Temario examen 3ª

XSL, XSLT y XPath

1. Introducción a XSL

XSL (eXtensible Stylesheet Language):

- **Descripción**: Lenguaje diseñado para trabajar con documentos XML.
- **Propósito**: Descripción de la presentación y estructura de contenido XML.
- **Funcionalidad**: Permite definir estilos y formatos para los datos en XML.

2. XSLT (Transformaciones XSL)

XSLT (XSL Transformations):

- **Descripción**: Permite transformar documentos XML en otros formatos (HTML, XML modificado).
- **Funcionalidades**: Manipulaciones como ordenación, filtrado y adición de estilos.

Definición de un archivo XSLT:

Cabecera:

```
xmlCopiar código
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

- **Descripción**: Define la versión de XSLT y el espacio de nombres utilizado.

Asociación con XML:

```
xmlCopiar código
<?xml-stylesheet type="text/xsl" href="archivo.xsl"?>
```

- **Descripción**: Asocia el archivo XML con su correspondiente hoja de estilo XSLT.

Ejemplo básico de transformación:

- **Transformar XML de libros en una página HTML**:

XML de ejemplo:

```
xmlCopiar código
<catalog>
    <book>
        <title>Harry Potter i la pedra filosofal</title>
        <author>J.K. Rowling</author>
        <price>20.00</price>
    </book>
    <book>
        <title>Crim i càstig</title>
        <author>Fyodor Dostoevsky</author>
        <price>18.50</price>
    </book>
</catalog>
```

- **Descripción**: Archivo XML con información de libros.

XSLT de transformación:

```
xmlCopiar código
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <!-- Plantilla para transformar elementos 'book' -->
    <xsl:template match="book">
        <div class="book">
            <!-- Extrae y muestra el título del libro -->
            <h2><xsl:value-of select="title"/></h2>
            <!-- Extrae y muestra el autor del libro -->
            <p>Autor: <xsl:value-of select="author"/></p>
            <!-- Extrae y muestra el precio del libro -->
            <p>Preu: $<xsl:value-of select="price"/></p>
        </div>
    </xsl:template>

    <!-- Plantilla para transformar el elemento 'catalog' -->
    <xsl:template match="catalog">
        <html>
            <head>
                <title>Llibreria Online</title>
            </head>
            <body>
                <h1>Col·lecció de Llibres</h1>
                <!-- Aplica la plantilla a cada elemento 'book' -->
                <xsl:apply-templates select="book"/>
            </body>
        </html>
    </xsl:template>
</xsl:stylesheet>
```

- **Descripción**: XSLT que transforma el XML de libros en una página HTML estructurada.

Plantillas (Templates):

- **Base de las transformaciones XSLT**:

```
xmlCopiar código
<xsl:template match="book">
    <!-- Contenido de la plantilla para 'book' -->
</xsl:template>
```

- **Descripción**: Define cómo transformar los elementos 'book'.

Aplicación de plantillas:

- **Utilización de** `<xsl:apply-templates>`:

```
xmlCopiar código
<xsl:apply-templates select="catalog/book"/>
```

- **Descripción**: Aplica la plantilla definida a los elementos 'book' dentro de 'catalog'.

3. XPath (Navegación y Obtención de Datos)

XPath (XML Path Language):

- **Descripción**: Lenguaje para navegar y obtener datos en documentos XML.
- **Sintaxis**: Similar a rutas de acceso de sistemas de archivos.

Estructura del documento XML como árbol de nodos:

```
xmlCopiar código
<llibreria>
    <llibre>
        <titol>Harry Potter i la pedra filosofal</titol>
```

```
        <autor>J. K. Rowling</autor>
        <any>1997</any>
        <preu>25.00</preu>
    </llibre>
    <llibre>
        <titol>El Senyor dels Anells</titol>
        <autor>J. R. R. Tolkien</autor>
        <any>1954</any>
        <preu>32.00</preu>
    </llibre>
    <llibre>
        <titol>Crònica de una mort anunciada</titol>
        <autor>Gabriel Garcia Márquez</autor>
        <any>1981</any>
        <preu>20.00</preu>
    </llibre>
</llibreria>
```

- **Descripción**: Ejemplo de documento XML estructurado como un árbol de nodos.

Tipos de nodos: Elemento, Atributo, Texto, Espacio de Nombres, Instrucción de Procesamiento, Comentario, Nodo Raíz.

Relación entre nodos: Padre, Hijo, Hermanos, Ancestros, Descendientes.

Selección de Nodos en XPath:

```
xpathCopiar código
//llibre/titol
/llibreria/llibre[1]
/llibreria/llibre[last()]
/llibreria/llibre[preu>25.00]
```

- **Descripción**: Ejemplos de expresiones XPath para seleccionar nodos específicos.

Predicados en XPath:

```
xpathCopiar código
/llibreria/llibre[@any='1997']
/llibreria/llibre[position()=2]
/llibreria/llibre[preu>20.00 and any<2000]
/llibreria/llibre[contains(titol, 'Harry')]
```

- **Descripción**: Ejemplos de predicados para filtrar y seleccionar nodos específicos.

4. Funciones y Operadores en XPath

Funciones de XPath:

- **Ejemplos de uso**:

```
xpathCopiar código
last(): /llibreria/llibre[last()]
position(): /llibreria/llibre[position()>1]
count(): count(/llibreria/llibre)
name(): name(/llibreria/llibre)
concat(): concat(/llibreria/llibre/titol, ' - ', /llibreria/llibre/autor)
contains(): /llibreria/llibre[contains(titol, 'Harry')]
starts-with(): /llibreria/llibre[starts-with(autor, 'J.')]
substring(): substring(/llibreria/llibre/titol, 1, 3)
sum(): sum(/llibreria/llibre/preu)
normalize-space(): normalize-space(/llibreria/llibre/titol)
not(): not(/llibreria/llibre[preu>50.00])
```

- **Descripción**: Ejemplos de funciones XPath aplicadas a nodos XML.

Operadores de XPath:

- **Ejemplos**:

```
xpathCopiar código
//llibre | //cd
6 + 4
6 - 4
6 * 4
8 div 4
preu=9.80
preu!=9.80
preu<9.80
preu<=9.80
preu>9.80
preu>=9.80
preu=9.80 or preu=9.70
preu>9.00 and preu<9.90
5 mod 2
```

- **Descripción**: Ejemplos de operadores utilizados en XPath.

5. Estructuras Condicionales y Bucles en XSLT

Estructuras Condicionales:

- **Ejemplo de** `<xsl:if>`:

```
xmlCopiar código
<xsl:if test="preu > 50">
    <span class="car">Car</span>
</xsl:if>
```

- **Descripción**: Condicional simple para evaluar y mostrar contenido basado en una condición.
- **Ejemplo de** `<xsl:choose>`:

```
xmlCopiar código
<xsl:choose>
    <xsl:when test="preu > 50">
        <span class="car">Car</span>
    </xsl:when>
    <xsl:when test="preu > 25 and preu <= 50">
        <span class="mitjà">Mitjà</span>
    </xsl:when>
    <xsl:otherwise>
        <span class="econòmic">Econòmic</span>
    </xsl:otherwise>
</xsl:choose>
```

- **Descripción**: Condicional múltiple para evaluar varias condiciones y mostrar contenido basado en estas.

Iteración con `<xsl:for-each>`:

- **Ejemplo**:

```
xmlCopiar código
<xsl:for-each select="catalog/book">
    <div class="book">
        <h2><xsl:value-of select="title"/></h2>
        <p>Autor: <xsl:value-of select="author"/></p>
        <p>Preu: $<xsl:value-of select="price"/></p>
    </div>
```

```
</xsl:for-each>
```

- **Descripción**: Permite recorrer múltiples nodos 'book' y aplicar plantillas a cada uno.

6. Variables y Parámetros en XSLT

Uso de Variables:

- **Definición y uso de variables**:

```
xmlCopiar código
<xsl:variable name="color" select="'#f2f2f2'"/>
<div style="background-color:{$color};">
    <!-- contenido -->
</div>
```

- **Descripción**: Definición de una variable y su uso en una plantilla.

Pasaje de Parámetros:

- **Definición de parámetros**:

```
xmlCopiar código
<xsl:param name="descompte"/>
```

- **Descripción**: Definición de un parámetro que puede ser pasado a una plantilla.
- **Uso y paso de parámetros**:

```
xmlCopiar código
<xsl:template match="book">
    <div style="background-color:{$color};">
        <h2><xsl:value-of select="title"/></h2>
        <p>Autor: <xsl:value-of select="author"/></p>
        <p>Preu amb Descompte: $<xsl:value-of select="price - (price * $descompte)"/></p>
    </div>
</xsl:template>
<xsl:apply-templates select="catalog/book">
    <xsl:with-param name="descompte" select="0.1"/>
</xsl:apply-templates>
```

- **Descripción**: Uso de parámetros en una plantilla y paso de valores a través de `<xsl:with-param>`.

7. Ejemplo Completo de XSLT

Transformación completa de XML a HTML:

- **XML**:

```
xmlCopiar código
<catalog>
    <book>
        <title>Harry Potter i la pedra filosofal</title>
        <author>J.K. Rowling</author>
        <price>20.00</price>
    </book>
    <book>
        <title>Crim i càstig</title>
        <author>Fyodor Dostoevsky</author>
        <price>18.50</price>
    </book>
</catalog>
```

- **Descripción**: Archivo XML con información de libros.
- **XSLT**:

