C200 Programming Assignment № 3

Dr. M.M. Dalkilic
Computer Science
School of Informatics, Computing, and Engineering

Indiana University, Bloomington, IN, USA

September 24, 2023

Due Date: October 1st, 2023 at 11:00 PM. **Please start this homework in a day or so.** Do not rush solving–it makes for unpleasant experience and interferes with learning. Please submit your work to the Autograder: `https://c200.luddy.indiana.edu/` and push to GitHub before the due date. **Student Pairs** are provided at the end of this document.

Instructions for submitting to the Autograder

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. 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.
4. Once you are done testing your code, comment out the tests i.e. the code under the __name__ == "__main__" section.

Some reminders from lecture

Please read through the problems carefully.

- A **constant** function is a function whose output value is the same for every input value. For example: f(x) = 3, irrespective of what we plug in for x, the function f will allways output 3.
- Now we know that a **constant** (or fixed) function $f(x)$ returns the same constant value, *e.g.*,

$$f(x, y) = 3 \tag{1}$$

  No matter the inputs, the value is three.

- We can convert between $\log_a, \log_k$:

$$\log_b(x) = \frac{\log_k(x)}{\log_k(b)} \tag{2}$$

  Why? Remember that if $\log_b(x) = r$, then $b^r = x$. So, let's first write

$$\log_b(x) = r \tag{3}$$
$$\log_k(x) = s \tag{4}$$
$$\log_k(b) = t \tag{5}$$

  This means

$$b^r = x \tag{6}$$
$$k^s = x \tag{7}$$
$$k^t = b \tag{8}$$

  We see that $b^r = x = k^s$. Since $b = k^t$, we can write:

$$(k^t)^r = k^{rt} = x = k^s \tag{9}$$

  Then

$$\log_k(k^{rt}) = \log_k(x) = \log_k(k^s) \tag{10}$$
$$rt = \log_k(x) = s \tag{11}$$

  Using equations 3,5 for r, t we have

$$\log_b(x) \log_k(b) = \log_k(x) \tag{12}$$
$$\log_b(x) = \frac{\log_k(x)}{\log_k(b)} \tag{13}$$

Problem 1: Functions and math module

All the following functions are drawn from real-world sources. This is an exercise for you to learn how to use a module on your own. From the math module use

- `math.exp(x)` for e^x
- `math.ceil(x)` for $\lceil x \rceil$ rounds x UP to the nearest integer k such that $x \leq k$
- `math.log(x)` for $\log_e(x) = \ln(x)$.

1. According to the Center for Disease Control (CDC) the bacteria *Salmonella* causes about 20K hospitalizations and nearly 400 deaths a year. The formula for how fast this bacteria grows is:

$$N(n_0, m, t) = n_0 e^{mt} \tag{14}$$
$$\tag{15}$$

   where n_0 is the initial number of bacteria, m is the growth rate *e.g.*, 100 per hr, and t is time in hours. Here is how you would calculate the size for an initial colony of 500 that grows at the rate of 100 per hr, for four hours.

$$N(500, 100, 4) = 2.610734844882072 \times 10^{176} \tag{16}$$

2. The number of teeth $N_t(t)$ after t days from incubation for *Alligator mississippiensis* is:

$$N_t(t) = 71.8e^{-8.96e^{-0.0685t}} \tag{17}$$
$$N_t(1000) = \lceil 71.8 \rceil = 72 \tag{18}$$

3. If we want to calculate the work done when an ideal gas expands isothermally (and reversibly) we use for initial and final pressure $P_i = 10\ bar, P_f = 1\ bar$ respectively at $300^o K$. In this problem we are using $\ln$ which is $\log_e$ ('math.log uses base e by default'). Because $\log_e$ is used so often, you'll see it just as often abbreivated as $\ln$.

$$W(P_i, P_f) = RT \ln(P_i/P_f) \tag{19}$$
$$W(10, 1) = \lceil 8.314(300)(\ln 10) \rceil = 5744 \tag{20}$$

   at temperature T (Kelvin) and $R = 8.314\ J/mol$ the universal gas constant.

4. The Wright Brothers are known for their Flyer and its maiden flight. The formula for lift is:

$$L(V, A, C_\ell) = kV^2 AC_\ell \tag{21}$$
$$L(33.8, 512, 0.515) = \lceil 0.0033(33.8)^2(512)0.515 \rceil = 995 \tag{22}$$

where k is Smeaton's Coefficient ($k = 0.0033$ from their wind tunnel), $V = 33.8\ mph$ is relative velocity over the wing, $A = 512\ ft$ area of wing, and $C_\ell = 0.515$ coefficient of lift. The Flyer weighed $600\,lbs$ and Orville was about $145\,lbs$. We can see that the lift was sufficient since $995 > 745$.

Deliverables for Problem 1

- Complete $N(), N_t(), W(), and\ L()$ functions described above.
- Don't forget to round up using $math.ceil(x)$ as indicated.

