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.)

```python
def is_prime(n):
    if n < 2:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True

def primepartition(m):
    if m <= 0:
        return False
    for i in range(2, m):
        if is_prime(i) and is_prime(m - i):
            return True
    return False
primepartition(185)
primepartition(7)
```

```
True
```

**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. Here are some examples to show how your function should work.

> matched("zb%78") True
>
> matched("(7)(a") False **

```python
def matched(s):
    stack = []
    for char in s:
        if char == '(':
            stack.append(char)
        elif char == ')':
            if len(stack) == 0 or stack[-1] != '(':
                return False
            stack.pop()
    return len(stack) == 0
matched("zb%78")
matched("a)*(?")
```

```
False
```

1. 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. Here are some examples to show how your function should work.

   rotatelist([1,2,3,4,5],1) [2, 3, 4, 5, 1]

rotatelist([1,2,3,4,5],3) [4, 5, 1, 2, 3]

```
def rotatelist(l, k):
    if k <= 0:
        return l
    n = len(l)
    rotations = k % n
    return l[rotations:] + l[:rotations]

rotatelist([1,2,3,4,5],1)
rotatelist([1,2,3,4,5],3)
```

```
[4, 5, 1, 2, 3]
```

1. 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:

   remdup([3,1,3,5]) [1, 3, 5]

remdup([7,3,-1,-5]) [7, 3, -1, -5]

```
def remdup(l):
    result = []
    seen = set()
    for num in reversed(l):
        if num not in seen:
            result.append(num)
            seen.add(num)
    return list(reversed(result))
remdup([3,1,3,5])
```

```
[1, 3, 5]
```

```
def remdup(l):
  result=[]
  for num in reversed(l):
    if num not in result:
      result.append(num)
  return list(reversed(result))

remdup([1,2,3,1,4,5,6,2])
```

```
[3, 1, 4, 5, 6, 2]
```

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. Here are some examples to show how your function should work.

splitsum([1,3,-5]) [10, -125]

splitsum([2,4,6]) [56, 0]

```
def splitsum(l):
    pos = sum([num**2 for num in l if num > 0])
    neg = sum([num**3 for num in l if num < 0])
    return [pos, neg]
splitsum([1,3,-5])
```

```
[10, -125]
```

1. 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. Here are some examples to show how your function should work.

   splitsum([1,3,-5]) [10, -125]

splitsum([2,4,6]) [56, 0]

splitsum([-19,-7,-6,0]) [0, -7418]

```
def splitsum(l):
    pos = sum([num**2 for num in l if num > 0])
    neg = sum([num**3 for num in l if num < 0])
    return [pos, neg]
splitsum([2,4,6])
```

```
[56, 0]
```

1. 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. Here are some examples to show how rainaverage(l) should work.

   rainaverage([(1,2),(1,3),(2,3),(1,1),(3,8)]) [(1, 2.0), (2, 3.0), (3, 8.0)]

rainaverage([('Bombay',848),('Madras',103),('Bombay',923),('Bangalore',201),('Madras',128)]) [('Bangalore', 201.0), ('Bombay', 885.5), ('Madras', 115.5)]

```
def rainaverage(l):
    city_totals = {}  # Dictionary to store total rainfall for each
city
    city_counts = {}  # Dictionary to store the count of rainfall
recordings for each city

    for city, rainfall in l:
        if city in city_totals:
            city_totals[city] += rainfall
            city_counts[city] += 1
        else:
            city_totals[city] = rainfall
            city_counts[city] = 1

    average_rainfall = []
    for city in sorted(city_totals.keys()):
        total = city_totals[city]
        count = city_counts[city]
        average = total / count  # Compute average rainfall
        average_rainfall.append((city, average))

    return average_rainfall

rainaverage([(1,2),(1,3),(2,3),(1,1),(3,8)])
rainaverage([('Bombay',848),('Madras',103),('Bombay',923),
('Bangalore',201),('Madras',128)])
```

```
[('Bangalore', 201.0), ('Bombay', 885.5), ('Madras', 115.5)]
```

1. 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(l) == type([])) Here are some examples to show how flatten(l) should work.

