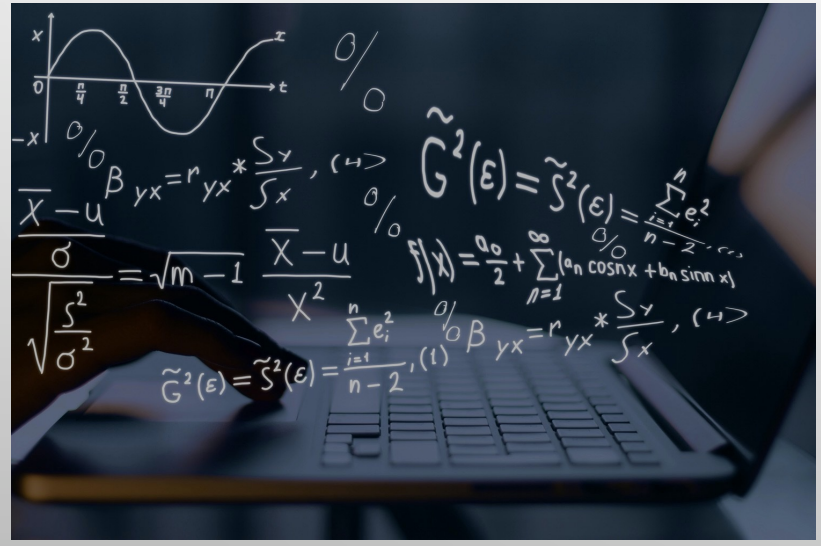
THE COLLEGE OF WOOSTER

Scientific Computing
CS 100



Fall Semester, 2021

TOPIC: RECURSION

OUTLINE

- Definition of recursion
- Parts of a recursive algorithm
- Why use recursion?
- Writing Recursive Programs
  - Recursive square – nested boxes
  - Recursive count of items in a list
  - Factorial Function
  - Palindrome
  - Drawing recursive tree
  - Drawing a Sierpinski triangle

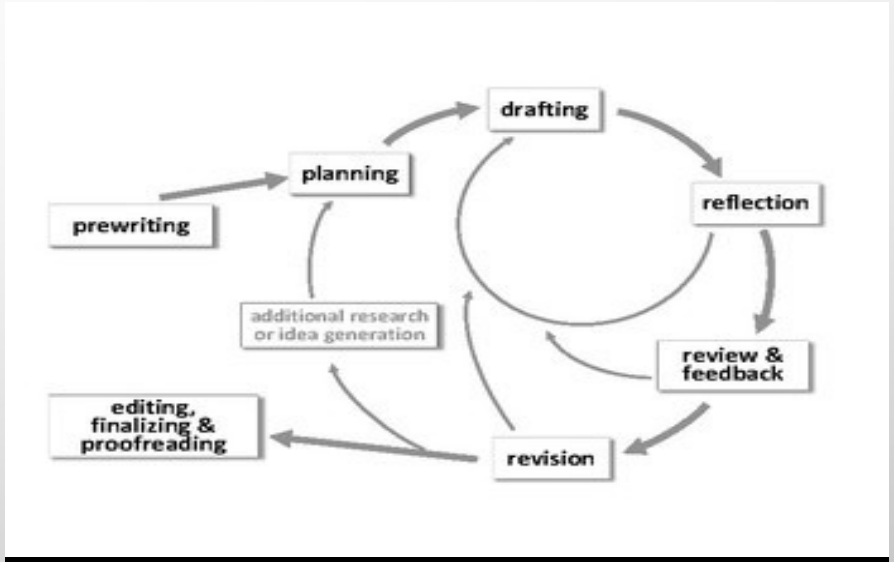
Definition of recursion

Recursion is the process of defining a problem (or the solution to a problem) in terms of (a simpler version of) itself.

Generally speaking, recursion is the concept of *well-defined* **self-reference**. It is the determination of a succession of elements by operating on one or more preceding elements according to a rule or a formula involving a finite number of steps.

