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

```python
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)
pensize(1)
home()
exitonclick()
```

Looping Turtles

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

```python
def polygon(numSides, sideLength):
    """ Draws a polygon with the specified number of sides, each with the specified length. """
```
Looping Turtles

```python
def polyFlow(numPetals, petalSides, petalLen):
    """Draws 'flowers' with numPetals arranged around a center point. Each petal is a polygon with petalSides sides of length petalLen. """
```

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

Spiraling Turtles: A Recursion Example

```python
def spiral(sideLen, angle, scaleFactor, minLength):
    """Draw a spiral recursively."""
    if sideLen >= minLength:
        fd(sideLen)
        lt(angle)
        spiral(sideLen*scaleFactor, angle, scaleFactor, minLength)
```

```
spiral(200, 90, 0.9, 10)
spiral(200, 90, 0.97, 10)
spiral(200, 90, 0.95, 10)
spiral(200, 90, 0.93, 10)
spiral(200, 121, 0.95, 15)
```

Discuss: Can we draw these patterns using loops?
spiral(625, 90, 0.8, 250)

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

spiral(625, 90, 0.8, 250)

if sideLen >= minLength:
    fd(sideLen)
    lt(angle)
    spiral(sideLen*scaleFactor, angle, scaleFactor, minLength)
```python
spiral(625, 90, 0.8, 250)

if True:
    fd(625)
    lt(90)
    spiral(500, 90, 0.8, 250)

spiral(500, 90, 0.8, 250)

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

spiral(500, 90, 0.8, 250)

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

Turtle Recursion 13-13

Turtle Recursion 13-14

Turtle Recursion 13-15

Turtle Recursion 13-16
```python
spiral(625, 90, 0.8, 250)
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(sideLen)
    lt(angle)
    spiral(sideLen*scaleFactor, angle, scaleFactor, minLength)
```
spiral(625, 90, 0.8, 250)

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(320, 90, 0.8, 250)

spiral(320, 90, 0.8, 250)

if True:
    if True:
        if True:
            spiral(sideLen*scaleFactor, angle, scaleFactor, minLength)
        spiral(sideLen*scaleFactor, angle, scaleFactor, minLength)
        lt(angle)
        spiral(sideLen*scaleFactor, angle, scaleFactor, minLength)
    if True:
        if True:
            if True:
                spiral(sideLen*2, angle, scaleFactor, minLength)
            spiral(sideLen*2, angle, scaleFactor, minLength)
            lt(angle)
            spiral(sideLen*2, angle, scaleFactor, minLength)
        spiral(sideLen*2, angle, scaleFactor, minLength)
        lt(angle)
        spiral(sideLen*2, angle, scaleFactor, minLength)
    if True:
        if True:
            if True:
                spiral(sideLen*3, angle, scaleFactor, minLength)
            spiral(sideLen*3, angle, scaleFactor, minLength)
            lt(angle)
            spiral(sideLen*3, angle, scaleFactor, minLength)
        spiral(sideLen*3, angle, scaleFactor, minLength)
        lt(angle)
        spiral(sideLen*3, angle, scaleFactor, minLength)
spiral(625, 90, 0.8, 250)
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(320, 90,
        0.8, 250)
spiral(320, 90, 0.8, 250)
if True:
    fd(320)
    lt(90)
    spiral(256, 90,
        0.8, 250)
spiral(256, 90, 0.8, 250)
if True:
    fd(256)
    lt(90)
    spiral(204.8, 90,
        0.8, 250)
Be turtle, draw.
spiral(625, 90, 0.8, 250)

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(320, 90, 0.8, 250)

spiral(320, 90, 0.8, 250)
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.

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

How does `spiralBack` work?

```python
spiralBack(625, 90, 0.8, 250)
```

```
spiralBack(625, 90, 0.8, 250)
if True:
    fd(sideLen)
    lt(90)
    spiralBack(500, 90, 0.8, 250)
    rt(angle)
    bk(sideLen)
spiralBack(500, 90, 0.8, 250)
if True:
    fd(sideLen)
    lt(90)
    spiralBack(400, 90, 0.8, 250)
    rt(angle)
    bk(sideLen)
spiralBack(400, 90, 0.8, 250)
if True:
    fd(sideLen)
    lt(90)
    spiralBack(320, 90, 0.8, 250)
    rt(angle)
    bk(sideLen)
spiralBack(320, 90, 0.8, 250)
if False:
    fd(sideLen)
    lt(angle)
    spiralBack(sideLen*scaleFactor, ...
    rt(angle)
    bk(sideLen)
```

Trees

```
tree(7, 75, 75, 0.8)
tree(10, 100, 90, 0.68)
tree(7, 75, 15, 0.8)
```

Essence of Invariance

<table>
<thead>
<tr>
<th>Do state change 1</th>
<th>Do state change 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Do state change n-1</td>
<td>Do state change n</td>
</tr>
</tbody>
</table>

Perform changes to state

Recursive call to function

```
spiralBack(625, 90, 0.8, 250)
```

```
spiralBack(625, 90, 0.8, 250)
if True:
    fd(sideLen)
    lt(90)
    spiralBack(500, 90, 0.8, 250)
    rt(angle)
    bk(sideLen)
spiralBack(500, 90, 0.8, 250)
if True:
    fd(sideLen)
    lt(90)
    spiralBack(400, 90, 0.8, 250)
    rt(angle)
    bk(sideLen)
spiralBack(400, 90, 0.8, 250)
if True:
    fd(sideLen)
    lt(90)
    spiralBack(320, 90, 0.8, 250)
    rt(angle)
    bk(sideLen)
spiralBack(320, 90, 0.8, 250)
if False:
    fd(sideLen)
    lt(angle)
    spiralBack(sideLen*scaleFactor, ...
    rt(angle)
    bk(sideLen)
```

Udoo state changes in opposite order

