempotools.integer_tempo_to_multiplier_tempo_pairs_report

abjad.tools.tempotools.**integer_tempo_to_multiplier_tempo_pairs_report**(*integer_tempo*, *maximum_numerator=None*, *maximum_denominator=None*)

New in version 1.1.2. Print all multiplier, tempo pairs possible from *integer_tempo*.

Allow no tempi less than `integer_tempo / 2` nor greater than `2 * integer_tempo`.

```
abjad> tempotools.integer_tempo_to_multiplier_tempo_pairs_report(58, 8, 8)
2:1     29
1:1     58
2:3     87
1:2     116
```

With more lenient numerator and denominator.

```
abjad> tempotools.integer_tempo_to_multiplier_tempo_pairs_report(58, 30, 30)
2:1     29
29:15   30
29:16   32
29:17   34
29:18   36
29:19   38
29:20   40
29:21   42
29:22   44
29:23   46
29:24   48
29:25   50
29:26   52
29:27   54
29:28   56
1:1     58
29:30   60
2:3     87
1:2     116
```

Return none.

threadtools

threadtools.component_to_thread_signature

abjad.tools.threadtools.**component_to_thread_signature**(*component*)

Return _ContainmentSignature giving the root and first voice, staff and score in parentage of component.

threadtools.iterate_thread_backward_from_component

abjad.tools.threadtools.**iterate_thread_backward_from_component**(*component*, *klass=None*)

New in version 1.1.2. Yield right-to-left components in the thread of *component* starting from *component*.

When `klass = None` return all components in the thread of *component*.

When *klass* is set to some other Abjad class, yield only *klass* instances in the thread of *component*.

```
abjad> container = Container(Voice(notetools.make_repeated_notes(2)) * 2)
abjad> container.is_parallel = True
abjad> container[0].name = 'voice 1'
abjad> container[1].name = 'voice 2'
abjad> staff = Staff(container * 2)
abjad> macros.diatonicize(staff)
abjad> print staff.format
\new Staff {
        <<
                \context Voice = "voice 1" {
                        c'8
                        d'8
                }
                \context Voice = "voice 2" {
                        e'8
                        f'8
                }
        >>
        <<
                \context Voice = "voice 1" {
                        g'8
                        a'8
                }
                \context Voice = "voice 2" {
                        b'8
                        c''8
                }
        >>
}
```

Starting from the last leaf in score.

```
abjad> for x in threadtools.iterate_thread_backward_from_component(staff.leaves[-1], Note):
...     x
Note("c''8")
Note("b'8")
Note("f'8")
Note("e'8")
```

Yield all components in thread:

```
abjad> for x in threadtools.iterate_thread_backward_from_component(staff.leaves[-1]):
...     x
Note("c''8")
Voice-"voice 2"{2}
Note("b'8")
Voice-"voice 2"{2}
Note("f'8")
Note("e'8")
```

Note that this function is a special type of depth-first search.

Compare with `threadtools.iterate_thread_backward_in_expr()`. Changed in version 1.1.2: renamed `iterate.thread_backward_from()` to `threadtools.iterate_thread_backward_from_component()`.Changed in version 1.1.2: renamed `iterate.thread_backward_from_component()` to `threadtools.iterate_thread_backward_from_component( )`.

threadtools.iterate_thread_backward_in_expr

`abjad.tools.threadtools.`**`iterate_thread_backward_in_expr`**`(`*expr*, *klass*, *thread_signature*`)`

New in version 1.1.2. Yield right-to-left instances of *klass* in *expr* with *thread_signature*:

```
abjad> container = Container(Voice(notetools.make_repeated_notes(2)) * 2)
abjad> container.is_parallel = True
abjad> container[0].name = 'voice 1'
abjad> container[1].name = 'vocie 2'
abjad> staff = Staff(container * 2)
abjad> macros.diatonicize(staff)
abjad> f(staff)
\new Staff {
        <<
                \context Voice = "voice 1" {
                        c'8
                        d'8
                }
                \context Voice = "vocie 2" {
                        e'8
                        f'8
                }
        >>
        <<
                \context Voice = "voice 1" {
                        g'8
                        a'8
                }
                \context Voice = "vocie 2" {
                        b'8
                        c''8
                }
        >>
}
```

```
abjad> signature = threadtools.component_to_thread_signature(staff[0])
abjad> for x in threadtools.iterate_thread_backward_in_expr(staff, Note, signature): # doctest:
...     x
Note("c''8")
Note("b'8")
Note("f'8")
Note("e'8")
```

The important thing to note is that the function yields only those leaves that sit in the same thread.

Compare with `componenttools.iterate_components_backward_in_expr( )`. Changed in version 1.1.2: renamed `iterate.thread_backward_in( )` to `threadtools.iterate_thread_backward_in_expr( )`.

threadtools.iterate_thread_forward_from_component

`abjad.tools.threadtools.`**`iterate_thread_forward_from_component`**`(`*component*, *klass=None*`)`

New in version 1.1.1. Yield left-to-right components in the thread of *component* starting from *component*.

When `klass = None` return all components in the thread of *component*.

When *klass* is set to some other Abjad class, yield only *klass* instances in the thread of *component*.

```
abjad> container = Container(Voice(notetools.make_repeated_notes(2)) * 2)
abjad> container.is_parallel = True
abjad> container[0].name = 'voice 1'
abjad> container[1].name = 'voice 2'
abjad> staff = Staff(container * 2)
abjad> macros.diatonicize(staff)
abjad> print staff.format
\new Staff {
        <<
                \context Voice = "voice 1" {
                        c'8
                        d'8
                }
                \context Voice = "voice 2" {
                        e'8
                        f'8
                }
        >>
        <<
                \context Voice = "voice 1" {
                        g'8
                        a'8
                }
                \context Voice = "voice 2" {
                        b'8
                        c''8
                }
        >>
}
```

Starting from the first leaf in score.

```
abjad> for x in threadtools.iterate_thread_forward_from_component(staff.leaves[0], Note):
...     x
...
Note("c'8")
Note("d'8")
Note("g'8")
Note("a'8")
```

Starting from the second leaf in score.

```
abjad> for x in threadtools.iterate_thread_forward_from_component(staff.leaves[1], Note):
...     x
...
Note("d'8")
Note("g'8")
Note("a'8")
```

Yield all components in thread.

```
abjad> for x in threadtools.iterate_thread_forward_from_component(staff.leaves[0]):
...     x
...
Note("c'8")
Voice-"voice 1"{2}
```

```
Note("d'8")
Voice-"voice 1"{2}
Note("g'8")
Note("a'8")
```

Note that this function is a special type of depth-first search.

Compare with `threadtools.iterate_thread_forward_in_expr( )`. Changed in version 1.1.2: renamed `iterate.thread_forward_from( )` to `threadtools.iterate_thread_forward_from_component( )`.Changed in version 1.1.2: renamed `iterate.thread_forward_from_component( )` to `threadtools.iterate_thread_forward_from_component( )`.

threadtools.iterate_thread_forward_in_expr

`abjad.tools.threadtools.`**`iterate_thread_forward_in_expr`**(*expr*, *klass*, *thread_signature*)

New in version 1.1.1. Yield left-to-right instances of *klass* in *expr* with *thread_signature*.

```
abjad> container = Container(Voice(notetools.make_repeated_notes(2)) * 2)
abjad> container.is_parallel = True
abjad> container[0].name = 'voice 1'
abjad> container[1].name = 'vocie 2'
abjad> staff = Staff(container * 2)
abjad> macros.diatonicize(staff)
abjad> print staff.format
\new Staff {
        <<
                \context Voice = "voice 1" {
                        c'8
                        d'8
                }
                \context Voice = "vocie 2" {
                        e'8
                        f'8
                }
        >>
        <<
                \context Voice = "voice 1" {
                        g'8
                        a'8
                }
                \context Voice = "vocie 2" {
                        b'8
                        c''8
                }
        >>
}
```

```
abjad> signature = threadtools.component_to_thread_signature(staff.leaves[0])
abjad> for x in threadtools.iterate_thread_forward_in_expr(staff, Note, signature):
...     x
...
Note("c'8")
Note("d'8")
Note("g'8")
Note("a'8")
```

The important thing to note is that the function yields only those leaves that sit in the same thread.

Compare with `componenttools.iterate_components_forward_in_expr( )`. Changed in version 1.1.2: renamed `iterate.thread_forward_in( )` to `threadtools.iterate_thread_forward_in_expr( )`.Changed in version 1.1.2: renamed `iterate.thread_forward_in_expr( )` to `threadtools.iterate_thread_forward_in_expr( )`.

tietools

tietools.TieSpanner

class `abjad.tools.tietools.`**`TieSpanner`**`(`*music=None*`)`
Bases: `abjad.tools.spannertools.Spanner.Spanner.Spanner`

Abjad tie spanner:

```
abjad> staff = Staff(notetools.make_repeated_notes(4))
abjad> tietools.TieSpanner(staff[:])
TieSpanner(c'8, c'8, c'8, c'8)
abjad> f(staff)
\new Staff {
   c'8 ~
   c'8 ~
   c'8 ~
   c'8
}
```

Return tie spanner.

tietools.add_or_remove_tie_chain_notes_to_achieve_scaled_written_duration

`abjad.tools.tietools.`**`add_or_remove_tie_chain_notes_to_achieve_scaled_written_duration`**`(`*tie_cha mul- ti- plier*`)`

Scale tie chain by multiplier. Wraps tie_chain_duration_change. Returns tie chain.

Changed in version 1.1.2: renamed `tietools.duration_scale( )` to `tietools.add_or_remove_tie_chain_notes_to_achieve_scaled_written_duration( )`.

tietools.add_or_remove_tie_chain_notes_to_achieve_written_duration

`abjad.tools.tietools.`**`add_or_remove_tie_chain_notes_to_achieve_written_duration`**`(`*tie_chain, new_written_dur*

Change the written duration of tie chain, adding and subtracting notes as necessary.

Return newly modified tie chain. Changed in version 1.1.2: renamed `tietools.duration_change( )` to `tietools.add_or_remove_tie_chain_notes_to_achieve_written_duration( )`.

tietools.apply_tie_spanner_to_leaf_pair

`abjad.tools.tietools.`**`apply_tie_spanner_to_leaf_pair`**`(`*left*, *right*`)`
Apply tie spanner to *left* leaf and *right* leaf:

```
abjad> staff = Staff(notetools.make_repeated_notes(4))
abjad> tietools.TieSpanner(staff[:2])
TieSpanner(c'8, c'8)
abjad> f(staff)
\new Staff {
   c'8 ~
   c'8
   c'8
   c'8
}
```

```
abjad> tietools.apply_tie_spanner_to_leaf_pair(staff[1], staff[2])
```

```
abjad> f(staff)
\new Staff {
   c'8 ~
   c'8 ~
   c'8
   c'8
}
```

Handle existing tie spanners intelligently.

Return none. Changed in version 1.1.2: renamed `tietools.span_leaf_pair( )` to `tietools.apply_tie_spanner_to_leaf_pair( )`.

tietools.are_components_in_same_tie_spanner

`abjad.tools.tietools.`**`are_components_in_same_tie_spanner`**`(`*components*`)`

True if all components in list share same tie spanner, otherwise False.

Changed in version 1.1.2: renamed `tietools.are_in_same_spanner( )` to `tietools.are_components_in_same_tie_spanner( )`.

tietools.get_leaves_in_tie_chain

`abjad.tools.tietools.`**`get_leaves_in_tie_chain`**`(`*tie_chain*`)`

Return Python list of leaves in tie chain.

tietools.get_preprolated_tie_chain_duration

`abjad.tools.tietools.`**`get_preprolated_tie_chain_duration`**`(`*tie_chain*`)`

Get sum of preprolated duration of all leaves in *tie_chain*.

Todo

write `tietools.get_preprolated_tie_chain_duration( )` tests.

Changed in version 1.1.2: renamed `tietools.get_duration_preprolated( )` to `tietools.get_preprolated_tie_chain_duration( )`.

tietools.get_prolated_tie_chain_duration

abjad.tools.tietools.**get_prolated_tie_chain_duration**(*tie_chain*)

Return sum of prolated duration of all leaves in chain.

Todo

Write tietools.get_prolated_tie_chain_duration( ) tests.

Changed in version 1.1.2: renamed `tietools.get_duration_prolated( )` to `tietools.get_prolated_tie_chain_duration( )`.Changed in version 1.1.2: renamed `tietools.get_tie_chain_prolated_duration( )` to `tietools.get_prolated_tie_chain_duration( )`.

tietools.get_tie_chain

abjad.tools.tietools.**get_tie_chain**(*component*)

New in version 1.1.2. Get tie chain from *component*.

tietools.get_tie_chain_duration_in_seconds

abjad.tools.tietools.**get_tie_chain_duration_in_seconds**(*tie_chain*)

Return sum of seconds duration of all leaves in chain.

Todo

Write tietools.get_tie_chain_duration_in_seconds( ) tests.

Changed in version 1.1.2: renamed `tietools.get_duration_seconds( )` to `tietools.get_tie_chain_duration_in_seconds( )`.

tietools.get_tie_chains_in_expr

abjad.tools.tietools.**get_tie_chains_in_expr**(*components*)

This function returns all tie chains in components. A tie chain may not encompass all the leaves spanned by its corresponding Tie spanner, but only those found in the given list. i.e. the function returns the intersection between all the leav es spanned by all tie spanners touching the components given and the leaves found in the given components list. Changed in version 1.1.2: renamed `tietools.get_tie_chains( )` to `tietools.get_tie_chains_in_expr( )`.

tietools.get_written_tie_chain_duration

abjad.tools.tietools.**get_written_tie_chain_duration**(*tie_chain*)

Return sum of written duration of all leaves in chain.

tietools.group_leaves_in_tie_chain_by_immediate_parents

abjad.tools.tietools.**group_leaves_in_tie_chain_by_immediate_parents**(*tie_chain*)

Group leaves in *tie_chain* by immediate parent:

```
abjad> staff = Staff(Measure((2, 8), notetools.make_repeated_notes(2)) * 2)
abjad> tietools.TieSpanner(staff.leaves)
TieSpanner(c'8, c'8, c'8, c'8)
abjad> f(staff)
\new Staff {
   {
      \time 2/8
      c'8 ~
      c'8 ~
   }
   {
      \time 2/8
      c'8 ~
      c'8
   }
}
```

```
abjad> tie_chain = tietools.get_tie_chain(staff.leaves[0])
abjad> tietools.group_leaves_in_tie_chain_by_immediate_parents(tie_chain)
[[Note("c'8"), Note("c'8")], [Note("c'8"), Note("c'8")]]
```

Return list of leaf group lists. Changed in version 1.1.2: renamed `tietools.group_by_parent( )` to `tietools.group_leaves_in_tie_chain_by_immediate_parents( )`.

tietools.is_component_with_tie_spanner_attached

`abjad.tools.tietools.`**`is_component_with_tie_spanner_attached`**`(`*expr*`)`

New in version 1.1.2. True when *expr* is component with tie spanner attached:

```
abjad> staff = Staff(notetools.make_repeated_notes(4))
abjad> tietools.TieSpanner(staff[:])
TieSpanner(c'8, c'8, c'8, c'8)
abjad> f(staff)
\new Staff {
   c'8 ~
   c'8 ~
   c'8 ~
   c'8
}
abjad> tietools.is_component_with_tie_spanner_attached(staff)
False
```

Otherwise false:

```
abjad> staff = Staff(notetools.make_repeated_notes(4))
abjad> tietools.TieSpanner(staff[:])
TieSpanner(c'8, c'8, c'8, c'8)
abjad> f(staff)
\new Staff {
   c'8 ~
   c'8 ~
   c'8 ~
   c'8
}
abjad> tietools.is_component_with_tie_spanner_attached(staff[1])
True
```

Return boolean.

tietools.is_tie_chain

`abjad.tools.tietools.is_tie_chain(expr)`

True when expr is a tie chain, otherwise False.

tietools.is_tie_chain_with_all_leaves_in_same_parent

`abjad.tools.tietools.is_tie_chain_with_all_leaves_in_same_parent(expr)`

True when expr is a tie chain with all leaves in same parent.

That is, True when tie chain crosses no container boundaries, otherwise False.

Example:

```
abjad> t = Staff(Measure((2, 8), notetools.make_repeated_notes(2)) * 2)
abjad> tietools.TieSpanner(t.leaves[1:3])
TieSpanner(c'8, c'8)

\new Staff {
      \time 2/8
      c'8
      c'8 ~
      \time 2/8
      c'8
      c'8
}

abjad> tie_chain = tietools.get_tie_chain(t.leaves[0])
abjad> assert tietools.is_tie_chain_with_all_leaves_in_same_parent(tie_chain)
abjad> tie_chain = tietools.get_tie_chain(t.leaves[1])
abjad> assert not tietools.is_tie_chain_with_all_leaves_in_same_parent(tie_chain)
abjad> tie_chain = tietools.get_tie_chain(t.leaves[2])
abjad> assert not tietools.is_tie_chain_with_all_leaves_in_same_parent(tie_chain)
abjad> tie_chain = tietools.get_tie_chain(t.leaves[3])
abjad> assert tietools.is_tie_chain_with_all_leaves_in_same_parent(tie_chain)
```

Changed in version 1.1.2: renamed `tietools.is_in_same_parent( )` to `tietools.is_tie_chain_with_all_leaves_in_same_parent( )`.

tietools.iterate_tie_chains_backward_in_expr

`abjad.tools.tietools.iterate_tie_chains_backward_in_expr(expr)`

Yield right-to-left tie chains in *expr*:

```
abjad> notes = notetools.make_notes([0], [(5, 16), (1, 8), (1, 8), (5, 16)])
abjad> staff = Staff(notes)
abjad> tuplet = tuplettools.FixedDurationTuplet((2, 16), staff[1:3])
abjad> macros.diatonicize(staff)
abjad> print staff.format
\new Staff {
        c'4 ~
        \times 2/3 {
                c'16
                d'8
        }
        e'8
        f'4 ~
```

```
        f'16
}

abjad> for x in tietools.iterate_tie_chains_backward_in_expr(staff):
...     x
...
(Note("f'4"), Note("f'16"))
(Note("e'8"),)
(Note("d'8"),)
(Note("c'4"), Note("c'16"))
```

Note that one-note tie chains yield the same as other tie chains.

