1. A positive integer m can be partitioned as primes if it can be written as p + q where p > 0, q > 0 and both p and q are prime numbers. Write a Python function primepartition(m) that takes an integer m as input and returns True if m can be partitioned as primes and False otherwise. (If m is not positive, your function should return False.)

```
def primepartition(m):
    primelist=[]
    if m<0:
        return False
    else:
        for i in range(2,m + 1):
            for p in range(2,i):
                if (i % p) == 0:
                    break
            else:
                primelist.append(i)


        for x in primelist:
            y= m-x
            if y in primelist:
                return True
        return False
```

2. Write a function matched(s) that takes as input a string s and checks if the brackets "(" and ")" in s are matched: that is, every "(" has a matching ")" after it and every ")" has a matching "(" before it. Your function should ignore all other symbols that appear in s. Your function should return True if s has matched brackets and False if it does not.

```
def matched(n):
    n=list(n)
    c=0
    for i in range(len(n)):
        if c==-1:
            return False
```

```
        if n[i]=="(":
            c=c+1
        if n[i]==")":
            c=c-1
    if c==0:
        return True
    else:
        return False
```

3. A list rotation consists of taking the first element and moving it to the end. For instance, if we rotate the list [1,2,3,4,5], we get [2,3,4,5,1]. If we rotate it again, we get [3,4,5,1,2]. Write a Python function rotatelist(l,k) that takes a list l and a positive integer k and returns the list l after k rotations. If k is not positive, your function should return l unchanged. Note that your function should not change l itself, and should return the rotated list.

```
def rotatelist(l,k):
    k=k%(len(l));
    return l[k:]+l[:k];
```

4. Define a Python function remdup(l) that takes a nonempty list of integers l and removes all duplicates in l, keeping only the last occurrence of each number. For instance:

```
def remdup(l):
  m=l[:]
  for w in range(len(l)-1):
    if l[w] in l[w+1:]:
        m.remove(l[w])
  return(m)
```

5. Write a Python function splitsum(l) that takes a nonempty list of integers and returns a list [pos,neg], where pos is the sum of squares all the positive numbers in l and neg is the sum of cubes of all the negative numbers in l

```
def splitsum(l):
  pos=0
  neg=0
  for v in l:
```

```
  if v >0:
    pos=pos+v*v
  if v< 0:
    neg=neg+v*v*v
 return ([pos,neg])
```

7. We have a list of annual rainfall recordings of cities. Each element in the list is of the form (c,r) where c is the city and r is the annual rainfall for a particular year. The list may have multiple entries for the same city, corresponding to rainfall recordings in different years. Write a Python function rainaverage(l) that takes as input a list of rainfall recordings and computes the avarage rainfall for each city. The output should be a list of pairs (c,ar) where c is the city and ar is the average rainfall for this city among the recordings in the input list. Note that ar should be of type float. The output should be sorted in dictionary order with respect to the city name.

```
def rainaverage(l):
 (t1,t2,t3)=({},{},{})
 for i,j in l:
  if i not in t1:
    (t1[i],t2[i])=(j,1)
  else:
    (t1[i], t2[i]) = (t1[i]+j,t2[i]+1)
 for i,j in t2.items():
  t3[i]=float(t1[i]/j)
 l2=[(i,j) for i,j in t3.items()]
 l2.sort(key=lambda  a:a[0])
 return l2
```

8. A list in Python can contain nested lists. The degree of nesting need not be uniform. For instance [1,2,[3,4,[5,6]]] is a valid Python list. Write a Python function flatten(l) that takes a nonempty list of lists and returns a simple list of all the elements in the nested lists, flattened out. You can make use of the following function that returns True if its input is of type list

```
def listtype(l):
 return type(1) == type([])
l3=[]
def flatten(l4):
 for i in l4:
```

```
    if listtype(i):
        flatten(i)
    else:
        l3.append(i)
  return l3
```

9. The library at the Hogwarts School of Witchcraft and Wizardry has computerized its book issuing process. The relevant information is provided as text from standard input in three parts: information about books, information about borrowers and information about checkouts. Each part has a specific line format, described below.
Information about books

Line format: Accession Number~Title

Information about borrowers

Line format: Username~Full Name

Information about checkouts

Line format: Username~Accession Number~Due Date

Note: Due Date is in YYYY-MM-DD format.

You may assume that the data is internally consistent. For every checkout, there is a corresponding username and accession number in the input data, and no book is simultaneously checked out by two people.

Each section of the input starts with a line containing a single keyword. The first section begins with a line containing Books. The second section begins with a line containing Borrowers. The third section begins with a line containing Checkouts. The end of the input is marked by a line containing EndOfInput.