```
tree(7, 75, 75, 0.8)
tree(10, 100, 90, 0.68)
tree(7, 75, 15, 0.8)
```
Draw a tree recursively

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

- \text{levels} is the number of branches on any path from the root to a leaf
- \text{trunkLen} is the length of the base trunk of the tree
- \text{angle} is the angle from the trunk for each subtree
- \text{shrinkFactor} is the shrinking factor for each subtree
How to make a 2 level tree: \texttt{tree(2, 36, 45, 0.6)}

and two level 1 trees
With 60\% trunks set at 45° angles

Make a trunk of size 36

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

and two level 0 trees set at 45° angles

Make a trunk of size 21.6

How to make a 0 level tree: \texttt{tree(0, 12.96, 45, 0.6)}

Do nothing!

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

"""Draw a 2-branch tree recursively."

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

"""
Tracing the invocation of \( \text{tree}(3, 60, 45, 0.6) \)

Draw trunk and turn to draw level 2 tree

Begin recursive invocation to draw level 2 tree

Draw trunk and turn to draw level 1 tree
Begin recursive invocation to draw level 1 tree

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

Turtle Recursion 13-53
Begin recursive invocation to draw level 0 tree

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

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

Begin recursive invocation to draw level 1 tree

Turtle Recursion 13-57

Turtle Recursion 13-58

Turtle Recursion 13-59

Turtle Recursion 13-60
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 turn to draw another level 2 tree
Draw trunk and turn to draw level 1 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
Trace the invocation of
\texttt{tree(3, 60, 45, 0.6)}

\begin{enumerate}
\item \texttt{tree(3, 60, 45, 0.6)}
\item \texttt{fd(60)} \texttt{rt(45)} \texttt{tree(2,36,45,0.6)} \texttt{lt(90)} \texttt{tree(2,36,45,0.6)} \texttt{rt(45)} \texttt{bk(60)}
\item \texttt{tree(2,36,45,0.6)} \texttt{fd(36)} \texttt{rt(45)} \texttt{tree(1,21.6,45,0.6)} \texttt{lt(90)} \texttt{tree(1,21.6,45,0.6)} \texttt{rt(45)} \texttt{bk(36)}
\item \texttt{tree(2,36,45,0.6)} \texttt{fd(21.6)} \texttt{rt(45)} \texttt{tree(1,12.96,45,0.6)} \texttt{lt(90)} \texttt{tree(1,12.96,45,0.6)} \texttt{rt(45)} \texttt{bk(21.6)}
\item \texttt{tree(2,36,45,0.6)} \texttt{fd(36)} \texttt{rt(45)} \texttt{tree(1,21.6,45,0.6)} \texttt{lt(90)} \texttt{tree(1,21.6,45,0.6)} \texttt{rt(45)} \texttt{bk(36)}
\item \texttt{tree(2,36,45,0.6)} \texttt{fd(21.6)} \texttt{rt(45)} \texttt{tree(1,12.96,45,0.6)} \texttt{lt(90)} \texttt{tree(1,12.96,45,0.6)} \texttt{rt(45)} \texttt{bk(21.6)}
\item \texttt{tree(2,36,45,0.6)} \texttt{fd(21.6)} \texttt{rt(45)} \texttt{tree(1,12.96,45,0.6)} \texttt{lt(90)} \texttt{tree(1,12.96,45,0.6)} \texttt{rt(45)} \texttt{bk(21.6)}
\item \texttt{tree(2,36,45,0.6)} \texttt{fd(21.6)} \texttt{rt(45)} \texttt{tree(1,12.96,45,0.6)} \texttt{lt(90)} \texttt{tree(1,12.96,45,0.6)} \texttt{rt(45)} \texttt{bk(21.6)}
\item \texttt{tree(2,36,45,0.6)} \texttt{fd(21.6)} \texttt{rt(45)} \texttt{tree(1,12.96,45,0.6)} \texttt{lt(90)} \texttt{tree(1,12.96,45,0.6)} \texttt{rt(45)} \texttt{bk(21.6)}
\end{enumerate}

\textbf{Random Trees}

\begin{lstlisting}
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)
\end{lstlisting}

\textbf{The squirrels aren't fooled…}

Be the turtle, draw the tree, label trunks with 7.

\textbf{More resources:}
http://cs111.wellesley.edu/lectures/lecture13

- All steps of recursion examples, drawn out.
- Exercises for drawing Koch curves and snowflakes with recursive turtle functions.
- History about turtles at Wellesley and elsewhere.
- Applying the turtle programming abstraction to control laser cutters in the WeLab (Wellesley engineering lab).
Drawing fractals – Koch Curve

```
koch(levels, size)
  koch(0, 150)
  koch(1, 150)
  koch(2, 150)
  koch(3, 150)
```

Turtle Recursion 13-77

Snowflakes

```
snowflake(levels, size)
  snowflake(0, 150)
  snowflake(1, 150)
  snowflake(2, 150)
  snowflake(3, 150)
```

Turtle Recursion 13-78

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).
- 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.

Turtle Recursion 13-79
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 with Turtle Blocks