birdears Documentation

Release 0.1.3

Iacchus Mercurius

Dec 19, 2017

CONTENTS

1 Features **3**

2 Installing birdears **5**
- 2.1 Installing the dependencies . . . 5
  - 2.1.1 Arch Linux . . . 5
- 2.2 Installing birdears . . . 5
  - 2.2.1 In-depth installation . . . 5

3 Using birdears **7**
- 3.1 What is Functional Ear Training . . . 7
- 3.2 The method . . . 7
- 3.3 birdears modes and basic usage . . . 7
  - 3.3.1 melodic . . . 8
  - 3.3.2 harmonic . . . 9
  - 3.3.3 dictation . . . 10
  - 3.3.4 instrumental . . . 11
- 3.4 Loading from config/preset files . . . 12
  - 3.4.1 Pre-made presets . . . 12
  - 3.4.2 Creating new preset files . . . 12
- 3.5 Keybindings . . . 12
  - 3.5.1 On the keybindings . . . 12
  - 3.5.2 Major (Ionian) . . . 12
  - 3.5.3 Dorian . . . 14
  - 3.5.4 Phrygian . . . 14
  - 3.5.5 Lydian . . . 14
  - 3.5.6 Mixolydian . . . 14
  - 3.5.7 Minor (Aeolian) . . . 14
  - 3.5.8 Locrian . . . 14

4 birdears package **17**
- 4.1 Subpackages . . . 17
  - 4.1.1 birdears.interfaces package . . . 17
  - 4.1.2 birdears.questions package . . . 18
- 4.2 Submodules . . . 24
- 4.3 birdears.interval module . . . 24
- 4.4 birdears.logger module . . . 26
- 4.5 birdears.prequestion module . . . 27
- 4.6 birdears.questionbase module . . . 28
- 4.7 birdears.resolution module . . . 29
- 4.8 birdears.scale module . . . 30

4.9 birdears.sequence module . . . 31

5 Features **33**

6 Installing birdears **35**
6.1 Installing the dependencies . . . 35
6.1.1 Arch Linux . . . 35
6.2 Installing birdears . . . 35
6.2.1 In-depth installation . . . 35

7 Using birdears **37**
7.1 What is Functional Ear Training . . . 37
7.2 The method . . . 37
7.3 birdears modes and basic usage . . . 37
7.3.1 melodic . . . 38
7.3.2 harmonic . . . 39
7.3.3 dictation . . . 40
7.3.4 instrumental . . . 41
7.4 Loading from config/preset files . . . 42
7.4.1 Pre-made presets . . . 42
7.4.2 Creating new preset files . . . 42
7.5 Keybindings . . . 42
7.5.1 On the keybindings . . . 42
7.5.2 Major (Ionian) . . . 42
7.5.3 Dorian . . . 44
7.5.4 Phrygian . . . 44
7.5.5 Lydian . . . 44
7.5.6 Mixolydian . . . 44
7.5.7 Minor (Aeolian) . . . 44
7.5.8 Locrian . . . 44

8 API **47**

9 birdears package **49**
9.1 Subpackages . . . 50
9.2 Submodules . . . 50
9.3 birdears.interval module . . . 50
9.4 birdears.logger module . . . 52
9.5 birdears.prequestion module . . . 53
9.6 birdears.questionbase module . . . 54
9.7 birdears.resolution module . . . 55
9.8 birdears.scale module . . . 56
9.9 birdears.sequence module . . . 57

10 birdears.questions package **59**
10.1 Submodules . . . 59
10.2 birdears.questions.harmonicinterval module . . . 59
10.3 birdears.questions.instrumentaldictation module . . . 61
10.4 birdears.questions.melodicdictation module . . . 63
10.5 birdears.questions.melodicinterval module . . . 64

11 birdears.interfaces package **67**
11.1 Submodules . . . 67
11.2 birdears.interfaces.commandline module . . . 67

Python Module Index **69**

Index **71**

Welcome to birdears documentation.

`birdears` is a software written in Python 3 for ear training for musicians (musical intelligence, transcribing music, composing). It is a clone of the method used by Funcitional Ear Trainer app for Android.

It comes with four modes, or four kind of exercises, which are: `melodic`, `harmonic`, `dictation` and `instrumental`.

In resume, with the *melodic* mode two notes are played one after the other and you have to guess the interval; with the *harmonic* mode, two notes are played simoutaneously (harmonically) and you should guess the interval.

With the *dictation* mode, more than 2 notes are played (*ie*., a melodic dictation) and you should tell what are the intervals between them.

With the *instrumental* mode, it is a like the *dictation*, but you will be expected to play the notes on your instrument, *ie*., birdears will not wait for a typed reply and you should prectice with your own judgement. The melody can be repeat any times and you can have as much time as you want to try it out.

Project at GitHub.

Download the PDF version of this book. Clicking here.

CHAPTER

ONE

FEATURES

- questions
- pretty much configurable
- load from config file
- you can make your own presets
- can be used interactively *(docs needed)*
- can be used as a library *(docs needed)*

CHAPTER

TWO

INSTALLING BIRDEARS

2.1 Installing the dependencies

2.1.1 Arch Linux

```
sudo pacman -Syu sox python python-pip
```

2.2 Installing birdears

To install,simple do this command with pip3

```
pip3 install --user --upgrade --no-cache-dir birdears
```

2.2.1 In-depth installation

You can choose to use a virtualenv to use birdears; this should give you an idea on how to setup one virtualenv.

You should first install virtualenv (for python3) using your distribution's package (supposing you're on linux), then issue on terminal:

```
virtualenv -p python3 ~/.venv # use the directory ~/.venv/ for the virtualenv

source ~/.venv/bin/activate   # activate the virtualenv; this should be done
                              # every time you may want to run the software
                              # installed here.

pip3 install birdears         # this will install the software

birdears --help               # and this will run it
```

CHAPTER

THREE

USING BIRDEARS

3.1 What is Functional Ear Training

write me!

3.2 The method

We can use abc language to notate music withing the documentation, ok

```
X: 1
T: Banish Misfortune
R: jig
M: 6/8
L: 1/8
K: Dmix
fed cAG| A2d cAG| F2D DED| FEF GFG|
AGA cAG| AGA cde|fed cAG| Ad^c d3:|
f2d d^cd| f2g agf| e2c cBc|e2f gfe|
f2g agf| e2f gfe|fed cAG|Ad^c d3:|
f2g e2f| d2e c2d|ABA GAG| F2F GED|
c3 cAG| AGA cde| fed cAG| Ad^c d3:|
```

3.3 birdears modes and basic usage

birdears actually has four modes:

- melodic interval question
- harmonic interval question
- melodic dictation question
- instrumental dictation question

To see the commands avaliable just invoke the command without any arguments:

```
birdears
```

```
Usage: birdears  <command> [options]

  birdears — Functional Ear Training for Musicians!
```

```
Options:
  --debug / --no-debug  Turns on debugging; instead you can set DEBUG=1.
  -h, --help            Show this message and exit.

Commands:
  dictation     Melodic dictation
  harmonic      Harmonic interval recognition
  instrumental  Instrumental melodic time-based dictation
  load          Loads exercise from .toml config file...
  melodic       Melodic interval recognition

  You can use 'birdears <command> --help' to show options for a specific
  command.

  More info at https://github.com/iacchus/birdears
```

```
birdears <command> --help
```

3.3.1 melodic

In this exercise birdears will play two notes, the tonic and the interval melodically, ie., one after the other and you should reply which is the correct distance between the two.

```
birdears melodic --help
```

```
Usage: birdears melodic [options]

  Melodic interval recognition

Options:
  -m, --mode <mode>               Mode of the question.
  -t, --tonic <tonic>             Tonic of the question.
  -o, --octave <octave>           Octave of the question.
  -d, --descending                Whether the question interval is descending.
  -c, --chromatic                 If chosen, question has chromatic notes.
  -n, --n_octaves <n max>         Maximum number of octaves.
  -v, --valid_intervals <1,2,..>  A comma-separated list without spaces
                                  of valid scale degrees to be chosen for the
                                  question.
  -q, --user_durations <1,0.5,n..>
                                  A comma-separated list without
                                  spaces with PRECISLY 9 floating values. Or
                                  'n' for default              duration.
  -p, --prequestion_method <prequestion_method>
                                  The name of a pre-question method.
  -r, --resolution_method <resolution_method>
                                  The name of a resolution method.
  -h, --help                      Show this message and exit.

  In this exercise birdears will play two notes, the tonic and the interval
  melodically, ie., one after the other and you should reply which is the
  correct distance between the two.

  Valid values are as follows:
```

```
  -m <mode> is one of: major, dorian, phrygian, lydian, mixolydian, minor,
  locrian

  -t <tonic> is one of: A, A#, Ab, B, Bb, C, C#, D, D#, Db, E, Eb, F, F#, G,
  G#, Gb

  -p <prequestion_method> is one of: none, tonic_only, progression_i_iv_v_i

  -r <resolution_method> is one of: nearest_tonic, repeat_only
```

3.3.2 harmonic

In this exercise birdears will play two notes, the tonic and the interval harmonically, ie., both on the same time and you should reply which is the correct distance between the two.

```
birdears harmonic --help
```

```
Usage: birdears harmonic [options]

  Harmonic interval recognition

Options:
  -m, --mode <mode>               Mode of the question.
  -t, --tonic <note>              Tonic of the question.
  -o, --octave <octave>           Octave of the question.
  -d, --descending                Whether the question interval is descending.
  -c, --chromatic                 If chosen, question has chromatic notes.
  -n, --n_octaves <n max>         Maximum number of octaves.
  -v, --valid_intervals <1,2,..>  A comma-separated list without spaces
                                  of valid scale degrees to be chosen for the
                                  question.
  -q, --user_durations <1,0.5,n..>
                                  A comma-separated list without
                                  spaces with PRECISLY 9 floating values. Or
                                  'n' for default              duration.
  -p, --prequestion_method <prequestion_method>
                                  The name of a pre-question method.
  -r, --resolution_method <resolution_method>
                                  The name of a resolution method.
  -h, --help                      Show this message and exit.

  In this exercise birdears will play two notes, the tonic and the interval
  harmonically, ie., both on the same time and you should reply which is the
  correct distance between the two.

  Valid values are as follows:

  -m <mode> is one of: major, dorian, phrygian, lydian, mixolydian, minor,
  locrian

  -t <tonic> is one of: A, A#, Ab, B, Bb, C, C#, D, D#, Db, E, Eb, F, F#, G,
  G#, Gb

  -p <prequestion_method> is one of: none, tonic_only, progression_i_iv_v_i
```

```
  -r <resolution_method> is one of: nearest_tonic, repeat_only
```

3.3.3 dictation

In this exercise birdears will choose some random intervals and create a melodic dictation with them. You should reply the correct intervals of the melodic dictation.

```
birdears dictation --help
```

```
Usage: birdears dictation [options]

  Melodic dictation

Options:
  -m, --mode <mode>               Mode of the question.
  -i, --max_intervals <n max>     Max random intervals for the dictation.
  -x, --n_notes <n notes>         Number of notes for the dictation.
  -t, --tonic <note>              Tonic of the question.
  -o, --octave <octave>           Octave of the question.
  -d, --descending                Wether the question interval is descending.
  -c, --chromatic                 If chosen, question has chromatic notes.
  -n, --n_octaves <n max>         Maximum number of octaves.
  -v, --valid_intervals <1,2,..>  A comma-separated list without spaces
                                  of valid scale degrees to be chosen for the
                                  question.
  -q, --user_durations <1,0.5,n..>
                                  A comma-separated list without
                                  spaces with PRECISLY 9 floating values. Or
                                  'n' for default              duration.
  -p, --prequestion_method <prequestion_method>
                                  The name of a pre-question method.
  -r, --resolution_method <resolution_method>
                                  The name of a resolution method.
  -h, --help                      Show this message and exit.

  In this exercise birdears will choose some random intervals and create a
  melodic dictation with them. You should reply the correct intervals of the
  melodic dictation.

  Valid values are as follows:

  -m <mode> is one of: major, dorian, phrygian, lydian, mixolydian, minor,
  locrian

  -t <tonic> is one of: A, A#, Ab, B, Bb, C, C#, D, D#, Db, E, Eb, F, F#, G,
  G#, Gb

  -p <prequestion_method> is one of: none, tonic_only, progression_i_iv_v_i

  -r <resolution_method> is one of: nearest_tonic, repeat_only
```

3.3.4 instrumental

In this exercise birdears will choose some random intervals and create a melodic dictation with them. You should play the correct melody in you musical instrument.

```
birdears instrumental --help
```

```
Usage: birdears instrumental [options]

  Instrumental melodic time-based dictation

Options:
  -m, --mode <mode>                Mode of the question.
  -w, --wait_time <seconds>        Time in seconds for next question/repeat.
  -u, --n_repeats <times>          Times to repeat question.
  -i, --max_intervals <n max>      Max random intervals for the dictation.
  -x, --n_notes <n notes>          Number of notes for the dictation.
  -t, --tonic <note>               Tonic of the question.
  -o, --octave <octave>            Octave of the question.
  -d, --descending                 Wether the question interval is descending.
  -c, --chromatic                  If chosen, question has chromatic notes.
  -n, --n_octaves <n max>          Maximum number of octaves.
  -v, --valid_intervals <1,2,..>   A comma-separated list without spaces
                                   of valid scale degrees to be chosen for the
                                   question.
  -q, --user_durations <1,0.5,n..>
                                   A comma-separated list without
                                   spaces with PRECISLY 9 floating values. Or
                                   'n' for default              duration.
  -p, --prequestion_method <prequestion_method>
                                   The name of a pre-question method.
  -r, --resolution_method <resolution_method>
                                   The name of a resolution method.
  -h, --help                       Show this message and exit.

  In this exercise birdears will choose some random intervals and create a
  melodic dictation with them. You should play the correct melody in you
  musical instrument.

  Valid values are as follows:

  -m <mode> is one of: major, dorian, phrygian, lydian, mixolydian, minor,
  locrian

  -t <tonic> is one of: A, A#, Ab, B, Bb, C, C#, D, D#, Db, E, Eb, F, F#, G,
  G#, Gb

  -p <prequestion_method> is one of: none, tonic_only, progression_i_iv_v_i

  -r <resolution_method> is one of: nearest_tonic, repeat_only
```

3.4 Loading from config/preset files

3.4.1 Pre-made presets

`birdears` cointains some pre-made presets in it's `presets/` subdirectory.

