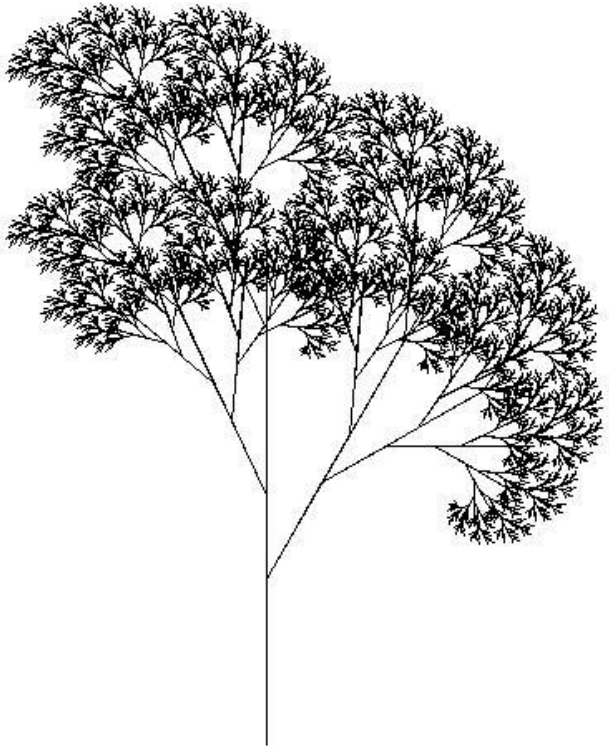
Graphics and Computer Vision

Recursion

Drawing Trees



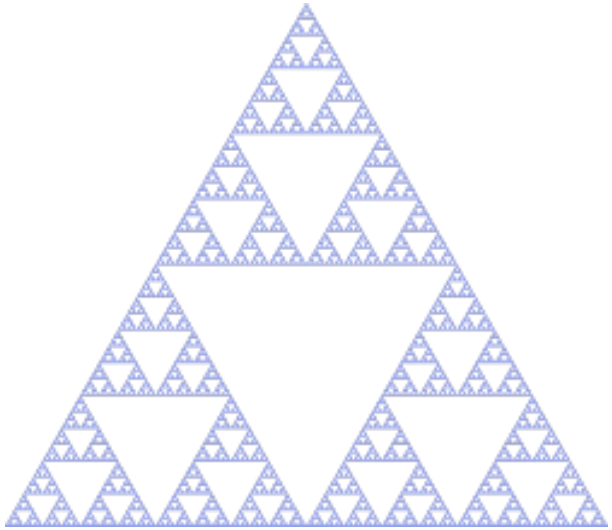
https://en.wikipedia.org/wiki/Recursion_(computer_science)

Recursion

Functions that call themselves

http://interactivepython.org/runestone/static/thinkcspy/IntroRecursion/intro-VisualizingRecursion.html

Recursion examples



Fractals
https://en.wikipedia.org/wiki/Recursion

The picture is defined by the picture itself.

http://interactivepython.org/runestone/static/pythonds/Recursion/WhatIsRecursion.html

What is recursion?

A concise, **elegant** way to code a complex algorithm, without needing loops.

You need 3 things:

1. a **function** with...
2. a **base (terminating) case**...
3. that **calls itself** and **moves towards the base case**.

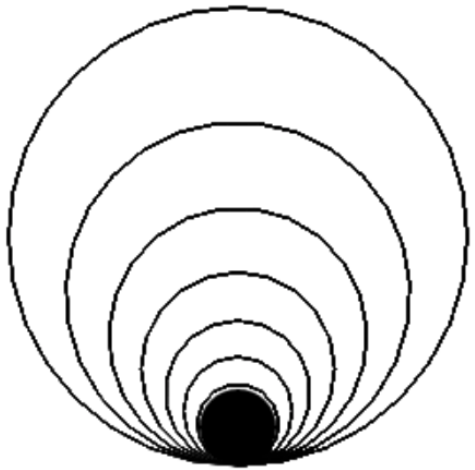
http://interactivepython.org/runestone/static/pythonds/Recursion/TheThreeLawsofRecursion.html

Recursive vortex

```
1   ##########################################
2   #
3   #  Visualizing recursion: a vortex, version 1
4   #
5   #  Author : CMPT 120
6   #  Date: Oct. 21, 2020
7
8   import turtle
9   pete = turtle.Turtle()
10
11  # recursive function!
12  def vortex (size):
13      if size <=20:
14          pete.left(90)
15          pete.forward(20)
16          pete.dot(20)
17      else:
18          pete.circle(size)
19          vortex(size*0.75)
20
21  # top level
22  vortex(120)
23
```