Example of recursive process: -Writing

Humans indulges in recursive process during writing



The steps involve in the recursive process.

Parts of a recursive algorithm

All recursive algorithms must have the following:

1. Base Case (i.e., when to stop)
2. Work toward Base Cases
3. Recursive Call (i.e., call self)

Erroneous Recursive algorithm

A recursive algorithm that has no terminating condition(Base Case) is very dangerous and viewed to be ill-constructed or erroneous. And, such a situation leads to a program crash.

What is wrong the program code below?

```
def hell():
    print("Hello Word")
    hello()
```

Other problems with recursive algorithm.

1. **Stack Overflow**
2. **Memory Error**

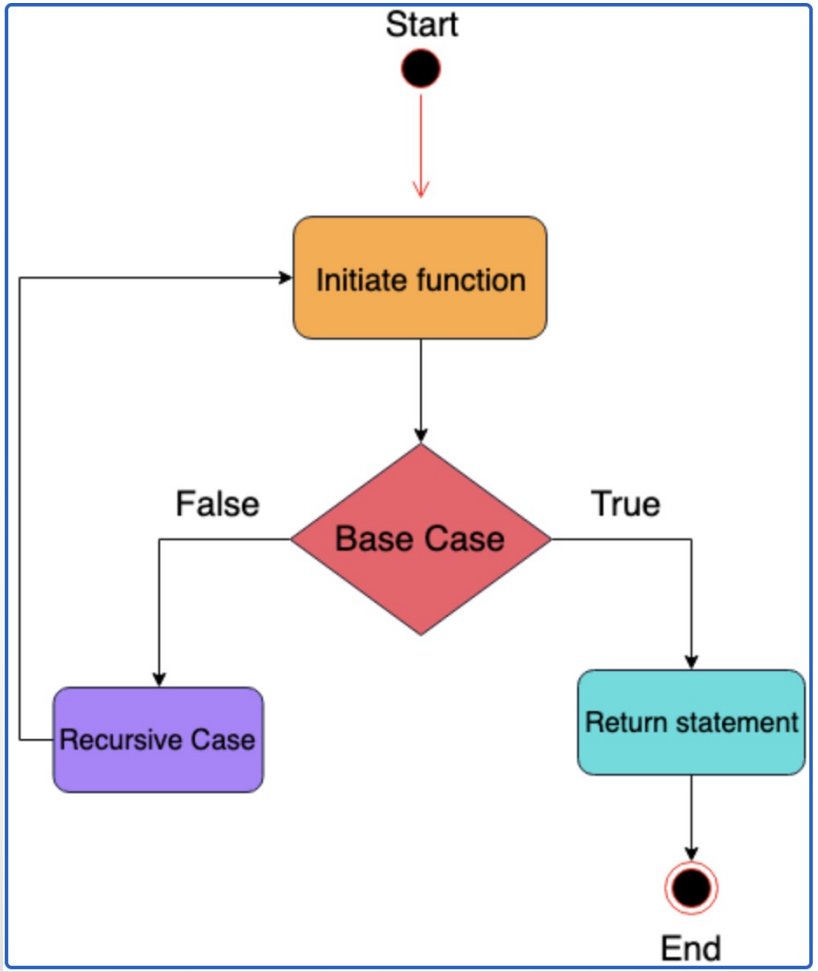


Fig 1.: Flow diagram of a recursive algorithm.

Why Use Recursion?

- Recursion is preferred when the problem can be broken down into smaller, repetitive tasks. These are the advantages of using recursion:

1. Complex tasks can be broken down into simpler problems.
2. Code using recursion is usually shorter and more elegant.
3. Sequence generation is cleaner with recursion than with iteration.

Writing Recursive Programs

At this stage and subsequently, we will illustrate how to write a recursive program that will solve the following tasks outlined below.

