Sheet 2

2023/09/26

1. Write a function called `modify_odd` that takes a mutable reference to an array slice of integers `slice` and sets all odd numbers to 0.
   Then write a second function that create a Vec, filled with all numbers from 0 to 100, and pass it to `modify_odd`;
2. Write a function `count_character` that takes a string consisting of ASCII characters `string` as input and returns a HashMap. The keys of the HashMap should be the characters in the string, and the values should be an u32 representing how many times each character appears in the string.
3. Write a function named `split_at_value` that takes two arguments: a slice of i32 called `slice` and a single i32 value called `value`. The function should find the first element equal to `value` inside `slice`. It should then split the slice at the corresponding index and return the two resulting slices wrapped in an Option.
   If `value` is not found in `slice`, the function should return None.
4. Write a function `sub_slice` that takes two `&Vec<i32>` as input. If the second vector is contained inside the first one it print the corresponding slice, otherwise it print `Not found`;
5. Write the following functions, for each of the functions think carefully about what is the best way to pass the arguments (&, &mut or passing ownership):

   - Write a function `max` that takes a Vec of i32 and returns the maximum value inside it.
   - Write a function `swap` that swaps the first and last element of a vector of i32.
   - Write a function `is_sorted` that takes a Vec of i32 and returns a boolean indicating whether the vector is sorted in non-decreasing order.
   - Write a function `insert_if_longer` that takes a Vec of String (`vec`) and a String (`string`). This function should insert `string` into `vec` only if the length of `string` is greater than 10.

   Also, when possible, implement these functions recursively, not iteratively.
6. Write a function `build_vector` that takes a `Iter<i32>` and returns the `Vec<&i32>` containing all the elements of the iterator;
7. Write a function `pancake_sort` that takes a `&mut Vec<i32>` and sorts it using the pancake sort algorithm;
8. Write a function `merge` that takes two `&[i32]` and returns a `Vec<i32>`. The returned vector should contain the elements of the two passed elements sorted, you can assume that the passed slices are sorted;
9. Create a `Vec` that can contain both an `i32` and a `String`;

10. Write these enums to represent a mathematical expression:

    - One enum is called `Operation` and can be: `Add`, `Sub`, `Mul`, `Div`.
    - One enum is called `Expression` an can be:
        - `Number` (contain inside an i32)
        - `Operation` (contain inside a left Expression, a right Expression and an Operation)

    Note: the left and right expression must be wrapped around a Box

    ```
    Box<Expression>.
    ```

    You will see Boxes further into the course, from now you just need to know that you can build a box using

    ```
    let my_box = Box::new(my_expression)
    ```

    and you can get the value inside the box by dereferencing it

    ```
    let value_inside = *my_box
    ```

    Write a function `evaluate_expression` that take as input an Expression, and return a Result with a i32 if the result is evaluated correctly, or a string if an error occurs.