**- Explain about the Object Model, and why it’s (very) relevant for modern Web-development**

Assuming this question refers to the DOM (Document Object Model), I would say it's relevant because of how simple and efficient it makes coding JavaScript and interacting/manipulating various HTML elements. It also comes with event bubbling, meaning actions from one element deep within multiple others will "bubble up", where you'll be able to grab it through the event object (created with an Eventlistener).

The model itself is cross-platform and language-independent as well, meaning there are no compatibility issues with different browsers etc. Structurally, it's like a tree where multiple elements can branch off of one root element, such as in a table, where you have table headers, rows and so on.

Overall, it's an effective tool that provides a lot of ways to manipulate and add elements to an HTML page in a very efficient manner.

**- Explain (using an example of your own choice) about JavaScript events, and Event Bubbling**

A JavaScript event is an object that you can use when you have an event listener tied to an HTML element, which can be used to manipulate various other elements within the event listening element or get information from them.

An example of event bubbling would be:  
  
You have a table with a number of rows wrapped in a div with an event listener on it. Using the event object from this event listener, you can log, modify or get all of the IDs or inner text values of the table, its headers, its rows and its individual cells through event bubbling.

Event bubbling makes it so whenever you click something within the element with an event listener, it becomes the event object (not sure that's how it actually is, but that's how I think of it).

This means that if you clicked a table row deep within the aforementioned div, its values will "bubble up" until they reach the div (which you have an event listener on), at which point they will be accessible to you. You can then do whatever you need to do with them.

**- Elaborate on how JSON or XML supports communication between subsystems, even when the subsystems are implemented on diﬀerent platforms.**

In short: language-independent. There are no limitations on which languages can use or read JSON, so it has high compatibility and is easy to understand and use.

**- Explain the topic AJAX and how it has changed the way modern web-applications are created**

AJAX (Asynchronous JavaScript And XML) allows us to create single-page applications (SPA). Previously, web developers could only create multi-page applications (MPA), which are slower since every request must fetch and render an entirely new HTML page (at least if you're being redirected/clicking a link).

Single-page applications, however, are downloaded in their entirety on the first request and contain just one HTML page. It can still have many different sub-pages and things to display, but it does not have to be rendered again at any point. This is possible through the use of JavaScript, where we can control what the single page looks like and acts like based on where the user clicks.

Naturally, this is also a lot faster than MPA and has made a significant impact on the efficiency and speed of modern websites. It allows for cleaner and faster web-applications, but it is of course also heavily reliant on JavaScript, where there are a lot of other options and frameworks for multi-page applications (e.g. Java servlets like we used in our 2nd semester).

**- Explain the Same Origin Policy (for AJAX), and different ways to work around it**

The Same Origin Policy basically restricts resource interactions through scripts and documents based on origin. In other words, if the origin is the same for a request and a response, there are no restrictions.

If they're not, the Same Origin Policy will stop any requests made and prevent the user from interacting with the server and getting responses from it.

The easiest way to work around it is with CORS (Cross Origin Resource Sharing). Through the use of CORS headers, such as Access-Control-Allow-Origin, you can bypass the Same Origin Policy restrictions and get resources from a server whose origin is not the same as the one you're requesting from (e.g. getting our database data from our droplet on our locally hosted web application).

The server which sends the requests has to manually add CORS headers to all of its responses to ensure that this is possible.