flatten([1,2,[3],[4,[5,6]]]) [1, 2, 3, 4, 5, 6]

```
def listtype(l):
    return type(l) == list

def flatten(l):
    flattened = []
    for item in l:
        if listtype(item):
            flattened.extend(flatten(item))
        else:
            flattened.append(item)
    return flattened
flatten([1,2,[3],[4,[5,6]]])
```

```
[1, 2, 3, 4, 5, 6]
```

1. 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. Here is a sample input and its corresponding output. Sample Input Books APM-001~Advanced Potion-Making GWG-001~Gadding With Ghouls APM-002~Advanced Potion-Making DMT-001~Defensive Magical Theory DMT-003~Defensive Magical Theory GWG-002~Gadding With Ghouls DMT-002~Defensive Magical Theory Borrowers SLY2301~Hannah Abbott SLY2302~Euan Abercrombie SLY2303~Stewart Ackerley SLY2304~Bertram Aubrey SLY2305~Avery SLY2306~Malcolm Baddock SLY2307~Marcus Belby SLY2308~Katie Bell SLY2309~Sirius Orion Black Checkouts SLY2304~DMT-002~2019-03-27 SLY2301~GWG-001~2019-03-27 SLY2308~APM-002~2019-03-14 SLY2303~DMT-001~2019-04-03 SLY2301~GWG-002~2019-04-03 EndOfInput Sample Output 2019-03-14~Katie Bell~APM-002~Advanced Potion-Making 2019-03-27~Bertram Aubrey~DMT-002~Defensive Magical Theory 2019-03-27~Hannah Abbott~GWG-001~Gadding With Ghouls 2019-04-03~Hannah Abbott~GWG-002~Gadding With Ghouls 2019-04-03~Stewart Ackerley~DMT-001~Defensive Magical Theory

```python
def process_checkout_data(books, borrowers, checkouts):
    # Create a dictionary to store borrower names
    borrower_names = {}
    for borrower in borrowers:
        username, full_name = borrower.split('~')
        borrower_names[username] = full_name

    # Create a list to store checked out books
    checked_out_books = []
    for checkout in checkouts:
        username, accession_number, due_date = checkout.split('~')
        title = books[accession_number]
        full_name = borrower_names[username]
        checked_out_books.append((due_date, full_name,
accession_number, title))

    # Sort the checked out books by due date and full name
    checked_out_books.sort(key=lambda x: (x[0], x[1]))

    # Print the details of checked out books
    for book in checked_out_books:
        print('~'.join(book))


def process_input_data(input_data):
    books = {}
    borrowers = []
    checkouts = []

    current_section = None
    for line in input_data:
        if line == 'Books':
            current_section = 'Books'
        elif line == 'Borrowers':
            current_section = 'Borrowers'
        elif line == 'Checkouts':
            current_section = 'Checkouts'
        elif line == 'EndOfInput':
            break
        else:
            if current_section == 'Books':
                accession_number, title = line.split('~')
                books[accession_number] = title
            elif current_section == 'Borrowers':
                borrowers.append(line)
            elif current_section == 'Checkouts':
                checkouts.append(line)

    process_checkout_data(books, borrowers, checkouts)


# Sample input data as a multiline string
input_data = """Books
APM-001~Advanced Potion-Making
GWG-001~Gadding With Ghouls
APM-002~Advanced Potion-Making
DMT-001~Defensive Magical Theory
DMT-003~Defensive Magical Theory
GWG-002~Gadding With Ghouls
DMT-002~Defensive Magical Theory
Borrowers
```

```
SLY2301~Hannah Abbott
SLY2302~Euan Abercrombie
SLY2303~Stewart Ackerley
SLY2304~Bertram Aubrey
SLY2305~Avery
SLY2306~Malcolm Baddock
SLY2307~Marcus Belby
SLY2308~Katie Bell
SLY2309~Sirius Orion Black
Checkouts
SLY2304~DMT-002~2019-03-27
SLY2301~GWG-001~2019-03-27
SLY2308~APM-002~2019-03-14
SLY2303~DMT-001~2019-04-03
SLY2301~GWG-002~2019-04-03
EndOfInput"""

# Split the input data into lines and process it
input_lines = input_data.split('\n')
process_input_data(input_lines)
```

```
2019-03-14~Katie Bell~APM-002~Advanced Potion-Making
2019-03-27~Bertram Aubrey~DMT-002~Defensive Magical Theory
2019-03-27~Hannah Abbott~GWG-001~Gadding With Ghouls
2019-04-03~Hannah Abbott~GWG-002~Gadding With Ghouls
2019-04-03~Stewart Ackerley~DMT-001~Defensive Magical Theory
```

