

Python与数据科学导论——大作业

量化回测系统与股票策略设计

回测系统：基于backtrader

- Backtrader是基于python语言的一个量化投资框架，可以用于各种资产的回测。
- 在本次作业中，库文件位于backtesting文件夹中，使用时直接import backtesting即可
- Backtrader的特点：

The platform has 2 main objectives:

1. Ease of use
2. Go back to 1

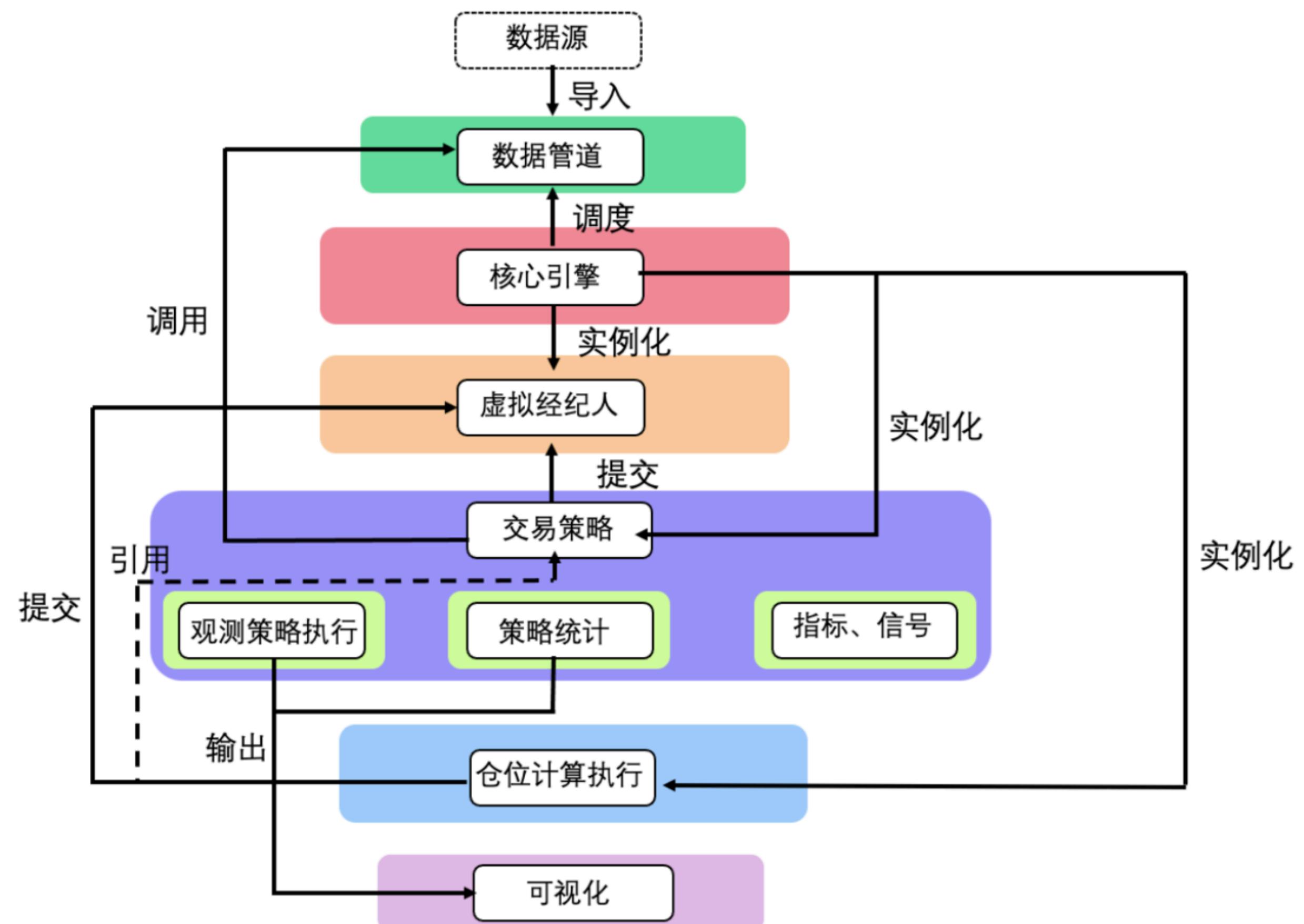
回测系统：基于backtrader

- 回测系统的使用--backtrader库的使用方法
 - 官方文档：<https://www.backtrader.com/docu/>
 - 非官方教程（仅供参考）：<https://blog.csdn.net/Castlehe/article/details/113378863>
 - 由于backtrader不支持matplotlib新版本，所以安装matplotlib==3.2.2的版本

回测系统：基于backtrader

- 基本架构：大脑+策略+数据+配套—模拟交易所的组织形式

系统基本架构



回测系统：基于backtrader

- 回测系统的使用--backtrader库的基本组成部分
 - Data Feeds 数据。资产的数据信息，例如股票K线信息，基本面信息。数据需要经过处理，采取一定的标准格式。
 - Strategy 、Cerebro是我们在使用这个库的时候主要操作的对象，对应了策略的设计和整体模拟的“大脑”。
 - Indicators 指标，某种判断形态。既可以是技术形态也可以是基本面。比如20日移动平均线和每股收益。
 - Analyzers 分析，交易结束之后需要做评价，比如收益率，风险收益等等。Analyzers内置了大量的金融分析模块。
 - Orders 订单。一般在策略使用的订单模块，进行买卖操作
 - Sizers 仓位。
 - Broker 代理人。证券交易商提供的接口
 - observers 观察者。观察回测中的状态
 - Plotting 绘制。绘制图形

回测系统：基于backtrader

- 关键概念：Line
- Line的本意就是一连串可以连接在一起的点，所有可以在坐标上形成一条线的数据，就称为Line。在量化投资的领域，这个Line通常指的是证券的open(开盘价)、high (最高价)、low (最低价)、close (收盘价)、volume (成交量)。例如如下我们下载的某股票的数据：

B	Line1C	D	E	F	Line5G
date	open	high	low	close	volume
1999-11-10 00:00:00	29.5	29.8	27	27.75	174085100
1999-11-11 00:00:00	27.58	28.38	27.53	27.71	29403500
1999-11-12 00:00:00	27.86	28.3	27.77	28.05	15008000
1999-11-15 00:00:00	28.2	28.25	27.7	27.75	11921100
1999-11-16 00:00:00	27.88	27.97	26.48	26.55	23223100
1999-11-17 00:00:00	26.5	27.18	26.37	27.18	10052600
1999-11-18 00:00:00	27.2	27.58	26.78	27.02	8446500
1999-11-19 00:00:00	27.5	27.53	26.8	26.88	5375000
1999-11-22 00:00:00	26.88	26.95	26.3	26.45	5535400
1999-11-23 00:00:00	26.45	26.55	26.1	26.45	3844000
1999-11-24 00:00:00	26.44	26.55	26.01	26.43	4098000
1999-11-25 00:00:00	26.3	26.66	26.02	26.4	5725300
1999-11-26 00:00:00	26.42	26.66	26.15	26.45	5725300

