Sign-in:
cadtx.pw/week9

Week 9

Intro to Python

Objectives

Search and sort lists

Introduce recursion

Create a few recursive functions

Solve a few problems using recursion

Searching lists

Built-in: *lst.index(key)*

Linear Search: Iterates through the list using a for loop to find the location of the desired value.

Note: There are other searching algorithms (e.g. binary search).

Write a function that takes in a list and a value, *linearSearch(lst, key)*, that returns the location, *i*, of the first appearance of the *key*. If the value isn't in the list return -1.

Sorting Lists

Built-in: *lst.sort()*

```
names = ['Eduardo', 'Anirudh', 'Dan', 'Alex']
names.sort()
print(names)
```

Note: Other sorting algorithms can be better (take Advanced Python next semester to learn more!)

Recursive Functions

Recursive functions invoke themselves to solve a problem by solving a simpler problem.

Three necessary parts:

1) Function header
2) Base case (stopping condition)
3) Recursive call to simpler subproblem

```
def summer(n):                          # function header
    if(n==0):                           # base case
        return 0
    else:                               # recursive call
        return n + recurse(n-1)         # recurse(n-1) = subproblem
```

Recursion vs. Loops

Many recursion problems (e.g. factorials) can also be solved by loops.

In these cases, loops are “better” as they use less memory and are faster.

However, in many other problems (e.g. Fibonacci numbers), solving with loops can be far more difficult and maybe even less efficient.

Example 1: Factorials

We are going to recursively compute the factorial of *n*.

5! (5 factorial) is computed this way:

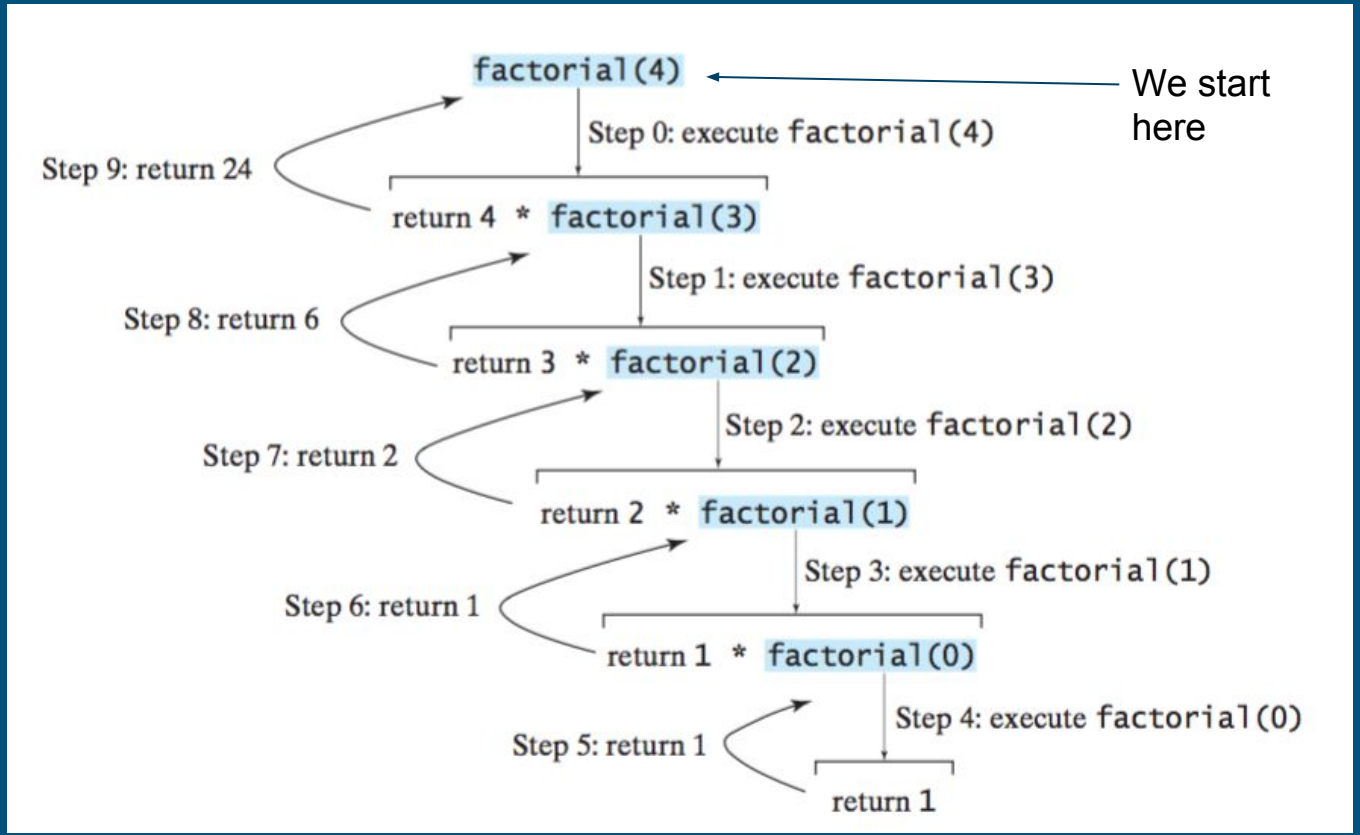
5*4*3*2*1 = 120

Instead of using a loop, we can create a function that calls on itself.

Loop example

```
def factorial(n):
    result = 1
    for i in range(n, 1, -1):
        result *= i
    return result
```

Visualization of previous example



Example 2: Fibonacci numbers

We are going to recursively compute the n^{th} Fibonacci number, f_n.

$f_n = f_{n-1} + f_{n-2}$, where $f_0 = 0$, $f_1 = 1$

Note: There is no intuitive way to accomplish this with a loop.

```
def fibonacci(n):
    if(n==0):
        return 0
    elif(n==1):
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)
```