Note also that nested structures are no problem. Changed in version 1.1.2: renamed `iterate.tie_chains_backward_in()` to `tietools.iterate_tie_chains_backward_in_expr()`.Changed in version 1.1.2: renamed `iterate.tie_chains_backward_in_expr()` to `tietools.iterate_tie_chains_backward_in_expr()`.

tietools.iterate_tie_chains_forward_in_expr

abjad.tools.tietools.**iterate_tie_chains_forward_in_expr**(*expr*)

Yield left-to-right tie chains in *expr*:

```
abjad> notes = notetools.make_notes([0], [(5, 16), (1, 8), (1, 8), (5, 16)])
abjad> staff = Staff(notes)
abjad> tuplet = tuplettools.FixedDurationTuplet((2, 16), staff[1:3])
abjad> macros.diatonicize(staff)
abjad> print staff.format
\new Staff {
        c'4 ~
        \times 2/3 {
                c'16
                d'8
        }
        e'8
        f'4 ~
        f'16
}

abjad> for x in tietools.iterate_tie_chains_forward_in_expr(staff):
...     x
...
(Note("c'4"), Note("c'16"))
(Note("d'8"),)
(Note("e'8"),)
(Note("f'4"), Note("f'16"))
```

Note that one-note tie chains yield the same as other tie chains.

Note also that nested structures are no problem. Changed in version 1.1.2: renamed `iterate.tie_chains_forward_in()` to `tietools.iterate_tie_chains_forward_in_expr()`.Changed in version 1.1.2: renamed `iterate.tie_chains_forward_in_expr()` to `tietools.iterate_tie_chains_forward_in_expr()`.

tietools.iterate_topmost_tie_chains_and_components_forward_in_expr

abjad.tools.tietools.**iterate_topmost_tie_chains_and_components_forward_in_expr**(*expr*)

Yield the left-to-right, top-level contents of *expr* with chain-wrapped leaves.

```
abjad> t = Staff(notetools.make_notes(0, [(5, 32)] * 4))
abjad> t.insert(4, tuplettools.FixedDurationTuplet((2, 8), notetools.make_repeated_notes(3)))
abjad> macros.diatonicize(t)
abjad> f(t)
\new Staff {
  c'8 ~
  c'32
  d'8 ~
  d'32
  \times 2/3 {
     e'8
     f'8
     g'8
  }
  a'8 ~
  a'32
  b'8 ~
  b'32
}

abjad> for x in tietools.iterate_topmost_tie_chains_and_components_forward_in_expr(t):
...     x
...
(Note("c'8"), Note("c'32"))
(Note("d'8"), Note("d'32"))
FixedDurationTuplet(1/4, [e'8, f'8, g'8])
(Note("a'8"), Note("a'32"))
(Note("b'8"), Note("b'32"))
```

Crossing ties raise `TieChainError`. Changed in version 1.1.2: renamed `iterate.chained_contents( )` to `tietools.iterate_topmost_tie_chains_and_components_forwar` `)`.Changed in version 1.1.2: renamed `iterate.topmost_tie_chains_and_components_forward_in_expr( )` to `tietools.iterate_topmost_tie_chains_and_components_forward_in_expr( )`.

tietools.label_tie_chains_in_expr_with_prolated_tie_chain_duration

abjad.tools.tietools.**label_tie_chains_in_expr_with_prolated_tie_chain_duration**(*expr*, *markup_directio*

Label tie chains in *expr* with prolated tie chain duration:

```
abjad> staff = Staff(notetools.make_repeated_notes(4))
abjad> tuplettools.FixedDurationTuplet((2, 8), staff[:3])
FixedDurationTuplet(1/4, [c'8, c'8, c'8])
abjad> tietools.TieSpanner(staff.leaves[:2])
TieSpanner(c'8, c'8)
abjad> tietools.TieSpanner(staff.leaves[2:])
TieSpanner(c'8, c'8)
abjad> tietools.label_tie_chains_in_expr_with_prolated_tie_chain_duration(staff)
abjad> f(staff)
\new Staff {
   \times 2/3 {
      c'8 _ \markup { \small 1/6 } ~
```

```
         c'8
         c'8 _ \markup { \small 5/24 } ~
      }
      c'8
   }
```

Return none.

tietools.label_tie_chains_in_expr_with_tie_chain_durations

abjad.tools.tietools.**label_tie_chains_in_expr_with_tie_chain_durations**(*expr*, *markup_direction='down'*)

Label tie chains in *expr* with both written tie chain duration and prolated tie chain duration:

```
abjad> staff = Staff(notetools.make_repeated_notes(4))
abjad> tuplettools.FixedDurationTuplet((2, 8), staff[:3])
FixedDurationTuplet(1/4, [c'8, c'8, c'8])
abjad> tietools.TieSpanner(staff.leaves[:2])
TieSpanner(c'8, c'8)
abjad> tietools.TieSpanner(staff.leaves[2:])
TieSpanner(c'8, c'8)
abjad> tietools.label_tie_chains_in_expr_with_tie_chain_durations(staff)
abjad> f(staff)
\new Staff {
   \times 2/3 {
      c'8 _ \markup { \column { \small 1/4 \small 1/6 } } ~
      c'8
      c'8 _ \markup { \column { \small 1/4 \small 5/24 } } ~
   }
   c'8
}
```

Return none.

tietools.label_tie_chains_in_expr_with_written_tie_chain_duration

abjad.tools.tietools.**label_tie_chains_in_expr_with_written_tie_chain_duration**(*expr*, *markup_direction=*

Label tie chains in *expr* with written tie chain duration.:

```
abjad> staff = Staff(notetools.make_repeated_notes(4))
abjad> tuplettools.FixedDurationTuplet((2, 8), staff[:3])
FixedDurationTuplet(1/4, [c'8, c'8, c'8])
abjad> tietools.TieSpanner(staff.leaves[:2])
TieSpanner(c'8, c'8)
abjad> tietools.TieSpanner(staff.leaves[2:])
TieSpanner(c'8, c'8)
abjad> tietools.label_tie_chains_in_expr_with_written_tie_chain_duration(staff)
abjad> f(staff)
\new Staff {
   \times 2/3 {
      c'8 _ \markup { \small 1/4 } ~
      c'8
      c'8 _ \markup { \small 1/4 } ~
   }
   c'8
}
```

Return none.

tietools.remove_all_leaves_in_tie_chain_except_first

`abjad.tools.tietools.`**`remove_all_leaves_in_tie_chain_except_first`**(*tie_chain*)

Detach all leaves of tie chain after the first. Unspan and return length-1 tie chain. Changed in version 1.1.2: renamed `tietools.truncate( )` to `tietools.remove_all_leaves_in_tie_chain_except_first( )`.

tietools.remove_tie_spanners_from_components_in_expr

`abjad.tools.tietools.`**`remove_tie_spanners_from_components_in_expr`**(*expr*)

Remove tie spanners components in *expr*:

```
abjad> staff = Staff(macros.scale(2, (5, 16)))
abjad> f(staff)
\new Staff {
   c'4 ~
   c'16
   d'4 ~
   d'16
}
```

```
abjad> tietools.remove_tie_spanners_from_components_in_expr(staff[:])
[Note("c'4"), Note("c'16"), Note("d'4"), Note("d'16")]
abjad> f(staff)
\new Staff {
   c'4
   c'16
   d'4
   d'16
}
```

Return *expr*. Changed in version 1.1.2: renamed `componenttools.untie_shallow( )` to `tietools.remove_tie_spanners_from_components_in_expr( )`.

tietools.tie_chain_to_augmented_tuplet_with_proportions_and_avoid_dots

`abjad.tools.tietools.`**`tie_chain_to_augmented_tuplet_with_proportions_and_avoid_dots`**(*tie_chain*, *proportions*)

New in version 1.1.2. Divide *tie_chain* into fixed-duration tuplet according to arbitrary integer *proportions*.

Interpret *proportions* as a ratio. That is, reduce integers in *proportions* relative to each other.

Return non-trivial tuplet as augmentation.

Where `proportions[i] == 1` for `i < len(proportions)`, do not allow tupletted notes to carry dots.

```
abjad> staff = Staff([Note(0, (1, 8)), Note(0, (1, 16)), Note(0, (1, 16))])
abjad> tietools.TieSpanner(staff[:2])
TieSpanner(c'8, c'16)
abjad> spannertools.BeamSpanner(staff[:])
BeamSpanner(c'8, c'16, c'16)
abjad> tie_chain = tietools.get_tie_chain(staff[0])
```

```
abjad> tietools.tie_chain_to_augmented_tuplet_with_proportions_and_avoid_dots(tie_chain, [1])
FixedDurationTuplet(3/16, [c'8])
abjad> f(staff)
\new Staff {
        \fraction \times 3/2 {
                c'8 [
        }
        c'16 ]
}

abjad> staff = Staff([Note(0, (1, 8)), Note(0, (1, 16)), Note(0, (1, 16))])
abjad> tietools.TieSpanner(staff[:2])
TieSpanner(c'8, c'16)
abjad> spannertools.BeamSpanner(staff[:])
BeamSpanner(c'8, c'16, c'16)
abjad> tie_chain = tietools.get_tie_chain(staff[0])
abjad> tietools.tie_chain_to_augmented_tuplet_with_proportions_and_avoid_dots(tie_chain, [1, 2])
FixedDurationTuplet(3/16, [c'16, c'8])
abjad> f(staff)
\new Staff {
        {
                c'16 [
                c'8
        }
        c'16 ]
}

abjad> staff = Staff([Note(0, (1, 8)), Note(0, (1, 16)), Note(0, (1, 16))])
abjad> tietools.TieSpanner(staff[:2])
TieSpanner(c'8, c'16)
abjad> spannertools.BeamSpanner(staff[:])
BeamSpanner(c'8, c'16, c'16)
abjad> tie_chain = tietools.get_tie_chain(staff[0])
abjad> tietools.tie_chain_to_augmented_tuplet_with_proportions_and_avoid_dots(tie_chain, [1, 2,
FixedDurationTuplet(3/16, [c'32, c'16, c'16])
abjad> f(staff)
\new Staff {
        \fraction \times 6/5 {
                c'32 [
                c'16
                c'16
        }
        c'16 ]
}
```

Changed in version 1.1.2: renamed `divide.tie_chain_into_arbitrary_augmentation_undotted( )` to `tietools.tie_chain_to_augmented_tuplet_with_proportions_and_avoid_dots( )`.

tietools.tie_chain_to_augmented_tuplet_with_proportions_and_encourage_dots

`abjad.tools.tietools.`**`tie_chain_to_augmented_tuplet_with_proportions_and_encourage_dots`**(*tie_c pro- por- tions*)

New in version 1.1.2. Divide *tie_chain* into fixed-duration tuplet according to arbitrary integer *proportions*.

Interpret *proportions* as a ratio. That is, reduce integers in *proportions* relative to each other.

Return non-trivial tuplet as augmentation.

Where `proportions[i] == 1` for `i < len(proportions)`, allow tupletted notes to carry dots.

```
abjad> staff = Staff([Note(0, (1, 8)), Note(0, (1, 16)), Note(0, (1, 16))])
abjad> tietools.TieSpanner(staff[:2])
TieSpanner(c'8, c'16)
abjad> spannertools.BeamSpanner(staff[:])
BeamSpanner(c'8, c'16, c'16)
abjad> tie_chain = tietools.get_tie_chain(staff[0])
abjad> tietools.tie_chain_to_augmented_tuplet_with_proportions_and_encourage_dots(tie_chain, [1]
FixedDurationTuplet(3/16, [c'8.])
abjad> f(staff)
\new Staff {
        {
                c'8. [
        }
        c'16 ]
}

abjad> staff = Staff([Note(0, (1, 8)), Note(0, (1, 16)), Note(0, (1, 16))])
abjad> tietools.TieSpanner(staff[:2])
TieSpanner(c'8, c'16)
abjad> spannertools.BeamSpanner(staff[:])
BeamSpanner(c'8, c'16, c'16)
abjad> tie_chain = tietools.get_tie_chain(staff[0])
abjad> tietools.tie_chain_to_augmented_tuplet_with_proportions_and_encourage_dots(tie_chain, [1,
FixedDurationTuplet(3/16, [c'16, c'8])
abjad> f(staff)
\new Staff {
        {
                c'16 [
                c'8
        }
        c'16 ]
}

abjad> staff = Staff([Note(0, (1, 8)), Note(0, (1, 16)), Note(0, (1, 16))])
abjad> tietools.TieSpanner(staff[:2])
TieSpanner(c'8, c'16)
abjad> spannertools.BeamSpanner(staff[:])
BeamSpanner(c'8, c'16, c'16)
abjad> tie_chain = tietools.get_tie_chain(staff[0])
abjad> tietools.tie_chain_to_augmented_tuplet_with_proportions_and_encourage_dots(tie_chain, [1,
FixedDurationTuplet(3/16, [c'64., c'32., c'32.])
abjad> f(staff)
\new Staff {
        \fraction \times 8/5 {
                c'64. [
                c'32.
                c'32.
        }
        c'16 ]
}
```

Changed in version 1.1.2: renamed `divide.tie_chain_into_arbitrary_augmentation_dotted( )` to `tietools.tie_chain_to_augmented_tuplet_with_proportions_and_encourage_dots( )`.

tietools.tie_chain_to_diminished_tuplet_with_proportions_and_avoid_dots

abjad.tools.tietools.**tie_chain_to_diminished_tuplet_with_proportions_and_avoid_dots**(*tie_chain*, *proportions*)

New in version 1.1.2. Divide *tie_chain* into fixed-duration tuplet according to arbitrary integer *proportions*.

Interpret *proportions* as a ratio. That is, reduce integers in *proportions* relative to each other.

Return non-trivial tuplet as diminution.

Where `proportions[i] == 1` for `i < len(proportions)`, do not allow tupletted notes to carry dots.

```
abjad> staff = Staff([Note(0, (1, 8)), Note(0, (1, 16)), Note(0, (1, 16))])
abjad> tietools.TieSpanner(staff[:2])
TieSpanner(c'8, c'16)
abjad> spannertools.BeamSpanner(staff[:])
BeamSpanner(c'8, c'16, c'16)
abjad> tie_chain = tietools.get_tie_chain(staff[0])
abjad> tietools.tie_chain_to_diminished_tuplet_with_proportions_and_avoid_dots(tie_chain, [1])
FixedDurationTuplet(3/16, [c'4])
abjad> f(staff)
\new Staff {
        \fraction \times 3/4 {
                c'4 [
        }
        c'16 ]
}

abjad> staff = Staff([Note(0, (1, 8)), Note(0, (1, 16)), Note(0, (1, 16))])
abjad> tietools.TieSpanner(staff[:2])
TieSpanner(c'8, c'16)
abjad> spannertools.BeamSpanner(staff[:])
BeamSpanner(c'8, c'16, c'16)
abjad> tie_chain = tietools.get_tie_chain(staff[0])
abjad> tietools.tie_chain_to_augmented_tuplet_with_proportions_and_avoid_dots(tie_chain, [1, 2])
FixedDurationTuplet(3/16, [c'16, c'8])
abjad> f(staff)
\new Staff {
        {
                c'16 [
                c'8
        }
        c'16 ]
}

abjad> staff = Staff([Note(0, (1, 8)), Note(0, (1, 16)), Note(0, (1, 16))])
abjad> tietools.TieSpanner(staff[:2])
TieSpanner(c'8, c'16)
abjad> spannertools.BeamSpanner(staff[:])
BeamSpanner(c'8, c'16, c'16)
abjad> tie_chain = tietools.get_tie_chain(staff[0])
abjad> tietools.tie_chain_to_diminished_tuplet_with_proportions_and_avoid_dots(tie_chain, [1, 2,
FixedDurationTuplet(3/16, [c'16, c'8, c'8])
abjad> f(staff)
\new Staff {
        \fraction \times 3/5 {
                c'16 [
                c'8
```

```
                c'8
        }
        c'16 ]
}
```

Changed in version 1.1.2: renamed `divide.tie_chain_into_arbitrary_diminution_undotted( )` to `tietools.tie_chain_to_diminished_tuplet_with_proportions_and_avoid_dots( )`.

tietools.tie_chain_to_diminished_tuplet_with_proportions_and_encourage_dots

`abjad.tools.tietools.`**`tie_chain_to_diminished_tuplet_with_proportions_and_encourage_dots`**`(`*tie_ pro por tion*

New in version 1.1.2. Divide *tie_chain* into fixed-duration tuplet according to arbitrary integer *proportions*.

Interpret *proportions* as a ratio. That is, reduce integers in *proportions* relative to each other.

Return non-trivial tuplet as diminution.

Where `proportions[i] == 1` for `i < len(proportions)`, allow tupletted notes to carry dots.

```
abjad> staff = Staff([Note(0, (1, 8)), Note(0, (1, 16)), Note(0, (1, 16))])
abjad> tietools.TieSpanner(staff[:2])
TieSpanner(c'8, c'16)
abjad> spannertools.BeamSpanner(staff[:])
BeamSpanner(c'8, c'16, c'16)
abjad> tie_chain = tietools.get_tie_chain(staff[0])
abjad> tietools.tie_chain_to_diminished_tuplet_with_proportions_and_encourage_dots(tie_chain, [1
FixedDurationTuplet(3/16, [c'8.])
abjad> f(staff)
\new Staff {
        {
                c'8. [
        }
        c'16 ]
}
```

```
abjad> staff = Staff([Note(0, (1, 8)), Note(0, (1, 16)), Note(0, (1, 16))])
abjad> tietools.TieSpanner(staff[:2])
TieSpanner(c'8, c'16)
abjad> spannertools.BeamSpanner(staff[:])
BeamSpanner(c'8, c'16, c'16)
abjad> tie_chain = tietools.get_tie_chain(staff[0])
abjad> tietools.tie_chain_to_diminished_tuplet_with_proportions_and_encourage_dots(tie_chain, [1
FixedDurationTuplet(3/16, [c'16, c'8])
abjad> f(staff)
\new Staff {
        {
                c'16 [
                c'8
        }
        c'16 ]
}
```

```
abjad> staff = Staff([Note(0, (1, 8)), Note(0, (1, 16)), Note(0, (1, 16))])
abjad> tietools.TieSpanner(staff[:2])
```

```
TieSpanner(c'8, c'16)
abjad> spannertools.BeamSpanner(staff[:])
BeamSpanner(c'8, c'16, c'16)
abjad> tie_chain = tietools.get_tie_chain(staff[0])
abjad> tietools.tie_chain_to_diminished_tuplet_with_proportions_and_encourage_dots(tie_chain, [1
FixedDurationTuplet(3/16, [c'32., c'16., c'16.])
abjad> f(staff)
\new Staff {
        \times 4/5 {
                c'32. [
                c'16.
                c'16.
        }
        c'16 ]
}
```

Changed in version 1.1.2: renamed `divide.tie_chain_into_arbitrary_diminution_dotted( )` to `tietools.tie_chain_to_diminished_tuplet_with_proportions_and_encourage_dots( )`.

tuplettools

tuplettools.FixedDurationTuplet

class `abjad.tools.tuplettools.`**`FixedDurationTuplet`**(*duration*, *music*, ***kwargs*)

Bases: `abjad.components.Tuplet.Tuplet.Tuplet`

Abjad tuplet of fixed duration and variable multiplier:

```
abjad> tuplettools.FixedDurationTuplet(Fraction(2, 8), "c'8 d'8 e'8")
FixedDurationTuplet(1/4, [c'8, d'8, e'8])
```

Return fixed-duration tuplet.