回测系统：基于backtrader

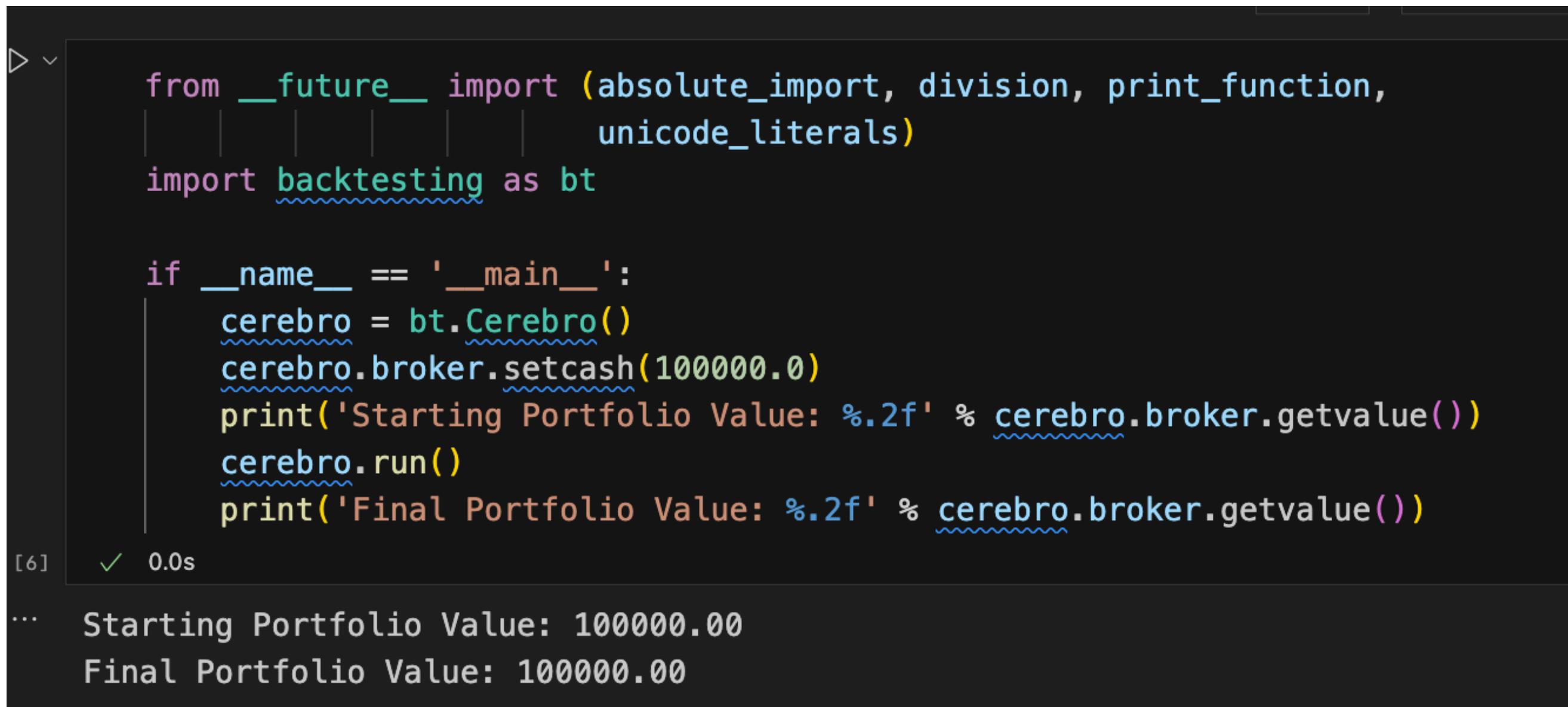
- 关键概念：Index=0
- Backtrader系统中，对Line的数据是逐行处理的，如下：
- 特别值得注意的是-1在python中用于访问一个列表的最后一个数据，而在backtrader中，-1指的是最后已经处理过得数据，在当前处理数据之前，值会随着系统的处理而不断变化。
- Eg：访问移动平均在当前时间点的值

```
self.sma = SimpleMovingAverage(.....)  
  
av = self.sma[0]
```

open	系统 处理 顺序
29.5	
27.58	
27.86	
28.2	-2 →
27.88	-1 →
26.5	0 →
27.2	1 →
27.5	
26.88	

回测系统：基于backtrader

- 创建第一个程序



```
from __future__ import (absolute_import, division, print_function,
                        unicode_literals)
import backtesting as bt

if __name__ == '__main__':
    cerebro = bt.Cerebro()
    cerebro.broker.setcash(100000.0)
    print('Starting Portfolio Value: %.2f' % cerebro.broker.getvalue())
    cerebro.run()
    print('Final Portfolio Value: %.2f' % cerebro.broker.getvalue())
[6]    ✓ 0.0s
...
Starting Portfolio Value: 100000.00
Final Portfolio Value: 100000.00
```

- 将Backtrader引入到程序中，命名为bt。
- 创建了一个机器人大脑（Cerebro），同时隐含创建了一个borker（券商）。
- 让机器人大脑开始运行。
- 显示了机器人在券商那里存有多少钱。

回测系统：基于backtrader

- 给空白的引擎增加数据

```
import backtesting as bt
import pandas as pd
from datetime import datetime
if __name__ == '__main__':
    cerebro = bt.Cerebro()
    #获取数据
    stock_hfq_df = pd.read_excel("./data/sh600000.xlsx", index_col='date', parse_dates=True)
    start_date = datetime(2010, 9, 30) # 回测开始时间
    end_date = datetime(2021, 9, 30) # 回测结束时间
    data = bt.feeds.PandasData(dataname=stock_hfq_df, fromdate=start_date, todate=end_date) # 加载数据
    cerebro.adddata(data) # 将数据传入回测系统

    cerebro.broker.setcash(100000.0)
    print('Starting Portfolio Value: %.2f' % cerebro.broker.getvalue())
    cerebro.run()
    print('Final Portfolio Value: %.2f' % cerebro.broker.getvalue())
```

Starting Portfolio Value: 100000.00
Final Portfolio Value: 100000.00

回测系统：基于backtrader

- 给引擎第一个策略

```
import backtesting as bt
import pandas as pd
from datetime import datetime

# 创建一个策略
class TestStrategy(bt.Strategy):

    def log(self, txt, dt=None):
        ''' 提供记录功能'''
        dt = dt or self.datas[0].datetime.date(0)
        print('%s, %s' % (dt.isoformat(), txt))

    def __init__(self):
        # 引用到输入数据的close价格
        self.dataclose = self.datas[0].close

    def next(self):
        # 目前的策略就是简单显示下收盘价。
        self.log('Close, %.2f' % self.dataclose[0])

if __name__ == '__main__':
    cerebro = bt.Cerebro()

    # 增加一个策略
    cerebro.addstrategy(TestStrategy)

    #获取数据
    stock_hfq_df = pd.read_excel("./data/sh600000.xlsx", index_col='date', parse_dates=True)
    start_date = datetime(2010, 9, 30) # 回测开始时间
    end_date = datetime(2021, 9, 30) # 回测结束时间
    data = bt.feeds.PandasData(dataname=stock_hfq_df, fromdate=start_date, todate=end_date) # 加载数据
    cerebro.adddata(data) # 将数据传入回测系统

    cerebro.broker.setcash(100000.0)
    print('Starting Portfolio Value: %.2f' % cerebro.broker.getvalue())

    cerebro.run()

    print('Final Portfolio Value: %.2f' % cerebro.broker.getvalue())
```

```
Starting Portfolio Value: 100000.00
2020-09-30, Close, 125.97
...(省略n行)
2021-09-27, Close, 127.11
2021-09-28, Close, 127.26
2021-09-29, Close, 127.11
2021-09-30, Close, 126.83
Final Portfolio Value: 100000.00
```

回测系统：基于backtrader

- 加入买的逻辑到Strategy中

```
import backtrader as bt
import pandas as pd
from datetime import datetime

# 创建一个测试策略
class TestStrategy(bt.Strategy):

    def log(self, txt, dt=None):
        ''' 记录策略信息 '''
        dt = dt or self.datas[0].datetime.date(0)
        print('%s, %s' % (dt.isoformat(), txt))

    def __init__(self):
        # 应用第一个数据源的收盘价
        self.dataclose = self.datas[0].close

    def next(self):
        # 打印每日的收盘价
        self.log('Close, %.2f' % self.dataclose[0])

        if self.dataclose[0] < self.dataclose[-1]:
            # 当前的价格比上一次价格（也就是昨天的价格）低

            if self.dataclose[-1] < self.dataclose[-2]:
                # 上一次的价格（昨天）比上上一次的价格（前天的价格）低

                # 开始买！
                self.log('BUY CREATE, %.2f' % self.dataclose[0])
                self.buy()

    def stop(self):
        self.log('STOPPING')
```

Starting Portfolio Value: 100000.00
...(省略n行)
2021-09-28, Close, 127.26
2021-09-29, Close, 127.11
2021-09-30, Close, 126.83 2021-09-30,
BUY CREATE, 126.83
Final Portfolio Value: 110441.06

