Lending Club API Documentation

Release 0.1.10

Jeremy Gillick

Jul 06, 2017

Contents

This is the API documentation for the stand-alone python module for interacting with your Lending Club account. In a nutshell, it lets you check your cash balance, search for notes, build orders, invest and more.

CHAPTER 1

Disclaimer

I have tested this tool to the best of my ability, but understand that it may have bugs. Use at your own risk!

CHAPTER 2

Download

This project is hosted on github

CHAPTER 3

API

LendingClub

The stand-alone python module for interacting with your Lending Club account.

class lendingclub.**LendingClub**(*email=None*, *password=None*, *logger=None*)

The main entry point for interacting with Lending Club.

Parameters **email** : string

The email of a user on Lending Club

password : string

The user's password, for authentication.

logger : Logger

A python logger used to get debugging output from this module.

Examples

Get the cash balance in your lending club account:

```
>>> from lendingclub import LendingClub
>>> lc = LendingClub()
>>> lc.authenticate()          # Authenticate with your lending club credentials
Email:test@test.com
Password:
True
>>> lc.get_cash_balance()      # See the cash you have available for investing
463.80000000000001
```

You can also enter your email and password when you instantiate the LendingClub class, in one line:

```
>>> from lendingclub import LendingClub
>>> lc = LendingClub(email='test@test.com', password='secret123')
>>> lc.authenticate()
True
```

Attributes

order	
session	

Methods

assign_to_portfolio(*portfolio_name*, *loan_id*, *order_id*)

Assign a note to a named portfolio. *loan_id* and *order_id* can be either integer values or lists. If choosing lists, they both **MUST** be the same length and line up. For example, *order_id[5]* must be the order ID for *loan_id[5]*

Parameters **portfolio_name** : string

The name of the portfolio to assign a the loan note to – new or existing

loan_id : int or list

The loan ID, or list of loan IDs, to assign to the portfolio

order_id : int or list

The order ID, or list of order IDs, that this loan note was invested with. You can find this in the dict returned from *get_note()*

Returns boolean

True on success

authenticate(*email=None*, *password=None*)

Attempt to authenticate the user.

Parameters **email** : string

The email of a user on Lending Club

password : string

The user's password, for authentication.

Returns boolean

True if the user authenticated or raises an exception if not

Raises **session.AuthenticationError**

If authentication failed

session.NetworkError

If a network error occurred

build_portfolio(*cash*, *max_per_note=25*, *min_percent=0*, *max_percent=20*, *filters=None*, *automatically_invest=False*, *do_not_clear_staging=False*)

Returns a list of loan notes that are diversified by your min/max percent request and filters. One way to

invest in these loan notes, is to start an order and use add_batch to add all the loan fragments to them. (see examples)

Parameters **cash** : int

The total amount you want to invest across a portfolio of loans (at least $25).

max_per_note : int, optional

The maximum dollar amount you want to invest per note. Must be a multiple of 25

min_percent : int, optional

THIS IS NOT PER NOTE, but the minimum average percent of return for the entire portfolio.

max_percent : int, optional

THIS IS NOT PER NOTE, but the maxmimum average percent of return for the entire portfolio.

filters : lendingclub.filters.*, optional

The filters to use to search for portfolios

automatically_invest : boolean, optional

If you want the tool to create an order and automatically invest in the portfolio that matches your filter. (default False)

do_not_clear_staging : boolean, optional

Similar to automatically_invest, don't do this unless you know what you're doing. Setting this to True stops the method from clearing the loan staging area before returning

Returns dict

A dict representing a new portfolio or False if nothing was found. If *automatically_invest* was set to *True*, the dict will contain an *order_id* key with the ID of the completed investment order.

Notes

The min/max_percent parameters

When searching for portfolios, these parameters will match a portfolio of loan notes which have an **AVERAGE** percent return between these values. If there are multiple portfolio matches, the one closes to the max percent will be chosen.

