

---

# **pychord Documentation**

**Author**

**Mar 12, 2023**



---

## Contents:

---

<b>1</b>	<b>pychord package</b>	<b>1</b>
1.1	Subpackages . . . . .	1
1.1.1	pychord.constants package . . . . .	1
1.1.1.1	Submodules . . . . .	1
1.1.1.2	pychord.constants.qualities module . . . . .	1
1.1.1.3	pychord.constants.scales module . . . . .	1
1.1.1.4	Module contents . . . . .	1
1.2	Submodules . . . . .	1
1.3	pychord.analyzer module . . . . .	1
1.4	pychord.chord module . . . . .	2
1.5	pychord.parser module . . . . .	3
1.6	pychord.progression module . . . . .	3
1.7	pychord.quality module . . . . .	4
1.8	pychord.utils module . . . . .	5
1.9	Module contents . . . . .	5
<b>2</b>	<b>Indices and tables</b>	<b>7</b>
	<b>Python Module Index</b>	<b>9</b>
	<b>Index</b>	<b>11</b>



# CHAPTER 1

---

## pychord package

---

### 1.1 Subpackages

#### 1.1.1 pychord.constants package

##### 1.1.1.1 Submodules

###### 1.1.1.2 pychord.constants.qualities module

###### 1.1.1.3 pychord.constants.scales module

###### 1.1.1.4 Module contents

### 1.2 Submodules

### 1.3 pychord.analyzer module

`pychord.analyzer.find_chords_from_notes (notes: List[str]) → List[pychord.chord.Chord]`  
Find possible chords consisted from notes

**Parameters** `notes` – List of note arranged from lower note. ex) [“C”, “Eb”, “G”]

**Returns** List of chord

`pychord.analyzer.get_all_rotated_notes (notes: List[str]) → List[List[str]]`  
Get all rotated notes

`get_all_rotated_notes([A,C,E]) -> [[A,C,E],[C,E,A],[E,A,C]]`

`pychord.analyzer.notes_to_positions (notes: List[str], root: str) → List[int]`  
Get notes positions from the root note

```
>>> notes_to_positions(["C", "E", "G"], "C")
[0, 4, 7]
```

#### Parameters

- **notes** – List of notes
- **root** – Root note

**Returns** List of note positions

## 1.4 pychord.chord module

**class** pychord.chord.Chord(*chord: str*)

Bases: object

Class to handle a chord.

**Attributes:** *\_chord*: Name of the chord. (e.g. C, Am7, F#m7-5/A) *\_root*: The root note of chord. (e.g. C, A, F#) *\_quality*: The quality of chord. (e.g. maj, m7, m7-5) *\_appended*: The appended notes on chord. *\_on*: The base note of slash chord.

#### appended

The appended notes on chord

#### chord

The name of chord

**components** (*visible: bool = True*) → Union[List[str], List[int]]

Return the component notes of chord

**Parameters** **visible** – returns the name of notes if True else list of int

**Returns** component notes of chord

**components\_with\_pitch** (*root\_pitch: int*) → List[str]

Return the component notes of chord formatted like [“C4”, “E4”, “G4”]

**Parameters** **root\_pitch** – the pitch of the root note

**Returns** component notes of chord

**classmethod from\_note\_index** (*note: int, quality: str, scale: str, diatonic: bool = False, chromatic: int = 0*) → pychord.chord.Chord

Create a Chord from note index in a scale

Chord.from\_note\_index(1, “”, “Cmaj”) returns I of C major => Chord(“C”) Chord.from\_note\_index(3, “m7”, “Fmaj”) returns IIImin of F major => Chord(“Am7”) Chord.from\_note\_index(5, “7”, “Amin”) returns Vmin of A minor => Chord(“E7”) Chord.from\_note\_index(2, “”, “Cmaj”) returns II of C major => Chord(“D”) Chord.from\_note\_index(2, “m”, “Cmaj”) returns IImin of C major => Chord(“Dm”) Chord.from\_note\_index(2, “”, “Cmaj”, diatonic=True) returns IImin of C major => Chord(“Dm”) Chord.from\_note\_index(2, “”, “Cmin”, chromatic=-1) returns bII of C minor => Chord(“Db”)

#### Parameters

- **note** – Scale degree of the chord’s root (1-7)
- **quality** – Quality of the chord (e.g. m7, sus4)
- **scale** – Base scale (e.g. Cmaj, Amin, F#maj, Ebmin)

- **diatonic** – If True, chord quality is determined using the base scale (overrides :param quality)
- **chromatic** – Lower or raise the scale degree (and all notes of the chord) by semitone(s)

**info()**

Return information of chord to display

**on**

The base note of slash chord

**quality**

The quality of chord

**root**

The root note of chord

**transpose (trans: int, scale: str = 'C')** → None

Transpose the chord

**Parameters**

- **trans** – Transpose key
- **scale** – key scale

## 1.5 pychord.parser module

**pychord.parser.parse (chord: str)** → Tuple[str, pychord.quality.Quality, List[str], str]

Parse a string to get chord component

**Parameters** **chord** – str expression of a chord

**Returns** (root, quality, appended, on)

## 1.6 pychord.progression module

**class pychord.progression.ChordProgression (initial\_chords: Union[str, pychord.chord.Chord], py-  
chord.chord.Chord, List[Union[str, pychord.chord.Chord]]] = [])**

Bases: object

Class to handle chord progressions.

