1	2	3	4	5	6	Σ
30	10	20	10	15	15	100

Student number	2	0						
Name								

Midterm Exam — Data Structures (CS206A)

October 18, 2017, 9:00–12:00

Instructions:

- This booklet has **three pages** with **6 problems** in total. Check that you have all!
- Before you start: Write your name and student number (one digit per square!) on **all pages** of this exam booklet (−5 points for missing names or unreadable numbers).
- The space provided should be sufficient for your answer. If you need scratch space, use the back side. If you need more space for your answer, you can also use the back side (but *indicate clearly on the front side* that your answer continues on the back).
- This is a **closed book** exam. You are not allowed to consult any book or notes.
- The questions have to be answered in **English**. Write clearly!
- To ensure a quiet exam environment, we will **not answer questions** during the exam. If you think there is a mistake in the question, write an explanation, and use common sense to answer the question.
- Relax. Breathe. This is just an easy, silly, and stupid midterm exam.

Problem 1: (30 pts) For each of the following statements, say whether they are *true* or *false* (2 points for correct answer, −2 points for wrong answer, 0 points for no answer).

- All objects are stored on the heap. True — False
- If `a` is a `set`, then `a == set(list(a))` always holds. True — False
- Local variables of a method are stored in the stack frame of the method. True — False
- If an object is created as `a = (1, 2, 3)`, then the object is immutable. True — False
- Stack frames are stored on the heap. True — False
- The garbage collector destroys objects that cannot be reached anymore. True — False
- When an exception happens inside a method, the method returns `None`. True — False
- The abstract data type (ADT) *Stack* is a LIFO. True — False
- A *sentinel* is a list node that stores no list element but simplifies program code. True — False
- If `a` is a `list`, then `a == list(set(a)` always holds. True — False
- A circular buffer is often used to implement the ADT *Queue*. True — False
- `enqueue` and `dequeue` are methods of the ADT *Stack*. True — False
- The exception `ValueError` is an object. It is stored on the heap. True — False
- A data structure is an implementation of an abstract data type. True — False
- The state of immutable objects cannot be changed. True — False

Problem 2: (10 pts) We are implementing a class `DoublyLinkedList` to store a doubly-linked list. The nodes are objects of the following node class:

```
class Node:
  def __init__(self, el, next, prev):
    self.el = el; self.next = next; self.prev = prev
```

Write a method `insert_before` for the `DoublyLinkedList` class. It inserts a node storing element `el` *in front* of node `node`, using the given `Node` class and assuming that there are *sentinels* at the front and rear of the list.

```
def insert_before(self, node, el):
  # fill in
```

Student number	2	0						
Name								

Problem 3: (20 pts) We are executing the following Python script. Draw the contents of the *heap* and of the *runtime stack* when execution reaches the marked line.

```
def fruit(g, s):
  t = "apple"
  g.append(s)
  g.append(t)
  # draw situation here

def garden(g):
  s = "cherry"
  g.append(s)
  if len(g) % 3 == 0:
    fruit(g, g[-3])
  else:
    garden(g)

garden(["banana"])
```

Problem 4: (10 pts) A convention for printing large numbers is to insert commas every three digits. For example:

```
123
12,456
1,234,567
123,456,789,012,345,678,901
```

Write a *recursive* function to print a given positive integer number $n > 0$ in this format. Only fill in the one empty gap in the function:

```
def print_with_commas(n):
  if n < 1000:
    print(n, end="")
  else:
    # Fill in this part
```

Student number	2	0						
Name								

Problem 5: (15 pts) We are implementing a class `SinglyLinkedList` to store a singly-linked list. The nodes are objects of the following node class:

```
class Node:
  def __init__(self, el, next):
    self.el = el; self.next = next
```

Write a method `remove_all` for the `SinglyLinkedList` class. It removes all nodes from the list whose element `el` is equal to `x`. For instance, if the list contains the elements 1, 9, 4, 9, 9, 7, 3, 9, then after calling `remove_all(9)`, the list will contain the elements 1, 4, 7, 3. Handle all special cases correctly! Note that it is not an error if `x` does not appear in the list, or if the list is empty. The list does not use sentinels, and the front of the list is in `self.front`. If the list is empty, then `self.front` is `None`.

```
  def remove_all(self, x):
```

Problem 6: (15 pts) Consider the *Towers of Hanoi* problem. As usual, we are given three pegs A, B, C. At the start, there are $2n$ disks on peg A, numbered from disk 1 (the smallest disk) to disk $2n$ (the largest disk). The problem is to move all disks to peg B.

This time, a disk with an *even number* can only move *clockwise*: from peg A to B, from peg B to C, or from peg C to A. On the other hand, a disk with an *odd number* can only move *counter-clockwise*: from peg A to C, from peg B to A, or from peg C to B. So disks $1, 3, 5, \ldots, 2n-1$ move counter-clockwise, disks $2, 4, 6, \ldots, 2n$ move clockwise. And, of course, a larger disk may not lie on top of a smaller disk.

Fill in the following function to print out a sequence of moves to solve this problem. You must solve the problem by only filling in the *two gaps*—you cannot create other functions or modify the function outside the two gaps.

The function will be called as `hanoiEvenOdd(n, 'A', 'B', 'C')`.

```
def hanoiEvenOdd(n, src, dest, spare):
  # src -> dest -> spare -> src is clockwise order
  if (n == 1):  # there are only two disks
    print("Move disk %d from %s to %s" % (1, src, spare))
    # Fill in here




  else:
    # Fill in here
```