1. What are arrow functions?

Solution:

Arrow function is one of the features introduced in the ES6 version of JavaScript. **It allows you to create functions in a cleaner way compared to regular functions.**

For Example:-

```
<!DOCTYPE html>

<html>

<body>


<h2>JavaScript Arrow Function</h2>


<p>This example shows the syntax of an Arrow Function, and how to use it.</p>


<p id="demo"></p>


<script>//this is syntax of arrow function

let myFunction=(x,y)=>x+y;

document.getElementById("demo").innerHTML=myFunction(60,55);  //OUTPUT:  115

</script>


</body>

</html>
```

2. Explain about let var and constant?

Solution:

Let:

1. **let is introduced in 2015 in the ES6 version of javascript.**

2. **It can be updated but not redeclare in same scope.**
3. **During hoisting declaration moved to top**
4. **It introduced in scope or block only.**

Var:

1.Var can be introduced in global and local scope.

2.It can be updated and redeclare in same scope.

3. before 2015 only var is used for declaring variables.

Const:

1.used to declare constant only.

2.it is initialised at same time of declaration.

3.can not redeclared.

3. Block-level element and inline element?

Solution:

Every HTML element has a default display value, depending on what type of element it is.

There are two display values: block and inline.

Block-level Elements

A block-level element always starts on a new line, and the browsers automatically add some space (a margin) before and after the element.

A block-level element always takes up the full width available (stretches out to the left and right as far as it can).

Two commonly used block elements are: `<p>` and `<div>`.

The `<p>` element defines a paragraph in an HTML document.

The `<div>` element defines a division or a section in an HTML document.

Inline Elements

An inline element does not start on a new line.

An inline element only takes up as much width as necessary.

This is a <span> element inside a paragraph

4.What are the HTML tags used to display the data in tabular form?

Solution:

HTML table tag is used to display data in tabular form (row * column). There can be many columns in a row. We can create a table to display data in tabular form, using **<table> element, with the help of <tr> , <td>, and <th> elements**.

<table>

<tr><th>First_Name</th><th>Last_Name</th><th>Marks</th></tr>

<tr><td>Sonoo</td><td>Jaiswal</td><td>60</td></tr>

<tr><td>James</td><td>William</td><td>80</td></tr>

<tr><td>Swati</td><td>Sironi</td><td>82</td></tr>

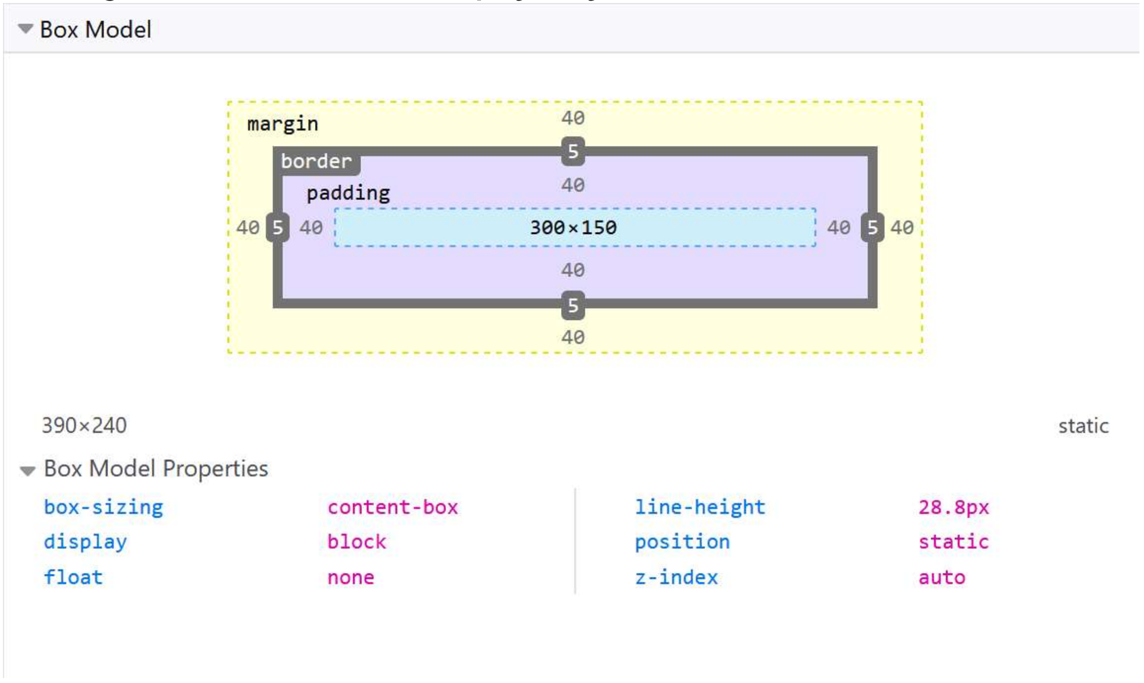
<tr><td>Chetna</td><td>Singh</td><td>72</td></tr>

</table>

5.What is the Box model in CSS?

Solution:

A box in CSS consists of a content area, which is where any text, images, or other HTML elements are displayed. This is optionally surrounded by padding, a border, and a margin, on one or more sides. The box model **describes how these elements work together to create a box as displayed by CSS**



box-sizing	content-box	line-height	28.8px
display	block	position	static
float	none	z-index	auto

6. What are the different types of Selectors in CSS?

In CSS, pattern matching rules determine which style rules apply to elements in a document. These patterns, called *selectors*

1.type selector

Eg. `H1 { font-family: Helvetica }`

2 Grouping

Eg. `H1, H2, H3 { font-family: Helvetica }`

3 Attribute selectors

3.1 Matching attributes, single values, and multiple values

Eg.

