birdears Documentation

Release latest

Iacchus Mmercurius

Jan 21, 2023

CONTENTS

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

SUPPORT

If you need help you can get in touch via IRC or file an issue on any matter regarding birdears at Github.

Media	Channel
IRC	#birdears
GitHub	https://github.com/iacchus/birdears
GH issues	https://github.com/iacchus/birdears/issues
ReadTheDocs	https://birdears.readthedocs.io
PyPI	https://pypi.python.org/pypi/birdears
TravisCI	https://travis-ci.org/iacchus/birdears
Coveralls	https://coveralls.io/github/iacchus/birdears

CHAPTER

TWO

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

THREE

INSTALLING BIRDEARS

3.1 Installing the dependencies

3.1.1 Arch Linux

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

3.2 Installing birdears

To install,simple do this command with pip3

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

3.2.1 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

FOUR

USING BIRDEARS

4.1 What is Functional Ear Training

write me!

4.2 The method

We can use abc language to notate music within 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:|
```

4.3 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
```

4.3.1 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.
```

(continues on next page)

(continued from previous page)

```
  -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
```

4.3.2 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
```

(continues on next page)

(continued from previous page)

```
  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
```

4.3.3 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
```

(continues on next page)

(continued from previous page)

```
  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
```

4.3.4 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
```

(continues on next page)

(continued from previous page)

```
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
```

4.4 Loading from config/preset files

4.4.1 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

4.4.2 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.

4.5 Keybindings

4.5.1 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.

4.5.2 Major (Ionian)

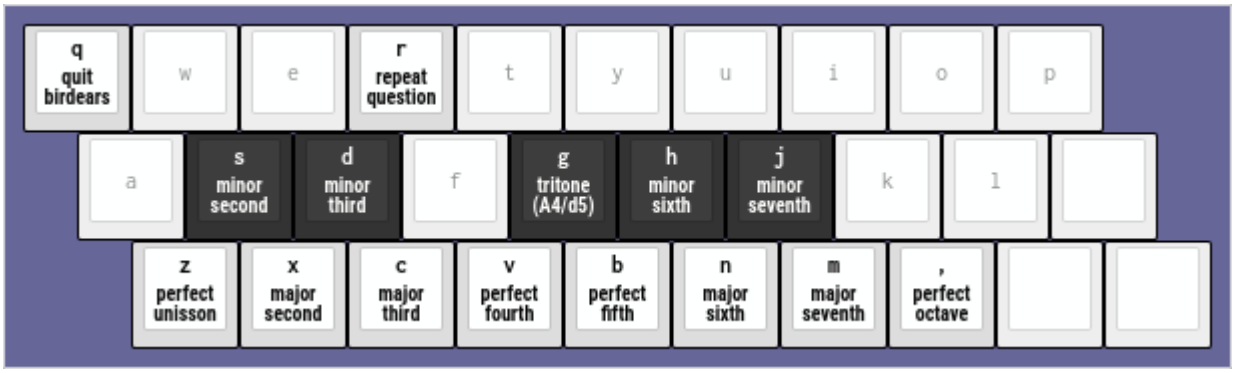

Fig. 1: Keyboard diagram for the `--mode major` (default).

4.5.3 Dorian

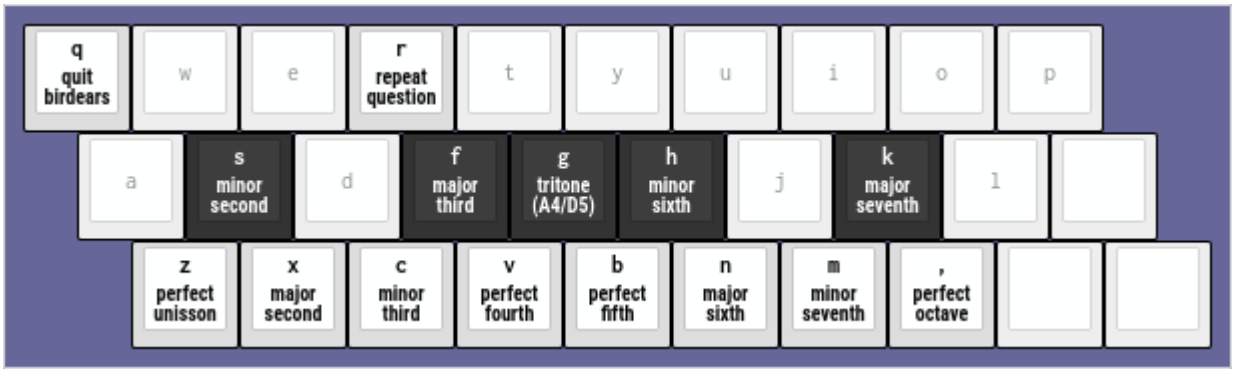

Fig. 2: Keyboard diagram for the `--mode dorian`.

4.5.4 Phrygian

4.5.5 Lydian

4.5.6 Mixolydian

4.5.7 Minor (Aeolian)

4.5.8 Locrian

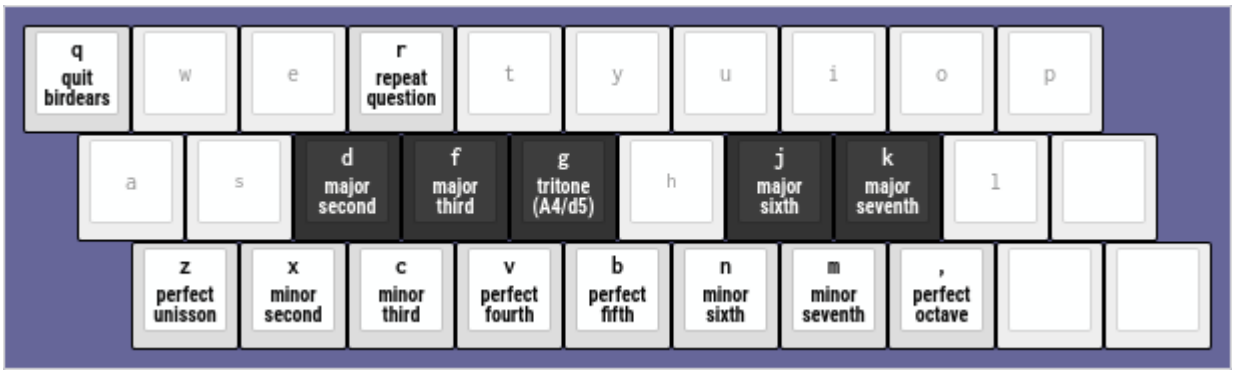

Fig. 3: Keyboard diagram for the `--mode phrygian`.