Examples

Here we want to invest $400 in a portfolio with only B, C, D and E grade notes with an average overall return between 17% - 19%. This similar to finding a portfolio in the 'Invest' section on lendingclub.com:

```
>>> from lendingclub import LendingClub
>>> from lendingclub.filters import Filter
>>> lc = LendingClub()
>>> lc.authenticate()
Email:test@test.com
Password:
True
```

```
>>> filters = Filter()                    # Set the search filters (only B, C,␣
↪D and E grade notes)
>>> filters['grades']['C'] = True
>>> filters['grades']['D'] = True
>>> filters['grades']['E'] = True
>>> lc.get_cash_balance()                 # See the cash you have available for␣
↪investing
463.80000000000001

>>> portfolio = lc.build_portfolio(400, # Invest $400 in a portfolio...
        min_percent=17.0,                 # Return percent average between 17 -␣
↪19%
        max_percent=19.0,
        max_per_note=50,                  # As much as $50 per note
        filters=filters)                  # Search using your filters

>>> len(portfolio['loan_fractions'])     # See how many loans are in this␣
↪portfolio
16
>>> loans_notes = portfolio['loan_fractions']
>>> order = lc.start_order()             # Start a new order
>>> order.add_batch(loans_notes)         # Add the loan notes to the order
>>> order.execute()                      # Execute the order
1861880
```

Here we do a similar search, but automatically invest the found portfolio. **NOTE** This does not allow you to review the portfolio before you invest in it.

```
>>> from lendingclub import LendingClub
>>> from lendingclub.filters import Filter
>>> lc = LendingClub()
>>> lc.authenticate()
Email:test@test.com
Password:
True
                                         # Filter shorthand
>>> filters = Filter({'grades': {'B': True, 'C': True, 'D': True, 'E': True}})
>>> lc.get_cash_balance()                # See the cash you have available for␣
↪investing
463.80000000000001
```

```
>>> portfolio = lc.build_portfolio(400,
        min_percent=17.0,
        max_percent=19.0,
        max_per_note=50,
        filters=filters,
        automatically_invest=True)       # Same settings, except invest␣
↪immediately
```

```
>>> portfolio['order_id']                # See order ID
1861880
```

get_cash_balance()

Returns the account cash balance available for investing

Returns float

The cash balance in your account.

get_investable_balance()
Returns the amount of money from your account that you can invest. Loans are multiples of $25, so this is your total cash balance, adjusted to be a multiple of 25.

Returns int

The amount of cash you can invest

get_note(*note_id*)
Get a loan note that you've invested in by ID

Parameters **note_id** : int

The note ID

Returns dict

A dictionary representing the matching note or False

Examples

```
>>> from lendingclub import LendingClub
>>> lc = LendingClub(email='test@test.com', password='secret123')
>>> lc.authenticate()
True
>>> notes = lc.my_notes()                    # Get the first 100 loan notes
>>> len(notes['loans'])
100
>>> notes['total']                            # See the total number of loan␣
↪notes you have
630
>>> notes = lc.my_notes(start_index=100)    # Get the next 100 loan notes
>>> len(notes['loans'])
100
>>> notes = lc.my_notes(get_all=True)        # Get all notes in one request␣
↪(may be slow)
>>> len(notes['loans'])
630
```

get_portfolio_list(*names_only=False*)
Get your list of named portfolios from the lendingclub.com

Parameters **names_only** : boolean, optional

If set to True, the function will return a list of portfolio names, instead of portfolio objects

Returns list

A list of portfolios (or names, if *names_only* is True)

get_saved_filter(*filter_id*)
Load a single saved search filter from the site by ID

Parameters **filter_id** : int

The ID of the saved filter

Returns SavedFilter

A *lendingclub.filters.SavedFilter* object or False

get_saved_filters()
Get a list of all the saved search filters you've created on lendingclub.com

Returns list

List of *lendingclub.filters.SavedFilter* objects

is_site_available()
Returns true if we can access LendingClub.com This is also a simple test to see if there's an internet connection

Returns boolean

my_notes(*start_index=0*, *limit=100*, *get_all=False*, *sort_by='loanId'*, *sort_dir='asc'*)
Return all the loan notes you've already invested in. By default it'll return 100 results at a time.

