# Debugging Mastery Problems - Set 3

## 1

### Problem description:

Write a function that calculates the area of a rectangle given its length and width.

### Buggy code:

def rectangle\_area(length, width):  
 return length + width

### One kind of right solution:

def rectangle\_area(length, width):  
 return length \* width

## 2

### Problem description:

Write a function that calculates the perimeter of a rectangle given its length and width.

### Buggy code:

def rectangle\_perimeter(length, width):  
 return 2 \* length + width

### One kind of right solution:

def rectangle\_perimeter(length, width):  
 return 2 \* (length + width)

## 3

### Problem description:

Write a function that checks if a string contains any uppercase letters.

### Buggy code:

def has\_uppercase(s):  
 for char in s:  
 if char.isupper:  
 return True  
 return False

### One kind of right solution:

def has\_uppercase(s):  
 for char in s:  
 if char.isupper():  
 return True  
 return False

## 4

### Problem description:

Write a function that counts the number of vowels in a string.

### Buggy code:

def count\_vowels(s):  
 vowels = "AEIOUaeiou"  
 count = 0  
 for char in s:  
 if char == vowels:  
 count += 1  
 return count

### One kind of right solution:

def count\_vowels(s):  
 vowels = "AEIOUaeiou"  
 count = 0  
 for char in s:  
 if char in vowels:  
 count += 1  
 return count

## 5

### Problem description:

Write a function that returns the first recurring character in a string.

### Buggy code:

def first\_recurring\_char(s):  
 seen = set()  
 for char in s:  
 if char not in seen:  
 return char  
 seen.add(char)  
 return None

### One kind of right solution:

def first\_recurring\_char(s):  
 seen = set()  
 for char in s:  
 if char in seen:  
 return char  
 seen.add(char)  
 return None

## 6

### Problem description:

Write a function that flattens a nested list.

### Buggy code:

def flatten(lst):  
 result = []  
 for item in lst:  
 if isinstance(item, list):  
 result.append(flatten(item))  
 else:  
 result.append(item)  
 return result

### One kind of right solution:

def flatten(lst):  
 result = []  
 for item in lst:  
 if isinstance(item, list):  
 result.extend(flatten(item))  
 else:  
 result.append(item)  
 return result

## 7

### Problem description:

Implement a basic Binary Search function.

### Buggy code:

def binary\_search(arr, target):  
 low, high = 0, len(arr)  
 while low <= high:  
 mid = (low + high) // 2  
 if arr[mid] == target:  
 return mid  
 elif arr[mid] < target:  
 low = mid + 1  
 else:  
 high = mid - 1  
 return None

### One kind of right solution:

def binary\_search(arr, target):  
 low, high = 0, len(arr) - 1  
 while low <= high:  
 mid = (low + high) // 2  
 if arr[mid] == target:  
 return mid  
 elif arr[mid] < target:  
 low = mid + 1  
 else:  
 high = mid - 1  
 return None

## 8

### Problem description:

Write a function that returns the power of a number using recursion.

### Buggy code:

def power(x, n):  
 if n == 0:  
 return 0  
 if n < 0:  
 return 1 / power(x, -n)  
 half = power(x, n // 2)  
 if n % 2 == 0:  
 return half \* half  
 else:  
 return half \* half \* x

### One kind of right solution:

def power(x, n):  
 if n == 0:  
 return 1  
 if n < 0:  
 return 1 / power(x, -n)  
 half = power(x, n // 2)  
 if n % 2 == 0:  
 return half \* half  
 else:  
 return half \* half \* x

## 9

### Problem description:

Implement a Merge Sort algorithm.

### 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:

def merge\_sort(arr):  
 if len(arr) <= 1:  
 return arr  
 mid = len(arr) // 2  
 left\_half = merge\_sort(arr[:mid])  
 right\_half = merge\_sort(arr[mid:])  
 return merge(left\_half, right\_half)  
  
def merge(left, right):  
 result = []  
 i = j = 0  
 while i < len(left) and j < len(right):  
 if left[i] < right[j]:  
 result.append(left[i])  
 i += 1  
 else:  
 result.append(right[j])  
 j += 1  
 result.extend(left[i:])  
 result.extend(right[j:])  
 return result

## 10

### Problem description:

Write a function that finds the longest substring without repeating characters.

### 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\_substring(s):  
 char\_index = {}  
 start, max\_len = 0, 0  
 for i, char in enumerate(s):  
 if char in char\_index and start <= char\_index[char]:  
 start = char\_index[char] + 1  
 else:  
 max\_len = max(max\_len, i - start + 1)  
 char\_index[char] = i  
 return max\_len