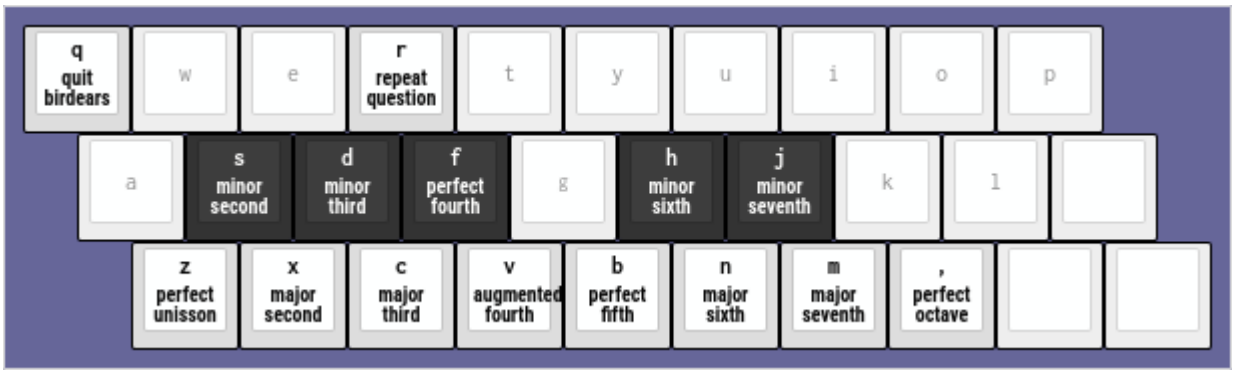

Fig. 4: Keyboard diagram for the `--mode lydian`.

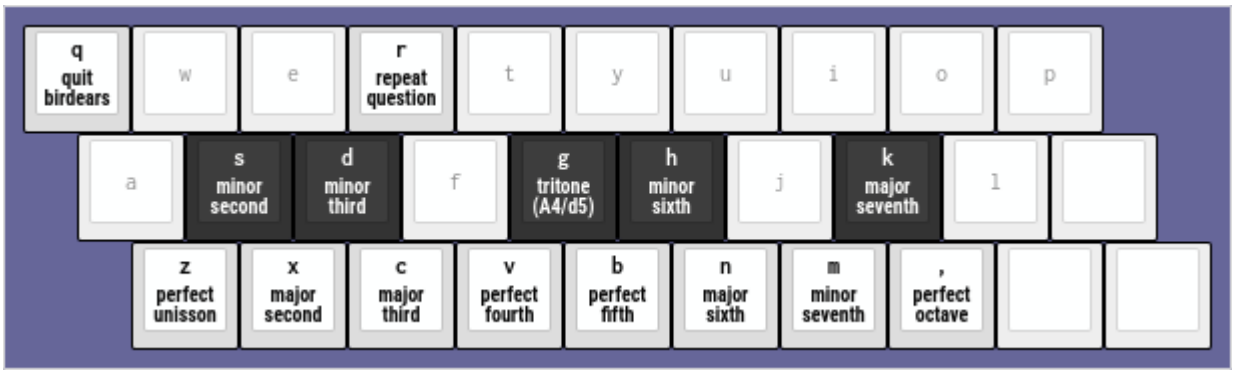

Fig. 5: Keyboard diagram for the `--mode mixolydian`.

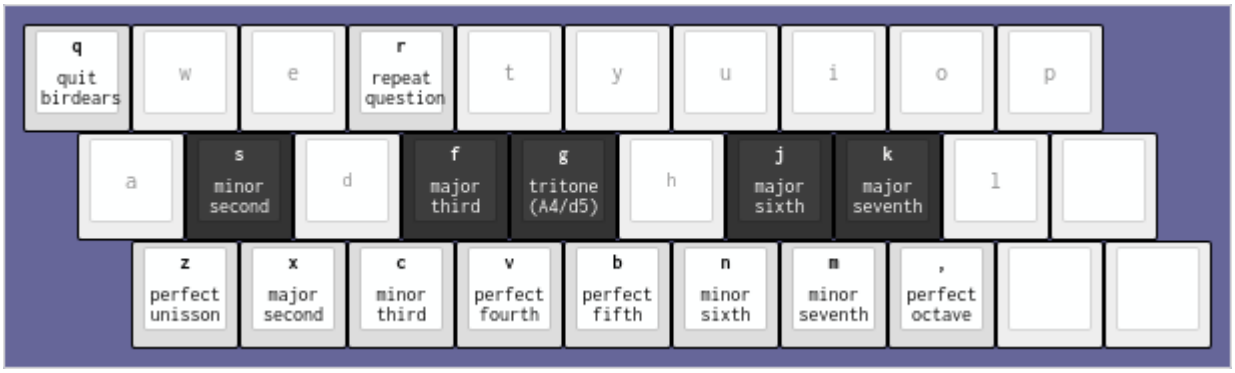

Fig. 6: Keyboard diagram for the `--mode minor`.

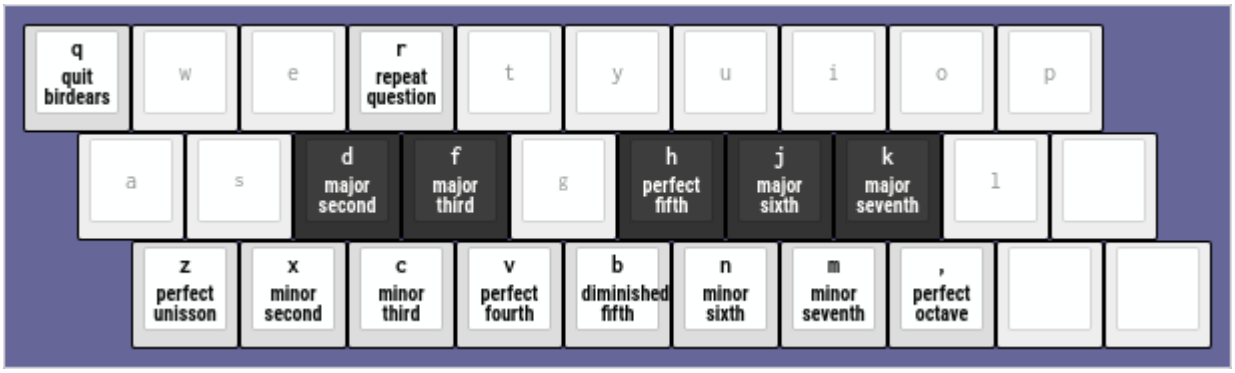

Fig. 7: Keyboard diagram for the `--mode locrian`.

CHAPTER

FIVE

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')**

Chromatic notes names using flats.

A mapping of the chromatic note names using flats.

Type
tuple

birdears.**CHROMATIC_SHARP = ('C', 'C#', 'D', 'D#', 'E', 'F', 'F#', 'G', 'G#', 'A', 'A#', 'B')**

Chromatic notes names using sharps.

A mapping of the chromatic note namesu sing sharps

Type
tuple

birdears.**CHROMATIC_TYPE = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11)**

A map of the chromatic scale.

