Katrina Mae Reyes

Master of Molecular Science and Software Engineering
Problem Set #1
CHEM 247B – Software Engineering Fundamentals for Molecular Science DUE Date: Friday, September 19th, 2025 by 11:59 PM Pacific
Individual Assignment

The main goal of this assignment is to provide you with an opportunity to practice some of the concepts covered in weeks 1 through 3 of this course. Each problem has a set of questions and programming tasks. You will submit a PDF file with your answers. Make sure to identify the questions that you are answering (e.g., "Problem 1.2.1"). Written answers do not need to be full paragraph form, i.e., bullet-point format is permissible. You can use your favorite text editor (e.g., Goodle Doc, MS word, plain text editor, Latex, etc) but submit only the corresponding PDF file by uploading it to BCourses. Name your PDF file with the following format: <Berkeley ID>-Answers Assignment1.pdf

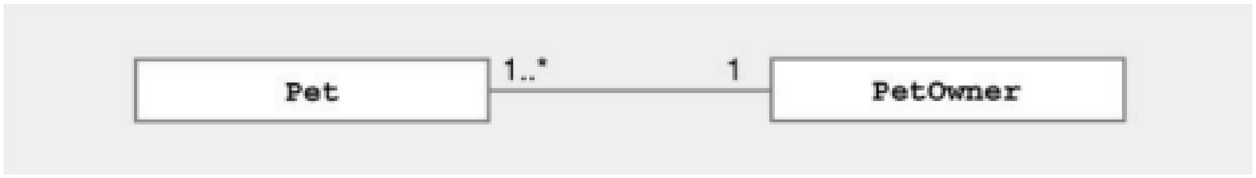
Document your programs well (e.g., using meaningful names for variables and functions, file names, relevant documentation). Make sure your computer programs are readable (i.e., exercise writing computer programs with code readability in mind). If a problem requests that you provide any programming files, place them all within a zipped folder containing all requested scripts for this problem assignment. Title the folder with the following format: <Berkeley ID>-Answers Assignment1-code and upload it to BCourses. If you use external code, e.g., StackOverflow (LLM outputs not allowed), please cite the source and provide rationale to prove you understand the code and that it is correct.

Assignment Points: 37

Unified Modeling Language

1. UML Comprehension (Part 1): Read each UML class diagram, try to understand the relationships being presented (pay attention to multiplicity), and describe it in natural language.

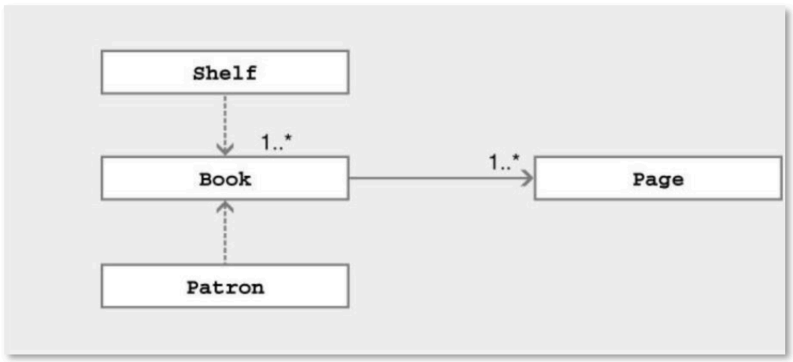
1.1 [1 point] Veterinary System



A pet owner can have multiple pets but a pet can only have one pet owner.

Katrina Mae Reyes

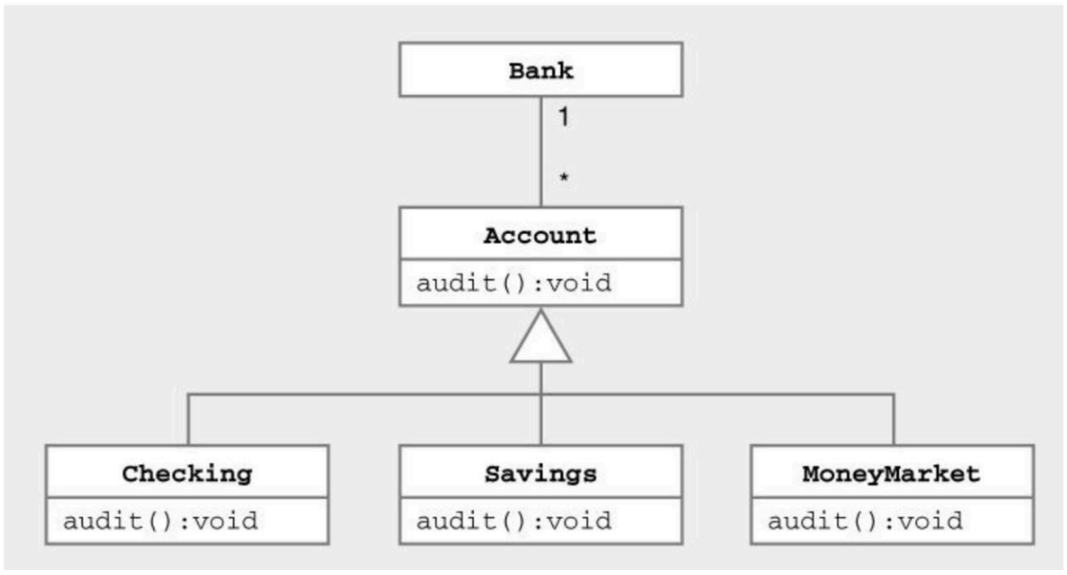
1.2 [1 point] Library System



There can be multiple pages in a book and multiple books in a shelf. A book is stored in a shelf and can be accessed by a patron.

