Recursive Tree Task

```
import turtle

def tree(branchLen,t):
    if branchLen > 5:
        t.forward(branchLen)
        t.right(20)
        tree(branchLen-15,t)
        t.left(40)
        tree(branchLen-15,t)
        t.right(20)
        t.backward(branchLen)

def main():
    t = turtle.Turtle()
    myWin = turtle.Screen()
    t.left(90)
    t.up()
    t.backward(100)
    t.down()
    t.color("green")
    tree(75,t)
    myWin.exitonclick()

main()
```

Modify the recursive tree program using one or all of the following ideas:

- Modify the thickness of the branches so that as the `branchLen` gets smaller, the line gets thinner.
- Modify the colour of the branches so that as the `branchLen` gets very short it is coloured like a leaf.
- Modify the angle used in turning the turtle so that at each branch point the angle is selected at random in some range. For example choose the angle between 15 and 45 degrees. Play around to see what looks good.
- Modify the `branchLen` recursively so that instead of always subtracting the same amount you subtract a random amount in some range.

Hints:

turtle.width(width)

turtle.color("blue")

Recursion Tasks

1. allStar

Given a string recursively create a new string where all the adjacent chars are now separated by a "*".
allStar("hello") → "h*e*l*l*o"
allStar("ab") → "a*b"
allStar("a") → "a"

Identify a way to break a problem up recursively….
Look for a Recursive Case and a Base Case

```
def allstar(string):
     if (??????):
          return ???????
     return (?????????????)
```

2. Triangle

We have triangle made of blocks. The topmost row has 1 block, the next row down has 2 blocks, the next row has 3 blocks, and so on. Compute recursively (no loops or multiplication) the total number of blocks in such a triangle with the given number of rows.
triangle(1) → 1
triangle(2) → 3
triangle(4) → 10

3. Factorial

The factorial function (!) of a number is the sum of all the numbers up to the number multiplied together

e.g f(5)=5*4*3*2*1

4. Elephant Count 1

We have a group of elephants and each elephant has two big ears. We want to compute the total number of ears across all the elephants recursively (without loops or multiplication).
elephants (0) → 0
elephants (1) → 2
elephants (2) → 4

5. Reverse

Write a recursive procedure that takes a string and reverses it

6. Elephant Count 2

We have elephants standing in a line, numbered 1, 2, ... The odd elephants (1, 3, ..) have the normal 2 ears. The even elephants (2, 4, ..) we'll say have 3 ears, because they each have a raised trunk. Recursively return the number of "ears" in the elephants line 1, 2, ... n (without loops or multiplication).

elephants (0) → 0
elephants 2(1) → 2
elephants 2(2) → 5

7. Lucky number 8's

Given a non-negative int n, return the count of the occurrences of 8 as a digit, so for example 818 yields 2. (no loops). Note that mod (%) by 10 yields the rightmost digit (126 % 10 is 6), while divide (/) by 10 removes the rightmost digit (126 / 10 is 12).
Lucky8(818) → 2
Lucky8 (8) → 1
Lucky8 (123) → 0

8. Hailstone Numbers

The hailstone sequence is a sequence of numbers that is generated by
For the positive integer *n*.
If *n* is even, divide it by 2 to get *n* / 2.
If *n* is odd, multiply it by 3 and add 1 to obtain 3*n* + 1.
Repeat the process until n=1

Implement a recursive hailstone sequence generator that prints the sequence of hailstone numbers from an input number