- Drawing nested boxes
- Count of items in a list
- Factorial Function
- Palindrome
- Drawing recursive tree
- Drawing a Sierpinski triangle

Drawing nested boxes

The objective of this task is to draw a series of squares using a recursive function. Figure 2 is a sample of what the output is ought to be.

Output

Fig. 2: Expected output

Hands-on Code

```
1   import turtle as aTurtle
2   def drawSquare(aTurtle, side):
3       for i in range(4):
4           aTurtle.forward(side)
5           aTurtle.right(90)
6
7   def nestedBox(aTurtle, side):
8       if side>=1:
9           drawSquare(aTurtle, side)
10          nestedBox(aTurtle, side-5)
11
12  nestedBox(aTurtle,150)
```

Additional Exercise

1. What statement is the base case?
2. Modify the hands-on code by reversing lines 8 and 10. Can you explain the different behavior?
3. Rewrite or modify the code to draw nested boxes, where each box is centered at the same point.

Count of items in a list

The objective of this task is to compute the number of elements in a list using a recursive call approach. Figure 3 is a diagram depicting the conceptual representation of a list.

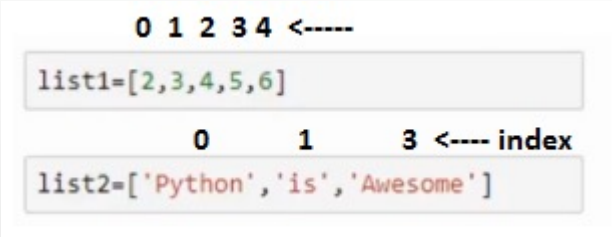


Fig. 3: Items in a list

Hands-on Code

```
def countList(aList):
    if aList==[]:  #base case, empty list
        return 0
    else:
        return 1+ countList(aList[1:])

aList=[2,3,4,5,6]
print('No of items in the list is:', countList(aList))
```

Output

```
No of items in the list is: 5
```

Additional Exercise

1. Write a recursive function to compute the sum of all the numbers in a list
2. Write a recursive function to find the minimum number in a list
3. Write a recursive function to find the maximum number in a list.
4. Write a recursive function to reverse the characters in a string

Factorial Function

A factorial function is defined as:

$$\mathbf{fact(n)} = \begin{cases} \boldsymbol{n * fact(n-1)} & \boldsymbol{n > 0} \\ \mathbf{1} & \boldsymbol{otherwise} \end{cases}$$

Our task is to implement this function a recursive call approach.

Hands-on Code

1

```
# Factorial of a number using recursion

def recur_factorial(n):
   if n == 1:
       return n
   else:
       return n*recur_factorial(n-1)

num = int(input("Enter a number>"))
```

2

```
# check if the number is negative


if num < 0:
   print("Sorry, factorial does not exist for negative
numbers")
elif num == 0:
   print("The factorial of 0 is 1")
else:
   print("The factorial of", num, "is", recur_factorial(num))
```

Output

```
Enter a number>4
The factorial of 4 is 24
```

Palindrome Function

A palindrome is a word or a phrase that is the same whether you read it backward or forward, for example, the word: refer, madam,

Hands-on Code

1

```
import string

def is_palindrome(phrase):
    # base case
    if len(phrase) < 2:
        return True

    # divide into easier calculation
    # and recursive operation
    return phrase[0] == phrase[-1] and \
        is_palindrome(phrase[1:len(phrase) - 1])
```

2

```
text="civic"
text_is_a_palindrome = is_palindrome(text)

if text_is_a_palindrome:
        print ("'{}' is a palindrome.".format(text))
else:
        print( "'{}' is not a palindrome.".format(text))
```

Output

```
'mum' is a palindrome.

'civic' is a palindrome.
```

Drawing recursive tree

This tree is drawn using a simple recursive instructions as below:

Steps:

1. Draw a trunk of n units long
2. Turn to the right 30 deg., and draw another tree with a trunk n-15 units long
3. Turn to the left 60 deg., and draw another tree with a trunk n-15 units long

