My Project

Generated by Doxygen 1.8.9.1

Sun Oct 18 2015 18:04:46

Contents

Chapter 1

Namespace Index

1.1 Packages

Here are the packages with brief descriptions (if available):

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 python Namespace Reference

Namespaces

- cards
- combinatorics
- primes

5.2 python.cards Namespace Reference

Classes

- class Hand

Functions

- def cards_cmp (a, b)
- def denom (card)

5.2.1 Function Documentation

5.2.1.1 def python.cards.cards_cmp (*a, b*)

```
Sorting helper, returns zero if a == b, a positive integer if
a > b and a negative integer otherwise.
```

5.2.1.2 def python.cards.denom (*card*)

```
The denominaton of the given card.  The denomination is defined as
follows:
    2 through 10 - the cards value
    J            - 11
    Q            - 12
    K            - 13
    A            - 14
```

5.3 python.combinatorics Namespace Reference

Functions

- def fib (n)
- def digits (n)
- def nub (nums)
- def has_square (n)

5.3.1 Function Documentation

5.3.1.1 def python.combinatorics.digits (*n*)

```
The digits of the argument, not counting the sign.
```

5.3.1.2 def python.combinatorics.fib (*n*)

```
It is possible to compute N'th Fibonacci number analytically,
but I was asked not to do this, and to keep it simple. So, here
you have the iterative version.
```

5.3.1.3 def python.combinatorics.has_square (*n*)

```
Returns True iff n contains a perfect square as a factor.
```

5.3.1.4 def python.combinatorics.nub (*nums*)

```
Returns a copy of nums with duplicates removed.  The numbers
in nums will be solrted in increasing order.
```

5.4 python.primes Namespace Reference

Functions

- def primes_seq (n)
- def is_next_prime (n, sieve)
- def is_prime (n, sieve)
- def next_prime (sieve)
- def primes
- def odd_composite ()
- def factors (n)

5.4.1 Function Documentation

5.4.1.1 def python.primes.factors (*n*)

```
Generates all prime factors of n.
```

5.4.1.2 def python.primes.is_next_prime (*n, sieve*)

```
Tests whether n is prime.  Sieve should contain all primes less than n.
```

5.4.1.3 def python.primes.is_prime (*n, sieve*)

```
Returns True if the number is a prime.  Sieve should contain some prime
numbers to start the testing (the more the better).
```

5.4.1.4 def python.primes.next_prime (*sieve*)

```
Given the sieve containing primes searches for the prime greater than the
last element of the sieve, such that there are no primes less than this
one which are not in the sieve.
```

5.4.1.5 def python.primes.odd_composite ( )

```
Iterator.  Generates odd composite numbers.
```

5.4.1.6 def python.primes.primes (*n* = None)

```
Iterator.  Generates primes.
```

5.4.1.7 def python.primes.primes_seq (*n*)

```
Generates a sequence of primes up to n.
```

Chapter 6

Class Documentation

6.1 card_t Struct Reference

This struct represents a playing card from a 52-cards deck.

Public Member Functions

- card_t ()
- card_t (unsigned char suit_, unsigned char denom_)

Public Attributes

- unsigned char suit
- unsigned char denom

6.1.1 Detailed Description

This struct represents a playing card from a 52-cards deck.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 card_t::card_t ( )

6.1.2.2 card_t::card_t (unsigned char *suit_*, unsigned char *denom_*)

6.1.3 Member Data Documentation

6.1.3.1 unsigned char card_t::denom

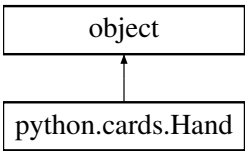
6.1.3.2 unsigned char card_t::suit

The documentation for this struct was generated from the following file:

- cxx/cards.cxx

6.2 python.cards.Hand Class Reference

Inheritance diagram for python.cards.Hand:



