C200 Programming Assignment № 4

Dr. M.M. Dalkilic
Computer Science
School of Informatics, Computing, and Engineering
Indiana University, Bloomington, IN, USA

October 1, 2023

Introduction

Due Date: 11:00 PM, Sunday, October 8th 2023 Submit your work to the Autograder `https://c200.luddy.indiana.edu/` and remember to add, commit and **push often** to GitHub **before the deadline. We do not accept late submissions.**

Instructions

1. Make sure that you are **following the instructions** in the PDF, especially the format of output returned by the functions. For example, if a function is expected to return a numerical value, then make sure that a numerical value is returned (not a list or a dictionary). Similarly, if a list is expected to be returned then return a list (not a tuple, set or dictionary).
2. Test **debug** the code well (syntax, logical and implementation errors), before submitting to the Autograder. These errors can be easily fixed by running the code in VSC and watching for unexpected behavior such as, program failing with syntax error or not returning correct output.
3. Make sure that the **code does not have infinite loop (that never exits)** before submitting to the Autograder. You can easily check for this by running in VSC and watching for program output, if it terminates timely or not.
4. Given that you already tried points 1-3, if you see that Autograder does not do anything (after you press 'submit') and waited for a while (30 seconds to 50 seconds), try refreshing the page or using a different browser.
5. Once you are done testing your code, comment out the tests i.e. the code under the __name__ == "__main__" section.

Problem 1: Day of the Week

The modulus operator is denoted by the symbol %. For $x\%y$ it returns the remainder after y is divided into x a whole number of times. While you've seen this before, the format is likely different.

To refresh your memory you can visit `https://realpython.com/python-modulo-operator/` or do your own search. For this problem, you will also need to use the floor function in python - think of floor as a way to round down a floating point number to the nearest integer. For example, floor(18.90) is 18 and floor(14.25) is 14.

An approximate (works generally okay for a number of dates, but **not** all) formula for computing the day of the week given dlst = [d,m,y] where d is day, m is month, and y is year is:

$$
\begin{aligned}
dlst &= [d, m, y] && (1)\\
a(dlst) &= y - \frac{(14-m)}{12} && (2)\\
x &= a(dlst) + \frac{a(dlst)}{4} - \frac{a(dlst)}{100} + \frac{a(dlst)}{400} && (3)\\
b(dlst) &= floor(x) && (4)\\
c(dlst) &= m + 12(\frac{14-m}{12}) - 2 && (5)\\
day((d, m, y)) &= (d + b(dlst) + (31 \cdot \frac{c(dlst)}{12}))\%7 && (6)
\end{aligned}
$$

The function day returns a number $1, 2, \ldots, 7$ where $1 = Monday, 2 = Tuesday, \ldots$. If we use this number as the key with the week dictionary, we are able to return the correct day of the week. Here is the dictionary used:

```
week = {1:"Mon", 2:"Tue", 3:"Wed", 4:"Thu", 5:"Fri", 6:"Sat", 7:"Sun"}
```

For example,

$$
\begin{aligned}
day([14, 2, 2000]) &= \text{Mon} && (7)\\
print(day([14, 2, 1963])) &= \text{Thu} && (8)\\
print(day([14, 2, 1972])) &= \text{Mon} && (9)
\end{aligned}
$$

2/14/2000 falls on a Monday; 2/14/1963 falls on a Thursday; 2/14/1972 falls on a Monday.

Deliverables for Problem 1

- Complete the functions described above.
- For calculating b(dlst), you can use the math.floor() function from the math library.
- Remember, the day function utilizes the week dictionary.

Problem 2: Starting quantum computing and quantitative finance

Quantum Computing is a different model of computation imagined by the physicist Feynman. Instead of whole numbers, like we use in the Turing model, it uses complex numbers. Complex numbers, if you remember, are actually pairs of real numbers written as:

$$\text{complex number} \quad = \quad x \pm y\, i \tag{10}$$

where $x, y \in \mathbb{R}$ (they're just numbers) and there's a lone $i = \sqrt{-1}$. The x is called the real part and the y is called the imaginary part. For example,

$$x^2 + 1 \quad = \quad 0 \tag{11}$$
$$x \quad = \quad \sqrt{-1} = i \tag{12}$$

Then

$$x^2 + 1 \quad = \quad (\sqrt{-1})^2 + 1 = -1 + 1 = 0 \tag{13}$$

In Python we use the function complex(x,y) to form a complex number $(x \pm yj)$ where j represents i and there's no space between the $\pm$ sign and numbers. Let's write the function for the values shown on (10), build the complex number, and show it's a solution.

```
>>> def f(x):
...     return x**2 + 1
...
>>> root = complex(0,1)
>>> f(root)
0j
>>> f(root) == 0
True
>>> f(root).real
0.0
>>> f(root).imag
0.0
```

For a quadratic from last homework, you know that the answer is either real or complex. To remind you, the answer is complex when the discriminant is negative. Fig. 1. visualizes what's happening–the curve for complex roots does **intersect** the abscissa. Observe that you'll get zero (on the x-axis or *abscissa*) if you put either of these two x values there. Sometimes the discriminant (the value in the squareroot) is negative. This is where i comes in. We simply multiply the value by -1, thereby allowing us to take the squareroot, but then we append an i to signal it's imaginary.

