

1 Introduction to HTML

- HTML(Hypertext markup language) is used to create the skeleton of a webpage .
- A group of web pages together we call a website .
- HTML is all about tags , there are around 140 - 145 tag names (reserved keywords in html) as of HTML5 .
- A tag is something which wraps tag names inside angular braces (<tagname>, </tagname>)
- **<tagname>** this is the opening tag & **</tagname>** is the closing tag
- Opening tag (<tagname>) , closing tag (</tagname>) and the content inside all together we call an HTML element.
- There are few HTML elements which don't have closing tags that we call self-closing tags.

Ex: <meta >,
, <hr>, etc.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Introduction to HTML</title>
</head>
<body>
</body>
</html>
```

Above is the sample Boilerplate code which is necessary for creating any web page .

- The `<!DOCTYPE html>` is not an HTML element. It is a document type declaration and is the first line of an HTML document. It informs the web browser which version of HTML is being used in the document.
- The head element will contain the information about the webpage which includes the title of the webpage `<title>Introduction to HTML</title>`, meta tags which contains the meta information of the webpage (keywords, description useful for SEO purpose).
- The body element will contain all the content that we wanted to display on the webpage.

```
<!DOCTYPE html>
```

```

<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Introduction to HTML</title>
</head>
<body>
  <b>This tag will bold the text inside it.</b>
  <hr><!-- Used to make horizontal rule (hr) in the webpage -->
  <br><!-- Used to make a line break(br) -->
  <!--
    To make headings in the webpage there are six tags available to us
    Note: Don't use more than one h1 tag in a single webpage.

    Using more than one h1 element in an HTML document is allowed, but it is
    generally not recommended. The h1 element is typically used as the main heading
    of a document and represents the highest level of section headings.

    Having multiple h1 elements can cause confusion for search engines and
    screen readers, as they may not know which h1 element represents the main heading
    of the page
  -->
  <h1>Heading 1</h1>
  <h2>Heading 2</h2>
  <h3>Heading 3</h3>
  <h4>Heading 4</h4>
  <h5>Heading 5</h5>
  <h6>Heading 6</h6>
</body>
</html>

```

Attributes in HTML

Attributes (properties) of an HTML element will provide more information to it.

Ex:

```

```

<!--

In the above src & alt are attributes(properties) where :

src attribute's value indicates the source image to be displayed

alt attribute's value indicates the text needs to be shown when the image url is broken(not available) also screen readers use this value to tell what that image is about.

note: src, alt are mandatory attributes for img tag

-->

