ITI 1120

Lab # 11

Recursion

Starting Lab 11

- Open a browser and log into Brightspace
- On the left hand side under Labs tab, find lab11 material contained in lab11-students.zip file
- Download that file to the Desktop and unzip it.

Before starting, always make sure you are running Python 3

This slide is applicable to all labs, exercises, assignments … etc

ALWAYS MAKE SURE FIRST that you are running Python 3.

That is, when you click on IDLE (or start python any other way) look at the first line that the Python shell displays. It should say Python 3.

If you do not know how to do this, read the material provided with Lab 1. It explains it step by step

Do all the exercises labeled as Task in your head i.e. on a paper

Later on if you wish, you can type them into a computer (or copy/paste from the solutions once I poste them)

Task 1:

(a) What do the following two programs print?

(b) What do functions orange and guave do when called with positive integer as input argument? (answer in plain language)

(c) http://www.pythontutor.com/visualize.html#mode

Open file t1.py and copy both programs into Python Vizualizer. Make sure you understand all the functions that are running (at the same time) as you step through the execution of the program.

```
def orange(n):
    if n > 0:
        print(n,end=" ")
        orange(n-2)

orange(10)
```

```
def guava(n):
    if n > 0:
        guava(n-2)
        print(n,end=" ")

guava(10)
```

Task 2:

(a) What does the following program print?

(b) Write one sentence that describes abstractly (in plain English) what the function mulberry does when called with positive integer as input argument.

(c) What would happen if we made mulberry(-3) call in the main, instead of mulberry (5)?

(d) http://www.pythontutor.com/visualize.html#mode

Open file t1.py and copy code below into Python Vizualizer. Make sure you understand how the program is running and what it is doing as you step through its execution of the program.

(e) What is the maximum number of mulberry functions running at any one time, i.e. what is the maximum number mulberry functions on the stack at any one time?

- What is the answer for general n?

- Make the call mulberry(1000) in Python shell instead mulberry(4) of and observe what happened? Why did that happen?

```python
def mulberry(n):
    if n == 1:
        return 1
    else:
      return n + mulberry(n - 1)

print( mulberry(4) )
```

Task 3:

(a) What does the following program print?

(b) Write one sentence that describes abstractly (in plain English) what the function cantaloupe does when called with positive integer as input argument.

(c) http://www.pythontutor.com/visualize.html#mode

Open file t1.py and copy code below into Python Vizualizer. Make sure you understand how the program is running and what it is doing as you step through the execution of the program.

(d) What is the maximum number of cantaloupe functions running at any one time, i.e. what is the maximum number cantaloupe functions on the stack at any one time? What is the answer for general n?

```
def cantaloupe(n):
    if n > 0:
        print( n % 10)
        cantaloupe(n // 10)

cantaloupe(7254)
```

