Recursion with Turtles

CS111 Computer Programming
Department of Computer Science
Wellesley College

Recursion with Turtles

Turtle Graphics

Python has a built-in module named turtle. See the Python turtle module API for details.

Use from turtle import * to use these commands:

- fd(dist)       turtle moves forward by dist
- bk(dist)       turtle moves backward by dist
- lt(angle)      turtle turns left angle degrees
- rt(angle)      turtle turns right angle degrees
- pu()           (pen up) turtle raises pen in belly
- pd()           (pen down) turtle lower pen in belly
- pensize(width) sets the thickness of turtle’s pen to width
- pencolor(color) sets the color of turtle’s pen to color
- shape(shp)     sets the turtle’s shape to shp
- home()         turtle returns to (0,0) (center of screen)
- clear()        delete turtle drawings; no change to turtle’s state
- reset()        delete turtle drawings; reset turtle’s state
- setup(width, height) create a turtle window of given width and height

A Simple Example with Turtles

from turtle import *
setup(400, 400)
fd(100)
lt(60)
shape('turtle')
pencolor('red')
fd(150)
rt(15)
pencolor('blue')
bk(100)
pu()
bk(50)
pd()
pensize(5)
bk(250)
bk(100)
pensize(1)
home()

Looping Turtles

Loops can be used in conjunction with turtles to make interesting designs.

# Draws a polygon with the specified number
# of sides, each with the specified length

def polygon(numSides, sideLength):

polygon(4, 100)
polygon(6, 60)
polygon(100, 3)
polygon(3, 100)
polygon(5, 75)
polygon(7, 50)
Looping Turtles

# Draws "flowers" with numPetals arranged around
# a center point. Each petal is a polygon with
# petalSides sides of length petalLen.

def polyFlow(numPetals, petalSides, petalLen):

polyFlow(7, 4, 80)
polyFlow(10, 5, 75)
polyFlow(11, 6, 60)

Spiraling Turtles: A Recursion Example

def spiral(sideLen, angle, scaleFactor, minLength):

• sideLen is the length of the current side
• angle is the amount the turtle turns left to draw the
  next side
• scaleFactor is the multiplicative factor by which to
  scale the next side (it is between 0.0 and 1.0)
• minLength is the smallest side length that the turtle
  will draw

spiral(625, 90, 0.8, 250)

spiral(200, 90, 0.9, 10)
spiral(200, 72, 0.97, 10)
spiral(200, 80, 0.95, 10)
spiral(200, 121, 0.95, 15)
spiral(200, 95, 0.93, 10)
spiral(200, 121, 0.95, 15)
spiral(200, 95, 0.93, 10)
if sideLen >= minLength:
    fd(sideLen)
    lt(angle)
    spiral(sideLen*scaleFactor, angle, scaleFactor, minLength)
if True:
  fd(sideLen)
  lt(angle)
  spiral(sideLen*scaleFactor, angle, scaleFactor, minLength)
if True:
    fd(625)
    lt(90)
    spiral(500, 90, 0.8, 250)

spiral(500, 90, 0.8, 250)

if True:
    fd(500)
    lt(90)
    spiral(400, 90, 0.8, 250)

spiral(400, 90, 0.8, 250)

if True:
    fd(400)
    lt(90)
    spiral(sideLen*scaleFactor, angle, scaleFactor, minLength)

spiral(320, 90, 0.8, 250)

if sideLen >= minLength:
    fd(sideLen)
    lt(angle)
    spiral(sideLen*scaleFactor, angle, scaleFactor, minLength)

10-21
Invariant Spiraling

A function is invariant relative to an object's state if the state of the object is the same before and after the function is invoked.

```
# Draws a spiral. The state of the turtle (position, color, heading, etc.) after drawing the spiral is the same as before drawing the spiral.
def spiralBack(sideLen, angle, scaleFactor, minLength):
```

Zigzags

```
def zigzag(num, length):
    if num>0:
        lt(45)
        fd(length)
        rt(90)
        fd(2*length)
        lt(90)
        fd(length)
        rt(45)
        zigzag(num-1, length)
```

Exercise: modify zigzag to make the turtle's state invariant.

Trees

```
tree(7, 75, 30, 0.8)
tree(7, 75, 15, 0.8)
tree(10, 80, 45, 0.7)
tree(10, 100, 90, 0.68)
```
How to make a 4 level tree: \texttt{tree(4, 100, 45, 0.6)}

Make a trunk of size 100 and two level 3 trees with 60\% trunks set at 45° angles.

How to make a 3 level tree: \texttt{tree(3, 60, 45, 0.6)}

Make a trunk of size 60 and two level 2 trees with 60\% trunks set at 45° angles.

How to make a 2 level tree: \texttt{tree(2, 36, 45, 0.6)}

Make a trunk of size 36 and two level 1 trees with 60\% trunks set at 45° angles.

