EJERCICIOS DE VALIDACIÓN 01

NOTA SOBRE LOS ARCHIVOS NECESARIOS:

Cada ejercicio necesita un archivo básico llamado index.html, que debe contener:

- Los campos necesarios (inputs) indicados en cada ejercicio.
- Una referencia al script JavaScript donde colocarás las funciones.

Si necesitas el archivo HTML básico, aquí tienes un ejemplo genérico para adaptarlo a cada ejercicio:

ARCHIVO HTML BÁSICO:

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Validación Ejercicios</title>
</head>
<body>
    <input type="text" id="campo" placeholder="Introduce dato">
    <button onclick="ejecutarValidacion()">Validar</button>
    <script src="validaciones.js"></script>
</body>
</html>
```

EJERCICIO 1

Enunciado:
Crea una función JavaScript que valide si un campo está vacío o no.

Archivos necesarios:

- index.html (con un input y un botón para ejecutar la validación)

Explicación:

- trim(): elimina espacios en blanco al inicio y al final.
- length: devuelve la longitud del texto

Solución:

```
function validarCampoVacio(texto) {
// Verifica que el texto sin espacios tenga longitud mayor que cero
    return texto.trim().length > 0;
}
```

EJERCICIO 2

Enunciado:
Realiza una validación que compruebe si la edad ingresada es un número válido entre 18 y 99 años.

Archivos necesarios:

- index.html (campo de edad)

Explicación:

- Number(): convierte una cadena a número.
- isNaN(): determina si un valor no es un número.
- Condicional (>=, <=): verifica el rango numérico.

Solución:

```
function validarEdad(edad) {
    const edadNum = Number(edad);
// Comprueba si es número y está dentro del rango permitido
    return !isNaN(edadNum) && edadNum >= 18 && edadNum <= 99;
}
```

EJERCICIO 3

Enunciado:
Crea una función para validar que un correo electrónico tenga formato válido.

Archivos necesarios:

- index.html (campo de email)

Explicación:

- Expresión regular (regex): valida patrones específicos.
- Método test(): verifica si una cadena cumple con la expresión regular.

Solución:

```
function validarEmail(email) {
// Patrón básico para validar formato de correo electrónico
    const regex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
    return regex.test(email);
}
```

EJERCICIO 4

Enunciado:
Implementa una función que valide la longitud mínima de un campo contraseña (mínimo 8 caracteres).

Archivos necesarios:

- index.html (campo contraseña)

Explicación:

- Propiedad length: evalúa cantidad de caracteres en una cadena.

Solución:

```
function validarLongitudContrasena(pass) {
// Comprueba que la contraseña tenga al menos 8 caracteres
    return pass.length >= 8;
}
```

EJERCICIO 5

Enunciado:
Realiza una función que valide que dos campos de contraseña coincidan (contraseña y repetir contraseña).

Archivos necesarios:

- index.html (2 campos de contraseña)

Explicación:

- Operador ===: verifica igualdad estricta (valor y tipo).

Solución:

```
function validarCoincidencia(pass1, pass2) {
// Verifica si las contraseñas coinciden exactamente
    return pass1 === pass2;
}
```

EJERCICIO 6

Enunciado:
Crea una validación personalizada para un número de teléfono con exactamente 9 dígitos numéricos.

Archivos necesarios:

- index.html (campo teléfono)

Explicación:

- Método replace(): reemplaza caracteres usando expresiones regulares.
- Expresión regular (\D): elimina caracteres no numéricos.

Solución:

```
function validarTelefono(telefono) {
// Elimina caracteres no numéricos y verifica longitud exacta
    const telLimpio = telefono.replace(/\D/g, '');
    return telLimpio.length === 9;
}
```

EJERCICIO 7

Enunciado:
Valida un nombre asegurando que no tenga números ni caracteres especiales (solo letras y espacios).

Archivos necesarios:

- index.html (campo nombre)

Explicación:

- Expresión regular (regex): [a-zA-Z\s] permite letras y espacios.
- Método test(): valida el patrón definido.

Solución:

```
function validarNombre(nombre) {
// Acepta solo letras mayúsculas, minúsculas y espacios
    const regex = /^[a-zA-Z\s]+$/;
    return regex.test(nombre);
}
```

EJERCICIO 8

Enunciado:
Implementa validación en tiempo real (on-the-fly) en JavaScript, resaltando el borde del campo en rojo si es incorrecto.

Archivos necesarios:

- index.html (campo texto)

Explicación:

- Evento input: activa función cuando cambia el contenido del input.
- Propiedad CSS (style.borderColor): modifica estilo visual en tiempo real.

Solución:

```
document.getElementById('campo').addEventListener('input',
function() {
// Cambia borde a rojo si el campo está vacío, normal si está lleno
    this.style.borderColor = this.value.trim() === '' ? 'red' : '';
});
```

EJERCICIO 9

Enunciado:
Realiza una función que valide un código postal (5 dígitos numéricos).

Archivos necesarios:

- index.html (campo código postal)

Explicación:

- Expresión regular (\d{5}): valida exactamente cinco dígitos.

Solución:

```
function validarCodigoPostal(cp) {
// Comprueba exactamente 5 dígitos numéricos
    return /^\d{5}$/.test(cp);
}
```

EJERCICIO 10

Enunciado:
Crea una función general que valide múltiples campos a la vez (nombre, edad, email) y devuelva mensajes específicos por cada error detectado.

Archivos necesarios:

- index.html (campos nombre, edad, email)

Explicación:

- Uso de array (errores): guarda mensajes específicos de validación.
- Condicionales: revisan individualmente cada regla de validación.
- Método push(): agrega elementos al array.

Solución:

```
function validarFormulario(nombre, edad, email) {
    let errores = [];
// Revisa individualmente cada campo y genera errores específicos
    if (!nombre.trim()) errores.push("Nombre no puede estar
vacío.");
    if (!(Number(edad) >= 1 && Number(edad) <= 120))
errores.push("Edad debe ser entre 1 y 120.");
    if (!/^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(email))
errores.push("Email no válido.");
    return errores;
}
```

EXPLICACIÓN GENERAL DE MÉTODOS Y FUNCIONES EMPLEADOS EN ESTOS EJERCICIOS:

- **trim()**: Método String para eliminar espacios al inicio y al final.
- **length**: Propiedad para obtener la cantidad de caracteres en cadenas.

- **Number()**: Convierte una cadena a número.
- **isNaN()**: Evalúa si un valor es o no numérico.
- **regex (expresiones regulares)**: Permite definir patrones específicos para validar cadenas.
- **test()**: Método para verificar si un string cumple un patrón dado por una expresión regular.
- **Eventos (input)**: Eventos de JavaScript que responden dinámicamente a cambios en elementos HTML.
- **Manipulación de estilos CSS**: Permite resaltar visualmente campos con errores.

EJERCICIOS DE VALIDACIÓN 02

ARCHIVOS BÁSICOS NECESARIOS:

Ejemplo genérico de archivo HTML:

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Ejercicios adicionales Validación</title>
</head>
<body>
    <input type="text" id="campo" placeholder="Introduce dato">
    <button onclick="ejecutarValidacion()">Validar</button>

    <!-- Vincula archivo JavaScript -->
    <script src="validaciones.js"></script>
</body>
</html>
```

EJERCICIO 11: VALIDAR URL

Enunciado:
Crea una función que valide si una cadena ingresada es una URL válida.

Explicación:

- Expresión regular para validación básica de URL.
- Método test() para verificar patrón.

Solución:

```
function validarURL(url) {
    const regex = /^(http|https):\/\/[^\s$.?#].[^\s]*$/;
    return regex.test(url);
}
```

EJERCICIO 12: VALIDAR NÚMERO ENTERO POSITIVO

Enunciado:
Crea una función que valide si el valor introducido es un número entero positivo.

Explicación:

- Método Number.isInteger() valida enteros.

- Condicional (> 0) para validar número positivo.

