Characteristics of scripting languages

Nature of the JavaScript language

The JavaScript language is object oriented and event driven. An object oriented language is where the code is broken into objects or pieces and each object has its own properties. This means that each object within a line or lines of code knows about itself and knows what it is able to do, therefore knowing what it can and cannot interact with.

With event driven languages, the code is split into events rather than objects. This means that the code is only executed when a particular event has been triggered, for example an event could be a mouse click or a press of a key on the keyboard and once the user has performed this event, the code is triggered and will run an action it is programmed to depending on what the event was.

Event Handlers

Event handlers are functions that enable events to be executed. These functions have program statements embedded into them as it is a software routine that is able to process actions. An event handler would be triggered when the actual event is performed, for example if a user clicks a button, the event handler ‘onClick’ would be triggered and would then allow certain code to be executed which is related to the event, for example an alert box would appear after the button (onclick) has been clicked by the user.

Hiding Scripts

Older browsers are not compatible with JavaScript and cannot recognise the scripting language or may cause errors in older web browsers due to confusion as an older browser would try to interpret the scripting language as HTML. This can be avoided by including the following code (shown below in red bold text) within the script tags.

```html
<script>
<!--
Script language is written here...
//-->
</script>
```

Older web browsers would be able to read the code above in red text only and successfully ignore everything else as they are not compatible with it and know to ignore it due to the tags in red. However, newer and dated browsers will ignore the compatibility tags and just read the script tags, meaning the JavaScript language would be able to be read successfully.
Security Issues

With JavaScript, there are many security issues that can form threats which can manipulate or even hack client’s computers. This is because scripting language or code is being executed on a user’s computer which is able to read and write from a client’s computer system. A common example of annoying JavaScript is the use of pop ups, if a user visits an untrusted website, they may click a button which executes JavaScript and alert boxes appear in a number of loops. Hackers can also perform malicious actions such as data mining, which is the process of extracting large amounts of data from a user’s computer system. Hackers can also open applications or read other browser windows, hence stealing information.

However, these threats are limited as the designers of JavaScript ensured that with the use of HTML and JavaScript alone, it is not possible to be able to access the local file system of a user’s computer system, therefore limiting the threats or damage that could be caused.

Including Scripts inside HTML

Including Scripts inside HTML is possible with the use of the script tag. The tag ‘<script>’ allows the browser and HTML know that after this tag, a scripting language will be written. The tag ‘</script>’ allows the browser and HTML know that this has ended the section of scripting and normal HTML tags will now be recognised. The script tags can also be used anywhere within the HTML tags and do not have to be after particular tags, for example JavaScript can be embedded after or before the body tag in HTML, unlike internal CSS where it must be within the header tag in HTML. An example of how scripts can be written in HTML is shown below. Note that just after the script tag, the type of scripting language (JavaScript, Python, etc.) has been declared, in this case it is JavaScript.

```html
<html>
<head>
</head>
<body>
<script type=text/javascript”>
    document.write ("This is JavaScript inside HTML!");
</script>
</body>
</html>
```
Scripting Language Constructs

Variables are used to store and retrieve data from a computer’s RAM (Random Access Memory). Variables must have a unique name and must not be the same name as others in different case letters, for example the variable ‘x’ and ‘X’ would be different as JavaScript is case sensitive. There are three different types of operators known as arithmetic operators, logical operators and relational operators.

Arithmetic Operators

Arithmetic operators are essential to be able to create arithmetic expressions such as calculating simple sums using numbers.

Logical Operators

Logical operators are used to define logic between two or more variables and values. The following logical operators are used to define logic.

&& - AND

|| – OR

! – Not

Relational Operators

These types of operators are used to compare values to one another. The comparisons result in two expressions which are true or false.

<table>
<thead>
<tr>
<th>Relational Operator Symbol</th>
<th>JavaScript Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>Equal to (for values such as numbers)</td>
</tr>
<tr>
<td>===</td>
<td>Equal to (for values and data types)</td>
</tr>
<tr>
<td>!=</td>
<td>Not equal to</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
</tr>
</tbody>
</table>

Assignment

When setting values with JavaScript, it is possible to assign the variable a value. For example, it is possible to set a variable known as age a value, such as a number.
**Dot operator**

The dot operator is used to as a separation technique which separates objects from their properties or methods. The example below shows a simple calculation with JavaScript using variables, arithmetic operators, a dot operator and assignment.