Public Member Functions

- def __init__ (self, cards)
- def n_of_a_kind (self, n)
- def royal_flush (self)
- def straight_flush (self)
- def four_of_a_kind (self)
- def full_house (self)
- def flush (self)
- def straight (self)
- def three_of_a_kind (self)
- def two_pairs (self)
- def one_pair (self)
- def value (self)
- def scoring_cards (self, score)
- def break_tie (self, other)
- def __lt__ (self, other)
- def __le__ (self, other)
- def __eq__ (self, other)
- def __ne__ (self, other)
- def __gt__ (self, other)
- def __ge__ (self, other)

6.2.1 Detailed Description

```
Stores a poker hand.  Poker hand contains five cards from the deck
of 52 playing cards.  This class also stores information on
how many cards of the same denomination are there in the hand.
```

6.2.2 Constructor & Destructor Documentation

6.2.2.1 def python.cards.Hand.__init__ (*self, cards*)

```
Creates new poker hand.  The argument must be a list with cards
represented as strings.
```

6.2.3 Member Function Documentation

6.2.3.1 def python.cards.Hand.__eq__ (*self, other*)

```
Implements the equality operator.
```

6.2.3.2 def python.cards.Hand.__ge__ (*self, other*)

```
Implements the greater or equal operator.
```

6.2.3.3 def python.cards.Hand.__gt__ (*self, other*)

```
Implements the greater than operator.
```

6.2.3.4 def python.cards.Hand.__le__ (*self, other*)

```
Implements the less than or equal operator.
```

6.2.3.5 def python.cards.Hand.__lt__ (*self, other*)

```
Implements the less than operator.
```

6.2.3.6 def python.cards.Hand.__ne__ (*self, other*)

```
Implements the not equal operator.
```

6.2.3.7 def python.cards.Hand.break_tie (*self, other*)

```
When two hands have the same value, this is used to break ties.  Note that
ties can still occur because the suit plays no role in this process.
```

6.2.3.8 def python.cards.Hand.flush (*self*)

```
Returns True if all cards in the hand come from the same suit.
```

6.2.3.9 def python.cards.Hand.four_of_a_kind (*self*)

```
Retruns True if the hand contains exactly four cards of the same
denomination.
```

6.2.3.10 def python.cards.Hand.full_house (*self*)

```
Returns True if the hand contains a pair and a triple of the same
denomination.
```

6.2.3.11 def python.cards.Hand.n_of_a_kind (*self, n*)

```
Returns True if the hand contains exactly n cards of the same
denomination.
```

6.2.3.12 def python.cards.Hand.one_pair (*self*)

```
Returns True if the hand has one pair.
```

6.2.3.13 def python.cards.Hand.royal_flush (*self*)

```
Returns True if all cards in the hand are of the same suit and
the smallest card is a ten.
```

6.2.3.14 def python.cards.Hand.scoring_cards (*self, score*)

```
When two hands have the same value, this procedure is used to compute
the cards which may potentially affect the tie break.  The cards are
returned in decreasing order (i.e. the first card should be
considered first to break the tie, then the second and so on).
```

6.2.3.15 def python.cards.Hand.straight (*self*)

```
Returns True if cards' values have no ''gaps'' in between.
```

6.2.3.16 def python.cards.Hand.straight_flush (*self*)

```
Similar to royal flush, except cards should be of consequtive
denominations (no need to start at 10).
```

6.2.3.17 def python.cards.Hand.three_of_a_kind (*self*)

```
Returns True if the hand has a triple.
```

6.2.3.18 def python.cards.Hand.two_pairs (*self*)

```
Returns True if the hand has two different pairs.
```

6.2.3.19 def python.cards.Hand.value (*self*)

```
Computes the value of the hand.  Hands are assigned value based on
some special combinations, such as ''Royal Flush'', ''Three Of A Kind''
etc, or by taking the denomination of the highest-ranked card.

See the documentation of other methods of this class for more details
on the meaning of special combinations.
```

The documentation for this class was generated from the following file:

- python/cards.py

6.3 hand Class Reference

This class provides operations on poker hand.