```
xmlCopiar código
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:variable name="colorFondo" select="'#f2f2f2'"/>
    <xsl:template match="book">
        <div style="background-color:{$colorFondo};">
            <h2><xsl:value-of select="title"/></h2>
            <p>Autor: <xsl:value-of select="author"/></p>
            <p>Preu: $<xsl:value-of select="price"/></p>
        </div>
    </xsl:template>
    <xsl:template match="catalog">
        <html>
            <head>
                <title>Llibreria Online</title>
            </head>
            <body>
                <h1>Col·lecció de Llibres</h1>
                <xsl:apply-templates select="book"/>
            </body>
        </html>
    </xsl:template>
</xsl:stylesheet>
```

- **Descripción**: XSLT que transforma el XML de libros en una página HTML estructurada, utilizando variables y plantillas.

Conclusión

Este temario proporciona una visión completa de los conceptos clave y técnicas de XSL, XSLT y XPath. Asegúrate de comprender cada sección y practicar con ejemplos reales para prepararte adecuadamente para tu examen. Puedes enviarme más documentos o temas específicos si necesitas más detalles o explicaciones adicionales.

JSON y JSON Schem

1. Introducción a JSON

JSON (JavaScript Object Notation)

- **Descripción**: Es un formato de notación de datos ligero utilizado tanto para guardar información como para el intercambio de datos entre aplicaciones.
- **Características**:
  - No es un lenguaje de marcas, no usa etiquetas como HTML o XML.
  - Los archivos JSON son archivos de texto y usan la extensión .json.
  - Autodescriptivo y fácil de leer y escribir tanto por máquinas como por humanos.
  - Alternativa más simple y ligera a XML con funciones similares.
  - Ampliamente aceptado en la comunicación entre clientes y servidores.
  - La mayoría de los lenguajes de programación tienen soporte integrado para JSON.
  - Versátil para el almacenamiento de datos y la configuración de aplicaciones.

1.1. Sintaxis JSON

- **Estructura basada en pares clave-valor**:

```
jsonCopiar código
{ "key": "value", "key1": "value1", "key2": "value2" }
```

- **Tipos de datos admitidos**:
  - Cadenas de texto (Strings):

    ```
    jsonCopiar código
    { "mensaje": "Hola Mundo" }
    ```

  - Números:

    ```
    jsonCopiar código
    { "entero": 42, "decimal": 3.14 }
    ```

  - Booleanos y Null:

    ```
    jsonCopiar código
    { "activo": true, "inactivo": false, "valorNulo": null }
    ```

  - Ejemplo completo:

    ```
    jsonCopiar código
    {
      "nom": "Aitor Tilla",
      "edat": 26,
      "estudiant": true,
      "ciutat": "La Vall d'Uixó",
      "puntuació": 5.5,
      "comentari": "Aquest és un exemple de JSON amb diverses dades.",
      "nulExemple": null}
    ```

1.2. Estructura de Datos en JSON

- **Objetos**:

  ```
  jsonCopiar código
  { "nom": "Aitor Tilla", "edat": 26, "moto": null }
  ```

  - Ejemplo de objeto JSON que representa un libro:

    ```
    jsonCopiar código
    {
      "títol": "El Gran Gatsby",
      "autor": "F. Scott Fitzgerald",
      "anyPublicació": 1925,
      "editorial": "Scribner",
      "disponible": true,
      "crítiques": [4.5, 5.0, 4.2]
    }
    ```

- **Arrays**:

  ```
  jsonCopiar código
  [ "Ford", "BMW", "Fiat" ]
  ```

  - Ejemplo de array dentro de un objeto:

    ```
    jsonCopiar código
    {
      "tasques": ["Comprar llet", "Anar al gimnàs", "Estudiar per l'examen"]
    ```

```
}
```

1.3. Combinaciones de Objetos y Arrays

- **Objetos simples**:

```
jsonCopiar código
{ "nom": "Jaume", "edat": 18, "ciutat": "La Vall d'Uixó" }
```

- **Arrays simples**:

```
jsonCopiar código
[ "blau", "verd", "groc" ]
```

- **Objetos dentro de objetos**:

```
jsonCopiar código
{
  "persona": { "nom": "Veronica", "edat": 28 },
  "adreça": { "carrer": "Carrer Gran, 123", "ciutat": "Fodeguilla" }
}
```

- **Arrays dentro de arrays**:

```
jsonCopiar código
[ [1, 2, 3], ["a", "b", "c"] ]
```

- **Objetos con arrays**:

```
jsonCopiar código
{ "noms": ["Gerard", "Jordi"], "punts": [8.5, 7.0] }
```

- **Arrays de objetos**:

```
jsonCopiar código
[ { "nom": "Vicent", "edat": 25 }, { "nom": "Carles", "edat": 29 } ]
```

- **Arrays con tipos de datos mixtos**:

```
jsonCopiar código
[42, "groc", true, { "objecte": "aninat" }]
```

- **Objetos con tipos de datos mixtos**:

```
jsonCopiar código
{ "nom": "MariCarmen", "edat": 30, "actiu": true, "aficions": ["lectura", "caminar"] }
```

- **Arrays dentro de objetos y viceversa**:

```
jsonCopiar código
{
  "persones": [
    { "nom": "Andrei", "edat": 35 },
    { "nom": "Esther", "edat": 28 }
  ],
```

```
  "colors": ["verd", "blau"]
}
```

- **Objetos y arrays vacíos**:

```
jsonCopiar código
{ "informacio": {}, "interessos": [], "historial_compres": [] }
```

- **Valores nulos y valores booleanos**:

```
jsonCopiar código
{ "valor1": null, "valor2": true, "valor3": false }
```

- **Arrays con elementos anidados**:o también

```
jsonCopiar código
[ [1, 2, 3], ["a", "b", "c", [true, false]] ]
```

```
jsonCopiar código
[
  "Texto",
  42,
  ["a", "b", "c", [1, 2, 3]],
  { "nombre": "Lorena", "edad": 30 }
]
```

- **Objetos con objetos y arrays con arrays anidados**:

```
jsonCopiar código
{
  "dades": {
    "noms": ["Nereida", "Juan"],
    "punts": [9.0, 8.5],
    "metadades": {
      "creador": "admin",
      "dataCreació": "1812-03-19"
    },
    "valors": [
      [1, 2],
      [3, 4]
    ]
  }
}
```

1.4. Consideraciones y Errores Comunes en JSON

- **Case-Sensitive**:

```
jsonCopiar código
{ "nom": "Fina", "Nom": "Pepe" }
```

- **Formato por un único elemento (Array o Objecto)**:
  Incorrecto:Correcto:

```
jsonCopiar código
[1, 2, 3], ["a", "b", "c"]
```

```
jsonCopiar código
[ [1, 2, 3], ["a", "b", "c"] ]
```

- **El último elemento no puede ir seguido de coma**:
  Incorrecto:Correcto:

```
jsonCopiar código
{ "nom": "Sergi", "edat": 21, }
["blau", "verd", "groc", ]
```

```
jsonCopiar código
{ "nom": "Sergi", "edat": 21 }
["blau", "verd", "groc"]
```

- **No permite claves duplicadas**:
  Incorrecto:

```
jsonCopiar código
{ "color": "vermell", "color": "blau" }
```

- **No admite comentarios**

1.5. Utilización de Caracteres Especiales y de Control

- **Ejemplo**:

```
jsonCopiar código
{
  "missatge": "Aquest és un exemple amb caràcters especials: \"cometes dobles\". També conté línie
s noves:\nLínia 1\nLínia 2",
  "barra_invertida": "Això és una barra invertida escapada: \\",
  "tabulacio": "Això és una tabulació: \tTabulació 1\tTabulació 2",
  "simbol": "Símbol de copyright © (c) representat com Unicode: \\u00A9",
  "caracter_control": "Això és un caràcter de control representat com Unicode: \\u20AC"
}
```

- **Validador de JSON online**: https://jsonlint.com/

1.6. Ejemplo Completo de JSON

```
jsonCopiar código
{
  "estudiant": {
    "nom": "Armando Bronca Segura",
    "edat": 20,
    "matriculat": true,
    "assignatures": ["Llenguatge de Marques", "Accés a Dades", "Programació Web"],
    "notes": {
      "llenguatge_de_marques": {
        "parcial_1": 9.5,
        "parcial_2": 8.7
      },
      "accés_a_dades": {
        "parcial_1": 8.0,
        "parcial_2": 9.2
      },
      "programacio_web": {
```

```
        "parcial_1": 7.8,
        "parcial_2": 8.9
      }
    }
  },
  "professor": {
    "nom": "Jaume Aragó",
    "especialitzacio": "Accés a Dades",
    "contacte": {
      "correu": "j.aragovalls@edu.gva.es",
      "telefon": "+34 555 555 555"
    }
  },
  "institut": "IES Benigasló",
  "curs_academic": "1936-1937",
  "comentaris": [
    {
      "data": "1936-10-15",
      "text": "Armando ha fet bons progressos en el curs fins ara."
    },
    {
      "data": "1936-11-30",
      "text": "És important seguir treballant l'accés a dades."
    }
  ]
}
```

2. Introducción a JSON Schema

JSON Schema

- **Descripción**: Herramienta para validar y describir la estructura de los datos y sus restricciones en JSON.
- **Funcionalidad**:
  - Establece reglas y especificaciones para los datos en un documento JSON.
  - Especifica tipos de datos, propiedades y restricciones.
  - Garantiza la integridad y consistencia de los datos JSON.

2.1. ¿Por qué usar JSON Schema?

- **Validación de Datos**: Asegura que los datos cumplen con las restricciones y estándares definidos.
- **Documentación**: Describe la estructura y restricciones de los datos.
- **Interoperabilidad**: Asegura que los datos sean compatibles con otros sistemas y aplicaciones.

2.2. ¿Cómo funciona?

1. **Creación del JSON Schema**: Documento en formato JSON que describe la estructura y restricciones de los datos.
2. **Validación de Datos**: Validar los datos del documento JSON utilizando el JSON Schema.
3. **Resultados de la Validación**: Los datos se consideran válidos si cumplen con todas las reglas del JSON Schema.
4. **Utilización en Aplicaciones**: Asegurar que los datos sean correctos y cumplan con los estándares definidos.

2.3. Ejemplos de JSON Schema

- **Ejemplo Simple**:

```
jsonCopiar código
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "properties": {
    "id": { "type": "integer" },
    "name": { "type": "string" },
```

```
    "price": { "type": "number" }
  },
  "required": ["id", "name", "price"]
}
```

- **Ejemplo Complejo**:

```
jsonCopiar código
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "Product",
  "description": "A product from Acme’s catalog",
  "type": "object",
  "properties": {
    "id": { "description": "The unique identifier for a product", "type": "integer" },
    "name": { "description": "Name of the product", "type": "string" },
    "price": { "type": "number" }
  },
  "required": ["id", "name", "price"]
}
```

2.4. Sintaxis de JSON Schema

- **Elementos Generales**:
  - **$schema**: Indica la versión de JSON Schema a utilizar.

    ```
    jsonCopiar código
    "$schema": "http://json-schema.org/draft-07/schema#"
    ```

  - **title**: Proporciona un título para identificar el objeto JSON.

    ```
    jsonCopiar código
    "title": "Exemple de Persona"
    ```

  - **description**: Describe lo que representa el esquema JSON.

    ```
    jsonCopiar código
    "description": "Esquema que descriu una estructura per a representar informació de persones."
    ```

  - **type**: Define el tipo de datos esperado.

    ```
    jsonCopiar código
    "type": "string"
    "type": "number"
    "type": "integer"
    "type": "boolean"
    "type": "object"
    "type": "array"
    ```

    - Combinaciones de tipos:

      ```
      jsonCopiar código
      "type": ["string", "number"]
      ```

    - Tipos polimórficos (complejos):

```
jsonCopiar código
{
  "type": "object",
  "properties": {
    "name": { "type": "string" },
    "age": { "type": "integer" }
  }
}
```

2.5. Elementos relacionados con Objetos

- **required**: Enumera las propiedades de un objeto que son obligatorias.

```
jsonCopiar código
"required": ["nom", "edat"]
```

- **properties**: Contiene una lista de propiedades y sus definiciones.

```
jsonCopiar código
{
  "type": "object",
  "properties": {
    "nom": { "type": "string" },
    "edat": { "type": "integer" }
  }
}
```

- **additionalProperties**: Especifica si se permiten propiedades adicionales no definidas en el esquema.

```
jsonCopiar código
"additionalProperties": { "type": "string" }
"additionalProperties": true"additionalProperties": false
```

  - Ejemplo:

```
jsonCopiar código
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "properties": {
    "nom": { "type": "string" },
    "edat": { "type": "integer" }
  },
  "additionalProperties": { "type": ["string", "integer"] }
}
```

2.6. Referencia a un esquema externo y definiciones

- **$ref**: Referencia a un esquema externo.

```
jsonCopiar código
"$ref": "http://example.com/schemas/person-schema"
```

- **definitions**: Define esquemas adicionales para reutilización.

```
jsonCopiar código
{
  "definitions": {
    "NomEsquema1": { /* Definición de esquema 1 */ },
```

```
    "NomEsquema2": { /* Definición de esquema 2 */ }
  }
}
```

- Utilización de definitions dentro del esquema principal:

```
jsonCopiar código
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "properties": {
    "prop1": { "$ref": "#/definitions/NomEsquema1" },
    "prop2": { "$ref": "#/definitions/NomEsquema2" }
  },
  "definitions": {
    "NomEsquema1": { /* Definición de esquema 1 */ },
    "NomEsquema2": { /* Definición de esquema 2 */ }
  }
}
```

2.7. Elementos relacionados con Arrays

- **items**: Contiene la definición de los elementos del array.
  - Valor único para todos los elementos del array:

```
jsonCopiar código
{
  "type": "array",
  "items": { "type": "string" }
}
```

  - Lista de tipos:

```
jsonCopiar código
{
  "type": "array",
  "items": [
    { "type": "string" },
    { "type": "number" },
    { "type": "boolean" }
  ]
}
```

- **minItems y maxItems**: Indican el número mínimo y máximo de elementos en una lista.

```
jsonCopiar código
{
  "type": "array",
  "items": { "type": "string" },
  "minItems": 2,
  "maxItems": 5
}
```

- **additionalItems**: Indica si se aceptan elementos adicionales que no están definidos en el esquema.

```
jsonCopiar código
"additionalItems": true"additionalItems": false
```

  - Ejemplo:

```
jsonCopiar código
{
  "type": "array",
  "items": [
    { "type": "string" },
    { "type": "number" }
  ],
  "additionalItems": { "type": "boolean" }
}
```

2.8. Enums

- **enum**: Restringe los valores de una propiedad a una lista.
  - Ejemplo con strings:

```
jsonCopiar código
{
  "color": {
    "type": "string",
    "enum": ["roig", "blau", "verd"]
  }
}
```

  - Ejemplo con arrays:

```
jsonCopiar código
{
  "noms": {
    "type": "array",
    "enum": [
      ["Juan", "Fina"],
      [1, 2, 3]
    ]
  }
}
```

  - Ejemplo con objetos:

```
jsonCopiar código
{
  "persones": {
    "type": "array",
    "items": {
      "type": "object",
      "properties": {
        "nom": { "type": "string" },
        "edat": { "type": "integer" }
      },
      "required": ["nom", "edat"],
      "additionalProperties": false},
    "enum": [
      [{ "nom": "Jaume", "edat": 55 }, { "nom": "Bel", "edat": 25 }],
      [{ "nom": "Sergi", "edat": 35 }, { "nom": "Saioa", "edat": 29 }]
    ]
  }
}
```

2.9. Ejemplo Completo de JSON Schema

- **Ejemplo**:

```
jsonCopiar código
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "properties": {
    "nom": { "type": "string" },
    "edat": { "type": "integer" },
    "poblacio": { "$ref": "#/definitions/Poblacio" },
    "punts": { "$ref": "#/definitions/Punts" }
  },
  "definitions": {
    "Poblacio": { "type": "string" },
    "Punts": { "type": "integer" }
  },
  "additionalProperties": { "type": ["string", "integer"] }
}
```

2.10. Uso de Expresiones Regulares

- **Pattern**: Campo en JSON Schema para aplicar una expresión regular a una cadena de texto.

```
jsonCopiar código
{
  "type": "string",
  "pattern": "^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}$"
}
```

  - Ejemplo: Validación de correo electrónico.

```
jsonCopiar código
{
  "type": "string",
  "pattern": "^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}$"
}
```

  - Ejemplo: Validación de números de teléfono.

```
jsonCopiar código
{
  "type": "string",
  "pattern": "^\\+?[1-9]\\d{1,14}$"
}
```

  - Ejemplo: Validación de nombre de usuario.

```
jsonCopiar código
{
  "type": "string",
  "pattern": "^[a-zA-Z0-9_-]{3,16}$"
}
```

  - Ejemplo: Validación de URL.

```
jsonCopiar código
{
  "type": "string",
  "pattern": "^(https?|ftp):\\/\\/[\\w\\d\\-]+(\\.[\\w\\d\\-]+)+([\\w\\d\\-.,@?^=%&:/~+#]*[\\w\\d
```

```
\\-@?^=%&/~+#])?$"
}
```

- Ejemplo: Validación de fecha (formato AAAA-MM-DD).