回测系统：基于backtrader

- 加入卖的逻辑到Strategy中

```
# 创建一个策略
class TestStrategy(bt.Strategy):

    def log(self, txt, dt=None):
        ''' 记录功能 '''
        dt = dt or self.datas[0].datetime.date(0)
        print('%.2f, %s' % (dt.isoformat(), txt))

    def __init__(self):
        # 引用到数据的close Line
        self.dataclose = self.datas[0].close

        # 跟踪订单
        self.order = None

    def notify_order(self, order):
        if order.status in [order.Submitted, order.Accepted]:
            # 订单提交和成交当前不做处理
            return

        # 检查订单是否成交
        # 注意，如果现金不够的话，订单会被拒绝
        if order.status in [order.Completed]:
            if order.isbuy():
                self.log('BUY EXECUTED, %.2f' % order.executed.price)
            elif order.issell():
                self.log('SELL EXECUTED, %.2f' % order.executed.price)

            self.bar_executed = len(self)

        elif order.status in [order.Canceled, order.Margin, order.Rejected]:
            self.log('Order Canceled/Margin/Rejected')

        # 记录没有挂起的订单
        self.order = None

    def next(self):
        #记录close的价格
        self.log('Close, %.2f' % self.dataclose[0])

        # 检查是否有挂起的订单，如果有的话，不能再发起一个订单
        if self.order:
            return

        # 检查是否在市场（有持仓）
        if not self.position:

            # 不在，那么连续3天价格下跌就买点
            if self.dataclose[0] < self.dataclose[-1]:
                # 当前价格比上一次低

                if self.dataclose[-1] < self.dataclose[-2]:
                    # 上一次的价格比上上次低

                    # 买入!!!
                    self.log('BUY CREATE, %.2f' % self.dataclose[0])

            # Keep track of the created order to avoid a 2nd order
            self.order = self.buy()

        else:

            # 已经在市场，5天后就卖掉。
            if len(self) >= (self.bar_executed + 5):
                # SELL, SELL, SELL!!! (with all possible default parameters)
                self.log('SELL CREATE, %.2f' % self.dataclose[0])

            # Keep track of the created order to avoid a 2nd order
            self.order = self.sell()
```

回测系统：基于backtrader

- 增加佣金

```
if __name__ == '__main__':
    cerebro = bt.Cerebro()

    # 增加一个策略
    cerebro.addstrategy(TestStrategy)

    # 获取数据
    stock_hfq_df = pd.read_excel("./data/sh600000.xlsx", index_col='date', parse_dates=True)
    start_date = datetime(2010, 9, 30) # 回测开始时间
    end_date = datetime(2021, 9, 30) # 回测结束时间
    data = bt.feeds.PandasData(dataname=stock_hfq_df, fromdate=start_date, todate=end_date) # 加载数据
    cerebro.adddata(data) # 将数据传入回测系统

    cerebro.broker.setcash(100000.0)
    # 设置佣金0.1% ... 除以100去掉%号。
    cerebro.broker.setcommission(commission=0.001)
    print('Starting Portfolio Value: %.2f' % cerebro.broker.getvalue())
    cerebro.run()
    print('Final Portfolio Value: %.2f' % cerebro.broker.getvalue())
```

```
Starting Portfolio Value: 100000.00 ...(省略n行)
2021-09-14, BUY CREATE, 129.79
2021-09-15, BUY EXECUTED, Price: 129.79, Cost: 129.79, Comm 0.13
2021-09-15, Close, 129.51
2021-09-16, Close, 128.52
2021-09-17, Close, 128.38
2021-09-22, Close, 127.26
2021-09-23, Close, 127.26
2021-09-24, Close, 127.11
2021-09-24, SELL CREATE, 127.11
2021-09-27, SELL EXECUTED, Price: 127.11, Cost: 129.79, Comm 0.13
2021-09-27, OPERATION PROFIT, GROSS -2.68, NET -2.94
2021-09-27, Close, 127.11 2021-09-28, Close, 127.26 2021-09-29, Close, 127.11
2021-09-30, Close, 126.83
2021-09-30, BUY CREATE, 126.83
Final Portfolio Value: 99992.00
```

回测系统：基于backtrader

- 增加一个指标

```
#创建一个策略
class TestStrategy(bt.Strategy):
    params = (
        ('maperiod', 20),
    )
    #记录功能
    def log(self, txt, dt=None):
        ''' Logging function fot this strategy'''
        dt = dt or self.datas[0].datetime.date(0)
        print('%s, %s' % (dt.isoformat(), txt))

    def __init__(self):
        # 引用到close line
        self.dataclose = self.datas[0].close

        # 跟踪订单状态以及买卖价格和佣金
        self.order = None
        self.buyprice = None
        self.buycomm = None

        # 增加移动均线
        self.sma = bt.indicators.SimpleMovingAverage(
            self.datas[0], period=self.params.maperiod)
```

```
Starting Portfolio Value: 100000.00 2020-11-04, Close, 125.30
2020-11-05, Close, 125.57
2020-11-06, Close, 125.97
2020-11-09, Close, 126.51
2020-11-10, Close, 126.91
2020-11-11, Close, 128.25
2020-11-11, BUY CREATE, 128.25
2020-11-12, BUY EXECUTED, Price: 127.71, Cost: 8939.70, Comm 0.00
...(省略n行)
2021-09-29, Close, 127.11
2021-09-30, Close, 126.83
Final Portfolio Value: 99648.60
```

回测系统：基于backtrader

- 可视化--画图
- 将cerebro.plot()放在cerebro.run()后即可



回测系统：基于backtrader

- 优化—尝试多个移动平均的宽度

```
if __name__ == '__main__':
    cerebro = bt.Cerebro()

    # 增加多参数的策略
    strats = cerebro.optstrategy(
        TestStrategy,
        maperiod=range(10, 31))

    #获取数据
    stock_hfq_df = pd.read_excel("./data/sh600000.xlsx", index_col='date', parse_dates=True)
    start_date = datetime(2020, 9, 30) # 回测开始时间
    end_date = datetime(2021, 9, 30) # 回测结束时间
    data = bt.feeds.PandasData(dataname=stock_hfq_df, fromdate=start_date, todate=end_date) # 加载数据
    cerebro.adddata(data) # 将数据传入回测系统

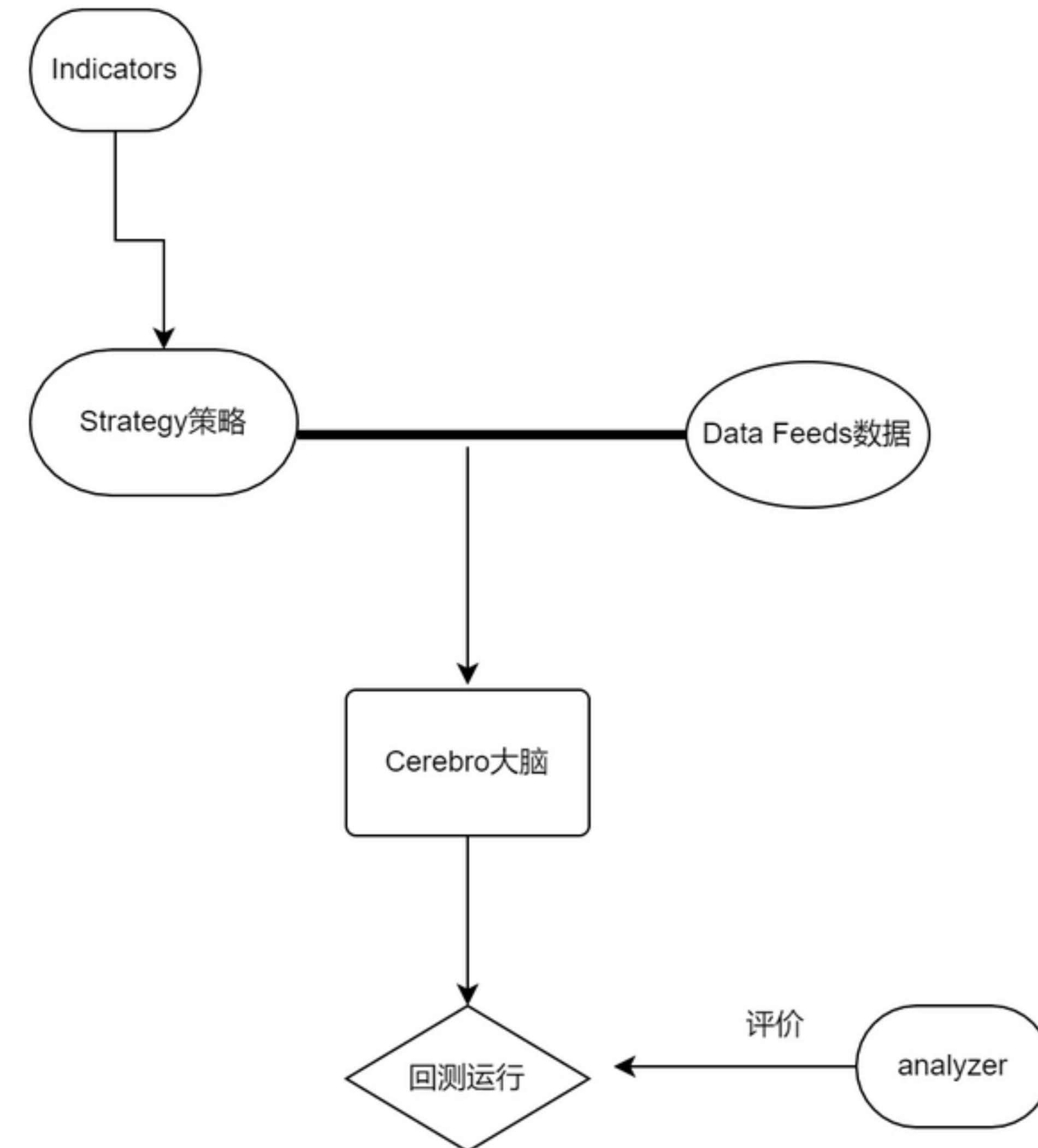
    cerebro.broker.setcash(100000.0)
    # Set the commission - 0.1% ... divide by 100 to remove the %
    cerebro.broker.setcommission(commission=0)
    # Add a FixedSize sizer according to the stake 每次买卖的股数量
    cerebro.addsizer(bt.sizers.FixedSize, stake=70)

    print('Starting Portfolio Value: %.2f' % cerebro.broker.getvalue())
    cerebro.run()

    print('Final Portfolio Value: %.2f' % cerebro.broker.getvalue())
```

