CSCI 1100 — Computer Science 1 Homework 2
Strings and Functions

Overview

This homework is worth **75 points** total toward your overall homework grade (each part is 25 points), and is due Thursday, September 22, 2016 at 11:59:59 pm. There are three parts to the homework, each to be submitted separately. All parts should be submitted by the deadline or your program will be considered late.

Note on grading. Make sure you read the comments from Homework # 1. They apply to this and all future homeworks and will be of increasing importance. In all parts of the homework, we will specify which functions you must provide. Make sure you write these functions, even if they can be very simple. Otherwise, you will lose points. We will write more complex functions as the semester goes on. In addition, we will also look at program structure (see Lecture 4), and at the names of variables and functions in grading this homework. Overall, autograding will account for 60 out of the 75 available points (20 points/part). The remaining 15 points (5 points / part) will be assigned by your TA based on program structure and understandability.

The homework submission server URL is below for your convenience:

```
https://submit.cs.rpi.edu/index.php?course=csci1100
```

Part 1: Geometry and breathing!

This problem is a warm up of the use of functions. You will be asked to write a few simple functions.

A number of countries and companies are planning manned trips to Mars in the next few decades. Let's think ahead to one of the more important considerations for the trip – breathable air. Now, the space station gets most of its oxygen from the electrolysis of water and I expect that a trip to Mars will use a similar method, but what if we needed to carry all the oxygen needed along with the astronauts? How much oxygen would we need to carry?

We will make some simplifying assumptions: the capsule is a sphere, 41% of the oxygen in the capsule is used each day, and oxygen is stored in a set of cylindrical tanks (essentially SCUBA tanks). These tanks are pressurized to 3000 psi, which means that each reservoir tank holds 210 times its internal volume of gas. Your problem is to determine the total volume of oxygen needed for the trip, the amount of oxygen per tank, and therefore the number of oxygen tanks.

Your program will ask the user for the dimensions of the spherical capsule (`radius`) and the dimensions of the reservoir cylinder, again (`radius, height`), reading these through `input`. The program will then produce the results as follows

- Calculate and output the total amount of oxygen needed during the journey to Mars. This is the volume in the spherical capsule times the amount of oxygen in the air (21%) times the percent of the oxygen used each day (41%) times 300 days for the one way trip.

  Remember the volume of a sphere is given by `pi* radius**3 * 4/3`. You must write a function `volume_sphere(radius)` to compute this and use it in your computation. **You must use the math module for the value of pi**.

- Calculate and output the amount of oxygen held in each cylindrical tank with the given dimensions `radius, height`. Remember, the volume of a cylinder is given by `pi* radius**2 * height`.

Write a function `volume_cylinder(radius, height)` that finds and returns the volume of a cylinder with the given measurements. (Here you are welcome to make use of the code from the lecture notes!) Output the amount of oxygen the cylinder holds at 3000 psi (210 times the volume of the cylindrical tank). **You must use the math module for the value of pi**.

- Calculate and output the number of oxygen tanks the capsule will need to carry with it on the journey. Always choose the next largest whole number, so 0.2 tanks becomes 1 tank, while 1 tank remains 1 tank. Look at the `ceil` function in the `math` module to help you do this.

As a final note, assume that all of the oxygen needed is contained in the reservoirs. In other words, you do not need to account for the oxygen already in the capsule at the start of the journey, nor do you need to be concerned with the amount of oxygen left in the tank at the end.

Here is example output of your program (how it will look on Wing IDE):

```
Radius of capsule (m) ==> 2.5
2.5
Radius of oxygen reservoir (m) ==> 0.093
0.093
Height of oxygen reservoir (m) ==> 0.633
0.633

Oxygen needed for the trip is 1690.570m^3.
Each cylinder holds 3.612m^3 at 3000 psi.
The trip will require 469 reservoir tanks.
```

Remember, you will need to use formating strings to generate the output in exactly the form we have shown it above. This is necessary to match our output.

