Q1. What is JavaScript?

- JavaScript was designed to add interactivity to HTML pages
- JavaScript is a scripting language
- A scripting language is a lightweight programming language
- JavaScript is usually embedded directly into HTML pages
- JavaScript is an interpreted language (means that scripts execute without preliminary compilation)
- Everyone can use JavaScript without purchasing a license

Q2. What can a JavaScript do?

- **JavaScript gives HTML designers a programming tool -** HTML authors are normally not programmers, but JavaScript is a scripting language with a very simple syntax! Almost anyone can put small "snippets" of code into their HTML pages
- **JavaScript can put dynamic text into an HTML page -** A JavaScript statement like this: document.write("<h1>" + name + "</h1>") can write a variable text into an HTML page
- **JavaScript can react to events -** A JavaScript can be set to execute when something happens, like when a page has finished loading or when a user clicks on an HTML element
- **JavaScript can read and write HTML elements -** A JavaScript can read and change the content of an HTML element
- **JavaScript can be used to validate data -** A JavaScript can be used to validate form data before it is submitted to a server. This saves the server from extra processing
- **JavaScript can be used to detect the visitor's browser** - A JavaScript can be used to detect the visitor's browser, and - depending on the browser - load another page specifically designed for that browser
- **JavaScript can be used to create cookies** - A JavaScript can be used to store and retrieve information on the visitor's computer

Q3. How To Put JavaScript into an HTML page ?

```
<html>
<body>
```

```
<script type="text/javascript">
document.write("Hello World!");
</script>
</body>
</html>
```

The example below shows how to add HTML tags to the JavaScript:

```
<html>
<body>
<script type="text/javascript">
document.write("<h1>Hello World!</h1>");
</script>
</body>
</html>
```

Q4. Explained This Example ?

To insert a JavaScript into an HTML page, we use the <script> tag. Inside the <script> tag we use the type attribute to define the scripting language.

So, the <script type="text/javascript"> and </script> tells where the JavaScript starts and ends:

```
<html>
<body>
<script type="text/javascript">
...
</script>
</body>
</html>
```

The **document.write** command is a standard JavaScript command for writing output to a page.

By entering the document.write command between the <script> and </script> tags, the browser will recognize it as a JavaScript command and execute the code line. In this case the browser will write Hello World! to the page:

```
<html>
<body>
<script type="text/javascript">
document.write("Hello World!");
</script>
```

```
</body>
</html>
```

Note: If we had not entered the <script> tag, the browser would have treated the document.write("Hello World!") command as pure text, and just write the entire line on the page. Try it yourself.

Q5. How to Handle Simple Browsers ?

Browsers that do not support JavaScript, will display JavaScript as page content.

To prevent them from doing this, and as a part of the JavaScript standard, the HTML comment tag should be used to "hide" the JavaScript.

Just add an HTML comment tag <!-- before the first JavaScript statement, and a --> (end of comment) after the last JavaScript statement, like this:

```
<html>
<body>
<script type="text/javascript">
<!--
document.write("Hello World!");
//-->
</script>
</body>
</html>
```

The two forward slashes at the end of comment line (//) is the JavaScript comment symbol. This prevents JavaScript from executing the --> tag.

JavaScripts in the body section will be executed WHILE the page loads.

JavaScripts in the head section will be executed when CALLED.

Q6.Where to Put the JavaScript ?

JavaScripts in a page will be executed immediately while the page loads into the browser. This is not always what we want. Sometimes we want to execute a script when a page loads, other times when a user triggers an event.