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

Type
tuple

birdears.**CIRCLE_OF_FIFTHS = [('C', 'G', 'D', 'A', 'E', 'B', 'Gb', 'Db', 'Ab', 'Eb', 'Bb', 'F'), ('C', 'F', 'Bb', 'Eb', 'Ab', 'C#', 'F#', 'B', 'E', 'A', 'D', 'G')]**

Circle of fifths.

These are the circle of fifth in both directions.

Type
list of tuples

birdears.**D**(*data*, *nlines=0*)

birdears.**DEGREE_INDEX = {'i': [0], 'ii': [1, 2], 'iii': [3, 4], 'iv': [5, 6], 'v': [6, 7], 'vi': [8, 9], 'vii': [10, 11], 'viii': [12]}**

A mapping of semitones of each degree.

A mapping of semitones which index to each degree roman numeral, major/minor, perfect, augmented/diminished

Type
dict of lists

birdears.**DIATONIC_MASK** = {'dorian': (1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0), 'locrian': (1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0), 'lydian': (1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1), 'major': (1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1), 'minor': (1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0), 'mixolydian': (1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0), 'phrygian': (1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0)}

A map of the diatonic scale.

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

Type
dict of tuples

birdears.**INTERVALS** = ((0, 'P1', 'Perfect Unison'), (1, 'm2', 'Minor Second'), (2, 'M2', 'Major Second'), (3, 'm3', 'Minor Third'), (4, 'M3', 'Major Third'), (5, 'P4', 'Perfect Fourth'), (6, 'A4', 'Augmented Fourth'), (7, 'P5', 'Perfect Fifth'), (8, 'm6', 'Minor Sixth'), (9, 'M6', 'Major Sixth'), (10, 'm7', 'Minor Seventh'), (11, 'M7', 'Major Seventh'), (12, 'P8', 'Perfect Octave'), (13, 'A8', 'Minor Ninth'), (14, 'M9', 'Major Ninth'), (15, 'm10', 'Minor Tenth'), (16, 'M10', 'Major Tenth'), (17, 'P11', 'Perfect Eleventh'), (18, 'A11', 'Augmented Eleventh'), (19, 'P12', 'Perfect Twelfth'), (20, 'm13', 'Minor Thirteenth'), (21, 'M13', 'Major Thirteenth'), (22, 'm14', 'Minor Fourteenth'), (23, 'M14', 'Major Fourteenth'), (24, 'P15', 'Perfect Double-octave'), (25, 'A15', 'Minor Sixteenth'), (26, 'M16', 'Major Sixteenth'), (27, 'm17', 'Minor Seventeenth'), (28, 'M17', 'Major Seventeenth'), (29, 'P18', 'Perfect Eighteenth'), (30, 'A18', 'Augmented Eighteenth'), (31, 'P19', 'Perfect Nineteenth'), (32, 'm20', 'Minor Twentieth'), (33, 'M20', 'Major Twentieth'), (34, 'm21', 'Minor Twenty-first'), (35, 'M21', 'Major Twenty-first'), (36, 'P22', 'Perfect Triple-octave'))

Data representing intervals.

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

Type
tuple of tuples

birdears.**INTERVAL_INDEX** = {1: [0], 2: [1, 2], 3: [3, 4], 4: [5, 6], 5: [6, 7], 6: [8, 9], 7: [10, 11], 8: [12]}

A mapping of semitones of each interval.

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

Type
dict of lists

birdears.**KEYS** = ('C', 'C#', 'Db', 'D', 'D#', 'Eb', 'E', 'F', 'F#', 'Gb', 'G', 'G#', 'Ab', 'A', 'A#', 'Bb', 'B')

Allowed keys

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

Type
tuple

5.1 Subpackages

5.1.1 birdears.interfaces package

Submodules

birdears.interfaces.commandline module

class birdears.interfaces.commandline.**CommandLine**(*cli_prompt_next=False*, *cli_no_scroll=False*, *cli_no_resolution=False*, *exercise=None*, **args*, ***kwargs*)

Bases: object

process_key(*user_input*)

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()

birdears.interfaces.urwid module

class birdears.interfaces.urwid.**Keyboard**(*scale*, *question_tonic_pitch*, *main_loop=None*, *keyboard_index=None*, **args*, ***kwargs*)

Bases: Filler

highlight_key(*element=None*)

class birdears.interfaces.urwid.**KeyboardButton**(*top=''*, *middle=''*, *bottom=''*, *pitch=None*, **args*, ***kwargs*)

Bases: Padding

highlight(*state=False*)

birdears.interfaces.urwid.**Pad**(*weight=1*)

class birdears.interfaces.urwid.**QuestionWidget**(*top_widget=None*, *keyboard=None*, *bottom_widget=None*, *display=None*, **args*, ***kwargs*)

Bases: Padding

draw_display(*question_display*)

redraw_display(*question_display*)

class birdears.interfaces.urwid.**TextUserInterface**(*exercise=None*, **args*, ***kwargs*)

Bases: object

check_question(*user_input*)

create_question(*exercise*, ***kwargs*)

draw_question()

keypress(*key*)

run_question()

update_input_display()

update_question_display()

class birdears.interfaces.urwid.**TextUserInterfaceWidget**(**args*, ***kwargs*)

Bases: Frame

birdears.interfaces.urwid.**is_chromatic**(*key*)

5.1.2 birdears.questions package

Submodules

birdears.questions.harmonicinterval module

class birdears.questions.harmonicinterval.**HarmonicIntervalQuestion**(*mode='major'*, *tonic='C'*, *octave=4*, *descending=False*, *chromatic=False*, *n_octaves=1*, *valid_intervals=(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11)*, *user_durations=None*, *prequestion_method='none'*, *resolution_method='nearest_tonic'*, **args*, ***kwargs*)

Bases: QuestionBase

Implements a Harmonic Interval test.

check_question(*user_input_char*)

Checks whether the given answer is correct.

make_pre_question(*method*)

make_question()

This method should be overwritten by the question subclasses.

make_resolution(*method*)

This method should be overwritten by the question subclasses.

name = 'harmonic'

play_question(*callback=None*, *end_callback=None*, **args*, ***kwargs*)

This method should be overwritten by the question subclasses.

play_resolution(*callback=None*, *end_callback=None*, **args*, ***kwargs*)

birdears.questions.instrumentaldictation module

class birdears.questions.instrumentaldictation.**InstrumentalDictationQuestion**(*mode='major'*, *wait_time=11*, *n_repeats=1*, *max_intervals=3*, *n_notes=4*, *tonic='C'*, *octave=4*, *descending=False*, *chromatic=False*, *n_octaves=1*, *valid_intervals=(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11)*, *user_durations=None*, *prequestion_method='progression_i_iv_v_*, *resolution_method='repeat_only'*, **args*, ***kwargs*)

Bases: QuestionBase

Implements an instrumental dictation test.

check_question()

Checks whether the given answer is correct.

