beansoup Documentation

Release 1.0a4

Filippo Tampieri

February 12, 2017

Contents

Contents:

CHAPTER 1

beansoup package

1.1 Subpackages

1.1.1 beansoup.importers package

Submodules

beansoup.importers.amex module

Importers for American Express statements.

class beansoup.importers.amex.**PdfFilingImporter**(*account*, *basename=None*, *first_day=1*, *filename_regexp=None*)

Bases: *beansoup.importers.filing.Importer*

A filing importer for American Express PDF monthly statements.

filename_regexp = '^Statement_(?P<month>Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec) (?P<year>\\d{4})

beansoup.importers.csv module

Utilities to implement CSV importers.

class beansoup.importers.csv.**Importer**(*account*, *currency='CAD'*, *basename=None*, *first_day=None*, *filename_regexp=None*, *account_types=None*)

Bases: beancount.ingest.importer.ImporterProtocol

An importer base class for CSV bank and credit card statements.

Unfortunately, CSV files often do not contain any information to easily identify the account; for this reason, this importer relies on the name of the file to associate it to a particular account.

Derived classes need to implement the 'parse' method.

See beansoup.importers.td.Importer for a full example of how to derive a concrete importer from this class.

create_balance_entry(*filename*, *date*, *balance*)

extract(*file*)

Return extracted entries and errors.

file_account(*_*)

Return the account associated with the file

file_date(*file*)
Return the filing date for the file.

file_name(*file*)
Return the optional renamed account file name.

identify(*file*)
Identify whether the file can be processed by this importer.

name()
Include the account in the name.

parse(*file*)
Parse the CSV file.

Derived classes must implement this method to parse their CSV files.

Consider using the helper function 'beansoup.importers.csv.parse' to implement your custom CSV parser.

Parameters **file** – A cache.FileMemo object.

Returns A list of Row objects; one object per row. The order of the parsed rows is irrelevant; they will be sorted in ascending chronological order in a way that agrees with the balance values associated to each row. It that is not possible, the balance values will be ignored and the importer will be unable to extract balance directive, but will otherwise work as expected.

class beansoup.importers.csv.**Row**(*lineno*, *date*, *description*, *amount*, *balance*)
Bases: tuple

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

__getstate__()
Exclude the OrderedDict from pickling

static __new__(*_cls*, *lineno*, *date*, *description*, *amount*, *balance*)
Create new instance of Row(lineno, date, description, amount, balance)

__repr__()
Return a nicely formatted representation string

amount
Alias for field number 3

balance
Alias for field number 4

date
Alias for field number 1

description
Alias for field number 2

lineno
Alias for field number 0

beansoup.importers.csv.**parse**(*file*, *dialect*, *parse_row*)
Parse a CSV file.

This utility function makes it easy to parse a CSV file format for bank or credit card accounts.

It takes advantage of the ability to cache the file contents, but it does not attempt to cache the parsed result. Be careful when you consider caching the result of your parser in a cache.FileMemo object; often your row parser

will adjust the sign of the row balance according to the sign of the account associated with the importer using the parser; this means that CSV importers for accounts of opposite signs should not share the parsed results!

Parameters

- **file** – A cache.FileMemo object; the CSV file to be parsed.
- **dialect** – The name of a registered CSV dialect to use for parsing.
- **parse_row** – A function taking a row (a list of values) and its line number in the input file and returning a Row object.

Returns A list of Row objects in the same order as encountered in the CSV file.

beansoup.importers.csv.**sort_rows**(*rows*)

Sort the rows of a CSV file.

This function can sort the rows of a CSV file in ascending chronological order such that the balance values of each row match the sequence of transactions.

Parameters **rows** – A list of objects with a lineno, date, amount, and balance attributes.

Returns A pair with a sorted list of rows and an error. The error is None if the function could find an ordering agreeing with the balance values of its rows; otherwise, it is the line number in the CSV file corresponding to the first row not agreeing with its balance value.

beansoup.importers.filing module

A file-only importer.

class beansoup.importers.filing.**Importer**(*account*, *basename=None*, *first_day=1*, *filename_regexp=None*)

Bases: beancount.ingest.importer.ImporterProtocol

A document-filing class for monthly files; it does not import anything.

