*Four Ages of Sand* Segment I.

import abjad

import itertools

import os

import pathlib

import time

import abjadext.rmakers

from MusicMaker import MusicMaker

from AttachmentHandler import AttachmentHandler

from random import random

from random import seed

print('Interpreting file ...')

# Define the time signatures we would like to apply against the timespan structure.

time\_signatures = [

abjad.TimeSignature(pair) for pair in [

(9, 8), (6, 8), (5, 4), (4, 4), (6, 8),

(3, 8), (4, 4), (3, 4), (5, 8), (4, 4),

(11, 8),

]

]

bounds = abjad.mathtools.cumulative\_sums([\_.duration for \_ in time\_signatures])

#Define Pitch Material

def reduceMod3(rw):

return [(x % 4) for x in rw]

def reduceMod7(rw):

return [(x % 8) for x in rw]

def reduceMod9(rw):

return [(x % 10) for x in rw]

def cyc(lst):

count = 0

while True:

yield lst[count%len(lst)]

count += 1

def grouper(lst1, lst2):

def cyc(lst):

c = 0

while True:

yield lst[c%len(lst)]

c += 1

lst1 = cyc(lst1)

return [next(lst1) if i == 1 else [next(lst1) for \_ in range(i)] for i in lst2]

seed(1)

flute\_random\_walk\_one = []

flute\_random\_walk\_one.append(-1 if random() < 0.5 else 1)

for i in range(1, 1000):

movement = -1 if random() < 0.5 else 1

value = flute\_random\_walk\_one[i-1] + movement

flute\_random\_walk\_one.append(value)

flute\_random\_walk\_one = [abs(x) for x in flute\_random\_walk\_one]

flute\_chord\_one = [1, 2, 9, 10, 18, 27, 18, 10, 9, 2, ]

flute\_notes\_one = [flute\_chord\_one[x] for x in reduceMod9(flute\_random\_walk\_one)]

seed(4)

saxophone\_random\_walk\_one = []

saxophone\_random\_walk\_one.append(-1 if random() < 0.5 else 1)

for i in range(1, 1000):

movement = -1 if random() < 0.5 else 1

value = saxophone\_random\_walk\_one[i-1] + movement

saxophone\_random\_walk\_one.append(value)

saxophone\_random\_walk\_one = [abs(x) for x in saxophone\_random\_walk\_one]

saxophone\_chord\_one = [-8, 1, 2, 9, 10, 9, 2, 1, ]

saxophone\_notes\_one = [saxophone\_chord\_one[x] for x in reduceMod7(saxophone\_random\_walk\_one)]

seed(8)

cello\_random\_walk\_one = []

cello\_random\_walk\_one.append(-1 if random() < 0.5 else 1)

for i in range(1, 1000):

movement = -1 if random() < 0.5 else 1

value = cello\_random\_walk\_one[i-1] + movement

cello\_random\_walk\_one.append(value)

cello\_random\_walk\_one = [abs(x) for x in cello\_random\_walk\_one]

cello\_chord\_one = [-24, -17, -16, -8, 1, 2, 1, -8, -16, -17, ]

cello\_notes\_one = [cello\_chord\_one[x] for x in reduceMod9(cello\_random\_walk\_one)]

seed(1)

flute\_random\_walk\_two = []

flute\_random\_walk\_two.append(-1 if random() < 0.5 else 1)

for i in range(1, 1000):

movement = -1 if random() < 0.5 else 1

value = flute\_random\_walk\_two[i-1] + movement

flute\_random\_walk\_two.append(value)

flute\_random\_walk\_two = [abs(x) for x in flute\_random\_walk\_two]

flute\_chord\_two = [1, 9, 18, 9, ]

flute\_notes\_two = [flute\_chord\_two[x] for x in reduceMod3(flute\_random\_walk\_two)]

seed(4)

saxophone\_random\_walk\_two = []

saxophone\_random\_walk\_two.append(-1 if random() < 0.5 else 1)

for i in range(1, 1000):

movement = -1 if random() < 0.5 else 1

value = saxophone\_random\_walk\_two[i-1] + movement

saxophone\_random\_walk\_two.append(value)

saxophone\_random\_walk\_two = [abs(x) for x in saxophone\_random\_walk\_two]