Base case

Recursive call



Recursive countdown

Function

Base case

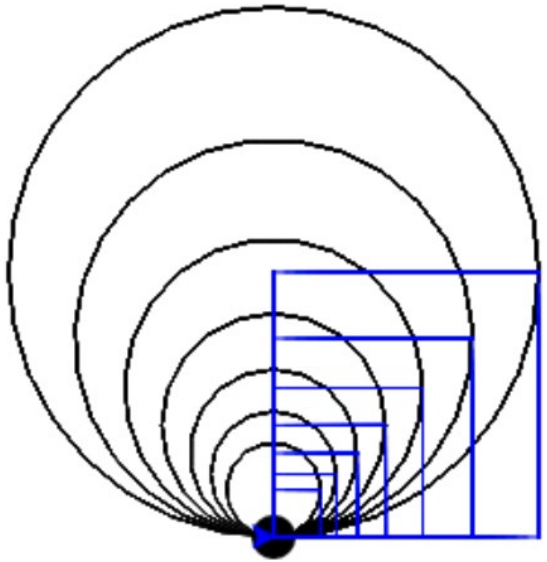
Calls itself moving toward base case

```
1  def print_special(n):
2      if n == 1:
3          print(n)
4      else:
5          print(n)
6          print_special(n-1)
7
8  print_special(5)
9
```

```
5
4
3
2
1
```

Recursive box up

```
1   ##########################################
2   #
3   #  Visualizing recursion: a vortex, version 2
4   #
5   #  Author : CMPT 120
6   #  Date: Oct. 21, 2020
7
8   import turtle
9   pete = turtle.Turtle()
10
11  def square(side):
12      pete.color("blue")
13      for i in range(4):
14          pete.forward(side)
15          pete.left(90)
16
17  # recursive function!
18  def vortex (size):
19      if size <=20:
20          pete.dot(10)
21      else:
22          pete.circle(size)
23          vortex(size*0.75)
24          square(size)
25
26  # top level
27  vortex(120)
```

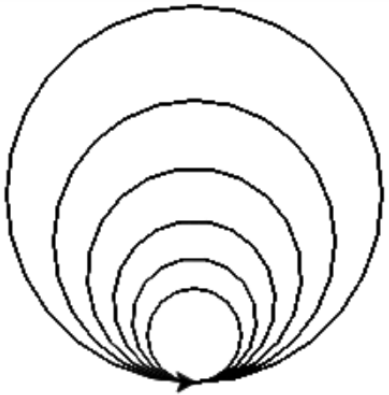


Recursive box up

```
1   ##########################################
2   #
3   #  Visualizing recursion: a vortex, version 3
4   #
5   #  Author : CMPT 120
6   #  Date: Oct. 21, 2020
7
8   import turtle
9   pete = turtle.Turtle()
10
11  def square(side):
12      pete.color("blue")
13      for i in range(4):
14          pete.forward(side)
15          pete.left(90)
16
17  # recursive function!
18  def vortex (size):
19      if size > 20:
20          pete.circle(size)
21          vortex(size*0.75)
22          square(size)
23
24  # top level
25  vortex(120)
```

else... do nothing!

The **base case** can be implicit. If the size is <=20, then just don't draw anymore.



Recursive countup

```
1  def print_special(n):
2      if n >= 1:
3          print_special(n-1)
4          print(n)
5  print_special(5)
```

```
1
2
3
4
5
>
```

print_special(1)

print_special(2)

print_special(3)

print_special(4)

print_special(5)

1

2

3

4

5

The **base case** can be implicit. If n < 1, then don't do anything.

This gets pushed onto the recursive stack first, waiting to be completed while other functions are called.

ExWeek11 > guess_draw.py > ...

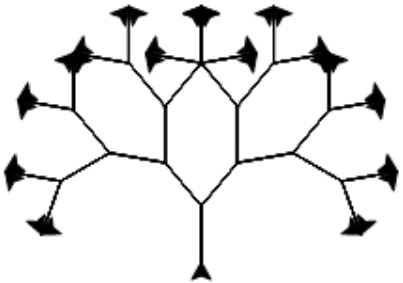
```
1   # What does this output?
2   import turtle
3   pete = turtle.Turtle()
4
5   SIZE = 20
6
7   def guess(n):
8       if n == 0:
9           pete.dot(10)
10      else:
11          pete.setheading(0)
12          pete.circle(SIZE, 180)
13
14          # Recurse towards base
15          guess(n-1)
16
17          pete.setheading(180)
18          pete.circle(SIZE, 180)
19
20  guess(3)
```

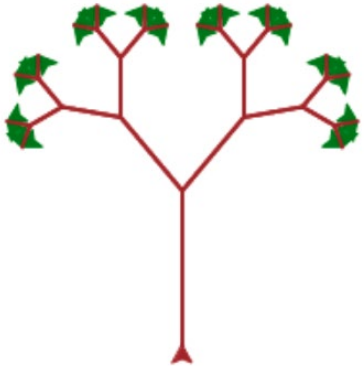
Draw half a circle.