`trim`(*start*, *stop='unused'*)

Trim fixed-duration tuplet elements from *start* to *stop*:

```
abjad> tuplet = tuplettools.FixedDurationTuplet(Fraction(2, 8), "c'8 d'8 e'8")
abjad> tuplet
FixedDurationTuplet(1/4, [c'8, d'8, e'8])

abjad> tuplet.trim(2)
abjad> tuplet
FixedDurationTuplet(1/6, [c'8, d'8])
```

Preserve fixed-duration tuplet multiplier.

Adjust fixed-duration tuplet duration.

Return none.

tuplettools.beam_bottommost_tuplets_in_expr

`abjad.tools.tuplettools.`**`beam_bottommost_tuplets_in_expr`**(*expr*)

Beam bottommost tuplets in *expr*:

```
abjad> staff = Staff(3 * Tuplet((2, 3), "c'8 d'8 e'8"))
```

```
f(staff)
\new Staff {
   \times 2/3 {
      c'8
      d'8
      e'8
   }
   \times 2/3 {
      c'8
      d'8
      e'8
   }
   \times 2/3 {
      c'8
      d'8
      e'8
   }
}

abjad> tuplettools.beam_bottommost_tuplets_in_expr(staff)

abjad> f(staff)
\new Staff {
   \times 2/3 {
      c'8 [
      d'8
      e'8 ]
   }
   \times 2/3 {
      c'8 [
      d'8
      e'8 ]
   }
   \times 2/3 {
      c'8 [
      d'8
      e'8 ]
   }
}
```

Return none.

tuplettools.change_augmented_tuplets_in_expr_to_diminished

abjad.tools.tuplettools.**change_augmented_tuplets_in_expr_to_diminished**(*tuplet*)

New in version 1.1.2. Multiply the written duration of the leaves in *tuplet* by the least power of 2 necessary to diminshed *tuplet*.

```
abjad> tuplet = tuplettools.FixedDurationTuplet((2, 4), macros.scale(3))
abjad> tuplet
FixedDurationTuplet(1/2, [c'8, d'8, e'8])
abjad> tuplettools.change_augmented_tuplets_in_expr_to_diminished(tuplet)
FixedDurationTuplet(1/2, [c'4, d'4, e'4])
```

Todo

make work with nested tuplets.

Changed in version 1.1.2: renamed `tuplettools.augmentation_to_diminution( )` to `tuplettools.change_augmented_tuplets_in_expr_to_diminished( )`.

tuplettools.change_diminished_tuplets_in_expr_to_augmented

`abjad.tools.tuplettools.`**`change_diminished_tuplets_in_expr_to_augmented`**`(`*tuplet*`)`

New in version 1.1.2. Divide the written duration of the leaves in *tuplet* by the least power of 2 necessary to augment *tuplet*.

```
abjad> tuplet = tuplettools.FixedDurationTuplet((2, 8), macros.scale(3))
abjad> tuplet
FixedDurationTuplet(1/4, [c'8, d'8, e'8])
abjad> tuplettools.change_diminished_tuplets_in_expr_to_augmented(tuplet)
FixedDurationTuplet(1/4, [c'16, d'16, e'16])
```

Todo

make work with nested tuplets.

Changed in version 1.1.2: renamed `tuplettools.diminution_to_augmentation( )` to `tuplettools.change_diminished_tuplets_in_expr_to_augmented( )`.

tuplettools.fix_contents_of_tuplets_in_expr

`abjad.tools.tuplettools.`**`fix_contents_of_tuplets_in_expr`**`(`*tuplet*`)`

Scale *tuplet* contents by power of two if tuplet multiplier less than 1/2 or greater than 2. Return tuplet.

```
abjad> tuplet = tuplettools.FixedDurationTuplet((2, 8), macros.scale(3, Duration(1, 4)))
abjad> tuplet
FixedDurationTuplet(1/4, [c'4, d'4, e'4])
abjad> tuplettools.fix_contents_of_tuplets_in_expr(tuplet)
FixedDurationTuplet(1/4, [c'8, d'8, e'8])
```

Changed in version 1.1.2: renamed `tuplettools.contents_fix( )` to `tuplettools.fix_contents_of_tuplets_in_expr( )`.

tuplettools.fuse_tuplets

`abjad.tools.tuplettools.`**`fuse_tuplets`**`(`*tuplets*`)`

Fuse parent-contiguous *tuplets*:

```
abjad> t1 = tuplettools.FixedDurationTuplet((2, 8), macros.scale(3))
abjad> spannertools.BeamSpanner(t1[:])
BeamSpanner(c'8, d'8, e'8)
abjad> t2 = tuplettools.FixedDurationTuplet((2, 16), macros.scale(3, Fraction(1, 16)))
abjad> spannertools.SlurSpanner(t2[:])
SlurSpanner(c'16, d'16, e'16)
abjad> staff = Staff([t1, t2])
abjad> f(staff)
\new Staff {
   \times 2/3 {
      c'8 [
      d'8
```

```
            e'8 ]
        }
        \times 2/3 {
            c'16 (
            d'16
            e'16 )
        }
    }

abjad> tuplettools.fuse_tuplets(staff[:])
FixedDurationTuplet(3/8, [c'8, d'8, e'8, c'16, d'16, e'16])

abjad> f(staff)
\new Staff {
    \times 2/3 {
        c'8 [
        d'8
        e'8 ]
        c'16 (
        d'16
        e'16 )
    }
}
```

Return new tuplet.

Fuse zero or more parent-contiguous *tuplets*.

Allow in-score *tuplets*.

Allow outside-of-score *tuplets*.

All *tuplets* must carry the same multiplier.

All *tuplets* must be of the same type. Changed in version 1.1.2: renamed `fuse.tuplets_by_reference( )` to `tuplettools.fuse_tuplets( )`.

tuplettools.get_first_tuplet_in_improper_parentage_of_component

`abjad.tools.tuplettools.`**`get_first_tuplet_in_improper_parentage_of_component`**(*component*)

New in version 1.1.2. Get first tuplet in improper parentage of *component*:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> Tuplet((2, 3), staff[:3])
Tuplet(2/3, [c'8, d'8, e'8])

abjad> f(staff)
\new Staff {
    \times 2/3 {
        c'8
        d'8
        e'8
    }
    f'8
}

abjad> tuplettools.get_first_tuplet_in_improper_parentage_of_component(staff.leaves[1])
Tuplet(2/3, [c'8, d'8, e'8])
```

Return tuplet or none.

tuplettools.get_first_tuplet_in_proper_parentage_of_component

abjad.tools.tuplettools.**get_first_tuplet_in_proper_parentage_of_component**(*component*)

New in version 1.1.2. Get first tuplet in proper parentage of *component*:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8")
abjad> Tuplet((2, 3), staff[:3])
Tuplet(2/3, [c'8, d'8, e'8])
```

```
abjad> f(staff)
\new Staff {
   \times 2/3 {
      c'8
      d'8
      e'8
   }
   f'8
}
```

```
abjad> tuplettools.get_first_tuplet_in_proper_parentage_of_component(staff.leaves[1])
Tuplet(2/3, [c'8, d'8, e'8])
```

Return tuplet or none.

tuplettools.is_proper_tuplet_multiplier

abjad.tools.tuplettools.**is_proper_tuplet_multiplier**(*multiplier*)

True when `1/2 < multiplier < 2`.

```
abjad> for n in range(17):
...     rational = fractions.Fraction(n, 8)
...     multiplier = tuplettools.is_proper_tuplet_multiplier(rational)
...     print '%s   %s' % (rational, multiplier)
...
0       False
1/8     False
1/4     False
3/8     False
1/2     False
5/8     True
3/4     True
7/8     True
1       True
9/8     True
5/4     True
11/8    True
3/2     True
13/8    True
7/4     True
15/8    True
2       False
```

This function models the idea that `4:3`, `4:5`, `4:6`, `4:7` are valid tuplet multipliers while `4:2` and `4:8` aren't. Changed in version 1.1.2: renamed `durtools.is_tuplet_multiplier( )` to `tuplettools.is_proper_tuplet_multiplier( )`.

tuplettools.iterate_tuplets_backward_in_expr

abjad.tools.tuplettools.**iterate_tuplets_backward_in_expr**(*expr*, *start=0*, *stop=None*)

New in version 1.1.2. Iterate tuplets backward in *expr*:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8 g'8 a'8 b'8 c''8")
abjad> Tuplet((2, 3), staff[:3])
Tuplet(2/3, [c'8, d'8, e'8])
abjad> Tuplet((2, 3), staff[-3:])
Tuplet(2/3, [a'8, b'8, c''8])

abjad> f(staff)
\new Staff {
   \times 2/3 {
      c'8
      d'8
      e'8
   }
   f'8
   g'8
   \times 2/3 {
      a'8
      b'8
      c''8
   }
}

abjad> for tuplet in tuplettools.iterate_tuplets_backward_in_expr(staff):
...     tuplet
...
Tuplet(2/3, [a'8, b'8, c''8])
Tuplet(2/3, [c'8, d'8, e'8])
```

Return generator.

tuplettools.iterate_tuplets_forward_in_expr

abjad.tools.tuplettools.**iterate_tuplets_forward_in_expr**(*expr*, *start=0*, *stop=None*)

New in version 1.1.2. Iterate tuplets forward in *expr*:

```
abjad> staff = Staff("c'8 d'8 e'8 f'8 g'8 a'8 b'8 c''8")
abjad> Tuplet((2, 3), staff[:3])
Tuplet(2/3, [c'8, d'8, e'8])
abjad> Tuplet((2, 3), staff[-3:])
Tuplet(2/3, [a'8, b'8, c''8])

abjad> f(staff)
\new Staff {
   \times 2/3 {
      c'8
      d'8
      e'8
   }
   f'8
   g'8
   \times 2/3 {
      a'8
```

```
        b'8
        c''8
    }
}
```

```
abjad> for tuplet in tuplettools.iterate_tuplets_forward_in_expr(staff):
...     tuplet
...
Tuplet(2/3, [c'8, d'8, e'8])
Tuplet(2/3, [a'8, b'8, c''8])
```

Return generator.

tuplettools.make_augmented_tuplet_from_duration_and_proportions_and_avoid_dots

abjad.tools.tuplettools.**make_augmented_tuplet_from_duration_and_proportions_and_avoid_dots**

New in version 1.1.2. Make augmented tuplet from *duration* and *proportions* and avoid dots.

Return tupletted leaves strictly without dots when all *proportions* equal `1`:

```
abjad> print tuplettools.make_augmented_tuplet_from_duration_and_proportions_and_avoid_dots(
... Fraction(3, 16), [1, 1, 1, -1, -1])
{@ 5:6 c'32, c'32, c'32, r32, r32 @}
```

Allow tupletted leaves to return with dots when some *proportions* do not equal `1`:

```
abjad> print tuplettools.make_augmented_tuplet_from_duration_and_proportions_and_avoid_dots(
... Fraction(3, 16), [1, -2, -2, 3, 3])
{@ 11:12 c'64, r32, r32, c'32., c'32. @}
```

Interpret nonassignable *proportions* according to *direction*:

```
abjad> print tuplettools.make_augmented_tuplet_from_duration_and_proportions_and_avoid_dots(
... Fraction(3, 16), [5, -1, 5], direction = 'little-endian')
{@ 11:12 c'64, c'16, r64, c'64, c'16 @}
```

Reduce *proportions* relative to each other.

Interpret negative *proportions* as rests.

Return fixed-duration tuplet. Changed in version 1.1.2: renamed `divide.duration_into_arbitrary_augmentation_undotted()` to `tuplettools.make_augmented_tuplet_from_duration_and_proportions_and_avoid_dots()`.

tuplettools.make_augmented_tuplet_from_duration_and_proportions_and_encourage_dots

abjad.tools.tuplettools.**make_augmented_tuplet_from_duration_and_proportions_and_encourage_d**

New in version 1.1.2. Make augmented tuplet from *duration* and *proportions* and encourage dots:

```
abjad> print tuplettools.make_augmented_tuplet_from_duration_and_proportions_and_encourage_dots(
... Fraction(3, 16), [1, 1, 1, -1, -1])
{@ 5:8 c'64., c'64., c'64., r64., r64. @}
```

Interpret nonassignable *proportions* according to *direction*:

```
abjad> print tuplettools.make_augmented_tuplet_from_duration_and_proportions_and_encourage_dots(
... Fraction(3, 16), [5, -1, 5], direction = 'little-endian')
{@ 11:16 c'32..., r128., c'32... @}
```

Reduce *proportions* relative to each other.

Interpret negative *proportions* as rests.

Return fixed-duration tuplet. Changed in version 1.1.2: renamed `divide.duration_into_arbitrary_augmentation_dotted( )` to `tuplettools.make_augmented_tuplet_from_duration_and_proportions_and_encourage_dots( )`.

tuplettools.make_diminished_tuplet_from_duration_and_proportions_and_avoid_dots

abjad.tools.tuplettools.**make_diminished_tuplet_from_duration_and_proportions_and_avoid_dots**

New in version 1.1.2. Make diminished tuplet from *duration* and nonzero integer *proportions*.

Return tupletted leaves strictly without dots when all *proportions* equal `1`:

```
abjad> print tuplettools.make_diminished_tuplet_from_duration_and_proportions_and_avoid_dots(
... Fraction(3, 16), [1, 1, 1, -1, -1])
{@ 5:3 c'16, c'16, c'16, r16, r16 @}
```

Allow tupletted leaves to return with dots when some *proportions* do not equal `1`:

```
abjad> print tuplettools.make_diminished_tuplet_from_duration_and_proportions_and_avoid_dots(
... Fraction(3, 16), [1, -2, -2, 3, 3])
{@ 11:6 c'32, r16, r16, c'16., c'16. @}
```

Interpret nonassignable *proportions* according to *direction*:

```
abjad> print tuplettools.make_diminished_tuplet_from_duration_and_proportions_and_avoid_dots(
... Fraction(3, 16), [5, -1, 5], direction = 'little-endian')
{@ 11:6 c'32, c'8, r32, c'32, c'8 @}
```

Reduce *proportions* relative to each other.

Interpret negative *proportions* as rets.

Return fixed-duration tuplet. Changed in version 1.1.2: renamed `divide.duration_into_arbitrary_diminution_undotted( )` to `tuplettools.make_diminished_tuplet_from_duration_and_proportions_and_avoid_dots( )`.

tuplettools.make_diminished_tuplet_from_duration_and_proportions_and_encourage_dots

abjad.tools.tuplettools.**make_diminished_tuplet_from_duration_and_proportions_and_encourage_

New in version 1.1.2. Make diminished tuplet from *duration* and *proportions* and encourage dots:

```
abjad> print tuplettools.make_diminished_tuplet_from_duration_and_proportions_and_encourage_dots
... Fraction(3, 16), [1, 1, 1, -1, -1])
{@ 5:4 c'32., c'32., c'32., r32., r32. @}
```

Interpret nonassignable *proportions* according to *direction*:

```
abjad> print tuplettools.make_diminished_tuplet_from_duration_and_proportions_and_encourage_dots
... Fraction(3, 16), [5, -1, 5], direction = 'little-endian')
{@ 11:8 c'16..., r64., c'16... @}
```

Reduce *proportions* relative to each other.

Interpret negative *proportions* as rests.

Return fixed-duration tuplet. Changed in version 1.1.2: renamed `divide.duration_into_arbitrary_diminution_dotted( )` to `tuplettools.make_diminished_tuplet_from_duration_and_proportions_and_encourage_dots( )`.

tuplettools.make_tuplet_from_proportions_and_pair

abjad.tools.tuplettools.**make_tuplet_from_proportions_and_pair**(*l*, *(n, d)*, *together=False*)

Divide *(n, d)* according to *l*.

Where no prolation is necessary, return container.

```
abjad> tuplettools.make_tuplet_from_proportions_and_pair([1], (7, 16))
{c'4..}
```

Where prolation is necessary, return fixed-duration tuplet.

```
abjad> tuplettools.make_tuplet_from_proportions_and_pair([1, 2], (7, 16))
FixedDurationTuplet(7/16, [c'8, c'4])

abjad> tuplettools.make_tuplet_from_proportions_and_pair([1, 2, 4], (7, 16))
FixedDurationTuplet(7/16, [c'16, c'8, c'4])

abjad> tuplettools.make_tuplet_from_proportions_and_pair([1, 2, 4, 1], (7, 16))
FixedDurationTuplet(7/16, [c'16, c'8, c'4, c'16])
```

```
abjad> tuplettools.make_tuplet_from_proportions_and_pair([1, 2, 4, 1, 2], (7, 16))
FixedDurationTuplet(7/16, [c'16, c'8, c'4, c'16, c'8])

abjad> tuplettools.make_tuplet_from_proportions_and_pair([1, 2, 4, 1, 2, 4], (7, 16))
FixedDurationTuplet(7/16, [c'16, c'8, c'4, c'16, c'8, c'4])
```

Note: function accepts a pair rather than a rational.

Note: function interprets *d* as tuplet denominator.

Changed in version 1.1.2: renamed `divide.pair( )` to `tuplettools.make_tuplet_from_proportions_and_pa` `).`

tuplettools.move_prolation_of_tuplet_to_contents_of_tuplet_and_remove_tuplet

`abjad.tools.tuplettools.`**`move_prolation_of_tuplet_to_contents_of_tuplet_and_remove_tuplet`**`(`*tu*

Scale `tuplet` contents and then bequeath in-score position of `tuplet` to contents.

Return orphaned `tuplet` emptied of all contents.

