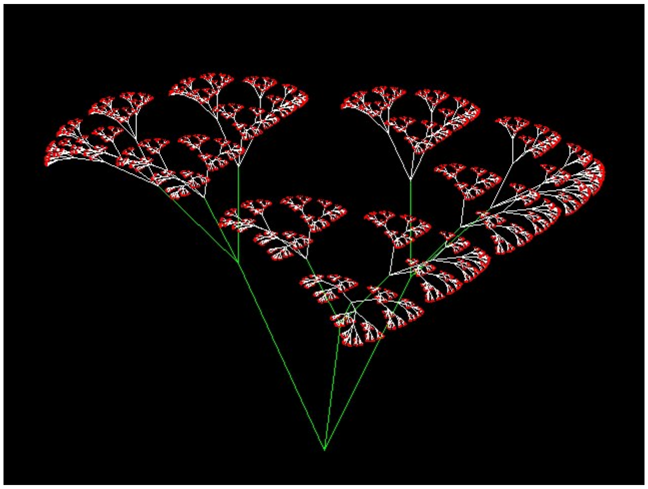
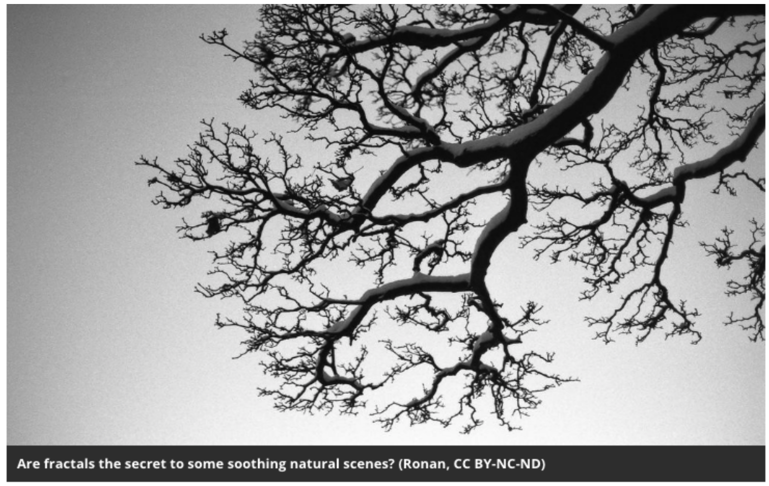
Lecture 14 - Recursion

Why Recursion?

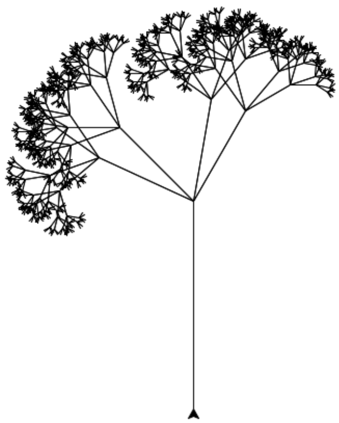



A fern repeats its pattern at various scales. (Michael , CC BY-NC)





Are fractals the secret to some soothing natural scenes? (Ronan, CC BY-NC-ND)



An example of a recursive addition

Let's define the sum of values from 0 to n as

```
sum(n) = { 0 if n <= 0
         { n + sum(n-1) if n > 0
```

Then we can build a function that matches this.

```
def recursive_sum ( n ):
    if n <= 0:
        return 0
    return n + recursive_sum(n-1)



# Automated Test
if __name__ == "__main__":
    n_err = 0
    x = recursive_sum ( 5 )
    if x !=  15:
        n_err = n_err + 1
        print ( "Error: Test 1: sum not working, expected {} got {}".format (  15, x ) )
    x = recursive_sum ( 0 )
    if x != 0:
        n_err = n_err + 1
        print ( "Error: Test 2: sum conversion not working, expected {} got {}".format ( 0, x ) )

    if n_err == 0 :
        print ( "PASS" )
    else:
        print ( "FAILED" )
```

What is a recursive function definition:

$$f(n) = \begin{cases} f(n-1) & n \geq 1 \\ 1 & n < 1 \end{cases}$$

For a positive integer:

```
n! = n * (n-1) * ... * 2 * 1
```

or

```
f(n) = n * (n-1) * ... * 2 * 1
```

or

```
f(n) = n * f(n-1)
```

or

```
f(n) = { n <= 1 : 1
       { n > 1  : n * f(n-1)
```

Now to Code:

```
def calc_factorial(x):
    # A recursive function to find the factorial of a number
    if x <= 1:
        return 1
    else:
        return (x * calc_factorial(x-1))

if __name__ == "__main__":
    num = 5
    print("The factorial of", num, "is", calc_factorial(num))

    err = False
    v = calc_factorial(num)
    if v != 120:
        err = True
        print ( "Incorrect result: {n}! Expected {good} got {bad}".format(n=num, good=120, bad=v))

    if not err :
        print ( "PASS" )
    else :
        print ( "FAIL" )
```