What will this draw?

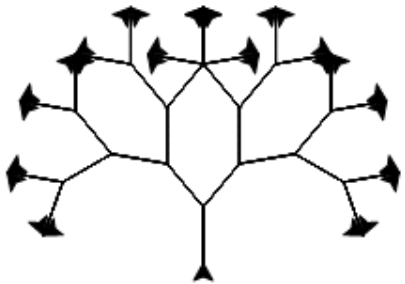
a) One circle
b) Stack of circles
c) Vortex
d) Row of half-circles

Recursion to draw a tree





Recursion to draw a tree



```
3  import turtle
4  bob = turtle.Turtle()
5
6  # Define a recursive function to draw trees
7  def draw_tree(level):
8      # Base case
9      if level == 0:
10         bob.stamp()
11
12     else:
13         # Draw a branch
14         bob.forward(30)
15
16         # Turn left and make mini-tree
17         bob.left(40)
18         draw_tree(level - 1)
19         bob.right(40)
20
21         # Turn right and make mini-tree
22         bob.right(40)
23         draw_tree(level - 1)
24         bob.left(40)
25
26         # Go back down branch
27         bob.back(30)
28
29 # Setup turtle direction and speed
30 bob.speed(0)
31 bob.setheading(90)
32
33 # Start drawing trees
34 draw_tree(5)
```

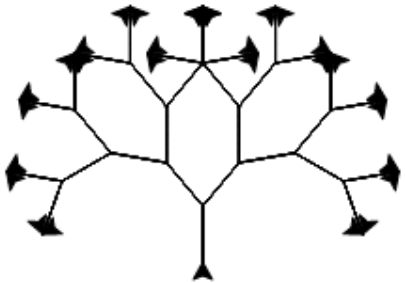
A **function**...

that has a **base case**...

and **calls itself**, moving closer to base case

The function call that **starts** it all!

Recursion to draw a tree

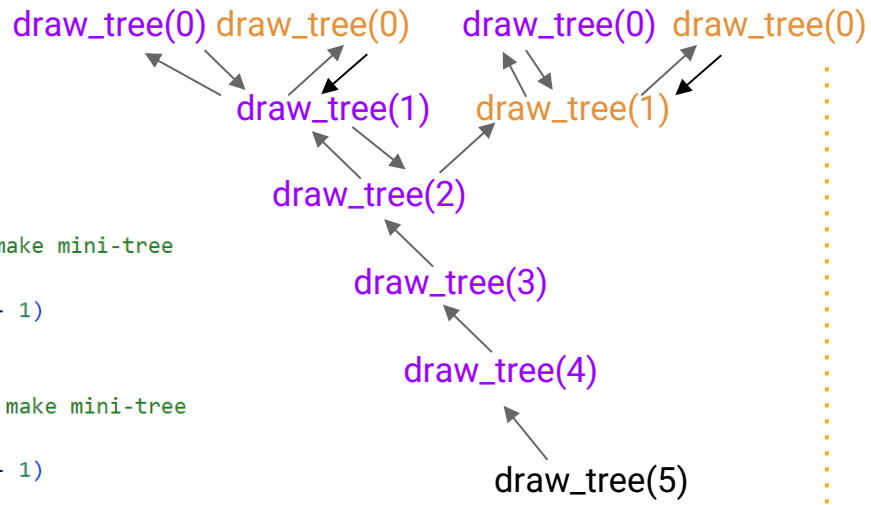


```
3   import turtle
4   bob = turtle.Turtle()
5
6   # Define a recursive function to draw trees
7   def draw_tree(level):
8       # Base case
9       if level == 0:
10          bob.stamp()
11
12      else:
13          # Draw a branch
14          bob.forward(30)
15
16          # Turn left and make mini-tree
17          bob.left(40)
18          draw_tree(level - 1)
            bob.right(40)

            # Turn right and make mini-tree
            bob.right(40)
            draw_tree(level - 1)
            bob.left(40)

            # Go back down branch
            bob.back(30)

    # Setup turtle direction and speed
    bob.speed(0)
31  bob.setheading(90)
32
33  # Start drawing trees
34  draw_tree(5)
```



Start here!

Let's review some concepts

What are some non-computing examples of recursion?

What does recursion allow us to do?

Give at least two components of a recursive function.

When a recursive function calls itself, what must we make sure of?

What happens if a recursive function doesn't have a base case?