Scripts in <head>

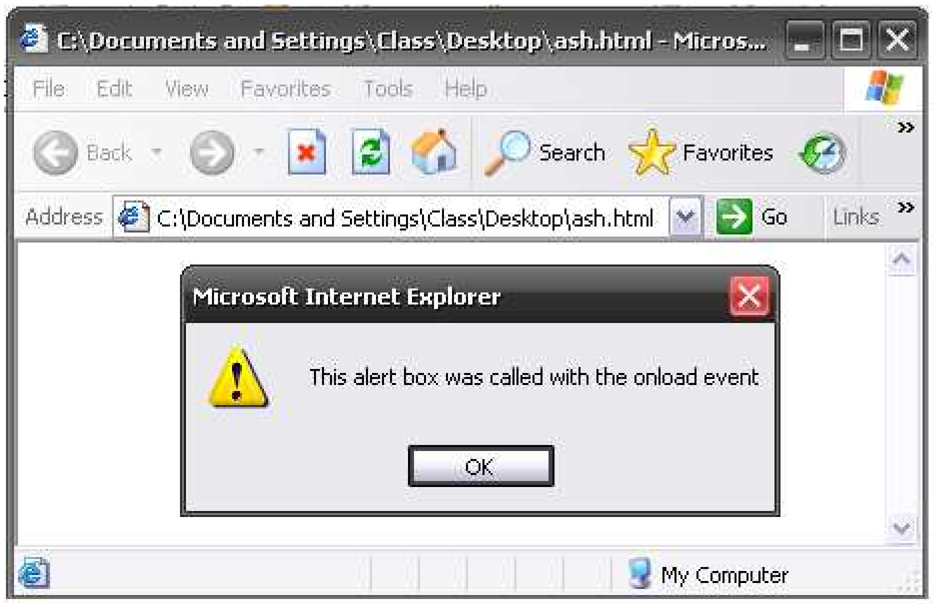
Scripts to be executed when they are called, or when an event is triggered, go in the head section.

If you place a script in the head section, you will ensure that the script is loaded before anyone uses it.

```
<html>
<head>
<script type="text/javascript">
function message()
{
alert("This alert box was called with the onload event");
}
</script>
</head>

<body onload="message()">
</body>
</html>
```

Output Window :-



C:\Documents and Settings\Class\Desktop\ash.html - Micros...
File Edit View Favorites Tools Help
Back
Search
Favorites
Address
C:\Documents and Settings\Class\Desktop\ash.html
Go
Links
Microsoft Internet Explorer
OK
My Computer

This alert box was called with the onload event

Q7. What is JavaScript Functions?

To keep the browser from executing a script when the page loads, you can put your script into a function.

A function contains code that will be executed by an event or by a call to the function.

You may call a function from anywhere within a page (or even from other pages if the function is embedded in an external .js file).

Functions can be defined both in the <head> and in the <body> section of a document. However, to assure that a function is read/loaded by the browser before it is called, it could be wise to put functions in the <head> section.

Q8. How to Define a Function?

```
function functionname(var1,var2,...,varX)
{
some code
}
```

The parameters var1, var2, etc. are variables or values passed into the function. The { and the } defines the start and end of the function.

```
<html>
<head>
<script type="text/javascript">
function displaymessage()
{
alert("Hello World!");
}
</script>
</head>

<body>
<form>
<input type="button" value="Click me!" onclick="displaymessage()" />
</form>
</body>
</html>
```

Q9. What is Events ?

Events are actions that can be detected by JavaScript.

By using JavaScript, we have the ability to create dynamic web pages. Events are actions that can be detected by JavaScript.

Every element on a web page has certain events which can trigger a JavaScript. For example, we can use the onClick event of a button element to indicate that a function will run when a user clicks on the button. We define the events in the HTML tags.

Examples of events:

- A mouse click
- A web page or an image loading
- Mousing over a hot spot on the web page
- Selecting an input field in an HTML form
- Submitting an HTML form
- A keystroke

Note: Events are normally used in combination with functions, and the function will not be executed before the event occurs!

For a complete reference of the events recognized by JavaScript, go to our complete Event reference.

onFocus, onBlur and onChange

The onFocus, onBlur and onChange events are often used in combination with validation of form fields.