1. Write a function intreverse(n) that takes as input a positive integer n and returns the integer obtained by reversing the digits in n. Here are some examples of how your function should work.

   intreverse(783) 387 intreverse(242789) 987242

```
def intreverse(n):
    reversed_num = 0
    while n > 0:
        # Extract the last digit from n
        last_digit = n % 10
        # Add the last digit to the reversed number
        reversed_num = (reversed_num * 10) + last_digit
        # Remove the last digit from n
        n //= 10
    return reversed_num
intreverse(783)
```

```
387
```

1. 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. Here are some examples to show how your function should work.

matched("zb%78") True matched("(7)(a") False

```
def matched(s):
    stack = []
    for char in s:
        if char == '(':
            stack.append(char)
        elif char == ')':
            if len(stack) == 0:
                return False
            stack.pop()

    return len(stack) == 0
matched("zb%78")
```

```
True
```

1. 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. Here are some examples to show how your function should work.

   sumprimes([3,3,1,13]) 19 sumprimes([2,4,6,9,11]) 13 sumprimes([-3,1,6]) 0

```
def is_prime(n):
    if n < 2:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True

def sumprimes(l):
    prime_sum = 0
    for num in l:
        if is_prime(num):
            prime_sum += num
    return prime_sum
sumprimes([3,3,1,13])
```

```
19
```

1. 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. Here are some examples of how your function should work.

   contracting([9,2,7,3,1]) True

contracting([-2,3,7,2,-1]) False

```
def contracting(l):
    for i in range(1, len(l)):
        diff1 = abs(l[i] - l[i-1])
        diff2 = abs(l[i+1] - l[i])
        if i > 1 and diff1 <= diff2:
            return False
    return True
contracting([-2,3,7,2,-1])
```

```
False
```

1. 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. Here are some examples to show how your function should work.

   counthv([1,2,1,2,3,2,1]) [2, 1]

   counthv([1,2,3,1]) [1, 0]

```
def counthv(l):
    hill_count = 0
    valley_count = 0

    for i in range(1, len(l) - 1):
        if l[i] > l[i-1] and l[i] > l[i+1]:
            hill_count += 1
        elif l[i] < l[i-1] and l[i] < l[i+1]:
            valley_count += 1

    return [hill_count, valley_count]
counthv([1,2,1,2,3,2,1])
```

```
[2, 1]
```

1. 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
2. 1 2 3 4 5 6 7 8 9
3. would be represented as [[1,2,3], [4,5,6], [7,8,9]].
4. 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
5. 3 6 9 2 5 8 1 4 7
6. Your function should not modify the argument m provided to the function rotate().
7. Here are some examples of how your function should work.
8. leftrotate([[1,2],[3,4]]) [[2, 4], [1, 3]]

   leftrotate([[1,2,3],[4,5,6],[7,8,9]]) [[3, 6, 9], [2, 5, 8], [1, 4, 7]] leftrotate([[1,1,1],[2,2,2],[3,3,3]]) [[1, 2, 3], [1, 2, 3], [1, 2, 3]]

```python
def leftrotate(m):
    n = len(m)  # Get the size of the square matrix

    # Step 1: Transpose the matrix
    for i in range(n):
        for j in range(i + 1, n):
            m[i][j], m[j][i] = m[j][i], m[i][j]

    # Step 2: Reverse each row
    for i in range(n):
        m[i] = m[i][::-1]

    return m
leftrotate([[1,2,3],[4,5,6],[7,8,9]])
```

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