XPATH y XSLT Cheatsheet

XPATH - Expresiones de Ruta

Selectores Básicos
```xpath
elemento          # Selecciona elementos con ese nombre
*                 # Cualquier elemento
@nombre           # Atributo específico
@*                # Cualquier atributo
texto()           # Nodos de texto
node()            # Cualquier tipo de nodo
```

Rutas
```xpath
/                 # Ruta absoluta desde la raíz
//                # Selecciona elementos en cualquier nivel
.                 # Nodo actual
..                # Nodo padre
/elemento         # Elemento hijo directo de la raíz
//elemento        # Elemento en cualquier nivel
elemento/hijo     # Hijo directo
elemento//hijo    # Hijo en cualquier nivel inferior
```

Predicados
```xpath
[1]               # Primer elemento
[last()]          # Último elemento
[position()=n]    # Elemento en posición n
[@atrib='valor']  # Elementos con atributo específico
[elemento='valor']# Elementos con hijo específico
```

Operadores
```xpath
|                 # Unión de conjuntos
and               # Y lógico
or                # O lógico
=, !=             # Igual, distinto
<, <=, >, >=      # Comparaciones
+, -, *, div      # Operaciones aritméticas
```

Funciones Comunes
```xpath
count()           # Cuenta nodos
sum()             # Suma valores
contains()        # Busca subcadena
starts-with()     # Comienza con
substring()       # Extrae parte de cadena
translate()       # Reemplaza caracteres
normalize-space() # Normaliza espacios
not()             # Negación
```

Sintaxis Avanzada XPath

Expresiones con Variables
```xpath
//libro[@id = $var]              # Comparación con variable
//venta[categoria = $cat-actual] # Filtrado usando variable
//*[precio > $precio-min]        # Comparación numérica con variable
```

Predicados Compuestos
```xpath
//libro[autor='Cervantes'][año>1600]    # Múltiples condiciones (AND)
//libro[@tipo='novela' or @tipo='poesía']# Condición OR
//autor[not(nacionalidad)]              # Negación
//libro[position() mod 2 = 0]           # Elementos pares
```

Expresiones de Ruta Complejas
```xpath
//libro[autor = preceding::autor[1]]     # Comparación con elemento anterior
//venta[categoria = following::categoria]# Comparación con elemento siguiente
```

```
//libro[count(autor) > 1]              # Conteo en predicado
//capitulo[not(. = preceding::capitulo)] # Elementos únicos
```

Funciones en Predicados
```xpath
//precio[sum(../cantidad) > 1000]       # Suma en predicado
//libro[contains(titulo, 'Don')]        # Búsqueda en texto
//autor[starts-with(nombre, 'A')]       # Comienza con
//fecha[substring(., 1, 4) = '2024']    # Substring en predicado
```

Expresiones de Comparación
```xpath
//libro[precio > //precio[1]]           # Comparación con otro elemento
//venta[importe = max(//importe)]       # Uso de funciones de agregación
//empleado[salario > avg(//salario)]    # Comparación con promedio
```

Navegación Contextual
```xpath
.//libro                    # Búsqueda desde nodo actual
../hermano                  # Elemento hermano
ancestor::seccion           # Ancestro específico
descendant::párrafo         # Descendiente específico
preceding-sibling::capitulo  # Hermano anterior
following-sibling::seccion   # Hermano siguiente
```

Ejemplos Prácticos
```xpath
# Encontrar ventas de una categoría específica
//venta[categoria = $cat-actual]

# Encontrar libros con precio mayor al promedio
//libro[precio > sum(//libro/precio) div count(//libro)]

# Encontrar elementos únicos
//elemento[not(. = preceding::elemento)]

# Encontrar elementos relacionados
//pedido[cliente = //cliente[ciudad='Madrid']/@id]

# Filtrado múltiple con variables
//producto[precio > $min and precio < $max]
[categoria = $cat][stock > 0]
```

Notas Importantes
1. **Uso de Variables**
   - Las variables siempre van precedidas de `$`
   - Deben estar definidas en XSLT con `<xsl:variable>`
   - Se usan para comparaciones dinámicas

2. **Predicados**
   - Se pueden encadenar múltiples predicados
   - Se evalúan de izquierda a derecha
   - Pueden contener expresiones complejas

