

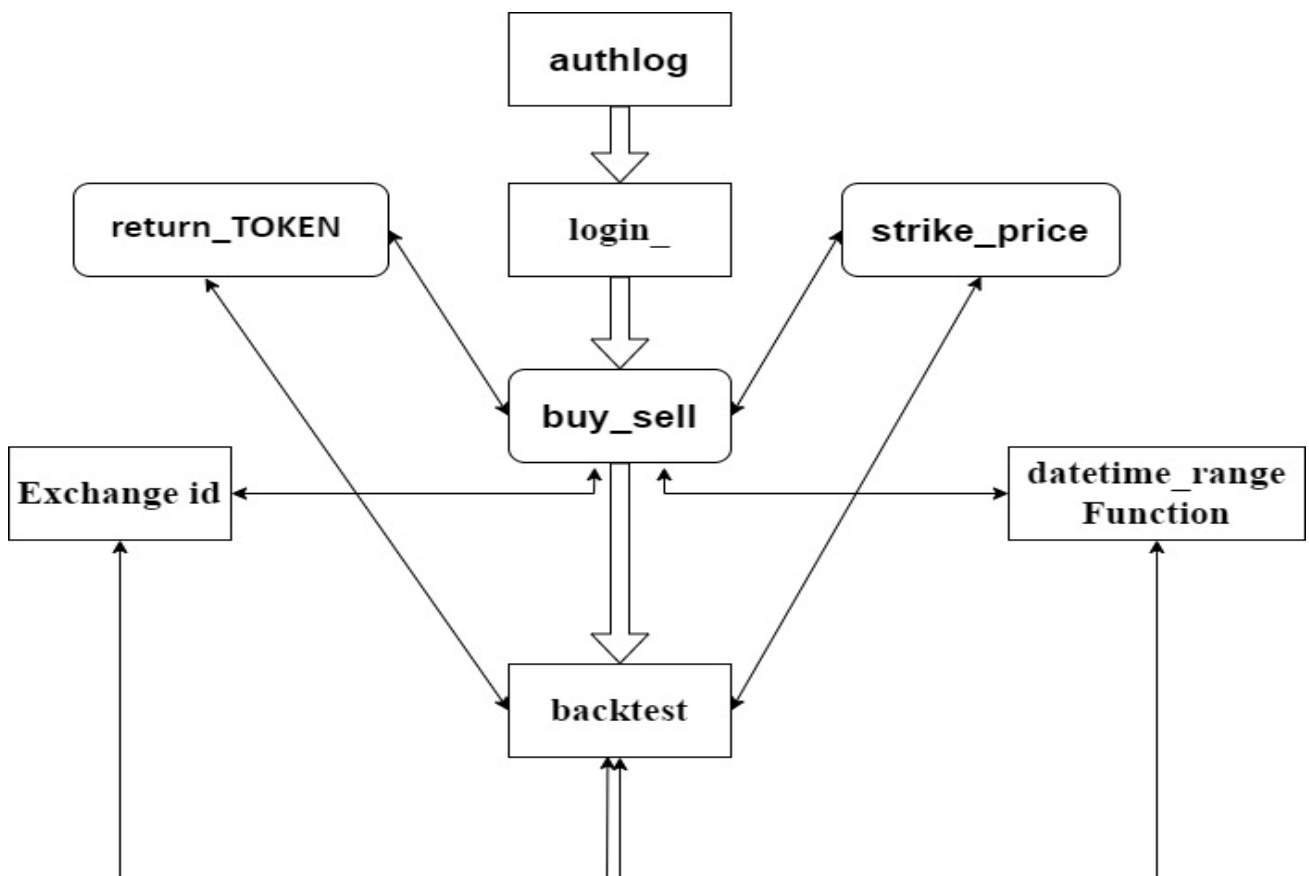
Packages Used:

1. pandas
2. numpy
3. py5paisa
4. mplfinance
5. matplotlib
6. datetime

Resources

1. <https://github.com/matplotlib/mplfinance>
2. <https://github.com/5paisa/py5paisa>

Flow Chart: -



User's credentials:

User can get this in the 5paisa trading website. And Datatype of all credentials are in string this includes {App name, app source, user id, password, user key, Encryption key: This will be in Dictionary} and {email id Password and DOB: this will be declared in function for login purpose through API's}


```
"APP_NAME": "5P56198212",  
"APP_SOURCE": "7312",  
"USER_ID": "bedaMQlrqg",  
"PASSWORD": "IBMfbkUlj65",  
"USER_KEY": "maNpe12f7OS212y31zGI8ToIV5zMgBWmousNA",  
"ENCRYPTION_KEY": "UEjdasrsadd2DVsdOda54tH5SYwp"
```

```
email="xyz@gmail.com", password of the user's 5paisa account ="xyz"  
date of birth {year/month/day} = "1234567"
```

So, for example this will look like this in function

```
def auth_  
    cred = {  
        "APP_NAME": "5P56798712",  
        "APP_SOURCE": "7311",  
        "USER_ID": "beTMrRQlrqg",  
        "PASSWORD": "IBMfbkUlj65",  
        "USER_KEY": "maNpeIRf7OS2yzGI8ToIV5zMgBWmouNA",  
        "ENCRYPTION_KEY": "UEjzxjrpwi2DVWfOa3GM4L454tH5SYwp"  
    }  
  
    client = FivePaisaClient(email="abcd@gmail.com", passwd="xxxxxxx",  
                             dob="11244119", cred=cred)  
  
    client.login()
```

How to get these login credentials aka API?

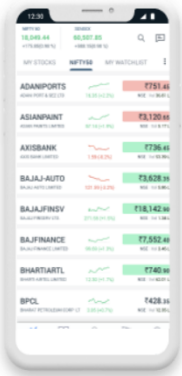


Company name / symbol

Help? **Login** Open Demat Account

Products ▾ Stocks Mutual Funds IPO News Blog FinSchool Queries ▾ English ▾

Nifty 17764.8 (-0.75%) ▾ | Sensex 59636.01 (-0.62%) ▾ | Nifty Bank 37976.2 (-0.17%) ▾ | Nifty IT 360



Stocks
Invest in individual companies that you believe in

Derivatives
Trade in Futures & Options with the best tools to be profitable

Commodities
Invest in commodities that power the world economies

Currencies
Trade in global currencies from the convenience of your couch!

Mutual Funds
Invest diversify and build wealth for long term! Pay 0% Commission

Online trading Made easy and rewarding!

0%* Brokerage Flat ₹20 per order

₹0 Commission Mutual Fund investments

Enter Name

Enter Mobile No

OPEN ACCOUNT NOW

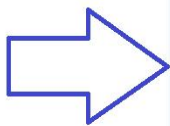
And start investing commission-free in just 5 mins. [Details](#)

Click on Login Button

Fill the Login details



Fill The Login Details.



Email / Client Code / Mobile

Password

Login

Don't have an account? [Register here.](#)
Forgot Password [click here](#)

Welcome to 5paisa

Simplify your investing experience with our advanced and user-friendly web-based online trading platform.

-  Powerful platform for lightning fast trading and easy trade execution
-  Intuitive design and single page view for complete details of a company.
-  Advanced charts and custom watchlist for informed investing decisions



Click On your profile and scroll down

Spaisa.com

Search/Add Scrip(min 3 characters required)

AVL MARGIN ₹17,737.06

NIFTY 50 17764.80 ▼-133.85 (-0.75%)

BANK NIFTY 37976.25 ▼-65.30 (-0.17%)

Switch Old Site Support KISHOR

Dashboard Watchlist Order & Position Portfolio Fund Ledger Reports Ideas Mutual Fund Market Analytics

NIFTY50 MY STOCKS NIFTY BANK

CPSEETF ₹29.72
CPSE ETF -0.39 (-1.30%) NSE BSE CASH

5m Alert Display Studies Views Events Share

CPSEETF 29.80 0.10 (0.34%)

+ Compare...