saxophone\_chord\_two = [-8, 2, 10, 2, ]

saxophone\_notes\_two = [saxophone\_chord\_two[x] for x in reduceMod3(saxophone\_random\_walk\_two)]

seed(8)

cello\_random\_walk\_two = []

cello\_random\_walk\_two.append(-1 if random() < 0.5 else 1)

for i in range(1, 1000):

movement = -1 if random() < 0.5 else 1

value = cello\_random\_walk\_two[i-1] + movement

cello\_random\_walk\_two.append(value)

cello\_random\_walk\_two = [abs(x) for x in cello\_random\_walk\_two]

cello\_chord\_two = [-24, -16, 1, -16, ]

cello\_notes\_two = [cello\_chord\_two[x] for x in reduceMod3(cello\_random\_walk\_two)]

seed(1)

flute\_random\_walk\_three = []

flute\_random\_walk\_three.append(-1 if random() < 0.5 else 1)

for i in range(1, 1000):

movement = -1 if random() < 0.5 else 1

value = flute\_random\_walk\_three[i-1] + movement

flute\_random\_walk\_three.append(value)

flute\_random\_walk\_three = [abs(x) for x in flute\_random\_walk\_three]

flute\_chord\_three = [2, 10, 27, 10, ]

flute\_notes\_three = [flute\_chord\_three[x] for x in reduceMod3(flute\_random\_walk\_three)]

seed(4)

saxophone\_random\_walk\_three = []

saxophone\_random\_walk\_three.append(-1 if random() < 0.5 else 1)

for i in range(1, 1000):

movement = -1 if random() < 0.5 else 1

value = saxophone\_random\_walk\_three[i-1] + movement

saxophone\_random\_walk\_three.append(value)

saxophone\_random\_walk\_three = [abs(x) for x in saxophone\_random\_walk\_three]

saxophone\_chord\_three = [1, 2, 10, 2, ]

saxophone\_notes\_three = [saxophone\_chord\_three[x] for x in reduceMod3(saxophone\_random\_walk\_three)]

seed(8)

cello\_random\_walk\_three = []

cello\_random\_walk\_three.append(-1 if random() < 0.5 else 1)

for i in range(1, 1000):

movement = -1 if random() < 0.5 else 1

value = cello\_random\_walk\_three[i-1] + movement

cello\_random\_walk\_three.append(value)

cello\_random\_walk\_three = [abs(x) for x in cello\_random\_walk\_three]

cello\_chord\_three = [-17, -8, 2, -8, ]

cello\_notes\_three = [cello\_chord\_three[x] for x in reduceMod3(cello\_random\_walk\_three)]

# Define rhythm-makers: two to be sued by the MusicMaker, one for silence.

rmaker\_one = abjadext.rmakers.TaleaRhythmMaker(

talea=abjadext.rmakers.Talea(

counts=[2, 7, 2, 3, 2, 4, 7, 2, 5, 6],

denominator=32,

),

beam\_specifier=abjadext.rmakers.BeamSpecifier(

beam\_divisions\_together=True,

beam\_rests=False,

),

extra\_counts\_per\_division=[0, 1, 0, -1, 1, ],

burnish\_specifier=abjadext.rmakers.BurnishSpecifier(

left\_classes=[abjad.Note, abjad.Rest],

left\_counts=[1, 0, 1],

),

tuplet\_specifier=abjadext.rmakers.TupletSpecifier(

trivialize=True,

extract\_trivial=True,

rewrite\_rest\_filled=True,

),

)

rmaker\_two = abjadext.rmakers.TaleaRhythmMaker(

talea=abjadext.rmakers.Talea(

counts=[1, 2, 2, 3, 3, 3, 2, 2, 1, ],

denominator=16,

),

beam\_specifier=abjadext.rmakers.BeamSpecifier(

beam\_divisions\_together=True,

beam\_rests=False,

),

extra\_counts\_per\_division=[1, 0, -1, 0, 1],

burnish\_specifier=abjadext.rmakers.BurnishSpecifier(

left\_classes=[abjad.Note, abjad.Rest],

left\_counts=[1, 0, 1],

),

tuplet\_specifier=abjadext.rmakers.TupletSpecifier(

trivialize=True,

extract\_trivial=True,

rewrite\_rest\_filled=True,

),

)

