CS 111: Practice Problems

Summer 2018

1 Miscellaneous

1. Write code to swap two values:

```
int x = 5;
int y = 10;

// your code here

// now x should be 10 and y should be 5
```

   (Note that "`x = 10; y = 5;`" works for these particular values, but is not a correct swap implementation in general.)

2. We saw that floating point types like `double` can have small inaccuracies. Write a `doubleEquals` function that determines whether two doubles are within a certain tolerance of each other (generally `1e-6` (that is, 0.000001) is used).

3. Write a program that simulates rolling two 6-sided dice and prints the resulting total. (Note that this is not the same as creating a random number from 2 to 12, since rolling a 2 is much less likely than rolling a 7.)

2 Conditional statements

1. Write a function that takes three integer arguments and determines their maximum.

2. Implement a game of rock-paper-scissors. Ask for two players' choices and print out who wins. Remember the difference between `==` and `.equals()` for strings. Does your program still work if the user enters "ROCK" / "Rock" / "roCK" instead of "rock"?

3. Write a schedule program that tells the user what they should be doing right now. Ask the user for the current time in 24-hour format and output their scheduled task according to this schedule:

0:00 – 8:29	sleep
8:30 – 8:59	breakfast
9:00 – 17:29	work on CS 111
17:30 – 18:29	dinner
18:30 – 23:59	catch Pokemon

3 Loops

1. Given a list of numbers, compute their average. First ask the user how many numbers will be in the list. Then input each one and at the end, output the average of the list. (This could be solved with or without arrays – can you do both?)

2. Given a list of numbers, compute their standard deviation:

$$\sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$$

   where N is the number of elements, μ is the average of the list, and x_i is each element in turn. First ask the user how many numbers will be in the list. Then input each one and at the end, output the standard deviation of the list.

3. Ask the user for a non-negative integer. Print the binary representation of that number. For example, if the user inputs "13", your program should print "1101".

4. Ask the user for a binary number. Print the decimal (base 10) representation of that number. For example, if the user inputs "1101", your program should print "13".

5. Write a function `prime` to determine if a given number is prime.

6. Write a function `factors` to return a list of numbers that evenly divide the given number, including 1, but not including the number itself. For example, for 14, your function should return 1, 2, and 7.

7. A *perfect number* is one whose factors add up to be the original number. For example, 6 is perfect because its factors are 1, 2, and 3, which sum to 6. But 12 is not, since its factors, 1, 2, 3, 4, and 6, sum to 16. Write a function `isPerfect` to check if a given number is perfect. (hint: use your solution to the previous problem)

8. All known perfect numbers are even, but it's unknown whether any odd perfect numbers exist. Write a program to find all perfect numbers from 1 to 10,000. Are any of them odd? (hint: probably not)

9. (Collatz conjecture) Given a positive integer n, we can repeatedly apply the following rules:

   - If n is even, divide it by 2
   - If n is odd, multiply it by 3, then add 1.
   - If n is 1, stop.

For example, given input 5, we would generate the following sequence: $5 \to 16 \to 8 \to 4 \to 2 \to 1$.

It's currently unknown whether the sequence always reaches one. Write a program that will check the numbers from 1 to 100,000 to see if they all reach one.

4 Strings

1. Write a function to reverse a string. For example, given input "cs111", your function should return "111sc".

2. Programmers sometimes name variables with underscores between each word (`number_of_students`) and sometimes using camel case (`numberOfStudents`). Write a function that converts a string from the underscore format to camel case. Write another function to convert in the other direction.

3. Write a function to convert a person's name from "First Last" format to "Last, First". Assume that the first and last names are each a single word with no spaces (note that this is not true in general – names are actually really tricky. See https://www.kalzumeus.com/2010/06/17/falsehoods-programmers-believe-about-names/).

4. Write a program that will act as a simple calculator. The input will be a string of the form "x ⟨op⟩ y", where x and y are integers and ⟨op⟩ can be `+`, `-`, `*`, or `/`. For example:

```
2 + 2
35 - 10
2 * 7
12 / 3
```

   Assume that there will always be exactly one space before and after the operator, and no other spaces. For division, as the designer of this program, you can decide whether to use integer or double division. Which do you think your users would expect?

5. Modify the previous program to support inputs with extra spaces (or a lack of them). For example (with quotes to clarify where spaces are, but the quotes are not part of the string itself):

```
"      2+   2"
"35-10"
"2 *7"
"  12  /  3  "
```

6. Write a program that reads two binary numbers as string from the user and adds them together. For example, on inputs "1101" and "11", your program should print "10000". (You could solve this by converting to decimal and back again, or by doing everything in binary – can you do both?)

5 Arrays

1. Write a function to append one array to another. For example, given inputs [1, 2, 3] and [4, 5], it should return the array [1, 2, 3, 4, 5].

2. Write a function to merge two sorted arrays into a single sorted array. For example, given inputs [1, 5, 10] and [2, 7], it should return the array [1, 2, 5, 7, 10].

3. Write a function to determine whether all elements in an array are the same.

4. Write a function to determine whether any duplicate elements exist in an array.

5. Modify the previous function to return a list of the indices where these duplicate elements exist. For example, for the array [3, 5, 3, 3, 7], your function should return [0, 2, 3]. If there are no duplicates, you should return the empty array [ ].

6. Modify the previous function further to support arrays with multiple duplicate elements and return a list of lists of indices. For example, for the array [3, 5, 3, 3, 7, 4, 5], your function should return [[0, 2, 3], [1, 6]]. If there are no duplicates, you should return the empty array [ ].

7. Write a function to remove all negative values from an array. For example, for the array [2, -5, 7, -3, -2, 8], it should return [2, 7, 8].

8. (Monte Carlo sampling) Run your two dice rolling function (Miscellaneous #3) 1,000,000 times and keep track of the results of each roll. Compute the probability of rolling each of the values from 2 to 12.

6 Objects

1. Implement a game of Tic-Tac-Toe. Write a `Board` class and a `Player` class. Instantiate two `Player` objects and ask each one for their move, until the game is over. Print the new board out after each move.