Que_1: Pairs.

Code:

```
def findPair(nums, target):


    for i in range(len(nums) - 1):


        for j in range(i + 1, len(nums)):

            if nums[i] + nums[j] == target:
                print('Pair found', (nums[i], nums[j]))
                return


    print('Pair not found')

if __name__ == '__main__':

    nums = [8, 7, 2, 5, 3, 1]
    target = 10

    findPair(nums, target)
```

Que_2: Array Reverse

Code:

```
from array import *
array_num = array('i', [1, 3, 5, 3, 7, 1, 9, 3])
print("Original array: "+str(array_num))
array_num.reverse()
print("Reverse the order of the items:")
print(str(array_num)
```

Que_3: rotationoftwostr

Code:

```
def checkRotation(s1, s2):
    temp = ''


    if len(s1) != len(s2):
        return False


    temp = s1 + s1


    if s2 in temp:
        return True
    else:
        return False
```

```
string1 = "HELLO"
string2 = "LOHEL"

if checkRotation(string1, string2):
    print("Given Strings are rotations of each other.")
else:
    print("Given Strings are not rotations of each other.")
```

Que_4: firstnonrepeatstr

Code:

```
def firstNonRepeatingChar(str1):
  char_order = []
  counts = {}
  for c in str1:
    if c in counts:
      counts[c] += 1
    else:
      counts[c] = 1
      char_order.append(c)
  for c in char_order:
    if counts[c] == 1:
     return c
  return None

print(firstNonRepeatingChar(input("Enter a String ")))
```

Que_5:Tower of hanoi

code:

```
def hanoi(disks, source, auxiliary, target):
    if disks == 1:
        print('Move disk 1 from peg {} to peg {}.'.format(source, target))
        return

    hanoi(disks - 1, source, target, auxiliary)
    print('Move disk {} from peg {} to peg {}.'.format(disks, source, target))
    hanoi(disks - 1, auxiliary, source, target)


disks = int(input('Enter number of disks: '))
hanoi(disks, 'A', 'B', 'C')
```

Que_6: PostfixtoPrefix

Code:

```
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 = "AB+CD-"


    print("Prefix : ", postToPre(post_exp))
```

Que_7: PrefixtoInfix

Code:

```
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 = "*-A/BC-/AKL"
    print(prefixToInfix(str))
```

Que_8: Bracketsclose

Code:

```
def isbalanced(s):
  c= 0
  ans=False
  for i in s:
    if i == "(":
     c += 1
    elif i == ")":
     c-= 1
    if c < 0:
     return ans
    if c==0:
     return not ans
  return ans
s="{[]}"
print("Given string brackets is closed :",isbalanced(s))
```

Que_9: reversing a stack

Code:

```
class Stack_structure:
  def __init__(self):
    self.items = []

  def check_empty(self):
    return self.items == []

  def push_val(self, data):
    self.items.append(data)

  def pop_val(self):
    return self.items.pop()

  def print_it(self):
    for data in reversed(self.items):
      print(data)

def insert_bottom(instance, data):
  if instance.check_empty():
    instance.push_val(data)
  else:
    deleted_elem = instance.pop_val()
    insert_bottom(instance, data)
    instance.push_val(deleted_elem)

def stack_reverse(instance):
  if not instance.check_empty():
    deleted_elem = instance.pop_val()
    stack_reverse(instance)
    insert_bottom(instance, deleted_elem)

my_instance = Stack_structure()
data_list = input('Enter the elements to add to the stack: ').split()
for data in data_list:
  my_instance.push_val(int(data))

print('The reversed stack is:')
my_instance.print_it()
stack_reverse(my_instance)
print('The stack is:')
my_instance.print_it()
```

Que_10: smallestnousingstack

Code:

```
from collections import deque


class MinStack:
 def __init__(self):

  self.s = deque()
```

```
  self.min = None


 def push(self, val):
  if not self.s:
   self.s.append(val)
   self.min = val
  elif val > self.min:
   self.s.append(val)
  else:
   self.s.append(2*val - self.min)
   self.min = val


 def pop(self):
  if not self.s:
   self.print('Stack underflow!!')
   exit(-1)
  top = self.s[-1]
  if top < self.min:
   self.min = 2*self.min - top
  self.s.pop()


 def getMin(self):
  return self.min


if __name__ == '__main__':

 s = MinStack()

 s.push(6)
 print(s.getMin())

 s.push(7)
 print(s.getMin())

 s.push(5)
 print(s.getMin())

 s.push(3)
 print(s.getMin())

 s.pop()
 print(s.getMin())

 s.pop()
 print(s.getMin())
```