Argyle

API Documentation

April 17, 2013

Contents

1 Module moira

MOIRA, the MOIRA Otto-matic Intelligent Reconniter of Assets, is an API for the Marketwatch Virtual Stock Exchange game.

Code is available on Github[1].

1.1 Functions

get_current_holdings(*token*, *game*, *s*=`<requests.sessions.Session object at 0x1ebb190>`)

Fetches and parses current holdings.

Parameters

`token`: Cookiejar returned by a call to `get_token`.

`game`: The *name* of the game (marketwatch.com/game/*XXXXXXX*).

Return Value

`Stock` data.

(type=Dict of `Stock` objects, keyed by id)

Warning: The stock price returned by a call to `get_current_holdings` is rounded to the nearest cent! This results in inaccuracies if you calculate things based on this number — don't. Use `stock_search` instead. Interestingly, Marketwatch itself never reports the full-precision stock price anywhere except in HTML attributes.

get_portfolio_data(*token*, *game*, *s*=`<requests.sessions.Session object at 0x1ebb910>`)

Grabs portfolio data.

Parameters

`token`: Cookiejar returned by `get_token`.

`game`: Game name (marketwatch.com/game/*XXXXXXX*)

Return Value

Portfolio data dictionary

(type=Dict with net_worth, overall_return_amount, overall_return_percent, daily_return_percent, purchasing_power, cash_left, cash_borrowed, short_reserve, rank, and time (last updated).)

Note: I probably won't be making this return a `Portfolio` object; it seems slightly redundant.

[1] http://github.com/brandonwu/moira

get_token(*username*, *password*, *returnsession*=`False`, *s*=`<requests.sessions.Session object at 0x1dd0950>`)

Issues a login request. The token returned by this function is required for all methods in this module.

Parameters

`username`: The marketwatch.com username (email).

`password`: The plaintext marketwatch.com password.

Return Value

Requests cookiejar containing authentication token.

Note: It's unknown what the expiry time for this token is - it is set to expire at end of session. It may be apt to request a new token daily, while the market is closed.

get_transaction_history(*token*, *game*, *s*=`<requests.sessions.Session object at 0x1ebb410>`)

DOES NOT FUNCTION YET: Downloads and parses the list of past transactions.

Parameters

`token`: Cookiejar returned by `get_token`.

`game`: The *name* of the game (marketwatch.com/game/*XXXXXXX*).

Return Value

A dict of all past transactions.

(type=Dict of `Trans` objects, keyed on an index (1, 2...).)

order(*token*, *game*, *type*, *id*, *amt*, *s*=`<requests.sessions.Session object at 0x1ebbb90>`)

Initiates a buy, sell, short, or cover order.

Parameters

`token`: Cookiejar returned by `get_token`.

`game`: Game name (marketwatch.com/game/*XXXXXXX*)

`id`: Security ID (not the ticker symbol). Obtain from `stock_search`

`amt`: Order amount.

`type`: Type of order - 'Sell', 'Buy', 'Short', or 'Cover'.

Return Value

Returns integer - 0 if success, nonzero if failure.

(type=integer)

Warning: If you have insufficient funds, the server will still respond that the order succeeded! Check the order and transaction list to make sure the order actually went through.

stock_search(*token*, *game*, *ticker*, *s*=`<requests.sessions.Session object at 0x1ebb690>`)

Queries Marketwatch for stock price and ID of a given ticker symbol.

Parameters

- `token`: Cookiejar returned by `get_token`.
- `game`: Game name (marketwatch.com/game/*XXXXXXX*).
- `ticker`: Ticker symbol of stock to query.

Return Value

Current stock price, stock id, and server time.

(type=Dict {'price':float, 'id':str, 'time':datetime object in EST}.)

Note: You must have joined a game in order to use this function.

1.2 Variables

Name	Description
__package__	**Value:** `None`
ch	**Value:** `<logging.StreamHandler object at 0x1d5fb10>`
fh	**Value:** `<logging.FileHandler object at 0x1dd0650>`
formatter	**Value:** `<logging.Formatter object at 0x1dd0890>`
from_zone	**Value:** `tzfile('/usr/share/zoneinfo/UTC')`
logger	**Value:** `<logging.Logger object at 0x1d5f050>`
to_zone	**Value:** `tzfile('/usr/share/zoneinfo/America/New_York')`

1.3 Class Portfolio

Stores portfolio data.

