|  |  |  |
| --- | --- | --- |
| **College of Engineering**  Computer Science & Eng. Dept.  **Course:** CMP 321L Programminglanguages Lab | A picture containing logo  Description automatically generated | **Course Professor:** Dr. Michel Pasquier  **Lab Instructor:** Praveena Kolli  **Office:** EB2-126  **Phone**: 971-6-5152352  **e-mail**: pkolli@aus.edu  **Semester**: Fall 2021 |

**Lab 3 – Python Functions**

**Objectives:**

* Make use of Python specific features
* Define functions and nested functions
* Define and use lambda expressions

**Due date: End of the lab. (**Only one team member needs to submit.)

**Rules:**

(1) Usage: **You should explore and make good use of the Python features you learned in class.** (2) Scope: **You should only use those features that have been explained in detail in class.**

(3) Style: Follow standard Python programming style and conventions.

(4) Logic: Add appropriate comments to your code to explain your solution.

(Code / answers that do not follow the above specifications will be penalized.)

***Warning:* You need to come to the lab properly prepared i.e.**

(1) Make sure you have studied and understood the class material.

(2) Read the lab doc, think about the problems, and prepare questions as needed.

If you do not, completing the lab in 2.45 hours may become too much of a challenge!

**Useful resources:**

* <https://docs.python.org/3/tutorial/datastructures.html>

**Exercise 1: Recursive, accumulator functions [2 Marks]**

1. Write a *classic* recursive function of N that returns the Nth Fibonacci number. Note that the function should be non-accumulator and non-tail.
2. Write an *accumulator* recursive function of N that returns the Nth Fibonacci number.
3. Write a recursive function of N that returns the Nth Fibonacci number, *calculating each number once only*. (Hint: use a dictionary.)
4. Write an *accumulator* recursive function of N that returns a *list* of all the Fibonacci numbers, *calculating each number once only*. (Hint: no dictionary needed.)

**Code:**

#part a)

def fib(n):

if n<=1:

return 1

return fib(n-1)+fib(n-2)

print([fib(i) for i in range(11)])

#part b)

def fibAcc(n):

def fibAcc(n,prev=0,acc=1):

if n<=0:

return acc

return fibAcc(n-1,acc,prev+acc)

return fibAcc(n)

print([fibAcc(i) for i in range(11)])

#part c)

def fibMemoized(n):

memo={0:1,1:1}

def fibMemoized(n):

if n in memo:

return memo[n]

return fibMemoized(n-1)+fibMemoized(n-2)

return fibMemoized(n)

print([fibMemoized(i) for i in range(11)])

#part d)

def fibAccList(n):

arr=[1,1]

def fibAccList(n,arr):

if n==0:

return [1]

if n==1:

return [1,1]

if len(arr)==n:

return arr

arr.append(arr[-1]+arr[-2])

return fibAccList(n,arr)

return fibAccList(n,arr)

print(fibAccList(11))

**Screenshot:**

**Exercise 2: User defined functions and zip [1 Marks]**

Write an iterative function that takes three lists of strings as arguments (students’ name, address, and major) and returns a list of strings mixing these three fields, formatted as per the example below. You may assume all three lists comprise the same number of elements. Use zip().

Example: >>> fun( ["Ahmed", "John", "Zina"], ["Dubai", "Sharjah", "Al Ain"],

["COE", "CMP", "ELE"] )

['Ahmed from COE lives in Dubai', 'John from CMP lives in Sharjah', 'Mohd from ELE lives in Al Ain']

**Code:**

def fun(names,locations,majors):

return [f'{name} from {maj} lives in {loc}' for name,maj,loc in zip(names,majors,locations)]

print(fun( ["Ahmed", "John", "Zina"], ["Dubai", "Sharjah", "Al Ain"],

["COE", "CMP", "ELE"] ))

**Screenshot:**



**Exercise 3: User defined functions with default parameters [1 Marks]**

Write a function enum() that takes a list, default starting value and default step for the counter and returns a list of tuples aggregating the given list elements with a counter. Do not use enumerate, zip or similar.

Example: >>> enum( ['A', 'B', 'C'] ) enum( ['A', 'B', 'C'], 5 )

[ (0, 'A'), (1, 'B'), (2, 'C') ] [ (5, 'A'), (6, 'B'), (7, 'C') ]

>>> enum( ['A', 'B', 'C'], 4, 2 )

[ (4, 'A'), (6, 'B'), (8, 'C') ]

**Code:**

def enum(arr,start=0,step=1):

return [(step\*i+start,arr[i]) for i in range(len(arr)) ]

print(enum( ['A', 'B', 'C'] ))

print(enum( ['A', 'B', 'C'], 5 ))

print(enum( ['A', 'B', 'C'], 4, 2 ))

**Screenshot:**

Text

Description automatically generated

**Exercise 4: Recursive functions [2 Marks]**

1. Write a recursive function that returns true if an object is a member of a given list, or false otherwise. Hint: use slicing.

Examples: >>> member( 2, [1, 3, 5] ) >>> member( 4, [1, 2, 3, 4, 5] )

False True

1. Write a recursive function indexOf() to return the index of the object if found, or -1 otherwise. Do not use enum function from ex(2) or enumerate or zip

Examples: >>> indexOf( 2, [1, 3, 5] ) >>> member( 4, [1, 2, 3, 4, 5] )

-1 3

**Code:**

#part a)

def member(x,arr):

if len(arr)==0:

return False

if x==arr[0]:

return True

return member(x,arr[1:])

print(member( 2, [1, 3, 5] ))

print(member( 4, [1, 2, 3, 4, 5] ))

#part b)

def indexOf(x,arr,acc=0):

if len(arr)==0:

return -1

if x==arr[0]:

return acc

return indexOf(x,arr[1:],acc+1)

print(indexOf( 2, [1, 3, 5] ))

print(indexOf( 4, [1, 2, 3, 4, 5] ))

**Screenshot:**

**Text

Description automatically generated**

**Exercise 5: Recursive function and lambda expression [2 Marks]**

1. Write a recursive function apply\_all() that takes a function and a list as arguments and returns a list comprising the results of applying the function to each elements of the list.

Example: >>> apply\_all( sqrt, [4, 9, 16, 25] ) # sqrt = square root function

[2, 3, 4, 5]

1. Use apply\_all function to check which string in a list contains numbers only.

apply\_all(lambda:…., ["1234", "hellow","class23"])

**Exercise 6: Returning a function or lambda expression as value [2 Marks]**

1. We want to code a function that computes binomial coefficients.

Binomial coeffients formula

Write a function T(n) that returns a nested function C(k) that computes binomial coefficients.

Example: >>> f = T(10) ; f(3) # f computes C(10,k)

120

Or

print(T(10)(3))

120

1. Rewrite part(a) C function that returns a lambda expression instead of a nested function