# Debugging Mastery Problems - Set 2

## 1

### Problem description:

Write a function that returns the smaller of two numbers.

### Buggy code:

def smaller\_number(a, b):  
 if a < b:  
 return a  
 else  
 return b

### One kind of right solution:

def smaller\_number(a, b):  
 if a < b:  
 return a  
 else:  
 return b

## 2

### Problem description:

Write a function that checks if a number is odd.

### Buggy code:

def is\_odd(n):  
 return n % 2 =! 0

### One kind of right solution:

def is\_odd(n):  
 return n % 2 != 0

## 3

### Problem description:

Write a function that returns the maximum element in a list.

### Buggy code:

def max\_element(lst):  
 return maximum(lst)

### One kind of right solution:

def max\_element(lst):  
 return max(lst)

## 4

### Problem description:

Write a function that checks if a list is sorted.

### Buggy code:

def is\_sorted(lst):  
 return lst = sorted(lst)

### One kind of right solution:

def is\_sorted(lst):  
 return lst == sorted(lst)

## 5

### Problem description:

Write a function that rotates a list to the left by k positions.

### Buggy code:

def rotate\_left(lst, k):  
 return lst[:k] + lst[k:]

### One kind of right solution:

def rotate\_left(lst, k):  
 return lst[k:] + lst[:k]

## 6

### Problem description:

Write a function that finds the most frequent element in a list.

### Buggy code:

def most\_frequent(lst):  
 return max(lst, key = lst.count())

### One kind of right solution:

def most\_frequent(lst):  
 return max(set(lst), key = lst.count)

## 7

### Problem description:

Implement a basic Queue class with enqueue and dequeue methods.

### Buggy code:

class Queue:  
 def \_\_init\_\_(self):  
 self.items = []  
  
 def enqueue(self, item):  
 self.items.append(item)  
  
 def dequeue(self):  
 return self.items.pop()

### One kind of right solution:

class Queue:  
 def \_\_init\_\_(self):  
 self.items = []  
  
 def enqueue(self, item):  
 self.items.append(item)  
  
 def dequeue(self):  
 return self.items.pop(0)

## 8

### Problem description:

Write a function that checks if a given string is a palindrome.

### Buggy code:

def is\_palindrome(s):  
 s = s.replace(" ", "")  
 return s == s[::-1]

### One kind of right solution:

def is\_palindrome(s):  
 s = s.replace(" ", "").lower()  
 return s == s[::-1]

## 9

### Problem description:

Implement a Trie data structure with methods to insert and search words.

### Buggy code:

class TrieNode:  
 def \_\_init\_\_(self):  
 self.children = {}  
 self.is\_end\_of\_word = False  
  
class Trie:  
 def \_\_init\_\_(self):  
 self.root = TrieNode()  
  
 def insert(self, word):  
 node = self.root  
 for char in word:  
 if char not in node.children:  
 node.children[char] = TrieNode()  
 node = node.children[char]  
  
 def search(self, word):  
 node = self.root  
 for char in word:  
 if char not in node.children:  
 return False  
 node = node.children[char]  
 return node.is\_end\_of\_word

### One kind of right solution:

class TrieNode:  
 def \_\_init\_\_(self):  
 self.children = {}  
 self.is\_end\_of\_word = False  
  
class Trie:  
 def \_\_init\_\_(self):  
 self.root = TrieNode()  
  
 def insert(self, word):  
 node = self.root  
 for char in word:  
 if char not in node.children:  
 node.children[char] = TrieNode()  
 node = node.children[char]  
 node.is\_end\_of\_word = True  
  
 def search(self, word):  
 node = self.root  
 for char in word:  
 if char not in node.children:  
 return False  
 node = node.children[char]  
 return node.is\_end\_of\_word

## 10

### Problem description:

Write a function that finds the longest common prefix of an array of strings.

### Buggy code:

def longest\_common\_prefix(strs):  
 if not strs:  
 return ""  
 prefix = strs[0]  
 for s in strs[1:]:  
 while s.find(prefix) = 0:  
 prefix = prefix[:-1]  
 if not prefix:  
 return ""  
 return prefix

### One kind of right solution:

def longest\_common\_prefix(strs):  
 if not strs:  
 return ""  
 prefix = strs[0]  
 for s in strs[1:]:  
 while s.find(prefix) != 0:  
 prefix = prefix[:-1]  
 if not prefix:  
 return ""  
 return prefix