Parameters **start_index** : int, optional

The result index to start on. By default only 100 records will be returned at a time, so use this to start at a later index in the results. For example, to get results 200 - 300, set *start_index* to 200. (default is 0)

limit : int, optional

The number of results to return per request. (default is 100)

get_all : boolean, optional

Return all results in one request, instead of 100 per request.

sort_by : string, optional

What key to sort on

sort_dir : {'asc', 'desc'}, optional

Which direction to sort

Returns dict

A dictionary with a list of matching notes on the *loans* key

search(*filters=None*, *start_index=0*, *limit=100*)
Search for a list of notes that can be invested in. (similar to searching for notes in the Browse section on the site)

Parameters **filters** : lendingclub.filters.*, optional

The filter to use to search for notes. If no filter is passed, a wildcard search will be performed.

start_index : int, optional

The result index to start on. By default only 100 records will be returned at a time, so use this to start at a later index in the results. For example, to get results 200 - 300, set *start_index* to 200. (default is 0)

limit : int, optional

The number of results to return per request. (default is 100)

Returns dict

A dictionary object with the list of matching loans under the *loans* key.

search_my_notes(*loan_id=None*, *order_id=None*, *grade=None*, *portfolio_name=None*, *status=None*, *term=None*)

Search for notes you are invested in. Use the parameters to define how to search. Passing no parameters is the same as calling *my_notes(get_all=True)*

Parameters **loan_id** : int, optional

Search for notes for a specific loan. Since a loan is broken up into a pool of notes, it's possible to invest multiple notes in a single loan

order_id : int, optional

Search for notes from a particular investment order.

grade : {A, B, C, D, E, F, G}, optional

Match by a particular loan grade

portfolio_name : string, optional

Search for notes in a portfolio with this name (case sensitive)

status : string, {issued, in-review, in-funding, current, charged-off, late, in-grace-period, fully-paid}, optional

The funding status string.

term : {60, 36}, optional

Term length, either 60 or 36 (for 5 year and 3 year, respectively)

Returns dict

A dictionary with a list of matching notes on the *loans* key

set_logger(*logger*)

Set a logger to send debug messages to

Parameters **logger** : Logger

A python logger used to get debugging output from this module.

start_order()

Start a new investment order for loans

Returns lendingclub.Order

The *lendingclub.Order* object you can use for investing in loan notes.

version()

Return the version number of the Lending Club Investor tool

Returns string

The version number string

Exceptions

exception lendingclub.**LendingClubError**(*value*, *response=None*)

Bases: exceptions.Exception

An error occurred. If the error was the result of an API call, the response attribute will contain the HTTP requests response object that was used to make the call to LendingClub.

Parameters **value** : string

The error message

response : requests.Response

Filters

Filters are used to search lending club for loans to invest in. There are many filters you can use, here are some examples with the main *Filter* class.

For example, to search for B grade loans, you could create a filter like this:

```
>>> filters = Filter()
>>> filters['grades']['B'] = True
```

Or, another more complex example:

```
>>> filters = Filter()
>>> filters['grades']['B'] = True
>>> filters['funding_progress'] = 90
>>> filters['term']['Year5'] = False
```

This would search for B grade loans that are at least 90% funded and not 5 year loans.

Filters currently do not support all search criteria. To see what is supported, create one and print it:

```
>>> filter = Filter()
>>> print filter
{'exclude_existing': True,
 'funding_progress': 0,
 'grades': {'A': False,
            'All': True,
            'B': False,
            'C': False,
            'D': False,
            'E': False,
            'F': False,
            'G': False},
 'term': {'Year3': True,
          'Year5': True}}
```

You can also set the values on instantiation:

```
>>> filters = Filter({'grades': {'B': True, 'C': True, 'D': True, 'E': True}})
```

Filter

class lendingclub.filters.**Filter**(*filters=None*)

Bases: dict

The default search filter that let's you refine your search based on a dictionary of search facets. Not all search options are supported yet.

Parameters **filters** : dict, optional

This will override any of the search filters you want on instantiation.

Examples

See the default filters:

```
>>> from lendingclub.filters import Filter
>>> from pprint import pprint
>>> filter = Filter()
>>> pprint(filter)
{'exclude_existing': True,
 'funding_progress': 0,
 'grades': {'A': False,
            'All': True,
            'B': False,
            'C': False,
            'D': False,
            'E': False,
            'F': False,
            'G': False},
 'term': {'Year3': True,
          'Year5': True}}
```

Set filters on instantiation:

```
>>> from lendingclub.filters import Filter
>>> from pprint import pprint
>>> filters = Filter({'grades': {'B': True, 'C': True, 'D': True, 'E': True}})
>>> pprint(filters['grades'])
{'All': False,
 'A': False,
 'B': True,
 'C': True,
 'D': True,
 'E': True,
 'F': False,
 'G': False}
```

Methods

search_string()
" Returns the JSON string that LendingClub expects for it's search

validate(*results*)
Validate that the results indeed match the filters. It's a VERY good idea to run your search results through this, even though the filters were passed to LendingClub in your search. Since we're not using formal APIs for LendingClub, they could change the way their search works at anytime, which might break the filters.

