In [1]:
```
import ast
import numpy as np
```

In [2]:
```
class UserMainCode():


    '''
    function that accepts list and return sorted list having values
    greater than average value of the list.
    '''

    @classmethod
    def list_avg(cls, input1):

        out_list = []
        if len(input1) > 0:
            average = sum(input1)/len(input1)
            out_list = [x for x in input1 if x > average]


        ''' #Alternative method using filter.
        average = sum(input1)/len(input1)
        out_list = li
        st(filter(lambda x:True if x > average else False, input1))
        '''

        return sorted(out_list)


    '''
    function that accepts list and return filtered list having values
    that is divisible by 4 or 9. Also same order as in orginal list.
    '''

    @classmethod
    def div_list(cls, input1):

        out_list = []
        if len(input1) > 0:
            out_list = [x for x in input1 if (x % 4 == 0) or ( x % 9 == 0)]

        return out_list


    '''
    function that makes diagonal of nxn matrix by 1 for input n
    '''
    @classmethod
    def diag_one(cls, input1):

        diag_arr = np.zeros((input1,input1),dtype=int)


        for i in range(input1):
            diag_arr[i][i] = 1
            diag_arr[i][(input1-1)-i] = 1
```

```
        '''
        #Alternative solution
        np.fill_diagonal(diag_arr, 1)
        np.fill_diagonal(np.fliplr(diag_arr), 1) #can use flipud as well
        '''


        return str(list(diag_arr))


    '''
    function that check whether input2 belongs to arithmetic progression
    in input1
    '''
    @classmethod
    def is_ap(cls, input1 , input2):

        out_str = 'Yes'


        if len(input1) > 1:
            while True:
                if input2 > input1[0] :

                    #this section for checking number on right side
                    if input2 == input1[-1]:
                        out_str = 'Yes'
                        break;
                    elif input2 < input1[-1]:
                        out_str = 'No'
                        break;
                    else:
                        input1.append(input1[-1] + (input1[1] - input1[0]))

                else:

                    #this section for checking number on left side
                    if input2 == input1[0]:
                        out_str = 'Yes'
                        break;
                    elif input2 > input1[-1]:
                        out_str = 'No'
                        break;
                    else:
                        input1.insert(0,input1[0] - (input1[1] - input1[0]))

            '''
            #Optimized and better solution
            diff = input1[1] - input1[0]
            if input2 > input1[0]:

                if input2 - input1[0] % diff ==0 :
                    out_str = 'Yes'
                else:
                    out_str = 'No'

            else :

                if input2 + input1[0] % diff ==0 :
                    out_str = 'Yes'
                else:
```

```
                out_str = 'No'
        '''

    return out_str


'''
function that checks whether from current location can we reach to
treasure.
'''
@classmethod
def treasure_hunt(cls, input1):

    out_str = 'No'
    if len(input1) > 1:

        t_map = np.array(input1,dtype=int)
        t_map_shape = t_map.shape
        visited_loc = np.zeros(t_map_shape,dtype=int)
        next_loc_queue = list()

        t_pos = tuple()

        for i in range(t_map_shape[0]):
            for j in range(t_map_shape[1]):
                if t_map[i][j] == 1:
                    next_loc_queue.append((i,j))
                elif t_map[i][j] == 2:
                    t_pos = (i,j)
                elif t_map[i][j] == -1:
                    visited_loc[i][j] = 1


        while len(next_loc_queue) > 0:
            #print(visited_loc)

            curr_loc = next_loc_queue[0]

            if t_pos in next_loc_queue:
                out_str='Yes'
                break;
            else :
                row = curr_loc[0]
                col = curr_loc[1]
                visited_loc[row][col] = 1
                next_loc_queue.remove((row,col))

                if row != 0:
                    if visited_loc[row-1][col] != 1:
                        next_loc_queue.append((row-1,col))

                if row != t_map_shape[0]-1:
                    if visited_loc[row+1][col] != 1:
                        next_loc_queue.append((row+1,col))

                if col != t_map_shape[1]-1:
                    if visited_loc[row][col+1] != 1:
                        next_loc_queue.append((row,col+1))

                if col != 0:
                    if visited_loc[row][col-1] != 1:
```

```
                            next_loc_queue.append((row,col-1))

        else:
            out_str = 'No'


        return out_str
```

