Institute for Advanced Computing And Software Development (IACSD) Akurdi, Pune

Java Script

Dr. D.Y. Patil Educational Complex, Sector 29, Behind Akurdi Railway Station,

Nigdi Pradhikaran, Akurdi, Pune - 411044.

JavaScript

History
JavaScript was designed to 'plug a gap' in the techniques available for creating web-pages.

HTML is relatively easy to learn, but it is static. It allows the use of links to load new pages, images, sounds, etc., but it provides very little support for any other type of interactivity.

To create dynamic material it was necessary to use either:

CGI (Common Gateway Interface) programs

Can be used to provide a wide range of interactive features, but...

Run on the server, i.e.:

A user-action causes a request to be sent over the internet from the client machine to the server.

The server runs a CGI program that generates a new page, based on the information supplied by the client.

The new page is sent back to the client machine and is loaded in place of the previous page.

Thus every change requires communication back and forth across the internet.

Written in languages such as Perl, which are relatively difficult to learn.

Java applets

Run on the client, so there is no need to send information back and forth over the internet for every change, but...

Written in Java, which is relatively difficult to learn.

Netscape Corporation set out to develop a language that:

Provides dynamic facilities similar to those available using CGI programs and Java applets.

Runs on the Client.

Is relatively easy to learn and use.

They came up with LiveScript.

Netscape subsequently teamed-up with Sun Microsystems (the company that developed Java) and produced JavaScript.

Javascript only runs on Netscape browsers (e.g., Netscape Navigator). However, Microsoft soon developed a version of JavaScript for their Internet Explorer browser. It is called JScript. The two languages are almost identical, although there are some minor differences.

Internet browsers such as Internet Explorer and Netscape Navigator provide a range of features that can be controlled using a suitable program. For example, windows can be opened and closed, items can be moved around the page, colours can be changed, information can be read or modified, etc..

However, in order to do this you need to know what items the browser contains, what operations can be carried out on each item, and the format of the necessary commands.

Therefore, in order to program internet browsers, you need to know:

How to program in a suitable language (e.g., Javascript/JScript)

The internal structure of the browser.

In this course we will be using JavaScript/JScript to program browsers. However, there are several other languages we could use should we wish to. Therefore, we shall try to distinguish clearly between those aspects of internet programming which are specific to JavaScript/JScript and those which remain the same regardles of which language we choose to use.

We'll start by looking at some of the basic features of the JavaScript language.

Variables & Literals

A variable is a container which has a name. We use variables to hold information that may change from one moment to the next while a program is running.

For example, a shopping website might use a variable called total to hold the total cost of the goods the customer has selected. The amount stored in this variable may change as the customer adds more goods or discards earlier choices, but the name total stays the same. Therefore we can find out the current total cost at any time by asking the program to tell us what is currently stored in total.

A literal, by contrast, doesn't have a name - it only has a value.

For example, we might use a literal to store the VAT rate, since this doesn't change very often. The literal would have a value of (e.g.) 0.21. We could then obtain the final cost to the customer in the following way:

VAT is equal to total x 0.21

final total is equal to total + VAT

JavaScript accepts the following types of variables:

Numeric Any numeric value, whether a whole number (an integer) or a number that includes a fractional part (a real), e.g.,
12
3.14159
etc.

String A group of text characters, e.g.,
Ian
Macintosh G4
etc.

Boolean A value which can only be either True or False, e.g.
completed
married
etc.

We create variables and assign values to them in the following way:

```
var christianName = "Fred"   (string)
var surname = "Jones"        (string)
var age = 37   (numeric)
var married = false    (Boolean)
```

Note that:

When a new variable is created (or declared) its name must be preceded by the word var

The type of the variable is determined by the way it is declared:

if it is enclosed within quotes, it's a string

if it is set to true or false (without quotes) it's a boolean

if it is a number (without quotes) it's numeric

We refer to the equals sign as the assignment operator because we use it to assign values to variables;

Variable names must begin with a letter or an underscore

Variable names must not include spaces

JavaScript is case-sensitive

Reserved words (i.e., words which indicate an action or operation in JavaScript) cannot be used as variable names.

Operators
Operators are a type of command. They perform operations on variables and/or literals and produce a result.

JavaScript understands the following operators:

+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus

(If you're not sure what a modulus operator does, here are some notes and an example)

These are known as binary operators because they require two values as input, i.e.:

4 + 3

7 / 2

15 % 4

In addition, JavaScript understands the following operators:

+ +	Increment	Increase value by 1
- -	Decrement	Decrease value by 1

-	Negation	Convert positive to negative, or vice versa

These are known as unary operators because they require only one value as input, i.e.:

4++	increase 4 by 1 so it becomes 5
7--	decrease 7 by 1 so it becomes 6
-5	negate 5 so it becomes -5

JavaScript operators are used in the following way:

```
var totalStudents = 60
var examPasses = 56
var resits = totalStudents - examPasses
```

Note that by carrying out this operation we have created a new variable - resits.

There is no need to declare this variable in advance.

It will be a numeric value because it has been created as a result of an operation performed on two numeric values.

We can also combine these operators (and the assignment operator, =) in certain ways.

For example:

```
total += price
```

performs the same task as:

```
total = total + price
```

Similarly,

```
total *= quantity
```

performs the same task as:

```
total = total * quantity
```

Below is a full list of these 'combined' operators.

+ =	'becomes equal to itself plus'
- =	'becomes equal to itself minus'
* =	'becomes equal to itself multiplied by'
/ =	'becomes equal to itself divided by'
% =	'becomes equal to the amount which is left when it is divided by'

You may find the descriptions helpful when trying to remember what each operator does. For example:

5 * = 3
can be thought of as meaning:
5 becomes equal to itself multiplied by 3
Part 2
Functions in JavaScript
In JavaScript, as in other languages, we can create functions. A function is a kind of mini-program that forms part of a larger program.

Functions:

consist of one or more statements (i.e., lines of program code that perform some operation)

are separated in some way from the rest of the program, for example, by being enclosed in curly brackets, {.....}

are given a unique name, so that they can be called from elsewhere in the program.

Functions are used:

Where the same operation has to be performed many times within a program.

Rather than writing the same code again and again, it can be written once as a function and used repeatedly. For example:

request_confirmation_from_user

To make it easier for someone else to understand your program.

Rather than writing long, rambling programs in which every single operation is listed in turn, it is usually better to divide programs up into small groups of related operations. For example:

set_variables_to_initial_values

welcome_user

obtain_user_input

perform_calculations

display_results

In JavaScript, functions are created in the following way:

```
function name()
{
        statement;
        statement;
        statement
}
```

Note that all the statements except the last statement must be followed by semi-colons. The last statement doesn't need one, but if you do put a semi-colon after the last statement it won't cause any problems.

Here is an example of a simple function:

```
function initialiseVariables()
{
        itemsSold = 0;
        nettPrice = 0;
        priceAfterTax = 0
}
```

When called, this function will set the three variables itemsSold, nettPrice and priceAfterTax to zero.

To run this function from somewhere else in a program, we would simply call it by name, e.g.:

```
initialiseVariables();
```

Note that the name must be followed by a pair of brackets. The purpose of these will become clear later.

Functions can be called from within other functions.

For example:

```
function sayGoodbye()
{
        alert("Goodbye!")
}

function sayHello()
{
```

```
    alert("Hi, there!");
    sayGoodbye()
}
```

When the function sayHello() is called, it first displays an alert on the screen. An alert is simply box containing some text and an 'OK' button that the user can press to make the box disappear when the text has been read. In this case the box will contain the words "Hi, there!".

The sayHello() function then calls the function sayGoodbye(), which posts another alert saying "Goodbye".

Click here to see this example working.

Note that the function sayGoodbye() is written first. Browsers interpet JavaScript code line-by-line, starting at the top, and some browsers will report an error if they find a reference to a function before they find the function itself. Therefore functions should be declared before the point in the program where they are used.

Passing Parameters to Functions

Some functions perform a simple task for which no extra information is needed.

However, it is often necessary to supply information to a function so that it can carry out its task.

For example, if we want to create a function which adds VAT to a price, we would have to tell the function what the price is.

To do this we would pass the price into the function as a parameter. Parameters are listed in between the brackets that follow the function name. For example:

```
function name(parameter_1, parameter_2)
{
    statement(s);
}
```

In this case two parameters are used, but it's possible to use more than this if necessary. The additional parameter names would simply be added on to the list of parameters inside the brackets, separated from one another by commas. It's also possible to use just one prameter if that's all that is needed.

Here's an example of a simple function that accepts a single parameter:

```
function addVAT(price)
{
    price *= 1.21;
```



```
    alert(price)
}
```

This function accepts a parameter called price, multiplies it by 1.21 (i.e., adds an extra 21% to it), and then displays the new value in an alert box.

We would call this function in the following way:

```
addVAT(nettPrice)
```

The parameter nettPrice could be either:

a literal - for example:

```
addVAT(5)
```

a variable - for example:

```
var nettPrice = 5;
addVAT(nettPrice)
```

Returning values from Functions

Sometimes we also need to get some information back from a function.

For example, we might want to add VAT to a price and then, instead of just displaying the result, pass it back to the user or display it in a table.

To get information back from a function we do the following:

```
function addVAT(price)
{
    price *= 1.21;
    return price
}
```

To call this function we would do the following:

```
var newPrice = addVAT(nettPrice)
```

The value returned by the funtion will be stored in the variable newPrice. Therefore this funtion will have the effect of making newPrice equal to nettPrice multiplied by 1.21.

Part 3

JavaScript Comparison Operators
As well as needing to assign values to variables, we sometime need to compare variables or literals.

We do this using Comparison Operators.

Comparison Operators compare two values and produce an output which is either true or false.

For example, suppose we have two variables which are both numeric (i.e., they both hold numbers):

examPasses
and

totalStudents

If we compare them, there are two possible outcomes:

They have the same value

They do not have the same value

Therefore, we can make statements like these:

They are the same

They are the different

The first is larger than the second

The first is smaller than the second

... and then perform a comparison to determine whether the statement is true or false.

The basic comparison operator is:

==
(i.e., two equals signs, one after the other with no space in between).

It means 'is equal to'. Compare this with the assignment operator, =, which means 'becomes equal to'. The assignment operator makes two things equal to one another, the comparison operator tests to see if they are already equal to one another.

Here's an example showing how the comparison operator might be used:

examPasses == totalStudents

If examPasses and totalStudents have the same value, the comparison would return true as a result.

If examPasses and totalStudents have different values, the comparison would return false as a result.

Another comparison operator is:

!=
(i.e., an exclamation mark followed by an equals sign, with no space in between).

It means 'is NOT equal to'.

For example:

examPasses != totalStudents

If examPasses and totalStudents have the same value, the comparison would return false as a result.

If examPasses and totalStudents have different values, the comparison would return true as a result.

Two other commonly-used comparison operators are:

<
and

>

< means 'less than'

> means 'greater than'

For example:

examPasses < totalStudents

If examPasses is less than totalStudents, the comparison would return true as a result.

If examPasses is more than totalStudents, the comparison would return false as a result.

Another example:

examPasses > totalStudents

If examPasses is more than totalStudents, the comparison would return true as a result.

If examPasses is less than totalStudents, the comparison would return false as a result.

As with some of the other 0perators we have encountered, comparison operators can be combined in various ways.

<=
means

'less than or equal to'.

For example:

examPasses <= totalStudents

If examPasses is less than or equal to totalStudents, the comparison would return true as a result.

If examPasses is more than totalStudents, the comparison would return false as a result.

Also:

>=
means

'greater than or equal to'.

For example:

examPasses >= totalStudents

If examPasses is more than or equal to totalStudents>, the comparison would return true as a result.

If examPasses is less than totalStudents, the comparison would return false as a result.

To summarise, JavaScript understands the following comparison operators:

==	'is equal to'
!=	'is NOT equal to'
<	'is less than'
>	'is greater than'
<=	'is less than or equal to'
>=	'is greater than or equal to'

If-Else Statements in JavaScript

Much of the power of programming languages comes from their ability to respond in different ways depending upon the data they are given.

Thus all programming languages include statements which make 'decisions' based upon data.

One form of decision-making statement is the If...Else statement.

It allows us to make decisions such as:

If I have more than £15 left in my account, I'll go to the cinema.

Otherwise I'll stay at home and watch television.

This might be expressed in logical terms as:

If (money > 15) go_to_cinema

Else watch_television

The If-Else statement in JavaScript has the following syntax:

```
if (condition)
{
        statement;
        statement
}
else
{
        statement;
        statement
};
```

The condition is the information on which we are basing the decision. In the example above, the condition would be whether we have more than £15. If the condition is true, the browser will carry out

the statements within the if... section; if the condition is false it will carry out the statements within the else... section.

The if... part of the statement can be used on its own if required. For example:

```
if (condition)
{
        statement;
        statement
};
```

Note the positioning of the semi-colons.

If you are using both the if... and the else... parts of the statement, it is important NOT to put a semi-colon at the end of the if... part. If you do, the else... part of the statement will never be used.

A semi-colon is normally placed at the very end of the if...else... statement, although this is not needed if it is the last or only statement in a function.

A practical If-Else statement in JavaScript might look like this:

```
if (score > 5)
{
        alert("Congratulations!")
}
else
{
        alert("Shame - better luck next time")
};
```

The for Loop

A for loop allows you to carry out a particular operation a fixed number of times.

The for loop is controlled by setting three values:

- an initial value
- a final value
- an increment

The format of a for loop looks like this:

```
for (initial_value; final_value; increment)
```



```
{
        statement(s);
}
```

A practical for loop might look like this:

```
for (x = 0; x <= 100; x++)
{
        statement(s);
}
```

Note that:

* The loop condition is tested using a variable, x, which is initially set to the start value (0) and then incremented until it reaches the final value (100).

* The variable may be either incremented or decremented.

* The central part of the condition, the part specifying the final value, must remain true throughout the required range. In other words, you could not use x = 100 in the for loop above because then the condition would only be true when x was either 0 or 100, and not for all the values in between. Instead you should use x <= 100 so that the condition remains true for all the values between 0 and 100.

Here are some practical examples of for loops.

The first example is a simple loop in which a value is incremented from 0 to 5, and reported to the screen each time it changes using an alert box. The code for this example is:

```
for (x = 0; x <= 5; x++)
{
        alert('x = ' + x);
}
```

Click here to see this example working.

The second example is the same except that the value is decremented from 5 to 0 rather than incremented from 0 to 5. The code for this example is:

```
for (x = 5; x >= 0; x--)
{
        alert('x = ' + x);
}
```

Click here to see this example working.

The start and finish values for the loop must be known before the loop starts.

However, they need not be written into the program; they can, if necessary, be obtained when the program is run.

For example:

```
var initialValue = prompt("Please enter initial value", "");
var finalValue = prompt("Please enter final value", "");

for (x = initialValue; x <= finalValue; x++)
{
        statement(s);
}
```

In this example, the user is prompted to type-in two numbers which are then assigned to the variables initialValue and finalValue. The loop then increments from initialValue to finalValue in exactly the same way as if these values had been written directly into the program.

The While Loop

Like the for loop, the while loop allows you to carry out a particular operation a number of times.

The format of a while loop is as follows:

```
while (condition)
{
        statement(s);
}
```

A practical while loop might look like this:

```
var x = 500000;

alert("Starting countdown...");

while (x > 0)
{
        x--;
};
alert("Finished!");
```


In this example, x is initially set to a high value (500,000). It is then reduced by one each time through the loop using the decrement operator (x--). So long as x is greater than zero the loop will continue to operate, but as soon as x reaches zero the loop condition (x > 0) will cease to be true and the loop will end.

The effect of this piece of code is to create a delay which will last for as long as it takes the computer to count down from 500,000 to 0. Before the loop begins, an 'alert' dialog-box is displayed with the message "Starting Countdown...". When the user clicks the 'OK' button on the dialog-box the loop will begin, and as soon as it finishes another dialog-box will be displayed saying "Finished!". The period between the first dialog box disappearing and the second one appearing is the time it takes the computer to count down from 500,000 to 0.

To see this example working, click here.

