*PitchHandler* Source Code.

import abjad

class PitchHandler:

def \_\_init\_\_(

self,

pitch\_list=None,

continuous=False,

):

def cyc(lst):

if self.continuous == False:

self.\_count = 0

while True:

yield lst[self.\_count % len(lst)]

self.\_count += 1

self.pitch\_list = pitch\_list

self.continuous = continuous

self.\_cyc\_pitches = cyc(pitch\_list)

self.\_count = 0

def \_\_call\_\_(self, selections):

return self.\_apply\_pitches(selections, self.pitch\_list)

def \_collect\_pitches\_durations\_leaves(self, logical\_ties, pitches):

def cyc(lst):

if self.continuous == False:

self.\_count = 0

while True:

yield lst[self.\_count % len(lst)]

self.\_count += 1

cyc\_pitches = cyc(pitches)

pitches, durations, leaves = [[], [], []]

for tie in logical\_ties:

if isinstance(tie[0], abjad.Note):

pitch = next(cyc\_pitches)

for leaf in tie:

pitches.append(pitch)

durations.append(leaf.written\_duration)

leaves.append(leaf)

else:

for leaf in tie:

pitches.append(None)

durations.append(leaf.written\_duration)

leaves.append(leaf)

return pitches, durations, leaves

def \_apply\_pitches(self, selections, pitches):

leaf\_maker = abjad.LeafMaker()

container = abjad.Container(selections)

old\_ties = [tie for tie in abjad.iterate(

container).logical\_ties()]

pitches, durations, old\_leaves = self.\_collect\_pitches\_durations\_leaves(

old\_ties, pitches)

new\_leaves = [leaf for leaf in leaf\_maker(pitches, durations)]

for old\_leaf, new\_leaf in zip(old\_leaves, new\_leaves):

indicators = abjad.inspect(old\_leaf).indicators()

for indicator in indicators:

abjad.attach(indicator, new\_leaf)

parent = abjad.inspect(old\_leaf).parentage().parent

parent[parent.index(old\_leaf)] = new\_leaf

return [container[:]]