Another problem to solve…

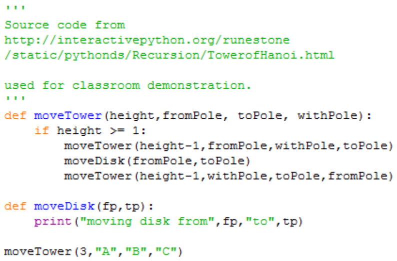
- We want to move a pile of discs of different size from one pole to another, with the condition that no larger disc can sit on top of a smaller disc!

href="https://commons.wikimedia.org/wiki/File:Tower_of_Hanoi.jpeg#/media/File:Tower_of_Hanoi.jpeg

Watch a video …

- https://en.wikipedia.org/wiki/Tower_of_Hanoi#/media/File:Tower_of_Hanoi_4.gif

A Python solution to this problem

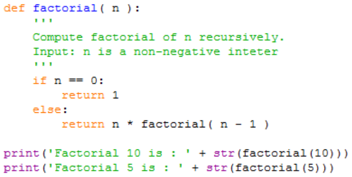


```
'''
Source code from
http://interactivepython.org/runestone
/static/pythonds/Recursion/TowerofHanoi.html

used for classroom demonstration.
'''
def moveTower(height,fromPole, toPole, withPole):
    if height >= 1:
        moveTower(height-1,fromPole,withPole,toPole)
        moveDisk(fromPole,toPole)
        moveTower(height-1,withPole,toPole,fromPole)

def moveDisk(fp,tp):
    print("moving disk from",fp,"to",tp)

moveTower(3,"A","B","C")
```

How to compute factorial ?

F(n) = n * F(n-1) for n > 0

F(0) = 1



```
def factorial( n ):
    '''
    Compute factorial of n recursively.
    Input: n is a non-negative inteter
    '''
    if n == 0:
        return 1
    else:
        return n * factorial( n - 1 )

print('Factorial 10 is : ' + str(factorial(10)))
print('Factorial 5 is : ' + str(factorial(5)))
```

These types of problems lead to a category of solution called *recursion*!

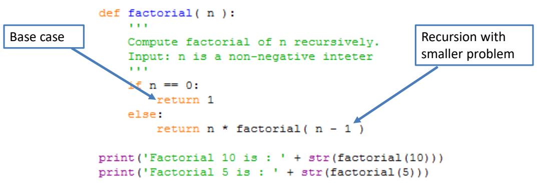
While some examples of recursion may be more complicated than the level of CSCI 203, we will learn the basic strategy in this class. Recursion will be studied in more depth in other CS courses.

Two important features of a recursive solution

- A recursive solution must have one or more base cases (when to stop)
  - E.g., factorial(0) == 1
- A recursive solution can be expressed in the exact same solution with a smaller problem size
  - E.g., factorial(n) = n * factorial(n-1)

In the case of computing factorial

F(n) = n * F(n-1) for n > 0
F(0) = 1

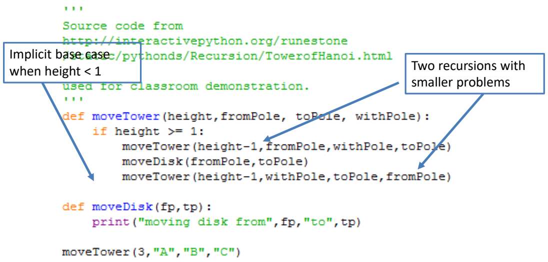


Base case

Recursion with smaller problem

```
def factorial( n ):
    '''
    Compute factorial of n recursively.
    Input: n is a non-negative inteter
    '''
    if n == 0:
        return 1
    else:
        return n * factorial( n - 1 )

print('Factorial 10 is : ' + str(factorial(10)))
print('Factorial 5 is : ' + str(factorial(5)))
```

In the case of Tower of Hanoi

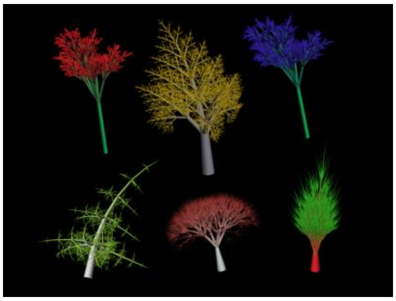


Implicit base case when height < 1

Two recursions with smaller problems

```
'''
Source code from
http://interactivepython.org/runestone
/static/pythonds/Recursion/TowerofHanoi.html

used for classroom demonstration.
'''
def moveTower(height,fromPole, toPole, withPole):
    if height >= 1:
        moveTower(height-1,fromPole,withPole,toPole)
        moveDisk(fromPole,toPole)
        moveTower(height-1,withPole,toPole,fromPole)

def moveDisk(fp,tp):
    print("moving disk from",fp,"to",tp)

moveTower(3,"A","B","C")
```

Recursion in nature



The key: self-similarity

Behind the curtain…

```
def fac(n):

    if n <= 1:
        return 1

    else:
        return n * fac(n-1)
```

```
>>> fac(1)
```

Result: 1

The base case is **No Problem**!

