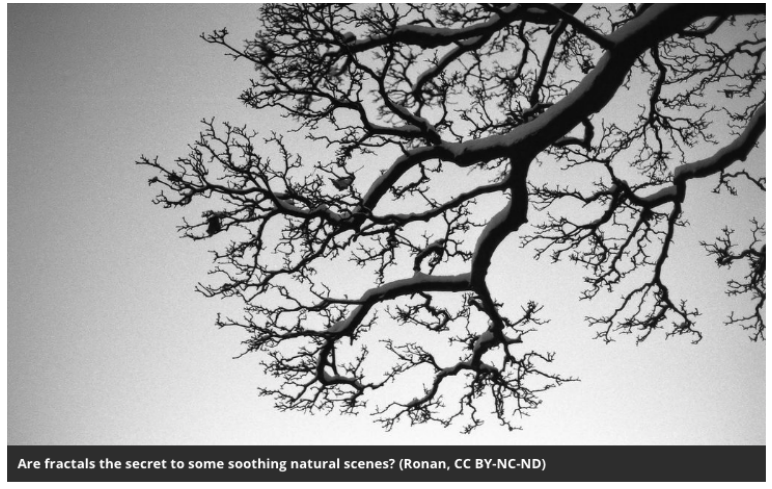
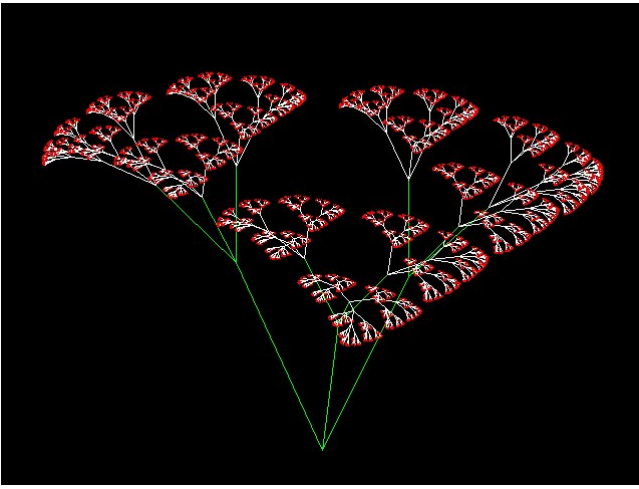


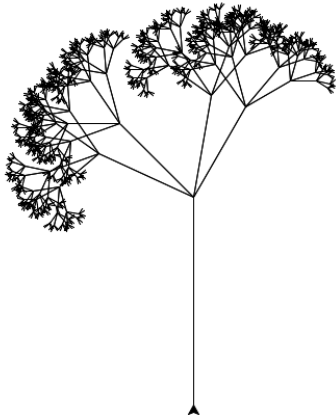
# Lecture 19 - Recursion

Why Recursion?





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

$$\text{sum}(n) = \begin{cases} 0 & \text{if } n \leq 0 \\ n + \text{sum}(n-1) & \text{if } n > 0 \end{cases}$$

Then we can build a function that matches this.

```

1: def recursive_sum ( n ):
2:     if n <= 0:
3:         return 0
4:     return n + recursive_sum(n-1)
5:
6:
7:
8: # Automated Test
9: if __name__ == "__main__":
10:     n_err = 0
11:     x = recursive_sum ( 5 )
12:     if x != 15:
13:         n_err = n_err + 1
14:         print ( "Error: Test 1: sum not working, expected {} got {}".format ( 15, x ) )
15:     x = recursive_sum ( 0 )
16:     if x != 0:
17:         n_err = n_err + 1
18:         print ( "Error: Test 2: sum conversion not working, expected {} got {}".format ( 0, x ) )
19:
20:     if n_err == 0 :
21:         print ( "PASS" )
22:     else:
23:         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:

$$n! = n * (n-1) * \dots * 2 * 1$$

or

$$f(n) = n * (n-1) * \dots * 2 * 1$$

or

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

or

$$f(n) = \begin{cases} n \leq 1 : 1 \\ n > 1 : n * f(n-1) \end{cases}$$

Now to Code:

```

1: def calc_factorial(x):
2:     # A recursive function to find the factorial of a number
3:     if x <= 1:
4:         return 1
5:     else:
6:         return (x * calc_factorial(x-1))
7:
8: if __name__ == "__main__":
9:     num = 5
10:    print("The factorial of", num, "is", calc_factorial(num))
11:
12:    err = False
13:    v = calc_factorial(num)
14:    if v != 120:
15:        err = True
16:        print ( "Incorrect result: {n}! Expected {good} got {bad}".format(n=num, good=120, bad=v))
17:
18:    if not err :
19:        print ( "PASS" )
20:    else :
21:        print ( "FAIL" )

```

Compare to an iterative version:

```

1: def factorial_iterative(x):
2:     if x <= 1:
3:         return 1
4:     nn = 2
5:     rv = 1
6:     while ( nn <= x ):
7:         rv = rv * nn
8:     return rv
9:
10: if __name__ == "__main__":
11:     num = 5
12:     print("The factorial of", num, "is", factorial_iterative(num))
13:
14:     err = False
15:     v = factorial_iterative(num)
16:     if v != 120:
17:         err = True
18:         print ( "Incorrect result: {n}! Expected {good} got {bad}".format(n=num, good=120, bad=v))
19:
20:     if not err :
21:         print ( "PASS" )
22:     else :
23:         print ( "FAIL" )

```

## Another Example

---

With a function like:

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

We can build a recursive soltion:

```
1: def harmonic_sum(n):
2:     if n < 2:
3:         return 1
4:     else:
5:         return 1 / n + (harmonic_sum(n - 1))
6:
7: nL = [ 4, 7, 11, 18 ]
8: for n in nL:
9:     print("harmonic sum for {} = {}".format(n,harmonic_sum(n)))
```

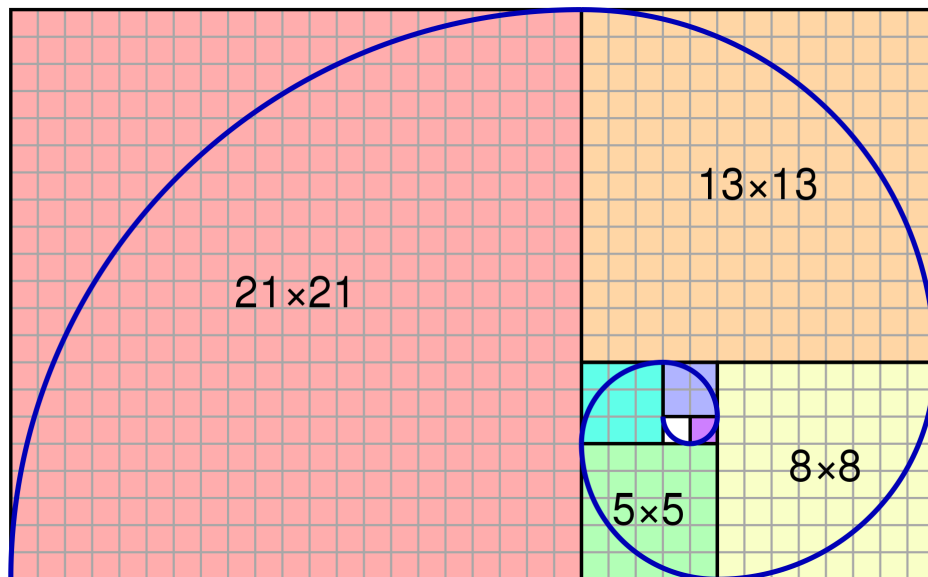
```
1: def harmonic_sum(n):
2:     hs = 0.0
3:     for i in range(n):
4:         hs = hs + 1/(i+1)
5:     return hs
6:
7: nL = [ 4, 7, 11, 18 ]
8: for n in nL:
9:     print("harmonic sum for {} is {}".format(n, harmonic_sum(n)))
10:
```

A better example is a fractal tree:

## Fibonacci Numbers

---





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

## Weed

---

```
1: import turtle
2:
3: def tree(length,n):
4:     if length < (length/n):
5:         return
6:     turtle.forward(length)
7:     turtle.left(45)
8:     tree(length * 0.5,length/n)
9:     turtle.left(20)
10:    tree(length * 0.5,length/n)
11:    turtle.right(75)
12:    tree(length * 0.5,length/n)
13:    turtle.right(20)
14:    tree(length * 0.5,length/n)
15:    turtle.left(30)
16:    turtle.backward(length)
17:    return
18:
19: turtle.left(90)
20: turtle.backward(30)
21: tree(200,4)
22:
23: input("Press Enter to continue...")
```

# The Koch curve.

---

So a program to run the Koch curve:

```
1: # Python program to print complete Koch Curve.
2: from turtle import *
3:
4: # function to create koch snowflake or koch curve
5: def snowflake(lengthSide, levels):
6:     if levels == 0:
7:         forward(lengthSide)
8:         return
9:     lengthSide /= 3.0
10:    snowflake(lengthSide, levels-1)
11:    left(60)
12:    snowflake(lengthSide, levels-1)
13:    right(120)
14:    snowflake(lengthSide, levels-1)
15:    left(60)
16:    snowflake(lengthSide, levels-1)
17:
18: # main function
19: if __name__ == "__main__":
20:     # defining the speed of the turtle
21:     speed(0)
22:     length = 300.0
23:
24:     # Pull the pen up no drawing when moving.
25:     # Move the turtle backward by distance, opposite
26:     # to the direction the turtle is headed.
27:     # Do not change the turtles heading.
28:     penup()
29:
30:     backward(length/2.0)
31:
32:     # Pull the pen down drawing when moving.
33:     pendown()
34:     for i in range(3):
35:         snowflake(length, 4)
36:         right(120)
37:
38:     # To control the closing windows of the turtle
39:     mainloop()
```