Name:
Student Number:

Please read the following guidelines carefully!

• Please write your name on the front and back of the exam.
• This examination has 4 questions. There are a total of 9 pages, DOUBLE-SIDED.
• You may always write helper functions/methods unless explicitly asked not to.
• Any question you leave blank or clearly cross out your work and write “I don’t know” is worth 10% of the marks.

Take a deep breath.

This is your chance to show us
How much you’ve learned.

We WANT to give you the credit
That you’ve earned.

A number does not define you.

Good luck!
1. The following questions test your understanding of the terminology and concepts from the course. You may answer in either point form or full sentences; you do not need to write much to get full marks!

(a) [2] What is a benefit of marking a class attribute as private?
   **Solution:** hide implementation details from client code; this allows the implementation to change without affecting external code. Also, this allows implementers to hide the complexity of implementation from the client code, so that the client only needs to know what the class does and how to use it, but not how it works.

(b) [2] Explain the difference between a **queue** and a **priority queue**.
   **Solution:** in a queue, you remove items in the same order you insert them. In a priority queue, you remove items based on their priority or “size”, with the smallest/biggest being removed first.
   Note: many students discussed differences at the implementation rather than abstract data type level. Keep in mind that queues, stacks, and priority queues are ADTs, and so they can be implemented in very different ways (not even using lists).

(c) [3] What is the running time of adding an element to the front of a Python list? State both the Big-Oh expression of the running time, as well as why this is the case. Be sure to define any variables you use in your response.
   **Solution:** O(n), where n is the length of the list. This is because when you insert items into the front of a Python list, all other items in the list must be shifted in memory to make room for that item.

(d) [3] Consider the following function, which operates on a Python list:

   ```python
def remove_middle(lst):
    new_lst = [lst[0], lst[1]]
    lst = new_lst
    return new_lst
```

   Fill in the output of the print statement below. Then underneath, explain why this happens.

   ```
   >>> lst = [1, 2, 3]
   >>> new_lst = remove_middle(lst)
   >>> print(lst)  # FILL IN THE OUTPUT IN THE NEXT LINE
   # SOLUTION
   [1, 2, 3]
   >>> print(copied)
   # SOLUTION
   [1, 2]
   ```

   **Explanation:**
   Assigning `lst` to `new_lst` inside the body of the function doesn’t actually change the argument that was passed in, it only changes what data `lst` refers to.
   Note: this is a common mistake we discussed in lecture; the line `lst = new_lst` doesn’t mutate the argument to the function.
2. Consider a function `divide` which takes a stack of integers and repeats the following sequence of operations until the stack is empty:

(i) Remove an item from the stack.
(ii) Print the item.
(iii) If the item is even, add two copies of (item / 2) to the stack.

(a) [2] Write the output of the following function call (no explanation necessary):

```python
global s
s = Stack()
s.push(12)
print(divide(s))
```

Solution:
```
12
6
3
3
6
3
3
```

Many students just wrote 12, forgetting about the “repeat until the stack is empty” part (although a significant number did implement the function correctly in part b).

(b) [4] Implement the function.

```python
def divide(stack):
    # SOLUTION
    while not stack.is_empty():
        item = stack.pop()
        print(item)
        if item % 2 == 0:
            stack.push(item)
            stack.push(item)
```
3. [7] You want to design a system which keeps track of cars and their owners. Each car in the system has a name, which is set when the car is created, and a mileage (the number of kilometres the car has been driven), which always starts at 0. Each car owner has an amount of money and any number of cars. A car owner can purchase a new car for a given price.

Below and on the next page, we have a very incomplete class design for this scenario. You have three tasks:

- Fill in the attributes of each class. These can all be public. Be sure to specify a type and description.
- Design and implement the constructor of the Car class. A docstring is not necessary.
- Implement the method buy_car of CarOwner, according to its docstring.

class Car:
    ""
    A car.
    
    Attributes:
    @type name: str
        The name of the car.
    @type mileage: int
        The number of kilometres the car has been driven.
    ""
    def __init__(self, name):
        # Note: the mileage of the car should *not* be a parameter of the constructor
        self.name = name
        self.mileage = 0

class CarOwner:
    ""
    A person who owns some cars.
    
    Attributes:
    @type money: int
        The amount of money the person has.
    @type cars: list[Car]
        The cars this person owns.
    ""
    def buy_car(self, name, price):
        ""
        Buy a new car with the given name for the given price.
        
        If this person cannot afford to buy the car, then do nothing.
        
        @type self: CarOwner
        @type name: str
        @type price: int
        @rtype: None
        ""
        if price <= self.money:
            self.cars.append(Car(name))
            self.money = self.money - price
4. (a) [1] Suppose we have access to the LinkedList class found on the aid sheet, and we have created a linked list by doing the following:

```python
>>> linky = LinkedList([10, 3, 100, 0])
```

Write an expression which accesses the node at index 3 in linky. You may refer to private attributes of the LinkedList class.

```python
>>> linky._first.next.next.next
```

```python
>>> len(linky.items)
```

**Solution**: attribute error, because linky doesn’t have an attribute called items. (This was another common mistake we discussed in lecture. The parameters of the constructor are not the attributes of the class.)

(c) [3] Consider the following linked list method:

```python
def my_method(self, other):
    
    @type self: LinkedList
    @type other: LinkedList
    @rtype: int
    
    curr = self._first
    curr2 = other._first
    while curr is not None or curr2 is not None:
        print('hi')
        if curr is not None:
            curr = curr.next
        if curr2 is not None:
            curr2 = curr2.next
```

What is the asymptotic (Big-Oh) running time of this method? Justify your answer.

**Solution**: $O(\max(m, n))$ where $m$ is the length of self and $n$ is the length of other. The running time is proportional to the number of time the loop runs (each individual line doesn’t depend on the size of either input). The loop runs $\max(m, n)$ times: each time the loop runs, each “current node” variables both advance by 1 position, and the loop stops when both variables have reached the end of their respective list.
**Bonus Question [2]**

**Warning:** this is a difficult question, and will be marked harshly. Only attempt it if you have finished all of the other questions!

What is the asymptotic running time of the `divide` function from Question 2, in the worst possible case? You must fully justify your answer to receive credit. Think carefully about what you mean by “input size” here.
Use this page for rough work.
Use this page for rough work.
Name:

<table>
<thead>
<tr>
<th>Grade</th>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Q4</th>
<th>Total</th>
<th>Bonus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out Of</td>
<td>10</td>
<td>6</td>
<td>7</td>
<td>5</td>
<td>28</td>
<td>2</td>
</tr>
</tbody>
</table>