Desarrollo 1: De los métodos o propiedades que tenemos para recorrer y modificar un árbol DOM indique cuales son las acordes a las siguientes descripciones :

a) Obtener una referencia al nodo padre:
b) Obtener una referencia al último nodo hijo:
c) Obtener una referencia al nodo anterior en el mismo nivel jerárquico:
d) Método para comprobar si un nodo tiene hijos:

a) let nodoReferencia = document.getElementById("idDelNodo");

let nodoPadre = nodoReferencia.parentNode;

b) let nodoHijo = nodoPadre.lastElementChild;

c) let nodoHermanoAnterior = nodoHijo.previousElementSibling;

d) let nodoPadre.hasChildNodes() == true, nodoPadre.children.length==0,

Desarrollo 2: Complemente el siguiente código en el cual se implementa una función capaz de crear un nodo párrafo, con texto, con un atributo "class" y por último, se añade este nodo párrafo como hijo del nodo que se pasa por parámetro. Para su implementación debe de usar los métodos y propiedades estudiados para tal cometido:

```
function crearParrafo( nodo ) {
            //Crear elemento párrafo:
            let parrafo =  ;
            //Crear contenido para el párrafo con el texto "Párrafo":
            let nodoTexto = ;
            //Añadir el contenido al nuevo nodo párrafo:
            //Añadir al párrafo el atributo "class" con el valor "clasecss":
            //Añadir párrafo al nodo pasado por parámetro:
      }

      crearParrafo(document.body);  //Ejemplo de uso
```

```
<script>
    function crearParrafo(nodo){
        //Crear elemento parrafo
        var parrafo=document.createElement("p");
        //Crear contenido para el parrafo con el texto "Parrafo":
        var nodoTexto=document.createTextNode("Parrafor");
        //Añadir el contenido al nuevo nodo parrafo
        parrafo.appendChild(nodoTexto);
        //Añadir al parrafo el atributo "class" con el valor "clasecss":
        parrafo.setAttribute("class","miClase");
        parrafo.setAttribute("src","http://blablabla.com")
    }
</script>
```

Desarrollo 3: Añada al nodo **h1** dos eventos: "**onmouseover**" y "**onmouseout**". Aplícales a ambos eventos **la misma función callback** con el nombre que quieras. En dicha función, debes de implementar la lógica para detectar cual de los dos eventos ha disparado la función. Muestra por consola un mensaje con el nombre del evento que haya "triggeado" la función.

```
<body>
      <h1 id="eventos">Obtener información de un evento</h1>
      <script>
      // Código aquí:
      </script>
</body>
```

```
<script>
    window.onload=iniciar;
    function iniciar(){
        let nodoH1=document.getElementById("eventos");
        nodoH1.addEventListener("mouseover", callback);
        nodoH1.addEventListener("mouseout", callback);
    }
    function callback(varMagica){
        console.log("¿Cual ha sido el evento producido?");
    }
    let tipoEvento=varMagica.type;
    console.log(tipoEvento);
</script>
```

Desarrollo 4: ¿De qué otra forma podemos controlar los estados del objeto XMLHttpRequest cuando hacemos una implementación alternativa al *"httpRequest.onreadystatechange"* ? Explica la alternativa

httpRequest.onload para hacer algo con la respuesta cuando todo haya ido bien o bien usar httpRequest.onerror para hacer algo cuando la respuesta ha ido mal…

httpRequest.onprogress

```
<script>
    function ajaxXhttp(){
        xhttp=new XMLHttpRequest();
        xhttp.onreadystatechange = function(){
            if(this.readyState==4 && this.status == 200){
                let json = JSON.parse(xhttp.responseText);
                displayData(json);
            }
        }
        xhttp.onload = function(){
            xhttp.onerror = function(){console.log("Algo ha ido mal");}
            xhttp.onprogress = function(){console.log("Estamos haciendo la llamada");}
        }
    }
</script>
```

