
robin_stocks Documentation

Release 1.0.0

Joshua M Fernandes

May 22, 2023

Contents

1	User Guide	3
1.1	Introduction	3
1.1.1	Philosophy	4
1.1.2	License	5
1.2	Installing	5
1.2.1	Using Pip	6
1.2.2	Get The Source Code	6
1.3	Quick Start	6
1.3.1	Importing and Logging In	7
1.3.2	Building Profile and User Data	8
1.3.3	Buying and Selling	8
1.3.4	Finding Options	9
1.3.5	Working With Orders	9
1.3.6	Saving to CSV File	9
1.3.7	Using Option Spreads	10
1.4	Advanced Usage	10
1.4.1	Making Custom Get and Post Requests	10
1.5	Robinhood Functions	11
1.5.1	Sending Requests to API	11
1.5.2	Logging In and Out	11
1.5.3	Loading Profiles	11
1.5.4	Getting Stock Information	12
1.5.5	Getting Option Information	12
1.5.6	Getting Market Information	12
1.5.7	Getting Positions and Account Information	12
1.5.8	Placing and Cancelling Orders	12
1.5.9	Getting Crypto Information	12
1.5.10	Export Information	12
1.6	TD Ameritrade Functions	12
1.6.1	Sending Requests to API	12
1.6.2	Logging In and Authentication	12
1.6.3	Getting Stock Information	12
1.6.4	Placing and Cancelling Orders	12
1.6.5	Getting Account Information	13
1.6.6	Getting Market Information	13
1.7	Gemini Functions	13

1.7.1	Sending Requests to API	13
1.7.2	Logging In and Authentication	13
1.7.3	Getting Crypto Information	13
1.7.4	Placing and Cancelling Orders	13
1.7.5	Getting Account Information	13
1.8	Example Scripts	13
2	Indices and tables	15



This library aims to create simple to use functions to interact with the Robinhood API. This is a pure python interface and it requires Python 3. The purpose of this library is to allow people to make their own robo-investors or to view stock information in real time.

Note: These functions make real time calls to your Robinhood account. Unlike in the app, there are no warnings when you are about to buy, sell, or cancel an order. It is up to **YOU** to use these commands responsibly.

CHAPTER 1

User Guide

Below is the table of contents for Robin Stocks. Use it to find example code or to scroll through the list of all the callable functions.

1.1 Introduction



1.1.1 Philosophy

I've written the code in accordance with what I consider the best coding practices. Some of these are part of [PEP 20](#) standards and some are my own. They are as follows:

- Explicit is better than implicit

When writing code for this project, you want other developers to be able to follow all function calls. A lot of times in C++ it can be confusing when trying to figure out if a function is built-in, defined in the same file, defined in another object, or an alias for another function. In Python, it's a lot easier to see where a function comes from, but care must still be taken to make code as readable as possible. This is the reason why my code uses `import robin_stocks.module as module` instead of `from module import *`. This means that calls to a function from the module must be written as `module.function` instead of the simply `function`. When viewing the code, it's easy to see which functions come from which modules. However users do not have to explicitly call functions because of the following reason...

- Flat is better than nested

The `__init__.py` file contains an import of all the functions I want to be made public to the user. This allows the user to call `robin_stocks.function` for all functions. Without the imports, the user would have to call `robin_stocks.module.function` and be sure to use the correct module name every single time. This may seem contradictory to the first standard, but the difference is that whereas I (the developer) must make explicit calls, for the end user it is unnecessary.

- Three strikes and you refactor

If you find yourself copying and pasting the same code 3 or more times, then it means you should put that code in its own function. As an example of this, I created the `robin_stocks.helper.request_get()` function, and then provided input parameters to handle different use cases. This means that although functions I write may have very different logic for how they handle the get requests from Robinhood, none of this logic is contained in the functions themselves. It's all been abstracted away to a single function which means the code is easier to debug, easier to propagate changes, and easier to read.