```

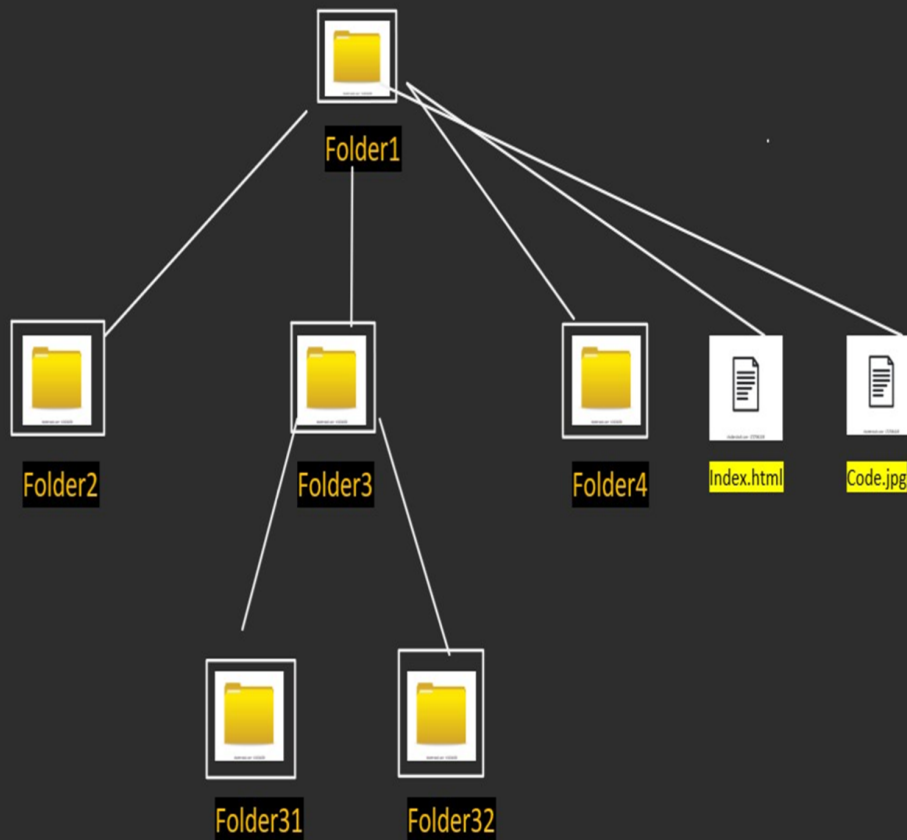
```

<!--

In the above width & height are optional attributes used to set the dimensions of the image(in pixels).

→

| Absolute path | Relative path |
|--|--|
| <p>An absolute path refers to the complete details needed to locate a file or folder, starting from the root element and ending with the other subdirectories.</p> <p>Example:</p> <p>C:\Users\Aravind\OneDrive\Desktop\sessions\test.html</p> | <p>Relative to current Folder / Directory .</p> <p>Example:</p> <p>../folder1/folder2/file.jpg</p> |



The folder system in a computer is a tree-like structure.

In the right side picture let if we are in folder31.

`./` : Refers to the same folder (folder31).

`../` : refers to the parent folder of the current folder i.e Folder3.

`../..` : Grand parent folder of the current folder i.e Folder1 .

`../..Folder4` : refers to inside folder4 w.r.t the current folder.

`../..index.html` : refers to the index.html file inside Folder1.

What is the relative path of the code.jpg file when you are in Folder2 ?

Text formatting

em vs I : em used for emphasized text , i used for italic text.

strong vs bold : Used to make text bold strong is used for important text, whereas bold is used for just making the text bold.

sub vs sup : For subscript and superscript.

ins vs del : ins used for underlined text , del is used for strikethrough the text.

Mark : Used to mark/Highlight some text .

Small : Defines smaller text.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Text formatting</title>
</head>
<body>
  <b>Bold Text</b><br>
  <strong>Important Text</strong><br>
  <i>Italic Text</i><br>
  <em>Emphasized Text</em><br>
  <mark>Marked Text</mark><br>
  <small>Smaller Text</small><br>
  <del>Deleted Text</del><br>
  <ins>Inserted Text</ins><br>
  Text<sub>Subscripted Text</sub><br>
  Text<sup>Superscripted Text</sup><br>
</body>
</html>
```

Output of the Above code :

Bold Text

Important Text

Italic Text

Emphasized Text

Marked Text

Smaller Text

~~Deleted Text~~

Inserted Text

Text_{Subscripted Text}

Text^{Superscripted Text}

HTML Forms

Forms in HTML are used to collect data from the user using input elements .

Different types of Input:

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Input tag in HTML</title>
</head>




<body>
  <!-- type attribute is mandatory for an input tag.
  The default value of type attribute is text. -->
  <input type="text"><br>
  <input type="number"><br>
```

```

<input type="email"><br>
<input type="password"><br>
<input type="url"><br>
<input type="time"><br>
<input type="date"><br>
<input type="datetime-local"><br>
<input type="checkbox"><br>
<input type="radio"><br>
<input type="file"><br>
</body>

</html>

```

| |
|---|
| Normal text |
| 3738383 |
| abc@gmail.com |
| ***** |
| https://google.com |
| 12:12 AM  |
| 02/02/2023  |
| 02/16/2023 11:14 PM  |
| <input checked="" type="checkbox"/> |
| <input type="radio"/> |
| Choose File index.html |

Student login form (assignment)

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">