How to make a 1 level tree: \texttt{tree(1, 21.6, 45, 0.6)}

Make a trunk of size 21.6 and two level 0 trees set at 45° angles.
How to make a 1 level tree: \texttt{tree(0, 12.96, 45, 0.6)}

Do nothing!

Trees

```python
def tree(levels, trunkLen, angle, shrinkFactor):
```

\texttt{tree(levels, trunkLen, angle, shrinkFactor)}

- \texttt{levels} is the number of branches on any path from the root to a leaf
- \texttt{trunkLen} is the length of the base trunk of the tree
- \texttt{angle} is the angle from the trunk for each subtree
- \texttt{shrinkFactor} is the shrinking factor for each subtree

Tracing the invocation of \texttt{tree(3, 60, 45, 0.6)}
Draw trunk and turn to draw level 2 tree

fd(60)
rt(45)
tree(3,60,45,0.6)

Begin recursive invocation to draw level 2 tree

fd(60)
rt(45)
tree(2,36,45,0.6)
tree(2,36,45,0.6)
tree(3,60,45,0.6)

10-49

10-50

Draw trunk and turn to draw level 1 tree

fd(60)
rt(45)
tree(2,36,45,0.6)
fd(36)
rt(45)
tree(2,36,45,0.6)
tree(3,60,45,0.6)

Begin recursive invocation to draw level 1 tree

fd(60)
rt(45)
tree(2,36,45,0.6)
fd(36)
rt(45)
tree(1,21.6,45,0.6)
tree(2,36,45,0.6)
tree(1,21.6,45,0.6)

10-51

10-52
Draw trunk and turn to draw level 0 tree

Begin recursive invocation to draw level 0 tree

Complete level 0 tree and turn to draw another level 0 tree

Begin recursive invocation to draw level 0 tree
Complete two level 0 trees and return to starting position of level 1 tree

Complete level 1 tree and return to starting position of level 2 tree

Complete level 2 tree and turn to draw another level 2 tree

Draw trunk and turn to draw level 1 tree
Draw trunk and turn to draw level 0 tree

Complete two level 0 trees and return to starting position of level 1 tree

Complete level 1 tree and turn to draw another level 1 tree

Draw trunk and turn to draw level 0 tree
Complete two level 0 trees and return to starting position of level 1 tree

Complete level 1 tree and return to starting position of level 2 tree

Complete level 2 tree and return to starting position of level 3 tree

The squirrels aren't fooled
Random Trees

def treeRandom(length, minLength, thickness, minThickness, 
               minAngle, maxAngle, minShrink, maxShrink):
  if (length < minLength) or (thickness < minThickness): # Base case
    pass # Do nothing
  else:
    angle1 = random.uniform(minAngle, maxAngle)
    angle2 = random.uniform(minAngle, maxAngle)
    shrink1 = random.uniform(minShrink, maxShrink)
    shrink2 = random.uniform(minShrink, maxShrink)
    pensize(thickness)
    fd(length)
    rt(angle1)
    treeRandom(length*shrink1, minLength, thickness*shrink1, 
               minThickness, minAngle, maxAngle, minShrink, maxShrink)
    lt(angle1 + angle2)
    treeRandom(length*shrink2, minLength, thickness*shrink2, 
               minThickness, minAngle, maxAngle, minShrink, maxShrink)
    rt(angle2)
    pensize(thickness)
    bk(length)

Turtle Ancestry

- "Floor turtles" used to teach children problem solving in late 1960s. Controlled by LOGO programming language created by Wally Feurzeig (BBN), Daniel Bobrow (BBN), and Seymour Papert (MIT).

- Logo-based turtles introduced around 1971 by Papert's MIT Logo Laboratory.

- Turtles play a key role in "constructionist learning" philosophy espoused by Papert in *Mindstorms* (1980).

Turtle Ancestry (cont'd)

- LEGO/Logo project at MIT (Mitchel Resnick and Steve Ocko, 1988); evolves into Handyboards (Fred Martin and Brian Silverman), Crickets (Robbie Berg @ Wellesley), and LEGO Mindstorms

Turtles, Buggles, & Friends At Wellesley

- In mid-1980s, Eric Roberts teaches programming using software-based turtles.
- In 1996, Randy Shull and Takis Metaxas use turtles to teach problem solving in CS110.
- In 1997, BuggleWorld introduced by Lyn Turbak when CS111 switches from Pascal to Java. Turtles are also used in the course
- In 2006, Robbie Berg and others introduce PICO Crickets: [http://www.picocricket.com](http://www.picocricket.com)
- In 2011, Lyn Turbak and the TinkerBlocks group introduce TurtleBlocks, a blocks-based turtle language whose designs can be turned into physical artifacts with laser and vinyl cutters.
Laser Cutting a Tree

regular mode

boundary mode

laser cutting

10-77