Poker Documentation

Release 0.30.0

Kiss György

Sep 09, 2019

Contents

A Python framework for poker related operations.

It contains classes for parsing card Suits, Cards, Hand combinations (called Combos), construct hand Ranges and check for syntax, parse Hand histories.

It can get information from poker related websites like Pocketfives, TwoplusTwo Forum, or PokerStars website by scraping them. In the long term, it will have a fast hand evaluator and an equity calculator.

It uses the MIT license, so its code can be used in any product without legal consequences.

It aims for quality, fully tested code and easy usability with nice APIs, suitable for beginners to play with.

CHAPTER 1

Contents

1.1 Installation

The package requires Python 3.6+ only from version 0.30.0.

Simple from PYPI:

```
$ pip install poker
```

Advanced, directly from package in development mode:

```
$ git clone git@github.com:pokerregion/poker.git
$ cd poker
$ pip install -e .
```

1.2 Basic operations

1.2.1 Card suits

Enumeration of suits:

```
>>> from poker import Suit
>>> list(Suit)
[Suit(''), Suit(''), Suit(''), Suit('')]
```

Suits are comparable:

```
>>> Suit.CLUBS < Suit.DIAMONDS
True
```

1.2.2 Card ranks

Enumeration of ranks:

```
>>> from poker import Rank
>>> list(Rank)
[Rank('2'), Rank('3'), Rank('4'), Rank('5'), Rank('6'), Rank('7'), Rank('8'), Rank('9
↪'), Rank('T'), Rank('J'), Rank('Q'), Rank('K'), Rank('A')]
```

Ranks are comparable:

```
>>> Rank('2') < Rank('A')
True
```

Making a random Rank:

```
>> Rank.make_random()
Rank('2')
```

1.2.3 Cards

Making a random Card:

```
>>> Card.make_random()
Card('As')
```

Comparing Cards:

```
>>> Card('As') > Card('Ks')
True
>>> Card('Tc') < Card('Td')
True
```

1.2.4 Implementing a deck

A deck is just a list of *poker.card.Card*s. Making a new deck and simulating shuffling is easy:

```
import random
from poker import Card

deck = list(Card)
random.shuffle(deck)

flop = [deck.pop() for __ in range(3)]
turn = deck.pop()
river = deck.pop()
```

1.2.5 Operations with Hands and Combos

```
>>> from poker.hand import Hand, Combo
```

List of all hands:

```
>>> list(Hand)
[Hand('32o'), Hand('32s'), Hand('42o'), Hand('42s'), Hand('43o'), Hand('43s'), Hand(
↪'52o'),
 ..., Hand('JJ'), Hand('QQ'), Hand('KK'), Hand('AA')]
```

Comparing:

```
>>> Hand('AAd') > Hand('KK')
True
>>> Combo('7s6s') > Combo('6d5d')
True
```

Sorting:

```
>>> sorted([Hand('22'), Hand('66'), Hand('76o')])
[Hand('76o'), Hand('22'), Hand('66')]
```

Making a random hand:

```
>>> Hand.make_random()
Hand('AJs')
```

1.3 Range parsing

The *poker.hand.Range* class parses human readable (text) ranges like `"22+ 54s 76s 98s AQo+"` to a set of *Hand*s and hand *Combo*s.

Can parse ranges and compose parsed ranges into human readable form again.
It's very fault-tolerant, so it's easy and fast to write ranges manually.
Can normalize unprecise human readable ranges into a precise human readable form, like `"22+ AQo+ 33 AKo"` –> `"22+ AQo+"`
Can tell how big a range is by *Percentage* or number of *Combo*s.

1.3.1 Defining ranges

Atomic signs

X	means "any card"
A K Q J T 9 8 7 6 5 4 3 2	Ace, King, Queen, Jack, Ten, 9, . . . , deuce
"s" or "o" after hands like AKo or 76s	suited and offsuit. Pairs have no suit (`''`)
-	hands worse, down to deuces
+	hands better, up to pairs

Available formats for defining ranges:

Format	Parsed range
22	one pair
44+	all pairs better than 33
66-	all pairs worse than 77
55-33	55, 44, 33

None of these below select pairs (for unambiguity):

AKo, J9o	offsuit hands
AKs, 72s	suited hands
AJo+ Q8o+	offsuit hands above this: AJo, AQo, AKo Q8o, Q9o, QTo, QJo
AJs+	same as offsuit
76s+	this is valid, although "+" is not neccessary, because there are no suited cards above 76s
A5o-	offsuit hands; A5o-A2o
A5s-	suited hands; A5s-A2s
K7	suited and offsuited version of hand; K7o, K7s
J8o-J4o	J8o, J7o, J6o, J5o, J4o
76s-74s	76s, 75s, 74s
J8-J4	both ranges in suited an offsuited form; J8o, J7o, J6o, J5o, J4o, J8s, J7s, J6s, J5s, J4s
A5+	either suited or offsuited hands that contains an Ace and the other is bigger than 5. Same as "A5o+ A5s+".
A5-	downward, same as above
XX	every hand (100% range) In this special case, pairs are also included (but only this)
AX	Any hand that contains an ace either suited or offsuit (no pairs)
AXo	Any offsuit hand that contains an Ace (equivalent to A2o+)
AXs	Any suited hand that contains an Ace (equivalent to A2s+)
QX+	Any hand that contains a card bigger than a Jack; Q2+, K2+, A2+
5X-	any hand that contains a card lower than 6
KXs+	Any suited hand that contains a card bigger than a Queen
KXo+	same as above with offsuit hands
7Xs-	same as above
8Xo-	same as above
2s2h, AsKc	exact hand *Combo*s

Note: "Q+" and "Q-" are invalid ranges, because in Hold'em, there are two hands to start with not one.

