Midterm #2 Review

CS 140 • Dr. Sam Vandervelde

On Thursday, November 15 we will hold our second midterm exam in CS 140 during our normal class time, from 2:20 to 3:50. I will write the exam to take around 70 minutes, so that you will (hopefully) have time to review your work. The midterm will consist of ten "predict the output" questions worth 3 points each, then two "spot the bug" questions at 4 points each, followed by six programs worth 7 points each, for a total of 80 points.

You should review your quizzes and the programming tasks below thoroughly, since over half of the midterm will be based directly on these questions, meaning that I will make only a minor variation (if any) to them. The remaining questions will be new, although they will only involve material presented during the semester, of course. I also recommend reviewing the code drills, homework programs, and class creations for further practice. The exam will emphasize topics covered since the first midterm, although by the nature of our subject it will necessarily be cumulative to a large extent.

The best way to review past programming assignments is to write the program from scratch. If necessary you can read the solution code for ideas, but then set the solution aside and return to your program to complete it without referring to the solution. Don't just read or memorize code; this will not help you overall!

Twelve Programming Tasks

1. *Pig Latin* Define a function called `piglatin` that removes the first letter from a word, pastes it onto the end of the word, then tacks on `ay` after that. The function should then return the result. Thus typing `print piglatin("birdbath")` should result in the output `irdbathbay`.

2. *Missing Numbers* Create a list called `randnums` consisting of one hundred numbers, each of which is a random number from 1 to 100. You might expect to obtain each number once, or at least close to it. Check whether this is the case by counting how many numbers from 1 to 100 do not appear in the list and announcing this quantity.

3. *Word Box* Write a program that asks the user to enter any word. The program should then create a word box such as the one shown to the right for the input `sunshine` in which the word makes up the sides of the box, with asterisks in the corners. Use a `for letter in word` type loop to create the vertical sides.

```
*sunshine*
s        s
u        u
n        n
s        s
h        h
i        i
n        n
e        e
*sunshine*
```

4. *Txtfy Wrds* Write a program which asks the user to enter a phrase. Then create a new phrase in which all the vowels have been removed, and give the "textified" version of the sentence. Thus if the user enters `Four score and seven years ago` then the program should respond with `Your textified phrase is Fr scr nd svn yrs g`

5. *Making a List* Write a program that inputs a positive integer called `number`. It should then repeatedly change the value of `number` by either dividing by two (if it is even) or tripling it and adding one (if it is odd). It should keep track of all the values along the way in a list called `values`, until `number` is finally equal to 1. So if the user enters 17 then at the end `print values` should give `[17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1]`.

6. *And Checking it Twice* Write a program that asks the user to enter any word. It should remove extra spaces, convert all letters to lower case, and then check whether or not there are two vowels in a row within the resulting string. So if the user types `inoculate` the program might respond `The word inoculate does not contain consecutive vowels.`

7. *Change of Heart* Suppose that `records` is an array in which each list within the array consists of a first name, last name, class year, color, and favorite number, in that order. Write a program that asks the user to enter a first name, then searches through the array to find a record matching that name. It should then mention this person's current favorite number, ask for a new number, and replace the old value with the new one.

8. *Randomizing a List* Define a function called `randomize` that takes a list of items and returns a list containing the same items, just listed in a random order. For instance, typing `print randomize([1, 2, 3, 4, 5, 6])` might yield `[5, 1, 3, 4, 6, 2]`.

9. *Head of the Class* Let `roster` be a list of names of the students in our class. Write code that asks the user to enter a name, then checks whether that name appears in the list and if so swaps the entered name with the first name in the list. If the name doesn't appear, then the program should simply point this out to the user.

10. *Parsing a String* Define a function called `parse` that takes a single long string, splits the string into words at the spaces, and returns a list of individual words. For instance, typing `print parse("have a nice day")` would give `["have", "a", "nice", "day"]`.

11. *Finding the Average* Suppose that `scores` is a list of quiz grades. Define a function `findaverage(scores)` that adds up all the numbers in this list, then divides by the total number of items in the list. The function should then announce the class average using a complete sentence, and finally return the average score.

12. *Character Counts* Write a function `count` that takes a string (consisting of lower case letters and spaces) as input and returns a list of three numbers: the first counts the number of vowels appearing in the string, the second counts the consonants, and the third counts the spaces. Thus `print count("have a nice thanksgiving")` should give `[8, 13, 3]`.

TIP: in several of these programs it will come in handy to define `vowels = "aeiou"` and use the `if letter in vowels:` construction.