**Attributes:** \_chords: component chords of chord progression.

**append (chord: Union[str, pychord.chord.Chord])** → None

Append a chord to chord progressions

**Parameters** **chord** – A chord to append

**chords**

Get component chords of chord progression

**insert (index: int, chord: Union[str, pychord.chord.Chord])** → None

Insert a chord to chord progressions

**Parameters**

- **index** – Index to insert a chord

- **chord** – A chord to insert

**pop** (*index: int = -1*) → pychord.chord.Chord  
Pop a chord from chord progressions

**Parameters index** – Index of the chord to pop (default: -1)

**transpose** (*trans: int*) → None  
Transpose whole chord progressions

**Parameters trans** – Transpose key

## 1.7 pychord.quality module

**class** pychord.quality.Quality (*name: str, components: Tuple[int, ...]*)

Bases: object

Chord quality

**append\_on\_chord** (*on\_chord, root*)

Append on chord

To create Am7/G q = Quality('m7') q.append\_on\_chord('G', root='A')

**Parameters**

- **on\_chord** (*str*) – bass note of the chord
- **root** (*str*) – root note of the chord

**get\_components** (*root='C', visible=False*)

Get components of chord quality

**Parameters**

- **root** (*str*) – the root note of the chord
- **visible** (*bool*) – returns the name of notes if True

**Return type** list[str/int]

**Returns** components of chord quality

**quality**

Get name of quality

**class** pychord.quality.QualityManager

Bases: object

Singleton class to manage the qualities

**find\_quality\_from\_components** (*components: List[int]*)

Find a quality from components

**Parameters components** – components of quality

**get\_quality** (*name: str, inversion: int = 0*) → pychord.quality.Quality

**load\_default\_qualities** ()

**set\_quality** (*name: str, components: Tuple[int, ...]*)

Set a Quality

This method will not affect any existing Chord instances. :param name: name of quality :param components: components of quality

## 1.8 pychord.utils module

`pychord.utils.display_appended(appended: List[str]) → str`

`pychord.utils.display_on(on_note: str) → str`

`pychord.utils.note_to_val(note: str) → int`

Get index value of a note

```
>>> note_to_val("C")
0
>>> note_to_val("B")
11
```

`pychord.utils.transpose_note(note: str, transpose: int, scale: str = 'C') → str`

Transpose a note

```
>>> transpose_note("C", 1)
"Db"
>>> transpose_note("D", 4, "A")
"F#"
```

`pychord.utils.val_to_note(val: int, scale: str = 'C') → str`

Return note by index in a scale

```
>>> val_to_note(0)
"C"
>>> val_to_note(11, "D")
"D#"
```

## 1.9 Module contents



## CHAPTER 2

---

### Indices and tables

---

- genindex
- modindex
- search



---

## Python Module Index

---

### p

pychord, 5  
pychord.analyzer, 1  
pychord.chord, 2  
pychord.constants, 1  
pychord.constants.qualities, 1  
pychord.constants.scales, 1  
pychord.parser, 3  
pychord.progression, 3  
pychord.quality, 4  
pychord.utils, 5



---

## Index

---

### A

append() (*pychord.progression.ChordProgression method*), 3  
append\_on\_chord() (*pychord.quality.Quality method*), 4  
appended (*pychord.chord.Chord attribute*), 2

### C

Chord (*class in pychord.chord*), 2  
chord (*pychord.chord.Chord attribute*), 2  
ChordProgression (*class in pychord.progression*), 3  
chords (*pychord.progression.ChordProgression attribute*), 3  
components() (*pychord.chord.Chord method*), 2  
components\_with\_pitch() (*pychord.chord.Chord method*), 2

### D

display\_appended() (*in module pychord.utils*), 5  
display\_on() (*in module pychord.utils*), 5

### F

find\_chords\_from\_notes() (*in module pychord.analyzer*), 1  
find\_quality\_from\_components() (*pychord.quality.QualityManager method*), 4  
from\_note\_index() (*pychord.chord.Chord class method*), 2

### G

get\_all\_rotated\_notes() (*in module pychord.analyzer*), 1  
get\_components() (*pychord.quality.Quality method*), 4  
get\_quality() (*pychord.quality.QualityManager method*), 4

### I

info() (*pychord.chord.Chord method*), 3

insert() (*pychord.progression.ChordProgression method*), 3

### L

load\_default\_qualities() (*pychord.quality.QualityManager method*), 4

### N

note\_to\_val() (*in module pychord.utils*), 5  
notes\_to\_positions() (*in module pychord.analyzer*), 1

### O

on (*pychord.chord.Chord attribute*), 3

### P

parse() (*in module pychord.parser*), 3  
pop() (*pychord.progression.ChordProgression method*), 4  
pychord (*module*), 5  
pychord.analyzer (*module*), 1  
pychord.chord (*module*), 2  
pychord.constants (*module*), 1  
pychord.constants.qualities (*module*), 1  
pychord.constants.scales (*module*), 1  
pychord.parser (*module*), 3  
pychord.progression (*module*), 3  
pychord.quality (*module*), 4  
pychord.utils (*module*), 5

### Q

Quality (*class in pychord.quality*), 4  
quality (*pychord.chord.Chord attribute*), 3  
quality (*pychord.quality.Quality attribute*), 4  
QualityManager (*class in pychord.quality*), 4

### R

root (*pychord.chord.Chord attribute*), 3

**S**

`set_quality()` (*pychord.quality.QualityManager method*), [4](#)

**T**

`transpose()` (*pychord.chord.Chord method*), [3](#)

`transpose()` (*pychord.progression.ChordProgression method*), [4](#)

`transpose_note()` (*in module pychord.utils*), [5](#)

**V**

`val_to_note()` (*in module pychord.utils*), [5](#)