What is Scalping?

Scalping is a trading style that specializes in profiting off of small price changes and making a fast profit off reselling. In day trading, scalping is a term for a strategy to prioritize making high volumes off small profits.1

The idea is that it is easier to capture a small change in the price of an asset than to predict a larger move. To execute this strategy successfully, a trader needs to have a tight control on their entry and exits.

What are we building?

We are building an algorithmic trading bot that will try to capture small profits trading `ETH/USD` but do it quite often (a few times within an hour hopefully). The bot will use data over the last 10 minutes to decide the buying and selling prices for our limit orders.
We will use the following logic to place our trades:

1. If the current price of the asset is above the minimum price over the last 10 minutes, place a buy limit order at a price 0.2% above the minimum price. In code, this price is referenced as `buying_price`.
2. The above step is immediately followed by placing a limit sell order at a price 0.2% below the maximum price of the asset over the last 10 minutes. In code, this price is referenced as `selling_price`.
3. If we do not have any position on the asset we are looking to trade, have a buy order in place for the asset but the current price of the asset is above 0.5% of our estimated price we would have sold our asset at, then we cancel our existing limit buy order and close our position. This indicates that the market is trending upwards and we need to recalibrate our buying and selling prices.
4. If we do have a position, have a sell order in place for the asset but the current price of the asset is 0.5% below the buying price of the asset, we cancel our sell limit order and close our position.

Ideally, we would like only checks 1 and 2 to execute alternatively (buy and sell). We might also encounter a scenario where the current price of the asset never reaches the limit price of our order. In such cases, checks 3 and 4 help limit our losses.

Let's Build

Before getting started, you will need to create an Alpaca account to use the paper trading as well as fetch market data for `ETH/USD`. You can get one by signing up here.

```
from alpaca.data.historical import CryptoHistoricalDataClient
```

```
from alpaca.data.requests import CryptoBarsRequest, CryptoQuotesRequest,
CryptoTradesRequest
from alpaca.trading.requests import GetOrdersRequest
from alpaca.data.timeframe import TimeFrame
from alpaca.trading.client import TradingClient
from alpaca.trading.requests import MarketOrderRequest, LimitOrderRequest
from alpaca.trading.enums import OrderSide, TimeInForce, OrderStatus
from datetime import datetime
from dateutil.relativedelta import relativedelta
import json
import logging
import config
import asyncio

# ENABLE LOGGING - options, DEBUG,INFO, WARNING?
logging.basicConfig(level=logging.INFO,
                    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s')
logger = logging.getLogger(__name__)
```

We start by importing the necessary libraries including Alpaca-py. This is the latest official python SDK from Alpaca. It provides us with the necessary market data and trading endpoints. Also, we enable logging to monitor the latest prices and bot status.

```
# Alpaca Trading Client
trading_client = TradingClient(
    config.APCA_API_KEY_ID, config.APCA_API_SECRET_KEY, paper=True)

# Alpaca Market Data Client
data_client = CryptoHistoricalDataClient()

# Trading variables
trading_pair = 'ETH/USD'
notional_size = 20000
spread = 0.00
total_fees = 0
buying_price, selling_price = 0.00, 0.00
buy_order, sell_order = None, None
current_price = 0.00
client_order_str = 'scalping'

# Wait time between each bar request
waitTime = 60

# Time range for the latest bar data
time_diff = 5

# Current position of the trading pair on Alpaca
current_position = 0

# Threshold percentage to cut losses (0.5%)
cut_loss_threshold = 0.005

# Alpaca trading fee
trading_fee = 0.003
```

Next, we define our trading client, market data client and trading variables we will use throughout the bot. Trading client needs Alpaca's secrets as inputs while Market data does not.

```
async def main():
    '''
    Main function to get latest asset data and check possible trade conditions
    '''

    # closes all position AND also cancels all open orders
    trading_client.close_all_positions(cancel_orders=True)
    logger.info("Closed all positions")

    while True:
        l1 = loop.create_task(get_crypto_bar_data(
            trading_pair))
        # Wait for the tasks to finish
        await asyncio.wait([l1])
        # Check if any trading condition is met
        await check_condition()
        # Wait for the a certain amount of time between each bar request
        await asyncio.sleep(waitTime)
```

In the main function we define our tasks using `asyncio` event loops. These tasks are defined to do 2 main things:

1. Get latest bar data for our trading pair and calculate possible buying/selling prices
2. Check for possible trade scenarios using the prices from the previous step.

In the snippet above, we are repeating the two steps every `waitTime` amount of seconds. We have defined it as 60 seconds for now but this can be tweaked further to optimize results.

```
async def get_crypto_bar_data(trading_pair):
    '''
    Get Crypto Bar Data from Alpaca for the last diff minutes
    '''
    time_diff = datetime.now() - relativedelta(minutes=diff)
    logger.info("Getting crypto bar data for {0} from {1}".format(
        trading_pair, time_diff))
    # Defining Bar data request parameters
    request_params = CryptoBarsRequest(
        symbol_or_symbols=[trading_pair],
        timeframe=TimeFrame.Minute,
        start=time_diff
    )
    # Get the bar data from Alpaca
    bars_df = data_client.get_crypto_bars(request_params).df
    # Calculate the order prices
    global buying_price, selling_price, current_position
    buying_price, selling_price = calc_order_prices(bars_df)

    if len(get_positions()) > 0:
        print(get_positions())
        current_position = get_positions()[0]['qty']
```

```
    buy_order = False
else:
    sell_order = False
return bars_df
```