Compare to an iterative version:

```
def factorial_iterative(x):
    if x <= 1:
        return 1
    nn = 2
    rv = 1
    while ( nn <= x ):
        rv = rv * nn
    return rv

if __name__ == "__main__":
    num = 5
    print("The factorial of", num, "is", factorial_iterative(num))

    err = False
    v = factorial_iterative(num)
    if v != 120:
        err = True
        print ( "Incorrect result: {n}! Expected {good} got {bad}".format(n=num, good=120, bad=v))

    if not err :
        print ( "PASS" )
    else :
        print ( "FAIL" )
```

Another Example

With a function like:

$$1+\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\frac{1}{5}+\cdots$$

We can build a recursive soltion:

```
def harmonic_sum(n):
  if n < 2:
    return 1
  else:
    return 1 / n + (harmonic_sum(n - 1))

nL = [ 4, 7, 11, 18 ]
for n in nL:
    print("harmonic sum for {} = {}".format(n,harmonic_sum(n)))
```

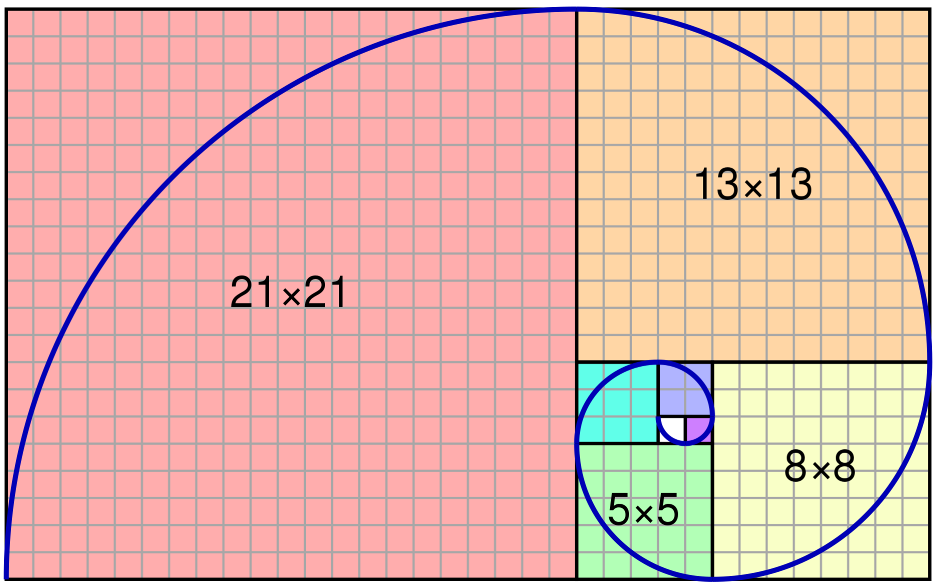
```
def harmonic_sum(n):
    hs = 0.0
    for i in range(n):
        hs = hs + 1/(i+1)
    return hs

nL = [ 4, 7, 11, 18 ]
for n in nL:
    print("harmonic sum for {} is {}".format(n, harmonic_sum(n)))

```

A better example is a fractal tree:

Fibonacci Numbers




```
fib(n) = { 0 : n = 0
         { 1 : n = 1
         { fib(n-1) + fib(n-2)
```

Weed

```
import turtle

def tree(length,n):
    if length < (length/n):
           return
    turtle.forward(length)
    turtle.left(45)
    tree(length * 0.5,length/n)
    turtle.left(20)
    tree(length * 0.5,length/n)
    turtle.right(75)
    tree(length * 0.5,length/n)
    turtle.right(20)
    tree(length * 0.5,length/n)
    turtle.left(30)
    turtle.backward(length)
    return

turtle.left(90)
turtle.backward(30)
tree(200,4)

input("Press Enter to continue...")
```

The Koch curve.

So a program to run the Koch curve:

```
# Python program to print complete Koch Curve.
from turtle import *

# function to create koch snowflake or koch curve
def snowflake(lengthSide, levels):
    if levels == 0:
        forward(lengthSide)
        return
    lengthSide /= 3.0
    snowflake(lengthSide, levels-1)
    left(60)
    snowflake(lengthSide, levels-1)
    right(120)
    snowflake(lengthSide, levels-1)
    left(60)
    snowflake(lengthSide, levels-1)

# main function
if __name__ == "__main__":
    # defining the speed of the turtle
    speed(0)
    length = 300.0

    # Pull the pen up  no drawing when moving.
    # Move the turtle backward by distance, opposite
    # to the direction the turtle is headed.
    # Do not change the turtles heading.
    penup()

    backward(length/2.0)

    # Pull the pen down  drawing when moving.
    pendown()
    for i in range(3):
        snowflake(length, 4)
        right(120)

    # To control the closing windows of the turtle
    mainloop()
```