cs1120: Exam 1

Due: Thursday, 11 October at 3:30pm (in class)

UVA ID (e.g., wrw6y) :

Directions

Work alone. You may not discuss these problems or anything related to the material covered by this exam with anyone except for the course staff between receiving this exam and turning it in.

Open resources. You may use any books you want, lecture notes, slides, your notes, and problem sets. You may ***not*** use PyCharm or any Python or Udacity Python, but it is not necessary to do so. You may also use external non-human sources including books and web sites. If you use anything other than the course books, slides, and notes, cite what you used. You may not obtain any help from other humans other than the course staff.

Answer well. Answer all questions 1-9 (question 0 is your UVA ID in two places and turning in a stapled, printed exam, which hopefully everyone will receive full credit for), and optionally answer questions 10-11.

You may either: (1) print out this exam and write your answers on it or (2) write your answers directly into the provided Word template and print the result out. Whichever one you choose, you must turn in your answers printed **on paper** and they must be clear enough for us to read and understand. You should not need more space than is provided to write good answers, but if you want more space you may attach extra sheets. If you do, make sure they are clearly marked.

The questions are not necessarily in order of increasing difficulty, so if you get stuck on one question you should continue on to the next question. There is no time limit on this exam, but it should not take a well-prepared student more than a few hours to complete. It may take you longer, though, so please do not delay starting the exam. There is no valid excuse (other than a medical or personal emergency) for running out of time on this exam.

No "snow jobs". If you leave a question **blank**, you will receive **three** points for it. If you have no idea and waste our time with long-winded guessing, we will be less sanguine and the grading will be more sanguine. :-)

Use any Python procedure from class. In your answers, you may use any Python procedure that appears in the lecture notes or in the book without redefining it (e.g., len, filter, sort, find_best, etc.). If there are multiple similar names (e.g., len vs. length, map vs. list_map), use whichever you like.

Full credit depends on the clarity and elegance of your answer, not just correctness. Your answers should be as short and simple as possible, but not simpler. Your programs will be judged for correctness, clarity and elegance, but you will not lose points for trivial errors (such as missing a closing parenthesis).

UVA ID *again* (e.g., wrw6y) :

Your Scores

0	1	2	3	4	5	6	7	8	9	EC	Total
10	10	10	10	10	10	10	10	10	10	2	100

(Your scores are recorded on the second page so that they are not visible to other students when tests are distributed or passed back.)

1. Consider the following grammar:

S ::= N
N ::= A B C X | D E F X
A ::= edith | simone
B ::= de | ε
C ::= wharton | beauvoir
D ::= percy
E ::= bysshe | ε
F ::= shelley
X ::= and S | ε

The start symbol is S. The symbol ε denotes the empty string.
(a) Is the language of this grammar infinite or finite?
(b) Give a string of length six (i.e., six words) that is in the language of the grammar.

(a)

(b)

2. Consider the following Python definition.

```
def strainer(pred, lst):
        if lst == []:
                return []
        elif pred(lst[0]):
                return [lst[0]] + strainer(pred, lst[1:])
        else:
                return [lst[0]]
```

Provide a convincing argument that **`strainer`** is not equivalent to **`filter`**. (Hint: provide inputs on which they behave differently.)

3. Write a Python procedure **`censor`** that accepts a list of strings as input. It should return the same list of strings in the same order, but with all instances of "profanity" replaced by "bleep". For example:

```
>>> censor(["with", "these", "profanity", "snakes"])
["with", "these", "bleep", "snakes"]
>>> censor(["this"] + ["profanity", "plane"])
["this", "bleep", "plane"]
```

```
def censor(lst):
```

4. Define a `make_list_min_cutoff` procedure that takes one input, a number, and *produces a procedure* as output. The output procedure is a procedure that takes a list of numbers as input, and produces as output that same list but with all entries *less than* the original element removed. (Hint: Review those curve transformations from Problem Set 3.) For example:

```
>>> (make_list_min_cutoff 5)
<function>
>>> (make_list_min_cutoff 3)( [1,2,3,4,5] )
[3, 4, 5]
>>> (make_list_min_cutoff 7)( [-1,8,-3,2] )
[8]
```

```
def make_list_min_cutoff(cutoff):
```

5. Define a procedure **`nested_catenate`** that takes one input: a list. The list may contain string elements, but it may also contain other lists (which may themselves contain other lists, hence "nested"). The procedure should return the string concatenation the elements anywhere in the list, in presentation order. For example:

```
>>> isinstance([1,2,3], list)
True
>>> isinstance("hello", list)
False
>>> nested_catenate(["a", "b", "c", "d"])
"abcd"
>>> nested_catenate(["w", ["x", "y"], "z"])
"wxyz"
>>> nested_catenate([])
""
>>> nested_catenate( ["1", ["2", ["3"], "4", []], "5"])
"12345"
```

(Hint 1: `rewrite_lcommands`. Hint 2: There are three cases. The list may be empty, its first element may itself be a list, or its first element may be a string. Two of those three cases involve recursive calls.)

```
def nested_catenate(lst):
```

6. Answer each of the following questions about function growth. Give an argument that explains each answer. For example, to prove that n is in O($n/2$) you might demonstrate the constants $n_0 = 1$ and $c = 2$. If there is any ambiguity, use the definitions from Chapter 7 of the course book.

