**Assignment**

**Write a program to find all pairs of an integer array whose sum is equal to a given number?**

In [ ]:

**class** **Pairs**():

**def** \_\_init\_\_(self, sum, n, arr ):

self.sum = sum

self.n = n

self.arr = arr

**def** find\_pairs(self):

**for** i **in** range(0, self.n):

**for** j **in** range(i+1, self.n):

**if** (self.arr[i] + self.arr[j] == self.sum):

print("(", self.arr[i], ",", " ", self.arr[j], ")", sep = "")

**def** Main():

arr=[]

count=int(input('Please enter the number of element in array'))

print("Please enter the numbers:")

**for** i **in** range(count):

ele=int(input())

arr.append(ele)

print(arr)

sum = int(input("Enter the sum for array elements: "))

n = len(arr)

obj\_Pairs = Pairs(sum, n, arr)

obj\_Pairs.find\_pairs()

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

Main()

**Write a program to reverse an array in place? In place means you cannot create a new array. You have to update the original array.**

In [ ]:

arr=[]

count=int(input('Please enter the number of element in array'))

print("Please enter the numbers:")

**for** i **in** range(count):

ele=int(input())

arr.append(ele)

print("Array is :",arr)

res = arr[::-1]

print("Resultant new reversed array:",res)

**Write a program to check if two strings are a rotation of each other?**

In [ ]:

**def** areRotations(string1, string2):

size1 = len(string1)

size2 = len(string2)

temp = ''

**if** size1 != size2:

**return** 0

temp = string1 + string1

**if** (temp.count(string2)> 0):

**return** 1

**else**:

**return** 0

string1 = input()

string2 = input()

**if** areRotations(string1, string2):

print ("Strings are rotations of each other")

**else**:

print ("Strings are not rotations of each other")

**Write a program to print the first non-repeated character from a string?**

In [ ]:

NO\_OF\_CHARS = 256

**def** getCharCountArray(string):

count = [0] \* NO\_OF\_CHARS

**for** i **in** string:

count[ord(i)]+= 1

**return** count

**def** firstNonRepeating(string):

count = getCharCountArray(string)

index = -1

k = 0

**for** i **in** string:

**if** count[ord(i)] == 1:

index = k

**break**

k += 1

**return** index

string = input()

index = firstNonRepeating(string)

**if** index == 1:

print ("Either all characters are repeating or string is empty")

**else**:

print ("First non-repeating character is " + string[index])

**Read about the Tower of Hanoi algorithm. Write a program to implement it.**

In [ ]:

**def** Tower\_Of\_Hanoi(n , from\_rod, to\_rod, aux\_rod):

**if** n == 1:

print("Move disk 1 from rod",from\_rod,"to rod",to\_rod)

**return**

TowerOfHanoi(n-1, from\_rod, aux\_rod, to\_rod)

print("Move disk",n,"from rod",from\_rod,"to rod",to\_rod)

TowerOfHanoi(n-1, aux\_rod, to\_rod, from\_rod)

n = int(input("Please enter the number of disks:**\n**"))

Tower\_Of\_Hanoi(n, 'A', 'C', 'B')

**Write a program to convert postfix to prefix expression.**

In [ ]:

**def** isOperator(x):

**if** x == "+":

**return** **True**

**if** x == "-":

**return** **True**

**if** x == "/":

**return** **True**

**if** x == "\*":

**return** **True**

**return** **False**

**def** postToPre(post\_exp):

s = []

length = len(post\_exp)

**for** i **in** range(length):

**if** (isOperator(post\_exp[i])):

op1 = s[-1]

s.pop()

op2 = s[-1]

s.pop()

temp = post\_exp[i] + op2 + op1

s.append(temp)

**else**:

s.append(post\_exp[i])

ans = ""

**for** i **in** s:

ans += i

**return** ans

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

post\_exp = input("Please enter the postfix equation:**\n**")

print("Prefix : ", postToPre(post\_exp))

**Write a program to convert prefix expression to infix expression.**

In [1]:

**def** prefixToInfix(prefix):

stack = []

i = len(prefix) - 1

**while** i >= 0:

**if** **not** isOperator(prefix[i]):

stack.append(prefix[i])

i -= 1

**else**:

str = "(" + stack.pop() + prefix[i] + stack.pop() + ")"

stack.append(str)

i -= 1

**return** stack.pop()

**def** isOperator(c):

**if** c == "\*" **or** c == "+" **or** c == "-" **or** c == "/" **or** c == "^" **or** c == "(" **or** c == ")":

**return** **True**

**else**:

**return** **False**

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

str = input("Please enter the prefix equation:**\n**")

print('Infix equatin is',prefixToInfix(str))