The study for beginners is recommended by following the numeric order of those files (000, 001, then 002 etc.)

Pre-made presets description

write me

3.4.2 Creating new preset files

You can open the files cointained in birdears premade `presets/` folder to have an ideia on how config files are made; it is simply the command line options written in a form `toml` standard.

3.5 Keybindings

3.5.1 On the keybindings

The following keyboard diagrams should give you an idea on how the keybindings work. Please note how the keys on the line from `z` (*unison*) to `,` (comma, *octave*) represent the notes that are *natural* to the mode, and the line above represent the chromatics.

Also, for exercises with two octaves, the **uppercased keys represent the second octave**. For example, `z` is *unison*, `,` is the *octave*, `Z` (uppercased) is the *double octave*. The same for all the other intervals.

3.5.2 Major (Ionian)

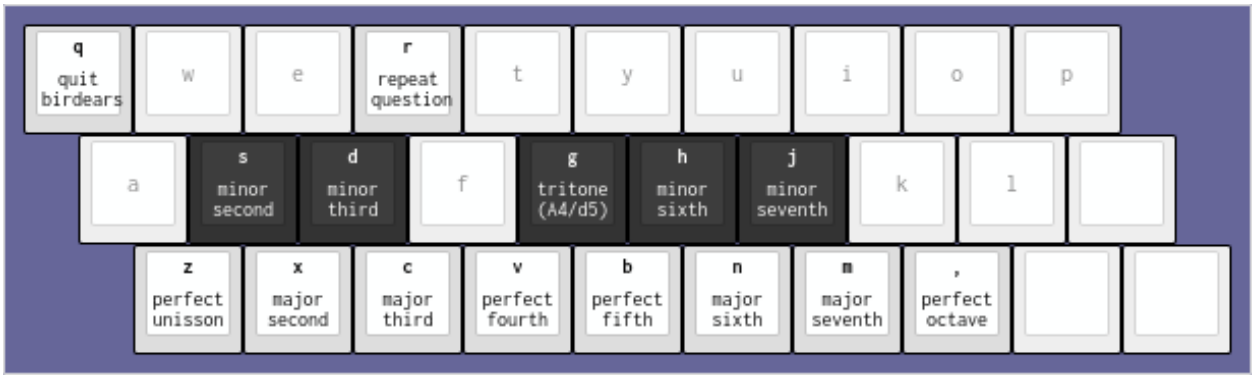

Fig. 3.1: Keyboard diagram for the `--mode major` (default).

Fig. 3.2: Keyboard diagram for the `--mode dorian`.

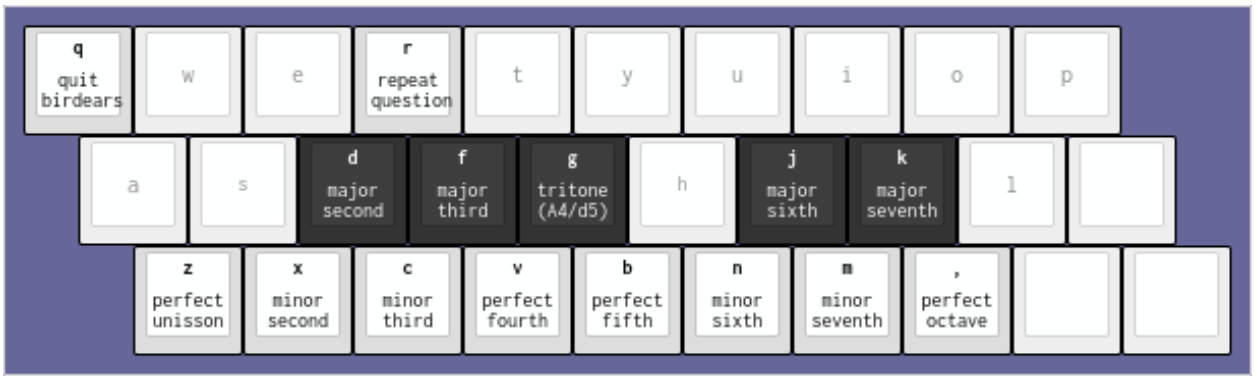

Fig. 3.3: Keyboard diagram for the `--mode phrygian`.

3.5.3 Dorian

3.5.4 Phrygian

3.5.5 Lydian

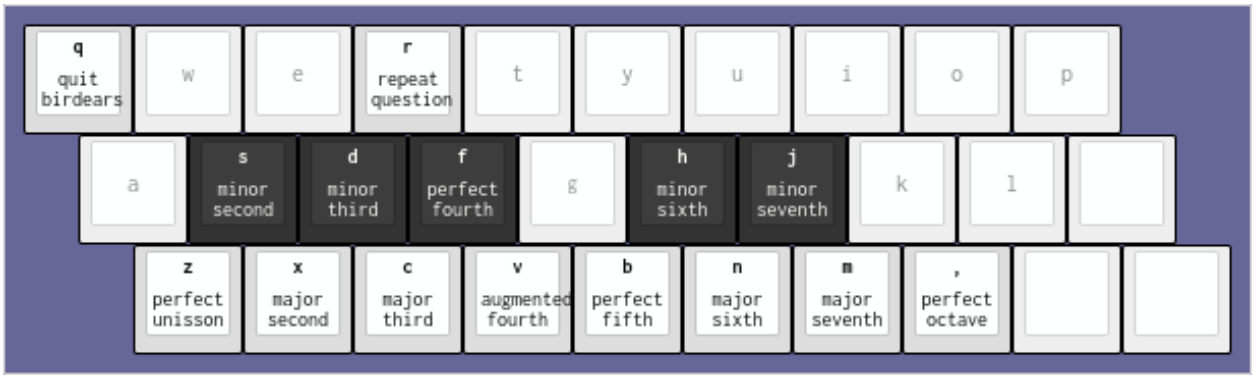

Fig. 3.4: Keyboard diagram for the `--mode lydian`.

3.5.6 Mixolydian

Fig. 3.5: Keyboard diagram for the `--mode mixolydian`.

3.5.7 Minor (Aeolian)

3.5.8 Locrian

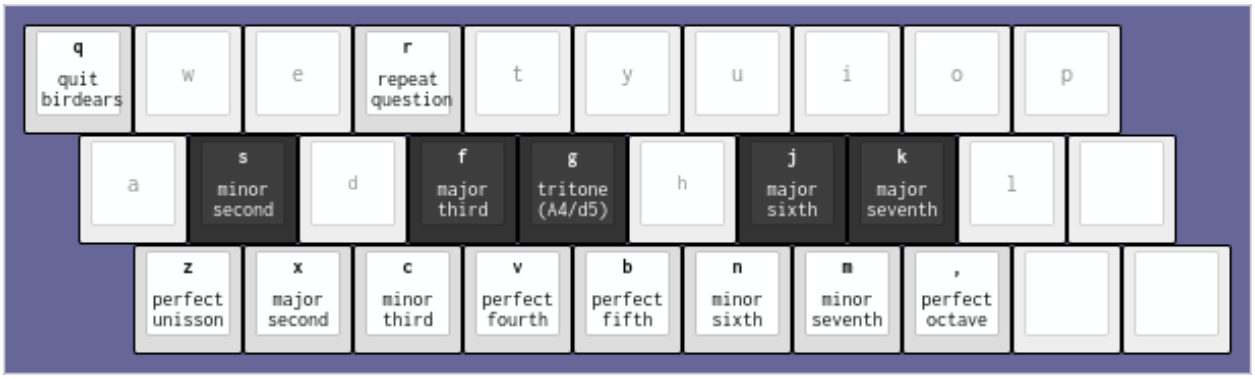

Fig. 3.6: Keyboard diagram for the `--mode minor`.

Fig. 3.7: Keyboard diagram for the `--mode locrian`.

CHAPTER

FOUR

BIRDEARS PACKAGE

birdears provides facilities to building musical ear training exercises.

birdears.**CHROMATIC_FLAT = ('C', 'Db', 'D', 'Eb', 'E', 'F', 'Gb', 'G', 'Ab', 'A', 'Bb', 'B')**
tuple – Chromatic notes names using flats.

A mapping of the chromatic note names using flats.

birdears.**CHROMATIC_SHARP = ('C', 'C#', 'D', 'D#', 'E', 'F', 'F#', 'G', 'G#', 'A', 'A#', 'B**
tuple – Chromatic notes names using sharps.

A mapping of the chromatic note namesu sing sharps

birdears.**CHROMATIC_TYPE = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12)**
tuple – A map of the chromatic chromatic scale.

A map of the the semitones which compound the chromatic scale.

birdears.**CIRCLE_OF_FIFTHS = [('C', 'G', 'D', 'A', 'E', 'B', 'Gb', 'Db', 'Ab', 'Eb', 'Bb',**
list of tuples – Circle of fifths.

These are the circle of fifth in both directions.

birdears.**DIATONIC_MODES = {'major': (0, 2, 4, 5, 7, 9, 11, 12), 'dorian': (0, 2, 3, 5, 7,**
dict of tuples – A map of the diatonic scale.

A mapping of the semitones which compound each of the greek modes.

birdears.**INTERVALS = ((0, 'P1', 'Perfect Unison'), (1, 'm2', 'Minor Second'), (2, 'M2', 'Ma**
tuple of tuples – Data representing intervals.

A tuple of tuples representing data for the intervals with format (semitones, short name, full name).

birdears.**INTERVAL_INDEX = {1: [0], 2: [1, 2], 3: [3, 4], 4: [5, 6], 5: [6, 7], 6: [8,**
dict of lists – A mapping of semitones of each interval.

A mapping of semitones which index to each interval name, major/minor, perfect, augmented/diminished

birdears.**KEYS = ('C', 'C#', 'Db', 'D', 'D#', 'Eb', 'E', 'F', 'F#', 'Gb', 'G', 'G#', 'Ab',**
tuple – Allowed keys

These are the allowed keys for exercise as comprehended by birdears.

4.1 Subpackages

4.1.1 birdears.interfaces package

Submodules

birdears.interfaces.commandline module

birdears.interfaces.commandline.**CommandLine**(*exercise*, ***kwargs*)
This function implements the birdears loop for command line.

Parameters

- **exercise** (*str*) – The question name.
- ****kwargs** (*kwargs*) – FIXME: The kwargs can contain options for specific questions.

birdears.interfaces.commandline.**center_text**(*text*, *sep=True*, *nl=0*)
This function returns input text centered according to terminal columns.

Parameters

- **text** (*str*) – The string to be centered, it can have multiple lines.
- **sep** (*bool*) – Add line separator after centered text (True) or not (False).
- **nl** (*int*) – How many new lines to add after text.

birdears.interfaces.commandline.**make_input_str**(*user_input*, *keyboard_index*)
Makes a string representing intervals entered by the user.

This function is to be used by questions which takes more than one interval input as MelodicDictation, and formats the intervals already entered.

Parameters

- **user_input** (*array_type*) – The list of keyboard keys entered by user.
- **keyboard_index** (*array_type*) – The keyboard mapping used by question.

birdears.interfaces.commandline.**print_instrumental**(*response*)
Prints the formatted response for 'instrumental' exercise.

Parameters **response** (*dict*) – A response returned by question's check_question()

birdears.interfaces.commandline.**print_question**(*question*)
Prints the question to the user.

Parameters **question** (*obj*) – A Question class with the question to be printed.

birdears.interfaces.commandline.**print_response**(*response*)
Prints the formatted response.

Parameters **response** (*dict*) – A response returned by question's check_question()

4.1.2 birdears.questions package

Submodules

birdears.questions.harmonicinterval module

class birdears.questions.harmonicinterval.**HarmonicIntervalQuestion**(*mode='major'*, *tonic=None*, *octave=None*, *descending=None*, *chromatic=None*, *n_octaves=None*, *valid_intervals=None*, *user_durations=None*, *prequestion_method='none'*, *resolution_method='nearest_tonic'*, **args*, ***kwargs*)

Bases: *birdears.questionbase.QuestionBase*

Implements a Harmonic Interval test.

__init__(*mode='major'*, *tonic=None*, *octave=None*, *descending=None*, *chromatic=None*, *n_octaves=None*, *valid_intervals=None*, *user_durations=None*, *prequestion_method='none'*, *resolution_method='nearest_tonic'*, **args*, ***kwargs*)

Inits the class.