```
abjad> t = Staff(tuplettools.FixedDurationTuplet((3, 8), macros.scale(2)) * 2)
abjad> spannertools.BeamSpanner(t.leaves)
BeamSpanner(c'8, d'8, c'8, d'8)
abjad> print t.format
\new Staff {
        \fraction \times 3/2 {
                c'8 [
                d'8
        }
        \fraction \times 3/2 {
                c'8
                d'8 ]
        }
}

abjad> tuplettools.move_prolation_of_tuplet_to_contents_of_tuplet_and_remove_tuplet(t[0])
FixedDurationTuplet(3/8, [ ])
abjad> print t.format
\new Staff {
        c'8. [
        d'8.
        \fraction \times 3/2 {
                c'8
                d'8 ]
        }
}
```

Changed in version 1.1.2: renamed `tuplettools.subsume( )` to `tuplettools.move_prolation_of_tuplet_to_contents_of_tuplet_and_remove_tuplet( )`.

tuplettools.remove_trivial_tuplets_in_expr

abjad.tools.tuplettools.**remove_trivial_tuplets_in_expr**(*expr*)

Remove trivial tuplets in *expr*:

```
abjad> t = tuplettools.FixedDurationTuplet((1, 4), macros.scale(3))
abjad> u = tuplettools.FixedDurationTuplet((1, 4), macros.scale(2))
abjad> s = Staff([t, u])
abjad> len(s)
2

abjad> s[0]
FixedDurationTuplet(1/4, [c'8, d'8, e'8])
abjad> s[1]
FixedDurationTuplet(1/4, [c'8, d'8])

abjad> tuplettools.remove_trivial_tuplets_in_expr(s)
abjad> len(s)
3
abjad> s[0]
FixedDurationTuplet(1/4, [c'8, d'8, e'8])
abjad> s[1]
Note("c'8")
abjad> s[2]
Note("d'8")

abjad> f(s)
\new Staff {
        \times 2/3 {
                c'8
                d'8
                e'8
        }
        c'8
        d'8
}
```

Replace trivial tuplets with plain leaves.

Return none. Changed in version 1.1.2: renamed `tuplettools.slip_trivial( )` to `tuplettools.remove_trivial_tuplets_in_expr( )`.

tuplettools.scale_contents_of_tuplets_in_expr_by_multiplier

abjad.tools.tuplettools.**scale_contents_of_tuplets_in_expr_by_multiplier**(*tuplet*, *multiplier*)

Scale fixed-duration tuplet by multiplier. Preserve tuplet multiplier. Return tuplet.

tuplettools.set_denominator_of_tuplets_in_expr_to_at_least

abjad.tools.tuplettools.**set_denominator_of_tuplets_in_expr_to_at_least**(*expr*, *n*)

New in version 1.1.2. Set denominator of tuplets in *expr* to at least *n*:

```
abjad> tuplet = Tuplet((3, 5), "c'4 d'8 e'8 f'4 g'2")

abjad> f(tuplet)
\fraction \times 3/5 {
   c'4
   d'8
   e'8
   f'4
   g'2
}

abjad> tuplettools.set_denominator_of_tuplets_in_expr_to_at_least(tuplet, 8)

abjad> f(tuplet)
\fraction \times 6/10 {
   c'4
   d'8
   e'8
   f'4
   g'2
}
```

Return none.

verticalitytools

verticalitytools.VerticalMoment

class `abjad.tools.verticalitytools.`**`VerticalMoment`**(*prolated_offset*, *governors*, *components*)

Bases: `abjad.core._Immutable._Immutable._Immutable`

Everything happening at a single moment in musical time:

```
abjad> score = Score([scoretools.PianoStaff([Staff("c'4 e'4 d'4 f'4"), Staff('g2 f2')])])
abjad> contexttools.ClefMark('bass')(score[0][1])
ClefMark('bass')(Staff{2})

f(score)
\new Score <<
   \new PianoStaff <<
      \new Staff {
         c'4
         e'4
         d'4
         f'4
      }
      \new Staff {
         \clef "bass"
         g2
         f2
      }
   >>
>>

abjad> for vertical_moment in verticalitytools.iterate_vertical_moments_forward_in_expr(score):
...     vertical_moment
...
```

```
VerticalMoment(0, <<2>>)
VerticalMoment(1/4, <<2>>)
VerticalMoment(1/2, <<2>>)
VerticalMoment(3/4, <<2>>)
```

Create vertical moments with the getters and iterators implemented in the verticalitytools module.

Vertical moments are immutable.

attack_count
Positive integer number of pitch carriers starting at vertical moment.

components
Read-only tuple of zero or more components happening at vertical moment.

It is always the case that `self.components = self.overlap_components + self.start_components`.

governors
Read-only tuple of one or more containers in which vertical moment is evaluated.

leaves
Read-only tuple of zero or more leaves at vertical moment.

measures
Read-only tuplet of zero or more measures at vertical moment.

next_vertical_moment
Read-only reference to next vertical moment forward in time.

notes
Read-only tuple of zero or more notes at vertical moment.

overlap_components
Read-only tuple of components in vertical moment starting before vertical moment, ordered by score index.

overlap_leaves
Read-only tuple of leaves in vertical moment starting before vertical moment, ordered by score index.

overlap_measures
Read-only tuple of measures in vertical moment starting before vertical moment, ordered by score index.

overlap_notes
Read-only tuple of notes in vertical moment starting before vertical moment, ordered by score index.

prev_vertical_moment
Read-only reference to prev vertical moment backward in time.

prolated_offset
Read-only rational-valued score offset at which vertical moment is evaluated.

start_components
Read-only tuple of components in vertical moment starting with at vertical moment, ordered by score index.

start_leaves
Read-only tuple of leaves in vertical moment starting with vertical moment, ordered by score index.

start_notes
Read-only tuple of notes in vertical moment starting with vertical moment, ordered by score index.

verticalitytools.get_vertical_moment_at_prolated_offset_in_expr

`abjad.tools.verticalitytools.`**`get_vertical_moment_at_prolated_offset_in_expr`**`(`*governor*, *prolated_offset*`)`

New in version 1.1.2. Get vertical moment at *prolated_offset* in *governor*.

```
abjad> score = Score([ ])
abjad> score.append(Staff([tuplettools.FixedDurationTuplet((4, 8), notetools.make_repeated_notes
abjad> piano_staff = scoretools.PianoStaff([ ])
abjad> piano_staff.append(Staff(notetools.make_repeated_notes(2, Duration(1, 4))))
abjad> piano_staff.append(Staff(notetools.make_repeated_notes(4)))
abjad> contexttools.ClefMark('bass')(piano_staff[1])
ClefMark('bass')(Staff{4})
abjad> score.append(piano_staff)
abjad> macros.diatonicize(list(reversed(score.leaves)))
abjad> f(score)
\new Score <<
        \new Staff {
                \fraction \times 4/3 {
                        d''8
                        c''8
                        b'8
                }
        }
        \new PianoStaff <<
                \new Staff {
                        a'4
                        g'4
                }
                \new Staff {
                        \clef "bass"
                        f'8
                        e'8
                        d'8
                        c'8
                }
        >>
>>
abjad> vertical_moment = verticalitytools.get_vertical_moment_at_prolated_offset_in_expr(piano_s
abjad> vertical_moment.leaves
(Note("a'4"), Note("e'8"))
```

Todo

optimize without full-component traversal.

Changed in version 1.1.2: renamed `iterate.get_vertical_moment_at_prolated_offset_in( )` to `verticalitytools.get_vertical_moment_at_prolated_offset_in_expr( )`.Changed in version 1.1.2: renamed `iterate.get_vertical_moment_at_prolated_offset_in_expr( )` to `verticalitytools.get_vertical_moment_at_prolated_offset_in_expr( )`.

verticalitytools.get_vertical_moment_starting_with_component

abjad.tools.verticalitytools.**get_vertical_moment_starting_with_component**(*expr*, *governor=None*)

New in version 1.1.2. When *governor* is none, get vertical moment at `expr._offset.start` in score root of *expr*:

```
abjad> score = Score([ ])
abjad> score.append(Staff([tuplettools.FixedDurationTuplet((4, 8), notetools.make_repeated_notes
abjad> piano_staff = scoretools.PianoStaff([ ])
abjad> piano_staff.append(Staff(notetools.make_repeated_notes(2, Duration(1, 4))))
abjad> piano_staff.append(Staff(notetools.make_repeated_notes(4)))
abjad> contexttools.ClefMark('bass')(piano_staff[1])
ClefMark('bass')(Staff{4})
abjad> score.append(piano_staff)
abjad> macros.diatonicize(list(reversed(score.leaves)))
abjad> f(score)
\new Score <<
        \new Staff {
                \fraction \times 4/3 {
                        d''8
                        c''8
                        b'8
                }
        }
        \new PianoStaff <<
                \new Staff {
                        a'4
                        g'4
                }
                \new Staff {
                        \clef "bass"
                        f'8
                        e'8
                        d'8
                        c'8
                }
        >>
>>
abjad> verticalitytools.get_vertical_moment_starting_with_component(piano_staff[1][1])
VerticalMoment(1/8, <<3>>)
```

When *governor* is not none, get vertical moment at `expr._offset.start` in *governor*.

```
abjad> verticalitytools.get_vertical_moment_starting_with_component(piano_staff[1][1], piano_sta
VerticalMoment(1/8, <<2>>)
```

Todo

optimize without full-component traversal.

Changed in version 1.1.2: renamed `iterate.get_vertical_moment_starting_with( )` to `verticalitytools.get_vertical_moment_starting_with_component( )`.Changed in version 1.1.2: renamed `iterate.get_vertical_moment_starting_with_component( )` to `verticalitytools.get_vertical_moment_starting_with_component( )`.

verticalitytools.iterate_vertical_moments_backward_in_expr

`abjad.tools.verticalitytools.`**`iterate_vertical_moments_backward_in_expr`**(*governor*)

New in version 1.1.2. Yield vertical moments forward in *governor*.

```
abjad> score = Score([ ])
abjad> score.append(Staff([tuplettools.FixedDurationTuplet((4, 8), notetools.make_repeated_notes
abjad> piano_staff = scoretools.PianoStaff([ ])
abjad> piano_staff.append(Staff(notetools.make_repeated_notes(2, Duration(1, 4))))
abjad> piano_staff.append(Staff(notetools.make_repeated_notes(4)))
abjad> contexttools.ClefMark('bass')(piano_staff[1])
ClefMark('bass')(Staff{4})
abjad> score.append(piano_staff)
abjad> macros.diatonicize(list(reversed(score.leaves)))
abjad> f(score)
\new Score <<
        \new Staff {
                \fraction \times 4/3 {
                        d''8
                        c''8
                        b'8
                }
        }
        \new PianoStaff <<
                \new Staff {
                        a'4
                        g'4
                }
                \new Staff {
                        \clef "bass"
                        f'8
                        e'8
                        d'8
                        c'8
                }
        >>
>>
abjad> for vertical_moment in verticalitytools.iterate_vertical_moments_backward_in_expr(score):
...     vertical_moment.leaves
...
(Note("b'8"), Note("g'4"), Note("c'8"))
(Note("b'8"), Note("g'4"), Note("d'8"))
(Note("c''8"), Note("g'4"), Note("d'8"))
(Note("c''8"), Note("a'4"), Note("e'8"))
(Note("d''8"), Note("a'4"), Note("e'8"))
(Note("d''8"), Note("a'4"), Note("f'8"))
abjad> for vertical_moment in verticalitytools.iterate_vertical_moments_backward_in_expr(piano_s
...     vertical_moment.leaves
...
(Note("g'4"), Note("c'8"))
(Note("g'4"), Note("d'8"))
(Note("a'4"), Note("e'8"))
(Note("a'4"), Note("f'8"))
```

Todo

optimize without multiple full-component traversal.

Changed in version 1.1.2: renamed `iterate.vertical_moments_backward_in()` to `verticalitytools.iterate_vertical_moments_backward_in_expr()`.Changed in version 1.1.2: renamed `iterate.vertical_moments_backward_in_expr()` to `verticalitytools.iterate_vertical_moments_backward_in_expr()`.

verticalitytools.iterate_vertical_moments_forward_in_expr

`abjad.tools.verticalitytools.`**`iterate_vertical_moments_forward_in_expr`**`(`*governor*`)`

New in version 1.1.2. Yield vertical moments forward in *governor*.

```
abjad> score = Score([ ])
abjad> score.append(Staff([tuplettools.FixedDurationTuplet((4, 8), notetools.make_repeated_notes
abjad> piano_staff = scoretools.PianoStaff([ ])
abjad> piano_staff.append(Staff(notetools.make_repeated_notes(2, Duration(1, 4))))
abjad> piano_staff.append(Staff(notetools.make_repeated_notes(4)))
abjad> contexttools.ClefMark('bass')(piano_staff[1])
ClefMark('bass')(Staff{4})
abjad> score.append(piano_staff)
abjad> macros.diatonicize(list(reversed(score.leaves)))
abjad> f(score)
\new Score <<
	\new Staff {
		\fraction \times 4/3 {
			d''8
			c''8
			b'8
		}
	}
	\new PianoStaff <<
		\new Staff {
			a'4
			g'4
		}
		\new Staff {
			\clef "bass"
			f'8
			e'8
			d'8
			c'8
		}
	>>
>>
abjad> for vertical_moment in verticalitytools.iterate_vertical_moments_forward_in_expr(score):
...     vertical_moment.leaves
...
(Note("d''8"), Note("a'4"), Note("f'8"))
(Note("d''8"), Note("a'4"), Note("e'8"))
(Note("c''8"), Note("a'4"), Note("e'8"))
(Note("c''8"), Note("g'4"), Note("d'8"))
(Note("b'8"), Note("g'4"), Note("d'8"))
(Note("b'8"), Note("g'4"), Note("c'8"))
abjad> for vertical_moment in verticalitytools.iterate_vertical_moments_forward_in_expr(piano_st
...     vertical_moment.leaves
...
(Note("a'4"), Note("f'8"))
(Note("a'4"), Note("e'8"))
(Note("g'4"), Note("d'8"))
```

```
(Note("g'4"), Note("c'8"))
```

Todo

optimize without multiple full-component traversal.

Changed in version 1.1.2: renamed `iterate.vertical_moments_forward_in( )` to `verticalitytools.iterate_vertical_moments_forward_in_expr( )`.Changed in version 1.1.2: renamed `iterate.vertical_moments_forward_in_expr( )` to `verticalitytools.iterate_vertical_moments_forward_in_expr( )`.

verticalitytools.label_vertical_moments_in_expr_with_chromatic_interval_classes

`abjad.tools.verticalitytools.`**`label_vertical_moments_in_expr_with_chromatic_interval_classes`**

New in version 1.1.2. Label harmonic chromatic interval-classes of every vertical moment in *expr*.

```
abjad> score = Score(Staff([ ]) * 3)
abjad> score[0].extend(macros.scale(4))
abjad> contexttools.ClefMark('alto')(score[1])
ClefMark('alto')(Staff{ })
abjad> score[1].extend([Note(-5, (1, 4)), Note(-7, (1, 4))])
abjad> contexttools.ClefMark('bass')(score[2])
ClefMark('bass')(Staff{ })
abjad> score[2].append(Note(-24, (1, 2)))
abjad> verticalitytools.label_vertical_moments_in_expr_with_chromatic_interval_classes(score)
abjad> f(score)
\new Score <<
	\new Staff {
		c'8
		d'8 _ \markup { \small { \column { 2 7 } } }
		e'8
		f'8 _ \markup { \small { \column { 5 5 } } }
	}
	\new Staff {
		\clef "alto"
		g4
		f4 _ \markup { \small { \column { 4 5 } } }
	}
	\new Staff {
		\clef "bass"
		c,2 _ \markup { \small { \column { 12 7 } } }
	}
>>
```

Changed in version 1.1.2: renamed `label.vertical_moment_chromatic_interval_classes( )` to `verticalitytools.label_vertical_moments_in_expr_with_chromatic_interval_classes( )`.

verticalitytools.label_vertical_moments_in_expr_with_chromatic_intervals

`abjad.tools.verticalitytools.`**`label_vertical_moments_in_expr_with_chromatic_intervals`**(*expr*, *markup_*

New in version 1.1.2. Label harmonic chromatic intervals of every vertical moment in *expr*.

```
abjad> score = Score(Staff([ ]) * 3)
abjad> score[0].extend(macros.scale(4))
abjad> contexttools.ClefMark('alto')(score[1])
ClefMark('alto')(Staff{ })
abjad> score[1].extend([Note(-5, (1, 4)), Note(-7, (1, 4))])
abjad> contexttools.ClefMark('bass')(score[2])
ClefMark('bass')(Staff{ })
abjad> score[2].append(Note(-24, (1, 2)))
abjad> verticalitytools.label_vertical_moments_in_expr_with_chromatic_intervals(score)
abjad> f(score)
\new Score <<
        \new Staff {
                c'8
                d'8 _ \markup { \small { \column { 26 19 } } }
                e'8
                f'8 _ \markup { \small { \column { 29 17 } } }
        }
        \new Staff {
                \clef "alto"
                g4
                f4 _ \markup { \small { \column { 28 17 } } }
        }
        \new Staff {
                \clef "bass"
                c,2 _ \markup { \small { \column { 24 19 } } }
        }
>>
```

Changed in version 1.1.2: renamed `label.vertical_moment_chromatic_intervals( )` to `verticalitytools.label_vertical_moments_in_expr_with_chromatic_intervals( )`.

verticalitytools.label_vertical_moments_in_expr_with_counterpoint_intervals

`abjad.tools.verticalitytools.`**`label_vertical_moments_in_expr_with_counterpoint_intervals`**(*exp* *ma*

New in version 1.1.2. Label counterpoint interval of every vertical moment in *expr*.

