A. Basic

1. **Prime numbers**

Write a Javascript function that takes an array of numbers and returns a new array of prime numbers filtered from the old array.

Note: if looping is needed, use only primitive for/while/do-while loop (forearch, map, etc are not allowed).

2. **Palindrome**

Write a function that takes a string and returns true/false based on whether it is a palindrome or not (you can use a dictionary to know what a palyndrom is).

3. **Array reversing**

Write a Javascript function that takes an array of numbers and return a reversed version

4. **Inplace Array reversing**

Write a Javascript function that takes an array of numbers and return a reversed version. Note: here you are not allowed to use an intermediary array. You need to use the same array passed in the parameter and just reverse its content. Do not use inbuilt functions like reverse(). Just use loops and conditions.

5. **Array & Object**

1. Write a function that takes a formatted array. The array is made of a string of people's identities in a predefined format,
   "first-name second-name, age, gender" . Ex "Patrick wyne, 30, male"

   ["Patrick wyne, 30, male", "lil wyne, 32, male","Eric mimi, 21, female","Dodos deck, 21,male","Alian Dwine, 22, male","Patrick wyne, 33, male","Patrick wyne, 10,male0","Patrick wyne, 40,male"]

From the array, the function returns a nested object of two arrays: one for all females and another for all males.. Each object in the array is object with keys, first-name and the corresponding child object is the rest of the info, *second name and age*

```
{
        females: [Eric: {second-name:mimi, age:21}],
        males: [    patrick: {second-name:wyne, age:30},lil: {second-name:wyne, age:30},etc]
```

}

B. Sorting

1. Custom sorting

Write a function that sorts an array in a descending order. The exception is that if it encounters a prime number, it should erase it from the array.

Note: this should be entirely from scratch no inbuilt sorting functions allowed.

Note: no String function or processing is allowed.
Only use raw mathematical operations, loops and conditional statements.

C. Time complexity

1. [Optiona] We would like to know if a particular array contains a majority element. An array has a majority element if there is at least one element that counts more than half of the size.

   Ex:
   a. [3,1,3,4,4,5,3,5,3,3,3,6,3]: this array contains a majority because the occurrence of 3 is 7 out of total 13 elements.
   b. [3,1,3,4,4] has no majority element as none of the elements counts more than the half of the array size.

Phase 2 : Advanced

1. Asynchronous Js & Error Handling

Let us mock a small API for setting age for an student,

```
let setStudentAgeApi = (student, age) => {  //async code below
   console.log("1. Starting ..")
```

```
    setTimeout(() => {
       console.log("2. setting age for the student")
       student.age = age;
     }, 500);
    console.log("3. Done ..")


}
```

a. Try to create a student object: let student = {name: “Eric”}
b. Then call the `setStudentAgeApi` function with the above parameter, with the age of your choice.
   Eg. of testing:
   `let student = { name: "denis" }`

```
c.    setStudentAgeApi(student, 20)
d.        console.log(student)
```

e. Analyze the results well and try to understand what is happening.
f. Take a screenshot of your results you will share with your Manager
g. I guess you realized that the order of logs & result of log(student) is not what you expected, that is what we call Asynchronous behavior of Javascript APIs.

We can force the code to behave async (linear). This is because in most of the cases we shall need to wait for a certain event to happen before we proceed with the next lines of our code.

The most popular way is to use **Promises**

Let us force our `setStudentAgeApi` function to return a promise that we can use later.

```
setStudentAgeApi = (student, age) => {


   return new Promise(function (resolve, reject) {
       setTimeout(() => {
           student.age = age;
           if(age < 0)
               reject ("Bad Age")
           else
                resolve(student)
       },
           500);

   });
}
```

Your task here is to

1. Call the newly customised function using **.then** notation. Here pass a positive age (>0)
2. Now, try to provide a negative age, found that there is an unhandled error/exception That I set to throw when the age is negative
3. Now handle the error/exception

2. Practicing what you have learnt

Do your own api that waits 1 second and it should do the following

1. It takes an array of objects.
    a. Each object is a family with keys **fatherName**, **MotherName** and **childrenNumber**

2. It sets to all objects of the array a new member, totalNumberofFamilyMembers
3. If any family has a father called Yves (case insensitive), it should throw an error of ***Yves is not an allowed dad in 2022.***
4. Make sure to handle all thrown errors when you call/invoke your api.

Observation
1. Invoke your method with a array of   family objects
2. Log the family object: console.log (family): this should log the object

Your task is to use await,

Note: keep remove the await   again and again and learn the behavior of what you have developed.