Below is the program code that implements our recursive instructions(steps 1-3)

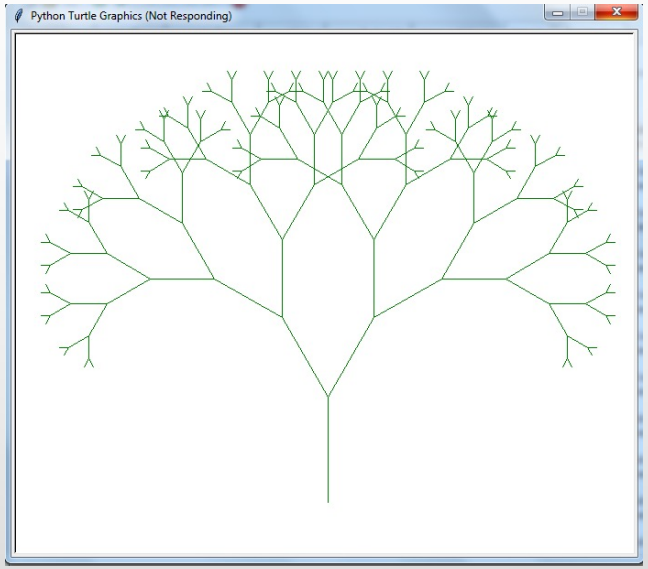
Hands-on Code

```
import turtle
t = turtle.Turtle()
def tree(t, trunkLength):
        if trunkLength < 5: # check  for base case
                return
        else:
                t.forward(trunkLength)
                t.right(30)
                tree(t, trunkLength-15)
                t.left(60)
                tree(t, trunkLength-15)
                t.right(30)
                t.backward(trunkLength)
t.up()
t.goto(0, -225)
t.down()
t.color("green", "green")
t.left(90)  # face up
tree(t, 115)
t.hideturtle()
```

Additional Exercise

1. Rewrite the tree function using the conditional truckLength >= 5 to check for the base case
2. Swap the rules for the trees so that it draws the left side of the tree before the right side
3. Randomize the turning angle to be between 15 and 45 degree. See hint on randomization at https://pynative.com/python-random-randrange/
4. Instead of always subtracting by 15, try subtracting a random amount between 5 and 25
5. Make the large branches brown and the small branches green(Hint: choose a threshold value for the length of the trunk and set the color accordingly.

Output



Drawing a Sierpinski triangle

The Sierpinski triangle is **a self-similar fractal**. It consists of an equilateral triangle, with smaller equilateral triangles recursively drawn on its remaining area. See figure 3 as example of Sierpinski triangle.

Our task is to implement a recursive Sierpinski function that we accept 3 points as parameter to draw a triangle and subdivide the triangle using the following rules:

1. For each of the corners of the larger triangle, create a small triangle using the corner and the point halfway between the given corner and the other two corners.
2. Specify the variable "depth" to indicate the number of times the original triangle will be sub-divided. The depth is reduced by one(1) whenever a triangle undergoes recursive division.
3. We calculate the mid-point of a line using the equations:

$$m_x = \frac{X_1 + X_2}{2} \text{ and } m_y = \frac{Y_1 + Y_2}{2}$$

Output

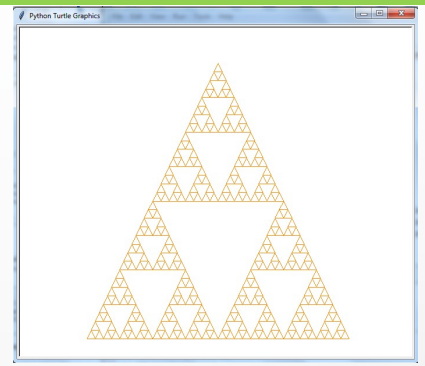


Fig. 3: A Sierpinski triangle

Hands-on Code

1

```
import turtle
def drawTriangle(t, p1, p2, p3):
    t.up()
    t.goto(p1)
    t.down()
    t.goto(p2)
    t.goto(p3)
    t.goto(p1)

def midPoint(p1, p2):
    return ((p1[0] +p2[0])/2.0, (p1[1]+p2[1])/2.0)

def sierpinski(t, p1, p2, p3, depth):
    if depth>0:
        sierpinski(t, p1, midPoint(p1, p2), midPoint(p1, p3), depth-1)
```