When you have tested your code, please submit it as Part 1 of HW 2.

Part 2: A simple trick (numerical functions)!

The original of this simple parlor trick is attributed to Albert Einstein, but it probably dates to an earlier date! We've modified it a little to test a few more math functions and give a little more challenge.

The trick: First, write the number 1089 on a piece of paper, fold it, and put it away. Next, ask your friend to write down a **five-digit** number, emphasizing that the first and third digits must differ by at least two. Don't watch your friend doing the arithmetic. After your friend has written down the five-digit number, ask them to reverse it, take the first three digits of the original number and the last three digits of the reversed number, then subtract the smaller from the larger.

Example: 32156 reversed is 65123
321 - 123 = 198.
Tell your friend to reverse the new number.
Example: 198 becomes 891.
Next ask your friend to add the new number and its reverse together.
Example: 198 + 891 = 1089.

If all goes as planned, your friend will be amazed. The number you wrote down at the start –1089– will always be the same as the end result of this mathematical trick.

Your job is to write a program that mimics this parlor trick by first predicting the outcome, then requesting a five-digit number from the user, applying the steps above, outputing the steps as it goes, and checking the result to make sure it is indeed 1089.

When looking at a problem like this, it is important to try to think about the central issue — the most challenging question — and try to solve it first and check your solution. Here, the most difficult question for us is reversing the digits in an integer. There are a number of ways to do this, but we want you to think about it by making use of integer division and integer remainders (the `//` and `%` operators). Here is code that will help you get started and help you understand as well how to grab the first three digits:

```
>>> x = 14926
>>> x // 100          # Gets the first three digits
149
>>> x % 100          # Gets the last two digits
26
>>> y = 75           # We are going to reverse a two digit number
>>> tens = y//10      # Gets the tens digit
>>> tens
7
>>> ones = y%10      # Gets the ones digit
>>> ones
5
>>> ones*10 + tens   # Gives us the reversed number
57
```

Start your programming effort by writing and testing a function called `reverse3` that calculates and returns the reverse of a three digit number. Test this function carefully. For example

```
>>> reverse3(123)
321
```

Next, write and test a function called `reverse5` that calculates and returns the reverse of a five digit number. Test this function carefully. For example,

```
>>> reverse5(26417)
71462
```

With these two functions done, write the complete program to mimic the parlor trick. The program should the produce output given below by asking the user for a five-digit integer, and then carrying out the computation outlined above, using the two reverse functions as needed. The program should output the steps of the computation. Then, add a print statement at the end of your program to say *"You see, I told you"*.

Your output must match the following (this is how it will look on Wing):

```
Enter a 5 digit number whose first and third digits must differ by at least 2.
The answer will be 1089, if your number is valid
Enter a value ==> 29467
29467
```

```
Here is the computation:
29467 reversed is 76492
492 - 294 = 198
198 + 891 = 1089
You see, I told you.
```

When you have tested your code, please submit it as Part 2 of HW 2.

Part 3: Find the hidden message! (string functions and an if)

Write a program that either encrypts or decrypts a string, depending on the choice that the user makes. First, the program should ask the user whether they want to encrypt ('E') or decrypt ('D'), then it should ask the user for a sentence using (`input`) written in a cipher to decrypt, or in clear text to either encrypt (if 'E' was input) or decrypt. The program should then carry out the encrypt or decrypt operation using the rules described below and print the resulting sentence. For both encrypt and decrypt, the program should print the difference in length between the cipher and clear text versions. The difference should always be printed as a positive number. The 'E' for encrypt, or 'D' for decrypt should be case insensitive, which means that 'e' means encrypt, just like 'E' does. If something other than 'E' or 'D' is entered in respone to the first query, just print *"I didn't understand ... exiting"* and exit the program. Processing the input will require an `if/elif/else` block. We will cover `if/elif/else` on Monday. Until then, you can write the `encrypt` and `decrypt` functions and apply both to any string that is given. **Do not let the fact that we have not covered if keep you from getting a jump on this homework.** All of parts 1 and 2 can be completed as well as most of part 3.

