海龟交易量化

量化需要的步骤：

明确策略，以及自己的交易逻辑
根据策略，明确自己的交易信号有哪些（你需要哪一些量）
根据交易信号，以及开源库，获取需要的数据
将策略用程序描述出来

海龟交易法则量化

策略：

买入：

价格超越**20日最高点**的一个最小单位则做多
价格跌破**20日最低点**的一个最小单位则做空

退出：

对于多头头寸系统而言，价格跌破过去**10日最低点**时退出。
对空头头寸，价格超过**10日最高点**时退出

止损： 超过或跌破价格的**2N**时止损

N = (19*PDN +TR)/20
N代表的是平均真实波动值

从而我们需要的**数据**有：

20日最高点，20日最低点，10日最低点，10日最高点，N

我们得将我们的想法用程序表达出来：

1. 创建一个class turtle
2. _init_ function always goes with class, 所以我们创建一个 init function初始化所有需要的数据：

交易账号
合约代码
唐奇安通道的天数周期（买入）
唐奇安通道天数周期（止盈）
计算atr的天数
最高风险度
N值
买卖单位
唐奇安通道上轨

唐奇安通道下轨

```python
class Turtle:
    #Initializing needed data 初始化需要的数据
    def __init__(self, symbol, account = "cgdjustin", donchian_channel_open_position = 20,donchian_channel_stop_profit=10, a
        self.account = account #交易账号
        self.symbol = symbol #合约代码
        self.donchian_channel_open_position = donchian_channel_open_position #唐奇安通道的天数周期（买入）
        self.donchian_channel_stop_profit = donchian_channel_stop_profit #唐奇安通道天数周期（止盈）
        self.atr_day_length =  atr_day_length #计算atr的天数
        self.max_risk_ratio = max_risk_ratio  # 最高风险度
        self.state = {
            "position": 0,  # 本策略净持仓数(正数表示多头，负数表示空头，0表示空仓)
            "last_price": float("nan"),  # 上次调仓价
        }
        self.n = 0 #N值
        self.unit = 0 #买卖单位
        self.donchian_channel_high = 0 #唐奇安通道上轨
        self.donchian_channel_low = 0 #唐奇安下轨

        #从库中获取数据
        self.api = TqApi(self.account)  #账号数据
        self.quote = self.api.get_quote(self.symbol)  #行情情况
        kline_length = max(donchian_channel_open_position + 1, donchian_channel_stop_profit + 1, atr_day_length * 5) #k线的根
        self.klines = self.api.get_kline_serial(self.symbol, 24 * 60 * 60, data_length=kline_length)  #指定合约及周期的K线数据.
        self.account = self.api.get_account()  #获取用户账户资金问题
        self.target_pos = TargetPosTask(self.api, self.symbol)  #创建目标持仓task实例，负责调整归属于该task的持仓
```

3. line 52: 通过使用tqsdk library，我们得到需要计算的参数：

```python
51      #交易信号的计算
52      def recalc_parameter(self):
53
54          #平均真实波动值 （使用库中的ATR函数，运用kline当成参数，使用需要的数据）
55          self.n = ATR(self.klines, self.atr_day_length)
56
57          #计算头寸规模单位 unit = 1% of the account/ 市场的绝对波动幅度
58          self.unit = int((self.account.balance * 0.01)/(self.quote.volume_multiple * self.n))
59
60          #唐奇安通道上轨：  前N个交易日的最高价，当价格突破此价格时则买入
61          self.donchian_channel_high = max(self.klines.high[-self.donchian_channel_open_position - 1:-1])
62
63          # #唐奇安通道下轨：  前N个交易日的最低价，当价格突破此价格时则做空
64          self.donchian_channel_low = min(self.klines.low[-self.donchian_channel_open_position - 1:-1])
65
66          print("唐其安通道上下轨: %f, %f" % (self.donchian_channel_high, self.donchian_channel_low))
67
68          return True
```

4. 设置持仓数，参数pos将为持仓数。其中还会同时update最新价格

```python
#设置持仓数
def set_position(self, pos):
    self.state["position"] = pos
    self.state["last_price"] = self.quote["last_price"]
    self.target_pos.set_target_volume(self.state["position"])
```

