 **Northwestern Polytechnic University**

**Python Programming**

**Homework Assignment #4**

**Due day: 10/28/2021**

**Instruction:**

1. **Push the source code to GitHub or answer sheet in word file**
2. **Please follow the code style rule like programs on handout.**
3. **Overdue homework submission could not be accepted.**

**4. Takes academic honesty and integrity seriously (Zero Tolerance of Cheating & Plagiarism)**

1. Write a high order function to make withdrawal from the balance in bank account by nonlocal variable.

***def*** ***mk\_wd(balance):***

*"""*

*Return the balance after withdrawal since inception*

*>>> rem = mk\_wd (100) # deposit $100*

*>>> rem(10)*

*90*

*>>> rem (20)*

*70*

*>>> rem (100)*

*'Insufficient funds'*

*"""*

Answer :

def mk\_wd(balance):  
 def draw(amt):  
 nonlocal balance  
 if amt > balance:  
 return 'Insufficient funds'  
 balance = balance-amt  
 return balance  
 return draw  
rem = mk\_wd (100)  
print(rem(10))  
print(rem(20))  
print(rem(100))

2. Write a function that deletes all instances of an element from a list.

***def***  ***rm\_all(elem, lst):***

*"""*

*>>> x = [3, 1, 2, 1, 5, 1, 1, 7]*

*>>> rm\_all (1, x)*

*>>> x*

*[3, 2, 5, 7]*

*"""*

Answer:

def rm\_all(elem,lst):  
 try:  
 while True:  
 lst.remove(elem)  
 except ValueError:  
 pass  
 print(lst)  
x = [3, 1, 2, 1, 5, 1, 1, 7]  
rm\_all (1, x)

3. Write a function that takes in three arguments *x, elem*, and a list, and adds as many *"elem"s* to the end of the list as there are *x’s*.

***def*** ***add\_many(x, elem, lst):***

*"""*

*Adds elem to the end of lst the number of times x occurs in lst.*

*>>> lst = [1, 2, 4, 2, 1]*

*>>> add\_many (2, 5, lst)*

*>>> lst*

*[1, 2, 4, 2, 1, 5, 5]*

"""

Answer :

def add\_many(x,elem,lst):  
 for i in range(x):  
 lst.append(elem)  
 return lst  
lst = [1, 2, 4, 2, 1]  
print(add\_many(2, 5, lst))

4. Write a function to create a new list from given a *"suits"* list and a number list

***def*** ***f (suits, numbers):***

*"""Creates a new list (2-element list as one element in a new list) with the given suits and numbers. Each element in the returned list should be of the form [suit, number].*

*>>> f (['S', 'C'], [1, 2, 3])*

*[['S', 1], ['S', 2], ['S', 3], ['C', 1], ['C', 2], ['C', 3]]*

*>>> f (['S', 'C'], [3, 2, 1])*

*[['S', 3], ['S', 2], ['S', 1], ['C', 3], ['C', 2], ['C', 1]]*

*>>> f ([], [3, 2, 1])*

*[]*

*>>> f (['S', 'C'], [])*

*[]*

*"""*

Answer :

def f (suits, numbers):  
 print( [[i, j] for i in suits for j in numbers])  
f (['S', 'C'], [3, 2, 1])

5. Write a function to merge *2* sorted lists *a* and *b*, and then return a new list with a sorted order by RECURSIVE calls.

***def*** ***mrg(ls1, ls2):***

*"""Merges two sorted lists recursively.*

*>>> mrg ([1, 3, 5], [2, 4, 6])*

*[1, 2, 3, 4, 5, 6]*

*>>> mrg ([], [2, 4, 6])*

*[2, 4, 6]*

*>>> mrg ([1, 2, 3], [])*

*[1, 2, 3]*

*>>> mrg ([5, 7], [2, 4, 6])*

*[2, 4, 5, 6, 7]*

*"""*

Answer :

def mrg(lst1, lst2):  
 if lst1==[]:  
 return lst2  
 elif lst2==[]:  
 return lst1  
 if lst1[0]>lst2[0]:  
 return [lst2[0]]+mrg(lst1,lst2[1:])  
 else:  
 return [lst1[0]]+mrg(lst1[1:], lst2)  
print(mrg ([1, 3, 5], [2, 4, 6]))  
print(mrg ([], [2, 4, 6]))  
print(mrg ([1, 2, 3], []))  
print(mrg ([5, 7], [2, 4, 6]))