rmaker\_three = abjadext.rmakers.EvenDivisionRhythmMaker(

denominators=[16, 16, 8, 16, 16, 8],

extra\_counts\_per\_division=[1, 0, 0, -1, 0, 1, -1, 0],

burnish\_specifier=abjadext.rmakers.BurnishSpecifier(

left\_classes=[abjad.Rest],

left\_counts=[1],

right\_classes=[abjad.Rest],

right\_counts=[1],

outer\_divisions\_only=True,

),

tuplet\_specifier=abjadext.rmakers.TupletSpecifier(

trivialize=True,

extract\_trivial=True,

rewrite\_rest\_filled=True,

),

)

# Initialize AttachmentHandler

attachment\_handler\_one = AttachmentHandler(

starting\_dynamic='mf',

ending\_dynamic='p',

hairpin\_indicator='--',

articulation='accent',

)

attachment\_handler\_two = AttachmentHandler(

starting\_dynamic='mf',

ending\_dynamic='ff',

hairpin\_indicator='<',

articulation='tenuto',

)

attachment\_handler\_three = AttachmentHandler(

starting\_dynamic='ff',

ending\_dynamic='mp',

hairpin\_indicator='|>',

articulation='',

)

# Initialize MusicMakers with the rhythm-makers.

#####oboe#####

flutemusicmaker\_one = MusicMaker(

rmaker=rmaker\_one,

pitches=flute\_notes\_one,

continuous=True,

attachment\_handler=attachment\_handler\_one,

)

flutemusicmaker\_two = MusicMaker(

rmaker=rmaker\_two,

pitches=flute\_notes\_two,

continuous=True,

attachment\_handler=attachment\_handler\_two,

)

flutemusicmaker\_three = MusicMaker(

rmaker=rmaker\_three,

pitches=flute\_notes\_three,

continuous=True,

attachment\_handler=attachment\_handler\_three,

)

#####saxophone#####

saxophonemusicmaker\_one = MusicMaker(

rmaker=rmaker\_one,

pitches=saxophone\_notes\_one,

continuous=True,

attachment\_handler=attachment\_handler\_one,

)

saxophonemusicmaker\_two = MusicMaker(

rmaker=rmaker\_two,

pitches=saxophone\_notes\_two,

continuous=True,

attachment\_handler=attachment\_handler\_two,

)

saxophonemusicmaker\_three = MusicMaker(

rmaker=rmaker\_three,

pitches=saxophone\_notes\_three,

continuous=True,

attachment\_handler=attachment\_handler\_three,

)

#####cello#####

cellomusicmaker\_one = MusicMaker(

rmaker=rmaker\_one,

pitches=cello\_notes\_one,

continuous=True,

attachment\_handler=attachment\_handler\_one,

)

cellomusicmaker\_two = MusicMaker(

rmaker=rmaker\_two,

pitches=cello\_notes\_two,

continuous=True,

attachment\_handler=attachment\_handler\_two,

)

cellomusicmaker\_three = MusicMaker(

rmaker=rmaker\_three,

pitches=cello\_notes\_three,

continuous=True,

attachment\_handler=attachment\_handler\_three,

)

silence\_maker = abjadext.rmakers.NoteRhythmMaker(

division\_masks=[

abjadext.rmakers.SilenceMask(

pattern=abjad.index([0], 1),

),

],

)

# Define a small class so that we can annotate timespans with additional

# information:

class MusicSpecifier:

def \_\_init\_\_(self, music\_maker, voice\_name):

self.music\_maker = music\_maker

self.voice\_name = voice\_name

# Define an initial timespan structure, annotated with music specifiers. This

# structure has not been split along meter boundaries. This structure does not

# contain timespans explicitly representing silence. Here I make four, one

# for each voice, using Python's list comprehension syntax to save some

# space.

print('Collecting timespans and rmakers ...')