```
jsonCopiar código
{
  "type": "string",
  "pattern": "^\\d{4}-\\d{2}-\\d{2}$"
}
```

2.11. Ejercicios Resueltos

- **Ejercicio 1: Creación de un esquema sencillo**:

```
jsonCopiar código
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "properties": {
    "persona": { "$ref": "#/definitions/Persona" }
  },
  "definitions": {
    "Persona": {
      "type": "object",
      "properties": {
        "nom": { "type": "string" },
        "edat": { "type": "integer" }
      },
      "required": ["nom", "edat"]
    }
  }
}
```

- **Ejercicio 2: Restricción de propiedades adicionales**:

```
jsonCopiar código
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "properties": {
    "persona": { "$ref": "#/definitions/Persona" },
    "dadesAddicionals": {
      "type": "object",
      "additionalProperties": { "type": "boolean" }
    }
  },
  "definitions": {
    "Persona": {
      "type": "object",
      "properties": {
        "nom": { "type": "string" },
        "edat": { "type": "integer" }
      },
      "required": ["nom", "edat"]
    }
  }
}
```

- **Ejercicio 3: Restricción de valores para una propiedad**:

```
jsonCopiar código
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "properties": {
    "colors": {
      "type": "array",
      "items": {
        "type": "string",
        "enum": ["verd", "blau", "groc"]
      }
    }
  },
  "required": ["colors"]
}
```

- **Ejercicio 4: Arrays con restricciones**:

```
jsonCopiar código
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "properties": {
    "numeros": {
      "type": "array",
      "items": { "type": "number" },
      "minItems": 2
    }
  },
  "required": ["numeros"]
}
```

- **Ejercicio 5: Referencia a un esquema externo**:

```
jsonCopiar código
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "$ref": "#/definitions/Persona",
  "definitions": {
    "Persona": {
      "type": "object",
      "properties": {
        "nom": { "type": "string" },
        "edat": { "type": "integer" }
      },
      "required": ["nom", "edat"]
    }
  }
}
```

JavaScript

1. Introducción a JavaScript

JavaScript

- **Descripción**: Lenguaje de programación interpretado y orientado a objetos para páginas web interactivas.

- **Características**:
  - Añadir funcionalidades dinámicas.
  - Validaciones de formularios.
  - Animaciones.
  - Interacción con el DOM.

Inicialización con `<script>`

- **Incorporación de scripts**:

```
htmlCopiar código
<script>
    // Código JavaScript
</script>
```

- **Archivo externo**:

```
htmlCopiar código
<script src="script.js"></script>
```

Ejemplo básico

- **Evento y función**:

```
htmlCopiar código
<script>
    function saludar() {
        alert('Hola, aquest és un script en JavaScript i HTML!');
    }
    window.onload = saludar;
</script>
```

---

2. Salida de Datos

Funciones de Salida

- **alert**:

```
javascriptCopiar código
alert("Açò és un missatge d'alerta!");
```

- **console.log**:

```
javascriptCopiar código
console.log("Aquest missatge apareix a la consola.");
```

- **document.write**:

```
javascriptCopiar código
document.write("Aquest missatge apareix al document HTML.");
```

- **innerHTML**:

```
javascriptCopiar código
document.getElementById("resultat").innerHTML = "Nou contingut dinàmic.";
```

Confirmación y Entrada de Datos

- **confirm**:

```
javascriptCopiar código
let confirmacio = confirm("Estàs segur que vols continuar?");
```

- **prompt**:

```
javascriptCopiar código
let resposta = prompt("Com et dius?");
```

---

3. Variables

Declaración de Variables

- **var, let, const**:

```
javascriptCopiar código
let edat = 30;
const ciutat = "Barcelona";
```

Ejemplos

- **Reasignación**:

```
javascriptCopiar código
edat = 35;
```

---

4. Tipos Primitivos

Tipos de Datos

- **Number**:

```
javascriptCopiar código
let numero1 = 10;
```

- **String**:

```
javascriptCopiar código
let frase = "Hola, món!";
```

- **Boolean**:

```
javascriptCopiar código
let esCert = true;
```

- **Undefined**:

```
javascriptCopiar código
let x; // x es undefined
```

- **Null**:

```
javascriptCopiar código
let y = null;
```

- **BigInt**:

```
javascriptCopiar código
let numeroGran1 = 9007199254740991n;
```

- **Symbol**:

```
javascriptCopiar código
let simbol = Symbol('Descripció del símbol');
```

---

5. Estructuras de Control

Condicionales

- **if, else if, else**:

```
javascriptCopiar código
let hora = 12;
if (hora < 12) {
    console.log("Bon dia!");
}
```

Bucles

- **for**:

```
javascriptCopiar código
for (let i = 0; i < 5; i++) {
    console.log(i);
}
```

- **while**:

```
javascriptCopiar código
let j = 0;
while (j < 5) {
    console.log(j);
    j++;
}
```

- **do-while**:

```
javascriptCopiar código
let k = 0;
do {
    console.log(k);
    k++;
} while (k < 5);
```

Switch

- **switch**:

```
javascriptCopiar código
let dia = 3;
switch (dia) {
    case 1:
        console.log("Dilluns");
        break;
    case 2:
        console.log("Dimarts");
        break;
    case 3:
        console.log("Dimecres");
        break;
    default:
        console.log("Altres dies");
}
```

6. Arrays

Declaración y Acceso a Elementos

- **Declaración**:

```
javascriptCopiar código
let colors = ["blau", "verd", "roig"];
```

- **Acceso**:

```
javascriptCopiar código
console.log(colors[0]); // Output: blau
```

Modificación

- **Modificar**:

```
javascriptCopiar código
colors[1] = "groc";
```

- **Longitud**:

```
javascriptCopiar código
console.log(colors.length); // Output: 3
```

Añadir y Eliminar Elementos

- **Añadir**:

```
javascriptCopiar código
colors.push("taronja");
```

- **Eliminar**:

```
javascriptCopiar código
colors.pop();
```

7. Iteración de Elementos de Arrays

Bucle For

- **Ejemplo**:

```
javascriptCopiar código
for (let i = 0; i < colors.length; i++) {
    console.log(colors[i]);
}
```

forEach

- **Ejemplo**:

```
javascriptCopiar código
colors.forEach(function(color) {
    console.log(color);
});
```

Métodos de Arrays

- **indexOf**:

```
javascriptCopiar código
console.log(colors.indexOf("groc")); // Output: 1
```

- **join**:

```
javascriptCopiar código
console.log(colors.join(", ")); // Output: blau, groc, roig
```

- **slice**:

```
javascriptCopiar código
let subarray = colors.slice(1, 2);
console.log(subarray); // Output: ["groc"]
```

- **splice**:

```
javascriptCopiar código
const array = ['a', 'b', 'c'];
array.splice(1, 2);
```

---

8. Objetos

Creación de Objetos

- **Object Literal**:

```
javascriptCopiar código
let persona = {
    nom: "Anna",
    edat: 30,
    ciutat: "Barcelona"
};
```

- **Función Constructora**:

```
javascriptCopiar código
function Persona(nom, edat, ciutat) {
    this.nom = nom;
    this.edat = edat;
    this.ciutat = ciutat;
}
let persona = new Persona("Anna", 30, "Barcelona");
```

Acceso y Modificación de Objetos

- **Acceso**:

```
javascriptCopiar código
console.log(persona.nom);    // Output: Anna
```

- **Modificación**:

```
javascriptCopiar código
persona.edat = 35;
```

- **Añadir**:

```
javascriptCopiar código
persona.professio = "Enginyer";
```

- **Eliminar**:

```
javascriptCopiar código
delete persona.ciutat;
```

Iteración y Operaciones

- **Bucle For-In**:

```
javascriptCopiar código
const cotxe = {
    marca: "Toyota",
    model: "Corolla",
    any: 2020
};
for (let propietat in cotxe) {
    console.log(propietat + ": " + cotxe[propietat]);
}
```

- **Comprobar Propiedad**:

```
javascriptCopiar código
console.log("nom" in persona);    // Output: true
```

- **Objetos Anidados**:

```
javascriptCopiar código
let cotxe = {
    marca: "Toyota",
    model: "Corolla",
    propietari: {
```

```
        nom: "Marc",
        edat: 40
    }
};
```

- **Clonar Objeto**:

```
javascriptCopiar código
let cotxe2 = Object.assign({}, cotxe);
```

- **Fusionar Objetos**:

```
javascriptCopiar código
let cotxe3 = { color: "blau" };
let cotxeCombinat = Object.assign({}, cotxe, cotxe3);
```

- **Convertir a Array**:

```
javascriptCopiar código
let arrayPersones = Object.values(persona);
```

Métodos Útiles de Objetos

- **Object.keys(obj)**: Retorna un array de las claves enumerables.
- **Object.values(obj)**: Retorna un array de los valores.
- **Object.entries(obj)**: Retorna un array de pares [clave, valor].
- **Object.assign(target, source1, ...)**: Copia valores al objeto destino.
- **Object.freeze(obj)**: Congela un objeto.
- **Object.seal(obj)**: Sella un objeto.
- **Object.getOwnPropertyNames(obj)**: Retorna un array de todas las propiedades.
- **Object.hasOwnProperty(prop)**: Retorna si una propiedad existe.
- **Object.create(proto[, propertiesObject])**: Crea un nuevo objeto con un prototipo.
- **Object.getPrototypeOf(obj)**: Retorna el prototipo de un objeto.
- **Object.setPrototypeOf(obj, proto)**: Establece el prototipo de un objeto.
- **Object.is(obj1, obj2)**: Compara si dos valores son iguales.

---

9. Funciones

Definición de Funciones

- **Declaración**:

```
javascriptCopiar código
function saludar(nom) {
    console.log("Hola, " + nom + "!");
}
```

- **Expresión**:

```
javascriptCopiar código
let saludar = function(nom) {
    console.log("Hola, " + nom + "!");
};
```

Llamada a Funciones

- **Llamada**:

```
javascriptCopiar código
saludar("Anna"); // Output: Hola, Anna!
```

- **Retorno**:

```
javascriptCopiar código
function suma(a, b) {
    return a + b;
}
let resultat = suma(5, 3);
```

Funciones Anónimas

- **Asignación a Variable**:

```
javascriptCopiar código
let saludar = function(nom) {
    console.log("Hola, " + nom + "!");
};
```

Funciones de Flecha

- **Sintaxis Básica**:

```
javascriptCopiar código
const nomFuncio = (parametres) => {
    // Código de la función
};
```

- **Ejemplo**:

```
javascriptCopiar código
let suma = (a, b) => a + b;
console.log(suma(5, 3)); // Output: 8
```

DOM (Document Object Model)

1. Introducción a DOM

Document Object Model (DOM)

- **Descripción**:
  - Estructura del documento HTML.
  - API de W3C que permite a los programas y scripts acceder y actualizar contenido, estilo y estructura de un documento HTML o XML.

1.1. Acceso al DOM

- **Acceso al objeto document**:

```
javascriptCopiar código
// Accede al objeto principal del DOM que representa el documento
const doc = document;
```

```
// Obtiene el elemento con el id 'titol'
const titol = doc.getElementById('titol');
```

---

2. Conceptos Principales del DOM

2.1. Document

- **Descripción**:
  - Objeto principal del DOM que representa todo el documento HTML o XML.
  - Punto de entrada principal para acceder a otros elementos del DOM.
  - Ejemplo:

```
javascriptCopiar código
// Accede al objeto principal del DOM
const doc = document;

// Obtiene el elemento con el id 'titol'
const titol = doc.getElementById('titol');
```

2.2. Node

- **Descripción**:
  - Cada elemento, atributo o texto del documento es considerado un nodo.
  - Ejemplo:

```
javascriptCopiar código
// Crea un nodo de texto con el contenido 'Este és un text'
const textNode = document.createTextNode('Este és un text');

// Crea un nuevo elemento div
const element = document.createElement('div');

// Añade el nodo de texto como hijo del elemento div
element.appendChild(textNode);
```

2.3. Element

- **Descripción**:
  - Tipo especial de nodo que representa etiquetas HTML como `<div>`, `<p>`, `<span>`, etc.
  - Ejemplo:

```
javascriptCopiar código
// Crea un nuevo elemento div
const divElement = document.createElement('div');

// Añade un atributo 'id' al elemento div
divElement.setAttribute('id', 'nouDiv');

// Asigna contenido de texto al elemento div
divElement.textContent = 'Aquest és un nou div';

// Añade el elemento div al cuerpo del documento
document.body.appendChild(divElement);
```

2.4. Atributo

- **Descripción**:

- Característica específica de un elemento HTML, como id o class.
- Ejemplo:

```
javascriptCopiar código
// Obtiene el elemento con el id 'meuElement'
const element = document.getElementById('meuElement');

// Cambia el atributo 'alt' del elemento a 'Nova descripció'
element.setAttribute('alt', 'Nova descripció');
```

2.5. Nivell d'Element

- **Descripción**:
  - Relación jerárquica entre elementos del documento.
  - Ejemplo:

```
javascriptCopiar código
// Obtiene el elemento hijo con el id 'fill'
const fill = document.getElementById('fill');

// Obtiene el padre del elemento hijo
const pare = fill.parentNode;

// Muestra el nombre de la etiqueta del padre en la consola
console.log('El pare de l\'element fill és:', pare.tagName);
```

---

3. Selección de Elementos

3.1. Métodos de Selección

- **getElementById**:

```
javascriptCopiar código
// Selecciona el elemento con el id 'idElement'
const elementId = document.getElementById('idElement');
```

- **getElementsByClassName**:

```
javascriptCopiar código
// Selecciona todos los elementos con la clase 'classeElement'
const elementsClasse = document.getElementsByClassName('classeElement');
```

- **getElementsByTagName**:

```
javascriptCopiar código
// Selecciona todos los elementos con la etiqueta 'p'
const elementsEtiqueta = document.getElementsByTagName('p');
```

- **querySelector**:

```
javascriptCopiar código
// Selecciona el primer elemento que coincide con el selector CSS '.selectorClasse'
const elementSelector = document.querySelector('.selectorClasse');
```

- **querySelectorAll**:

```
javascriptCopiar código
// Selecciona todos los elementos que coinciden con el selector CSS 'p'
const elementsSelector = document.querySelectorAll('p');
```

4. Manipulación del DOM

4.1. Crear y Eliminar Elementos

- **Crear Elemento**:

```
javascriptCopiar código
// Crea un nuevo elemento div
const nouElement = document.createElement('div');

// Añade el nuevo elemento como hijo de otro elemento
pare.appendChild(nouElement);
```

- **Eliminar Elemento**:

```
javascriptCopiar código
// Elimina el elemento hijo del elemento padre
pare.removeChild(fill);
```

5. Esdeveniments del DOM

5.1. Gestionar Esdeveniments

- **Añadir un Evento**:

```
javascriptCopiar código
// Obtiene el elemento con el id 'meuElement'
const meuElement = document.getElementById('meuElement');

// Añade un evento de clic al elemento
meuElement.addEventListener('click', function() {
    // Muestra una alerta cuando el elemento es clicado
    alert('S\'ha clicat l\'element!');
});
```

6. Ejercicios

6.1. Ejercicio 1

- **Cambiar color de fondo del título a verde oscuro**:

```
javascriptCopiar código
// Obtiene el elemento con el id 'titol'
const titol = document.getElementById('titol');

// Cambia el color de fondo del elemento a verde oscuro
titol.style.backgroundColor = '#006400';
```

6.2. Ejercicio 2

- **Añadir un nuevo elemento al menú**:

```
javascriptCopiar código
// Crea un nuevo elemento 'li'
const nouElement = document.createElement('li');

// Crea un nuevo enlace 'a'
const enllaç = document.createElement('a');

// Configura el atributo href del enlace
enllaç.href = "#";

// Establece el texto del enlace
enllaç.textContent = "Sobre nosaltres";

// Añade el enlace como hijo del elemento 'li'
nouElement.appendChild(enllaç);

// Añade el nuevo elemento 'li' al menú
document.getElementById('menu').appendChild(nouElement);
```

6.3. Ejercicio 3

- **Cambiar color de fondo de todos los artículos a amarillo claro**:

```
javascriptCopiar código
// Selecciona todos los elementos 'article'
const articles = document.querySelectorAll('article');

// Itera a través de cada elemento 'article'
articles.forEach(article => {
    // Cambia el color de fondo de cada artículo a amarillo claro
    article.style.backgroundColor = '#FFFF99';
});
```

6.4. Ejercicio 4

- **Cambiar color del texto del menú a rojo**:

```
javascriptCopiar código
// Selecciona todos los enlaces 'a' dentro del menú
const links = document.querySelectorAll('#menu a');

// Itera a través de cada enlace
links.forEach(link => {
    // Cambia el color del texto de cada enlace a rojo
    link.style.color = '#ee0808';
});
```

6.5. Ejercicio 5

- **Crear un nuevo párrafo y añadirlo al final del main**:

```
javascriptCopiar código
// Crea un nuevo elemento 'p'
const nouParagraf = document.createElement('p');

// Establece el texto del nuevo párrafo
nouParagraf.textContent = "Segueix-nos a les xarxes socials!";

// Añade el nuevo párrafo al final del elemento 'main'
```

```
document.querySelector('main').appendChild(nouParagraf);
```

6.6. Ejercicio 6

- **Eliminar el enlace "Serveis" del menú**:

```
javascriptCopiar código
// Selecciona el enlace con el atributo href igual a "#" que contiene el texto "Serveis"
const enllaçServeis = document.querySelector('#menu a[href="#"]');

// Elimina el enlace del menú
enllaçServeis.parentNode.removeChild(enllaçServeis);
```

6.7. Ejercicio 7

- **Cambiar color de fondo del footer a gris oscuro**:

```
javascriptCopiar código
// Selecciona el elemento 'footer'
const footer = document.querySelector('footer');

// Cambia el color de fondo del footer a gris oscuro
footer.style.backgroundColor = '#9b9797';
```

6.8. Ejercicio 8

- **Cambiar tamaño de fuente de los párrafos del artículo 1 a 36 píxeles**:

```
javascriptCopiar código
// Selecciona todos los párrafos dentro del primer artículo
const paragrafs = document.querySelectorAll('article:nth-child(1) p');

// Itera a través de cada párrafo
paragrafs.forEach(p => {
    // Cambia el tamaño de la fuente de cada párrafo a 36 píxeles
    p.style.fontSize = '36px';
});
```

6.9. Ejercicio 9

- **Cambiar color de fondo del título y del menú al hacer clic**:

```
javascriptCopiar código
// Selecciona el elemento con el id 'titol'
const titol = document.getElementById('titol');

// Selecciona el elemento con el id 'menu'
const menu = document.getElementById('menu');

// Añade un evento de clic al título
titol.addEventListener('click', function() {
    // Cambia el color de fondo del título a azul
    titol.style.backgroundColor = '#336699';
});

// Añade un evento de clic al menú
menu.addEventListener('click', function() {
    // Cambia el color de fondo del menú a rojo
    menu.style.backgroundColor = '#FF0000';
});
```

Esdeveniments (Eventos)

1. Introducción a los Eventos en el DOM

Eventos en el DOM

- **Descripción**:
  - Interacciones en la página web que se pueden capturar y gestionar mediante código JavaScript.
  - Permiten crear interactividad y responder a acciones de los usuarios como clics, teclas presionadas, movimientos del ratón, etc.
  - Usando eventos, se pueden crear interacciones ricas y dinámicas en las páginas web.

---

2. Tipos de Eventos Comunes

2.1. Evento `click`

- **Descripción**:
  - Se dispara cuando el usuario hace clic sobre un elemento con el botón izquierdo del ratón.
- **Ejemplo**: Capturar un clic en un botón para validar un formulario.

```
javascriptCopiar código
// Obtiene el elemento botón con el id 'myButton'
const btn = document.getElementById('myButton');

// Añade un evento de clic al botón
btn.addEventListener('click', function() {
    // Muestra una alerta cuando el botón es clicado
    alert('Has clicat el botó!');
});
```

2.2. Evento `dblclick`

- **Descripción**:
  - Se dispara cuando el usuario hace doble clic sobre un elemento con el botón izquierdo del ratón.
- **Ejemplo**: Capturar un doble clic para abrir una ventana de detalles.