In [3]:

```
#Testcases to just take input and call neccessary function.
class TestCode():

    @classmethod
    def avg_test(cls):
        inp_str = input('Enter input in list format e.g [1,2] - ')
        inp_list = ast.literal_eval(inp_str);

        print(UserMainCode.list_avg(inp_list))

    @classmethod
    def div_test(cls):
        inp_str = input('Enter input in list format e.g [1,2] - ')
        inp_list = ast.literal_eval(inp_str);

        print(UserMainCode.div_list(inp_list))

    @classmethod
    def diag_test(cls):
        inp_str = input('Enter a no  - ')
        print(UserMainCode.diag_one(int(inp_str)))

    @classmethod
    def is_ap_test(cls):
        inp_str1 = input('Enter input in list format e.g [1,2] - ')
        inp_list = ast.literal_eval(inp_str1);

        inp_str2 = input('Enter a long value e.g 1l - ')
        no_to_check = int(inp_str2.strip('l'))

        print(UserMainCode.is_ap(inp_list,no_to_check))

    @classmethod
    def treasure_hunt_test(cls):
        inp_str1 = input('Enter input in 2d map i.e nxn nest list format e.g [[-1,2],[1
        inp_list = ast.literal_eval(inp_str1);


        print(UserMainCode.treasure_hunt(inp_list))
```

In [5]:

```
#To Test Program1
TestCode.avg_test()

# Sample Test cases
# i/p - [39,60,55,78,88,43,84,25]
# o/p - [60, 78, 84, 88]
```

```
Enter input in list format e.g [1,2] - [39,60,55,78,88,43,84,25]
[60, 78, 84, 88]
```

In [6]:
```
#To Test Program2
TestCode.div_test()

# Sample Test cases
# i/p - [3,8,12,15,18,9,4,27,36]
# o/p - [8, 12, 18, 9, 4, 27, 36]

# i/p - [3,6,10,17]
# o/p - []
```

```
Enter input in list format e.g [1,2] - [3,8,12,15,18,9,4,27,36]
[8, 12, 18, 9, 4, 27, 36]
```

In [7]:
```
#To Test Program3
TestCode.diag_test()

# Sample Test cases
# i/p - 3
# o/p - [[1, 0, 1],[0, 1, 0]),[1, 0, 1]]

# i/p - 4
# o/p - [[1, 0, 0, 1],[0, 1, 1, 0],[0, 1, 1, 0],[1, 0, 0, 1]]
```

```
Enter a no  - 4
[array([1, 0, 0, 1]), array([0, 1, 1, 0]), array([0, 1, 1, 0]), array([1, 0, 0, 1])]
```

In [8]:
```
#To Test Program4
TestCode.is_ap_test()

# Sample Test cases
# i/p -
#     [1,8,15]
#     36
# o/p - Yes

# i/p -
#     [1,3,5]
#     -1
# o/p - Yes

# i/p -
#     [1,8,15]
#     21
# o/p - No
```

```
Enter input in list format e.g [1,2] - [1,8,15]
Enter a long value e.g 1l - 36
Yes
```

In [9]:
```
#To Test Program5
TestCode.treasure_hunt_test()

# Sample Test cases
# i/p - [[-1,2],[1,0]]
# o/p - Yes

# i/p - [[-1,2],[1,-1]]
# o/p - No
```

```
# i/p - [[-1,0,0],[1,0,-1],[0,0,2]]
# o/p - Yes
```

```
Enter input in 2d map i.e nxn nest list format e.g [[-1,2],[1,0]] - [[-1,0,0],[1,0,-1],
[0,0,2]]
Yes
```

In [ ]:

```
### END ####
```