1.3.1 Methods

___init___(*self*, *time*, *cash*, *leverage*, *net_worth*, *purchasing_power*, *starting_cash*, *return_amt*, *rank*)

Parameters

`time:`	Last updated time (server time from HTTP headers).
`cash:`	Amount of *cash* (not purchasing power!) remaining.
`leverage:`	Amount available to borrow.
`net_worth:`	Sum of assets and liabilities.
`purchasing_power:`	Amount (credit + cash) available to buy.
`starting_cash:`	Cash amount provided at game start.
`return_amt:`	Dollar amount of returns over `starting_cash`.

1.4 Class Stock

Stores portfolio data for a single stock.

1.4.1 Methods

___init___(*self*, *id*, *ticker*, *security_type*, *current_price*, *shares*, *purchase_type*, *returns*)

Parameters

`id:`	Unique ID assigned by Marketwatch to each security.
`ticker:`	The ticker symbol of the stock.
`security_type:`	"ExchangeTradedFund" or "Stock"
`current_price:`	Current price per share, *rounded to the cent.*
`shares:`	Number of shares held.
`purchase_type:`	"Buy" or "Short"
`returns:`	Total return on your investment. @see See the warnings at `get_current_holdings` about price rounding.

1.5 Class Trans

Stores transaction data for a single transaction.

1.5.1 Methods

__init__(*self*, *ticker*, *order_time*, *trans_time*, *trans_type*, *trans_amt*, *exec_price*)

Parameters

`ticker:`	The ticker symbol of the security.
`order_time:`	The time the order was issued.
`trans_time:`	The time the order was executed.
`trans_type:`	"Buy", "Short", "Sell", or "Cover"
`trans_amt:`	Number of shares sold/purchased.
`exec_price:`	Price of security at time of order.

2 Module nukaquant

Nukaquant is a library for technical and quant analysis of stock data. It is intended to be used with its companion Marketwatch API library, moira.

2.1 Variables

Name	Description
__package__	**Value:** `None`

2.2 Class Bollinger

Calculates the high and low Bollinger bands for a data stream.

2.2.1 Methods

__init__(*self*, *mavg_obj*, *num_sd*=`2`)

Parameters

`mavg_obj`: A MovingAverage object containing the data.

get_bollinger(*self*)

Returns the high and low Bollinger bands.

Return Value

Tuple(lowband, midband, highband)

2.3 Class LocalExtrema

Attempts to find price pivot points over a given interval in a stream of data.

2.3.1 Methods

__init__(*self*, *auto_period*=`False`, *period*=`20`, *max_period*=`100`, *dec_threshold*=`0.05`)

Parameters

`auto_period:`	If true, this dynamically increases the period to fit price cycles.
`max_period:`	The max value that `auto_period` will increase the period to.
`period:`	Size of window for pivot point determination.
`dec_threshold:`	Amount of change to happen before the window is decreased. Values of 0.4-0.7 will work for volatile stocks.

add_value(*self*, *value*)

clear_data(*self*)

2.3.2 Instance Variables

Name	Description
data	Current window of data inputted
high	Predicted current high point
low	Predicted current low point
slope	Current price direction

2.4 Class MovingAverage

Calculates the moving average for a data stream.

2.4.1 Methods

__init__(*self*, *period*=`30`)

Parameters

`period`: The number of samples to average; if the actual number of samples provided is less than this, the mavg attribute will be the simple average.

add_value(*self*, *value*)

Adds a sample to the moving average calculation window.

Parameters

`value`: The numerical value of the sample to add.

2.4.2 Instance Variables

Name	Description
data	List of data inputted
mavg	The moving average of the data added with `add_value`.

2.5 Class OrderQueue

Trying this out. Don't use it yet.

2.5.1 Methods

__init__(*self*)

add_order(*self*, *position*, *type*, *amount*, *price*)

Adds an order to the OrderQueue.

Parameters

`position:`	When to execute the order ('high' or 'low')
`type:`	'Buy', 'Sell', 'Short', or 'Cover'.
`amount:`	Number of securities to order.

clear_orders(*self*)

get_latest_order(*self*, *position*)

2.5.2 Instance Variables

Name	Description
nextaction	When the next order is scheduled.

Index