1. Write a **function** that loops through a word and returns a list with every other letter of the word starting with the **first** letter. The function will take a single argument *word* (a string representing the word to process).

   Examples:

   - skip_letter("counterattack") → ["c","u","t","r","t","a","c"]
   - skip_letter("banana sunday") → ["b","n","n","s","n","a"]

2. Write a **function** that loops through a word and returns a list with every other letter of the word starting with the **second** letter. The function will take a single argument *word* (a string representing the word to process).

   Examples:

   - skip_letter("counterattack") → ["o","n","e","a","t","c"]
   - skip_letter("banana sunday") → ["a","a","a","s","n","a"]

3. Write a **function** that loops through and returns a list with every even number between two integers (inclusive). The arguments to the function will be *smaller_num* and *larger_num*.

   Examples:

   - output_even(37, 1050) → [38, 40, 42, . . . 1048, 1050],
   - output_even(1, 2000) → [2, 4, 6, . . . 1998, 2000],
   - output_even(50, 199) → [50, 52, 54, . . . 196, 198]

4. Write a **function** that loops through and returns a list with every odd number between two integers (inclusive). The arguments to the function will be *smaller_num* and *larger_num*.

   Examples:

   - output_odd(37, 1050) → [37, 39, 41, . . . , 1049],
   - output_odd(1, 2000) → [1, 3, 5, . . . , 1999],
   - output_odd(50, 199) → [51, 53, 55, . . . , 197, 199]

5. Given a positive integer n, the following rules will always create a sequence that ends with 1, called Hailstone Sequence:

   (a) If n is even, divide by 2
   (b) If n is odd, multiply by 3 and add 1 (i.e. $3n + 1$)
   (c) Continue until n is 1

   Write a **function** that returns a list with the hailstone sequence starting at n. The argument to the function will be n (the integer to start the sequence from). **Examples:**

   - hailstone_seq(25) → [25, 76, 38, 19, 58 ... 8, 4, 2, 1],
   - hailstone_seq(40) → [40, 20, 10, 5, 16, 8, 4, 2, 1]

6. Write a **function** that returns a list with the factors of a given integer. The argument of the function will be *num* (integer to find factors for).

   Examples:

- find_factors(12) → [1, 2, 3, 4, 6, 12],
- find_factors(17) → [1, 17],
- find_factors(36) → [1, 2, 3, 4, 6, 9, 12, 18, 36]

7. Write a **function** that takes 3 numbers as arguments, *num_1* (first number), *num_2* (second number), and *num_3* (third number). Return a list of the integers in ascending order. You may **not** use the built-in functions *max*(), *min*(), *sort*(), or *sorted*().

   Examples:

   - ascending_order(2, 3, 1) → [1, 2, 3],
   - ascending_order(10, 1, 25) → [1, 10, 25],
   - ascending_order(2, 45, 4) → [2, 4, 45]

8. Write a **function** that takes 3 numbers as arguments, *num_1* (first number), *num_2* (second number), and *num_3* (third number). Return a list of the integers in descending order. You may **not** use the built-in functions *max*(), *min*(), *sort*(), or *sorted*().

   Examples:

   - descending_order(2, 3, 1) → [3, 2, 1],
   - descending_order(10, 1, 25) → [25, 10, 1],
   - descending_order(2, 45, 4) → [45, 4, 2]

9. Write a function named *add_lists* that takes two lists *lyst*1 and *lyst*2 and adds the first element in *lyst*1 with the first element in *lyst*2, the second element *lyst*1 with the second element *lyst*2, etc. Return a new list containing the corresponding sums of the list1 and list2. You may assume both lists have the same length.

   Examples:

   - add_lists([1, 3, 3, 1], [4, 3, 6, 1]) → [5, 6, 8, 2] (since 1+4=5; 3+3=6; 3+6=9; 1+1=2)
   - add_lists([1, 8, 5, 0, -7], [0, -7, 4, 2, -6]) → [1, 1, 9, 2, -13]
   - add_lists([1, 2], [-1, 1]) → [0, 3]

10. Write a **function** that finds the largest even number in a list *numbers*. Return -1 if not found. You may **not** use the built-in functions *max*(), *min*(), *sort*(), or *sorted*().

    Examples:

    - largest_even([3, 7, 2, 1, 7, 9, 10, 13]) → 10,
    - largest_even([1, 3, 5, 7]) → -1,
    - largest_even([0, 19, 18973623]) → 0

11. Write a **function** that finds the largest odd number in a list *numbers*. Return -1 if not found. You may **not** use the built-in functions *max*(), *min*(), *sort*(), or *sorted*().

    Examples:

    - largest_odd([3, 7, 2, 1, 7, 9, 10, 13]) → 13,
    - largest_odd([2, 4, 6, 8]) → -1,
    - largest_odd([0, 19, 18973623]) → 18973623

12. YouTube currently displays a like and a dislike button, allowing you to express your opinions about particular content. It's set up in such a way that you cannot like and dislike a video at the same time. There are two other interesting rules to be noted about the interface:

(a) Pressing a button, which is already active, will undo your press.

(b) If you press the like button after pressing the dislike button, the like button overwrites the previous "dislike" state. The same is true for the other way round.

Write a **function** that takes in a list of button inputs *events* and returns the final state.