The principal difference between for loops and while loops is:

with a while loop, the number of times the loop is to be executed need not be known in advance.

while loops are normally used where an operation must be carried out repeatedly until a particular situation arises.

For example:

```
var passwordNotVerified = true;

while (passwordNotVerified == true)
{
        var input = prompt("Please enter your password", "");
        if (input == password)
        {
                passwordNotVerified = false;
        }
        else
        {
                alert("Invalid password - try again")
        }
}
```

In this example, the variable passwordNotVerified is initially set to the Boolean value true. The user is then prompted to enter a password, and this password is compared with the correct password stored in the variable called password. If the password entered by the user matches the stored password, the variable passwordNotVerified is set to false and the while loop ends. If the password entered by the user does not match the stored password, the variable passwordNotVerified remains set to true and a warning message is displayed, after which the loop repeats.

To try this piece of code, click here.

PS The password is CS7000 - and don't forget that the 'CS' must be capitalised.

Testing Boolean Variables
In the while loop example above we used the line:

```
var passwordNotVerified = true;
```

and then tested this variable in a conditional statement as follows:

```
while (passwordNotVerified == true)
```

We could also have written the conditional statement like this:

```
while(passwordNotVerified)
```

In other words, if we don't specify true or false in a conditional statement, the JavaScript interpreter will assume we mean true and test the variable accordingly.

This allows us to make our code a little shorter and, more importantly, to make it easier for others to understand. The line:

```
while(passwordNotVerified)
```

is much closer to the way in which such a condition might be expressed in English than:

```
while(passwordNotVerified == true)
```

Logical Operators
We have met a number of operators that can be used when testing conditions, e.g., == , < , > , <= , >= .

Two more operators that are particularly useful with while loops are:

&& Logical AND

|| Logical OR

These operators are used to combine the results of other conditional tests.

For example:


```
if (x > 0 && x < 100)
```

means...
if x is greater than 0 and less than 100...

Placing the && between the two conditions means that the if statement will only be carried out if BOTH conditions are true. If only one of the conditions is true (e.g., x is greater than 0 but also greater than 100) the condition will return false and the if statement won't be carried out.
Similarly:

```
if (x == 0 || x == 1)
```

means...
if x is 0 or x is 1...

Placing the || between the two conditions means that the if statement will be executed if EITHER of the conditions are true.

The ability to combine conditions in this way can be very useful when setting the conditions for while loops.

For example:

```
var amount = prompt ("Please enter a number between 1 and 9", "");

while (amount < 1 || amount > 9)
{
    alert("Number must be between 1 and 9");
    amount = prompt ("Please enter a number between 1 and 9", "");
}
```

In this example, the variable amount is initially set to the value typed-in by the user in response to the 'prompt' dialog-box. If the amount entered by the user is between 1 and 9, the loop condition becomes false and the loop ends. If the amount entered by the user is less than 1 or greater than 9, the loop condition remains true and a warning is displayed, after which the user is prompted to enter another value.

To try this piece of code, click here.

Note that if you enter a correct value immediately, the while loop never executes. The condition is false the first time it is tested, so the loop never begins.

Part 4

JavaScript: Window, Document & Form Objects
JavaScript is an object-oriented (or, as some would argue, object-based) language.

An object is a set of variables, functions, etc., that are in some way related. They are grouped together and given a name

Objects may have:

Properties

A variable (numeric, string or Boolean) associated with an object. Most properties can be changed by the user.

Example: the title of a document

Methods

Functions associated with an object. Can be called by the user.

Example: the alert() method

Events

Notification that a particular event has occurred. Can be used by the programmer to trigger responses.

Example: the onClick() event.

Internet browsers contain many objects. In the last few years the object structure of internet browsers has become standardised, making programming easier. Prior to this, browsers from different manufacturers had different object structures. Unfortunately, many such browsers are still in use.

The objects are arranged into a hierarchy as shown below:

The Document Object Model. Objects shown in green are common to both Netscape Navigator and Internet Explorer; objects shown in yellow are found only in Internet Explorer while objects shown in blue are found only in Netscape Navigator.
The hierarchy of objects is known as the Document Object Model (DOM).

The Window Object

Window is the fundamental object in the browser. It represents the browser window in which the document appears

Its properties include:

status The contents of the status bar (at the bottom of the browser window). For example:

window.status = "Hi, there!";

will display the string "Hi, there!" on the status bar.
Click here to see this line of code in operation.

location The location and URL of the document currently loaded into the window (as displayed in the location bar). For example:

alert(window.location);

will display an alert containing the location and URL of this document.
Click here to see this line of code in operation.

length The number of frames (if any) into which the current window is divided. For example:

alert(window.length);

will display an alert indicating the number of frames in the current window.

See under parent (below) for an example.
parent The parent window, if the current window is a sub-window in a frameset. For example:

```
var parentWindow = window.parent;
alert(parentWindow.length);
```

will place a string representing the parent window into the variable parentWindow, then use it to report the number of frames (if any) in the parent window.

Click here to see an example of the use of parent and length.

top The top-level window, of which all other windows are sub-windows. For example:

```
var topWindow = window.top;
alert(topWindow.length);
```

will place a string representing the top-level window into the variable topWindow, then use it to report the number of frames (if any) in the top-level window.

top behaves in a very similar way to parent. Where there are only two levels of windows, top and parent will both indicate the same window. However, if there are more than two levels of windows, parent will indicate the parent of the current window, which may vary depending upon which window the code is in. However, top will always indicate the very top-level window.

Window methods include:

alert() Displays an 'alert' dialog box, containing text entered by the page designer, and an 'OK' button. For example:

```
alert("Hi, there!");
```

will display a dialog box containing the message "Hi, there!".
Click here to see this line of code in operation.

confirm() Displays a 'confirm' dialog box, containing text entered by the user, an 'OK' button, and a 'Cancel' button. Returns true or false. For example:

```
var response = confirm("Delete File?");
alert(response);
```

will display a dialog box containing the message "Delete File?" along with an 'OK' button and a 'Cancel' button. If the user clicks on 'OK' the variable response will contain the Boolean value true, and this will appear in the 'alert' dialog-box; If the user clicks on 'Cancel' the variable response will contain the Boolean value false and this will appear in the 'alert' dialog-box.
Click here to see this code in operation.

prompt() Displays a message, a box into which the user can type text, an 'OK' button, and a 'Cancel' button. Returns a text string. The syntax is:

```
prompt(message_string, default_response_string)
```

For example:

```
var fileName = prompt("Select File", "file.txt");
alert(fileName);
```

will display a dialog box containing the message "Select File" along with an 'OK' button, a 'Cancel' button, and an area into which the user can type. This area will contain the string "file.txt", but this can be overwritten with a new name. If the user clicks on 'OK' the variable fileName will contain the string "file.txt" or whatever the user entered in its place, and this will be reported using an alert dialog box.
Click here to see this code in operation.

open() Opens a new browser window and loads either an existing page or a new document into it. The syntax is:

```
open(URL_string, name_string, parameter_string)
```

For example:

```
var parameters = "height=100,width=200";
newWindow = open("05_JS4nw.html", "newDocument", parameters);
```

will open a new window 100 pixels high by 200 pixels wide. An HTML document called '05_JS4nw.html' will be loaded into this window.

Note that the variable newWindow is not preceded by the word var. This is because newWindow is a global variable which was declared at the start of the script. The reason for this is explained below.

Click here to see this code in operation.

close() Closes a window. If no window is specified, closes the current window. The syntax is:

```
window_name.close()
```

For example:

```
newWindow.close()
```

will close the new window opened by the previous example.

Note that the window name (i.e., newWindow) must be declared as a global variable if we want to open the window using one function and close it using another function. If it had been declared as a local variable, it would be lost from the computer's memory as soon as the first function ended, and we would not then be able to use it to close the window.

Click here to see this code in operation.

Window events include:

onLoad() Message sent each time a document is loaded into a window. Can be used to trigger actions (e.g., calling a function). Usually placed within the <body> tag, for example:

```
<body onLoad="displayWelcome()">
```

would cause the function displayWelcome() to execute automatically every time the document is loaded or refreshed.

onUnload() Message sent each time a document is closed or replaced with another document. Can be used to trigger actions (e.g., calling a function). Usually placed within the <body> tag, for example:

<body onUnload="displayFarewell()">

would cause the function displayFarewell() to execute automatically every time the document is closed or refreshed.

The Document Object

The Document object represents the HTML document displayed in a browser window. It has properties, methods and events that allow the programmer to change the way the document is displayed in response to user actions or other events.

Document properties include:

bgColor The colour of the background. For example:

document.bgColor = "lightgreen";

would cause the background colour of the document to change to light-green.

Click here to change the background colour of this document.
Click here to change it back again.

fgColor The colour of the text. For example:

document.fgColor = "blue";

will cause the colour of the text in the document to change to blue.

Click here to change the foreground colour of this document.
Click here to change it back again.

(Note that this will not work with all browsers).

linkColor The colour used for un-visited links (i.e., those that have not yet been clicked-upon by the user). For example:

document.linkColor = "red";

will change the colour of all the un-visited links in a document to red.

alinkColor The colour used for an active link (i.e., the one that was clicked-upon most recently, or is the process of being clicked). For example:

document.alinkColor = "lightred";

will change the colour of active links in a document to light-red.

vlinkColor The colour used for visited links (i.e., those that have previously been clicked-upon by the user). For example:

document.vlinkColor = "darkred";

will change the colour of all the visited links in a document to dark-red.

title The title of the document, as displayed at the top of the browser window. For example:

document.title = "This title has been changed";

will replace the existing page title with the text "This title has been changed".

Click here to see this code in operation.

(Note that some browsers do not display a title bar. On such browsers this code will have no effect.)

forms An array containing all the forms (if any) in the document. It accepts an index number in the following way:

forms[index-number]

where index-number is the number of a particular form. Forms are automatically numbered from 0, starting at the beginning of the document, so the first form in an HTML document will always have the index-number 0.

An example of the use of the forms property is given below, in the section on the form object.

Document methods include:

write() Allows a string of text to be written to the document. Can be used to generate new HTML code in response to user actions. For example:

```
document.write("<h1>Hello</h1> ");
document.write("<p>Welcome to the new page</p>");
document.write("<p>To return to the lecture notes,");
document.write("<a href='05_JS4.html'>click here </a></p>");
```

will replace the existing page display with the HTML code contained within the brackets of the

document.write() methods. This code will display the text "Hi, there!" and "Welcome to the new page", followed by a link back to this page.

Note that all the HTML code within the brackets is enclosed within double-quotes. Note too that the link declaration ('05_JS4.html') is enclosed within single quotes. You can use either single or double quotes in both cases, but you must be careful not to mix them up when placing one quoted string inside another.

Click here to see this code in operation.

The Form Object

When you create a form in an HTML document using the <form> and </form> tags, you automatically create a form object with properties, methods and events that relate to the form itself and to the individual elements within the form (e.g., text boxes, buttons, radio-buttons, etc.). Using JavaScript, you can add behaviour to buttons and other form elements and process the information contained in the form.

Form properties include:

name The name of the form, as defined in the HTML <form> tag when the form is created, for example:

```
<form name="myForm">
```

This property can be accessed using JavaScript. For example, this paragraph is part of a form that contains the example buttons. It is the third form in the document (the others contain the buttons for the Window and Document object examples). To obtain the name of this form, we could use the following code:

```
alert(document.forms[2].name);
```

This code uses the document.forms property described earlier. Since this is the third form in the document it will have the index-number 2 (remember that forms are numbered from 0).

Thus the code above will display the name property of the example form, which is simply "formExamples".

Click here to see this code in operation.

methodThe method used to submit the information in the form, as defined in the HTML <form> tag when the form is created, for example:

```
<form method="POST">
```

The method property can be set either to POST or GET (see under 'forms' in any good HTML

reference book if you're not sure about the use of the POST and GET methods).

This property can be accessed using JavaScript. For example, the present form has its method attribute set to "POST" (even though it's not actually going to be submitted). So the code:

```
alert(document.forms[2].method);
```

will display the method property of the example form, which is "POST".

Click here to see this code in operation.

action The action to be taken when the form is submitted, as defined in the HTML <form> tag when the form is created, for example:

```
<form action="mailto:sales@bigco.com">
```

The action property specifies either the URL to which the form data should be sent (e.g., for processing by a CGI script) or mailto: followed by an email address to which the data should be sent (for manual processing by the recipient). See under 'forms' in any good HTML reference book for more information on the use of the action attribute.

This property can be accessed using JavaScript. For example, the present form has its action attribute set to "mailto:sales@bigco.com" (even though it's not actually going to be submitted). So the code:

```
alert(document.forms[2].action);
```

will display the action property of the example form, which is "mailto:sales@bigco.com".

Click here to see this code in operation.

length The number of elements (text-boxes, buttons, etc.) in the form. For example:

```
alert(document.forms[2].length);
```

will display the number of elements in this form (there are 22).

Click here to see this code in operation.

elements An array of all the elements in the form. Individual elements are referenced by index-number.

Elements are automatically numbered from 0, starting at the beginning of the form, so the first element in a form will always have the index-number 0. For example:

```
alert(document.forms[2].elements[0].name);
```

will display the name of the first element in this form, which is the button labelled "Get Form Name". It's name is "get_form_name".

Click here to see this code in operation.

Form methods include:

submit() Submits the form data to the destination specified in the action attribute using the method specified in the method attribute. As such it performs exactly the same function as a standard 'submit' button, but it allows the programmer greater flexibility. For example, using this method it is possible to create a special-purpose 'submit' button that has more functionality than a standard 'submit' button, perhaps checking the data or performing some other processing before submitting the form.

Form events include:

onSubmit Message sent each time a form is submitted. Can be used to trigger actions (e.g., calling a function). Usually placed within the <form> tags, for example:

<form onSubmit="displayFarewell()">

would cause the function displayFarewell() to execute automatically every time the form is submitted.

Text-boxes and text-areas
Each element within a form is an object in its own right, and each has properties, methods and events that can be accessed using JavaScript.

Text-boxes and text-areas have almost identical sets of properties, methods and events, so they will be considered together.

Text-box and text-area properties include:

name The name of the text-box or text-area, as defined in the HTML <input> tag when the form is created, for example:

<input type=text name="textBox1">

The name property of a text-box or other form element can be accessed using JavaScript in the manner shown under the section on document.length, above.

value Whatever is typed-into a text-box by the user. For example, here is a simple text-box:

This text-box is named textBox1. Therefore, we can obtain any text typed into it using the following line of code:

alert(document.forms[2].textBox1.value);

Type something into the text-box, then click here to see this code in operation.

Text-box and text-area events include:

onFocus Event signal generated when a user clicks in a text-box or text-area. For example, here is a simple text-box:

This text-box was declared using the following HTML code:

<input type=text name="textBox2" onFocus="alertOnFocus()">

The function called alertOnFocus() displays an alert box, so clicking in the text-box above should trigger the function and cause the alert to appear.

onBlur Event signal generated when a user clicks outside a text-box or text-area having previously clicked inside it. For example, here is a simple text-box:

This text-box was declared using the following HTML code:

<input type=text name="textBox3" onBlur="alertOnBlur()">

The function called alertOnBlur() displays an alert box, so clicking in the text-box above and then clicking outside it should trigger the function and cause the alert to appear.

Buttons, Radio-buttons and Checkboxes
Buttons, Radio-buttons and Checkboxes have almost identical sets of properties, methods and events, so they will be considered together.

Button, Radio-button and Checkbox properties include:

name The name of the button, radio-button or checkbox, as defined in the HTML <input> tag when the form is created, for example:

<input type=button name="button1">

The name property of a button, radio-button or checkbox can be accessed using JavaScript in the manner shown under the section on document.length, above.

value The value given to the button when it is created. On standard buttons the value is displayed as a label. On radio-buttons and check-boxes the value is not displayed. For example, here is a button:

This button is named button1 and has the value Original value, original label. We can change the value of the button, and hence it's label, using the following code:

document.forms[2].button1.value = "New value, new label";

Click here to see this code in operation.

checked This property - which is used with radio-buttons and check-boxes but not standard buttons - indicates whether or not the button has been selected by the user. For example, here is a checkbox:

This checkbox is named checkbox 1. We can determine whether it has been selected or not using the following code:

```
if (document.forms[2].checkbox1.checked == true)
{
  alert("Checked");
}
else
{
  alert("Not checked");
};
```

Try clicking on the check-box to select and un-select it, then click here to see this code in operation.

