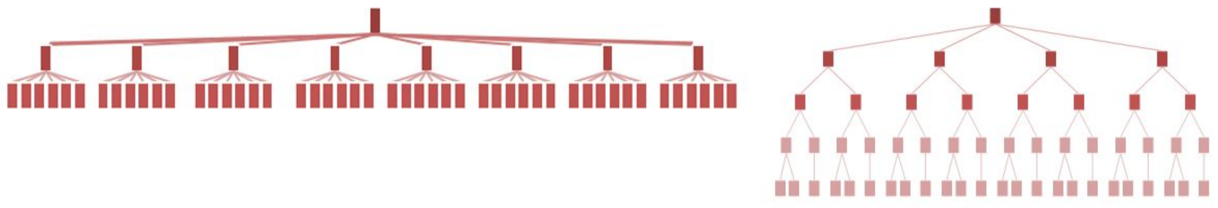
1. **Explain the Flat and Deep architecture of Website.**

   Virtually every website that has more than a few pages uses some structure for organizing the content. The most common (and most easily understood) structure is to categorize pages into groups, often with distinct subgroups. The end result is a hierarchy of content, a structure familiar to most of us from our interactions with organizations, families, and the natural world.

   Decisions about exactly how content should be grouped can have dramatic consequences for how your site's structure works (or doesn't work) for users, but these nuances are difficult to understand at first glance. To analyze how a structure will work, we often need to create a visualization that shows a high-level view of how the different pages of a site relate to each other.

   Consider these 2 structures: each represents the same amount of information, and shows a perfectly logical way of organizing the content for a website. Yet the end-user's experience of browsing these 2 hierarchies — even if they contain exactly the same information — will be very different.



Left: a flat site hierarchy, with few vertical levels. Right: a deep site hierarchy has the same information organized into more sublevels.

Both of these site hierarchies start at the top with a single homepage, but the information below that page is organized quite differently: the website on the left has 8 major categories, but the site on the right has only 4. This side-by-side comparison illustrates what we mean when we talk about flat and deep hierarchies. The flat (or broad) hierarchy on the left looks wide and short, because it has only 3 layers. The structure on the right is deep, with 5 levels, fewer categories and subcategories at each level, and ends up appearing taller and narrower.

Although website visitors never see this type of visualization, the shape of the hierarchy has a huge impact on the end user's experience, for 2 reasons:

1. Content is more discoverable when it's not buried under multiple intervening layers. All other things being equal, deep hierarchies are more difficult to use.

2. Categories that are specific and do not overlap are the easiest to understand. This cuts both ways: In deep hierarchies, when there are only a few categories on each level, they

tend to be more generic and, thus, more confusing.
A flatter hierarchy with more categories at each level usually has more-specific labels that are easier to understand; but in broad hierarchies with a very large number of items, there is often some conceptual overlap between at least a few of the categories. Users can also become overwhelmed with long, cluttered menus.

2. **Discuss the Advanced search techniques.**
Keyword difficulty
When building a web page, it can be useful to know how competitive the keyword is that you are going after, yet this can be difficult to obtain. The intitle: operator shows pages that are more focused on your search term than the pages returned without that operator (e.g., intitle:"dress boots"). You can use different ratios to give you a sense of how competitive a keyword market is (higher results mean that it is more competitive). For example: dress boots (20,900,000) versus “dress boots” (424,000) versus intitle:"dress boots" (37,000) Ratio: 20,900/37 = 565:1 Exact phrase ratio: 424/37 = 11:1 Another significant parameter you can look at is the inanchor: operator; for example, inanchor:"dress boots". You can use this operator in the preceding equation instead of the intitle: operator.
Using number ranges
The number range operator can help restrict the results set to a set of model numbers, product numbers, price ranges, and so forth. For example: site:stevespanglerscience.com "product/1700..1750" Unfortunately, the number range combined with inurl: is not supported. So, the product number must be on the page. The number range operator is also great for copyright year searches (to find abandoned sites to acquire). Combine it with the intext: operator to improve the signal-to-noise ratio; for example, intext:"copyright 1993..2005" -2008 blog.
Advanced doc type search
The filetype: operator is useful for looking for needles in haystacks. Here are a couple of examples:
confidential business plan -template filetype:doc forrester research grapevine filetype:pdf
Determine listing age
You can label results with dates that give a quick sense of how old (and thus trusted) each listing is; for example, by appending the &as_qdr=m199 parameter to the end of a Google SERP URL, you can restrict results to those within the past 199 months
Uncover subscriber-only or deleted content
You can get to subscriber-only or deleted content from the Cached link in the listing in the SERPs or by using the cache: operator. Don’t want to leave a footprint? Add &strip=1 to the end of the Google cached URL. Images on the page won’t load. If no Cached link is available, use Google Translate to take your English document and