Plots

1D 1M 3M 1Y 5Y Max

Market Depth

Best Bid			Best Ask		
3 @ 29.60			100 @ 29.98		
Order	Price	Volume	Order	Price	Volume
3	29.60	3	1	29.98	100
2	29.30	2	1	29.99	100
1	29.20	1	1	30.00	1
1	29.15	1	2	30.02	11

Detail

Open	High	Low	Close
30.29	30.29	29.67	30.11

DPR 24.14 - 36.20

Volume 0

52 week High 34.40

Ask Spaisa

Click on your profile

After Scroll down you will see trading API click on it.

Spaisa.com

Search/Add Scrip(min 3 characters required)

AVL MARGIN ₹17,737.06

NIFTY 50 17764.80 ▼-133.85 (-0.75%)

BANK NIFTY 37976.25 ▼-65.30 (-0.17%)

Switch Old Site Support KISHOR

Dashboard Watchlist Order & Position Portfolio Fund Ledger Reports Ideas Mutual Fund Market Analytics

CPSEETF ₹29.72
CPSE ETF -0.39 (-1.30%) NSE BSE CASH

5m Alert Display Studies

CPSEETF 29.80 0.10 (0.34%)

+ Compare...

Plots

Click on Trading API

Market Depth

Best Bid			Best Ask		
3 @ 29.60			100 @ 29.98		
Order	Price	Volume	Order	Price	Volume
3	29.60	3	1	29.98	100
2	29.30	2	1	29.99	100
1	29.20	1	1	30.00	1
1	29.15	1	2	30.02	11

Detail

Open	High	Low	Close
30.29	30.29	29.67	30.11

DPR 24.14 - 36.20

Volume 0

52 week High 34.40

Ask Spaisa

US Investment Wealth

Research

Smallcase Sensibull Smart Investor Trader Smith

Streak

Utilities

Trading API Span Margin Calculator Market Status Approved Scrip

Logout

Click on get API keys



Dashboard Research Financial Products Reports Tools More

Developer APIs

Overview

Authorization

Customer Login

Market Feeds

Historical Data

Live Market Streaming

Order

Report

Fees

API Limits

Get API Keys

Downloads

ScripMaster

Python (SDK)

NodeJS (SDK)

Colang (SDK)

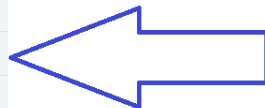
OVERVIEW

Use our APIs at no additional costs and integrate with your own trading tools.

You can execute orders in real time, monitor your positions, manage your portfolio and much more.

Our REST APIs can help you build a complete investment and trading platform in no time. Some key benefits:

- Build your own trading interface while using your 5paisa Trading account
- No additional costs
- Secure and easy to implement
- Flexible REST API compatible with multiple languages
- With SDK's for different programming languages like Python, Node JS and Colang



Click on Get API Keys

And Here you will find all the credential's, copy and paste it at same location.



Dashboard Research Financial Products Reports Tools More

Developer APIs

Overview

Authorization

Customer Login

Market Feeds

Historical Data

Live Market Streaming

Order

Report

Fees

API Limits

Get API Keys

Downloads

ScripMaster

Python (SDK)

NodeJS (SDK)



App Name

App Source

User ID

Password

User Key

Encryption Key

Valid up to

Please copy and store these details for future use.

In this Algo we take, at the money strike Price means if we get a buy single at particular value for e.g., 39650, let's say this is buying value of option so in order to get strike price at the money we use following user defined function which will return 39600 int value

```
def strike_price(snum):
    base = 100
    strike_f = base * round(snum/base)
    print(strike_f)
    return strike_f
```

Exchange id, this is a unique number provided by broker to track the order or trade status so that if we want to cancel the trade or modify the trade (for modifying stoploss especially) we can pass this as a parameter in there modify function i.e., `client.modify_order()`, whenever we place an order through API, by this function:- `client.place_order`, this returns a dictionary which have exchange id as blank or null value, cause broker don't provide exchange code immediately after placing a order but they do provide broker code or order id so with the help of order id we can get exchange id, (note: - there is only exchange id parameter to modify the order, we can't pass order id or broker id to track the order)

