Cryptocurrency Algorithmic Trading

HKU-SCF FinTech Academy

Python Object-Oriented Programming

The QuantConnect (QC) examples can be ignored if you have not started reading the modules on quantconnect. You can return to this module and take a look at the QC examples later.

In the QuantConnect (QC) platform, algorithms are inplemented with classes. In order for you to design an algorithm there, you will need to know the basics of Object-Oriented Prgramming (OOP).

In This chapter, we will cover the basic python OOP skills you'll need so you can understand the structure of QC algorithms and be able to read and learn from other's algorithm.

Note that there are quite some optional external references included in this Module, feel free to go through or skip them at your own interest.

Estimated Time to Finish:

3-5 hours (excluding Optional Materials)

Main Learning Objectives:

Learning how to:

- Define a **class**, which is a sort of blueprint for an object
- Instantiate an **object** from a class
- Use **attributes** and **methods** to define the properties and behaviors of an object
- Use **inheritance** to create **child classes** from a **parent class**

The Tutorial

The tutorial will be mainly based on these 2 sources.

freeCodeCamp - scientific-computing-with-python (4 videos)

1. Python Objects
2. An example class
3. object lifecycle
4. Object inheritance

How to Think Like a Computer Scientist: Interactive Edition - Chapter 17

- Object-oriented programming
- A change of perspective
- Objects Revisited
- User Defined Classes
- Improving our Constructor
- Adding Other Methods to our Class
- Objects as Arguments and Parameters
- Glossary
- Exercises

Same as the last chapter, a cheatsheet (Link to be fixed) from Python Crash Course 2nd Edition is provided (Credit goes to the author Eric Matthes)

Overview

1. Introduction
2. Classes
3. Constructors and Instances
4. Methods
5. Objects as Arguments and Parameters
6. Inheritance

---

1. Introduction

thinkcspy - object-oriented programming

Python is an **object-oriented programming language**. That means it provides features that support object-oriented programming (**OOP**).

It was developed as a way to handle the rapidly increasing size and complexity of software systems and to make it easier to modify these large and complex systems over time.

A paradigm shift

thinkcspy - A change in perspective

Up to now, some of the programs we have been writing use a procedural programming paradigm. In procedural programming the focus is on writing functions or *procedures* which operate on data. In object-oriented programming the focus is on the creation of **objects** which contain both data and functionality together. Usually, each object definition corresponds to

some object or concept in the real world and the functions that operate on that object correspond to the ways real-world objects interact.

> Tips: Don't worry if these sound too overwhelming and abstract, the upcoming examples and exercises will make them easy to understand

---

2. Classes

Jargon alert: There will be quite some jargons in this chapter. For a quick recap, please visit the Glossary

What are classes?

Classes are the foundation of **object-oriented programming**. Classes represent **real-world things you want to model** in your programs such as dogs, cars, and robots. You use a class to make **objects**, which are specific **instances** of dogs, cars, and robots.

An example class

thinkcspy - 17.4. User Defined Classes

A class defines the *general behavior that a whole category of objects can have*, and the information that can be associated with those objects. Classes can **inherit** from each other – you can write a class that **extends the functionality** of an existing class. This allows you to code efficiently for a wide variety of situations. Even if you don't write many of your own classes, you'll frequently find yourself working with classes that others have written.

Terminalogy

- **Class**: A template for objects
- **Attribute**: A variable within a class
- **Method**: A function within a class
- **Object**: An instance of a class
- **Constructor**: Code that runs when an object is created
    - see the explanation of `__init__()` below
- **Inheritance**: The ability to extend a class to make a new class

---

3. Constructors and Instances

Instances

object lifecycle

thinkcspy - 17.5. Improving Our Constructor

- We can create a lot of objects
  - reminder: A **class** is the template for **objects**
- We can store each **distinct object** in its own variable
- This is called "having multiple **instances** of the same class"
- Each **instance** has its own copy of the **instance variable**

To illustrate these concepts, It would be helpful to imagine a `Car` class that will serve as a blueprint for the different car instances that we want to create.

```
class Car:
    def __init__(self, brand, model, color, fuel):
        self.brand=brand
        self.model=model
        self.color=color
        self.fuel=fuel

cybertruck=Car('Tesla','Cybertruck','Silver','Electricity')
cybertruck.fuel
```

```
'Electricity'
```

Understanding `__init__()`

The `__init__()` method is a function that's part of a class, just like any other method. The only special thing about `__init__()` is that it's called automatically every time you make a new instance from a class. If you accidentally misspell `__init__()`, the method won't be called and your object may not be created correctly.