Behind the curtain…

```
def fac(n):

    if n <= 1:
        return 1

    else:
        return n * fac(n-1)
```

```
fac(5)
```

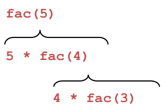
Behind the curtain…

```
def fac(n):

    if n <= 1:
        return 1

    else:
        return n * fac(n-1)
```

$$\overset{\text{fac(5)}}{\overbrace{\text{5 * fac(4)}}}$$

Behind the curtain…

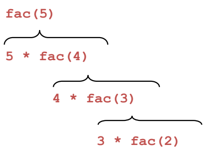
```
def fac(n):

    if n <= 1:
        return 1

    else:
        return n * fac(n-1)
```



```
fac(5)
5 * fac(4)
      4 * fac(3)
```

Behind the curtain…

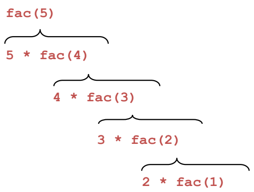
```
def fac(n):

    if n <= 1:
        return 1

    else:
        return n * fac(n-1)
```



```
fac(5)
5 * fac(4)
      4 * fac(3)
           3 * fac(2)
```

Behind the curtain…

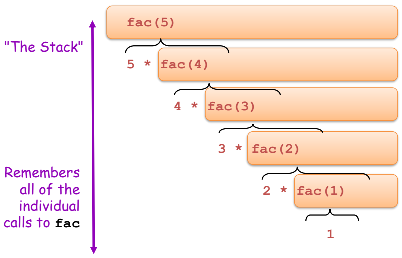
```
def fac(n):

    if n <= 1:
        return 1

    else:
        return n * fac(n-1)
```



```
fac(5)
5 * fac(4)
      4 * fac(3)
           3 * fac(2)
                2 * fac(1)
```

Behind the curtain…

```
def fac(n):

    if n <= 1:
        return 1

    else:
        return n * fac(n-1)
```



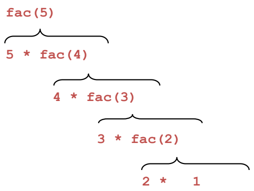
"The Stack"

Remembers all of the individual calls to `fac`

```
fac(5)
5 * fac(4)
      4 * fac(3)
           3 * fac(2)
                2 * fac(1)
                       1
```

Behind the curtain…

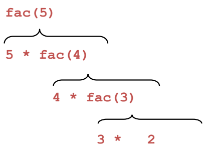
```
def fac(n):

    if n <= 1:
        return 1

    else:
        return n * fac(n-1)
```



```
fac(5)
5 * fac(4)
      4 * fac(3)
           3 * fac(2)
                2 *   1
```

Behind the curtain…

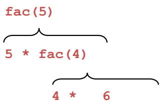
```
def fac(n):

    if n <= 1:
        return 1

    else:
        return n * fac(n-1)
```



```
fac(5)
5 * fac(4)
      4 * fac(3)
           3 *   2
```

Behind the curtain…

```
def fac(n):

    if n <= 1:
        return 1

    else:
        return n * fac(n-1)
```



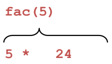
fac(5)

5 * fac(4)

4 *   6

Behind the curtain…

```
def fac(n):

    if n <= 1:
        return 1

    else:
        return n * fac(n-1)
```



fac(5)

5 *   24

Behind the curtain…

```
def fac(n):

    if n <= 1:
        return 1

    else:
        return n * fac(n-1)
```

fac(5)

Result: 120

Look familiar?

```
0 N*** -> X 1     Base Case
0 x*** -> N 0     Recursive Step
```

Let recursion do the work for you!

Exploit self-similarity
Produce short, elegant code } **Less work !**

```
def factorial(n):

    if n <= 1:
        return 1
    else:
        return n * factorial(n-1)
```

You handle the base case – the easiest possible case to think of!

Recursion does almost all of the rest of the problem!

Always a "smaller" problem!

Question: Sum

- How to modify factorial function to give us a sum of integers from 1 to n?

```
def factorial(n):

    if n <= 1:
        return 1
    else:
        return n * factorial(n-1)
```

Exercise 1

- Write a function called `my_len` that computes the length of a string
- Example:
  `my_len("aliens")` → 6
- What is the base case?
  Empty string
  `my_len('')` → 0
- Recursive call:
  `1 + my_len(s[1:])`

my_len

```
def my_len(s):
    """ input: any string, s
        output: the number of characters in s
    """
    if

    else:
```

my_len

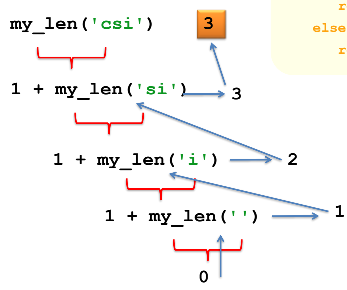
```
def my_len(s):
    """ input: any string, s
        output: the number of characters in s
    """
    if s == '':
        return 0
    else:
        rest = s[1:]
        return 1 + my_len( rest )
```

my_len

```
def my_len(s):
    """ input: any string, s
        output: the number of characters in s
    """
    if s == '':
        return 0
    else:
        return 1 + my_len(s[1:])
```

Behind the curtain:
how recursion works...

```
def my_len(s):
    if s == '':
        return 0
    else:
        return 1 + my_len(s[1:])
```



```
my_len('csi')              3

1 + my_len('si')      3

    1 + my_len('i')      2

        1 + my_len('')      1

                 0
```

MORE RECURSION!!

```
def sum_of_digits(s):
    """ input: a string of numbers -‘252674’
        output: the sum of the numbers 26
    """
     if

     else:
```