So, when we place an order with this `client.place_order()` function. It returns this dictionary

```
{'BrokerOrderID': 576977910, 'ClientCode': '56798712', 'Exch': 'N', 'ExchOrderID': 'O', 'ExchType': 'D',
'LocalOrderID': 0, 'Message': 'Order rejected by RMS. Quantity should be in multiple of 25',
'RMSResponseCode': -95, 'ScripInfo': '53963', 'Status': 1, 'Time': '/Date(1637260200000+0530)/'}
```

1.1

As you can see there is no exchange id, so in order to get it we have user defined function as `return_EXCHANGECODE` :-

```
def return_EXCHANGECODE(bidict):
    bidict1 = bidict['BrokerOrderID']
    jk = client.order_book() # module function (explanation for this function is below)
    for i in range(len(jk)):
        if str(jk[i]['BrokerOrderId']) == str(bidict1):
            print('exchange id ',jk[i]['ExchOrderID'])
            return jk[i]['ExchOrderID']
```

`client.order_book()` this is a module function which return list with dictionary elements for e.g., this is a list of two trade;

```
[{'AHProcess': 'N', 'AfterHours': 'N', 'AtMarket': 'N', 'BrokerOrderId': 576975686, 'BrokerOrderTime':
'/Date(1637305818340+0530)/', 'BuySell': 'B', 'DelvIntra': 'I', 'DisClosedQty': 1, 'Exch': 'N',
'ExchOrderID': 'O', 'ExchOrderTime': '/Date(3155130000000+0530)/', 'ExchType': 'D', 'MarketLot': 0,
'OldorderQty': 0, 'OrderRequesterCode': '56798712', 'OrderStatus': 'Rejected By 5P',
'OrderValidUpto': '19 Nov 2021', 'OrderValidity': 0, 'PendingQty': 1, 'Qty': 1, 'Rate': 1, 'Reason':
'Order rejected by RMS. Limit price should be within circuit limit(17.8 - 324.6)', 'RemoteOrderID': '1',
'RequestType': 'P', 'SLTriggerRate': 0, 'SLTriggered': 'N', 'SMOProfitRate': 0, 'SMOSLLimitRate': 0,
```

```
'SMOSLTriggerRate': 0, 'SMOTrailingSL': 0, 'ScripCode': 48594, 'ScripName': 'NIFTY 20 Jan 2022 PE 16600.00', 'TerminalId': 0, 'TradedQty': 0, 'WithSL': 'N'},
```

```
{'AHProcess': 'N', 'AfterHours': 'N', 'AtMarket': 'N', 'BrokerOrderId': 576977910, 'BrokerOrderTime': '/Date(1637308874377+0530)/', 'BuySell': 'B', 'DelvIntra': 'I', 'DisClosedQty': 1, 'Exch': 'N', 'ExchOrderID': '0', 'ExchOrderTime': '/Date(315513000000+0530)/', 'ExchType': 'D', 'MarketLot': 0, 'OldorderQty': 0, 'OrderRequesterCode': '56798712', 'OrderStatus': 'Rejected By 5P', 'OrderValidUpto': '19 Nov 2021', 'OrderValidity': 0, 'PendingQty': 1, 'Qty': 1, 'Rate': 1, 'Reason': 'Order rejected by RMS. Quantity should be in multiple of 25', 'RemoteOrderID': '1', 'RequestType': 'P', 'SLTriggerRate': 0, 'SLTriggered': 'N', 'SMOProfitRate': 0, 'SMOSLLimitRate': 0, 'SMOSLTriggerRate': 0, 'SMOTrailingSL': 0, 'ScripCode': 53963, 'ScripName': 'BANKNIFTY 25 Nov 2021 PE 38000.00', 'TerminalId': 0, 'TradedQty': 0, 'WithSL': 'N'}}
```

So the parameter of return_EXCHANGECODE is a dictionary, this parameter is nothing but the return of the client.place_order() (1.1) function so we are just equating a element of dictionary to another element of dictionary for e.g.,

element of client.place_order() dictionary will be 'BrokerOrderId': 576977910,

```
{'BrokerOrderId': 576977910, 'ClientCode': '56798712', 'Exch': 'N', 'ExchOrderID': '0', 'ExchType': 'D', 'LocalOrderID': 0, 'Message': 'Order rejected by RMS. Quantity should be in multiple of 25', 'RMSResponseCode': -95, 'ScripInfo': '53963', 'Status': 1, 'Time': '/Date(1637260200000+0530)/'}
```