Button, Radio-button and Checkbox methods include:

focus() Give the button focus (i.e., make it the default button that will be activated if the return key is pressed). For example, here is a button that displays an alert when clicked:

This button is named button2 and has the value Hello. We can give it focus using the following code:

document.forms[2].button2.focus();

Click here to see this code in operation. A dotted border should appear around the label on the button, indicating that the button now has focus. Pressing the RETURN key should now activate the button, causing it to display the alert just as if it had been clicked.

blur() Removes focus from a button. For example, the code:

document.forms[2].button2.blur();

will remove the focus (indicated by the dotted line) from the button above. Click here to see this code in operation.

click() Simulates the effect of clicking the button. For example, below is a button that has the following code:

document.forms[2].button2.click();

Clicking on this button will have the same effect as clicking directly on the button labelled 'Hello', i.e., it will display the 'Hello to you too' dialog box.

Button, Radio-button and Checkbox events include:

onClick Signal sent when the button is clicked. Can be used to call a function. Probably the most frequently-used of all the button events (all the example buttons in this document use this method).

For example:

This button was declared using the following code:

<input type=button name="button3" value="Click Here" onClick="alert('onClick event received')">

The code onClick="alert('onClick event received')" will cause an alert dialog box to appear whenever the button is pressed.

onFocus Signal sent when the button receives focus (i.e., when it becomes the default

button, the one that is activated by pressing the RETURN key). For example:

This button was declared using the following code:

<input type=button name="button4" value="Click Here" onFocus="alert('This button is now the default')">

The first time you click the button it will gain focus, and the alert will appear. However, if you click again, no alert will appear because the button still has focus as a result of the previous click. To make the alert appear again, you will have to remove focus from the button (e.g., by clicking somewhere else on the document) then restore it by clicking the button again.

onBlur Signal sent when the button loses focus. For example:

This button was declared using the following code:

<input type=button name="button5" value="Click Here" onBlur="alert('This button is no longer the default')">

Clicking the button will not cause the alert to appear because it will only give the button focus. However, if you remove focus from the button (e.g., by clicking somewhere else on the document) the alert will appear.

The Select Object

Selection-boxes behave in a very similar fashion to radio-buttons: they present several options, of which only one can be selected at a time. They also have a similar set of properties, methods and events.

The principal difference from a programming perspective is that selection-boxes don't have a checked property. Instead, to find out which option has been selected, you must use the SelectedIndex property.

SelectedIndex Returns an integer indicating which of a group of options has been selected by the user. For example:

Something or other

This selection-box is named selectBox1. We can find out which option is currently selected by using the following code:

```
alert(document.forms[2].selectBox1.selectedIndex);
```


Click here to see this code in operation. You should find that the value of selectedIndex (as shown in the dialog box) varies from 0 to 2 depending upon which of the three items in the selection-box is currently selected.

Part 5

Other Objects

In addition to the objects we have already encountered, there are a number of other objects that form part of the JavaScript language. Among the more important and useful of these are the Date and Math objects.

The Date object

The Date object allows us to obtain the current date and time, and to perform various timing operations.

In order to use the Date object, we must first create a new 'instance' of it. This is done in the following way:

```
var myDateObject = new Date;
```

This creates an object called myDateObject that contains information about the date, time, etc., at the instant it was created. The information in myDateObject doesn't change as time passes, so if you want to know the correct time a few minutes later you will have to create a new instance of the Date object.

Once you have created an instance of the Date object, you can use any of the methods below to obtain information from it:

getFullYear() Returns the current year as a four-digit number (e.g., 2000). For example:

```
var myDateObject = new Date;
var currentYear = myDateObject.getFullYear();
alert(currentYear);
```

Click here to see this example working.

getMonth() Returns the current month as an integer from 0-11 (e.g., November = 10). For example:

```
var myDateObject = new Date;
var currentMonth = myDateObject.getMonth();
alert(currentMonth);
```

Click here to see this example working.

getDate() Returns the day of the month as an integer between 1 and 31. For example:

```
var myDateObject = new Date;
var currentDate = myDateObject.getDate();
alert(currentDate);
```

Click here to see this example working.

getDay() Returns the day of the week as an integer between 0 and 6, starting from Sunday (e.g., Tuesday = 2). For example:

```
var myDateObject = new Date;
var currentDay = myDateObject.getDay();
alert(currentDay);
```

Click here to see this example working.

getHours() Returns the hour of the day as an integer between 0 and 23. For example:

```
var myDateObject = new Date;
var currentHour = myDateObject.getHours();
alert(currentHour);
```

Click here to see this example working.

getMinutes() Returns the number of minutes since the beginning of the hour as an integer. For example:

```
var myDateObject = new Date;
var currentMinute = myDateObject.getMinutes();
alert(currentMinute);
```

Click here to see this example working.

getSeconds() Returns the number of seconds since the start of the minute as an integer. For example:

```
var myDateObject = new Date;
var currentSecond = myDateObject.getSeconds();
alert(currentSecond);
```

Click here to see this example working.

In order to use the Date object, it is often necessary to convert the data it produces, e.g., to obtain the names of days and months rather than just numbers. To see an example of the Date object in use, click here.

The Date object also has methods to obtain time intervals as small as milliseconds, to convert between various time systems, to parse dates and times in various formats into individual elements, and to perform various other time-related operations.

The Math object

The Math object allows us to perform various mathematical operations not provided by the basic operators we have already looked at.

Its methods include the following:

sqrt(x) Returns the square root of x. For example:

```
var inputValue = prompt("Please enter a value", "");
var squareRoot = Math.sqrt(inputValue);
alert(squareRoot);
```

Click here to see this example working.

log(x) Returns the natural logarithm of x. For example:

```
var inputValue = prompt("Please enter a value", "");
var logOfX = Math.log(inputValue);
alert(logOfX);
```

Click here to see this example working.

max(x,y) Returns whichever is the larger of x and y. For example:

```
var inputX = prompt("Please enter a value for X", "");
var inputY = prompt("Please enter a value for Y", "");
var largerOfXY = Math.max(inputX, inputY);
alert(largerOfXY);
```

Click here to see this example working.

min(x,y) Returns whichever is the smaller of x and y.
Works in a similar way to max(x,y), above.

round(x) Returns the value of x rounded to the nearest integer. For example:

```
var inputvalue = prompt("Please enter a value", "");
var roundedValue = Math.round(inputValue);
alert(roundedValue);
```

Click here to see this example working.

ceil(x) Returns the absolute value of x rounded up to the next integer value.
Works in a similar way to round(x), above.

floor(x) Returns the absolute value of x rounded down to the next integer value.
Works in a similar way to round(x), above.

abs(x) Returns the absolute value of x. For example:

```
var rawValue = prompt("Please enter a value", "")
var absValue = Math.abs(rawValue)
alert(absValue);
```

Click here to see this example working.

pow(x,y) Returns the value of x raised to the power y. For example:

```
var baseValue = prompt ("Please enter base value", "");
var expValue = prompt ("Please enter exponent", "");
var baseToPower = Math.pow(baseValue, expValue);
alert(baseToPower);
```

Click here to see this example working.

The Math object also has methods to perform trigonmetrical operations such as sin(), cos(), tan(), etc., and a set of properties that include values for pi and other constants.

Part 6
The String Object
JavaScript includes a String object that allow us to manipulate strings in a variety of ways, for example, searching a string to see if it contains certain patterns of letters, extracting part of it to form new string, and much more.

A String object is created in the following way:

```
var myString = new String("Hello World");
```

However, most browsers regard any string as an instance of the String object. Therefore you can declare a string in the normal way, e.g.:


```
var myString = "Hello World";
```

...and in so doing you will automatically create a String object, complete with its associated methods, properties, etc..
String properties include:

length Returns the length of a string. For example, here is a text box:

It is called textBox1 and is part of a form called form1.

If someone types into the box, you could find out how many characters they typed using the following code:

```
var stringLength = document.form1.textBox1.value.length
```

Type some text into the box, then click here to see this code in operation.

String methods include:

charAt() Returns the character at a specified position in a string. The syntax is:

charAt(index)

where index is a number representing the position of a character in the string. For example, here is a text box:

It is called textBox2 and is part of form1.

You could find out what the third character in the text-box is using the following code:

```
var thirdCharacter = document.form1.textBox2.value.charAt(2);
```

(Note that the characters in a string are numbered from zero, not one. Therefore the third character will be character number two.)

Type some text into the box, then click here to see this code in operation.

indexOf() Searches a string to see if it contains a specified character, and if it does, returns the position of that character. If the specified character occurs more than once in the string, it returns the

position of the first occurrence of that character. The syntax is:

indexOf(character)

For example, here is a text box:

It is called textBox3 and is part of form1.

To find out whether the text-box contains the letter 'c', we could use the following code:

```
var positionOfC = document.form1.textBox3.value.indexOf("c");
```

Type some text into the box, then click here to see this code in operation.

Again, bear in mind that the characters in the string are numbered from zero, not one, so the index will be one less than the character's actual position. Also note that if there is no 'c' in the string, the value returned will be -1.

lastIndexOf() Searches a string to see if it contains a specified character, and if it does, returns the position of that character. It performs the same function as indexOf(), except that if the specified character occurs more than once in the string, it returns the position of the last occurrence of that character rather than the first.

substring() Returns the portion of a string between two specified positions. The syntax is:

substring(start_position, end_position)

where start_position is the position of the first character in the wanted portion of the string (counting from zero), and end_position is the position after the last character of the wanted portion of the string.

For example, here is a text box:

It is called textBox4 and is part of form1.

Suppose we wish to extract the third, fourth and fifth characters from a string in the text-box. Counting from zero, these would be the characters in positions 2, 3 and 4. Therefore we would set the first parameter to 2 and the second parameter to 5 (one position after the last character we want). For example:

```
var extract = document.form1.textBox4.value.substring(2,5);
```


Type some text into the box, then click here to see this code in operation.

substr() Returns a portion of a string, starting from a specified positions and continuing for a specified number of characters. The syntax is:

substr(start_position, no_of_characters)

where start_position is the position of the first character in the wanted portion of the string (counting from zero), and no_of_characters is the number of characters to extract, starting at that position.

In other words, it behaves in a very similar way to the substring( ) method, except that second parameter specifies the number of characters to extract from the string rather than the position after the last character.

charCodeAt() Returns the numerical (ASCII) value of the character at the specified position.

For example, here is a text box:

It is called textBox5 and is part of form1.

To find the ASCII value of a single character typed into this box, we could use the following code:

```
var asciiValue = document.form1.textBox5.value.charCodeAt(0);
```

Type a character into the box, then click here to see this code in operation.

fromCharCode() Returns the characters represented by a sequence of numerical (ASCII) values.

For example:

```
var asciiString = String.fromCharCode(73,97,110);
```

This code will create a string containing the ASCII characters whose numerical valies are 73, 97 and 110. Click here to see this code in operation.

toString() Converts a number into a string. The syntax is:

toString(number)

For example, consider the following code:

```
var aNumber = 12345;
alert("The length is " + aNumber.length);

var aNumber = aNumber.toString();
alert("The length is " + aNumber.length);
```

The variable aNumber is not enclosed in quotes or otherwise declared as a string, so it is a numeric variable. Therefore it doesn't have a length property, and the first alert() dialog in the example will report that the length is 'undefined'.

However, once we have converted aNumber into a string using the toString() method, it will have a length property just like any other string, so the second alert() dialog in the example should report the length correctly.

Click here to see this code in operation.

In addition to the methods shown above, the String object also provides methods to add HTML formatting to strings. These methods include:

italics() Formats a string with and tags. The syntax is:

```
string_name.italics()
```

For example:

```
var myString = "Hello World";
document.write(myString.italics());
```

Would have the same effect as:

```
var myString = "Hello World";
document.write("<i>" + myString + "</i>");
```

bold() Formats the string with and tags. Works in a similar fashion to the italics() method (see above).

sup() Formats the string with ^{and} (i.e., super-script) tags. Works in a similar fashion to the italics() method (see above).

sub() Formats the string with _{and} (i.e., sub-script) tags. Works in a similar fashion to the italics() method (see above).

Regular Expressions

A regular expression describes a pattern of characters or character types that can be used when searching and comparing strings.

For example, consider a web-site that allows the user to purchase goods online using a credit-card. The credit-card number might be checked before submission to make sure that it is of the right type, e.g., that it is 16 digits long, or consists of four groups of four digits separated by spaces or dashes.

It would be quite complicated to perform such checks using just the string-handling functions described above. However, using regular expressions we could create a pattern that means 'four digits followed by a space or a dash, followed by four more digits...', etc.. It is then quite simple to compare this pattern with the value entered by the user and see if they match or not.

The special characters that can be used to create regular expressions include:

\d Represents any numerical character

{x} Indicates that the preceding item should occur x times consecutively

In addition, any letter or other character can be used explcitly. For example, if you place an 'a' in a regular expression pattern it means that an 'a' is expected as a match at that point.

Using these pattern-matching characters, we could create a simple pattern that checks for a credit-card number in the following way:

```
\d{4}-\d{4}-\d{4}-\d{4}
```

The first '\d' means 'any number'. This is followed by '{4}', which extends the pattern to mean 'any four consecutive numbers'. After this comes a '-' which simply means that a dash character is expected at this point. Then comes the 'any four consecutive numbers' pattern again, followed by the dash, and so on.

To create such a pattern, we would declare a regular expression object and give it the pattern as a value. This can be done in two ways:

```
var myRegExp = new RegExp("\d{4}-\d{4}-\d{4}-\d{4}");
```

Or:

```
var myRegExp = /\d{4}-\d{4}-\d{4}-\d{4}/;
```

In the first example, the new regular expression object (RegExp) is declared explicitly.

In the second example, the pattern is declared in much the same way a string might be declared, except that forward-slashes (/) are used at the beginning and end instead of quote-marks. The forward-slashes

indicate that this sequence of characters is being declared as a regular expression rather than as a string.

Once the regular expression has been created, we can use the test() method to compare it with a string. For example:

```
var myRegExp = /\d{4}-\d{4}-\d{4}-\d{4}/;

var inputString = prompt("Please enter Credit Card number","");

var result = myRegExp.test(inputString);

alert(result);
```

In this example, the regular expression is assigned to a variable called myRegExp. The user is then prompted to enter a string which is assigned to the variable inputString. The test() method is then used to compare the two strings, with the result being passed to the variable result which is displayed in an alert() dialog-box. If the string matches the pattern, the result will be true; if not, it will be false.

Click here to see this example working. Try entering various numbers into the box and see the result. You should find that the code only returns true if you enter a number that consists of four groups of four digits separated by dashes.

We can also compare a string with several different patterns using the Logical OR operator. For example:

```
\d{16}|\d{4}-\d{4}-\d{4}-\d{4}
```

This example is similar to the previous one except that the following characters have been added to the start of the string:

```
\d{16}|
```

The first few characters, \d{16}, mean 'any sixteen consecutive numbers'. The | character indicates a logical OR, meaning that the whole expression will be true if either the part before this symbol OR the part after it are true.

In other words, a string will be regarded as a valid match if it contains 16 consecutive digits OR four groups of four digits separated by dashes.

Click here to see this example working. You should find that the code returns true if you enter a number that consists of sixteen consecutive digits or four groups of four digits separated by dashes.

This is fine if the user enters consecutive numbers or groups of numbers separated by dashes, but what if

the user enters groups of numbers separated by spaces?

It is possible test a character in a string to see if it is any one of several specified characters. This can be done in the following way:

[xyz] Match any of the characters within the brackets, e.g., if the characters within the brackets are x, y and z, the test will return true if the character at that point in the string is either x, y or z.