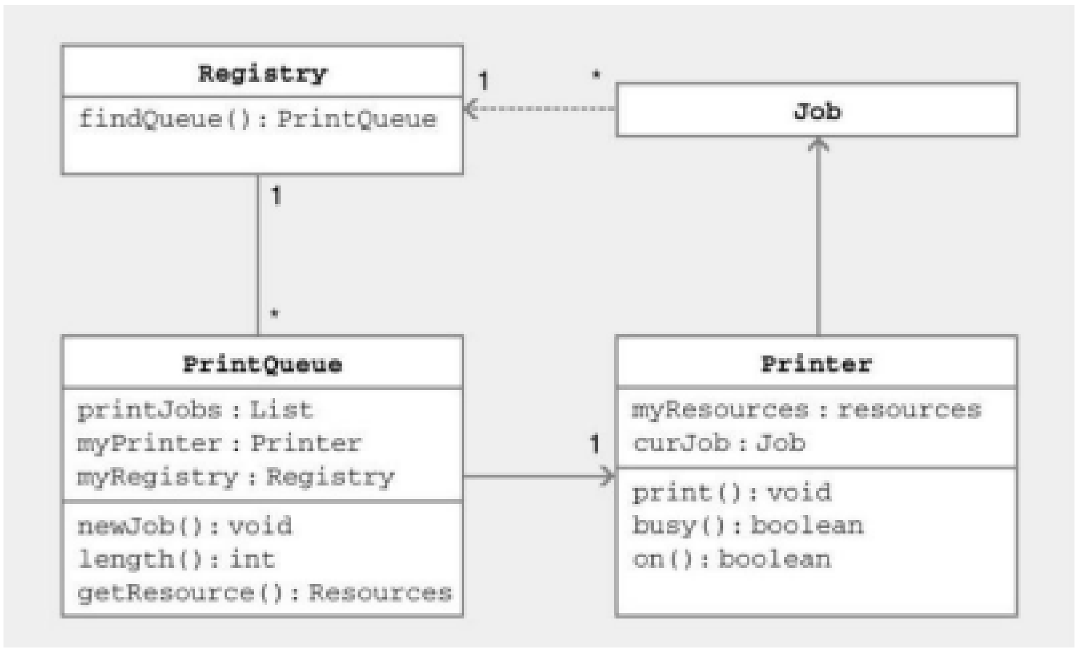
1.3 [1 point] Banking System

Checking, savings and MoneyMarket have a dependent relationship on the account. Many accounts can be found in this one bank. Checking, Savings, MoneyMarket, and account all perform an audit function denoted by audit(): void. This function does not return anything and only performs a task.



Katrina Mae Reyes

1.4 [1 point] Printing System



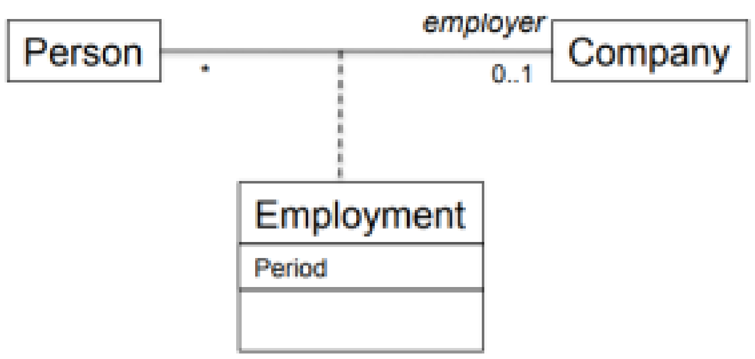
The job or task that needs to be performed by the printer is sent to exactly one registry. The registry performs a function findQueue() (which finds and stores the task it received), and returns this as PrintQueue. The registry can return many PrintQueues. Each PrintQueues stores the attributes printJobs, myPrinter, and my Registry and performs the functions newJob(), length() and getResources(). The PrintQueue instructs the printer what to print. The printer can be printing the tasks (print() function), be busy (busy() function), or is just on standby (on() function). Printer also stores the attributes myResources and curJob. There is exactly one printer in this case.

2. UML Comprehension (Part 2): For each question, designate the best answer from the choices given.

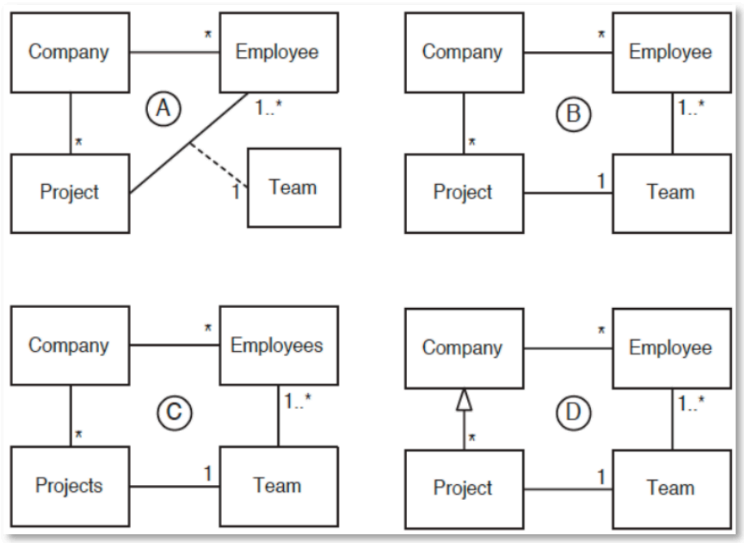
2.1 [1 point] Which sentences are coherent with the below class diagram?