Please enter the prefix equation:

\*-A/BC-/AKL

Infix equatin is ((A-(B/C))\*((A/K)-L))

**Write a program to check if all the brackets are closed in a given code snippet.**

In [ ]:

**def** areBracketsBalanced(expr):

stack = []

**for** char **in** expr:

**if** char **in** ["(", "{", "["]:

stack.append(char)

**else**:

**if** **not** stack:

**return** **False**

current\_char = stack.pop()

**if** current\_char == '(':

**if** char != ")":

**return** **False**

**if** current\_char == '{':

**if** char != "}":

**return** **False**

**if** current\_char == '[':

**if** char != "]":

**return** **False**

**if** stack:

**return** **False**

**return** **True**

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

expr = input("Please enter the expression:**\n**")

**if** areBracketsBalanced(expr):

print("Balanced")

**else**:

print("Not Balanced")

**Write a program to reverse a stack.**

In [2]:

**def** insertAtBottom(stack, item):

**if** isEmpty(stack):

push(stack, item)

**else**:

temp = pop(stack)

insertAtBottom(stack, item)

push(stack, temp)

**def** reverse(stack):

**if** **not** isEmpty(stack):

temp = pop(stack)

reverse(stack)

insertAtBottom(stack, temp)

**def** createStack():

stack = []

**return** stack

**def** isEmpty( stack ):

**return** len(stack) == 0

**def** push( stack, item ):

stack.append( item )

**def** pop( stack ):

**if**(isEmpty( stack )):

print("Stack Underflow ")

exit(1)

**return** stack.pop()

**def** prints(stack):

**for** i **in** range(len(stack)-1, -1, -1):

print(stack[i], end = ' ')

print()

stack = createStack()

count=int(input("Please enter the desired size of stack:"))

**for** i **in** range(count):

ele=input()

push(stack,(ele))

print("Original Stack ",stack)

reverse(stack)

print("Reversed Stack ",stack)

Please enter the desired size of stack:4

12

2

3

Original Stack ['12', '', '2', '3']

Reversed Stack ['3', '2', '', '12']

**Write a program to find the smallest number using a stack.**

In [ ]:

**class** **Node**:

**def** \_\_init\_\_(self, value):

self.value = value

self.next = **None**

**def** \_\_str\_\_(self):

**return** "Node(**{}**)".format(self.value)

\_\_repr\_\_ = \_\_str\_\_

**class** **Stack**:

**def** \_\_init\_\_(self):

self.top = **None**

self.count = 0

self.minimum = **None**

**def** \_\_str\_\_(self):

temp = self.top

out = []

**while** temp:

out.append(str(temp.value))

temp = temp.next

out = '**\n**'.join(out)

**return** ('Top **{}** **\n\n**Stack :**\n{}**'.format(self.top,out))

\_\_repr\_\_=\_\_str\_\_

**def** getMin(self):

**if** self.top **is** **None**:

**return** "Stack is empty"

**else**:

print("**\n**Minimum Element in the stack is: **{}**" .format(self.minimum))

**def** isEmpty(self):

**if** self.top == **None**:

**return** **True**

**else**:

**return** **False**

**def** \_\_len\_\_(self):

self.count = 0

tempNode = self.top

**while** tempNode:

tempNode = tempNode.next

self.count+=1

**return** self.count

**def** peek(self):

**if** self.top **is** **None**:

print ("Stack is empty")

**else**:

**if** self.top.value < self.minimum:

print("Top Most Element is: **{}**" .format(self.minimum))

**else**:

print("Top Most Element is: **{}**" .format(self.top.value))

**def** push(self,value):

**if** self.top **is** **None**:

self.top = Node(value)

self.minimum = value

**elif** value < self.minimum:

temp = (2 \* value) - self.minimum

new\_node = Node(temp)

new\_node.next = self.top

self.top = new\_node

self.minimum = value

**else**:

new\_node = Node(value)

new\_node.next = self.top

self.top = new\_node

print("Number Inserted: **{}**" .format(value))

**def** pop(self):

**if** self.top **is** **None**:

print( "Stack is empty")

**else**:

removedNode = self.top.value

self.top = self.top.next

**if** removedNode < self.minimum:

print ("Top Most Element Removed :**{}** " .format(self.minimum))

self.minimum = ( ( 2 \* self.minimum ) - removedNode )

**else**:

print ("Top Most Element Removed : **{}**" .format(removedNode))

stack = Stack()

count=int(input("Please enter the desired size of stack:"))

**for** i **in** range(count):

ele=int(input("**\n**Enter the number:"))

stack.push(ele)

stack.getMin()