This importer only supports bean-identify and bean-file. It does not extract any transactions; in fact, it does not even open the file. It uses a regular expression to match a filename to an account and to a date (interpreted as the last day of a billing period).

extract(*file*)

Do not attempt to extract any transactions from the file.

file_account(*_*)

Return the account associated with the file

file_date(*file*)

Return the filing date for the file.

file_name(*file*)

Return the optional renamed account file name.

identify(*file*)

Identify whether the file can be processed by this importer.

name()

Include the filing account in the name.

beansoup.importers.mixins module

Mixins for importer classes.

class beansoup.importers.mixins.**FilterChain**(**args*, ***kwargs*)
Bases: object

A mixin to pass imported entries through a pipeline of filters.

This mixin modifies the extract method of a concrete instance of ImporterProtocol to run the extracted entries through a chain of arbitrary filters.

extract(*file*)
Extract the entries using the main importer and then run all the filters on them.

beansoup.importers.td module

Importers for TD Canada Trust.

class beansoup.importers.td.**Importer**(*account*, *currency='CAD'*, *basename=None*, *first_day=None*, *filename_regexp=None*, *account_types=None*)
Bases: *beansoup.importers.csv.Importer*

An importer for TD Canada Trust CSV statements.

parse(*file*)
Parse a TD Canada Trust CSV file.

Parameters **file** – A beansoup.ingest.cache.FileMemo instance; the CSV file to be parsed.

Returns A list of Row objects.

parse_row(*row*, *lineno*)
Parse a row of a TD Canada Trust CSV file.

Parameters

- **row** – A list of field values for the row.
- **lineno** – The line number where the row appears in the CSV file

Returns A beansoup.importers.csv.Row object.

Module contents

1.1.2 beansoup.plugins package

Submodules

beansoup.plugins.clear_transactions module

Work in progress. It works, but needs documentation and some cleaning.

class beansoup.plugins.clear_transactions.**AccountPairType**(*entries*)
Bases: object

class beansoup.plugins.clear_transactions.**Processor**(*args*)
Bases: object

clear_transaction_group(*txn_postings*)

clear_transactions(*entries*)

get_txn_clearing_posting(*txn*)

match_txn_postings(*txn_posting*, *txn_posting2*)

max_matching_date(*txn*)

beansoup.plugins.clear_transactions.**clear_transactions**(*entries*, *options_map*, *config_string*)

beansoup.plugins.config module

Utilities to help parse a plugin configuration string.

class beansoup.plugins.config.**ArgumentParser**(**args*, ***kwargs*)
Bases: argparse.ArgumentParser

error(*message*)

exit(*status=0*, *message=None*)

exception beansoup.plugins.config.**ParseError**(*source*, *message*)
Bases: Exception

beansoup.plugins.config.**re_type**(*string*)
Argument type for regular expressions.

It returns a compiled regular expression if string is not empty; None, otherwise. It raises argparse.ArgumentTypeError if the string is not a valid regular expression.

beansoup.plugins.deposit_in_transit module

Work in progress. It works, but needs documentation and some cleaning.

A plugin that automatically ties split deposit-in-transit transactions.

usage: beansoup.plugins.deposit_in_transit [–dit_component NAME] [–auto_open] [–same_day_merge] [–flag_pending] [–cleared_tag TAG] [–pending_tag TAG] [–ignored_tag TAG] [–link_prefix PREFIX] [–skip_re REGEX]

optional arguments:

--dit_component NAME	use NAME as the component name distinguishing deposit- in-transit accounts (default: DIT)
--auto_open	automatically open deposit-in-transit accounts (default: False)
--same_day_merge	merge same-day transactions with matching deposit-in- transit postings (default: False)
--flag_pending	annotate pending transactions with a ! flag (default: False)
--cleared_tag TAG	tag cleared transactions with TAG (default: DEPOSITED)
--pending_tag TAG	tag pending transactions with TAG (default: IN- TRANSIT)
--ignored_tag TAG	ignore transactions that have a TAG tag (default: IGNORED)
--link_prefix PREFIX	link pairs of cleared transactions with PREFIX string followed by increasing count; otherwise it uses UUIDs (default: None)
--skip_re REGEX	disable plugin if REGEX matches any sys.argv (default: None)

class beansoup.plugins.deposit_in_transit.**DITError**(*source*, *message*, *entry*)
Bases: tuple