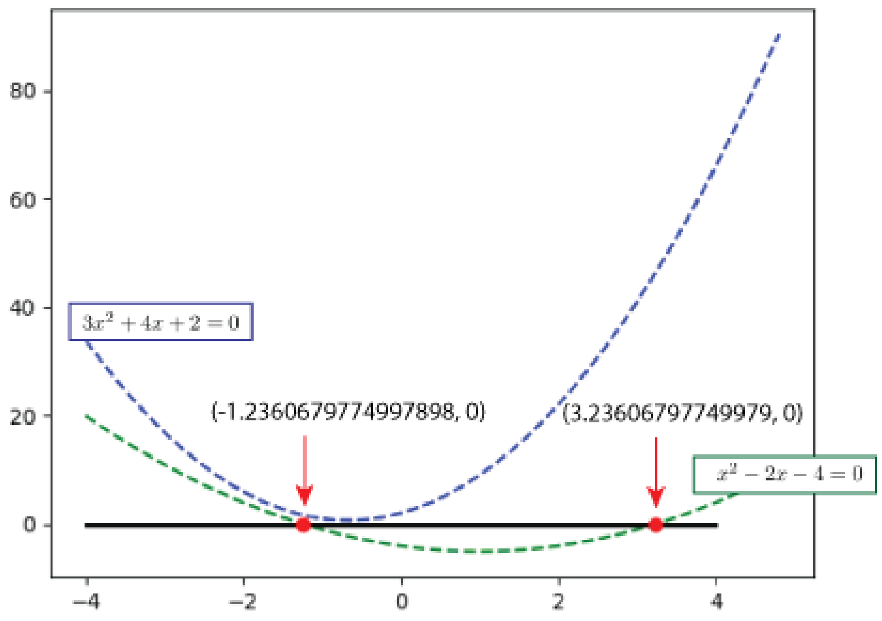

Figure 1: The red dots (with red arrows) are where your solutions are to $x^2 - 2x - 4 = 0$. The dash curve is the function itself. The horizontal line just makes it easier to see the x-axis. The two values are -1.2360679774997898 and 3.23606797749979. For the other function $3x^2+4x+2 = 0$ shown in blue, because it has an imaginary part, it cannot cross the axis. You'll use the matplotlib library you were introduced last homework to actually plot this!

For example, suppose we have $3x^2 + 4x + 2 = 0$. Then we find, after some algebra:

$$\begin{aligned} x &= \frac{-4 \pm \sqrt{-8}}{6} && (14)\\ &= \frac{-4 \pm \sqrt{-2 \times 4}}{6} && (15)\\ &= \frac{-4 \pm \sqrt{-2} \times \sqrt{4}}{6} && (16)\\ &= \frac{-4 \pm 2\sqrt{-2}}{6} && (17)\\ &= -\tfrac{2}{3} \pm \tfrac{\sqrt{2}}{3} i && (18)\\ &= (-\tfrac{2}{3} + \tfrac{\sqrt{2}}{3} i, -\tfrac{2}{3} - \tfrac{\sqrt{2}}{3} i) && (19) \end{aligned}$$

In Python we would write:

```
>>> import math
>>> complex(-2/3,math.sqrt(2)/3),complex(-2/3,-math.sqrt(2)/3)
((-0.6666666666666666+0.47140452079103173j), ↩
    (-0.6666666666666666-0.47140452079103173j))
>>> x = round(-2/3,2)
>>> y = round(math.sqrt(2)/3)
>>> complex(x,y), complex(x,-y)
((-0.67+0j), (-0.67+0j))
>>> def f(x):
...     return 3*(x**2) + 4*x + 2
```

```
...
>>> f(complex(x,y))
(0.6667000000000001+0j)
>>> f(complex(x,-y))
(0.6667000000000001+0j)
>>> f(complex(-2/3,math.sqrt(2)/3))
0j
```

Observe the difference between using the full root and only to two decimal places. You'll write a function q(t) where:

$$q((a,b,c)) = \begin{cases} (r_0, r_1) & \text{real roots if } b^2 - 4ac \geq 0 \\ (c_0, c_1) & \text{complex roots if } b^2 - 4ac < 0 \end{cases} \tag{20}$$

When returning the roots, round to two decimal places, *i.e.*, round(x,2). Here are a few examples:

$$q((3,4,2)) = ((-0.67 + 0.47j), (-0.67 - 0.47j)) \tag{21}$$
$$q((1,3,-4)) = (-4.0, 1.0) \tag{22}$$
$$q((1,-2,-4)) = (-1.24, 3.24) \tag{23}$$

Deliverables for Problem 2

- Complete the function according to the equations on line 20.
- Don't forget to round to two decimal places.