Desarrollo 5: Implemente en el siguiente código el código necesario para ejecutar la función **"doTask"** y sea capaz de mostrar por consola, si todo va bien, el campo **"value"**, y si hay un error, el campo **"message"**.

```
function doTask (iterations) {
      return new Promise( (resolve, reject) => {
            const numbers = [];
            for (let i = 0; i < iterations; i++) {
                const number = 1 + Math.floor(Math.random() * 6);
                numbers.push(number);
                if (number === 6) {
                    reject({
                        error: true,
                        message: "Se ha sacado un 6"
                    });
                }
            }
            resolve({
                error: false,
                value: numbers ,
                valores : [ wfwef, effwef, wewef, wefwef]
            });
        })};
```

doTask(2)

. then ((objDevRes) => { console.log (objDevRes.value) })

.catch ((objError) => { console.log (objError.message)})

Desarrollo 6: Explique para qué se utilizan los métodos **Promise.any() y Promise.race()**. Deje bien claro **la diferencia principal** que existe entre estos 2 métodos estáticos del objeto Promise.

- La diferencia con el operador "any" es que al comando "race" le da igual que la primera promesa que termina haya sido "resolved" o "rejected". Sea como sea, la primera que termina es la que se devuelve.

Código a desacoplar: libreriaDatos.js

```
function llamadaServidor(url )
{
        fetch( url )
        .then(  response => {
            if (response.status == 200) {
                return response.json();
            }
            else {
                throw new Error(response.status);
            }
        })
        .then(   json =>
             obtenerDatosAlumnos(json);
        )
}

function obtenerDatosAlumnos( json )
{
      console.log(  json.nombre );
}

llamadaServidor("alumnos.json");
```

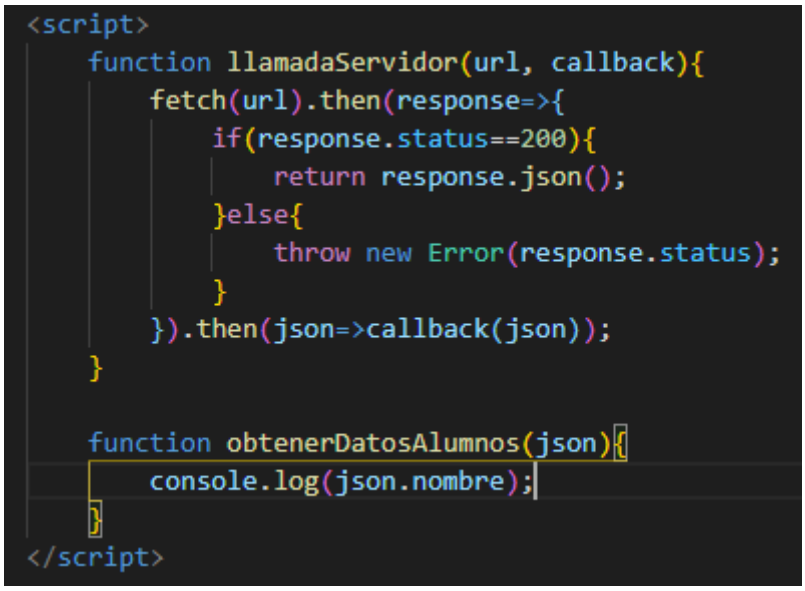
Y aquí el desacoplado a completar:

libreriaDatos.js

```
<script srt="libreriaNetwork.js"></script>


function obtenerDatosAlumnos( ...  )
{

       console.log(  .....   .nombre );



}


llamadaServidor( ... );
```

libreriaNetwork.js

```
function llamadaServidor(    )
{
        fetch(url)
        .then(  response => {
            if (response.status == 200) {

                return response.json();

            }
            else {

                throw new Error(response.status);

            }
        })
        .then(   json => {

        ] )
}
```


<script>
function llamadaServidor(url, callback){
fetch(url).then(response=>{
if(response.status==200){
return response.json();
}else{
throw new Error(response.status);
}
}).then(json=>callback(json));
}
function obtenerDatosAlumnos(json){
console.log(json.nombre);
}
</script>