Parameters **results** : list

A list of loan note records returned from LendingClub

Returns boolean

True or raises FilterValidationError

Raises **FilterValidationError**

If a loan does not match the filter criteria

validate_one(*loan*)
Validate a single loan result record against the filters

Parameters **loan** : dict

A single loan note record

Returns boolean

True or raises FilterValidationError

Raises **FilterValidationError**

If the loan does not match the filter criteria

FilterByLoanID

class lendingclub.filters.**FilterByLoanID**(*loan_id*)
Bases: *lendingclub.filters.Filter*

Creates a filter to search by loan ID. You can either search by 1 loan ID or for multiple loans by ID.

Parameters **loan_id** : int or list

The loan ID or a list of loan IDs

Examples

Search for 1 loan by ID:

```
>>> from lendingclub import LendingClub
>>> from lendingclub.filters import FilterByLoanID
>>> lc = LendingClub(email='test@test.com', password='secret123')
>>> lc.authenticate()
True
>>> filter = FilterByLoanID(1234)  # Search for the loan 1234
>>> results = lc.search(filter)
>>> len(results['loans'])
1
```

Search for multiple loans by ID:

```
>>> from lendingclub import LendingClub
>>> from lendingclub.filters import FilterByLoanID
>>> lc = LendingClub(email='test@test.com', password='secret123')
>>> lc.authenticate()
True
>>> filter = FilterByLoanID(54321, 76432)  # Search for two loans: 54321 and 76432
>>> results = lc.search(filter)
>>> len(results['loans'])
2
```

Methods

SavedFilter

class lendingclub.filters.**SavedFilter**(*lc*, *filter_id*)
Bases: *lendingclub.filters.Filter*

Load a saved search filter from the site. Since this is loading a filter from the server, the individual values cannot be modified. Most often it is easiest to load the saved filters from LendingClub, via *get_saved_filters* and *get_saved_filter*. See examples.

Parameters **lc** : *lendingclub.LendingClub*

An instance of the LendingClub class that will be used to communicate with the site

filter_id : int

The ID of the filter to load

Examples

The SavedFilter needs to use an instance of LendingClub to access the site, so the class has a couple wrappers you can use to load SavedFilters. Here are a couple examples of loading saved filters from the LendingClub instance.

Load all saved filters:

```
>>> from lendingclub import LendingClub
>>> from lendingclub.filters import SavedFilter
>>> lc = LendingClub(email='test@test.com', password='secret123')
>>> lc.authenticate()
True
>>> filters = SavedFilter.all_filters(lc)     # Get a list of all saved␣
↪filters on LendinClub.com
>>> print filters
[<SavedFilter: 12345, '90 Percent'>, <SavedFilter: 23456, 'Only A loans'>]
```

Load a single saved filter:

```
>>> from lendingclub import LendingClub
>>> from lendingclub.filters import SavedFilter
>>> lc = LendingClub(email='test@test.com', password='secret123')
>>> lc.authenticate()
True
>>> filter = lc.get_saved_filter(23456)     # Get a single saved search filter␣
↪from the site by ID
>>> filter.name
u'Only A'
```

Attributes

id	
json	
json_text	
lc	
name	
response	

Methods

static **all_filters**(*lc*)
Get a list of all your saved filters

Parameters **lc** : *lendingclub.LendingClub*

An instance of the authenticated LendingClub class

Returns list

A list of lendingclub.filters.SavedFilter objects

load()
Load the filter from the server

reload()
Reload the saved filter

search_string()
Get the search JSON string to send to the server

Exceptions

exception lendingclub.filters.**FilterValidationError**(*value=None*, *loan=None*, *criteria=None*)
Bases: exceptions.Exception

A loan note does not match the filters set.

