```python
# Reverse Array

def reverse_array(letters, first=0, last=None):
    "reverses the letters in an array in-place"
    if last is None:
        last = len(letters)
    last -= 1
    while first < last:
        letters[first], letters[last] = letters[last], letters[first]
        first += 1
        last -= 1

# Reverse string: reverse char but not order of words

def reverse_words(string):
    s='The dog ran'
    ' '.join(w[::-1] for w in s.split())

# Reverse String

'hello world'[::-1]

# First Question:
# If you have only one room, what is the maximum number of meetings you can
# scheduled into that room.
#
# Solution:
# 1. sort the meetings by finishing time, this is because we greedily choose the
# meeting that finishes first.
# 2. go through all the meetings in order of finishing time, schedule the meeting into
# the room if the room is not occupied at its start time, and increase the count by one.
# 3. no of count will be the max number of meetings you can schedule into the room.
#
# Second Question:
# You are given a set of meetings with start time and end time, what is the minimum
# number of meeting rooms you need to have to hold all the meetings.
#
# A better solution using the greedy approach
# 1. We sort the meetings by start time
# 2. Then step through all the meetings in order of start time, keep a set of meeting
# rooms, if all the rooms are occupied, then we schedule a new room. To check all the
# previous scheduled meetings, we keep a priority queue by finishing time of all the
# scheduled meetings. Assume there are d number of rooms, then checking takes logd time.
# 3. count the number of rooms.

# If we rob house[i], we couldn't rob house[i-1], but we could rob house[i-2].
# If we don't rob house[i], we could rob house[i-1].
# Choose the one gets more money.

rob_house[i]= max(house[i] + rob_house[i-2], rob_house[i-1])
```

```python
# Reverse singly linked list

# https://leetcode.com/problems/reverse-linked-list/#/description

def reverse_list(head):
    new_head = None
    while head:
        temp = head
        head = temp.next
        temp.next = new_head
        new_head = temp
    return new_head



# Two Sum

# Given an array of integers, return indices of the two numbers such that they
# add up to a specific target. You may assume that each input would have exactly
# one solution, and you may not use the same element twice.

# Given nums = [2, 7, 11, 15], target = 9,

# Because nums[0] + nums[1] = 2 + 7 = 9,
# return [0, 1].

def twoSum(self, nums, target):
    """
    :type nums: List[int]
    :type target: int
    :rtype: List[int]
    """
    matches = {}
    for i in xrange(1, len(nums)):
        print(nums[i])
        print(matches)
        if nums[i] in matches:
            return [matches[nums[i]], i]
        else:
            matches[target - nums[i]] = i
    return "no matches :("

# Add Two Numbers II

# Input: (7 -> 2 -> 4 -> 3) + (5 -> 6 -> 4)
# Output: 7 -> 8 -> 0 -> 7
# You are given two non-empty linked lists representing two non-negative integers.
# The most significant digit comes first and each of their nodes contain a single digit.
# Add the two numbers and return it as a linked list.
# https://github.com/kamyu104/LeetCode/blob/master/Python/add-two-numbers.py
```

```
def addTwoNumbers(self, l1, l2):
        """
        :type l1: ListNode
        :type l2: ListNode
        :rtype: ListNode
        """
        dummy = ListNode(0)
        current = dummy
        carry = 0

        while l1 or l2:
            val = carry
            if l1:
                val = val + l1.val
                l1 = l1.next
            if l2:
                val = val + l2.val
                l2 = l2.next
            carry = val / 10
            val = val % 10
            current.next = ListNode(val)
            current = current.next

        if carry == 1:
            current.next = ListNode(1)

        return dummy.next


# Directory Walk

# print out directory contents

def print_directory_contents(sPath):
    import os
    for sChild in os.listdir(sPath):
        sChildPath = os.path.join(sPath,sChild)
        if os.path.isdir(sChildPath):
            print_directory_contents(sChildPath)
        else:
            print(sChildPath)


# Move Zeroes

# https://leetcode.com/problems/move-zeroes/#/description

# Given an array nums, write a function to move all 0's to the end of it while
# maintaining the relative order of the non-zero elements.
# # For example, given nums = [0, 1, 0, 3, 12], after calling your function,
# nums should be [1, 3, 12, 0, 0].

# You must do this in-place without making a copy of the array.
# Minimize the total number of operations.

def move_zeros(lst):
  n = len(lst)
  lst[:] = filter(None, lst)
  lst.extend([0] * (n - len(lst)))
```

```

# Add two array of digit numbers

# https://leetcode.com/problems/add-two-numbers-ii/#/description

# Definition for singly-linked list.
# class ListNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution(object):
    def addTwoNumbers(self, l1, l2):
        """
        :type l1: ListNode
        :type l2: ListNode
        :rtype: ListNode
        """
        list1, list2 = []
        count1, count2 = 0

        while (l1 != None) {
            list1[count] = l1.val
            l1 = l1.next
        }
        while (l2 != None) {
            list2[count] = l2.val
            l2 = l2.next
        }

        i,j=count1, count2
        carry = 0
        sum = 0

        while (i>0 or j>0 or carry>0) :
            sum = carry;
            if i>=0: sum += list1[i]
            if j>=0: sum += list2[j]

            carry = sum / 10
            print(sum%10 + " ")

            i=i-1
            j=j-1

        return 0

# Missing number

# Given an array containing n distinct numbers taken from 0, 1, 2, ..., n,
# find the one that is missing from the array.

def missingNumber(self, nums):
    n = len(nums)
    return n * (n+1) / 2 - sum(nums)
```

```python
# Merge 2 Sorted Array

def merge(self, nums1, m, nums2, n):
        while m > 0 and n > 0:
            if nums1[m-1] >= nums2[n-1]:
                nums1[m+n-1] = nums1[m-1]
                m -= 1
            else:
                nums1[m+n-1] = nums2[n-1]
                n -= 1
        if n > 0:
            nums1[:n] = nums2[:n]
```