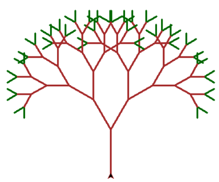
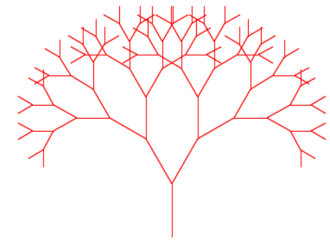
More Recursion



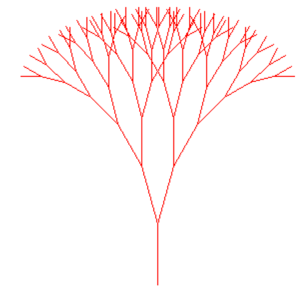
CS111 Computer Programming

Department of Computer Science
Wellesley College

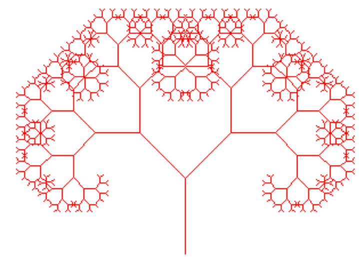
Trees



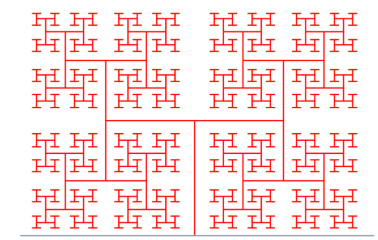
`tree(7, 75, 30, 0.8)`



`tree(7, 75, 15, 0.8)`



`tree(10, 80, 45, 0.7)`



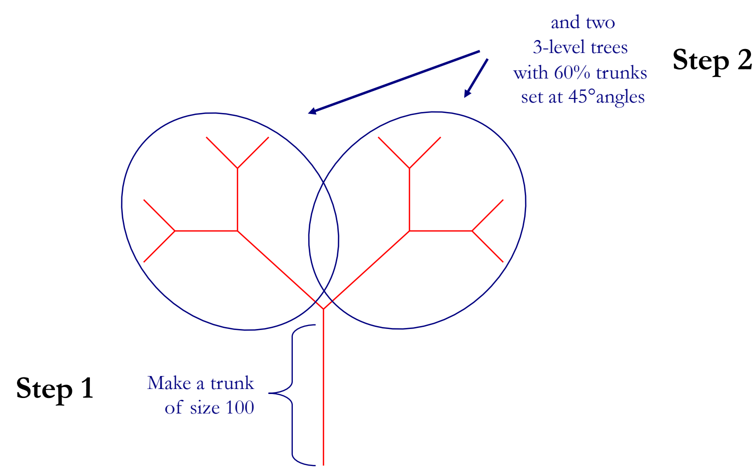
`tree(10, 100, 90, 0.68)`

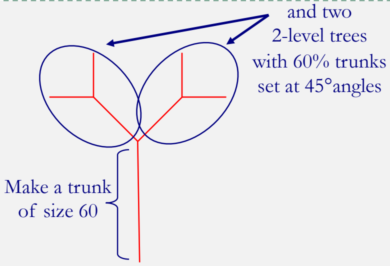
Draw a tree recursively

`tree(levels, trunkLen, angle, shrinkFactor)`

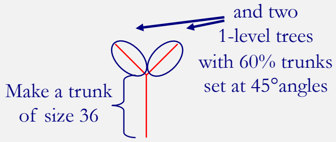
- `levels` is the number of branches on any path from the root to a leaf
- `trunkLen` is the length of the base trunk of the tree
- `angle` is the angle from the trunk for each subtree
- `shrinkFactor` is the shrinking factor for each subtree

How to make a 4-level tree: `tree(4, 100, 45, 0.6)`

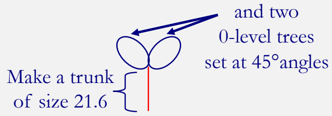




How to make a **3-level tree:**
`tree(3, 60, 45, 0.6)`



How to make a **2-level tree:**
`tree(2, 36, 45, 0.6)`



How to make a **1-level tree:**
`tree(1, 21.6, 45, 0.6)`

Do nothing!