Problem 2: More Quadratic

We saw, and also know from basic algebra that for $ax^2 + bx + c = 0$ the roots (values that make the equation zero) are given by:

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{23}$$

The expression $b^2 - 4ac$ is called the discriminant. By looking at the discriminant we can deteremine properties of the roots:

- If $b^2 - 4ac > 0$, then both roots are real
- If $b^2 - 4ac = 0$, then both roots are $-b/2a$
- If $b^2 - 4ac < 0$, then both roots are imaginary

There are other properties we can determine from the coeficients. For example, the quadratic as a maximum when $a < 0$, since the graph opens down; the graph opens up when $a > 0$. The vertex of the parabola–the maximum or minimum–is given by:

$$v((a, b, c)) = \frac{-b}{2a} \tag{24}$$

where the coefficients are given as a tuple (a, b, c). By substituting the vertex, you'll find the y-coordinate:

$$f(v((a, b, c))) = f(\frac{-b}{2a}) \tag{25}$$

For example, if $f(x) = -2.6x^2 + 7.6x - 10$, then the maximum point is:

$$(v((a, b, c)), f(v((a, b, c)))) = (\frac{-7.6}{2(-2.6)}, f(\frac{-7.6}{2(-2.6)})) \tag{26}$$

$$\approx (1.46, -4.45) \tag{27}$$

Another example, for $f(x) = x^2 - 10.2 + 26.01$ opens up and has vertex at (5.1,0).

For this function, you'll return a tuple (x,y) where x is a string that says either “down" or “up" and y is a tuple (vertex, f(vertex)) for the quadratic function f.

Deliverables for Problem 2

- Complete the functions.
- You are not allowed to use the module `cmath`.
- You will not be asked to examine roots with complex answers.
- When **returning** the vertex,y-coordinate pair, round to two decimal places.

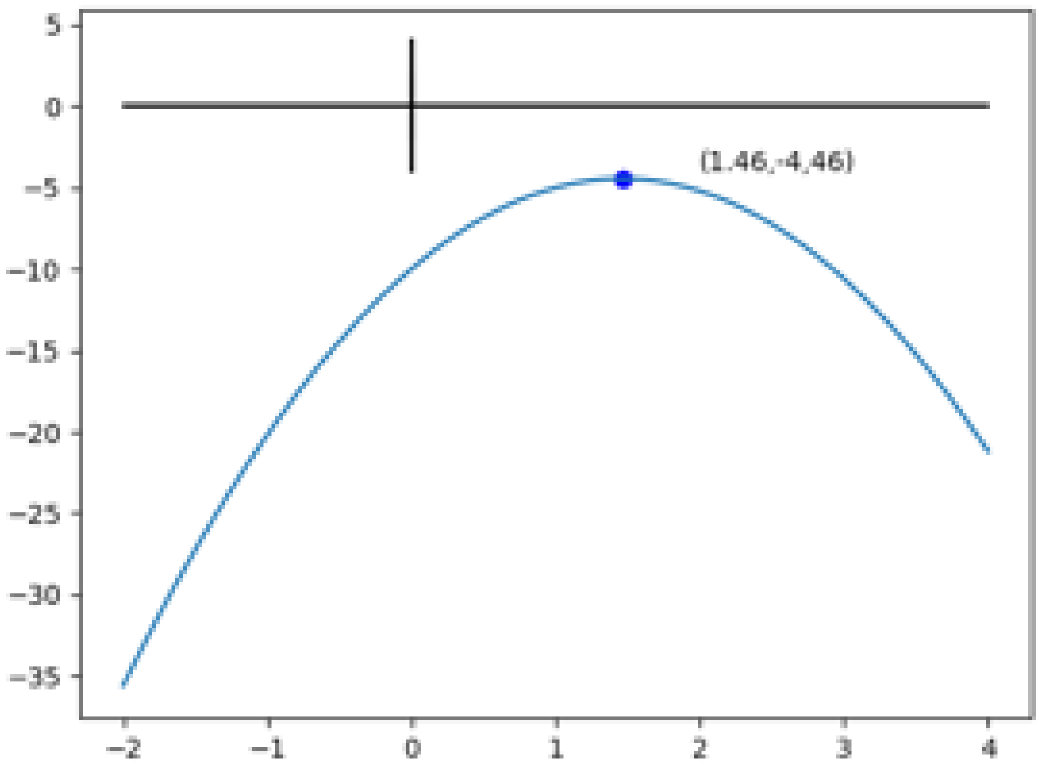


Figure 1: Graphing $f(x) = -2.6x^2 + 7.6x - 10$ observe whethe it opens down (it does) or up and the vertex, y-coordinate pair.

Problem 3: Something Wicked This Way Comes

Search on this phrase "what food isn't taxed in Indiana". You're writing software that allows for customer checkout at a grocery store. A customer will have a have receipt r which is a list of pairs $r = [[i_0, p_0], [i_1, p_1], \ldots, [i_n, p_n]]$ where i is an item and p the cost. You have access to a list of items that are not taxed $no\_tax = [j_0, j_1, \ldots, j_m]$. The tax rate in Indiana is 7%. Write a function amt that takes the receipt and the items that are not taxed and gives the total amount owed. For this function you will have to implement the member function $m(x, lst)$ that returns True if x is a member of lst.

For example, let (receipt) $r = [[1, 1.45], [3, 10.00], [2, 1.45], [5, 2.00]]$, $tax\_rate = \frac{7}{100}$ and $no\_tax = [33, 5, 2]$. Then

$$amt(r, no\_tax) = round(((1.45 + 10.00)1.07 + 1.45 + 2.00), 2) \tag{28}$$
$$= \$15.7 \tag{29}$$

If all items are taxable (for the same receipt), but the tax is 10%, then the total will be \$16.39.

Deliverables for Problem 3

- Complete $m(x, lst)$ function. You must search for x in lst by looping through lst
- You are free to use either an iterable or subscripting
- Complete the $amt()$ function.
- You cannot use the Python's x **in** y predicate. For example, you cannot use

```
>>> x = [1,2,3,4]
>>> if 1 in x:
...     "gotcha"
...
'gotcha'
>>>
```

  If you use Python's x **in** y member function, you will recieve a 0.

