Homework 3: Recursion in Python

See Webcourses and the syllabus for due dates.

General Directions

This homework should be done individually. Note the grading policy on cooperation carefully if you choose to work together (which we don't recommend).
In order to practice for exams, we suggest that you first write out your solution to each problem on paper, and then check it typing that into the computer.
You should take steps to make your Python code clear, including using symbolic names for important constants and using helping functions or procedures to avoid duplicate code.
Follow the grammar; we will take points off if you do not.
Also, it is a good idea to test your code yourself first, before running our tests, as it is easier to debug when you run the code yourself.
Tests that are provided in hw3-tests.zip, which consists of several python files with names of the form `test_`*f*`.py`, where *f* is the name of the function you should be writing, and also some other files. Your function *f* should go in a file named *f*`.py` and the function itself should be named *f*. These conventions will make it possible to test using pytest.
Pytest is installed already on the Eustis cluster. If you need help installing pytest on your own machine, see the course staff or the running Python page.

Running Pytest from the Command Line

After you have pytest installed, and after you have written your solution for a problem that asks for a function named *f*, you can run pytest on our tests for function *f* by executing at a command line

```
pytest test_f.py > f_tests.txt
```

which puts the output of the testing into the file *f*`_tests.txt`.

Running Pytest from within IDLE

You can also run pytest from within IDLE. To do that first edit a test file with IDLE (so that IDLE is running in the same directory as the directory that contains the files), then from the Run menu select "Run module" (or press the F5 key), and then execute the following statements:

```
import pytest
pytest.main(["test_f.py", "--capture=sys"])
```

which should produce the same output as the command line given above. Then you can copy and past the test output into the file *f*`_tests.txt` to hand in.

What to turn in

For problems that ask you to write a Python procedure, upload your code as an ASCII file with suffix `.py`, and also upload the output of running our tests (as an ASCII file with suffix `.txt`).

Problems

These problems use the `LispList` type defined in the file `LispList.py`, which is included in the homework zip file. To make this work, put the following import in each file you write.

```
from LispList import *
```

Note that you are *not* allowed to use the built-in lists of Python; doing so will result in loss of all points for the problem. Also, do not attempt to modify the LispList objects passed as arguments to the functions you write; that will result in incorrect solutions.

1. (10 points) [Programming] Write a Python function, `sub_from_each(loi, n)` of

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

   that takes a LispList of ints, `loi` and an int, `n`, and returns a new LispList that is like `loi`, except that the i^{th} element of the result is the i^{th} element of `loi` minus the value of `n`. Tests for this are shown in Figure 1.

```
# $Id: test_sub_from_each.py,v 1.2 2017/02/09 18:57:34 leavens Exp $
from LispList import *
from sampleLispLists import *  # test data, such as loi1_6, used below
from sub_from_each import *
def test_sub_from_each():
    assert sub_from_each(loi1_6, 2) \
        == Cons(-1, Cons(0, Cons(1, Cons(2, Cons(3, Cons(4, nil))))))
    assert sub_from_each(nil, 99) == nil
    assert sub_from_each(Cons(5, Cons(5, nil)), 3) == Cons(2, Cons(2, nil))
    assert sub_from_each(Cons(5, nil), 3) == Cons(2, nil)
    assert sub_from_each(loi999_twice_3, 99) \
        == Cons(900, Cons(900, Cons(3-99, nil)))
    assert sub_from_each(loi5down, 1) \
        == Cons(4, Cons(3, Cons(2, Cons(1, Cons(0, nil)))))
```

Figure 1: Tests for `sub_from_each.py`. This uses the module `sampleLispLists` shown in Figure 2 on the following page.

   The tests in Figure 1 use some sample Lisp Lists that are collected into the module `sampleLispLists`, which is shown in Figure 2 on the following page. (This is done to avoid repetition in the testing code.)

   Remember to turn in both your file `sub_from_each.py` and the output of running our tests with `pytest`. Your code and also the output of running our tests should be submitted to webcourses as ASCII text files that you upload.

2. (10 points) [Programming] Define a Python function, `removeAll(val, loi)`, of

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

   which takes an int, `val`, and a Lisp list of ints, `loi`, and returns a new Lisp list that is like `loi` except that the every occurrence of `val` in `loi` (as determined by `==`) is not present in the result. Note that, despite the name, the argument list is unaffected by this function. Tests for `removeAll()` are found in Figure 3 on the next page.

   Hint: since you are not to modify the argument list in any way, your code should, in essence, make a copy of `loi`, but omit copying every element that is `==` to `val`, if any.