2021-09-30, (MA Period 10) Ending Value 99330.80
2021-09-30, (MA Period 12) Ending Value 103441.20
2021-09-30, (MA Period 11) Ending Value 100841.40
2021-09-30, (MA Period 13) Ending Value 102013.20
2021-09-30, (MA Period 14) Ending Value 103860.50
2021-09-30, (MA Period 17) Ending Value 103322.20
2021-09-30, (MA Period 15) Ending Value 105178.60
2021-09-30, (MA Period 16) Ending Value 104009.60
2021-09-30, (MA Period 20) Ending Value 103149.30
2021-09-30, (MA Period 19) Ending Value 104309.20
2021-09-30, (MA Period 18) Ending Value 103423.00
2021-09-30, (MA Period 21) Ending Value 102899.40
2021-09-30, (MA Period 24) Ending Value 104294.50
2021-09-30, (MA Period 22) Ending Value 103868.20
2021-09-30, (MA Period 25) Ending Value 103544.80
2021-09-30, (MA Period 23) Ending Value 103264.10
2021-09-30, (MA Period 26) Ending Value 102888.20
2021-09-30, (MA Period 27) Ending Value 102410.10
2021-09-30, (MA Period 28) Ending Value 101978.20
2021-09-30, (MA Period 29) Ending Value 102212.70
2021-09-30, (MA Period 30) Ending Value 101871.10

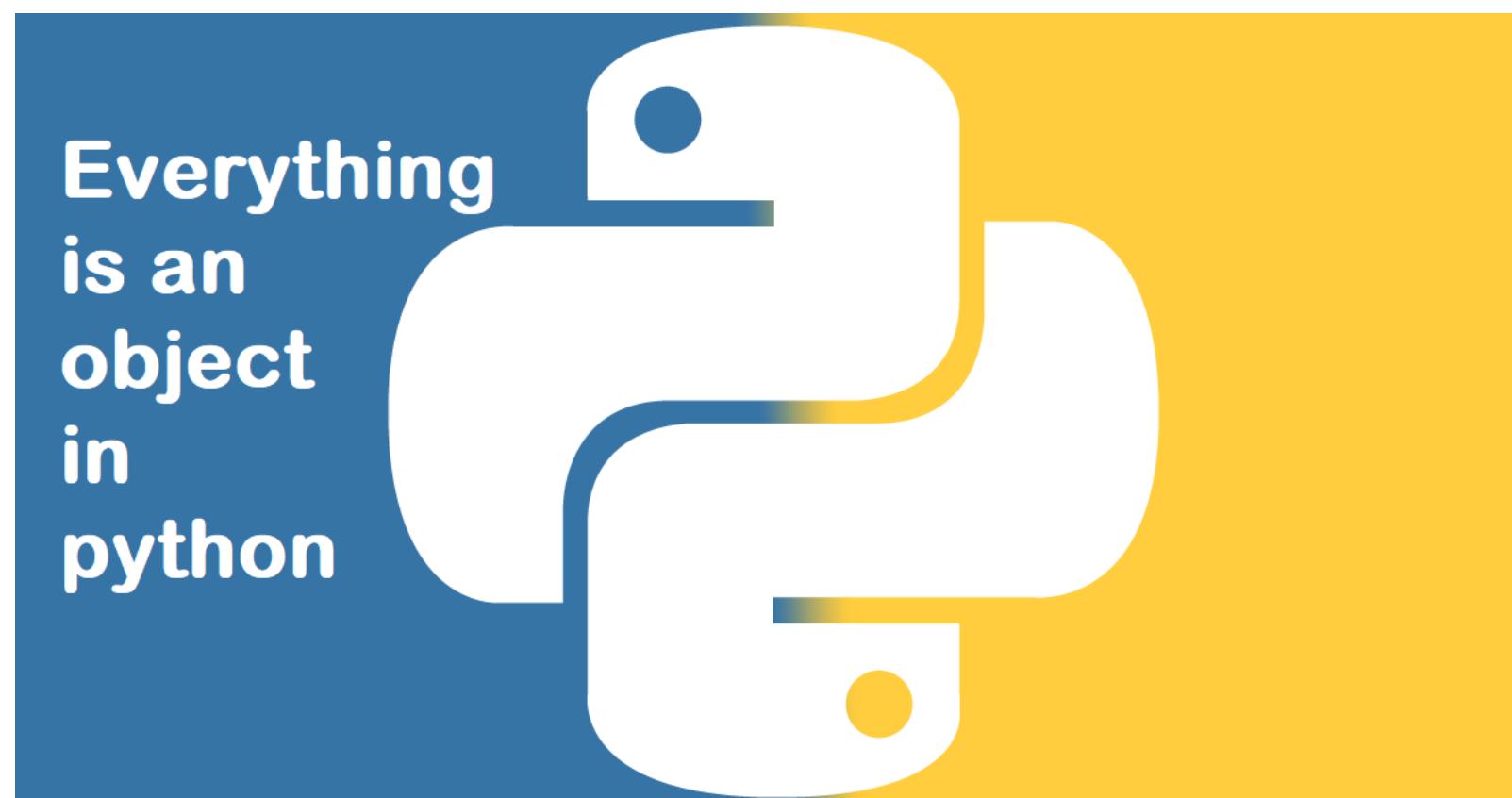
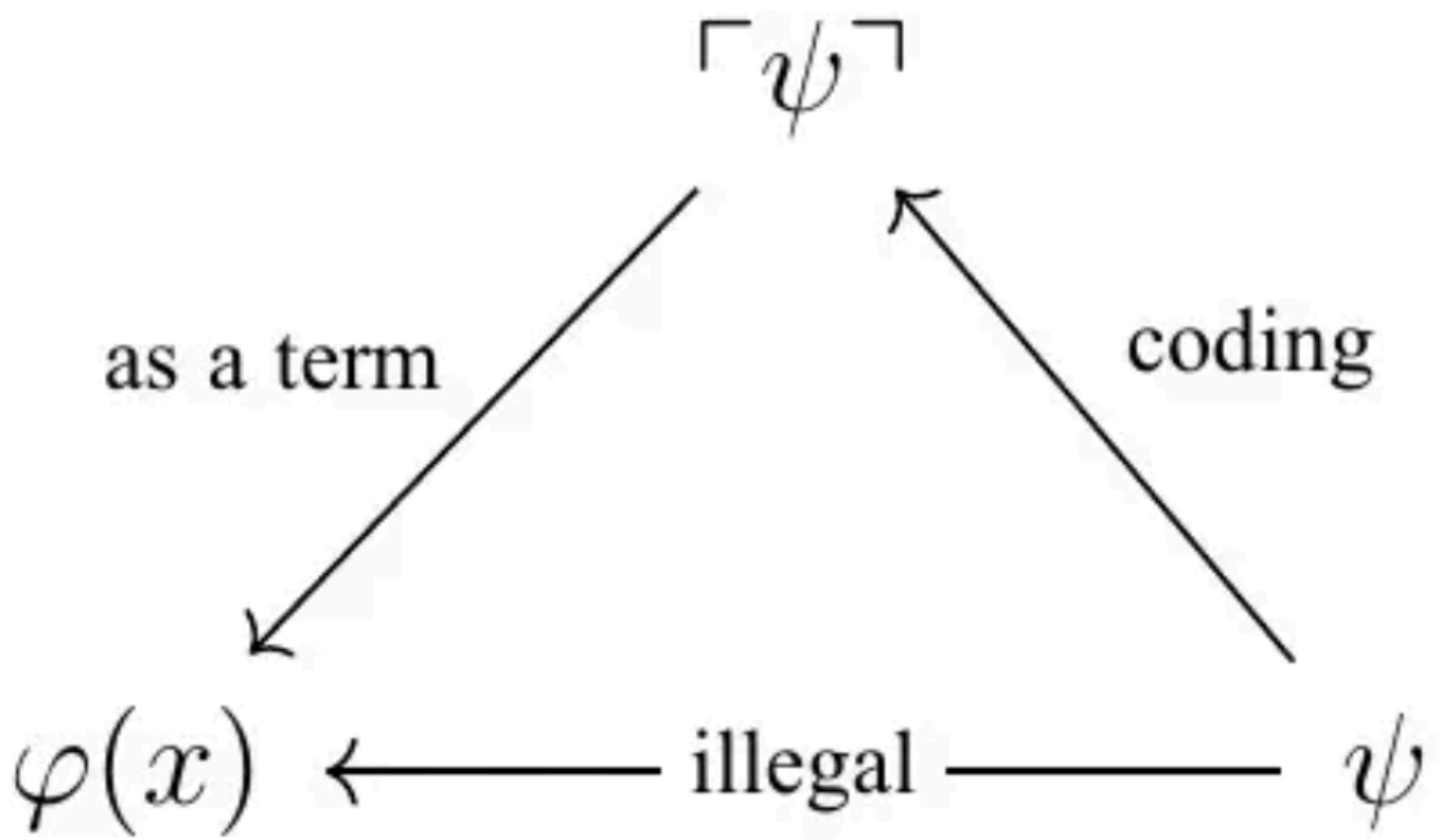
回测系统：基于backtrader