Ranges are case insensitive, so `"AKs"` and `"aks"` and `"aKS"` means the same. Also the order of the cards doesn't matter. `"AK"` is the same as `"KA"`. Hands can be separated by space (even multiple), comma, colon or semicolon, and Combo of them (multiple spaces, etc.).

1.3.2 Normalization

Ranges should be rearranged and parsed according to these rules:

- hands separated with one space only in repr, with ", " in str representation
- in any given hand the first card is bigger than second (except pairs of course)
- pairs first, if hyphened, bigger first
- suited hands after pairs, descending by rank
- offsuited hands at the end

1.3.3 Printing the range as an HTML table

Range has a method *to_html()*. When you print the result of that, you get a simple HTML table representation of it.

`Range('XX').to_html()` looks like this:

You can format it with CSS, you only need to define `td.pair`, `td.offsuit` and `td.suited` selectors. It's easy to recreate PokerStove style colors:

```
<style>
   td {
      /* Make cells same width and height and centered */
      width: 30px;
      height: 30px;
      text-align: center;
      vertical-align: middle;
   }
   td.pair {
      background: #aaff9f;
   }
   td.offsuit {
      background: #bbced3;
   }
   td.suited {
      background: #e37f7d;
   }
</style>
```

1.3.4 Printing the range as an ASCII table

to_ascii() can print a nicely formatted ASCII table to the terminal:

```
>>> print(Range('22+ A2+ KT+ QJ+ 32 42 52 62 72').to_ascii())
AA  AKs AQs AJs ATs A9s A8s A7s A6s A5s A4s A3s A2s
AKo KK  KQs KJs KTs
AQo KQo QQ  QJs
AJo KJo QJo JJ
ATo KTo         TT
A9o                 99
A8o                     88
A7o                         77                  72s
A6o                             66              62s
A5o                                 55          52s
A4o                                     44      42s
A3o                                         33  32s
A2o                         72o 62o 52o 42o 32o 22
```

1.4 Hand history parsing

The classes in `poker.room` can parse hand histories for different poker rooms. Right now for PokerStars, Full Tilt Poker and PKR, very efficiently with a simple API.

1.4.1 Parsing from hand history text

In one step:

```
from poker.room.pokerstars import PokerStarsHandHistory
# First step, only raw hand history is saved, no parsing will happen yet
hh = PokerStarsHandHistory(hand_text)
# You need to explicitly parse. This will parse the whole hh at once.
hh.parse()
```

Or in two steps:

```
from poker.room.pokerstars import PokerStarsHandHistory
hh = PokerStarsHandHistory(hand_text)
# parse the basic information only (really fast)
hh.parse_header()
# And later parse the body part. This might happen e.g. in a background task
>>> hh.parse()
```

I decided to implement this way, and not parse right away at object instantiation, because probably the most common operation will be looking into the hand history as fast as possible for basic information like hand id, *or* deferring the parsing e.g. to a message queue. This way, you basically just save the raw hand history in the instance, pass it to the queue and it will take care of parsing by the parse() call.

And also because "Explicit is better than implicit."

1.4.2 Parsing from file

```
>>> hh = PokerStarsHandHistory.from_file(filename)
>>> hh.parse()
```

1.4.3 Example

```
>>> from poker.room.pokerstars import PokerStarsHandHistory
>>> hh = PokerStarsHandHistory(hand_text)
>>> hh.parse()
>>> hh.players
[_Player(name='flettl2', stack=1500, seat=1, combo=None),
 _Player(name='santy312', stack=3000, seat=2, combo=None),
 _Player(name='flavio766', stack=3000, seat=3, combo=None),
 _Player(name='strongi82', stack=3000, seat=4, combo=None),
 _Player(name='W2lkm2n', stack=3000, seat=5, combo=Combo('AJ')),
 _Player(name='MISTRPerfect', stack=3000, seat=6, combo=None),
 _Player(name='blak_douglas', stack=3000, seat=7, combo=None),
 _Player(name='sinus91', stack=1500, seat=8, combo=None),
 _Player(name='STBIJUJA', stack=1500, seat=9, combo=None)]
>>> hh.date
datetime.datetime(2013, 10, 4, 19, 18, 18, tzinfo=<DstTzInfo 'US/Eastern' EDT-1 day,␣
↪20:00:00 DST>)
>>> hh.hero
_Player(name='W2lkm2n', stack=3000, seat=5, combo=Combo('AJ')),
>>> hh.limit, hh.game
('NL', 'HOLDEM')
>>> hh.board
```

(continues on next page)

(continued from previous page)

```
(Card('2'), Card('6'), Card('6'))
>>> hh.flop.is_rainbow
True
>>> hh.flop.has_pair
True
>>> hh.flop.actions
(('W2lkm2n', <Action.BET: ('bet', 'bets')>, Decimal('80')),
 ('MISTRPerfect', <Action.FOLD: ('fold', 'folded', 'folds')>),
 ('W2lkm2n', <Action.RETURN: ('return', 'returned', 'uncalled')>, Decimal('80')),
 ('W2lkm2n', <Action.WIN: ('win', 'won', 'collected')>, Decimal('150')),
 ('W2lkm2n', <Action.MUCK: ("don't show", "didn't show", 'did not show', 'mucks')>))
```

1.4.4 API

See all the room specific classes in them *Hand history parsing API* documentation.