```

```

<title>Forms</title>
</head>
<body>

<h1>Student login form</h1>
<!--
    name, branch, email , dob , gender(male, female , other)
    subjects (DSA, Frontend1, Frontend2)

    /login/submit

    action attribute will be the target fileAddress / REST endpoint to which
you wanted to submit data .
-->
<form action="./aravind.html" method="GET">
    <div>
        <label for="name">Name*</label>
        <input type="text" id="name" placeholder="Enter Name" name="username"
required>
    </div>
    <div>
        <label for="branch">Branch</label>
        <input type="text" id="branch" placeholder="Enter branch" name="abc">
    </div>
    <div>
        <label for="email">Email*</label>
        <input type="email" id="email" name="email" required>
    </div>
    <div>
        <label for="dob">Date of birth*</label>
        <input type="date" id="dob" name="dateofbirth" required>
    </div>
    <div>
        <!-- radio -->
        <label for="male">Male</label>
        <input type="radio" id="male" name="gender" value="m">

        <label for="female">female</label>
        <input type="radio" id="female" name="gender" value="f">

```



```

        <label for="other">other</label>
        <input type="radio" id="other" name="gender" value="oth">
    </div>
    <!-- <div>
        <label for="dsa">DSA</label>
        <input type="checkbox" id="dsa" name="dsa">

        <label for="frontend1">Frontend1</label>
        <input type="checkbox" id="frontend1" name="frontend1">

        <label for="frontend2">Frontend 2</label>
        <input type="checkbox" id="">
    </div> -->
    <!-- <button type="submit">Submit Data</button> -->
    <button type="reset">Reset</button>
    <input type="submit" value="Submit Data">
</form>

```

```

<!-- <input type="radio" name="gender">
<input type="radio" name="gender"> -->

```

```

<!--

```

whenever you need to submit data :
 you should give the name attribute for all the inputs

these name attribute value will be the keys for the data

we have two inputs (name = "username") , second input (name = "x")

when we submit the above two input form

username: "Aravind" ,
 password: "Abc@123"

```

-->

```

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

CSS

Css is used to style the webpages.

Box Model:

```
<!DOCTYPE html>  
<html lang="en">  
  <head>  
    <meta charset="UTF-8" />  
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />  
    <meta name="viewport" content="width=device-width,  
initial-scale=1.0" />  
    <title>Document</title>  
    <style>  
      div {  
        width: 200px;  
        height: 100px;  
        padding: 20px;  
        border: 1px solid black;  
        margin: 10px;  
      }  
    </style>  
  </head>  
  <body>  
    <div>Hello World!</div>  
  
  <!--  
    CSS box model:
```

The CSS box model is a fundamental concept in web design that defines how HTML elements are displayed on a web page. It describes how the dimensions of an element, such as its width, height, padding, and border, are calculated.

The CSS box model consists of four layers, which wrap around an HTML element:

1. Content: The actual content of the element, such as text or an image.
2. Padding: The space between the content and the element's border.
3. Border: The line that separates the padding from the margin.
4. Margin: The space between the element's border and other elements on the page.

The dimensions of an element are calculated by adding up the width and height of the content, padding, and border. The margin is not included in these dimensions.

In the above code example:

. We have a div element with a width of 200px and a height of 100px .

. It also has 20px of padding, a 1px border, and 10px of margin.

. The total width of the element will be 242 pixels (200px for the content, 20px on each side for the padding, and 1px on each side for the border).

. The total height of the element will be 142 pixels (100px for the content, 20px on each side for the padding, and 1px on each side for the border). The margin is not included in these dimensions.

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

block , inline & inline-block:

In CSS, there are three main display values that can be used to control how an element is rendered on a web page: block, inline, and inline-block.

by default every HTML element will be one of the above three types

For example:

div, p, h1, h2 etc are by default block level elements

span, bold, label etc are by default inline elements

button, input, img etc are by default inline-block elements.

We can overwrite the default behavior of an element by using the CSS "display" property .

```
display: block;
```

An element with `display: block;` takes up the full width of its container by default and starts on a new line.

Block-level elements are often used to create structural elements on a page, such as headings, paragraphs, and divs.

```
display: inline;
```

An element with `display: inline;` only takes up as much width as necessary and does not start on a new line. Inline-level elements are often used for text and other content that should flow alongside other inline elements.

```
display: inline-block;
```

An element with `display: inline-block;` is similar to an inline element, but it can have a width and height specified. This can be useful for creating elements that need to be positioned alongside other inline elements but still require a specific size or layout.

