List Comprehensions

1. **Negative Numbers**: Create a list comprehension that multiplies each number in a given list by -1.
2. **Filter Vowels**: Generate a list of characters which are vowels from a given string.
3. **Create Tuples**: Form a list of tuples (n, n*n) for each number n in a given range.
4. **Square Numbers:** Create a list comprehension that squares each number in a given list.
5. **Even Numbers:** Use a list comprehension to filter out odd numbers from a list.
6. **Capitalize Words:** Given a list of words, use a list comprehension to capitalize each word.
7. **Divisible by Three:** Create a list comprehension that includes numbers from a list that are divisible by 3.
8. **Reverse Strings:** Reverse each string in a list of strings.
9. **Square Even, Cube Odd:** For a list of numbers, square the even numbers and cube the odd numbers.

Map

1. **Convert to Upper Case**: Use `map` to convert a list of strings to upper case.
2. **Inverse Numbers**: Given a list of non-zero numbers, use `map` to find their inverses (1/x).
3. **Boolean Conversion**: Convert a list of “truthy” and “falsy” values to their corresponding boolean values.
4. **Length of Each Word:** Use map to create a list containing the length of each word in a given list of words.
5. **Square Roots:** Given a list of numbers, use map to find the square root of each number.
6. **String to Integer:** Convert a list of numeric strings to a list of integers.
7. **Length of Words:** Use map to create a list of lengths of each word in a given sentence.
8. **Convert Fahrenheit to Celsius:** Given a list of temperatures in Fahrenheit, convert them to Celsius.
9. **Add Suffix:** Add a common suffix to all strings in a list.

Filter

1. **Prime Numbers**: Filter a list to include only prime numbers.
2. **Non-Empty Strings**: Filter out all empty strings from a list of strings.
3. Filter Specific Characters: Given a string, filter out all occurrences of a specified character.
4. **Filter Even Numbers:** Write a filter to get only even numbers from a list.

5. **Filter Positive Numbers:** Filter out all non-positive numbers from a list.
6. **Filter Long Words:** Given a list of words, filter all the words that have more than 4 characters.
7. **Filter Negative Numbers:** Write a filter to remove negative numbers from a list.
8. **Words Without Letter 'e':** Filter out words that contain the letter 'e' from a list of words.
9. **Keep Non-Zero:** Filter out zero from a list of numbers.

Reduce

1. **Sum of a List:** Use reduce to find the sum of all numbers in a list.
2. **Product of a List**: Use `reduce` to find the product of all numbers in a list.
3. **Find Minimum**: Use `reduce` to find the smallest number in a list.
4. **Find Maximum:** Use reduce to find the largest number in a list.
5. **Concatenate Words with Separator**: Given a list of words, use `reduce` to concatenate them into a single string, separated by a specified separator (like a comma or space).
6. **Concatenate Strings:** Given a list of strings, use reduce to concatenate them into a single string.
7. **Concatenate Strings with Condition:** Use reduce to concatenate strings in a list that are longer than 2 characters.
8. **Find Longest Word:** Use reduce to find the longest word in a list of words.
9. **Cumulative Multiplication:** Use reduce to multiply all numbers in a list, cumulatively (like factorial but not limited to integers)

Extra strings

1. **Ends with Certain Characters**: Create a list comprehension that includes strings from a given list only if they end with certain characters (e.g., 'ing', 'ed').
2. **Title Case Conversion**: Use a list comprehension to convert each string in a list to title case (first letter of each word capitalized), but only for strings that are not already in title case.
3. **Strings Containing Numbers**: Generate a list of strings that contain any numerical digits, filtering out those that don't contain digits.
4. **Alternate Uppercase and Lowercase**: For each string in the list, use a list comprehension to create a new string where the characters alternate between uppercase and lowercase.
5. **Strings without Specific Characters**: Create a list of strings that do not contain specific characters (e.g., 'a', 'b', 'c').
6. **Count Vowels**: Generate a list that contains the number of vowels in each string of the given list.