And another element will be of client.order_book() that is 'BrokerOrderId': 576977910,

```
{'AHProcess': 'N', 'AfterHours': 'N', 'AtMarket': 'N', 'BrokerOrderId': 576977910, 'BrokerOrderTime': '/Date(1637308874377+0530)/', 'BuySell': 'B', 'DelvIntra': 'I', 'DisClosedQty': 1, 'Exch': 'N', 'ExchOrderID': '0', 'ExchOrderTime': '/Date(315513000000+0530)/', 'ExchType': 'D', 'MarketLot': 0, 'OldorderQty': 0, 'OrderRequesterCode': '56798712', 'OrderStatus': 'Rejected By 5P', 'OrderValidUpto': '19 Nov 2021', 'OrderValidity': 0, 'PendingQty': 1, 'Qty': 1, 'Rate': 1, 'Reason': 'Order rejected by RMS. Quantity should be in multiple of 25', 'RemoteOrderID': '1', 'RequestType': 'P', 'SLTriggerRate': 0, 'SLTriggered': 'N', 'SMOProfitRate': 0, 'SMOSLLimitRate': 0, 'SMOSLTriggerRate': 0, 'SMOTrailingSL': 0, 'ScripCode': 53963, 'ScripName': 'BANKNIFTY 25 Nov 2021 PE 38000.00', 'TerminalId': 0, 'TradedQty': 0, 'WithSL': 'N'}}
```

if str(jk[i]['BrokerOrderId']) == str(bidict1): if this condition gets satisfy then it well returns exchange id

As you can now clearly see both BrokerOrderId gets match then return_EXCHANGECODE function will return exchange id from order book (return jk[i]['ExchOrderID'])

return_TOKEN Function

This Is a user defined function (return token) which return token code aka script code and this script code is used as a parameter in client.place_order() function, script code is a code of particular contract in option's for e.g., Bank Nifty 18th NOV 39000. (note 39000 is a strike price) this whole thing is packed in 4–9 digit of code and that code is script code or scrip code, as you can see return token has a strike price as a parameter (which is strike_p). and pc is nothing but which contract or option type you want to buy, i.e., Put or call

```

def return_TOKEN(pc,strike_p):
    from datetime import date
    client = authlog()
    today = date.today()
    offset = (today.weekday() - 3)
    mdate = today - timedelta(days=offset)

    month={
        "01":'JAN',
        "02":'FEB',
        "03":'MAR',
        "04":'APR',
        "05":'MAY',
        "06":'JUN',
        "07":'JUL',
        "08":'AUG',
        "09":'SEP',
        "10":'OCT',
        "11":'NOV',
        "12":'DEC'
    }

    date = mdate.day
    iyear = mdate.year
    imon = mdate.month
    mon = month[f'{iyear}{imon}{date}'][4:6]

    symbol = "BANKNIFTY".upper()

    strike_f = "{:.2f}".format(float(strike_price(strike_p)))

    PE_CE = pc

    sym=f'{symbol} {25} {mon} {iyear} {PE_CE} {strike_f}'
    print(sym)
    day_year_mon = str(iyear) + str(imon)+str(date)
    print('dayyearmon:',day_year_mon)
    req=[{"Exch": "N", "ExchType": "D", "Symbol": sym, "Expiry": 20211125, "StrikePrice":strike_f,
"OptionType" : f'{PE_CE}']}
    res = client.fetch_market_feed(req)
    token=res['Data'][0]['Token']
    return token

