1 Recursion

Every Recursive function has three things.
1. One or more base cases
2. One or more ways to break the problem down into a smaller problem
   • E.g. Given a number as input, we need to break it down into a smaller number
3. Solve the smaller problem recursively; from that, form a solution to the original problem

1. What is wrong with the following function? How can we fix it?
   def factorial(n):
       return n * factorial(n)

2. Complete the definition for all_true, which takes in a list lst and returns True if there are no False-y values in the list and False otherwise. Make sure that your implementation is recursive.
   def all_true(lst):
       ""
       >>> all_true([True, 1, "True"])
       True
       >>> all_true([1, 0, 1])
       False
       >>> all_true([])
       True
       """
3. Write a function `is_sorted` that takes in an integer `n` and returns true if the digits of that number are nondecreasing from right to left.

```python
def is_sorted(n):
    """
    >>> is_sorted(2)
    True
    >>> is_sorted(22222)
    True
    >>> is_sorted(9876543210)
    True
    >>> is_sorted(9087654321)
    False
    """
```
4. Draw the environment diagram that results from running the code.

```python
def bar(f, x):
    if x == 1:
        return f(x)
    else:
        return f(x) + bar(f, x - 1)

f = 4
bar(lambda x: x + f, 2)
```
5. Write a function that takes as input a number, \( n \), and a list of numbers, \( \text{lst} \), and returns \( \text{True} \) if we can find a subset of \( \text{lst} \) that sums up to \( n \).

```python
def add_up(n, lst):
    """
    >>> add_up(10, [1, 2, 3, 4, 5])
    True
    >>> add_up(8, [2, 1, 5, 4, 3])
    True
    >>> add_up(-1, [1, 2, 3, 4, 5])
    False
    >>> add_up(100, [1, 2, 3, 4, 5])
    False
    """
    if ________________________________:
        return True

    if lst == []:
        ________________________________

    else:
        first, rest = ____________________, ____________________
        return ________________________________
```