After a search is performed, each loan returned from the server will be validate against the filter's criteria, for good measure. If it doesn't match, this exception is thrown.

Parameters **value** : string

The error message

loan : dict

The loan that did not match

criteria : string

The filter item that the loan failed on.

exception lendingclub.filters.**SavedFilterError**(*value*, *request=None*)
Bases: exceptions.Exception

An error occurred while loading or processing a :class:SavedFilter

Parameters **value** : string

The error message

response : requests.Response

The Response object from the HTTP request to find the saved filter.

Order

class lendingclub.**Order**(*lc*)

Used to create an order for one or more loan notes. It's best to create the Order instance through the *lendingclub.LendingClub.start_order()* method (see examples below).

Parameters **lc** : *lendingclub.LendingClub*

The LendingClub API object that is used to communicate with lendingclub.com

Examples

Invest in a single loan:

```
>>> from lendingclub import LendingClub
>>> lc = LendingClub()
>>> lc.authenticate()
Email:test@test.com
Password:
True
>>> order = lc.start_order()            # Start a new investment order
>>> order.add(654321, 25)               # Add loan 654321 to the order with a $25␣
↪investment
>>> order.execute()                     # Execute the order
1861879
>>> order.order_id                      # See the order ID
1861879
>>> order.assign_to_portfolio('Foo')    # Assign the loan in this order to a␣
↪portfolio called 'Foo'
True
```

Invest $25 in multiple loans:

```
>>> from lendingclub import LendingClub
>>> lc = LendingClub(email='test@test.com', password='mysecret')
>>> lc.authenticate()
True
>>> loans = [1234, 2345, 3456]        # Create a list of loan IDs
>>> order = lc.start_order()           # Start a new order
>>> order.add_batch(loans, 25)         # Invest $25 in each loan
>>> order.execute()                    # Execute the order
1861880
```

Invest different amounts in multiple loans:

```
>>> from lendingclub import LendingClub
>>> lc = LendingClub(email='test@test.com', password='mysecret')
>>> lc.authenticate()
True
>>> loans = [
```

```
    {'loan_id': 1234, invest_amount: 50},  # $50 in 1234
    {'loan_id': 2345, invest_amount: 25},  # $25 in 2345
    {'loan_id': 3456, invest_amount: 150}  # $150 in 3456
]
>>> order = lc.start_order()
>>> order.add_batch(loans)                 # Do not pass `batch_amount` parameter␣
↪this time
>>> order.execute()                        # Execute the order
1861880
```

Attributes

lc	
loans	

Methods

add(*loan_id*, *amount*)

Add a loan and amount you want to invest, to your order. If this loan is already in your order, it's amount will be replaced with the this new amount

Parameters **loan_id** : int or dict

The ID of the loan you want to add or a dictionary containing a *loan_id* value

amount : int % 25

The dollar amount you want to invest in this loan, as a multiple of 25.

add_batch(*loans*, *batch_amount=None*)

Add a batch of loans to your order.

Parameters **loans** : list

A list of dictionary objects representing each loan and the amount you want to invest in it (see examples below).

batch_amount : int, optional

The dollar amount you want to set on ALL loans in this batch. **NOTE:** This will override the invest_amount value for each loan.

Examples

Each item in the loans list can either be a loan ID OR a dictionary object containing *loan_id* and *invest_amount* values. The invest_amount value is the dollar amount you wish to invest in this loan.

List of IDs:

```
# Invest $50 in 3 loans
order.add_batch([1234, 2345, 3456], 50)
```

List of Dictionaries:

```
# Invest different amounts in each loans
order.add_batch([
    {'loan_id': 1234, invest_amount: 50},
    {'loan_id': 2345, invest_amount: 25},
    {'loan_id': 3456, invest_amount: 150}
])
```

assign_to_portfolio(*portfolio_name=None*)
Assign all the notes in this order to a portfolio

Parameters **portfolio_name – The name of the portfolio to assign it to (new or existing)**

Returns boolean

True on success

Raises **LendingClubError**

execute(*portfolio_name=None*)
Place the order with LendingClub

Parameters **portfolio_name** : string

The name of the portfolio to add the invested loan notes to. This can be a new or existing portfolio name.