   Remember to turn in both your file `remove_first.py` and the output of running our tests with `pytest`.

```
# $Id: sampleLispLists.py,v 1.1 2017/02/09 18:57:34 leavens Exp $
from LispList import *
# Several samples of LispLists
nil = Nil()
loi12321 = Cons(1, Cons(2, Cons(3, Cons(2, Cons(1, nil)))))
loi999999 = Cons(999999, nil)
loi999_twice_3 = Cons(999, Cons(999, Cons(3, nil)))
loi1_6 = Cons(1, Cons(2, Cons(3, Cons(4, Cons(5, Cons(6, nil))))))
loi55 = Cons(5, Cons(5, nil))
loi5down = Cons(5, Cons(4, Cons(3, Cons(2, Cons(1, nil)))))

def fromToOn(n, m, lst):
    """type: (int, int, LispList(int)) -> LispList(int)
    Return the LispList(int) of form Cons(n, Cons(n+1, ..., Cons(m, lst)...))"""
    if n > m:
        return lst
    else:
        return Cons(n, fromToOn(n+1, m, lst))

loi1_12 = fromToOn(1, 12, Nil())
loi2_9 = fromToOn(2, 9, Nil())
loiup10down5 = fromToOn(1,10, loi5down)
```

Figure 2: The module `sampleLispLists`, which is used in testing for this homework.

```
# $Id: test_removeAll.py,v 1.2 2017/02/21 20:26:48 leavens Exp leavens $
from LispList import *
from sampleLispLists import *  # names like nil and lst12321 defined there
from removeAll import *
def test_removeAll():
    """Testing for removeAll."""
    assert removeAll(1, loi12321) == Cons(2, Cons(3, Cons(2, nil)))
    # Note: the argument is unchanged!
    assert loi12321 == Cons(1, Cons(2, Cons(3, Cons(2, Cons(1, nil)))))
    assert removeAll(2, loi12321) == Cons(1, Cons(3, Cons(1, nil)))
    assert removeAll(3, loi12321) == Cons(1, Cons(2, Cons(2, Cons(1, nil))))
    assert removeAll(5, loi12321) == loi12321
    assert removeAll(999999, nil) == nil
    assert removeAll(1, Cons(2, Cons(3, Cons(2, Cons(1, nil))))) \
        == Cons(2, Cons(3, Cons(2, nil)))
    assert removeAll(999, loi999_twice_3) == Cons(3, nil)
    assert removeAll(1, loiup10down5) \
        == fromToOn(2,10,Cons(5, Cons(4, Cons(3, Cons(2, nil)))))
    assert removeAll(-1, Cons(-1, Cons(-1, Cons(-1, nil))))
```

Figure 3: Tests for `removeAll`. This uses the module `sampleLispLists` shown in Figure 2.

3. (10 points) [Programming] Define a Python function, `remove_first(val, loi)`, of

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

   which takes an int, `val`, and a Lisp list of ints, `loi`, and returns a new Lisp list that is like `loi` except that the first occurrence of `val` in `loi` (as determined by ==) is not present in the result. Note that, despite the name, the argument list is unaffected by this function. Tests for `remove_first()` are found in Figure 4.

   Hint: this is like `removeAll`, but the code needs to not remove any occurrences of `val` after the first one is found.

```
# $Id: test_remove_first.py,v 1.3 2017/02/21 20:26:48 leavens Exp leavens $
from LispList import *
from sampleLispLists import *
from remove_first import *
def test_remove_first():
    """Testing for remove_first"""
    assert remove_first(1, loi12321) == Cons(2, Cons(3, Cons(2, Cons(1, nil))))
    # Note: the argument is unchanged!
    assert loi12321 == Cons(1, Cons(2, Cons(3, Cons(2, Cons(1, nil)))))
    assert remove_first(2, loi12321) == Cons(1, Cons(3, Cons(2, Cons(1, nil))))
    assert remove_first(3, loi12321) == Cons(1, Cons(2, Cons(2, Cons(1, nil))))
    assert remove_first(5, loi12321) == loi12321
    assert remove_first(999999, nil) == nil
    assert remove_first(1, Cons(2, Cons(3, Cons(2, Cons(1, nil))))) \
        == Cons(2, Cons(3, Cons(2, nil)))
    assert remove_first(999, loi999_twice_3) == Cons(999, Cons(3, nil))
    assert remove_first(-1, Cons(-1, Cons(-1, Cons(-1, nil)))) \
        == Cons(-1, Cons(-1, nil))
```

Figure 4: Tests for `remove_first`. This uses the module `sampleLispLists` shown in Figure 2 on page 3.

   Remember to turn in both your file `remove_first.py` and the output of running our tests with `pytest`.

4. (10 points) [Programming] Define a Python function, `length(lst)`, which for any type T is of

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

   and which takes a Lisp list of elements of type T, `lst`, and returns the number of elements in `lst`. Tests for `length()` are found in Figure 5 on the next page.

   Remember to turn in both your file `length.py` and the output of running our tests with `pytest`.