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

    function addNumbers (a, b) { //Setting a function with two parameters: 'a and b'
        var c = a+b; //the variable 'c' is assigned to the calculation of a+b (note the '+' is an arithmetic operator
        return c; //Return the result of the calculation of 'a' and 'b' in the text area below
    }

    document.write (addNumbers (3, 6)); //calling the function and setting two numbers to add, in which the result should be 9

    //document.write is a dot operator and displays any text in the brackets or the result, in this case the result of the function called (the number 9)

</script>
</body>
</html>
```
Loops

Loops allow actions happen repeatedly and do not end until an action has been successfully performed, this when they are told to stop by JavaScript code by using a conditional statement. An example of this is with the use of relational operators and variables as shown below.

```html
<html>
<body>
<script type=text/javascript">
counter = 1; //counter starts at the number 1

do
{
    document.write ("The number is " + counter + "<br>"); //display the text "the number is" and the actual number in loop with the counter.

    //An example of this is as follows:
    //The number is 1
    //The number is 2
    //and so on...
    //This is displayed in a new line each time due to the <br> tag
    counter++;
}
while (counter <= 10); //the loop will be repeated while the counter is less than or equal to the number 10

//When this condition is no longer true (number reaching 11 or above), the counter will stop, hence stopping the loop
</script>
</body>
</html>
```
The example above shows a ‘do…while’ loop and a conditional statement has been made so that a counter is repeated while the counter number is less than or equal to 10. This means that until the counter has reached the number 10, it will not stop displaying numbers and will increase by 1 from the number 1. This was defined in the beginning with the line of code `counter = 1`. In the JavaScript code, when the counter reaches 11, it will not display this on the browser as the conditional statement will no longer be true.

**Decision Making**

‘IF’ statements, or decision making code follow a particular set of rules when being executed. These statements are only executed if an ‘IF’ statement is true or false. An example of an IF statement is shown below.

```html
<html>
<head>
</head>
<body>
<script type=text/javascript”>
    var x = 24; // Setting a variable as 24
    var y = 56; // Setting a variable as 56

    if(x<=y) { // if x (24) is less than y (56)
        document.write ("This 'if' statement works!"); // then the following sentence will be displayed on the browser window
    }

</script>
</body>
</html>
```
As you can see, the statement shows that if the variable ‘x’ which is equal to 24 is less than the variable ‘y’ which is equal to 56, then the text "This 'if' statement works!" will be displayed on the web browser. In this case, the ‘IF’ statement is true as 24 is less than 56, meaning the text will successfully display on the browser. An ‘IF else’ statement is optional, if for example there must be an action for the ‘false’ part of a statement, then one would create an 'IF else' statement and declare another action to be executed.

**Functions**

Functions are pieces of code that can be written separately or within the ‘<script>’ tags and can be called or executed whenever programmed to. Functions are never executed unless they are called. Below is an example of a function being declared within the ‘<script>’ tags and being called.

```html
<html>
<head>
</head>
<body>
<script type="text/javascript">
function characters (one, two) { // name of the function and declaring two values within the function parameter.
    document.write(one + " is better than " + two + "<br/>") // a sentence in which the functions can be executed into.
}
characters ("Thor", "Loki"); // the function is called here and the values "one" and "two" are replaced with "Thor" and "Loki" automatically.
characters ("Batman", "The Joker");
characters ("Superman", "Lex Luthor");
</script>
</body>
</html>
```
The name of the function is ‘characters’ and there are two simple values passed into the function parameter, meaning the function will use these values while it is running. The function is called after the braces, by declaring ‘characters’ each time. As you can see, “Thor” and “Loki” replace the values “one” and “two” automatically and are implemented into the ‘write’ method to form the sentence; “Thor is better than Loki”. The same method applies to the next two times the function is called.

Functions can also be used to handle events, for example, a function can be used to create an alert box which alerts the user that something has happened when clicking a button. An example of this is shown below.

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

function SubmitForm () { // function has been named.
    alert (“Your details have been saved!”); // an alert box will be created and include the text declared.
}
</script>

<form>
<input type=”button” value=”Submit” onclick=”SubmitForm ()”>

</form>
</body>
</html>

The code above shows how a function has been made and called that creates an alert box with the ‘onclick’ event handler within a form. After the user clicks the ‘Submit’ button, an alert box will appear as the function will execute by this event.