Módulo 6 - Algoritmos I - Práctica

Algoritmo para calcular la letra del DNI

- La entrada debe ser un número entre 0 y 99999999.
- Debemos calcular el resto de la división entera entre el número introducido y el número 23.
- Según el resultado, de 0 a 22, le corresponderá una letra de las siguientes: `(T, R, W, A, G, M, Y, F, P, D, X, B, N, J, Z, S, Q, V, H, L, C, K, E)`
- Si lo introducido no es un número deberá indicarse con un `alert` al usuario y volver a preguntar los datos.
- Deberá repetirse el proceso hasta que el usuario pulse "Cancelar".

Vamos por partes. Definimos la entrada del algoritmo que será el número que introduzca el usuario.

```
var number;
// Paso 1: pedir datos al usuario
number = prompt("Introduce tu número de DNI");
```

Primer requisito: `La entrada debe ser un número`. Sabemos que la función `prompt` recogerá el texto introducido por el usuario como un `string` y nosotros vamos a trabajar con `number`, así que vamos a comprobar que efectivamente, lo que ha introducido puede ser convertido a número.

```
var number;
number = prompt("Introduce tu número de DNI");

// Paso 2: comprobar que es un número
if (parseInt(number) === Number(number)) {
    number = Number(number); // Lo reasignamos transformado en número
} else {
    alert(number + " No es un número");
}
```

Una vez tenemos controlado que es un número, debemos acotar los valores al requisito `La entrada debe ser un número entre 0 y 99999999`.

```
var number;
number = prompt("Introduce tu número de DNI");

if (parseInt(number) === Number(number)) {
    number = Number(number);

    // Paso 3: comprobar que es un número válido
    if (number >= 0 && number <= 99999999) {

    } else {
        alert("El número de DNI introducido es erróneo");
    }

} else {
    alert(number + " No es un número");
}
```

Una vez hemos comprobado que se trata de un número de DNI válido, vamos a ver cómo podemos devolver la letra correspondiente.

Según la web del Ministerio del Interior, nos dice que el resto como resultado de dividir el número de DNI entre 23, se corresponde a una letra en la siguiente tabla:

RESTO	0	1	2	3	4	5	6	7	8	9	10	11
LETRA	T	R	W	A	G	M	Y	F	P	D	X	B

RESTO	12	13	14	15	16	17	18	19	20	21	22
LETRA	N	J	Z	S	Q	V	H	L	C	K	E

Por ejemplo, si el número del DNI es 12345678, dividido entre 23 da como resto 14, luego la letra sería la Z: 12345678Z.

Así que, sabiendo esto, vamos a actualizar nuestro código.

```
var number;
// Agregamos las variables necesarias
var rest;
var letter;

number = prompt("Introduce tu número de DNI");

if (parseInt(number) === Number(number)) {
    number = Number(number);

    if (number >= 0 && number <= 99999999) {
        // Paso 4: obtener el resto y la letra correspondiente
        rest = number % 23;
        switch (rest) {
            case 0:
                letter = "T";
                break;
            case 1:
                letter = "R";
                break;
            case 2:
                letter = "W";
                break;
            case 3:
                letter = "A";
                break;
            case 4:
                letter = "G";
                break;
            case 5:
                letter = "M";
                break;
            case 6:
                letter = "Y";
                break;
            case 7:
                letter = "F";
                break;
            case 8:
                letter = "P";
                break;
            case 9:
                letter = "D";
                break;
            case 10:
                letter = "X";
                break;
            case 11:
                letter = "B";
                break;
            case 12:
                letter = "N";
                break;
            case 13:
                letter = "J";
                break;
            case 14:
                letter = "Z";
                break;
            case 15:
                letter = "S";
                break;
```

```
            case 16:
                letter = "Q";
                break;
            case 17:
                letter = "V";
                break;
            case 18:
                letter = "H";
                break;
            case 19:
                letter = "L";
                break;
            case 20:
                letter = "C";
                break;
            case 21:
                letter = "K";
                break;
            case 22:
                letter = "E";
                break;
        }
    alert("Número: " + number + ", Letra: " + letter);
    number = null;
    } else {
        alert("El número de DNI introducido es erróneo");
    }
} else {
    alert(number + " No es un número");
}
```

Siguiente requisito: Si lo introducido no es un número deberá indicarse con un `alert` al usuario y `volver a preguntar los datos`.

Y además: `Deberá repetirse el proceso hasta que el usuario pulse "Cancelar"`.

Pues como vemos, tenemos que editar nuestro algoritmo para que, si lo que el usuario ha introducido no es un número, vuelva a preguntar.

