PyLendingClub

Release 3.0.4.dev5

Feb 01, 2019

Contents

CHAPTER 1

Overview

docs	
tests	
package	

- Free software: BSD 3-Clause License

1.1 Installation

```
pip install pylendingclub
```

1.2 About

A Python based wrapper for Lending Club's API that enables easier programmatic use of the API. Also extends the functionality of the API through a higher-level wrapper for ease of use, and an AutoInvestor. More features to come.

See the API documentation here: https://www.lendingclub.com/developers/api-overview.

1.3 Getting Started

To get started, download the package with pip:

```
pip install pylendingclub
```

Once the package is installed, you will need a Session object. You can create one directly, by passing your api-key and investor-id.

```
from pylendingclub.session import LendingClubSession
session = LendingClubSession(api_key, investor_id)
```

Alternatively, you can create environment variables for both of these values. Make sure they are created as 'LC_API_KEY' and 'LC_INVESTOR_ID'.

With environment variables set, you can create a *Session* with them like so:

```
from pylendingclub import Session
session = LendingClubSession.from_environment_variables()
```

1.4 Using the Session Object

1.5 Sessions and Responses

Calls to the API through the *Session* will return a Response object. You can then work with this response as needed. If you just want the JSON data from the response, use the following syntax:

```
response = session.resource.property
json_data = response.json()
```

or

```
response = session.resource.method()
json_data = response.json()
```

You can also chain the *.json()* call directly onto the property, or method, but this won't allow you to handle an error with the response without making a separate call to get the original response. Especially when working with the *POST* methods, it is recommended to store the response separate from the JSON, but it is not required.

1.6 Accessing Resources

There are two primary resources available within the API. These are the *Account* and *Loan* resources. You can access them within the *Session* like so:

```
account = session.account
loan = session.loan
```

These two resources expose the sub-resources/services within the API. More on this below.

Remember, all of these services will return a 'Response'.

1.7 Account Resource

1.8 Account Summary

API Documentation: https://www.lendingclub.com/developers/summary

Method Type: GET

Syntax:

```
account_summary = session.account.summary
```

1.9 Available Cash

API Documentation: https://www.lendingclub.com/developers/available-cash

Method Type: GET

Syntax:

```
available_cash = session.account.available_cash
```

1.10 Notes

API Documentation: https://www.lendingclub.com/developers/notes-owned

Method Type: GET

Syntax:

```
notes = session.account.notes
```

1.11 Detailed Notes

API Documentation: https://www.lendingclub.com/developers/detailed-notes-owned

Method Type: GET

Syntax:

```
detailed_notes = session.account.detailed_notes
```

1.12 Portfolios Owned

API Documentation: https://www.lendingclub.com/developers/portfolios-owned

Method Type: GET

Syntax:

```
portfolios_owned = session.account.portfolios_owned
```

1.13 Filters

API Documentation: https://www.lendingclub.com/developers/filters

Method Type: GET

Syntax:

```
filters = session.account.filters
```

1.14 Create Portfolio

API Documentation: https://www.lendingclub.com/developers/create-portfolio

Method Type: POST

Syntax:

```
create_portfolio = session.account.create_portfolio(portfolio_name, [portfolio_
↪description])
```

1.15 Submit Orders

API Documentation: https://www.lendingclub.com/developers/submit-order

Note:

The orders must be a list of dicts in the format:

```
[
  {
    'loanId' : loan_id,
    'requestedAmount' : amount,
    'portfolioId' : portfolio_id
  }
]
```

Where *loanId* and *requestedAmount* are required, and *requestedAmount* must be a denomination of 25.

For example:

```
[
  {
    'loanId' : 1234,
    'requestedAmount' : 25,
  },
  {
    'loanId' : 1345,
    'requestedAmount' : 50,
    'portfolioId' : 12345
```

(continues on next page)

(continued from previous page)

```
  }
]
```

Method Type: POST

Syntax:

```
submit_orders = session.account.submit_orders(orders)
```

1.16 Submit Order

API Documentation: https://www.lendingclub.com/developers/submit-order

Method Type: POST

Note: The *requested_amount* must be a denomination of $25.00. For example, 25, 100, and 2000 are all accepted values but 26, 115, and 2010 are not.