```
abjad> score = Score(Staff([ ]) * 3)
abjad> score[0].extend(macros.scale(4))
abjad> contexttools.ClefMark('alto')(score[1])
ClefMark('alto')(Staff{ })
abjad> score[1].extend([Note(-5, (1, 4)), Note(-7, (1, 4))])
abjad> contexttools.ClefMark('bass')(score[2])
ClefMark('bass')(Staff{ })
abjad> score[2].append(Note(-24, (1, 2)))
abjad> verticalitytools.label_vertical_moments_in_expr_with_counterpoint_intervals(score)
abjad> f(score)
\new Score <<
        \new Staff {
                c'8
                d'8 _ \markup { \small { \column { 2 5 } } }
                e'8
                f'8 _ \markup { \small { \column { 4 4 } } }
        }
        \new Staff {
                \clef "alto"
```

```
                g4
                f4 _ \markup { \small { \column { 3 4 } } }
        }
        \new Staff {
                \clef "bass"
                c,2 _ \markup { \small { \column { 8 5 } } }
        }
>>
```

Changed in version 1.1.2: renamed `label.vertical_moment_counterpoint_intervals( )` to `verticalitytools.label_vertical_moments_in_expr_with_counterpoint_intervals( )`.

verticalitytools.label_vertical_moments_in_expr_with_diatonic_intervals

`abjad.tools.verticalitytools.`**`label_vertical_moments_in_expr_with_diatonic_intervals`**`(`*expr*, *markup_d*

New in version 1.1.2. Label diatonic intervals of every vertical moment in *expr*.

```
abjad> score = Score(Staff([ ]) * 3)
abjad> score[0].extend(macros.scale(4))
abjad> contexttools.ClefMark('alto')(score[1])
ClefMark('alto')(Staff{ })
abjad> score[1].extend([Note(-5, (1, 4)), Note(-7, (1, 4))])
abjad> contexttools.ClefMark('bass')(score[2])
ClefMark('bass')(Staff{ })
abjad> score[2].append(Note(-24, (1, 2)))
abjad> verticalitytools.label_vertical_moments_in_expr_with_diatonic_intervals(score)
abjad> f(score)
\new Score <<
        \new Staff {
                c'8
                d'8 _ \markup { \small { \column { 16 12 } } }
                e'8
                f'8 _ \markup { \small { \column { 18 11 } } }
        }
        \new Staff {
                \clef "alto"
                g4
                f4 _ \markup { \small { \column { 17 11 } } }
        }
        \new Staff {
                \clef "bass"
                c,2 _ \markup { \small { \column { 15 12 } } }
        }
>>
```

Changed in version 1.1.2: renamed `label.vertical_moment_diatonic_intervals( )` to `verticalitytools.label_vertical_moments_in_expr_with_diatonic_intervals( )`.

verticalitytools.label_vertical_moments_in_expr_with_interval_class_vectors

`abjad.tools.verticalitytools.`**`label_vertical_moments_in_expr_with_interval_class_vectors`**`(`*exp*
mar

New in version 1.1.2. Label interval-class vector of every vertical moment in *expr*.

```
abjad> score = Score(Staff([ ]) * 3)
abjad> score[0].extend(macros.scale(4))
abjad> contexttools.ClefMark('alto')(score[1])
ClefMark('alto')(Staff{ })
abjad> score[1].extend([Note(-5, (1, 4)), Note(-7, (1, 4))])
abjad> contexttools.ClefMark('bass')(score[2])
ClefMark('bass')(Staff{ })
abjad> score[2].append(Note(-24, (1, 2)))
abjad> verticalitytools.label_vertical_moments_in_expr_with_interval_class_vectors(score)
abjad> f(score)
\new Score <<
        \new Staff {
                c'8
                d'8 _ \markup { \tiny { 0010020 } }
                e'8
                f'8 _ \markup { \tiny { 1000020 } }
        }
        \new Staff {
                \clef "alto"
                g4
                f4 _ \markup { \tiny { 0100110 } }
        }
        \new Staff {
                \clef "bass"
                c,2 _ \markup { \tiny { 1000020 } }
        }
>>
```

Changed in version 1.1.2: renamed `label.vertical_moment_interval_class_vectors( )` to `verticalitytools.label_vertical_moments_in_expr_with_interval_class_vectors( )`.

verticalitytools.label_vertical_moments_in_expr_with_numbered_chromatic_pitch_classes

abjad.tools.verticalitytools.**label_vertical_moments_in_expr_with_numbered_chromatic_pitch_c**

New in version 1.1.2. Label pitch-classes of every vertical moment in *expr*.

```
abjad> score = Score(Staff([ ]) * 3)
abjad> score[0].extend(macros.scale(4))
abjad> contexttools.ClefMark('alto')(score[1])
ClefMark('alto')(Staff{ })
abjad> score[1].extend([Note(-5, (1, 4)), Note(-7, (1, 4))])
abjad> contexttools.ClefMark('bass')(score[2])
ClefMark('bass')(Staff{ })
abjad> score[2].append(Note(-24, (1, 2)))
abjad> verticalitytools.label_vertical_moments_in_expr_with_numbered_chromatic_pitch_classes(sco
abjad> f(score)
\new Score <<
        \new Staff {
                c'8
                d'8 _ \markup { \small { \column { 7 2 0 } } }
                e'8
                f'8 _ \markup { \small { \column { 5 0 } } }
        }
        \new Staff {
                \clef "alto"
```

```
                g4
                f4 _ \markup { \small { \column { 5 4 0 } } }
        }
        \new Staff {
                \clef "bass"
                c,2 _ \markup { \small { \column { 7 0 } } }
        }
>>
```

Changed in version 1.1.2: renamed `label.vertical_moment_pitch_classes( )` to `verticalitytools.label_vertical_moments_in_expr_with_numbered_chromatic_pitch_classes( )`.

verticalitytools.label_vertical_moments_in_expr_with_pitch_numbers

abjad.tools.verticalitytools.**label_vertical_moments_in_expr_with_pitch_numbers**(*expr*, *markup_direction*

New in version 1.1.2. Label pitch numbers of every vertical moment in *expr*.

```
abjad> score = Score(Staff([ ]) * 3)
abjad> score[0].extend(macros.scale(4))
abjad> contexttools.ClefMark('alto')(score[1])
ClefMark('alto')(Staff{ })
abjad> score[1].extend([Note(-5, (1, 4)), Note(-7, (1, 4))])
abjad> contexttools.ClefMark('bass')(score[2])
ClefMark('bass')(Staff{ })
abjad> score[2].append(Note(-24, (1, 2)))
abjad> verticalitytools.label_vertical_moments_in_expr_with_pitch_numbers(score)
abjad> f(score)
\new Score <<
        \new Staff {
                c'8
                d'8 _ \markup { \small { \column { 2 -5 -24 } } }
                e'8
                f'8 _ \markup { \small { \column { 5 -7 -24 } } }
        }
        \new Staff {
                \clef "alto"
                g4
                f4 _ \markup { \small { \column { 4 -7 -24 } } }
        }
        \new Staff {
                \clef "bass"
                c,2 _ \markup { \small { \column { 0 -5 -24 } } }
        }
>>
```

Changed in version 1.1.2: renamed `label.vertical_moment_pitch_numbers( )` to `verticalitytools.label_vertical_moments_in_expr_with_pitch_numbers( )`.

voicetools

voicetools.get_first_voice_in_improper_parentage_of_component

abjad.tools.voicetools.**get_first_voice_in_improper_parentage_of_component**(*component*)

New in version 1.1.2. Get first voice in improper parentage of *component*:

```
abjad> voice = Voice("c'8 d'8 e'8 f'8")
abjad> staff = Staff([voice])

abjad> f(staff)
\new Staff {
   \new Voice {
      c'8
      d'8
      e'8
      f'8
   }
}

abjad> voicetools.get_first_voice_in_improper_parentage_of_component(staff.leaves[0])
Voice{4}
```

Return voice or none.

voicetools.get_first_voice_in_proper_parentage_of_component

abjad.tools.voicetools.**get_first_voice_in_proper_parentage_of_component**(*component*)
New in version 1.1.2. Get first voice in proper parentage of *component*:

```
abjad> voice = Voice("c'8 d'8 e'8 f'8")
abjad> staff = Staff([voice])

abjad> f(staff)
\new Staff {
   \new Voice {
      c'8
      d'8
      e'8
      f'8
   }
}

abjad> voicetools.get_first_voice_in_proper_parentage_of_component(staff.leaves[0])
Voice{4}
```

Return voice or none.

voicetools.iterate_semantic_voices_backward_in_expr

abjad.tools.voicetools.**iterate_semantic_voices_backward_in_expr**(*expr*)
New in version 1.1.2. Iterate semantic voices backward in *expr*:

```
abjad> measures = measuretools.make_measures_with_full_measure_spacer_skips([(3, 8), (5, 16), (5
abjad> meter_voice = Voice(measures)
abjad> meter_voice.name = 'MeterVoice'
abjad> meter_voice.is_nonsemantic = True
abjad> music_voice = Voice("c'4. d'4 e'16 f'4 g'16")
abjad> music_voice.name = 'MusicVoice'
abjad> staff = Staff([meter_voice, music_voice])
abjad> staff.is_parallel = True
```

```
abjad> f(staff)
\new Staff <<
   \context Voice = "MeterVoice" {
      {
         \time 3/8
         s1 * 3/8
      }
      {
         \time 5/16
         s1 * 5/16
      }
      {
         \time 5/16
         s1 * 5/16
      }
   }
   \context Voice = "MusicVoice" {
      c'4.
      d'4
      e'16
      f'4
      g'16
   }
>>

abjad> for voice in voicetools.iterate_semantic_voices_backward_in_expr(staff):
...   voice
Voice-"MusicVoice"{5}
```

Return generator.

voicetools.iterate_semantic_voices_forward_in_expr

abjad.tools.voicetools.**iterate_semantic_voices_forward_in_expr**(*expr*)
New in version 1.1.2. Iterate semantic voices forward in *expr*:

```
abjad> measures = measuretools.make_measures_with_full_measure_spacer_skips([(3, 8), (5, 16), (5
abjad> meter_voice = Voice(measures)
abjad> meter_voice.name = 'MeterVoice'
abjad> meter_voice.is_nonsemantic = True
abjad> music_voice = Voice("c'4. d'4 e'16 f'4 g'16")
abjad> music_voice.name = 'MusicVoice'
abjad> staff = Staff([meter_voice, music_voice])
abjad> staff.is_parallel = True

abjad> f(staff)
\new Staff <<
   \context Voice = "MeterVoice" {
      {
         \time 3/8
         s1 * 3/8
      }
      {
         \time 5/16
         s1 * 5/16
      }
      {
```

```
            \time 5/16
            s1 * 5/16
        }
    }
    \context Voice = "MusicVoice" {
        c'4.
        d'4
        e'16
        f'4
        g'16
    }
>>

abjad> for voice in voicetools.iterate_semantic_voices_forward_in_expr(staff):
...    voice
Voice-"MusicVoice"{5}
```

Return generator.

voicetools.iterate_voices_backward_in_expr

`abjad.tools.voicetools.`**`iterate_voices_backward_in_expr`**`(`*expr*`)`

New in version 1.1.2. Iterate voices backward in *expr*:

```
abjad> voice_1 = Voice("c'8 d'8 e'8 f'8")
abjad> voice_2 = Voice("c'4 b4")
abjad> staff = Staff([voice_1, voice_2])
abjad> staff.is_parallel = True

abjad> f(staff)
\new Staff <<
    \new Voice {
        c'8
        d'8
        e'8
        f'8
    }
    \new Voice {
        c'4
        b4
    }
>>

abjad> for voice in voicetools.iterate_voices_backward_in_expr(staff):
...    voice
Voice{2}
Voice{4}
```

Return generator.

voicetools.iterate_voices_forward_in_expr

`abjad.tools.voicetools.`**`iterate_voices_forward_in_expr`**`(`*expr*`)`

New in version 1.1.2. Iterate voices forward in *expr*:

```
abjad> voice_1 = Voice("c'8 d'8 e'8 f'8")
abjad> voice_2 = Voice("c'4 b4")
abjad> staff = Staff([voice_1, voice_2])
abjad> staff.is_parallel = True

abjad> f(staff)
\new Staff <<
   \new Voice {
      c'8
      d'8
      e'8
      f'8
   }
   \new Voice {
      c'4
      b4
   }
>>

abjad> for voice in voicetools.iterate_voices_forward_in_expr(staff):
...   voice
Voice{4}
Voice{2}
```

Return generator.

63.1.3 Additional Abjad composition packages (load manually)

pitcharraytools

pitcharraytools.PitchArray

class `abjad.tools.pitcharraytools.`**`PitchArray`**(**args*)

Bases: `abjad.core._StrictComparator._StrictComparator._StrictComparator` New in version 1.1.2. Two-dimensional array of pitches.

`append_column`(*column*)

`append_row`(*row*)

`apply_pitches_by_row`(*pitch_lists*)

`cell_tokens_by_row`

`cell_widths_by_row`

`cells`

`columns`

`copy_subarray`(*upper_left_pair*, *lower_right_pair*)

`depth`

`dimensions`

`has_spanning_cell_over_index`(*index*)

`has_voice_crossing`

`is_rectangular`

pad_to_depth (*depth*)

pad_to_width (*width*)

pitches

pitches_by_row

pop_column (*column_index*)

pop_row (*row_index=-1*)

remove_row (*row*)

rows

size

voice_crossing_count

weight

width

pitcharraytools.PitchArrayCell

class `abjad.tools.pitcharraytools.`**`PitchArrayCell`**(*cell_token=None*)

Bases: `abjad.core._StrictComparator._StrictComparator._StrictComparator`

One cell in a pitch array.

```
abjad> from abjad.tools import pitcharraytools

abjad> array = pitcharraytools.PitchArray([[1, 2, 1], [2, 1, 1]])
abjad> print array
[ ] [     ] [ ]
[     ] [ ] [ ]
abjad> cell = array[0][1]
abjad> cell
PitchArrayCell(x2)

abjad> cell.column_indices
(1, 2)

abjad> cell.indices
(0, (1, 2))

abjad> cell.is_first_in_row
False

abjad> cell.is_last_in_row
False

abjad> cell.next
PitchArrayCell(x1)

abjad> cell.parent_array
PitchArray(PitchArrayRow(x1, x2, x1), PitchArrayRow(x2, x1, x1))

abjad> cell.parent_column
PitchArrayColumn(x2, x2)
```

```
abjad> cell.parent_row
PitchArrayRow(x1, x2, x1)

abjad> cell.pitches
[]

abjad> cell.prev
PitchArrayCell(x1)

abjad> cell.row_index
0

abjad> cell.token
2

abjad> cell.width
2
```

Return pitch array cell.

column_indices
Read-only tuple of one or more nonnegative integer indices.

indices

is_first_in_row

is_last_in_row

matches_cell (*arg*)

next

parent_array

parent_column

parent_row

pitches

prev

row_index

token

weight

width

pitcharraytools.PitchArrayColumn

class abjad.tools.pitcharraytools.**PitchArrayColumn** (*cells*)
Bases: abjad.core._StrictComparator._StrictComparator._StrictComparator New in version 1.1.2. Column in a pitch array:

```
abjad> from abjad.tools import pitcharraytools

abjad> array = pitcharraytools.PitchArray([
...    [1, (2, 1), (-1.5, 2)],
...    [(7, 2), (6, 1), 1]])
```

```
abjad> print array
[  ] [d'] [bqf    ]
[g'     ] [fs'] [ ]

abjad> array.columns[0]
PitchArrayColumn(x1, g' x2)

abjad> print array.columns[0]
[  ]
[g'     ]
```

Return pitch array column.

append(*cell*)

cell_tokens

cell_widths

cells

column_index

depth

dimensions

extend(*cells*)

has_voice_crossing

is_defective

parent_array

pitches

remove_pitches()

start_cells

start_pitches

stop_cells

stop_pitches

weight

width

pitcharraytools.PitchArrayRow

class abjad.tools.pitcharraytools.**PitchArrayRow**(*cells*)

Bases: abjad.core._StrictComparator._StrictComparator._StrictComparator New in version 1.1.2. One row in pitch array.