Know thy`self`

You may wonder what does the `self` variable represents. The self variable is a **reference to an object** that's been created from the class.

The `self` variable provides a way to make other variable and objects available everywhere in a class. The self
variable is **automatically passed to each method that's called through an object**, which is why you see it listed first in every method definition. **Any variable attached to self is available everywhere in the class.**

Examples on QuantConnect.

```
class SampleAlgorithm(QCAlgorithm):
    def Initialize(self):
        ''' Initialise the data and resolution required, as well as the cash and start-

        self.SetStartDate(2013,10,7) #Set Start Date
        self.SetEndDate(2013,10,11) #Set End Date
        self.SetCash(100000) #Set Strategy Cash
        ...
```

`Initialize(self)` is the constructor of the `SampleAlgorithm` class we defined.

- Note that there is no need to define `__init__()` for algorithms in quantconnect.

`self.SetCash` is a method from the `QCalgorithm` class, the parent class `SampleAlgorithm` inherits from.

- methods and Inheritance will be explained below.

---

4. Methods

thinkcspy - 17.6. Adding Other Methods to our Class

To reiterate, classes are like blueprints for object, just like how a rocket is made from referring to its blueprint according to the plans. It has all the properties mentioned in the plan, and behaves accordingly.

- A Python method is just like a Python function, but it must be called on an object.
  - It must be put inside a class.
  - It is a label that you can call on an object; it is a piece of code to execute on that object.

Let's continue with the `Car` class examples. we expect the cars to have certain behaviours like start(), halt(), drift(), speedup(), and turn().

Examples on QuantConnect.

```
class SampleAlgorithm(QCAlgorithm):

    def Initialize(self):
        ...

    def OnData(self, data):
```

```
        '''OnData event is the primary entry point for your algorithm.
        Each new data point will be pumped in here.

        Arguments:
            data: Slice object keyed by symbol containing the stock data
        '''
        ...

    def OnEndOfAlgorithm(self):
        '''Called when the algorithm ends'''
        ...
```

In our `SampleAlgorithm`, we expect the algorithm to perfect some actions when it is fed data or when it ends. So we define 2 methods that will be called when those conditions are met. On QuantConnect, said logic and conditions have already been implemented on their platform so you just have to focus on designing what the algorithms would do under these situations later.

---

5. Objects as Arguments and Parameters

thinkcspy - 17.7. Objects as Arguments and Parameters

We can pass objects as arguments in methods of a class so that the different objects can interact.

Exercise

thinkcspy - 17.11. Exercises

Examples on QuantConnect.

```
class SampleAlgorithm(QCAlgorithm):

    def Initialize(self):
        ...

    def OnData(self, data):
        ...

    def OnEndOfAlgorithm(self):
        ...
```

```
    def OnOrderEvent(self, orderEvent):
        # Log filled orders
        if orderEvent.Status == OrderStatus.Filled:
            self.Log("{}: {} Filled".format(self.Time, orderEvent.ToString()))
```

we can define `OnOrderEvent()` that uses `orderEvent` objects as parameter
Each order made by the algorithm generates order events, which are passed to the `OnOrderEvent()` method for logging the information about order states. For more information, you can refer to QuantConnect Docs

> Forgot about how .format() works? Revisit the string subtopic from the last chapter Module 1a - Python Basics

---

6. Inheritance:

freeCodeCamp: Inheritance

from GeeksforGeeks:

> Inheritance is the capability of one class to derive or inherit the properties from another class. The benefits of inheritance are:
>
> 1. It represents real-world relationships well.
> 2. It provides **reusability** of a code. We don’t have to write the same code again and again. Also, it allows us to add more features to a class without modifying it.
> 3. It is transitive in nature, which means that if class B inherits from another class A, then all the subclasses of B would automatically inherit from class A.

You are not required to dig too deeply into inheritance. It only requires basic understanding of inheritance in order to for you to use QuantConnect.

Examples on QuantConnect.

```
class SampleAlgorithm(QCAlgorithm):
    ...
```

The `SampleAlgorithm` we have seen earlier inherits from the `QCalgorithm` class created by QuantConnect. This enables our algorithm to use methods defined in the `QCAlgorithm` parent class.

Optional Resources

Real Python - Object-Oriented Programming (OOP) in Python 3

- A comprehensive text guide for basic OOP in python with examples (but no exercises.)

Next up:

You will be learning data science libraries in Python link here