- Type is in the name

A person should be able to look at the code and know the purpose of all the names they see. For this reason I have written names of functions as `snake_case`, the names of input parameters and local function variables as `camelCase`, the names of class names and enum names as `PascalCase`, and the names of global variables as `UPPER_SNAKE_CASE`.

In addition, the naming of each function is standardized in order to make searching for functions easier. Functions that load user account information begin with “load”, functions that place orders begin with “order”, functions that cancel orders begin with “cancel”, functions that query begin with “find”, and so on. If you are using a text editor/IDE with auto-complete (which I highly recommend!), then this naming convention makes it even easier to find the function you want. As long as you know what you want the function to do, then you know what word it starts with.

1.1.2 License

Copyright (c) 2018 Joshua M. Fernandes

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.

1.2 Installing



1.2.1 Using Pip

This is the simplest method. To install Robin Stocks globally or inside a virtual environment, open terminal and run the command:

```
$ pip install robin_stocks
```

1.2.2 Get The Source Code

If you prefer to install the source code directly, it can be found [here](https://github.com/jmfernandes/robin_stocks), or you can clone the repository using:

```
$ git clone https://github.com/jmfernandes/robin_stocks.git
```

Once the file has been downloaded or cloned, cd into the directory that contains the `setup.py` file and install using:

```
$ pip install .
```

1.3 Quick Start



1.3.1 Importing and Logging In

The first thing you will need to do is to import Robin Stocks by typing:

```
>>> import robin_stocks
```

`robin_stocks` will need to be added as a preface to every function call in the form of `robin_stocks.function`. If you don't want to have to type `robin_stocks` at the beginning of every call, then import Robin Stocks by typing

```
>>> from robin_stocks import *
```

Keep in mind that this method is not considered good practice as it obfuscates the distinction between Robin Stocks' functions and other functions. For the rest of the documentation, I will assume that Robin Stocks was imported as `import robin_stocks`.

Once you have imported Robin Stocks, you will need to login in order to store an authentication token.

Basic

```
>>> import robin_stocks.robinhood as r
>>> login = r.login(<username>, <password>)
```

You will be prompted for your MFA token if you have MFA enabled and choose to do the above basic example.

With MFA entered programmatically from Time-based One-Time Password (TOTP)

NOTE: to use this feature, you will have to sign into your robinhood account and turn on two factor authentication. Robinhood will ask you which two factor authorization app you want to use. Select "other". Robinhood will present you with an alphanumeric code. This code is what you will use for "My2factorAppHere" in the code below. Run the following code and put the resulting MFA code into the prompt on your robinhood app.

```
>>> import pyotp
>>> totp = pyotp.TOTP("My2factorAppHere").now()
>>> print("Current OTP:", totp)
```

Once you have entered the above MFA code (the totp variable that is printed out) into your Robinhood account, it will give you a backup code. Make sure you do not lose this code or you may be locked out of your account!!! You can also take the exact same “My2factorAppHere” from above and enter it into your phone’s authentication app, such as Google Authenticator. This will cause the exact same MFA code to be generated on your phone as well as your python code. This is important to do if you plan on being away from your computer and need to access your Robinhood account from your phone.

Now you should be able to login with the following code,

```
>>> import pyotp
>>> import robin_stocks.robinhood as r
>>> totp = pyotp.TOTP("My2factorAppHere").now()
>>> login = r.login('joshsmith@email.com', 'password', mfa_code=totp)
```

Not all of the functions contained in the module need the user to be authenticated. A lot of the functions contained in the modules ‘stocks’ and ‘options’ do not require authentication, but it’s still good practice to log into Robinhood at the start of each script.