Parameters

- **mode** (*str*) – A string representing the mode of the question. Eg., 'major' or 'minor'
- **tonic** (*str*) – A string representing the tonic of the question, eg.: 'C'; if omitted, it will be selected randomly.
- **octave** (*int*) – A scienfic octave notation, for example, 4 for 'C4'; if not present, it will be randomly chosen.
- **descending** (*bool*) – Is the question direction in descending, ie., intervals have lower pitch than the tonic.
- **chromatic** (*bool*) – If the question can have (True) or not (False) chromatic intervals, ie., intervals not in the diatonic scale of tonic/mode.
- **n_octaves** (*int*) – Maximum number of octaves of the question.
- **valid_intervals** (*list*) – A list with intervals (int) valid for random choice, 1 is 1st, 2 is second etc. Eg. [1, 4, 5] to allow only tonics, fourths and fifths.
- **user_durations** (*str*) – A string with 9 comma-separated *int* or *float's to set the default duration for the notes played. The values are respectively for: pre-question duration (1st), pre-question delay (2nd), and pre-question pos-delay (3rd); question duration (4th), question delay (5th), and question pos-delay (6th); resolution duration (7th), resolution delay (8th), and resolution pos-delay (9th). duration is the duration in of the note in seconds; delay is the time to wait before playing the next note, and pos_delay is the time to wait after all the notes of the respective sequence have been played. If any of the user durations is 'n*, the default duration for the type of question will be used instead. Example:

```
"2,0.5,1,2,n,0,2.5,n,1"
```

- **prequestion_method** (*str*) – Method of playing a cadence or the exercise tonic before the question so to affirm the question musical tonic key to the ear. Valid ones are registered in the *birdears.prequestion.PREQUESION_METHODS* global dict.
- **resolution_method** (*str*) – Method of playing the resolution of an exercise. Valid ones are registered in the *birdears.resolution.RESOLUTION_METHODS* global dict.

check_question(*user_input_char*)
Checks whether the given answer is correct.

make_pre_question(*method*)

make_question()

make_resolution(*method*)

play_question()

play_resolution()

birdears.questions.instrumentaldictation module

class birdears.questions.instrumentaldictation.**InstrumentalDictationQuestion**(*mode='major'*, *wait_time=11*, *n_repeats=1*, *max_intervals=3*, *n_notes=4*, *tonic=None*, *octave=None*, *descending=None*, *chromatic=None*, *n_octaves=None*, *valid_intervals=No*
user_durations=No
prequestion_method='progr
resolution_method='repea
**args*, ***kwargs*)

Bases: *birdears.questionbase.QuestionBase*

Implements an instrumental dictation test.

__init__(*mode='major'*, *wait_time=11*, *n_repeats=1*, *max_intervals=3*, *n_notes=4*, *tonic=None*, *octave=None*, *descending=None*, *chromatic=None*, *n_octaves=None*, *valid_intervals=None*, *user_durations=None*, *prequestion_method='progression_i_iv_v_i'*, *resolution_method='repeat_only'*, **args*, ***kwargs*)
Inits the class.

Parameters

- **mode** (*str*) – A string representing the mode of the question. Eg., 'major' or 'minor'.
- **wait_time** (*float*) – Wait time in seconds for the next question or repeat.
- **n_repeats** (*int*) – Number of times the same dictation will be repeated before the end of the exercise.
- **max_intervals** (*int*) – The maximum number of random intervals the question will have.
- **n_notes** (*int*) – The number of notes the melodic dictation will have.
- **tonic** (*str*) – A string representing the tonic of the question, eg.: 'C'; if omitted, it will be selected randomly.
- **octave** (*int*) – A scienfic octave notation, for example, 4 for 'C4'; if not present, it will be randomly chosen.
- **descending** (*bool*) – Is the question direction in descending, ie., intervals have lower pitch than the tonic.
- **chromatic** (*bool*) – If the question can have (True) or not (False) chromatic intervals, ie., intervals not in the diatonic scale of tonic/mode.
- **n_octaves** (*int*) – Maximum number of octaves of the question.
- **valid_intervals** (*list*) – A list with intervals (int) valid for random choice, 1 is 1st, 2 is second etc. Eg. [1, 4, 5] to allow only tonics, fourths and fifths.
- **user_durations** (*str*) – A string with 9 comma-separated *int* or *float's to set the default duration for the notes played. The values are respectively for: pre-question duration (1st), pre-question delay (2nd), and pre-question pos-delay (3rd); question duration (4th), question delay (5th), and question pos-delay (6th); resolution duration (7th), resolution delay (8th), and resolution pos-delay (9th). duration is the duration in of the note in seconds; delay is the time to wait before playing the next note, and pos_delay is the time to wait after all the notes of the respective sequence have been played. If any of the user durations is 'n*, the default duration for the type of question will be used instead. Example:

```
"2,0.5,1,2,n,0,2.5,n,1"
```

- **prequestion_method** (*str*) – Method of playing a cadence or the exercise tonic before the question so to affirm the question musical tonic key to the ear. Valid ones are registered in the *birdears.prequestion.PREQUESION_METHODS* global dict.
- **resolution_method** (*str*) – Method of playing the resolution of an exercise. Valid ones are registered in the *birdears.resolution.RESOLUTION_METHODS* global dict.

check_question()

Checks whether the given answer is correct.

This currently doesn't applies to instrumental dictation questions.

make_pre_question(*method*)

make_question(*phrase_semitones*)

make_resolution(*method*)

play_question()

birdears.questions.melodicdictation module

class birdears.questions.melodicdictation.**MelodicDictationQuestion**(*mode='major'*, *max_intervals=3*, *n_notes=4*, *tonic=None*, *octave=None*, *descending=None*, *chromatic=None*, *n_octaves=None*, *valid_intervals=None*, *user_durations=None*, *prequestion_method='progression_i_iv_v_*
resolution_method='repeat_only', **args*, ***kwargs*)

Bases: *birdears.questionbase.QuestionBase*

Implements a melodic dictation test.

__init__(*mode='major'*, *max_intervals=3*, *n_notes=4*, *tonic=None*, *octave=None*, *descending=None*, *chromatic=None*, *n_octaves=None*, *valid_intervals=None*, *user_durations=None*, *prequestion_method='progression_i_iv_v_i'*, *resolution_method='repeat_only'*, **args*, ***kwargs*)

Inits the class.

Parameters

- **mode** (*str*) – A string representing the mode of the question. Eg., 'major' or 'minor'.
- **max_intervals** (*int*) – The maximum number of random intervals the question will have.
- **n_notes** (*int*) – The number of notes the melodic dictation will have.
- **tonic** (*str*) – A string representing the tonic of the question, eg.: 'C'; if omitted, it will be selected randomly.
- **octave** (*int*) – A scienfic octave notation, for example, 4 for 'C4'; if not present, it will be randomly chosen.
- **descending** (*bool*) – Is the question direction in descending, ie., intervals have lower pitch than the tonic.
- **chromatic** (*bool*) – If the question can have (True) or not (False) chromatic intervals, ie., intervals not in the diatonic scale of tonic/mode.
- **n_octaves** (*int*) – Maximum number of octaves of the question.
- **valid_intervals** (*list*) – A list with intervals (int) valid for random choice, 1 is 1st, 2 is second etc. Eg. [1, 4, 5] to allow only tonics, fourths and fifths.
- **user_durations** (*str*) – A string with 9 comma-separated *int* or *float's to set the default duration for the notes played. The values are respectively for: pre-question duration (1st), pre-question delay (2nd), and pre-question pos-delay (3rd); question duration*

(4th), question delay (5th), and question pos-delay (6th); resolution duration (7th), resolution delay (8th), and resolution pos-delay (9th). duration is the duration in of the note in seconds; delay is the time to wait before playing the next note, and pos_delay is the time to wait after all the notes of the respective sequence have been played. If any of the user durations is 'n, the default duration for the type of question will be used instead. Example:

```
"2,0.5,1,2,n,0,2.5,n,1"
```

- **prequestion_method** (*str*) – Method of playing a cadence or the exercise tonic before the question so to affirm the question musical tonic key to the ear. Valid ones are registered in the *birdears.prequestion.PREQUESION_METHODS* global dict.
- **resolution_method** (*str*) – Method of playing the resolution of an exercise. Valid ones are registered in the *birdears.resolution.RESOLUTION_METHODS* global dict.

check_question(*user_input_keys*)
Checks whether the given answer is correct.

make_pre_question(*method*)

make_question(*phrase_semitones*)

make_resolution(*method*)

play_question()

play_resolution()

birdears.questions.melodicinterval module

class birdears.questions.melodicinterval.**MelodicIntervalQuestion**(*mode='major'*, *tonic=None*, *octave=None*, *descending=None*, *chromatic=None*, *n_octaves=None*, *valid_intervals=None*, *user_durations=None*, *prequestion_method='tonic_only'*, *resolution_method='nearest_tonic'*, **args*, ***kwargs*)

Bases: *birdears.questionbase.QuestionBase*

Implements a Melodic Interval test.

check_question(*user_input_char*)
Checks whether the given answer is correct.

make_pre_question(*method*)

make_question()

make_resolution(*method*)

play_question()

play_resolution()

4.2 Submodules

4.3 birdears.interval module

class birdears.interval.**ChromaticInterval**(*mode*, *tonic*, *octave*, *n_octaves=None*, *descending=None*, *valid_intervals=None*)

Bases: *birdears.interval.IntervalBase*

Chooses a diatonic interval for the question.

tonic_octave
int – Scientific octave for the tonic. For example, if the tonic is a 'C4' then *tonic_octave* is 4.

interval octave
int – Scientific octave for the interval. For example, if the interval is a 'G5' then *tonic_octave* is 5.

chromatic_offset
int – The offset in semitones inside one octave; maybe it will be deprecated in favour of *distance['semitones']* which is the same.

note_and_octave
str – Note and octave of the interval, for example, if the interval is G5 the note name is 'G5'.

note_name
str – The note name of the interval, for example, if the interval is G5 then the name is 'G'.

semitones
int – Semitones from tonic to octave. If tonic is C4 and interval is G5 the number of semitones is 19.

is_chromatic
bool – If the current interval is chromatic (True) or if it exists in the diatonic scale which key is tonic.

is_descending
bool – If the interval has a descending direction, ie., has a lower pitch than the tonic.

diatonic_index
int – If the interval is chromatic, this will be the nearest diatonic interval in the direction of the resolution (closest tonic.) From II to IV degrees, it is the ditonic interval before; from V to VII it is the diatonic interval after.

distance
dict – A dictionary which the distance from tonic to interval, for example, if tonic is C4 and interval is G5:

```
{
    'octaves': 1,
    'semitones': 7
}
```

data
tuple – A tuple representing the interval data in the form of (semitones, short_name, long_name), for example:

```
(19, 'P12', 'Perfect Twelfth')
```

Todo:

- **Maybe we should refactor some of the attributes with a tuple** (note, octave)
- Maybe remove *chromatic_offset* in favor of *distance['semitones']*

__init__(*mode*, *tonic*, *octave*, *n_octaves=None*, *descending=None*, *valid_intervals=None*)
Inits the class and choses a random interval with the given args.

Parameters

- **mode** (*str*) – Diatonic mode for the interval. (eg.: 'major' or 'minor')
- **tonic** (*str*) – Tonic of the scale. (eg.: 'Bb')
- **octave** (*str*) – Scientific octave of the scale (eg.: 4)
- **interval** (*str*) – Not implemented. The interval.
- **chromatic** (*bool*) – Can have chromatic notes? (eg.: F# in a key of C; default: false)
- **n_octaves** (*int*) – Maximum number os octaves (eg. 2)
- **descending** (*bool*) – Is the interval descending? (default: false)
- **valid_intervals** (*int*) – A list with inervals valid for random choice, 1 is 1st, 2 is second etc.

class birdears.interval.**DiatonicInterval**(*mode*, *tonic*, *octave*, *n_octaves=None*, *descending=None*, *valid_intervals=None*)
Bases: *birdears.interval.IntervalBase*

Chooses a diatonic interval for the question.

tonic_octave
int – Scientific octave for the tonic. For example, if the tonic is a 'C4' then *tonic_octave* is 4.

interval octave
int – Scientific octave for the interval. For example, if the interval is a 'G5' then *tonic_octave* is 5.

chromatic_offset
int – The offset in semitones inside one octave. Relative semitones to tonic.

note_and_octave
str – Note and octave of the interval, for example, if the interval is G5 the note name is 'G5'.

note_name
str – The note name of the interval, for example, if the interval is G5 then the name is 'G'.

semitones
int – Semitones from tonic to octave. If tonic is C4 and interval is G5 the number of semitones is 19.

is_chromatic
bool – If the current interval is chromatic (True) or if it exists in the diatonic scale which key is tonic.

is_descending
bool – If the interval has a descending direction, ie., has a lower pitch than the tonic.

diatonic_index
int – If the interval is chromatic, this will be the nearest diatonic interval in the direction of the resolution (closest tonic.) From II to IV degrees, it is the ditonic interval before; from V to VII it is the diatonic interval after.

distance
dict – A dictionary which the distance from tonic to interval, for example, if tonic is C4 and interval is G5:

```
{
    'octaves': 1,
    'semitones': 7
}
```

data
tuple – A tuple representing the interval data in the form of (semitones, short_name, long_name), for example:

```
(19, 'P12', 'Perfect Twelfth')
```

__init__(*mode*, *tonic*, *octave*, *n_octaves=None*, *descending=None*, *valid_intervals=None*)
Inits the class and choses a random interval with the given args.

Parameters

- **mode** (*str*) – Diatonic mode for the interval. (eg.: 'major' or 'minor')
- **tonic** (*str*) – Tonic of the scale. (eg.: 'Bb')
- **octave** (*str*) – Scientific octave of the scale (eg.: 4)
- **n_octaves** (*int*) – Maximum number os octaves (eg. 2)
- **descending** (*bool*) – Is the interval descending? (default: false)
- **valid_intervals** (*int*) – A list with intervals (int) valid for random choice, 1 is 1st, 2 is second etc.

class birdears.interval.**IntervalBase**
Bases: object

__init__()
Base class for interval classes.

return_simple(*keys*)
This method returns a dict with only the values passed to *keys*.