7. **Swap Case**: Use a list comprehension to swap the case of each character in each string in the list (lowercase to uppercase and vice versa).
8. **Filter by String Length**: Create a list of strings that are longer than a certain length, say 10 characters.
9. **String Reversal**: Write a list comprehension that reverses each string in the given list.
10. **Extract Email Domains**: From a list of email addresses, use a list comprehension to extract the domain of each email (the part after '@').
11. **String Lengths**: Create a list comprehension that gives the length of each string in a list of strings but only for strings longer than 5 characters.
12. **Uppercase Strings**: Use a list comprehension to convert all strings in a list to uppercase, but only if the string starts with a vowel.
13. **Strip Whitespace**: Generate a list of strings with leading and trailing whitespaces removed, using a list comprehension.
14. **Find Digits in Strings**: Extract all the digits from each string in a list. The result should be a list of lists, each containing the digits of a respective string.
15. **Concatenate with Conditions**: Using a list comprehension, concatenate adjacent strings in a list, but only if both strings have the same length.
16. **String Contains Substring**: Create a list of booleans, where each boolean indicates whether a corresponding string in a list contains a given substring.
17. **Replace Substrings**: Use a list comprehension to replace a specific substring with another substring in each string of the list, but only if the string ends with a certain suffix.
18. **Palindrome Check**: Generate a list of booleans, each indicating whether a corresponding string in the list is a palindrome.
19. **Count Specific Characters**: Create a list that contains the count of a specific character (like 'a') in each string of the given list.
20. **Sort by Inner Character**: Sort the strings in a list based on the second character of each string (assume all strings are at least two characters long).

Extra conjuntos

1. **Union of All Sets (Reduce)**: Given a list of sets, use `reduce` to compute the union of all sets.
2. **Intersection of Sets (Reduce)**: Apply `reduce` to find the intersection of a list of sets.
3. **Filter Disjoint Sets (Filter)**: From a list of sets, use `filter` to retain only those sets that are disjoint with a given set.
4. **Length of Each Set (Map)**: Use `map` to create a list containing the lengths of each set in a given list of sets.
5. **Set of Unique Characters (Map & List Comprehension)**: Given a list of strings, use `map` and a list comprehension to create a list of sets, each containing the unique characters of a corresponding string.
6. **Subset Check (Map)**: Given two lists of sets, A and B, use `map` to create a list of booleans, each indicating whether a set in A is a subset of the corresponding set in B.
7. **Unique Elements in Lists (List Comprehension)**: Use a list comprehension to convert each sublist in a list of lists into a set, thereby keeping only unique elements.

8. **Symmetric Difference (List Comprehension)**: For a list containing pairs of sets, use a list comprehension to create a list of sets, each representing the symmetric difference between the sets in each pair.
9. **Max and Min in Sets (Map)**: Apply `map` to a list of sets to create a list of tuples, each tuple containing the maximum and minimum of each set.
10. **Filter Sets by Size (Filter)**: Use `filter` to remove sets from a list of sets that have fewer than a specified number of elements.

Extra dictionaries

1. **Merge Two Dictionaries**: Write a function that merges two dictionaries into one. If there is an overlap in keys, prefer the values from the second dictionary.
2. **Key with Maximum Value**: Find the key associated with the highest value in a dictionary. If there are multiple keys with the same maximum value, return a list of those keys.
3. **Dictionary Filtering**: Create a new dictionary from a given dictionary by filtering out items based on a specific condition (e.g., values greater than a certain number).
4. **Invert a Dictionary**: Write a function to invert a dictionary, swapping its keys and values. If multiple keys have the same value, store the values in a list under the new key.
5. **Frequency Counter**: Given a string or list, create a dictionary where keys are unique elements and values are counts of their occurrences.
6. **Nested Dictionary Flattening**: Flatten a nested dictionary (a dictionary containing dictionaries) into a single-level dictionary by concatenating keys.
7. **Dictionary from Two Lists**: Create a dictionary using two lists: one for keys and the other for values. Assume both lists are of the same length.
8. **Default Value for Missing Keys**: Write a function to retrieve a value from a dictionary, but return a default value if the key is not found.
9. **Sort Dictionary by Value**: Sort a dictionary based on its values in ascending or descending order.
10. **Dictionary Comprehension**: Use dictionary comprehension to transform the values of a dictionary based on a certain condition (e.g., square the values if they are numbers).
11. **Sum Values in Dictionary (Map & Reduce):** Given a dictionary with numerical values, use `map` to extract the values and `reduce` to sum them up.
12. **Concatenate String Values (Map & Reduce)**: For a dictionary with string values, concatenate them into a single string using `map` and `reduce`.
13. **Count Word Frequency (Map & Reduce)**: Given a dictionary where each key is a document ID and each value is a list of words, use `map` and `reduce` to create a word frequency count across all documents.
14. **Find Maximum Value (Map & Reduce)**: Use `map` to extract values from a dictionary and `reduce` to find the maximum value.
15. **Merge Dictionaries (Map & Reduce)**: Given a list of dictionaries, merge them into a single dictionary.
16. **Average Value (Map & Reduce)**: Calculate the average of values in a dictionary.