Using this method, we could modify our previous pattern as follows:

```
\d{16}|\d{4}[ -]\d{4}[ -]\d{4}[ -]\d{4}
```

In this example, each of the dashes has been replaced with a pair of square brackets containing both a space and a dash. This means that either a space or a dash will be accepted at these points in the string. Thus the complete string will regarded as a valid match if it contains 16 consecutive digits OR four groups of four digits separated by spaces or dashes.

Click here to see this example working. You should find that the code returns true if you enter a number that consists of sixteen consecutive digits or four groups of four digits separated either by dashes or spaces.

The examples given above indicate only a few of the possibilities offered by regular expressions.

Some of the most commonly-used pattern-matching characters are shown below. For a more complete list you should consult a good reference book (the 'Pure JavaScript' book recommended for use with this course has quite an extensive list).

\w	Represents any alphanumerical character
\W	Represents any non-alphanumerical character
\d	Represents any numerical character
\D	Represents any non-numerical character
\s	Represents any 'whitespace' character (e.g., carriage-return, tab, etc.)
\S	Represents any non-whitespace character
[..]	Match any one of the characters within the brackets
[^..]	Match any one character other than those within the brackets
[x-y]	Match any character within the range x to y
[^x-y]	Match any one character other than those within the range x to y
{x}	Match the previous item x times
{x,}	Match the previous item at least x times

Using Regular Expressions with Strings

In addition to the methods described above, the String object has a number of methods that are specifically designed to work with regular expressions.

search(regExp) Searches for any occurrences of the regular expression in the string. If any are found, it returns an integer indicating where within the string the matching sequence begins. If no match is found it returns -1.

For example:

The textbox above is called textBox6 and it is part of a form called form2.

If someone typed a string into the box containing a two-digit number, you could find where the number begins using the following code:

```
var myRegExp = /\d\d/;

var numStart = document.form2.textBox6.value.search(myRegExp);

alert("The number starts at position " + numStart);
```

Type some text into the box, then click here to see this code in operation.

replace(regExp, newString) Searches for any occurrences of the regular expression in the string. If any are found, it replaces them with newString.

For example:

The textbox above is called textBox7 and it is part of a form called form2.

If someone typed a string into the box containing a two-digit number, you could replace the number with 00 using the following code:

```
var myRegExp = /\d\d/;

var newString = document.form2.textBox7.value.replace(myRegExp, "00");

alert("The modified string is: " + newString);
```

Type some text into the box, then click here to see this code in operation.

Part 7

Arrays

An array is a set of variables (e.g., strings or numbers) that are grouped together and given a single

name.

For example, an array could be used to hold a set of strings representing the names of a group of people. It might be called people and hold (say) four different strings, each representing the name of a person:

Sarah Patrick Jane Tim

Items are held in an array in a particular order, usually the order in which they were added to the array. However, one of the great advantages of arrays is that the ordering can be changed in various ways. For example, the array above could easily be sorted so that the names are arranged in alphabetical order.

Creating Arrays

To create an array, a new Array object must be declared. This can be done in two ways:

```
var myArray = new Array("Sarah","Patrick","Jane","Tim");
```

Or:

```
var myArray = ["Sarah","Patrick","Jane","Tim"];
```

In the first example, the new Array object is declared explicitly.

In the second example, the array is declared in much the same way a string might be declared, except that square brackets ([]) are used at the beginning and end instead of quote-marks. The square brackets indicate to the JavaScript interpreter that this sequence of characters is being declared as an array.

Arrays are often used to hold data typed-in or otherwise collected from a user. Therefore, it may be necessary to create the array first and add data to it later. An empty array may be created in the following way:

```
var myArray = new Array();
```

The array thus created has no elements at all, but elements can be added as necessary later.

If it is not known exactly what data will be stored in the array, but the number of items is known, it may be appropriate to create an array of a specific size. This may be done in the following way:

```
var myArray = new Array(4);
```

The array thus created has four elements, all of which are empty. These empty elements can be filled with data later on.

Viewing and Modifying Array Elements

Suppose an array has been created using the following code:

```
var demoArray = new Array("Sarah","Patrick","Jane","Tim");
```

The number of elements in the array can be determined using the length property. For example:

```
alert(demoArray.length);
```

Click here to see this example working.

The entire contents of the array can be viewed using the valueOf() method. For example:

```
alert(demoArray.valueOf())
```

This piece of code will display an alert showing the entire contents of the array demoArray, with the various elements separated by commas.

Click here to see this example working.

The value of a particular element in the array can be obtained by using its position in the array as an index. For example

```
var indexNumber = prompt ("Please enter a number between 0 and 3","");
alert("Element " + indexNumber + " = " + demoArray[indexNumber]);
```

This piece of code will prompt the user to enter a number between 0 and 3 (the elements in an array are numbered from zero, so the four elements in this array will be numbered 0, 1, 2 and 3). It will then display the corresponding element from the array.

Click here to see this example working.

It is also possible to change the value of an array element using its position in the array as an index. For example:

```
var newValue = prompt("Please enter your name","");
demoArray[0] = newValue;
alert(demoArray.valueOf());
```

This piece of code will prompt the user to enter their name, then place this string into element 0 of the array, over-writing the string previously held in that element. It will then display the modified array using the valueOf() method described earlier.

Click here to see this example working.

Adding and Removing Elements

The length property of an array can be altered as well as read:

increasing the length property adds extra (empty) elements onto the end of the array.

decreasing the length property removes some of the existing elements from the end of the array.

Consider the following examples:

(1)
```
var currentLength = demoArray.length;
demoArray[currentLength] = "Fred";
alert(demoArray.valueOf());
```

This piece of code first determines the number of elements in the array using the length property. It then uses this information to identify the next element position after the end of the existing array and places the string "Fred" into that position, thus creating a new element. Finally, it displays the modified array using the valueOf() method described earlier.

Note that it isn't necessary to add 1 to the value of length in order to identify the next position in the array. This is because length indicates the actual number of elements, even though the elements are numbered from zero. For example, if an array has two elements, the value of length will be 2; however, those two elements will be numbered 0 and 1. Therefore, if length is used as an index, it will indicate the third element in the array, not the second one.

Click here to see this example working.

(2)
```
var currentLength = demoArray.length;
demoArray.length = currentLength - 1;
alert(demoArray.valueOf());
```

This piece of code first determines the number of elements in the array using the length property. It then resets length to one less than its previous value, thus removing the last element in the array. Finally, it displays the modified array using the valueOf() method described earlier.

Click here to see this example working.

There are also several methods that add and remove elements directly, some of which are listed below. However, it should be noted that these methods only work with Netscape Navigator, so it is generally

preferable to use the methods described above since they work with most browsers.

push() Adds one or more elements onto the end of an array. For example:

```
var lastElement = demoArray.push("Fred", "Lisa");
```

This piece of code would add two new elements, "Fred" and "Lisa" onto the end of the array. The variable lastElement would contain the value of the last element added, in this case "Lisa".

pop() Removes the last element from the end of an array. For example:

```
var lastElement = demoArray.pop();
```

This piece of code would remove the last element from the end of the array and return it in the variable lastElement.

unshift() Adds one or more new elements to the beginning of an array, shifting the existing elements up to make room. Operates in a similar fashion to push(), above.

shift() Removes the first element from the beginning of an array, shifting the existing elements down to fill the space. Operates in a similar fashion to pop(), above.

Splitting and Concatenating Arrays

Arrays can be split and concatenated using the following methods:

slice(x,y) Copies the elements between positions x and y in the source array into a new array. For example:

```
newArray = demoArray.slice(0,2);
alert(newArray.valueOf());
```

This piece of code will copy elements 0 and 1 from the array called demoArray into a new array called newArray. It will then display the contents of newArray using the valueOf() method described earlier.

Note that the slice starts at x but stops at the last position before y rather than at position y itself.

Click here to see this example working.

concat(array) Concatenates the specified array and the array to which it is applied into a new array. For example:

```
combinedArray = demoArray.concat(newArray);
alert(combinedArray.valueOf());
```


This piece of code will concatenate demoArray and the new array created in the last example (newArray) to form another array called combinedArray. It will then display the contents of combinedArray using the valueOf() method described earlier.

Click here to see this example working.

Rearranging Array Elements

The order of the elements in an array can be modified using the following methods:

reverse() Reverses the order of the elements within an array. For example:

```
demoArray.reverse();
alert(demoArray.valueOf());
```

This piece of code will reverse the order of the elements in the array, then display the re-ordered array using the valueOf() method described earlier.

Click here to see this example working.

sort() Sorts the elements within the array. Unless otherwise specified, the elements will be sorted alphabetically. For example:

```
demoArray.sort();
alert(demoArray.valueOf());
```

This piece of code will sort the elements of the array into alphabetical order, then display the re-ordered array using the valueOf() method described earlier.

Click here to see this example working.

Although the sort() method normally sorts arrays alphabetically, it can be modified to sort in other ways. This is done by creating a special function and passing the name of that function to the sort() method as a parameter, e.g.:

```
demoArray.sort(bylength);
```

The sorting function must be written in such a way that it takes two elements of the array as parameters, compares them, then returns one of the following values indicating what order they should be placed in:

-1 : The first element should be placed before the second.

0 : The two elements are the same so the ordering does not matter.

1 : The first element should be placed after the second.

For example, here is a sorting function that sorts array elements by length:

```
function bylength(item1, item2)
{
if(item1.length < item2.length)
{
return -1;
};
if(item1.length == item2.length)
{
return 0;
};
if(item1.length > item2.length)
{
return 1;
};
}
```

In accordance with the rules for sorting functions, this example accepts two parameters, item1 and item2. Each of these parameters will be an element of the array, passed to it by the sort() method.

Since the demonstration array contains strings, each element will have a length property. The length properties of the two parameters are compared using three if statements, and either -1, 0 or 1 is returned depending upon their relative lengths.

The sort() method will apply this function to each pair of strings in the array in turn until all have been sorted.

Click here to see this example working.

Multi-Dimensional Arrays

The arrays described so far are One-Dimensional Arrays. They are effectively just lists.

Sometimes, however, we need to store information which has more than one dimension, for example, the scores in a game:

Sarah 18
Patrick 16
Jane 12
Tim 13

To store data of this type we use multi-dimensional arrays. In JavaScript this is done by using arrays as elements of other arrays.

For example, the game scores could be stored in the following way:

Each person's data would be stored in a separate, two-element array (one element for the name and one element for the score).

These four arrays (one for each person) would then be stored in a four-element array.

We could create such an array in the following way:

```
var person1 = new Array("Sarah", 18);
var person2 = new Array("Patrick", 16);
var person3 = new Array("Jane", 12);
var person4 = new Array("Tim", 13);

var scores = new Array (person1,person2,person3,person4);
```

To identify the individual elements within a multi-dimensional array, we use two index values, one after the other. The first refers to an element in the outer array (scores in this example), and the second refers to an element in the inner array (person1, person2, person3 or person4 in this example). For example:

```
function displayMDarray()
{
for(x = 0; x <=3; x++)
{
alert(scores[x][0] + " has " + scores[x][1] + " points");
}
}
```

In this example, the array is accessed using pairs of values in square brackets, e.g.:

scores[x][0]

The value in the first pair of square-brackets indicates one of the four 'person' arrays. The value in the

second pair of square-brackets indicates one of the elements within that 'person' array, either the name (0) or the score (1).

The variable called x is incremented from 0 to 3 using a for loop. Therefore, x will point to a different one of the four 'person' arrays each time through the loop. The second value, 0 or 1, then selects either the name or score from within that array.

Click here to see this example working.

Multi-dimensional arrays can be accessed and modified using the same methods as one-dimensional arrays.

For example, a multi-dimensional array can be sorted using the reverse() and sort() methods in just the same way as a one-dimensional array, e.g.:

scores.sort()
Click here to sort the array, then click on the "View Elements of Multi-Dimensional Array" button to see the effects.

Note that the array is sorted by the first element of each sub-array, i.e., by names rather than scores.

Part 8
Differences between Browsers
Browsers vary considerably in:

The HTML tags they recognise, and the way they respond to certain HTML tags.

The version of the Document Object Model (DOM) they support.

The versions of JavaScript (and other scripting languages) they support.

Therefore, creating web pages that will display correctly on all (or even most) browsers is not easy, particularly if the pages contain dynamic material.

However, use of a scripting language such as JavaScript allows us to create pages that will load and run correctly on a wide range of browsers.

Major differences between browsers include:

The mechanism used to address elements

Using Dynamic HTML it is possible to modify various attributes of elements - usually style attributes such as size, position, visibility, etc. - using scripts. For this to work, there has to be a means of uniquely identifying each element within a web-page so that information can be sent to it or retrieved from it.

Some elements - such as forms and their components - are represented in the object hierarchy and can be addressed directly. However, not all elements are represented in the object hierarchy, and it was not considered practical to introduce new objects for each type of element. Therefore, browser manufacturers introduced other mechanisms for addressing elements:

Microsoft added the all object to Internet Explorer. This is an array of all the elements of a web-page, allowing the individual elements to be addressed in the following way:

```
document.all.myElement.style.attribute
```

Netscape added a mechanism whereby any element could be addressed directly, provided it had been given a name. This allows elements to be addressed in the following way:

```
document.myElement.style.attribute
```

The World-Wide Web Consortium (W3C) subsequently agreed a standard (part of the DOM Level 1 Standard) whereby elements are assigned a recognised name and henceforth addresed directly by that name. For example:

```
var myElement = document.getElementById("myElementName")

myElement.style.attribute
```

This standard has been adopted in newer browsers (e.g., Netscape 6), but earlier browsers use one of the proprietary systems described above.

The use of Divisions and Layers

The tag is recognised by both Internet Explorer and Netscape Navigator, but they offer different levels of support for it:

In Internet Explorer 4 & 5 and Netscape Navigator 6, divisions are dynamic (i.e., they can be repositioned and/or have their size, visibility, etc., modified).

In Netscape Navigator 4, divisions can only be static.

The <layer> tag is specific to Netscape Navigator and was supported on versions of Netscape Navigator prior to version 6. On these browsers it offers similar functionality to that available using divisions in Internet Explorer, including dynamic-repositioning, etc..

Event handling

In Internet Explorer, events propagate up the object hierarchy.

For example, if the mouse is clicked whilst over a form button, the click event is received by the button object and then passed on to objects further up the hierarchy, such as the Document object.

In Netscape Navigator, events propagate down the object hierarchy.

For example, if the mouse is clicked whilst over a form button, the click event is received by the Document object and passed down to the button object.

(In Netscape Navigator, it is possible to 'capture' events at the Document level and thus prevent them being sent down the hierarchy to form elements such as buttons; this can be used, for example, to disable all the elements in a form until such time as they are required.)

Browsers also differ in the support they provide for the JavaScript language. The level of compatibility offered by the various versions of Netscape and Internet Explorer (for JavaScript and Jscript respectively) is summarised below:

Navigator version JavaScript version Comments

2.0 1.0 First version of JavaScript.
Included some bugs/limitations, e.g.:

* window.open() method doesn't work on Mac and Unix platforms.
* Poor support for arrays.
* onLoad() event triggered by re-sizing.

3.0 1.1 Support for <src> attribute, allowing use of external JavaScript source code.

4.0 - 4.05 1.2 Support for dynamic positioning of elements.
Signed scripts for security.

4.06 - 4.5 1.3 ECMAScript 1.0 Compatible

5.0 1.4 ECMAScript 2.0 Compatible
More advanced error-checking and control functions.

Explorer version JScript versionComments
3.0 1.0 Used a separate JScript Scripting Engine which could be upgraded separately from the browser. Many versions in use.
Particular problems when handling image attributes.

4.0 - 4.5 3.0 ECMAScript 1.0 Compatible (and therefore largely identical to JavaScript 1.3).
Support for <src> attribute (but not until v3.02, and even then with bugs), allowing use of external JavaScript source code.
Support for dynamic positioning of elements.

5.0 5.0 ECMAScript 2.0 Compatible (and therefore largely identical to JavaScript 1.4)

In addition to the differences listed above, there are many minor and not-so-minor differences in the way the various browsers suppport and interpret HTML and JavaScript. Most of these will be listed in any good HTML or JavaScript reference book.

