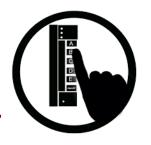


Lecture 14: More Recursion!

CS 1110

Introduction to Computing Using Python

How was A3?



- A. Didn't finish all 7 functions but got some of them done.
- B. Finished the 7 functions and tested using a3_test.py
- C. Tested all 7 and also got a3_scramble.py working.
- D. I was able to decode the secret message.
- E. Was? I'm still working on it.

Announcements

Recursion

Recursive Function:

A function that calls itself (directly or indirectly)

Recursive Definition:

A definition that is defined in terms of itself

From previous lecture: Factorial

Non-recursive definition:

$$n! = n \times n-1 \times ... \times 2 \times 1$$

= $n (n-1 \times ... \times 2 \times 1)$

Recursive definition:

$$n! = n (n-1)!$$
 for $n > 0$ Recursive case $0! = 1$ Base case

Recursive Call Frames

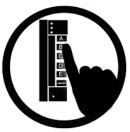
```
def factorial(n):
    """Returns: factorial of n.
    Precondition: n ≥ 0 an int"""
    if n == 0:
        return 1
        return n*factorial(n-1)
factorial(3)
```

Recursive Call Frames

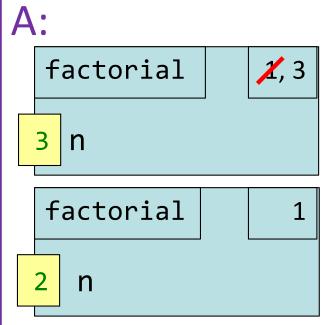
```
1, 3
def factorial(n):
                                     factorial
   """Returns: factorial of n.
                                       n
   Precondition: n ≥ 0 an int"""
   if n == 0:
      return 1
  return n*factorial(n-1)
factorial(3)
                           Now what?
```

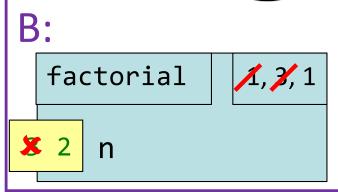
Each call is a new frame!

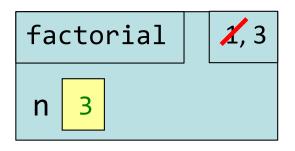
What happens next? (Q)

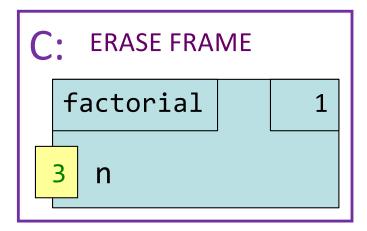


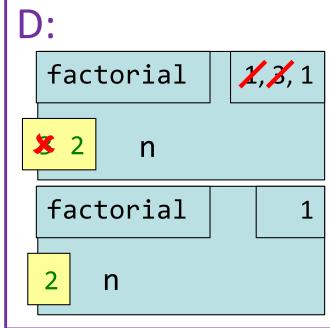
```
def factorial(n):
    """Returns: factorial o
    Precondition: n ≥ 0 an
    if n == 0:
        return 1
    return n*factorial(n-1)
```











Recursive Call Frames (n==2, execute line 1)

```
factorial
 def factorial(n):
                                                    1, 3
    """Returns: factorial of n.
                                        3
                                         n
   ▶Precondition: n ≥ 0 an int"""
   if n == 0:
                                        factorial
       return 1
                                        2 n
3
    return n*factorial(n-1)
```

factorial(3)

11

Recursive Call Frames (n==2, execute line 3)

```
def factorial(n):
    """Returns: factorial of n.
    Precondition: n ≥ 0 an int"""

if n == 0:
    return 1
    return n*factorial(n-1)
factorial ¼,3
factorial ¼,3
```

factorial(3)

Recursive Call Frames (n==1, execute line 1)

```
factorial
                                                     1, 3
 def factorial(n):
    """Returns: factorial of n.
                                        3
                                          n
   ▶Precondition: n ≥ 0 an int"""
   if n == 0:
                                        factorial
                                                     1, 3
       return 1
                                        2 n
3
    return n*factorial(n-1)
                                        factorial
 factorial(3)
                                          n
```

Recursive Call Frames (n==1, execute line 3)

```
factorial
                                                   1, 3
def factorial(n):
   """Returns: factorial of n.
                                       3
                                         n
   Precondition: n ≥ 0 an int"""
   if n == 0:
                                       factorial
                                                   1, 3
      return 1
                                       2 n
  return n*factorial(n-1)
                                       factorial
                                                   1, 3
factorial(3)
                                         n
```

