NAME ____ **ICA-17**

NetID ____________________@email.arizona.edu

Problem 1. Write a recursive function `reverse(alist)` that takes a built-in Python list `alist` as an argument and returns a new list that is the reverse of `alist`. The original list should be unchanged.

Problem 2. Write a recursive function `make_tuples(a, b)` that takes two lists `a` and `b` and returns a list of 2-tuples where the first tuple is `(a[0], b[0])`, and the second is `(a[1], b[1])`, and so on. The function stops when the shorter list runs out. For example, if `len(a)` is 3 and `len(b)` is 5, then `len(make_tuples(a,b))` is 3.

Problem 3. Write a function `sum_leaves(t)` that takes a binary tree `t` and sums the values of the leaf nodes. For example, if `t` is the following tree,

```
(8 (3 (2 None None) (5 None None)) (11 (9 None None) None))
```

then `sum_leaves(t)` would return 16. You may assume that all of the values of the nodes are integers and that the initial argument tree will not be empty.

Problem 4.
a)What is an invariant?

b) State two invariants for y at **Point A**.

```
x = len(input())
y = len(input())
y = y % x
if (y % 2 != 0):
    y += 1
else:
    y += 2
# Point A
print(y)
```

You may not use the following invariant: y > 0.

First invariant: ______________________________________

Second invariant: ______________________________________

Problem 5. For this problem, assume the following definitions of `LinkedList` and `Node` classes:

```
class LinkedList:
    def __init__(self):
         self._head = None
```

```
class Node:
    def __init__(self,value):
        self._value = value
        self._next = None
```

First, implement the following two methods for the `LinkedList` class:

- `add(self, node)`: adds `node` to the head of a linked list
- `remove(self)`: removes a node from the **tail** of a linked list; returns the node removed

Second, implement the queue ADT using a linked list as the underlying implementation. Specifically, you must implement the following methods of the queue ADT:

- `init(self)`
- `enqueue(self, item)`
- `dequeue(self)`

Note: You may access the attributes directly without getter and setter methods.

Problem 6. Consider the table below:

Key	Hash value	Probe decrement
10	3	1
2	2	1
19	5	2
14	0	2
24	3	3
23	2	3

Give the final configuration of a hash table T that results from inserting the keys listed above into an empty table T using double hashing. The order of insertion of the keys is: 23, 14, 10, 2, 19, 24.

0	1	2	3	4	5	6

Problem 7.

Assume that we have the following encodings for the three characters in the table:

Character	Encoding
A	1
B	11
C	101

a) What are the sequences of characters that could be represented by 111?

b) Write a well-formed encoded sequence that represents an encoding of a combination of four characters selected from the table above.

Problem 8. The following function counts the strings in *wordlist* that end with *tail*. It then returns the ratio of that count to the length of *wordlist*. However, the function has a bug. Write **four** unit tests for this function, including one test that **exposes** the bug.

Note: To specify a unit test, provide the arguments that are passed to the function for testing.

```
# buggy code
def ratio_ending_with(wordlist, tail):
    count = 0
    for word in wordlist:
        if word.endswith(tail):
            count += 1
    return count/len(wordlist)
```

value for `wordlist`	value for `tail`

Problem 9.

If there is time left, work in groups to review a topic covered in class, create a multiple choice exam question, and present the question to the class.