1.3.2 Building Profile and User Data

The two most useful functions are `build_holdings` and `build_user_profile`. These condense information from several functions into a single dictionary. If you wanted to view all your stock holdings then type:

```
>>> my_stocks = robin_stocks.build_holdings()
>>> for key,value in my_stocks.items():
>>>     print(key,value)
```

1.3.3 Buying and Selling

Trading stocks, options, and crypto-currencies is one of the most powerful features of Robin Stocks. There is the ability to submit market orders, limit orders, and stop orders as long as Robinhood supports it. Here is a list of possible trades you can make

```
>>> #Buy 10 shares of Apple at market price
>>> robin_stocks.order_buy_market('AAPL',10)
>>> #Sell half a Bitcoin is price reaches 10,000
>>> robin_stocks.order_sell_crypto_limit('BTC',0.5,10000)
>>> #Buy $500 worth of Bitcoin
>>> robin_stocks.order_buy_crypto_by_price('BTC',500)
>>> #Buy 5 $150 May 1st, 2020 SPY puts if the price per contract is $1.00. Good until_
↪cancelled.
>>> robin_stocks.order_buy_option_limit('open','debit',1.00,'SPY',5,'2020-05-01',150,
↪'put','gtc')
```

Now let’s try a slightly more complex example. Let’s say you wanted to sell half your Tesla stock if it fell to 200.00. To do this you would type

```
>>> positions_data = robin_stocks.get_current_positions()
>>> ## Note: This for loop adds the stock ticker to every order, since Robinhood
>>> ## does not provide that information in the stock orders.
```

(continues on next page)

(continued from previous page)

```
>>> ## This process is very slow since it is making a GET request for each order.
>>> for item in positions_data:
>>>     item['symbol'] = robin_stocks.get_symbol_by_url(item['instrument'])
>>> TSLAData = [item for item in positions_data if item['symbol'] == 'TSLA']
>>> sellQuantity = float(TSLAData['quantity'])//2.0
>>> robin_stocks.order_sell_limit('TSLA',sellQuantity,200.00)
```

Also be aware that all the order functions default to ‘gtc’ or ‘good until cancelled’. To change this, pass one of the following in as the last parameter in the function: ‘gfd’(good for the day), ‘ioc’(immediate or cancel), or ‘opg’(execute at opening).

1.3.4 Finding Options

Manually clicking on stocks and viewing available options can be a chore. Especially, when you also want to view additional information like the greeks. Robin Stocks gives you the ability to view all the options for a specific expiration date by typing

```
>>> optionData = robin_stocks.find_options_for_list_of_stocks_by_expiration_date(['fb
↳','aapl','tsla','nflx'],
>>>                                     expirationDate='2018-11-16',optionType='call')
>>> for item in optionData:
>>>     print(' price -',item['strike_price'],' exp - ',item['expiration_date'],'_
↳symbol - ',
>>>           item['chain_symbol'],' delta - ',item['delta'],' theta - ',item['theta
↳'])
```

1.3.5 Working With Orders

You can also view all orders you have made. This includes filled orders, cancelled orders, and open orders. Stocks, options, and cryptocurrencies are separated into three different locations. For example, let’s say that you have some limit orders to buy and sell Bitcoin and those orders have yet to be filled. If you want to cancel all your limit sells, you would type

```
>>> positions_data = robin_stocks.get_all_open_crypto_orders()
>>> ## Note: Again we are adding symbol to our list of orders because Robinhood
>>> ## does not include this with the order information.
>>> for item in positions_data:
>>>     item['symbol'] = robin_stocks.get_crypto_quote_from_id(item['currency_pair_id
↳'], 'symbol')
>>> btcOrders = [item for item in positions_data if item['symbol'] == 'BTCUSD' and_
↳item['side'] == 'sell']
>>> for item in btcOrders:
>>>     robin_stocks.cancel_crypto_order(item['id'])
```

1.3.6 Saving to CSV File

Users can also export a list of all orders to a CSV file. There is a function for stocks and options. Each function takes a directory path and an optional filename. If no filename is provided, a date stamped filename will be generated. The directory path can be either absolute or relative. To save the file in the current directory, simply pass in “.” as the directory. Note that “.csv” is the only valid file extension. If it is missing it will be added, and any other file extension will be automatically changed. Below are example calls.


```
>>> # let's say that I am running code from C:/Users/josh/documents/  
>>> r.export_completed_stock_orders(".") # saves at C:/Users/josh/documents/stock_  
↳orders_Jun-28-2020.csv  
>>> r.export_completed_option_orders("../", "toplevel") # save at C:/Users/josh/  
↳toplevel.csv
```

1.3.7 Using Option Spreads

When viewing a spread in the robinhood app, it incorrectly identifies both legs as either “buy” or “sell” when closing a position. The “direction” has to reverse when you try to close a spread position.