This currently doesn't applies to instrumental dictation questions.

make_pre_question(*method*)

make_question()

This method should be overwritten by the question subclasses.

make_resolution(*method*)

This method should be overwritten by the question subclasses.

name = 'instrumental'

play_question(*callback=None*, *end_callback=None*, **args*, ***kwargs*)

This method should be overwritten by the question subclasses.

birdears.questions.melodicdictation module

class birdears.questions.melodicdictation.**MelodicDictationQuestion**(*mode='major'*, *max_intervals=3*, *n_notes=4*, *tonic='C'*, *octave=4*, *descending=False*, *chromatic=False*, *n_octaves=1*, *valid_intervals=(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11)*, *user_durations=None*, *prequestion_method='progression_i_iv_v_i'*, *resolution_method='repeat_only'*, **args*, ***kwargs*)

Bases: QuestionBase

Implements a melodic dictation test.

check_question(*user_input_keys*)

Checks whether the given answer is correct.

make_pre_question(*method*)

make_question()

This method should be overwritten by the question subclasses.

make_resolution(*method*)

This method should be overwritten by the question subclasses.

name = 'dictation'

play_question(*callback=None*, *end_callback=None*, **args*, ***kwargs*)

This method should be overwritten by the question subclasses.

play_resolution(*callback=None*, *end_callback=None*, **args*, ***kwargs*)

birdears.questions.melodicinterval module

class birdears.questions.melodicinterval.**MelodicIntervalQuestion**(*mode='major'*, *tonic='C'*, *octave=4*, *descending=False*, *chromatic=False*, *n_octaves=1*, *valid_intervals=(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11)*, *user_durations=None*, *prequestion_method='tonic_only'*, *resolution_method='nearest_tonic'*, **args*, ***kwargs*)

Bases: QuestionBase

Implements a Melodic Interval test.

check_question(*user_input_char*)

Checks whether the given answer is correct.

make_pre_question(*method*)

make_question()

This method should be overwritten by the question subclasses.

make_resolution(*method*)

This method should be overwritten by the question subclasses.

name = 'melodic'

play_question(*callback=None*, *end_callback=None*, **args*, ***kwargs*)

This method should be overwritten by the question subclasses.

play_resolution(*callback=None*, *end_callback=None*, **args*, ***kwargs*)

birdears.questions.notename module

class birdears.questions.notename.**NoteNameQuestion**(*mode='major'*, *tonic='C'*, *octave=4*, *descending=False*, *chromatic=False*, *n_octaves=1*, *valid_intervals=(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11)*, *user_durations=None*, *prequestion_method='tonic_only'*, *resolution_method='nearest_tonic'*, **args*, ***kwargs*)

Bases: QuestionBase

Implements a Note Name test.

check_question(*user_input_char*)

Checks whether the given answer is correct.

make_pre_question(*method*)

make_question()

This method should be overwritten by the question subclasses.

make_resolution(*method*)

This method should be overwritten by the question subclasses.

name = 'notename'

play_question(*callback=None*, *end_callback=None*, **args*, ***kwargs*)

This method should be overwritten by the question subclasses.

play_resolution(*callback=None*, *end_callback=None*, **args*, ***kwargs*)

5.2 Submodules

5.3 birdears.exception module

exception birdears.exception.**InvalidNote**

Bases: Exception

exception birdears.exception.**InvalidOctave**

Bases: Exception

exception birdears.exception.**InvalidPitch**

Bases: Exception

exception birdears.exception.**InvalidSequenceElement**

Bases: Exception

5.4 birdears.interval module

class birdears.interval.**Interval**(*pitch_a*, *pitch_b*)

Bases: dict

This class represents the interval between two pitches..

tonic_octave

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

Type
int

interval octave

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

Type
int

chromatic_offset

The offset in semitones inside one octave. Relative semitones to tonic.

Type
int

note_and_octave

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

Type
str

note_name

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

Type
str

semitones

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

Type
int

is_chromatic

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

Type
bool

is_descending

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

Type
bool

diatonic_index

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.

Type
int

distance

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

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

Type
dict

data

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

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

Type
tuple

birdears.interval.**get_interval_by_semitones**(*semitones*)

5.5 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
```

(continues on next page)

(continued from previous page)

```
                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.

5.6 birdears.note_and_pitch module

class birdears.note_and_pitch.**Chord**(*iterable*)

Bases: list

append(*obj*)

Append object to the end of the list.

delay = None

duration = None

extend(*iterable*)

Extend list by appending elements from the iterable.

class birdears.note_and_pitch.**Note**(*note='C'*, *accident='sharp'*)

Bases: object

property pitch_class

class birdears.note_and_pitch.**Pitch**(*note='C'*, *octave=4*, *accident='sharp'*)

Bases: Note

delay = None

distance(*other*)

duration = None

property pitch_number

birdears.note_and_pitch.**get_abs_chromatic_offset**(*pitch1*, *pitch2*)

birdears.note_and_pitch.**get_pitch_by_number**(*numeric*, *accident='sharp'*)

birdears.note_and_pitch.**get_pitch_class**(*note*)

birdears.note_and_pitch.**get_pitch_number**(*note*, *octave*)

5.7 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

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

5.8 birdears.questionbase module

class birdears.questionbase.**QuestionBase**(*mode='major'*, *tonic='C'*, *octave=4*, *descending=False*, *chromatic=False*, *n_octaves=1*, *valid_intervals=(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11)*, *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.

check_question()

This method should be overwritten by the question subclasses.

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.**get_valid_pitches**(*scale*, *valid_intervals=(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11)*)

birdears.questionbase.**register_question_class**(*cls*, **args*, ***kwargs*)

Decorator for question classes.

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

5.9 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 resolves to the tonic.

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)

5.10 birdears.scale module

class birdears.scale.**ChromaticScale**(*tonic='C'*, *octave=4*, *n_octaves=1*, *descending=False*, *dont_repeat_tonic=False*)

Bases: ScaleBase

Builds a musical chromatic scale.

scale

The array of notes representing the scale.

Type
array_type

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='C'*, *mode='major'*, *octave=4*, *n_octaves=1*, *descending=False*, *dont_repeat_tonic=False*)

Bases: ScaleBase

Builds a musical diatonic scale.

scale

The array of notes representing the scale.

Type
array_type

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: list

5.11 birdears.sequence module

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

Bases: list

Register a Sequence of notes and/or chords.

elements

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

Type
array_type

async_play(*callback*, *end_callback*, *args*, *kwargs*)

Plays the Sequence elements of notes and/or chords and wait for *Sequence.pos_delay* seconds.

make_chord_progression(*tonic_pitch*, *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*, **args*, ***kwargs*)

5.12 birdears.utils module

class birdears.utils.**DictCallback**(*other=None*, ***kwargs*)

Bases: dict

callback = None

callback_args = []

callback_kwargs = {}

update([*E*], ***F*) → None. Update D from dict/iterable E and F.

If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

CHAPTER

SIX

SUPPORT

If you need help you can get in touch via IRC or file an issue on any matter regarding birdears at Github.