Recursive Call Frames (n==0, execute line 1)

```
factorial
                                                     1, 3
 def factorial(n):
    """Returns: factorial of n.
                                          n
   ▶Precondition: n ≥ 0 an int"""
   if n == 0:
                                         factorial
                                                     1, 3
        return 1
                                         2 n
    return n*factorial(n-1)
3
                                         factorial
                                                     1, 3
 factorial(3)
                                           n
                                         factorial
                                           n
```

Recursive Call Frames (n==0, execute line 2)

```
factorial
 def factorial(n):
                                                      1, 3
    """Returns: factorial of n.
                                         3
                                           n
    Precondition: n ≥ 0 an int"""
    if n == 0:
                                                      1, 3
                                         factorial
      return 1
                                         2 n
3
    return n*factorial(n-1)
                                         factorial
                                                      1, 3
 factorial(3)
                                           n
                                                      1, 2
                                         factorial
                                           n
```

Recursive Call Frames (n==0, RETURN 1)

```
factorial
                                                     1, 3
 def factorial(n):
    """Returns: factorial of n.
                                         3
                                          n
    Precondition: n ≥ 0 an int"""
    if n == 0:
                                                     1, 3
                                         factorial
     →return 1
                                         2 n
    return n*factorial(n-1)
3
                                                     1, 3
                                         factorial
 factorial(3)
                                           n
                                         factorial
                                                 RETURN
```

Recursive Call Frames (n==1, finish line 3)

```
factorial
                                                      1, 3
 def factorial(n):
    """Returns: factorial of n.
                                           n
    Precondition: n ≥ 0 an int"""
    if n == 0:
                                         factorial
                                                      1, 3
                                         2 n
    return n*factorial(n-1)
3
                                                      1, 3
                                         factorial
 factorial(3)
                                           n
                                         factorial
                                                  RETURN
                                                            18
```

Recursive Call Frames (n==1, RETURN 1)

```
factorial
                                                     1, 3
 def factorial(n):
    """Returns: factorial of n.
                                          n
    Precondition: n ≥ 0 an int"""
    if n == 0:
                                                     1, 3
                                         factorial
        return 1
                                         2 n
3
    return n*factorial(n-1)
                                         factorial
 factorial(3)
                                                 RETURN
                                         factorial
                                                 RETURN
```

Recursive Call Frames (n==2, finish line 3)

```
factorial
                                                      1, 3
 def factorial(n):
    """Returns: factorial of n.
                                           n
    Precondition: n ≥ 0 an int"""
    if n == 0:
                                         factorial
                                                      1, 3
                                         2 n
    return n*factorial(n-1)
3
                                         factorial
 factorial(3)
                                                 RETURN
                                         factorial
                                                 RETURN
                                                            20
```

Recursive Call Frames (n==2, RETURN 6)

```
factorial
                                                     1, 3
 def factorial(n):
    """Returns: factorial of n.
                                           n
    Precondition: n ≥ 0 an int"""
    if n == 0:
                                         factorial
        return 1
                                                 RETURN
3
    return n*factorial(n-1)
                                         factorial
 factorial(3)
                                                 RETURN
                                         factorial
                                                 RETURN
                                                            21
```

Recursive Call Frames (n==3, finish line 3)

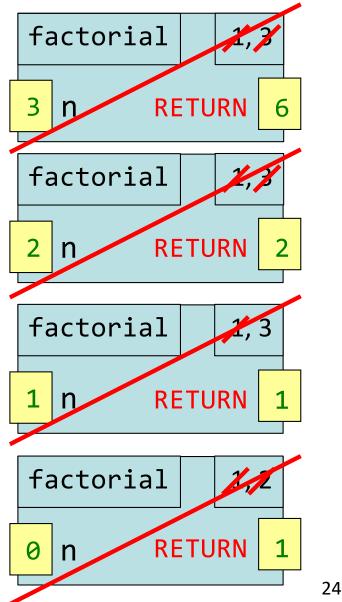
```
factorial
                                                     1, 3
 def factorial(n):
    """Returns: factorial of n.
                                          n
    Precondition: n ≥ 0 an int"""
    if n == 0:
                                         factorial
                                                 RETURN
    return n*factorial(n-1)
3
                                         factorial
 factorial(3)
                                                 RETURN
                                         factorial
                                                 RETURN
                                                            22
```

Recursive Call Frames (n==3, RETURN 6)

```
def factorial(n):
                                         factorial
                                                     1,3
    """Returns: factorial of n.
                                                 RETURN
    Precondition: n ≥ 0 an int"""
    if n == 0:
                                         factorial
        return 1
                                                 RETURN
3
    return n*factorial(n-1)
                                         factorial
 factorial(3)
                                                 RETURN
                                         factorial
                                                 RETURN
```