A. A company may employ 0 or 1 person
B. A person may work for a single company
C. A person has one employment
D. A company has one employer that is a person
E. A company may have zero employers

Katrina Mae Reyes



2.2 [1 point] Consider a company that is involved in several projects. Each project is executed by a team of employees. Which of the following is the most suitable UML diagram? Why?≥



B - The project is not a subclass of the company, rendering D a poor choice. C has plural titles, which is redundant due to the scales already present next to the appropriate objects. For instance, the connection between company and employees has an asterisk next to employees, indicating that in this company, there are multiple employees. This also applies to “Projects” on part C where the plurality of the object is repeated by the asterisk. ‘A’ misleadingly suggests that there is an associated team to the employees. There should be a direct link between the employee and the team given that the employees are the members of the team. For these reasons, ‘B’ is the best choice.

Katrina Mae Reyes

2.3 [1 point] Which of the following are illegal combinations of subtypes in this class diagram?

A. Female, Patient, Nurse

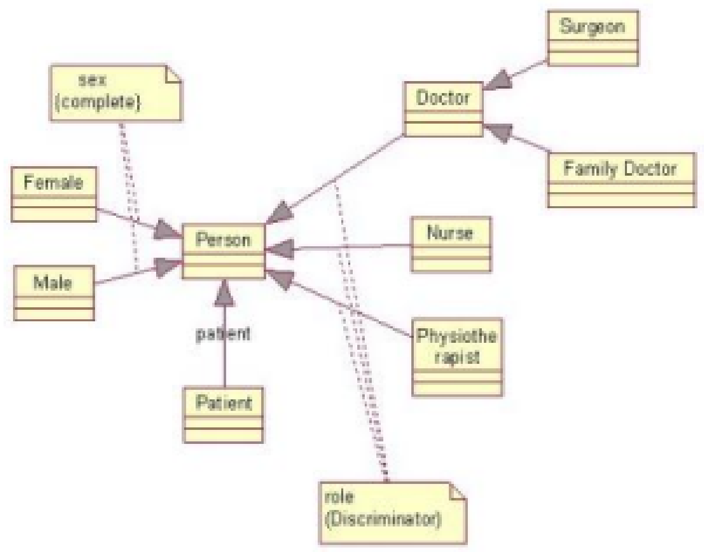
B. Male, Physiotherapist

C. Female, Patient

D. Female, Doctor, Surgeon

E. Patient, Doctor

F. Male, Doctor, Nurse



2.4 [1 point] Below is a class diagram for an order system. Give a one-sentence description of what each term is referring to in the context of the class diagram.

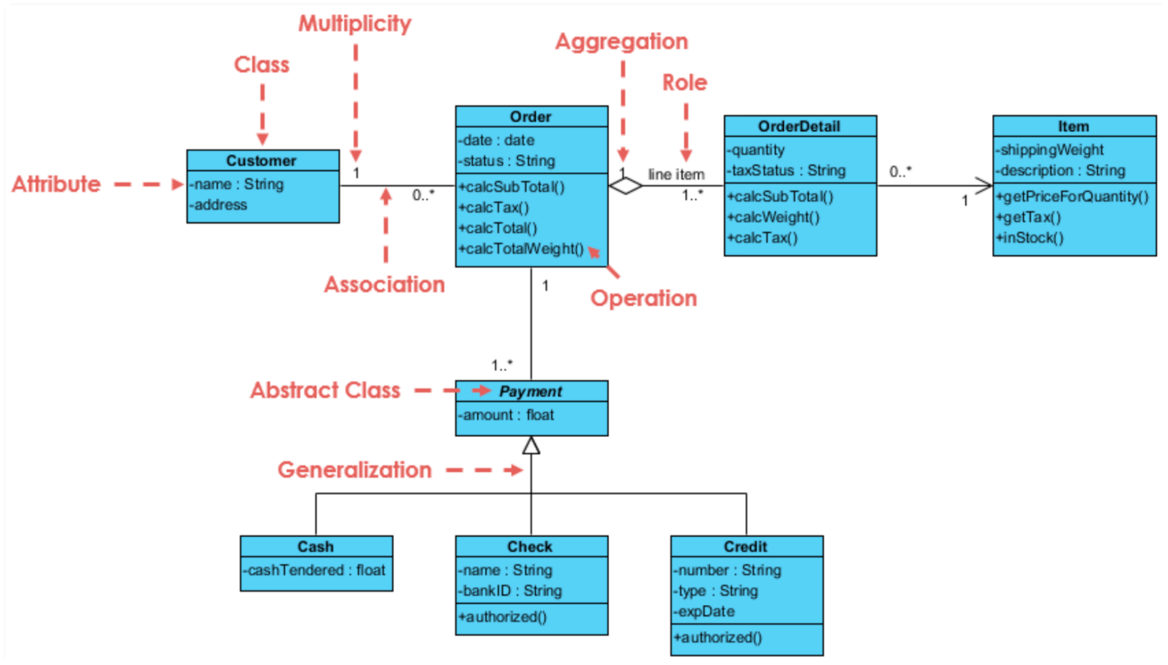
- Attribute - This is a variable of the class Customer.
- Class - This is a class that holds a variable name of type string and an address variable.
- Multiplicity - The amount of class associated with an order (which is one).
- Association - Customer and order are related but no direction of inheritance or other relationship.
- Aggregation - This is a one sided relationship in which the order contains an order detail but the reciprocal relationship isn't true.
- Role - This describes the role of the order detail in the context of the class Order.