Public Member Functions

- hand (std::string raw)

  Creates new poker hand.
- bool n_of_a_kind (int n) const

  Verifies if the hand has n cards of the same denomination.
- bool royal_flush () const

  Verifies whether the hand has a T,J,Q,K,A sequence all of the same suit.
- bool straight_flush () const

  Verifies whether the hand has five cards of consequent denomination, all of same suit.
- bool four_of_a_kind () const

  Verifies whether the hand contains four card of the same denomination.
- bool full_house () const

  Verifies whether the hand contains a pair and that other three cards are of the same denomination.
- bool flush () const

  Verifies whether all cards in the hand are of the same suit.
- bool straight () const

  Verifies that there are no two cards of the same denomination and that the difference between the highest-valued card and the lowest-valued one is exactly five.
- bool three_of_a_kind () const

  Verifies whether the hand contains exactly three cards of the same denomination.
- bool two_pairs () const

  Verifies whether the hand contains two different pairs of cards of the same denomination.
- bool one_pair () const

  Verifies whether the hand contains a pair of cards of the same denomination.
- int value () const

  Calculates the value of the hand.
- std::vector< card_t > scoring_cards () const

  Returns only the cards that can affect a tie-break.
- int break_tie (const hand &other) const

  Computes amount of points by which one hand outweighs the other.
- bool operator> (const hand &other) const

  Establishes whether this hand's value is greater than the value of the other hand.

6.3.1 Detailed Description

This class provides operations on poker hand.

This class provides methods assessing the value of one's poker hand.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 hand::hand (std::string *raw*)

Creates new poker hand.

Parameters

raw	Raw poker hand.

Constructor will parse the hand from the given string and initialize its fields based on the parsing results.

6.3.3 Member Function Documentation

6.3.3.1 int hand::break_tie (const hand & *other*) const

Computes amount of points by which one hand outweighs the other.

Returns

The number of points by which this hand outweighs the other.

Once the tie is detected, this function is given the `other` hand, which has the same score, but might have different cards supporting that score. Ties are broken by first looking into the highest-ranked cards in the special combinations and later by looking at the remaining cards.

Note that it is still possible to draw (we don't establish any rules which assign weights to suits).

6.3.3.2 bool hand::flush ( ) const

Verifies whether all cards in the hand are of the same suit.

Returns

true iff the hand contains the “Flush" combination.

6.3.3.3 bool hand::four_of_a_kind ( ) const

Verifies whether the hand contains four card of the same denomination.

Returns

true iff the hand contains the “Four Of A Kind" combination.

6.3.3.4 bool hand::full_house ( ) const

Verifies whether the hand contains a pair and that other three cards are of the same denomination.

Returns

true iff the hand contains the “Full House" combination.

6.3.3.5 bool hand::n_of_a_kind (int *n*) const

Verifies if the hand has `n` cards of the same denomination.

Parameters

n	Number of same cards needed to be found.

Returns

true iff the hand has this exact number of cards of the same denomination.

Looks for at least one stretch of the required size. Note that the hand that contains 3 cards with the same denomination will not be considered as having 2 cards of the same denomination, unless it also contains exactly 2 cards of the same denomination. Other methods rely on this behavior.

6.3.3.6 bool hand::one_pair ( ) const

Verifies whether the hand contains a pair of cards of the same denomination.

Returns

true iff the hand contains the “One Pair" combination.

6.3.3.7 bool hand::operator> (const hand & *other*) const

Establishes whether this hand's value is greater than the value of the other hand.

Returns

`true` iff the value of this hand is greate than that of the other.

Invoking this operator will set in motion `hand::value()` and possibly `hand::break_tie()` in order to establish the winning hand.

6.3.3.8 bool hand::royal_flush ( ) const

Verifies whether the hand has a T,J,Q,K,A sequence all of the same suit.

Returns

true iff the hand contains the “Royal Flush" combination.

6.3.3.9 std::vector< card_t > hand::scoring_cards ( ) const

Returns only the cards that can affect a tie-break.

Returns

A vector of cards which can affect a tie-break.

Ties are broken by first considering the cards that participate in forming one of the special combinations

See also

hand::value. The cards are also considered in the order from highest to lowest value.

6.3.3.10 bool hand::straight ( ) const

Verifies that there are no two cards of the same denomination and that the difference between the highest-valued card and the lowest-valued one is exactly five.

Returns

true iff the hand contains the “straight" combination.