Returns int

The completed order ID

Raises **LendingClubError**

remove(*loan_id*)
Remove a loan from your order

Parameters **loan_id** : int

The ID of the loan you want to remove

remove_all()
Remove all loans from your order

update(*loan_id*, *amount*)
Update a loan in your order with this new amount

Parameters **loan_id** : int or dict

The ID of the loan you want to update or a dictionary containing a *loan_id* value

amount : int % 25

The dollar amount you want to invest in this loan, as a multiple of 25.

Session

Manage the LendingClub user session and all raw HTTP calls to the LendingClub site. This will almost always be accessed through the API calls in *lendingclub.LendingClub* instead of directly.

class lendingclub.session.**Session**(*email=None*, *password=None*, *logger=None*)

Attributes

email	
last_response	

Methods

authenticate(*email=None*, *password=None*)
Authenticate with LendingClub and preserve the user session for future requests. This will raise an exception if the login appears to have failed, otherwise it returns True.

Since Lending Club doesn't seem to have a login API, the code has to try to decide if the login worked or not by looking at the URL redirect and parsing the returned HTML for errors.

Parameters **email** : string

The email of a user on Lending Club

password : string

The user's password, for authentication.

Returns boolean

True on success or throws an exception on failure.

Raises **session.AuthenticationError**

If authentication failed

session.NetworkError

If a network error occurred

base_url = 'https://www.lendingclub.com/'
The root URL that all paths are appended to

build_url(*path*)
Build a LendingClub URL from a URL path (without the domain).

Parameters **path** : string

The path part of the URL after the domain. i.e. https://www.lendingclub.com/<path>

clear_session_order()
Clears any existing order in the LendingClub.com user session.

get(*path*, *query=None*, *redirects=True*)
GET request wrapper for *request()*

head(*path*, *query=None*, *data=None*, *redirects=True*)
HEAD request wrapper for *request()*

is_site_available()
Returns true if we can access LendingClub.com This is also a simple test to see if there's a network connection

Returns boolean

True or False

json_success(*json*)
Check the JSON response object for the success flag

Parameters **json** : dict

A dictionary representing a JSON object from lendingclub.com

last_request_time = 0
The timestamp of the last HTTP request

post(*path*, *query=None*, *data=None*, *redirects=True*)
POST request wrapper for *request()*

request(*method*, *path*, *query=None*, *data=None*, *redirects=True*)
Sends HTTP request to LendingClub.

Parameters **method** : {GET, POST, HEAD, DELETE}

The HTTP method to use: GET, POST, HEAD or DELETE

path : string

The path that will be appended to the domain defined in *base_url*.

query : dict

A dictionary of query string parameters

data : dict

A dictionary of POST data values

redirects : boolean

True to follow redirects, False to return the original response from the server.

Returns requests.Response

A requests.Response object

session_timeout = 10
Minutes until the session expires. The session will attempt to reauth before the next HTTP call after timeout.

set_logger(*logger*)
Have the Session class send debug logging to your python logging logger. Set to None stop the logging.

Parameters **logger** : Logger

The logger to send debug output to.

Exceptions

exception lendingclub.session.**SessionError**(*value*, *origin=None*)
Bases: exceptions.Exception

Base exception class for *lendingclub.session*

Parameters **value** : string

The error message

origin : Exception

The original exception, if this exception was caused by another.

exception lendingclub.session.**AuthenticationError**(*value*, *origin=None*)
Bases: *lendingclub.session.SessionError*

Authentication failed

exception lendingclub.session.**NetworkError**(*value*, *origin=None*)
Bases: *lendingclub.session.SessionError*

An error occurred while making an HTTP request

CHAPTER 4

Examples

Here are a few examples, in the python interactive shell, of using the Lending Club API module.