Media	Channel
IRC	#birdears
GitHub	https://github.com/iacchus/birdears
GH issues	https://github.com/iacchus/birdears/issues
ReadTheDocs	https://birdears.readthedocs.io
PyPI	https://pypi.python.org/pypi/birdears
TravisCI	https://travis-ci.org/iacchus/birdears
Coveralls	https://coveralls.io/github/iacchus/birdears

CHAPTER

SEVEN

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

EIGHT

INSTALLING BIRDEARS

8.1 Installing the dependencies

8.1.1 Arch Linux

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

8.2 Installing birdears

To install,simple do this command with pip3

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

8.2.1 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

NINE

USING BIRDEARS

9.1 What is Functional Ear Training

write me!

9.2 The method

We can use abc language to notate music within 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:|
```

9.3 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
```

9.3.1 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.
```

(continues on next page)

(continued from previous page)

```
  -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
```

9.3.2 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
```

(continues on next page)

(continued from previous page)

```
  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
```

9.3.3 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
```

(continues on next page)

(continued from previous page)

```
  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
```

9.3.4 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
```

(continues on next page)

(continued from previous page)

```
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
```

9.4 Loading from config/preset files

9.4.1 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

9.4.2 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.

9.5 Keybindings

9.5.1 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.

9.5.2 Major (Ionian)

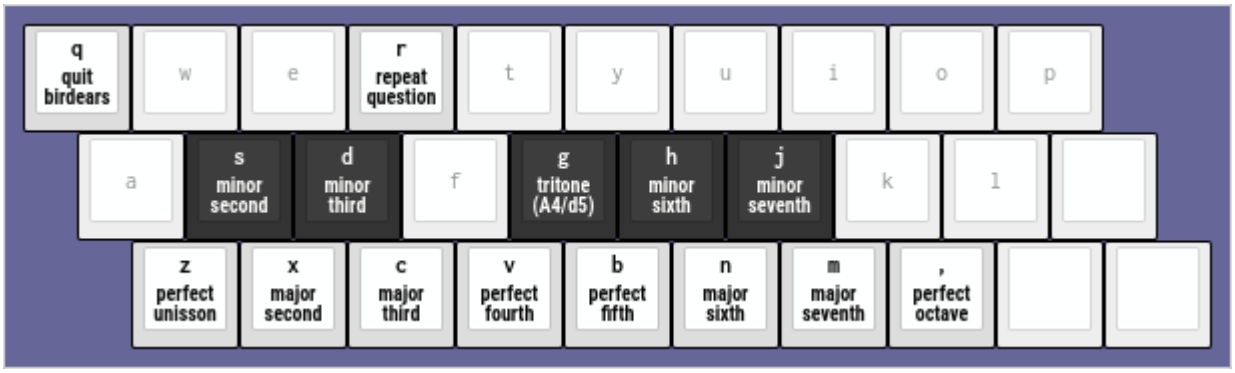

Fig. 1: Keyboard diagram for the `--mode major` (default).

9.5.3 Dorian

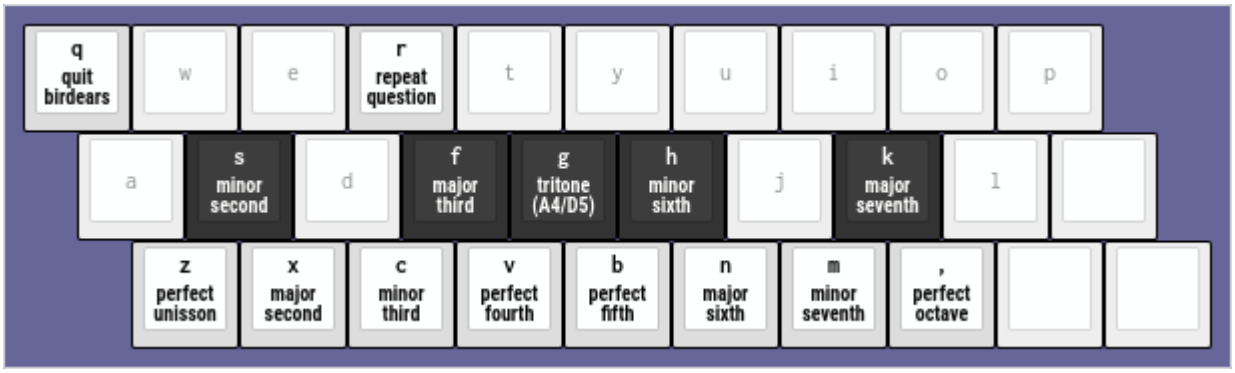

Fig. 2: Keyboard diagram for the `--mode dorian`.

9.5.4 Phrygian

9.5.5 Lydian

9.5.6 Mixolydian

9.5.7 Minor (Aeolian)

9.5.8 Locrian

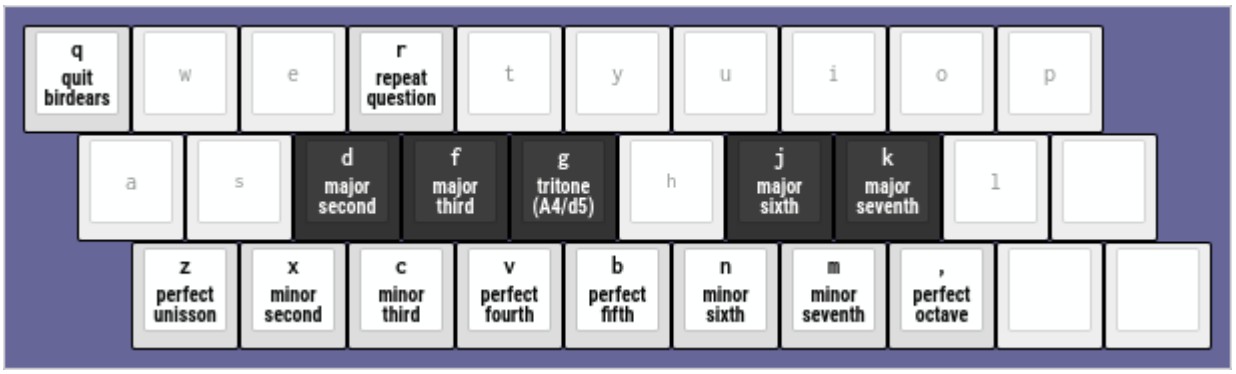

Fig. 3: Keyboard diagram for the `--mode phrygian`.