###group one###

voice\_1\_timespan\_list = abjad.TimespanList([

abjad.AnnotatedTimespan(

start\_offset=start\_offset,

stop\_offset=stop\_offset,

annotation=MusicSpecifier(

music\_maker=music\_maker,

voice\_name='Voice 1',

),

)

for start\_offset, stop\_offset, music\_maker in [

[(0, 8), (3, 8), flutemusicmaker\_one],

[(4, 8), (8, 8), flutemusicmaker\_two],

[(10, 8), (12, 8), flutemusicmaker\_three],

[(12, 8), (15, 8), flutemusicmaker\_one],

[(18, 8), (24, 8), flutemusicmaker\_two],

[(28, 8), (33, 8), flutemusicmaker\_three],

[(33, 8), (35, 8), flutemusicmaker\_one],

[(40, 8), (42, 8), flutemusicmaker\_two],

[(42, 8), (44, 8), flutemusicmaker\_three],

[(44, 8), (48, 8), flutemusicmaker\_one],

[(54, 8), (55, 8), flutemusicmaker\_two],

[(62, 8), (64, 8), flutemusicmaker\_three],

[(72, 8), (75, 8), flutemusicmaker\_one],

[(76, 8), (79, 8), flutemusicmaker\_two],

[(79, 8), (80, 8), silence\_maker],

]

])

voice\_3\_timespan\_list = abjad.TimespanList([

abjad.AnnotatedTimespan(

start\_offset=start\_offset,

stop\_offset=stop\_offset,

annotation=MusicSpecifier(

music\_maker=music\_maker,

voice\_name='Voice 3',

),

)

for start\_offset, stop\_offset, music\_maker in [

[(9, 8), (12, 8), saxophonemusicmaker\_one],

[(20, 8), (24, 8), saxophonemusicmaker\_two],

[(31, 8), (33, 8), saxophonemusicmaker\_three],

[(33, 8), (36, 8), saxophonemusicmaker\_one],

[(42, 8), (48, 8), saxophonemusicmaker\_two],

[(53, 8), (56, 8), saxophonemusicmaker\_three],

[(56, 8), (60, 8), saxophonemusicmaker\_one],

[(64, 8), (69, 8), saxophonemusicmaker\_two],

[(69, 8), (72, 8), saxophonemusicmaker\_three],

[(75, 8), (79, 8), saxophonemusicmaker\_one],

]

])

voice\_8\_timespan\_list = abjad.TimespanList([

abjad.AnnotatedTimespan(

start\_offset=start\_offset,

stop\_offset=stop\_offset,

annotation=MusicSpecifier(

music\_maker=music\_maker,

voice\_name='Voice 8',

),

)

for start\_offset, stop\_offset, music\_maker in [

[(15, 8), (18, 8), cellomusicmaker\_one],

[(18, 8), (22, 8), cellomusicmaker\_two],

[(25, 8), (29, 8), cellomusicmaker\_three],

[(29, 8), (32, 8), cellomusicmaker\_one],

[(35, 8), (39, 8), cellomusicmaker\_two],

[(39, 8), (42, 8), cellomusicmaker\_three],

[(45, 8), (50, 8), cellomusicmaker\_one],

[(50, 8), (52, 8), cellomusicmaker\_two],

[(55, 8), (56, 8), cellomusicmaker\_three],

[(56, 8), (61, 8), cellomusicmaker\_one],

[(61, 8), (62, 8), cellomusicmaker\_two],

[(65, 8), (69, 8), cellomusicmaker\_three],

[(69, 8), (72, 8), cellomusicmaker\_one],

[(75, 8), (79, 8), cellomusicmaker\_two],

]

])

# Create a dictionary mapping voice names to timespan lists so we can

# maintain the association in later operations:

all\_timespan\_lists = {

'Voice 1': voice\_1\_timespan\_list,

'Voice 3': voice\_3\_timespan\_list,

'Voice 8': voice\_8\_timespan\_list,

}

# Determine the "global" timespan of all voices combined:

global\_timespan = abjad.Timespan(

start\_offset=0,

stop\_offset=max(\_.stop\_offset for \_ in all\_timespan\_lists.values())

)

# Using the global timespan, create silence timespans for each timespan list.

# We don't need to create any silences by-hand if we now the global start and

# stop offsets of all voices combined:

for voice\_name, timespan\_list in all\_timespan\_lists.items():