6.3.3.11 bool hand::straight_flush ( ) const

Verifies whether the hand has five cards of consequent denomination, all of same suit.

Returns

true iff the hand contains the “Straight Flush" combination.

6.3.3.12 bool hand::three_of_a_kind ( ) const

Verifies whether the hand contains exactly three cards of the same denomination.

Returns

true iff the hand contains the “Three Of A Kind" combination.

6.3.3.13 bool hand::two_pairs ( ) const

Verifies whether the hand contains two different pairs of cards of the same denomination.

Returns

true iff the hand contains the “Two Pairs" combination.

6.3.3.14 int hand::value ( ) const

Calculates the value of the hand.

Returns

An integer in range [3,23] indicating the total value of the hand.

The value of the hand is determined by it either having one of the eight known special combinations, or by its highest-ranked card. The combinations are ranked as follows:

- *One Pair*: 15
- *Two Pairs*: 16
- *Three Of A Kind*: 17
- *Straight*: 18
- *Flush*: 19
- *Full House*: 20
- *Four Of A Kind*: 21
- *Straight Flush*: 22
- *Royal Flush*: 23

The documentation for this class was generated from the following file:

- cxx/cards.cxx

6.4 primes Class Reference

This class is a container for the sieve of Erathosphenes.

Public Member Functions

- primes ()

  Default constructor.
- primes (int n)

  Creates a sieve, where the largest element is at most `n`*.*
- int next ()

  Returns the next element of the sieve.
- bool is_prime (int n) const

  Tests whether `n` *is prime.*
- std::size_t size () const

  The size of this sieve.
- int & operator[] (std::size_t n)

  Returns the referenct to the element of the sieve at index `n`*.*

6.4.1 Detailed Description

This class is a container for the sieve of Erathosphenes.

This class is used to iterate or add elements to the sieve of Erathosphenes. It can also efficiently assert containment (which would be equivalent to primality test in the given range).

6.4.2 Constructor & Destructor Documentation

6.4.2.1 primes::primes ( )

Default constructor.

Creates a sieve of all primes up to 10.

6.4.2.2 primes::primes (int *n*)

Creates a sieve, where the largest element is at most `n`.

Creates a sieve of with a number of elements up to (and possibly including) `n`.

6.4.3 Member Function Documentation

6.4.3.1 bool primes::is_prime (int *n*) const

Tests whether `n` is prime.

Returns

`true` iff `n` is a prime in this sieve.

This test will not try to extend the sieve in case the input is greater than its greates element. This means that this method will give incorect answers for primes outside the range.

6.4.3.2 int primes::next ( )

Returns the next element of the sieve.

Returns

Next prime number.

Once the existing sieve is exhausted, this function will attempt to extend it with the next prime.

6.4.3.3 int & primes::operator[ ] (std::size_t *n*)

Returns the referenct to the element of the sieve at index `n`.

Returns

A reference to an element in the sieve.

The the reference returned is suitable for assignment, so beware!

6.4.3.4 std::size_t primes::size ( ) const

The size of this sieve.

Returns

the size of this sieve.

The size of the sieve is the number of primes it contains, note that this will always be less then the largest prime it contains (not to be confused with the constructor's argument).

The documentation for this class was generated from the following file:

- cxx/primes.cxx

Chapter 7

File Documentation

7.1 cxx/cards.cxx File Reference

```
#include <vector>
#include <cmath>
#include <algorithm>
#include <string>
```

Classes

- struct card_t

  This struct represents a playing card from a 52-cards deck.
- class hand

  This class provides operations on poker hand.

Functions

- bool card_cmp (card_t a, card_t b)

  This is a helper function used to sort cards based on their denomination.
- std::vector< card_t > parse_cards (std::string raw)

  Parses hand (5 cards) from string.
- std::vector< unsigned char > group_cards (std::vector< card_t > cards)

  Groups cards by denomination.

7.1.1 Function Documentation

7.1.1.1 bool card_cmp (card_t *a*, card_t *b*)

This is a helper function used to sort cards based on their denomination.

Parameters

a	The card prior in the sequence.
b	The card following in the sequence.