1. Block-level elements are typically used for larger structural elements on a web page, while inline-level elements are used for smaller bits of content such as links or text.

2. Block-level elements can have margins, padding, and borders applied to them, while inline-level elements only allow padding and borders to be applied.
3. Inline-level elements are often used for creating links, because they can be easily positioned next to other inline elements like text or images.
4. Inline-block elements are useful for creating elements like buttons or form fields that need to have a specific size and layout but still flow with other inline elements.
5. Understanding the different display values in CSS is an important part of creating a well-designed and responsive web page. By choosing the right display value for each element, you can control how it is displayed on the page and ensure that it works well with other elements in your layout.

Position:

Display: flex

```
<div class="container">
  <div class="item">1</div>
  <div class="item">2</div>
  <div class="item">3</div>
  <div class="item">4</div>
  <div class="item">5</div>
</div>
```

```

/* .container {
    padding: 20px;
    background-color: blue ;
    display: flex ; */
    /* flex-direction: row ;
    flex-wrap: wrap; */
    /* flex-flow: row wrap ;
    justify-content: space-around;
} */
/* body{
    margin-top: 0 ;
}
.item {
    padding: 30px;
    margin: 2px;
    background-color: orange ;
} */

```

```

.container {
    color: red ;
    display: inline ;
}

```

```

#test{
    display: block ;
}

```

/*

flex is a layout mode 1

display: flex ;

there many props which we can apply in a flex container

these will help to adjust it's children

flex-direction : row (default value)

flex-wrap : nowrap (default value)

flex-flow : shorthand prop for the above two

```
flex-direction : row
horizontal axis => main axis
vertical axis => cross axis

flex-direction : column

=> cross
vertical => main

justify-content : main axis
flex-start(default)
flex-end(moves all the items together to end of the main axis)
center (centers elements about main axis )

space-around
space-evenly
space-between

align-items : cross axis
align-content: cross axis

*/
```

[Flex Practice link :](#)

Solution for Login form:

HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
```



```

    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <link rel="stylesheet" href="./index.css">
</head>
<body>
    <div class="root-container">
        <div class="container">
            <label for="email">Email</label>
            <input type="email" placeholder="Email">
            <div class="box1">
                <label for="password">Password</label>
                <a href="#">Forgot Password ?</a>
            </div>
            <input type="password" placeholder="Password">
            <button>Login</button>
            <div class="box1 box2">
                <span>Don't have an account ? </span>
                <a href="#">Create new account</a>
            </div>
        </div>
    </div>
</body>
</html>

```

CSS:

```

.container {
    background-color: white ;
    padding: 30px ;
    display: flex;
    flex-flow: column wrap;
    width: 350px;
    height: 250px;
    justify-content: space-evenly;
    border-radius: 10px;
}

input, button{

```

```
border: none ;
background-color: #e7e7e7;
padding: 8px;
border-radius: 4px;
}

button {
  background-color: dodgerblue;
  color: white ;
}

.box1 {
  display: flex;
  flex-flow: row wrap;
  justify-content: space-between;
}

.box2 {
  justify-content: space-around;
}

a {
  text-decoration: none ;
  color: dodgerblue ;
}

body {
  margin: 0 ;
}

.root-container {
  background-color: #e7e7e7;
  height: 100vh ;
  display: flex;
  justify-content: center;
  align-items: center;
}
```

Grid:

Units In CSS:

1) . Absolute Units:

px (pixels): A single pixel on the screen. 1px is always equal to 1/96th of an inch, regardless of screen size or resolution.

in (inches): A physical inch on the screen.

cm (centimeters): A physical centimeter on the screen.

mm (millimeters): A physical millimeter on the screen.

pt (points): A unit commonly used in print media. 1pt is equal to 1/72nd of an inch.

pc (picas): Another unit commonly used in print media. 1pc is equal to 12pt.

2) .Relative Units:

em: A unit that's equal to the computed font size of the element it's used on. If no font size is specified, the default is usually 16px. For example, if an element has a font size of 16px, 1em is equal to 16px, and 2em is equal to 32px.

rem: A unit that's similar to em, but instead of being based on the font size of the current element, it's based on the font size of the root element (usually the <html>

element). This means that rem units are more predictable and easier to work with than em units.

vh (viewport height): A unit that's equal to 1% of the height of the viewport. For example, 50vh is equal to 50% of the viewport height.

vw (viewport width): A unit that's equal to 1% of the width of the viewport. For example, 50vw is equal to 50% of the viewport width.

vmin (viewport minimum): A unit that's equal to the smaller of vh and vw. For example, if the viewport height is smaller than the viewport width, 50vmin is equal to 50% of the viewport height.

vmax (viewport maximum): A unit that's equal to the larger of vh and vw. For example, if the viewport height is larger than the viewport width, 50vmax is equal to 50% of the viewport height.

3) .Percentage Units:

%: A unit that's relative to the size of the parent element. For example, if a child element has a width of 50% and its parent element has a width of 500px, the child element will have a width of 250px.

[Practice flex & grid figma](#)

[Dashboard using grid & flex](#)

Switch making:

```
<input type="checkbox" id="check" >
<label for="check">
  <span class="circle"></span>
</label>
```

```
input {
  width: 0;
  height: 0 ;
}
input:checked + label{
  background-color: green ;
}
input:checked + label > .circle {
  left: 20px;
/*  translateX: 20px; */
}
label {
  display: inline-block;
  width: 40px;
  height: 20px;
  border-radius: 10px;
  box-shadow: 2px 2px 4px gray ;
  background-color: gray ;
  position: relative ;
}
.circle {
  display: inline-block ;
  width: 20px;
  height: 20px;
  border-radius: 10px; background-color: white ;
  position: absolute ;
}
```

CSS Animations:

```
/* .container {  
    color: red ;  
    width: 100px;  
    height: 100px;  
    background-color: blue ;  
    cursor: pointer;  
    transition: 3s linear all 1s;  
}  
.container:hover {  
    background-color: yellow ;  
    width: 200px;  
    height: 200px;  
} */  
  
/*  
what is the time taken to change the bg-color ??  
  
0s time to change the color  
  
delay - waiting time to start the transition  
duration - the time duration for the transition  
start - end ;  
*/  
/*  
.animate {  
    width: 200px;  
    height: 200px;  
    background-color: tomato;  
    animation-name: aravind;  
    animation-duration: 8s;  
    animation-iteration-count: 4;  
    position: relative;  
    animation-direction: alternate;  
    animation-timing-function: ease-in;  
}
```

```
@keyframes rauch {
  from {
    width: 200px;
  }
  to {
    width: 400px;
  }
}
```

```
@keyframes aravind {
  0% {
    top: 0;
    left: 0;
  }
  25%{
    top: 0;
    left: 50vw;
  }
  50% {
    top: 0;
    left: 50vw;
  }
  75%{
    left: 50vw;
    top: 200px;
  }
  80%{
    top: 200px;
    left: 0;
  }
  100%{
    top: 0;
    left: 0;
  }
} */
```

```
/*
transform:
```

```

*/

.animate {
    width: 200px;
    height: 200px;
    background-color: red ;
    margin: 20px;
    transform: rotate(0deg);
}

.loader {
    width: 25px;
    height: 25px;
    border-radius: 50% ;
    border-left: 6px solid blue ;
    border-right: 6px solid rgb(189, 189, 234) ;
    border-top: 6px solid rgb(189, 189, 234) ;
    border-bottom: 6px solid rgb(189, 189, 234) ;
    animation: loading 1s 3s infinite linear;
}

@keyframes loading {
    from {
        transform: rotate(0deg) ;
    }
    to{
        transform: rotate(360deg) ;
    }
}

/* create animation where car moves from left to right wheels should rotate

car-body
two wheels

*/

```



```

.point {
  margin-left: 200px;
  width: 100px;
  height: 100px;
  background-color: blue ;
  transform-origin: 40px 30px;
  animation: rotate 2s infinite;
}

@keyframes rotate {
  from {
    transform: rotate(0deg);
  }
  to{
    transform: rotate(50deg);
  }
}

```

Bootstrap assignment:

```

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Bootstrap Example</title>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/bootstrap.min.css"
rel="stylesheet">
  <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/js/bootstrap.bundle.min.js"
"></script>
  <link rel="stylesheet" href="./index.css">
</head>
<body>

```