Problem 3: Computing Relationships

To match people, companies use trigonometry. Assume you have a list of people $[p_0, p_1, \ldots, p_m]$, we find the two different people who have the smallest angle. The way we calculate this is to assume each person is a list (mathematically vector) of 0s and 1s. We need three basic functions: inner product, magnitude, and $\cos^{-1}$. We assume two lists $x = [x_0, x_1, \ldots, x_n], y = [y_0, y_1, \ldots, y_n]$ of 0s and 1s of the same length:

$$inner\_prod(x, y) = x_0y_0 + x_1y_1 + \cdots + x_ny_n \tag{24}$$

$$mag(x) = \sqrt{inner\_prod(x, x)} \tag{25}$$

For the last function (where we calculate the angle), we know that:

$$\cos(\theta) = \frac{inner\_prod(x, y)}{mag(x)mag(y)} \tag{26}$$

In class we learned that we can invert a function and the inverted function is denoted as f^{-1}. So we can:

$$\cos^{-1}(\cos(\theta)) = \cos^{-1}(\frac{inner\_prod(x, y)}{mag(x)mag(y)}) \tag{27}$$

$$\theta = \cos^{-1}(\frac{inner\_prod(x, y)}{mag(x)mag(y)}) \tag{28}$$

The math module has math.acos() for $\cos^{-1}()$. Further, Python returns θ in radians. We have:

$$\pi \text{ radians} = 180 \text{ degrees} \tag{29}$$

Thus, to convert from radians to degree, you **must** multiply your answer by $\frac{180}{\pi}$.

Your task is to first complete the functions “inner_prd“, “mag“ and “angle“ and then use them to write the “match“ and “best_match“ functions. The match function takes a list of people pi where each person is a list of 0s 1s, and returns all unique pairs with the angle in degrees. Note that “match“ returns the output in the format: [[person 1, person2, angle], [person2, person3, angle], [person1, person3, angle]], where each person i.e., person1, person2-is also a list for example, [1,1,1] or [0,1,0]. So it returns a list of lists. For “angle“ function-round your answer to 2 decimal digits.

```python
people0 = [[0,1,1],[1,0,0],[1,1,1]]
print(match(people0))
print(best_match(match(people0)))
```

gives an output

```
[[[0, 1, 1], [1, 0, 0], 90.0],
 [[0, 1, 1], [1, 1, 1], 35.26],
 [[1, 0, 0], [1, 1, 1], 54.74]]
([0, 1, 1], [1, 1, 1], 35.26)
```

We're displaying the result of match so you can see the structure. Each unique pair as an angle. For example:

$$\begin{aligned}
inner\_prod([1,0,0],[1,1,1]) &= 1(1)+0(1)+0(1)=1 && (30)\\
mag([1,0,0]) &= \sqrt{inner\_prod([1,0,0],[1,0,0])}=\sqrt{1}=1 && (31)\\
mag([1,1,1]) &= \sqrt{inner\_prod([1,1,1],[1,1,1])}=\sqrt{3} && (32)\\
\theta &= (\frac{180}{\pi})\text{math.acos}(\frac{1}{1\sqrt{3}})\approx 54.74 && (33)
\end{aligned}$$

Deliverables for Problem 3

- Complete the functions.
- Keep in mind that function angle() utilizes equation 28.
- Round the output of angle to two decimal places.

Problem 4: Completing the Square

When the roots of a quadratic are real, we can transform the quadratic $ax^2+bx+c=0$ into a simpler form and find the roots more easily:

$$ax^2+bx+c=(x+m)^2+n$$

To find the roots then:

$$\begin{aligned}
(x+m)^2+n &= 0\\
(x+m)^2 &= -n\\
x+m &= \pm\sqrt{-n}\\
x &= -m\pm\sqrt{-n}
\end{aligned}$$

Through some simple algebra we find that

$$\begin{aligned}
m &= \frac{b}{2a}\\
n &= c-\frac{b^2}{4a}
\end{aligned}$$

We can then call our transform function that takes a, b, c and returns m, n as seen below:

```python
def c_s(coefficients):
    pass

def q_(coefficients):
    m,n = c_s(coefficients)
    pass
```

Deliverables for Problem 4

- Complete the functions.
- Round the output from c_s() and q_() function to 2 decimal places.
- The q_ function must call the c_s function.

Problem 5: Toward Statistical Analysis

In this problem, you'll write fundamental statistical functions. We assume a list of numbers $lst = [x_0, x_1, \ldots, x_n]$.

$$\begin{aligned} mean(lst) &= (x_0 + x_1 + \cdots + x_n)/\text{len}(lst) && (34) \\ \mu &= mean(lst) && (35) \\ var(lst) &= \frac{1}{\text{len}(lst)}((x_0 - \mu)^2 + (x_1 - \mu)^2 + \cdots + (x_n - \mu)^2) && (36) \\ std(lst) &= \sqrt{var(lst)} && (37) \end{aligned}$$

For example, $lst = [1, 3, 3, 2, 9, 10]$, rounding to two places

$$\begin{aligned} mean(lst) &= 4.67 && (38) \\ var(lst) &= 12.22 && (39) \\ std(lst) &= 3.5 && (40) \end{aligned}$$

The last function mean_centered takes a list of numbers $lst = [x_0, x_1, \ldots, x_n]$ and returns a new list $[x_0 - \mu, x_1 - \mu, \ldots, x_n - \mu]$. An interesting feature of the mean-centered list is that if you try to calculate its mean, it's zero:

$$\begin{aligned} \mu &= (x_0 + x_1 + \cdots + x_n)/n && (41) \\ lst &= [x_0 - \mu, x_1 - \mu, \ldots x_n - \mu] && (42) \\ mean(lst) &= ((x_0 - \mu) + \cdots (x_n - \mu))/n && (43) \\ &= ((x_0 + \ldots + x_n) + n\mu)/n && (44) \\ &= \mu - \mu = 0 && (45) \end{aligned}$$

Using the same list we have

$$mean(mean\_centered(lst)) = -0.0 = 0 \qquad (46)$$

Deliverables for Problem 5

- Complete the functions.
- Round-up to 2 decimal places the ouptut of mean, var and std functions.