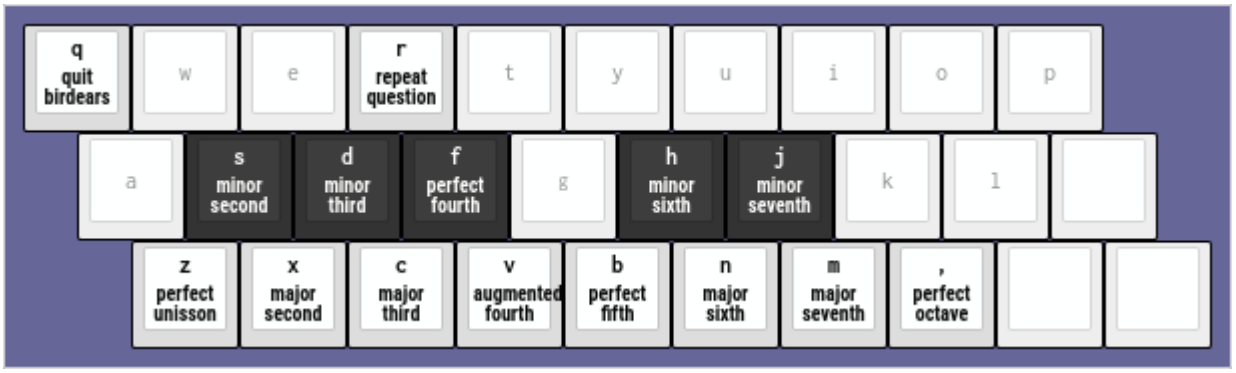

Fig. 4: Keyboard diagram for the `--mode lydian`.

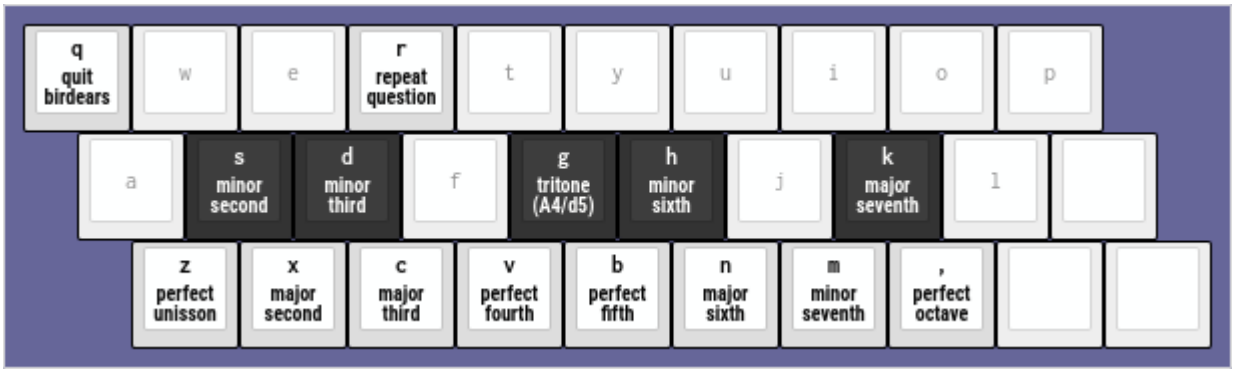

Fig. 5: Keyboard diagram for the `--mode mixolydian`.

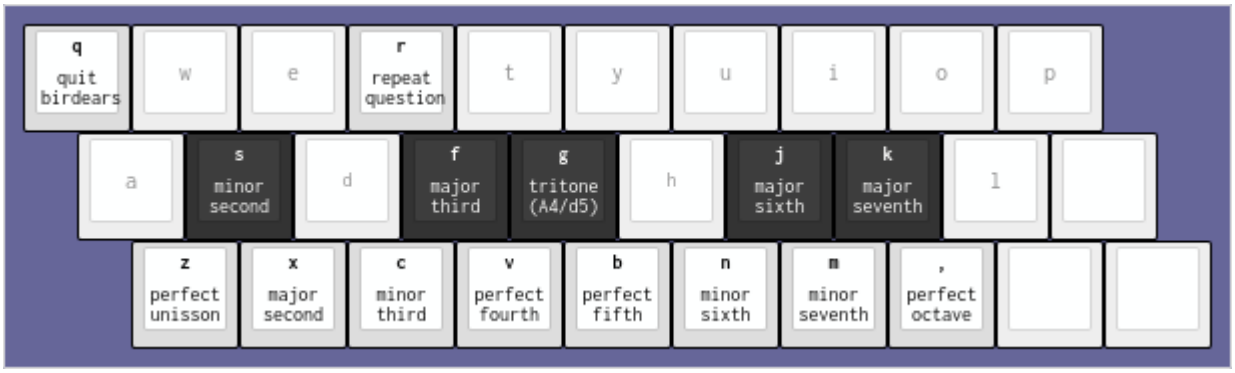

Fig. 6: Keyboard diagram for the `--mode minor`.

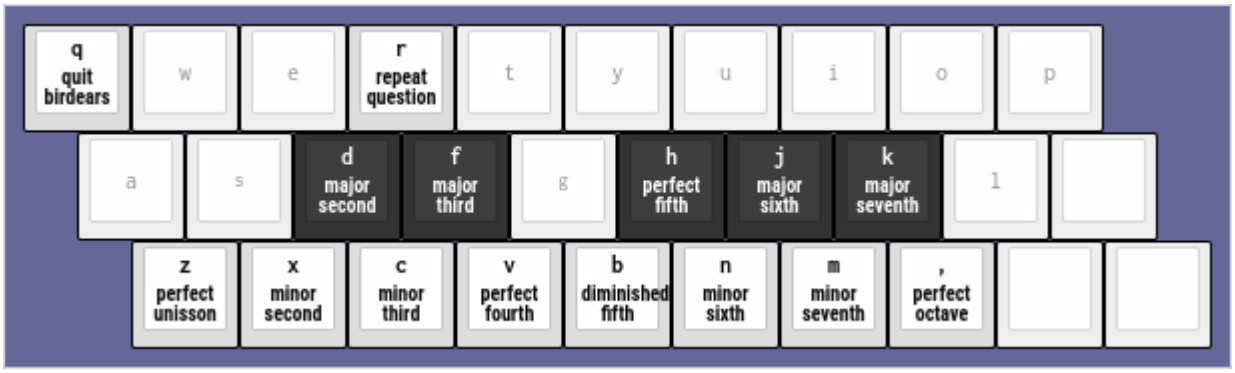

Fig. 7: Keyboard diagram for the `--mode locrian`.

CHAPTER

TEN

API

CHAPTER

ELEVEN

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')**

Chromatic notes names using flats.

A mapping of the chromatic note names using flats.

Type
tuple

birdears.**CHROMATIC_SHARP = ('C', 'C#', 'D', 'D#', 'E', 'F', 'F#', 'G', 'G#', 'A', 'A#', 'B')**

Chromatic notes names using sharps.

A mapping of the chromatic note namesu sing sharps

Type
tuple

birdears.**CHROMATIC_TYPE = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11)**

A map of the chromatic scale.

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

Type
tuple

birdears.**CIRCLE_OF_FIFTHS = [('C', 'G', 'D', 'A', 'E', 'B', 'Gb', 'Db', 'Ab', 'Eb', 'Bb', 'F'), ('C', 'F', 'Bb', 'Eb', 'Ab', 'C#', 'F#', 'B', 'E', 'A', 'D', 'G')]**

Circle of fifths.