Solución:

```
function validarEnteroPositivo(numero) {
    const num = Number(numero);
    return Number.isInteger(num) && num > 0;
}
```

EJERCICIO 13: VALIDAR FECHA FORMATO DD/MM/AAAA

Enunciado:
Valida que una fecha esté en formato día/mes/año (DD/MM/AAAA).

Explicación:

- Expresión regular específica para formato de fecha.
- Método test().

Solución:

```
function validarFecha(fecha) {
    const regex = /^(0[1-9]|[12][0-9]|3[01])\/(0[1-9]|1[0-
2])\/\d{4}$/;
    return regex.test(fecha);
}
```

EJERCICIO 14: VALIDAR SELECCIÓN DE OPCIÓN EN MENÚ DESPLEGABLE

Enunciado:
Verifica si el usuario ha seleccionado una opción válida distinta al valor por defecto.

Explicación:

- Compara valor seleccionado con valor por defecto.

Solución:

```
function validarSeleccion(valorSeleccionado, valorPorDefecto) {
    return valorSeleccionado !== valorPorDefecto;
}
```

EJERCICIO 15: VALIDAR CADENA SIN ESPACIOS

Enunciado:
Valida que una cadena no contenga espacios en blanco.

Explicación:

- Método includes() revisa si cadena contiene espacios.

Solución:

```
function validarSinEspacios(texto) {
    return !texto.includes(' ');
}
```

EJERCICIO 16: VALIDAR CAMPO NUMÉRICO CON DECIMALES

Enunciado:
Valida que un campo numérico permita números con hasta dos decimales.

Explicación:

- Expresión regular específica para números con decimales.

Solución:

```
function validarNumeroDecimal(numero) {
    const regex = /^\d+(\.\d{1,2})?$/;
    return regex.test(numero);
}
```

EJERCICIO 17: VALIDAR NOMBRE DE USUARIO ALFANUMÉRICO (SIN CARACTERES ESPECIALES)

Enunciado:
Valida que el nombre de usuario sea solo letras y números.

Explicación:

- Expresión regular [a-zA-Z0-9]+ restringe a letras y números.

Solución:

```
function validarUsuario(usuario) {
    const regex = /^[a-zA-Z0-9]+$/;
    return regex.test(usuario);
}
```

EJERCICIO 18: VALIDAR TEXTO QUE CONTENGA MÍNIMO UNA LETRA MAYÚSCULA

Enunciado:
Crea una función que valide si el texto ingresado contiene al menos una mayúscula.

Explicación:

- Expresión regular [A-Z] revisa existencia de letras mayúsculas.

Solución:

```
function contieneMayuscula(texto) {
    const regex = /[A-Z]/;
    return regex.test(texto);
}
```

EJERCICIO 19: VALIDAR FORMATO DNI ESPAÑOL (8 NÚMEROS Y 1 LETRA)

Enunciado:
Valida que un DNI español tenga el formato correcto (ej.: 12345678Z).

Explicación:

- Expresión regular específica para formato DNI.

Solución:

```
function validarDNI(dni) {
    const regex = /^\d{8}[A-Za-z]$/;
    return regex.test(dni);
}
```

EJERCICIO 20: VALIDAR CHECKBOX OBLIGATORIO (TÉRMINOS Y CONDICIONES)

Enunciado:
Valida que un checkbox obligatorio (como términos y condiciones) esté seleccionado.

Explicación:

- Propiedad checked de elementos input tipo checkbox.

Solución:

```
function validarCheckboxObligatorio(checkbox) {
    return checkbox.checked;
}
```

RESUMEN EXPLICATIVO DE MÉTODOS Y FUNCIONES UTILIZADOS EN ESTOS EJERCICIOS ADICIONALES:

- **Expresiones regulares (regex)**:
  Permiten validar patrones específicos en cadenas de texto.

- **test()**:
  Evalúa si un string cumple una expresión regular definida.

- **Number()** y **Number.isInteger():**
  Convierte texto en número y verifica si es entero.

- **includes()**:
  Comprueba existencia de un substring dentro de una cadena.

- **checked**:
  Propiedad específica de inputs tipo checkbox para validar si están marcados.