CSCI 1133
Exercise Set 3

You should attempt as many of these problems as you can. Practice is *essential* to learning and reinforcing computational problem solving skills. We encourage you to do these without any outside assistance. Create a directory named `exercise3` and save each of your problem solutions in this directory. Name your individual Python source files using "`ex3`" followed by the letter of the exercise, e.g., "`ex3a.py`", "`ex3b.py`", "`ex3c.py`", etc.

Note that we automatic testing of your solutions is performed using GitHub, so it is necessary to name your source files precisely as shown and push them to your repository.

Recall from Lab3 that you need to include the following statement in the global namespace in order to automatically have the `main()` function executed when the module is loaded and executed in script mode:

```
if __name__ == '__main__':
    main()
```

A. **Compound Interest**
Write a well-structured Python program that will compute the number of years it takes to reach a specified target amount given an interest rate that is compounded annually. Your program should prompt the user and input the following:

- A starting amount of money in dollars.cents (float)
- A target amount of money in dollars.cents (float)
- An annual interest rate expressed as a percentage (float)

Use a while loop to determine the number of *full* years it takes until the initial amount of money equals or exceeds the target amount at the given annual interest rate. Your program must use iteration. Do not use compound interest formulae.

Constraints:

- Do not import or use any module functions
- Use a while loop and an iterative method, do not solve the problem analytically

Examples:

```
$ python3 ex3a.py
enter starting amount: 1000.00
enter target amount: 10000.00
enter interest rate: .02
117 years

$ python3 ex3a.py
enter starting amount: 1000.00
enter target amount: 10000.00
enter interest rate: .05
48 years
```

B. **Perfect Numbers**

An integer is said to be a *perfect number* if the sum of its divisors, including 1 but not the number itself, is equal to the number. For example, 6 is a perfect number because 6 = 1 + 2 + 3. Write a well-structured Python program that will output all the perfect numbers in a given closed interval. Your program should prompt the user and input the following:

- The lower limit of the closed interval (positive integer)
- The upper limit of the closed interval (positive integer)

Your program should include a pure function named `perfect(n)` that determines whether any particular number is a perfect number. The function should have an integer as its only argument and return the *Boolean* value `True` if the number is perfect, `False` otherwise.

(HINT: finding all of the integer divisors requires a loop that checks every value starting from 1 in a "brute force" fashion. Simply accumulate those that are found to be divisors of the target integer)

Thoroughly test your function to make sure it works!

Examples:

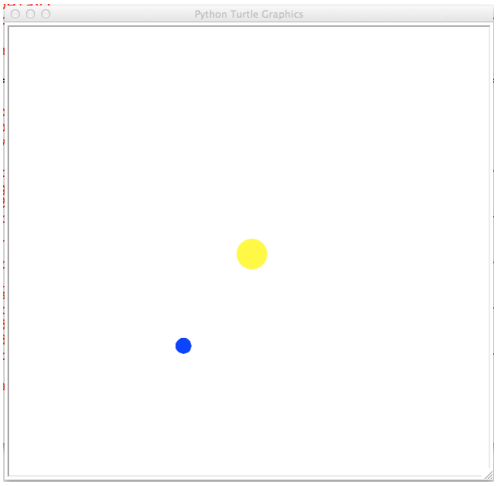
```
$ python3 ex3b.py
enter lower bound: 1
enter upper bound: 100
6
28
```

C. **Orbiting Turtles**

Write a well-structured Python program that will use turtle graphics to create an animated simulation of a ball rotating around an object. Your program should do the following:

- Draw a dot at the center of the turtle display with radius 40 in the color of your choosing.
- Change the shape of the turtle to a circle.
- Using a `while` loop, move the turtle *one degree* at a time in a circular "orbit" around the origin at a radius of 150 for 3 complete revolutions.
- You must use the turtle `.goto()` command, do not use the `.circle()` command

HINT: use the `sin` and `cos` functions from the math library to determine the turtle's next position in the orbit.



D. **More Orbiting Turtles**

Rewrite the orbiting turtle program to simulate *two* orbiting objects, one orbiting around another (i.e., a moon orbiting a planet and a planet orbiting a sun). Your program should have the "moon" make one complete orbit for each advance of the "planet" in its orbit. To speed things up a bit, advance the moon 5 degrees per iteration and have the planet advance 2 degrees per iteration.