Is $2n-8$ in O(n) ? Why or why not?

Is n^5 in O(n^4) ? Why or why not?

Is n in Ω($n/2 + 7$) ? Why or why not?

Is log n in Ω(n^4) ? Why or why not?

Is $5n^2$ in Θ(n^2+n^3) ? Why or why not?

7. Consider the following four procedures:

```
def filter(f, lst):           # Same as in class/notes/book
      if lst == []: return []
      if f(lst[0]): return [lst[0]] + filter(f, lst[1:])
      return filter(f, lst[1:])

def my_reverse(lst):          # Reverse the list
      def reverse_helper(x,y):
            if x == []: return y
            return reverse_helper(x[1:], [x[0]] + y)
      return reverse_helper(lst, [])

def revfilter_alpha(f, lst):  # Reverse and filter ...
      return my_reverse(filter(f, lst))

def revfilter_beta(f, lst):   # Reverse and filter ...
      if lst == []: return []
      return revfilter_beta(f, lst[1:]) + \
            (lst[0]  if f(lst[0])  else [])
```

Give the running time of `my_reverse` and `revfilter_alpha` and `revfilter_beta` in Big Theta **Θ** notation. Assume `f` runs in constant time. Which of `revfilter_alpha` and `revfilter_beta` is faster? Can there be an asymptotically faster `revfilter` procedure? (Hint: either demonstrate "yes" by giving the procedure, or argue that no such faster procedure is possible.)

(a) my_reverse is Θ

(b) revfilter_alpha is Θ

(c) revfilter_beta is Θ

(d) The faster procedure is

(e)

8. Consider the task of sorting a music playlist for a portable music player or cell phone. We will represent each song as a list of its artist and its title. We wish to sort the playlist by *title first*, and then by artist. Write a procedure `sort_playlist` that accepts one argument, a list of songs, and returns that same list of songs, but sorted as per the description above. Example:

```
>>> song1 = ["ABBA", "Mamma Mia"]
>>> song2 = ["Taylor Swift", "Love Story"]
>>> song3 = ["Beyonce", "Single Ladies"]
>>> song4 = ["Randy Newman", "Love Story"]
>>> song5 = ["Lonely Island", "I'm On A Boat"]
>>> sort_playlist([song1, song2, song3, song4, song5])
[     ["Lonely Island", "I'm On A Boat"],
      ["Randy Newman", "Love Story"],
      ["Taylor Swift", "Love Story"],
      ["ABBA", "Mamma Mia"],
      ["Beyonce", "Single Ladies" ] ]
>>> "ABBA" < "Beyonce"  # < does dictionary ordering on strings
True
>>> "ABBA" == "Beyonce" # check string equality
False
```

(No points off if you sort Z-A instead of A-Z. All points off if you sort by artist first.)

```
def sort_playlist(lst):
```

9. In the not too distant future (next Sunday A.D.), YouTube, Twitter and Facebook have merged together to form YouTwitFace, one social networking site to rule them all and in the darkness bind them. You have been asked to write a friend analysis procedure for this site. Lists of friends are available. Given such a list, you are to find the person with the most friends.

You should write two procedures. The first, `number_of_friends`, takes as input a lists of pairs of strings (representing friends) and a particular string (the person under consideration) and returns an integer (the number of friends of that person). The second, `most_friends`, takes as input a **non-empty** list of pairs of strings (representing friends, as before). It should return the name of the person with the most friends. (This can be done in under 10 lines total.) Example:

```
>>> friends = [ [ "rachel", "monica" ],    # rachel & monica are friends
                [ "phoebe", "rachel" ],    # as are phoebe & rachel, ...
                [ "rachel", "joey" ],
                [ "phoebe", "joey" ],
                [ "monica", "chandler" ] ]
>>> number_of_friends(friends, "rachel")
3                                          # monica, phoebe, joey
>>> number_of_friends(friends, "chandler")
1                                          # monica
>>> most_friends(friends)
"rachel"
```

```
def number_of_friends(lst, who):




def most_friends(lst):
```

10. (Extra credit, two points max, no points for leaving this blank.) Give the running time for your implementation of `most_friends` using Θ notation (or a hypothetical implementation if you did not complete #9). Assume that the friend list contains *F* friend pairs.

11. (Zero points.) Do you feel your performance on this exam will fairly reflect your understanding of the course material so far? If not, explain why. You will not lose any points for your answer.