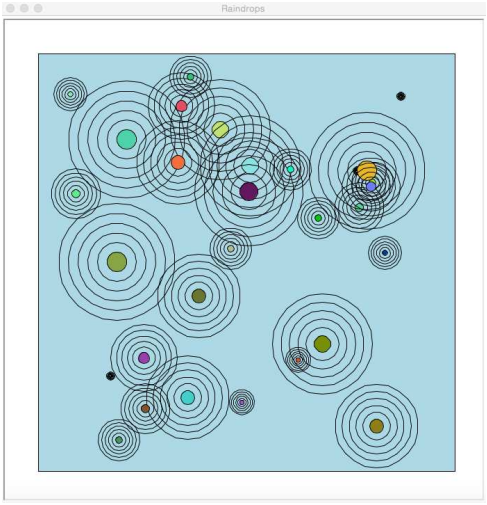
Computer Science I
Raindrops

CSCI-141
Lab 3

01/25/2022

Problem

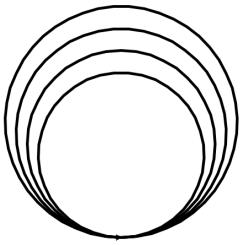
Using Python's turtle graphics module, write a program **raindrops.py** to generate a colorful sequence of raindrops rippling on a smooth pond. Your program must use both tail-recursion and iteration to complete its task. When drawing, the raindrops and ripples must stay within the boundaries of a square representing the borders of the pond.



1 Problem-solving Session (15%)

Work in a group of three or four students as determined by your instructor. Each group will work together to complete the following activities. Note that for problem solving, we'll consider minor variations of the problem you will ultimately implement.

1. Write code for a recursive function, `draw_rec`, that draws circles of increasing radius from a common base point, as illustrated. Assume as a precondition that the turtle starts out facing East, pen down, at the common base point.

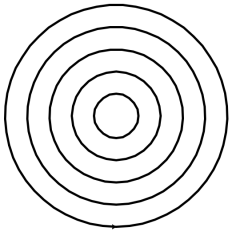


Your function should take at least two parameters: a positive integer that represents the number of circles to draw, and another positive integer that represents the radius of the smallest circle to draw. (You also may decide to use additional parameters,

and you may wish to write auxiliary helper/utility functions.) Each successive circle should have a radius one unit larger than the previously drawn circle.
The recursive function should return the sum of the areas of the circles drawn. You may assume the `math` library has been imported and you can use `math.pi`. Your code may not loop using any mechanism other than recursion.

2. Provide a substitution trace for `draw_rec(3, 3)`. Leave your answer in terms of π.

3. Write code for an iterative function `draw_iter` that draws circles, as illustrated. Assume as a precondition that the turtle starts out facing to the right, pen down, at the center of the circles.



Your function should take two parameters: a positive integer that represents the number of circles to draw, and another positive integer that represents the radius of the first circle as well as the distance between circles.
The iterative function should return the sum of the *circumferences* of the circles drawn instead of the areas. You may assume the `math` library has been imported and you can use `math.pi`. Your function must use a `while` loop instead of recursion.

4. Create a timeline of changes to the parameters and local variables used in `draw_iter(4, 5)`. Leave appropriate answers in terms of π.

5. Suppose for turtle drawing purposes that the 'pond' has legal values $[-p, p]$ in both x and y dimensions.

   You want to draw a raindrop that has radius r, along with k ripples. The first ripple has radius $2r$, and each subsequent ripple has a radius that is larger than the radius of the previous ripple by an additional amount r.

   In terms of p, r, and k, what is the interval of values where the center of the raindrop could be located such that the raindrop and all of its ripples fall inside the dimensions of the pond?

   Hint: the answer is the same for both the x and y dimensions.

Substitution Traces (Review)

Recall from the lecture what a substitution trace looks like. It works by tracing the code as a sequence of substitutions of argument values as recursion progresses. Here's an example, in both non-tail-recursive, and tail-recursive form (but omitting the wrapper function); the function is on the left, and the trace is on the right:

```
def fact(n):                          |    fact(3) = 3 * fact(2)
    if n == 0:                        |            = 3 * (2 * fact(1))
        return 1                      |            = 3 * (2 * (1 * fact(0)))
    else:                             |            = 3 * (2 * (1 * 1))
        return n * fact(n-1)          |            = 3 * (2 * 1)
                                      |            = 3 * 2
                                      |            = 6
                                      |
def fact_accum(n,a)                    |    fact_accum(3,1) = fact_accum(2,3)
    if n == 0:                        |                    = fact_accum(1,6)
        return a                      |                    = factAccum(0,6)
    else:                             |                    = 6
        return fact_accum(n-1,n*a)    |
```

2 In-Lab Session (10%)

During the in-lab time, you are expected to work on the assignment described below. At the end of the time, you need to have either an SLI or instructor check off with you on the progress you have made. Students who demonstrate having put in a reasonable amount of effort will receive full credit for this component. Students who do not demonstrate reasonable effort, leave early, or do not get verification, will not get credit.