4.4 birdears.logger module

This submodule exports *logger* to log events.

Logging messages which are less severe than *lvl* will be ignored:

```
Level       Numeric value
-----       -------------
CRITICAL    50
ERROR       40
WARNING     30
INFO        20
DEBUG       10
NOTSET      0

Level       When it's used
-----       --------------
DEBUG       Detailed information, typically of interest only when
                diagnosing problems.
```

```
INFO        Confirmation that things are working as expected.
WARNING     An indication that something unexpected happened, or indicative
                of some problem in the near future (e.g. 'disk space low').
                The software is still working as expected.
ERROR       Due to a more serious problem, the software has not been able
                to perform some function.
CRITICAL    A serious error, indicating that the program itself may be
                unable to continue running.
```

birdears.logger.**log_event**(*f*, **args*, ***kwargs*)

Decorator. Functions and method decorated with this decorator will have their signature logged when birdears is executed with *–debug* mode. Both function signature with their call values and their return will be logged.

4.5 birdears.prequestion module

This module implements pre-questions' progressions.

Pre questions are chord progressions or notes played before the question is played, so to affirmate the sound of the question's key.

For example a common cadence is chords I-IV-V-I from the diatonic scale, which in a key of *C* is *CM-FM-GM-CM* and in a key of *A* is *AM-DM-EM-AM*.

Pre-question methods should be decorated with *register_prequestion_method* decorator, so that they will be registered as a valid pre-question method.

class birdears.prequestion.**PreQuestion**(*method*, *question*)

Bases: object

__call__(**args*, ***kwargs*)

Calls the resolution method and pass arguments to it.

Returns a *birdears.Sequence* object with the pre-question generated by the method.

__init__(*method*, *question*)

This class implements methods for different types of pre-question progressions.

Parameters

- **method** (*str*) – The method used in the pre question.
- **question** (*obj*) – Question object from which to generate the
- **sequence.** (*pre-question*) –

birdears.prequestion.**none**(*question*, **args*, ***kwargs*)

Pre-question method that return an empty sequence with no delay. :param question: Question object from which to generate the

pre-question sequence. (this is provided by the *Resolution* class when it is '__call__'ed)

birdears.prequestion.**progression_i_iv_v_i**(*question*, **args*, ***kwargs*)

Pre-question method that play's a chord progression with triad chords built on the grades I, IV, V the I of the question key.

Parameters **question** (*obj*) – Question object from which to generate the pre-question sequence. (this is provided by the *Resolution* class when it is '__call__'ed)

birdears.prequestion.**register_prequestion_method**(*f*, **args*, ***kwargs*)
Decorator for prequestion method functions.

Functions decorated with this decorator will be registered in the *PREQUESTION_METHODS* global dict.

birdears.prequestion.**tonic_only**(*question*, **args*, ***kwargs*)
Pre-question method that only play's the question tonic note before the question.

Parameters **question** (*obj*) – Question object from which to generate the pre-question sequence. (this is provided by the *Resolution* class when it is '__call__'ed)

4.6 birdears.questionbase module

class birdears.questionbase.**QuestionBase**(*mode='major'*, *tonic=None*, *octave=None*, *descending=None*, *chromatic=None*, *n_octaves=None*, *valid_intervals=None*, *user_durations=None*, *prequestion_method=None*, *resolution_method=None*, *default_durations=None*, **args*, ***kwargs*)
Bases: object

Base Class to be subclassed for Question classes.

This class implements attributes and routines to be used in Question subclasses.

__init__(*mode='major'*, *tonic=None*, *octave=None*, *descending=None*, *chromatic=None*, *n_octaves=None*, *valid_intervals=None*, *user_durations=None*, *prequestion_method=None*, *resolution_method=None*, *default_durations=None*, **args*, ***kwargs*)
Inits the class.

Parameters

- **mode** (*str*) – A string represnting the mode of the question. Eg., 'major' or 'minor'
- **tonic** (*str*) – A string representing the tonic of the question, eg.: 'C'; if omitted, it will be selected randomly.
- **octave** (*int*) – A scienfic octave notation, for example, 4 for 'C4'; if not present, it will be randomly chosen.
- **descending** (*bool*) – Is the question direction in descending, ie., intervals have lower pitch than the tonic.
- **chromatic** (*bool*) – If the question can have (True) or not (False) chromatic intervals, ie., intervals not in the diatonic scale of tonic/mode.
- **n_octaves** (*int*) – Maximum numbr of octaves of the question.
- **valid_intervals** (*list*) – A list with intervals (int) valid for random choice, 1 is 1st, 2 is second etc. Eg. [1, 4, 5] to allow only tonics, fourths and fifths.
- **user_durations** (*dict*) – A string with 9 comma-separated *int* or *float's to set the default duration for the notes played. The values are respectively for: pre-question duration (1st), pre-question delay (2nd), and pre-question pos-delay (3rd); question duration (4th), question delay (5th), and question pos-delay (6th); resolution duration (7th), resolution delay (8th), and resolution pos-delay (9th). duration is the duration in of the note in seconds; delay is the time to wait before playing the next note, and pos_delay is the time to wait after all the notes of the respective sequence have been played. If any of the user durations is 'n*, the default duration for the type of question will be used instead. Example:

```
"2,0.5,1,2,n,0,2.5,n,1"
```

- **prequestion_method** (*str*) – Method of playing a cadence or the exercise tonic before the question so to affirm the question musical tonic key to the ear. Valid ones are registered in the *birdears.prequestion.PREQUESION_METHODS* global dict.
- **resolution_method** (*str*) – Method of playing the resolution of an exercise Valid ones are registered in the *birdears.resolution.RESOLUTION_METHODS* global dict.
- **user_durations** – Dictionary with the default durations for each type of sequence. This is provided by the subclasses.

check_question()
This method should be overwritten by the question subclasses.

get_valid_semitones()
Returns a list with valid semitones for question.

make_question()
This method should be overwritten by the question subclasses.

make_resolution()
This method should be overwritten by the question subclasses.

play_question()
This method should be overwritten by the question subclasses.

birdears.questionbase.**register_question_class**(*f*, **args*, ***kwargs*)
Decorator for question classes.

Classes decorated with this decorator will be registered in the *QUESTION_CLASSES* global.

4.7 birdears.resolution module

class birdears.resolution.**Resolution**(*method*, *question*)
Bases: object

This class implements methods for different types of question resolutions.

A resolution is an answer to a question. It aims to create a mnemonic on how the inverval resvolver to the tonic.

__call__(**args*, ***kwargs*)
Calls the resolution method and pass arguments to it.

Returns a *birdears.Sequence* object with the resolution generated by the.method.

__init__(*method*, *question*)
Inits the resolution class.

Parameters

- **method** (*str*) – The method used in the resolution.
- **question** (*obj*) – Question object from which to generate the
- **sequence.** (resolution) –

birdears.resolution.**nearest_tonic**(*question*)
Resolution method that resolve the intervals to their nearest tonics.

Parameters **question** (*obj*) – Question object from which to generate the resolution sequence. (this is provided by the *Prequestion* class when it is '__call__'ed)

birdears.resolution.**register_resolution_method**(*f*, **args*, ***kwargs*)
Decorator for resolution method functions.

Functions decorated with this decorator will be registered in the *RESOLUTION_METHODS* global dict.

birdears.resolution.**repeat_only**(*question*)
Resolution method that only repeats the sequence elements with given durations.

Parameters **question** (*obj*) – Question object from which to generate the resolution sequence. (this is provided by the *Prequestion* class when it is '__call__'ed)

4.8 birdears.scale module

class birdears.scale.**ChromaticScale**(*tonic*, *octave=None*, *n_octaves=None*, *descending=None*, *dont_repeat_tonic=None*)
Bases: *birdears.scale.ScaleBase*

Builds a musical chromatic scale.

scale
array_type – The array of notes representing the scale.

__init__(*tonic*, *octave=None*, *n_octaves=None*, *descending=None*, *dont_repeat_tonic=None*)
Returns a chromatic scale from tonic.

Parameters

- **tonic** (*str*) – The note which the scale will be built upon.
- **octave** (*int*) – The scientific octave the scale will be built upon.
- **n_octaves** (*int*) – The number of octaves the scale will contain.
- **descending** (*bool*) – Whether the scale is descending.
- **dont_repeat_tonic** (*bool*) – Whether to skip appending the last note (octave) to the scale.

get_triad(*mode*, *index=0*, *degree=None*)
Returns an array with notes from a scale's triad.

Parameters

- **mode** (*str*) – Mode of the scale (eg. 'major' or 'minor')
- **index** (*int*) – Triad index (eg.: 0 for 1st degree triad.)
- **degree** (*int*) – Degree of the scale. If provided, overrides the *index* argument. (eg.: *1* for the 1st degree triad.)

Returns A list with three pitches (str), one for each note of the triad.

class birdears.scale.**DiatonicScale**(*tonic*, *mode=None*, *octave=None*, *n_octaves=None*, *descending=None*, *dont_repeat_tonic=None*)
Bases: *birdears.scale.ScaleBase*

Builds a musical diatonic scale.

scale
array_type – The array of notes representing the scale.

__init__(*tonic*, *mode=None*, *octave=None*, *n_octaves=None*, *descending=None*, *dont_repeat_tonic=None*)
Returns a diatonic scale from tonic and mode.

Parameters

- **tonic** (*str*) – The note which the scale will be built upon.
- **mode** (*str*) – The mode the scale will be built upon. ('major' or 'minor')
- **octave** (*int*) – The scientific octave the scale will be built upon.
- **n_octaves** (*int*) – The number of octaves the scale will contain.
- **descending** (*bool*) – Whether the scale is descending.
- **dont_repeat_tonic** (*bool*) – Whether to skip appending the last note (octave) to the scale.

get_triad(*index=0*, *degree=None*)
Returns an array with notes from a scale's triad.

Parameters

- **index** (*int*) – triad index (eg.: 0 for 1st degree triad.)
- **degree** (*int*) – Degree of the scale. If provided, overrides the *index* argument. (eg.: *1* for the 1st degree triad.)

Returns An array with three pitches, one for each note of the triad.

class birdears.scale.**ScaleBase**
Bases: object

4.9 birdears.sequence module

class birdears.sequence.**Sequence**(*elements=[]*, *duration=2*, *delay=1.5*, *pos_delay=1*)
Bases: object

Register a Sequence of notes and/or chords.

elements
array_type – List of notes (strings) ou chords (list of strings) in this Sequence.

append(*elements*)
Appends *elements* to Sequence.elements

Parameters **elements** (*array_type*) – Elements to be appended to the class.

async_play(*callback*, *end_callback*)
Plays the Sequence elements of notes and/or chords and wait for *Sequence.pos_delay* seconds.

extend(*elements*)
Extends Sequence.elements with *elements*.

Parameters **elements** (*array_type*) – elements extend the class with.

make_chord_progression(*tonic*, *mode*, *degrees*)
Appends triad chord(s) to the Sequence.

Parameters

- **tonic** (*str*) – Tonic note of the scale.

- **mode** (*str*) – Mode of the scale from which build the triads upon.
- **degrees** (*array_type*) – List with integers represending the degrees of each triad.

play(*callback=None*, *end_callback=None*)

play_element(*index*)
Plays element *sequence.elements[index]*.

CHAPTER

FIVE

FEATURES

- questions
- pretty much configurable
- load from config file
- you can make your own presets
- can be used interactively *(docs needed)*
- can be used as a library *(docs needed)*

CHAPTER

SIX

INSTALLING BIRDEARS

6.1 Installing the dependencies

6.1.1 Arch Linux

```
sudo pacman -Syu sox python python-pip
```

6.2 Installing birdears

To install,simple do this command with pip3

```
pip3 install --user --upgrade --no-cache-dir birdears
```

6.2.1 In-depth installation

You can choose to use a virtualenv to use birdears; this should give you an idea on how to setup one virtualenv.

You should first install virtualenv (for python3) using your distribution's package (supposing you're on linux), then issue on terminal:

```
virtualenv -p python3 ~/.venv # use the directory ~/.venv/ for the virtualenv

source ~/.venv/bin/activate   # activate the virtualenv; this should be done
                              # every time you may want to run the software
                              # installed here.

pip3 install birdears         # this will install the software

birdears --help               # and this will run it
```

CHAPTER

SEVEN

USING BIRDEARS

7.1 What is Functional Ear Training

write me!