Katrina Mae Reyes

- Operation - This is a method inside a class, describing functions that will be carried out.

- Abstract class - General class encompassing all forms of payment (cash, check, credit)

- Generalization - Relationship where subclasses inherit their superclass payment.

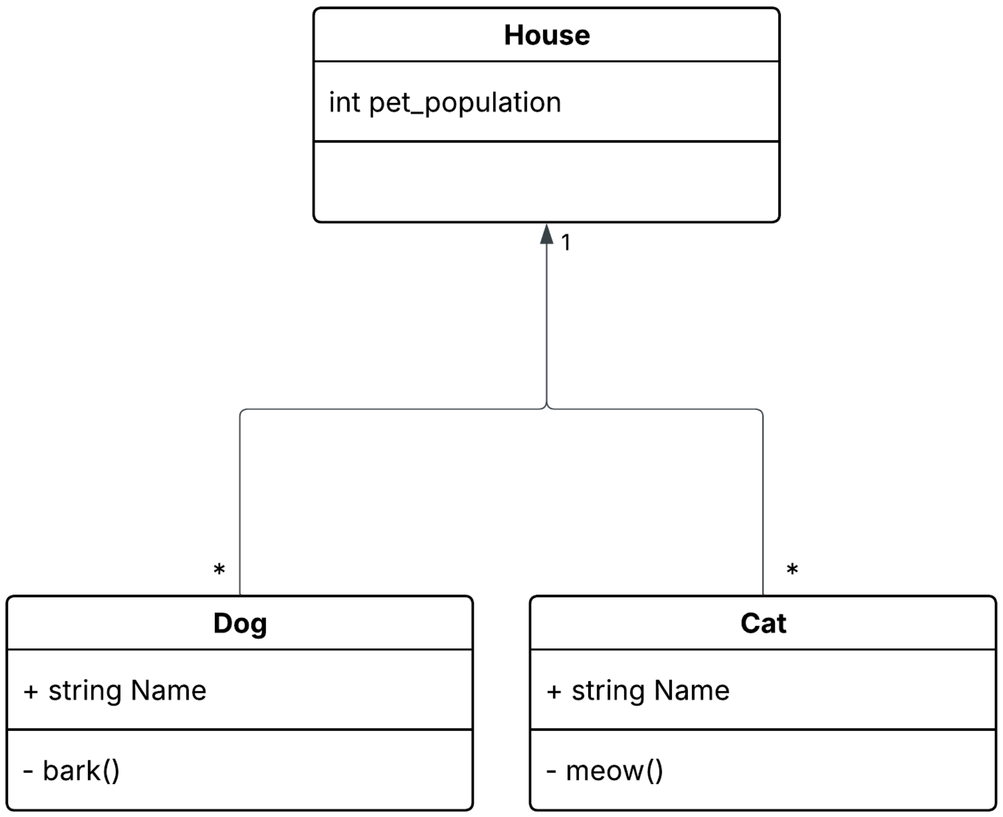


3. UML DIY: The following questions present an open-ended scenario and require you to design your own diagram.

3.1 [2 points] Design a class diagram for a system that tracks the animals within a house, fulfilling the following criteria:

- A house may have any number of pets living in it
- The two possible types of pets that can live in a house are dogs and cats
- Each dog or cat has a name
- An animal's house is its one and only home
- You can tell an animal to make noise and it will do its thing

