Day 2 lab python programs

1.def climbStairs(n):

if n == 1:

return 1

dp = [0] \* (n + 1)

dp[1] = 1

dp[2] = 2

for i in range(3, n + 1):

dp[i] = dp[i - 1] + dp[i - 2]

return dp[n]

# Test cases

print(climbStairs(2)) # Output: 2

print(climbStairs(3)) # Output: 3

print(climbStairs(4)) # Output: 5

print(climbStairs(1)) # Output: 1

print(climbStairs(5)) # Output: 8

2.

def is\_leap\_year(year):

if year % 400 == 0:

return True

if year % 100 == 0:

return False

if year % 4 == 0:

return True

return False

# Input

year = int(input("Enter a year: "))

# Check if it's a leap year and print the output

if is\_leap\_year(year):

print("Given year is a Leap Year")

else:

print("Given year is a Non Leap Year")

3.

def max\_words\_in\_sentence(sentences):

max\_words = 0

for sentence in sentences:

words = sentence.split()

max\_words = max(max\_words, len(words))

return max\_words

# Test cases

print(max\_words\_in\_sentence(["alice and bob love leetcode", "i think so too", "this is great thanks very much"])) # Output: 6

print(max\_words\_in\_sentence(["please wait", "continue to fight", "continue to win"])) # Output: 3

print(max\_words\_in\_sentence(["the heads", "of", "two", "sorted linked lists"])) # Output: 4

print(max\_words\_in\_sentence(["python", "is", "an object-oriented programming language"])) # Output: 5

print(max\_words\_in\_sentence(["python", "is", "an interactive language"])) # Output: 3

4.class ListNode:

def \_\_init\_\_(self, val=0, next=None):

self.val = val

self.next = next

def mergeTwoLists(list1, list2):

dummy = ListNode(-1)

current = dummy

while list1 and list2:

if list1.val < list2.val:

current.next = list1

list1 = list1.next

else:

current.next = list2

list2 = list2.next

current = current.next

current.next = list1 or list2

return dummy.next

# Helper function to print linked list

def print\_linked\_list(head):

result = []

while head:

result.append(head.val)

head = head.next

return result

# Test cases

list1 = ListNode(1, ListNode(2, ListNode(4)))

list2 = ListNode(1, ListNode(3, ListNode(4)))

result = mergeTwoLists(list1, list2)

print(print\_linked\_list(result)) # Output: [1, 1, 2, 3, 4, 4]

list1 = None

list2 = None

result = mergeTwoLists(list1, list2)

print(print\_linked\_list(result)) # Output: []

list1 = None

list2 = ListNode(0)

result = mergeTwoLists(list1, list2)

print(print\_linked\_list(result)) # Output: [0]

list1 = None

list2 = ListNode(1, ListNode(2, ListNode(3, ListNode(4, ListNode(5)))))

result = mergeTwoLists(list1, list2)

print(print\_linked\_list(result)) # Output: [1, 2, 3, 4, 5]

list1 = ListNode(0, ListNode(1, ListNode(9)))

list2 = ListNode(3, ListNode(4, ListNode(5)))

result = mergeTwoLists(list1, list2)

print(print\_linked\_list(result)) # Output: [0, 1, 3, 4, 5, 9]

5.

def calculate(s):

stack = []

num = 0

sign = '+'

for i in range(len(s)):

char = s[i]

if char.isdigit():

num = num \* 10 + int(char)

if (not char.isdigit() and char != ' ') or i == len(s) - 1:

if sign == '+':

stack.append(num)

elif sign == '-':

stack.append(-num)

elif sign == '\*':

stack[-1] \*= num

elif sign == '/':

stack[-1] = int(stack[-1] / num)

num = 0

sign = char

return sum(stack)

# Test cases

print(calculate("3+2\*2")) # Output: 7

print(calculate(" 3/2 ")) # Output: 1

print(calculate(" 3+5 / 2 ")) # Output: 5

print(calculate("-1+5")) # Output: 4

print(calculate("2+3+5")) # Output: 10

7.

def generateParenthesis(n):

def backtrack(s, left, right):

if len(s) == 2 \* n:

result.append(s)

return

if left < n:

backtrack(s + '(', left + 1, right)

if right < left:

backtrack(s + ')', left, right + 1)

result = []

backtrack('', 0, 0)

return result

# Test cases

print(generateParenthesis(3)) # Output: ["((()))","(()())","(())()","()(())","()()()"]

print(generateParenthesis(1)) # Output: ["()"]

print(generateParenthesis(5)) # Output: ["((((()))))","(((()())))","(((())()))","(((()))())","((((()))))","((((())))",

# "(((())))()","((()(())))","((()()))()","((())(()))","((())())()","((()))(())",

# "(()((())))","(()(()()))","(()(())())","(()(()))()","(()()(()))","(()()())()",

# "(()())(())","(())((()))","(())(()())","(())(())()","()(())(())","()((()))()",

# "()(()())()","()((())())","()((()))()","()()(()())","()()((()))","()()(())()",

# "()()()()"]

print(generateParenthesis(-1)) # Output: []

print(generateParenthesis(0)) # Output: []

8.

def isMatch(s, p):

dp = [[False] \* (len(p) + 1) for \_ in range(len(s) + 1)]

dp[0][0] = True

for i in range(len(s) + 1):

for j in range(1, len(p) + 1):

if p[j - 1] == '\*':

dp[i][j] = dp[i][j - 2] or (i > 0 and (s[i - 1] == p[j - 2] or p[j - 2] == '.') and dp[i - 1][j])

else:

dp[i][j] = i > 0 and (s[i - 1] == p[j - 1] or p[j - 1] == '.') and dp[i - 1][j - 1]

return dp[len(s)][len(p)]

# Test cases

print(isMatch("aa", "a")) # Output: False

print(isMatch("aa", "a\*")) # Output: True

print(isMatch("ab", ".\*")) # Output: True

print(isMatch("aaa", "aa")) # Output: False

print(isMatch("aab", "c\*a\*b")) # Output: True

9.

def get\_season(month, day):

if (month == "January" and day >= 20) or (month == "February"):

return "Winter"

elif (month == "February" and day < 20) or (month == "March" or month == "April"):

return "Spring"

elif (month == "April" and day < 21) or (month == "May" or month == "June"):

return "Summer"

elif (month == "June" and day < 21) or (month == "July" or month == "August"):

return "Fall"

else:

return "Winter"

# Input

month = input("Enter the month: ").capitalize()

day = int(input("Enter the date: "))

# Determine and output the season

season = get\_season(month, day)

print("The season is currently", season)

10.

def remove\_common\_words(s1, s2):

words\_s1 = set(s1.split())

words\_s2 = set(s2.split())

common\_words = words\_s1.intersection(words\_s2)

s1\_modified = ' '.join([word for word in s1.split() if word not in common\_words])

s2\_modified = ' '.join([word for word in s2.split() if word not in common\_words])

return s1\_modified, s2\_modified

# Test cases

test\_cases = [

("sky is blue in color", "Raj likes sky blue color"),

("learn python", "python is easy to learn"),

("raju likes apple", "apple is red in color"),

("sita likes orange", "orange is rich in anti-oxidants"),

("raj is travelling to Chennai in train", "the rain will reach Chennai at 8 pm")

]

for s1, s2 in test\_cases:

modified\_s1, modified\_s2 = remove\_common\_words(s1, s2)

print("Output:")

print(modified\_s1)

print(modified\_s2)

print()