**Logo

Description automatically generated**

**San Francisco Bay University**

**Python Programming**

**Homework Assignment #3**

**Due day: 7/7/2023**

**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'*

*"""*

Program

def mk\_wd(balance):

def withdraw(amount):

nonlocal balance

if amount > balance:

return 'Insufficient funds'

balance -= amount

return balance

return withdraw

def main():

initial\_balance = float(input("Enter the initial balance: "))

rem = mk\_wd(initial\_balance)

withdrawals = []

for i in range(3):

withdrawal = float(input(f"Enter withdrawal {i+1}: "))

withdrawals.append(withdrawal)

for withdrawal in withdrawals:

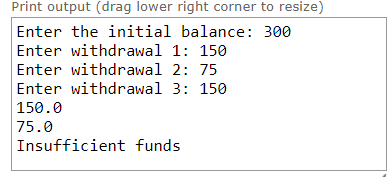
result = rem(withdrawal)

print(result)

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Result**



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]*

*"""*

***Program***

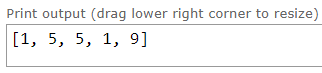
def rm\_all(elem, lst):

return [i for i in lst if i != elem]

x = [2, 1, 2, 5, 5, 2, 1, 9]

print(rm\_all(2, x))

**Result**



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]*

*"""*

**Program**

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

count\_x = lst.count(x)

for \_ in range(count\_x):

lst.append(elem)

return lst

def main():

lst = [1, 3, 4, 3, 1]

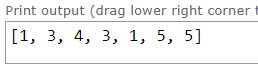
result = add\_many(3, 5, lst)

print(result)

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Result**



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'], [])*

*[]*

*"""*

**Program**

def f(suits, numbers):

new\_list = []

for suit in suits:

for number in numbers:

new\_list.append([suit, number])

return new\_list

# Test case 1

suits1 = ['S', 'C']

numbers1 = [2, 4, 6]

print("Suits:", suits1)

print("Numbers:", numbers1)

result1 = f(suits1, numbers1)

print("New List:", result1)

print()

# Test case 2

suits2 = ['S', 'C']

numbers2 = [1, 3, 5]

print("Suits:", suits2)

print("Numbers:", numbers2)

result2 = f(suits2, numbers2)

print("New List:", result2)

print()

# Test case 3

suits3 = []

numbers3 = [7, 6, 5]

print("Suits:", suits3)

print("Numbers:", numbers3)

result3 = f(suits3, numbers3)

print("New List:", result3)

print()

# Test case 4

suits4 = ['S', 'C']

numbers4 = []

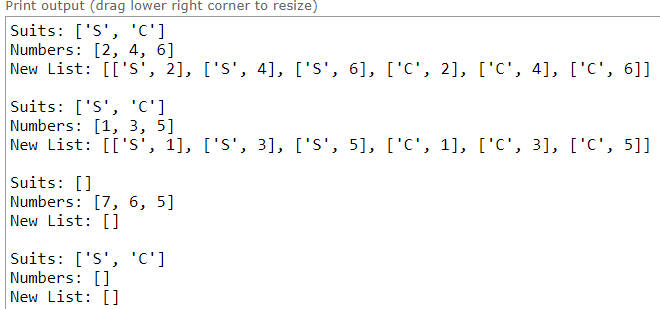
print("Suits:", suits4)

print("Numbers:", numbers4)

result4 = f(suits4, numbers4)

print("New List:", result4)

**Result**



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]*

*"""*

**Program**

def mrg(ls1, ls2):

if not ls1:

return ls2

if not ls2:

return ls1

if ls1[0] < ls2[0]:

return [ls1[0]] + mrg(ls1[1:], ls2)

else:

return [ls2[0]] + mrg(ls1, ls2[1:])

def main():

# Test case 1

ls1 = [3, 1, 8]

ls2 = [4, 2, 7]

print("List 1:", ls1)

print("List 2:", ls2)

merged\_list1 = mrg(ls1, ls2)

print("Merged list:", merged\_list1)

print()

# Test case 2

ls3 = []

ls4 = [8, 5, 6]

print("List 1:", ls3)

print("List 2:", ls4)

merged\_list2 = mrg(ls3, ls4)

print("Merged list:", merged\_list2)

print()

# Test case 3

ls5 = [3, 6, 9]

ls6 = []

print("List 1:", ls5)

print("List 2:", ls6)

merged\_list3 = mrg(ls5, ls6)

print("Merged list:", merged\_list3)

print()

