Using recursion

COSC 101, 2018-04-11

Announcements
● Exam 2 Thursday @ 7pm in 207 Lathrop Hall

Outline
● Warm-up
● Visualizing recursion
● Rules of recursion

Warm-up
Assume you have been provided the following function (which we wrote in class on Monday):

```python
def downupN(string, n):
    if n == 1:
        print(string)
    else:
        print(string[:n])
downupN(string,n-1)
        print(string[:n])
```

a) List the sequence of statements that are executed when the following lines of code are executed:

```python
word = "cosc"
downupN(word, len(word))

downupN("cosc", 4)
print("cosc")
downupN("cosc", 3)
print("cos")
downupN("cosc", 2)
print("co")
downupN("cosc", 1)
print("c")
print("co")
print("cos")
print("cosc")
```

b) Modify the `downupN` function to eliminate the parameter `n` and instead use the `len` function.

```python
def downupN(string):
    if len(string) == 1:
        print(string)
    else:
        print(string[:len(string)])
downupN(string[:len(string)-1])
print(string[:len(string)])
```
Visualizing recursion

- Recursion occurs naturally in nature --- e.g., trees, leaves, snowflakes
  - Use recursion to draw trees
    - Draw branch; recurse on right fork; recurse on left fork
    - t = turtle.Turtle()
      def treeN(n):
        if n == 0:
          t.stamp()
        else:
          t.forward(n*15) # Branch
          t.right(20)
          treeN(n-1,t) # Right fork
          t.left(40)
          treeN(n-1,t) # Left fork
          t.right(20)
          t.backward(n*15) # Back down branch
      treeN(1)
      treeN(2)
      treeN(3)
      treeN(4)
Recursion occurs in mathematics --- fractals
  - Koch snowflake
  - Sierpinski triangle
    - Connected midpoints of sides to create four smaller triangles
    - Recurse on triangles in each of the three corners
Recursion occurs in art --- e.g., Matryoshka dolls

List the recursive calls for each of the following programs. Also, what does each program output?

a) def string_mystery(s):
   if len(s) <= 1:
     return s
   else:
     one = s[0] * len(s)
     rest = string_mystery(s[1:])
     return one + rest

print(string_mystery("ABCD"))

Recursive calls:
string_mystery("ABCD") # Returns AAAABBBCCD
string_mystery("BCD") # Returns BBBCCD
string_mystery("CD") # Returns CCD
string_mystery("D") # Returns D

Output:
AAAABBBCCD
b) def half_mystery(lst):
    if len(lst) == 1:
        return lst[0]
    else:
        mid = len(lst)//2
        left_half = half_mystery(lst[:mid])
        right_half = half_mystery(lst[mid:])
        if left_half < right_half:
            return left_half
        else:
            return right_half

print(half_mystery([5,2,4,3]))
Recursive calls:
    half_mystery([5,2,4,3]) # Returns 2
        half_mystery([5,2]) # Returns 2
            half_mystery([5]) # Returns 5
            half_mystery([2]) # Returns 2
        half_mystery([4,3]) # Returns 3
            half_mystery([4]) # Returns 4
            half_mystery([3]) # Returns 3
    Output:
        2

Rules of recursion

- Every recursive function must have:
  1) Base case --- problem is simple, often when one or nothing is left
     a) Solve problem directly
  2) Recursive case --- problem is not simple enough to solve directly
     a) Divide --- break into two pieces, a simple piece to handle now and a “harder” piece that is a smaller version of the same problem
     b) Recurse --- make a recursive function call (i.e., function calls itself) with harder piece and trust that it will come together correctly
     c) Combine --- put the result of the recursive call and the simple piece together into a complete solution

- For each of the following problems, what is the condition and result for the base case?
  a) Count the number of people between you and the left wall of the classroom, including yourself
     Condition: no person to your left
     Result: 1
  b) downup
     Condition: string with one character (i.e., n = 1)
     Result: print string
  c) Draw a tree
     Condition: leaf (i.e, n = 0)
     Result: draw (or stamp) leaf
  d) Sum the numbers in a list
     Condition: list with one (or zero) number(s)
     Result: the number (or zero)
e) Reverse the characters in a string --- e.g., "COLGATE" becomes "ETAGLOC"
   Condition: string with one character
   Result: single character

For each of the following problems, what is the smaller problem on which you recurse?
   a) Count the number of people between you and the left wall of the classroom, including yourself
      Count people to the left of your neighbor
   b) downup
      String without last character
   c) Draw a tree
      Left fork and right fork
   d) Sum the numbers in a list
      List without the first (or last) number
   e) Reverse the characters in a string --- e.g., "COLGATE" becomes "ETAGLOC"
      String without first (or last) character

For each of the following problems, how do you combine the simple piece and the result of the recursive call?
   a) Count the number of people between you and the left wall of the classroom, including yourself
      Add one to your neighbor's count
   b) Sum the numbers in a list
      Add the first number in the list to the sum of the rest of the list,
      OR add the last number in the list to the sum of the rest of the list
   c) Reverse the characters in a string --- e.g., "COLGATE" becomes "ETAGLOC"
      Add the reverse of the rest of the string to the first character of the string,
      OR add the last character of the string to the reverse of the rest of the string