Recursive Call Frames (all calls complete!)

```
def factorial(n):
    """Returns: factorial of n.
    Precondition: n ≥ 0 an int"""
    if n == 0:
       return 1
3
    return n*factorial(n-1)
 factorial(3)
```

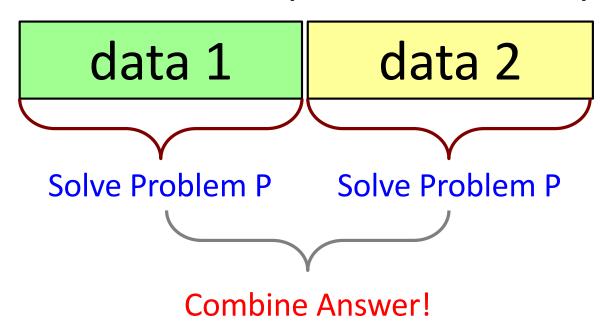


Divide and Conquer

Goal: Solve problem P on a piece of data

data

Idea: Split data into two parts and solve problem

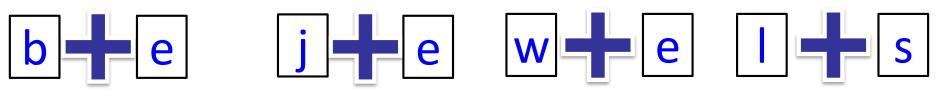


From Last Time: Divide and Conquer Example

Count the number of 'e's in a string:

bejewels 3

Watch in the Python Tutor



0 1 0 1 0 0 26

Example: Palindromes

Example:

AMANAPLANACANALPANAMA

MOM

A

- Dictionary definition: "a word that reads (spells) the same backward as forward"
- Can we define recursively?

Example: Palindromes

- Strings with <= 1 character are palindromes
- String with ≥ 2 characters is a palindrome if:
 - its first and last characters are equal, and
 - the rest of the characters form a palindrome
- Example:

 have to be the same

 AMANAPLANACANALPANAMA

 has to be a palindrome
- Implement:

```
def ispalindrome(s):
"""Returns: True if s is a palindrome""28
```

Example: Palindromes (1)

Strings with <= 1 character are palindromes String with ≥ 2 characters is a palindrome if:

its first and last characters are equal, and

the rest of the characters form a palindrome Definit

What is the simple case?

```
def ispalindrome(s):
   """Returns: True if s is a palindrome"""
```

Palindrome Base Case



```
What is the simple case?
  def ispalindrome(s):
     """Returns: True if s is a palindrome"
                     return True
             if len(s) == 1:
                     return True
             if len(s) == 2 \&\& s[0] == s[1]:
                     return True
             if len(s) < 2:
                     return True
```

E: I don't know

Example: Palindromes (1)

Strings with <= 1 character are palindromes String with ≥ 2 characters is a palindrome if:

```
    its first and last characters are equal, and
```

the rest of the characters form a palindrome pefinition

```
What is the simple case? What is the complex case? def ispalindrome(s):
```

```
"""Returns: True if s is a palindrome"""

if len(s) < 2:
    return True

Base case
```

```
endsAreSame = _____
middleIsPali = _____
return ____
```

Example: Palindromes (2)

Strings with <= 1 character are palindromes String with ≥ 2 characters is a palindrome if:

```
its first and last characters are equal, and
```

the rest of the characters form a palindrome pefinit

```
What is the simple case? What is the complex case?
  def ispalindrome(s):
```

```
"""Returns: True if s is a palindrome"""
if len(s) < 2:
                      Base case
       return True
                                      Recursive case
endsAreSame = s[0] == s[-1]
middleIsPali = ispalindrome(s[1:-1])
return endsAreSame and middleIsPali
```

Recursion and Objects

Class Person, 3 attributes

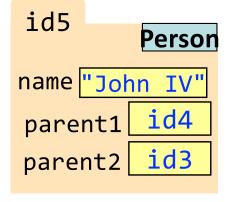
name: String

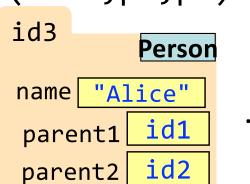
parent1: Person (or None)

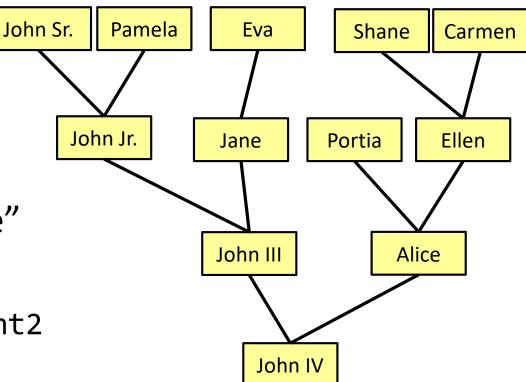
parent2: Person (or None)