6. Write the function to flatten the deep list.

***def*** ***fltn(ls):***

*"""Return a new version of list as follows.*

*>>> fltn ([1, 2, 3]) # normal list*

*[1, 2, 3]*

*>>> x = [1, [2, 3], 4] # deep list*

*>>> fltn (x)*

*[1, 2, 3, 4]*

*>>> x = [[1, [1, 1]], 1, [1, 1]] # deep list*

*>>> fltn (x)*

*[1, 1, 1, 1, 1, 1]*

*"""*

Answer :

def fltn(ls):  
 if not ls:  
 return []  
 elif type(ls[0]) == list:  
 return fltn(ls[0]) + fltn(ls[1:])  
 else:  
 return [ls[0]] + fltn(ls[1:])  
print(fltn ([1, 2, 3]))  
x = [1, [2, 3], 4]  
print(fltn (x))  
x = [[1, [1, 1]], 1, [1, 1]]  
print(fltn (x))

7. Define a function to check if the element exists in the list or not.

***def***  ***chk\_elm(lst, n):***

*"""*

*>>> a = [ [1,[2]], 3, [ [4], [5,[6] ] ] ]*

*>>> chk\_elm (a, 6)*

*True*

*"""*

def chk\_elem(lst, n):  
 def flattn(lst):  
 ans = []  
 for i in lst:  
 if isinstance(i, list): # item is list then recursively flatten the list then append to the flattened list  
 x = flattn(i)  
 for j in x:  
 ans.append(j)  
 else: # otherwise add to the list  
 ans.append(i)  
 lst = flattn(lst) # flatten the list  
 print (lst)  
 if n in lst: # check if element exist  
 print (True)  
 return True  
 print(False)  
 return False  
a = [ [1,[2]], 3, [ [4], [5,[6] ] ] ]  
chk\_elem (a, 6)

8. Write a function to check whether the input argument list is symmetric or not in recursive call.

***def*** ***sym(l):***

*"""Returns whether a list is symmetric or not*

*>>> sym ([])*

*True*

*>>> sym ([1])*

*True*

*>>> sym ([1, 4, 5, 1])*

*False*

*>>> sym ([1, 4, 4, 1])*

*True*

*>>> sym (['l', 'o', 'l'])*

*True*

*"""*

Answer :

def sym(l):  
 if l == None or len(l) == 0 or len(l) == 1:  
 return True  
 if l[0] == l[len(l)-1]:  
 return sym(l[1:len(l)-1])  
 return False  
  
print(sym ([]))  
print(sym ([1, 4, 5, 1]))  
print(sym ([1, 4, 4, 1]))  
print(sym (['l', 'o', 'l']))

9. Write a function in recursive call that takes in a list *lst*, a function *g*, and an initial value *m*. This function will fold *lst* starting at the beginning. If *lst* is *[1, 2, 3, 4, 5]* then the function *g* is applied as follows:

*g (g (g (g (g (m, 1), 2), 3), 4), 5)*

from operator import add, sub, mul

***def*** ***fld (lst, g, m):***

*"""Return the result of applying the function* *g* *to the initial value* *m* *and the first element in* *lst*, *and repeatedly applying* *g* *to this result and the next element in* *lst* *until it reaches the end of the list.*

*>>> s = [3, 2, 1]*

*>>> fld (s, sub, 0) # sub(sub(sub(0, 3), 2), 1)*

*-6*

*>>> fld (s, add, 0) # add(add(add(0, 3), 2), 1)*

*6*

*>>> fld (s, mul, 1) # mul(mul(mul(1, 3), 2), 1)*

*6*

*>>> fld ([], sub, 100) # return m if s is empty*

*100*

*"""*

Answer:

def add(a, b):  
 return a+b  
  
def mul(a, b):  
 return a\*b  
  
def sub(a, b):  
 return a-b  
  
  
def fld(lst, f, val):  
 if lst is None or len(lst) == 0:  
 return val  
 return fld(lst[1:len(lst)], f, f( val, lst[0]))  
  
s = [3, 2, 1]  
print(fld(s,add,0))  
print(fld(s,sub,0))  
print(fld(s,mul,1))  
print(fld([],sub,100))

10. Implement a function to create *2D* array as follows

***def***  ***crte\_2d\_arr(rows, columns):***

*"""*

*>>> crte\_2d\_arr(3, 5)*

*[['-', '-', '-', '-', '-'], ['-', '-', '-', '-', '-'], ['-', '-', '-', '-', '-']]*

*"""*

Answer:

def crte\_2d\_arr(rows, columns):  
 arr = [["-"] \* columns] \* rows  
 print(arr)  
 return arr  
crte\_2d\_arr(3, 5)