Below is an example of how to use the onChange event. The checkEmail() function will be called whenever the user changes the content of the field:

```
<input type="text" size="30" id="email" onchange="checkEmail()">
```

onSubmit

The onSubmit event is used to validate ALL form fields before submitting it.

Below is an example of how to use the onSubmit event. The checkForm() function will be called when the user clicks the submit button in the form. If the field values are not accepted, the submit should be cancelled. The function checkForm() returns either true or false. If it returns true the form will be submitted, otherwise the submit will be cancelled:

```
<form method="post" action="xxx.htm" onsubmit="return checkForm()">
```

onMouseOver and onMouseOut

onMouseOver and onMouseOut are often used to create "animated" buttons.

Below is an example of an onMouseOver event. An alert box appears when an onMouseOver event is detected:

```
<a href="http://www.w3schools.com" onmouseover="alert('An onMouseOver
event');return false"><img src="w3s.gif" alt="W3Schools" /></a>
```

Q10. What is a Cookie?

A cookie is a variable that is stored on the visitor's computer. Each time the same computer requests a page with a browser, it will send the cookie too. With JavaScript, you can both create and retrieve cookie values.

Examples of cookies:

- Name cookie - The first time a visitor arrives to your web page, he or she must fill in her/his name. The name is then stored in a cookie. Next time the visitor arrives at your page, he or she could get a welcome message like "Welcome John Doe!" The name is retrieved from the stored cookie
- Password cookie - The first time a visitor arrives to your web page, he or she must fill in a password. The password is then stored in a cookie. Next time the visitor arrives at your page, the password is retrieved from the cookie

Date cookie - The first time a visitor arrives to your web page, the current date is stored in a cookie. Next time the visitor arrives at your page, he or she could get a message like "Your last visit was on Tuesday August 11, 2005!" The date is retrieved from the stored cookie

Create and Store a Cookie

In this example we will create a cookie that stores the name of a visitor. The first time a visitor arrives to the web page, he or she will be asked to fill in her/his name. The name is then stored in a cookie. The next time the visitor arrives at the same page, he or she will get welcome message.

First, we create a function that stores the name of the visitor in a cookie variable:

```
function setCookie(c_name,value,expiredays)
{
var exdate=new Date();
exdate.setDate(exdate.getDate()+expiredays);
document.cookie=c_name+ "=" +escape(value)+
((expiredays==null) ? "" : ";expires="+exdate.toGMTString());
}
```

The parameters of the function above hold the name of the cookie, the value of the cookie, and the number of days until the cookie expires.

In the function above we first convert the number of days to a valid date, then we add the number of days until the cookie should expire. After that we store the cookie name, cookie value and the expiration date in the document.cookie object.

Then, we create another function that checks if the cookie has been set:

```
function getCookie(c_name)
{
if (document.cookie.length>0)
  {
  c_start=document.cookie.indexOf(c_name + "=");
  if (c_start!=-1)
    {
    c_start=c_start + c_name.length+1;
    c_end=document.cookie.indexOf(";",c_start);
    if (c_end==-1) c_end=document.cookie.length;
    return unescape(document.cookie.substring(c_start,c_end));
    }
  }
return "";
}
```

The function above first checks if a cookie is stored at all in the document.cookie object. If the document.cookie object holds some cookies, then check to see if our specific cookie is stored. If our cookie is found, then return the value, if not - return an empty string.

Last, we create the function that displays a welcome message if the cookie is set, and if the cookie is not set it will display a prompt box, asking for the name of the user:

```
function checkCookie()
{
username=getCookie('username');
if (username!=null && username!="")
  {
  alert('Welcome again '+username+'!');
  }
else
  {
  username=prompt('Please enter your name:',"");
  if (username!=null && username!="")
    {
    setCookie('username',username,365);
    }
  }
}
```