These are the circle of fifth in both directions.

Type
list of tuples

birdears.**D**(*data*, *nlines=0*)

birdears.**DEGREE_INDEX = {'i': [0], 'ii': [1, 2], 'iii': [3, 4], 'iv': [5, 6], 'v': [6, 7], 'vi': [8, 9], 'vii': [10, 11], 'viii': [12]}**

A mapping of semitones of each degree.

A mapping of semitones which index to each degree roman numeral, major/minor, perfect, augmented/diminished

Type
dict of lists

birdears.**DIATONIC_MASK** = {'dorian': (1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0), 'locrian': (1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0), 'lydian': (1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1), 'major': (1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1), 'minor': (1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0), 'mixolydian': (1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0), 'phrygian': (1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0)}

A map of the diatonic scale.

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

Type
dict of tuples

birdears.**INTERVALS** = ((0, 'P1', 'Perfect Unison'), (1, 'm2', 'Minor Second'), (2, 'M2', 'Major Second'), (3, 'm3', 'Minor Third'), (4, 'M3', 'Major Third'), (5, 'P4', 'Perfect Fourth'), (6, 'A4', 'Augmented Fourth'), (7, 'P5', 'Perfect Fifth'), (8, 'm6', 'Minor Sixth'), (9, 'M6', 'Major Sixth'), (10, 'm7', 'Minor Seventh'), (11, 'M7', 'Major Seventh'), (12, 'P8', 'Perfect Octave'), (13, 'A8', 'Minor Ninth'), (14, 'M9', 'Major Ninth'), (15, 'm10', 'Minor Tenth'), (16, 'M10', 'Major Tenth'), (17, 'P11', 'Perfect Eleventh'), (18, 'A11', 'Augmented Eleventh'), (19, 'P12', 'Perfect Twelfth'), (20, 'm13', 'Minor Thirteenth'), (21, 'M13', 'Major Thirteenth'), (22, 'm14', 'Minor Fourteenth'), (23, 'M14', 'Major Fourteenth'), (24, 'P15', 'Perfect Double-octave'), (25, 'A15', 'Minor Sixteenth'), (26, 'M16', 'Major Sixteenth'), (27, 'm17', 'Minor Seventeenth'), (28, 'M17', 'Major Seventeenth'), (29, 'P18', 'Perfect Eighteenth'), (30, 'A18', 'Augmented Eighteenth'), (31, 'P19', 'Perfect Nineteenth'), (32, 'm20', 'Minor Twentieth'), (33, 'M20', 'Major Twentieth'), (34, 'm21', 'Minor Twenty-first'), (35, 'M21', 'Major Twenty-first'), (36, 'P22', 'Perfect Triple-octave'))

Data representing intervals.

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

Type
tuple of tuples

birdears.**INTERVAL_INDEX** = {1: [0], 2: [1, 2], 3: [3, 4], 4: [5, 6], 5: [6, 7], 6: [8, 9], 7: [10, 11], 8: [12]}

A mapping of semitones of each interval.

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

Type
dict of lists

birdears.**KEYS** = ('C', 'C#', 'Db', 'D', 'D#', 'Eb', 'E', 'F', 'F#', 'Gb', 'G', 'G#', 'Ab', 'A', 'A#', 'Bb', 'B')

Allowed keys

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

Type
tuple

11.1 Subpackages

11.2 Submodules

11.3 birdears.exception module

exception birdears.exception.**InvalidNote**

Bases: Exception

exception birdears.exception.**InvalidOctave**

Bases: Exception

exception birdears.exception.**InvalidPitch**

Bases: Exception

exception birdears.exception.**InvalidSequenceElement**

Bases: Exception

11.4 birdears.interval module

class birdears.interval.**Interval**(*pitch_a*, *pitch_b*)

Bases: dict

This class represents the interval between two pitches..

tonic_octave

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

Type
int

interval octave

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

Type
int

chromatic_offset

The offset in semitones inside one octave. Relative semitones to tonic.

Type
int

note_and_octave

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

Type
str

note_name

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

Type
str

semitones

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

Type
int

is_chromatic

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

Type
bool

is_descending

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

Type
bool

diatonic_index

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.

Type
int

distance

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

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

Type
dict

data

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

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

Type
tuple

birdears.interval.**get_interval_by_semitones**(*semitones*)

11.5 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.

11.6 birdears.note_and_pitch module

class birdears.note_and_pitch.**Chord**(*iterable*)

Bases: list

append(*obj*)

Append object to the end of the list.

delay = None

duration = None

extend(*iterable*)

Extend list by appending elements from the iterable.

class birdears.note_and_pitch.**Note**(*note='C'*, *accident='sharp'*)

Bases: object

property pitch_class

class birdears.note_and_pitch.**Pitch**(*note='C'*, *octave=4*, *accident='sharp'*)

Bases: Note

delay = None

distance(*other*)

duration = None

property pitch_number

birdears.note_and_pitch.**get_abs_chromatic_offset**(*pitch1*, *pitch2*)

birdears.note_and_pitch.**get_pitch_by_number**(*numeric*, *accident='sharp'*)

birdears.note_and_pitch.**get_pitch_class**(*note*)

birdears.note_and_pitch.**get_pitch_number**(*note*, *octave*)

11.7 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

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

11.8 birdears.questionbase module

class birdears.questionbase.**QuestionBase**(*mode='major'*, *tonic='C'*, *octave=4*, *descending=False*, *chromatic=False*, *n_octaves=1*, *valid_intervals=(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11)*, *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.

check_question()

This method should be overwritten by the question subclasses.

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.**get_valid_pitches**(*scale*, *valid_intervals=(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11)*)

birdears.questionbase.**register_question_class**(*cls*, **args*, ***kwargs*)

Decorator for question classes.

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

11.9 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 resolves to the tonic.

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)

11.10 birdears.scale module

class birdears.scale.**ChromaticScale**(*tonic='C'*, *octave=4*, *n_octaves=1*, *descending=False*, *dont_repeat_tonic=False*)

Bases: ScaleBase

Builds a musical chromatic scale.

scale

The array of notes representing the scale.

Type

array_type

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='C'*, *mode='major'*, *octave=4*, *n_octaves=1*, *descending=False*, *dont_repeat_tonic=False*)

Bases: ScaleBase

Builds a musical diatonic scale.

scale

The array of notes representing the scale.

Type

array_type

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: list

11.11 birdears.sequence module

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

Bases: list

Register a Sequence of notes and/or chords.

elements

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

Type
array_type

async_play(*callback*, *end_callback*, *args*, *kwargs*)

Plays the Sequence elements of notes and/or chords and wait for *Sequence.pos_delay* seconds.

make_chord_progression(*tonic_pitch*, *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*, **args*, ***kwargs*)

11.12 birdears.utils module

class birdears.utils.**DictCallback**(*other=None*, ***kwargs*)

Bases: dict

callback = None

callback_args = []

callback_kwargs = {}

update([*E*], ***F*) → None. Update D from dict/iterable E and F.