Problem 6: Market Equilibrium

When modeling market behavior we use two curves to indicate supply and demand. You can also think of supply as quantity and demand as want. When the two curves intersect (the common point where they cross), see Fig. 2, there is a point that satisfies both equations. Review the problem of intersecting two lines. Here we have specifically two quadratic functions. Assume

$$s(x) = -.025x^2 - .05x + 60 \tag{47}$$
$$d(x) = 0.02x^2 + .6x + 20 \tag{48}$$

for supply s and demand d. We have a function $equi$ that takes the coefficients of s, d and returns the solution:

$$equi((-.025, -.5, 60), (0.02, .6, 20)) = (20.0, -44.44) \tag{49}$$

Our solution returns the roots to a quadratic equation. Since s, d models the presence of items, only 20 makes sense. HINT: use your q function in Problem 2.

Deliverables for Problem 6

- Complete the function.
- Use q in Problem 2 to make the solution very quick.

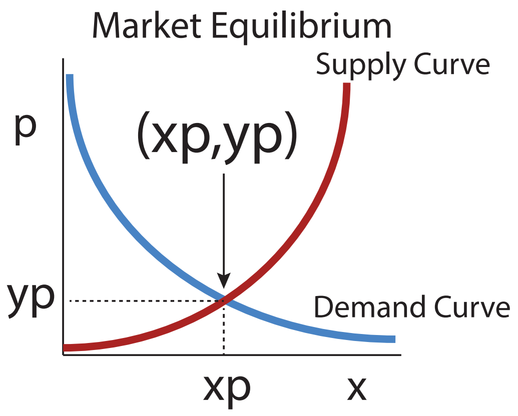

Figure 2: Market equilibrium with supply and demand

Problem 7: Predator-Prey Model

The Lotka-Volterra predator-prey model chacterizes two populations of animals–in this example, we'll assume the predator is a fox and the prey is a rabbit. We examine the total populations of each as they interact. For this, I'll write $r[t], f[t]$ to mean the populations of rabbit and fox respectively at $t = 0, 1, 2, 3, \ldots, k$. We're interested in the following, given $r[t], f[t]$ how do we compute $r[t+1], f[t+1]$? The models states that we must include four rates:

- The growth rate of the rabbit population. We'll use b_r for birth of rabbits.
- The death rate of rabbits due to fox predation. We'll use d_r to indicate this value.
- The death rate of foxes, *e.g.*, bad hunting skills, disease, scare food. We'll use d_f for this value.
- Then an odd rate: When foxes catch and consume rabbits, they have litters (kits). So, there's a rate of turning rabbits into foxes. We'll call this b_f.

The model is:

$$r[t+1] = \lceil r[t] + r[t]b_r - r[t]f[t]d_r \rceil \tag{50}$$

$$f[t+1] = \lceil f[t] + b_f d_r r[t]f[t] - f[t]d_f \rceil \tag{51}$$

We can ignore the [t] part and simply make a list of values–the location of the variables in the list is the time.

```
def rabbit_fox(br,dr,df,bf,rabbit,fox,time_limit):
    i = 0
    history_rabbit = []
    history_fox = []
    while i < time_limit:
        history_rabbit.append(rabbit)
        history_fox.append(fox)
        #marvelous code
    return history_rabbit,history_fox

br = 0.03
dr = 0.0004
df = 0.25
bf = 0.11
rabbit = 3000 #initial population size
fox = 200 #initial population size
time_limit = 2000
history_rabbit, history_fox = rabbit_fox(br,dr,df,bf,rabbit,fox, ↩
    time_limit)
```