Write a Python program to read the data as described above and print out details about books that have been checked out. Each line should describe to one currently issued book in the following format:

Due Date~Full Name~Accession Number~Title

Your output should be sorted in increasing order of due date. For books due on the same date, sort in increasing order of full name.

```
def sortdates(checkout):
    key = list(map(int,checkout[1].split('-')))
    #print(key)
    key.append(borrowers[checkout[0][0]])
    key.append(checkout[0][1])
    print (tuple(key))
    return tuple(key)
```

```
books = dict()
borrowers = dict()
checkouts = dict()
temp = None
while(True):
    line = input()
    if(line == 'Books'):
        temp = 'Books'
        continue
    elif(line == 'Borrowers'):
        temp = 'Borrowers'
        continue
    elif(line == 'Checkouts'):
        temp = 'Checkouts'
        continue
    elif(line == 'EndOfInput'):
        break
    elif(temp == 'Books'):
        access_no,title = line.split('~')
        books[access_no] = title
    elif(temp == 'Borrowers'):
        usercode,name = line.split('~')
        borrowers[usercode] = name
    elif(temp == 'Checkouts'):
        usercode,access_no,due_date = line.split('~')
        checkouts[(usercode,access_no)] = due_date

checkouts = sorted(checkouts.items(), key=sortdates)
```

```
for checkout in checkouts:
    line = checkout[1]+'~'+borrowers[checkout[0][0]]+'~'+\
          checkout[0][1]+'~'+books[checkout[0][1]]
    print(line)
```

10. Write a function intreverse(n) that takes as input a positive integer n and returns the integer obtained by reversing the digits in n.

```
def intreverse(n):
  a=""
  while n>0:
    b=str(n%10)
    a=a+b
    n=n//10
  return int(a)
```

12. Write a function sumprimes(l) that takes as input a list of integers l and retuns the sum of all the prime numbers in l.

```
def sumprimes(l):
    sum=0
    for i in range(0, len(l)):
        num=l[i]
        flag=0
        for j in range(2, int(num/2)+1):
            if (num % j )== 0:
                flag=1
                break
        if(flag==0 and (num!=1) and num>=0):
            sum+=num
    return sum
```

13. Write a function contracting(l) that takes as input a list of integer l and returns True if the absolute difference between each adjacent pair of elements strictly decreases.

```
def contracting(l):


for z in range(0,len(l)-3):
   A=abs(l[z+2]-l[z+1])
   B=abs(l[z+1]-l[z])
   if A<B:
      continue
   else:
      return (False)
return (True)
```

14. In a list of integers l, the neighbours of l[i] are l[i-1] and l[i+1]. l[i] is a hill if it is strictly greater than its neighbours and a valley if it is strictly less than its neighbours.

Write a function counthv(l) that takes as input a list of integers l and returns a list [hc,vc] where hc is the number of hills in l and vc is the number of valleys in l.

```
def counthv(l):
 hc=0
 vc=0
 if len(l)>2:
  for i in range(1,len(l)-1):
    if l[i]>l[i-1] and l[i]>l[i+1]:
      hc+=1
    elif l[i]<l[i-1] and l[i]<l[i+1]:
      vc+=1
 return [hc,vc]
```

15. A square n×n matrix of integers can be written in Python as a list with n elements, where each element is in turn a list of n integers, representing a row of the matrix. For instance, the matrix 1.

1 2 3

4 5 6

7 8 9

2. would be represented as

[[1,2,3], [4,5,6], [7,8,9]].

3. Write a function leftrotate(m) that takes a list representation m of a square matrix as input, and returns the matrix obtained by rotating the original matrix counterclockwize by 90 degrees.

For instance, if we rotate the matrix above, we get

4.

3 6 9

2 5 8

1 4 7

5. Your function should not modify the argument m provided to the function rotate().

Method 1:

```
def leftrotate(m):
    b=[]
    for i in range(len(m)):
        b.append([])
        for j in range(len(m)):
            b[i].append(m[j][len(m)-i-1])
    return(b)
```

Method 2

```
def leftrotate(a):
    new_matrix = []
    for i in range(1, len(a)+1):
        temp = []
        for j in range(len(a)):
            temp.append(a[j][-i])
        new_matrix.append(temp)
    return new_matrix
```

Method 3

```
def leftrotate(m) :
    N = len(m)
    for i in range(0, int(N / 2)):
        for j in range(i, N-i-1):
            temp = m[i][j]
            m[i][j] = m[j][N-1-i]
            m[j][N-1-i] = m[N-1-i][N-1-j]
            m[N-1-i][N-1-j] = m[N-1-j][i]
            m[N-1-j][i] = temp
    return m
```