- Round the output of amt() function to 2 decimal places.

Problem 4: 2D intersecting lines

A line in 2D Euclidean space can be described by $y = mx + b$ (m and b are the slope and intercept respectively). You are provided with a function that takes two points and returns the line as a dictionary:

```python
#INPUT two points in 2D
#OUTPUT a dictionary {'m':m, 'b':b} holding slope & intercept
#values are rounded to two decimal places
def make_line(p0,p1):
    x0,y0,x1,y1 = *p0,*p1
    m = round((y1 - y0)/(x1 - x0),2)
    b = round(y0 - (m*x0),2)
    return {'m':m, 'b':b}
```

We'll assume all lines are functions. You'll implement a function that takes two lines described above and return the point of intersection (x, y). Here is the function on two inputs:

```python
p0 = (32,32)
p1 = (29,5)
p2 = (15,10)
p3 = (49,25)
p4 = (15,30)
p5 = (50,15)

l0,l1 = make_line(p0,p1),make_line(p2,p3)
print(intersection(l0,l1))
l0 = make_line(p4,p5)
print(intersection(l0,l1))
```

has output:

```
(30.3, 16.73)
(37.99, 20.11)
```

The other two outcomes are either the lines are the same or parallel. In this case return the strings “same lines" or “parallel lines" as shown below:

```python
p6,p7,p8 = (0,0),(1,1),(2,2)
p9,p10 = (0,1),(1,2)
print(intersection(make_line(p6,p7),make_line(p7,p8))) # same line
print(intersection(make_line(p6,p7),make_line(p9,p10))) # parallel lines
```

has outcome:

```
same line
parallel lines
```

Deliverables for Problem 4

- Complete the function.
- You cannot use any math module or cmath module function.
- Round the values of slope and intercept to two decimal places.

Problem 5: Means

When analyzing data, we often want to summarize it: make it concise. Each of these functions takes a list of n numbers as nlst. The mean of a list of numbers gives a summary through one number. You're probably aware of the arithmetic mean:

$$\text{arithmetic\_mean(nlst)} = \frac{(x_0 + x_1 + \cdots + x_{n-1})}{n} \tag{30}$$

For example, the arithmetic mean of [1,2,3] is 2.0.

The geometric mean (usually done with logs) is:

$$\text{geo\_mean(nlst)} = a^{sum/n} \tag{31}$$

$$sum = \log_a(x_0) + \log_a(x_1) + \cdots + \log_a(x_{n-1}) \tag{32}$$

where $\log_a$ is an arbitrary log to base a. For example, the geometric mean of [2,4,8] is 4.0. Use $\log_{10}$ as default–but it doesn't actually matter. The harmonic mean is:

$$\text{har\_mean(nlst)} = \frac{n}{1/x_0 + 1/x_1 + \cdots + 1/x_{n-1}} \tag{33}$$

For example, the harmonic mean of [1,2,3] is approximately 1.64.

The root mean square is:

$$\text{RMS\_mean(nlst)} = \sqrt{\frac{sum}{n}} \tag{34}$$

$$sum = x_0^2 + x_1^2 + \cdots + x_{n-1}^2 \tag{35}$$

For example, the root mean square of [1,3,4,5,7] is approximately 4.47.

All of these functions take a (possibly empty) list of numbers. If there is a list of numbers, then return the mean. If the list is empty, return the string **"Data Error: 0 values"**. If there is a zero in the list of numbers for the harmonic mean, then return the string **"Data Error: 0 in data"**. We give these two errors, because we face division by zero if we don't.

Deliverables for Problem 5

- Complete the functions as specified above.
- You cannot use the Python's **in** keyword to search for 0s.
- In case of empty list or 0 in harmonic mean-return the error string exactly as mentioned in the problem description. Do not add any extra white spaces.

Problem 6: Occurs

In this problem you'll write one function occur_at_least(x,lst,y) that returns True if object x occurs at least y times in lst, False otherwise. We assume lst is either a list or a tuple. Here are a couple of runs:

```
data = [[1,[1,2,1,2,1,1],4], [1,[1,2,1,2,1,1],3],
        [1,(1,2,1,2,1,0),4], ]

for d in data:
    print(occur_at_least(*d))
```

has output:

```
True
True
False
```

Deliverables for Problem 6

- Complete the function.

Problem 7: Looping

We will describe succinctly each kind of function and the type of looping you're required to use. The first function occurs_more(x,y,lst) takes two objects, x,y and a list lst and returns True if x occurs strictly more than y in lst. If the list is empty, then it should return True (since it's not provably false). The second function equal_remove(x,y,lst) returns a list where the number of occurrences of x,y in lst are the equal. If x occurs more, then it is removed from the lst (from left to right) until it occurs the same amount as y. If y occurs, more, then similarly. For example, the following code:

```
    print(occurs_more(1,2,lst))
    print(occurs_more(2,3,lst))
    print(occurs_more(2,3,[]))


    print(equal_remove(1,2,lst))
    print(equal_remove(1,3,lst))
    print(equal_remove(2,3,lst))
    print(occurs_more(2,3,(equal_remove(2,3,lst))))
```

produces

```
False
True
True
[2, 3, 1, 2, 1, 1, 2]
[2, 2, 3, 2, 1, 2]
[3, 1, 1, 1, 2]
False
```

When implementing this function, you're encouraged to use local helper functions. In my code I have this:

```
def equal_remove(x,y,lst):
   tmp = lst
   def cnt_occurs(x,lst): #returns the count of x in lst
        pass

   x_c,y_c = cnt_occurs(x,lst),cnt_occurs(y,lst) #find the counts

   def re_k(x,cnt,lst):  #removes the first cnt occurrences of x in lst
        pass


```

```python
    if x_c < y_c:
        tmp = re_k(y,y_c-x_c,tmp)
    elif x_c > y_c:
        tmp = re_k(x,x_c - y_c,tmp)

    return tmp
```

This made solving the problem *much* easier. You're free to solve it how you'd like.