Tasks:

1. Read over the entire lab.

2. Create a new project for this lab and open a new file for coding.

3. Near the top of your program create a constant value indicating the size of your 'pond'. The constant should be outside the scope of all functions (i.e. globally visible). Choose a value that will fit well on your display.

4. Write one or more functions to initialize your drawing, choosing a size for the overall window that provides some border beyond the size of your 'pond'. Draw the bounding box 'pond', using a light-colored fill for the interior. Experiment with different fill colors to get one you like.

5. Write a (not recursive) function that draws a single raindrop (no ripples yet), and that returns the value of the circumference of the raindrop drawn. (You will eventually change this so that it returns the total circumferences of all of the ripples only, *excluding* the original raindrop.) For now, you can hardcode the raindrop radius to 10. Use the `random` module to generate a random fill color for the raindrop, and to generate a random location for the raindrop. Make sure it will fit in the 'pond'. (You will eventually update this to make sure the raindrop *and* all of its ripples fit in the 'pond'.)

6. Write a recursive function that draws n raindrops, and returns the total circumference of the raindrops. (This is a very short function - it uses the function you wrote in the previous question.)

7. Write a main function that calls your initialization function and then calls your non-tail-recursive drawing function. Have it draw 20 raindrops. Run your program, calling your main function, and confirm that you draw your bounding box 'pond' along with 20 randomly located and colored raindrops. Have your main function print the return value from your non-tail-recursive drawing function. If you set the raindrop radius at 10, the total circumference of 20 raindrops is $400\pi = 1256.6370614359173$.

For the postPSS, zip your write a `raindrops.py` work in progress to a file named **lab03.zip** and upload it to mycourses. You could do this at the end of the in-lab or later before the postPSS Sunday deadline.

3 Implementation (75%)

Each student will *individually implement* and submit their own solution to the problem as a Python program named `raindrops.py`.

Note: some of the requirements will require you to make small modifications to the code your wrote for the in-lab activity. Make sure you read the requirements carefully.

Unlike what you did in class in the previous sections, your final program adds up the circumferences of all the ripples of all raindrops, but excludes the innermost circle in each one, which is considered the raindrop itself rather than a ripple.

Because of the nature of the problem, *teleportation* is allowed in this lab. You may use a function such as `setpos()` to move the turtle to where each next raindrop will be drawn.

Your program must satisfy the following requirements:

- The program should run correctly using Python3.
- Prompt the user for the number of raindrops. If the value is outside of the range $[1, 100]$, output an appropriate message and exit. For example,

```
Raindrops (1-100): 278
Raindrops must be between 1 and 100 inclusive.
```

- Use the default turtle window.
- Draw a square bounding box that is visible inside the borders of your window. Using a fill color for the bounding box is optional.
- Use recursion to control the number of raindrops that are generated.
- Use iteration (a `while` loop) to control the number of ripples generated for each raindrop.
- Each raindrop should be drawn in the following manner:
  - Generate a random positive integer in the range $[1, 20]$ inclusive, that represents the radius of the raindrop to be drawn.
  - Generate a random location for the center of the raindrop. The location must be within the bounding box sufficiently so that the raindrop fits inside the bounding box. (Note that the location does not guarantee that any of the subsequent ripples will also stay inside the bounding box.)
  - Generate a random fill color for the raindrop. This can be accomplished by generating three random numbers that represent the red, green and blue components of the fill color. The line color of your raindrop should always be the default black.
  - Generate a random number of ripples in the range $[3, 8]$ inclusive, to be drawn for a raindrop. Each ripple is drawn as a black circle having the same center as the raindrop. The radius of the first ripple is twice the radius of the raindrop. The distance from one ripple to the next is equal to the radius of the raindrop.

