code step by step problem 1:

```
numbers = [1, 2, 3, 4, 5, 1, 2, 3, 6, 7, 8, 9, 10, 1]

def count_duplicates(numbers):
    count = 0
    seen = set() # create a set for checking numbers
    for number in numbers: # iterate through numbers list
        if number in seen: # checking if that number is also in seen
list indicating duplicate in the number list
            count += 1 # if a duplicate is spotted then increase
duplicate count + 1
        else: # if no duplicates yet check next number
            seen.add(number)
    return count

print(count_duplicates(numbers))
```

```
4
```

code step by step problem 2:

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

def is_stack_sorted(numbers):
    restoration_list = list()
    if len(numbers) <= 1: # check to see if numbers list empty or only
one number
        return True
    sorted_boolean = True
    compare_number = numbers.pop() # first number
    while len(numbers) > 0:
        next_number = numbers.pop() # next number to compare against
        if compare_number > next_number: # comparison to test sorting
            sorted_boolean = False # test result false, stack not
sorted
        restoration_list.append(compare_number) # banking numbers for
restoration
        compare_number = next_number # next number to check
    restoration_list.append(compare_number)
    while len(restoration_list) > 0:
        numbers.append(restoration_list.pop()) # restoring the numbers
list from restoration list bank
    return sorted_boolean

print(is_stack_sorted(numbers))
```

```
False
```

code step by step problem 3:

```
numbers = [5, 2, 4, 17, 55, 4, 3, 26, 18, 2, 17]

def median(numbers): # partially from class code repository insertion
sort
    for i in range(1, len(numbers)):
        j = i

        # Insert numbers[i] into sorted part
        # stopping once numbers[i] in correct position
        while j > 0 and numbers[j] < numbers[j - 1]:
            # Swap numbers[j] and numbers[j - 1]
            temp = numbers[j]
            numbers[j] = numbers[j - 1]
            numbers[j - 1] = temp
            j -= 1
    middle = len(numbers) // 2 # define middle of sorted list
    if len(numbers) % 2 == 0: # check for list being even
        median_value = (numbers[middle - 1] + numbers[middle] / 2) #
if even median is average of two middle numbers
    else:
        median_value = numbers[middle] # if odd median is middle
    return median_value

median(numbers)
```

```
5
```

data lemur problem:

```
a = [1, 2, 3, 4, 5]
b = [0, 1, 3, 7]

def intersection(a, b):
  intersection_list = [] # list to be returned
  for value in a: # iterate through a
    if value in b: # compare if the number in a is a number in b
      intersection_list.append(value) # if true then add to return
list
  return intersection_list

intersection(a, b)
```

```
[1, 3]
```

leet code problem:

```
class MyQueue(object):
    def __init__(self): #initialize our two stacks
        self.s1 = []
        self.s2 = []
    def push(self, x):
        while self.s1:
            self.s2.append(self.s1.pop()) # populate our stack 2 with
stack 1 numbers
        self.s1.append(x) # stack 1 holds only x
        while self.s2:
            self.s1.append(self.s2.pop()) # add the numbers back to
stack 1 behind x

    def pop(self):
        return self.s1.pop()

    def peek(self):
        return self.s1[-1] # the last number in stack 1 is the top

    def empty(self):
        return not self.s1

NewQueue = MyQueue()
NewQueue.push(1)
NewQueue.push(2)
print(NewQueue.peek())
print(NewQueue.pop())
print(NewQueue.empty())
```

```
1
1
False
```