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 an abstract class, and what is the purpose of abstract classes in this course?

**Solution:** an abstract is a class with at least one unimplemented method. We use them to define a public interface which is then implemented by the subclasses of that class.

(b) [2] Explain the difference between a **stack** and a **priority queue**.

**Solution:** in a stack, you remove items in the reverse order you insert them (most recently added is removed first). 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 removing an element from 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 remove items from the front of a Python list, all other items in the list must be shifted in memory to fill in the gap made by removing that item.

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

```python
def copy_and_add(lst, item):
    copy = lst
    copy.append(item)
    return copy
```

Fill in the outputs of the two print statements below. Then underneath, explain why this happens

```python
>>> lst = [1, 2, 3]
>>> copied = copy_and_add(lst, 10)
>>> print(copied)  # FILL IN THE OUTPUT IN THE NEXT LINE

# SOLUTION
[1, 2, 3, 10]

>>> print(lst)  # FILL IN THE OUTPUT IN THE NEXT LINE

# SOLUTION
[1, 2, 3, 10]
```

**Explanation:**
When we set `copy = lst` inside `copy_and_add`, we make `copy` an alias of the argument data which was passed to the function ([1, 2, 3]).

Note that this is a common mistake we discussed in lecture: assignment statements change references, but don’t copy any data.
2. Consider a function `divide` which takes a queue of integers and repeats the following sequence of operations until the queue is empty:

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

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

```python
>>> q = Queue()
>>> q.enqueue(12)
>>> divide(q)
```

Solution:

```
12
6
6
3
3
3
```

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

(b) [4] Implement the function. You do not need to write a docstring.

```python
1   def divide(queue):
2       # SOLUTION
3
4       while not queue.is_empty():
5           item = queue.dequeue()
6           print(item)
7           if item % 2 == 0:
8               queue.enqueue(item / 2)
9               queue.enqueue(item / 2)
```
3. [7] You want to design a system which keeps track of cats and their owners. Each cat in the system has a name, which is set when the cat is created. Each person in the system can own either 0 or 1 cats. A cat owner can obtain a new cat, which replaces the previous cat the person owned, if any. Finally, the system should keep track of how long a cat has been owned by its owner.

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 Cat class. A docstring is not necessary.
- Implement the method get_cat of CatOwner, according to its docstring.

class Cat:
    """A cat.
    === Attributes ===
    @type name: str
    The name of the cat.
    
    NOTE: It was also acceptable to store the amount of time the cat was owned in this class.
    """
    # TODO: Design and implement a constructor here!
    def __init__(self, name):
        self.name = name

class CatOwner:
    """A person who can own one cat.
    === Attributes ===
    @type cat: Cat | None
    The cat this person owns, or None if this person doesn’t own a cat.
    Note: Many students forgot to express the "None" in the type. This is quite important - None behaves very differently than any other class, after all!

    @type time_owned: int
    The amount of time the person’s current cat has been owned, or -1 if the person has no cat.
    """
    def get_cat(self, name):
        """Get a new cat with the given name.
        Return True if the owner already had a cat, and False otherwise.
        
        @type self: CatOwner
        @type name: str
        @rtype: bool
        """
        had_cat = self.cat is not None
        self.cat = Cat(name)
        self.time_owned = 0
        return had_cat

Two common errors: not linking the two classes through composition (e.g., storing the name of the cat as an attribute rather than the Cat object itself), and not replacing the previous cat object with a new cat.
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 `item` at index 3 in `linky`. You may use all attributes of the `LinkedList` and `Node` classes.

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

(b) [1] State and explain the output of the following expression, based on the description of the `LinkedList` class found on the aid sheet.

```python
>>> linky.items[0]
```

**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: None
    """
    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>