1.4.5 About hand history changes

Poker rooms sometimes change the hand history format significally. My goal is to cover all hand histories after 2014.01.01., because it is the best compromise between fast development and good coverage. This way we don't have to deal with ancient hand history files and overcomplicate the code and we can concentrate on the future instead of the past. Also, hopefully hand history formats are stable enough nowadays to follow this plan, less and less new game types coming up.

One of the "recent" changes made by Full Tilt is from 2013.05.10.:

> "In the software update from Wednesday, changed the format of the . This means that Hold'em Manager does no longer import these hands, and the HUD is not working. … B.t.w. They just renamed "No Limit Hold'em" to "NL Hold'em", and swapped position with the blinds, inside the handhistory files."

Details: http://www.bankrollmob.com/forum.asp?mode=thread&id=307215

1.5 Getting information from poker related websites

1.5.1 PokerStars status

You can get information about PokerStars online players, active tournaments, number of tables currently running:

```
>>> from poker.website.pokerstars import get_status
>>> status = get_status()
>>> status.players, status.tables
(110430, 16427)
```

See the possible attributes in the *API documentation*.

1.5.2 List of upcoming tournaments from PokerStars

```
>>> from poker.website.pokerstars import get_current_tournaments
# get_current_tournaments is a generator, so if you want a list, you need to cast it
# otherwise, you can iterate over it
>>> list(get_current_tournaments())
[_Tournament(start_date=datetime.datetime(2014, 8, 16, 8, 2, tzinfo=tzoffset(None, -
↪14400)), name="Copernicus' FL Omaha H/L Freeroll", game='Omaha', buyin='$0 + $0',␣
↪players=2509),
_Tournament(start_date=datetime.datetime(2014, 8, 16, 8, 2, tzinfo=tzoffset(None, -
↪14400)), name='500 Cap: $0.55 NLHE', game="Hold'em", buyin='$0.50 + $0.05',␣
↪players=80),
_Tournament(start_date=datetime.datetime(2014, 8, 16, 8, 2, tzinfo=tzoffset(None, -
↪14400)), name='Sunday Million Sat [Rd 1]: $0.55+R NLHE [2x-Turbo], 3 Seats Gtd',␣
↪game="Hold'em", buyin='$0.50 + $0.05', players=14),
_Tournament(start_date=datetime.datetime(2014, 8, 16, 8, 2, tzinfo=tzoffset(None, -
↪14400)), name='$11 NLHE [Phase 1] Sat: 5+R FPP NLHE [2x-Turbo], 2 Seats Gtd', game=
↪"Hold'em", buyin='$0 + $0', players=45),
...
]
```

Or you can iterate over it and use specific data:

```
>>> from poker.website.pokerstars import get_current_tournaments
>>> for tournament in get_current_tournaments():
...     print(tournament.name)
Play Money, No Limit Hold'em + Knockout (5,000)
Sunday Million Sat [Rd 1]: $2.20 NLHE [Turbo]
Play Money, No Limit Omaha (100k)
Play Money, Hourly 1K, NLHE
$11 NL Hold'em [Time: 15 Minutes]
$2.20 NL Hold'em [Heads-Up,128 Cap, Winner-Take-All]
$2.20 NL Hold'em [4-Max, Turbo,5x-Shootout]
$11+R NL Hold'em [Action Hour], $5K Gtd
...
```

1.5.3 Information about a Two plus two forum member

If you want to download all the available public information about a forum member (e.g. http://forumserver.twoplustwo.com/members/115014/) all you need to do is:

```
>>> from poker.website.twoplustwo import ForumMember
>>> forum_member = ForumMember('Walkman_')
>>> vars(forum_member)
{'public_usergroups': ('Marketplace Approved',),
 'username': 'Walkman_ ',
 'location': 'Hungary',
 'download_date': datetime.datetime(2014, 8, 29, 16, 30, 45, 64197, tzinfo=datetime.
↪timezone.utc),
 'rank': 'enthusiast',
 'total_posts': 92,
 'id': '115014',
 'join_date': datetime.date(2008, 3, 10),
 'posts_per_day': 0.04,
 'profile_picture': 'http://forumserver.twoplustwo.com/customprofilepics/
↪profilepic115014_1.gif',
 'avatar': 'http://forumserver.twoplustwo.com/customavatars/thumbs/avatar115014_1.gif
↪',
```

(continues on next page)

(continued from previous page)

```
 'last_activity': datetime.datetime(2014, 8, 26, 2, 49, tzinfo=<UTC>)}
```

1.5.4 Getting the top 100 players from Pocketfives

```
>>> from poker.website.pocketfives import get_ranked_players
>>> list(get_ranked_players())
[_Player(name='pleno1', country='United Kingdom', triple_crowns=1, monthly_win=0,␣
↪biggest_cash='$110,874.68', plb_score=7740.52, biggest_score=817.0, average_
↪score=42.93, previous_rank='2nd'),
_Player(name='p0cket00', country='Canada', triple_crowns=6, monthly_win=0, biggest_
↪cash='$213,000.00', plb_score=7705.61, biggest_score=1000.0, average_score=47.23,␣
↪previous_rank='1st'),
_Player(name='r4ndomr4gs', country='Sweden', triple_crowns=2, monthly_win=1, biggest_
↪cash='$174,150.00', plb_score=7583.38, biggest_score=803.0, average_score=46.59,␣
↪previous_rank='3rd'),
_Player(name='huiiiiiiiiii', country='Austria', triple_crowns=1, monthly_win=0,␣
↪biggest_cash='$126,096.00', plb_score=7276.52, biggest_score=676.0, average_
↪score=39.26, previous_rank='11th'),
_Player(name='TheClaimeer', country='United Kingdom', triple_crowns=1, monthly_win=0,␣
↪biggest_cash='$102,296.00', plb_score=6909.56, biggest_score=505.0, average_
↪score=41.68, previous_rank='4th'),
```