17. **Create a Reverse Lookup Dictionary (Map & Reduce)** For a dictionary, create a reverse lookup dictionary where each value points to a list of keys that had that value.
18. **Aggregate Nested Dictionary Values (Map & Reduce)**: Given a dictionary with nested dictionaries as values, use `map` and `reduce` to aggregate inner dictionary values based on some criteria.

Problemas de pesquisa

1. **Number of full houses**

Consider a list of tuples representing a deck of cards to play the game of Poker:

```
deck = [('2', 'S'), ('3', 'S'), ('4', 'S'), ('5', 'S'), ('6', 'S'),
 ('7', 'S'), ('8', 'S'), ('9', 'S'), ('T', 'S'), ('J', 'S'),
 ('Q', 'S'), ('K', 'S'), ('A', 'S'),
 ('2', 'C'), ('3', 'C'), ('4', 'C'), ('5', 'C'), ('6', 'C'),
 ('7', 'C'), ('8', 'C'), ('9', 'C'), ('T', 'C'), ('J', 'C'),
 ('Q', 'C'), ('K', 'C'), ('A', 'C'),
 ('2', 'H'), ('3', 'H'), ('4', 'H'), ('5', 'H'), ('6', 'H'),
 ('7', 'H'), ('8', 'H'), ('9', 'H'), ('T', 'H'), ('J', 'H'),
 ('Q', 'H'), ('K', 'H'), ('A', 'H'),
 ('2', 'D'), ('3', 'D'), ('4', 'D'), ('5', 'D'), ('6', 'D'),
 ('7', 'D'), ('8', 'D'), ('9', 'D'), ('T', 'D'), ('J', 'D'),
 ('Q', 'D'), ('K', 'D'), ('A', 'D')]
```

For each tuple the first item is the card rank (2,3, dama, valete, rei) and the second one o tipo de carta (copas (H), espadas (S), paus (C), ouros (D)).
In the end of the poker game each player has 5 cards. Count all the “full-house” sets: the ones that have three cards of the same rank and two others of the same rank. Example: three aces and two kings.

2. **Subset sum problem**

Count all subsets of a set whose elements sum results in a certain number.

3. **Optimal seat arrangement**

Consider the existence of 4 seats in a theatre, where 4 persons shall be seated:

_ _ _ _

Seats are marked 0 to 3 from left to right. The preferences of seating for each person range from 1 to 4 and are kept in a dictionary, where values are 4-sized lists containing the preferences for each person respective to each seat.

{“P1”: [1,2,3,4], “P2”: [2, 3, 1, 4], “P3”: [3, 4, 1, 2], “P4”: [4, 1, 3, 2]}

Implement a brute force program that finds the seat arrangement that maximizes the satisfaction of the people.

4. **All possible outcomes of a rolling dice**

Generate all possible sets of k dice rolls that results in, at least, n sixes.

5. **Backpack problem**

You have several items each with a specific weight given in a list of tuples. For instance:

lw = [(1, 20), (2, 30), (3, 10), (4, 5), (5, 50)]

where the first item of the tuple is the identification of the item and the second item is the weight. Knowing that you are limited to a certain weight W, find the solution that has more items, and in the case of a draw between two solutions, choose the one that has less weight remaining.