# Test case 4

ls7 = [4, 8]

ls8 = [3, 4, 9]

print("List 1:", ls7)

print("List 2:", ls8)

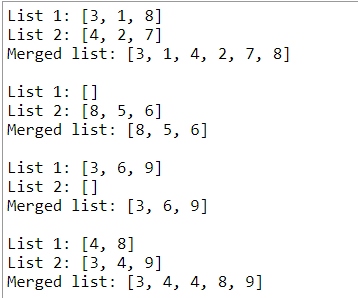
merged\_list4 = mrg(ls7, ls8)

print("Merged list:", merged\_list4)

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Result**



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]*

*"""*

**Program**

def fltn(ls):

result = []

for item in ls:

if isinstance(item, list):

result.extend(fltn(item))

else:

result.append(item)

return result

def main():

# Test case 1: Normal list

ls1 = [5, 10, 15]

print("Original list:", ls1)

flattened\_list1 = fltn(ls1)

print("Flattened list:", flattened\_list1)

print()

# Test case 2: Deep list

ls2 = [4, [8, 16], 24]

print("Original list:", ls2)

flattened\_list2 = fltn(ls2)

print("Flattened list:", flattened\_list2)

print()

# Test case 3: Deep list

ls3 = [[4, [2, 2]], 4, [1, 1]]

print("Original list:", ls3)

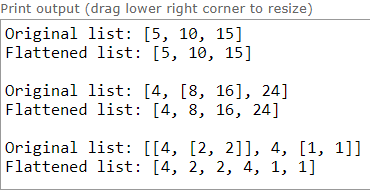
flattened\_list3 = fltn(ls3)

print("Flattened list:", flattened\_list3)

if \_\_name\_\_ == '\_\_main\_\_':

main()

**Result**



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*

*"""*

**Program**

def chk\_elm(lst, n):

for item in lst:

if item == n:

return True

elif isinstance(item, list):

if chk\_elm(item, n):

return True

return False

def main():

a = [[2, [7]], 0, [[5], [1, [6]]]]

element = 0

print("List:", a)

print("Element:", element)

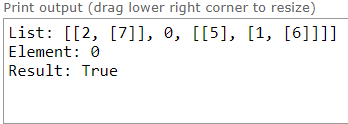
result = chk\_elm(a, element)

print("Result:", result)

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Result**



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*

*"""*

**Program**

def sym(l):

if len(l) <= 1:

return True

if l[0] != l[-1]:

return False

return sym(l[1:-1])

def main():

# Test case 1

list1 = []

print("List:", list1)

result1 = sym(list1)

print("Symmetric:", result1)

print()

# Test case 2

list2 = [1]

print("List:", list2)

result2 = sym(list2)

print("Symmetric:", result2)

print()

# Test case 3

list3 = [2, 6, 4, 2]

print("List:", list3)

result3 = sym(list3)

print("Symmetric:", result3)

print()

# Test case 4

list4 = [3, 5, 5, 3]

print("List:", list4)

result4 = sym(list4)

print("Symmetric:", result4)

print()

# Test case 5

list5 = ['l', 'o', 'l']

print("List:", list5)

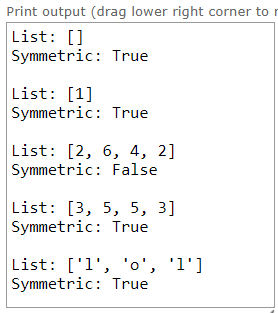
result5 = sym(list5)

print("Symmetric:", result5)

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Result**



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*

*"""*

***Program***

from operator import add, sub, mul

def fld(lst, g, m):

if not lst:

return m

else:

return fld(lst[1:], g, g(m, lst[0]))

# Testing

s = [3, 2, 1]

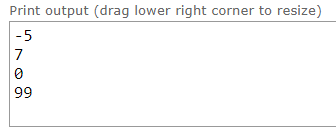
print(fld(s, sub, 1))

print(fld(s, add, 1))

print(fld(s, mul, 0))

print(fld([], sub, 99))

**Result**



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

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

*"""*

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

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

*"""*

**Program**

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

*return [['-' for \_ in range(columns)] for \_ in range(rows)]*

*def main():*

*array = crte\_2d\_arr(3, 5)*

*print(array)*

*if \_\_name\_\_ == '\_\_main\_\_':*

*main()*

***Result***

