Recursion

Draw tree, 2 branches at 45o angles, at evey segnemntS

Recursive function=function that is defined in terms of itself,

Or

A function that calls itself

Seems like an endlessneed to incorporate basecase to end loop

First try::

Depth = one = first trunk

Depth = zero = draw nothing

Depth = 2, branch

Def drawTree(size)+

Fd(size)

Def drawTree2(size):

Fd(size)

Lt(45)

Fd(size/2)

Fd(-size/2)

Rt(90)

fd(size/2)

Fd(-sizs/2)

Lt(45)

fd(-size)

Def drawTree3(size):

Fd(size)

ft(45)

drawTree2(size/2)

rt(90)

drawtree2(size/2)

lt(45)

bod(9)

Def main():

Size = 100

depth = 5

lt(90)

drawTreeLarger(size, depth)

“””

main()

single function

refi]]

“””

% is modulo, what is remainder

Recursive fx calls itself inside itself

def drawTree(size, depth)

if (depth == zero): ----#-basecase

pass

else

fd(sizde)

fd(90)

drawTreee(size\2, depth – 1)

rt(90)

drawTree(size/2, depth-1)

lt(45)

fd(-size)

needs basecase, otherwise it runs perpetually

we could use size as a determining factor

we could use depth

Turtle.setup( w, h)

80 \* numchars, 80 \* numlines 160x160

Turtle.setworld coordinates(11x, 11y, urx, ury) each letter gets 30 units

Goes from -4 up to 64 in both x and y

Each letter assumes drawing from ll corner

Scale: main()

Scale = 2

Or

Scale=input(…)

Needs to be sent to all functions

First to initcanvas

Coordinates \* scale too

Letters: drawletter( …,..,..,,(scale))

Or

Coordinate system already scales drawin letters to fit canvas

Parameter as an argument to function

Window, coordinates, letters

“execution trace:

DrawTree (size: = 100, depth: = 2)

Depth!=0

Fd(100,left(45)

drawTree(size = 50, depth)

depth !=50

fd(50)

lt(45)

drawTree(size :=25, depth:=0)

rt(90)

drawTree(size:=25, depth:=0)

depth==0

lt(45)

fd(-50)

rt(90)

drawTree(size:=50, depth:=1)

lt(45)

fd(100)

“””indicate what the function does, also add parameter indications, pre-conditions and post-conditions

Fruitful functions

Def addOne(x):

Return x+1

Factorial, fibonachi

N!=n(n-1)(n-2)\*….\*1

N!=n\*(n-1)!

1!=1

Or (basecases)

0!=1

Def fact(n):

If(n==0):

Return n

Else:

Return(n\*fact(n-1))

Substitution trace

If we want !3

Then we want to trace

Factorial 3 = 3\*fact(2)

=

=3(2\*fact(1)

=3(2\*fact0

=3(2\*1\*1=6

Fibonacci #s

Sequence number starting iwith0,1 or 1,1

Fo = 0

F1=1

Fn=Fn-1 + Fn-2….

Returned vals: 0.1.1.2.3.5.8.13.21

Fib numbers: 1,2,3,4,5,6,7,8

Previous 2 added together

Def Fib(n):

If(n==0):

Return0

Elif(n==1):

Return 1

Else

Return (fib(n-1)+fib(n-2))

Subs. Trace

Fib(3)=fib(2)+fib(1)

(Fib(1) + fib(0)) +1

1 + 0 + 1 = 2