Deliverables for Problem 7

- Complete the functions.
- Use either subscripting or iterables

Problem 8: Geometric Series

A geometric series is the sum of an infinite number of terms that have a constant ratio between successive terms. For example,

$$\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} \tag{36}$$

is apparently a geometric series since:

$$\frac{1/4}{1/2} = \frac{1/8}{1/4} = \frac{1/16}{1/8} = \frac{1}{2} \tag{37}$$

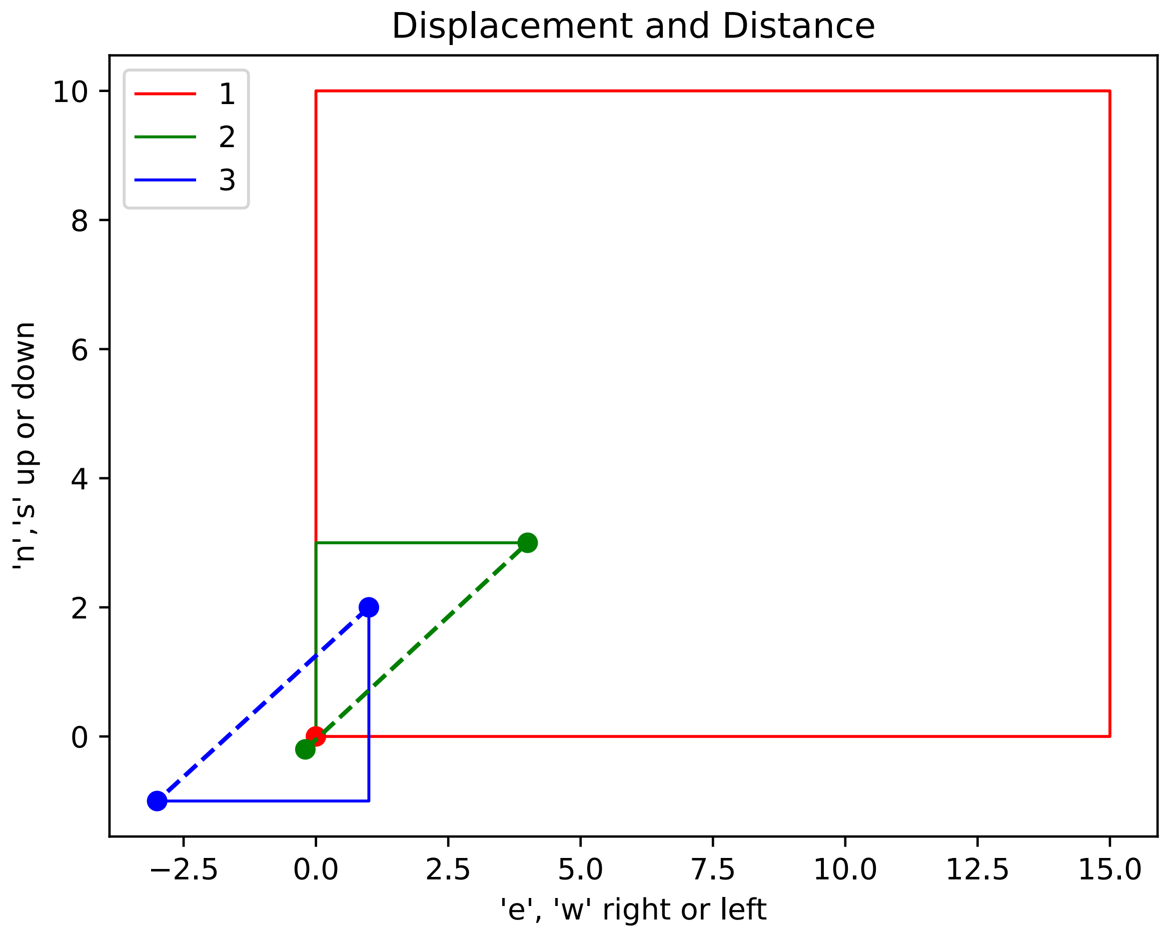
Write a function $is\_geo(xlst)$ that takes a list of numbers and returns 1 only if the series is geometric; otherwise 0. If the list has 2 or fewer members, is_geo returns 0. For example,

$$\begin{aligned}
is_geo([1/2, 1/4, 1/8, 1/16, 1/32]) &= 1 && (38)\\
is_geo([1, -3, 9, -27]) &= 1 && (39)\\
is_geo([625, 125, 25]) &= 1 && (40)\\
is_geo([1/2, 1/4, 1/8, 1/16, 1/31]) &= 0 && (41)\\
is_geo([1, -3, 9, -26]) &= 0 && (42)\\
is_geo([625, 125, 24]) &= 0 && (43)\\
is_geo([1/2, 1/4]) &= 0 && (44)
\end{aligned}$$

Deliverables for Problem 8

- Complete the function
- Assume none of the numbers are zero
- If the series has 2 or fewer the function returns 0

Problem 9: Tracking movement of objects



Each color is a run. Since green and red start at the same location, the green is slightly offset so it is visible. The dots show the starting and ending positions. Blue and green have dashed lines, because their starting and ending points are different. The dashed lines both have size 5.