Here is a series of examples running of the program (what you will see on Wing):

Run 1:

```
Enter 'E' for encrypt or 'D' for decrypt ==> d
d
Enter cipher text ==> wh*%$ s7654 s2(*2(ri7654@@@s
wh*%$ s7654 s2(*2(ri7654@@@s

Deciphered as ==> why so serious
Difference in length ==> 14
```

Run 2:

```
Enter 'E' for encrypt or 'D' for decrypt ==> e
e
Enter regular text ==> back off man I am a scientist
back off man I am a scientist

Encrypted as ==> back 7654ff m-? I%4%m%4% sci2(*2(ntist
Difference in length ==> 9
```

Run 3:

```
Enter 'E' for encrypt or 'D' for decrypt ==> D
D
```

```
Enter cipher text ==> wh*%$ s7654 s2(*2(ri7654@@@s
wh*%$ s7654 s2(*2(ri7654@@@s

Deciphered as ==> why so serious
Difference in length ==> 14
```

Run 4:

```
Enter 'E' for encrypt or 'D' for decrypt ==> E
E
Enter regular text ==> back off man I am a scientist
back off man I am a scientist

Encrypted as ==> back 7654ff m-? I%4%m%4% sci2(*2(ntist
Difference in length ==> 9
```

Run 5:

```
Enter 'E' for encrypt or 'D' for decrypt ==> a
a
I didn't understand ... exiting
```

The encryption rules are based on a set of string replacements, they should be applied in this order exactly:

`' a' => '%4%'`	replace any `a` after a space with `%4%`.
`'he' => '7!'`	replace all occurrences of string `he` with `7!`
`'e' => '9(*9('`	replace any remaining `e` with `9(*9(`
`'y' => '*%$'`	replace all occurrences of string `y` with `*%$`
`'u' => '@@@'`	replace all occurrences of string `u` with `@@@`
`'an' => '-?'`	replace all occurrences of string `an` with `-?`
`'th' => '!@+3'`	replace all occurrences of string `th` with `!@+3`
`'o' => '7654'`	replace all occurrences of string `o` with `7654`
`'9' => '2'`	replace all occurrences of string `9` with `2`

For example the cipher for `methane` is `m2(*2(!@3-?2(*2(+`. Here is how we get this:

```
>>> 'methane'.replace('e','9(*9(')
'm9(*9(than9(*9('
>>> 'm9(*9(than9(*9('.replace('an','-?')
'm9(*9(th-?9(*9('
>>> 'm9(*9(th-?9(*9('.replace('th','!@+3')
'm9(*9(!@+3-?9(*9('
'm9(*9(!@+3-?9(*9('.replace('9', '2')
'm2(*2(!@+3-?2(*2('
```

Decrypting will involve using the rules in reverse order.

Your program must use two functions:

- Write one function `encrypt(word)` that takes as an argument a string in plain English, and returns a ciphered version of it as a string.

- Write a second function `decrypt(word)` that does the reverse: takes a string in cipher and returns the plain English version of it.

Both functions will be very similar in structure, but they will use the string replacement rules in different order. You can now test whether your functions are correct by first encrypting a string, and then decrypting. The result should be identical to the original string (assuming the replacement rules are not ambiguous).

Use these functions to implement the above program. When you have tested your code, please submit it as Part 2 of HW 2.

Finished and still want extra challenges to sharpen your programming skills? These are not extra credit but for you to do additional practice.

- Revise part 2 so that if the person enters a number that is not valid (i.e. first and third digit do not differ by two), you immediately print a statement informing the user and do not compute anything. Otherwise, you carry out the above computation.
- Revise part 3 so that it takes an input sentence, encrypts it and then decrypts it, and then checks whether the result is the same. It should print "same" if they are the same and "different" otherwise.