Homework 3

1 Without Coding [Show your work when asked]

Problem 1 (5 points each)

Calculate the result of the following function calls. Make sure to **show your work** tracing the recursive calls.

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

[*1*]

```
def x(n):
    if n <= 1:
        return 1
    else:
        return n*(n-1)*x(n-1)
def y(m):
    if m == 1:
        return 1
    else:
        n = x(m - 1)
        return n
```

[*2*]

```
# Function that finds the nth tribonacci number
def tribonacci(n):
    if n == 1:
        return 0
    elif n == 2:
        return 1
    elif n == 3:
        return 2
    else:
        return tribonacci(n-1) + tribonacci(n-2) + tribonacci(n-3)
```

[*3*]

Problems to Trace

1. factorial(5)
2. factorial(8)
3. y(8)
4. tribonacci(4)
5. tribonacci(7)

Problem 2 (10 points)

Find the expected result of the two list comprehension by hand (**show your work**):

```
list_1 = [10-thing for thing in range(20,1,-1)]
list_2 = [val**3 for val in [1,1,2,2,3,3,4,4] if val!=2]
```

[4]

2 Coding Problems

NOTE, you may NOT import any external modules in this assignment

Problem 3 (10 points)

Write a program that will take a list as input and return a list where the neighbors of each element are added to it. You do not have to use any specific methods.
The method signature must be neighbor_sum(a_list).

For example, neighbor_sum([1, 3, 5, 6, 9]) = [4, 9, 14, 20, 15]
The first index (1) adds only 3 to the original number and becomes 4.
The second index (3) adds 1 and 5 to the original number and becomes 9.
The third index (5) adds 3 and 6 to the original number and becomes 14.
The fourth index (6) adds 5 and 9 to the original number and becomes 20.
The fifth index (9) adds only 6 to the original number and becomes 15.

Problem 4 (15 points)

Using ONLY List Comprehension with a helper function, write a program that will take a list of annual-incomes for an individual and return the total federal income tax.

The method signature must be get_income_tax(list_of_incomes) and should be at most 2 lines of code (you may write a helper function for the expression inside the list comprehension). You are given that the list of incomes is for individuals filing as "Single" (and you can assume no deduction, exemption, or any other modifications apply). We will assume that all individuals will get the standard minimum deduction(In 2021 it is 12,550).

Below are the appropriate 2021 tax brackets.

Tax Rate	Single
10%	\$0 to \$9,950
12%	\$9,951 to \$40,525
22%	\$40,526 to \$86,375
24%	\$86,376 to \$164,925
32%	\$164,926 to \$209,425
35%	\$209,426 to \$523,600
37%	\$523,601 and higher

This means that someone who makes \$112,550 in a year owes,

$10\% * (9,950) + 12\% * (40,525 - 9,950) + 22\% * (86,375 - 40,525) + 24\% * (100,000 - 86,375) = 18021.00$ (effectively only 16%)

That is income fills in each of the tax brackets (paying that respective rate) until the final amount is reached. The final amount is just the income - the standard deduction.

Problem 5 (15 points)

Write a set operation class called SetSuite. Users of this class will only use and see lists, but under the hood we will use sets to make it work. It should have the following methods:

- __init__(list_of_lists): the input should be a list of sets (represented as lists that may have duplicates. Init should convert the lists into sets and store a list of sets
- add_set(a_list): adds a new set (input as a list with possible duplicates) to the internal list of sets
- get_sets(): returns the internal sets as a list of lists
- union_all(): returns a set that is the union of all sets in the class as a list of items
- intersection_all (): returns a set that is the intersection of all sets in the class as a list of items

Problem 6 (10 points)

Write a function that will generate the ith row of pascal's triangle (using **recursion**)

The method signature should be pascal(row). It should return a list of elements that would be in that row of Pascal's Triangle. For example the 0th row is [1] and the 2nd row is [1,2,1]

Problem 7 (10 points)

Write a function that will check if a number is a perfect power of another (using **recursion**).

The method signature should be perfect_power(num_1, num_2) and return a boolean of whether num_1 is a perfect power of num_1.
Examples:
perfect_power(5, 125) = True (5^3)
perfect_power(5, 25) = True (5^2)
perfect_power(5, 10) = False
perfect_power(2, 8) = True (2^3)

Extra Credit (10 points available)

Write a robust function that will convert a given number and base to base 10. It should be able to handle any base from 2 to 10. To convert from base b to base 10, go from right to left through the string and multiply each number by $b^{(position-1)}$ where position is the position in the string and add the results.

For example to convert 231 base 5 to base 10: $1 * 5^0 + 3 * 5^1 + 2 * 5^2 = 1 + 15 + 50 = 66$

It should expect input to be a string(for the represented number) and an int (for the base). It should use exception handling to not crash if the inputs are not valid.

The inputs it should be able to not crash on are:
convert_to_10("12x", 5): "Error: Not a Number" (entered something that is not a number)
convert_to_10("123", 2): "Error: Invalid Number" (123 is not a valid number in base 2)
convert_to_10("123", 1.5) "Error Invalid Base"(1.5 is not a valid base)
convert_to_10("123", "a"): "Error: Not A Number" (entered something that is not a number)

Instead of crashing on these inputs it should return the correct string from above (there are 4 errors to catch, and you should have 3 different exceptions defined). You must also throw an error internally (although this will be graded manually)

The method signature should be convert_to_10(number, base)

5 points are available for solving the problem
5 additional points are available if you solve it without using for/while loops (using recursion)

Submission Instructions

Submit 2 files:
hw3.py - For the coding part
hw3.pdf - For the written part

Submit both parts on **Gradescope.**

At the top of hw3.pdf and hw3.py, include the following information:

- Your name
- The name of any classmates you discussed the assignment with, or the words "no collaborators"
- A list of sources you used (textbooks, wikipedia, research papers, etc.) to solve the assignment, or the words "no sources"
- Whether you're using the extension or not

Grading Methodology

The **written part** of the assignment will be graded for correctness. In the case that your answer is incorrect, partial credit may be awarded based on the provided work.

The **code part** of the assignment will be graded by an automatic grading program. It will use the method signatures specified in the assignment and use multiple test cases.

5 points are allocated for proper comments and styling:

- Descriptive variable names
- Comments for what functions do
- Comments for complex parts of code
- Proper naming conventions for any Variables, Functions, and Classes