For this problem, we're tracking movement of objects. This is first step in autonomous vehicle movement. You'll write two functions. The first function determines the distance in 2D space:

$$net\_displacement(p_0, p_1) \quad = \quad [(x_0 - x_1)^2 + (y_0 - y_1)^2]^{1/2}$$

where $p_0 = (x_0, y_0)$ and $p_1 = (x_1, y_1)$. The second function called track(start_pos, movement) takes a starting point (x, y) and a list of movements of size one. The list is the first letter of "east", "west", "north", "south". If the current position is (x, y), then each letter changes the values:

'e'	move right (add one to x)
'w'	move left (subtract one from x)
's'	move down (subtract one from y)
'n'	move up (add one to y)

For example, if you're starting at $(3, 0)$ then ['e','e','w','s','s','s'] results in $(4, -3)$. The total distance travelled is 6 (units). The function returns a tuple: the current location, the distance traveled, and the actual distance between the start and end points. We use net displacement to find how far we are from our starting position. The graphs below show three calls: red, blue, green. Look at the calls of the function with the graphic (built from Python too!). Here are the three calls:

```
data_m = [[(0,0), list(10*'n' + 15*'e' + 10*'s'+15*'w')],
          [(0,0), list(3*'n' + 4*'e')],
          [(1,2), list(3*'s' + 4*'w')]]

for d in data_m:
    print(track(*d))
```

has output:

```
((0, 0), 50, 0.0)
((4, 3), 7, 5.0)
((-3, -1), 7, 5.0)
```

So, the first run started at (0,0), took 50 steps, and ended-up back where it started. To make encoding more convenient, we take advantage of typecasting from strings to lists. For a string “abc...xyz”, list(“abc...xyz”) returns [‘a’,‘b’,‘c’,...,‘x’,‘y’,‘z’].

Deliverables for Problem 9

- Complete the two functions.
- Round the output of net_displacement to two decimal places.

Problem 10: Distance between two objects

Assume we are tracking two vehicles using the system in problem 9. Write a function final_distance(m0,m1) that takes two tuples (outputs from track) and determines the final distance between the two vehicles. For example,

```
data_m = [[(0,0), list(10*'n' + 15*'e' + 10*'s'+15*'w')],
          [(0,0), list(3*'n' + 4*'e')],
          [(1,2), list(3*'s' + 4*'w')]]

print(final_distance(track(*data_m[1]),track(*(data_m[2]))))
```

gives output:

```
8.06
```

because

$$net\_displacement((4,3),(-3,-1)) \quad = \quad [(4-(-3))^2+(3-(-1))^2]^{1/2} = [49+16]^{1/2} \approx 8.06$$

As a reminder, the distance between two objects is computed generally as

$$net\_displacement(p_0,p_1) \quad = \quad [(x_0-x_1)^2+(y_0-y_1)^2]^{1/2}$$

Deliverables for Problem 10

- Complete the function.
- While you are free to fully implement this, problem 9 already has the function(s) you need obviating a lot of coding. You can call the same functions to solve this function and only write new code as needed.

Problem 11: Big 10 Women's College Hoops: IU

The IU women's team is ranked 2nd. Tonight they beat Iowa ranked 5th. You'll write a function update(conference, game) that updates the conference standings using the output of the game. A game is a dictionary {team0:score0, team1:score1, ... } where team0,team1 are strings and score0,score1 are non-negative integers. For this problem, you'll have to loop over the elements of the dictionary. To remind you, for a dictionary d

- $d.keys()$ returns an iterable of dictionary keys
- $d.values()$ returns an iterable of dictionary values
- $d.items()$ returns an iterable of tuples that are key,value pairs

Since the dictionary is mutable, if you send it as a parameter and change it in the function, it'll be changed globally. The function does **not** return any value. The function updates the wins, losses ('W','L') and percentage wins: $\frac{wins}{wins+losses}$ ('PCT'), while retaining the number of games played at their home campus versus off-campus ('Home'). Here's a run (some of the data is modified to make the problem easier)

```
big_10_women = {'IU':{'W':12,'L':1,'PCT':.923, 'Home':(13,0)},
                'PU':{'W':6,'L':6,'PCT':.500, 'Home':(8,4)},
                'IOWA':{'W':11,'L':1,'PCT':.917, 'Home':(11,1)},
                'NW':{'W':1,'L':11,'PCT':.083,'Home':(6,6)}}

print(big_10_women['IU'],big_10_women['IOWA'])
update(big_10_women,{'IU':87,'IOWA':78})
print(big_10_women['IU'],big_10_women['IOWA'])
```

has output:

```
{'W': 12, 'L': 1, 'PCT': 0.923, 'Home': (13, 0)} {'W': 11, 'L': 1, 'PCT': 0.917, 'Home': (11, 1)}
{'W': 13, 'L': 1, 'PCT': 0.929, 'Home': (13, 0)} {'W': 11, 'L': 2, 'PCT': 0.846, 'Home': (11, 1)}
```

You can see we added one to the wins for IU, changed the percentage, added one to the losses for Iowa, and changed the percentage.

Deliverables for Problem 11

- Complete the function.
- Return the dictionary (conference) after updating it with the game info i.e. wins, loses and PCT.

Problem 12: Go Fund Me

In this problem, you'll be writing code that takes donations for a goal amount that is initially posted. The values are assumed to be dollars. The list contains the individual donations. You'll keep taking donations until the goal is met. The function returns a tuple: the first value is the original goal, the second value is the list of possible donations that are left if the goal is minimally met, and the last value is the amount that is needed to meet the goal. If the goal is not met, the last value returned should be negative. For this problem, using enumerate is very useful. Here are a couple of runs:

```python
data = [[100,[10,15,20,30,29,13,15,40]],
          [100,[]],
          [100,[30,4]]]

for d in data:
    print(go_fund_me(*d))
```

has output:

```
(100, [13, 15, 40], 4)
(100, [], -100)
(100, [], -66)
```

The first tuple says the goal of $100 has been met with 10+15+20+30+29 = 104. No more donations are needed. The remainder [13,15,40] are returned to the people and 4 is the amount over. For the second donation, nobody donated so the goal remains unchanged. For the third, only two people donated. The goal that's left is $66.

Deliverables for Problem 12

- Complete the function.

Problem 13: FICO Credit Score

Your access to credit is determined by your FICO (Fair Isaac Corporation) credit score. This is a number [0,850] that reflects your creditworthiness (a number reflecting what is believed to be the probability you'll not honor your debt obligations–in other words, if you borrow, will you pay it back? This is determined through three ways: model (using AI, statistics, historical data), a human (using tools), or hybrid (both). Five inputs P,A,L,N,C are weighted and summed:

- (.35) P (Payment History)
- (.30) A (Amounts Owed)
- (.15) L (Length of Credit History)
- (.10) N (New Credit)
- (.10) C (Credit Mix)

We won't go into detail here, but certainly the first two are the most significant. In this problem we'll write a function loan(cs, lst) which take a credit score and lst as [n,cd] where n is a name and cd is a dictionary that contains a person's five inputs (unweighted). The function loan returns the list of names of people who are will be given a loan through our bank Republic United National Bank or RUN B. Here is a run of on some applicants:

```
data = [['x',{'P':600, 'L':700,'A': 500, 'N': 170, 'C': 250}],
        ['y',{'P':550, 'L':720,'A': 500, 'N': 230, 'C': 250}],
        ['b',{'P':560, 'L':710,'A': 500, 'N': 221, 'C': 250}],
        ['c',{'P':800, 'L':700,'A': 200, 'N': 100, 'C': 150}],
        ['a',{'P':800, 'L':800,'A': 600, 'N': 250, 'C': 150}],
        ['z',{'P':800, 'L':800,'A': 500, 'N': 250, 'C': 150}]]
print(loan(550,data))
```

producing:

```
[['a', 620.0], ['z', 590.0]]
```

Deliverables for Problem 13

- Complete the function
- The model is very conservative. How might you modify it to find more people?

Problem 14: Agatha Christie and Miss Marple

Agatha Christie is among the most widely read mystery novelist. The Miss Marple series is my favorite. In these charming novels, there always is a *murder* to be solved. Let's visit a mystery to be solved with data science and computing.

We have three suspects: Ursala, the famous Malamute animal; Kaiser, her seemingly good-natured German Shephard assistant, Shilah, the sister of Ursala, but kind of a wild child. When our eccentric detective Dr. D arrives on the scene, at 4:00P, the body of a gold fish is found–a large goldfish. The temperature is taken and found to be 81^o F. The dwelling is much colder than the average home–the thermostat says 65^o F. After the suspects are given a cookie and asked to go outside in the backyard, the temperature of the hapless water breather is taken 1 hour later and found to be 77^o F. We find out that the normal body temperature of the gold fish is 85^o F

We have a model that describes how something warmer than the environment becomes cooler.

$$T(t) = T_e + (T_0 - T_e)e^{-kt} \tag{45}$$

where $T(t)$ is the temperature at time t, T_e is the steady temperature of the environment, T_0 is the initial temperature of what's being studied, and k a constant that has to be determined. The data for this problem is:

$$81 = 65 + (85 - 65)e^{-kt} \tag{46}$$

when the fish was discovered. An hour later we found

$$77 = 65 + (85 - 65)e^{-k(t+1)} \tag{47}$$

We have two equations with two unknowns. The suspects couldn't be account for during these times:

Name	Missing
Ursala	3:00P to 4:00P
Kaiser	1:00P to 2:00P
Shilah	2:00P to 2:30P

When we run the numbers, the evidence points to Ursie–she's absconded with her food bowl and my credit cards–be on the lookout.

Deliverables for Problem 14

- Complete the function time that returns how many hours elapsed.
- To do that, you'll need to calculate k. Find k to six decimal places
- The starter code includes the math module. To remind you, math.log() is the computational representative of $\ln()$ (or $\log_e()$).

Student Pairs

huhasan@iu.edu, crmoll@iu.edu
phjhess@iu.edu, ajtse@iu.edu
zguising@iu.edu, vmungara@iu.edu
amkhatri@iu.edu, samrile@iu.edu
deombeas@iu.edu, zshamo@iu.edu
leokurtz@iu.edu, cinivo@iu.edu
eghough@iu.edu, tpandey@iu.edu
gcopus@iu.edu, dmmullin@iu.edu
mbekkem@iu.edu, pateishi@iu.edu
cfampo@iu.edu, tarturnm@iu.edu
jdc6@iu.edu, dyashwar@iu.edu
kjj6@iu.edu, orrostew@iu.edu
tfreson@iu.edu, aveluru@iu.edu

elybrewe@iu.edu, fshamrin@iu.edu
jacobben@iu.edu, iperine@iu.edu
leolin@iu.edu, gepearcy@iu.edu
bencho@iu.edu, mzagotta@iu.edu
rcaswel@iu.edu, bmpool@iu.edu
adhuria@iu.edu, cialugo@iu.edu
zhatfie@iu.edu, gmpierce@iu.edu
caegrah@iu.edu, wilsonnf@iu.edu
nihanas@iu.edu, mnimmala@iu.edu
jhar@iu.edu, emmmccle@iu.edu
skunduru@iu.edu, jlopezmo@iu.edu
aaragga@iu.edu, gavilleg@iu.edu
austdeck@iu.edu, apavlako@iu.edu
jdemirci@iu.edu, epautsch@iu.edu
althart@iu.edu, rvinzant@iu.edu
apbabu@iu.edu, marystre@iu.edu
arnadutt@iu.edu, ijvelmur@iu.edu
spgerst@iu.edu, deturne@iu.edu
agawrys@iu.edu, sasayini@iu.edu
tcconnol@iu.edu, sahimann@iu.edu
sfuneno@iu.edu, tzuyyen@iu.edu
aakindel@iu.edu, justyou@iu.edu
gandhira@iu.edu, kamaharj@iu.edu
loggreen@iu.edu, reedkier@iu.edu
jwdrew@iu.edu, maglowe@iu.edu
ddrotts@iu.edu, madymcsh@iu.edu
celechav@iu.edu, ap52@iu.edu
anlego@iu.edu, patel89@iu.edu
jtbland@iu.edu, abiparri@iu.edu
laharden@iu.edu, rtrammel@iu.edu
lawmat@iu.edu, cltran@iu.edu
wgurley@iu.edu, cmvanhov@iu.edu
seangarc@iu.edu, ruska@iu.edu
hermbrar@iu.edu, wtrucker@iu.edu
johnguen@iu.edu, amanocha@iu.edu
dkkosim@iu.edu, savebhat@iu.edu
kaihara@iu.edu, lmeldgin@iu.edu
wilcusic@iu.edu, woodsky@iu.edu
criscruz@iu.edu, jwember@iu.edu
mwclawso@iu.edu, rorymurp@iu.edu
marcchao@iu.edu, awsaunde@iu.edu

browdeon@iu.edu, patedev@iu.edu
ek37@iu.edu, wtubbs@iu.edu
ethbrock@iu.edu, krbpatel@iu.edu
mdiazrey@iu.edu, ysanghi@iu.edu
alelefeb@iu.edu, smremmer@iu.edu
ovadeley@iu.edu, msorsor@iu.edu
mohiambu@iu.edu, jpochyly@iu.edu
rl29@iu.edu, myeralli@iu.edu
abellah@iu.edu, shrrao@iu.edu
mrfehr@iu.edu, gszopin@iu.edu
hh35@iu.edu, perezand@iu.edu
arklonow@iu.edu, rwan@iu.edu
vkommar@iu.edu, kvpriede@iu.edu
quecox@iu.edu, benprohm@iu.edu
eakanle@iu.edu, jdw14@iu.edu
aberkun@iu.edu, jtsuter@iu.edu
dja1@iu.edu, dukthang@iu.edu
dblackme@iu.edu, vpolu@iu.edu
josespos@iu.edu, bdyiga@iu.edu
braybrya@iu.edu, gmichna@iu.edu
tchapell@iu.edu, ntuhl@iu.edu
diebarro@iu.edu, rnschroe@iu.edu
adwadash@iu.edu, hasiddiq@iu.edu
apchavis@iu.edu, antando@iu.edu
garcied@iu.edu, evataylo@iu.edu
mbeigie@iu.edu, jnzheng@iu.edu
kdembla@iu.edu, rafir@iu.edu
gmhowell@iu.edu, liwitte@iu.edu
aroraarn@iu.edu, emgward@iu.edu
ridbhan@iu.edu, gsilingh@iu.edu
lflenoy@iu.edu, lukastef@iu.edu
liansia@iu.edu, giomayo@iu.edu
mdonato@iu.edu, asidda@iu.edu
ohostet@iu.edu, aamathew@iu.edu
tfleuran@iu.edu, aayushah@iu.edu
kaneai@iu.edu, btasa@iu.edu
ryanbren@iu.edu, cnyarko@iu.edu
ayoajayi@iu.edu, ansakrah@iu.edu
jkielcz@iu.edu, fmahamat@iu.edu
oeichenb@iu.edu, patekek@iu.edu
ageorgio@iu.edu, schwajaw@iu.edu

khanni@iu.edu, ksadiq@iu.edu
stkimani@iu.edu, pricemo@iu.edu
mdoxsee@iu.edu, mmunaf@iu.edu
jabbarke@iu.edu, asaokho@iu.edu
nharkins@iu.edu, rosenbbj@iu.edu
blacount@iu.edu, sahaan@iu.edu
cgkabedi@iu.edu, pp31@iu.edu
nbernot@iu.edu, jomeaghe@iu.edu
leegain@iu.edu, sahishah@iu.edu
micedunn@iu.edu, emluplet@iu.edu
saganna@iu.edu, surapapp@iu.edu
nolakim@iu.edu, wardjohn@iu.edu
bjdahl@iu.edu, vyeruba@iu.edu
egoldsto@iu.edu, aledminc@iu.edu
masharre@iu.edu, myswill@iu.edu
adiyer@iu.edu, erschaef@iu.edu
amkurz@iu.edu, nmr1@iu.edu
earuland@iu.edu, dernguye@iu.edu
simadams@iu.edu, asultano@iu.edu
marganey@iu.edu, aidnschi@iu.edu
howardbw@iu.edu, chrimanu@iu.edu
brhint@iu.edu, lqadan@iu.edu
alchatz@iu.edu, annaum@iu.edu
daxbills@iu.edu, ltmckinn@iu.edu
scbik@iu.edu, wattsra@iu.edu
evacoll@iu.edu, jwetherb@iu.edu
ajeeju@iu.edu, impofujr@iu.edu
spgreenf@iu.edu, audtravi@iu.edu
kapgupta@iu.edu, nichojop@iu.edu
coopjose@iu.edu, anassour@iu.edu
maxklei@iu.edu, awolkind@iu.edu
greenpat@iu.edu, dsummit@iu.edu
howamatt@iu.edu, ragmahaj@iu.edu
spdamani@iu.edu, gavsteve@iu.edu
sgaladim@iu.edu, nsatti@iu.edu
delkumar@iu.edu, cstancom@iu.edu
bellcol@iu.edu, reddyrr@iu.edu
hk120@iu.edu, adsize@iu.edu
lcoveney@iu.edu, coenthom@iu.edu
anrkram@iu.edu, tt13@iu.edu
saecohen@iu.edu, clmcevil@iu.edu

