# conchord reference

## 1. Chord tabstring generation

```
get-chords(name: str, tuning: str, at: int none, true-bass) -> array[str]
```

Gets all possible chord strings with given tuning (and optionally at given fret) Complex chord with omitted perfect fifth will have ? in end

```
#get-chords("Cmaj7").slice(0, 10)
```

```
(
    "x32000",
    "x35000",
    "x32003",
    "x32400?",
    "x35400",
    "x35500",
    "x32500?",
    "x32403",
    "x3x000",
    "x35450",
)
```

**Parameters:**

name ( str ) – Chord name

tuning ( str = default-tuning) – Tuning in format "A B C"

at ( int or none = none) – What fret to find chords at

true-bass (= true) – Whether to require the lowest note to be the root note. Note that doesn't affect chords with / that set bass, like A/E. You can abuse it to make chords have true bass with Am/A.

Best to leave true for guitar, but false for ukulele, where the bas is not as important

```
get-chord(name, n, tuning, at)
```

Gets individual chord string

```
default-tuning
```

Classic 6-string Guitar tuning: E1 A1 D2 G2 B2 E3

## 2. Chord drawing

```
new-chordgen(
    shadow-barre: int ,
    string-number: int ,
    scale-length: length ,
    colors: dictionary ,
    number-to-left: boolean ,
    thick-nut: boolean ,
    use-shadow-barre: bool
  ) -> function[(tabs, name, scale-l)→chord]
```

1. Creates a new chordgen: a new function that takes tabstring, name and scale-length and returns a rendered chord block

**Parameters:**

shadow-barre ( int = 0) – length of semi-visible upper part of barre (default 0)

string-number ( int = 6) – number of strings of the instrument, default is 6

scale-length ( length = 1pt) – outputs canvas with roughly height=80 * scale-length and width=((string-number + 1)**10 + 5)** scale-length

colors ( dictionary = (:)) – colors: dictionary with colors for image
- grid: color of grid, default is gray.darken(20%)
- open: color of circles for open strings, default is black
- muted: color of crosses for muted strings, default is black
- hold: color of held positions, default is #5d6eaf
- barre: color of main barre part, default is #5d6eaf
- shadow-barre: color of "unnecessary" barre part, default is #5d6eaf.lighten(30%)

colors and other properties of fret and chord name you can specify using show rules for text and raw (fret is raw)

number-to-left ( boolean = false) – whether to display to the left

thick-nut ( boolean = true) – whether to draw thick nut

use-shadow-barre ( bool = true) – Whether to use shadow barre

```
get-chordgram-width-scale(n-strings: int ) -> float
```

2. The width of the chord diagram will be roughly this * scape-length

**Parameters:**

n-strings ( int ) – Number of strins in chord

```
parse-tabstring(string-tab) -> (array, boolean)
```

3. Parses tabstring

```
generate-chord(
    tabs:  array[int | "x"],
    name:  str,
    string-number:  int,
    force-barre:  int,
    use-shadow-barre:  bool,
    scale-length,
    colors,
    number-to-left,
    thick-nut
  ) -> chord
```

4. Generates chord image with simple rules, for inner use mostly

**Parameters:**

tabs ( array[int | "x"] ) – array of parsed tabstring, "x" (mute) and numbers are accepted

name ( str = "" ) – displayed name

string-number ( int = 6) – total number of strings instrument has

force-barre ( int = 0) – 0 → standard algorithm, 1 → force add barre, –1 → force avoid barre

scale-length (= 1pt) – see new-chordgen for this and other parameters

```
render-chord(
    hold:  array[(int, int)],
    open:  array[int],
    muted:  array[int],
    fret-number:  int,
    name:  str,
    barre:  int,
    barre-shift:  int,
    shadow-barre:  int,
    string-number:  int,
    scale-length:  length,
    colors:  dictionary,
    number-to-left:  boolean,
    thick-nut:  boolean
  ) -> chord
```

5. Renders the chord

   *Important*: for the convenience there all strings are numbered *from the top* (e.g. A will be 1)

**Parameters:**

hold ( array[(int, int)] ) – array of coords of positions held; string first, then shift

open ( array[int] ) – array of numbers of opened strings

muted ( array[int] ) – array of numbers for muted

fret-number ( int ) – the starting fret

name ( str ) – displayed name

`barre` (`int` = `0`) – length of barre if present; ZERO means NO

`barre-shift` (`int` = `0`) – shift of the barre; usually no, but there are exceptions