7.2 The method

We can use abc language to notate music withing the documentation, ok

```
X: 1
T: Banish Misfortune
R: jig
M: 6/8
L: 1/8
K: Dmix
fed cAG| A2d cAG| F2D DED| FEF GFG|
AGA cAG| AGA cde|fed cAG| Ad^c d3:|
f2d d^cd| f2g agf| e2c cBc|e2f gfe|
f2g agf| e2f gfe|fed cAG|Ad^c d3:|
f2g e2f| d2e c2d|ABA GAG| F2F GED|
c3 cAG| AGA cde| fed cAG| Ad^c d3:|
```

7.3 birdears modes and basic usage

birdears actually has four modes:

- melodic interval question
- harmonic interval question
- melodic dictation question
- instrumental dictation question

To see the commands avaliable just invoke the command without any arguments:

```
birdears
```

```
Usage: birdears  <command> [options]

  birdears — Functional Ear Training for Musicians!
```

```
Options:
  --debug / --no-debug  Turns on debugging; instead you can set DEBUG=1.
  -h, --help            Show this message and exit.

Commands:
  dictation     Melodic dictation
  harmonic      Harmonic interval recognition
  instrumental  Instrumental melodic time-based dictation
  load          Loads exercise from .toml config file...
  melodic       Melodic interval recognition

  You can use 'birdears <command> --help' to show options for a specific
  command.

  More info at https://github.com/iacchus/birdears
```

```
birdears <command> --help
```

7.3.1 melodic

In this exercise birdears will play two notes, the tonic and the interval melodically, ie., one after the other and you should reply which is the correct distance between the two.

```
birdears melodic --help
```

```
Usage: birdears melodic [options]

  Melodic interval recognition

Options:
  -m, --mode <mode>               Mode of the question.
  -t, --tonic <tonic>             Tonic of the question.
  -o, --octave <octave>           Octave of the question.
  -d, --descending                Whether the question interval is descending.
  -c, --chromatic                 If chosen, question has chromatic notes.
  -n, --n_octaves <n max>         Maximum number of octaves.
  -v, --valid_intervals <1,2,..>  A comma-separated list without spaces
                                  of valid scale degrees to be chosen for the
                                  question.
  -q, --user_durations <1,0.5,n..>
                                  A comma-separated list without
                                  spaces with PRECISLY 9 floating values. Or
                                  'n' for default              duration.
  -p, --prequestion_method <prequestion_method>
                                  The name of a pre-question method.
  -r, --resolution_method <resolution_method>
                                  The name of a resolution method.
  -h, --help                      Show this message and exit.

  In this exercise birdears will play two notes, the tonic and the interval
  melodically, ie., one after the other and you should reply which is the
  correct distance between the two.

  Valid values are as follows:
```

```
  -m <mode> is one of: major, dorian, phrygian, lydian, mixolydian, minor,
  locrian

  -t <tonic> is one of: A, A#, Ab, B, Bb, C, C#, D, D#, Db, E, Eb, F, F#, G,
  G#, Gb

  -p <prequestion_method> is one of: none, tonic_only, progression_i_iv_v_i

  -r <resolution_method> is one of: nearest_tonic, repeat_only
```

7.3.2 harmonic

In this exercise birdears will play two notes, the tonic and the interval harmonically, ie., both on the same time and you should reply which is the correct distance between the two.

```
birdears harmonic --help
```

```
Usage: birdears harmonic [options]

  Harmonic interval recognition

Options:
  -m, --mode <mode>               Mode of the question.
  -t, --tonic <note>              Tonic of the question.
  -o, --octave <octave>           Octave of the question.
  -d, --descending                Whether the question interval is descending.
  -c, --chromatic                 If chosen, question has chromatic notes.
  -n, --n_octaves <n max>         Maximum number of octaves.
  -v, --valid_intervals <1,2,..>  A comma-separated list without spaces
                                  of valid scale degrees to be chosen for the
                                  question.
  -q, --user_durations <1,0.5,n..>
                                  A comma-separated list without
                                  spaces with PRECISLY 9 floating values. Or
                                  'n' for default              duration.
  -p, --prequestion_method <prequestion_method>
                                  The name of a pre-question method.
  -r, --resolution_method <resolution_method>
                                  The name of a resolution method.
  -h, --help                      Show this message and exit.

  In this exercise birdears will play two notes, the tonic and the interval
  harmonically, ie., both on the same time and you should reply which is the
  correct distance between the two.

  Valid values are as follows:

  -m <mode> is one of: major, dorian, phrygian, lydian, mixolydian, minor,
  locrian

  -t <tonic> is one of: A, A#, Ab, B, Bb, C, C#, D, D#, Db, E, Eb, F, F#, G,
  G#, Gb

  -p <prequestion_method> is one of: none, tonic_only, progression_i_iv_v_i
```

```
  -r <resolution_method> is one of: nearest_tonic, repeat_only
```

7.3.3 dictation

In this exercise birdears will choose some random intervals and create a melodic dictation with them. You should reply the correct intervals of the melodic dictation.

```
birdears dictation --help
```

```
Usage: birdears dictation [options]

  Melodic dictation

Options:
  -m, --mode <mode>               Mode of the question.
  -i, --max_intervals <n max>     Max random intervals for the dictation.
  -x, --n_notes <n notes>         Number of notes for the dictation.
  -t, --tonic <note>              Tonic of the question.
  -o, --octave <octave>           Octave of the question.
  -d, --descending                Wether the question interval is descending.
  -c, --chromatic                 If chosen, question has chromatic notes.
  -n, --n_octaves <n max>         Maximum number of octaves.
  -v, --valid_intervals <1,2,..>  A comma-separated list without spaces
                                  of valid scale degrees to be chosen for the
                                  question.
  -q, --user_durations <1,0.5,n..>
                                  A comma-separated list without
                                  spaces with PRECISLY 9 floating values. Or
                                  'n' for default              duration.
  -p, --prequestion_method <prequestion_method>
                                  The name of a pre-question method.
  -r, --resolution_method <resolution_method>
                                  The name of a resolution method.
  -h, --help                      Show this message and exit.

  In this exercise birdears will choose some random intervals and create a
  melodic dictation with them. You should reply the correct intervals of the
  melodic dictation.

  Valid values are as follows:

  -m <mode> is one of: major, dorian, phrygian, lydian, mixolydian, minor,
  locrian

  -t <tonic> is one of: A, A#, Ab, B, Bb, C, C#, D, D#, Db, E, Eb, F, F#, G,
  G#, Gb

  -p <prequestion_method> is one of: none, tonic_only, progression_i_iv_v_i

  -r <resolution_method> is one of: nearest_tonic, repeat_only
```

7.3.4 instrumental

In this exercise birdears will choose some random intervals and create a melodic dictation with them. You should play the correct melody in you musical instrument.

```
birdears instrumental --help
```

```
Usage: birdears instrumental [options]

  Instrumental melodic time-based dictation

Options:
  -m, --mode <mode>                Mode of the question.
  -w, --wait_time <seconds>        Time in seconds for next question/repeat.
  -u, --n_repeats <times>          Times to repeat question.
  -i, --max_intervals <n max>      Max random intervals for the dictation.
  -x, --n_notes <n notes>          Number of notes for the dictation.
  -t, --tonic <note>               Tonic of the question.
  -o, --octave <octave>            Octave of the question.
  -d, --descending                 Wether the question interval is descending.
  -c, --chromatic                  If chosen, question has chromatic notes.
  -n, --n_octaves <n max>          Maximum number of octaves.
  -v, --valid_intervals <1,2,..>   A comma-separated list without spaces
                                   of valid scale degrees to be chosen for the
                                   question.
  -q, --user_durations <1,0.5,n..>
                                   A comma-separated list without
                                   spaces with PRECISLY 9 floating values. Or
                                   'n' for default              duration.
  -p, --prequestion_method <prequestion_method>
                                   The name of a pre-question method.
  -r, --resolution_method <resolution_method>
                                   The name of a resolution method.
  -h, --help                       Show this message and exit.

  In this exercise birdears will choose some random intervals and create a
  melodic dictation with them. You should play the correct melody in you
  musical instrument.

  Valid values are as follows:

  -m <mode> is one of: major, dorian, phrygian, lydian, mixolydian, minor,
  locrian

  -t <tonic> is one of: A, A#, Ab, B, Bb, C, C#, D, D#, Db, E, Eb, F, F#, G,
  G#, Gb

  -p <prequestion_method> is one of: none, tonic_only, progression_i_iv_v_i

  -r <resolution_method> is one of: nearest_tonic, repeat_only
```

7.4 Loading from config/preset files

7.4.1 Pre-made presets

`birdears` cointains some pre-made presets in it's `presets/` subdirectory.

The study for beginners is recommended by following the numeric order of those files (000, 001, then 002 etc.)

Pre-made presets description

write me

7.4.2 Creating new preset files

You can open the files cointained in birdears premade `presets/` folder to have an ideia on how config files are made; it is simply the command line options written in a form `toml` standard.

7.5 Keybindings

7.5.1 On the keybindings

The following keyboard diagrams should give you an idea on how the keybindings work. Please note how the keys on the line from `z` (*unison*) to `,` (comma, *octave*) represent the notes that are *natural* to the mode, and the line above represent the chromatics.

Also, for exercises with two octaves, the **uppercased keys represent the second octave**. For example, `z` is *unison*, `,` is the *octave*, `Z` (uppercased) is the *double octave*. The same for all the other intervals.

7.5.2 Major (Ionian)

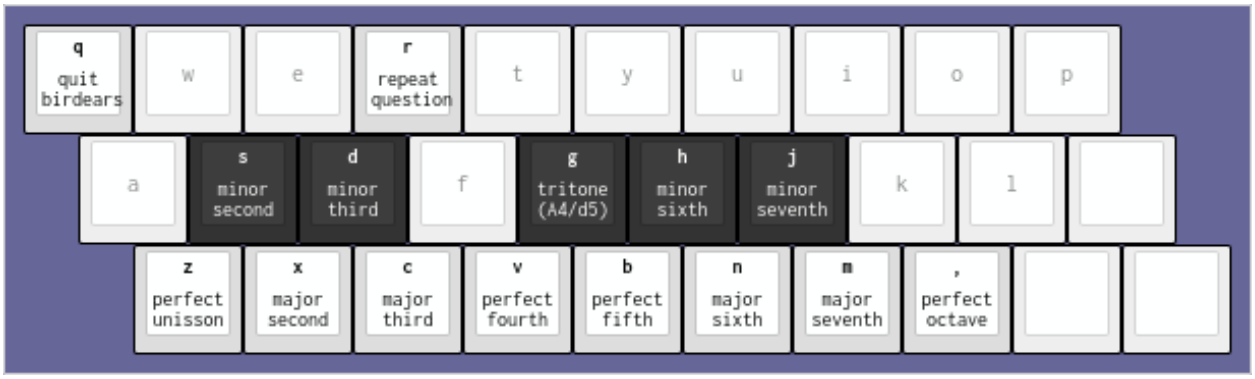

Fig. 7.1: Keyboard diagram for the `--mode major` (default).

Fig. 7.2: Keyboard diagram for the `--mode dorian`.

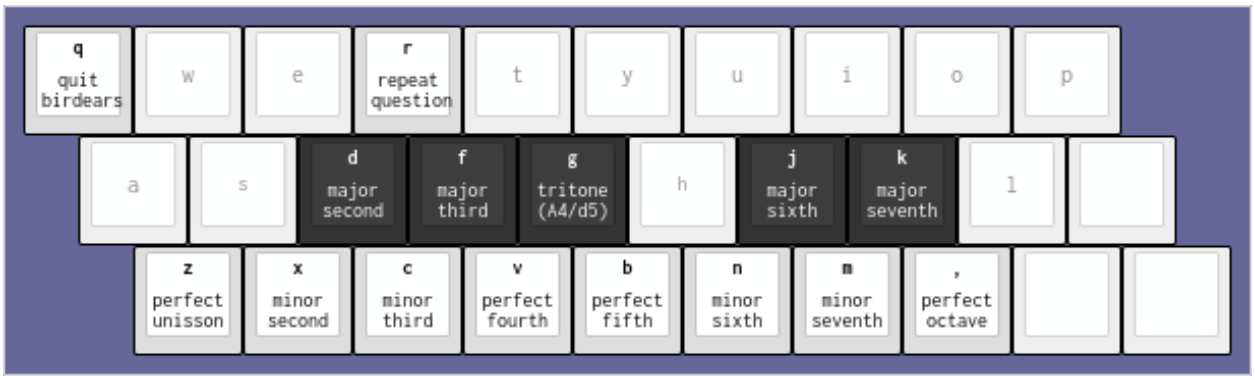

Fig. 7.3: Keyboard diagram for the `--mode phrygian`.

7.5.3 Dorian

7.5.4 Phrygian

7.5.5 Lydian

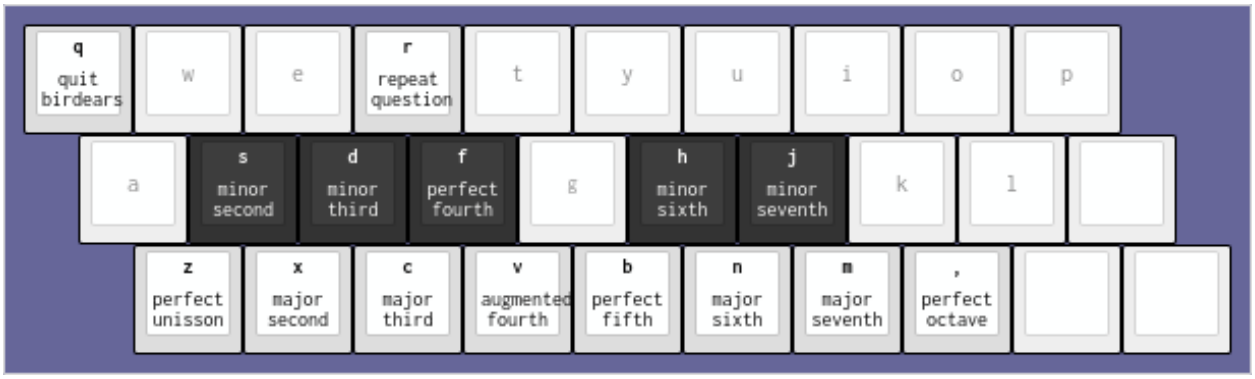

Fig. 7.4: Keyboard diagram for the `--mode lydian`.

7.5.6 Mixolydian

Fig. 7.5: Keyboard diagram for the `--mode mixolydian`.