poker.website.pocketfives._Player is a named tuple, so you can look up attributes on it:

```
>>> for player in get_ranked_players():
...     print(player.name, player.country)
pleno1 United Kingdom
p0cket00 Canada
r4ndomr4gs Sweden
huiiiiiiiiii Austria
TheClaimeer United Kingdom
Romeopro Ukraine
PokerKaiser Chile
dipthrong Canad
...
```

1.6 Room specific operations

1.6.1 Manipulating PokerStars player notes

poker.room.pokerstars.Notes class is capable of handling PokerStars Players notes.

You can add and delete labels, and notes, save the modifications to a new file or just print the object instance and get the full XML.

```
>>> from poker.room.pokerstars import Notes
>>> notes = Notes.from_file('notes.W2lkm2n.xml')
>>> notes.players
('regplayer', 'sharkplayer', 'fishplayer', '"htmlchar"', '$dollarsign',
↪'nonoteforplayer',
 '-=strangename=-', '//ÄMGS', '0bullmarket0', 'CarlGardner', 'µ (x+t)', 'Walkman')
```

(continues on next page)

(continued from previous page)

```
>>> notes.labels
(_Label(id='0', color='30DBFF', name='FISH'),
 _Label(id='1', color='30FF97', name='SHARK'),
 _Label(id='2', color='E1FF80', name='REG'),
 _Label(id='3', color='E1FF80', name='GENERAL'))

>>> notes.add_label('NIT', 'FF0000')
>>> notes.labels
(_Label(id='0', color='30DBFF', name='FISH'),
 _Label(id='1', color='30FF97', name='SHARK'),
 _Label(id='2', color='E1FF80', name='REG'),
 _Label(id='3', color='E1FF80', name='GENERAL'))
 _Label(id='4', color='FF0000', name='NIT'))
```

For the full API, see the *Room specific classes API*.

1.7 Development

1.7.1 Git repository

You find the repository on github: https://github.com/pokerregion/poker

In the `dend/` branches, there are ideas which doesn't work or has been abandoned for some reason. They are there for reference as "this has been tried".

I develop in a very simple Workflow. (Before 662f5d73be1efbf6eaf173da448e0410da431b2c you can see bigger merge bubbles, because I handled hand history parser code and the rest as two separate projects, but made a subtree merge and handle them in this package.) Feature branches with rebases on top of master. Only merge stable code into master.

The repository tags will match PyPi release numbers.

1.7.2 Versioning

I use Semantic Versioning, except for major versions like 1.0, 2.0, because I think 1.0.0 looks stupid :)

1.7.3 Coding style

PEP8 except for line length, which is 99 max (hard limit). If your code exceeds 99 characters, you do something wrong anyway, you need to refactor it (e.g. to deeply nested, harder to understand)

1.7.4 Dates and times

Every datetime throughout the library is in UTC with tzinfo set to `pytz.UTC`. If you found a case where it's not, it's a bug, please report it on GitHub! The right way for setting a date correctly e.g. from PokerStars ET time is:

```
>>> import pytz
>>> ET = pytz.timezone('US/Eastern')
>>> ET.localize(some_datetime).astimezone(pytz.UTC)
```

This will consider DST settings and ambiguous times. For more information, see pytz documentation!

1.7.5 New hand history parser

Note: Hand history parsing API will change for sure until 1.0 is done.

If you want to support a new poker room you have to subclass the appropriate class from `poker.handhistory` like `poker.handhistory._SplittableHandHistory` depending on the type of hand history file, like XML, or similar to pokerstars and FTP, define a couple of methods and done.

```
class NewPokerRoomHandHistory(HandHistory):
"""Implement PokerRoom specific parsing."""

   def parse_header(self):
      # Parse header only! Usually just the first line. The whole purpose is to do it␣
↪fast!
      # No need to call super()

   def _parse_table(self):
      # parses table name

   def _parse_players(self):
      # parses players, player positions, stacks, etc
      # set self.players attribute

   def _parse_button(self):

   def _parse_hero(self):

   def _parse_preflop(self):

   def _parse_street(self):

   def _parse_showdown(self):

   def _parse_pot(self):

   def _parse_board(self):

   def _parse_winners(self):

   def _parse_extra(self):
```

You **have to** provide all common attributes, and *may* provide PokerRoom specific extra attributes described in the base `poker.handhistory.HandHistory` class API documentation.

1.7.6 Testing

The framework contains a lot of tests (over 400). The basic elements like Card, Hand, Range, etc. are fully tested.

All the unit tests are written in pytest. I choose it because it offers very nice functionality, and no-boilerplate code for tests. No need to subclass anything, just prefix classes with `Test` and methods with `test_`.

All assertion use the default python `assert` keyword.

You need to install the `poker` package in development mode:

```
# from directory where setup.py file is
$ pip install -e .
```

and install pytest and run it directly:

```
$ pip install pytest
$ py.test
```

from the poker module directory and pytest will automatically pick up all unit tests.

1.8 Glossary

Suit One of 'c', 'd', 'h', or 's'. Alternatively '♣', '♦', '♥', '♠'. According to Wikipedia, suits are ranked as:

spades > hearts > diamonds > clubs

Shape A hand can have three "Shapes" according to Wikipedia.

'o' for offsuit, 's' for suited hands '' for pairs.

Rank One card without suit. One of '2', '3', '4', '5', '6', '7', '8', '9', 'T', 'J', 'Q', 'K', 'A'.

Card One exact card with a suit. e.g. 'As', '2s'. It has a *Rank* and a *Suit*.

Hand Consists two *Rank*s without precise suits like "AKo", "22".

Hand comparisons Comparisons in this library has nothing to do with equities or if a hand beats another. They are only defined so that a consistent ordering can be ensured when representing objects. If you want to compare hands by equity, use pypoker-eval instead.

Comparison rules:

- pairs are 'better' than none-pairs
- non-pairs are better if at least one of the cards are bigger
- suited better than offsuit

Combo Exact two cards with suits specified like "2s2c", "7s6c". There are total of 1326 Combos.

Range A range of hands with either in *Hand* form or *Combo*. e.g. "55+ AJo+ 7c6h 8s6s", "66-33 76o-73o AsJc 2s2h" or with other speical notation. (See above.)

Range percent Compared to the total of 1326 hand *Combo*s, how many are in the range?

Range length

Range size How many concrete hand *Combo*s are in the range?

Range is "bigger" than another If there are more hand *Combo*s in it. (Equity vs each other doesn't matter here.)

Token Denote one part of a range. In a "66-33 76o-73o AsJc 2s2h" range, there are 4 tokens: - "66-33" meaning 33, 44, 55, 66 - "AsJc" specific *Combo* - "2s2h" a specific pair of deuces - "76o-73o" several offsuit *Hand*s

Broadway card T, J, Q, K, A

Face card Only: J, Q, K.

> **Warning:** Ace is not a face card!

1.9 License

CHAPTER 2

API

2.1 Card API

The `poker.card` module has three basic classes for dealing with card suits, card ranks and cards. It also has a DECK, which is just a tuple of Cards.

2.1.1 Suit

class poker.card.**Suit**
An enumeration.

Enumeration of the four *Suit*s.

2.1.2 Rank

class poker.card.**Rank**
An enumeration.

Enumeration of the 13 *Rank*s.

classmethod difference(*first*, *second*)
Tells the numerical difference between two ranks.

Parameters

- **first** (*str*, Rank) –
- **second** (*str*, Rank) –

Returns value of the difference (always positive)

Return type int

poker.card.**FACE_RANKS**
See *Face card*

poker.card.**BROADWAY_RANKS**
See *Broadway card*

2.1.3 Card

class poker.card.**Card**
Represents a Card, which consists a Rank and a Suit.

classmethod make_random()
Returns a random Card instance.

Return type *Card*

is_face

Type bool

is_broadway

Type bool

rank

Type *Rank*

suit

Type *Suit*

2.2 Hand API

2.2.1 Shape

class poker.hand.**Shape**
An enumeration.

See: *Shape*

Warning: This might be removed in future version for simplify API.

2.2.2 Hand

class poker.hand.**Hand**(*hand*)
General hand without a precise suit. Only knows about two ranks and shape.

Parameters **hand** (*str*) – e.g. 'AKo', '22'

Variables

- ***first*** (Rank) – first Rank
- ***second*** (Rank) – second Rank
- ***shape*** (Shape) – Hand shape (pair, suited or offsuit)

rank_difference
The difference between the first and second rank of the Hand.

Type int

first

Type *poker.card.Rank*

second

Type *poker.card.Rank*

shape

Type *Shape*

is_broadway

is_connector

is_offsuit

is_one_gapper

is_pair

is_suited

is_suited_connector

is_two_gapper

to_combos()

poker.hand.**PAIR_HANDS = (Hand('22'), Hand('33'), Hand('44'), Hand('55'), Hand('66'), Hand(**
Tuple of all pair hands in ascending order.

poker.hand.**OFFSUIT_HANDS = (Hand('32o'), Hand('42o'), Hand('43o'), Hand('52o'), Hand('53o')**
Tuple of offsuit hands in ascending order.

poker.hand.**SUITED_HANDS = (Hand('32s'), Hand('42s'), Hand('43s'), Hand('52s'), Hand('53s'),**
Tuple of suited hands in ascending order.

2.2.3 Combo

class poker.hand.**Combo**
Hand combination.

See *Combo*

first

Type *poker.card.Card*

second

Type *poker.card.Card*

shape

Type *Shape*

classmethod from_cards(*first*, *second*)

is_broadway

is_connector

is_offsuit

is_one_gapper

is_pair

is_suited

is_suited_connector

is_two_gapper

rank_difference
The difference between the first and second rank of the Combo.

to_hand()
Convert combo to *Hand* object, losing suit information.

2.2.4 Range

class poker.hand.**Range**(*range=''*)
Parses a str range into tuple of Combos (or Hands).

Parameters **range** (*str*) – Readable range in unicode

Note: All of the properties below are cached_property, so make sure you invalidate the cache if you manipulate them!

---

hands
Tuple of hands contained in this range. If only one combo of the same hand is present, it will be shown here. e.g. `Range('2s2c').hands == (Hand('22'),)`

Type tuple of *poker.hand.Hand*s

combos

Type tuple of *poker.hand.Combo*s

percent
What percent of combos does this range have compared to all the possible combos.

