In [1]:

```
'Week 3 Practice Programming Assignment'

def descending(L):
    'List must be monotonically descending'
    'L[i-1] >= L[i]'
    for i in range(1,len(L)):
        if L[i-1]<L[i]:
            return False
    else: # else of loop executes, if terminates from top, not inbetween 'break'
        return True

def valley(L):
    'Strictly Decreasing then increasing, both non-empty'
    lowest_index = L.index(min(L))
    if L[:lowest_index+1] and L[lowest_index:]:
        # Both DSC & ASC lists are non-empty
        cond_1 = sorted(L[:lowest_index+1],reverse=True)  == L[:lowest_index+1]
        cond_2 = sorted(L[lowest_index:],  reverse=False) == L[lowest_index:]
        if cond_1 and cond_2:
            # 1st slice is reverse(DSC) sorted, 2nd is default (ASC) sorted
            return True
        else:
            return False #Both non-empty, but any one violates condition
    else:
        # False if any list is empty, or any condition won't hold
        return False

def transpose(L):
    'Matrix transpose of matrices'
    # Changing just row_indx & col_indx won't work for rectangular matrix
    return [ [ L[ri][ci] for ri in range(len(L))] for ci in range(len(L[0])) ]
```

In [2]:

```
_ = '''DO NOT COPY THIS CODE'''

a = '''Test Case 1 descending([]) True
Test Case 2 descending([4,4,3]) True
Test Case 3 descending([19,17,18,7]) False
Test Case 4 valley([2]) False
Test Case 5 valley([13,12,14,14]) False
Test Case 6 valley([5,4,3,2,1,2]) True
Test Case 7 valley([17,1,2,3,4,5]) True
Test Case 8 transpose([[19,14]]) [[19],[14]]
Test Case 9 transpose([[21,33,11],[42,64,16]]) [[21,42],[33,64],[11,16]]
Test Case 10 transpose([[16,31,42],[26,82,73],[84,53,38]]) [[16,26,84],[31,82,53],[42,7
'''


for i1,i2,i3,j,k in [x.split(' ') for x in a.split('\n')]:
    print(eval(j),eval(k))

_ = '''NON-COPYING REGION END'''
```

```
True True
True True
False False
True False
True False
True True
True True
[[19], [14]] [[19], [14]]
[[21, 42], [33, 64], [11, 16]] [[21, 42], [33, 64], [11, 16]]
[[16, 26, 84], [31, 82, 53], [42, 73, 38]] [[16, 26, 84], [31, 82, 53],
 [42, 73, 38]]
```

In [ ]:

In [3]:

```
'Week 3 Programming Assignment'

def progression(L):
    'check if given list is an Arithmetic Progression'
    if len(L)<=1:
        return True
    a = L[0]
    d = L[1]-L[0]
    for i in range(1,len(L)):
        d_chk = L[i]-L[i-1]
        if d != d_chk:
            return False
    else:
        return True

def primesquare(L):
    # Snippet of Sieve of Eratosthenes (prev. assignment, populat algo)
    mx = max(L)+1
    prime_index = [1]*mx
    prime_index[0:2] = [0,0]
    for i in range(2,round(mx**0.5)+1):
        if prime_index[i]:
            for j in range(i*2,mx,i):
                prime_index[j] = 0
    # Finding how list begins, prime, prsqr or none
    first = L[0]
    if prime_index[first]:
        'List begins with Primes'
        prime_indx_parity,prsqr_indx_parity  = 0,1
    elif (int(first**0.5)**2) == first:
        'List begins with Perfect square'
        prime_indx_parity,prsqr_indx_parity  = 1,0
    else:
        'Not any above, must be false'
        return False
    # Lambda functions to check primes & perfect squares
    prime_chk = lambda x: prime_index[x]
    prsqr_chk = lambda x: (int(x**0.5)**2) == x
    # Generating expression of primes & prsqrs positions
    primes = ( L[i] for i in range(len(L)) if i%2==prime_indx_parity )
    prsqrs = ( L[i] for i in range(len(L)) if i%2==prsqr_indx_parity )
    # Condition for checking if primes & prsqr are at correct places
    prime_valid_pos = all(map( prime_chk , primes ))
    prsqr_valid_pos = all(map( prsqr_chk , prsqrs ))

    if prime_valid_pos and prsqr_valid_pos:
        return True
    else:
        return False

def matrixflip(m,d):
    'Note logical horizontal & vertical flip differ from testcases given'
    # Actually return arguments of IF & ELIF should be swapped, but not for assignment
    if d=='h': # Horizontal flip be reversing rows
        return [row[::-1] for row in m]
    elif d=='v': # Vertical flip by reversing cols (entries in each row)
        return m[::-1]
    else: # This case should never occur
        return m
```

In [4]:

```
_ = '''DO NOT COPY THIS CODE'''

a = '''Test Case 1 progression([3]) True
Test Case 2 progression([7,3,-1,-5]) True
Test Case 3 progression([3,5,7,9,10]) False
Test Case 4 primesquare([4]) True
Test Case 5 primesquare([4,5,16,101,64]) True
Test Case 6 primesquare([5,16,101,36,27]) False
Test Case 7 matrixflip([[1,2],[3,4]],'h') [[2,1],[4,3]]
Test Case 8 matrixflip([[1,2],[3,4]],'v') [[3,4],[1,2]]'''


for i1,i2,i3,j,k in [x.split(' ') for x in a.split('\n')]:
    print(eval(j),eval(k))

_ = '''NON-COPYING REGION END'''
```

```
True True
True True
False False
True True
True True
False False
[[2, 1], [4, 3]] [[2, 1], [4, 3]]
[[3, 4], [1, 2]] [[3, 4], [1, 2]]
```

In [ ]: