**BY: Fadi Alahmad Alomar 120180049**

**# Question 2**  
function expressionChecker(s):  
 """  
 this function uses one helper stack and two arrays one with the opening bracket and one with a bracket pair   
 if the next element in the string is a closing brackets   
 it is taking with the last element in the stack and checking   
 if they are a bracket pair in the closing array if not then the  
 string is invalid if they are then we pop the last element  
 of the stack   
 """  
 myStack <- Stack() # initializing an aux stack  
 op <- ["{","[","("] # an array with the opening brackets in it  
 cl <- ["{}", "[]", "()"] # an array with the brackets as pairs  
 for i<-0: s.length, do # iterating through the string character by character  
 check1 <- True # a flag value to check if the character is in the op array  
 for j<-0:3, do # iterating through the array  
 if s[i] equal op[j],do # checking if the character is in the op array  
 myStack.push(s[i]) # pushing the char to the stack  
 check1 <- False # changing the flag value  
 break # breaking out if the for loop  
 end  
 end  
 if check1, do # checking to see if the char was found in the op array using the flag  
 if !myStack.isEmpty(), do # checking to see if the stack is empty  
 st = myStack.peek() + s[i] # constructing a string from the last element in the stack and the character in the string iteration  
 check2 = True # a flag value to see if the constructed string was found in the array cl  
 for j<-0:3, do # iterating through the cl array  
 if st equal cl[j], do # checking if the constructed string is in the cl array  
 myStack.pop() # popping the last element in the stack  
 check2=False # changing the flag value  
 break # breaking out of the for loop  
 end  
 end  
 if check2, do # if the constructed string was not found in cl returning Invalid  
 return "Invalid"  
 end  
 else, do # if the stack is empty return invalid  
 return "Invalid"  
 end  
 end  
 return "Valid" # if the operation is done successfully through the string then it is valid  
  
  
  
**# Question 6:**  
  
function Sum(st):  
 """  
 this function returns the sum of all the elements in a stack by place holding them in an aux stack   
 then re-pushing them to the original one  
 """  
 auxStack <- Stack() # initializing an aux stack  
 s<-0 # initializing a sum value  
 while !st.isEmpty(), do # while the stack is not empty do  
 p <- st.pop() # pop from the stack and assign it to a variable  
 s <- s+p # incrementing the sum value with the popped element  
 auxStack.push(p) # pushing the popped element into the aux stack  
 end  
 while !auxStack.isEmpty(), do # while the aux stack is not empty do  
 st.push(auxStack.pop()) # pop and element from the aux stack and push it to the original stack  
 end  
 return s # return the sum  
  
**# Question 7:**  
  
function palindromeCheck(s):  
 """  
 this function takes a string and pushes its element to a stack thus they are inverted   
 and starts popping from the stack and iterating through the list if the string   
 and the stack which holds the same string but are the same then it is palindrome   
 """  
 auxStack<- Stack() # initializing an aux stack  
 for i<-0:s.length() ,do # iterating through the string to add it to the aux stack  
 auxStack.push(s[i]) # adding the ith element of the string to the stack  
 end  
 for i <- 0: s.length(), do # iterating through teh string  
 if s[i] not equal aux.pop(),do # checking to see if the the element is equal to the popped element from the stack  
 return False # if one element mismatches then return False  
 end  
 end  
 return True # if the operation is completed then the entire string is palindrome  
  
**# Question 8:**  
function insert\_sorted(st, val):  
 """  
 this function inserts the value while keeping the sorted order by using an aux stack to hold all items bigger   
 than the value then adding the value to the original stack and pushing all elements in the aux stack back to the original one  
 """  
 auxStack <- Stack() # initializing an aux stack  
 while !st.isEmpty() and val > st.peek(), do# while the stack is not empty and the val is bigger that the one on top of the stack  
 auxStack.push(st.pop()) # pop the first element from the original stack and pushing it to the aux stack  
 end  
 st.push(val) # pushing the wanted value  
 while !auxStack.isEmpty(), do # while the aux stack is not empty do  
 st.push(auxStack.pop()) # pop the element from the aux stack and pushing it to the original stack  
 end  
 return st # returning the original stack

**# Question 9:**  
class Stack:  
 def \_\_init\_\_(self, ma=10000):  
 self.s = [None] \* ma  
 self.last = -1  
 self.ma = ma  
  
 def isEmpty(self):  
 if self.last < 0:  
 return True  
 return False  
  
 def isFull(self):  
 if self.last + 1 == self.ma:  
 return True  
 return False  
  
 def push(self, val):  
 if self.isFull():  
 raise Exception("Stack is full")  
 else:  
 self.last += 1  
 self.s[self.last] = val  
  
 def peek(self):  
 if not self.isEmpty():  
 return self.s[self.last]  
 else:  
 return  
  
 def pop(self):  
 if not self.isEmpty():  
 self.last -= 1  
 return self.s[self.last + 1]  
 else:  
 raise Exception("Stack is empty")