There are 1326 total combos in Hold'em: 52 * 51 / 2 (because order doesn't matter) Precision: 2 decimal point

Type float (1-100)

rep_pieces
List of str pieces how the Range is represented.

Type list of str

to_html()
Returns a 13x13 HTML table representing the range.

The table's CSS class is `range`, pair cells (td element) are `pair`, offsuit hands are `offsuit` and suited hand cells has `suited` css class. The HTML contains no extra whitespace at all. Calculating it should not take more than 30ms (which takes calculating a 100% range).

Return type str

to_ascii(*border=False*)
Returns a nicely formatted ASCII table with optional borders.

Return type str

classmethod from_file(*filename*)
Creates an instance from a given file, containing a range. It can handle the PokerCruncher (.rng extension) format.

classmethod from_objects(*iterable*)
Make an instance from an iterable of Combos, Hands or both.

slots = ('_hands', '_combos')

2.3 Hand history parsing API

Note: Hand history parsing API will change for sure until 1.0 is done.

2.3.1 Constant values

These enumerations are used to identify common values like limit types, game, etc. By unifying these into groups of enumeration classes, it's possible to have common values accross the whole framework, even when parsing totally different kind of hand histories, which uses different values. (Data normalization) It's recommended to use keys (name property) to save in database, and print them to the user. (E.g. in a web application template, `{{ PokerRoom.STARS }}` will be converted to `'PokerStars'`.)

class poker.constants.**Action**
An enumeration.

BET = ('bet', 'bets')

CALL = ('call', 'calls')

CHECK = ('check', 'checks')

FOLD = ('fold', 'folded', 'folds')

MUCK = ("don't show", "didn't show", 'did not show', 'mucks')

RAISE = ('raise', 'raises')

RETURN = ('return', 'returned', 'uncalled')

SHOW = ('show',)

THINK = ('seconds left to act',)

WIN = ('win', 'won', 'collected')

class poker.constants.**Currency**
An enumeration.

EUR = ('EUR', '€')

GBP = ('GBP', '£')

STARS_COIN = ('SC', 'StarsCoin')

USD = ('USD', '$')

class poker.constants.**Game**
An enumeration.

HOLDEM = ("Hold'em", 'HOLDEM')

OHILO = ('Omaha Hi/Lo',)

OMAHA = ('Omaha',)

RAZZ = ('Razz',)

STUD = ('Stud',)

class poker.constants.**GameType**
An enumeration.

CASH = ('Cash game', 'CASH', 'RING')

SNG = ('Sit & Go', 'SNG', 'SIT AND GO', 'Sit&go')

TOUR = ('Tournament', 'TOUR')

class poker.constants.**Limit**
An enumeration.

FL = ('FL', 'Fixed limit', 'Limit')

NL = ('NL', 'No limit')

PL = ('PL', 'Pot limit')

class poker.constants.**MoneyType**
An enumeration.

PLAY = ('Play money',)

REAL = ('Real money',)

class poker.constants.**PokerRoom**
An enumeration.

EIGHT = ('888', '888poker')

FTP = ('Full Tilt Poker', 'FTP', 'FULL TILT')

PKR = ('PKR', 'PKR POKER')

STARS = ('PokerStars', 'STARS', 'PS')

class poker.constants.**Position**
An enumeration.

BB = ('BB', 'big blind')

BTN = ('BTN', 'bu', 'button')

CO = ('CO', 'cutoff', 'cut off')

HJ = ('HJ', 'hijack', 'utg+5', 'utg + 5')

SB = ('SB', 'small blind')

UTG = ('UTG', 'under the gun')

UTG1 = ('UTG1', 'utg+1', 'utg + 1')

UTG2 = ('UTG2', 'utg+2', 'utg + 2')

UTG3 = ('UTG3', 'utg+3', 'utg + 3')

UTG4 = ('UTG4', 'utg+4', 'utg + 4')

class poker.constants.**TourFormat**
An enumeration.

ACTION = ('Action Hour',)

ONEREB = ('1R1A',)

REBUY = ('Rebuy', '+R')

SECOND = ('2x Chance',)

class poker.constants.**TourSpeed**
An enumeration.

DOUBLE = ('2x-Turbo',)

HYPER = ('Hyper-Turbo',)

REGULAR = ('Regular',)

SLOW = ('Slow',)

TURBO = ('Turbo',)

2.3.2 Base classes

class poker.handhistory.**_BaseHandHistory**(*hand_text*)
Abstract base class for *all* kinds of parser.

Parameters **hand_text** (*str*) – poker hand text

The attributes can be iterated.
The class can read like a dictionary.
Every attribute default value is `None`.

Variables

- **date_format** (*str*) – default date format for the given poker room
- **ident** (*str*) – hand id
- **game_type** (poker.constants.GameType) – `"TOUR"` for tournaments or `"SNG"` for Sit&Go-s
- **tournament_ident** (*str*) – tournament id
- **tournament_level** (*str*) – level of tournament blinds
- **currency** (poker.constants.Currency) – 3 letter iso code `"USD"`, `"HUF"`, `"EUR"`, etc.
- **buyin** (*decimal.Decimal*) – buyin **without** rake
- **rake** (*decimal.Decimal*) – if game_type is `"TOUR"` it's buyin rake, if `"CASH"` it's rake from pot
- **game** (poker.constants.Game) – `"HOLDEM"`, `"OMAHA"`, `"STUD"`, `"RAZZ"`, etc.
- **limit** (poker.constants.Limit) – `"NL"`, `"PL"` or `"FL"`
- **sb** (*decimal.Decimal*) – amount of small blind
- **bb** (*decimal.Decimal*) – amount of big blind
- **date** (*datetime*) – hand date in UTC