produces

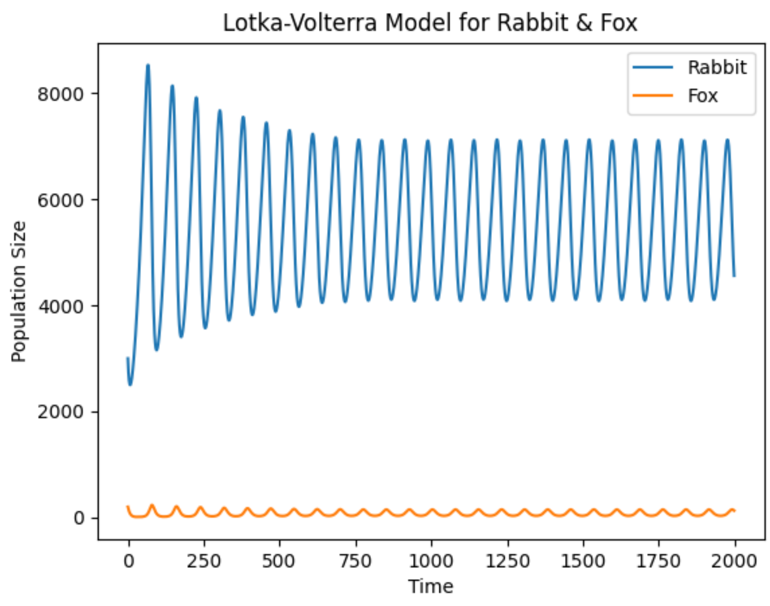

Figure 3: Predator-Prey Lotka-Volterra model for 2000 time points.

```
0 3000 200
200 5118 18
400 4496 150
600 6713 42
800 4274 46
1000 6204 140
1200 5902 33
1400 4094 77
1600 7050 94
1800 5132 31
```

which describes the time point, rabbit population, fox population at some points and the graphic in Fig. 3. Your task is to complete the #marvelous code using equations 50,51 and from inspection of the code. You can see the populations reach an equilibrium.

Deliverables for Problem 7

- Observe the ceiling function.
- You should experiement and see what are the populations constraints–does this *always* become stable?
- After testing the code but before submitting to the Autograder - make sure to comment lines 249-255 (the code used for creating the plot) because Autograder can not interpret visual output i.e. plots and will likely fail due to the code trying to display plots in the web browser.

Problem 8: Counts of Substring

We know that a string in Python is any number of characters enclosed by a set of double or single quotes: Given a string s a substring t is an any slice $t = s[i:j] \ \wedge \ len(t) \neq 0$ for non-

“abcabc”
“”
“ccccc”

negative integers i, j. For the initial strings the counts are shown below. Implement sub_string

“abcabc”	’a’, ’ab’, ’abc’, ’abca’, ’abcab’, ’abcabc’, ’b’, ’bc’, ’bca’, ’bcab’, ’bcabc’, ’c’, ’ca’, ’cab’, ’cabc’
“”	no substrings
“ccccc”	’c’, ’cc’, ’ccc’, ’cccc’, ’ccccc’

to take a string and return a dictionary of counts of all the substrings. For example:

“abc”	’a’: 1, ’ab’: 1, ’abc’: 1, ’b’: 1, ’bc’: 1, ’c’: 1
“”	
“ccccc”	’c’: 5, ’cc’: 4, ’ccc’: 3, ’cccc’: 2, ’ccccc’: 1

Deliverables for Problem 8

- Complete the function.
- You cannot use other string functions other than slicing (which is a strong hint).

Problem 9: Using Loop to Calculate Sinking Fund Schedule

A **sinking fund** is an financial instrument to discharge a future debt. For example, a manufacturing company expects to replace obsolescent machinary in some number of years. In this problem, you are building your finance company to compute payment schedules for a sinking fund. We use the general formula for annuity:

$$\begin{aligned} S &= R\frac{(1+i)^n - 1}{i} \\ i &= \frac{r}{m} \\ n &= my \end{aligned}$$

where S is the future debt, r the (percentage) interest rate, m the number of payments per year, y the number of years, n is the total number of payments (number of payments in a year multiplied by total number of years) and R is the monthly deposit made. Here is the particular problem. Acme Sundials believes replacing their sun dial mold-press will cost $30,000 in two years. They can currently get 10% interest compounded quarterly (four times a year). You build a Python program that, given the initial data will:

- Determine the monthly deposit (R).
- Determine the total schedule of payments that includes the interest accrued and total amount. This should be stored as a list of lists (see example output below).

Here is the run:

```
def deposit(S,r,i,n):
   pass

def sinking_fund(final_amt,r,m,y):
    pass

for i in sinking_fund(30000,.1,4,2):
    print(i)
```

with output:

```
[[0, 3434.02, 0, 3434.02],
[1, 3434.02, 85.85, 6953.89],
[2, 3434.02, 173.85, 10561.76],
[3, 3434.02, 264.04, 14259.82],
[4, 3434.02, 356.5, 18050.34],
[5, 3434.02, 451.26, 21935.62],
[6, 3434.02, 548.39, 25918.03],
[7, 3434.02, 647.95, 30000.0]]
```

- The function deposit returns the monthly deposit needed for the fund (R).
- Clearly you must solve for R (monthly deposit) first and use this value in sinking_fund() to solve for S.
- Once you have the deposit value, you should find n and i.
- After you have all the values, i..e, i, n, R, you can start calculating your payment schedule.
- Make sure that the payment schedule is stored as a list of lists where each sublist represent one payment as follows: [period, deposit, interest, total fund].

Observe there are eight payments. On the last payment, the fund is \$30,000. The first value in the list is the period, the second is the deposit amount, the third is the interest accrued on the **previous** fund, the last value is the curent total fund. Let's construct period 1 from 0. The total fund in period 0 is \$3434.02, or the initial deposit value for this particular example.

$$\begin{aligned} i \times fund &= \frac{r}{m} \times 3434.02 \\ &= \frac{.10}{4} \times 3434.02 = 0.025(3434.02) = 85.85 \end{aligned}$$

The total t_1 for that period is the previous total, plus interest, plus deposit:

$$t_1 = 3434.02 + 85.85 + 3434.02 = 6953.89$$

Deliverables for Problem 9

- Complete the functions deposit and sinking_fund.
- We are rounding to two decimal places all values.
- You are free to use a single bounded (hint), unbounded, nested loops–whatever you find most comfortable.

