Solutions - Practical Lesson 4

Matteo Sani
matteosan1@gmail.com

October 11, 2019

1 Solutions

1.1 Exercises

1.1.1 Exercise 4.1

Write two classes, `Circle` and `Rectangle` that given the radius and height, width respectively allow to compute area and perimeter of the two shapes. Test them with the following:

```
a_circle = Circle(5)
print ("My circle has an area of {} m**2".format(a_circle.area()))

a_rectangle = Rectangle(3, 6)
print ("My rectangle has a perimeter of {} m and an area of {} m**2" \
    .format(a_rectangle.perimeter(), a_rectangle.area()))
```

```
In [1]: from math import pi

        class Circle:
            def __init__(self, radius):
                self.radius = radius

            def area(self):
                return pi*self.radius**2

        class Rectangle:
            def __init__(self, width, height):
                self.height = height
                self.width = width

            def area(self):
                return self.width*self.height

            def perimeter(self):
                return self.width*2 + self.height*2

        circle = Circle(5)
```

```
        print ("My circle area is {:.1f} m**2".format(circle.area()))

        rect = Rectangle(3, 6)
        print ("My rect area is {:.1f} m**2 and the "\
               "perimeter is {} m".format(rect.area(), rect.perimeter()))
```

```
My circle area is 78.5 m**2
My rect area is 18.0 m**2 and the perimeter is 18 m
```

1.1.2 Exercise 4.2

Define a class `Songs`, its `__init__` should take as input a dictionary (`lyrics` that contains lyrics line by line). Define a method, `sing_me_a_song` that prints each element of the lyrics in his own line. Also test it with the follwing input.

```
lyrics = {"Wonderwall":["Today is gonna be the day",
                        "That they're gonna throw it back to you",
                        "By now you should've somehow", "..."],
          "Shpalman":  ["Attenti cattivissimi",
                        "perchè è arrivato Shpalma",
                        "che shpalma la merda in faccia a", "..."]}
```

```
In [2]: class Songs:
            def __init__(self, lyrics):
                self.lyrics = lyrics

            def sing_me_a_song(self, title):
                song = self.lyrics[title]
                print ("Title: {}".format(title))
                print ("********************")
                for line in song:
                    print (line)

        lyrics = {"Wonderwall":["Today is gonna be the day",
                                "That they're gonna throw it back to you",
                                "By now you should've somehow", "..."],
                  "Shpalman":  ["Attenti cattivissimi",
                                "perchè è arrivato Shpalma",
                                "che shpalma la merda in faccia a", "..."]}

        songs = Songs(lyrics)
        songs.sing_me_a_song("Wonderwall")
```

```
Title: Wonderwall
********************
Today is gonna be the day
That they're gonna throw it back to you
By now you should've somehow
```

...

1.1.3 Exercise 4.3

Define a Point2D class that represent a point in a plane. Its `__init__` method should accept the point coordinates `x` and `y`. Write a method `distanceTo` that compute the distance of the point to another passed as input. Test the class by printing the distance of the point $P = (4, 5)$ to the origin $P = (0, 0)$ and to $P = (3, 4)$.

```
In [3]: from math import sqrt

        class Point2D:
            def __init__(self, x, y):
                self.x = x
                self.y = y

            def distanceTo(self, x, y):
                dist = sqrt((self.x-x)**2 + (self.y - y)**2)
                return dist

            def distanceTo_v2(self, p):
                dist = sqrt((self.x-p[0])**2 + (self.y - p[1])**2)
                return dist

            def distanceTo_v3(self, p):
                dist = sqrt((self.x-p.x)**2 + (self.y - p.y)**2)
                return dist


        point = Point2D(4, 5)
        p0 = (0, 0)
        point0 = Point2D(0, 0)
        print ("distance to p0: {}".format(point.distanceTo(p0[0], p0[1])))
        print ("distance_v2 to p0: {}".format(point.distanceTo_v2(p0)))
        print ("distance_v3 to p0: {}".format(point.distanceTo_v3(point0)))

        p1 = (3, 4)
        point1 = Point2D(3, 4)
        print ("distance to p1: {}".format(point.distanceTo(p1[0], p1[1])))
        print ("distance_v2 to p1: {}".format(point.distanceTo_v2(p1)))
        print ("distance_v3 to p1: {}".format(point.distanceTo_v3(point1)))
```

```
distance to p0: 6.4031242374328485
distance_v2 to p0: 6.4031242374328485
distance_v3 to p0: 6.4031242374328485
distance to p1: 1.4142135623730951
distance_v2 to p1: 1.4142135623730951
```

```
distance_v3 to p1: 1.4142135623730951
```