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

__getstate__()
Exclude the OrderedDict from pickling

static __new__(*_cls*, *source*, *message*, *entry*)
Create new instance of DITError(source, message, entry)

__repr__()
Return a nicely formatted representation string

entry
Alias for field number 2

message
Alias for field number 1

source
Alias for field number 0

beansoup.plugins.deposit_in_transit.**is_pair_mergeable**(*pair*)

beansoup.plugins.deposit_in_transit.**match_dit**(*dit*, *candidate_dits*, *dit_component*)

beansoup.plugins.deposit_in_transit.**open_dit_accounts**(*entries*, *dit_component*)
Minimally adapted from beancount.plugins.auto_accounts.

beansoup.plugins.deposit_in_transit.**pair_dits**(*dits*, *dit_component*)

beansoup.plugins.deposit_in_transit.**plugin**(*entries*, *options_map*, *config_string*)

beansoup.plugins.deposit_in_transit.**process_entries**(*entries*, *args*)

beansoup.plugins.deposit_in_transit.**process_pair**(*pair*, *cleared_tag*, *cleared_links*, *same_day_merge*)

beansoup.plugins.deposit_in_transit.**process_singleton**(*singleton*, *flag_pending*, *pending_tag*)

beansoup.plugins.deposit_in_transit.**split_entries**(*entries*, *dit_component*, *ignored_tag*)

Module contents

1.1.3 beansoup.utils package

Submodules

beansoup.utils.dates module

Utilities for working with dates.

beansoup.utils.dates.**MONTHS**
Dict[str, int]

a map from month names to their ordinal values, starting at 1. The names are lowercase and can be full names, three-letter abbreviations, or one- or two-digit representations.

beansoup.utils.dates.**add_biz_days**(*date*, *num_biz_days*)
Add a number of business days to a date.

If the starting date falls on a weekend, it is moved to the next business day before adding the delta.

Parameters

- **date** (*datetime.date*) – The starting date.
- **num_biz_days** (*int*) – The number of business days to add to the starting date; it must be non-negative.

Returns the offset date.

Return type datetime.date

beansoup.utils.dates.**month_number**(*month*)
Turns a month name into its corresponding month number.

It recognizes full and abbreviated (three letters) English month names (case insensitive) as well as month number with or without a leading 0.

Parameters **month** (*str*) – The name of a month or its three-letter abbreviation or its numerical equivalent.

Returns The number in [1,12] corresponding to the given month name, or None if it does not recognize the given name.

Return type Optional[int]

beansoup.utils.links module

Utilities for working with links.

beansoup.utils.links.**count**(*link_prefix=None*, *start=1*)
A generator of unique link names.

Parameters

- **link_prefix** (*Optional[str]*) – If a string, link names will be of the form link_prefix-#; otherwise, they will be UUIDs.
- **start** (*int*) – The start of the number sequence when used with a link prefix.

Yields *str* – the next link name in the sequence.

beansoup.utils.periods module

Utilities to work with monthly billing periods.

beansoup.utils.periods.**count**(*date*, *reverse=False*)
A generator of monthly-spaced dates.

It enumerates monthly-spaced dates, starting at the given *date*. If the starting date falls on a day that is not in a given month, the date for that month will be the last day of that month.

Parameters

- **date** (*datetime.date*) – The starting date.
- **reverse** (*bool*) – If True, it generates dates in reverse chronological order.

Yields *datetime.date* – the next date in the sequence.

Example

```
>>> import datetime
>>> import itertools
>>> start = datetime.date(2016, 1, 31)
>>> [date.isoformat() for date in itertools.islice(count(start), 5)]
['2016-01-31', '2016-02-29', '2016-03-31', '2016-04-30', '2016-05-31']
```

beansoup.utils.periods.**enclose_date**(*date*, *first_day=1*)

Compute the monthly period containing the given date.

Parameters

- **date** (*datetime.date*) – The date to be contained.
- **first_day** (*int*) – The first day of the monthly cycle. It must fall in the interval [1,28].