3. **Funciones**
   - Se pueden usar dentro de predicados
   - Permiten operaciones complejas
   - Pueden anidarse

4. **Contexto**
   - `.` se refiere al nodo actual
   - `..` se refiere al nodo padre
   - `//` busca en todo el documento
   - `/` busca desde la raíz

XSLT - Transformaciones

Estructura Básica XSLT
```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="xml/html" indent="yes"/>
```

```xml
    <xsl:template match="/">
        <!-- Contenido -->
    </xsl:template>
</xsl:stylesheet>
```

Elementos XSLT Principales
```xml
<xsl:template>              # Define plantillas
<xsl:value-of>             # Extrae valor
<xsl:for-each>             # Itera elementos
<xsl:if>                  # Condiciones
<xsl:choose>               # Switch
<xsl:when>                # Case del switch
<xsl:otherwise>            # Default del switch
<xsl:sort>                # Ordenamiento
<xsl:apply-templates>      # Aplica plantillas
```

Atributos Comunes
```xml
select=""         # Selecciona nodos
match=""          # Patrón de coincidencia
mode=""           # Modo de procesamiento
test=""          # Condición en if/when
order=""          # Orden (ascending/descending)
```

Transformación XML a HTML

Estructura Básica
```xml
<xsl:template match="/">
    <html>
        <head>
            <title>Título</title>
        </head>
        <body>
            <xsl:apply-templates/>
        </body>
    </html>
</xsl:template>
```

Ejemplo Tabla HTML
```xml
<xsl:template match="elementos">
    <table>
        <tr>
            <th>Título</th>
            <th>Autor</th>
        </tr>
        <xsl:for-each select="elemento">
            <tr>
                <td><xsl:value-of select="titulo"/></td>
                <td><xsl:value-of select="autor"/></td>
            </tr>
        </xsl:for-each>
    </table>
</xsl:template>
```

Transformación XML a XML

Estructura Básica
```xml
<xsl:output method="xml" indent="yes"/>
<xsl:template match="/">
    <nuevo-root>
        <xsl:apply-templates/>
    </nuevo-root>
</xsl:template>
```

Manejo de Atributos
```xml
<!-- Forma directa -->
<elemento atributo="{expresión-xpath}"/>
```

```xml
<!-- Con xsl:attribute -->
<elemento>
   <xsl:attribute name="nombre">
      <xsl:value-of select="expresión"/>
   </xsl:attribute>
</elemento>
```

Sintaxis de Llaves {...} en XSLT

Uso Básico
```xml
<!-- Las llaves {} permiten evaluar expresiones XPath dentro de atributos -->
<elemento atributo="{expresión-xpath}"/>

<!-- Ejemplos básicos -->
<div id="{@id}"/>                  # Copia el valor del atributo id
<span class="{nombre}"/>           # Usa el valor del elemento nombre
<p title="{.}"/>                   # Usa el valor del nodo actual
```

Casos Comunes
```xml
<!-- {.} - Valor del nodo actual -->
<categoria nombre="{.}"/>          # El valor del texto del nodo actual
<elemento valor="{./texto}"/>      # Valor de un hijo directo

<!-- {@atributo} - Valor de atributos -->
<div id="{@id}"/>                  # Copia un atributo
<span class="{@tipo}"/>            # Usa un atributo como clase

<!-- Expresiones compuestas -->
<div class="precio-{@tipo}"/>      # Concatena texto con valor
<span title="{nombre} - {apellido}"/> # Combina múltiples valores
```

Ejemplos Avanzados
```xml
<!-- Funciones dentro de llaves -->
<p class="{concat(tipo, '-', @id)}"/>

<!-- Operaciones matemáticas -->
<precio total="{precio * cantidad}"/>

<!-- Condiciones -->
<div class="{if (@tipo='especial') then 'destacado' else 'normal'}"/>

<!-- Variables -->
<span valor="{$variable}"/>
```

Notas Importantes
1. Las llaves `{...}` solo funcionan en valores de atributos
2. Para texto dentro de elementos, usar `<xsl:value-of>`
3. No se pueden anidar llaves
4. Las llaves evalúan la expresión XPath inmediatamente

