**Logo

Description automatically generated**

**San Francisco Bay University**

**Python Programming**

**Homework Assignment #4**

**Due day: 8/6/2023**

**Instruction:**

1. **DON’T allow to call the functions from the existing library**
2. **Push the source code to Github platform**
3. **Please follow the code style rule like programs on handout.**
4. **Overdue homework submission could not be accepted.**

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

1. Write a function to check if the element exists or not in the linked list.

***def*** ***cntn\_link(s, elm):***

*"""Return True if elm is in the linked list s*

*>>> cntn\_link (link(1, link(2, link(3, empty))), 1)*

*True*

*>>> cntn\_link (link(1, link(2, link(3, empty))), 4)*

*False*

"""

**Program**

*def link(value, next\_link):*

*return [value, next\_link]*

*def cntn\_link(s, elm):*

*if s is None:*

*return False*

*elif s[0] == elm:*

*return True*

*else:*

*return cntn\_link(s[1], elm)*

*def main():*

*empty = None*

*result1 = cntn\_link(link(7, link(6, link(5, empty))), 6)*

*print(result1)*

*result2 = cntn\_link(link(3, link(2, link(1, empty))), 0)*

*print(result2)*

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

*main()*

**Result**

A white rectangle with black text

Description automatically generated

1. Create a function to print linked list as follows.

***def******prnt\_lnk(s):***

*"""*

*>>> prnt\_lnk ( link(1, link(2, link(3, link(4, empty)))) )*

*<1 2 3 4>*

*"""*

*Program*

*class Link:*

*def \_\_init\_\_(self, value, next=None):*

*self.value = value*

*self.next = next*

*def prnt\_lnk(s):*

*result = "<"*

*while s is not None:*

*result += str(s.value) + " "*

*s = s.next*

*result = result.strip() + ">"*

*print(result)*

*def link(value, next=None):*

*return Link(value, next)*

*empty = None*

*def main():*

*prnt\_lnk(link(7, link(8, link(2, link(0, empty)))))*

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

*main()*

*Result*

*A white rectangular object with black text

Description automatically generated*

1. Implement a function to create a new linked list in the reverse order.

***def******rvrs\_lnk(s):***

*"""Return linked list reversed*

*>>> rvrs\_lnk (link(1, link(2, link(3, link(4, empty)))))*

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

*"""*

*Program*

*class Link:*

*def \_\_init\_\_(self, value, next=None):*

*self.value = value*

*self.next = next*

*def rvrs\_lnk(s):*

*def reverse\_helper(s, acc):*

*if s is None:*

*return acc*

*else:*

*return reverse\_helper(s.next, [s.value, acc])*

*return reverse\_helper(s, [])*

*def link(value, next=None):*

*return Link(value, next)*

*empty = None*

*def main():*

*reversed\_list = rvrs\_lnk(link(9, link(1, link(0, link(7, empty)))))*

*print(reversed\_list)*

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

*main()*

*Result*

*A screenshot of a computer

Description automatically generated*

1. Write a function *srt (lnk)* function, which returns True if the linked list *lnk* is sorted ascendingly from the left to right. If two adjacent elements are equal, the linked list can still be considered sorted.

***def*** ***srt (lnk):***

*""" if the linked list lnk is sorted, then return True*

*>>> lnk1 = link(1, link(2, link(3, link(4,empty))))*

*>>> srt (lnk1)*

*True*

*>>> lnk2 = link(1, link(3, link(2, link(4, link(5, empty)))))*

*>>> srt (lnk2)*

*False*

*>>> lnk3 = link(3, link(3, link(3, empty)))*

*>>> srt (lnk3)*

*True*

*"""*

***Program***

*def link(value, next\_link):*

*return [value, next\_link]*

*def srt(lnk):*

*if is\_empty(lnk) or is\_empty(rest(lnk)):*

*return True*

*elif first(lnk) <= first(rest(lnk)):*

*return srt(rest(lnk))*

*else:*

*return False*

*def is\_empty(s):*

*return s == []*

*def first(s):*

*return s[0]*

*def rest(s):*

*return s[1]*

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

*lnk1 = link(5, link(10, link(15, link(20, []))))*

*print(srt(lnk1))*

*lnk2 = link(7, link(3, link(8, link(5, link(10, [])))))*

*print(srt(lnk2))*

*lnk3 = link(3, link(3, link(6, [])))*

*print(srt(lnk3))*

***Result***

***A white background with black text

Description automatically generated***

1. Write a function with arguments a linked list *lnk* and a function *g*, which is applied to each number in *lnk* and returns the sum. If the linked list is empty, the sum is *0*.

***def*** ***sum\_lnk(lnk, g):***

*"""Applies a function g to each element in lnk and returns the sum*

*of them*

*>>> sqr = lambda x: x \* x*

*>>> dbl = lambda y: 2 \* y*

*>>> lnk1 = link(1, link(2, link(3, link(4, empty))))*

*>>> sum\_lnk (lnk1, sqr)*

*30 # sqr(1) + sqr(2) + sqr(3) + sqr(4)*

*>>> lnk2 = link(3, link(5, link(4, link(6, empty))))*

*>>> sum\_lnk (lnk2, dbl)*

*36 # dbl(3)+ dbl(5)+ dbl(4)+ dbl(6)*

*"""*

***Program***

*def link(value, next\_link):*

*return [value, next\_link]*

*def sum\_lnk(lnk, g):*

*if is\_empty(lnk):*

*return 0*

*else:*

*return g(first(lnk)) + sum\_lnk(rest(lnk), g)*

*def is\_empty(s):*

*return s == []*

*def first(s):*

*return s[0]*

*def rest(s):*

*return s[1]*

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

*sqr = lambda x: x \* x*

*dbl = lambda y: 2 \* y*

*lnk1 = link(4, link(5, link(6, link(7, []))))*

*print(sum\_lnk(lnk1, sqr))*

*lnk2 = link(8, link(10, link(9, link(11, []))))*

*print(sum\_lnk(lnk2, dbl))*

***Result***

*A white rectangle with black text

Description automatically generated*

1. Define a function with input parameters a linked list, *lnk*, and two elements, *u* & *v*. The function returns linked list but with all elements of *u*substituted with *v*.

***def******change(lnk, u, v):***

*"""Returns a linked list matching lnk but with all elements of u replaced by v. If u does not appear in lnk, then return lnk*

*>>> l = link(1, link(2, link(3, empty)))*

*>>> n=change(l, 3, 1)*

*>>> n*

*[1, [2, [1, [ ] ]]]*

*>>> m=change(n, 1, 2)*

*>>> m*

*[2, [2, [2, [ ]]]]*

*>>> change(m, 5, 1)*

*[2, [2, [2, [ ]]]]*

*"*

***Program***

*# Define a linked list node*

*def link(value, next\_link):*

*return [value, next\_link]*

*def is\_empty(s):*

*return s == []*

*def first(s):*

*return s[0]*

*def rest(s):*

*return s[1]*

*def change(lnk, u, v):*

*if is\_empty(lnk):*

*return lnk*

*if first(lnk) == u:*

*return link(v, change(rest(lnk), u, v))*

*else:*

*return link(first(lnk), change(rest(lnk), u, v))*

*def convert\_to\_list(lnk):*

*def helper(lst):*

*if is\_empty(lst):*

*return []*

*else:*

*return [first(lst), helper(rest(lst))]*

*return helper(lnk)*

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

*l = link(2, link(4, link(6, [])))*

*n = change(l, 6, 2)*

*print(convert\_to\_list(n))*

*m = change(n, 2, 3)*

*print(convert\_to\_list(m))*

*result = change(m, 8, 1)*

*print(convert\_to\_list(result))*

*Result*

A screenshot of a computer

Description automatically generated

1. Generate a function to append element to the end of linked list.

***def******apnd(lnk, m):***

*"""Adds the element m to the end of lnk  
  
 >>> l = link(1, link(2, link(3, empty)))  
 >>> n = apnd (l, 4) # n = [1, [2, [3, [4, [] ]]]]  
 >>> first(rest(rest(rest(n))))  
 4  
 """*

***Program***

***def link(value, next\_link):***

***return [value, next\_link]***

***def is\_empty(s):***

***return s == []***

***def first(s):***

***return s[0]***

***def rest(s):***

***return s[1]***

***def apnd(lnk, m):***

***if is\_empty(lnk):***

***return link(m, [])***

***else:***

***return link(first(lnk), apnd(rest(lnk), m))***

***def convert\_to\_list(lnk):***

***def helper(lst):***

***if is\_empty(lst):***

***return []***

***else:***

***return [first(lst), helper(rest(lst))]***

***return helper(lnk)***

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

***l = link(5, link(8, link(11, [])))***

***n = apnd(l, 14)***

***print(convert\_to\_list(n))***

***print(first(rest(rest(rest(n)))))***

***Result***

*A screenshot of a computer error

Description automatically generated*

1. Implement the insert function that creates a copy of the original list with an item inserted at the specific index. If the index is greater than the current length, you should insert the item at the end of the list.

***def******insrt(l, elm, ind):***

*"""*

*>>> l = link(11, link(12, link(13, empty)))*

*>>> n = insrt (l, 2021, 1)*

*>>> n*

*[11, [2021, [12, [13, [ ] ]]]]*

*>>> m = insrt(n, 2022, 20)*

*>>> m*

*[11 [2021 [12 [13 [2022, [ ] ]]]]]*

*"""*

***Program***

***def link(value, next\_link):***

***return [value, next\_link]***

***def is\_empty(s):***

***return s == []***

***def first(s):***

***return s[0]***

***def rest(s):***

***return s[1]***

***def insrt(l, elm, ind):***

***if ind == 0 or is\_empty(l):***

***return link(elm, l)***

***else:***

***return link(first(l), insrt(rest(l), elm, ind - 1))***

***def convert\_to\_list(lnk):***

***def helper(lst):***

***if is\_empty(lst):***

***return []***

***else:***

***return [first(lst), helper(rest(lst))]***

***return helper(lnk)***

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

***l = link(7, link(9, link(12, [])))***

***n = insrt(l, 2023, 2)***

***print(convert\_to\_list(n))***

***m = insrt(n, 2024, 0)***

***print(convert\_to\_list(m))***

***Result***

A white box with black text

Description automatically generated