Document Object Model:

Förenklat kan man se *Document* som själva HTML-filen och *Object Model* är då som en datastruktur som omfamnar hela HTML-dokumentet. Objektmodellen har på ett hierarkisk vis inkapslat alla beståndsdelar av HTML-dokumentet som vi kan komma åt på olika sätt genom Javascript. Det gör att HTML-dokumentet kan ses som en ”array” i objektmodellen som innehåller alla olika element i dokumentet.

Just att den är hierarkisk blir tydligt när vi i denna uppgift skapade matchraderna för stryktipset.
Målet var att i en befintlig tabell lägga till nya rader och kolumner baserat på data från ett API-anrop. När vi genom DOM vill lägga till ett nytt HTML-element måste vi ange vem som är ”föräldern” till elementet.
Om vi ser till koden i metoden ”SkapaMatchRader” är detta tydligt att för varje ny kolumn som skapas, anger vi att kolumnen är ett barn till föräldern ”raden”. Exempelvis: newRow.appendChild(NrCol). Raderna måste i sin tur också läggas till i sin förälder vilket är tabellen. matchTable.append(newRow);

Användningsområden för DOM i denna uppgift är väldig lik de övningar som föranlett examinationen. Antingen skapar vi nya HTML-element genom createElement eller så väljer vi ut ett befintligt HTML-element genom querySelector. QuerySelector är användbart genom att vi kommer åt hela HTML-dokumentet i objektmodellen och väljer ut ett enskilt element genom att exempelvis peka på ett id eller klass kopplat till ett element. Detta gör vi exempelvis för den befintliga tabellen som har ID ”#table”. Genom att instanserna tabellen i Javascript kan vi säga att de nya raderna som skapas är ”barn” till tabellen och därmed följer vi HTML strukturen.

Asynchronous Javascript and XML:

AJAX är en teknik som tillåter våra hemsidor att bli uppdaterade asynkront genom ett utbyte med en webbserver. Det gör det möjligt att ändra delar av hemsidan utan att behöva ladda om hela sidan.
Den asynkrona biten gör att vi effektivt kan starta olika ”tidsaxlar” i vår exekvering av kod vilket både kan förbättra användarupplevelsen och säkerställa att bearbetning av datainhämtning från webbservar blir korrekta. Jag beskriver i min kodkommentering i större detalj hur vi programmerar asynkront genom fetch och then (promises). Om vi inte hade implementerat AJAX och endast exekverat koden synkront hade det inneburit att datainhämtningen hade blivit inkomplett då koden för att skapa matchraderna hade försökt bearbeta datan utan att vänta in slutförd datainhämtning vilket troligen hade inneburit problem.
Nedanstående implementation som återfinns i min kod är en såkallad ”callback”-funktion. Eftersom att API-anropet för att få fram stryktipsdatan är en asynkronuppgift säger vi att skapa först matchraderna när datan är fullständigt inhämtad.

```
getStrykTipsData().then(function (data) {

   SkapaMatchRader(data);
});
```

Namnet AJAX i sig är kanske lite missvisande i modern tid utifrån X:et som står för XML men som vanligtvis är ersatt av JSON som vi fokuserat på i denna kurs men även uppgiften. HTTP-svaret som vi får genom fetch konverterar vi till en JSON-fil som genom Javascript kan bearbeta och ta fram värden ur. XML må vara lite mer läsbart då det följer likt HTML en hierarkisk struktur med taggnamn. Samtidigt är JSON också enkelt att läsa och tycks enligt många ha en enklare syntax vilket har blivit att föredra i modern webbutveckling.