Técnicas Avanzadas

Agrupación
```xml
<xsl:for-each select="grupo[not(./valor = preceding::grupo/valor)]">
   <grupo>
      <xsl:for-each select="//grupo[./valor = current()/valor]">
         <!-- Elementos del grupo -->
      </xsl:for-each>
   </grupo>
</xsl:for-each>
```

Ordenamiento
```xml
<xsl:sort select="campo"
        order="ascending"
        data-type="text/number"/>
```

Condicionales
```xml
```

```xml
<xsl:choose>
   <xsl:when test="condición1">
      <!-- resultado 1 -->
   </xsl:when>
   <xsl:when test="condición2">
      <!-- resultado 2 -->
   </xsl:when>
   <xsl:otherwise>
      <!-- resultado por defecto -->
   </xsl:otherwise>
</xsl:choose>
```

Buenas Prácticas

1. **Rutas XPath**
   - Usar rutas específicas (`biblioteca/libro` vs `//libro`)
   - Evitar rutas absolutas cuando sea posible
   - Usar predicados para filtrar con precisión

2. **Plantillas**
   - Usar plantillas nombradas para código reutilizable
   - Separar la lógica en plantillas diferentes
   - Usar modos para diferentes contextos

3. **Organización**
   - Mantener una estructura clara
   - Comentar secciones complejas
   - Usar indentación consistente

4. **Rendimiento**
   - Evitar procesamiento redundante
   - Usar keys para búsquedas eficientes
   - Minimizar el uso de //

Ejemplos Comunes

Listado con Filtro
```xml
<xsl:template match="libros">
   <ul>
      <xsl:for-each select="libro[precio &lt; 20]">
         <xsl:sort select="titulo"/>
         <li><xsl:value-of select="titulo"/></li>
      </xsl:for-each>
   </ul>
</xsl:template>
```

Tabla con Totales
```xml
<xsl:template match="ventas">
   <table>
      <xsl:for-each select="venta">
         <tr>
            <td><xsl:value-of select="producto"/></td>
            <td><xsl:value-of select="importe"/></td>
         </tr>
      </xsl:for-each>
      <tr>
         <td>Total:</td>
         <td><xsl:value-of select="sum(//importe)"/></td>
      </tr>
   </table>
</xsl:template>
```

Ejemplos Detallados con apply-templates

XML a HTML - Ejemplos

1. Estructura Jerárquica
```xml
<!-- XML de entrada
<empresa>
   <departamento nombre="Ventas">
      <empleado>
         <nombre>Juan</nombre>
         <cargo>Vendedor</cargo>
```

```
        </empleado>
        <empleado>
            <nombre>Ana</nombre>
            <cargo>Gerente</cargo>
        </empleado>
    </departamento>
</empresa>
-->

<xsl:template match="/">
    <html>
        <body>
            <h1>Estructura de la Empresa</h1>
            <xsl:apply-templates select="empresa/departamento"/>
        </body>
    </html>
</xsl:template>

<xsl:template match="departamento">
    <div class="departamento">
        <h2>Departamento: <xsl:value-of select="@nombre"/></h2>
        <xsl:apply-templates select="empleado"/>
    </div>
</xsl:template>

<xsl:template match="empleado">
    <div class="empleado">
        <h3><xsl:value-of select="nombre"/></h3>
        <p>Cargo: <xsl:value-of select="cargo"/></p>
    </div>
</xsl:template>
```

2. Listas Anidadas
```xml
<!-- XML de entrada
<categorias>
    <categoria nombre="Libros">
        <subcategoria nombre="Ficción">
            <item>Novela</item>
            <item>Poesía</item>
        </subcategoria>
        <subcategoria nombre="No Ficción">
            <item>Historia</item>
            <item>Ciencia</item>
        </subcategoria>
    </categoria>
</categorias>
-->

<xsl:template match="/">
    <html>
        <body>
            <h1>Catálogo</h1>
            <xsl:apply-templates select="categorias/categoria"/>
        </body>
    </html>
</xsl:template>

<xsl:template match="categoria">
    <div>
        <h2><xsl:value-of select="@nombre"/></h2>
        <xsl:apply-templates select="subcategoria"/>
    </div>
</xsl:template>

<xsl:template match="subcategoria">
    <h3><xsl:value-of select="@nombre"/></h3>
    <ul>
        <xsl:apply-templates select="item"/>
    </ul>
</xsl:template>

<xsl:template match="item">
    <li><xsl:value-of select="."/></li>
</xsl:template>
```