- **table_name** (*str*) – name of the table. it's "tournament_number table_number"
- **max_players** (*int*) – maximum players can sit on the table, 2, 4, 6, 7, 8, 9
- **button** (poker.handhistory._Player) – player on the button
- **hero** (poker.handhistory._Player) – hero player
- ***players*** (*list*) – list of *poker.handhistory._Player*. the sequence is the seating order at the table at the start of the hand
- **flop** (_Flop) – room specific Flop object
- **turn** (poker.card.Card) – turn card, e.g. Card('Ah')
- **river** (poker.card.Card) – river card, e.g. Card('2d')
- **board** (*tuple*) – board cards, e.g. (Card('4s'), Card('4d'), Card('4c'), Card('5h'))
- **preflop_actions** (*tuple*) – action lines in str
- **turn_actions** (*tuple*) – turn action lines
- **turn_pot** (*decimal.Decimal*) – pot size before turn
- **turn_num_players** (*int*) – number of players seen the turn
- **river_actions** (*tuple*) – river action lines
- **river_pot** (*decimal.Decimal*) – pot size before river
- **river_num_players** (*int*) – number of players seen the river
- **tournament_name** (*str*) – e.g. "$750 Guarantee", "$5 Sit & Go (Super Turbo)"
- **total_pot** (*decimal.Decimal*) – total pot after end of actions (rake included)
- **show_down** (*bool*) – There was show_down or wasn't
- **winners** (*tuple*) – winner names, tuple if even when there is only one winner. e.g. ('W2lkm2n',)
- **extra** (*dict*) – Contains information which are specific to a concrete hand history and not common accross all. When iterating through the instance, this extra attribute will not be included. default value is None

class poker.handhistory.**_Player**(*name*, *stack*, *seat*, *combo*)

Player participating in the hand history.

Variables

- **name** (*str*) – Player name
- **stack** (*int*) – Stack size (sometimes called as chips)
- **seat** (*int*) – Seat number
- **combo** (Combo, *None*) – If the player revealed his/her hand, this property hold's it. None for players didn't show... autoclass:: poker.handhistory._Player

Every hand history has an attribute `flop` which is an instance of the room specific *_Flop* object which has the following attributes:

class _Flop

Variables

- **cards** (*tuple*) – tuple of *poker.card.Card*s
- **pot** (*decimal.Decimal*) – pot size after actions
- ***players*** (*tuple*) – tuple of player names
- **actions** (*tuple*) –

  tuple of *poker.constants.Action* in the order of happening.
  Form:
  (Player name, Action, Amount) or
  (Player name, Action) if no amount needed (e.g. in case of Check)

It also has properties about flop texture like:

Variables

- **is_rainbow** (*bool*) –
- **is_monotone** (*bool*) –
- **is_triplet** (*bool*) –
- **has_pair** (*bool*) –
- **has_straightdraw** (*bool*) –
- **has_gutshot** (*bool*) –
- **has_flushdraw** (*bool*) –

2.3.3 PokerStars

class poker.room.pokerstars.**PokerStarsHandHistory**(*hand_text*)
Parses PokerStars Tournament hands.

2.3.4 Full Tilt Poker

class poker.room.fulltiltpoker.**FullTiltPokerHandHistory**(*hand_text*)
Parses Full Tilt Poker hands the same way as PokerStarsHandHistory class.

PokerStars and Full Tilt hand histories are very similar, so parsing them is almost identical. There are small differences though.

Class specific

Variables

- **tournament_level** – None
- **buyin** – None: it's not in the hand history, but the filename
- **rake** – None: also
- **currency** – None
- **table_name** (*str*) – just a number, but str type

Extra

Variables

- **turn_pot** (*Decimal*) – pot size before turn
- **turn_num_players** (*int*) – number of players seen the turn
- **river_pot** (*Decimal*) – pot size before river
- **river_num_players** (*int*) – number of players seen the river
- **tournament_name** (*str*) – e.g. `"$750 Guarantee"`, `"$5 Sit & Go (Super Turbo)"`

2.3.5 PKR

class poker.room.pkr.**PKRHandHistory**(*hand_text*)

Parses PKR hand histories.

Class specific

Variables **table_name** (*str*) – "#table_number - name_of_the_table"

Extra

Variables

- **last_ident** (*str*) – last hand id
- **money_type** (*str*) – `"R"` for real money, `"P"` for play money

2.4 Room specific classes API

2.4.1 Pokerstars player notes

class poker.room.pokerstars.**Notes**(*notes: str*)

Class for parsing pokerstars XML notes.

get_note(*player*)

Return *_Note* tuple for the player.

Raises ***poker.room.pokerstars.NoteNotFoundError*** –

add_note(*player*, *text*, *label=None*, *update=None*)

Add a note to the xml. If update param is None, it will be the current time.

Raises ***poker.room.pokerstars.LabelNotFoundError*** – if there is no such label name

del_note(*player*)

Delete a note by player name.

Raises ***poker.room.pokerstars.NoteNotFoundError*** –

add_label(*name*, *color*)

Add a new label. It's id will automatically be calculated.

append_note(*player*, *text*)

Append text to an already existing note.

del_label(*name*)

Delete a label by name.

classmethod from_file(*filename*)
Make an instance from a XML file.

get_label(*name*)
Find the label by name.

get_note_text(*player*)
Return note text for the player.

label_names
Tuple of label names.

labels
Tuple of labels.

notes
Tuple of notes..

players
Tuple of player names.

prepend_note(*player*, *text*)
Prepend text to an already existing note.

replace_note(*player*, *text*)
Replace note text with text. (Overwrites previous note!)

save(*filename*)
Save the note XML to a file.

class poker.room.pokerstars.**_Label**(*id*, *color*, *name*)
Labels in Player notes.