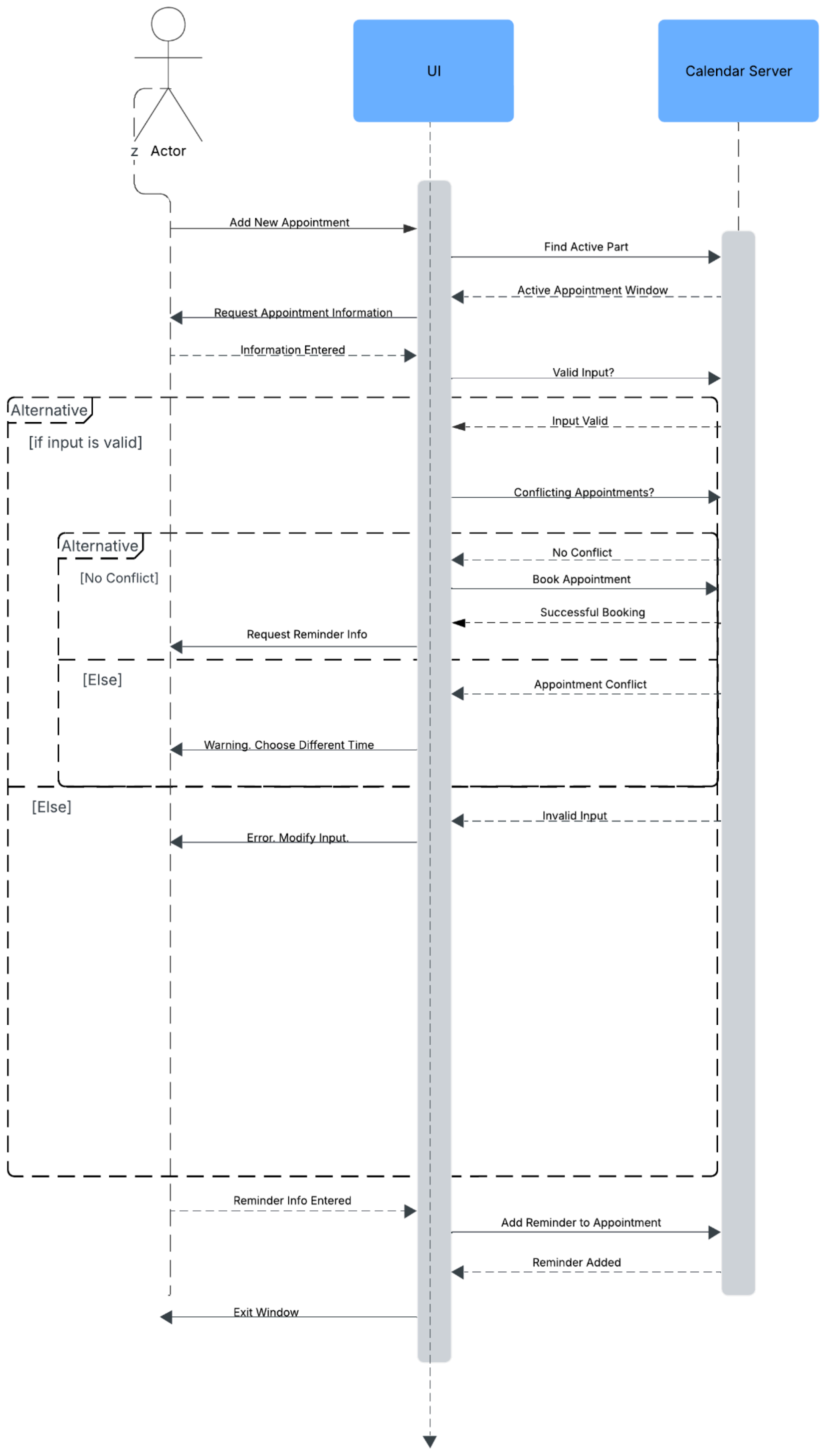
Katrina Mae Reyes



3.2 [4 points] Design a sequence diagram for a piece of software that adds an appointment to a calendar:

- The scenario begins when the user chooses to add a new appointment in the UI
- The UI notices which part of the calendar is active and pops up an Add Appointment window for that date and time
- The user enters the necessary information about the appointment's name, location, start and end times
- The UI will prevent the user from entering an appointment that has invalid information such as an entering an appointment that has invalid information, such as an empty name or negative duration
- The calendar records the new appointment in the user's list of appointments
- Any reminder selected by the user is added to the list of reminders
- If the user already has an appointment at that time, the user is shown a warning message and asked to choose an available time or replace the previous appointment
- If the user enters an appointment with the same name and duration as an existing group meeting, the calendar asks the user whether he/she intended to join that group meeting instead
- If so, the user is added to that group meeting's list of participants.

Katrina Mae Reyes



4. UML Conversions

4.1 [1 point] For the below UML diagram, outline its class structure in Python. You can write it or you can provide the programming file in your code submission folder.

Katrina Mae Reyes

Please check file 4.1_UML.py on the submission folder for the code.

Analysis of Algorithms

5. Analysis Warm-ups

5.1 [1 point] Suppose the running time of an algorithm was the following for the given inputs:

Input Size	Running Time
1,000	5 seconds
2,000	20 seconds
3,000	45 seconds
4,000	80 seconds

5,000	???

Estimate how long it will take to solve a problem of size 5,000. Is the order of growth of the running time linear, linearithmic, quadratic, cubic, or exponential? Explain your rationale.
It will take 125 seconds for an input size of 5000. This is a quadratic growth. When the input size is doubled, the runtime increases by a factor of 4. If the input is tripled, the runtime increases by a factor of 9. This shows a quadratic relationship between input size and runtime. Thus, if the input size is increased by a factor of 5 (from 1000 to 5000), quadratically, the runtime will increase by a factor of 25.

5 seconds x 25 = 125 seconds

5.2 [1 point] The following function is intended to return a random string of length *n*.