7.5.7 Minor (Aeolian)

7.5.8 Locrian

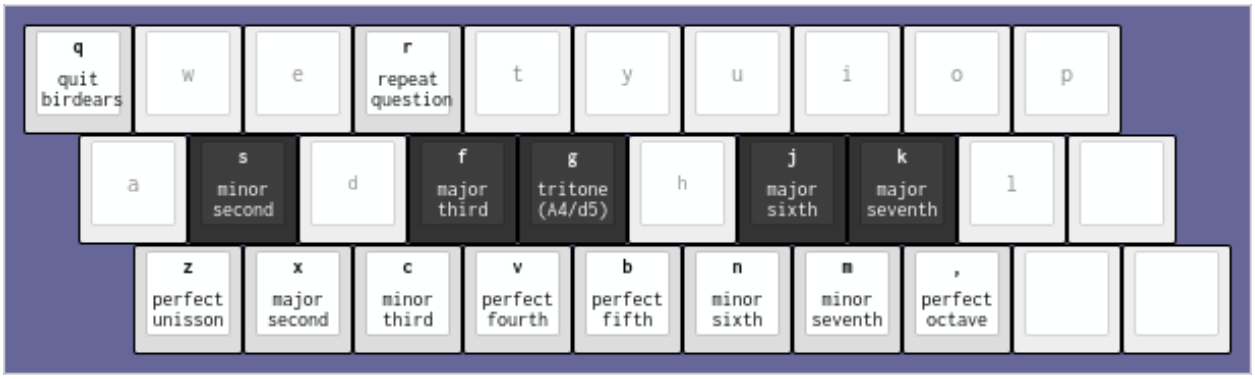

Fig. 7.6: Keyboard diagram for the `--mode minor`.

Fig. 7.7: Keyboard diagram for the `--mode locrian`.

CHAPTER

EIGHT

API

CHAPTER

NINE

BIRDEARS PACKAGE

birdears provides facilities to building musical ear training exercises.

birdears.**CHROMATIC_FLAT = ('C', 'Db', 'D', 'Eb', 'E', 'F', 'Gb', 'G', 'Ab', 'A', 'Bb', 'B')**
tuple – Chromatic notes names using flats.

A mapping of the chromatic note names using flats.

birdears.**CHROMATIC_SHARP = ('C', 'C#', 'D', 'D#', 'E', 'F', 'F#', 'G', 'G#', 'A', 'A#', 'B**
tuple – Chromatic notes names using sharps.

A mapping of the chromatic note namesu sing sharps

birdears.**CHROMATIC_TYPE = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12)**
tuple – A map of the chromatic chromatic scale.

A map of the the semitones which compound the chromatic scale.

birdears.**CIRCLE_OF_FIFTHS = [('C', 'G', 'D', 'A', 'E', 'B', 'Gb', 'Db', 'Ab', 'Eb', 'Bb',**
list of tuples – Circle of fifths.

These are the circle of fifth in both directions.

birdears.**DIATONIC_MODES = {'major': (0, 2, 4, 5, 7, 9, 11, 12), 'dorian': (0, 2, 3, 5, 7,**
dict of tuples – A map of the diatonic scale.

A mapping of the semitones which compound each of the greek modes.

birdears.**INTERVALS = ((0, 'P1', 'Perfect Unison'), (1, 'm2', 'Minor Second'), (2, 'M2', 'Ma**
tuple of tuples – Data representing intervals.

A tuple of tuples representing data for the intervals with format (semitones, short name, full name).

birdears.**INTERVAL_INDEX = {1: [0], 2: [1, 2], 3: [3, 4], 4: [5, 6], 5: [6, 7], 6: [8,**
dict of lists – A mapping of semitones of each interval.

A mapping of semitones which index to each interval name, major/minor, perfect, augmented/diminished

birdears.**KEYS = ('C', 'C#', 'Db', 'D', 'D#', 'Eb', 'E', 'F', 'F#', 'Gb', 'G', 'G#', 'Ab',**
tuple – Allowed keys

These are the allowed keys for exercise as comprehended by birdears.

9.1 Subpackages

9.2 Submodules

9.3 birdears.interval module

class birdears.interval.**ChromaticInterval**(*mode*, *tonic*, *octave*, *n_octaves=None*, *descending=None*, *valid_intervals=None*)

Bases: *birdears.interval.IntervalBase*

Chooses a diatonic interval for the question.

tonic_octave
int – Scientific octave for the tonic. For example, if the tonic is a 'C4' then *tonic_octave* is 4.

interval octave
int – Scientific octave for the interval. For example, if the interval is a 'G5' then *tonic_octave* is 5.

chromatic_offset
int – The offset in semitones inside one octave; maybe it will be deprecated in favour of *distance['semitones']* which is the same.

note_and_octave
str – Note and octave of the interval, for example, if the interval is G5 the note name is 'G5'.

note_name
str – The note name of the interval, for example, if the interval is G5 then the name is 'G'.

semitones
int – Semitones from tonic to octave. If tonic is C4 and interval is G5 the number of semitones is 19.

is_chromatic
bool – If the current interval is chromatic (True) or if it exists in the diatonic scale which key is tonic.

is_descending
bool – If the interval has a descending direction, ie., has a lower pitch than the tonic.

diatonic_index
int – If the interval is chromatic, this will be the nearest diatonic interval in the direction of the resolution (closest tonic.) From II to IV degrees, it is the ditonic interval before; from V to VII it is the diatonic interval after.

distance
dict – A dictionary which the distance from tonic to interval, for example, if tonic is C4 and interval is G5:

```
{
    'octaves': 1,
    'semitones': 7
}
```

data
tuple – A tuple representing the interval data in the form of (semitones, short_name, long_name), for example:

```
(19, 'P12', 'Perfect Twelfth')
```

Todo:

- **Maybe we should refactor some of the attributes with a tuple** (note, octave)
- Maybe remove *chromatic_offset* in favor of *distance['semitones']*

__init__ (*mode*, *tonic*, *octave*, *n_octaves=None*, *descending=None*, *valid_intervals=None*)
Inits the class and choses a random interval with the given args.

Parameters

- **mode** (*str*) – Diatonic mode for the interval. (eg.: 'major' or 'minor')
- **tonic** (*str*) – Tonic of the scale. (eg.: 'Bb')
- **octave** (*str*) – Scientific octave of the scale (eg.: 4)
- **interval** (*str*) – Not implemented. The interval.
- **chromatic** (*bool*) – Can have chromatic notes? (eg.: F# in a key of C; default: false)
- **n_octaves** (*int*) – Maximum number os octaves (eg. 2)
- **descending** (*bool*) – Is the interval descending? (default: false)
- **valid_intervals** (*int*) – A list with inervals valid for random choice, 1 is 1st, 2 is second etc.

class birdears.interval.**DiatonicInterval**(*mode*, *tonic*, *octave*, *n_octaves=None*, *descending=None*, *valid_intervals=None*)

Bases: *birdears.interval.IntervalBase*

Chooses a diatonic interval for the question.

tonic_octave
int – Scientific octave for the tonic. For example, if the tonic is a 'C4' then *tonic_octave* is 4.

interval octave
int – Scientific octave for the interval. For example, if the interval is a 'G5' then *tonic_octave* is 5.

chromatic_offset
int – The offset in semitones inside one octave. Relative semitones to tonic.

note_and_octave
str – Note and octave of the interval, for example, if the interval is G5 the note name is 'G5'.

note_name
str – The note name of the interval, for example, if the interval is G5 then the name is 'G'.

semitones
int – Semitones from tonic to octave. If tonic is C4 and interval is G5 the number of semitones is 19.

is_chromatic
bool – If the current interval is chromatic (True) or if it exists in the diatonic scale which key is tonic.

is_descending
bool – If the interval has a descending direction, ie., has a lower pitch than the tonic.

diatonic_index
int – If the interval is chromatic, this will be the nearest diatonic interval in the direction of the resolution (closest tonic.) From II to IV degrees, it is the ditonic interval before; from V to VII it is the diatonic interval after.

distance
dict – A dictionary which the distance from tonic to interval, for example, if tonic is C4 and interval is G5:

```
{
    'octaves': 1,
    'semitones': 7
}
```

data
tuple – A tuple representing the interval data in the form of (semitones, short_name, long_name), for example:

```
(19, 'P12', 'Perfect Twelfth')
```

__init__(*mode*, *tonic*, *octave*, *n_octaves=None*, *descending=None*, *valid_intervals=None*)
Inits the class and choses a random interval with the given args.

Parameters

- **mode** (*str*) – Diatonic mode for the interval. (eg.: 'major' or 'minor')
- **tonic** (*str*) – Tonic of the scale. (eg.: 'Bb')
- **octave** (*str*) – Scientific octave of the scale (eg.: 4)
- **n_octaves** (*int*) – Maximum number os octaves (eg. 2)
- **descending** (*bool*) – Is the interval descending? (default: false)
- **valid_intervals** (*int*) – A list with intervals (int) valid for random choice, 1 is 1st, 2 is second etc.

class birdears.interval.**IntervalBase**
Bases: object

__init__()
Base class for interval classes.

return_simple(*keys*)
This method returns a dict with only the values passed to *keys*.

9.4 birdears.logger module

This submodule exports *logger* to log events.

Logging messages which are less severe than *lvl* will be ignored:

```
Level       Numeric value
-----       -------------
CRITICAL    50
ERROR       40
WARNING     30
INFO        20
DEBUG       10
NOTSET      0

Level       When it's used
-----       --------------
DEBUG       Detailed information, typically of interest only when
                diagnosing problems.
INFO        Confirmation that things are working as expected.
WARNING     An indication that something unexpected happened, or indicative
```

```
                of some problem in the near future (e.g. 'disk space low').
                The software is still working as expected.
ERROR       Due to a more serious problem, the software has not been able
                to perform some function.
CRITICAL    A serious error, indicating that the program itself may be
                unable to continue running.
```

birdears.logger.**log_event**(*f*, **args*, ***kwargs*)

Decorator. Functions and method decorated with this decorator will have their signature logged when birdears is executed with *–debug* mode. Both function signature with their call values and their return will be logged.

9.5 birdears.prequestion module

This module implements pre-questions' progressions.

Pre questions are chord progressions or notes played before the question is played, so to affirmate the sound of the question's key.

For example a common cadence is chords I-IV-V-I from the diatonic scale, which in a key of *C* is *CM-FM-GM-CM* and in a key of *A* is *AM-DM-EM-AM*.

Pre-question methods should be decorated with *register_prequestion_method* decorator, so that they will be registered as a valid pre-question method.

class birdears.prequestion.**PreQuestion**(*method*, *question*)

Bases: object

__call__(**args*, ***kwargs*)

Calls the resolution method and pass arguments to it.

Returns a *birdears.Sequence* object with the pre-question generated by the method.

__init__(*method*, *question*)

This class implements methods for different types of pre-question progressions.

Parameters

- **method** (*str*) – The method used in the pre question.
- **question** (*obj*) – Question object from which to generate the
- **sequence.** (*pre-question*) –

birdears.prequestion.**none**(*question*, **args*, ***kwargs*)

Pre-question method that return an empty sequence with no delay. :param question: Question object from which to generate the

pre-question sequence. (this is provided by the *Resolution* class when it is '__call__'ed)

birdears.prequestion.**progression_i_iv_v_i**(*question*, **args*, ***kwargs*)

Pre-question method that play's a chord progression with triad chords built on the grades I, IV, V the I of the question key.

Parameters **question** (*obj*) – Question object from which to generate the pre-question sequence. (this is provided by the *Resolution* class when it is '__call__'ed)

birdears.prequestion.**register_prequestion_method**(*f*, **args*, ***kwargs*)
Decorator for prequestion method functions.

Functions decorated with this decorator will be registered in the *PREQUESTION_METHODS* global dict.

birdears.prequestion.**tonic_only**(*question*, **args*, ***kwargs*)
Pre-question method that only play's the question tonic note before the question.

Parameters **question** (*obj*) – Question object from which to generate the pre-question sequence. (this is provided by the *Resolution* class when it is '__call__'ed)

9.6 birdears.questionbase module

class birdears.questionbase.**QuestionBase**(*mode='major'*, *tonic=None*, *octave=None*, *descending=None*, *chromatic=None*, *n_octaves=None*, *valid_intervals=None*, *user_durations=None*, *prequestion_method=None*, *resolution_method=None*, *default_durations=None*, **args*, ***kwargs*)
Bases: object

Base Class to be subclassed for Question classes.

This class implements attributes and routines to be used in Question subclasses.

__init__(*mode='major'*, *tonic=None*, *octave=None*, *descending=None*, *chromatic=None*, *n_octaves=None*, *valid_intervals=None*, *user_durations=None*, *prequestion_method=None*, *resolution_method=None*, *default_durations=None*, **args*, ***kwargs*)
Inits the class.

Parameters

- **mode** (*str*) – A string represnting the mode of the question. Eg., 'major' or 'minor'
- **tonic** (*str*) – A string representing the tonic of the question, eg.: 'C'; if omitted, it will be selected randomly.
- **octave** (*int*) – A scienfic octave notation, for example, 4 for 'C4'; if not present, it will be randomly chosen.
- **descending** (*bool*) – Is the question direction in descending, ie., intervals have lower pitch than the tonic.
- **chromatic** (*bool*) – If the question can have (True) or not (False) chromatic intervals, ie., intervals not in the diatonic scale of tonic/mode.
- **n_octaves** (*int*) – Maximum numbr of octaves of the question.
- **valid_intervals** (*list*) – A list with intervals (int) valid for random choice, 1 is 1st, 2 is second etc. Eg. [1, 4, 5] to allow only tonics, fourths and fifths.
- **user_durations** (*dict*) – A string with 9 comma-separated *int* or *float's to set the default duration for the notes played. The values are respectively for: pre-question duration (1st), pre-question delay (2nd), and pre-question pos-delay (3rd); question duration (4th), question delay (5th), and question pos-delay (6th); resolution duration (7th), resolution delay (8th), and resolution pos-delay (9th). duration is the duration in of the note in seconds; delay is the time to wait before playing the next note, and pos_delay is the time to wait after all the notes of the respective sequence have been played. If any of the user durations is 'n*, the default duration for the type of question will be used instead. Example:

```
"2,0.5,1,2,n,0,2.5,n,1"
```

- **prequestion_method** (*str*) – Method of playing a cadence or the exercise tonic before the question so to affirm the question musical tonic key to the ear. Valid ones are registered in the *birdears.prequestion.PREQUESION_METHODS* global dict.
- **resolution_method** (*str*) – Method of playing the resolution of an exercise Valid ones are registered in the *birdears.resolution.RESOLUTION_METHODS* global dict.
- **user_durations** – Dictionary with the default durations for each type of sequence. This is provided by the subclasses.

check_question()
This method should be overwritten by the question subclasses.

get_valid_semitones()
Returns a list with valid semitones for question.

make_question()
This method should be overwritten by the question subclasses.

make_resolution()
This method should be overwritten by the question subclasses.

play_question()
This method should be overwritten by the question subclasses.

birdears.questionbase.**register_question_class**(*f*, **args*, ***kwargs*)
Decorator for question classes.