Obtaining Browser Parameters
We can tackle differences between browsers using the Navigator object

This object returns properties related to the browser, such as its type, version, etc.. It also returns properties indicating the platform on which the browser is running (Macintosh, Windows PC, Linux PC, etc.),

Navigator object properties include:

appName Returns a string representing the browser. For example, the code:

```
alert(navigator.appName);
```

should return a string containing the name of the browser you are using (e.g., "Microsoft Internet Explorer" or "Netscape")

appCodeNameReturns a string representing the code name of the browser. For example, the code:

```
alert(navigator.appCodeName);
```

should return a string containing the code name of the browser you are using (e.g., "Mozilla")

appVersion Returns a string representing the browser version. For example, the code:

alert(navigator.appVersion);

should return a string indicating which version of the browser you are using (e.g., "4.0 (compatible: MSIE 5.0)")

platform Returns a string representing the computer platform on which the browser is running. For example, the code:

alert(navigator.platform);

should return a string containing the name of the platform on which the browser is running (e.g., "Win32" or "MacPPC")

Using some or all of the Navigator object properties, It is possible to determine which browser is being used and then generate the appropriate code.

This example includes a movable bar that has to be coded differently depending upon which browser it is being viewed under. The web-page tests to see what browser is being used, then generates appropriate code. The code used is quite simple and may not accommodate all browsers, but it should work under most widely-used browsers.

Have a look at the source code of the example to see how it works. Note the following points:

The page contains a script in the <head> section. The script declares two global variables and two functions. The purpose of these is discussed below.

The body of the page contains several sections:

A piece of JavaScript code that uses the navigator object to determine what browser is being used. If the browser is found to be Netscape Navigator, a further test is performed to find out what version is being used. This is done using the charAt() method to obtain the first character of the string (which is always the browser version number). Once obtained, this information is stored in one of the global variables, browserType.

This is followed by another piece of JavaScript code that uses document.write() methods to create a new page. The content of the page varies depending upon the browser used (as determined by the previous piece of code): if the browser is Netscape Navigator 4, the page content is created using a <layer> tag; if

the browser is Internet Explorer or Netscape Navigator 6, the page content is created using a tag.

Following this is another piece of code that creates a form with two buttons. This code works equally well with all the target browsers, so it can be written directly to the document without checking the browser type.

Finally, there is a piece of code that checks the browser type again and writes either a closing </layer> tag or a closing tag to the page.

Because these pieces of code are placed in the body of the document they will be executed automatically when the document is loaded.

Having created the page, the division or layer can be moved around using the two functions in the <head> section of the document. These interrogate the global variable browserType to find out what browser is in use, then use one of the following techniques to move the division or layer:

If the code is being viewed under Internet Explorer, the division can be moved by altering the value of the pixelLeft attribute in its style.

In order to locate the division, the code uses the document.all array. Thus the pixelLeft attribute of the division called myDivision can be identified using the following code:

```
document.all.myDivision.style.pixelLeft
```

If the code is being viewed under Netscape Navigator 4, the layer can be moved by altering the value of its pageX attribute.

Netscape Navigator does not support the document.all array used in Internet Explorer. However, <layer> is represented as an object in the Document Object Model. Therefore the value of the pageX attribute of the layer called myLayer can be read or written simply by specifiying it in terms of its place within the object hierarchy, e.g.:

```
document.mylayer.pageX
```

If the code is being viewed under Netscape Navigator 6, the division can be moved by altering the value of the left attribute in its style.

However, unlike Internet Explorer, Netscape Navigator 6 does not support the document.all array, nor does it provide a special object for the tag as earlier versions did for the <layer> tag.

Instead, Netscape Navigator 6 uses the getElementByID() method, in accordance with the current

World-Wide Web Consortium (W3C) standard. The getElementByID() method is called and given the name of the division as a parameter, and the result is stored in a variable. Once this has been done, the variable can be used to address the division and its style attributes, e.g.:

```
var myDiv = document.getElementByID("myDivision");
...
mydiv.style.left
```

One slight complication with Netscape Navigator 6 is that the left style attribute is stored with the suffix 'px' (meaning pixels). This is also in accordance with the current W3C standard. Before any arithmetical operations can be performed on the left attribute, the suffix must be removed. This is done using the parseInt() method, which extracts integers from strings.

What are the possible ways to create objects in JavaScript

There are many ways to create objects in javascript as below

Object constructor:

The simplest way to create an empty object is using the Object constructor. Currently this approach is not recommended.

```
var object = new Object();
```

Object's create method:

The create method of Object creates a new object by passing the prototype object as a parameter

```
var object = Object.create(null);
```

Object literal syntax:

The object literal syntax (or object initializer), is a comma-separated set of name-value pairs wrapped incurly braces.

```
var object = {
    name: "Sudheer",age:
    34
};
```

Object literal property values can be of any data type, including array, function, and nested object.

Note: This is an easiest way to create an object

Function constructor:

Create any function and apply the new operator to create object instances,

```
function Person(name) {
  this.name = name;
  this.age = 21;
}
var object = new Person("Sudheer");
```

Function constructor with prototype:

This is similar to function constructor but it uses prototype for their properties and methods,

```
function Person() {}
Person.prototype.name = "Sudheer";var
object = new Person();
```

Prototype chaining is used to build new types of objects based on existing ones. It is similar to inheritance in a class based language.
The prototype on object instance is available through Object.getPrototypeOf(object) or __proto__ property whereas prototype on constructors function is available through Object.prototype.

What is the difference between Call, Apply and Bind
The difference between Call, Apply and Bind can be explained with below examples,
Call: The call() method invokes a function with a given this value and arguments provided one by one var employee1 = { firstName: "John", lastName: "Rodson" };
var employee2 = { firstName: "Jimmy", lastName: "Baily" };

```
function invite(greeting1, greeting2) { console.log(
greeting1 + " " + this.firstName + " " + this.lastName + ", " + greeting2
);
}
```

```
invite.call(employee1, "Hello", "How are you?"); // Hello John Rodson, How are you?
invite.call(employee2, "Hello", "How are you?"); // Hello Jimmy Baily, How are you?
```

Apply: Invokes the function with a given this value and allows you to pass in arguments as an array
var employee1 = { firstName: "John", lastName: "Rodson" };
var employee2 = { firstName: "Jimmy", lastName: "Baily" };

```
function invite(greeting1, greeting2) { console.log(
greeting1 + " " + this.firstName + " " + this.lastName + ", " + greeting2
);
}
```

```
invite.apply(employee1, ["Hello", "How are you?"]); // Hello John Rodson, How are you?
invite.apply(employee2, ["Hello", "How are you?"]); // Hello Jimmy Baily, How are you?
```

bind: returns a new function, allowing you to pass any number of arguments var employee1 = { firstName: "John", lastName: "Rodson" };
var employee2 = { firstName: "Jimmy", lastName: "Baily" };

```
function invite(greeting1, greeting2) { console.log(
greeting1 + " " + this.firstName + " " + this.lastName + ", " + greeting2
);
}
```


var inviteEmployee1 = invite.bind(employee1); var inviteEmployee2 = invite.bind(employee2);
inviteEmployee1("Hello", "How are you?"); // Hello John Rodson, How are you?
inviteEmployee2("Hello", "How are you?"); // Hello Jimmy Baily, How are you?
Call and apply are pretty interchangeable. Both execute the current function immediately. You need to decide whether it's easier to send in an array or a comma separated list of arguments. You can remember by treating Call is for comma (separated list) and Apply is for Array.
Whereas Bind creates a new function that will have this set to the first parameter passed to bind().

What is JSON and its common operations
JSON is a text-based data format following JavaScript object syntax, which was popularized by Douglas Crockford. It is useful when you want to transmit data across a network and it is basically just a text file with an extension of .json, and a MIME type of application/json
Parsing: Converting a string to a native object JSON.parse(text);
Stringification: converting a native object to a string so it can be transmitted across the network JSON.stringify(object);

What is the purpose of the array slice method
The slice() method returns the selected elements in an array as a new array object. It selects the elements starting at the given start argument, and ends at the given optional end argument without including the last element. If you omit the second argument then it selects till the end.

Some of the examples of this method are, let arrayIntegers = [1, 2, 3, 4, 5];
let arrayIntegers1 = arrayIntegers.slice(0, 2); // returns [1,2] let arrayIntegers2 = arrayIntegers.slice(2, 3); // returns [3] let arrayIntegers3 = arrayIntegers.slice(4); //returns [5]
Note: Slice method won't mutate the original array but it returns the subset as a new array.

What is the purpose of the array splice method
The splice() method is used either adds/removes items to/from an array, and then returns the removed item. The first argument specifies the array position for insertion or deletion whereas the optional second argument indicates the number of elements to be deleted. Each additional argument is added to the array.
Some of the examples of this method are, let arrayIntegersOriginal1 = [1, 2, 3, 4, 5];
let arrayIntegersOriginal2 = [1, 2, 3, 4, 5];
let arrayIntegersOriginal3 = [1, 2, 3, 4, 5];

```
let arrayIntegers1 = arrayIntegersOriginal1.splice(0, 2); // returns [1, 2]; original array: [3, 4, 5]
let arrayIntegers2 = arrayIntegersOriginal2.splice(3); // returns [4, 5]; original array: [1, 2, 3]
let arrayIntegers3 = arrayIntegersOriginal3.splice(3, 1, "a", "b", "c"); //returns [4]; original array:
[1, 2, 3, "a", "b", "c", 5]
```

Note: Splice method modifies the original array and returns the deleted array.

What is the difference between slice and splice Some of the major difference in a tabular form Slice Splice

Doesn't modify the original array(immutable) Modifies the original array(mutable) Returns the subset of original array Returns the deleted elements as array
Used to pick the elements from array Used to insert or delete elements to/from array

How do you compare Object and Map

Objects are similar to Maps in that both let you set keys to values, retrieve those values, delete keys, and detect whether something is stored at a key. Due to this reason, Objects have been used as Maps historically. But there are important differences that make using a Map preferable in certain cases.
The keys of an Object are Strings and Symbols, whereas they can be any value for a Map, including functions, objects, and any primitive.
The keys in Map are ordered while keys added to Object are not. Thus, when iterating over it, a Map object returns keys in order of insertion.
You can get the size of a Map easily with the size property, while the number of properties in an Object must be determined manually.
A Map is an iterable and can thus be directly iterated, whereas iterating over an Object requires obtaining its keys in some fashion and iterating over them.
An Object has a prototype, so there are default keys in the map that could collide with your keys if you're not careful. As of ES5 this can be bypassed by using map = Object.create(null), but this is seldom done.
A Map may perform better in scenarios involving frequent addition and removal of key pairs.

What is the difference between == and === operators
JavaScript provides both strict(===, !==) and type-converting(==, !=) equality comparison. The strict operators take type of variable in consideration, while non-strict operators make type correction/conversion based upon values of variables. The strict operators follow the below conditions for different types,
Two strings are strictly equal when they have the same sequence of characters, same length, and same characters in corresponding positions.
Two numbers are strictly equal when they are numerically equal. i.e, Having the same number value. There are two special cases in this,
NaN is not equal to anything, including NaN.
Positive and negative zeros are equal to one another.
Two Boolean operands are strictly equal if both are true or both are false. Two objects are strictly equal if they refer to the same Object.
Null and Undefined types are not equal with ===, but equal with ==. i.e, null===undefined --> false but null==undefined --> true
Some of the example which covers the above cases,

```
0 == false  // true 0 === false // false 1 == "1"        // true
1 === "1" // false
null == undefined // true null === undefined // false '0' == false // true
'0' === false // false
[]==[] or []===[] //false, refer different objects in memory
```


{}=={} or {}==={} //false, refer different objects in memory

What are lambda or arrow functions
An arrow function is a shorter syntax for a function expression and does not have its own this, arguments, super, or new.target. These functions are best suited for non-method functions, and they cannot be used as constructors.

What is a first class function
In Javascript, functions are first class objects. First-class functions means when functions in that language are treated like any other variable.
For example, in such a language, a function can be passed as an argument to other functions, can be returned by another function and can be assigned as a value to a variable. For example, in the below example, handler functions assigned to a listener

```
const handler = () => console.log("This is a click handler function");
document.addEventListener("click", handler);
```

What is a first order function
First-order function is a function that doesn't accept another function as an argument and doesn't return a function as its return value.

```
const firstOrder = () => console.log("I am a first order function!");
```

What is a higher order function

Higher-order function is a function that accepts another function as an argument or returns a function as a return value or both.

```
const firstOrderFunc = () =>
console.log("Hello, I am a First order function");
const higherOrder = (ReturnFirstOrderFunc) => ReturnFirstOrderFunc();
higherOrder(firstOrderFunc);
```

What is a unary function
Unary function (i.e. monadic) is a function that accepts exactly one argument. It stands for a single argument accepted by a function.
Let us take an example of unary function,

```
const unaryFunction = (a) => console.log(a + 10); // Add 10 to the given argument and display the value
```

What is the currying function
Currying is the process of taking a function with multiple arguments and turning it into a sequence of functions each with only a single argument. Currying is named after a mathematician Haskell Curry. By applying currying, a n-ary function turns it into a unary function.
Let's take an example of n-ary function and how it turns into a currying function, const

```
multiArgFunction = (a, b, c) => a + b + c;
console.log(multiArgFunction(1, 2, 3)); // 6
```

const curryUnaryFunction = (a) => (b) => (c) => a + b + c; curryUnaryFunction(1); // returns a function: b => c => 1 + b + c curryUnaryFunction(1)(2); // returns a function: c => 3 + c curryUnaryFunction(1)(2)(3); // returns the number 6
Curried functions are great to improve code reusability and functional composition.

What is a pure function
A Pure function is a function where the return value is only determined by its arguments without any side effects. i.e, If you call a function with the same arguments 'n' number of times and 'n' number of places in the application then it will always return the same value.
Let's take an example to see the difference between pure and impure functions,

```
//Impure
let numberArray = [];
const impureAddNumber = (number) => numberArray.push(number);
//Pure
const pureAddNumber = (number) => (argNumberArray) => argNumberArray.concat([number]);

//Display the results console.log(impureAddNumber(6)); // returns 1 console.log(numberArray); // returns [6]
console.log(pureAddNumber(7)(numberArray)); // returns [6, 7] console.log(numberArray); // returns [6]
```

As per the above code snippets, the Push function is impure itself by altering the array and returning a push number index independent of the parameter value. . Whereas Concat on the other hand takes the array and concatenates it with the other array producing a whole new array without side effects. Also, the return value is a concatenation of the previous array.
Remember that Pure functions are important as they simplify unit testing without any side effects and no need for dependency injection. They also avoid tight coupling and make it harder to break your application by not having any side effects. These principles are coming together with Immutability concept of ES6 by giving preference to const over let usage.

What is the purpose of the let keyword
The let statement declares a block scope local variable. Hence the variables defined with let keyword are limited in scope to the block, statement, or expression on which it is used. Whereas variables declared with the var keyword used to define a variable globally, or locally to an entire function regardless of block scope.
Let's take an example to demonstrate the usage, let counter = 30;

```
if (counter === 30) { let counter = 31;
console.log(counter); // 31
}

console.log(counter); // 30 (because the variable in if block won't exist here)
```

What is the difference between let and var
You can list out the differences in a tabular format var let

It is been available from the beginning of JavaScriptIntroduced as part of ES6 It has function scope
It has block scope
Variables will be hoisted Hoisted but not initialized Let's take an example to see the difference,

```
function userDetails(username) { if (username) {
console.log(salary); // undefined due to hoisting
console.log(age); // ReferenceError: Cannot access 'age' before initialization let age = 30;
var salary = 10000;
}
console.log(salary); //10000 (accessible to due function scope) console.log(age); //error: age is not defined(due to block scope)
}
userDetails("John");
```

What is the reason to choose the name let as a keyword
let is a mathematical statement that was adopted by early programming languages like Scheme and Basic. It has been borrowed from dozens of other languages that use let already as a traditional keyword as close to var as possible.