For example, the following rule matches all H1 elements that specify the "href" attribute, whatever its value:

```
   H1[href] { color: blue; }
```

In the following example, the rule matches all SPAN elements whose "class" attribute has the value "example":

```
   SPAN[class=example] { color: blue; }
```

3.2 The class and id attribute in HTML

For example, we can assign style information to all elements with `class="pastoral"`:

```
  .pastoral { color: green }  /* all elements with class=pastoral */
```

or just to H1 elements with `class="pastoral"`:

```
  H1.pastoral { color: green }  /* H1 elements with class=pastoral */
```

Given these rules, the first H1 instance below would not have green text, while the second would:

```
  <H1>Not green</H1>
  <H1 class="pastoral">Very green</H1>
```

Note that "H1.pastoral" is equivalent to "H1[class~=pastoral]".

3.3 The class attribute in other document languages: @class

For instance, to specify that the "type" attribute of XML has the role of assigning class information, authors should include the following declaration in their style sheets:

```
   @class type;
```

Then, a rule for XML such as:

```
   PARA.romeo { ... }
```

would be equivalent to:

```
   PARA[type~=romeo] { ... }
```

3.4 The id attribute

In the following example, the style rule contains no selector information and therefore matches any element that has `id="z98y"`. The rule will thus match for the P element:

```
  <HEAD>
  <STYLE>
  #z98y { letter-spacing: 0.3em }
  </STYLE>
  </HEAD>
  <BODY>
  <P id=z98y>Wide text</P>
  </BODY>
```

In the next example, however, the style rule will only match an H1 element that has `id="z98y"`. The rule will not match the P element in this example:

```
  <HEAD>
  <STYLE>
  H1#z98y { letter-spacing: 0.5em }
  </STYLE>
  </HEAD>
  <BODY>
  <P id=z98y>Wide text</P>
  </BODY>
```

4 Contextual selectors

For example, consider the following rules:

```
  H1 { color: red }
  EM { color: red }
```

5 Parent-child selectors

```
BODY ~ P { line-height: 1.3 }
```

6 Sequential selectors

```
/MATH ~ P/ { text-indent: 0 }
```

```
   7. What is VH/VW (viewport height/ viewport width) in CSS?
      Solution:
```

vw and vh are a percentage of the viewport width and height, respectively: **100vw is 100% of the width**, 80vw is 80%, etc. This seems like the exact same as the % unit, which is more common.

8. Difference between Java script and java?
   Solution:

- Java is an OOP programming language while Java Script is an OOP scripting language.
- Java creates applications that run in a virtual machine or browser while JavaScript code is run on a browser only.
- Java code needs to be compiled while JavaScript code are all in text.

9. What are the ways to define a variable in JavaScript?

Basically we can declare variables in three different ways by using **var, let and const keyword**. Each keyword is used in some specific conditions. var: This keyword is used to declare variable globally. If you used this keyword to declare variable then the variable can accessible globally and changeable also.

9. What are the data types supported by JavaScript?

Solution:

JavaScript types

- Boolean type.
- Null type.
- Undefined type.
- Number type.
- BigInt type.
- String type.
- Symbol type.

11.How can you create an object in JavaScript?

Solution:

1. Creating objects using object literal syntax

This is really simple. All you have to do is throw your key value pairs separated by ':' inside a set of curly braces({ }) and your object is ready to be served (or consumed), like below:

```
const person = {
  firstName: 'testFirstName',
  lastName: 'testLastName'
};
```

2. Creating objects using the 'new' keyword

Here are 2 ways you can use the 'new' keyword pattern —

a) Using the 'new' keyword with' in-built Object constructor function

To create an object, use the new keyword with `Object()` constructor, like this:

```
const person = new Object();
person.firstName = 'testFirstName';//giving properties
person.lastName = 'testLastName';
```

b) Using 'new' with user defined constructor function

We first create a constructor function and then use the 'new' keyword to get objects:

```
function Person(fname, lname) {
  this.firstName = fname;
  this.lastName = lname;
}
```

Now, anytime you want a 'Person' object, just do this:

```
const personOne = new Person('testFirstNameOne', 'testLastNameOne');
const personTwo = new Person('testFirstNameTwo', 'testLastNameTwo');
```

3. Using Object.create() to create new objects

```
const orgObject = { company: 'ABC Corp' };
```

And you want to create employees for this organization. Clearly, you want all the employee objects.

```
const employee = Object.create(orgObject, { name: { value: 'EmployeeOne' } });

console.log(employee); // { company: "ABC Corp" }
console.log(employee.name); // "EmployeeOne"
```

4. Using Object.assign() to create new objects

Assume you have two objects as below:

```
const orgObject = { company: 'ABC Corp' }
const carObject = { carName: 'Ford' }
```

Now, you want an employee object of 'ABC Corp' who drives a 'Ford' car. You can do that with the help of `Object.assign` as below:

```
const employee = Object.assign({}, orgObject, carObject);
```

Now, you get an `employee` object that has `company` and `carName` as its property.

5. Using ES6 classes to create objects

```
class Person {
  constructor(fname, lname) {
    this.firstName = fname;
    this.lastName = lname;
  }
}

const person = new Person('testFirstName', 'testLastName');

console.log(person.firstName); // testFirstName
console.log(person.lastName); // testLastName
```