If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

CHAPTER

TWELVE

BIRDEARS.QUESTIONS PACKAGE

12.1 Submodules

12.2 birdears.questions.harmonicinterval module

class birdears.questions.harmonicinterval.**HarmonicIntervalQuestion**(*mode='major'*, *tonic='C'*, *octave=4*, *descending=False*, *chromatic=False*, *n_octaves=1*, *valid_intervals=(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11)*, *user_durations=None*, *prequestion_method='none'*, *resolution_method='nearest_tonic'*, **args*, ***kwargs*)

Bases: QuestionBase

Implements a Harmonic Interval test.

check_question(*user_input_char*)

Checks whether the given answer is correct.

make_pre_question(*method*)

make_question()

This method should be overwritten by the question subclasses.

make_resolution(*method*)

This method should be overwritten by the question subclasses.

name = 'harmonic'

play_question(*callback=None*, *end_callback=None*, **args*, ***kwargs*)

This method should be overwritten by the question subclasses.

play_resolution(*callback=None*, *end_callback=None*, **args*, ***kwargs*)

12.3 birdears.questions.instrumentaldictation module

class birdears.questions.instrumentaldictation.**InstrumentalDictationQuestion**(*mode='major'*, *wait_time=11*, *n_repeats=1*, *max_intervals=3*, *n_notes=4*, *tonic='C'*, *octave=4*, *descending=False*, *chromatic=False*, *n_octaves=1*, *valid_intervals=(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11)*, *user_durations=None*, *prequestion_method='progression_i_iv_v_*
resolution_method='repeat_only', **args*, ***kwargs*)

Bases: QuestionBase

Implements an instrumental dictation test.

check_question()

Checks whether the given answer is correct.

This currently doesn't applies to instrumental dictation questions.

make_pre_question(*method*)

make_question()

This method should be overwritten by the question subclasses.

make_resolution(*method*)

This method should be overwritten by the question subclasses.

name = 'instrumental'

play_question(*callback=None*, *end_callback=None*, **args*, ***kwargs*)

This method should be overwritten by the question subclasses.

12.4 birdears.questions.melodicdictation module

class birdears.questions.melodicdictation.**MelodicDictationQuestion**(*mode='major'*, *max_intervals=3*, *n_notes=4*, *tonic='C'*, *octave=4*, *descending=False*, *chromatic=False*, *n_octaves=1*, *valid_intervals=(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11)*, *user_durations=None*, *prequestion_method='progression_i_iv_v_i'*, *resolution_method='repeat_only'*, **args*, ***kwargs*)

Bases: QuestionBase

Implements a melodic dictation test.

check_question(*user_input_keys*)

Checks whether the given answer is correct.

make_pre_question(*method*)

make_question()

This method should be overwritten by the question subclasses.

make_resolution(*method*)

This method should be overwritten by the question subclasses.

name = 'dictation'

play_question(*callback=None*, *end_callback=None*, **args*, ***kwargs*)

This method should be overwritten by the question subclasses.

play_resolution(*callback=None*, *end_callback=None*, **args*, ***kwargs*)

12.5 birdears.questions.melodicinterval module

class birdears.questions.melodicinterval.**MelodicIntervalQuestion**(*mode='major'*, *tonic='C'*, *octave=4*, *descending=False*, *chromatic=False*, *n_octaves=1*, *valid_intervals=(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11)*, *user_durations=None*, *prequestion_method='tonic_only'*, *resolution_method='nearest_tonic'*, **args*, ***kwargs*)

Bases: QuestionBase

Implements a Melodic Interval test.

check_question(*user_input_char*)

Checks whether the given answer is correct.

make_pre_question(*method*)

make_question()

This method should be overwritten by the question subclasses.

make_resolution(*method*)

This method should be overwritten by the question subclasses.

name = 'melodic'

play_question(*callback=None*, *end_callback=None*, **args*, ***kwargs*)

This method should be overwritten by the question subclasses.

play_resolution(*callback=None*, *end_callback=None*, **args*, ***kwargs*)

12.6 birdears.questions.notename module

class birdears.questions.notename.**NoteNameQuestion**(*mode='major'*, *tonic='C'*, *octave=4*, *descending=False*, *chromatic=False*, *n_octaves=1*, *valid_intervals=(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11)*, *user_durations=None*, *prequestion_method='tonic_only'*, *resolution_method='nearest_tonic'*, **args*, ***kwargs*)

Bases: QuestionBase

Implements a Note Name test.

check_question(*user_input_char*)

Checks whether the given answer is correct.

make_pre_question(*method*)

make_question()

This method should be overwritten by the question subclasses.

make_resolution(*method*)

This method should be overwritten by the question subclasses.

name = 'notename'

play_question(*callback=None*, *end_callback=None*, **args*, ***kwargs*)

This method should be overwritten by the question subclasses.

play_resolution(*callback=None*, *end_callback=None*, **args*, ***kwargs*)

CHAPTER

THIRTEEN

BIRDEARS.INTERFACES PACKAGE

13.1 Submodules

13.2 birdears.interfaces.commandline module

class birdears.interfaces.commandline.**CommandLine**(*cli_prompt_next=False*, *cli_no_scroll=False*, *cli_no_resolution=False*, *exercise=None*, **args*, ***kwargs*)

Bases: object

process_key(*user_input*)

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()

13.3 birdears.interfaces.urwid module

class birdears.interfaces.urwid.**Keyboard**(*scale*, *question_tonic_pitch*, *main_loop=None*, *keyboard_index=None*, **args*, ***kwargs*)

Bases: Filler

highlight_key(*element=None*)

class birdears.interfaces.urwid.**KeyboardButton**(*top=''*, *middle=''*, *bottom=''*, *pitch=None*, **args*, ***kwargs*)

Bases: Padding

highlight(*state=False*)

birdears.interfaces.urwid.**Pad**(*weight=1*)

class birdears.interfaces.urwid.**QuestionWidget**(*top_widget=None*, *keyboard=None*, *bottom_widget=None*, *display=None*, **args*, ***kwargs*)

Bases: Padding

draw_display(*question_display*)

redraw_display(*question_display*)

class birdears.interfaces.urwid.**TextUserInterface**(*exercise=None*, **args*, ***kwargs*)

Bases: object

check_question(*user_input*)

create_question(*exercise*, ***kwargs*)

draw_question()

keypress(*key*)

run_question()

update_input_display()

update_question_display()

class birdears.interfaces.urwid.**TextUserInterfaceWidget**(**args*, ***kwargs*)

Bases: Frame

birdears.interfaces.urwid.**is_chromatic**(*key*)

INDEX

D

E

G

H

I

K

L

M

N

P

Q

R

S

T

U