Problem 10: Geometric Progression

A geometric progression is described by $ar^0, ar^1, ar^2, \ldots$ for positive integers a, r. The ratio of successive terms is a constant *viz.*, $\frac{ar^1}{ar^0}, \frac{ar^2}{ar^1}, \ldots$. Thus, given a (finite) sequence we can determine whether it is geometric or not. In this problem you'll implement a function is_geometric_sequence(lst) that takes a possibly empty list of numbers. If there are at least three, then you can determine if it is a geometric series. For example,

```python
data = [[1,2,4,6],[2,4,8,16],[10,30,90,270,810,2430]]
for d in data:
    print(is_geometric_sequence(d))
```

produces

```
False
True
True
```

Deliverables for Problem 10

- Complete the function.
- As always you cannot use any exotic or functions not discussed in class.

Problem 11: Portfolio Valuation

In this problem, you'll implement a function value(portfolio, market) that takes a person's portfolio of stocks and the current market and determines the value as a percent. A stock is a portion of ownership of a business. Since it's a portion, it is a fraction of what the business is worth in total. A stock as a price per share as a dollar amount and the number of shares (how much of the portion is owned). For example, if you have a stock that you purchased for $2.25/share and you purchased 11 shares then $\$2.25 \times 11 = \$24.75$ in total. A portfolio consists of the stock name, share price that was purchased, and the number of shares. We'll model this a dictionary:

```
portfolios =  {'A':{'stock':{'x':(41.45,45),'y':(22.20,1000)}},
               'B':{'stock':{'x':(33.45,15),'y':(12.20,400)}}}
```

shows two portfolios, one for 'A' and one for 'B'. 'A' has 45 shares of 'x' valued at $41.45 and 'y' valued at $22.20 with 1000 shares. The total value is: $41.45(45) + $22.20(1000) = $24065.25. The market value for stocks changes. In this example, our market is:

```
market = {'x':43.00, 'y':22.50}
```

This means that 'A's holdings in 'x' is worth now $\$43.0 \times 45 = \$1935.0$ instead of $\$41.45 \times 45 = 1865.25$. The value of a portfolio is the ratio of $\frac{\text{Market}-\text{Portfolio}}{\text{Portfolio}}$. If we were solely looking at the 'x' stock, then

$$\frac{\text{Market} - \text{Portfolio}}{\text{Portfolio}} = \frac{43.00(45) - 41.45(45)}{41.45(45)} \quad (52)$$

$$\approx 3.7\% \quad (53)$$

which means the value today is 3.7% more. Implement the function value(portfolio, market) that determines the change in percentage of worth. For example,

```
portfolios =  {'A':{'stock':{'x':(41.45,45),'y':(22.20,1000)}},'B':{'↩
    stock':{'x':(33.45,15),'y':(12.20,400)}}}

market = {'x':43.00, 'y':22.50}

for name, portfolio in portfolios.items():
    print(f"{name} {value(portfolio,market)}")
```

produces

```
A 2.0
B 79.0
```

that gives 2% for A and 79% for B.

Deliverables for Problem 11

- Complete the function.
- The function returns a percent.
- Round the percentage change to two decimal places.

Student pairs

ameydesh@iu.edu, wtrucker@iu.edu
rcaswel@iu.edu, justyou@iu.edu
leegain@iu.edu, aveluru@iu.edu
ag69@iu.edu, blswing@iu.edu
vkommar@iu.edu, drsnid@iu.edu
lcoveney@iu.edu, sahishah@iu.edu
jtbland@iu.edu, aptheria@iu.edu
stkimani@iu.edu, ryarram@iu.edu
alelefeb@iu.edu, rnschroe@iu.edu, sezinnkr@iu.edu
aberkun@iu.edu, smremmer@iu.edu
gandhira@iu.edu, avraya@iu.edu
tychid@iu.edu, cjwaller@iu.edu
hawkjod@iu.edu, reedkier@iu.edu
cfampo@iu.edu, amyawash@iu.edu
mrcoons@iu.edu, pravulap@iu.edu
loggreen@iu.edu, dsummit@iu.edu
dkkosim@iu.edu, maklsmit@iu.edu
migriswo@iu.edu, jcn1@iu.edu
garcied@iu.edu, wlyzun@iu.edu
howamatt@iu.edu, vyeruba@iu.edu
arklonow@iu.edu, muyusuf@iu.edu
maudomin@iu.edu, reddyrr@iu.edu
nolakim@iu.edu, snyderjk@iu.edu
greenpat@iu.edu, lukastef@iu.edu
hermbrar@iu.edu, audtravi@iu.edu
ceub@iu.edu, voram@iu.edu
zguising@iu.edu, nichojop@iu.edu
braybrya@iu.edu, lvansyck@iu.edu
scbik@iu.edu, vmungara@iu.edu
agrevel@iu.edu, fmahamat@iu.edu
matgarey@iu.edu, isaramir@iu.edu
mbekkem@iu.edu, rosenbbj@iu.edu

