CSCI 121: Computer Science Fundamentals I
Pratice Second Midterm Exam

The next pages give practice problems for the second midterm exam being held in lecture on Monday April 27th, 2026. The exam covers these topics:

- recursive functions
- higher-order functions and `lambda`
- object orientation
- class inheritance
- linked lists

You can use these to test your knowledge in preparation for taking the exam. I will post my solutions to these problems on Friday, April 24th.

---

1. Write a *recursive* function `ones(n)` that, when given a positive integer `n`, reports the number of times a digit of 1 appears in its decimal representation.

   ```
   >>> ones(123417)
   2
   >>> ones(375)
   0
   >>> ones(11011)
   4
   ```

   You cannot use any string operations to write this code and your function must be recursive.

2. Write the code for `make_reporter` that takes a single value as a parameter. It should return a function that, when given a parameter, outputs (using `print`) whether that parameter is larger or smaller than the value originally given to `make_reporter`. If it is equal, it should not do anything.

```
>>> r = make_reporter(8)
>>> r(6)
smaller
>>> r(8)
>>> r(10)
larger
```

3. Write, from scratch, a `Bus` class that represents the activity of a bus along a bus line. The bus line is a series of n bus stops numbered from 0 to $n - 1$. When the `Bus` object is first created with `__init__`, **it is given a list of the number of passengers at each stop**. The bus is placed at stop 0 and has no passengers yet. Other than `__init__`, it has three methods: `pick_up`, `drop_off`, and `next_stop`. It might help you to **check the next page of the exam** to see a `Bus` object in action.

   When `pick_up` is called at a stop, the number of passengers at that stop is decremented, and the number of passengers in the bus is incremented. When `drop_off` is called at a stop, the number of passengers at that stop is incremented, and the number of passengers in the bus is decremented. When `next_stop` is called and it is at stop s, it moves to stop $s + 1$. With each call of `next_stop`, then, it moves from stop 0, to stop 1, to stop 2, etc. The exception is when it is at stop $n - 1$. In that case another call to `next_stop` makes it drop off all its passengers at stop $n - 1$ and then return to stop 0.

   Note that `drop_off` does nothing if the bus has no passengers and `pick_up` does nothing if there are no passengers at the bus's current stop.

   Write the code below. Again, the **next page of this exam shows the transcript** of an example use of a `Bus` object.

An example use of the `Bus` object class:

```
>>> stops = [10,0,4,0,0]
>>> b = Bus(stops)
>>> b.pick_up()
>>> stops
[9, 0, 4, 0, 0]
>>> b.pick_up()
>>> b.pick_up()
>>> stops
[7, 0, 4, 0, 0]
>>> b.next_stop() # The bus moves to stop 1 with 3 passengers.
>>> b.drop_off()
>>> stops
[7, 1, 4, 0, 0]
>>> b.drop_off()
>>> b.drop_off()
>>> b.drop_off()  # None dropped off since there are no passengers.
>>> stops
[7, 3, 4, 0, 0]
>>> b.next_stop() # The bus moves to stop 2.
>>> b.pick_up()
>>> stops
[7, 3, 3, 0, 0]
>>> b.next_stop()
>>> b.next_stop()
>>> b.next_stop() # The bus loses its passenger and returns to stop 0.
>>> stops
[7, 3, 3, 0, 1]
>>> b.pick_up()
>>> stops
[6, 3, 3, 0, 1]
```

4. Below is some code for the `Car` class, similar to the one from the homework. Write the code for a new class, `Delivery`, used to represent delivery trucks for a particular company. Delivery trucks are like cars, but rather than allowing for customization, these trucks should always have a gas tank that holds 50 gallons and get 15 miles per gallon. Furthermore, there should be an additional method, `idle`, which takes as input a time measured in minutes and represents the truck idling at a stop for that many minutes. The truck, when idling, uses gas at a rate of two gallons per hour. An idling truck that runs out of gas should just sit with no gas. The `idle` method should return `None`.

```
from math import sqrt

class Car:

     def __init__(self, mpg, tank):
          self.x = 0.0
          self.y = 0.0
          self.mpg = mpg
          self.tank = tank
          self.gas = tank

      def moveTo(self, newX, newY):
          dx = self.x - newX
          dy = self.y - newY
          distance = sqrt(dx**2 + dy**2)
          if self.gas < distance / self.mpg:
              return False
          else:
              self.gas -= distance / self.mpg
              self.x = newX
              self.y = newY
              return True
```

5. Write a method for `LinkedList` called `apply`. It should take a function as a parameter and apply that function to the contents of each of the list nodes, changing them to that function's value.

```
>>> xs = LinkedList()
>>> xs.append(20); xs.append(100); xs.append(15); xs.append(87)
>>> print(xs.asString())
<20, 100, 15, 87>
>>> xs.apply(lambda x: x - 10)
>>> print(xs.asString())
<10, 90, 5, 77>
```

   No new nodes should be created as a result of this method being called.