How do you redeclare variables in switch block without an error
If you try to redeclare variables in a switch block then it will cause errors because there is only one block. For example, the below code block throws a syntax error as below,

```
let counter = 1; switch (x) { case 0:

let name; break;

case 1:
let name; // SyntaxError for redeclaration. break;
}
```

To avoid this error, you can create a nested block inside a case clause and create a new block scoped lexical environment.

```
let counter = 1; switch (x) { case 0: {
let name; break;
}
case 1: {
let name; // No SyntaxError for redeclaration. break;
}
}
```

What is the Temporal Dead Zone
The Temporal Dead Zone is a behavior in JavaScript that occurs when declaring a variable with the let and const keywords, but not with var. In ECMAScript 6, accessing a let or const variable before its declaration (within its scope) causes a ReferenceError. The time span when that happens, between the creation of a variable's binding and its declaration, is called the temporal dead zone.
Let's see this behavior with an example, function somemethod() { console.log(counter1); // undefined console.log(counter2); // ReferenceError var counter1 = 1;

```
let counter2 = 2;

}
```

What is IIFE(Immediately Invoked Function Expression)
IIFE (Immediately Invoked Function Expression) is a JavaScript function that runs as soon as it is defined. The signature of it would be as below,

```
(function () {
// logic here
})();
```

The primary reason to use an IIFE is to obtain data privacy because any variables declared within the IIFE cannot be accessed by the outside world. i.e, If you try to access variables with IIFE then it throws an error as below,

```
(function () {
var message = "IIFE"; console.log(message);
})();
console.log(message); //Error: message is not defined
```

How do you decode or encode a URL in JavaScript?
encodeURI() function is used to encode an URL. This function requires a URL string as a parameter and return that encoded string. decodeURI() function is used to decode an URL. This function requires an encoded URL string as parameter and return that decoded string.
Note: If you want to encode characters such as / ? : @ & = + $ # then you need to use encodeURIComponent().

```
let uri = "employeeDetails?name=john&occupation=manager"; let encoded_uri = encodeURI(uri);
let decoded_uri = decodeURI(encoded_uri);
```

What is memoization
Memoization is a programming technique which attempts to increase a function's performance by caching its previously computed results. Each time a memoized function is called, its parameters are used to index the cache. If the data is present, then it can be returned, without executing the entire

function. Otherwise the function is executed and then the result is added to the cache. Let's take an example of adding function with memoization,

```
const memoizAddition = () => {
 let cache = {};
 return (value) => {if
  (value in cache) {
   console.log("Fetching from cache");
  return cache[value]; // Here, cache.value cannot be used as property name starts with the number
```

which is not a valid JavaScript identifier. Hence, can only be accessed using the square bracket notation.


```
  } else {
   console.log("Calculating result");let
   result = value + 20; cache[value] =
   result;
   return result;
  }
 };
};
// returned function from memoizAddition
const addition = memoizAddition();
console.log(addition(20)); //output: 40 calculated
console.log(addition(20)); //output: 40 cached
```

What is Hoisting
Hoisting is a JavaScript mechanism where variables, function declarations and classes are moved to the top of their scope before code execution. Remember that JavaScript only hoists declarations, not initialisation. Let's take a simple example of variable hoisting,

```
console.log(message); //output : undefined
var message = "The variable Has been hoisted";
```

The above code looks like as below to the interpreter, var message;

```
console.log(message);
message = "The variable Has been hoisted";
```

In the same fashion, function declarations are hoisted too message("Good morning"); //Good morning

```
function message(name) { console.log(name);
}
```

This hoisting makes functions to be safely used in code before they are declared.

What are classes in ES6
In ES6, Javascript classes are primarily syntactic sugar over JavaScript's existing prototype-based inheritance. For example, the prototype based inheritance written in function expression as below,

```
function Bike(model, color) {
this.model = model; this.color = color;
}


Bike.prototype.getDetails = function () {
return this.model + " bike has" + this.color + " color";
};
```

Whereas ES6 classes can be defined as an alternative class Bike {

```
constructor(color, model) { this.color = color; this.model = model;
```

```
}

getDetails() {
return this.model + " bike has" + this.color + " color";
}
}
```

What are closures

A closure is the combination of a function and the lexical environment within which that function was declared. i.e, It is an inner function that has access to the outer or enclosing function's variables. The closure has three scope chains
Own scope where variables defined between its curly brackets Outer function's variables
Global variables
Let's take an example of closure concept, function Welcome(name) {

```
var greetingInfo = function (message) { console.log(message + " " + name);
};
return greetingInfo;
}
var myFunction = Welcome("John"); myFunction("Welcome "); //Output: Welcome John
myFunction("Hello Mr."); //output: Hello Mr.John
```

As per the above code, the inner function(i.e, greetingInfo) has access to the variables in the outer function scope(i.e, Welcome) even after the outer function has returned.

What are modules
Modules refer to small units of independent, reusable code and also act as the foundation of many JavaScript design patterns. Most of the JavaScript modules export an object literal, a function, or a constructor

Why do you need modules
Below are the list of benefits using modules in javascript ecosystem Maintainability
Reusability Namespacing

What is scope in javascript

Scope is the accessibility of variables, functions, and objects in some particular part of your code during runtime. In other words, scope determines the visibility of variables and other resources in areas of your code.

What is a service worker
A Service worker is basically a script (JavaScript file) that runs in the background, separate from a web page and provides features that don't need a web page or user interaction. Some of the major features of service workers are Rich offline experiences(offline first web application development), periodic background syncs, push notifications, intercept and handle network requests and

programmatically managing a cache of responses.

How do you manipulate DOM using a service worker
Service worker can't access the DOM directly. But it can communicate with the pages it controls by responding to messages sent via the postMessage interface, and those pages can manipulate the DOM.

How do you reuse information across service worker restarts
The problem with service worker is that it gets terminated when not in use, and restarted when it's next needed, so you cannot rely on global state within a service worker's onfetch and onmessage handlers. In this case, service workers will have access to IndexedDB API in order to persist and reuse across restarts.

What is IndexedDB
IndexedDB is a low-level API for client-side storage of larger amounts of structured data, including files/blobs. This API uses indexes to enable high-performance searches of this data.

What is web storage
Web storage is an API that provides a mechanism by which browsers can store key/value pairs locally within the user's browser, in a much more intuitive fashion than using cookies. The web storage provides two mechanisms for storing data on the client.
Local storage: It stores data for current origin with no expiration date.
Session storage: It stores data for one session and the data is lost when the browser tab is closed.

What is a post message

Post message is a method that enables cross-origin communication between Window objects.(i.e, between a page and a pop-up that it spawned, or between a page and an iframe embedded within it). Generally, scripts on different pages are allowed to access each other if and only if the pages follow same-origin policy(i.e, pages share the same protocol, port number, and host).

What is a Cookie
A cookie is a piece of data that is stored on your computer to be accessed by your browser. Cookies are saved as key/value pairs. For example, you can create a cookie named username as below,
document.cookie = "username=John";

Why do you need a Cookie
Cookies are used to remember information about the user profile(such as username). It basically involves two steps,
When a user visits a web page, the user profile can be stored in a cookie. Next time the user visits the page, the cookie remembers the user profile.

What are the options in a cookie
There are few below options available for a cookie,

By default, the cookie is deleted when the browser is closed but you can change this behavior by setting expiry date (in UTC time).
document.cookie = "username=John; expires=Sat, 8 Jun 2019 12:00:00 UTC";
By default, the cookie belongs to a current page. But you can tell the browser what path the cookie belongs to using a path parameter.
document.cookie = "username=John; path=/services";

How do you delete a cookie
You can delete a cookie by setting the expiry date as a passed date. You don't need to specify a cookie value in this case. For example, you can delete a username cookie in the current page as below. document.cookie =
"username=; expires=Fri, 07 Jun 2019 00:00:00 UTC; path=/;";

Note: You should define the cookie path option to ensure that you delete the right cookie. Some browsers doesn't allow to delete a cookie unless you specify a path parameter.

What are the differences between cookie, local storage and session storage
Below are some of the differences between cookie, local storage and session storage, Feature Cookie Local storage Session storage
Accessed on client or server side Both server-side & client-side client-side only client-side only Lifetime As configured using Expires option until deleted until tab is closed
SSL support Supported Not supported Not supported Maximum data size 4KB 5 MB 5MB

What is the main difference between localStorage and sessionStorage
LocalStorage is the same as SessionStorage but it persists the data even when the browser is closed and reopened(i.e it has no expiration time) whereas in sessionStorage data gets cleared when the page session ends.

How do you access web storage
The Window object implements the WindowLocalStorage and WindowSessionStorage objects which has localStorage(window.localStorage) and sessionStorage(window.sessionStorage) properties respectively. These properties create an instance of the Storage object, through which data items can be set, retrieved and removed for a specific domain and storage type (session or local). For example, you can read and write on local storage objects as below
localStorage.setItem("logo", document.getElementById("logo").value);
localStorage.getItem("logo");

What are the methods available on session storage
The session storage provided methods for reading, writing and clearing the session data
// Save data to sessionStorage sessionStorage.setItem("key", "value");

// Get saved data from sessionStorage let data = sessionStorage.getItem("key");

// Remove saved data from sessionStorage sessionStorage.removeItem("key");

// Remove all saved data from sessionStorage sessionStorage.clear();

What is a storage event and its event handler
The StorageEvent is an event that fires when a storage area has been changed in the context of another document. Whereas onstorage property is an EventHandler for processing storage events. The syntax would be as below
window.onstorage = functionRef;
Let's take the example usage of onstorage event handler which logs the storage key and it's values
window.onstorage = function (e) {
console.log("The " + e.key +
" key has been changed from " + e.oldValue +
" to " + e.newValue + "."
);
};

Why do you need web storage
Web storage is more secure, and large amounts of data can be stored locally, without affecting website performance. Also, the information is never transferred to the server. Hence this is a more recommended approach than Cookies.

How do you check web storage browser support
You need to check browser support for localStorage and sessionStorage before using web storage,

if (typeof Storage !== "undefined") {
// Code for localStorage/sessionStorage.
} else {
// Sorry! No Web Storage support..
}

How do you check web workers browser support
You need to check browser support for web workers before using it if (typeof Worker !== "undefined") {
// code for Web worker support.
} else {
// Sorry! No Web Worker support..
}

Give an example of a web worker
You need to follow below steps to start using web workers for counting example
Create a Web Worker File: You need to write a script to increment the count value. Let's name it as counter.js
let i = 0;

```
function timedCount() { i = i + 1; postMessage(i);
setTimeout("timedCount()", 500);
}


timedCount();
```

Here postMessage() method is used to post a message back to the HTML page
Create a Web Worker Object: You can create a web worker object by checking for browser support. Let's name this file as web_worker_example.js

```
if (typeof w == "undefined") {
w = new Worker("counter.js");

}
and we can receive messages from web worker w.onmessage = function (event) {
document.getElementById("message").innerHTML = event.data;
};
```

Terminate a Web Worker: Web workers will continue to listen for messages (even after the external script is finished) until it is terminated. You can use the terminate() method to terminate listening to the messages.

```
w.terminate();
```

Reuse the Web Worker: If you set the worker variable to undefined you can reuse the code w = undefined;

What are the restrictions of web workers on DOM
WebWorkers don't have access to below javascript objects since they are defined in an external files Window object
Document object Parent object

What is a promise
A promise is an object that may produce a single value some time in the future with either a resolved value or a reason that it's not resolved(for example, network error). It will be in one of the 3 possible states: fulfilled, rejected, or pending.
The syntax of Promise creation looks like below,

```
const promise = new Promise(function (resolve, reject) {
// promise description
});
```

The usage of a promise would be as below, const promise = new Promise(

```
(resolve) => { setTimeout(() => {
resolve("I'm a Promise!");
}, 5000);

},
(reject) => {}
);
```


promise.then((value) => console.log(value)); The action flow of a promise will be as below,

Why do you need a promise
Promises are used to handle asynchronous operations. They provide an alternative approach for callbacks by reducing the callback hell and writing the cleaner code.

What are the three states of promise Promises have three states:
Pending: This is an initial state of the Promise before an operation begins Fulfilled: This state indicates that the specified operation was completed.
Rejected: This state indicates that the operation did not complete. In this case an error value will be thrown.

What is a callback function
A callback function is a function passed into another function as an argument. This function is invoked inside the outer function to complete an action. Let's take a simple example of how to use callback function

```
function callbackFunction(name) { console.log("Hello " + name);
}


function outerFunction(callback) {
let name = prompt("Please enter your name."); callback(name);
}

outerFunction(callbackFunction);
```

Why do we need callbacks
The callbacks are needed because javascript is an event driven language. That means instead of waiting for a response javascript will keep executing while listening for other events. Let's take an example with the first function invoking an API call(simulated by setTimeout) and the next function which logs the message.

```
function firstFunction() {
// Simulate a code delay setTimeout(function () { console.log("First function called");
}, 1000);
}
function secondFunction() { console.log("Second function called");
}
firstFunction(); secondFunction();

Output;
// Second function called
// First function called
```

As observed from the output, javascript didn't wait for the response of the first function and the remaining code block got executed. So callbacks are used in a way to make sure that certain code doesn't execute until the other code finishes execution.

What is a callback hell
Callback Hell is an anti-pattern with multiple nested callbacks which makes code hard to read and debug when dealing with asynchronous logic. The callback hell looks like below,

```
async1(function(){ async2(function(){
async3(function(){

async4(function(){
....

});
```

What are server-sent events
Server-sent events (SSE) is a server push technology enabling a browser to receive automatic updates from a server via HTTP connection without resorting to polling. These are a one way communications channel - events flow from server to client only. This has been used in Facebook/Twitter updates, stock price updates, news feeds etc.

How do you receive server-sent event notifications
The EventSource object is used to receive server-sent event notifications. For example, you can receive messages from server as below,

```
if (typeof EventSource !== "undefined") {
var source = new EventSource("sse_generator.js"); source.onmessage = function (event) {
document.getElementById("output").innerHTML += event.data + "<br>";
};
}
```

How do you check browser support for server-sent events
You can perform browser support for server-sent events before using it as below,

```
if (typeof EventSource !== "undefined") {
// Server-sent events supported. Let's have some code here!
} else {
// No server-sent events supported
}
```

What are the events available for server sent events

Below are the list of events available for server sent events Event Description
onopen It is used when a connection to the server is opened onmessage This event is used when a message is received onerror It happens when an error occurs

What are the main rules of promise
A promise must follow a specific set of rules,
A promise is an object that supplies a standard-compliant .then() method A pending promise may transition into either fulfilled or rejected state
A fulfilled or rejected promise is settled and it must not transition into any other state. Once a promise is settled, the value must not change.

What is callback in callback
You can nest one callback inside in another callback to execute the actions sequentially one by one. This is known as callbacks in callbacks.

```
loadScript("/script1.js", function (script) { console.log("first script is loaded");

loadScript("/script2.js", function (script) { console.log("second script is loaded");

loadScript("/script3.js", function (script) { console.log("third script is loaded");
// after all scripts are loaded
});
});
});
```

What is promise chaining
The process of executing a sequence of asynchronous tasks one after another using promises is known as Promise chaining. Let's take an example of promise chaining for calculating the final result,

```
new Promise(function (resolve, reject) { setTimeout(() => resolve(1), 1000);
})
.then(function (result) { console.log(result); // 1 return result * 2;
})
.then(function (result) { console.log(result); // 2 return result * 3;
})
.then(function (result) { console.log(result); // 6 return result * 4;
});
```

In the above handlers, the result is passed to the chain of .then() handlers with the below work flow,
The initial promise resolves in 1 second,
After that .then handler is called by logging the result(1) and then return a promise with the value of result * 2.
After that the value passed to the next .then handler by logging the result(2) and return a promise with result * 3.
Finally the value passed to the last .then handler by logging the result(6) and return a promise with result * 4.

What is promise.all
Promise.all is a promise that takes an array of promises as an input (an iterable), and it gets

resolved when all the promises get resolved or any one of them gets rejected. For example, the syntax of promise.all method is below,
Promise.all([Promise1, Promise2, Promise3]) .then(result) => { console.log(result) }) .catch(error => console.log(`Error in promises ${error}`))
Note: Remember that the order of the promises(output the result) is maintained as per input order.