Classes decorated with this decorator will be registered in the *QUESTION_CLASSES* global.

9.7 birdears.resolution module

class birdears.resolution.**Resolution**(*method*, *question*)
Bases: object

This class implements methods for different types of question resolutions.

A resolution is an answer to a question. It aims to create a mnemonic on how the inverval resvolver to the tonic.

__call__(**args*, ***kwargs*)
Calls the resolution method and pass arguments to it.

Returns a *birdears.Sequence* object with the resolution generated by the.method.

__init__(*method*, *question*)
Inits the resolution class.

Parameters

- **method** (*str*) – The method used in the resolution.
- **question** (*obj*) – Question object from which to generate the
- **sequence.** (resolution) –

birdears.resolution.**nearest_tonic**(*question*)
Resolution method that resolve the intervals to their nearest tonics.

Parameters **question** (*obj*) – Question object from which to generate the resolution sequence. (this is provided by the *Prequestion* class when it is '__call__'ed)

birdears.resolution.**register_resolution_method**(*f*, **args*, ***kwargs*)
Decorator for resolution method functions.

Functions decorated with this decorator will be registered in the *RESOLUTION_METHODS* global dict.

birdears.resolution.**repeat_only**(*question*)
Resolution method that only repeats the sequence elements with given durations.

Parameters **question** (*obj*) – Question object from which to generate the resolution sequence. (this is provided by the *Prequestion* class when it is '__call__'ed)

9.8 birdears.scale module

class birdears.scale.**ChromaticScale**(*tonic*, *octave=None*, *n_octaves=None*, *descending=None*, *dont_repeat_tonic=None*)
Bases: *birdears.scale.ScaleBase*

Builds a musical chromatic scale.

scale
array_type – The array of notes representing the scale.

__init__(*tonic*, *octave=None*, *n_octaves=None*, *descending=None*, *dont_repeat_tonic=None*)
Returns a chromatic scale from tonic.

Parameters

- **tonic** (*str*) – The note which the scale will be built upon.
- **octave** (*int*) – The scientific octave the scale will be built upon.
- **n_octaves** (*int*) – The number of octaves the scale will contain.
- **descending** (*bool*) – Whether the scale is descending.
- **dont_repeat_tonic** (*bool*) – Whether to skip appending the last note (octave) to the scale.

get_triad(*mode*, *index=0*, *degree=None*)
Returns an array with notes from a scale's triad.

Parameters

- **mode** (*str*) – Mode of the scale (eg. 'major' or 'minor')
- **index** (*int*) – Triad index (eg.: 0 for 1st degree triad.)
- **degree** (*int*) – Degree of the scale. If provided, overrides the *index* argument. (eg.: *1* for the 1st degree triad.)

Returns A list with three pitches (str), one for each note of the triad.

class birdears.scale.**DiatonicScale**(*tonic*, *mode=None*, *octave=None*, *n_octaves=None*, *descending=None*, *dont_repeat_tonic=None*)
Bases: *birdears.scale.ScaleBase*

Builds a musical diatonic scale.

scale
array_type – The array of notes representing the scale.

__init__(*tonic*, *mode=None*, *octave=None*, *n_octaves=None*, *descending=None*, *dont_repeat_tonic=None*)
Returns a diatonic scale from tonic and mode.

Parameters

- **tonic** (*str*) – The note which the scale will be built upon.
- **mode** (*str*) – The mode the scale will be built upon. ('major' or 'minor')
- **octave** (*int*) – The scientific octave the scale will be built upon.
- **n_octaves** (*int*) – The number of octaves the scale will contain.
- **descending** (*bool*) – Whether the scale is descending.
- **dont_repeat_tonic** (*bool*) – Whether to skip appending the last note (octave) to the scale.

get_triad(*index=0*, *degree=None*)
Returns an array with notes from a scale's triad.

Parameters

- **index** (*int*) – triad index (eg.: 0 for 1st degree triad.)
- **degree** (*int*) – Degree of the scale. If provided, overrides the *index* argument. (eg.: *1* for the 1st degree triad.)

Returns An array with three pitches, one for each note of the triad.

class birdears.scale.**ScaleBase**
Bases: object

9.9 birdears.sequence module

class birdears.sequence.**Sequence**(*elements=[]*, *duration=2*, *delay=1.5*, *pos_delay=1*)
Bases: object

Register a Sequence of notes and/or chords.

elements
array_type – List of notes (strings) ou chords (list of strings) in this Sequence.

append(*elements*)
Appends *elements* to Sequence.elements

Parameters **elements** (*array_type*) – Elements to be appended to the class.

async_play(*callback*, *end_callback*)
Plays the Sequence elements of notes and/or chords and wait for *Sequence.pos_delay* seconds.

extend(*elements*)
Extends Sequence.elements with *elements*.

Parameters **elements** (*array_type*) – elements extend the class with.

make_chord_progression(*tonic*, *mode*, *degrees*)
Appends triad chord(s) to the Sequence.

Parameters

- **tonic** (*str*) – Tonic note of the scale.

- **mode** (*str*) – Mode of the scale from which build the triads upon.
- **degrees** (*array_type*) – List with integers represending the degrees of each triad.

play(*callback=None*, *end_callback=None*)

play_element(*index*)
Plays element *sequence.elements[index]*.

CHAPTER

TEN

BIRDEARS.QUESTIONS PACKAGE

10.1 Submodules

10.2 birdears.questions.harmonicinterval module

class birdears.questions.harmonicinterval.**HarmonicIntervalQuestion**(*mode='major'*, *tonic=None*, *octave=None*, *descending=None*, *chromatic=None*, *n_octaves=None*, *valid_intervals=None*, *user_durations=None*, *prequestion_method='none'*, *resolution_method='nearest_tonic'*, **args*, ***kwargs*)

Bases: *birdears.questionbase.QuestionBase*

Implements a Harmonic Interval test.

__init__(*mode='major'*, *tonic=None*, *octave=None*, *descending=None*, *chromatic=None*, *n_octaves=None*, *valid_intervals=None*, *user_durations=None*, *prequestion_method='none'*, *resolution_method='nearest_tonic'*, **args*, ***kwargs*)

Inits the class.

Parameters

- **mode** (*str*) – A string representing the mode of the question. Eg., 'major' or 'minor'
- **tonic** (*str*) – A string representing the tonic of the question, eg.: 'C'; if omitted, it will be selected randomly.
- **octave** (*int*) – A scienfic octave notation, for example, 4 for 'C4'; if not present, it will be randomly chosen.
- **descending** (*bool*) – Is the question direction in descending, ie., intervals have lower pitch than the tonic.

- **chromatic** (*bool*) – If the question can have (True) or not (False) chromatic intervals, ie., intervals not in the diatonic scale of tonic/mode.
- **n_octaves** (*int*) – Maximum number of octaves of the question.
- **valid_intervals** (*list*) – A list with intervals (int) valid for random choice, 1 is 1st, 2 is second etc. Eg. [1, 4, 5] to allow only tonics, fourths and fifths.
- **user_durations** (*str*) – A string with 9 comma-separated *int* or *float's to set the default duration for the notes played. The values are respectively for: pre-question duration (1st), pre-question delay (2nd), and pre-question pos-delay (3rd); question duration (4th), question delay (5th), and question pos-delay (6th); resolution duration (7th), resolution delay (8th), and resolution pos-delay (9th). duration is the duration in of the note in seconds; delay is the time to wait before playing the next note, and pos_delay is the time to wait after all the notes of the respective sequence have been played. If any of the user durations is 'n*, the default duration for the type of question will be used instead. Example:

```
"2,0.5,1,2,n,0,2.5,n,1"
```

- **prequestion_method** (*str*) – Method of playing a cadence or the exercise tonic before the question so to affirm the question musical tonic key to the ear. Valid ones are registered in the *birdears.prequestion.PREQUESION_METHODS* global dict.
- **resolution_method** (*str*) – Method of playing the resolution of an exercise. Valid ones are registered in the *birdears.resolution.RESOLUTION_METHODS* global dict.

check_question(*user_input_char*)

Checks whether the given answer is correct.

make_pre_question(*method*)

make_question()

make_resolution(*method*)

play_question()

play_resolution()

10.3 birdears.questions.instrumentaldictation module

class birdears.questions.instrumentaldictation.**InstrumentalDictationQuestion**(*mode='major'*, *wait_time=11*, *n_repeats=1*, *max_intervals=3*, *n_notes=4*, *tonic=None*, *octave=None*, *descending=None*, *chromatic=None*, *n_octaves=None*, *valid_intervals=No*
user_durations=No
prequestion_method='progr
resolution_method='repea
**args*, ***kwargs*)

Bases: *birdears.questionbase.QuestionBase*

Implements an instrumental dictation test.

__init__(*mode='major'*, *wait_time=11*, *n_repeats=1*, *max_intervals=3*, *n_notes=4*, *tonic=None*, *octave=None*, *descending=None*, *chromatic=None*, *n_octaves=None*, *valid_intervals=None*, *user_durations=None*, *prequestion_method='progression_i_iv_v_i'*, *resolution_method='repeat_only'*, **args*, ***kwargs*)

Inits the class.

Parameters

- **mode** (*str*) – A string representing the mode of the question. Eg., 'major' or 'minor'.
- **wait_time** (*float*) – Wait time in seconds for the next question or repeat.
- **n_repeats** (*int*) – Number of times the same dictation will be repeated before the end of the exercise.
- **max_intervals** (*int*) – The maximum number of random intervals the question will have.
- **n_notes** (*int*) – The number of notes the melodic dictation will have.
- **tonic** (*str*) – A string representing the tonic of the question, eg.: 'C'; if omitted, it will be selected randomly.
- **octave** (*int*) – A scienfic octave notation, for example, 4 for 'C4'; if not present, it will be randomly chosen.
- **descending** (*bool*) – Is the question direction in descending, ie., intervals have lower pitch than the tonic.

- **chromatic** (*bool*) – If the question can have (True) or not (False) chromatic intervals, ie., intervals not in the diatonic scale of tonic/mode.
- **n_octaves** (*int*) – Maximum number of octaves of the question.
- **valid_intervals** (*list*) – A list with intervals (int) valid for random choice, 1 is 1st, 2 is second etc. Eg. [1, 4, 5] to allow only tonics, fourths and fifths.
- **user_durations** (*str*) – A string with 9 comma-separated *int* or *float's to set the default duration for the notes played. The values are respectively for: pre-question duration (1st), pre-question delay (2nd), and pre-question pos-delay (3rd); question duration (4th), question delay (5th), and question pos-delay (6th); resolution duration (7th), resolution delay (8th), and resolution pos-delay (9th). duration is the duration in of the note in seconds; delay is the time to wait before playing the next note, and pos_delay is the time to wait after all the notes of the respective sequence have been played. If any of the user durations is 'n*, the default duration for the type of question will be used instead. Example:

```
"2,0.5,1,2,n,0,2.5,n,1"
```

- **prequestion_method** (*str*) – Method of playing a cadence or the exercise tonic before the question so to affirm the question musical tonic key to the ear. Valid ones are registered in the *birdears.prequestion.PREQUESION_METHODS* global dict.
- **resolution_method** (*str*) – Method of playing the resolution of an exercise. Valid ones are registered in the *birdears.resolution.RESOLUTION_METHODS* global dict.

check_question()

Checks whether the given answer is correct.

This currently doesn't applies to instrumental dictation questions.

make_pre_question(*method*)

make_question(*phrase_semitones*)

make_resolution(*method*)

play_question()

10.4 birdears.questions.melodicdictation module

class birdears.questions.melodicdictation.**MelodicDictationQuestion**(*mode='major'*, *max_intervals=3*, *n_notes=4*, *tonic=None*, *octave=None*, *descending=None*, *chromatic=None*, *n_octaves=None*, *valid_intervals=None*, *user_durations=None*, *prequestion_method='progression_i_iv_v_*, *resolution_method='repeat_only'*, **args*, ***kwargs*)

Bases: *birdears.questionbase.QuestionBase*

Implements a melodic dictation test.

__init__(*mode='major'*, *max_intervals=3*, *n_notes=4*, *tonic=None*, *octave=None*, *descending=None*, *chromatic=None*, *n_octaves=None*, *valid_intervals=None*, *user_durations=None*, *prequestion_method='progression_i_iv_v_i'*, *resolution_method='repeat_only'*, **args*, ***kwargs*)

Inits the class.