```


login_function

So login_function is a user defined function and it takes day as a argument so that we can get a dataframe from client.historical_data() (module defined function), client.historical_data() takes 6 arguments in which 'N' stands for NSE, 'C' stands for Commodity or cash, '999920005' this is a script code, '5m' means time frame, e.g., 5 minute, 10 minute, 1 minute and so on, here m stands for minute, so last and second last argument means date range from to - where, , f'2021-{mon}-{dayf}', f'2021-{mon}-{dayf}' here mon is a global variable and dayf is a argument, this whole means we are taking one day data of 5 minute time frame of Nifty bank of NSE {National Stock Exchange}.

```
def login_(dayf):
    df1 = None
    try:
        global client
        client = authlog()
        df1 = client.historical_data('N', 'C', 999920005, '5m', f'2021-{mon}-{dayf}', f'2021-{mon}-{dayf}')
        print(df1)
    except Exception as e:
        print(e)
    return df1
```

here we return dataframe.

buy_sell(date) Funtion

We are going to calculate an initial buy and a sell signal. So this a user defined function that takes date as a argument to get dataframe from login_(dayf):, we are inheriting the functions.

```
def buy_sell(date)
    df = login_(date)
    < SOME CALCULATION >
    test_order = Order(order_type='S', exchange='N', exchange_segment='D', scrip_code=48595,
        quantity=25, price=0, is_intraday=True) # explanation for this is given down in bactest
    .
    ipsc = client.place_order(test_order)
    return df
```

after calculation and placing order It will return a dataframe.

def backtest(date) Function

After getting buy sell signals from buy_sell(date) function we again going to pass the dataframe to next user defined function which will put a stop loss and further signals. This is the last function of the code

```
def backtest(date):
    df = buy_sell(date)

    <SOME CALCULATION >

    test_order = Order(order_type='S', exchange='N', exchange_segment='D', scrip_code=48595,
                        quantity=25, price=0, is_intraday=True)

    ipsc = client.place_order(test_order)
```

so this is how we are going to place a order, Order() is a class, so order_type can be buy or sell represented by capital B or S respectively datatype as string, exchange means NSE OR BSE, N for NSE, exchange segment means derivative or equity so 'D' stands for derivative. Scrip code is contract code. quantity :Quantity to buy, price=0 means buy or sell at market price, is_intraday= true, means day trading and not a delivery or long term holding.

datetime_range Function

```
def datetime_range(start, end, delta):
    current = start
    while current < end:
        yield current
        current += delta
```

datetime_range Is a user defined function and it is used before placing order and it returns list of 5min interval, after calling a function through this way,

```
dts = [dt.strftime('%Y-%m-%dT%H:%M:00') for dt in
        datetime_range(datetime(yr1, mon, cday, 9), datetime.now(), # cday, mon, yr1 are global
                        timedelta(minutes=5))]                      # variable's mon=month,
                                                                    # yr1 = year, cday= current day
```

we get,

```
['2021-11-21T09:00:00', '2021-11-21T09:05:00', '2021-11-21T09:10:00', '2021-11-21T09:15:00',
'2021-11-21T09:20:00', '2021-11-21T09:25:00', '2021-11-21T09:30:00', '2021-11-21T09:35:00',
'2021-11-21T09:40:00', '2021-11-21T09:45:00', '2021-11-21T09:50:00', '2021-11-21T09:55:00',
'2021-11-21T10:00:00', '2021-11-21T10:05:00', '2021-11-21T10:10:00', '2021-11-21T10:15:00']
```

It will only return the element: - '2021-11-21T09:05:00', after 9:05 or at 9:05 so before 9:05 we will have list which will look like this: - ['2021-11-21T09:00:00'].

we are only interested in last second element of the list which we will be equating with dataframe index which is also a datetime index. We used this functions to avoid placing of order in for loop else it will place order each time we run the code. so this function is used in such a way that when we get signal to buy it will first check whether it is at current time or not by picking a element form index (we get index from dataframe) and equating with a last second element of returned list by function (datetime_range(start, end, delta)) so by this we are secured from placing random orders.

```
if df['Datetime'][i] == dts[-2]:
    test_order = Order(order_type='B', exchange='N', exchange_segment='D', scrip_code=48594,
                        quantity=25, price=0, is_intraday=True)
    icsc = client.place_order(test_order)
```

Summary

we first use authlog function to login and by using authlog function we authorize and get dataframe from login_ function, After that we use buy_sell function to generate and place order and for further signal we give that dataframe to another function which is backtest function and this function put a stoploss and order according to algorithm.