# Here is another technique for finding where the silence timespans are. We

# create a new timespan list consisting of the global timespan and all the

# timespans from our current per-voice timespan list. Then we compute an

# in-place logical XOR. The XOR will replace the contents of the "silences"

# timespan list with a set of timespans representing those periods of time

# where only one timespan from the original was present. This has the

# effect of cutting out holes from the global timespan wherever a per-voice

# timespan was found, but also preserves any silence before the first

# per-voice timespan or after the last per-voice timespan. Then we merge

# the newly-created silences back into the per-voice timespan list.

silences = abjad.TimespanList([global\_timespan])

silences.extend(timespan\_list)

silences.sort()

silences.compute\_logical\_xor()

# Add the silences into the voice timespan list. We create new \*annotated\*

# timespans so we can maintain the voice name information:

for silence\_timespan in silences:

timespan\_list.append(

abjad.AnnotatedTimespan(

start\_offset=silence\_timespan.start\_offset,

stop\_offset=silence\_timespan.stop\_offset,

annotation=MusicSpecifier(

music\_maker=None,

voice\_name=voice\_name,

),

)

)

timespan\_list.sort()

# Split the timespan list via the time signatures and collect the shards into a

# new timespan list

for voice\_name, timespan\_list in all\_timespan\_lists.items():

shards = timespan\_list.split\_at\_offsets(bounds)

split\_timespan\_list = abjad.TimespanList()

for shard in shards:

split\_timespan\_list.extend(shard)

split\_timespan\_list.sort()

# We can replace the original timespan list in the dictionary of

# timespan lists because we know the key it was stored at (its voice

# name):

all\_timespan\_lists[voice\_name] = timespan\_list

# Create a score structure

score = abjad.Score([

abjad.Staff(lilypond\_type='TimeSignatureContext', name='Global Context'),

abjad.StaffGroup(

[

abjad.Staff([abjad.Voice(name='Voice 1')],name='Staff 1', lilypond\_type='Staff',),

abjad.Staff([abjad.Voice(name='Voice 3')],name='Staff 3', lilypond\_type='Staff',),

abjad.Staff([abjad.Voice(name='Voice 8')],name='Staff 8', lilypond\_type='Staff',),

],

name='Staff Group 1',

),

],

)

# Teach each of the staves how to draw analysis brackets

# for staff in score['Staff Group']:

# staff.consists\_commands.append('Horizontal\_bracket\_engraver')

# Add skips and time signatures to the global context

for time\_signature in time\_signatures:

skip = abjad.Skip(1, multiplier=(time\_signature))

abjad.attach(time\_signature, skip)

score['Global Context'].append(skip)

# Define a helper function that takes a rhythm maker and some durations and

# outputs a container. This helper function also adds LilyPond analysis

# brackets to make it clearer where the phrase and sub-phrase boundaries are.

print('Making containers ...')

def make\_container(music\_maker, durations):

selections = music\_maker(durations)

container = abjad.Container([])

container.extend(selections)

# # Add analysis brackets so we can see the phrasing graphically

# start\_indicator = abjad.LilyPondLiteral('\startGroup', format\_slot='after')

# stop\_indicator = abjad.LilyPondLiteral('\stopGroup', format\_slot='after')

# for cell in selections:

# cell\_first\_leaf = abjad.select(cell).leaves()[0]

# cell\_last\_leaf = abjad.select(cell).leaves()[-1]

# abjad.attach(start\_indicator, cell\_first\_leaf)

# abjad.attach(stop\_indicator, cell\_last\_leaf)

# # The extra space in the literals is a hack around a check for whether an

# # identical object has already been attached

# start\_indicator = abjad.LilyPondLiteral('\startGroup ', format\_slot='after')

# stop\_indicator = abjad.LilyPondLiteral('\stopGroup ', format\_slot='after')

# phrase\_first\_leaf = abjad.select(container).leaves()[0]

# phrase\_last\_leaf = abjad.select(container).leaves()[-1]

# abjad.attach(start\_indicator, phrase\_first\_leaf)

# abjad.attach(stop\_indicator, phrase\_last\_leaf)

return container

# Loop over the timespan list dictionaries, spitting out pairs of voice

# names and per-voice timespan lists. Group timespans into phrases, with

# all timespans in each phrase having an identical rhythm maker. Run the

# rhythm maker against the durations of the timespans in the phrase and

# add the output to the voice with the timespan lists's voice name.

def key\_function(timespan):

"""

Get the timespan's annotation's rhythm-maker.