Python: 是什么?

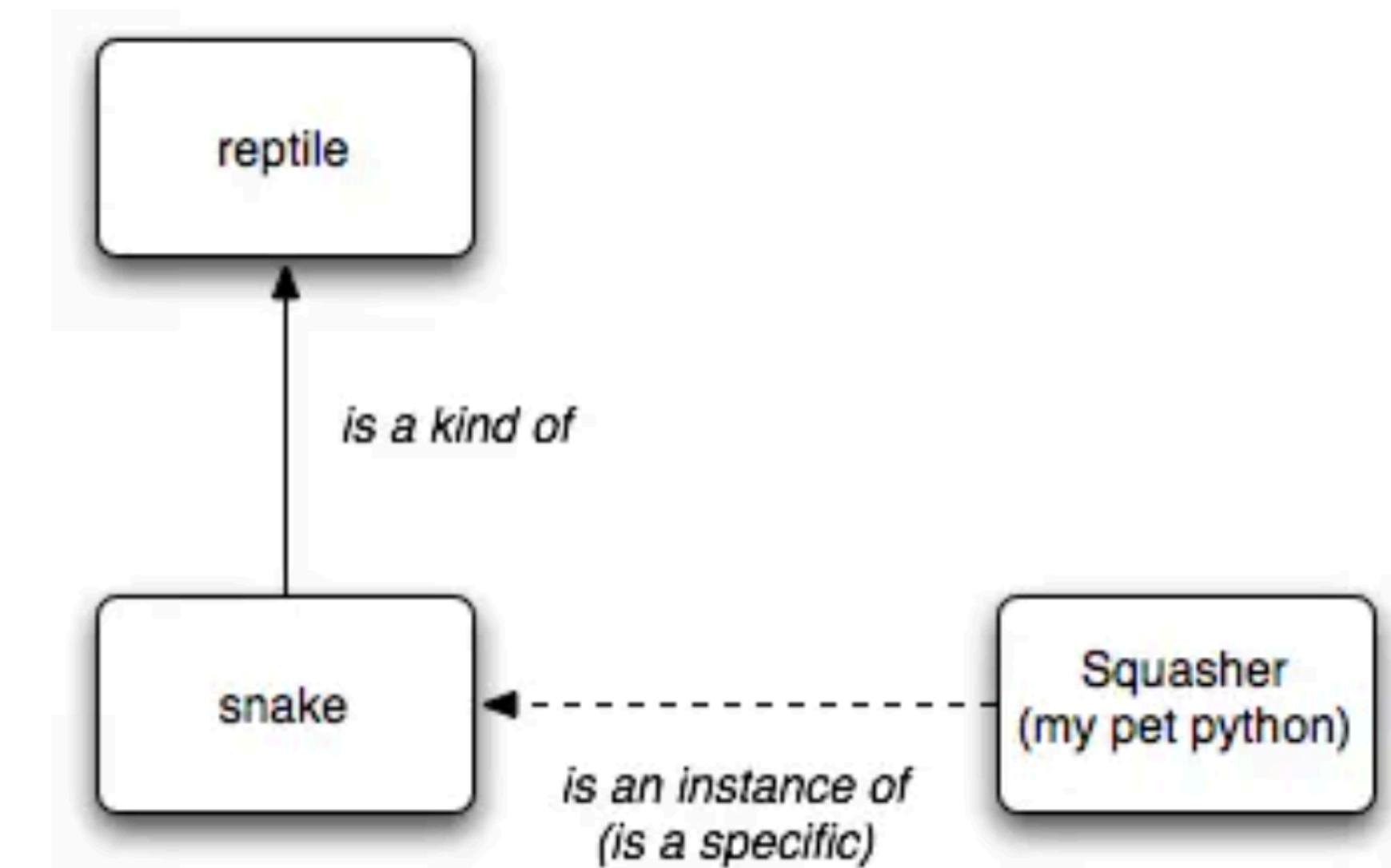
- 语言? --和自然语言有区别
- 形式逻辑系统--完备性?
- 公理体系?--存在需要承认的公理--哥德尔不完备定理
- Python中必然成立的性质是什么?
 - 课程一开始提到的:

形式上基本继承了C语言的表达方案，但本质上是面向对象的。
底层有比较复杂的实现规范。



Python: 最基本的元素

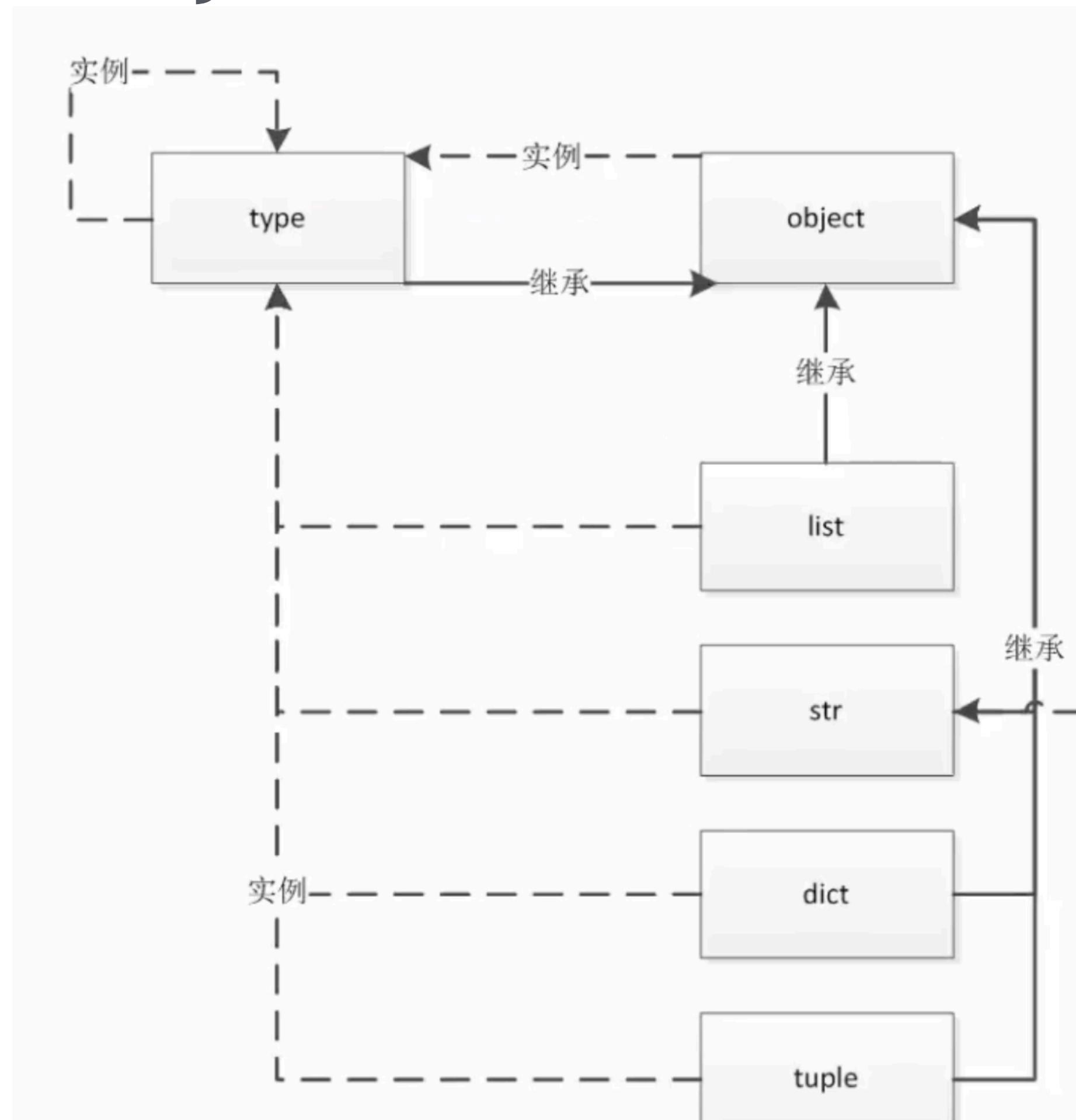
- 一种存在形式：对象
- 两种关系：实例化关系（有没有）、继承关系（有什么）
- 两种对象：类型对象（type object）、非类型对象（nontype object）
 - 类型对象：可以拥有继承关系且可以作为实例化关系中父类的对象
 - 非类型对象：不可以拥有继承关系且只能作为实例化关系中子类的对象



对上述逻辑的实现手段：`object`和`type`

Python: 矛盾所在——type & object

- 七条必然成立的性质：
 - 一切皆为对象，对象之间必有关系—**图的连通性**
 - 对象只有两类，类型对象与非类型对象—**节点的分类**
 - 关系只有两类，继承关系和实例化关系—**关系的分类**
 - 一切对象都是type的实例—**类型关系的根节点**
 - 除了object外，一切类型对象继承自object—**继承关系的根节点**
 - 类型对象的**子类的实例**是该类型对象的实例—**关系传递性1**
 - 类型对象的**实例的子类**是该类型对象的子类—**关系传递性2**



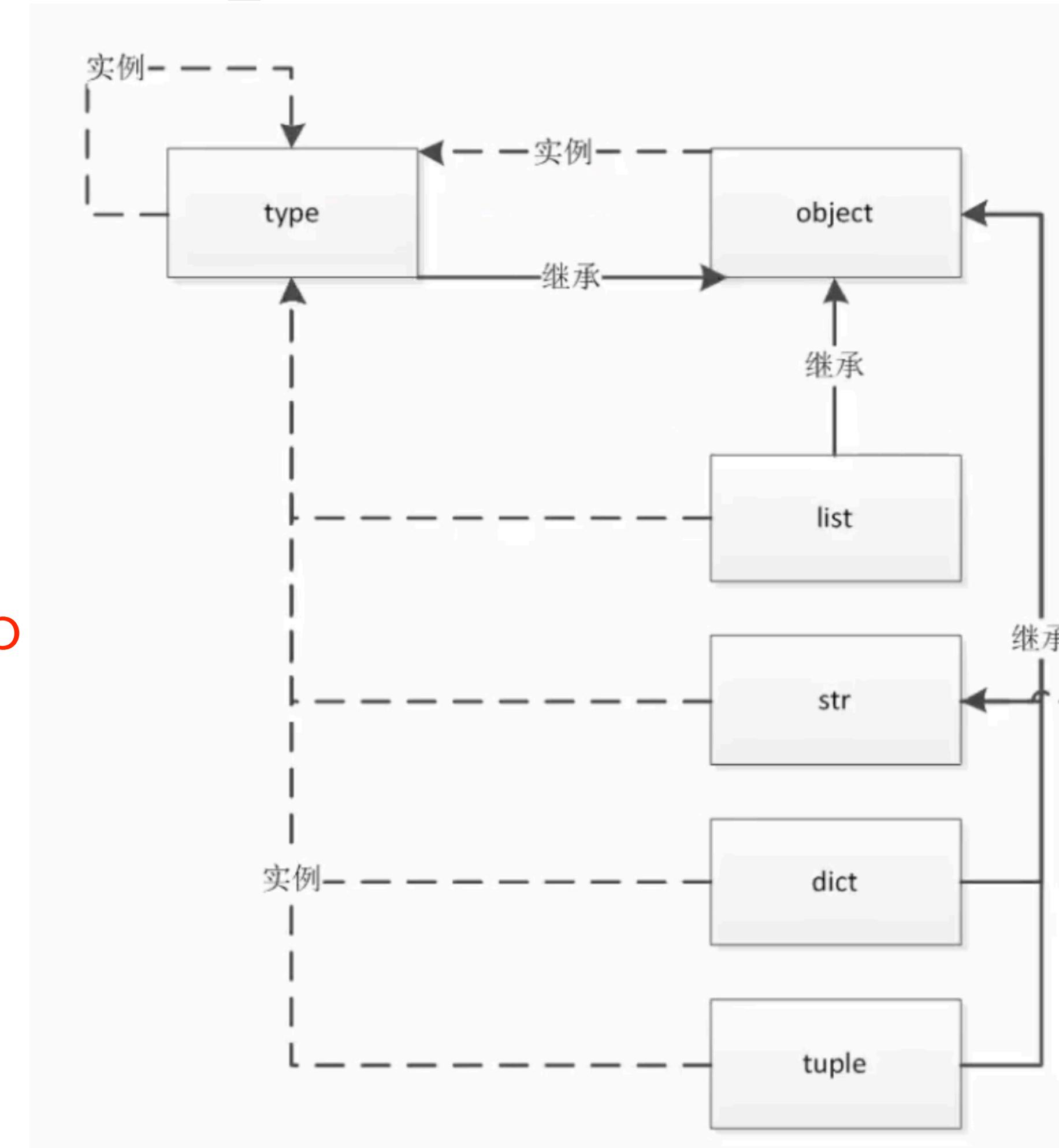
<https://docs.python.org/3/glossary.html#term-type>
<https://docs.python.org/3/glossary.html#term-object>