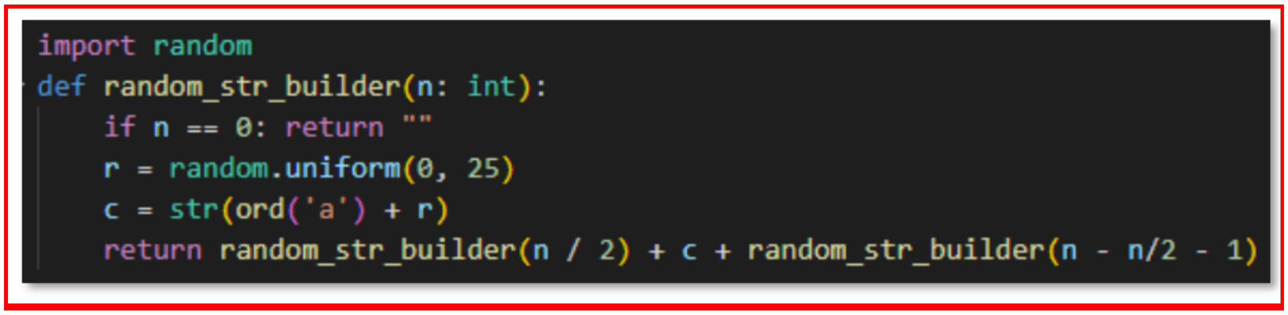
5.2.1 What is problematic about the function's current implementation? What lines need changed and how would you correct the implementation?

This will never hit the base case of n==0. You can change the if statement to n<=0. Random.uniform should be random.randint since this intends to index into the alphabet, so there should be no floating points. Instead of 'n/2', there it should be n//2 so the input into the recursive case will be an int, as indicated in the argument for n.

Katrina Mae Reyes

5.2.2 For the correct implementation, what is the order of growth of its running time as a function of *n*? Explain your rationale.

It is O(n) since each recursion call calls n once.



```python
import random
def random_str_builder(n: int):
    if n == 0: return ""
    r = random.uniform(0, 25)
    c = str(ord('a') + r)
    return random_str_builder(n / 2) + c + random_str_builder(n - n/2 - 1)
```

5.3 [1 point] Suppose you were interested in understanding the order of growth of a method whose implementation you do not have access to (e.g., a private method). Devise an experiment to verify what the order of growth is for both run time and memory.
You can implement a time function before and after a function to see the elapsed time while the method is running. You can then measure the elapsed time with different sizes of inputs. You can then graph this as a function (elapsed time vs input size) and see the order of growth. A similar function that graphs memory usage over time can also be implemented.

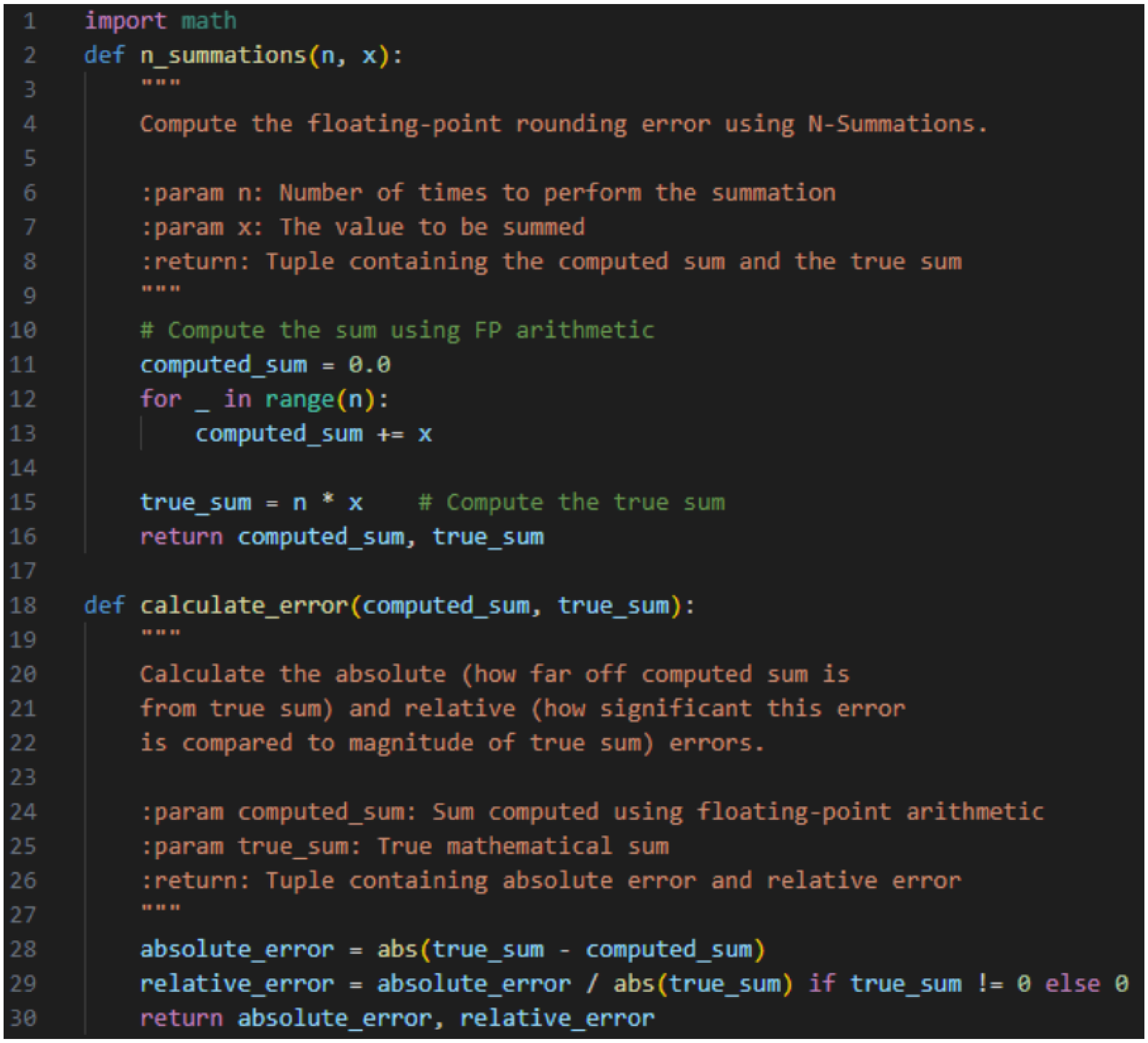
5.4 [1 point] Knowing nothing else, which order of growth would you prefer: quadratic, logarithmic, or linear? How does your answer change if you know that the value of the leading coefficients of the running time, such that the running times are ��^{2}seconds, 100��������$_2$�� seconds, and 10000�� seconds? Think about which algorithm would be fastest for a given range of problem sizes, ��.