If the annotation's rhythm-maker is None, return the silence maker.

"""

return timespan.annotation.music\_maker or silence\_maker

for voice\_name, timespan\_list in all\_timespan\_lists.items():

for music\_maker, grouper in itertools.groupby(

timespan\_list,

key=key\_function,

):

# We know the voice name of each timespan because a) the timespan

# list is in a dictionary, associated with that voice name and b)

# each timespan's annotation is a MusicSpecifier instance which

# knows the name of the voice the timespan should be used for.

# This double-reference to the voice is redundant here, but in a

# different implementation we could put \*all\* the timespans into

# one timespan list, split them, whatever, and still know which

# voice they belong to because their annotation records that

# information.

durations = [timespan.duration for timespan in grouper]

container = make\_container(music\_maker, durations)

voice = score[voice\_name]

voice.append(container)

print('Splitting and rewriting ...')

# split and rewite meters

for voice in abjad.iterate(score['Staff Group 1']).components(abjad.Voice):

for i , shard in enumerate(abjad.mutate(voice[:]).split(time\_signatures)):

time\_signature = time\_signatures[i]

abjad.mutate(shard).rewrite\_meter(time\_signature)

print('Beaming runs ...')

for voice in abjad.select(score).components(abjad.Voice):

for run in abjad.select(voice).runs():

if 1 < len(run):

# use a beam\_specifier to remove beam indicators from run

specifier = abjadext.rmakers.BeamSpecifier(

beam\_each\_division=False,

)

specifier(run)

# then attach new indicators at the 0 and -1 of run

abjad.attach(abjad.StartBeam(), run[0])

abjad.attach(abjad.StopBeam(), run[-1])

for leaf in run:

# continue if leaf can't be beamed

if abjad.Duration(1, 4) <= leaf.written\_duration:

continue

previous\_leaf = abjad.inspect(leaf).leaf(-1)

next\_leaf = abjad.inspect(leaf).leaf(1)

# if next leaf is quarter note (or greater) ...

if (isinstance(next\_leaf, (abjad.Chord, abjad.Note)) and

abjad.Duration(1, 4) <= next\_leaf.written\_duration):

left = previous\_leaf.written\_duration.flag\_count

right = leaf.written\_duration.flag\_count # right-pointing nib

beam\_count = abjad.BeamCount(

left=left,

right=right,

)

abjad.attach(beam\_count, leaf)

continue

# if previous leaf is quarter note (or greater) ...

if (isinstance(previous\_leaf, (abjad.Chord, abjad.Note)) and

abjad.Duration(1, 4) <= previous\_leaf.written\_duration):

left = leaf.written\_duration.flag\_count # left-pointing nib

right = next\_leaf.written\_duration.flag\_count

beam\_count = abjad.BeamCount(

left=left,

right=right,

)

abjad.attach(beam\_count, leaf)

print('Stopping Hairpins ...')

for staff in abjad.iterate(score['Staff Group 1']).components(abjad.Staff):

for rest in abjad.iterate(staff).components(abjad.Rest):

previous\_leaf = abjad.inspect(rest).leaf(-1)

if isinstance(previous\_leaf, abjad.Note):

abjad.attach(abjad.StopHairpin(), rest)

elif isinstance(previous\_leaf, abjad.Chord):

abjad.attach(abjad.StopHairpin(), rest)

elif isinstance(previous\_leaf, abjad.Rest):

pass

#attach instruments and clefs

print('Adding attachments ...')

bar\_line = abjad.BarLine('||')

metro = abjad.MetronomeMark((1, 4), 60)

markup1 = abjad.Markup(r'\bold { A }')

markup2 = abjad.Markup(r'\bold { B }')

markup3 = abjad.Markup(r'\bold { C }')

markup4 = abjad.Markup(r'\bold { D }')

markup5 = abjad.Markup(r'\bold { E }')

markup6 = abjad.Markup(r'\bold { F }')

mark1 = abjad.RehearsalMark(markup=markup1)

mark2 = abjad.RehearsalMark(markup=markup2)

mark3 = abjad.RehearsalMark(markup=markup3)