`shadow-barre` (`int` = `0`) – length of semi-visible upper part of barre (default 0)

`string-number` (`int` = `6`) – number of strings of the instrument, default is 6

`scale-length` (`length` = `1pt`) – outputs canvas with roughly `height=80 * scale-length` and width=((string-number + 1)**10 + 5)** scale-length

`colors` (`dictionary` = `(:)`) – colors: dictionary with colors for image
- grid: color of grid, default is `gray.darken(20%)`
- open: color of circles for open strings, default is `black`
- muted: color of crosses for muted strings, default is `black`
- hold: color of held positions, default is `#5d6eaf`
- barre: color of main barre part, default is `#5d6eaf`
- shadow-barre: color of "unnecessary" barre part, default is `#5d6eaf.lighten(30%)`

colors and other properties of fret and chord name you can specify using show rules for text and raw (fret is `raw`)

`number-to-left` (`boolean` = `false`) – whether to display to the left

`thick-nut` (`boolean` = `true`) – whether to draw thick nut
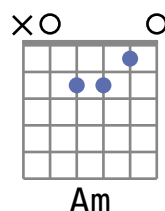
# 3. Smart chord

```
smart-chord(
    name: str,
    chordgen,
    n: int,
    tuning,
    true-bass,
    at,
    scale-l
) -> chord
```

1. Function that renders chord by its name



```
#smart-chord("Am")
```

**Parameters:**

`name` (`str`) – chord name

`chordgen` (= `red-missing-fifth`) – chordgen to use, the default one marks imperfect chords with red hold points

`n` (`int` = `0`) – number of chord to select, the "best" is zero

tuning (= `default-tuning`) – tuning string in format "A B C D E"

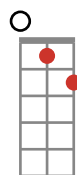true-bass (= `true`) – whether to require the lowest note to be the root note

at (= `none`) – at which fret to search chord

scale-l (= `1pt`) – see `draw-chord` for reference

```
red-missing-fifth(tabs, name, scale-l) -> chord
```

2. A chordgen that marks missing perfect fifth chords with red hold points. That means chords with ? in the end will be *red*.



```
#red-missing-fifth("012?")
```

# 4. Song sheets

```
overchord(
    text:    str,
    styling: (text <chord>) => content,
    align:   alignment,
    height:  length,
    width:   length
) -> chord
```

1. A simple function to place chord over text. Attaches tag to the text to apply tonality and make a chordlib. May be replaced with any custom.

   Just add chord labels above lyrics in arbitrary place, don't think about what letter exactly it should be located. By default `overchord` aligns the chord label to the left, so it produces pretty results out-of-box. You can pass other alignments to `align` argument, or use the chords straight inside words.

   Feel free to use it for your purposes outside of the package.
   It takes on default `-0.25em` width to remove one adjacent space, so
   • To make it work on monospace/other special fonts, you will need to adjust `width` argument.
      The problem is that I can't `measure` space, but maybe that will be eventually fixed.
   • To add chord inside word, you have to add *one* space, like wo #chord[Am]rd.

**Parameters:**

text ( `str` ) – chord name to attach. Should be plain string for tagging to work

styling ( `(text <chord>) => content` = `strong`) – styling function that is applied to the string

align ( `alignment` = `start`) – alignment of the word above the point

height ( `length` = `1em`) – height of the chords

width ( `length` = `-0.25em`) – width of space in current font, may be set to zero if you don't put any spaces between chords and words

```
inlinechord(text, styling) -> content
```

1a. A replacement for overchord, displays chords inline in (double) square brackets

```
fulloverchord(
    name: string,
    styling: (text <chord>) => content,
    align: alignment,
    height: length,
    width: length,
    smart-chord,
    scale-length,
    ..args
) -> content
```

1b. An overchord alternative, displays a chord above line that is changed with tonality

**Parameters:**

name ( `string` ) – chord name

styling ( `(text <chord>) => content` = `strong`) – styling function that is applied to the string

align ( `alignment` = `start`) – alignment of the word above the point

height ( `length` = `40pt`) – height of the chords

width ( `length` = `-0.25em`) – width of space in current font, may be set to zero if you don't put any spaces between chords and words

```
chordify(doc: content, squarechords: boolean, line-chord: function(name) → content,
  heading-reset-tonality) -> content
```

2. Use `#show:` `chordify` in your document to allow auto square chords formatting and automatic tonality change inspired by soxfox42's chordish

**Parameters:**

doc ( `content` ) – the document to apply show rule

squarechords ( `boolean` = `true`) – enable square brackets for chords writing

line-chord ( `function(name) → content` = `overchord`) – function applying to the chord names when square brackets are used

```
chordlib(
    smart-chord,
    chordgen,
    tuning: str,
    true-bass,
    exclude: array[str],
    switch: dictionary[int],
    at: dictionary[int   none],
    scale-l: length,
    heading-level: int
) -> sequence[content]
```

3.  Render all chords of current song.
    - Set `header-level` to set headings that separate the different songs. If none, all chords in document will be rendered.