XML a XML - Ejemplos

1. Reestructuración con Atributos
```xml
<!-- XML de entrada
<biblioteca>
   <libro id="1">
      <titulo>Don Quijote</titulo>
      <autor>Cervantes</autor>
      <año>1605</año>
   </libro>
</biblioteca>
-->

<xsl:template match="/">
   <catalogo>
      <xsl:apply-templates select="biblioteca/libro"/>
   </catalogo>
</xsl:template>

<xsl:template match="libro">
   <obra tipo="libro" id="{@id}">
      <xsl:apply-templates select="titulo"/>
      <xsl:apply-templates select="autor"/>
      <fecha-publicacion><xsl:value-of select="año"/></fecha-publicacion>
   </obra>
</xsl:template>

<xsl:template match="titulo">
   <nombre-obra><xsl:value-of select="."/></nombre-obra>
</xsl:template>

<xsl:template match="autor">
   <escritor><xsl:value-of select="."/></escritor>
</xsl:template>
```

2. Transformación con Modos
```xml
<!-- XML de entrada
<datos>
   <persona>
      <nombre>Juan</nombre>
      <edad>30</edad>
   </persona>
</datos>
-->

<xsl:template match="/">
   <resultado>
      <formato-detallado>
         <xsl:apply-templates select="//persona" mode="detallado"/>
      </formato-detallado>
      <formato-resumido>
         <xsl:apply-templates select="//persona" mode="resumido"/>
      </formato-resumido>
   </resultado>
</xsl:template>

<xsl:template match="persona" mode="detallado">
   <persona-detalle>
      <nombre-completo><xsl:value-of select="nombre"/></nombre-completo>
      <años><xsl:value-of select="edad"/></años>
   </persona-detalle>
</xsl:template>

<xsl:template match="persona" mode="resumido">
   <persona nombre="{nombre}" edad="{edad}"/>
</xsl:template>
```

3. Agrupación Avanzada
```xml
<!-- XML de entrada
<ventas>
   <venta>
      <producto>A</producto>
      <categoria>Electrónica</categoria>
      <precio>100</precio>
   </venta>
```

```
</ventas>
-->

<xsl:template match="/">
   <resumen-ventas>
      <xsl:apply-templates select="//categoria[not(. = preceding::categoria)]" mode="grupo"/>
   </resumen-ventas>
</xsl:template>

<xsl:template match="categoria" mode="grupo">
   <xsl:variable name="cat-actual" select="."/>
   <categoria-ventas nombre="{.}">
      <productos>
         <xsl:apply-templates select="//venta[categoria = $cat-actual]"/>
      </productos>
      <total>
         <xsl:value-of select="sum(//venta[categoria = $cat-actual]/precio)"/>
      </total>
   </categoria-ventas>
</xsl:template>

<xsl:template match="venta">
   <producto>
      <nombre><xsl:value-of select="producto"/></nombre>
      <precio><xsl:value-of select="precio"/></precio>
   </producto>
</xsl:template>
```
```

### Consejos para apply-templates

1. **Uso de Modos**
   - Permite procesar el mismo elemento de diferentes formas
   - Útil para generar múltiples vistas del mismo dato
   - Mantiene el código organizado

2. **Prioridad de Plantillas**
   - Las plantillas más específicas tienen prioridad
   - Se puede usar el atributo `priority` para control explícito
   - El orden de declaración importa cuando hay igual especificidad

3. **Contexto**
   - `current()` para referirse al nodo actual en procesamiento
   - `position()` para obtener la posición en el conjunto actual
   - Variables para almacenar valores temporales

4. **Buenas Prácticas**
   - Mantener plantillas pequeñas y específicas
   - Usar nombres descriptivos para los modos
   - Documentar el propósito de cada plantilla
   - Estructurar jerárquicamente las transformaciones

### Control de Espacios en Blanco
```xml
<xsl:strip-space elements="*"/>        # Elimina espacios en blanco de todos los elementos
<xsl:preserve-space elements="pre"/>    # Preserva espacios en elementos específicos
<xsl:text>
</xsl:text>             # Insertar salto de línea explícito
```

Variables y Parámetros
```xml
<!-- Definición de variables -->
<xsl:variable name="nombre" select="expresión"/>
<xsl:variable name="nombre">
   <!-- contenido complejo -->
</xsl:variable>