Preferably logarithmic given that the runtime grows the slowest out of the 3 functions. The coefficients of the run time will not change given that this does not affect the general behavior of the runtime functions. For instance, a quadratic runtime will still grow faster with increasing input sizes than a linear runtime even with coefficients.

6. Floating Point Rounding Errors and Data Representation: Consider the code block following this problem statement, which shows how floating-point arithmetic can accumulate to rounding errors, especially when dealing with many operations ($n \rightarrow \infty$) or with numbers that can't be represented precisely in binary floating-point (e.g., 0.1). Consider the example output for this code for 10^6summations of 0.1:

Number of summations: 1000000
Value being summed: 0.1
Computed sum: 100000.00000133288
True sum: 100000.0
Absolute error: 1.3328826753422618e-06
Relative error: 1.3328826753422619e-11

Katrina Mae Reyes



```
1   import math
2   def n_summations(n, x):
3       """
4       Compute the floating-point rounding error using N-Summations.
5
6       :param n: Number of times to perform the summation
7       :param x: The value to be summed
8       :return: Tuple containing the computed sum and the true sum
9       """
10      # Compute the sum using FP arithmetic
11      computed_sum = 0.0
12      for _ in range(n):
13          computed_sum += x
14
15      true_sum = n * x    # Compute the true sum
16      return computed_sum, true_sum
17
18  def calculate_error(computed_sum, true_sum):
19      """
20      Calculate the absolute (how far off computed sum is
21      from true sum) and relative (how significant this error
22      is compared to magnitude of true sum) errors.
23
24      :param computed_sum: Sum computed using floating-point arithmetic
25      :param true_sum: True mathematical sum
26      :return: Tuple containing absolute error and relative error
27      """
28      absolute_error = abs(true_sum - computed_sum)
29      relative_error = absolute_error / abs(true_sum) if true_sum != 0 else 0
30      return absolute_error, relative_error
```

6.1 [2 points] Algorithmic analysis: For the `n_summations(n, x)` function, (1) complete the "Times" column of the following table, which reflects the number of times that line is executed. We assign a constant cost to every step. Now that you have the cost and time for each line, (2) write the corresponding numerical formulation, �����������������(��), for the number of times every step gets executed.

Line	Cost	Time
11	c_1	1
12	c_2	n-1
13	c_3	n-1

Katrina Mae Reyes

15	c_4	1

TotalCost(n)) = c1 + c2(n-1) + c3(n-1) + c4

6.2 [2 points] Write a test script to call the n_summations(n, x) function for different values of n and x. Fill in the table below for the relative error for different values of n and x. What happens to the relative error as n increases? What happens to the relative error as n decreases? Even though the numbers are similar in magnitude, why is the relative error so different for those values of n with base 16 compared to the other values of n (e.g., 9, 99, 999, etc.)? You do not need to submit your test program code. If the error is large, just report the leading digit and magnitude, e.g., report 1.6734946621243942e-11 as 1e-11.

		n							
		9	99	999	9999	99999	16	16^2	16^3
x	0.4	1e-16	1e-15	1e-14	1e-13	1e-12	1e-16	3e-15	6e-14
	0.1	1e-16	1e-15	1e-14	1e-13	1e-12	1e-16	3e-15	6e-14
	0.01	0	6e-16	1e-14	1e-13	7e-13	0	4e-15	1e-14

As n increases, the relative error increases. Base 16 numbers tends to have higher error than other numbers due to the floating point calculation.

Linear Data Structures

7. Warm Ups:

7.1 [1 point] What does the following code fragment print when n is 50? Give a high-level description of what the code fragment does when presented with a positive integer *n*.

```
#include <iostream>
#include <stack>

int main() {
    int n = 50;
    std::stack<int> stack;

    while (n > 0) {
        stack.push(n % 2);
        n /= 2;
    }

    while (!stack.empty()) {
        std::cout << stack.top();
        stack.pop();
    }

    std::cout << std::endl;
    return 0;
}
```

	n%2	n
Last in	1	1
	1	3
	0	6
	0	12
	1	25
	0	50

Stack goes by a last-in-first-out order of operation. The n%2 column was created from stack.push(n%2). 'Stack.top' reads the element at the very top or the element that that was last in and returns in. 'stack.pop' then removes the top element, and then stack.top reads the newest top element and returns it again. This continues until the stack is empty.

We get the binary code for 50.