    *This must be inside `context` to work*

**Parameters:**

`smart-chord` (= `smart-chord`) – smart chord function to use

`chordgen` (= `red-missing-fifth`) – chordgen for smart-chord

`tuning` (`str` = `default-tuning`) – tuning to use in "A B C D" format

`true-bass` (= `true`) – whether to require the lowest note to be the root note

`exclude` (`array[str]` = `()`) – chords not to draw, can be added manually in format ("Am", …)

`switch` (`dictionary[int]` = `(:)`) – versions of chords to use (default zero is the "best") in format (Am: 2, …)

`at` (`dictionary[int` or `none]` = `(:)`) – at witch fret to find the best chord in format (Am: 5, …)

`scale-l` (`length` = `1pt`) – scale length, see `draw-chord`

`heading-level` (`int` = `none`) – heading level to search chords within

```
sized-chordlib(
    N: int,
    width: length,
    prefix: content,
    postfix: content,
    inset,
    ..args
) -> content
```

4.  Draw a nice box with chords inside

**Parameters:**

`N` (`int` = `2`) – number of chords inside one line

`width` (`length` = `130pt`) – width of the box

`prefix` (`content` = `none`) – content to add to chordbox start

postfix ( `content` = `none`) – content to add to chords end (e.g., some excluded chords)

inset (= `10pt`) – inset for block to use

..args () – all the other args of `chordlib`

```
change-tonality(tonality-shift: int ) -> content
```

5. Changes current tonality shift to given number This is just metadata, so you need to put into document to have any effect

**Parameters:**

`tonality-shift` ( `int` ) – number of halftones to move tonality

```
auto-tonality-chord(name: str , smart-chord: function(name, ..args) → chord ,
  ..args: any ) -> chord
```

6. Smart chord that changes tonality automatically

**Parameters:**

`name` ( `str` ) – chord name

`smart-chord` ( `function(name, ..args) → chord` = `smart-chord`) – smart chord method to use

..args ( `any` ) – arguments for smart-chord

```
get-tonality(loc: content location ) -> int
```

7. get current tonality in document

**Parameters:**

`loc` ( `content` or `location` ) – Element that has location or `location`

```
shift-chord-tonality(chord: str , tonality: int ) -> str
```

8. Shifts tonality of given chord name by given amount with regexes

**Parameters:**

`chord` ( `str` ) – chord name

`tonality` ( `int` ) – number of halftones to move tonality

```
inside-level-selector(select, heading-level) -> selector
```

Utility function Selects all things inside current "chapter"

# 5. Tabs

```
new(
    tabs: raw,
    preamble: cetz drawing,
    extra: cetz drawing,
    eval-scope: dictionary,
    scale-length: length,
    s-num: int,
    one-beat-length: float,
    line-spacing: float,
    enable-scale: boolean,
    colors: dictionary,
    autoscale-max: float,
    autoscale-min: float,
    draw-rhythm,
    debug-render: int none,
    debug-numbers: bool
  ) -> content
```

Creates a new tab line

**Parameters:**

tabs ( raw ) – the tab code; see README for rough specification

preamble ( cetz drawing = none) – what to add at the "start" of tab canvas

extra ( cetz drawing = none) – what to add at the "end" of tab canvas

eval-scope ( dictionary = (:)) – scope for your code for custom elements

scale-length ( length = 0.3cm) – canvas scale length

s-num ( int = 6) – number of strings

one-beat-length ( float = 8) – length in cetz points of one beat

line-spacing ( float = 3) – spacing between the lines

enable-scale ( boolean = true) – enable smart scaling for better fitting to line

colors ( dictionary = (:)) – colors of things, see README

autoscale-max ( float = 3.0) – maximum scaling for smart scale

autoscale-min ( float = 0.9) – minimal scaling for smart scale

draw-rhythm (= false) – draw "rhythm" bar

debug-render ( int or none = none) – render this number of notes only

debug-numbers ( bool = false) – draw numbers of step