¿Cómo podemos repetir nuestro código MIENTRAS algo ocurra?

Pues para este caso usaremos la sentencia `do...while`. Vamos a envolver nuestro código con un `do...while`

```
var number;
// Paso 5: envolvemos el código con un do...while. La declaración de number la dejamos fuera para usarla en el `whil
do {
    var rest;
    var letter;

  number = prompt("Introduce tu número de DNI");

    if (parseInt(number) === Number(number)) {
        number = Number(number);

        if (number >= 0 && number <= 99999999) {

            rest = number % 23;
            switch (rest) {
                case 0:
                    letter = "T";
                    break;
                case 1:
                    letter = "R";
                    break;
                case 2:
                    letter = "W";
                    break;
                case 3:
                    letter = "A";
                    break;
                case 4:
                    letter = "G";
```

```
                    break;
                case 5:
                    letter = "M";
                    break;
                case 6:
                    letter = "Y";
                    break;
                case 7:
                    letter = "F";
                    break;
                case 8:
                    letter = "P";
                    break;
                case 9:
                    letter = "D";
                    break;
                case 10:
                    letter = "X";
                    break;
                case 11:
                    letter = "B";
                    break;
                case 12:
                    letter = "N";
                    break;
                case 13:
                    letter = "J";
                    break;
                case 14:
                    letter = "Z";
                    break;
                case 15:
                    letter = "S";
                    break;
                case 16:
                    letter = "Q";
                    break;
                case 17:
                    letter = "V";
                    break;
                case 18:
                    letter = "H";
                    break;
                case 19:
                    letter = "L";
                    break;
                case 20:
                    letter = "C";
                    break;
                case 21:
                    letter = "K";
                    break;
                case 22:
                    letter = "E";
                    break;
            }

            alert("Número: " + number + ", Letra: " + letter);
            number = null; // Finalizamos el código
        } else {
            alert("El número de DNI introducido es erróneo");
        }
    } else {
        alert(number + " No es un número");
    }
} while (number !== null); // <- condición a comprobar después de cada ejecución
```

Veamos qué hemos hecho. A parte de envolver el código con la sentencia, hemos agregado como condición a verificar que `number` sea distinto de `null`

¿Recordáis por qué?

Si `prompt` tenía como selección Cancelar, recordad que devolvía un `null`. Pues estamos aprovechando este resultado para que la ejecución del código no pare hasta que el usuario cancele la petición de datos.

Una vez el usuario nos ha respondido correctamente, estamos forzando la no repetición del bucle asignando `null` a la variable number.

Y para finalizar, si no queremos que le muestre al usuario que `null No es un número`, podemos agregar esta condición antes de mostrar el texto por pantalla:

```
var number;

do {
    var rest;
    var letter;
    // Paso 1: pedir datos al usuario
    number = prompt("Introduce tu número de DNI");

    // Paso 2: comprobar que es un número
    if (parseInt(number) === Number(number)) {
        number = Number(number);

        // Paso 3: comprobar que es un número válido
        if (number >= 0 && number <= 99999999) {
            // Paso 4: obtener el resto y la letra correspondiente
            rest = number % 23;
            switch (rest) {
                case 0:
                    letter = "T";
                    break;
                case 1:
                    letter = "R";
                    break;
                case 2:
                    letter = "W";
                    break;
                case 3:
                    letter = "A";
                    break;
                case 4:
                    letter = "G";
                    break;
                case 5:
                    letter = "M";
                    break;
                case 6:
                    letter = "Y";
                    break;
                case 7:
                    letter = "F";
                    break;
                case 8:
                    letter = "P";
                    break;
                case 9:
                    letter = "D";
                    break;
                case 10:
                    letter = "X";
                    break;
                case 11:
                    letter = "B";
                    break;
                case 12:
                    letter = "N";
                    break;
                case 13:
                    letter = "J";
                    break;
                case 14:
                    letter = "Z";
                    break;
                case 15:
                    letter = "S";
```

```
                    break;
                case 16:
                    letter = "Q";
                    break;
                case 17:
                    letter = "V";
                    break;
                case 18:
                    letter = "H";
                    break;
                case 19:
                    letter = "L";
                    break;
                case 20:
                    letter = "C";
                    break;
                case 21:
                    letter = "K";
                    break;
                case 22:
                    letter = "E";
                    break;
            }

            alert("Número: " + number + ", Letra: " + letter);
            number = null; // Finalizamos el código
        } else {
            alert("El número de DNI introducido es erróneo");
        }
    } else {
    // Si pulsó "Aceptar" sin introducir un número...

    // paso 6: Si pulsó "Cancelar", no mostrar el mensaje
        if (number !== null) {
            alert(number + " No es un número");
        }
    }
} while (number !== null);
```

No obstante, aunque este algoritmo es correcto y cumple los requisitos, tiene bastante espacio de mejora. ¿Puedes pensar en una alternativa más optimizada?