5. 入市策略

```
def try_open(self):
    #开仓策略:入市策略：  价格超越 20 日突破为基础的短期系统.只要价格超越20日最高或最低点的一个最小单位，则买入或做空
    while self.state["position"] == 0 :
        self.api.wait_update()

        # 如果产生新k线,则重新计算唐奇安通道及买卖单位
        if self.api.is_changing(self.klines.iloc[-1],"datetime"):
            self.recalc_parameter
        if self.api.is_changing(self.quote, "last_price"):
            print("最新价: %f" % self.quote.last_price)
            if self.quote.last_price > self.donchian_channel_high:  # 当前价>唐奇安通道上轨，买入1个Unit；(持多仓)
                print("当前价>唐奇安通道上轨，买入1个Unit(持多仓): %d 手" % self.unit)
                self.set_position(self.state["position"] + self.unit)
            elif self.quote.last_price < self.donchian_channel_low:  # 当前价<唐奇安通道下轨，卖出1个Unit；(持空仓)
                print("当前价<唐奇安通道下轨，卖出1个Unit(持空仓): %d 手" % self.unit)
                self.set_position(self.state["position"] - self.unit)
```

6. 交易策略

加仓，止损，止盈

```
    #交易策略
    # 退出：  对于多头头寸系统而言，系统1在价格跌破过去10日最低点时退出。对空头头寸，价格超过10日最高点时退出
def try_close(self):
    while self.state["position"] != 0:
        self.api.wait_update()
        if self.api.is_changing(self.quote,"last_price"):
            print("最新价: ", self.quote.last_price)

            #做多海龟策略
            if self.state["position"] > 0:  #持多单
                # 加仓策略：如果是多仓且行情最新价在上一次建仓（或者加仓）的基础上又上涨了0.5N，就再加一个Unit的多仓,并且风险度在设定范围内
                if self.quote.last_price >= self.state["last_price"] + 0.5 * self.n and self.account.risk_ratio <= self.
                    print("加仓:加1个Unit的多仓")
                    self.set_position(self.state["position"] + self.unit)
                #止损策略：  如果是多仓且行情最新价在上一次建仓的基础下又跌了2N，就卖出全部头寸止损
                elif self.quote.last_price <= self.state["last_price"] - 2*self.n:
                    print("止损:卖出全部头寸")
                    self.set_position(0)
                # 止盈策略：如果是多仓且行情最新价跌破了10日唐奇安通道的下轨，就清空所有头寸结束策略,离场
                if self.quote.last_price <= min(self.klines.low[-self.donchian_channel_stop_profit - 1:-1]):
                    print("止盈:清空所有头寸结束策略,离场")
                    self.set_position(0)

            #做空海龟策略
            elif self.state["position"] < 0:  # 持空单
                # 加仓策略：如果是空仓且行情最新价在上一次建仓（或者加仓）的基础上又下跌了0.5N，就再加一个Unit的空仓,并且风险度在设定范围内
                if self.quote.last_price <= self.state["last_price"] - 0.5 * self.n and self.account.risk_ratio <= self.
                    print("加仓:加1个Unit的空仓")
                    self.set_position(self.state["position"] - self.unit)
                # 止损策略：如果是空仓且行情最新价在上一次建仓（或者加仓）的基础上又上涨了2N，就平仓止损
                elif self.quote.last_price >= self.state["last_price"] + 2 * self.n:
                    print("止损:卖出全部头寸")
                    self.set_position(0)
                # 止盈策略：如果是空仓且行情最新价升破了10日唐奇安通道的上轨，就清空所有头寸结束策略,离场
                if self.quote.last_price >= max(self.klines.high[-self.donchian_channel_stop_profit - 1:-1]):
                    print("止盈:清空所有头寸结束策略,离场")
                    self.set_position(0)
```

Needed Python
Python classed and objects

The _init_() function

- All classes have a function called __init__(), which is always executed **when the class is being initiated.**
- Use the __init__() function to assign values to object properties, or other operations that are necessary to do when the object is being created: