CS2321 Lab 1

Lab Instructions:
Save the code you write for each exercise in this lab as a *library* -- that is, a textfile with a .py extension containing only executable python code (i.e. no angle-bracket prompts, etc). Name each file according to the exercise number (e.g. ex1.py, ex2.py, etc.) and save them to a directory containing the report file, when completed, compress them together in a single zip file to be submitted on D2L.

Each function should have a docstring explaining what the function does.
Any Follow-up Questions and their Answers should be included in a **docstring** in the `main()` function.

e.g. the definition for a function should look like:

```
def fct1():
  '''
      This is the documentation-string explaining what this function does.
      Questions posed by the exercise/lab instructor are also answered here.
  '''
  your function-code here…
  etc...
```

Lab Deliverable: Once all your programs run correctly, collect their code and the results of their test-cases in a nicely-formatted **PDF** file exported from Word Processing document (e.g. Word or LibreOffice are fine) to be included in the submission on D2L.

This report should consist of each lab exercise, clearly **labeled** in order, consisting of code, then copy/pasted output of its four provided test-cases.

Paired Programming:
We will work today's lab assignments in pairs -- assigned by the instructor -- on a single computer in one partner's account. One person will start out as the ***typist***, the other as the ***verifier***. These roles will switch.
For each problem, partners should decide upon their proposed algorithm to solve the given problem *before* the typist begins to type. Sketch it out on a sheet of paper, perhaps.
For ten minute periods, the typist will type the code, while the other verifies and suggests corrections (typist has final decision). Under no circumstances may the verifier ever touch the mouse or keyboard. (Note: the *instructor* may not touch your input devices either!)
On the instructor's ten-minute signal, partners will trade responsibilities. This should allow both partners to benefit from each other's strengths. Future paired-programming labs will be with different partners, to spread the gained experience around.
Each partner should post the resulting code in their own D2L folder. You may transmit partnership-generated code to the other partner (only!) by email or thumb-drive.

Exercises

Using the **turtle** library and interface (from the book, the help command, or the internet), to design the following functions. Include a screen-capture of two different sizes of each in your final Lab Report -- to show that the scale parameter is working. Please crop your screencaptures to only show the Turtle's graphics window.

Ex #1:
Drawing circles

1a) Implement Project #1 on page 240 of your 2nd edition Textbook. Have your function print out the Turtle's traversed distance in the Shell.

Page 240 #1:

> Define a function **drawCircle**. This function should expect a Turtle object, the coordinates of the circle's center point, and the circle's radius as arguments. The function should draw the specified circle. The algorithm should draw the circle's circumference by turning 3 degrees and moving a given distance 120 times. Calculate the distance moved with the formula 2.0 * π * radius / 120.0.

Explain your algorithm (e.g. as pseudocode) that leads the Turtle object to draw a circle according to the author's suggested method.

1b) What Python code would draw the same circle using Turtle's .circle() method? Show me this code, and a screen-capture of the results.

Ex #2:
Define a function to draw a simple house using your turtle object. It should consist of a square body, a triangle roof, a rectangular door (with a round doorknob), and a square window with four window panes. Name this function: **drawHouse(t, blc, scale=1)** The parameter **blc** should be a tuple (x,y) specifying the screen coordinates of the house's bottom left corner.
scale is assumed to be a floating-point value between (0 ... 5] that should be applied to each of your turtle's drawn distances, where 1 means 100%. Changing this value should scale the size of your total drawing up/down (e.g. scale = 0.5 should draw a house half the size of that drawn by your original code). As illustrated in the function signature above, it should be an optional parameter with a default value of one (1).

Ex #3: Define a function **drawXmasTree(t, blc, scale=1)** You may add further parameters beyond the first three if you wish (note: give any additional parameters default values!).
Your tree should resemble three filled, superimposed green triangles (containing colored ball ornaments) over a brown trunk.
blc and **scale** should work as in the preceding exercise.

Ex #4: Define a function **drawSnowman(t, blc, scale=1)** Again, decide on your own parameter-list beyond the first three.
Your snowman should consist of a three-circle body, an orange triangular nose, round black coal for eyes and mouth, a black stovepipe hat, and stick arms with three-fingered hands. The fingers should be drawn at random angles from each other.
As snowmen generally have few corners, let **blc** here represent the tuple at the bottom-most point of the bottom-circle of its body.

Ex #5: Define a function **winterScene(t)**
Again, you may decide on your own parameter-list (if any) beyond the first argument.
Your rendered WinterScene should contain a house, a Christmas tree, and a snowman -- in any (reasonable) arrangement.

Include **two** example screenshots in your lab report (accompanied by annotations/explanations) of differently-scaled results of the functions in Ex#2-5