Examples:

- like_or_dislike(["dislike"]) → "dislike",
- like_or_dislike(["like", "like"]) → "nothing",
- like_or_dislike(["dislike", "like"]) → "like",
- like_or_dislike(["like", "dislike", "dislike"]) → "nothing",

13. Write a **function** that takes two arguments, a list and an item. The function should return the indices of all occurrences of the *item* in the list.

    Examples:

    - get_indices([1, 5, 5, 2, 7], 7) → [4]
    - get_indices([1, 5, 5, 2, 7], 5) → [1, 2]
    - get_indices([1, 5, 5, 2, 7], 8) → []
    - get_indices(["a", "a", "b", "a", "b", "a"], "a") → [0, 1, 3, 5]

14. Write a **function** that takes two numbers as arguments *num* and *length* and returns a list of multiples of *num* until the list length reaches *length*.

    Examples:

    - list_of_multiples(7, 5) → [7, 14, 21, 28, 35]
    - list_of_multiples(12, 10) → [12, 24, 36, 48, 60, 72, 84, 96, 108, 120]
    - list_of_multiples(17, 6) → [17, 34, 51, 68, 85, 102]

15. In BlackJack, cards are counted with -1, 0, 1 values:

    - 2, 3, 4, 5, 6 are counted as +1
    - 7, 8, 9 are counted as 0
    - 10, j, q, k, a are counted as -1

    Write a **function** that takes a list called *cards*, counts the number, and returns it from the list of cards provided.

    Examples:

    - count([5, 9, 10, 3, "j", "a", 4, 8, 5]) → 1,
    - count(["a", "a", "k", "q", "q", "j"]) → -6,
    - count(["a", 5, 5, 2, 6, 2, 3, 8, 9, 7]) → 5

16. There's a great war between the even and odd numbers. Many numbers already lost their lives in this war and it's your task to end this. You have to determine which group sums larger: the evens or the odds. The larger group wins.

    Write a **function** that takes a list of integers named *numbers*, sums the even numbers and odd numbers separately, then returns which of the two sums is larger.

    Examples:

- war_of_numbers([2, 8, 7, 5]) → "odds", (since 2 + 8 = 10, 7 + 5 = 12, odds is larger)
- war_of_numbers([12, 90, 75, 1, 1]) → "evens", (12 + 90 = 102, 75 + 1 + 1 = 77, evens is larger)
- war_of_numbers([2, 10, 22, 243]) → "odds"

17. To train for an upcoming marathon, Johnny goes on one long-distance run each Saturday. He wants to track how often the number of miles he runs exceeds the previous Saturday. This is called a progress day. Write a **function** that takes in a list of miles run every Saturday and returns Johnny's total number of progress days.

    Examples:

    - progress_days([3, 4, 1, 2]) → 2, (Two progress days, day 2 since (4 > 3) and day 4 since (2 > 1))
    - progress_days([10, 11, 12, 9, 10]) → 3,
    - progress_days([6, 5, 4, 3, 2, 9]) → 1,
    - progress_days([9, 9]) → 0

18. To train for an upcoming marathon, Samuel goes on one long-distance run each Saturday. He wants to track how often the number of miles he runs fall short of the previous Saturday. This is called a lag day. Write a **function** that takes in a list of miles run every Saturday and returns Samuel's total number of lag days.

    Examples:

    - lag_days([5, 3, 2, 1]) → 3, (3 lag days, day2 since (3<5), day3 since (2<3), and day4 since (1<2))
    - lag_days([10, 11, 12, 9, 10]) → 1,
    - lag_days([6, 5, 4, 3, 2, 9]) → 4,
    - lag_days([9, 9]) → 0

19. Let ***s*** be a string and ***words*** be a list of strings. The string ***s*** is considered an acronym of ***words*** if it can be formed by concatenating the first character of each string in ***words*** in order. For example, `"ab"` can be formed from [`"apple"`, `"banana"`], but it can't be formed from [`"bear"`, `"aardvark"`]. Write a function that takes a string ***s*** and a list of strings ***words***, and returns True if ***s*** is an acronym of ***words***, and False otherwise.

    Examples:

    - is_acronym(`"abc"`, [`"alice"`, `"bob"`, `"charlie"`]) → True
      Explanation: The first character in the words `"alice"`, `"bob"`, and `"charlie"` are `"a"`, `"b"`, and `"c"`, respectively. Hence, s = `"abc"` is the acronym.
    - is_acronym(`"a"`, [`"an"`, `"apple"`]) → False
      Explanation: The first character in the words `"an"` and `"apple"` are `"a"` and `"a"`, respectively. The acronym formed by concatenating these characters is `"aa"`. Hence, s = `"a"` is not the acronym.
    - is_acronym(`"ngguoy"`, [`"never"`, `"gonna"`, `"give"`, `"up"`, `"on"`, `"you"`]) → True
      Explanation: By concatenating the first character of the words in the array, we get the string `"ngguoy"`. Hence, s = `"ngguoy"` is the acronym.
    - is_acronym(`"ab"`, [`"apple"`, `"banana"`, `cat`]) → False
      Explanation: Wrong length
    - is_acronym(`"ab"`, [`"apple"`, `""`, `cat`]) → False
      Explanation: Can't get the first letter in the second string of ***words***.