
University of New Haven

University of New Haven

WELCOME
TO
Intro to Script Programming/Python

CSCI 6651-03 Spring 2022

Bibek Upadhayay

University of New Haven

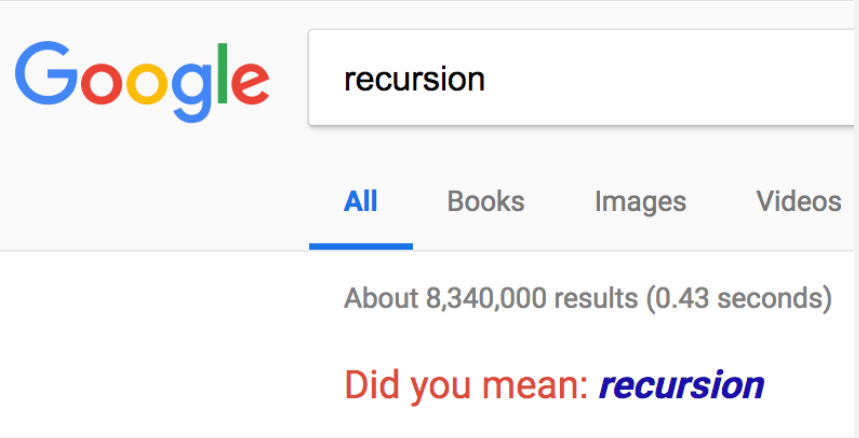
University of New Haven

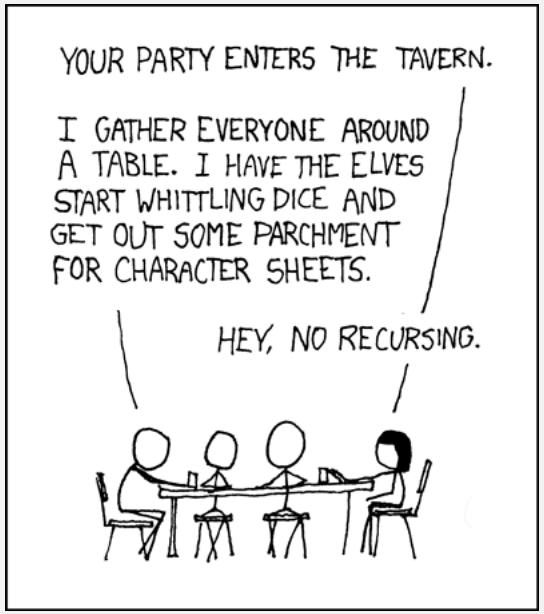
Recursive Function

INPUT AND OUTPUT WITH FILES

What is Recursion?

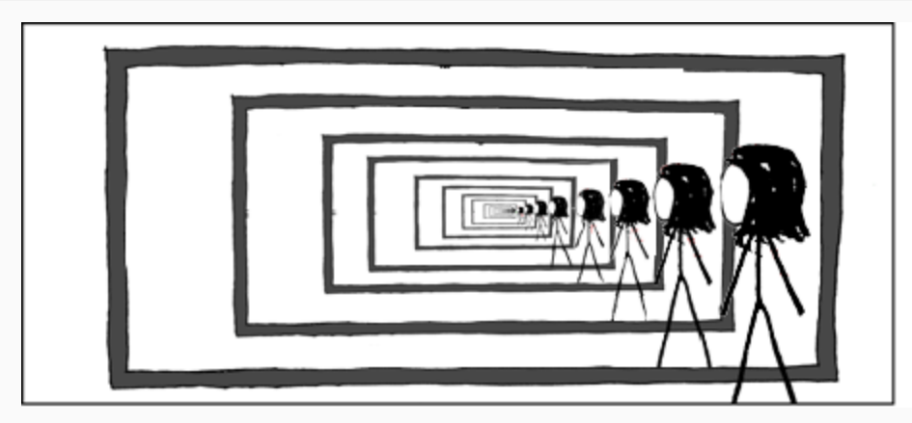
- To understand recursion, you must first understand recursion

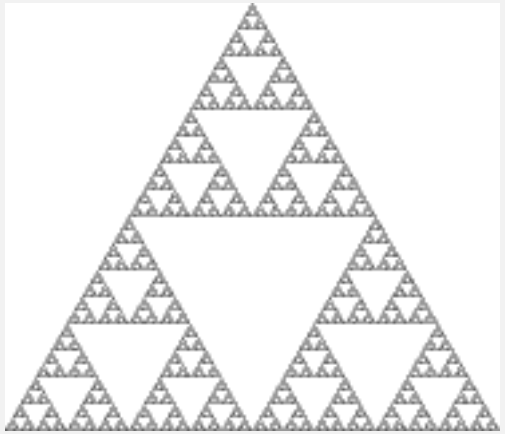




What is Recursion?

- Recursion is when something is defined in terms of itself. Think of self similarity
- For programming, it is when a solution to a problem depends on solutions to smaller instances of the same problem. More concretely, when a function's definition depends on subsequent calls to itself.





What is Recursion?

University of New Haven

- If we can think of a solution to a problem that involves solving a smaller sub problem, we can define a recursive solution. A recursive function is one where the function references itself in the body solution
- If we keep breaking the problem down into smaller and smaller pieces, eventually we will have a small enough problem where the solution is trivial
  - E.g. Sorting an array of size 1 is trivial... It is already sorted
  - We call this the **base case**
- Aside from the base case, we have the **recursive** or **general case**.
- A recursive algorithm expresses a solution in terms of itself

Computing N!

University of New Haven

- n! (factorial) is defined as the following:

$$n! = \begin{cases} 1, & \text{if } n = 0 \\ n * (n-1) * (n-2) * \cdots * 1, & \text{if } n > 0 \end{cases}$$

- So 4! = 4 * 3 * 2 * 1
  - Factorial is often used to compute **permutations**, or possible ordering of n items

$$n! = \begin{cases} 1, & \text{if } n = 0 \\ n * (n-1)! & \text{if } n > 0 \end{cases}$$

- The recursive definition of n! (self-referencing):

Computing N!

- What is our base case?
- What is our general case?

fact.py > factorial

```
1
2   def factorial(n):
3       if n==1:
4           return n
5       else:
6           return n*factorial(n-1)
7
8
```

How Recursion Works ?

University of New Haven

- Recursion relies on the idea of a call stack.
- When a function is invoked, need to keep track of some information
  - Parameters
  - Local variables
  - Return address (where to resume execution in the calling code)
- This is maintained in an **activation record** or **stack frame**

How Recursion Works

University of New Haven

- As functions call other functions, these activation records are stacked (top one being the current context). This is the **call stack**.
  - It's a stack because we only push context to the top, and when a function is done processing, we pop from the top and return context to the calling code.

Verification

University of New Haven

- Ask yourself three questions:
- The Base-Case Question:
  - Is there a non-recursive way out of the function, and does the routine work for this base case?
- The Smaller-Case Question:
  - Does each recursive call to the function involve a smaller case of the original problem, leading inescapably to the base case?
- The General-Case Question:
  - Assuming the recursive calls work correctly does the entire function work correctly?

Code

University of New Haven

```
1
2   def factorial(n):
3       if n==1:
4           return n
5       else:
6           return n*factorial(n-1)
7
8
9   n= 5
10
11  if n <0:
12      print("Please enter number greater than 0")
13  elif n==0:
14      print(" The factorial of 0 is 1")
15  else:
16      print("The factorial is: ", factorial(n))
```

University of New Haven

Thank you!