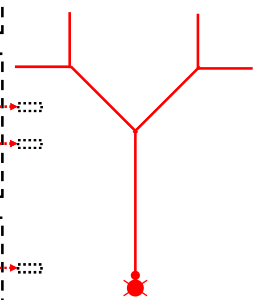
How to make a **0-level tree:**
`tree(0, 12.96, 45, 0.6)`


```
def 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
    """
    if levels > 0:
        # 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()
```


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

See full slides for trace

```
tree(3,60,45,0.6)
fd(60)
rt(45)
tree(2,36,45,0.6)
lt(90)
tree(2,36,45,0.6)
rt(45)
bk(60)

tree(2,36,45,0.6)
fd(36)
rt(45)
tree(1,21.6,45,0.6)
lt(90)
tree(1,21.6,45,0.6)
rt(45)
bk(36)

tree(1,21.6,45,0.6)
fd(21.6)
rt(45)
tree(0,12.96,45,0.6)
lt(90)
tree(0,12.96,45,0.6)
rt(45)
bk(21.6)
```



Trace the invocation of
`tree(3, 60, 45, 0.6)`

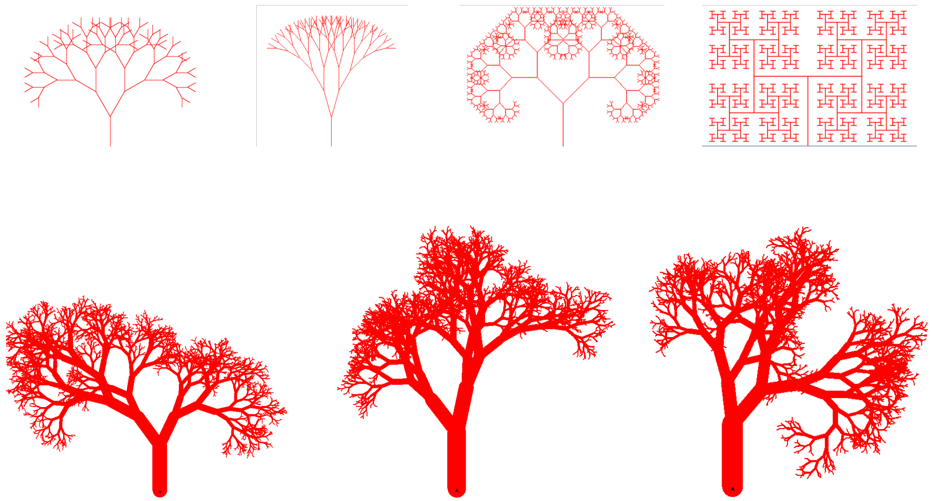
Fill in the BLANK

Be the turtle, draw the tree, label trunks with *i*.

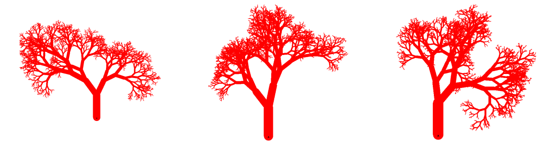
1 2 3 4 5 6 7

```
tree(3,60,45,0.6)
fd(60)
rt(45)
tree(2,36,45,0.6)
lt(90)
tree(2,36,45,0.6)
rt(45)
bk(60)

tree(2,36,45,0.6)
fd(36)
rt(45)
tree(1,21.6,45,0.6)
lt(90)
tree(1,21.6,45,0.6)
rt(45)
bk(36)

tree(1,21.6,45,0.6)
fd(21.6)
rt(45)
tree(0,12.96,45,0.6)
lt(90)
tree(0,12.96,45,0.6)
rt(45)
bk(21.6)
```

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)
```


it's your turn

Fruitful Trees

As with spiral, we can return counts of the drawings we make using fruitful recursion. Try this example below in the notebook and check the notebook solution for answers.