<https://www.eecg.toronto.edu/~jzhu/csc326/readings/metaklass-class-instance.pdf>

矛盾：type和object，先有鸡还是先有蛋？

Python: 矛盾的解决——bootstrap

- 矛盾的根源: type和object能否被解构? (Python下不行)
- 矛盾的解决: Python的底层语言--C
 - 在C下, type和object可被解构!
 - 本质: 先解构, 再划分物理内存, 最后建立映射关系(map)
 - Py_InitializeEx()->_Py_ReadyTypes()->PyType_Ready()
 - Eg: type对应了Cpython中的PyTypeObject
 - 实现的效果: 由静态的C实现了动态的Python --元编程



Python元编程：基本想法与大作业任务

- 基本想法：用类来生成类——本质上是type的子类可以实例化类型对象
- 实现方式：设计继承自type的类，在`__new__`方法中对类进行加工
- 使用场景：实际业务中能用类就用类！设计解决实际问题的类的架构，才用元类！
 - 元类->类->实例 & 理论->工具->实操

- 举例：

```
1 def __repr__(self):  
2     return f'{self.__class__.__name__}(wheel={self.wheel}, type={self.type})'  
3  
4  
5 Vehicle = type('Vehicle', (), {'wheel': 4, '__repr__': __repr__})  
6 Bus = type('Bus', (Vehicle,), {'type': 1})  
7 Truck = type('Truck', (Vehicle,), {'type': 2})  
8  
9 bus = Bus()  
10 truck = Truck()  
11 print(bus, truck)  
  
Bus(wheel=4, type=1) Truck(wheel=4, type=2)
```

Python元编程：基本想法与大作业任务

```
1 class VehicleMeta(type):
2     @classmethod
3     def __prepare__(meta_cls, name, bases, **kwds):
4         print('__prepare__', meta_cls, name, bases, kwds)
5         return kwds
6
7     def __new__(meta_cls, name, bases, dicts):
8         print('__new__', meta_cls, name, bases, dicts)
9
10    def __repr__(self):
11        return f'{self.__class__.__name__}(wheel={self.wheel}, type={self.type})'
12
13    dicts.update({'wheel': 4, '__repr__': __repr__})
14
15    clazz = super().__new__(meta_cls, name, bases, dicts)
16    return clazz
17
```

Python元编程：基本想法与大作业任务

```
19 class Vehicle(metaclass=VehicleMeta):
20     ...
21
22
23 class Bus(Vehicle):
24     type = 1
25
26
27 class Truck(Vehicle):
28     type = 2
29 |
```

耗时: 11ms

```
__prepare__ <class '__main__.VehicleMeta'> Vehicle () {}
__new__ <class '__main__.VehicleMeta'> Vehicle () {'__module__': '__main__', '__qualname__': 'Vehicle'}
__prepare__ <class '__main__.VehicleMeta'> Bus (<class '__main__.Vehicle'>,) {}
__new__ <class '__main__.VehicleMeta'> Bus (<class '__main__.Vehicle'>,) {'__module__': '__main__', '__qualname__': 'Bus', 'type': 1}
__prepare__ <class '__main__.VehicleMeta'> Truck (<class '__main__.Vehicle'>,) {}
__new__ <class '__main__.VehicleMeta'> Truck (<class '__main__.Vehicle'>,) {'__module__': '__main__', '__qualname__': 'Truck',
'type': 2}
```

```
31 bus = Bus()
32 truck = Truck()
33 print(bus, truck)
```

Bus(wheel=4, type=1) Truck(wheel=4, type=2)

Python元编程：基本想法与大作业任务

- 任务一（15分）：补全给出的基于元编程的回测系统的元类代码（metabase.py文件中）
 - 补全**findbases**函数（5分）：实现**寻找类的所有基（父）类**，包括间接的基类--递归实现
 - 出题的思路：让同学了解python中**继承关系**的遍历方式与实际继承情况
 - 补全**findowner**函数（5分）：在堆栈帧中查找拥有**特定类实例**的对象
 - 出题的思路：让同学了解python中**实例化关系**的储存模式--栈的理解
 - 补全**AutoInfoClass**类中的**_derive**函数（5分）：实现了类型对象间**信息的传递**
 - 出题的思路：让同学了解Python中**对象性质的解构方式**--map，也就是字典
- 额外任务（有助于理解回测架构）：描述**数据**进入回测系统后在各组件间的**传递情况**

数据科学：传统股票策略的实现

- 任务二（25分）：在给定的股票数据下，将以下两种策略结合起来实现一个综合性的策略，用量化回测系统进行回测，呈现结果
 - （如果要使用Pyfolio库进行可视化，因为其中有若干bug，需要手动修复）<https://blog.csdn.net/Yangxh2004/article/details/121451734>
 - 股票策略（推荐参考《151 trading strategies》）
 - **动量策略**：是分析过去一段时间中，多个股票上的收益率，做多收益率最高的一组；做空收益率最低的一组，首先，计算过去60个交易日股票的累计收益率，根据累计收益率的高低，把股票分为10组，做多收益率高的一组，做空收益率低的一组。资金在各个股票之间等权重分配。
 - **均线策略**：我们使用全市场的A股日数据进行测试，只做多头。首先，在当前交易日，分析一下，有多少个股票的均线已经存在，然后把资金平分成多少份；当前一个收盘价小于均线，这个收盘价大于均线的时候，做多这支股票，根据资金，计算出应该买的股票数目；当前一个收盘价大于均线，当前的收盘价小于均线的时候，平仓这支股票。
 - 假设：手续费万分之二，初始资金一个亿。

数据科学：传统股票策略的实现

- 我们提供的数据——行情+基本面——5149支股票从2020-01-02到2022-12-30的数据

股票日行情 `stk_daily`

列名	含义
stk_id	股票ID
date	日期
open	开盘价
high	最高价
low	最低价
close	收盘价
volume	成交量
amount	成交额
cumadj	累积复权因子

stk_id	date	open	high	low	close	volume	amount	cumadj
0 000001.SZ	2020-01-02	16.65	16.95	16.55	16.87	153023000.0	2.571200e+09	98.0986
1 000001.SZ	2020-01-03	16.94	17.31	16.92	17.18	111619000.0	1.914500e+09	98.0986
2 000001.SZ	2020-01-06	17.01	17.34	16.91	17.07	86208400.0	1.477930e+09	98.0986
3 000001.SZ	2020-01-07	17.13	17.28	16.95	17.15	72860800.0	1.247050e+09	98.0986
4 000001.SZ	2020-01-08	17.00	17.05	16.63	16.66	84782400.0	1.423610e+09	98.0986

股票资产负债表 `stk_fin_balance`

stk_id	type	date	adj	publish_date	BALANCESTATEMENT_9
0 000001.SZ	一般企业	1993-06-30	0	1993-08-11	NaN
1 000001.SZ	银行	1993-12-31	0	1994-04-16	1.115580e+08
2 000001.SZ	银行	1994-06-30	0	1994-08-30	1.395527e+08
3 000001.SZ	银行	1994-12-31	0	1995-03-10	2.382906e+08
4 000001.SZ	银行	1995-06-30	0	1995-08-11	NaN
5 000001.SZ	银行	1995-12-31	0	1996-02-09	3.862612e+08
6 000001.SZ	银行	1996-06-30	0	1996-08-29	3.392828e+09
7 000001.SZ	银行	1996-12-31	0	1997-04-21	8.717299e+08
8 000001.SZ	银行	1997-06-30	0	1997-08-28	3.646123e+08
9 000001.SZ	银行	1997-12-31	0	1998-03-12	4.005841e+08

