COSC 1336 Lab: 7

Relevant reading: Chapter 6

Due: Oct. 31, 2:30 pm

(Late date: Nov. 6, 2:30 pm)

50 Points

Problem 1. [**8 points**] Download the file `color_mixer_validation_func.py`. In it, you will find a version of the color mixer program we have looked at several times on past labs. Right now the `main` function does no input validation on the user's input. Make the following changes:

a. [**5 points**] Write a function called `read_color`, which prompts the user to enter a primary color and then displays an error message and another prompt until the user inputs a valid primary color. This function should return the valid color that is finally entered by the user. It should also use the `is_blue`, `is_red`, and `is_yellow` functions.

b. [**3 points**] When you have written this function, modify the `main` function to use it.

Problem 2. [**22 points**] **Prime factorization:** Every integer can be uniquely written as the product of its prime factors raised to some power. This is known as the number's prime factorization. For example, $49 = 7^2$, and $756 = 2^2 \cdot 3^3 \cdot 7$. In this problem, you will write a program that displays the prime factorization of a number input by the user. Create a file called `prime_factorization.py` and then follow these steps:

a. [**4 points**] Write a function called `is_prime`, which receives a non-negative integer and returns a Boolean indicating whether or not the integer is prime. Recall that you wrote a similar function for problem 2 of lab 5, based on a provided design, but that function displayed the answer rather than returning a value. Feel free to modify the previous solution, but you are also welcome to write this function from scratch; it does not have to match the previously provided design, so long as it is correct. Test and debug this function **thoroughly** before continuing. If you get stuck, get help so that you are able to do the rest of the problem.

b. [**8 points**] Now write a function called `prime_factor_power` which receives a positive integer `num` and a prime number `p` and returns the number of times that `p` divides evenly into `num`. Again, thoroughly test and debug this function before moving on, and get help if you get stuck.

c. [**6 points**] Write a function called `display_factorization` that receives a positive integer `num` and displays the prime factorization of the number. Here is a pseudocode design:

```
for p from 2 to num-1:
    if p is prime:
        set e to be the number of times that p divides evenly into num
        if e is not zero:
           display p^e
end for
```

Again, test and debug this function throughly. For example, the function call `display_factorization(756)` should result in the output

```
2^2 3^3 7^1
```

Notice that this function should display the result on the screen rather than returning the result.

d. [**4 points**] Lastly, write a `main` that has a sentinel loop that allows the user to enter positive integers and displays the prime factorization. An input of zero or less should terminate the program.

Problem 3. [20 points] Slot machine simulation: In a file called `one_armed_bandit.py`, create a slot machine simulation. A slot machine is a gambling device that the user inserts money into and then pulls a lever (or presses a button). The slot machine then displays a set of random images. If two or more of the images match, the user wins an amount of money, which the slot machine dispenses back to the user. Below is a description of the way the program should behave when it is run, followed by a description of the functions you need to write in your implementation. I encourage you to write, test, and debug each function in the order they are described.

The program should first ask the user to enter their bet (that is, how much money they would be entering into the machine). Then the program should display the three randomly generated "images" (just text, really, unless you decide to get crazy with the graphics library). Next the program will tell the user how much, if anything, the user has won on that lever pull. The program will ask the user whether she wants to play again (and repeat these steps if so). If the user is finished, the program should display the total amount of money lost or won, and the total percentange payout. That is, if the user bet a total of $20 and won $17 over the course of playing the game, they should be informed that they lost $3, and the total percentage payout was 85% (17/20).

Here are the functions you should write:

a. **[4 points]** `getBet`: takes no parameters, and prompts the user to enter the amount they want to insert, and returns the entered value. The function should use an input validation loop to force the bet to be positive.

b. **[5 points]** `getRandomSymbol`: takes no parameters, chooses a random string from among the possible symbols, and returns it. The possibilities are `Cherries, Oranges, Plums, Bells, Melons,` and `Bars`. Remember that you will need to put `import random` at the top of your file to get access to the `random.randrange` function.

c. **[6 points]** `pullLever`: takes a parameter indicating the amount inserted into the machine, and returns the amount won. If all three symbols match, the user wins 20 times the input bet. If only two symbols match, the user wins twice the input bet. If no symbols match, the user wins nothing.

   This function should use the `getRandomSymbol` function to get the three symbols, display them, and then do the calculation required to determine how much (if anthing) the user won. This value should be *returned*, not displayed on the screen.

d. **[5 points]** Write the `main` function to implement the entire simulation as described above. The `main` function should call the `getBet` and `pullLever` functions, as well as asking the user whether he wants to continue and keeping track of the total amount bet and won by the user. When the user is finished playing, it should display the results as described.

For this lab, you should submit the following files:

- `color_mixer_validation_func.py`
- `prime_factorization.py`
- `one_armed_bandit.py`