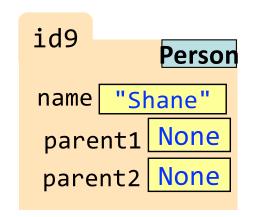
Represents the "family tree"

- Goes as far back as known
- Attributes parent1 and parent2 are None if not known
- Constructor: Person(name, p1, p2)









Recursion and Objects: Setup

```
def count_ancestors(p):
                                      John Sr.
                                               Pamela
                                                         Eva
                                                                         Carmen
                                                                  Shane
    """Returns: num of known ances
    Pre: p is a Person"""
    # 1. Handle base case.
                                            John Jr.
                                                                         Ellen
                                                       Jane
                                                                Portia
    # No parents (no ancestors)
                                                                     Alice
                                                       John III
    # 2. Break into two parts
    # Has parent1 or parent2
    # Count ancestors of each one
                                                          John IV
    # (plus parent1, parent2 themselves)
                                                                    11 ancestors
                                                         id5
                                                                  Person
                                                        name "John IV"
                                                                  id4
                                                         parent1
    # 3. Combine the result
                                                                           34
                                                                  id3
                                                         parent2
```

Recursion and Objects: Implementation

```
def count_ancestors(p):
                                             Pamela
                                     John Sr.
                                                       Eva
                                                                       Carmen
                                                                Shane
    """Returns: num of known ances
    Pre: p is a Person"""
    # 1. Handle base case.
                                          John Jr.
                                                                       Ellen
                                                      Jane
                                                              Portia
    # No parents (no ancestors)
    if p.parent1 == None and p.parent2 == None.
        return 0
                                                                   Alice
                                                     John III
    # 2. Break into two parts
    # Has parent1 or parent2
    # Count ancestors of each one
                                                        John IV
    # (plus parent1, parent2 themselves)
    parent1s_fam = 0
                                                                 11 ancestors
    if p.parent1 != None:
        parent1s_fam = 1 + count_ancestors(p.parent1)
                                                            id8
                                                                    Person
    parent2s_fam = 0
                                                           name "Jane"
    if p.parent2 != None:
        parent2s fam = 1 + count ancestors(p.parent2)
                                                           parent1 None
    # 3. Combine the result
                                                           parent2 id3
    return parent1s_fam + parent2s_fam
```

Recursion and Objects: Finishing Touches

```
def count_ancestors(p):
    """Returns: num of known ancestors
    Pre: p is a Person"""
    # 1. Handle base case.
    # No parents (no ancestors)
    if p.parent1 == None and p.parent2 == None:
        return 0
    # 2. Break into two parts
    # Has parent1 or parent2
    # Count ancestors of each one
    # (plus parent1, parent2 themselves)
    parent1s_fam = 0
    if p.parent1 != None:
        parent1s_fam = 1 + count_ancestors(p.parent1)
    parent2s_fam = 0
    if p.parent2 != None:
        parent2s fam = 1 + count_ancestors(p.parent2)
    # 3. Combine the result
    return parent1s_fam + parent2s_fam
```

We don't actually need this.
It is handled by the conditionals in #2.

"It Takes a Village" Version: Lots of Parents

```
def count_ancestors(p):
    """Returns: num of known ancestors
    Pre: p is a Person with attribute parents, a list of parents
    # 1. Handle base case. (We decided this wasn't necessary)
                                                             id3
                                          id9
                                                                      list
    # 2. Break into parts
                                                    Person
    # For each parent, count ancestors
                                                              0
                                                                   id5
                                          name | "Ming"
    # (plus parent, parent2 themselves)
                                                                   id6
                                           parents id3
    n ancestors = 0
                                                                  id15
    for parent in p.parents:
       n_ancestors += (1 + count_ancestors(parent))
                                                          id5
                                                                    Persor
                                                           name "MingMom"
    # 3. Combine the result : FREE!
                                                           parents id7
    return n ancestors
```

Notice when you have no parents, you return n_ancestors with the # value 0. (the parent list is empty so you don't go in the loop) 37

Exercise: Find Ancestors

def list ancestors(p): """Returns: list of all ancestors of p""" # 1. Handle base case. John Sr. Pamela Eva Shane Carmen # 2. Break into parts. # 3. Combine answer. John Jr. Ellen **Portia** Jane John III Alice

John IV