<!-- Uso de variables -->
<xsl:value-of select="$nombre"/>

<!-- Parámetros -->
<xsl:param name="titulo" select="'Valor por defecto'"/>
```

Manipulación de Texto
```xml
<xsl:value-of select="concat(string1, string2)"/>     # Concatenar strings
<xsl:value-of select="substring(string, start, len)"/> # Extraer subcadena
<xsl:value-of select="translate(string, 'abc', 'ABC')"/> # Reemplazar caracteres
```
```

```
<xsl:value-of select="normalize-space(string)"/>       # Normalizar espacios
```

Procesamiento Condicional Avanzado
```xml
<!-- Selección múltiple con when -->
<xsl:choose>
   <xsl:when test="@tipo='libro'">
      <libro><xsl:apply-templates/></libro>
   </xsl:when>
   <xsl:when test="@tipo='revista'">
      <revista><xsl:apply-templates/></revista>
   </xsl:when>
   <xsl:otherwise>
      <documento><xsl:apply-templates/></documento>
   </xsl:otherwise>
</xsl:choose>

<!-- Condiciones con if -->
<xsl:if test="precio > 100">
   <span class="caro"><xsl:value-of select="precio"/></span>
</xsl:if>
```

Numeración y Conteo
```xml
<xsl:number value="position()" format="1"/>    # Números (1,2,3...)
<xsl:number value="position()" format="a"/>    # Letras (a,b,c...)
<xsl:number value="position()" format="i"/>    # Números romanos (i,ii,iii...)
<xsl:value-of select="count(//libro)"/>        # Contar elementos
<xsl:value-of select="sum(//precio)"/>         # Sumar valores
```
Manejo de Errores
```xml
<!-- Verificar existencia antes de procesar -->
<xsl:if test="autor">
   <xsl:value-of select="autor"/>
</xsl:if>

<!-- Valores por defecto -->
<xsl:value-of select="autor"/>
<xsl:text>Autor desconocido</xsl:text>
```

Ordenamiento Avanzado
```xml
<!-- Ordenamiento múltiple -->
<xsl:for-each select="libro">
   <xsl:sort select="autor"/>
   <xsl:sort select="titulo"/>
   <!-- contenido -->
</xsl:for-each>

<!-- Ordenamiento personalizado -->
<xsl:sort select="precio"
         data-type="number"
         order="descending"/>
```

Comentarios y Documentación
```xml
<!-- Comentario normal -->

<!--
   Documentación de plantilla:
   - Propósito: Procesa elementos libro
   - Parámetros: ninguno
   - Salida: elemento HTML div
-->
<xsl:template match="libro">
   <!-- implementación -->
</xsl:template>
```