1.1.4 Exercise 4.4

So now that we have an idea of what a class is, try to write a `DiscountCurve` class which contains the pillar dates and pillar discount factors as attributes and which has methods for calculating the discount factor and forward libor rate at arbitrary dates.

Solution:

```
In [4]: import math
        import numpy
        from datetime import date

        class DiscountCurve:

            # the special __init__ method defines
            # how to construct instances of the class
            def __init__(self, today, pillar_dates, discount_factors):
                # we just store the arguments as attributes of the instance
                self.today = today
                self.pillar_dates = pillar_dates
                self.discount_factors = discount_factors

            # calculates a discount factor at an arbitrary
            #value date using the data stored in the instance
            def df(self, d):
                # these remain local variables,
                # i.e. they are only available within the function.
                # to read (or write) instance attributes,
                # you always need to use the self. syntax
                log_discount_factors = \
                  [math.log(discount_factor)
                   for discount_factor in self.discount_factors]
                pillar_days = [(pillar_date - self.today).days
                               for pillar_date in self.pillar_dates]
                d_days = (d - self.today).days
                interpolated_log_discount_factor = \
                    numpy.interp(d_days, pillar_days, log_discount_factors)
                return math.exp(interpolated_log_discount_factor)

            # calculates a forward libor rate based on the discount
            # curve data stored in the instance
            def forward_libor(self, d1, d2):
                # we use the df method of the current instance to calculate
                # the forward rate
```

```
                return (self.df(d1) / self.df(d2) - 1.0) * \
                        (365.0 / ((d2 - d1).days))
```

In [5]:
```
# build the EONIA curve object
# n.b. here we use the 'parameter=argument' syntax
# (today=..., pillar_dates=...)
# just so it's really clear what we're doing - it's not necessary,
# it's only for clarity

eonia_curve = DiscountCurve(today=date(2017, 10, 1),
                            pillar_dates=[date(2017, 10, 1),
                                          date(2018, 10, 1),
                                          date(2019, 10, 1)],
                            discount_factors=[1.0, 0.95, 0.8])

# build the Libor curve object
libor_curve = DiscountCurve(today=date(2017, 10, 1),
                            pillar_dates=[date(2017, 10, 1),
                                          date(2018, 4, 1),
                                          date(2018, 10, 1)],
                            discount_factors=[1.0, 0.98, 0.82])

# Let's compute the discount factor of the two curves
# on the 1-6-2018
print (eonia_curve.df(date(2018, 6, 1)))
print (libor_curve.df(date(2018, 6, 1)))
```

```
0.9664277992834573
0.9234683197742072
```

In [6]:
```
# Let's compute now the 6m forward rate at 1-4-2018
print (libor_curve.forward_libor(date(2018, 10, 1),
                                 date(2018, 4, 1)))
print (eonia_curve.forward_libor(date(2018, 10, 1),
                                 date(2018, 4, 1)))
```

```
0.3256384521021525
0.050639359540235025
```

In [7]:
```
# Compute the NPV of the 6m forward libor coupon
npv = eonia_curve.df(date(2018, 10, 1)) * \
      libor_curve.forward_libor(date(2018,10, 1),
                                date(2018, 4, 1))

# Compute it in the pre-2008 way
npv_pre_2008 = libor_curve.df(date(2018, 10, 1)) *\
               libor_curve.forward_libor(date(2018, 10, 1),
```

```
                                                    date(2018, 4, 1))
        print (npv)
        print (npv_pre_2008)
```

```
0.30935652949704484
0.267023530723765
```