What is the purpose of the race method in promise
Promise.race() method will return the promise instance which is firstly resolved or rejected. Let's take an example of race() method where promise2 is resolved first

```
var promise1 = new Promise(function (resolve, reject) { setTimeout(resolve, 500, "one");
});
var promise2 = new Promise(function (resolve, reject) { setTimeout(resolve, 100, "two");
});


Promise.race([promise1, promise2]).then(function (value) {
console.log(value); // "two" // Both promises will resolve, but promise2 is faster
});
```

What is a strict mode in javascript
Strict Mode is a new feature in ECMAScript 5 that allows you to place a program, or a function, in a "strict" operating context. This way it prevents certain actions from being taken and throws more exceptions. The literal expression "use strict"; instructs the browser to use the javascript code in the Strict mode.

Why do you need strict mode
Strict mode is useful to write "secure" JavaScript by notifying "bad syntax" into real errors. For example, it eliminates accidentally creating a global variable by throwing an error and also throws an error for assignment to a non-writable property, a getter-only property, a non-existing property, a non-existing variable, or a non-existing object.

How do you declare strict mode
The strict mode is declared by adding "use strict"; to the beginning of a script or a function. If declared at the beginning of a script, it has global scope.

```
"use strict";
x = 3.14; // This will cause an error because x is not declared and if you declare inside a function, it has local scope

x = 3.14; // This will not cause an error. myFunction();

function myFunction() { "use strict";
y = 3.14; // This will cause an error
}
```


What is the purpose of double exclamation
The double exclamation or negation(!!) ensures the resulting type is a boolean. If it was falsey (e.g. 0, null, undefined, etc.), it will be false, otherwise, true. For example, you can test IE version using this expression as below,

```
let isIE8 = false;
isIE8 = !!navigator.userAgent.match(/MSIE 8.0/); console.log(isIE8); // returns true or false
```

If you don't use this expression then it returns the original value.

```
console.log(navigator.userAgent.match(/MSIE 8.0/)); // returns either an Array or null Note: The expression !! is not an operator, but it is just twice of ! operator.
```

What is the purpose of the delete operator
The delete keyword is used to delete the property as well as its value. var user = { name: "John", age: 20 };

```
delete user.age;


console.log(user); // {name: "John"}
```

What is typeof operator
You can use the JavaScript typeof operator to find the type of a JavaScript variable. It returns the type of a variable or an expression.

```
typeof "John Abraham"; // Returns "string" typeof (1 + 2); // Returns "number"
```

What is undefined property
The undefined property indicates that a variable has not been assigned a value, or declared but not initialized at all. The type of undefined value is undefined too.

```
var user; // Value is undefined, type is undefined console.log(typeof user); //undefined
```

Any variable can be emptied by setting the value to undefined. user = undefined;

What is null value
The value null represents the intentional absence of any object value. It is one of JavaScript's primitive values. The type of null value is object. You can empty the variable by setting the value to null.

```
var user = null; console.log(typeof user); //object
```

What is the difference between null and undefined
Below are the main differences between null and undefined, Null Undefined
It is an assignment value which indicates that variable points to no object. It is not an assignment value where a variable has been declared but has not yet been assigned a value.
Type of null is object Type of undefined is undefined
The null value is a primitive value that represents the null, empty, or non-existent reference.
The undefined value is a primitive value used when a variable has not been assigned a value.
Indicates the absence of a value for a variable Indicates absence of variable itself

Converted to zero (0) while performing primitive operations Converted to NaN while performing primitive operations

What is eval
The eval() function evaluates JavaScript code represented as a string. The string can be a JavaScript expression, variable, statement, or sequence of statements.
console.log(eval("1 + 2")); // 3

What is the difference between window and document

Below are the main differences between window and document, Window Document
It is the root level element in any web page It is the direct child of the window object. This is also known as Document Object Model(DOM)
By default window object is available implicitly in the page You can access it via window.document or document.
It has methods like alert(), confirm() and properties like document, location It provides methods like getElementById, getElementsByTagName, createElement etc

How do you access history in javascript
The window.history object contains the browser's history. You can load previous and next URLs in the history using back() and next() methods.
function goBack() { window.history.back();
}
function goForward() { window.history.forward();
}
Note: You can also access history without window prefix.

How do you detect caps lock key turned on or not
The mouseEvent getModifierState() is used to return a boolean value that indicates whether the specified modifier key is activated or not. The modifiers such as CapsLock, ScrollLock and NumLock are activated when they are clicked, and deactivated when they are clicked again.
Let's take an input element to detect the CapsLock on/off behavior with an example,
<input type="password" onmousedown="enterInput(event)" />

<script>
function enterInput(e) {
var flag = e.getModifierState("CapsLock");

if (flag) {
document.getElementById("feedback").innerHTML = "CapsLock activated";
} else { document.getElementById("feedback").innerHTML = "CapsLock not activated";

}
}
</script>

What is isNaN
The isNaN() function is used to determine whether a value is an illegal number (Not-a-Number) or not. i.e, This function returns true if the value equates to NaN. Otherwise it returns false.
isNaN("Hello"); //true isNaN("100"); //false

What are the differences between undeclared and undefined variables
Below are the major differences between undeclared(not defined) and undefined variables,
undeclared undefined
These variables do not exist in a program and are not declared These variables declared in the program but have not assigned any value
If you try to read the value of an undeclared variable, then a runtime error is encountered If you try to read the value of an undefined variable, an undefined value is returned.

What are global variables
Global variables are those that are available throughout the length of the code without any scope. The var keyword is used to declare a local variable but if you omit it then it will become global variable msg = "Hello"; // var is missing, it becomes global variable

What are the problems with global variables
The problem with global variables is the conflict of variable names of local and global scope. It is also difficult to debug and test the code that relies on global variables.

What is NaN property
The NaN property is a global property that represents "Not-a-Number" value. i.e, It indicates that a value is not a legal number. It is very rare to use NaN in a program but it can be used as return value for few cases
Math.sqrt(-1); parseInt("Hello");

What is the purpose of isFinite function
The isFinite() function is used to determine whether a number is a finite, legal number. It returns false if the value is +infinity, -infinity, or NaN (Not-a-Number), otherwise it returns true.
isFinite(Infinity); // false isFinite(NaN); // false isFinite(-Infinity); // false

isFinite(100); // true

What is an event flow
Event flow is the order in which event is received on the web page. When you click an element that is nested in various other elements, before your click actually reaches its destination, or target element, it must trigger the click event for each of its parent elements first, starting at the top with the global window object. There are two ways of event flow

Top to Bottom(Event Capturing) Bottom to Top (Event Bubbling)

What is event bubbling
Event bubbling is a type of event propagation where the event first triggers on the innermost target element, and then successively triggers on the ancestors (parents) of the target element in the same nesting hierarchy till it reaches the outermost DOM element.

What is event capturing

Event capturing is a type of event propagation where the event is first captured by the outermost element, and then successively triggers on the descendants (children) of the target element in the same nesting hierarchy till it reaches the innermost DOM element.

How do you submit a form using JavaScript
You can submit a form using document.forms[0].submit(). All the form input's information is submitted using onsubmit event handler
function submit() { document.forms[0].submit();
}

How do you find operating system details
The window.navigator object contains information about the visitor's browser OS details. Some of the OS properties are available under platform property,
console.log(navigator.platform);

What is the difference between document load and DOMContentLoaded events
The DOMContentLoaded event is fired when the initial HTML document has been completely loaded and parsed, without waiting for assets(stylesheets, images, and subframes) to finish loading. Whereas The load event is fired when the whole page has loaded, including all dependent resources(stylesheets, images).

What is the difference between native, host and user objects
Native objects are objects that are part of the JavaScript language defined by the ECMAScript specification. For example, String, Math, RegExp, Object, Function etc core objects defined in the ECMAScript spec. Host objects are objects provided by the browser or runtime environment (Node). For example, window, XmlHttpRequest, DOM nodes etc are considered as host objects. User objects are objects defined in the javascript code. For example, User objects created for profile information.

What are the tools or techniques used for debugging JavaScript code You can use below tools or techniques for debugging javascript Chrome Devtools

debugger statement
Good old console.log statement

What are the pros and cons of promises over callbacks

Below are the list of pros and cons of promises over callbacks, Pros:
It avoids callback hell which is unreadable
Easy to write sequential asynchronous code with .then() Easy to write parallel asynchronous code with Promise.all()
Solves some of the common problems of callbacks(call the callback too late, too early, many times and swallow errors/exceptions)
Cons:
It makes little complex code
You need to load a polyfill if ES6 is not supported

What is the difference between an attribute and a property
Attributes are defined on the HTML markup whereas properties are defined on the DOM. For example, the below HTML element has 2 attributes type and value,
<input type="text" value="Name:">
You can retrieve the attribute value as below, const input = document.querySelector("input");
console.log(input.getAttribute("value")); // Good morning console.log(input.value); // Good morning
And after you change the value of the text field to "Good evening", it becomes like
console.log(input.getAttribute("value")); // Good evening console.log(input.value); // Good evening

What is same-origin policy
The same-origin policy is a policy that prevents JavaScript from making requests across domain boundaries. An origin is defined as a combination of URI scheme, hostname, and port number. If you enable this policy then it prevents a malicious script on one page from obtaining access to sensitive data on another web page using Document Object Model(DOM).

What is the purpose of void 0
Void(0) is used to prevent the page from refreshing. This will be helpful to eliminate the unwanted side- effect, because it will return the undefined primitive value. It is commonly used for HTML documents that use href="JavaScript:Void(0);" within an <a> element. i.e, when you click a link, the browser loads a new page or refreshes the same page. But this behavior will be prevented using this expression. For example, the below link notify the message without reloading the page
 Click Me!

Is JavaScript a compiled or interpreted language
JavaScript is an interpreted language, not a compiled language. An interpreter in the browser reads over the JavaScript code, interprets each line, and runs it. Nowadays modern browsers use a technology known as Just-In-Time (JIT) compilation, which compiles JavaScript to executable bytecode just as it is about to run.

Is JavaScript a case-sensitive language
Yes, JavaScript is a case sensitive language. The language keywords, variables, function & object names, and any other identifiers must always be typed with a consistent capitalization of letters.

Is there any relation between Java and JavaScript
No, they are entirely two different programming languages and have nothing to do with each other. But both of them are Object Oriented Programming languages and like many other languages, they follow similar syntax for basic features(if, else, for, switch, break, continue etc).

What are events
Events are "things" that happen to HTML elements. When JavaScript is used in HTML pages, JavaScript can react on these events. Some of the examples of HTML events are,
Web page has finished loading Input field was changed Button was clicked

Let's describe the behavior of click event for button element,

```
<!doctype html>
<html>
<head>
<script>
function greeting() { alert('Hello! Good morning');
}
</script>
</head>
<body>
<button type="button" onclick="greeting()">Click me</button>
</body>
</html>
```

Who created javascript
JavaScript was created by Brendan Eich in 1995 during his time at Netscape Communications. Initially it was developed under the name Mocha, but later the language was officially called LiveScript when it first shipped in beta releases of Netscape.

What is the use of preventDefault method
The preventDefault() method cancels the event if it is cancelable, meaning that the default action or behaviour that belongs to the event will not occur. For example, prevent form submission when clicking on submit button and prevent opening the page URL when clicking on hyperlink are some common use cases.

```
document
.getElementByld("link")
.addEventListener("click", function (event) { event.preventDefault();
});
```

Note: Remember that not all events are cancelable.

What is the use of stopPropagation method
The stopPropagation method is used to stop the event from bubbling up the event chain. For example, the below nested divs with stopPropagation method prevents default event propagation when clicking on nested div(Div1)

```
<p>Click DIV1 Element</p>
<div onclick="secondFunc()">DIV 2
<div onclick="firstFunc(event)">DIV 1</div>
</div>


<script>
function firstFunc(event) { alert("DIV 1"); event.stopPropagation();
}


function secondFunc() { alert("DIV 2");
}
</script>
```

What are the steps involved in return false usage
The return false statement in event handlers performs the below steps, First it stops the browser's default action or behaviour.
It prevents the event from propagating the DOM
Stops callback execution and returns immediately when called.

What is BOM
The Browser Object Model (BOM) allows JavaScript to "talk to" the browser. It consists of the objects navigator, history, screen, location and document which are children of the window. The Browser Object Model is not standardized and can change based on different browsers.

What is the use of setTimeout
The setTimeout() method is used to call a function or evaluate an expression after a specified number of milliseconds. For example, let's log a message after 2 seconds using setTimeout method,

```
setTimeout(function () {
console.log("Good morning");
}, 2000);
```

What is the use of setInterval
The setInterval() method is used to call a function or evaluate an expression at specified intervals (in milliseconds). For example, let's log a message after 2 seconds using setInterval method,

```
setInterval(function () {
console.log("Good morning");
}, 2000);
```

Why is JavaScript treated as Single threaded
JavaScript is a single-threaded language. Because the language specification does not allow the programmer to write code so that the interpreter can run parts of it in parallel in multiple threads or processes. Whereas languages like java, go, C++ can make multi-threaded and multi-process programs.

What is an event delegation
Event delegation is a technique for listening to events where you delegate a parent element as the listener for all of the events that happen inside it.
For example, if you wanted to detect field changes in inside a specific form, you can use event delegation technique,
var form = document.querySelector("#registration-form");

// Listen for changes to fields inside the form form.addEventListener(
"input",
function (event) {
// Log the field that was changed console.log(event.target);

},
false
);

What is ECMAScript
ECMAScript is the scripting language that forms the basis of JavaScript. ECMAScript standardized by the ECMA International standards organization in the ECMA-262 and ECMA-402 specifications. The first edition of ECMAScript was released in 1997.

What is JSON
JSON (JavaScript Object Notation) is a lightweight format that is used for data interchanging. It is based on a subset of JavaScript language in the way objects are built in JavaScript.

What are the syntax rules of JSON Below are the list of syntax rules of JSON The data is in name/value pairs
The data is separated by commas Curly braces hold objects
Square brackets hold arrays

What is the purpose JSON stringify
When sending data to a web server, the data has to be in a string format. You can achieve this by converting JSON object into a string using stringify() method.
var userJSON = { name: "John", age: 31 }; var userString = JSON.stringify(userJSON);
console.log(userString); //"{"name":"John","age":31}"

How do you parse JSON string

When receiving the data from a web server, the data is always in a string format. But you can convert this string value to a javascript object using parse() method.
var userString = '{"name":"John","age":31}'; var userJSON = JSON.parse(userString);
console.log(userJSON); // {name: "John", age: 31}

Why do you need JSON
When exchanging data between a browser and a server, the data can only be text. Since JSON is text only, it can easily be sent to and from a server, and used as a data format by any programming language.

What are PWAs
Progressive web applications (PWAs) are a type of mobile app delivered through the web, built using common web technologies including HTML, CSS and JavaScript. These PWAs are deployed to servers, accessible through URLs, and indexed by search engines.