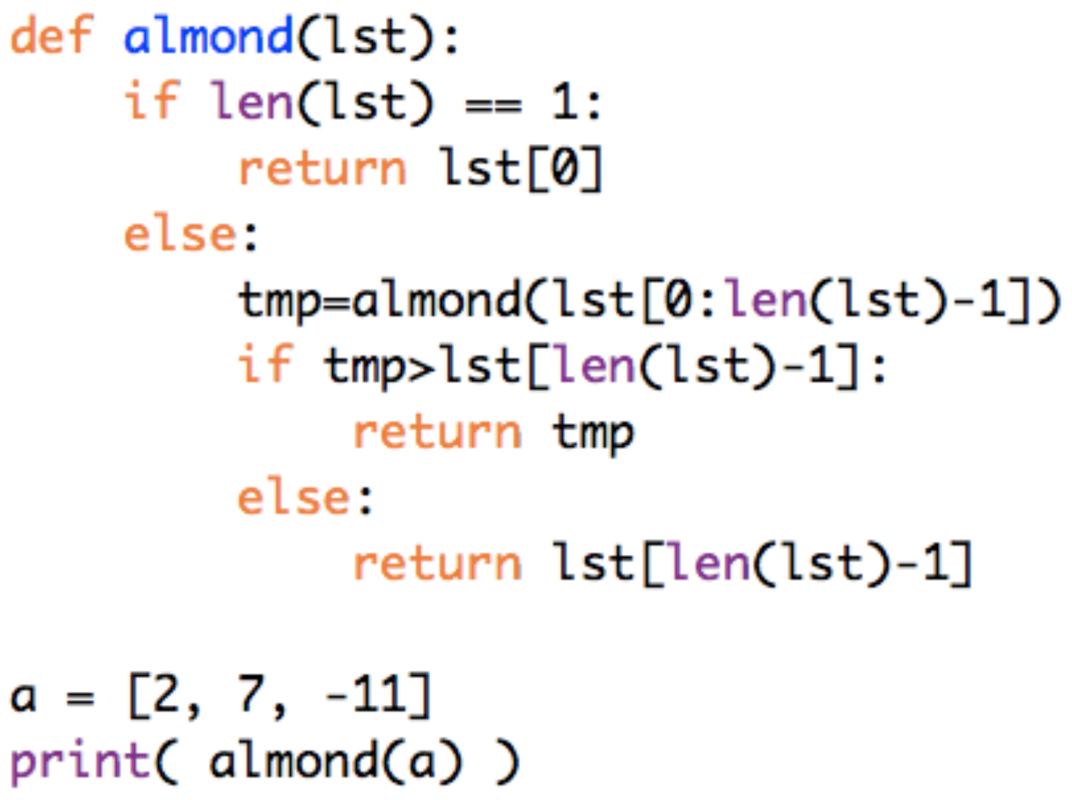
Task 4:

(a) What does the following program print?

(b) Write one sentence that describes abstractly (in plain English) what the function almond does given a list of numbers lst as input argument?

(c) http://www.pythontutor.com/visualize.html#mode

Open file t1.py and copy code below into Python Vizualizer. Make sure you understand how the program is running and what it is doing as you step through the execution of the program.



```python
def almond(lst):
    if len(lst) == 1:
        return lst[0]
    else:
        tmp=almond(lst[0:len(lst)-1])
        if tmp>lst[len(lst)-1]:
            return tmp
        else:
            return lst[len(lst)-1]

a = [2, 7, -11]
print( almond(a) )
```

Task 5:

(a) What does the following program print?

(b) Write one sentence that describes abstractly (in plain English) what the function fig does given a list of numbers lst as input argument and high=len(lst)-1.

(c) http://www.pythontutor.com/visualize.html#mode

Open file t1.py and copy code below into Python Vizualizer. Make sure you understand how the program is running and what it is doing as you step through the execution of the program.

```
def fig(lst, high):
    if high == 0:
        return lst[0]
    else:
        tmp=fig(lst, high - 1)
        if tmp>lst[high]:
            return tmp
        else:
            return lst[high]

a = [2, 7, -11]
print( fig (a, len(a)-1) )
```

Task 6:

(a) What does the following program print?

(b) Write one sentence that describes abstractly (in plain English) what the function almond does (given strings s and ch as input and assuming ch contains only one character)

(c) http://www.pythontutor.com/visualize.html#mode

Open file t1.py and copy code below into Python Vizualizer. Make sure you understand how the program is running and what it is doing as you step through the execution of the program.

```python
def nox(s, ch):
    if len(s)==0:
        return s
    elif s[0]==ch:
        return nox(s[1:], ch)
    else:
        return s[0]+nox(s[1:], ch)

print( nox('Cacic', 'c' ))
```

Recursion: Paper Study

- Open file sum_prod.py

It contains contains 2 functions that both compute the sum: 1+2+3 …+n. One computes it in the 'usual' way using python's loops (this is called, iterative solution), and the other in a recursive way (i.e. using function calls that solve a problem on the smaller problem instance)

Similarly sum_prod.py contains 2 function that both compute the product 1*2*3…*n (thus they compute n!, i.e. n factorial) in iterative and recursive way (as we have seen in class)

Study with TAs and understand well all these 4 solutions.

