Rust Developer Profile Set-1

// 1. Check whether a given string is a palindrome or not

```
fn is_palindrome(s: &str) -> bool {
   let cleaned: String = s.chars()
                    .filter(|c| c.is_alphanumeric())
                    .map(|c| c.to_ascii_lowercase())
                    .collect();
   cleaned == cleaned.chars().rev().collect::<String>()
}

fn main() {
   println!("1. Check whether a given string is a palindrome or not:");
   println!("{}", is_palindrome("A man, a plan, a canal, Panama"));  // true
   println!("{}", is_palindrome("racecar"));                  // true
   println!("{}", is_palindrome("hello"));                   // false
   println!();
```

// 2. Return the index of the first occurrence of a given number in a sorted array

```
   fn first_occurrence(arr: &[i32], target: i32) -> Option<usize> {
      let mut left = 0;
      let mut right = arr.len();

      while left < right {
```

```
        let mid = (left + right) / 2;
        if arr[mid] < target {
            left = mid + 1;
        } else {
            right = mid;
        }
    }

    if left < arr.len() && arr[left] == target {
        Some(left)
    } else {
        None
    }
}

println!("2. Return the index of the first occurrence of a given number in a sorted array:");
println!("{:?}", first_occurrence(&[1, 2, 2, 3, 4, 5], 2));  // Some(1)
println!("{:?}", first_occurrence(&[1, 2, 3, 4, 5], 3));     // Some(2)
println!("{:?}", first_occurrence(&[1, 2, 3, 4, 5], 6));     // None
println!();
```

// 3. Return the shortest word in a string

```
fn shortest_word(s: &str) -> &str {
    s.split_whitespace().min_by_key(|word| word.len()).unwrap_or("")
}

println!("3. Return the shortest word in a string:");
println!("{}", shortest_word("The quick brown fox jumps over the lazy dog"));  // "The"
```

```
println!("{}", shortest_word("Lorem ipsum dolor sit amet"));          // "sit"
println!("{}", shortest_word("Hello world"));                         // "world"
println!();
```

// 4. Check whether a given number is prime or not

```
fn is_prime(n: u32) -> bool {
    if n <= 1 {
        return false;
    }
    if n <= 3 {
        return true;
    }
    if n % 2 == 0 || n % 3 == 0 {
        return false;
    }
    let mut i = 5;
    while i * i <= n {
        if n % i == 0 || n % (i + 2) == 0 {
            return false;
        }
        i += 6;
    }
    true
}

println!("4. Check whether a given number is prime or not:");
println!("{}", is_prime(11));  // true
println!("{}", is_prime(4));   // false
```

```
println!();
```

// 5. Return the median of a sorted array

```
fn median_of_sorted_array(arr: &[i32]) -> f64 {
    let n = arr.len();
    if n % 2 == 0 {
        (arr[n / 2 - 1] as f64 + arr[n / 2] as f64) / 2.0
    } else {
        arr[n / 2] as f64
    }
}

println!("5. Return the median of a sorted array:");
println!("{}", median_of_sorted_array(&[1, 2, 3, 4, 5]));   // 3.0
println!("{}", median_of_sorted_array(&[1, 2, 3, 4, 5, 6]));  // 3.5
println!();
```

// 6. Find the longest common prefix of a given set of strings

```
fn longest_common_prefix(strs: &[&str]) -> String {
    if strs.is_empty() {
        return String::new();
    }
    let mut prefix = strs[0].to_string();
    for s in &strs[1..] {
        while !s.starts_with(&prefix) {
            prefix.pop();
            if prefix.is_empty() {
                return prefix;
```

```
        }
      }
    }
    prefix
}

println!("6. Find the longest common prefix of a given set of strings:");
println!("{}", longest_common_prefix(&["flower", "flow", "flight"]));  // "fl"
println!("{}", longest_common_prefix(&["dog", "racecar", "car"]));    // ""
println!();
```

// 7. Return the kth smallest element in a given array

```
fn kth_smallest(mut arr: Vec<i32>, k: usize) -> Option<i32> {
    arr.sort();
    arr.get(k - 1).cloned()
}

println!("7. Return the kth smallest element in a given array:");
println!("{:?}", kth_smallest(vec![3, 2, 1, 5, 6, 4], 2));  // Some(2)
println!("{:?}", kth_smallest(vec![3, 2, 3, 1, 2, 4, 5, 5, 6], 4));  // Some(3)
println!();
```

// 8. Return the maximum depth of a binary tree

```
#[derive(Debug, PartialEq, Eq)]
struct TreeNode {
    val: i32,
    left: Option<Box<TreeNode>>,
    right: Option<Box<TreeNode>>,
```

```
}

impl TreeNode {
    fn new(val: i32) -> Self {
        TreeNode { val, left: None, right: None }
    }
}

fn max_depth(root: Option<Box<TreeNode>>) -> i32 {
    match root {
        Some(node) => 1 + std::cmp::max(max_depth(node.left), max_depth(node.right)),
        None => 0,
    }
}

let root = Some(Box::new(TreeNode {
    val: 1,
    left: Some(Box::new(TreeNode::new(2))),
    right: Some(Box::new(TreeNode::new(3))),
}));

println!("8. Return the maximum depth of a binary tree:");
println!("{}", max_depth(root));  // 2
println!();

// 9. Reverse a string in Rust
fn reverse_string(s: &str) -> String {
    s.chars().rev().collect()
```

```
}

println!("9. Reverse a string in Rust:");
let s = "hello";
println!("{}", reverse_string(s));  // "olleh"
println!();
```

// 10. Check if a number is prime in Rust

```
println!("10. Check if a number is prime in Rust:");
println!("{}", is_prime(11));  // true
println!("{}", is_prime(4));   // false
println!();
```

// 11. Merge two sorted arrays in Rust

```
fn merge_sorted_arrays(arr1: Vec<i32>, arr2: Vec<i32>) -> Vec<i32> {
   let mut merged = Vec::with_capacity(arr1.len() + arr2.len());
   let mut i = 0;
   let mut j = 0;

   while i < arr1.len() && j < arr2.len() {
      if arr1[i] < arr2[j] {
         merged.push(arr1[i]);
         i += 1;
      } else {
         merged.push(arr2[j]);
         j += 1;
      }
   }
```

```
    while i < arr1.len() {
        merged.push(arr1[i]);
        i += 1;
    }

    while j < arr2.len() {
        merged.push(arr2[j]);
        j += 1;
    }

    merged
}

println!("11. Merge two sorted arrays in Rust:");
let arr1 = vec![1, 3, 5];
let arr2 = vec![2, 4, 6];
println!("{:?}", merge_sorted_arrays(arr1, arr2));  // [1, 2, 3, 4, 5, 6]
println!();
```

// 12. Find the maximum subarray sum in Rust

```
fn max_subarray_sum(nums: Vec<i32>) -> i32 {
    let mut max_sum = nums[0];
    let mut current_sum = nums[0];

    for &num in nums.iter().skip(1) {
        current_sum = std::cmp::max(num, current_sum + num);
        max_sum = std::cmp::max(max_sum, current_sum);
```

```
        }

        max_sum
    }

    println!("12. Find the maximum subarray sum in Rust:");
    let nums = vec![-2, 1, -3, 4, -1, 2, 1, -5, 4];
    println!("{}", max_subarray_sum(nums));  // 6
}
```