```
abjad> from abjad.tools import pitcharraytools

abjad> array = pitcharraytools.PitchArray([[1, 2, 1], [2, 1, 1]])
abjad> array[0].cells[0].pitches.append(0)
abjad> array[0].cells[1].pitches.append(2)
abjad> array[1].cells[2].pitches.append(4)
abjad> print array
```

```
[c'] [d'    ] [  ]
[       ] [ ] [e']

abjad> array[0]
PitchArrayRow(c', d' x2, x1)

abjad> array[0].cell_widths
(1, 2, 1)

abjad> array[0].dimensions
(1, 4)

abjad> array[0].pitches
(NamedChromaticPitch("c'"), NamedChromaticPitch("d'"))
```

Return pitch array row.

append (*cell_token*)

apply_pitches (*pitch_tokens*)

cell_tokens

cell_widths

cells

copy_subrow (*start=None*, *stop=None*)

depth

dimensions

empty_pitches ()

extend (*cell_tokens*)

has_spanning_cell_over_index (*i*)

index (*cell*)

is_defective

is_in_range

merge (*cells*)

pad_to_width (*width*)

parent_array

pitch_range

pitches

pop (*cell_index*)

remove (*cell*)

row_index

weight

width

withdraw ()

pitcharraytools.concatenate_pitch_arrays

abjad.tools.pitcharraytools.**concatenate_pitch_arrays**(*pitch_arrays*)

New in version 1.1.2. Concatenate *pitch_arrays*:

```
abjad> from abjad.tools import pitcharraytools

abjad> array_1 = pitcharraytools.PitchArray([[1, 2, 1], [2, 1, 1]])
abjad> print array_1
[ ] [     ] [ ]
[     ] [ ] [ ]

abjad> array_2 = pitcharraytools.PitchArray([[3, 4], [4, 3]])
abjad> print array_2
[     ] [         ]
[         ] [     ]

abjad> array_3 = pitcharraytools.PitchArray([[1, 1], [1, 1]])
abjad> print array_3
[ ] [ ]
[ ] [ ]

abjad> merged_array = pitcharraytools.concatenate_pitch_arrays([array_1, array_2, array_3])
abjad> print merged_array
[ ] [     ] [ ] [     ] [         ] [ ] [ ]
[     ] [ ] [ ] [         ] [     ] [ ] [ ]
```

Return pitch array.

pitcharraytools.list_nonspanning_subarrays_of_pitch_array

abjad.tools.pitcharraytools.**list_nonspanning_subarrays_of_pitch_array**(*pitch_array*)

New in version 1.1.2. List nonspanning subarrays of *pitch_array*:

```
abjad> from abjad.tools import pitcharraytools

abjad> array = pitcharraytools.PitchArray([
...     [2, 2, 3, 1],
...     [1, 2, 1, 1, 2, 1],
...     [1, 1, 1, 1, 1, 1, 1, 1]])
abjad> print array
[     ] [     ] [         ] [ ]
[ ] [     ] [ ] [ ] [     ] [ ]
[ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]

abjad> subarrays = pitcharraytools.list_nonspanning_subarrays_of_pitch_array(array)
abjad> len(subarrays)
3

abjad> print subarrays[0]
[     ] [     ]
[ ] [     ] [ ]
[ ] [ ] [ ] [ ]

abjad> print subarrays[1]
[         ]
[ ] [     ]
[ ] [ ] [ ]
```

```
abjad> print subarrays[2]
[ ]
[ ]
[ ]
```

Return list.

pitcharraytools.make_empty_pitch_array_from_list_of_pitch_lists

abjad.tools.pitcharraytools.**make_empty_pitch_array_from_list_of_pitch_lists**(*leaf_iterables*)

New in version 1.1.2. Make empty pitch array from *leaf_iterables*:

```
abjad> from abjad.tools import pitcharraytools

abjad> score = Score([ ])
abjad> score.append(Staff(macros.scale(4)))
abjad> score.append(Staff(macros.scale(2, Duration(1, 4))))
abjad> score.append(Staff(tuplettools.FixedDurationTuplet((2, 8), macros.scale(3)) * 2))
abjad> f(score)
\new Score <<
        \new Staff {
                c'8
                d'8
                e'8
                f'8
        }
        \new Staff {
                c'4
                d'4
        }
        \new Staff {
                \times 2/3 {
                        c'8
                        d'8
                        e'8
                }
                \times 2/3 {
                        c'8
                        d'8
                        e'8
                }
        }
>>

abjad> array = pitcharraytools.make_empty_pitch_array_from_list_of_pitch_lists(score)
abjad> print array
[     ] [     ] [     ] [     ]
[             ] [             ]
[ ] [     ] [ ] [ ] [     ] [ ]
```

Return pitch array.

pitcharraytools.make_populated_pitch_array_from_list_of_pitch_lists

abjad.tools.pitcharraytools.**make_populated_pitch_array_from_list_of_pitch_lists**(*leaf_iterables*)

New in version 1.1.2. Make populated pitch array from *leaf_iterables*:

```
abjad> from abjad.tools import pitcharraytools

abjad> score = Score([ ])
abjad> score.append(Staff(macros.scale(4)))
abjad> score.append(Staff(macros.scale(2, Duration(1, 4))))
abjad> score.append(Staff(tuplettools.FixedDurationTuplet((2, 8), macros.scale(3)) * 2))
abjad> f(score)
\new Score <<
        \new Staff {
                c'8
                d'8
                e'8
                f'8
        }
        \new Staff {
                c'4
                d'4
        }
        \new Staff {
                \times 2/3 {
                        c'8
                        d'8
                        e'8
                }
                \times 2/3 {
                        c'8
                        d'8
                        e'8
                }
        }
>>

abjad> array = pitcharraytools.make_populated_pitch_array_from_list_of_pitch_lists(score)
abjad> print array
[c'     ] [d'     ] [e'     ] [f'     ]
[c'               ] [d'               ]
[c'] [d'     ] [e'] [c'] [d'     ] [e']
```

Return pitch array.

sievetools

sievetools.ResidueClass

class `abjad.tools.sievetools.`**`ResidueClass`**(**args*)

Bases: `abjad.tools.sievetools._BaseResidueClass._BaseResidueClass._BaseResidueClass`, `abjad.core._Immutable._Immutable._Immutable`

Residue class (or congruence class). Residue classes form the basis of Xenakis sieves. They can be used to construct any complex periodic integer (or boolean) sequence as a combination of simple periodic sequences.

Example from the opening of Xenakis's *Psappha* for solo percussion:

```
abjad> from abjad.tools.sievetools import ResidueClass as RC

abjad> s1 = (RC(8, 0) | RC(8, 1) | RC(8, 7)) & (RC(5, 1) | RC(5, 3))
abjad> s2 = (RC(8, 0) | RC(8, 1) | RC(8, 2)) & RC(5, 0)
abjad> s3 = RC(8, 3)
```

```
abjad> s4 = RC(8, 4)
abjad> s5 = (RC(8, 5) | RC(8, 6)) & (RC(5, 2) | RC(5, 3) | RC(5, 4))
abjad> s6 = (RC(8, 1) & RC(5, 2))
abjad> s7 = (RC(8, 6) & RC(5, 1))

abjad> y = s1 | s2 | s3 | s4 | s5 | s6 | s7
abjad> y
{{{ResidueClass(8, 0) | ResidueClass(8, 1) | ResidueClass(8, 7)} & {ResidueClass(5, 1) | Residue

abjad> y.get_congruent_bases(40)
   [0, 1, 3, 4, 6, 8, 10, 11, 12, 13, 14, 16, 17, 19, 20, 22, 23, 25, 27,
   28, 29, 31, 33, 35, 36, 37, 38, 40]
abjad> y.get_boolean_train(40)
   [1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0,
   1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0]
```

Return residue class.

get_boolean_train (**min_max*)

Returns a boolean train with 0s mapped to the integers that are not congruent bases of the residue class and 1s mapped to those that are. The method takes one or two integer arguments. If only one is given, it is taken as the max range and the min is assumed to be 0.

Example:

```
abjad> from abjad.tools.sievetools import ResidueClass as RC

abjad> r = RC(3, 0)
abjad> r.get_boolean_train(6)
[1, 0, 0, 1, 0, 0]
abjad> r.get_congruent_bases(-6, 6)
[-6, -3, 0, 3, 6]
```

Return list.

get_congruent_bases (**min_max*)

Returns all the congruent bases of this residue class within the given range. The method takes one or two integer arguments. If only one it given, it is taken as the max range and the min is assumed to be 0.

Example:

```
abjad> from abjad.tools.sievetools import ResidueClass as RC

abjad> r = RC(3, 0)
abjad> r.get_congruent_bases(6)
[0, 3, 6]
abjad> r.get_congruent_bases(-6, 6)
[-6, -3, 0, 3, 6]
```

Return list.

modulo

Period of residue class.

residue

Residue of residue class.

sievetools.ResidueClassExpression

class abjad.tools.sievetools.**ResidueClassExpression**(*rcs*, *operator='or'*)
Bases: abjad.tools.sievetools._BaseResidueClass._BaseResidueClass._BaseResidueClass, abjad.core._Immutable._Immutable._Immutable

get_boolean_train(**min_max*)
Returns a boolean train with 0s mapped to the integers that are not congruent bases of the RC expression and 1s mapped to those that are. The method takes one or two integer arguments. If only one is given, it is taken as the max range and min is assumed to be 0.

Example:

```
abjad> from abjad.tools.sievetools import ResidueClass as RC

abjad> e = RC(3, 0) | RC(2, 0)
abjad> e.get_boolean_train(6)
[1, 0, 1, 1, 1, 0]
abjad> e.get_congruent_bases(-6, 6)
[-6, -4, -3, -2, 0, 2, 3, 4, 6]
```

Return list.

get_congruent_bases(**min_max*)
Returns all the congruent bases of this RC expression within the given range. The method takes one or two integer arguments. If only one it given, it is taken as the max range and min is assumed to be 0.

Example:

```
abjad> from abjad.tools.sievetools import ResidueClass as RC

abjad> e = RC(3, 0) | RC(2, 0)
abjad> e.get_congruent_bases(6)
[0, 2, 3, 4, 6]
abjad> e.get_congruent_bases(-6, 6)
[-6, -4, -3, -2, 0, 2, 3, 4, 6]
```

Return list.

is_congruent_base(*integer*)

operator
Operator of residue class expression.

period

rcs
Residue classes of expression.

representative_boolean_train

representative_congruent_bases

sievetools.cycle_tokens_to_sieve

abjad.tools.sievetools.**cycle_tokens_to_sieve**(**cycle_tokens*)
New in version 1.1.2. Make Xenakis sieve from arbitrarily many *cycle_tokens*.

```
abjad> from abjad.tools import sievetools
```

```
abjad> cycle_token_1 = (6, [0, 4, 5])
abjad> cycle_token_2 = (10, [0, 1, 2], 6)
abjad> sievetools.cycle_tokens_to_sieve(cycle_token_1, cycle_token_2)
{ResidueClass(6, 0) | ResidueClass(6, 4) | ResidueClass(6, 5) | ResidueClass(10, 6) | ResidueCla
```

Cycle token comprises mandatory *modulo*, mandatory *residues* and optional *offset*.

tonalitytools

tonalitytools.ChordClass

class `abjad.tools.tonalitytools.`**`ChordClass`**

Bases: `abjad.tools.pitchtools.NamedChromaticPitchClassSet.NamedChromaticPitchClassSet.Nar`

New in version 1.1.2. Abjad model of tonal chords like G 7, G 6/5, G half-diminished 6/5, etc.

Note that notions like G 7 represent an entire *class of* chords because there are many different spacings and registrations of a G 7 chord.

`bass`

`cardinality`

`extent`

`figured_bass`

`inversion`

`markup`

`quality_indicator`

`quality_pair`

`root`

`root_string`

`transpose`(*mdi*)

tonalitytools.ChordQualityIndicator

class `abjad.tools.tonalitytools.`**`ChordQualityIndicator`**

Bases: `abjad.tools.pitchtools.HarmonicDiatonicIntervalSegment.HarmonicDiatonicIntervalSeg`

New in version 1.1.2. Chord quality indicator.

`cardinality`

`extent`

`extent_name`

`inversion`

`position`

`quality_string`

`rotation`

tonalitytools.DoublingIndicator

class `abjad.tools.tonalitytools.`**`DoublingIndicator`**(*doublings*)

Bases: `abjad.core._Immutable._Immutable._Immutable` New in version 1.1.2. Indicator of chord doubling.

Value object that can not be changed after instantiation.

`doublings`

tonalitytools.ExtentIndicator

class `abjad.tools.tonalitytools.`**`ExtentIndicator`**(*arg*)

Bases: `abjad.core._Immutable._Immutable._Immutable` New in version 1.1.2. Indicator of chord extent, such as triad, seventh chord, ninth chord, etc.

Value object that can not be changed after instantiation.

`name`

`number`

tonalitytools.InversionIndicator

class `abjad.tools.tonalitytools.`**`InversionIndicator`**(*arg=0*)

Bases: `abjad.core._Immutable._Immutable._Immutable` New in version 1.1.2. Indicator of the inversion of tertian chords: 5, 63, 64 and also 7, 65, 43, 42, etc. Also root position, first, second, third inversions, etc.

Value object that can not be changed once initialized.

`extent_to_figured_bass_string`(*extent*)

`name`

`number`

`title`

tonalitytools.Mode

class `abjad.tools.tonalitytools.`**`Mode`**(*arg*)

Bases: `abjad.core._Immutable._Immutable._Immutable` New in version 1.1.2. Diatonic mode. Can be extended for nondiatonic mode.

Modes with different ascending and descending forms not yet implemented.

`melodic_diatonic_interval_segment`

`mode_name_string`

tonalitytools.OmissionIndicator

class `abjad.tools.tonalitytools.`**`OmissionIndicator`**

Bases: `abjad.core._Immutable._Immutable._Immutable` New in version 1.1.2. Indicator of missing chord tones.

Value object that can not be chnaged after instantiation.

tonalitytools.QualityIndicator

class `abjad.tools.tonalitytools.`**`QualityIndicator`**(*quality_string*)

Bases: `abjad.core._Immutable._Immutable._Immutable` New in version 1.1.2. Indicator of chord quality, such as major, minor, dominant, diminished, etc.

Value object that can not be changed after instantiation.

`is_uppercase`

`quality_string`

tonalitytools.Scale

class `abjad.tools.tonalitytools.`**`Scale`**

Bases: `abjad.tools.pitchtools.NamedChromaticPitchClassSegment.NamedChromaticPitchClassSeg`
New in version 1.1.2. Abjad model of diatonic scale.

`diatonic_interval_class_segment`

`dominant`

`key_signature`

`leading_tone`

`mediant`

`named_chromatic_pitch_class_to_scale_degree`(**args*)

`scale_degree_to_named_chromatic_pitch_class`(**args*)

`subdominant`

`submediant`

`superdominant`

`tonic`

tonalitytools.ScaleDegree

class `abjad.tools.tonalitytools.`**`ScaleDegree`**(**args*)

Bases: `abjad.core._Immutable._Immutable._Immutable` New in version 1.1.2. Abjad model of diatonic scale degrees 1, 2, 3, 4, 5, 6, 7 and also chromatic alterations including flat-2, flat-3, flat-6, etc.

`accidental`

Read-only accidental applied to scale degree.

`apply_accidental`(*accidental*)

Apply accidental to self and emit new instance.

`name`

Read-only name of scale degree.

`number`

Read-only number of diatonic scale degree from 1 to 7, inclusive.

`roman_numeral_string`

`symbolic_string`

`title_string`

tonalitytools.SuspensionIndicator

class `abjad.tools.tonalitytools.`**`SuspensionIndicator`**`(`**args*`)`

Bases: `abjad.core._Immutable._Immutable._Immutable` New in version 1.1.2. Indicator of 9-8, 7-6, 4-3, 2-1 and other types of suspension typical of, for example, the Bach chorales.

Value object that can not be changed after instantiation.

`chord_name_string`

`figured_bass_pair`

`figured_bass_string`

`is_empty`

`start`

`stop`

`title_string`

tonalitytools.TonalFunction

class `abjad.tools.tonalitytools.`**`TonalFunction`**`(`**args*`)`

Bases: `abjad.core._Immutable._Immutable._Immutable` New in version 1.1.2. Abjad model of functions in tonal harmony: I, I6, I64, V, V7, V43, V42, bII, bII6, etc., also i, i6, i64, v, v7, etc.

Value object that can not be cahnged after instantiation.

`bass_scale_degree`

`extent`

`figured_bass_string`

`inversion`

`markup`

`quality`

`root_scale_degree`

`scale_degree`

`suspension`

`symbolic_string`

tonalitytools.analyze_chord

`abjad.tools.tonalitytools.`**`analyze_chord`**`(`*expr*`)`

New in version 1.1.2. Analyze *expr* and return chord class.

```
abjad> from abjad.tools import tonalitytools

abjad> chord = Chord([7, 10, 12, 16], (1, 4))
abjad> tonalitytools.analyze_chord(chord)
CDominantSeventhInSecondInversion
```

Return none when no tonal chord is understood.

```
abjad> chord = Chord(['c', 'cs', 'd'], (1, 4))
abjad> tonalitytools.analyze_chord(chord) is None
True
```

Raise tonal harmony error when chord can not analyze.

tonalitytools.analyze_incomplete_chord

abjad.tools.tonalitytools.**analyze_incomplete_chord**(*expr*)

New in version 1.1.2. Analyze *expr* and return chord class based on incomplete pitches.

```
abjad> from abjad.tools import tonalitytools

abjad> tonalitytools.analyze_incomplete_chord(Chord([7, 11], (1, 4)))
GMajorTriadInRootPosition

abjad> tonalitytools.analyze_incomplete_chord(Chord(['fs', 'g', 'b'], (1, 4)))
GMajorSeventhInSecondInversion
```

Return chord class.

tonalitytools.analyze_incomplete_tonal_function

abjad.tools.tonalitytools.**analyze_incomplete_tonal_function**(*expr*, *key_signature*)

New in version 1.1.2. Analyze tonal function of *expr* according to *key_signature*:

```
abjad> from abjad.tools import tonalitytools

abjad> chord = Chord("<c' e'>4")
abjad> key_signature = contexttools.KeySignatureMark('g', 'major')
abjad> tonalitytools.analyze_incomplete_tonal_function(chord, key_signature)
IVMajorTriadInRootPosition
```

Return tonal function.

tonalitytools.analyze_tonal_function

abjad.tools.tonalitytools.**analyze_tonal_function**(*expr*, *key_signature*)

New in version 1.1.2. Analyze *expr* and return tonal function according to *key_signature*.

```
abjad> from abjad.tools import tonalitytools

abjad> chord = Chord(['ef', 'g', 'bf'], (1, 4))
abjad> key_signature = contexttools.KeySignatureMark('c', 'major')
abjad> tonalitytools.analyze_tonal_function(chord, key_signature)
FlatIIIMajorTriadInRootPosition
```

Return none when no tonal function is understood.

```
abjad> chord = Chord(['c', 'cs', 'd'], (1, 4))
abjad> key_signature = contexttools.KeySignatureMark('c', 'major')
abjad> tonalitytools.analyze_tonal_function(chord, key_signature) is None
True
```

Return tonal function or none.

tonalitytools.are_scalar_notes

`abjad.tools.tonalitytools.`**`are_scalar_notes`**`(*expr)`

New in version 1.1.2. True when notes in *expr* are scalar.

```
abjad> from abjad.tools import tonalitytools

abjad> t = Staff(macros.scale(4))
abjad> tonalitytools.are_scalar_notes(t[:])
True
```

Otherwise false.

```
abjad> tonalitytools.are_scalar_notes(Note(0, (1, 4)), Note(0, (1, 4)))
False
```

Changed in version 1.1.2: renamed `tonalitytools.are_scalar( )` to `tonalitytools.are_scalar_notes( )`.

tonalitytools.are_stepwise_ascending_notes

`abjad.tools.tonalitytools.`**`are_stepwise_ascending_notes`**`(*expr)`

New in version 1.1.2. True when notes in *expr* are stepwise ascneding.

```
abjad> from abjad.tools import tonalitytools

abjad> t = Staff(macros.scale(4))
abjad> tonalitytools.are_stepwise_ascending_notes(t[:])
True
```

Otherwise false.