  - For any raindrop, you should only draw those ripples that remain completely inside the bounding box.
- Your recursive function must *return* (not print) the circumference of the raindrops generated. Don't include any ripples in this calculation. Just the raindrops.
- Output (print) the value returned from your recursive function. For good style, it is recommended that you have a `main` function that prompts the user, validates the input, initializes the turtle window, calls the recursive function, and prints the output.
  Here is a transcript of the output after running the program to produce the initial picture in this document.

  ```
  Raindrops (1-100): 30
  The total circumference of all ripples is 29405.30723760046 units.
  ```

  Because of the randomness involved, your answer will almost certainly be different, but it should be a similar value.
- Use the `turtle.done()` function to allow the user to close the window at the conclusion of the program.

3.1 Constants

In the requirements section, you have been provided some constant values as **bounds on the range of a legal value**.

Do not hardcode these as magic numbers in your code! You should instead create a variable outside of all functions, normally located at the top of your file. This improves readability and maintainability of your code. If you were to choose to change one of these constants, you would only need to change one piece of code (the previously mentioned variable), rather than searching through your program for all the places in the code where a hardcoded value was used.

Names for the constant value must be *fully capitalized* as shown.

- Your code prompts the user for the number of raindrops. The valid range for the number of raindrops is between `1` and `100` inclusive. Therefore `MAX_RAINDROPS` should be `100`.

  The corresponding code should be:

  ```
  """Maximum number of raindrops"""
  MAX_RAINDROPS = 100
  ```

- You should have a constant for the maximum radius of a raindrop.
- You should have a constant for the number of ripples.
- You should have a constant for the size of the bounding box used in your window.

- You **do not** need constants (e.g. ZERO or ONE) whose usage is obvious from the context and unlikely to change. You can just hardcode such values.

3.2 Random

There are two functions in the `random` module that will be useful:

- `randint(a,b)`: Returns a random integer in the range `[a,b]`, including the endpoints.

```
>>> random.randint(1,15)
4
```

- `random()`: Returns a random floating point value in the range `[0,1)`. This is useful for randomizing the color of each raindrop (see the Turtle section below for more detail).

```
>>> random.random()
0.2128164618278381
```

3.3 Turtle

The following turtle functions may be useful:

- `fillcolor(red, green, blue)`: By default, the `colormode` for turtle is `1`. This means that turtle expects values for each color parameter to be in the range of `[0,1]`. Think of this as the intensity of each color channel (0 is the lowest, 1 is the highest). Use the `random.random()` function three times to generate a value for each color channel parameter. Alternatively you can use `turtle.colormode(255)` to input color in the ranges of `[0,255]`.
- `setheading(angle)`: Face the turtle a specific direction. The value `0` is East, `90` is North, `180` is West, `270` is South, and other values are in-between.
- `hideturtle()`: Hide the turtle so that it does not interfere with the drawing.
- `reset()`: Reset the window by clearing everything and setting the default state.
- `goto(x,y)` or `setpos(x,y)`: Move the turtle from its current position to the specified position. Note that if the pen is down, the straight line connecting the current and specified positions is drawn. The turtle's orientation is not changed. This *teleportation* is allowed for this lab.
- The code sequence:

```
fillcolor( ... )
begin_fill()
# code to draw a single, closed-curve figure element.
end_fill()
```

is used to draw a single, closed-curve figure element of a specific fill color (the line color can also be specified, but we are leaving the line color as the default black). Note that these are Python 3 specific. You must follow the exact order above to draw a filled figure element, and the figure must define an enclosed area.

4 Grading

Your 75% implementation grade is based on these factors:

- 15%: The program uses recursion to control the number of raindrops drawn.
- 20%: The program uses iteration to control the number of ripples for a raindrop and correctly draws a raindrop and its ripples.
- 10%: The specified value is computed and returned from the recursive functions.
- 5%: The program prompts for and uses the number of raindrops, and displays an error message if the number of raindrops is out of range.
- 5%: The program displays an appropriate output message after successful completion (doesn't have to exactly match the sample output).
- 5%: Usage of randomization as specified.
- 5%: All raindrops and ripples stay inside the bounding box.
- 5%: The program handles constants without using *magic numbers*.
- 5%: The code follows the style guidelines on the course web site, including docstrings for each function.

5 Submission

Zip your file `raindrops.py` into a file called `lab03.zip`. Submit the `lab03.zip` file to the MyCourses dropbox for this assignment. You must use *zip* format and no other.