Returns

true if a must precede b.

Helps `std::sort()` to sort a collection of `card_t` based on card's denomination.

7.1.1.2 std::vector< unsigned char > group_cards (std::vector< card_t > *cards*)

Groups cards by denomination.

Parameters

cards	Vector containing a sorted sequence of cards (not modified).

Returns

A vector with stretches of cards of the same denomination.

This function implements something very similar to run-length encoding. Even though this is a canonical group-by functionality, this function exploits the fact that the input sequence is sorted, so it is trivially linear.

7.1.1.3 std::vector< card_t > parse_cards (std::string *raw*)

Parses hand (5 cards) from string.

Parameters

raw	Taw string containing 5 card codes.

Returns

A vector with parsed cards.

Given a string with five card codes will create a sorted vector with five cards. Cards are sorted by their denomination.

7.2 cxx/combinatorics.cxx File Reference

```
#include <gmpxx.h>
```

Functions

- mpz_class fib (int n)
  Calculates N^{th} Fibonacci number.

7.2.1 Function Documentation

7.2.1.1 mpz_class fib (int *n*)

Calculates N^{th} Fibonacci number.

Parameters

n	The index into Fibonacci sequence you want to generate.

Returns

A biginteger (aka `mpz_class`) from GNU MP library.

This function uses iterative algorithm, and is also, technically, incorrect for the value of $N = 3$, but we never care for this value, and I prefer to keep the code short.

7.3 cxx/primes.cxx File Reference

```
#include <vector>
#include <cmath>
#include <algorithm>
```

Classes

- class primes

  This class is a container for the sieve of Erathosphenes.

Functions

- bool is_zero (int x)

  Helper for checking whether the given integer is a zero.
- std::vector< int > primes_seq (int n)

  Generates a sequence of prime numbers.
- bool decomposes_goldbach (int n)

  Asserts Goldbach's conjecture.

7.3.1 Detailed Description

This file contains some functions to compute prime numbers and sequences of primes.

7.3.2 Function Documentation

7.3.2.1 bool decomposes_goldbach (int *n*)

Asserts Goldbach's conjecture.

Parameters

n	The integer to which the conjecture's constraings are applied.

Returns

`true` iff the conjecture holds for `n`.

Goldbach conjectured that for every composite integer i it holds that $i = p + n^2$ for some prime p and an integer n. This function is an implementation of this conjecture.

7.3.2.2 bool is_zero (int *x*)

Helper for checking whether the given integer is a zero.

Parameters

x	The integer suspected to be zero

Returns

true iff `x = 0`.

This function is used to filter out zeros when generating Eratosphenes sieve. But, you are welcome to use it wherever you like.

7.3.2.3 std::vector< int > primes_seq (int *n*)

Generates a sequence of prime numbers.

Parameters

n	The largest prime in the generaetd sequence will be at most this big.

Returns

A vector containing primes in ascending order.

This function is a simple implementation or the sieve of Eratosphenes. It at first generates a vector of `n` integers in increasing order, then it looks into consequtive elements of this vector and:

if the element is a zero, it skips it.

otherwise, for all indices which are multiple of this element, except for the first one it replaces the elements with zeros.

Finally, it removes all zeros.

7.4 python/__init__.py File Reference

Namespaces

- python

7.5 python/cards.py File Reference

Classes

- class python.cards.Hand

Namespaces

- python.cards

Functions

- def python.cards.cards_cmp (a, b)
- def python.cards.denom (card)

7.6 python/combinatorics.py File Reference

Namespaces

- python.combinatorics

Functions

- def python.combinatorics.fib (n)
- def python.combinatorics.digits (n)
- def python.combinatorics.nub (nums)
- def python.combinatorics.has_square (n)

7.7 python/primes.py File Reference

Namespaces

- python.primes

Functions

- def python.primes.primes_seq (n)
- def python.primes.is_next_prime (n, sieve)
- def python.primes.is_prime (n, sieve)
- def python.primes.next_prime (sieve)
- def python.primes.primes
- def python.primes.odd_composite ()
- def python.primes.factors (n)

Index