股票利润表 `stk_fin_income`

股票现金流量表 `stk_fin_cashflow`

股票财务报表附注 `stk_fin_annotation`

数据科学：传统股票策略的实现

狗股策略实现的效果：指标呈现

<https://zhuanlan.zhihu.com/p/603614811>

Start date	2006-01-04		
End date	2019-12-30		
In-sample months	150		
Out-of-sample months	11		
	In-sample	Out-of-sample	All
Annual return	18.2%	31.5%	19.1%
Cumulative returns	713.3%	30.3%	959.4%
Annual volatility	26.1%	18.7%	25.6%
Sharpe ratio	0.77	1.56	0.81
Calmar ratio	0.29	1.69	0.31
Stability	0.81	0.04	0.83
Max drawdown	-61.7%	-18.7%	-61.7%
Omega ratio	1.16	1.31	1.17
Sortino ratio	1.06	2.37	1.12
Skew	-0.66	-0.15	-0.65
Kurtosis	4.97	1.82	5.04
Tail ratio	0.98	1.18	1.00
Daily value at risk	-3.2%	-2.2%	-3.1%
Gross leverage	0.86	0.91	0.86
Daily turnover	0.6%	0.7%	0.6%

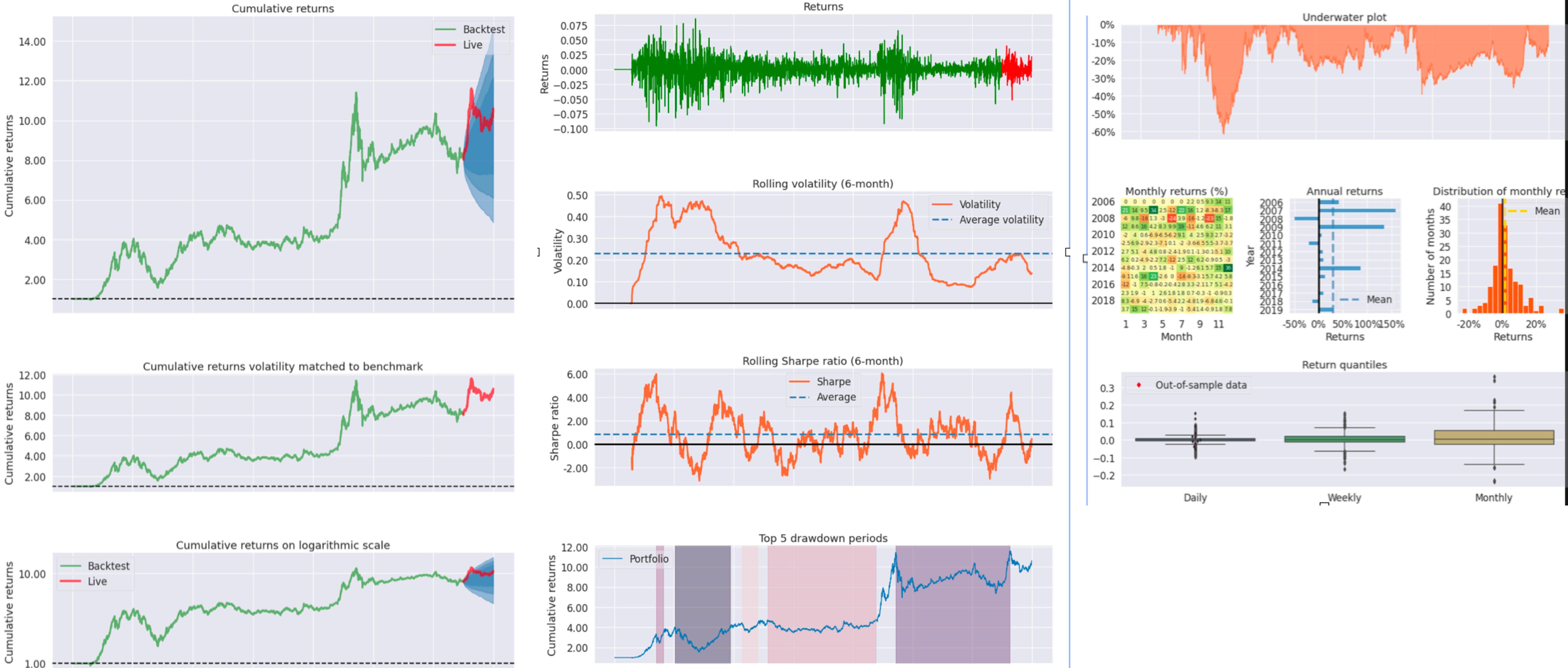
<https://www.zhihu.com/question/27264526>

Worst drawdown periods	Net drawdown in %	Peak date	Valley date	Recovery date	Duration
0	61.72	2008-01-15	2008-11-04	2009-11-18	482
1	39.22	2015-06-08	2015-08-25	2019-04-04	999
2	27.81	2007-05-29	2007-07-05	2007-08-20	60
3	27.56	2011-02-21	2012-12-03	2014-10-08	948
4	21.47	2010-04-12	2010-07-05	2010-10-25	141

Top 10 long positions of all time	max
000581	16.28%
601669	13.48%
601800	12.54%
000876	11.81%
600018	10.47%
000155	10.21%
600005	9.05%
000550	9.02%
000983	8.74%
000002	8.72%

数据科学：传统股票策略的实现

狗股策略实现的效果：可视化呈现



数据科学：股票数据的预处理与股票聚类

- 任务三（30分）：股票数据的预处理与股票的聚类：请参考以下链接中的基本框架进行优化

https://github.com/areed1192/sigma_coding_youtube/blob/master/python/python-data-science/machine-learning/k-means/Clustering%20Stocks%20-%20KMeans.ipynb

- 读取数据、清洗数据（5分）：从stock中读取股票的行情和基本面数据、处理各支股票中的缺失值和异常值
- 特征提取：使用以下几类方法对时间序列数据特征提取
 - 基本统计特征：均值、中位数、方差、极值、峰度和偏度等（5分）
 - 时域特征：差分、移动均值、滞后特征、自相关特征等（5分） <https://zhuanlan.zhihu.com/p/398752292>
 - 频域特征：傅里叶变换或小波变换提取频域信息等（5分） <https://www.zhihu.com/question/24021704/answer/2245867156>
 - 深度学习特征提取：利用预训练的模型进行特征提取（如基于BERT等预训练模型）--选做（加分最多5分）--<https://zhuanlan.zhihu.com/p/686888794>
- 特征融合：归一化数据，需要将不同的特征向量组合成一个整体特征向量（直接拼接、借助神经网络进行特征融合等），并使用PCA方法降维（可设置设置累积方差贡献率阈值确定留下多少主成分）（5分）
- 股票聚类：使用K-means进行股票聚类。肘部法确定最佳簇数量--输出每个簇的代表股票、簇中股票的数量、年化回报分布等，也可对结果进行可视化展示。（5分）

数据科学：股票策略设计与实施

- 任务四（30分）：基于以上提取出的股票特征、聚类信息和使用过的传统策略，设计股票策略并评估结果
 - 按照时间顺序划分测试和训练集，测试集在前，训练集在后，训练集:测试集=7:3
 - 设计策略，尝试将聚类信息及其中的因果关系与均线、动量策略结合，指导股票的分组处理（10分）
 - 在已知数据集对策略进行训练和调优
 - 将策略应用于未知数据集，并评估绩效（20分）
 - 传统金融指标：年化回报率、年化波动率
 - 投资效用：计算效用函数，值越高越好

数据科学：股票策略设计与实施

- 对于第*i*个有交易的天数，定义：

$$p_i = \sum_j weight_{ij} * resp_{ij},$$

$$t = \frac{\sum p_i}{\sqrt{\sum p_i^2}} \sqrt{\frac{T}{|i|}}$$

- 其中， $weight_{ij}$ 是该天编号为j的一笔交易的量占当天总交易量的比重， $resp_{ij}$ 是这笔交易的回报率，T是测试集的总天数， $|i|$ 是有交易的总天数
- 则效用函数定义如下： $u = min(max(t, 0), 6) \sum p_i.$

使用该指标进行评估的参考比赛：<https://www.kaggle.com/competitions/jane-street-market-prediction>

最后请摘出与作业要求对应的关键代码和结果，写一个方便助教理解并对照要求给分的实验报告

作业提交截止时间：2024年5月5号 晚上23点

提交方式：将作业的notebook和报告打包

作业打包以学号.rar压缩提交（拓展名是.rar）