Your program should do the following:

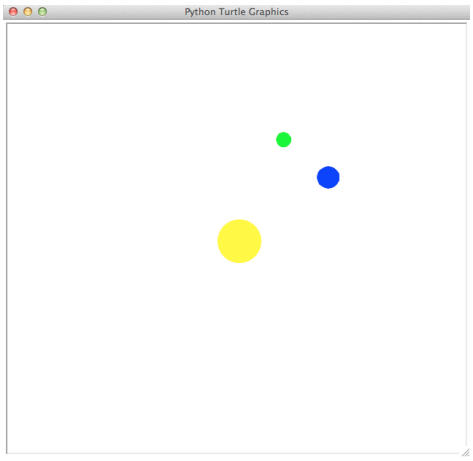
- Use a moon 'radius' (distance from center of planet) of 80 pixels
- Use a planet 'radius' (distance from center of sun) of 150 pixels

Constraints:

- You must *nested* `while` loops to solve this problem, do not use `for` loops
- Use the `.goto` turtle method to move the 'moon' and 'planet' objects

HINT: you will need to construct 2 separate turtle objects, one for the 'planet' and one for the 'moon'

Example Screen Shot:



E. **Greatest Common Divisor**

The *Greatest Common Divisor*, GCD(*a,b*), of two integers *a* and *b* not both of which are zero, is the largest *positive* integer that evenly divides both *a* and *b*. The Greek philosopher and mathematician Euclid devised an incredibly clever and efficient algorithm to determine the GCD of two integers using a minimum number of steps.

The Euclidean algorithm for finding the Greatest Common Divisor of *a* and *b* is as follows:

If *b*=0, then GCD(*a*,*b*) = *a*.

Otherwise, divide *a* by *b* to obtain an integer quotient *q* and remainder *r*. By definition, $a = bq + r$.

Since GCD(*a*,*b*) = GCD(*b*,*r*), simply replace *a* with *b* and *b* with *r* and repeat the procedure.

Note carefully that the remainders are decreasing at each iteration, therefore a remainder of 0 will eventually result. The last *nonzero* remainder is GCD(*a,b*).

Example: GCD(1260,198): *a*=1260, *b*=198

GCD(1260,198)	1260 = 198 x 6 + 72	*b*=198, *r*=7	= GCD(198,72)
GCD(198,72)	198 = 72 x 2 + 54	*b*=72, *r*=54	= GCD(72,54)
GCD(72,54)	72 = 54 x 1 + 18	*b*=54, *r*=18	= GCD(54,18)
GCD(54,18)	54 = 18 x 3 + 0	*b*=18, *r*=0	= GCD(18,0)
GCD(18,0)			= 18 (by rule 1)

Write a well-structured Python program that inputs two integers and then calls a pure function named `gcd(a,b)` to calculate and return the greatest common divisor. Display the two integers and the greatest common divisor on the terminal screen. Include a loop that will continue to solicit input until the user presses the `enter` key without entering anything (the Python `input()` function will return a null string, "", if this is the case).

Note: If either *a* or *b* are negative, use their absolute value. Also note that you need to carefully identify which of the two arguments is the largest.

Examples:

```
$ python3 ex3b.py
enter first value: 1260
enter second value: 198
the GCD of 1260 and 198 is 18

enter first value: 13
enter second value: 26
the GCD of 13 and 26 is 13

enter first value:
```

F. **Craps**

The casino game *Table Craps* is a dice game in which a number of betting options are available depending on the eventual roll of a pair of dice. Players may bet "for" or "against" certain outcomes in a dizzying array of combinations with varying odds. Each round of *Craps* begins with an initial roll of the dice (the so-called 'come-out' roll). The player 'wins' if the initial roll totals 7 or 11, and *'Craps out'* (loses) if the roll is 2, 3 or 12. Any other initial roll (4, 5, 6, 8, 9, 10) becomes the *point* value. The player continues rolling dice until either the *point* is matched or a 7 is rolled (whichever occurs first). If the *point* is matched the player wins, if a 7 is rolled the player loses.

Write a well-structured Python program to simulate the game of *Craps*. Your program must do the following:

- Simulate the initial (come-out) roll by generating two pseudo-random integer values between 1 and 6 (inclusive). Determine if the result is 'win', 'lose' or 'point' and display the result along with the dice-roll "values".
- If a *point* is established, output a message to indicate the value of the *point* (4, 5, 6, 8, 9, 10) and continue to simulate play until either a '7' is rolled or the *point* is matched. Display the result of each roll as you continue. Finally, indicate if the player wins or loses at the end.

Examples:

```
$ python3 ex3f.py
Initial roll is: [6,5] = 11
  You win!

$ python3 ex3f.py
Initial roll is: [5,2] = 7
  You win!

$ python3 ex3f.py
Initial roll is: [1,1] = 2
  Craps!  Sorry, you lose

$ python3 ex3f.py
Initial roll is: [4,4] = 8
  Point is 8. Roll again
  Roll is: [5,4] = 9
  Roll is: [5,1] = 6
  Roll is: [2,1] = 3
  Roll is: [4,2] = 6
  Roll is: [1,6] = 7  Sorry, you lose

$ python3 ex3f.py
Initial roll is: [5,1] = 6
  Point is 6. Roll again
  Roll is: [3,3] = 6  You win!
```