CS3101-1 Python, Fall 2014: Problem Set 3

Daniel Bauer

Total points: 20
Due date: Sep 30, 11:59pm EST

Submission instructions:
Put the code for each part in its own file (`part1.py` etc.).

Place the files for all problems in a directory named `[your_uni]_week[X]`, where X is the number of the problem set. For instance if your uni is xy1234 and you are submitting the problem set for the first week, the directory should be called `xy1234_week1`. Either zip or tar and gzip the directory (using `tar -c xy1234_week1 | gzip > xy1234_week1.tgz`) and upload it to your Drop Box on the Courseworks page for this class.

Please pay attention to the general guidelines/homework policy on the course website.

Part1 (16pt) - Functional Programming

A nested list is a list that can contain other lists. Consider the following example for a nested list.

```
nlist = [1, [2, [3, [4, 5]]], [6, [7, [8, [9]], 10]]]
```

a. (4pt) **Printing Nested Lists**: Nested lists are somewhat difficult to read. Write a function `print_nlist(nlist)` that prints a nested list `nlist` in the following format

```
>>> print_nlist(nlist)
.1
.....2
.........3
.............4
.............5
.....6
.........7
.............8
..............
```

Hint: Use a nested function within `print_nlist` that calls itself recursively for each sub-list. The nested function should have an additional parameter `indent` which keeps track of the indentation for each recursion level.

b. (4pt) **Mapping Nested Lists**: Write a function `map_nlist(nlist, fun)` that returns a new nested list in which each element `n` that is not a list (i.e. a number in the example) has been replaced with the result of applying `fun` to `n`. The new list has the same nesting structure as the old list. For instance:

```
>>> map_nlist(nlist, lambda x: x*2)
[2, [4, [6, [8, 10]]], [12, [14, [16, [18]], 20]]]
```

c. (4pt) **Combining nested lists**: Write a function `combine_nlist(nlist, init, combiner)`. In each recursion step `combine_nlist` keeps track of some current value, initialized to `init`. It then updates the current value by applying `combiner` repeatedly to the current value and the result of processing the next element of `nlist` recursively. Eventually the recursion step returns the current value.

For instance, you can call `combine_nlist` to compute the sum of all integers in a nested list:

```
>>> combine_nlist(nlist, 0, lambda x,y: x+y)
55
```

d. (4pt) **Flattening nested lists**: write a function `flatten_nlist(nlist)`, that produces a 'flattened' representation of a nested list.

```
>>> flatten_nlist(nlist)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Hint: While all correct solutions to part (d) will receive full credit, this is easy to solve using `combine_nlist`.

Part2 (4pt) - Generators

Write a generator function `ngrams(n,s)` that takes a list of words (a sentence) and acts as an iterator over all n-grams in this sentence. An `n`-gram is any contiguous sequence of words of length `n` that occurs in the sentence.

```
>>> s = "the quick red fox jumps over the lazy brown dog"
>>> ngrams(3, s)
>>> <generator object ngrams at 0x109a36ca8>
>>> for x in ngrams(3,s.split()):
...    print(x)
...
['the', 'quick', 'red']
['quick', 'red', 'fox']
['red', 'fox', 'jumps']
['fox', 'jumps', 'over']
['jumps', 'over', 'the']
['over', 'the', 'lazy']
['the', 'lazy', 'brown']
['lazy', 'brown', 'dog']
```