7.2 [1 point] What does the following code fragment do to the queue "queue"?

```
#include <stack>
#include <queue>

std::stack<int> stack;
std::queue<int> queue;

while (!queue.empty()) {
    stack.push(queue.front());
    queue.pop();
}
```

Katrina Mae Reyes

```
while (!stack.empty()) {
    queue.push(stack.top());
    stack.pop();
}
```

stack.push(queue.front()) - reads the front of the queue and pushes that to the top of the stack.
queue.pop() removes the front-most element and the next element is then pushed to the top of the stack. This cycle repeats until the queue is empty.

When the queue is empty:
queue.push(stack.top()) - reads the front of the stack and pushes this to the back of the queue.

stack.pop() removes the top most element and then the cycle repeats with queue.push(stack.pop())

The output is a sequence that is the reverse of the original order.

7.3 [1 point] Which code block correctly inserts node *t* immediately after node *x* in a linked list? Justify your answer.

Option A:

```
t.next = x.next;
x.next = t;
```

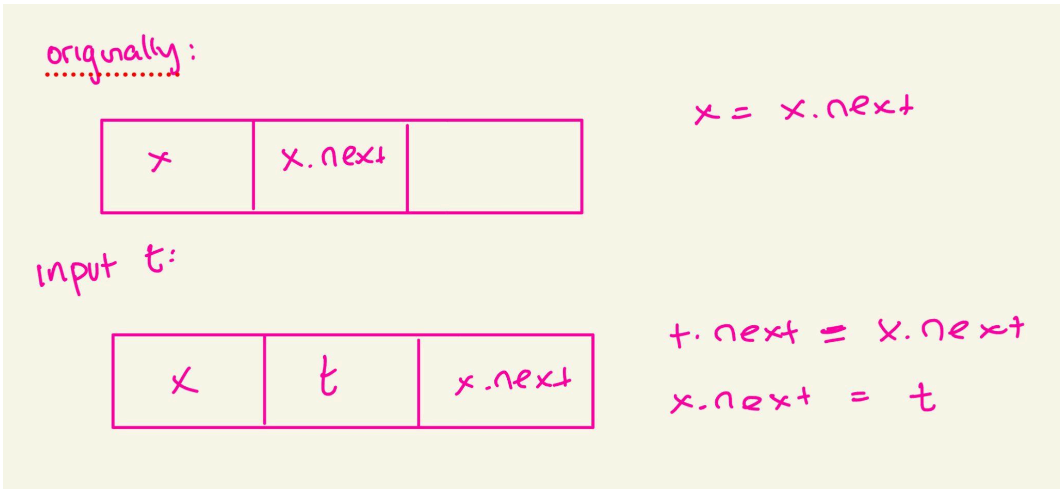
Option B:

```
x.next = t;
t.next = x.next;
```

Option C:

```
t = x.next.next;
x.next = t;
```

Katrina Mae Reyes

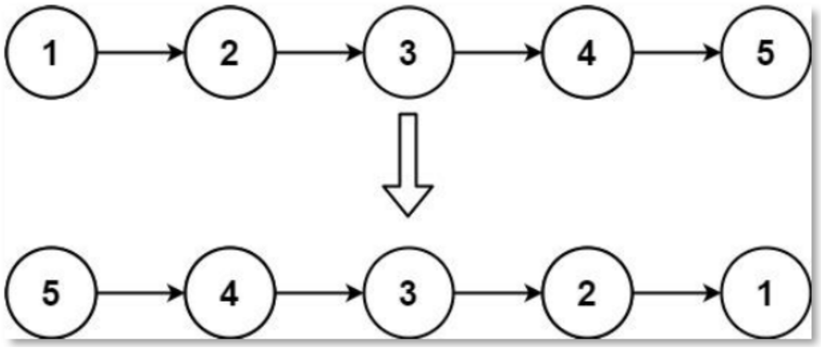


After inserting t, the link to x.next has to be preserved, so t.next=x.next has to be defined first. Then, since x is now pointing to t, x.next = t. The right answer is A.

8. GradeScope Problems: You will implement code to complete several programming problems. The starter code files are available in the starter_code directory that is in the same location as this problem set 1 file. Follow the instructions within each question to complete the implementation.

Once you have completed **all** of the problems, upload a submission containing all files to the corresponding GradeScope assignment: "Problem Set 1: Linear Data Structures" Note that the autograder score is to facilitate grading and does not necessarily reflect your final score.

8.1 [2 points] Reverse a linked-list -- Given the head of a singly linked list, reverse the list, and return *the reversed list*. You may provide either the iterative or recursive solution. Implement your solution where specified in the file reversed_linked_list.py that was given as starter code. The image below visualizes an input and expected output:



Katrina Mae Reyes

8.2 [3 points] Valid symbols -- Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid. An input string is valid if:

1. Open brackets must be closed by the same type of brackets.
2. Open brackets must be closed in the correct order.
3. Every close bracket has a corresponding open bracket of the same type. Implement your

solution where specified in the file valid_symbols.py that was given as starter code.

8.3 Queuing -- Consider the starter Python implementation of a queue using a linked list structure, provided in queue_linked_list.py.

8.3.1 [3 points] Implement the is_empty, enqueue, and dequeue functions so that they run in constant time. Make sure to consider edge cases and do not modify the Node class. Implement your solution where specified in the file queue_linked_list.py that was given as starter code.

8.3.2 [3 points] Now implement your own queue class using two stacks so that each queue operations takes a constant amortized number of stack operations. Implement your solution where specified in the file queue_two_stacks.py that was given as starter code.