I.e. direction=“credit” when “action”:“sell”,“effect”:“close”

in the case of a long call or put spread.

1.4 Advanced Usage



1.4.1 Making Custom Get and Post Requests

Robin Stocks depends on Requests which you are free to call and use yourself, or you could use it within the Robin Stocks framework by using `robin_stocks.helper.request_get()`, `robin_stocks.helper.request_post()`, `robin_stocks.helper.request_document()`, and `robin_stocks.helper.request_delete()`. For example, if you wanted to make your own get request to the option instruments API endpoint in order to get all calls you would type:

```
>>> url = 'https://api.robinhood.com/options/instruments/'
>>> payload = { 'type' : 'call' }
>>> robin_stocks.request_get(url, 'regular', payload)
```

Robinhood returns most data in the form:

```
{ 'previous' : None, 'results' : [], 'next' : None }
```

where ‘results’ is either a dictionary or a list of dictionaries. However, sometimes Robinhood returns the data in a different format. To compensate for this, I added the **dataType** parameter which defaults to return the entire dictionary listed above. There are four possible values for **dataType** and their uses are:

```
>>> robin_stocks.robinhood.request_get(url, 'regular')      # For when you want
>>>                                                         # the whole dictionary
>>>                                                         # to view 'next' or
>>>                                                         # 'previous' values.
>>>
>>> robin_stocks.robinhood.request_get(url, 'results')      # For when results contains a
>>>                                                         # list or single dictionary.
>>>
>>> robin_stocks.robinhood.request_get(url, 'pagination')  # For when results contains a
>>>                                                         # list, but you also want to
>>>                                                         # append any information in
>>>                                                         # 'next' to the list.
>>>
>>> robin_stocks.robinhood.request_get(url, 'indexzero')   # For when results is a list
>>>                                                         # of only one entry.
```

Also keep in mind that the results from the Robinhood API have been decoded using `.json()`. There are instances where the user does not want to decode the results (such as retrieving documents), so I added the `robin_stocks.helper.request_document()` function, which will always return the raw data, so there is no **dataType** parameter. `robin_stocks.helper.request_post()` is similar in that it only takes a url and payload parameter.

1.5 Robinhood Functions

Note: Even though the functions are written as `robin_stocks.module.function`, the module name is unimportant when calling a function. Simply use `robin_stocks.function` for all functions.

1.5.1 Sending Requests to API

1.5.2 Logging In and Out

1.5.3 Loading Profiles

1.5.4 Getting Stock Information

1.5.5 Getting Option Information

1.5.6 Getting Market Information

1.5.7 Getting Positions and Account Information

1.5.8 Placing and Cancelling Orders

1.5.9 Getting Crypto Information

1.5.10 Export Information

1.6 TD Ameritrade Functions

Note: Even though the functions are written as `robin_stocks.module.function`, the module name is unimportant when calling a function. Simply use `robin_stocks.function` for all functions.

1.6.1 Sending Requests to API

1.6.2 Logging In and Authentication

1.6.3 Getting Stock Information

1.6.4 Placing and Cancelling Orders

1.6.5 Getting Account Information

1.6.6 Getting Market Information

1.7 Gemini Functions

Note: Even though the functions are written as `robin_stocks.module.function`, the module name is unimportant when calling a function. Simply use `robin_stocks.function` for all functions.

1.7.1 Sending Requests to API

1.7.2 Logging In and Authentication

1.7.3 Getting Crypto Information

1.7.4 Placing and Cancelling Orders

1.7.5 Getting Account Information

1.8 Example Scripts



Example python scripts can be found at https://github.com/jmfernandes/robin_stocks

CHAPTER 2

Indices and tables

- `genindex`
- `modindex`
- `search`