Parameters

- **mode** (*str*) – A string representing the mode of the question. Eg., 'major' or 'minor'.
- **max_intervals** (*int*) – The maximum number of random intervals the question will have.
- **n_notes** (*int*) – The number of notes the melodic dictation will have.
- **tonic** (*str*) – A string representing the tonic of the question, eg.: 'C'; if omitted, it will be selected randomly.
- **octave** (*int*) – A scienfic octave notation, for example, 4 for 'C4'; if not present, it will be randomly chosen.
- **descending** (*bool*) – Is the question direction in descending, ie., intervals have lower pitch than the tonic.
- **chromatic** (*bool*) – If the question can have (True) or not (False) chromatic intervals, ie., intervals not in the diatonic scale of tonic/mode.
- **n_octaves** (*int*) – Maximum number of octaves of the question.
- **valid_intervals** (*list*) – A list with intervals (int) valid for random choice, 1 is 1st, 2 is second etc. Eg. [1, 4, 5] to allow only tonics, fourths and fifths.
- **user_durations** (*str*) – A string with 9 comma-separated *int* or *float's to set the default duration for the notes played. The values are respectively for: pre-question duration (1st), pre-question delay (2nd), and pre-question pos-delay (3rd); question duration*

(4th), question delay (5th), and question pos-delay (6th); resolution duration (7th), resolution delay (8th), and resolution pos-delay (9th). duration is the duration in of the note in seconds; delay is the time to wait before playing the next note, and pos_delay is the time to wait after all the notes of the respective sequence have been played. If any of the user durations is 'n, the default duration for the type of question will be used instead. Example:

```
"2,0.5,1,2,n,0,2.5,n,1"
```

- **prequestion_method** (*str*) – Method of playing a cadence or the exercise tonic before the question so to affirm the question musical tonic key to the ear. Valid ones are registered in the *birdears.prequestion.PREQUESION_METHODS* global dict.
- **resolution_method** (*str*) – Method of playing the resolution of an exercise. Valid ones are registered in the *birdears.resolution.RESOLUTION_METHODS* global dict.

check_question(*user_input_keys*)
Checks whether the given answer is correct.

make_pre_question(*method*)

make_question(*phrase_semitones*)

make_resolution(*method*)

play_question()

play_resolution()

10.5 birdears.questions.melodicinterval module

class birdears.questions.melodicinterval.**MelodicIntervalQuestion**(*mode='major'*, *tonic=None*, *octave=None*, *descending=None*, *chromatic=None*, *n_octaves=None*, *valid_intervals=None*, *user_durations=None*, *prequestion_method='tonic_only'*, *resolution_method='nearest_tonic'*, **args*, ***kwargs*)

Bases: *birdears.questionbase.QuestionBase*

Implements a Melodic Interval test.

check_question(*user_input_char*)
Checks whether the given answer is correct.

make_pre_question(*method*)

make_question()

make_resolution(*method*)

play_question()

play_resolution()

CHAPTER

ELEVEN

BIRDEARS.INTERFACES PACKAGE

11.1 Submodules

11.2 birdears.interfaces.commandline module

birdears.interfaces.commandline.**CommandLine**(*exercise*, ***kwargs*)
This function implements the birdears loop for command line.

Parameters

- **exercise** (*str*) – The question name.
- ****kwargs** (*kwargs*) – FIXME: The kwargs can contain options for specific questions.

birdears.interfaces.commandline.**center_text**(*text*, *sep=True*, *nl=0*)
This function returns input text centered according to terminal columns.

Parameters

- **text** (*str*) – The string to be centered, it can have multiple lines.
- **sep** (*bool*) – Add line separator after centered text (True) or not (False).
- **nl** (*int*) – How many new lines to add after text.

birdears.interfaces.commandline.**make_input_str**(*user_input*, *keyboard_index*)
Makes a string representing intervals entered by the user.

This function is to be used by questions which takes more than one interval input as MelodicDictation, and formats the intervals already entered.

Parameters

- **user_input** (*array_type*) – The list of keyboard keys entered by user.
- **keyboard_index** (*array_type*) – The keyboard mapping used by question.

birdears.interfaces.commandline.**print_instrumental**(*response*)
Prints the formatted response for 'instrumental' exercise.

Parameters **response** (*dict*) – A response returned by question's check_question()

birdears.interfaces.commandline.**print_question**(*question*)
Prints the question to the user.

Parameters **question** (*obj*) – A Question class with the question to be printed.

birdears.interfaces.commandline.**print_response**(*response*)
Prints the formatted response.

Parameters **response** (*dict*) – A response returned by question's check_question()

PYTHON MODULE INDEX

b

birdears, 49
birdears.interfaces, 67
birdears.interfaces.commandline, 67
birdears.interval, 50
birdears.logger, 52
birdears.prequestion, 53
birdears.questionbase, 54
birdears.questions, 59
birdears.questions.harmonicinterval, 59
birdears.questions.instrumentaldictation, 61
birdears.questions.melodicdictation, 63
birdears.questions.melodicinterval, 64
birdears.resolution, 55
birdears.scale, 56
birdears.sequence, 57

INDEX

Symbols

__call__() (birdears.prequestion.PreQuestion method), 27, 53
__call__() (birdears.resolution.Resolution method), 29, 55
__init__() (birdears.interval.ChromaticInterval method), 25, 51
__init__() (birdears.interval.DiatonicInterval method), 26, 52
__init__() (birdears.interval.IntervalBase method), 26, 52
__init__() (birdears.prequestion.PreQuestion method), 27, 53
__init__() (birdears.questionbase.QuestionBase method), 28, 54
__init__() (birdears.questions.harmonicinterval.HarmonicIntervalQuestion method), 19, 59
__init__() (birdears.questions.instrumentaldictation.InstrumentalDictationQuestion method), 20, 61
__init__() (birdears.questions.melodicdictation.MelodicDictationQuestion method), 22, 63
__init__() (birdears.resolution.Resolution method), 29, 55
__init__() (birdears.scale.ChromaticScale method), 30, 56
__init__() (birdears.scale.DiatonicScale method), 30, 56

A

append() (birdears.sequence.Sequence method), 31, 57
async_play() (birdears.sequence.Sequence method), 31, 57

B

birdears (module), 17, 49
birdears.interfaces (module), 17, 67
birdears.interfaces.commandline (module), 18, 67
birdears.interval (module), 24, 50
birdears.logger (module), 26, 52
birdears.prequestion (module), 27, 53
birdears.questionbase (module), 28, 54
birdears.questions (module), 18, 59
birdears.questions.harmonicinterval (module), 19, 59
birdears.questions.instrumentaldictation (module), 20, 61
birdears.questions.melodicdictation (module), 22, 63
birdears.questions.melodicinterval (module), 23, 64
birdears.resolution (module), 29, 55
birdears.scale (module), 30, 56
birdears.sequence (module), 31, 57

C

center_text() (in module birdears.interfaces.commandline), 18, 67
check_question() (birdears.questionbase.QuestionBase method), 29, 55
check_question() (birdears.questions.harmonicinterval.HarmonicIntervalQuestion method), 20, 60
check_question() (birdears.questions.instrumentaldictation.InstrumentalDictationQuestion method), 21, 62
check_question() (birdears.questions.melodicdictation.MelodicDictationQuestion method), 23, 64
check_question() (birdears.questions.melodicinterval.MelodicIntervalQuestion method), 23, 64
CHROMATIC_FLAT (in module birdears), 17, 49
chromatic_offset (birdears.interval.ChromaticInterval attribute), 24, 50
chromatic_offset (birdears.interval.DiatonicInterval attribute), 25, 51
CHROMATIC_SHARP (in module birdears), 17, 49
CHROMATIC_TYPE (in module birdears), 17, 49
ChromaticInterval (class in birdears.interval), 24, 50
ChromaticScale (class in birdears.scale), 30, 56
CIRCLE_OF_FIFTHS (in module birdears), 17, 49
CommandLine() (in module birdears.interfaces.commandline), 18, 67

D

data (birdears.interval.ChromaticInterval attribute), 24, 50
data (birdears.interval.DiatonicInterval attribute), 26, 52
diatonic_index (birdears.interval.ChromaticInterval attribute), 24, 50

diatonic_index (birdears.interval.DiatonicInterval attribute), 25, 51
DIATONIC_MODES (in module birdears), 17, 49
DiatonicInterval (class in birdears.interval), 25, 51
DiatonicScale (class in birdears.scale), 30, 56
distance (birdears.interval.ChromaticInterval attribute), 24, 50
distance (birdears.interval.DiatonicInterval attribute), 25, 51

E

elements (birdears.sequence.Sequence attribute), 31, 57
extend() (birdears.sequence.Sequence method), 31, 57

G

get_triad() (birdears.scale.ChromaticScale method), 30, 56
get_triad() (birdears.scale.DiatonicScale method), 31, 57
get_valid_semitones() (birdears.questionbase.QuestionBase method), 29, 55

H

HarmonicIntervalQuestion (class in birdears.questions.harmonicinterval), 19, 59

I

InstrumentalDictationQuestion (class in birdears.questions.instrumentaldictation), 20, 61
INTERVAL_INDEX (in module birdears), 17, 49
IntervalBase (class in birdears.interval), 26, 52
INTERVALS (in module birdears), 17, 49
is_chromatic (birdears.interval.ChromaticInterval attribute), 24, 50
is_chromatic (birdears.interval.DiatonicInterval attribute), 25, 51
is_descending (birdears.interval.ChromaticInterval attribute), 24, 50
is_descending (birdears.interval.DiatonicInterval attribute), 25, 51

K

KEYS (in module birdears), 17, 49

L

log_event() (in module birdears.logger), 27, 53

M

make_chord_progression() (birdears.sequence.Sequence method), 31, 57
make_input_str() (in module birdears.interfaces.commandline), 18, 67
make_pre_question() (birdears.questions.harmonicinterval.HarmonicIntervalQuestion method), 20, 60
make_pre_question() (birdears.questions.instrumentaldictation.InstrumentalDictationQuestio method), 21, 62
make_pre_question() (birdears.questions.melodicdictation.MelodicDictationQuestion method), 23, 64
make_pre_question() (birdears.questions.melodicinterval.MelodicIntervalQuestion method), 23, 64
make_question() (birdears.questionbase.QuestionBase method), 29, 55
make_question() (birdears.questions.harmonicinterval.HarmonicIntervalQue method), 20, 60
make_question() (birdears.questions.instrumentaldictation.InstrumentalDict method), 21, 62
make_question() (birdears.questions.melodicdictation.MelodicDictationQue method), 23, 64
make_question() (birdears.questions.melodicinterval.MelodicIntervalQuesti method), 23, 64
make_resolution() (birdears.questionbase.QuestionBase method), 29, 55
make_resolution() (birdears.questions.harmonicinterval.HarmonicIntervalQuestion method), 20, 60
make_resolution() (birdears.questions.instrumentaldictation.InstrumentalDictationQuestio method), 21, 62
make_resolution() (birdears.questions.melodicdictation.MelodicDictationQuestion method), 23, 64
make_resolution() (birdears.questions.melodicinterval.MelodicIntervalQuestion method), 23, 64
MelodicDictationQuestion (class in birdears.questions.melodicdictation), 22, 63
MelodicIntervalQuestion (class in birdears.questions.melodicinterval), 23, 64

N

nearest_tonic() (in module birdears.resolution), 29, 55
none() (in module birdears.prequestion), 27, 53
note_and_octave (birdears.interval.ChromaticInterval attribute), 24, 50
note_and_octave (birdears.interval.DiatonicInterval attribute), 25, 51
note_name (birdears.interval.ChromaticInterval attribute), 24, 50
note_name (birdears.interval.DiatonicInterval attribute), 25, 51

P

play() (birdears.sequence.Sequence method), 32, 58
play_element() (birdears.sequence.Sequence method), 32, 58
play_question() (birdears.questionbase.QuestionBase method), 29, 55
play_question() (birdears.questions.harmonicinterval.HarmonicIntervalQuestion method), 20, 60
play_question() (birdears.questions.instrumentaldictation.InstrumentalDictationQuestion method), 21, 62
play_question() (birdears.questions.melodicdictation.MelodicDictationQuestion method), 23, 64
play_question() (birdears.questions.melodicinterval.MelodicIntervalQuestion method), 23, 64
play_resolution() (birdears.questions.harmonicinterval.HarmonicIntervalQuestion method), 20, 60
play_resolution() (birdears.questions.melodicdictation.MelodicDictationQuestion method), 23, 64
play_resolution() (birdears.questions.melodicinterval.MelodicIntervalQuestion method), 24, 65
PreQuestion (class in birdears.prequestion), 27, 53
print_instrumental() (in module birdears.interfaces.commandline), 18, 67
print_question() (in module birdears.interfaces.commandline), 18, 67
print_response() (in module birdears.interfaces.commandline), 18, 67
progression_i_iv_v_i() (in module birdears.prequestion), 27, 53

Q

QuestionBase (class in birdears.questionbase), 28, 54

R

register_prequestion_method() (in module birdears.prequestion), 27, 53
register_question_class() (in module birdears.questionbase), 29, 55
register_resolution_method() (in module birdears.resolution), 30, 56
repeat_only() (in module birdears.resolution), 30, 56
Resolution (class in birdears.resolution), 29, 55
return_simple() (birdears.interval.IntervalBase method), 26, 52

S

scale (birdears.scale.ChromaticScale attribute), 30, 56
scale (birdears.scale.DiatonicScale attribute), 30, 56
ScaleBase (class in birdears.scale), 31, 57
semitones (birdears.interval.ChromaticInterval attribute), 24, 50
semitones (birdears.interval.DiatonicInterval attribute), 25, 51
Sequence (class in birdears.sequence), 31, 57

T

tonic_octave (birdears.interval.ChromaticInterval attribute), 24, 50
tonic_octave (birdears.interval.DiatonicInterval attribute), 25, 51
tonic_only() (in module birdears.prequestion), 28, 54