translate it from Spanish to English (this will reveal the content even though no Cached link is available):
http://translate.google.com/translate?prev=&hl=en&u=URL-GOES-HERE&sl=es&tl=en
Identify neighborhoods
The related: operator will look at the sites linking (the "Linking Sites") to the specified site, and then see which other sites are commonly linked to by the Linking Sites. This list is usually limited to between 25 and 31 results. These are commonly referred to as neighborhoods, as there is clearly a strong relationship between sites that share similar link graphs.
Find Creative Commons (CC) licensed content
Use the as_rights parameter in the URL to find Creative Commons licensed content. Here are some example scenarios to find CC-licensed material on the Web: Permit commercial use
http://google.com/search?as_rights=(cc_publicdomain|cc_attribute|cc_sharealike|cc_nonderived).-(cc_noncommercial)&q=KEYWORDS
Permit derivative works
http://google.com/search?as_rights=(cc_publicdomain|cc_attribute|cc_sharealike|cc_noncommercial).-(cc_nonderived)&q=KEYWORDS Permit commercial and derivative use
http://google.com/search?as_rights=(cc_publicdomain|cc_attribute|cc_sharealike).-(cc_noncommercial|cc_nonderived)&q=KEYWORDS
Make sure you replace KEYWORDS with the keywords that will help you find content that is of relevance to your site. The value of this to SEO is an indirect one. Creative Commons content can potentially be a good source of content for a website.

3. **Describe the significance & working of WSDL with an example.**
The Web Services Description Language (WSDL) is an XML-based interface definition language that is used for describing the functionality offered by a web service. The acronym is also used for any specific WSDL description of a web service (also referred to as a WSDL file), which provides a machine-readable description of how the service can be called, what parameters it expects, and what data structures it returns. Therefore its purpose is roughly similar to that of a method signature in a programming language.'

WSDL Elements
A WSDL document contains the following elements:

- Definition: It is the root element of all WSDL documents. It defines the name of the web service, declares multiple namespaces used throughout the remainder of the document, and contains all the service elements described here.
- Data types: The data types to be used in the messages are in the form of XML schemas.
- Message: It is an abstract definition of the data, in the form of a message presented either as an entire document or as arguments to be mapped to a method invocation.
- Operation: It is the abstract definition of the operation for a message, such as naming a method, message queue, or business process, that will accept and process the message.
- Port type: It is an abstract set of operations mapped to one or more end-points, defining the collection of operations for a binding; the collection of operations, as it is abstract, can be mapped to multiple transports through various bindings.
- Binding: It is the concrete protocol and data formats for the operations and messages defined for a particular port type.
- Port: It is a combination of a binding and a network address, providing the target address of the service communication.
- Service: It is a collection of related end-points encompassing the service definitions in the file; the services map the binding to the port and include any extensibility definitions.
- In addition to these major elements, the WSDL specification also defines the following utility elements:
- **Documentation:** This element is used to provide human-readable documentation and can be included inside any other WSDL element.
- **Import:** This element is used to import other WSDL documents or XML Schemas.

Example (WSDL File):

```
<?xml version="1.0" encoding="UTF-8"?>
<description xmlns="http://www.w3.org/ns/wsdl"
          xmlns:tns="http://www.tmsws.com/wsdl20sample"
          xmlns:whttp="http://schemas.xmlsoap.org/wsdl/http/"
          xmlns:wsoap="http://schemas.xmlsoap.org/wsdl/soap/"
          targetNamespace="http://www.tmsws.com/wsdl20sample">

<documentation>
   This is a sample WSDL 2.0 document.
</documentation>
```

```
<!-- Abstract type -->
  <types>
    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
             xmlns="http://www.tmsws.com/wsdl20sample"
             targetNamespace="http://www.example.com/wsdl20sample">

      <xs:element name="request"> ... </xs:element>
      <xs:element name="response"> ... </xs:element>
    </xs:schema>
  </types>

<!-- Abstract interfaces -->
  <interface name="Interface1">
    <fault name="Error1" element="tns:response"/>
    <operation name="Get" pattern="http://www.w3.org/ns/wsdl/in-out">
      <input messageLabel="In" element="tns:request"/>
      <output messageLabel="Out" element="tns:response"/>
    </operation>
  </interface>

<!-- Concrete Binding Over HTTP -->
  <binding name="HttpBinding" interface="tns:Interface1"
          type="http://www.w3.org/ns/wsdl/http">
    <operation ref="tns:Get" whttp:method="GET"/>
  </binding>

<!-- Concrete Binding with SOAP-->
  <binding name="SoapBinding" interface="tns:Interface1"
          type="http://www.w3.org/ns/wsdl/soap"
          wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP/"
          wsoap:mepDefault="http://www.w3.org/2003/05/soap/mep/request-response">
    <operation ref="tns:Get" />
  </binding>

<!-- Web Service offering endpoints for both bindings-->
  <service name="Service1" interface="tns:Interface1">
    <endpoint name="HttpEndpoint"
             binding="tns:HttpBinding"
             address="http://www.example.com/rest/"/>
    <endpoint name="SoapEndpoint"
              binding="tns:SoapBinding"
              address="http://www.example.com/soap/"/>
  </service>
</description>
```

