CSC 236 T13: Visualizing Recursion

**This is a paired programming assignment designed as an in-class activity.**

[[1]](#footnote-0)

**Directions for use:**

* To use this form effectively, sign into a Google account.
* Then under “File” choose “Make a Copy” in order to be able to edit.
* Share with both team members, but allow the recorder to do the recording.
* Each yellow box should be filled with an appropriate response..
* Download as *yourteamname-T13.docx* and upload to Moodle

This activity is to be done as a modified paired programming assignment, so one person is the facilitator and the other is the recorder. The facilitator roughly corresponds to the “Navigator” and Recorder to the “Driver” in paired programming. If there is a third team member, use a “Sceptic” who poses questions in order to ensure accuracy. **If you are working with the same partner as in T12, be sure to change roles!**

|  |  |
| --- | --- |
| **Team Roles** | **Member Name** |
| **Facilitator:** | Tradd Schmidt |
| **Recorder:** | Tradd Schmidt |

### Visualizing Recursion

Recall that a recursive function is a function that directly or indirectly calls itself. All good recursive definitions have these key characteristics:

1. One or more recursive expressions.
2. One or more base cases for which no recursion is required.
3. Flow such that every chain of recursion eventually ends up at one of the base cases.

|  |
| --- |
| Run the following code in your favorite Python IDE: [recursive-spiral.py](http://cs.berea.edu/courses/csc236/code/recursion-spiral.py)  In the following code, use comments to identify:   1. Every recursive expression. 2. Every base case. 3. If the function is not recursive, explain why.   Also, write an appropriate docstring for each function. |
| import turtle  def drawSpiral(myTurtle, lineLen):  """  Draws a line and turns right until the length of the line is less than or  equal to 0  :param myTurtle: A turtle  :param lineLen: The length of the line to be drawn  :return: none  """  if lineLen > 0: # Base case is when lineLen <= 0  myTurtle.forward(lineLen)  myTurtle.right(90)  drawSpiral(myTurtle,lineLen-5) # Recursive expression  def main():  """  Creates a turtle and a turtle screen. Then calls the drawSpiral()  function.  :return: none  """  myTurtle = turtle.Turtle() # This is not recursive as main() isn’t called  myWin = turtle.Screen() # anywhere in this function  drawSpiral(myTurtle, 100)  myWin.exitonclick()  main() |
| **Additional explanation:** |
| This particular function would not be difficult to code using iteration rather than recursion. Explain the main idea of how you could use iteration to draw the same spiral. |
| **You would have a for loop that will run 100/5 times. Each time you will draw the line using lineLen and then subtract 5 from lineLen** |

|  |
| --- |
| Run the following code in your favorite Python IDE: [recursion-tree.py](http://cs.berea.edu/courses/csc236/code/recursion-tree.py)  In the following code, use comments to identify:   1. Every recursive expression (if none exists use the box below to to explain whether or not this is properly recursive.) 2. Every base case (if none exists use the box below to explain whether or not this is properly recursive.)   Also, write an appropriate docstring for each function. |
| import turtle  def tree(branchLen,t):  """  Draws a tree with using recursive calls  :param branchLen: The length of a branch to be drawn  :param t: A turtle  :return: none  """  if branchLen > 5: # The base case is when branchLen is <= 5  t.forward(branchLen)  t.right(20)  tree(branchLen-15,t) # This is a recursive expression  t.left(40)  tree(branchLen-15,t) # This is a recursive expression  t.right(20)  t.backward(branchLen)  def main():  """  Sets up the positioning to draw a tree using a turtle and the recursive  function tree()  :return: none  """  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() |
| **Additional explanation:** |
| When a recursive call is made, a copy is made of:   * the code * the local variables along with their initial values * the parameters * the current position in the code   The current position in the code which was saved is used upon return.  Thinking about the current position, any would see coding this tree iteratively as much more difficult than the coding using recursion. Explain why this might be. |
| You would have to save a lot of information each time you would want to iterate. |
| Modify the above recursive tree program as follows:   * Modify the thickness of the branches so that as the branchLen()gets smaller, the line gets thinner. * 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 that looks good.   Place this modified code below. |
| #-------------------------------------------------------------------------------  # Purpose: Recursive tree  #-------------------------------------------------------------------------------  import turtle  import random  def tree(branchLen, t, size, angle):  *"""*  *Draws a tree with using recursive calls*  ***:param*** *branchLen: The length of a branch to be drawn*  ***:param*** *t: A turtle*  ***:return****: none*  *"""*  if branchLen > 5:  t.pensize(size)  t.forward(branchLen)  t.right(angle)  tree(branchLen-random.randint(5, 15), t, size - 2, random.randint(15, 60))  t.left(angle\*2)  tree(branchLen-random.randint(5, 15), t, size - 2, random.randint(15, 60))  t.right(angle)  t.backward(branchLen)  def main():  *"""*  *Sets up the positioning to draw a tree using a turtle and the recursive function tree()*  ***:return****: none*  *"""*  t = turtle.Turtle()  myWin = turtle.Screen()  t.speed(0)  t.left(90)  t.up()  t.backward(100)  t.down()  t.color("green")  tree(75, t, 25, random.randint(15, 60))  myWin.exitonclick()  main() |

|  |
| --- |
| Run the following code in your favorite Python IDE: [recursion-koch-snowflakes.py](http://cs.berea.edu/courses/csc236/code/recursion-koch-snowflakes.py)  In the following code, use comments to identify:   1. Every recursive expression (if none exists use the box below to to explain whether or not this is properly recursive.) 2. Every base case (if none exists use the box below to explain whether or not this is properly recursive.)   Also, write an appropriate docstring for each class, method, and function. |
| import turtle  import random  class Koch:  def \_\_init\_\_(self, order=3, size=200):  self.order = order  self.size = size  self.turtle = turtle.Turtle()  self.turtle.color("pink")  self.turtle.penup()  self.turtle.goto(-1\*size//2, size//2)  self.turtle.pendown()  def draw\_koch(self, t, neworder, newsize):  if neworder == 0: # The base case is when neworder == 0  t.forward(newsize)  else:  for angle in [60, -120, 60, 0]:  self.draw\_koch(t, neworder-1, newsize//3) # This is a  t.left(angle) # recursive statement because it calls itself  def snowflake(self):  for i in range(3):  self.draw\_koch(self.turtle, self.order, self.size)  self.turtle.right(120)  def main():  wn=turtle.Screen()  k=Koch(random.randrange(4), random.randrange(100)+50)  k.snowflake()  wn.exitonclick()  main() |
| **Additional explanation:** |
| This particular function would also difficult to code using iteration rather than recursion. Explain why. |
| **Because you would have to manage a large amount of variables and ensure they are updated and used correctly. Recursion makes this management easy.** |

## Suggestions and Submission

Please offer any suggestions for improvement of this activity from the team:

|  |
| --- |
| **Suggestions for improvement** |
|  |

To submit, the recorder will download as *yourteamname-T13.docx* and upload to Moodle while all other person should simply upload the name of the assignment (T13) and the names of both team members.

1. This assignment is modified by Dr. Jan Pearce, Berea College, from <http://interactivepython.org/runestone/static/pythonds/Recursion/pythondsintro-VisualizingRecursion.html> and from <http://openbookproject.net/thinkcs/python/english3e/recursion.html> so carries the same share and share alike copyright. [↑](#footnote-ref-0)