Ejercicios Completos de Examen
```xml
<!-- XML de entrada:
<empresa>
   <empleado departamento="Ventas">
```

```
      <nombre>Juan</nombre>
      <salario>30000</salario>
      <antiguedad>5</antiguedad>
   </empleado>
   <empleado departamento="IT">
      <nombre>Ana</nombre>
      <salario>35000</salario>
      <antiguedad>3</antiguedad>
   </empleado>
</empresa>
-->

<!-- Crear un informe HTML que:
    1. Agrupe por departamento
    2. Calcule salario promedio por departamento
    3. Resalte empleados con más de 4 años
    4. Muestre estadísticas generales
-->

<xsl:template match="/">
   <html>
      <head>
         <style>
            .veterano { color: green; }
            .departamento { margin: 20px; }
         </style>
      </head>
      <body>
         <h1>Informe de Empleados</h1>

         <!-- Procesamiento por departamentos -->
         <xsl:for-each select="//empleado[not(@departamento = preceding::empleado/@departamento)]">
            <xsl:variable name="dept-actual" select="@departamento"/>
            <div class="departamento">
               <h2>Departamento: <xsl:value-of select="$dept-actual"/></h2>

               <!-- Lista de empleados -->
               <ul>
                  <xsl:for-each select="//empleado[@departamento = $dept-actual]">
                     <li>
                        <xsl:choose>
                           <xsl:when test="antiguedad > 4">
                              <span class="veterano">
                                 <xsl:value-of select="nombre"/>
                                 (<xsl:value-of select="antiguedad"/> años)
                              </span>
                           </xsl:when>
                           <xsl:otherwise>
                              <xsl:value-of select="nombre"/>
                              (<xsl:value-of select="antiguedad"/> años)
                           </xsl:otherwise>
                        </xsl:choose>
                     </li>
                  </xsl:for-each>
               </ul>

               <!-- Estadísticas del departamento -->
               <p>
                  Salario promedio:
                  <xsl:value-of
                     select="format-number(sum(//empleado[@departamento = $dept-actual]/salario)
                           div count(//empleado[@departamento = $dept-actual]), '#,##0')"/>€
               </p>
            </div>
         </xsl:for-each>

         <!-- Estadísticas generales -->
         <div class="estadisticas">
            <h2>Estadísticas Generales</h2>
            <ul>
               <li>Total empleados: <xsl:value-of select="count(//empleado)"/></li>
               <li>Salario promedio general:
                  <xsl:value-of select="format-number(sum(//salario) div count(//empleado), '#,##0')"/>€
               </li>
               <li>Empleados veteranos: <xsl:value-of select="count(//empleado[antiguedad > 4])"/></li>
            </ul>
         </div>
      </body>
   </html>
```

```
</xsl:template>
```
Transformación XML a HTML con agrupación:
```xml
<!-- XML de entrada:
<biblioteca>
   <libro>
      <genero>Ficción</genero>
      <titulo>Don Quijote</titulo>
      <autor>Cervantes</autor>
   </libro>
   <libro>
      <genero>Ficción</genero>
      <titulo>Cien años de soledad</titulo>
      <autor>García Márquez</autor>
   </libro>
   <libro>
      <genero>No Ficción</genero>
      <titulo>Historia de España</titulo>
      <autor>Pérez</autor>
   </libro>
</biblioteca>
-->

<!-- Solución: Agrupar libros por género en una tabla HTML -->
<xsl:template match="/">
   <html>
      <body>
         <h1>Biblioteca por Géneros</h1>
         <xsl:for-each select="//libro[not(genero = preceding::libro/genero)]">
            <xsl:variable name="genero-actual" select="genero"/>
            <h2><xsl:value-of select="genero"/></h2>
            <table border="1">
               <tr><th>Título</th><th>Autor</th></tr>
               <xsl:for-each select="//libro[genero = $genero-actual]">
                  <tr>
                     <td><xsl:value-of select="titulo"/></td>
                     <td><xsl:value-of select="autor"/></td>
                  </tr>
               </xsl:for-each>
            </table>
         </xsl:for-each>
      </body>
   </html>
</xsl:template>
```
Transformación con cálculos y condiciones:
```xml
<!-- XML de entrada:
<ventas>
   <venta>
      <producto>A</producto>
      <precio>100</precio>
      <cantidad>2</cantidad>
   </venta>
   <venta>
      <producto>B</producto>
      <precio>50</precio>
      <cantidad>3</cantidad>
   </venta>
</ventas>
-->

<xsl:template match="/">
   <resumen>
      <xsl:variable name="total" select="sum(//venta/precio * //venta/cantidad)"/>
      <ventas-totales>
         <xsl:for-each select="//venta">
            <venta>
               <producto><xsl:value-of select="producto"/></producto>
               <subtotal>
                  <xsl:value-of select="precio * cantidad"/>
               </subtotal>
               <porcentaje-del-total>
                  <xsl:value-of select="format-number((precio * cantidad) div $total * 100, '##.##')"/>%
               </porcentaje-del-total>
            </venta>
         </xsl:for-each>
         <total><xsl:value-of select="$total"/></total>
```

```
        </ventas-totales>
    </resumen>
</xsl:template>
```