Variables

- **id** (*str*) – numeric id for the label. None when no label ('-1' in XML)
- **color** (*str*) – color code for note
- **name** (*str*) – name of the note

class poker.room.pokerstars.**_Note**(*player*, *label*, *update*, *text*)
Player note.

Variables

- **player** (*str*) – player name
- **label** (*str*) – Label name of note
- **update** (*datetime.datetime*) – when was the note last updated
- **text** (*str*) – Note text

exception poker.room.pokerstars.**NoteNotFoundError**
Note not found for player.

exception poker.room.pokerstars.**LabelNotFoundError**
Label not found in the player notes.

2.5 Website API

This package contains mostly scraping tools for well known websites like Two Plus Two forum, Pocketsfives, etc...

2.5.1 Two Plus Two Forum API

poker.website.twoplustwo.**FORUM_URL**
http://forumserver.twoplustwo.com

poker.website.twoplustwo.**FORUM_MEMBER_URL**
http://forumserver.twoplustwo.com/members

class poker.website.twoplustwo.**ForumMember**(*username*)
Download and store a member data from the Two Plus Two forum.

Parameters **id** (*int*, *str*) –

Forum id (last part of members URL, e.g. in case of
http://forumserver.twoplustwo.com/members/407153/
the id is 407153)

Variables

- **id** (*str*) – Forum id
- **username** (*str*) – Forum username
- ***rank*** (*str*) – Forum rank like `'enthusiast'`
- **profile_picture** (*str*, *None*) – URL of profile if set.
- **location** (*str*) – Location (country)
- **total_posts** (*int*) – Total posts
- **posts_per_day** (*float*) – Posts per day on account page
- **last_activity** (*datetime*) – Last activity with the website timezone
- **join_date** (*date*) – Join date on account page
- **public_usergroups** (*tuple*) – Public usergroup permission as in the box on the top right
- **donwload_date** (*datetime*) – When were the data downloaded from TwoplusTwo

2.5.2 Pocketfives API

class poker.website.pocketfives.**_Player**(*name*, *country*, *triple_crowns*, *monthly_win*, *biggest_cash*, *plb_score*, *biggest_score*, *average_score*, *previous_rank*)
Pocketfives player data.

Variables

- **name** (*str*) – Player name
- **country** (*str*) – Country name
- **triple_crowns** (*int*) – Number of triple crowns won
- **monthly_win** (*int*) –
- **biggest_cash** (*str*) –
- **plb_score** (*float*) –
- **biggest_score** (*float*) – Biggest Pocketfives score

- **average_score** (*float*) – Average pocketfives score
- **previous_rank** (*str*) – Previous pocketfives rank

poker.website.pocketfives.**get_ranked_players**()
Get the list of the first 100 ranked players.

Returns generator of *_Player*s

Note: Downloading this list is a slow operation!

2.5.3 PokerStars website API

poker.website.pokerstars.**WEBSITE_URL**
http://www.pokerstars.eu

poker.website.pokerstars.**TOURNAMENTS_XML_URL**
http://www.pokerstars.eu/datafeed_global/tournaments/all.xml

poker.website.pokerstars.**STATUS_URL**
http://www.psimg.com/datafeed/dyn_banners/summary.json.js

poker.website.pokerstars.**get_current_tournaments**()
Get the next 200 tournaments from pokerstars.

Returns generator of *_Tournament*

Note: Downloading this list is an extremly slow operation!

poker.website.pokerstars.**get_status**()
Get pokerstars status: players online, number of tables, etc.

Returns *_Status*

class poker.website.pokerstars.**_Tournament**(*start_date*, *name*, *game*, *buyin*, *players*)
Upcoming pokerstars tournament.

Variables

- **start_date** (*datetime*) –
- **name** (*str*) – Tournament name as seen in PokerStars Lobby
- **game** (*str*) – Game Type
- **buyin** (*str*) – Buy in with fee
- ***players*** (*int*) – Number of players already registered

class poker.website.pokerstars.**_Status**(*updated*, *tables*, *next_update*, *players*, *clubs*, *active_tournaments*, *total_tournaments*, *sites*, *club_members*)
PokerStars status.

Variables

- **updated** (*datetime*) – Status last updated
- **tables** (*int*) – Number of tournament tables
- ***players*** (*int*) – Number of players logged in to PokerStars

- **clubs** (*int*) – Total number of Home Game clubs created
- **club_members** (*int*) – Total number of Home Game club members
- **active_tournaments** (*int*) –
- **total_tournaments** (*int*) –
- **sites** (*tuple*) – Tuple of *_SiteStatus*

class poker.website.pokerstars.**_SiteStatus**(*id*, *tables*, *players*, *active_tournaments*)

PokerStars status on different subsites like FR, ES IT or Play Money.

Variables

- **id** (*str*) – ID of the site (".IT", ".FR", "Play Money")
- **tables** (*int*) –
- **player** (*int*) –
- **active_tournaments** (*int*) –

CHAPTER 3

Repo and contact

Repo: https://github.com/pokerregion/poker
Issues: https://github.com/pokerregion/poker/issues
@kissgyorgy on twitter
or you can reach me on my public Github e-mail.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

W