Recursion: Programming Exercise 1

- Write a recursive function (do not use python's loops), called m, that computes the following series, given positive integer i:

$$m(i) = \frac{1}{3} + \frac{2}{5} + \frac{3}{7} + \frac{4}{9} + \frac{5}{11} + \frac{6}{13} + \ldots + \frac{i}{2i + 1}$$

- In the "main" write a loop that tests your function m by displaying values **m(i)** for **i = 1, 2, . . . , 10**, as follows

```
m(1)= 0.3333333333333333
m(2)= 0.7333333333333334
m(3)= 1.161904761904762
m(4)= 1.6063492063492064
m(5)= 2.060894660894661
m(6)= 2.5224331224331227
m(7)= 2.9890997890997895
m(8)= 3.459688024393907
m(9)= 3.933372234920223
m(10)= 4.409562711110699
```

Recursion: Programming Exercise 2

- Write a recursive function, called count_digits, that counts the number of digits in a given positive integer n.

- Test your function:

```
>>> count_digits(0)
1
>>> count_digits(7)
1
>>> count_digits(73)
2
>>> count_digits(13079797)
8
>>>
```

Recursion: Programming Exercise 3

- A string is a palindrome if it reads the same from the left and from the right. For example, word "kayak" is a palindrome, so is a name "Anna", so is a word "a". Word "uncle" is not a palindrome.

- Write a recursive function, called is_palindrome, that returns True if the input string is a palindrome and otherwise returns False. Test your function.

- Notice: a word of length n is a palindrom if 1st and nth letter are the same, AND 2nd and (n-1)st are the same, and so on … until we get to the "middle" of the word.

```
>>> is_palindrome('blurb')
False
>>> is_palindrome('a')
True
>>> is_palindrome('anna')
True
>>> is_palindrome('Anna')
True
>>> is_palindrome("A man, a plan, a canal --
Panama!")
False
>>> is_palindrome("Madam, I'm Adam")
False
```

Idea/Strategy

Checking if a string is a palindrome can be divided into two subproblems:

1. Check if the 1st and the last character in the string are the same
2. Ignore the two end characters and check if the rest of the substring is a palindrome.

Notice that the 2nd subproblem is the same as the original problem but smaller in size.

Useful string methods: lower(), and string slicing

Recursion: Programming Exercise 4

- Refine your function is_palindrome, and call the modified version, is_palindrome_v2, such that it ignores all characters but the characters that correspond to the letters of English alphabet. You may find Python's string method .isalpha() useful.

- Test your function with the following at least:

```
>>> is_palindrome_v2("A man, a plan, a canal -- Panama!")
True
>>> is_palindrome_v2("Go hang a salami, I'm a lasagna
hog")
True
>>> is_palindrome_v2("Madam, I'm Adam")
True
>>> is_palindrome_v2("Madam, I'm")
False
>>> is_palindrome_v2('blurb')
False
>>> is_palindrome_v2('a')
True
>>> is_palindrome_v2('Anna')
True
```

Recursion: Programming Exercise 5: GCD

The greatest common divisor (GCD) of two integers (at least one of which is not zero), is the largest positive integer that divides the numbers without a remainder. For example, the GCD of 12 and 8 is 4.

The following technique is known as Euclid's Algorithm because it appears in Euclid's Elements (Book 7, ca. 300 BC). It may be the oldest nontrivial algorithm.

The process is based on the observation that, if r is the remainder when a is divided by b, then the common divisors of a and b are the same as the common divisors of b and r. Thus we can use the equation

gcd(a,b) = gcd(b,r)

to successively reduce the problem of computing a GCD to the problem of computing the GCD of smaller and smaller pairs of integers. For example,

gcd(36, 20) = gcd(20, 16) = gcd(16, 4) = gcd(4, 0) = 4

implies that the GCD of 36 and 20 is 4. It can be shown that for any two starting numbers, this repeated reduction eventually produces a pair where the second number is 0. Then the GCD is the other number in the pair.

Write a recursive function called gcd that takes two integer parameters a and b (you can assume a>=b) and that uses Euclid's algorithm to compute and return the greatest common divisor of the two numbers. Test your function.

Difficult question: GCD

What is the depth of the recursion (i.e. the maximum number of gcd methods on the stack/memory) of your gcd function if you called it with gcd(36, 20)? You can find this out by drawing the diagrams as we did in class and or running your function and the gcd(36, 20) in Python Visualizer.

Here is a difficult question and food for thought: What is the maximum depth of the recursion of your gcd method for general a and b (where a>=b)?