What is the purpose of clearTimeout method
The clearTimeout() function is used in javascript to clear the timeout which has been set by setTimeout()function before that. i.e, The return value of setTimeout() function is stored in a variable and it's passed into the clearTimeout() function to clear the timer.
For example, the below setTimeout method is used to display the message after 3 seconds. This timeout can be cleared by the clearTimeout() method.
<script> var msg;
function greeting() { alert('Good morning');
}
function start() {
msg =setTimeout(greeting, 3000);

}

function stop() { clearTimeout(msg);
}
</script>

What is the purpose of clearInterval method
The clearInterval() function is used in javascript to clear the interval which has been set by setInterval() function. i.e, The return value returned by setInterval() function is stored in a variable and it's passed into the clearInterval() function to clear the interval.
For example, the below setInterval method is used to display the message for every 3 seconds. This interval can be cleared by the clearInterval() method.
<script> var msg;
function greeting() { alert('Good morning');
}
function start() {
msg = setInterval(greeting, 3000);

```
}


function stop() { clearInterval(msg);
}
</script>
```

How do you redirect new page in javascript
In vanilla javascript, you can redirect to a new page using the location property of window object. The syntax would be as follows,

```
function redirect() {
window.location.href = "newPage.html";
}
```

How do you check whether a string contains a substring
There are 3 possible ways to check whether a string contains a substring or not,
Using includes: ES6 provided String.prototype.includes method to test a string contains a substring

```
var mainString = "hello",

subString = "hell"; mainString.includes(subString);
```

Using indexOf: In an ES5 or older environment, you can use String.prototype.indexOf which returns the index of a substring. If the index value is not equal to -1 then it means the substring exists in the main string.

```
var mainString = "hello", subString = "hell";
mainString.indexOf(subString) !== -1;
```

Using RegEx: The advanced solution is using Regular expression's test method(RegExp.test), which allows for testing for against regular expressions

```
var mainString = "hello", regex = /hell/; regex.test(mainString);
```

How do you validate an email in javascript
You can validate an email in javascript using regular expressions. It is recommended to do validations on the server side instead of the client side. Because the javascript can be disabled on the client side. function validateEmail(email) {

```
var re =
/^(([^<>()\[\]\\.,;:\s@"]+(\.[^<>()\[\]\\.,;:\s@"]+)*)|(".+"))@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-
9]{1,3}\.[0- 9]{1,3}\])|(([a-zA-Z\-0-9]+\.)+[a-zA-Z]{2,}))$/;
return re.test(String(email).toLowerCase());
}
```

The above regular expression accepts unicode characters. How do you get the current url with javascript
You can use window.location.href expression to get the current url path and you can use the same expression for updating the URL too. You can also use document.URL for read-only purposes but

this solution has issues in FF.

```
console.log("location.href", window.location.href); // Returns full URL
```

What are the various url properties of location object

The below Location object properties can be used to access URL components of the page, href - The entire URL
protocol - The protocol of the URL
host - The hostname and port of the URL hostname - The hostname of the URL port - The port number in the URL pathname - The path name of the URL search - The query portion of the URL hash - The anchor portion of the URL

How do get query string values in javascript
You can use URLSearchParams to get query string values in javascript. Let's see an example to get the client code value from URL query string,

```
const urlParams = new URLSearchParams(window.location.search); const clientCode =
urlParams.get("clientCode");
```

How do you check if a key exists in an object
You can check whether a key exists in an object or not using three approaches,
Using in operator: You can use the in operator whether a key exists in an object or not "key" in obj; and If you want to check if a key doesn't exist, remember to use parenthesis,

```
!("key" in obj);
```

Using hasOwnProperty method: You can use hasOwnProperty to particularly test for properties of the object instance (and not inherited properties)

```
obj.hasOwnProperty("key"); // true
```

Using undefined comparison: If you access a non-existing property from an object, the result is undefined. Let's compare the properties against undefined to determine the existence of the property. const user = {

```
name: "John",
};


console.log(user.name !== undefined); // true

console.log(user.nickName !== undefined); // false
```

How do you loop through or enumerate javascript object
You can use the for-in loop to loop through javascript object. You can also make sure that the key you get is an actual property of an object, and doesn't come from the prototype using hasOwnProperty method.

```
var object = { k1: "value1",
k2: "value2",
k3: "value3",
};
```

```
for (var key in object) {
if (object.hasOwnProperty(key)) {
console.log(key + " -> " + object[key]); // k1 -> value1 ...
}
}
```

How do you test for an empty object
There are different solutions based on ECMAScript versions
Using Object entries(ECMA 7+): You can use object entries length along with constructor type.
Object.entries(obj).length === 0 && obj.construetor === Object; // Since date object length is 0, you need to check constructor check as well
Using Object keys(ECMA 5+): You can use object keys length along with constructor type.
Object.keys(obj).length === 0 && obj.constructor === Object; // Since date object length is 0, you need to check constructor check as well
Using for-in with hasOwnProperty(Pre-ECMA 5): You can use a for-in loop along with hasOwnProperty. function isEmpty(obj) {

```
for (var prop in obj) {
if (obj.hasOwnProperty(prop)) { return false;
}

}


return JSON.stringify(obj) === JSON.stringify({});
}
```

What is an arguments object
The arguments object is an Array-like object accessible inside functions that contains the values of the arguments passed to that function. For example, let's see how to use arguments object inside sum function,

```
function sum() { var total = 0;
for (var i = 0, len = arguments.length; i < len; ++i) { total += arguments[i];
}
return total;
}


sum(1, 2, 3); // returns 6
```

Note: You can't apply array methods on arguments object. But you can convert into a regular array as below.

```
var argsArray = Array.prototype.slice.call(arguments);
```

How do you make first letter of the string in an uppercase

You can create a function which uses a chain of string methods such as charAt, toUpperCase and slice methods to generate a string with the first letter in uppercase.

```
function capitalizeFirstLetter(string) {
return string.charAt(0).toUpperCase() + string.slice(1);
}
```

What are the pros and cons of for loop
The for-loop is a commonly used iteration syntax in javascript. It has both pros and cons Pros

Works on every environment
You can use break and continue flow control statements Cons
Too verbose Imperative
You might face one-by-off errors

How do you display the current date in javascript
You can use new Date() to generate a new Date object containing the current date and time. For example, let's display the current date in mm/dd/yyyy

```
var today = new Date();
var dd = String(today.getDate()).padStart(2, "0");
var mm = String(today.getMonth() + 1).padStart(2, "0"); //January is 0! var yyyy =
today.getFullYear();

today = mm + "/" + dd + "/" + yyyy; document.write(today);
```

How do you compare two date objects
You need to use date.getTime() method to compare date values instead of comparison operators (==, !=,
===, and !== operators) var d1 = new Date(); var d2 = new Date(d1);
console.log(d1.getTime() === d2.getTime()); //True console.log(d1 === d2); // False

How do you check if a string starts with another string
You can use ECMAScript 6's String.prototype.startsWith() method to check if a string starts with another string or not. But it is not yet supported in all browsers. Let's see an example to see this usage,

```
"Good morning".startsWith("Good"); // true "Good morning".startsWith("morning"); // false
```

How do you trim a string in javascript
JavaScript provided a trim method on string types to trim any whitespaces present at the beginning or ending of the string.

```
" Hello World ".trim(); //Hello World
```

If your browser(<IE9) doesn't support this method then you can use below polyfill. if

```
(!String.prototype.trim) {
(function () {
// Make sure we trim BOM and NBSP
var rtrim = /^[\s\uFEFF\xA0]+|[\s\uFEFF\xA0]+$/g; String.prototype.trim = function () {
```

```
return this.replace(rtrim, "");
};
})();
}
```

How do you add a key value pair in javascript
There are two possible solutions to add new properties to an object. Let's take a simple object to explain these solutions.

```
var object = { key1: value1, key2: value2,
};
```

Using dot notation: This solution is useful when you know the name of the property object.key3 = "value3";
Using square bracket notation: This solution is useful when the name of the property is dynamically determined.

```
obj["key3"] = "value3";
```

Is the !-- notation represents a special operator
No,that's not a special operator. But it is a combination of 2 standard operators one after the other,
A logical not (!)
A prefix decrement (--)

At first, the value decremented by one and then tested to see if it is equal to zero or not for determining the truthy/falsy value.

How do you assign default values to variables
You can use the logical or operator || in an assignment expression to provide a default value. The syntax looks like as below,

```
var a = b || c;
```

As per the above expression, variable 'a 'will get the value of 'c' only if 'b' is falsy (if is null, false, undefined, 0, empty string, or NaN), otherwise 'a' will get the value of 'b'.

How do you define multiline strings
You can define multiline string literals using the '\' character followed by line terminator. var str = "This is a \ very lengthy \ sentence!";
But if you have a space after the '\' character, the code will look exactly the same, but it will raise a SyntaxError.

What is an app shell model
An application shell (or app shell) architecture is one way to build a Progressive Web App that reliably and instantly loads on your users' screens, similar to what you see in native applications. It is useful for getting some initial HTML to the screen fast without a network.

Can we define properties for functions
Yes, We can define properties for functions because functions are also objects. fn = function (x) {
//Function code goes here


```
};


fn.name = "John";

fn.profile = function (y) {
//Profile code goes here
};
```

What is the way to find the number of parameters expected by a function
You can use function.length syntax to find the number of parameters expected by a function. Let's take an example of sum function to calculate the sum of numbers,

```
function sum(num1, num2, num3, num4) { return num1 + num2 + num3 + num4;
}
sum.length; // 4 is the number of parameters expected.
```

What is a polyfill
A polyfill is a piece of JS code used to provide modern functionality on older browsers that do not natively support it. For example, Silverlight plugin polyfill can be used to mimic the functionality of an HTML Canvas element on Microsoft Internet Explorer 7.

What are break and continue statements
The break statement is used to "jump out" of a loop. i.e, It breaks the loop and continues executing the code after the loop.

```
for (i = 0; i < 10; i++) { if (i === 5) {
break;
}
text += "Number: " + i + "<br>";
}
```

The continue statement is used to "jump over" one iteration in the loop. i.e, It breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

```
for (i = 0; i < 10; i++) { if (i === 5) { continue;
}

text += "Number: " + i + "<br>";
}
```

What are js labels
The label statement allows us to name loops and blocks in JavaScript. We can then use these labels to refer back to the code later. For example, the below code with labels avoids printing the numbers when they are same,

```
var i, j;


loop1: for (i = 0; i < 3; i++) { loop2: for (j = 0; j < 3; j++) { if (i === j) {
```

```
continue loop1;
}
console.log("i = " + i + ", j = " + j);
}
}


// Output is:
// "i = 1, j = 0"
// "i = 2, j = 0"
// "i = 2, j = 1"
```

What are the benefits of keeping declarations at the top

It is recommended to keep all declarations at the top of each script or function. The benefits of doing this are,

Gives cleaner code

It provides a single place to look for local variables Easy to avoid unwanted global variables

It reduces the possibility of unwanted re-declarations

What are the benefits of initializing variables

It is recommended to initialize variables because of the below benefits, It gives cleaner code

It provides a single place to initialize variables

Avoid undefined values in the code

What are the recommendations to create new object

It is recommended to avoid creating new objects using new Object(). Instead you can initialize values based on it's type to create the objects.

Assign {} instead of new Object()

Assign "" instead of new String() Assign

0 instead of new Number() Assign false

instead of new Boolean()Assign []

instead of new Array() Assign /()/

instead of new RegExp()

Assign function (){} instead of new Function()You

can define them as an example,

```
var v1 = {};
var v2 = ""; var v3
= 0; var v4 = false;
var v5 = []; var v6
= /()/;
var v7 = function () {};
```

How do you define JSON arrays

JSON arrays are written inside square brackets and arrays contain javascript objects. For example, the JSON array of users would be as below,

```
"users":[
  {"firstName":"John", "lastName":"Abrahm"},
  {"firstName":"Anna", "lastName":"Smith"},
  {"firstName":"Shane", "lastName":"Warn"}
]
```

How do you generate random integers

You can use Math.random() with Math.floor() to return random integers. For example, if you want generate random integers between 1 to 10, the multiplication factor should be 10,

```
Math.floor(Math.random() * 10) + 1; // returns a random integer from 1 to 10
Math.floor(Math.random() * 100) + 1; // returns a random integer from 1 to 100
```

Note: Math.random() returns a random number between 0 (inclusive), and 1 (exclusive)

Can you write a random integers function to print integers with in a range

Yes, you can create a proper random function to return a random number between min and max (both included)

```
function randomInteger(min, max) {
return Math.floor(Math.random() * (max - min + 1)) + min;
}
randomInteger(1, 100); // returns a random integer from 1 to 100 randomInteger(1, 1000); // returns a random integer from 1 to 1000
```

What is tree shaking

Tree shaking is a form of dead code elimination. It means that unused modules will not be included in the bundle during the build process and for that it relies on the static structure of ES2015 module syntax,(i.e. import and export). Initially this has been popularized by the ES2015 module bundler rollup.

What is the need of tree shaking

Tree Shaking can significantly reduce the code size in any application. i.e, The less code we send over the wire the more performant the application will be. For example, if we just want to create a "Hello World" Application using SPA frameworks then it will take around a few MBs, but by tree shaking it can bring down the size to just a few hundred KBs. Tree shaking is implemented in Rollup and Webpack bundlers.

Is it recommended to use eval

No, it allows arbitrary code to be run which causes a security problem. As we know that the eval() function is used to run text as code. In most of the cases, it should not be necessary to use it.

What is a Regular Expression
A regular expression is a sequence of characters that forms a search pattern. You can use this search pattern for searching data in a text. These can be used to perform all types of text search and text replace operations. Let's see the syntax format now,
/pattern/modifiers;
For example, the regular expression or search pattern with case-insensitive username would be,
/John/i;

What are the string methods available in Regular expression
Regular Expressions has two string methods: search() and replace(). The search() method uses an expression to search for a match, and returns the position of the match.
var msg = "Hello John";
var n = msg.search(/John/i); // 6
The replace() method is used to return a modified string where the pattern is replaced. var msg = "Hello John";
var n = msg.replace(/John/i, "Buttler"); // Hello Buttler

What are modifiers in regular expression
Modifiers can be used to perform case-insensitive and global searches. Let's list down some of the modifiers,
Modifier Description
i Perform case-insensitive matching
g Perform a global match rather than stops at first match m Perform multiline matching
Let's take an example of global modifier, var text = "Learn JS one by one";
var pattern = /one/g;
var result = text.match(pattern); // one,one

What are regular expression patterns
Regular Expressions provide a group of patterns in order to match characters. Basically they are categorized into 3 types,

Brackets: These are used to find a range of characters. For example, below are some use cases,
[abc]: Used to find any of the characters between the brackets(a,b,c)
[0-9]: Used to find any of the digits between the brackets (a|b): Used to find any of the alternatives separated with |
Metacharacters: These are characters with a special meaning For example, below are some use cases,
\d: Used to find a digit
\s: Used to find a whitespace character
\b: Used to find a match at the beginning or ending of a word
Quantifiers: These are useful to define quantities For example, below are some use cases, n+: Used to find matches for any string that contains at least one n
n*: Used to find matches for any string that contains zero or more occurrences of n n?: Used to find matches for any string that contains zero or one occurrences of n

What is a RegExp object
RegExp object is a regular expression object with predefined properties and methods. Let's see the simple usage of RegExp object,
var regexp = new RegExp("\\w+"); console.log(regexp);
// expected output: /\w+/

How do you search a string for a pattern
You can use the test() method of regular expression in order to search a string for a pattern, and return true or false depending on the result.
var pattern = /you/; console.log(pattern.test("How are you?")); //true

What is the purpose of exec method
The purpose of exec method is similar to test method but it executes a search for a match in a specified string and returns a result array, or null instead of returning true/false.
var pattern = /you/;
console.log(pattern.exec("How are you?")); //["you", index: 8, input: "How are you?", groups: undefined]

How do you change the style of a HTML element
You can change inline style or classname of a HTML element using javascript Using style property: You can modify inline style using style property
document.getElementById("title").style.fontSize = "30px";
Using ClassName property: It is easy to modify element class using className property
document.getElementById("title").className = "custom-title";

What would be the result of 1+2+'3'
The output is going to be 33. Since 1 and 2 are numeric values, the result of the first two digits is going to be a numeric value 3. The next digit is a string type value because of that the addition of numeric value 3 and string type value 3 is just going to be a concatenation value 33.

What is a debugger statement
The debugger statement invokes any available debugging functionality, such as setting a breakpoint. If no debugging functionality is available, this statement has no effect. For example, in the below function a debugger statement has been inserted. So execution is paused at the debugger statement just like a breakpoint in the script source.
function getProfile() {
// code goes here debugger;
// code goes here
}

What is the purpose of breakpoints in debugging
You can set breakpoints in the javascript code once the debugger statement is executed and the debugger window pops up. At each breakpoint, javascript will stop executing, and let you examine

the JavaScript values. After examining values, you can resume the execution of code using the play button.

Can I use reserved words as identifiers
No, you cannot use the reserved words as variables, labels, object or function names. Let's see one simple example,

```
var else = "hello"; // Uncaught SyntaxError: Unexpected token else
```