Syntax:

```
submit_order = session.account.submit_order(loan_id, requested_amount, [portfolio_id])
```

1.17 Account/Funds

1.18 Pending Transfers

API Documentation: https://www.lendingclub.com/developers/pending-transfers

Method Type: GET

Syntax:

```
pending_transfers = session.account.funds.pending
```

1.19 Add

API Documentation: https://www.lendingclub.com/developers/add-funds

Method Type: POST

Notes:

The *transfer_frequency* argument must be one of *[LOAD_NOW, LOAD_ONCE, LOAD_WEEKLY, LOAD_BIWEEKLY, LOAD_ON_DAY_1_AND_16, LOAD_MONTHLY]*

The ‘start_date’ argument is required for recurring transfers, and for *LOAD_ONCE*.

Syntax:

```
add_funds = session.account.funds.add(amount, transfer_frequency, [start_date], [end_
↪date])
```

1.20 Withdraw

API Documentation: https://www.lendingclub.com/developers/add-funds

Method Type: POST

Syntax:

```
withdraw_funds = session.account.funds.withdraw(amount)
```

1.21 Cancel Transfer

API Documentation: https://www.lendingclub.com/developers/cancel-transfers

Method Type: POST

Syntax:

```
cancel_transfer = session.account.funds.cancel(transfer_id)
```

1.22 Loan Resource

1.23 Listed Loans

API Documentation: https://www.lendingclub.com/developers/listed-loans

Method Type: GET

Notes:

The *show_all* argument will determine whether all loans are shown, or only the loans from the most recent listing period are shown.

The *filter_id* argument, if provided, will only show loans matching the filter.

Syntax:

```
listed_loans = session.loan.listed_loans([filter_id], [show_all]=True)
```

CHAPTER 2

Installation

At the command line:

```
pip install pylendingclub
```

CHAPTER 3

Usage

To use PyLendingClub in a project:

```
import pylendingclub
```

CHAPTER 4

Reference

4.1 pylendingclub

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

5.1 Bug reports

When reporting a bug please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.2 Documentation improvements

PyLendingClub could always use more documentation, whether as part of the official PyLendingClub docs, in docstrings, or even on the web in blog posts, articles, and such.

5.3 Feature requests and feedback

The best way to send feedback is to file an issue at https://github.com/bbarney213/PyLendingClub/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that code contributions are welcome :)

5.4 Development

To set up *PyLendingClub* for local development:

1. Fork PyLendingClub (look for the "Fork" button).
2. Clone your fork locally:

   ```
   git clone git@github.com:your_name_here/PyLendingClub.git
   ```

3. Create a branch for local development:

   ```
   git checkout -b name-of-your-bugfix-or-feature
   ```

   Now you can make your changes locally.

4. When you're done making changes, run all the checks, doc builder and spell checker with tox one command:

   ```
   tox
   ```

5. Commit your changes and push your branch to GitHub:

   ```
   git add .
   git commit -m "Your detailed description of your changes."
   git push origin name-of-your-bugfix-or-feature
   ```

6. Submit a pull request through the GitHub website.

5.4.1 Pull Request Guidelines

If you need some code review or feedback while you're developing the code just make the pull request.

For merging, you should:

1. Include passing tests (run `tox`)[1].
2. Update documentation when there's new API, functionality etc.
3. Add a note to `CHANGELOG.rst` about the changes.
4. Add yourself to `AUTHORS.rst`.

5.4.2 Tips

To run a subset of tests:

```
tox -e envname -- pytest -k test_myfeature
```

To run all the test environments in *parallel* (you need to `pip install detox`):

```
detox
```

[1] If you don't have all the necessary python versions available locally you can rely on Travis - it will run the tests for each change you add in the pull request.

It will be slower though ...

CHAPTER 6

Authors

- Brandon Dean Barney - https://blog.ionelmc.ro

CHANGELOG

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index