ceub@iu.edu, jneblett@iu.edu
ahavlin@iu.edu, jarlmint@iu.edu
davgourl@iu.edu, pravulap@iu.edu
migriswo@iu.edu, ism1@iu.edu
achordi@iu.edu, mjroelle@iu.edu, sezinnkr@iu.edu
clearle@iu.edu, nrizvi@iu.edu
sakalwa@iu.edu, amyawash@iu.edu
fdonfrio@iu.edu, jwu6@iu.edu
nokebark@iu.edu, aditpate@iu.edu
swconley@iu.edu, gs29@iu.edu
sydecook@iu.edu, megapaul@iu.edu
ejhaas@iu.edu, snyderjk@iu.edu
linjaso@iu.edu, rpoludas@iu.edu
oakinsey@iu.edu, utwade@iu.edu
brownset@iu.edu, ntorpoco@iu.edu
fkeele@iu.edu, jacmanto@iu.edu
lpfritsc@iu.edu, arirowe@iu.edu
ajgrego@iu.edu, avraya@iu.edu
skatiyar@iu.edu, majtorm@iu.edu
schinitz@iu.edu, clscheum@iu.edu
ethickma@iu.edu, aselki@iu.edu
dce@iu.edu, lvansyck@iu.edu
twfine@iu.edu, emisimps@iu.edu
nmcastan@iu.edu, qshamsid@iu.edu
daminteh@iu.edu, maklsmit@iu.edu
hamac@iu.edu, aptheria@iu.edu
ckdiallo@iu.edu, anajmal@iu.edu
aaamoako@iu.edu, jwmullis@iu.edu
ruqchen@iu.edu, brayrump@iu.edu
mkleinke@iu.edu, nniranj@iu.edu
mrcoons@iu.edu, keasandl@iu.edu
micahand@iu.edu, wlyzun@iu.edu
jdgonzal@iu.edu, mz24@iu.edu
alscarr@iu.edu, thnewm@iu.edu
ag69@iu.edu, lpelaez@iu.edu
joehawl@iu.edu, apathma@iu.edu
coopelki@iu.edu, drsnid@iu.edu
hawkjod@iu.edu, nlippman@iu.edu
kyhigg@iu.edu, bcmarret@iu.edu
krisgupt@iu.edu, jlzhao@iu.edu
grafe@iu.edu, etprince@iu.edu

aketcha@iu.edu, ammulc@iu.edu
flynncj@iu.edu, jcn1@iu.edu
avulas@iu.edu, skp2@iu.edu
wanjiang@iu.edu, violuway@iu.edu
ameydesh@iu.edu, ao9@iu.edu
cuizek@iu.edu, wilsori@iu.edu
tychid@iu.edu, mveltri@iu.edu
milhavil@iu.edu, scotbray@iu.edu
mbrockey@iu.edu, mszczas@iu.edu
cannan@iu.edu, mmarotti@iu.edu
edfran@iu.edu, jaslnu@iu.edu
efritch@iu.edu, tangtom@iu.edu
fu7@iu.edu, owysmit@iu.edu
bcdutka@iu.edu, ir1@iu.edu
escolber@iu.edu, asteini@iu.edu
keswar@iu.edu, cmarcuka@iu.edu
jc168@iu.edu, ryarram@iu.edu
jwcase@iu.edu, dwo@iu.edu
nfelici@iu.edu, mehtriya@iu.edu
zacbutle@iu.edu, wtatoole@iu.edu
jacklapp@iu.edu, trenstev@iu.edu
maudomin@iu.edu, muyusuf@iu.edu
nfarhat@iu.edu, rt11@iu.edu
pyphealy@iu.edu, vrradia@iu.edu
kekchoe@iu.edu, patelsak@iu.edu
matgarey@iu.edu, aranjit@iu.edu
kraus@iu.edu, anemlunc@iu.edu
colrkram@iu.edu, jactrayl@iu.edu
cpkerns@iu.edu, voram@iu.edu
jakchap@iu.edu, lmadiraj@iu.edu
bencalex@iu.edu, aptarin@iu.edu
maladwa@iu.edu, tolatinw@iu.edu
harmonnc@iu.edu, cjwaller@iu.edu
sg40@iu.edu, joshroc@iu.edu
agrevel@iu.edu, samyuan@iu.edu
bkante@iu.edu, thomps16@iu.edu
makinap@iu.edu, wwtang@iu.edu
mkames@iu.edu, isaramir@iu.edu
fkanmogn@iu.edu, masmatth@iu.edu
jhhudgin@iu.edu, pw18@iu.edu
allencla@iu.edu, blswing@iu.edu

willhowa@iu.edu, ap79@iu.edu
laburkle@iu.edu, jsadiq@iu.edu