mark4 = abjad.RehearsalMark(markup=markup4)

mark5 = abjad.RehearsalMark(markup=markup5)

mark6 = abjad.RehearsalMark(markup=markup6)

instruments1 = cyc([

abjad.Flute(),

abjad.AltoSaxophone(),

abjad.Cello(),

])

clefs1 = cyc([

abjad.Clef('treble'),

abjad.Clef('treble'),

abjad.Clef('bass'),

])

abbreviations1 = cyc([

abjad.MarginMarkup(markup=abjad.Markup('fl.'),),

abjad.MarginMarkup(markup=abjad.Markup('sx.'),),

abjad.MarginMarkup(markup=abjad.Markup('vc.'),),

])

names1 = cyc([

abjad.StartMarkup(markup=abjad.Markup('Flute'),),

abjad.StartMarkup(markup=abjad.Markup('Saxophone'),),

abjad.StartMarkup(markup=abjad.Markup('Violoncello'),),

])

for staff in abjad.iterate(score['Staff Group 1']).components(abjad.Staff):

leaf1 = abjad.select(staff).leaves()[0]

abjad.attach(next(instruments1), leaf1)

abjad.attach(next(abbreviations1), leaf1)

abjad.attach(next(names1), leaf1)

abjad.attach(next(clefs1), leaf1)

for staff in abjad.select(score['Staff Group 1']).components(abjad.Staff)[0]:

leaf1 = abjad.select(staff).leaves()[0]

last\_leaf = abjad.select(staff).leaves()[-1]

abjad.attach(metro, leaf1)

abjad.attach(bar\_line, last\_leaf)

# for staff in abjad.iterate(score['Global Context']).components(abjad.Staff):

# leaf1 = abjad.select(staff).leaves()[7]

# abjad.attach(mark1, leaf1)

#

# for staff in abjad.iterate(score['Global Context']).components(abjad.Staff):

# leaf2 = abjad.select(staff).leaves()[16]

# abjad.attach(mark2, leaf2)

#

# for staff in abjad.iterate(score['Global Context']).components(abjad.Staff):

# leaf3 = abjad.select(staff).leaves()[22]

# abjad.attach(mark3, leaf3)

#

# for staff in abjad.iterate(score['Global Context']).components(abjad.Staff):

# leaf4 = abjad.select(staff).leaves()[29]

# abjad.attach(mark4, leaf4)

#

# for staff in abjad.iterate(score['Global Context']).components(abjad.Staff):

# leaf5 = abjad.select(staff).leaves()[34]

# abjad.attach(mark5, leaf5)

#

# for staff in abjad.iterate(score['Global Context']).components(abjad.Staff):

# leaf6 = abjad.select(staff).leaves()[39]

# abjad.attach(mark6, leaf6)

for staff in abjad.iterate(score['Staff Group 1']).components(abjad.Staff):

abjad.Instrument.transpose\_from\_sounding\_pitch(staff)

# Make a lilypond file and show it:

score\_file = abjad.LilyPondFile.new(

score,

includes=['first\_stylesheet.ily', '/Users/evansdsg2/abjad/docs/source/\_stylesheets/abjad.ily'],

)

abjad.SegmentMaker.comment\_measure\_numbers(score)

###################

directory = '/Users/evansdsg2/Scores/four\_ages\_of\_sand/four\_ages\_of\_sand/Segments/Segment\_I'

pdf\_path = f'{directory}/Segment\_I.pdf'

path = pathlib.Path('Segment\_I.pdf')

if path.exists():

print(f'Removing {pdf\_path} ...')

path.unlink()

time\_1 = time.time()

print(f'Persisting {pdf\_path} ...')

result = abjad.persist(score\_file).as\_pdf(pdf\_path)

print(result[0])

print(result[1])

print(result[2])

success = result[3]

if success is False:

print('LilyPond failed!')

time\_2 = time.time()

total\_time = time\_2 - time\_1

print(f'Total time: {total\_time} seconds')

if path.exists():

print(f'Opening {pdf\_path} ...')

os.system(f'open {pdf\_path}')

# for staff in abjad.iterate(score['Staff Group']).components(abjad.Staff):

# abjad.show(staff)

# abjad.show(score)

# abjad.play(score)