```
<div class="container mt-5">
  <h1 class="text-center">Internship</h1>
  <form class="was-validated" class="row">
    <div class="row">
      <div class="col-4">
        <label for="firstname">First Name</label>
        <input type="text" required id="firstname"
class="form-control">
        <div class="invalid-feedback">First name required!</div>
      </div>

      <div class="col-4">
        <label for="lastname">Last Name</label>
        <input type="text" required id="lastname"
class="form-control">
        <div class="invalid-feedback">Last name required!</div>
      </div>

      <div class="col-4">
        <label for="email">Email Address</label>
        <input type="text" required id="email" class="form-control">
        <div class="invalid-feedback">Email required!</div>
      </div>

      <div class="col-6 mt-4">
        <label for="college">College</label>
        <input type="text" required id="college"
class="form-control">
        <div class="invalid-feedback">
          Please provide a valid college name.
        </div>
      </div>

      <div class="col-3 mt-4">
        <label for="grad">Graduation Year</label>
        <select id="grad" class="form-select" required>
          <option value="">Choose...</option>
```

```
        <option value="2022">2022</option>
        <option value="2023">2023</option>
    </select>
    <div class="invalid-feedback">
        Please select a valid Graduation year
    </div>
</div>

<div class="col-3 mt-4">
    <label for="roll">Roll Number</label>
    <input type="text" id="roll" class="form-control" required>
    <div class="invalid-feedback">
        Please provide a valid roll no.
    </div>
</div>

<div class="col-12 mt-4">
    <input type="checkbox" label="check" class="form-check-inline
checkbox" required>

    <label for="check" class="checkbox-label">Agree to terms and
conditions</label>
    <div class="invalid-feedback">You must be agree before
submitting</div>
</div>

<div class="col-3 mt-4">
    <button class="btn btn-primary">Submit</button>

</div>
</div>
</form>
</div>

</body>
</html>
```

Javascript- Hoisting:

```
// const , let both are same except
// 1) in const need to initialize with a value and can't be updated

// console.log(x) ;
// var x = 20 ;
// console.log(y) ;
// var y = 100 ;

// Hoisting:
/*
1) Creatinon phase
2) Execution Phase

*/

/*
CODE:

console.log(x + y) ;
var y = 10 ;
console.log(x, y , x+y) ;
var x = 20 ;
console.log(x, y, x+y);

OUTPUT:

NaN
index.js:21 undefined 10 NaN
index.js:23 20 10 30

*/

/*
```

1) Creation phase

x = undefined, y = undefined

*/

/*

CODE:

console.log(a+b);

let c = 90 ;

var a = 200 ;

console.log(a, b , c, a+b+c);

var b = 900 ;

console.log(a, b , c, a+b+c) ;

OUTPUT:

NaN

index.js:47 200 undefined 90 NaN

index.js:49 200 900 90 1190

*/

/*

1) all the varriables will be hoisted(let, const , var)

2) variables which are declared with let and const will be hoisted but they will be in temporal deadzone until their declaration line gets executed

3) variable which are declared with var keyword will also be hoisted but they'll not be in temporal deadzone(they'll be open).

*/

/*

let , const are block scoped

var is a context scoped

In the same scope you're not allowed to re declare the same variable with let & const

```
*/  
  
// var x = 300 ;  
// if(true) {  
//     console.log(x, a, x+a) ;  
// }  
// var a = 900 ;  
// console.log(x, a , x + a) ;  
// var x = 90 ;  
// console.log(x)  
// if(true) {  
//     var x = 900 ;  
// }  
// var x = 1000 ;  
// console.log(x) ;
```

/*

CODE:

```
console.log(x) ;  
let a = 200 ;  
if(true){  
    console.log(x) ;  
    let a = 900;  
    console.log(a+x) ;  
    if(true){  
        var x = 940 ;  
        console.log(x+a) ;  
    }  
    console.log(2*a) ;  
}
```

OUTPUT:

undefined

```
index.js:96 undefined
```

```
index.js:98 NaN
```

```
index.js:101 1840
```

```
index.js:103 1800
```

```
*/
```