```
def branchCount(levels, trunkLen, angle, shrinkFactor):
    """Draw a 2-branch tree recursively and returns a
    count of the branches.
    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
    """
    # your code here
```


it's your turn

Digging Deeper

Drawing fractals – Koch Curve

koch(levels, size)

koch(0, 150)

koch(1, 150)

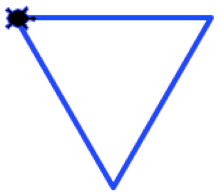
koch(2, 150)

koch(3, 150)

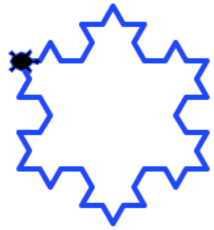
Snowflakes

it's your turn

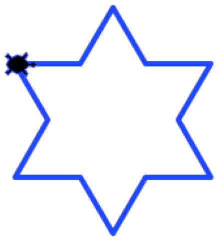
Digging Deeper



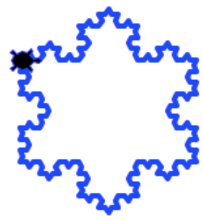
`snowflake(0,150)`



`snowflake(2,150)`



`snowflake(1,150)`



`snowflake(3,150)`

Turtle Ancestry

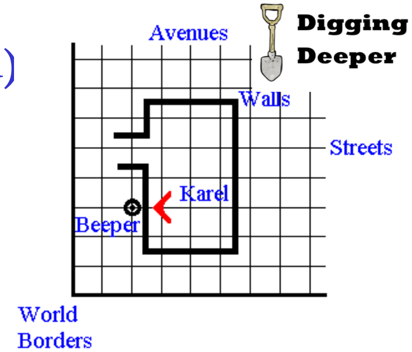
Digging Deeper

- "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)

Digging Deeper

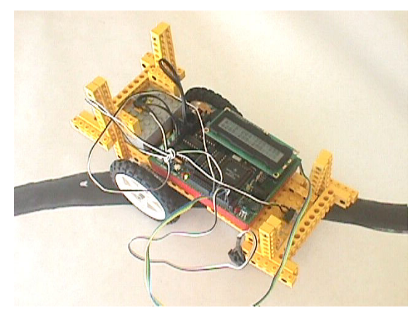
- Richard Pattis's Karel the Robot (1981) teaches problem-solving using Pascal robots that manipulate beepers in a grid world.
- *Turtle Geometry* book by Andrea diSessa and Hal Abelson (1986).
- LEGO/Logo project at MIT (Mitchel Resnick and Steve Ocko, 1988); evolves into Handyboards (Fred Martin and Brian Silverman), Crickets (Robbie Berg @ Wellesley), and LEGO Mindstorms
- StarLogo – programming with thousands of turtles in Resnick's *Turtles, Termites, and Traffic Jams* (1997).



Turtles, Buggles, & Friends At Wellesley

Digging Deeper

- In mid-1980s, Eric Roberts teaches programming using software-based turtles.
- In 1996, Robbie Berg and Lyn Turbak start teaching Robotic Design Studio with Sciborgs.
- 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
- 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.



List of numbers from n down to 1

Define a function **countDownList** to return the list of numbers from n down to 1

```
countDownList(0) → [ ]
countDownList(5) → [5, 4, 3, 2, 1]
countDownList(8) → [8, 7, 6, 5, 4, 3, 2, 1]
```

Apply the wishful thinking strategy on n = 4:

- **countDownList(4)** should return **[4, 3, 2, 1]**
- By wishful thinking, assume **countDownList(3)** returns **[3, 2, 1]**
- How to combine **4** and **[3, 2, 1]** to yield **[4, 3, 2, 1]**?
  [4] + [3, 2, 1]
- Generalize: **countDownList(n) = [n] + countDownList(n-1)**

countDownList(n)

```python
def countDownList(n):
    """Returns a list of numbers from n down to 1.
       For example, countDownList(5) returns
       [5,4,3,2,1]."""
    if n <= 0:
        return []
    else:
        return [n] + countDownList(n-1)
```


Exercise: Define countDownListPrintResults(n)

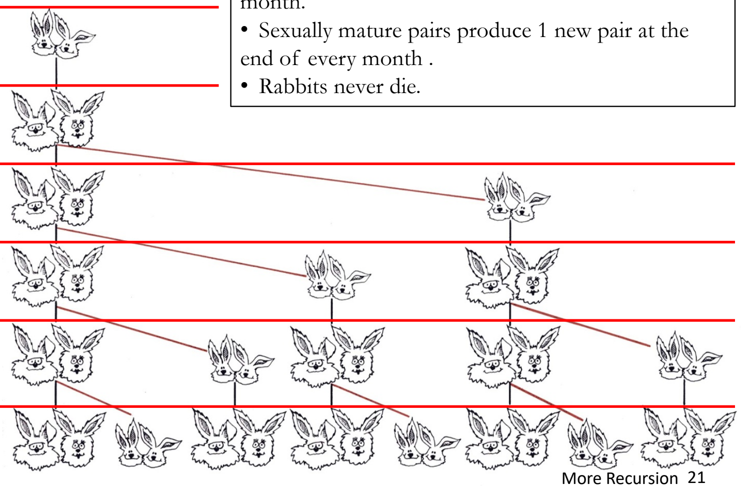
```python
def countDownListPrintResults(n):
    """Returns a list of numbers from n down to 1
       and also prints each recursive result along
       the way."""
    if n <= 0:
        print([])
        result = []
    else:
        result = [n] + countDownListPrintResults(n-1)
        print(result)
        return result
```


Exercise: Define countUpList(n)