Un poco más adelante encontrarás una versión más depurada.

Formulario de registro

Para este ejercicio, debemos comprobar si los datos en un formulario de registro son correctos.

Los datos que introducirá el usuario serán:

- Nombre completo.
- Fecha de nacimiento.
- DNI.
- Teléfono.

* Comenzamos con una estructura HTML básica.

./src/index.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1" />
        <title>Bootcamp Javascript - Módulo 6 - Algoritmos</title>
        <link rel="stylesheet" href="app.css" />
    </head>

    <body>
```

```
        <header>
            <h1>Formulario de registro</h1>
        </header>
        <main>
            <form id="register" autocomplete="off">
                <label for="fullName">Nombre y apellidos *</label>
                <input
                    type="text"
                    id="fullName"
                    name="fullName"
                    placeholder="Tu nombre completo"
                />

                <label for="birthday">Fecha de nacimiento *</label>
                <input
                    type="text"
                    id="birthday"
                    name="birthday"
                    placeholder="dd/mm/aaaa"
                />

                <label for="dni">DNI *</label>
                <input type="text" id="dni" name="dni" placeholder="12345678-A" />

                <label for="mobile">Teléfono *</label>
                <input
                    type="text"
                    id="mobile"
                    name="mobile"
                    placeholder="Tu teléfono"
                />

                <span class="age">Es mayor de edad <b id="age"></b></span>

                <input type="submit" id="submit" value="Enviar" class="btn" />
            </form>
        </main>
        <script src="app.js"></script>
    </body>
</html>
```

* Agregamos el css.

./src/app.css

```
body {
    margin: 0;
    font-family: Arial, Helvetica, sans-serif;
}

header {
    text-align: center;
}

main {
    margin: auto;
    width: 70%;
    border-radius: 5px;
    background-color: #f2f2f2;
    padding: 20px;
}

input[type="text"] {
    width: 100%;
    padding: 12px;
    border: 1px solid #ccc;
    border-radius: 4px;
    box-sizing: border-box;
    margin-top: 6px;
    margin-bottom: 16px;
    resize: vertical;
}
```

```
input[type="text"].error {
    color: red;
    border: 1px solid red;
}

input[type="submit"] {
    background-color: #4caf50;
    color: white;
    padding: 12px 20px;
    border: none;
    border-radius: 4px;
    cursor: pointer;
}

input[type="submit"]:hover:not([disabled]) {
    background-color: #45a049;
}

input[type="submit"]:disabled {
    background-color: #ddd;
}
```

* Y un fichero con el código JavaScript.

./src/app.js

```
document
    .getElementById("register")
    .addEventListener("submit", validateForm, true);

function validateForm(event) {
    console.log("COMIENZA PROCESO VALIDACIÓN");
}
```

Una vez hemos dado forma a nuestra página de registro, vamos a probar que tenemos enganchado la función que validará el formulario. Si hacemos click en el botón de "Enviar", deberíamos ver por consola "COMIENZA PROCESO VALIDACIÓN".

Comprobado esto último, vamos a comenzar con elaborar nuestro algoritmo para validar los datos del formulario.

Análisis preliminar del problema

Primero, analizamos el problema y vemos que tenemos una serie de campos que debemos comprobar para enviar nuestro formulario:

- Nombre y apellidos
- Fecha de nacimiento
- DNI
- Teléfono

Requisitos

- Debemos comprobar que el campo `Nombre y apellidos` tiene una longitud mínima de 5 caracteres.
  - Como opcionales: comprobar que tiene la longitud mínima quitando los espacios al inicio y al final.
- El campo `Fecha de nacimiento` debe ser una fecha válida. Ten en cuenta que es un input de texto, por lo que debemos validar que el string escrito siga el formato de fecha "dd/mm/aaaa". Además, mostraremos por pantalla si es mayor de edad o no en base a la fecha introducida.
- Para el campo `DNI`, debemos validar que el número introducido junto con su letra son correctos.
  - Como opcionales: podríamos comprobar el número con distintos formatos 12345678-A, 12345678A, etc...
- Finalmente el campo `Teléfono` será válido si efectivamente es un número de 9 dígitos y que comience por 9, 8, 7 o 6.

Identificación de los apartados

Tenemos varios apartados que serán cada uno de los distintos tipos de validaciones.

Además, tenemos un apartado que será para la ejecución de la validación principal del formulario, es decir, cuando se comienza el proceso del resto de validaciones.

También hay que comunicar al usuario si se ha producido un error en la introducción de los campos.

Como opcional, podemos decir el tipo de error que ha ocurrido en el proceso de validación.

Creación del algoritmo

Como ya tenemos definido los apartados vamos a comenzar con el sub-algoritmo de validación de `Nombre y apellidos`:

```
// 1. Full name validation.
var isValidFullName = fullname => fullname && fullname.length >= 5;

var validateFullName = () => {
    var fullNameField = document.getElementById("fullName");
    var valid = isValidFullName(fullNameField.value);

    if (valid) {
        fullNameField.classList.remove("error");
    } else {
        fullNameField.classList.add("error");
    }

    return valid;
};
```

Añadimos el sub-algoritmo para validar `Fecha de nacimiento`:

```
// 2a. Birthdate validaton.
var isValidYear = year =>
    year && year >= 1850 && year < new Date().getFullYear();
var isValidMonth = month => month >= 1 && month <= 12;
var isLeapYear = year =>
    year % 400 === 0 || (year % 100 !== 0 && year % 4 === 0);
var isValidDay = (day, month, year) => {
    var monthLength = [
        31,
        isLeapYear(year) ? 29 : 28,
        31,
        30,
        31,
        30,
        31,
        31,
        30,
        31,
        30,
        31,
    ];
    return day > 0 && day <= monthLength[month - 1];
};

var splitDateInParts = date => {
    var parts = date.split("/");
    return [parseInt(parts[0]), parseInt(parts[1]), parseInt(parts[2])];
};

var isValidDate = date => {
    if (date.length < 8 || date.length > 10) return false;
    var parts = splitDateInParts(date); // [day, month, year]
    return (
        isValidYear(parts[2]) &&
        isValidMonth(parts[1]) &&
        isValidDay(parts[0], parts[1], parts[2])
    );
};

var validateBirthdate = () => {
    var birthdayField = document.getElementById("birthday");
    var valid = isValidDate(birthdayField.value);

    if (isValidDate(birthdayField.value)) {
        birthdayField.classList.remove("error");
    } else {
```

```
        birthdayField.classList.add("error");
    }

    return valid;
};
```

Añadiremos un pequeño apartado para chequear si el usuario es mayor de edad en base a la fecha de nacimiento introducida, y en tal caso, lo indicaremos por pantalla:

```
// 2b. Birthdate over 18.
var isOver18 = date => {
    var parts = splitDateInParts(date); // [day, month, year]

    var nowDate = new Date();
    var limitDate = nowDate.setFullYear(nowDate.getFullYear() - 18);

    return limitDate >= new Date(parts[2], parts[1] - 1, parts[0]); // Months in 0 base
};

var showIsAdult = () => {
    var birthdayField = document.getElementById("birthday");
    document.getElementById("age").innerHTML = isOver18(birthdayField.value)
        ? "SÍ"
        : "NO";
};
```

También un sub-algoritmo para comprobar que `DNI` es correcto:

```
// 3. DNI validation.
var DNI_LETTERS = "TRWAGMYFPDXBNJZSQVHLCKET";

var isValidDNILetter = (dniLetter, dniNumber) =>
    DNI_LETTERS[dniNumber % 23] === dniLetter.toUpperCase();
var isValidDNINumber = number => number >= 0 && number <= 99999999;

var isValidDNI = dni => {
    if (dni.length !== 9) return false;

    var dniNumber = dni.slice(0, 8);
    var dniLetter = dni.slice(8, 9);

    return isValidDNINumber(dniNumber) && isValidDNILetter(dniLetter, dniNumber);
};

var validateDNI = () => {
    var dniField = document.getElementById("dni");
    var valid = isValidDNI(dniField.value);

    if (valid) {
        dniField.classList.remove("error");
    } else {
        dniField.classList.add("error");
    }

    return valid;
};
```

Y finalmente comprobamos el campo `Teléfono`:

```
// 4. Mobile validation.
var isValidMobile = mobile => {
    return (
        parseInt(mobile) &&
        mobile.length === 9 &&
        (mobile.startsWith(9) ||
            mobile.startsWith(8) ||
            mobile.startsWith(7) ||
            mobile.startsWith(6))
    );
};
```