```
abjad> tonalitytools.are_stepwise_ascending_notes(Note(0, (1, 4)), Note(0, (1, 4)))
False
```

Changed in version 1.1.2: renamed `tonalitytools.are_stepwise_ascending( )` to `tonalitytools.are_stepwise_ascending_notes( )`.

tonalitytools.are_stepwise_descending_notes

`abjad.tools.tonalitytools.`**`are_stepwise_descending_notes`**`(*expr)`

New in version 1.1.2. True when notes in *expr* are stepwise descending.

```
abjad> from abjad.tools import tonalitytools

abjad> t = Staff(list(reversed(macros.scale(4))))
abjad> tonalitytools.are_stepwise_descending_notes(t[:])
True
```

Otherwise false.

```
abjad> tonalitytools.are_stepwise_descending_notes(Note(0, (1, 4)), Note(0, (1, 4)))
False
```

Changed in version 1.1.2: renamed `tonalitytools.are_stepwise_descending( )` to `tonalitytools.are_stepwise_descending_notes( )`.

tonalitytools.are_stepwise_notes

`abjad.tools.tonalitytools.`**`are_stepwise_notes`**`(*expr)`

New in version 1.1.2. True when notes in *expr* are stepwise.

```
abjad> from abjad.tools import tonalitytools

abjad> t = Staff(macros.scale(4))
abjad> tonalitytools.are_stepwise_notes(t[:])
True
```

Otherwise false.

```
abjad> tonalitytools.are_stepwise_notes(Note(0, (1, 4)), Note(0, (1, 4)))
False
```

Changed in version 1.1.2: renamed `tonalitytools.are_stepwise( )` to `tonalitytools.are_stepwise_notes( )`.

tonalitytools.chord_class_cardinality_to_extent

`abjad.tools.tonalitytools.`**`chord_class_cardinality_to_extent`**`(cardinality)`

..versionadded:: 1.1.2

Change integer chord class *cardinality* to integer chord class extent:

```
abjad> from abjad.tools import tonalitytools

abjad> tonalitytools.chord_class_cardinality_to_extent(4)
7
```

The function above indicates that a tertian chord with 4 unique pitches qualifies as a seventh chord.

tonalitytools.chord_class_extent_to_cardinality

`abjad.tools.tonalitytools.`**`chord_class_extent_to_cardinality`**`(extent)`

..versionadded:: 1.1.2

Change integer chord class *extent* to integer chord class cardinality:

```
abjad> from abjad.tools import tonalitytools

abjad> tonalitytools.chord_class_extent_to_cardinality(7)
4
```

The call above shows that a seventh chord comprises 4 unique pitch-classes.

tonalitytools.chord_class_extent_to_extent_name

`abjad.tools.tonalitytools.`**`chord_class_extent_to_extent_name`**`(extent)`

New in version 1.1.2. Change integer chord class *extent* to extent name string.

```
abjad> from abjad.tools import tonalitytools

abjad> tonalitytools.chord_class_extent_to_extent_name(7)
'seventh'
```

The call above shows that a tertian chord subtending 7 staff spaces qualifies as a seventh chord.

tonalitytools.diatonic_interval_class_segment_to_chord_quality_string

abjad.tools.tonalitytools.**diatonic_interval_class_segment_to_chord_quality_string**(*dic_seg*)

New in version 1.1.2. Change diatonic interval-class segment *dic_seg* to chord quality string:

```
abjad> from abjad.tools import tonalitytools

abjad> dic_seg = pitchtools.InversionEquivalentDiatonicIntervalClassSegment([
...   pitchtools.InversionEquivalentDiatonicIntervalClass('major', 3),
...   pitchtools.InversionEquivalentDiatonicIntervalClass('minor', 3),])
abjad> tonalitytools.diatonic_interval_class_segment_to_chord_quality_string(dic_seg)
'major'
```

Todo

Implement `diatonic_interval_class_set_to_chord_quality_string( )`.

tonalitytools.is_neighbor_note

abjad.tools.tonalitytools.**is_neighbor_note**(*note*)

New in version 1.1.2. True when *note* is preceeded by a stepwise interval in one direction and followed by a stepwise interval in the other direction. Otherwise false.

```
abjad> from abjad.tools import tonalitytools

abjad> t = Staff(macros.scale(4))
abjad> for note in t:
...     print '%s\t%s' % (note, tonalitytools.is_neighbor_note(note))
...
c'8     False
d'8     False
e'8     False
f'8     False
```

Return boolean.

tonalitytools.is_passing_tone

abjad.tools.tonalitytools.**is_passing_tone**(*note*)

New in version 1.1.2. True when *note* is both preceeded and followed by scalewise sibling notes. Otherwise false.

```
abjad> from abjad.tools import tonalitytools

abjad> t = Staff(macros.scale(4))
abjad> for note in t:
...     print '%s\t%s' % (note, tonalitytools.is_passing_tone(note))
...
c'8     False
d'8     True
e'8     True
f'8     False
```

Return boolean.

tonalitytools.is_unlikely_melodic_diatonic_interval_in_chorale

abjad.tools.tonalitytools.**is_unlikely_melodic_diatonic_interval_in_chorale**(*mdi*)

New in version 1.1.2. True when *mdi* is unlikely melodic diatonic interval in JSB chorale.

```
abjad> from abjad.tools import tonalitytools

abjad> mdi = pitchtools.MelodicDiatonicInterval('major', 7)
abjad> tonalitytools.is_unlikely_melodic_diatonic_interval_in_chorale(mdi)
True
```

Otherwise False.

```
abjad> mdi = pitchtools.MelodicDiatonicInterval('major', 2)
abjad> tonalitytools.is_unlikely_melodic_diatonic_interval_in_chorale(mdi)
False
```

Return boolean.

tonalitytools.make_all_notes_in_ascending_and_descending_diatonic_scale

abjad.tools.tonalitytools.**make_all_notes_in_ascending_and_descending_diatonic_scale**(*key_signa*

New in version 1.1.2. Construct one up-down period of scale according to *key_signature*:

```
abjad> from abjad.tools import tonalitytools

abjad> score = tonalitytools.make_all_notes_in_ascending_and_descending_diatonic_scale(contextto
abjad> f(score)
\new Score \with {
	tempoWholesPerMinute = #(ly:make-moment 30 1)
} <<
	\new Staff {
		\key e \major
		e'8
		fs'8
		gs'8
		a'8
		b'8
		cs''8
		ds''8
		e''8
		ds''8
		cs''8
		b'8
		a'8
		gs'8
		fs'8
		e'4
	}
>>
```

Changed in version 1.1.2: renamed `construct.scale_period( )` to `tonalitytools.make_all_notes_in_ascending_and_descending_diatonic_scale( )`.Changed in version 1.1.2: renamed `leaftools.make_all_notes_in_ascending_and_descending_diatonic_` `)` to `tonalitytools.make_all_notes_in_ascending_and_descending_diatonic_scale( )`.

tonalitytools.make_first_n_notes_in_ascending_diatonic_scale

`abjad.tools.tonalitytools.`**`make_first_n_notes_in_ascending_diatonic_scale`**(*count*, *written_duration=Duration(1, 8)*, *key_signature=None*)

Construct *count* notes with *written_duration* according to *key_signature*:

```
abjad> macros.scale(4)
[Note("c'8"), Note("d'8"), Note("e'8"), Note("f'8")]
```

Allow nonassignable *written_duration*:

```
abjad> staff = Staff(macros.scale(2, (5, 16)))
abjad> f(staff)
\new Staff {
        c'4 ~
        c'16
        d'4 ~
        d'16
}
```

New in version 1.1.2: Optional *key_signature* keyword parameter.Changed in version 1.1.2: renamed `construct.scale( )` to `macros.scale( )`.Changed in version 1.1.2: renamed `leaftools.make_first_n_notes_in_ascending_diatonic_scale( )` to `tonalitytools.make_first_n_notes_in_ascending_diatonic_scale( )`.

treetools

treetools.Block

class `abjad.tools.treetools.`**`Block`**(**args*, ***kwargs*)

Bases: `abjad.tools.treetools.BoundedInterval.BoundedInterval.BoundedInterval`

An abstract block of musical material occupying some amount of time.

`duration`

`start_offset`

`stop_offset`

treetools.BoundedInterval

class `abjad.tools.treetools.`**`BoundedInterval`**(**args*, ***kwargs*)

Bases: `abjad.core._Immutable._Immutable._Immutable`

A low / high pair, carrying some metadata.

`centroid`

Center point of low and high bounds.

`data`

Payload.

`get_overlap_with_interval`(*interval*)

Return amount of overlap with *interval*.

high
High bound.

is_contained_by_interval (*interval*)
True if interval is contained by *interval*.

is_container_of_interval (*interval*)
True if interval contains *interval*.

is_overlapped_by_interval (*interval*)
True if interval is overlapped by *interval*.

is_tangent_to_interval (*interval*)
True if interval is tangent to *interval*.

low
Low bound.

magnitude
High bound minus low bound.

scale_by_rational (*rational*)

scale_to_rational (*rational*)

shift_by_rational (*rational*)

shift_to_rational (*rational*)

signature
Tuple of low bound and high bound.

split_at_rational (*rational*)

treetools.IntervalTree

class `abjad.tools.treetools.`**IntervalTree** (*intervals*=[ ])
Bases: `abjad.tools.treetools._RedBlackTree._RedBlackTree._RedBlackTree`

An augmented red-black tree for storing and searching for intervals of time. Allows for the arbitrary placement of blocks of material along a time-line. While this functionality could be achieved with Python's built-in collections, this class reduces the complexity of the search process, such as locating overlapping intervals.

bounds

find_intervals_intersecting_or_tangent_to_interval (**args*)

find_intervals_intersecting_or_tangent_to_offset (*offset*)

find_intervals_starting_after_offset (*offset*)

find_intervals_starting_and_stopping_within_interval (**args*)

find_intervals_starting_at_offset (*offset*)

find_intervals_starting_before_offset (*offset*)

find_intervals_starting_or_stopping_at_offset (*offset*)

find_intervals_starting_within_interval (**args*)

find_intervals_stopping_after_offset (*offset*)

find_intervals_stopping_at_offset (*offset*)

find_intervals_stopping_before_offset (*offset*)

find_intervals_stopping_within_interval(**args*)

high

high_max

high_min

low

low_max

low_min

magnitude

treetools.all_are_intervals_or_trees_or_empty

abjad.tools.treetools.**all_are_intervals_or_trees_or_empty**(*input*)
Recursively test if all elements of *input* are BoundedIntervals or IntervalTrees. An empty result also return as True.

treetools.all_interval_payloads_contain_key_of_klass

abjad.tools.treetools.**all_interval_payloads_contain_key_of_klass**(*intervals*, *key*, *klass*)
True if all intervals in *intervals* use a dictionary as their payload, have a key named *key* in that dictionary, and the key's value is an instance of *klass*.

treetools.all_intervals_are_contiguous

abjad.tools.treetools.**all_intervals_are_contiguous**(*intervals*)
True when all intervals in *intervals* are contiguous and non-overlapping.

treetools.all_intervals_are_nonoverlapping

abjad.tools.treetools.**all_intervals_are_nonoverlapping**(*intervals*)
True when all intervals in *intervals* in tree are non-overlapping.

treetools.calculate_density_of_attacks_in_interval

abjad.tools.treetools.**calculate_density_of_attacks_in_interval**(*intervals*, *interval*)
Return a Fraction of number of attacks in *interval* over the magnitude of *interval*.

treetools.calculate_density_of_releases_in_interval

abjad.tools.treetools.**calculate_density_of_releases_in_interval**(*intervals*, *interval*)
Return a Fraction of the number of releases in *interval* divided by the magnitude of *interval*.

treetools.calculate_depth_centroid_of_intervals

`abjad.tools.treetools.`**`calculate_depth_centroid_of_intervals`**(*intervals*)

Return a weighted mean, such that the centroids of each interval in the depth tree of *intervals* are the values, and the depth of each interval in the depth tree of *intervals* are the weights.

treetools.calculate_depth_centroid_of_intervals_in_interval

`abjad.tools.treetools.`**`calculate_depth_centroid_of_intervals_in_interval`**(*intervals*, *interval*)

Return the weighted mean of the depth tree of *intervals* in *interval*, such that the centroids of each interval of the depth tree are the values, and the weights are the depths at each interval of the depth tree.

treetools.calculate_depth_density_of_intervals

`abjad.tools.treetools.`**`calculate_depth_density_of_intervals`**(*intervals*)

Return a Fraction, of the magnitude of each interval in the depth tree of *intervals*, multiplied by the depth at that interval, divided by the overall magnitude of *intervals*.

The depth density of a single interval is 1

```
abjad> from abjad.tools import treetools
abjad> from abjad.tools.treetools import BoundedInterval
abjad> from abjad.tools.treetools import IntervalTree

abjad> a = BoundedInterval(0, 1)
abjad> b = BoundedInterval(0, 1)
abjad> c = BoundedInterval(Fraction(1, 2), 1)
abjad> treetools.calculate_depth_density_of_intervals(a)
Fraction(1, 1)
abjad> treetools.calculate_depth_density_of_intervals([a, b])
Fraction(2, 1)
abjad> treetools.calculate_depth_density_of_intervals([a, c])
Fraction(3, 2)
abjad> treetools.calculate_depth_density_of_intervals([a, b, c])
Fraction(5, 2)
```

Return fraction.

treetools.calculate_depth_density_of_intervals_in_interval

`abjad.tools.treetools.`**`calculate_depth_density_of_intervals_in_interval`**(*intervals*, *interval*)

Return a Fraction, of the magnitude of each interval in the depth tree of *intervals* within *interval*, multiplied by the depth at that interval, divided by the overall magnitude of *intervals*.

treetools.calculate_mean_attack_of_intervals

`abjad.tools.treetools.`**`calculate_mean_attack_of_intervals`**(*intervals*)

Return Fraction of the average attack offset of *intervals*

treetools.calculate_mean_release_of_intervals

abjad.tools.treetools.**calculate_mean_release_of_intervals**(*intervals*)
Return a Fraction of the average release offset of *intervals*.

treetools.calculate_min_mean_and_max_depth_of_intervals

abjad.tools.treetools.**calculate_min_mean_and_max_depth_of_intervals**(*intervals*)
Return a 3-tuple of the minimum, mean and maximum depth of *intervals*. If *intervals* is empty, return None. "Mean" in this case is a weighted mean, where the magnitudes of the intervals in depth tree of *intervals* are the weights

treetools.calculate_min_mean_and_max_magnitude_of_intervals

abjad.tools.treetools.**calculate_min_mean_and_max_magnitude_of_intervals**(*intervals*)
Return a 3-tuple of the minimum, mean and maximum magnitude of all intervals in *intervals*. If *intervals* is empty, return None.

treetools.calculate_sustain_centroid_of_intervals

abjad.tools.treetools.**calculate_sustain_centroid_of_intervals**(*intervals*)
Return a weighted mean, such that the centroid of each interval in *intervals* are the values, and the weights are their magnitudes.

treetools.clip_interval_magnitudes_to_range

abjad.tools.treetools.**clip_interval_magnitudes_to_range**(*intervals*, *min=None*, *max=None*)

treetools.compute_depth_of_intervals

abjad.tools.treetools.**compute_depth_of_intervals**(*intervals*)
Compute a tree whose intervals represent the depth (level of overlap) in each boundary pair of *intervals*:

```
abjad> from abjad.tools.treetools import *
abjad> a = BoundedInterval(0, 3)
abjad> b = BoundedInterval(6, 12)
abjad> c = BoundedInterval(9, 15)
abjad> tree = IntervalTree([a, b, c])
abjad> compute_depth_of_intervals(tree)
IntervalTree([
   BoundedInterval(0, 3, {'depth': 1}),
   BoundedInterval(3, 6, {'depth': 0}),
   BoundedInterval(6, 9, {'depth': 1}),
   BoundedInterval(9, 12, {'depth': 2}),
   BoundedInterval(12, 15, {'depth': 1})
])
```

Return interval tree.

treetools.compute_depth_of_intervals_in_interval

abjad.tools.treetools.**compute_depth_of_intervals_in_interval**(*intervals*, *interval*)

Compute a tree whose intervals represent the depth (level of overlap) in each boundary pair of *intervals*, cropped within *interval*:

```
abjad> from abjad.tools.treetools import *
abjad> a = BoundedInterval(0, 3)
abjad> b = BoundedInterval(6, 12)
abjad> c = BoundedInterval(9, 15)
abjad> tree = IntervalTree([a, b, c])
abjad> d = BoundedInterval(-1, 16)
abjad> compute_depth_of_intervals_in_interval(tree, d)
IntervalTree([
   BoundedInterval(-1, 0, {'depth': 0}),
   BoundedInterval(0, 3, {'depth': 1}),
   BoundedInterval(3, 6, {'depth': 0}),
   BoundedInterval(6, 9, {'depth': 1}),
   BoundedInterval(9, 12, {'depth': 2}),
   BoundedInterval(12, 15, {'depth': 1}),
   BoundedInterval(15, 16, {'depth': 0})
])
```

Return interval tree.

treetools.compute_logical_and_of_intervals

abjad.tools.treetools.**compute_logical_and_of_intervals**(*intervals*)

Compute the logical AND of a collection of intervals.

treetools.compute_logical_and_of_intervals_in_interval

abjad.tools.treetools.**compute_logical_and_of_intervals_in_interval**(*intervals*, *interval*)

Compute the logical AND of a collection of intervals, cropped within *interval*.

treetools.compute_logical_not_of_intervals

abjad.tools.treetools.**compute_logical_not_of_intervals**(*intervals*)

Compute the logical NOT of some collection of intervals.

treetools.compute_logical_not_of_intervals_in_interval

abjad.tools.treetools.**compute_logical_not_of_intervals_in_interval**(*intervals*, *interval*)

Compute the logical NOT of some collection of intervals, cropped within *interval*.

treetools.compute_logical_or_of_intervals

abjad.tools.treetools.**compute_logical_or_of_intervals**(*intervals*)

Compute the logical OR of a collection of intervals.

treetools.compute_logical_or_of_intervals_in_interval

`abjad.tools.treetools.`**`compute_logical_or_of_intervals_in_interval`**(*intervals*, *interval*)

Compute the logical OR of a collection of intervals, cropped within *interval*.

treetools.compute_logical_xor_of_intervals

`abjad.tools.treetools.`**`compute_logical_xor_of_intervals`**(*intervals*)

Compute the logical XOR of a collections of intervals.

treetools.compute_logical_xor_of_intervals_in_interval

`abjad.tools.treetools.`**`compute_logical_xor_of_intervals_in_interval`**(*intervals*, *interval*)

Compute the logical XOR of a collections of intervals, cropped within *interval*.

treetools.concatenate_trees

`abjad.tools.treetools.`**`concatenate_trees`**(*trees*, *padding=0*)

Merge all trees in *trees*, offsetting each subsequent tree to start after the previous.

treetools.explode_intervals_compactly

`abjad.tools.treetools.`**`explode_intervals_compactly`**(*intervals*)

Explode the intervals in *intervals* into n non-overlapping trees, where n is the maximum depth of *intervals*.

Returns an array of *IntervalTree* instances.

The algorithm will attempt to insert the exploded intervals into the lowest-indexed resultant tree with free space.

treetools.explode_intervals_into_n_trees_heuristically

`abjad.tools.treetools.`**`explode_intervals_into_n_trees_heuristically`**(*intervals*, *n*)

Explode *intervals* into *n* trees, avoiding overlap when possible, and distributing intervals so as to equalize density across the trees.

treetools.explode_intervals_uncompactly

`abjad.tools.treetools.`**`explode_intervals_uncompactly`**(*intervals*)

Explode the intervals in *intervals* into n non-overlapping trees, where n is the maximum depth of *intervals*.

Returns an array of *IntervalTree* instances.

The algorithm will attempt to insert the exploded intervals cyclically, making its insertion attempt at the next resultant tree in the array, rather than always beginning its search from index 0.

treetools.fuse_overlapping_intervals

abjad.tools.treetools.**fuse_overlapping_intervals**(*intervals*)

Fuse the overlapping intervals in *intervals* and return an *IntervalTree* of the result

