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:

```python
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

The only two commands that draw lines are `fd` and `bk`.

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

Looping Turtles (1)

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. """
    pass
```

Will solve this in the Notebook.
**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
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
def polyFlow(7, 4, 80)
def polyFlow(5, 5, 75)
def 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
def spiral(sideLen, angle, scaleFactor, minLength):
    """Draw a spiral recursively."""
    if sideLen >= minLength:
        fd(sideLen)
        lt(angle)
        spiral(sideLen*scaleFactor, angle, scaleFactor, minLength)
```

```
def spiral(625, 90, 0.8, 250)
```

**Answer this:**

How would you create these shapes using loops? Recursion makes easier solving certain problems that involve a repeating pattern.

**Concepts in this slide:**

- Drawing function call frames helps us follow the execution of recursion.
if True:
    fd(sideLen)
    lt(angle)
    spiral(sideLen*scaleFactor, angle, scaleFactor, minLength)

if sideLen >= minLength:
    fd(sideLen)
    lt(angle)
    spiral(sideLen*scaleFactor, angle, scaleFactor, minLength)
```python
if True:
    fd(625)
    lt(90)
    spiral(500, 90, 0.8, 250)
spiral(500, 90, 0.8, 250)
spiral(625, 90, 0.8, 250)
```

```python
if sideLen >= minLength:
    fd(sideLen)
    lt(angle)
    spiral(sideLen*scaleFactor, angle, scaleFactor, minLength)
spiral(400, 90, 0.8, 250)
```

```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)
```
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(400, 90, 0.8, 250)

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

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

spiral(500, 90, 0.8, 250)

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

spiral(400, 90, 0.8, 250)

if True:
    fd(400)
l(t(90)
spiral(320, 90, 0.8, 250)

spiral(320, 90, 0.8, 250)

if True:
    fd(320)
l(t(angle)
spiral(sideLen*scaleFactor, angle, scaleFactor, minLength)

spiral(320, 90, 0.8, 250)

if True:
    fd(320)
l(t(angle)
spiral(sideLen*scaleFactor, angle, scaleFactor, minLength)

spiral(256, 90, 0.8, 250)

if sideLen >= minLength:
    fd(sideLen)
l(t(angle)
spiral(sideLen*scaleFactor, angle, scaleFactor, minLength)
spiral(625, 90, 0.8, 250)

Important
Initially all execution frames co-exist in the memory. Only once a function has returned (implicitly), the execution frame is deleted.

Turtle Recursion 19-30
Important
All execution frames were one by one deleted after their completion. This terminates the invocation of the function and has created as a "side-effect" the turtle image at the top of the slide.
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. """
```
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:
\[ \text{tree}(4, 100, 45, 0.6) \]

Step 1
- Make a trunk of size 100
- Make two 3-level trees with 60% trunks set at 45° angles

Step 2
- And two 3-level trees with 60% trunks set at 45° angles

How to make a 3-level tree:
\[ \text{tree}(3, 60, 45, 0.6) \]

Make a trunk of size 60
- And two 2-level trees with 60% trunks set at 45° angles

How to make a 2-level tree:
\[ \text{tree}(2, 36, 45, 0.6) \]

Make a trunk of size 36
- And two 1-level trees with 60% trunks set at 45° angles

How to make a 1-level tree:
\[ \text{tree}(1, 21.6, 45, 0.6) \]

Make a trunk of size 21.6
- And two 0-level trees set at 45° angles

Do nothing!

How to make a 0-level tree:
\[ \text{tree}(0, 12.96, 45, 0.6) \]
How to make a 2 level tree: \textbf{tree}(2, 36, 45, 0.6)

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

Make a trunk of size 36

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

and two level 0 trees
set at 45^\circ angles

Make a trunk of size 21.6

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

Do nothing!

\begin{verbatim}
def tree(levels, trunkLen, angle, shrinkFactor):
    """Draw a 2-branch tree recursively.
    \textbf{levels:} number of branches on any path from the root to a leaf
    \textbf{trunkLen:} length of the base trunk of the tree
    \textbf{angle:} angle from the trunk for each subtree
    \textbf{shrinkFactor:} shrinking factor for each subtree
    """
    if 0 < levels:
        # Draw the trunk.
        fd(trunkLen)
        # Turn and draw the right subtree.
        rt(angle)
        tree(levels-1, trunkLen*shrinkFactor, angle, shrinkFactor)
        # Turn and draw the left subtree.
        lt(angle * 2)
        tree(levels-1, trunkLen*shrinkFactor, angle, shrinkFactor)
        # Turn back and back up to root without drawing.
        rt(angle)
        pu()
        bk(trunkLen)
        pd()
\end{verbatim}
Tracing the invocation of \texttt{tree(3,60, 45,0.6)}

Begin recursive invocation to draw level 2 tree

Draw trunk and turn to draw level 2 tree

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

- **Turtle Recursion 19-61**

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

- **Turtle Recursion 19-62**

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

- **Turtle Recursion 19-63**

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

- **Turtle Recursion 19-64**
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
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)
```

More resources

- 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 19-77

Snowflakes

snowflake(0,150)
snowflake(1,150)
snowflake(2,150)
snowflake(3,150

Turtle Recursion 19-78

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 19-79

Turtle Recursion 19-80