```python
def countUpList(n):
    """Returns a list of numbers from 1 up to n.
       For example, countUpList(5) returns
       [1,2,3,4,5]."""
    if n <= 0:
        return []
    else:
        return countUpList(n-1) + [n]
```

Leonardo Pisano Fibonacci counts Rabbits

Month	# Pairs
0	0
1	1
2	1
3	2
4	3
5	5
6	8



Assume:
- Start with one pair of newborn rabbits in month 1.
- Newborn rabbits become sexually mature after 1 month.
- Sexually mature pairs produce 1 new pair at the end of every month .
- Rabbits never die.

Exercise: Fibonacci Numbers fib(n)

it's your turn

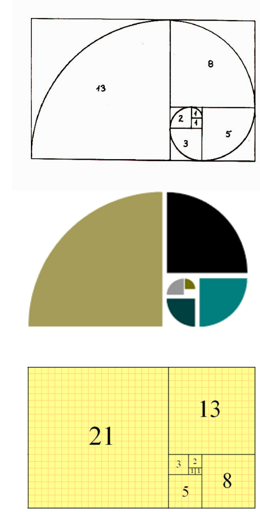
The **nth** Fibonacci number fib(n) is the number of pairs of rabbits alive in the **nth** month.

Formula:

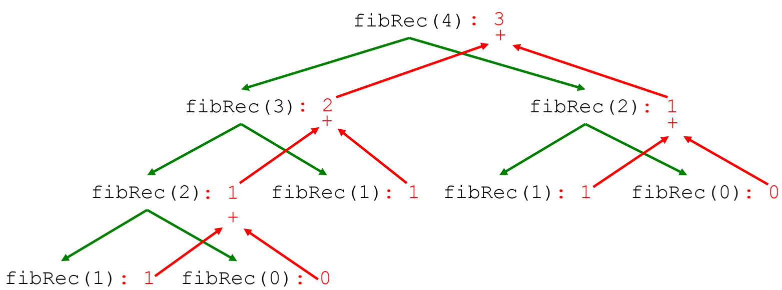
fib(0) = 0 ; no pairs initially
fib(1) = 1 ; 1 pair introduced the first month
fib(n) = fib(n-1) ; pairs never die, so live to next month
+ fib(n-2) ; all sexually mature pairs produce
; a pair each month

Now write the program:

```
def fibRec(n):
    '''Returns the nth Fibonacci number.'''
    if n <= 1:
        return n
    else:
        return fibRec(n-1) + fibRec(n-2)
```



Fibonacci: Efficiency



How long would it take to calculate `fibRec(100)`?

Is there a better way to calculate Fibonacci numbers?

Iteration leads to a more efficient fib(n)

The Fibonacci sequence: 0, 1, 1, 2, 3, 5, 8, 13, 21, …

Iteration table for calculating the 8th Fibonacci number:

i	fibi	fibi_next
0	0	1
1	1	1
2	1	2
3	2	3
4	3	5
5	5	8
6	8	13
7	13	21
8	21	34

Exercise: `fibLoop(n)`

Use iteration to calculate Fibonacci numbers more efficiently:

i	fibi	fibi_next
0	0	1
1	1	1
2	1	2
3	2	3
4	3	5
5	5	8
6	8	13
7	13	21
8	21	34

```
def fibLoop(n):
    '''Returns the nth Fibonacci number.'''
    fibi = 0
    fibi_next = 1
    for i in range(1, n+1):
        fibi, fibi_next = fibi_next, fibi+fibi_next
        # tuple assignment simultaneously updates state vars
    return fibi
```