Returns The start and end dates (inclusives) of the monthly period containing the given date.

Return type Tuple[datetime.date, datetime.date]

beansoup.utils.periods.**greatest_start**(*date*, *first_day=1*)

Compute the starting date of the monthly period containing the given date.

More formally, given a monthly cycle starting on *first_day* day of the month, it computes the greatest starting date that is less than or equal to the given *date*.

Parameters

- **date** (*datetime.date*) – An arbitrary date.
- **first_day** (*int*) – The first day of the monthly cycle. It must fall in the interval [1,28].

Returns The starting date of the monthly period containing the given date.

Return type datetime.date

beansoup.utils.periods.**lowest_end**(*date*, *first_day=1*)

Compute the ending date of the monthly period containing the given date.

More formally, given a monthly cycle starting on *first_day* day of the month, it computes the lowest ending date that is greater than or equal to the given *date*. Note that the ending date is inclusive, i.e. it is included in the monthly period.

Parameters

- **date** (*datetime.date*) – An arbitrary date.
- **first_day** (*int*) – The first day of the monthly cycle. It must fall in the interval [1,28].

Returns The ending date of the monthly period containing the given date.

Return type datetime.date

beansoup.utils.periods.**next**(*date*)

Add one month to the given date.

Parameters **date** (*datetime.date*) – The starting date.

Returns One month after the starting date unless the starting date falls on a day that is not in the next month; in that case, it returns the last day of the next month.

Return type datetime.date

Example

```
>>> import datetime
>>> next(datetime.date(2016, 1, 31))
datetime.date(2016, 2, 29)
```

beansoup.utils.periods.**prev**(*date*)

Subtract one month from the given date.

Parameters **date** (*datetime.date*) – The starting date.

Returns One month before the starting date unless the starting date falls on a day that is not in the previous month; in that case, it returns the last day of the previous month.

Return type datetime.date

Example

```
>>> import datetime
>>> prev(datetime.date(2016, 3, 31))
datetime.date(2016, 2, 29)
```

Module contents

1.2 Submodules

1.3 beansoup.transactions module

Utilities to work with beancount.core.data.Transaction objects.

class beansoup.transactions.**TransactionCompleter**(*existing_entries*, *account*, *min_score=0.5*, *max_age=None*, *interpolated=False*)

Bases: object

A class capable of completing partial transactions.

Importers typically generate incomplete transactions with a single posting to the main account related to the imported data. This class attempts to complete those transaction by adding a second posting to an account chosen based on the existing transaction history for the main account.

It looks for existing transactions that have exactly two postings and where one of the two postings is to the main account. It scores each of these transactions based on the similarity of its payee and narration fields to the narration field of the incomplete transaction and selects the one with the highest score as a model to fill in the missing posting of the incomplete transaction. Equal scores are broken by selecting the most recent transaction.

__call__(*entries*)

Same as *complete_entries* method.

complete_entries(*entries*)

Complete the given entries.

Only transactions with a single posting to the account bound to the completer may be modified.

Parameters **entries** – The entries to be completed.

Returns A list of completed entries

complete_entry(*entry*)
Complete the given entry.

This method attempts to complete the entry only if it is a transaction with a single posting to the account bound to the completer. The entry will be completed only if a suitable model transaction can be found. If multiple model transactions are found that balance the transaction against different account, the missing posting will be flagged for review.

Parameters **entry** – The entry to be completed.

Returns: True is the entry was completed; False, otherwise.

find_best_model(*txn*)
Return the best model for the given incomplete transaction.

Parameters **txn** – A beancount.core.data.Transaction object; an incomplete transaction with a single posting.

Returns A pair of a beancount.core.data.Transaction object and a set of account strings; the first part is the model transaction or None, if no suitable model could be found; the second part is a set of the different accounts used by top-scoring transaction to balance the posting to the target account.

score_model(*model_txn*, *txn*)
Score an existing transaction for its ability to provide a model for an incomplete transaction.

Parameters

- **model_txn** – The transaction to be scored.
- **txn** – The incomplete transaction.

Returns A float number representing the score, normalized in [0,1].

1.4 beansoup.version module

Project version.

1.5 Module contents

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

b

Symbols

A

B

C

D

E