deombeas@iu.edu, asultano@iu.edu
jacklapp@iu.edu, owysmit@iu.edu
evacoll@iu.edu, nlippman@iu.edu
kdembla@iu.edu, awsaunde@iu.edu
khannni@iu.edu, pricemo@iu.edu
tfreson@iu.edu, jwetherb@iu.edu
mdoxsee@iu.edu, pateishi@iu.edu
adhuria@iu.edu, evataylo@iu.edu
diebarro@iu.edu, ltmckinn@iu.edu
rl29@iu.edu, anemlunc@iu.edu
nokebark@iu.edu, orrostew@iu.edu
sydecook@iu.edu, keasandl@iu.edu
jdgonzal@iu.edu, emgward@iu.edu
kraus@iu.edu, lpelaez@iu.edu
skatiyar@iu.edu, tarturnm@iu.edu
jwcase@iu.edu, rorymurp@iu.edu
twfine@iu.edu, asaokho@iu.edu
harmonnc@iu.edu, shrrao@iu.edu
spgreenf@iu.edu, mzagotta@iu.edu
edfran@iu.edu, myeralli@iu.edu
lflenoy@iu.edu, violuway@iu.edu
jkielcz@iu.edu, anassour@iu.edu
kaneai@iu.edu, clscheum@iu.edu
colrkram@iu.edu, rafir@iu.edu
ryanbren@iu.edu, cmvanhov@iu.edu
huhasan@iu.edu, skp2@iu.edu
bkante@iu.edu, sasayini@iu.edu
hk120@iu.edu, patel89@iu.edu
mbrockey@iu.edu, krbpatel@iu.edu
caegrah@iu.edu, aselki@iu.edu
bellcol@iu.edu, patedev@iu.edu
tchapell@iu.edu, asidda@iu.edu
linjaso@iu.edu, myswill@iu.edu
cgkabedi@iu.edu, rpoludas@iu.edu
ahavlin@iu.edu, pp31@iu.edu
mohiambu@iu.edu, jnzheng@iu.edu
coopjose@iu.edu, amanocha@iu.edu
pyphealy@iu.edu, nmr1@iu.edu
willhowa@iu.edu, thnewm@iu.edu
abellah@iu.edu, wtubbs@iu.edu
cuizek@iu.edu, vpolu@iu.edu

liansia@iu.edu, epautsch@iu.edu
egoldsto@iu.edu, jactrayl@iu.edu
blacount@iu.edu, tzuyyen@iu.edu
jabbarke@iu.edu, jneblett@iu.edu
daxbills@iu.edu, emluplet@iu.edu
adwadash@iu.edu, ajtse@iu.edu
sgaladim@iu.edu, gavilleg@iu.edu
mwclawso@iu.edu, nrizvi@iu.edu
mrfehr@iu.edu, dernguye@iu.edu
coopelki@iu.edu, savebhat@iu.edu
kekchoe@iu.edu, dmmullin@iu.edu
zacbutle@iu.edu, ap79@iu.edu
aakindel@iu.edu, thomps16@iu.edu
brhint@iu.edu, jtsuter@iu.edu
kjj6@iu.edu, mszczas@iu.edu
kapgupta@iu.edu, mnimmala@iu.edu
simadams@iu.edu, ammulc@iu.edu
ethbrock@iu.edu, jwu6@iu.edu
aaragga@iu.edu, nsatti@iu.edu
nihanas@iu.edu, lmadiraj@iu.edu
mkames@iu.edu, dukthang@iu.edu
hamac@iu.edu, tt13@iu.edu
swconley@iu.edu, gavsteve@iu.edu
ek37@iu.edu, jdw14@iu.edu
wilcusic@iu.edu, cstancom@iu.edu
masharre@iu.edu, joshroc@iu.edu
ageorgio@iu.edu, emisimps@iu.edu
wanjiang@iu.edu, rt11@iu.edu
bcdutka@iu.edu, apathma@iu.edu
dja1@iu.edu, lmeldgin@iu.edu
eakanle@iu.edu, wardjohn@iu.edu
arnadutt@iu.edu, giomayo@iu.edu
ohostet@iu.edu, fshamrin@iu.edu
sfuneno@iu.edu, aidnschi@iu.edu
daminteh@iu.edu, vrradia@iu.edu
skunduru@iu.edu, sahimann@iu.edu
ckdiallo@iu.edu, ansakrah@iu.edu
jhar@iu.edu, dwo@iu.edu
anrkram@iu.edu, pw18@iu.edu
quecox@iu.edu, ism1@iu.edu
nbernot@iu.edu, gepearcy@iu.edu

