CS61A Lab 5

Week 5, 2012

Map and Filter

0.) Try to guess what the following print. Check your solution using the Python interpreter.

```
>>> tuple(map(lambda x: x * x, (1, 2, 3, 4, 5, 6)))

>>> tuple(filter(lambda x: x % 2 == 0, (1, 2, 3, 4, 5, 6)))

>>> def is_even(x):
...     return x % 2 == 0
...
>>> def square(x):
...     return x * x
...
>>> tuple(map(square, filter(is_even, (1, 2, 3, 4, 5, 6))))
```

1.) Write the `sizes` procedure using `map`. This procedure will take a sequence of tuples and will return a sequence of the sizes of each of the tuples in the original input.

```
def sizes(seq):
    """Takes an input sequence of tuples, seq, and returns a sequence with the
    corresponding lengths of each tuple in seq.

    Arguments:
    seq -- A sequence of tuples.

    >>> sizes(((1,), (2, 3), (4, 5, 6)))
    (1, 2, 3)
    """
    "*** Your code here. ***"
```

2.) Write the `odd_len_only` procedure using `filter`. This procedure will take a sequence of tuples and return a sequence containing only those tuples in the original sequence which had an odd length.

```
def odd_len_only(seq):
    """Takes an input sequence of tuples, seq, and returns a sequence with only
    the tuples which had odd length.

    Arguments:
    seq -- A sequence of tuples.

    >>> odd_len_only(((1,), (2, 3), (4, 5, 6)))
    ((1,), (4, 5, 6))
    """
    "*** Your code here. ***"
```

More fun with Rlists and Recursion!

Recall the implementation of Rlists from lecture and discussion section:

```
empty_rlist = None

def make_rlist(first, rest = empty_rlist):
    return first, rest

def first(r):
    return r[0]

def rest(r):
    return r[1]
```

3) First, let's implement the len_rlist function. You have seen this in both lecture and discussion section, but try to write it from scratch using rescursion (ie, no loops!).

```
def len_rlist(r):
    """Return the length of recursive list r.

    >>> len_rlist(empty_rlist)
    0
    >>> len_rlist(make_rlist(6, make_rlist(2, make_rlist(-12))))
    3
    """
    "*** Your code here. ***"
```

4) Now use a similar technique to write a (recursive) sum function for recursive lists.

```
def sum_rlist(r):
    """Return the sum of items in the recursive list r, assuming they are numbers.

    >>> sum_rlist(empty_rlist)
    0
    >>> sum_rlist(make_rlist(-1, make_rlist(3, make_rlist(8))))
    10
    """
```

5) With only a very small modification to the previous function, you can also create a str_concat_rlist function that concatenates strings in a recursive list.

```
def str_concat_rlist(r):
    """Returns a string that is the concatenation of the strings in the recursive list r

    >>> str_concat_rlist(empty_rlist)
    ''
    >>> str_concat_rlist(make_rlist('How ', make_rlist('do ', make_rlist('you ', make_rlist('sum ', make_rlist('strings?')))))
    'How do you sum strings?'
    """
    "*** Your code here. ***"
```

6) Now let's write an insert function that inserts an item at a specific index in the recursive list. If the index is greater than the current length, you should insert the item at the end of the list. HINT: This will be much easier to implement using recursion than a loop!

```
def insert_rlist(r, item, index):
    """ Returns a recursive list matching r but with the given item inserted at the specified index.
    If the index is greater than the current length, the item is appended to
    the end of the list.

    Arguments:
    r -- A recursive list.
    item -- the item to be inserted
    index -- the index at which to insert the item
```

```
    >>> r = make_rlist('I', make_rlist(' love', make_rlist(' recursion')))
    >>> str_concat_rlist(insert_rlist(r, ' using', 2))
    'I love using recursion'
    >>> str_concat_rlist(insert_rlist(r, '!', 100))
    'I love recursion!'
    """
    "*** Your code here. ***"
```

7) Let's check to make sure you are using the recursive list abstract data type properly. Replace the initial implementation with the one below in your code:

```
empty_rlist = 'empty_rlist'

def make_rlist(first, rest = empty_rlist):
    return rest, first

def first(r):
    return r[1]

def rest(r):
    return r[0]
```

See what we did there? We simpy changed the order of how the list data is stored in the tuple. We also modified the way an empty rlist is represented by using a specific string instead of the NoneType. This is just one of the (literally) infinite ways we could have implemented RIists.
You should now re-run our doctests with the new implementation. Does everything still work? If not, fix any code that breaks an abstraction barrier so that your functions will be agnostic to the underlying implementation of RIists.

8) Finally, since you're on a roll, see if you can implement a filter function for recursive lists. Again, recursion is your friend!

```
def filter_rlist(predicate, r):
    """ Returns a recursive list matching r but with items failing the predicate removed.

    Arguments:
    predicate -- A function that takes a single argument and returns True or False.
    r -- A recursive list.

    >>> from math import sqrt
    >>> r = make_rlist(53, make_rlist(16, make_rlist(625, make_rlist(50, make_rlist(49)))))
    >>> sum_rlist(filter_rlist(lambda x : sqrt(x)%1==0, r))
    690
    >>> s = make_rlist('I', make_rlist(' love', make_rlist(' recursion', make_rlist('!'))))
    >>> str_concat_rlist(filter_rlist(lambda x: len(x) < 7, s))
    'I love!'
    """
    "*** Your code here. ***"
```

Now that you are a master of recursion, show it off to all of your friends and use it in your daily life. Can you find a way to recursively walk to your next class?