6.S189 Sample Exam

Introduction

The purpose of this test is to give you an idea of the kind of comfort with programming that we'd expect you to get to very quickly in 6.01. It's definitely okay to look at your notes, Python references, and your homework/tutor solutions to help you finish the exercises on this "exam". Knowing where to look for help is a good skill to have in computer science!

If you do fine on this test you should have no difficulty with the programming aspects of 6.01 on start-up.

Please actually write the programs - and *run and debug* them! The most important skill that a good software engineer can have is the ability to write good tests that find all conceivable bugs in the code. You should work on these problems alone; the purpose of the diagnostic is for you to gauge your ability, not the ability of a team. If you need help, feel free to visit us in lab or ask on Piazza.

Programming Problems

For the following problems, write the stated functions. For those that take input, provide at least 4 comprehensive test cases. Cover every edge condition you can think of.

1. Warmups
   (a) Write a function that prints out the odd numbers 1 through 99. Bonus: Use list comprehension.
   (b) Write a function that takes in a dictionary and outputs the most frequent value in the dictionary.
   (c) Write a function to reverse a string. The string can be of any length and might be empty. Bonus: Implement recursively.
   (d) Write a function that takes in two sorted lists and merges them (don't use any Python built in functions or methods, such as the list.sort method). So the input ([1, 5, 8], [4, 7, 10, 12]) should yield the output [1, 4, 5, 7, 8, 10, 12]. The lists may not be of the same length, and one or both may be empty.
2. A clerk works in a store where the cost of each item is a positive integer number of dollars. So, for example, something might cost $21, but nothing costs $9.99. In order to make change a clerk has an unbounded number of bills in each of the following denominations: $1, $2, $5, $10, and $20. Write a procedure that takes two arguments, the cost of an item and the amount paid, and prints how to make change using the smallest possible number of bills.
3. (a) Write a procedure that takes a string of words separated by spaces (assume no punctuation or capitalization), together with a "target" word, and shows the position of the target word in the string of words. For example, if the string is:

```
we dont need no education we dont need no thought control no we dont
```

and the target is the word "dont" then your procedure should return the list 1, 6, 13 because "dont" appears at the 1st, 6th, and 13th position in the string. (We start counting positions of words in the string from 0.) Your procedure should return False if the target word doesn't appear in the string.

(b) Suppose you are repeatedly looking up targets in a fixed long list. It might help to build an "inverted index", that shows the positions of all targets, so that this information can be retrieved quickly. For example, an inverted index for the above string of words would be:

```
we: 0, 5, 12
dont: 1, 6, 13
need: 2, 7
no: 3, 8, 11
education: 4
thought: 9
control: 10
```

Use an appropriate data structure to represent an inverted index. Write a procedure that, given a string of words, constructs an inverted index, and show how to use the index to look up target words as in part (a).

4. One classic method for composing secret messages is called a square code. The spaces and punctuation are removed from the english text and the characters are written into a square (or rectangle). For example, the sentence "If man was meant to stay on the ground, God would have given us roots" is 54 letters long, so it is written into a rectangle with 7 rows and 8 columns (the number of rows is $\lfloor\sqrt{54}\rfloor$, and the number of columns is $\lceil\frac{54}{\texttt{numrows}}\rceil$):

```
ifmanwas
meanttos
tayonthe
groundgo
dwouldha
vegivenu
sroots
```

The coded message is obtained by reading down the columns going left to right. For example, the message above is coded as:

```
imtgdvs fearwer mayoogo anouuio ntnnlvt wttddes aohghn  sseoau
```

In your program, have the user enter a message in English (they might enter capital letters, spaces, and punctuation - you will need to remove these!). Have the maximum message length be 81 letters, and print an error message if they exceed this length.

Display the encoded message. (Watch out that no "garbage" characters are printed.) Here are some more examples:

```
Input                     Output
Have a nice day!          hae and via ecy
Feed the dog.             fto ehg ee  dd
CHILL OUT!!!!             cl ho iu lt
```