CS 331/401 Summer 2018

Midterm Exam

Instructions:

• This exam is closed-book, closed-notes. Computers of any kind are not permitted.

• For numbered, multiple-choice questions, fill your answer in the corresponding row on the “bubble” sheet.

• For problems that require a written solution (labeled with the prefix “WP”), write your answer in the space provided on the written solution sheet. Please write legibly and clearly indicate your final answer.

• Turn in the exam question packet, bubble sheet, and written solution sheet separately.
Basic Concepts (24 points):

1. Consider the following function definition:

   ```python
def foo(a='a', b='b', c='c', d='d', e='e'):  
       print(a + b + c + d + e)
   ```

   What is the output of `foo('p', *['q', 'r'], e='s')`?
   (a) abcde  
   (b) sdrqp  
   (c) pqrds  
   (d) p[qr]se

2. What are the contents of the list `lst` after the following code is executed?

   ```python
lst = [(x,2**x) for x in range(1,8,2)]
   ```
   (a) [(2, 4), (4, 16), (6, 64), (8, 256)]  
   (b) [(1, 2), (3, 8), (5, 32), (7, 128)]  
   (c) [(1, 3, 5, 7), (2, 8, 32, 128)]  
   (d) ([2, 4, 6, 8], [4, 16, 64, 256])

3. What are the contents of the dictionary `dct` after the following code is executed?

   ```python
dct = {}
   for x in 'a man a plan a canal'.split():
       if len(x) not in dct:
           dct[len(x)] = [x]
       else:
           dct[len(x)].append(x)
   ```
   (a) {'a': 1, 'man': 3, 'plan': 4, 'canal': 5}  
   (b) {1: 'a', 3: 'man', 4: 'plan', 5: 'canal'}  
   (c) {3: ['a', 'man'], 4: ['a', 'plan'], 5: ['a', 'canal']}  
   (d) {1: ['a', 'a', 'a'], 3: ['man'], 4: ['plan'], 5: ['canal']}

Page 2 of 6
4. What is the worst-case runtime complexity of retrieving the last element in an unsorted array-backed list of $N$ elements?
   
   (a) $O(1)$
   (b) $O(\log N)$
   (c) $O(N)$
   (d) $O(N^2)$

5. What is the worst-case runtime complexity of deleting an arbitrary element from an array-backed list of $N$ elements?
   
   (a) $O(1)$
   (b) $O(\log N)$
   (c) $O(N)$
   (d) $O(N^2)$

6. In order to keep an array-backed list of $N$ elements sorted after inserting a new value, a student proposes to (a) search for the insertion spot using binary search, then (b) insert the new value at that position. What is the runtime complexity of this operation?
   
   (a) $O(1)$
   (b) $O(\log N)$
   (c) $O(N)$
   (d) $O(N^2)$

7. What is the run-time complexity of inserting a new element at the beginning of a circular, doubly-linked list with a sentinel head?
   
   (a) $O(1)$
   (b) $O(\log N)$
   (c) $O(N)$
   (d) $O(N^2)$

8. Which implementation of the list ADT is best suited to an application where the most common operations are to insert and remove values from the beginning and end of the list?
   
   (a) built-in Python list
   (b) array-backed list
   (c) singly-linked list
   (d) doubly-linked list
9. Which of the following operations on the built-in Python list mutates the list?
   (a) extends
   (b) contains
   (c) __iter__
   (d) __getitem__

10. Which of the following is not true of all iterators in Python?
    (a) next can be called on them to obtain the next value (when available)
    (b) iter can be called on them to obtain an iterator
    (c) they raise StopIteration exceptions when they are out of elements
    (d) they are implemented using the yield keyword

11. Which correctly prepends value to a circular, doubly-linked list with a sentinel head?
    (a) n = LinkedList.Node(value, prior=self.head.prior, next=self.head.next)
        n.prior = n.prior = n
    (b) n = LinkedList.Node(value, prior=self.head, next=self.head.next)
        n.next = n.next = n
    (c) n = LinkedList.Node(value, prior=self.head, next=self.head.prior)
        n.next, n.prior = n, n.next.prior
    (d) n = LinkedList.Node(value, prior=self.head.next, next=self.head.next.next)
        n.next.prior = n.prior.next = n

12. If to_del refers to a node in a circular, doubly-linked list with a sentinel head, which of the following correctly removes the node from the list?
    (a) to_del.next = to_del.prior
        to_del.prior = to_del.next
    (b) to_del.prior.next = to_del.next
        to_del.next.prior = to_del.prior
    (c) to_del.prior.prior = to_del.next.next
        to_del.next.next = to_del.prior.prior
    (d) to_del.next, to_del.prior = to_del.prior, to_del.next
Lists and Dicts (8 points):

WP1 Implement dictify, which accepts two equal-length lists — ks and vs — and returns a dictionary with keys drawn from ks and lists of corresponding values drawn from vs.

E.g., dictify(['a', 'b', 'c'], ['apples', 'bees', 'cars']) returns the dictionary {'a': ['apples'], 'b': ['bees'], 'c': ['cars']}.  

E.g., dictify([1, 2, 3, 2, 1, 2], [10, 20, 30, 200, 100, 2000]) returns the dictionary {1: [10, 100], 2: [20, 200, 2000], 3: [30]}.  

Array-backed List (8 points):

WP2 Implement the array-backed list method del_span, which accepts parameters idx and n, and removes n elements starting at index idx. Your implementation should not use any other array-list methods, and may only perform the following operations on the backing list (named data in the provided skeleton code):

- len(self.data)
- Accessing a valid list index i, e.g., self.data[i]
- self.data.append(None)
- del self.data[len(self.data)-1]

E.g., calling del_span(2, 5) on the list with elements [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] results in the list being shortened to [0, 1, 7, 8, 9].  

E.g., calling del_span(7, 1) on the list with elements [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] results in the list being shortened to [0, 1, 2, 3, 4, 5, 6, 8, 9].
Linked List (8 points):

WP3 Implement the linked list method \texttt{reverse\_iter}, which returns an iterator that traverses all the elements in the circular, doubly-linked list \textit{backwards}. Assume that \texttt{self.head} refers to a sentinel node.

E.g., the following code would print out the contents of the linked list in reverse order (9, 8, \ldots, 1, 0):

```python
l = LinkedList()
for i in range(10):
    l.append(i)
for x in l.reverse_iter():
    print(x)
```