Spring, 2017 Name: _____

(Please *don't* write your id number!)

Exam 2: Python Programming with Recursion and Loops

Directions

The code for the `LispList` type used in the homework and in some problems on this exam is found in Figure 1 on the following page.
For this exam you are permitted one page of notes. It is a good idea to condense your notes into a small amount of ready reference material.
If you need more space, use the back of a page. Note when you do that on the front.
Before you begin, please take a moment to look over the entire test so that you can budget your time.
Clarity is important; if your answers are sloppy and hard to read, you may lose some points.

For Grading

Question:	1	2	3	Total
Points:	30	35	35	100
Score:				

```
# $Id: LispList.py,v 1.1 2017/02/27 04:52:50 leavens Exp $
import abc
class LispList(abc.ABC):
    pass
class Nil(LispList):
    def __init__(self):
        """Initialize this empty list"""
        pass
    def __eq__(self, lst):
        """Return True just when lst is also an instance of Nil."""
        return isinstance(lst, Nil)
    def __repr__(self):
        """Return a string representing this Nil instance."""
        return "Nil()"
    def __str__(self):
        """Return a string showing the elements of self."""
        return "[]"
    def isEmpty(self):
        """Return whether this list is empty."""
        return True

class Cons(LispList):
    def __init__(self, hd, tl):
        """Initialize this Cons with head hd and tail tl."""
        self.car = hd
        self.cdr = tl
    def __eq__(self, lst):
        """Return True just when self is structurally equivalent to lst."""
        return isinstance(lst, Cons) and lst.first() == self.first() \
            and lst.tail() == self.tail()
    def __repr__(self):
        """Return a string representing this list."""
        return "Cons(" + repr(self.first()) + ", " + repr(self.tail()) + ")"
    def elements_str(self):
        """Return a string of the elements of self, separated by commas."""
        if self.tail().isEmpty():
            return str(self.first())
        else:
            return str(self.first()) + ", " + self.tail().elements_str()
    def __str__(self):
        """Return a string showing the elements of self."""
        return "[" + self.elements_str() + "]"
    def isEmpty(self):
        """Return whether this list is empty."""
        return False
    def first(self):
        """Return the first element of this list."""
        return self.car
    def tail(self):
        """Return the rest of this list."""
        return self.cdr
```

Figure 1: Code for `LispList`, which is used in some problems.

1. (30 points) [Programming] Define a Python function, `multBy(lst,val)`, of

   type: `(LispList(int),int) -> LispList(int)`

   that takes a LispList of ints, `lst`, and an int, `val`, and returns a new LispList of ints that is like `lst` but with each element multiplied by `val`; that is, the n^{th} element of the result should be `val` times the n^{th} element of `lst`. (The argument `lst` should not be modified at all.)

   Tests for this problem appear in Figure 2.

```
# $Id: test_multBy.py,v 1.1 2017/02/27 04:52:50 leavens Exp $
from LispList import *
from multBy import *
def test_multBy():
    """Testing for multBy()."""
    assert multBy(Nil(), 8) == Nil()
    lst3223 = Cons(3, Cons(2, Cons(2, Cons(3, Nil()))))
    assert multBy(lst3223, 5) == Cons(15, Cons(10, Cons(10, Cons(15, Nil()))))
    # multBy does not change the argument list
    assert lst3223 == Cons(3, Cons(2, Cons(2, Cons(3, Nil()))))
    assert multBy(Cons(10, Cons(20, Cons(30, Nil()))), 7) \
        == Cons(70, Cons(140, Cons(210, Nil())))
    assert multBy(Cons(3, Cons(4, Nil())), 0) == Cons(0, Cons(0, Nil()))
```

Figure 2: Tests for `multBy`.

---

2. (35 points) [Programming] Define a Python function, `sumSqDiffs(x, lst)` of

   type: (**int**, LispList(**int**)) -> **int**

   that when given an int, x, and a LispList of ints, `lst`, returns the sum of the squares of the differences between x and each element of `lst`. Tests for this problem appear in Figure 3.

```
# $Id: test_sumSqDiffs.py,v 1.1 2017/02/27 04:52:50 leavens Exp $
from LispList import *
from sumSqDiffs import *
def test_sumSqDiffs():
    """Testing for sumSqDiffs()."""
    assert sumSqDiffs(5, Nil()) == 0
    assert sumSqDiffs(4, Cons(1, Nil())) == (4-1)**2
    lst3223 = Cons(3, Cons(2, Cons(2, Cons(3, Nil()))))
    assert sumSqDiffs(7, lst3223) == (7-3)**2 + (7-2)**2 + (7-2)**2 + (7-3)**2
    # sumSqDiffs does not change the argument list
    assert lst3223 == Cons(3, Cons(2, Cons(2, Cons(3, Nil()))))
    assert sumSqDiffs(20, Cons(10, Cons(20, Cons(30, Nil())))) \
        == (20-20)**2 + (20-10)**2 + (20-20)**2 + (20-30)**2
    assert sumSqDiffs(5, Cons(3, Cons(4, Nil()))) == 5
```

Figure 3: Tests for `sumSqDiffs`.

3. (35 points) [Programming] Define a function, `count5s(num)`, of

   `type: int -> int`

   that takes a non-negative int, `num`, and returns the number of 5's that appear in the decimal representation of `num`.

   In your solution you must *not* convert `num` into a string or a sequence, and thus you must *not* use formatting or the functions **`str()`** or **`repr()`**. However, you may use Python's modulo operator, `%`, and Python's integer division operator `//`. (Recall that in Python `15 % 10` is 5 and that `15 // 10` is 1.)

   Examples appear in Figure 4.

```
# $Id: test_count5s.py,v 1.1 2017/03/01 02:38:11 leavens Exp leavens $
from count5s import *
def test_count5s():
    """Testing for count5s()"""
    assert count5s(0) == 0
    assert count5s(4) == 0
    assert count5s(5) == 1
    assert count5s(10) == 0
    assert count5s(15) == 1
    assert count5s(55) == 2
    assert count5s(99550005) == 3
    assert count5s(55555) == 5
    assert count5s(525153545000321) == 5
    assert count5s(555555555555555555555) == 21
```

Figure 4: Tests for `count5s`.