2

```
        sierpinski(t, p2, midPoint(p2, p3), midPoint(p2, p1), depth-1)
        sierpinski(t, p3, midPoint(p3, p1), midPoint(p3, p2), depth-1)
 else: # base case
        drawTriangle(t, p1, p2, p3)

t=turtle.Turtle()
t.color('darkorange')
sierpinski(t, [-225, -250], [225, -250], [0, 225] , 5)
t.hideturtle()
```

References

- *Python Programming in Context*, 3rd edition, B. Miller, D. Ranum, & J. Anderson
- Recursion: https://www.cs.utah.edu/~germain/PPS/Topics/recursion.html
- Understanding Recursion With Examples: https://betterprogramming.pub/understanding-recursion-with-examples-f74606fd6be0

Extra Example on Drawing Fractal Tree

Code

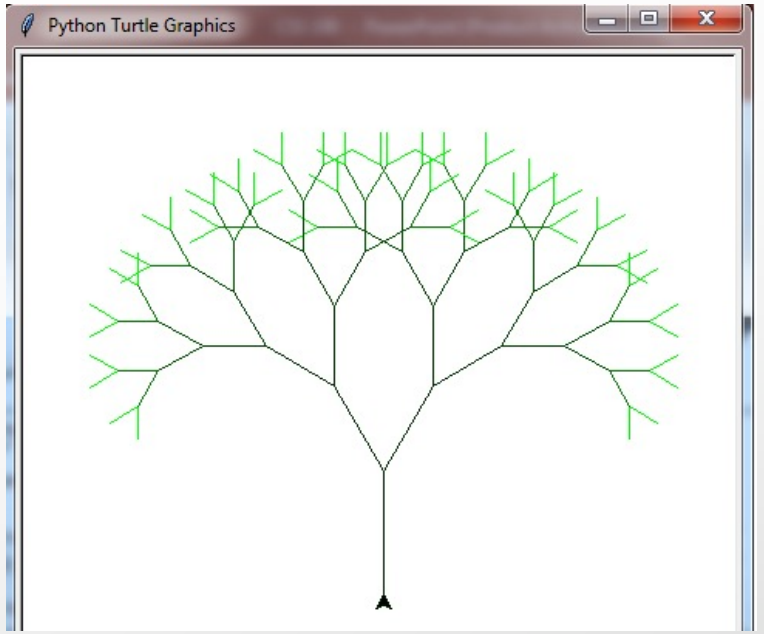
1

```
from turtle import *

speed('fastest')

# turning the turtle to face upwards
rt(-90)

# the acute angle between
# the base and branch of the Y
angle = 30

# function to plot a Y
def y(sz, level):

        if level > 0:
                colormode(255)
                # splitting the rgb range for green
                # into equal intervals for each level
                # setting the colour according
                # to the current level
                pencolor(0, 255//level, 0)

                # drawing the base
                fd(sz)
```

2

```
                rt(angle)

                # recursive call for
                # the right subtree
                y(0.8 * sz, level-1)

                pencolor(0, 255//level, 0)

                lt( 2 * angle )

                # recursive call for
                # the left subtree
                y(0.8 * sz, level-1)

                pencolor(0, 255//level, 0)

                rt(angle)
                fd(-sz)

# tree of size 80 and level 7
y(80, 7)
```

Output



Output from the execution the code

The code used the turtle library to implement the drawing of the tree and for more information about its usage, visit the link below:
https://docs.python.org/3/library/turtle.html