Example :

```
<html>
<head>
<script type="text/javascript">
function getCookie(c_name)
{
if (document.cookie.length>0)
  {
  c_start=document.cookie.indexOf(c_name + "=");
  if (c_start!=-1)
    {
    c_start=c_start + c_name.length+1;
    c_end=document.cookie.indexOf(";",c_start);
    if (c_end==-1) c_end=document.cookie.length;
    return unescape(document.cookie.substring(c_start,c_end));
    }
  }
return "";
}

function setCookie(c_name,value,expiredays)
{
var exdate=new Date();
exdate.setDate(exdate.getDate()+expiredays);
document.cookie=c_name+ "=" +escape(value)+
((expiredays==null) ? "" : ";expires="+exdate.toGMTString());
}

function checkCookie()
```

```
{
username=getCookie('username');
if (username!=null && username!="")
  {
  alert('Welcome again '+username+'!');
  }
else
  {
  username=prompt('Please enter your name:',"");
  if (username!=null && username!="")
    {
    setCookie('username',username,365);
    }
  }
}
</script>
</head>

<body onload="checkCookie()">
</body>
</html>
```

Q11.How to save state with cookies?

Maintaining state means remembering information while the user moves from page to page within a Web site. With this information in hand, you can set user preferences, fill in default form values track visit counts, and do many other thing that made browsing easier for users and that give you more information about how your pages are used.

You can maintain state information in a number of ways:

- ➢ Store it in cookies
- ➢ Encode it in URL links
- ➢ Send it in hidden from variables
- ➢ Store it in variables in other frames
- ➢ Store it on the Web server

Q12. Explain document object model (DOM).

1. The HTML DOM is a W3C standard and it is an abbreviation for the Document Object Model for HTML.
2. The HTML DOM defines a standard set of objects for HTML, and a standard way to access and manipulate HTML documents.
3. All HTML elements, along with their containing text and attributes, can be accessed through the DOM. The contents can be modified or deleted, and new elements can be created.
4. The HTML DOM is platform and language independent. It can be used by any programming language like Java, JavaScript, and VBScript.

Q13. Explain form tag in detail.

i. A form is an area that can contain form elements.
ii. Form elements are elements that allow the user to enter information (like text fields, text area fields, drop-down menus, radio buttons, checkboxes, etc.) in a form.
iii. A form is defined with the <form> tag.

General format

```
<form>
.
input elements
.
</form>
```

Input

The most used form tag is the <input> tag. The type of input is specified with the type attribute. The most commonly used input types are explained below.

Text Fields

Text fields are used when you want the user to type letters, numbers, etc. in a form.

```
<form>
First name:
<input type="text" name="firstname" />
<br />
```

```
Last name:
<input type="text" name="lastname" />
</form>
```

How it looks in a browser:

First name:

Last name:

Radio Buttons

Radio Buttons are used when you want the user to select one of a limited number of choices.

only one option can be chosen.

```
<form>
<input type="radio" name="sex" value="male" /> Male
<br />
<input type="radio" name="sex" value="female" /> Female
</form>
```

How it looks in a browser:

○ Male

○ Female

Checkboxes

Checkboxes are used when you want the user to select one or more options of a limited number of choices.

```
<form>
I have a bike:
<input type="checkbox" name="vehicle" value="Bike" />
<br />
I have a car:
```

```
<input type="checkbox" name="vehicle" value="Car" />
<br />
I have an airplane:
<input type="checkbox" name="vehicle" value="Airplane" />
</form>
```

How it looks in a browser:

I have a bike: ☐

I have a car: ☐

I have an airplane: ☐

The Form's Action Attribute and the Submit Button

i. When the user clicks on the "Submit" button, the content of the form is sent to the server.
ii. The form's action attribute defines the name of the file to send the content to.
iii. The file defined in the action attribute usually does something with the received input.

Code:

```
<form name="input" action="html_form_submit.asp" method="get">
Username:
<input type="text" name="user" />
<input type="submit" value="Submit" />
</form>
```

How it looks in a browser:

Username:

If we type some characters in the text field above, and click the "Submit" button, the browser will send your input to a page called "html_form_submit.asp". The page will show you the received input.