```
abjad> from abjad.tools import treetools
abjad> from abjad.tools.treetools import BoundedInterval
abjad> from abjad.tools.treetools import IntervalTree

abjad> a = BoundedInterval(0, 10)
abjad> b = BoundedInterval(5, 15)
abjad> c = BoundedInterval(15, 25)
abjad> tree = IntervalTree([a, b, c])
abjad> treetools.fuse_overlapping_intervals(tree)
IntervalTree([
   BoundedInterval(0, 15, {}),
   BoundedInterval(15, 25, {})
])
```

Return interval tree.

treetools.fuse_tangent_or_overlapping_intervals

abjad.tools.treetools.**fuse_tangent_or_overlapping_intervals**(*intervals*)

Fuse all tangent or overlapping intervals and return an *IntervalTree* of the result

```
abjad> from abjad.tools import treetools
abjad> from abjad.tools.treetools import BoundedInterval
abjad> from abjad.tools.treetools import IntervalTree

abjad> a = BoundedInterval(0, 10)
abjad> b = BoundedInterval(5, 15)
abjad> c = BoundedInterval(15, 25)
abjad> tree = IntervalTree([a, b, c])
abjad> treetools.fuse_tangent_or_overlapping_intervals(tree)
IntervalTree([
   BoundedInterval(0, 25, {})
])
```

Return interval tree.

treetools.get_all_unique_bounds_in_intervals

abjad.tools.treetools.**get_all_unique_bounds_in_intervals**(*intervals*)

Return all unique starting and ending boundaries in *intervals*.

treetools.group_overlapping_intervals_and_yield_groups

abjad.tools.treetools.**group_overlapping_intervals_and_yield_groups**(*intervals*)

Group overlapping intervals in *intervals* and return tuples.

treetools.group_tangent_or_overlapping_intervals_and_yield_groups

abjad.tools.treetools.**group_tangent_or_overlapping_intervals_and_yield_groups**(*intervals*)

Group tangent or overlapping intervals in *intervals* and return tuples.

treetools.make_monophonic_percussion_score_from_nonoverlapping_intervals

abjad.tools.treetools.**make_monophonic_percussion_score_from_nonoverlapping_intervals**(*intervals, colorkey=N*

Create a monophonic percussion score from nonoverlapping interval collection *intervals*.

treetools.make_polyphonic_percussion_score_from_nonoverlapping_trees

abjad.tools.treetools.**make_polyphonic_percussion_score_from_nonoverlapping_trees**(*trees, colorkey=None*)

Make a polyphonic percussion score from a collections of non-overlapping trees.

treetools.mask_intervals_with_intervals

abjad.tools.treetools.**mask_intervals_with_intervals**(*masked_intervals, mask_intervals*)

Clip or remove all intervals in *masked_intervals* outside of the bounds defined in *mask_intervals*, while maintaining *masked_intervals*' payload contents

```
abjad> from abjad.tools import treetools
abjad> from abjad.tools.treetools import BoundedInterval
abjad> from abjad.tools.treetools import IntervalTree

abjad> a = BoundedInterval(0, 10, 'a')
abjad> b = BoundedInterval(5, 15, 'b')
abjad> tree = IntervalTree([a, b])
abjad> mask = BoundedInterval(4, 11)
abjad> treetools.mask_intervals_with_intervals(tree, mask)
IntervalTree([
   BoundedInterval(4, 10, 'a'),
   BoundedInterval(5, 11, 'b')
])
```

Return interval tree.

treetools.resolve_overlaps_between_nonoverlapping_trees

abjad.tools.treetools.**resolve_overlaps_between_nonoverlapping_trees**(*trees*)

Create a nonoverlapping IntervalTree from *trees*. Intervals in higher-indexed trees in *trees* only appear in part or whole where they do not overlap intervals from lower-indexed trees

```
abjad> from abjad.tools import treetools
abjad> from abjad.tools.treetools import BoundedInterval
abjad> from abjad.tools.treetools import IntervalTree

abjad> a = IntervalTree(BoundedInterval(0, 4, 'a'))
abjad> b = IntervalTree(BoundedInterval(1, 5, 'b'))
abjad> c = IntervalTree(BoundedInterval(2, 6, 'c'))
abjad> d = IntervalTree(BoundedInterval(1, 3, 'd'))
abjad> treetools.resolve_overlaps_between_nonoverlapping_trees([a, b, c, d])
IntervalTree([
   BoundedInterval(0, 4, 'a'),
   BoundedInterval(4, 5, 'b'),
```

```
   BoundedInterval(5, 6, 'c')
])
```

Return interval tree.

treetools.resolve_overlaps_between_nonoverlapping_trees_excluding_remainders_less_than_rational

`abjad.tools.treetools.resolve_overlaps_between_nonoverlapping_trees_excluding_remainders_le`

Create a nonoverlapping IntervalTree from *trees*. Intervals in higher-indexed trees in *trees* only appear in part or whole where they do not overlap intervals from lower-indexed trees, and then only where their magnitudes are equal to or greater than *rational*

```
abjad> from abjad.tools import treetools
abjad> from abjad.tools.treetools import BoundedInterval
abjad> from abjad.tools.treetools import IntervalTree

abjad> a = IntervalTree(BoundedInterval(0, 1, 'a'))
abjad> b = IntervalTree(BoundedInterval(Fraction(1, 32), Fraction(33, 32), 'b'))
abjad> c = IntervalTree(BoundedInterval(Fraction(1, 16), Fraction(17, 16), 'c'))
abjad> treetools.resolve_overlaps_between_nonoverlapping_trees_excluding_remainders_less_than_ra
IntervalTree([
   BoundedInterval(0, 1, 'a'),
   BoundedInterval(1, Fraction(17, 16), 'c')
])
```

Return interval tree.

treetools.round_interval_bounds_to_nearest_multiple_of_rational

`abjad.tools.treetools.round_interval_bounds_to_nearest_multiple_of_rational`(*intervals*, *rational*)

treetools.scale_aggregate_magnitude_by_rational

`abjad.tools.treetools.scale_aggregate_magnitude_by_rational`(*intervals*, *rational*)

Scale the aggregate magnitude of all intervals in *intervals* by *rational*, maintaining the original low offset

```
abjad> from abjad.tools import treetools
abjad> from abjad.tools.treetools import BoundedInterval
abjad> from abjad.tools.treetools import IntervalTree

abjad> a = BoundedInterval(-1, 3)
abjad> b = BoundedInterval(6, 12)
abjad> c = BoundedInterval(9, 16)
abjad> tree = IntervalTree([a, b, c])
abjad> treetools.scale_aggregate_magnitude_by_rational(tree, Fraction(1, 3))
IntervalTree([
   BoundedInterval(-1, Fraction(1, 3), {}),
   BoundedInterval(Fraction(4, 3), Fraction(10, 3), {}),
```

```
    BoundedInterval(Fraction(7, 3), Fraction(14, 3), {})
])
```

Return interval tree.

treetools.scale_aggregate_magnitude_to_rational

abjad.tools.treetools.**scale_aggregate_magnitude_to_rational**(*intervals*, *rational*)

Scale the aggregate magnitude of all intervals in *intervals* to *rational*, maintaining the original low offset

```
abjad> from abjad.tools import treetools
abjad> from abjad.tools.treetools import BoundedInterval
abjad> from abjad.tools.treetools import IntervalTree

abjad> a = BoundedInterval(-1, 3)
abjad> b = BoundedInterval(6, 12)
abjad> c = BoundedInterval(9, 16)
abjad> tree = IntervalTree([a, b, c])
abjad> treetools.scale_aggregate_magnitude_to_rational(tree, Fraction(16, 7))
IntervalTree([
    BoundedInterval(-1, Fraction(-55, 119), {}),
    BoundedInterval(Fraction(-1, 17), Fraction(89, 119), {}),
    BoundedInterval(Fraction(41, 119), Fraction(9, 7), {})
])
```

Return interval tree.

treetools.scale_interval_magnitudes_by_rational

abjad.tools.treetools.**scale_interval_magnitudes_by_rational**(*intervals*, *rational*)

Scale the magnitude of each interval in *intervals* by *rational*, maintaining their low offsets

```
abjad> from abjad.tools import treetools
abjad> from abjad.tools.treetools import BoundedInterval
abjad> from abjad.tools.treetools import IntervalTree

abjad> a = BoundedInterval(-1, 3)
abjad> b = BoundedInterval(6, 12)
abjad> c = BoundedInterval(9, 16)
abjad> tree = IntervalTree([a, b, c])
abjad> treetools.scale_interval_magnitudes_by_rational(tree, Fraction(6, 5))
IntervalTree([
    BoundedInterval(-1, Fraction(19, 5), {}),
    BoundedInterval(6, Fraction(66, 5), {}),
    BoundedInterval(9, Fraction(87, 5), {})
])
```

Return interval tree.

treetools.scale_interval_magnitudes_to_rational

abjad.tools.treetools.**scale_interval_magnitudes_to_rational**(*intervals*, *rational*)

Scale the magnitude of each interval in *intervals* to *rational*, maintaining their low offsets

```
abjad> from abjad.tools import treetools
abjad> from abjad.tools.treetools import BoundedInterval
abjad> from abjad.tools.treetools import IntervalTree

abjad> a = BoundedInterval(-1, 3)
abjad> b = BoundedInterval(6, 12)
abjad> c = BoundedInterval(9, 16)
abjad> tree = IntervalTree([a, b, c])
abjad> treetools.scale_interval_magnitudes_to_rational(tree, Fraction(1, 7))
IntervalTree([
   BoundedInterval(-1, Fraction(-6, 7), {}),
   BoundedInterval(6, Fraction(43, 7), {}),
   BoundedInterval(9, Fraction(64, 7), {})
])
```

Return interval tree.

treetools.scale_interval_offsets_by_rational

abjad.tools.treetools.**scale_interval_offsets_by_rational**(*intervals*, *rational*)
Scale the offset of each interval in *intervals* by *rational*, maintaining the lowest offset in *intervals*

```
abjad> from abjad.tools import treetools
abjad> from abjad.tools.treetools import BoundedInterval
abjad> from abjad.tools.treetools import IntervalTree

abjad> a = BoundedInterval(-1, 3)
abjad> b = BoundedInterval(6, 12)
abjad> c = BoundedInterval(9, 16)
abjad> tree = IntervalTree([a, b, c])
abjad> treetools.scale_interval_offsets_by_rational(tree, Fraction(4, 5))
IntervalTree([
   BoundedInterval(-1, 3, {}),
   BoundedInterval(Fraction(23, 5), Fraction(53, 5), {}),
   BoundedInterval(Fraction(7, 1), Fraction(14, 1), {})
])
```

Return interval tree.

treetools.shift_aggregate_offset_by_rational

abjad.tools.treetools.**shift_aggregate_offset_by_rational**(*intervals*, *rational*)
Shift the aggregate offset of *intervals* by *rational*

```
abjad> from abjad.tools import treetools
abjad> from abjad.tools.treetools import BoundedInterval
abjad> from abjad.tools.treetools import IntervalTree

abjad> a = BoundedInterval(-1, 3)
abjad> b = BoundedInterval(6, 12)
abjad> c = BoundedInterval(9, 16)
abjad> tree = IntervalTree([a, b, c])
abjad> treetools.shift_aggregate_offset_by_rational(tree, Fraction(1, 3))
IntervalTree([
   BoundedInterval(Fraction(-2, 3), Fraction(10, 3), {}),
   BoundedInterval(Fraction(19, 3), Fraction(37, 3), {}),
```

```
    BoundedInterval(Fraction(28, 3), Fraction(49, 3), {})
])
```

Return interval tree.

treetools.shift_aggregate_offset_to_rational

abjad.tools.treetools.**shift_aggregate_offset_to_rational**(*intervals*, *rational*)

Shift the aggregate offset of *intervals* to *rational*

```
abjad> from abjad.tools import treetools
abjad> from abjad.tools.treetools import BoundedInterval
abjad> from abjad.tools.treetools import IntervalTree

abjad> a = BoundedInterval(-1, 3)
abjad> b = BoundedInterval(6, 12)
abjad> c = BoundedInterval(9, 16)
abjad> tree = IntervalTree([a, b, c])
abjad> treetools.shift_aggregate_offset_to_rational(tree, Fraction(10, 7))
IntervalTree([
  BoundedInterval(Fraction(10, 7), Fraction(38, 7), {}),
  BoundedInterval(Fraction(59, 7), Fraction(101, 7), {}),
  BoundedInterval(Fraction(80, 7), Fraction(129, 7), {})
])
```

Return interval tree.

treetools.split_intervals_at_rationals

abjad.tools.treetools.**split_intervals_at_rationals**(*intervals*, *rationals*)

Split *intervals* at each rational in *rationals*

```
abjad> from abjad.tools import treetools
abjad> from abjad.tools.treetools import BoundedInterval
abjad> from abjad.tools.treetools import IntervalTree

abjad> a = BoundedInterval(-1, 3)
abjad> b = BoundedInterval(6, 12)
abjad> c = BoundedInterval(9, 16)
abjad> tree = IntervalTree([a, b, c])
abjad> treetools.split_intervals_at_rationals(tree, [1, Fraction(19, 2)])
IntervalTree([
  BoundedInterval(-1, 1, {}),
  BoundedInterval(1, 3, {}),
  BoundedInterval(6, Fraction(19, 2), {}),
  BoundedInterval(9, Fraction(19, 2), {}),
  BoundedInterval(Fraction(19, 2), 12, {}),
  BoundedInterval(Fraction(19, 2), 16, {})
])
```

Return interval tree.

BIBLIOGRAPHY

[Adan2006] Víctor Adán. Music <-> Geometry <-> Meta-Music. Draft February 12, 2006.

[AgonHaddadAssayag2002] Carlos Agon, Karim Haddad & Gerard Assayag. Répresentation et rendu de structures rhythmiques. Journées d'Informatique Musicale, 9th ed., Marseille, 29 - 31 May 2002.

[Alegant1993] Brian Alegant. The seventy-seven partitions of the aggregate: Analytical and theoretical implications. Doctoral Dissertation. The University of Rochester, Eastman School of Muisc. 1993.

[Ariza2005] Christopher Ariza. An Open Design for Computer-Aided Algorithmic Music Composition: athenaCL. Dissertation.com, Boca Raton. 2005.

[BacaAdan2007] Trevor Bača & Víctor Adán. Cuepatlahto and Lascaux: two approaches to the formalized control of musical score. Draft June 7, 2007.

[BressonAgonAssayag2008] Jean Bresson, Carlos Agon, Gérard Assayag. The OM Composer's Book 2. Éditions Delatour, Paris. 2008

[Carter2002] Eliot Carter. Harmony Book. Nicholas Hopkins and John F. Link, eds. Carl Fischer, New York. 2002.

[Haddad] Karim Haddad. Le Temps comme Territoire: pour une géographie temporelle.

[Kampela1998] Arthur Kampela. Uma Faca Só Lâmina. Doctoral Dissertation. Columbia University, NY, NY. 1998.

[Malt2008] Mikhaïl Malt. Some Considerations on Brian Ferneyhough's Musical Language Through His Use of CAC – Part I: Time and Rhythmic Structures. In [BressonAgonAssayag2008].

[Morris1987] Robert Morris. Composition with Pitch-Classes. Yale University Press, New Haven. 1987.

[Nauert1997] Paul Nauert. Timespan Formation in Nonmetric, Posttonal Music. Doctoral Dissertation. Columbia University, NY, NY. 1997.

[NienhuysNieuwenhuizen2003] Han-Wen Nienhuys & Jan Nieuwenhuizen. Lilypond: A system for automated music engraving. Proceedings of the XIV Colloquium on Musical Informatics. Firenze, Italy. May 8 - 10, 2003.

[Ross1987] Ted Ross. Teach Yourself The Art of Music Engraving and Processing. Hansen House, Miami Beach. 1987.

[Selfridge-Field1997] Eleanor Selfridge-Field, ed. Beyond MIDI: The Handbook of Musical Codes. The MIT Press. Cambridge, Massachusetts. 1997.

[Valle] Andrea Valle. GeoGraphy: Notazione musicale e composizione algorithmica. Centro Interdipartimentale di Ricerca sulla Multimedialità e l'Audiovisivo. Università degli Studi di Torino.

[WulfsonBarrettWinter] Harris Wulfson, G. Douglas Barrett & Michael Winter. Automatic Notation Generators.

INDEX

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z