```
javascriptCopiar código
// Obtiene el elemento con el id 'myElement'
const element = document.getElementById('myElement');

// Añade un evento de doble clic al elemento
element.addEventListener('dblclick', function() {
    // Muestra una alerta cuando el elemento es doble clicado
    alert('Has fet doble clic sobre l\'element!');
});
```

2.3. Evento `mouseenter`

- **Descripción**:
  - Se dispara cuando el cursor del ratón entra a la zona de un elemento.
- **Ejemplo**: Cambiar el estilo de un elemento cuando el ratón entra.

```
javascriptCopiar código
// Obtiene el elemento con el id 'myElement'
const element = document.getElementById('myElement');
```

```
// Añade un evento de entrada del ratón al elemento
element.addEventListener('mouseenter', function() {
    // Cambia el color de fondo del elemento a azul claro cuando el ratón entra
    element.style.backgroundColor = 'lightblue';
});
```

2.4. Evento `mouseleave`

- **Descripción**:
  - Se dispara cuando el cursor del ratón sale de la zona de un elemento.
- **Ejemplo**: Restablecer el estilo de un elemento cuando el ratón sale.

```
javascriptCopiar código
// Obtiene el elemento con el id 'myElement'
const element = document.getElementById('myElement');

// Añade un evento de salida del ratón al elemento
element.addEventListener('mouseleave', function() {
    // Restablece el color de fondo del elemento cuando el ratón sale
    element.style.backgroundColor = '';
});
```

---

3. Otros Eventos Útiles

3.1. Evento `keydown`

- **Descripción**:
  - Se dispara cuando una tecla del teclado es presionada.
- **Ejemplo**:

```
javascriptCopiar código
// Añade un evento de presión de tecla al documento
document.addEventListener('keydown', function(event) {
    // Muestra el código de la tecla presionada en la consola
    console.log('Tecla presionada:', event.key);
});
```

3.2. Evento `keyup`

- **Descripción**:
  - Se dispara cuando una tecla del teclado es liberada.
- **Ejemplo**:

```
javascriptCopiar código
// Añade un evento de liberación de tecla al documento
document.addEventListener('keyup', function(event) {
    // Muestra el código de la tecla liberada en la consola
    console.log('Tecla liberada:', event.key);
});
```

3.3. Evento `submit`

- **Descripción**:
  - Se dispara cuando se envía un formulario.
- **Ejemplo**:

```
javascriptCopiar código
// Obtiene el formulario con el id 'myForm'
const form = document.getElementById('myForm');

// Añade un evento de envío al formulario
form.addEventListener('submit', function(event) {
    // Previene el comportamiento predeterminado de envío del formulario
    event.preventDefault();
    // Muestra un mensaje en la consola
    console.log('Formulario enviado!');
});
```

3.4. Evento `focus`

- **Descripción**:
  - Se dispara cuando un elemento recibe el foco.
- **Ejemplo**:

```
javascriptCopiar código
// Obtiene el elemento de entrada con el id 'myInput'
const input = document.getElementById('myInput');

// Añade un evento de foco al elemento de entrada
input.addEventListener('focus', function() {
    // Cambia el color de fondo del elemento cuando recibe el foco
    input.style.backgroundColor = 'yellow';
});
```

3.5. Evento `blur`

- **Descripción**:
  - Se dispara cuando un elemento pierde el foco.
- **Ejemplo**:

```
javascriptCopiar código
// Obtiene el elemento de entrada con el id 'myInput'
const input = document.getElementById('myInput');

// Añade un evento de pérdida de foco al elemento de entrada
input.addEventListener('blur', function() {
    // Restablece el color de fondo del elemento cuando pierde el foco
    input.style.backgroundColor = '';
});
```

3.6. Evento `scroll`

- **Descripción**:
  - Se dispara cuando se hace scroll en un elemento.
- **Ejemplo**:

```
javascriptCopiar código
// Añade un evento de scroll al documento
window.addEventListener('scroll', function() {
    // Muestra un mensaje en la consola cuando se hace scroll
    console.log('Se está haciendo scroll!');
});
```

3.7. Evento `load`

- **Descripción**:
  - Se dispara cuando una página web o un recurso ha terminado de cargarse completamente.
- **Ejemplo**:

```
javascriptCopiar código
// Añade un evento de carga a la ventana
window.addEventListener('load', function() {
    // Muestra un mensaje en la consola cuando la página ha cargado completamente
    console.log('Página cargada completamente!');
});
```

3.8. Evento `resize`

- **Descripción**:
  - Se dispara cuando se redimensiona la ventana del navegador.
- **Ejemplo**:

```
javascriptCopiar código
// Añade un evento de redimensionamiento a la ventana
window.addEventListener('resize', function() {
    // Muestra las nuevas dimensiones de la ventana en la consola
    console.log('Tamaño de la ventana cambiado:', window.innerWidth, 'x', window.innerHei
ght);
});
```

JavaScript y JSON

Ejemplo 1: Crear un objeto JSON simple y acceder a sus propiedades

Código

```
javascriptCopiar código
// Paso 1: Definir un objeto JSON
var persona = {
  nom: "Juan",
  edat: 30,
  ciutat: "Nova York"
};

// Paso 2: Acceder a las propiedades del objeto JSON
console.log(persona.nom);   // Imprime 'Juan'
console.log(persona.edat);  // Imprime '30'
console.log(persona.ciutat);// Imprime 'Nova York'
```

Comentarios

- **Definición**: Se crea un objeto JSON sencillo con tres propiedades: `nom`, `edat` y `ciutat`.
- **Acceso**: Se utiliza la notación de puntos (`persona.nom`) para acceder a las propiedades del objeto y mostrar sus valores utilizando `console.log()`.

Ejemplo 2: Convertir un objeto JavaScript en JSON y viceversa

Código

```
javascriptCopiar código
// Paso 1: Definir un objeto JavaScript
var persona = {
  nom: "Juan",
  edat: 30,
  ciutat: "Nova York"
};

// Paso 2: Convertir el objeto JavaScript a JSON
var jsonString = JSON.stringify(persona);
console.log(jsonString); // Imprime '{"nom":"Juan","edat":30,"ciutat":"Nova York"}'

// Paso 3: Convertir el JSON de vuelta a un objeto JavaScript
var objecte = JSON.parse(jsonString);
console.log(objecte); // Imprime { nom: "Juan", edat: 30, ciutat: "Nova York" }
```

Comentarios

- **JSON.stringify()**: Convierte el objeto JavaScript `persona` a una cadena JSON.
- **JSON.parse()**: Convierte la cadena JSON `jsonString` de vuelta a un objeto JavaScript.

Ejemplo 3: Iterar sobre un array de objetos JSON

Código

```
javascriptCopiar código
// Paso 1: Definir un array de objetos JSON
var persones = [
  { nom: "Juan", edat: 30 },
  { nom: "María", edat: 25 },
  { nom: "Pedro", edat: 35 }
];

// Paso 2: Iterar sobre el array y acceder a las propiedades de cada objeto
persones.forEach(function(persona) {
  console.log(persona.nom + " té " + persona.edat + " anys.");
});
```

Comentarios

- **Array de objetos**: Se define un array `persones` que contiene varios objetos JSON.
- **Iteración**: Se usa el método `forEach` para recorrer cada objeto en el array y acceder a sus propiedades.

Ejemplo 4: Filtrar datos de un array de objetos JSON

Código

```
javascriptCopiar código
// Paso 1: Definir un array de objetos JSON
var persones = [
  { nom: "Juan", edat: 30 },
  { nom: "María", edat: 25 },
  { nom: "Pedro", edat: 35 }
];

// Paso 2: Filtrar personas mayores de 30 años
var personesMajors = persones.filter(function(persona) {
  return persona.edat > 30;
});
```

```
console.log(personesMajors); // Imprime [{ nom: "Pedro", edat: 35 }]
```

Comentarios

- **Filtrado**: Se utiliza el método `filter` para obtener un nuevo array `personesMajors` con personas cuya `edat` es mayor a 30.

Ejemplo 5: Modificar datos dentro de un objeto JSON existente

Código

```
javascriptCopiar código
// Paso 1: Definir un objeto JSON
var persona = {
  nom: "Juan",
  edat: 30,
  ciutat: "Nova York"
};

// Paso 2: Modificar la propiedad "edat"
persona.edat = 31;

console.log(persona); // Imprime { nom: "Juan", edat: 31, ciutat: "Nova York" }
```

Comentarios

- **Modificación**: Se accede directamente a la propiedad `edat` del objeto `persona` y se asigna un nuevo valor.

Ejemplo 6: Añadir y eliminar propiedades de un objeto JSON

Código

```
javascriptCopiar código
// Paso 1: Definir un objeto JSON
var persona = {
  nom: "Juan",
  edat: 30,
  ciutat: "Nova York"
};

// Paso 2: Añadir una nueva propiedad al objeto
persona.professio = "Enginyer";

// Paso 3: Eliminar una propiedad del objeto
delete persona.ciutat;

console.log(persona); // Imprime { nom: "Juan", edat: 30, professio: "Enginyer" }
```

Comentarios

- **Añadir propiedad**: Se agrega la propiedad `professio` al objeto `persona`.
- **Eliminar propiedad**: Se elimina la propiedad `ciutat` del objeto `persona` usando `delete`.

Ejemplo 7: Encadenar objetos JSON

Código