5. (10 points) [Programming] Define a Python function, `member(val, lst)`, which for any type T is of

   ```
   type: (T, LispList(T)) -> bool
   ```

   and which takes a value of type T, `val`, and a Lisp list of elements of type T, `lst`, and returns a Boolean that indicates whether `val` is == to any element of `lst`. Tests for `member()` are found in Figure 6 on the following page.

   Remember to turn in both your file `member.py` and the output of running our tests with `pytest`.

6. (15 points) [Programming] Define a Python function, `subseteq(lst1, lst2)`, which for any type T is of

   ```
   type: (LispList(T), LispList(T)) -> bool
   ```

```
# $Id: test_length.py,v 1.2 2017/02/21 20:26:48 leavens Exp leavens $
from LispList import *
from sampleLispLists import *
from length import *
def test_length():
    assert length(nil) == 0
    assert length(loi12321) == 5
    # Note: the argument is unchanged!
    assert loi12321 == Cons(1, Cons(2, Cons(3, Cons(2, Cons(1, nil)))))
    assert length(loi12321.tail()) == 4
    assert length(loi1_12) == 12
    assert length(loi2_9) == 8
    assert length(fromToOn(1,100,Nil())) == 100
    assert length(loiup10down5) == 15
    assert length(Cons("also", Cons("string", Cons("lists", nil)))) == 3
    assert length(Cons(3.14, Cons(2.78, nil))) == 2
```

Figure 5: Tests for `length`. This uses the module `sampleLispLists` shown in Figure 2 on page 3.

```
# $Id: test_member.py,v 1.2 2017/02/21 20:48:23 leavens Exp leavens $
from LispList import *
from sampleLispLists import *  # names like nil and lst12321 defined there
from member import *
def test_member():
    assert member(1, loi12321)
    # Note: the argument is unchanged!
    assert loi12321 == Cons(1, Cons(2, Cons(3, Cons(2, Cons(1, nil)))))
    assert member(2, loi12321)
    assert member(3, loi12321)
    assert not member(5, loi12321)
    assert member(6, loi1_6)
    assert member(999, loi999_twice_3)
    assert member(3, loi999_twice_3)
    assert member(5, loiup10down5)
    assert member(10, loiup10down5)
    assert member(150, fromToOn(10,151, Nil()))
    assert not member(nil, nil)  # where T is a LispList type
    assert member(nil, Cons(nil, nil))
    assert member(Cons(1, nil), Cons(nil, Cons(Cons(1, nil), nil)))
```

Figure 6: Tests for `member`. This uses the module `sampleLispLists` shown in Figure 2 on page 3.

and which takes as arguments two Lisp lists whose elements are of type `T`, and returns a Boolean that indicates whether every element of `lst1` is also an element of `lst2`. That is, if you regard the lists as representing sets, then the set of the elements in the first list is a subset (or equal to) the set of elements in the second set. All tests comparing elements of type `T` should use == (that is elements x and y are considered equal if they are such that $x == y$). Tests for `subseteq()` are found in Figure 7 on the following page.

Do *not* use `length` or any Python lists in your solution.

Hint: you may want to use `member` in your solution.

```
# $Id: test_subseteq.py,v 1.1 2017/02/09 20:47:15 leavens Exp leavens $
from LispList import *
from sampleLispLists import *  # names like nil and lst12321 defined there
from subseteq import *
def test_member():
    assert subseteq(Cons(1, nil), loi12321)
    # Note: the argument is unchanged!
    assert loi12321 == Cons(1, Cons(2, Cons(3, Cons(2, Cons(1, nil)))))
    assert not subseteq(loi12321, Cons(1, nil))
    # Note: the argument is unchanged!
    assert loi12321 == Cons(1, Cons(2, Cons(3, Cons(2, Cons(1, nil)))))
    assert subseteq(Cons(3, Cons(1, nil)), loi12321)
    assert not subseteq(Cons(3, Cons(4, Cons(1, nil))), loi12321)
    assert subseteq(Cons(1, Cons(4, nil)), loi1_6)
    assert subseteq(loi1_6, loi1_6)
    assert subseteq(Cons(3, Cons(999, nil)), loi999_twice_3)
    assert subseteq(Cons(3, nil), loi999_twice_3)
    assert subseteq(Cons(999, nil), loi999_twice_3)
    assert subseteq(Cons(10, Cons(9, Cons(7, nil))), loiup10down5)
    assert not subseteq(Cons(1, Nil()), Nil())
    assert not subseteq(Cons(2, Cons(1, Nil())), Nil())
    assert not subseteq(Cons(2, Cons(1, Nil())), Cons(2, Nil()))
```

Figure 7: Tests for `subseteq`. This uses the module `sampleLispLists` shown in Figure 2 on page 3.

Remember to turn in both your file `subseteq.py` and the output of running our tests with `pytest`.

Points

This homework's total points: 65.