jc168@iu.edu, jlopezmo@iu.edu
krisgupt@iu.edu, patekek@iu.edu
dce@iu.edu, mmarotti@iu.edu
flynncj@iu.edu, cinivo@iu.edu
joehawl@iu.edu, ao9@iu.edu
gcopus@iu.edu, mmunaf@iu.edu
adiyer@iu.edu, kvpriede@iu.edu
grafe@iu.edu, ijvelmur@iu.edu
phjhess@iu.edu, mz24@iu.edu
leokurtz@iu.edu, btasa@iu.edu
leolin@iu.edu, jsadiq@iu.edu
dblackme@iu.edu, ysanghi@iu.edu
achordi@iu.edu, ksadiq@iu.edu
mdonato@iu.edu, ntorpoco@iu.edu
fkeele@iu.edu, annaum@iu.edu
fkanmogn@iu.edu, zshamo@iu.edu
davgourl@iu.edu, anajmal@iu.edu
fu7@iu.edu, etprince@iu.edu
aketcha@iu.edu, majtorm@iu.edu
nfelici@iu.edu, ntuhl@iu.edu
efritch@iu.edu, hasiddiq@iu.edu
aroraarn@iu.edu, aranjit@iu.edu
mbeigie@iu.edu, mjroelle@iu.edu
wgurley@iu.edu, bdyiga@iu.edu
amkhatri@iu.edu, megapaul@iu.edu
alscarr@iu.edu, rtrammel@iu.edu
jdemirci@iu.edu, jpochyly@iu.edu
jwdrew@iu.edu, clmcevil@iu.edu
kaihara@iu.edu, jaslnu@iu.edu
ridbhan@iu.edu, erschaef@iu.edu
makinap@iu.edu, surapapp@iu.edu
nfarhat@iu.edu, deturne@iu.edu
cpkerns@iu.edu, schwajaw@iu.edu
ayoajayi@iu.edu, jlzhao@iu.edu
maladwa@iu.edu, tolatinw@iu.edu
nmcastan@iu.edu, mveltri@iu.edu
agawrys@iu.edu, impofujr@iu.edu
spgerst@iu.edu, bcmarret@iu.edu
bencalex@iu.edu, antando@iu.edu
bjdahl@iu.edu, samrile@iu.edu
lawmat@iu.edu, madymcsh@iu.edu

tcconnol@iu.edu, cialugo@iu.edu
kyhigg@iu.edu, tangtom@iu.edu
nharkins@iu.edu, samyuan@iu.edu
marcchao@iu.edu, ragmahaj@iu.edu
jakchap@iu.edu, coenthom@iu.edu
mkleinke@iu.edu, abiparri@iu.edu
micahand@iu.edu, gsilingh@iu.edu
alchatz@iu.edu, iperine@iu.edu
allencla@iu.edu, awolkind@iu.edu
delkumar@iu.edu, cmarcuka@iu.edu
althart@iu.edu, mehtriya@iu.edu
laharden@iu.edu, ruska@iu.edu
elybrewe@iu.edu, gmpierce@iu.edu
anlego@iu.edu, wilsonnf@iu.edu
maxklei@iu.edu, aamathew@iu.edu
apbabu@iu.edu, qshamsid@iu.edu
hh35@iu.edu, jacmanto@iu.edu
spdamani@iu.edu, maglowe@iu.edu
avulas@iu.edu, brayrump@iu.edu
ajgrego@iu.edu, bmpool@iu.edu
mdiazrey@iu.edu, trenstev@iu.edu
earuland@iu.edu, jwember@iu.edu
apchavis@iu.edu, jarlmint@iu.edu
johnguen@iu.edu, wtatoole@iu.edu
cannan@iu.edu, chrimanu@iu.edu
zhatfie@iu.edu, crmoll@iu.edu
ethickma@iu.edu, patelsak@iu.edu
seangarc@iu.edu, sahaan@iu.edu
clearle@iu.edu, woodsky@iu.edu
ajeeju@iu.edu, ir1@iu.edu
austdeck@iu.edu, aditpate@iu.edu
sakalwa@iu.edu, scotbray@iu.edu
saecohen@iu.edu, rwan@iu.edu
keswar@iu.edu, utwade@iu.edu
ovadeley@iu.edu, cltran@iu.edu
ejhaas@iu.edu, adsize@iu.edu
jhhudgin@iu.edu, masmatth@iu.edu
fdonfrio@iu.edu, asteini@iu.edu
ddrotts@iu.edu, aledminc@iu.edu
bencho@iu.edu, apavlako@iu.edu
josespos@iu.edu, cnyarko@iu.edu

ruqchen@iu.edu, nniranj@iu.edu
sg40@iu.edu, jwmullis@iu.edu
milhavil@iu.edu, marystre@iu.edu
schinitz@iu.edu, tpandey@iu.edu
aaamoako@iu.edu, rvinzant@iu.edu
marganey@iu.edu, benprohm@iu.edu
jdc6@iu.edu, kamaharj@iu.edu
brownset@iu.edu, jomeaghe@iu.edu
criscruz@iu.edu, wilsori@iu.edu
oeichenb@iu.edu, gszopin@iu.edu
micedunn@iu.edu, perezand@iu.edu
gmhowell@iu.edu, gs29@iu.edu
jacobben@iu.edu, gmichna@iu.edu
escolber@iu.edu, arirowe@iu.edu
celechav@iu.edu, dyashwar@iu.edu
howardbw@iu.edu, wwtang@iu.edu
saganna@iu.edu, emmmccle@iu.edu
oakinsey@iu.edu, wattsra@iu.edu
lpfritsc@iu.edu, liwitte@iu.edu
laburkle@iu.edu, lqadan@iu.edu