```
javascriptCopiar código
// Paso 1: Definir un objeto JSON con objetos encadenados
var persona = {
```

```
  nom: "Juan",
  edat: 30,
  direccio: {
    carrer: "123 Carrer Principal",
    ciutat: "Nova York",
    codi_postal: "10001"
  }
};

// Paso 2: Acceder a propiedades de objetos encadenados
console.log(persona.direccio.carrer); // Imprime '123 Carrer Principal'
```

Comentarios

- **Encadenamiento**: Se define un objeto `persona` que contiene otro objeto `direccio` con sus propias propiedades.
- **Acceso**: Se accede a propiedades del objeto encadenado usando notación de puntos.

Ejemplo 8: Validar y manipular datos JSON provenientes de una API

Código

```
javascriptCopiar código
// Paso 1: Simular datos JSON de una API
var dadesJSON = '{"nom": "Juan", "edat": 30, "professio": null}';

// Paso 2: Validar y manipular los datos
try {
  var dades = JSON.parse(dadesJSON);

  // Verificar si la propiedad "professio" está definida
  if (dades.professio !== null) {
    console.log("Professió:", dades.professio);
  } else {
    console.log("La professió no està definida.");
  }
} catch (error) {
  console.error("Error en analitzar les dades JSON:", error);
}
```

Comentarios

- **Simulación**: Se simulan datos JSON que podrían provenir de una API.
- **Validación**: Se usa `JSON.parse()` para convertir la cadena JSON en un objeto y se verifica si la propiedad `professio` está definida.
- **Manejo de errores**: Se utiliza `try...catch` para manejar posibles errores durante el análisis.

Ejemplo 9: Ordenar un array de objetos JSON

Código

```
javascriptCopiar código
// Paso 1: Definir un array de objetos JSON
var persones = [
  { nom: "Juan", edat: 30 },
  { nom: "María", edat: 25 },
  { nom: "Pedro", edat: 35 }
];

// Paso 2: Ordenar el array por edad de forma ascendente
persones.sort(function(a, b) {
```

```
  return a.edat - b.edat;
});

console.log(persones); // Imprime [{ nom: "María", edat: 25 }, { nom: "Juan", edat: 30 }, { nom: "Pedro", edat: 35 }]
```

Comentarios

- **Ordenación**: Se usa el método `sort` para ordenar el array `persones` según la propiedad `edat` en orden ascendente.
- **Función de comparación**: Se define una función de comparación que retorna la diferencia entre las edades de dos objetos.

Ejemplo 10: Combinar múltiples objetos JSON en un solo objeto

Código

```
javascriptCopiar código
// Paso 1: Definir varios objetos JSON
var dades1 = { a: 1, b: 2 };
var dades2 = { c: 3, d: 4 };
var dades3 = { e: 5, f: 6 };

// Paso 2: Combinar los objetos en un solo objeto
var dadesCombinades = Object.assign({}, dades1, dades2, dades3);

console.log(dadesCombinades); // Imprime { a: 1, b: 2, c: 3, d: 4, e: 5, f: 6 }
```

Comentarios

- **Combinación**: Se usa `Object.assign()` para combinar múltiples objetos (`dades1`, `dades2`, `dades3`) en un solo objeto `dadesCombinades`.
- **Objeto destino**: El primer parámetro `{}` especifica un objeto vacío como el objeto destino donde se combinarán las propiedades de los otros objetos.

jQuery

1. Introducción a jQuery – DOM – AJAX

jQuery

- **Descripción**: jQuery es una biblioteca de JavaScript que simplifica la manipulación del DOM y la gestión de eventos en páginas web.
- **Características**:
  - Sintaxis sencilla y funciones optimizadas.
  - Selección y manipulación de elementos del DOM.
  - Manejo de eventos.
  - Efectos y animaciones.
  - Compatibilidad con AJAX para llamadas asíncronas al servidor.

---

2. Inicialización de jQuery

Inicializar jQuery

- **Incluir jQuery en una página web usando un CDN**:

```
htmlCopiar código
<script src="https://code.jquery.com/jquery-3.7.1.min.js"></script>
```

- **Esperar a que la página esté cargada**:

```
javascriptCopiar código
// La función dentro de $(document).ready() se ejecuta cuando el documento HTML está comp
letamente cargado.
$(document).ready(function() {
    // Aquí va el código jQuery
});
```

3. Sintaxis de jQuery

Sintaxis Básica

- **Ejemplos**:

```
javascriptCopiar código
// Oculta todos los elementos <p>
$("p").hide();

// Muestra todos los elementos con class="test"
$(".test").show();

// Muestra lentamente el elemento con id="test"
$("#test").fadeIn();

// Alterna entre hide y show para todos los elementos <button>
$(":button").toggle();
```

Eventos en jQuery

- **Ejemplos de manejo de eventos**:

```
javascriptCopiar código
// Muestra una alerta cuando el ratón entra en cualquier <p>
$(document).on("mouseenter", "p", function(event) {
    alert("Mouse Enter!");
});

// Previene el comportamiento por defecto y muestra una alerta al hacer clic en enlaces c
on clase "actiu"
$(document).on("click", "a.actiu", function(event) {
    event.preventDefault();
    alert("Clic!");
});
```

4. Obtención y Modificación de Datos

Métodos de jQuery

- **Ejemplos**:

```
javascriptCopiar código
// HTML de referencia
<!DOCTYPE html>
<html lang="ca">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
    <title>Exemple jQuery</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
<div class="container">
    <h1>Exemple jQuery</h1>
    <p id="paragraf">En un lugar de <strong>la Mancha</strong> de cuyo nombre no <em>quie
ro acordarme</em>.</p>
    <input type="text" id="inputText" value="Valor de l'input">
    <a href="https://www.exemple.com" id="link">Enllaç d'exemple</a>
    <button id="btnText">Mostrar Text</button>
    <button id="btnHtml">Mostrar HTML</button>
    <button id="btnVal">Mostrar Valor</button>
    <button id="btnAttr">Mostrar Atribut</button>
    <div id="resultat" class="result"></div>
</div>
<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
<script src="script.js"></script>
</body>
</html>

// JavaScript de referencia
$(document).ready(function() {
    // Muestra el texto del párrafo
    $('#btnText').click(function() {
        let text = $('#paragraf').text();
        $('#resultat').text('Text del paràgraf: ' + text);
    });

    // Muestra el HTML del párrafo
    $('#btnHtml').click(function() {
        let html = $('#paragraf').html();
        $('#resultat').text('HTML del paràgraf: ' + html);
    });

    // Muestra el valor del input
    $('#btnVal').click(function() {
        let valor = $('#inputText').val();
        $('#resultat').text('Valor de l\'input: ' + valor);
    });

    // Muestra el atributo href del enlace
    $('#btnAttr').click(function() {
        let href = $('#link').attr('href');
        $('#resultat').text('Atribut "href" de l\'enllaç: ' + href);
    });
});
```

5. Añadir Datos

Métodos para Añadir Contenido

- **Ejemplos**:

```
javascriptCopiar código
$(document).ready(function() {
    // Añadir contenido al final de los párrafos
    $("p").append(" <span>Adeu</span>");

    // Añadir contenido al principio de los párrafos
    $("p").prepend("<span>Hola</span> ");

    // Añadir contenido antes de los párrafos
```

```
    $("p").before("<b>Salutació: </b>");

    // Añadir contenido después de los párrafos
    $("p").after("<br>");

    // Envolver los párrafos con un div con clase 'container'
    $("p").wrap("<div class='container'></div>");
});
```

6. Eliminar Datos

Métodos para Eliminar Contenido

- **Ejemplos**:

```
javascriptCopiar código
$(document).ready(function(){
    // Vacía el contenido del div 1 al hacer clic en el botón "Buida Contingut Div 1"
    $("#buida").click(function(){
        $("div:nth-child(1)").empty(); // Div 1 estará vacío, pero seguirá existiendo en
el DOM
    });

    // Elimina el div 2 completamente al hacer clic en el botón "Elimina Div 2"
    $("#elimina").click(function(){
        $("div:nth-child(2)").remove(); // Div 2 y todo su contenido serán eliminados com
pletamente del DOM
    });

    // Elimina el contenedor directo de los elementos seleccionados al hacer clic en el b
otón "Elimina Contenidor Directe"
    $("#unwrap").click(function(){
        $("p").unwrap(); // Elimina el div.contenidor dejando solo el párrafo
    });

    // Elimina el atributo href de todos los enlaces al hacer clic en el botón "Elimina A
tribut Href"
    $("#eliminar-attr").click(function(){
        $("a").removeAttr("href"); // Se eliminará el atributo href de todos los enlaces
<a>
    });
});
```

7. Manipulación de CSS

Métodos para Manipular CSS

- **Ejemplos**:

```
javascriptCopiar código
$(document).ready(function(){
    // Añadir clase "fons-roig" al hacer clic en el botón "Afegir Classe"
    $("#afegir").click(function(){
        $("div").addClass("fons-roig");
    });

    // Eliminar clase "mida-gran" al hacer clic en el botón "Eliminar Classe"
    $("#eliminar").click(function(){
        $("p").removeClass("mida-gran");
    });
```

