# Debugging Mastery Problems - Set 4

## 1

### Problem description:

Write a function that calculates the area of a triangle given its base and height.

### Buggy code:

def triangle\_area(base, height):  
 return 0.5 + base \* height

### One kind of right solution:

def triangle\_area(base, height):  
 return 0.5 \* base \* height

## 2

### Problem description:

Write a function that returns the square of a number.

### Buggy code:

def square\_number(n):  
 return n \* n \* n

### One kind of right solution:

def square\_number(n):  
 return n \*\* 2

## 3

### Problem description:

Write a function that reverses a list.

### Buggy code:

def reverse\_list(lst):  
 return list[::-1]

### One kind of right solution:

def reverse\_list(lst):  
 return lst[::-1]

## 4

### Problem description:

Write a function that checks if a list is a palindrome.

### Buggy code:

def is\_palindrome\_list(lst):  
 return lst == reversed(lst)

### One kind of right solution:

def is\_palindrome\_list(lst):  
 return lst == lst[::-1]

## 5

### Problem description:

Write a function that calculates the factorial of a number.

### Buggy code:

def factorial(n):  
 if n == 0:  
 return 0  
 else:  
 return n \* factorial(n-1)

### One kind of right solution:

def factorial(n):  
 if n == 0:  
 return 1  
 else:  
 return n \* factorial(n-1)

## 6

### Problem description:

Write a function that finds the nth Fibonacci number.

### Buggy code:

def fibonacci(n):  
 if n <= 1:  
 return n  
 else:  
 return fibonacci(n-1) - fibonacci(n-2)

### One kind of right solution:

def fibonacci(n):  
 if n <= 1:  
 return n  
 else:  
 return fibonacci(n-1) + fibonacci(n-2)

## 7

### Problem description:

Implement a function that performs insertion sort on a list.

### Buggy code:

def insertion\_sort(lst):  
 for i in range(1, len(lst)):  
 key = lst[i]  
 j = i - 1  
 while j > 0 and key < lst[j]:  
 lst[j + 1] = lst[j]  
 j -= 1  
 lst[j + 1] = key  
 return lst

### One kind of right solution:

def insertion\_sort(lst):  
 for i in range(1, len(lst)):  
 key = lst[i]  
 j = i - 1  
 while j >= 0 and key < lst[j]:  
 lst[j + 1] = lst[j]  
 j -= 1  
 lst[j + 1] = key  
 return lst

## 8

### Problem description:

Write a function that checks if two strings are anagrams.

### Buggy code:

def are\_anagrams(s1, s2):  
 return s1 == sorted(s2)

### One kind of right solution:

def are\_anagrams(s1, s2):  
 return sorted(s1) == sorted(s2)

## 9

### Problem description:

Implement a function that performs quick sort on a list.

### Buggy code:

def quick\_sort(lst):  
 if len(lst) <= 1:  
 return lst  
 pivot = lst[len(lst) // 2]  
 left = [x for x in lst if x < pivot]  
 middle = [x for x in lst if x == pivot]  
 right = [x for x in lst if x > pivot]  
 return quick\_sort(left) + right + quick\_sort(middle)

### One kind of right solution:

def quick\_sort(lst):  
 if len(lst) <= 1:  
 return lst  
 pivot = lst[len(lst) // 2]  
 left = [x for x in lst if x < pivot]  
 middle = [x for x in lst if x == pivot]  
 right = [x for x in lst if x > pivot]  
 return quick\_sort(left) + middle + quick\_sort(right)

## 10

### Problem description:

Write a function that determines if a given string is a valid IP address.

### Buggy code:

def is\_valid\_ip(ip):  
 parts = ip.split('.')  
 if len(parts) != 4:  
 return False  
 for part in parts:  
 if not part.isdigit():  
 return False  
 num = int(part)  
 if num <= 0 or num >= 255:  
 return False  
 return True

### One kind of right solution:

def is\_valid\_ip(ip):  
 parts = ip.split('.')  
 if len(parts) != 4:  
 return False  
 for part in parts:  
 if not part.isdigit():  
 return False  
 num = int(part)  
 if num < 0 or num > 255:  
 return False  
 return True