4. **What do you mean by JSON?Why use JSON over XML.**

   What is JSON?

JavaScript Object Notation (JSON) is a lightweight text-based open standard designed for human-readable data interchange. It is derived from the JavaScript programming language for representing simple data structures and associative arrays, called objects. Despite its relationship to JavaScript, it is language-independent, with parsers available for most programming languages.

What is XML?

Extensible Markup Language (XML) is a set of rules for encoding documents in machine-readable form. XML's design goals emphasize simplicity, generality, and usability over the Internet.
Further reading: http://en.wikipedia.org/wiki/XML

Reasons to choose JSON over XML

1. JSON requires less tags than XML – XML items must be wrapped in open and close tags whereas JSON you just name the tag once
2. Because JSON is transportation-independent, you can just bypass the XMLHttpRequest object for getting your data.
3. JavaScript is not just data – you can also put methods and all sorts of goodies in JSON format.
4. JSON is better at helping procedural decisions in your JavaScript based on objects and their values (or methods).
5. You can get JSON data from anywhere, not just your own domain. There's no more proxy server nonsense.
6. Yahoo has a really good YUI2 JSON API.
7. JSON is easier to read than XML – Obviously a personal preference

5.

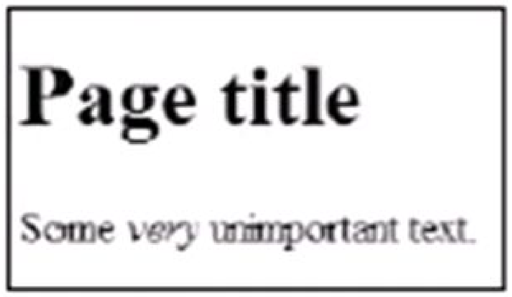
6. **Define DOM.Explain in node tree for HTML document and list level of DOM.**
   The Document Object Model (DOM) is an object-oriented representation of an HTML or XML document. The structure of an HTML and XML document is hierarchical, so the DOM structure resembles that of a tree. DOM provides an API to access and modify this tree of objects. The DOM API is specified in a language- independent manner by the W3C, and mappings are available for most programming languages. The DOM API is used in a number of situations, like on the server side in a Java program to perform some manipulation on an XML document. For the scope of this and the next chapter, we will look in more details at the DOM in the context of the browser and see how with DOM API you can dynamically modify the HTML or XHTML page.

In order to look at the structure of the DOM, consider this short, yet complete and valid XHTML page:

Example :

```
<?xml version=”1.0” encoding=”ISO-8859-1”?>
<!DOCTYPE html PUBLIC “-//W3C//DTD XHTML 1.0 Strict//EN”
 “http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd”>
 <html xmlns=”http://www.w3.org/1999/xhtml”>
  <head>
  <title>DOM Example</title>
  </head>
  <body>
   <h1>Page title</h1>
   <p>
   Some <i>very</i> unimportant text.
   </p>
    </body>
      </html>
```

Output :



Page title

Some very unimportant text.

Levels of DOM:
The DOM has three levels of support; each level corresponds to subsequent recommendations by the W3C:

- Level 1 was first released in 1998, when browsers were already providing a DOM-like API. It was released in part as an effort to define a common API that would be implemented by web browsers. Adoption has been slow, but today it is generally accepted that DOM level 1 is supported by all the mainstream browsers.
- Level 2 was first released in 2000, and most of the API is supported by all the mainstream browsers, except Internet Explorer which implements a smaller but still significant subset of the specification. If you are developing an application targeting the browsers deployed today, using level 2 recommendations as a reference is your best bet.
- Level 3 is still a work in progress. Starting with level 2, the DOM specification has been split into a number of documents: one specification for core components, one for HTML-specific aspects of the API, one dealing with events, and so on. Only a subset of those has been published by the W3C as recommendations, while others are still works in progress. Most browsers only support a minimal subset of level 3, and Internet Explorer does not support level 3 at all. For those reasons, at this point most web developers consider that it is still too early to use the DOM level 3.