```
    // Alternar clase "negreta" al hacer clic en el texto
    $("#alternar").click(function(){
        $(this).toggleClass("negreta");
    });
});
```

Resumen

1. **Inicialización de jQuery**:
   - Incluir jQuery mediante un CDN.
   - Usar `$(document).ready()` para asegurarse de que el código se ejecute después de cargar la página.
2. **Sintaxis de jQuery**:
   - Seleccionar elementos HTML y aplicar acciones sobre ellos.
   - Ejemplos: `hide()`, `show()`, `fadeIn()`, `toggle()`.
3. **Obtención y Modificación de Datos**:
   - Funciones para obtener y modificar datos del contenido HTML: `text()`, `html()`, `val()`, `attr()`.
4. **Añadir Datos**:
   - Funciones para añadir contenido HTML: `append()`, `prepend()`, `before()`, `after()`, `wrap()`.
5. **Eliminar Datos**:
   - Métodos para eliminar datos o elementos HTML: `empty()`, `remove()`, `unwrap()`, `removeAttr()`.
6. **Manipulación de CSS**:
   - Añadir, eliminar y alternar clases CSS: `addClass()`, `removeClass()`, `toggleClass()`.

JsRender

1. Introducción a JsRender

JsRender

- **Descripción**:
  - Biblioteca JavaScript que permite generar HTML fácilmente utilizando plantillas.
  - Parte de la familia de herramientas JsViews, que incluye JsViews para la vinculación de datos.
  - Permite un mayor control sobre la generación de contenido dinámico en HTML.

---

2. Sintaxis Básica de JsRender

2.1. Interpolación de Datos

- **Descripción**:
  - Proceso de sustitución de variables o expresiones con sus valores correspondientes en una plantilla.
- **Ejemplo**:

```
htmlCopiar código
<script id="myTemplate" type="text/x-jsrender">
    <div>
        <p>El meu nom és {{:nom}} i tinc {{:edat}} anys.</p>
    </div>
</script>
```

```
javascriptCopiar código
// Dades per a la plantilla
var dades = { nom: "Anna", edat: 25 };
```

```
// Renderitzar la plantilla amb les dades
var plantilla = $.templates("#myTemplate");
var htmlSortida = plantilla.render(dades);

// Afegir l'HTML generat al DOM
$("#sortida").html(htmlSortida);
```

2.2. Sentencias Condicionales

- **Descripción**:
  - Controlar qué contenido se renderiza en la plantilla mediante condicionales.
- **Ejemplo**:

```
htmlCopiar código
<script id="myTemplate" type="text/x-jsrender">
    {{if edat >= 18}}
        <p>Major d'edat</p>
    {{else}}
        <p>Menor d'edat</p>
    {{/if}}
</script>
```

```
javascriptCopiar código
// Dades per a la plantilla
var dades = { edat: 20 };

// Renderitzar la plantilla amb les dades
var plantilla = $.templates("#myTemplate");
var htmlSortida = plantilla.render(dades);

// Afegir l'HTML generat al DOM
$("#sortida").html(htmlSortida);
```

2.3. Iteración sobre Arrays

- **Descripción**:
  - Renderizar una lista de datos almacenados en un array.
- **Ejemplo**:

```
htmlCopiar código
<script id="myTemplate" type="text/x-jsrender">
    <ul>
        {{for persones}}
            <li>{{:nom}} - {{:edat}}</li>
        {{/for}}
    </ul>
</script>
```

```
javascriptCopiar código
// Dades per a la plantilla
var dades = {
    persones: [
        { nom: "Joan", edat: 30 },
        { nom: "Maria", edat: 25 }
    ]
};
```

```
// Renderitzar la plantilla amb les dades
var plantilla = $.templates("#myTemplate");
var htmlSortida = plantilla.render(dades);

// Afegir l'HTML generat al DOM
$("#sortida").html(htmlSortida);
```

2.4. Iteración sobre las Propiedades de un Objeto

- **Descripción**:
  - Renderizar las propiedades de un objeto.
- **Ejemplo**:

```
htmlCopiar código
<script id="myTemplate" type="text/x-jsrender">
    <ul>
        {{props persona}}
            <li>{{:prop}}: {{:~val}}</li>
        {{/props}}
    </ul>
</script>
```

```
javascriptCopiar código
// Dades per a la plantilla
var dades = {
    persona: {
        nom: "Anna",
        edat: 25,
        ciutat: "València"
    }
};

// Renderitzar la plantilla amb les dades
var plantilla = $.templates("#myTemplate");
var htmlSortida = plantilla.render(dades);

// Afegir l'HTML generat al DOM
$("#sortida").html(htmlSortida);
```

3. Utilización de JsRender

3.1. Configuración

- **Descripción**:
  - Incluir JsRender en el proyecto descargando el archivo desde el sitio web oficial o utilizando un CDN.
- **Ejemplo**:

```
htmlCopiar código
<script src="https://cdnjs.cloudflare.com/ajax/libs/jsrender/1.0.13/jsrender.min.js"></sc
ript>
```

3.2. Crear una Plantilla JsRender

- **Descripción**:
  - Definir una plantilla dentro de una etiqueta `<script>` con un tipo específico.
- **Ejemplo**:

```
htmlCopiar código
<script id="plantilla-persona" type="text/x-jsrender">
    <div>
        <p>Nom: {{:nom}}</p>
        <p>Edat: {{:edat}}</p>
    </div>
</script>
```

3.3. Renderizar la Plantilla

- **Descripción**:
  - Renderizar la plantilla con datos específicos.
- **Ejemplo**:

```
htmlCopiar código
<div id="sortida"></div>

<script>
    // Dades per a la plantilla
    var dades = { nom: "Jaume Aragó", edat: 30 };

    // Renderitzar la plantilla amb les dades
    var plantilla = $.templates("#plantilla-persona");
    var htmlSortida = plantilla.render(dades);

    // Afegir l'HTML generat al DOM
    $("#sortida").html(htmlSortida);
</script>
```

3.4. Utilizar Bucles y Condicionales

- **Descripción**:
  - Usar bucles y condicionales en las plantillas para manejar datos de manera más dinámica.
- **Ejemplo**:

```
htmlCopiar código
<script id="plantilla-persones" type="text/x-jsrender">
    {{for persones}}
        <div>
            <p>Nom: {{:nom}}</p>
            <p>Edat: {{:edat}}</p>
        </div>
    {{/for}}
</script>

<div id="sortida"></div>

<script>
    // Dades per a la plantilla
    var dades = {
        persones: [
            { nom: "Jaume Aragó", edat: 30 },
            { nom: "Verònica Mascarós", edat: 25 },
            { nom: "Andrei Micleusanu", edat: 40 }
        ]
    };

    // Renderitzar la plantilla amb les dades
    var plantilla = $.templates("#plantilla-persones");
    var htmlSortida = plantilla.render(dades);
```

```
    // Afegir l'HTML generat al DOM
    $("#sortida").html(htmlSortida);
</script>
```

4. JsRender con JSON Externo

4.1. Cargar Datos JSON Externos

- **Descripción**:
  - Cargar datos desde un archivo JSON externo y renderizar plantillas con esos datos.
- **Ejemplo**:

```
htmlCopiar código
<!DOCTYPE html>
<html lang="ca">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Exemple JsRender</title>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.7.1/jquery.min.js"></scr
ipt>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/jsrender/1.0.13/jsrender.min.js">
</script>
</head>
<body>

    <div id="persona"></div>
    <hr>
    <div id="persones"></div>

    <script id="plantilla-persona" type="text/x-jsrender">
        <div>
            <p>Nom: {{:nom}}</p>
            <p>Edat: {{:edat}}</p>
            {{if casada}}
                <p>Està casada</p>
            {{else}}
                <p>No està casada</p>
            {{/if}}
        </div>
    </script>

    <script id="plantilla-persones" type="text/x-jsrender">
        {{for persones}}
            <div>
                <p>Nom: {{:nom}}</p>
                <p>Edat: {{:edat}}</p>
            </div>
        {{/for}}
    </script>

    <script>
        $(function() {
            // Carregar les dades JSON externes
            $.getJSON("dades.json", function(dades) {
                // Renderitzar la plantilla per a una sola persona
                var plantillaPersona = $.templates("#plantilla-persona");
                var htmlPersona = plantillaPersona.render(dades.persona);

                $("#persona").html(htmlPersona);
```

```
                // Renderitzar la plantilla per a l'array de persones
                var plantillaPersones = $.templates("#plantilla-persones");
                var htmlPersones = plantillaPersones.render(dades);

                $("#persones").html(htmlPersones);
            });
        });
    </script>

</body>
</html>
```

Resumen

1. **Interpolación de Datos**: Uso de `{{:variable}}` para insertar valores dinámicos en plantillas.
2. **Sentencias Condicionales**: Uso de `{{if condició}}...{{else}}...{{/if}}` para renderizar contenido condicionalmente.
3. **Iteración sobre Arrays**: Uso de `{{for array}}...{{/for}}` para recorrer y renderizar elementos de un array.
4. **Iteración sobre Propiedades de un Objeto**: Uso de `{{props object}}...{{/props}}` para recorrer y renderizar propiedades de un objeto.
5. **Configuración de JsRender**: Incluir JsRender mediante CDN y definir plantillas en etiquetas `<script>`.
6. **Renderizar Plantillas**: Uso de `$.templates("#template-id").render(data)` para generar HTML a partir de plantillas y datos.
7. **JsRender con JSON Externo**: Cargar datos JSON desde un archivo externo y renderizar plantillas con esos datos utilizando `$.getJSON`.