Simple Search and Order

Searching for grade B loans and investing $25 in the first one you find:

```
>>> from lendingclub import LendingClub
>>> from lendingclub.filters import Filter
>>> lc = LendingClub()
>>> lc.authenticate()
Email:test@test.com
Password:
True
>>> filters = Filter()
>>> filters['grades']['B'] = True       # Filter for only B grade loans
>>> results = lc.search(filters)        # Search using this filter
>>> len(results['loans'])               # See how many results returned
100
>>> results['loans'][0]['loan_id']      # See the loan_id of the first loan
1763030
>>> order = lc.start_order()            # Start a new investment order
>>> order.add(1763030, 25)              # Add the first loan to the order with a $25␣
↪investment
>>> order.execute()                     # Execute the order
1861879
>>> order.order_id                      # See the order ID
1861879
>>> order.assign_to_portfolio('Foo')    # Assign the loans in this order to a␣
↪portfolio called 'Foo'
True
```

Invest in a Portfolio of Loans

Here we want to invest $400 in a portfolio with only B, C, D and E grade notes with an average overall return between 17% - 19%. This similar to finding a portfolio in the 'Invest' section on lendingclub.com:

```
>>> from lendingclub import LendingClub
>>> from lendingclub.filters import Filter
>>> lc = LendingClub(email='test@test.com', password='secret123')
>>> lc.authenticate()
True
>>> filters = Filter()                # Set the filters
>>> filters['grades']['B'] = True     # See Pro Tips for a shorter way to do this
>>> filters['grades']['C'] = True
>>> filters['grades']['D'] = True
>>> filters['grades']['E'] = True
>>> lc.get_cash_balance()             # See the cash you have available for investing
463.80000000000001
                                      # Find a portfolio to invest in ($400, between␣
↪17-19%, $25 per note)
>>> portfolio = lc.build_portfolio(400,
        min_percent=17.0,
        max_percent=19.0,
        max_per_note=25,
        filters=filters)

>>> len(portfolio['loan_fractions'])  # See how many loans are in this portfolio
16
>>> loans_notes = portfolio['loan_fractions']
>>> order = lc.start_order()          # Start a new order
>>> order.add_batch(loans_notes)      # Add the loan notes to the order
>>> order.execute()                   # Execute the order
1861880
```

Your Loan Notes

Get a list of the loan notes that you have **already** invested in (by default this will only return 100 at a time):

```
>>> from lendingclub import LendingClub
>>> lc = LendingClub(email='test@test.com', password='secret123')
>>> lc.authenticate()
True
>>> notes = lc.my_notes()                  # Get the first 100 loan notes
>>> len(notes['loans'])
100
>>> notes['total']                          # See the total number of loan notes you␣
↪have
630
>>> notes = lc.my_notes(start_index=100)   # Get the next 100 loan notes
>>> len(notes['loans'])
100
>>> notes = lc.my_notes(get_all=True)      # Get all notes in one request (may be␣
↪slow)
>>> len(notes['loans'])
630
```

Using Saved Filters

Use a filter saved on lendingclub.com to search for loans **SEE NOTE BELOW**:

```
>>> from lendingclub import LendingClub
>>> from lendingclub.filters import SavedFilter
>>> lc = LendingClub()
>>> lc.authenticate()
Email:test@test.com
Password:
True
>>> filters = SavedFilter.all_filters(lc)     # Get a list of all saved filters on␣
↪LendinClub.com
>>> print filters                             # I've pretty printed the output for you
[
    <SavedFilter: 12345, '90 Percent'>,
    <SavedFilter: 23456, 'Only A loans'>
]
>>> filter = SavedFilter(lc, 7611034)         # Load a saved filter by ID 7611034
>>> filter.name
u'Only A'
>>> results = lc.search(filter)               # Search for loan notes with that filter
>>> len(results['loans'])
100
```

NOTE: When using saved search filters you should always confirm that the returned results match your filters. This is because LendingClub's search API is not very forgiving. When we get the saved filter from the server and then send it to the search API, if any part of it has been altered or becomes corrupt, LendingClub will do a wildcard search instead of using the filter. The code in this python module takes great care to keep the filter pristine and check for inconsistencies, but that's no substitute for the individual investor's diligence.

Batch Investing

Invest in a list of loans in one action:

```
>>> from lendingclub import LendingClub
>>> lc = LendingClub(email='test@test.com', password='secret123')
>>> lc.authenticate()
True
>>> loans = [1234, 2345, 3456]  # Create a list of loan IDs
>>> order = lc.start_order()    # Start a new order
>>> order.add_batch(loans, 25)  # Invest $25 in each loan
>>> order.execute()             # Execute the order
1861880
```

CHAPTER 5

License

The MIT License (MIT)

Copyright (c) 2013 Jeremy Gillick

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Python Module Index

l

Index

S

U

V