

## Lab Questions: Lab Session 10

Deadline: 01.11.2017 11:59pm SGT

Complete all assignments below. For those questions that are marked with an asterisk \*, i.e. **Questions ?? and ??**, create the script files as requested. Once you are done with it, submit the file(s) via iNTU. Remember to put plenty of comments for all assignments, as this helps us to better understand your program (which might give you higher marks).

**Important!!! Make sure your scripts work properly, as we give 0 marks otherwise. Please name the scripts according to the requirements, and upload each file separately and not in a Zip file or similar. Check that your files properly have been uploaded to iNTU. The submission system closes at the deadline. Hence after that, you will get no marks for your solution.**

1. Test your knowledge and understanding of the set operations. Create two sets `v1, v2` filled with random integer elements and use the four set operators `union`, `intersect`, `difference`, `symmetric difference` to see if the outputs are as expected.
2. Create anonymous functions that calculate the areas of a rectangle, trapezoid, and sphere. Save all of the anonymous functions in a file called `my_functions.py`. Try to call the functions just created from another script. Do you get an error message? Then import the file `my_functions.py`, and try to call them again using the prefix `my_functions`.
3. The hyperbolic sine for an argument `x` is defined as:

$$\text{hyperbolicsine}(x) = (e^x - e^{-x})/2$$

Write an anonymous function to implement this. Compare yours to the function `math.sinh` from the `math` module.

4. A vector can be represented by its rectangular coordinates `x` and `y` or by its polar coordinates `r` and `θ`. For positive values of `x` and `y`, the conversions from rectangular to polar coordinates in the range from 0 to  $2\pi$  are  $r = \sqrt{x^2 + y^2}$  and  $\theta = \arctan(y/x)$ . The function for arctan is `math.atan`. Write an anonymous function `recpol` to receive as input arguments the rectangular coordinates and that returns the corresponding polar coordinates as a 2-element list.
5. A recursive definition of  $a^n$  where `a` is an integer and `n` is a non-negative integer is:  
 $a^n = 1$  if `n == 0`  
 $a * a^{n-1}$  if `n > 0`

Write a recursive function called `mypower`, which receives `a` and `n` and returns the value of  $a^n$  by implementing the above definition. Note: The program should **not** use the exponentiation operator anywhere; this is to be done recursively instead! Test the function.

6. \* Combinatorial coefficients  $C(n, m)$  can be defined recursively as follows:

$$C(n,m) = \begin{cases} 1 & , \text{if } m = 0 \text{ or } m = n \\ C(n-1, m-1) + C(n-1, m) & , \text{otherwise} \end{cases}$$

Write a recursive function `combi.py` to implement this definition. Return the error value  $-1$  if the condition  $0 \leq m \leq n$  is not respected.

7. A word is a palindrome, if it reads the same forward or reversed. For instance, 'rotator', 'rotor', 'radar', are all palindromes. Write a **recursive** function `isPalindrome` that checks if the input string is a palindrome. That is, the function takes as an input a string, and returns true (logical 1), if the input string is a palindrome, or false (logical 0) if it is not.

Example of execution:

```
>>> isPalindrome( 'aba')
True
>>> isPalindrome( 'taat')
True
>>> isPalindrome( 'tata')
False
```

8. What does this function do?

mystery.py

```
def mystery(x,y):
    if y == 1: return x
    else: return x + mystery(x,y-1)
```

Explain what this mystery function does with its two arguments.

9. Write a recursive function `compute_gcd.py` that takes as input two positive integers  $a$  and  $b$ , and outputs the greatest common divisor of  $a$  and  $b$  (hint: the greatest common divisor of  $a$  and  $b$  is the same as the greatest common divisor of  $a$  and  $b - a$ ). Return the error value  $-1$  if any of the inputs is not a positive integer.
10. \* The Fibonacci numbers is a sequence of numbers  $F_i$ :

0 1 1 2 3 5 8 13 21 34 ...

where  $F_0$  is 0,  $F_1$  is 1,  $F_2$  is 1,  $F_3$  is 2, and in general  $F_i = F_{i-1} + F_{i-2}$ . That is, the sequence starts with 0 and 1, and all other Fibonacci numbers are obtained by adding the previous two Fibonacci numbers.

In a file `fibonacci.py`, write two independent functions that implement this definition: recursive function `r_fibonacci` and iterative function `i_fibonacci`. Each of the functions will receive one integer argument  $n$ , and will return one integer value, which is the  $n$ -th Fibonacci number  $F_n$ . To test if your functions are correct call them with the same input value.

Which of the functions is easier to write? Which one is faster (measure the time taken when calling each function for  $n = 35$ )? Draw the tree of the successive recursive calls to the function `r_fibonacci` (starting from  $F_4$ ) and provide an explanation of the difference in timing (do not submit the drawing on NTU learn, just submit your explanations as comment in the python file submitted).

Hint: for the recursive function, in the definition there is one general case but two base cases.

11. The Ackermann-Peter function is defined as follows for non-negative integers  $m$  and  $n$ :

$$\begin{aligned} A(m,n) &= n+1 && , \text{ if } m = 0 \\ &= A(m-1, 1) && , \text{ if } m>0 \text{ and } n=0 \\ &= A(m-1, A(m,n-1)) && , \text{ if } m>0 \text{ and } n>0 \end{aligned}$$

In a file `ackermann.py`, write a recursive function `A` that implements this definition. Return the error value `-1` if the condition that  $m$  and  $n$  are non-negative integers is not respected. Note: for testing, do not use values  $m > 3$  as the function grows very quickly.

12. Write a recursive function `perm.py` that takes as input a string `S`, and that will output a list of strings containing of all possible permutations of the letter positions in `S`. Here are some execution examples:

```
>>> print(perm('bat'))
['bat', 'bta', 'abt', 'atb', 'tba', 'tab']
>>> print(perm('jean'))
['jean', 'jena', 'jaen', 'jane', 'jnea', 'jnae', 'ejan', 'ejna',
 'eajn', 'eanj', 'enja', 'enaj', 'ajen', 'ajne', 'aejn', 'aenj',
 'anje', 'anej', 'njea', 'njae', 'neja', 'neaj', 'naje', 'naej']
```

Hint: Assume that you have a function that generates the set  $S_n$  of all permutations of  $n$  elements, you can generate the set  $S_{n+1}$  of all permutations of  $n + 1$  elements by inserting the new element in every possible position of every member of  $S_n$ .