```
var validateMobile = () => {
    var mobileField = document.getElementById("mobile");
    var valid = isValidMobile(mobileField.value);

    if (valid) {
        mobileField.classList.remove("error");
    } else {
        mobileField.classList.add("error");
    }

    return valid;
};
```

Ahora solo nos queda implementar el algoritmo general que valida todo el formulario. Dicha validación tendrá lugar cuando se haga click en el botón `Enviar`. Por tanto:

```
// 5. General Algorithm.
var validateForm = event => {
    event.preventDefault();

    validateFullName();
    if (validateBirthdate()) showIsAdult();
    validateDNI();
    validateMobile();
};

// EVENTS
document.getElementById("register").addEventListener("submit", validateForm);
```

* Probamos nuestro código para comprobar que todo funciona correctamente.

Empezamos etapa de refinamiento, y para ello nos fijamos si hay código que podamos optimizar. Si observamos detenidamente, todas las funciones de validación principales, como `validateDNI`, `validateMobile`, etc., hacen exactamente lo mismo. Es código que se repite, con ligeras variaciones:

- Extrae el elemento HTML que representa un campo a través de su ID.
- Valida el valor de dicho campo.
- Cambia su CSS en función de la validación.
- Devuelve el resultado de la validación.

Podríamos implementar una función más genérica que haga esto mismo, es decir, validar cualquier input. Para ello debe solicitar por parámetros aquellos elementos que cambien de un input a otro:

```
var validateInput = (id, validationFunction, onValidCallback) => {
    var field = document.getElementById(id);
    var valid = validationFunction(field.value);

    if (valid) {
        field.classList.remove("error");
        onValidCallback && onValidCallback();
    } else {
        field.classList.add("error");
    }

    return valid;
};
```

Y ahora adaptamos la validación de formulario consecuentemente:

```
var validateForm = event => {
    event.preventDefault();

    validateInput("fullName", isValidFullName);
    validateInput("birthday", isValidDate, showIsAdult);
    validateInput("dni", isValidDNI);
```

```
    validateInput("mobile", isValidMobile);
};
```

* Vayamos un paso más alla **¿Podrías actualizar el código para que validara los campos según se escribieran en ellos?**

Aprovechando que hemos dividido y encapsulado en funciones concisas y sencillas, esta tarea ahora es inmediata, y nos proporciona validación en vivo en nuestro formulario a medida que los campos cambian.

En primer lugar definimos funciones de validación para cada input:

```
var validateFullName = () => validateInput("fullName", isValidFullName);
var validateBirthdate = () =>
    validateInput("birthday", isValidDate, showIsAdult);
var validateDNI = () => validateInput("dni", isValidDNI);
var validateMobile = () => validateInput("mobile", isValidMobile);
```

Y ahora las utilizamos en nuestro `validateForm` y también las registraremos como `eventListener` de los eventos `change` de cada input:

```
var validateForm = event => {
    event.preventDefault();
    validateFullName();
    validateBirthdate();
    validateDNI();
    validateMobile();
};

// EVENTS
document.getElementById("register").addEventListener("submit", validateForm);
document
    .getElementById("fullName")
    .addEventListener("change", validateFullName);
document
    .getElementById("birthday")
    .addEventListener("change", validateBirthdate);
document.getElementById("dni").addEventListener("change", validateDNI);
document.getElementById("mobile").addEventListener("change", validateMobile);
```

* Comprobemos el funcionamiento.

Este último paso no es necesario para este ejemplo donde tenemos pocos campos en nuestro formulario. Pero si pensamos en formularios complejos, con muchos campos, o que se generan dinámicamente sin saber a priori cuantos campos hay, podríamos hacer nuestro código más robusto y automático si registramos previamente los IDs de cada input con su correspondiente función de validación.

Esto se puede hacer con un simple objeto a modo de mapa del siguiente modo:

```
var inputs = {
    fullName: () => validateInput("fullName", isValidFullName),
    birthday: () => validateInput("birthday", isValidDate, showIsAdult),
    dni: () => validateInput("dni", isValidDNI),
    mobile: () => validateInput("mobile", isValidMobile),
};
```

La idea, es que sólo tengamos que tocar dicho registro en un futuro, si hay nuevos campos o si se eliminan otros. Con esta abstracción, ahora podemos hacer el `validateForm` y el registro de eventos 100% automático en base a dicho registro:

```
var validateForm = event => {
    event.preventDefault();
    for (const id in inputs) {
        inputs[id]();
    }
};

// EVENTS
document.getElementById("register").addEventListener("submit", validateForm);
```

```
for (const id in inputs) {
    document.getElementById(id).addEventListener("change", inputs[id]);
}
```