Recursion with Turtles

CS111 Computer Programming
Department of Computer Science
Wellesley College

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

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`

Looping Turtles (1)

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

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

![](turtle.png)
**Looping Turtles (2)**

```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. """
```

```python
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)
```

```python
spiral(625, 90, 0.8, 250)
```
Substitute computed values (not expressions) to show call frame model of this invocation.

```
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(625)
    lt(90)
    spiral(500, 90, 0.8, 250)
```

```
spiral(500, 90, 0.8, 250)
```

```
if True:
    fd(500)
    lt(90)
    spiral(320, 90, 0.8, 250)
```

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

Essence of Invariance

Do state change 1
Do state change 2
...  
Do state change n-1
Do state change n

Perform changes to state

Recursive call to function

Undo state change n
Undo state change n-1
...  
Undo state change 2
Undo state change 1

Undo state changes in opposite order

Trees
Draw a tree recursively

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

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

How to make a 4-level tree: \text{tree}(4, 100, 45, 0.6)

def \text{tree}(levels, trunkLen, angle, shrinkFactor):
    """Draw a 2-branch tree recursively.
    levels: number of branches on any path from the root to a leaf
    trunkLen: length of the base trunk of the tree
    angle: angle from the trunk for each subtree
    shrinkFactor: shrinking factor for each subtree
    """
Trace the invocation of 
\textbf{tree}(3, 60, 45, 0.6)