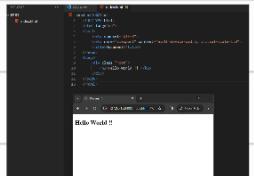


# Part - 1



Here we're seeing how we're printing Hello World on the screen just with the help of  $\rightarrow$  HTML &  $\rightarrow$  CSS

Now, we'll see what can we do if we want to achieve the same result but only with JAVASCRIPT  
But before that we should know this -

ChatGPT  
To use `querySelector` for selecting elements by class name, you can use the period symbol followed by the class name. Here's an example:

```
// Select the class element with the class name "example"
const element = document.querySelector('.example');

// If you want to select multiple elements with the same class, you can use "elements" instead of "element" like this:
const elements = document.querySelectorAll('.example');
```

ChatGPT  
To use `querySelector` for selecting elements by ID, you can use the hash (#) symbol followed by the ID in the argument. Here's an example:

```
// Select the element with the ID "example"
const element = document.querySelector('#example');

// Note: IDs must be unique in a document, so querySelector will return one element.
```

Replace "`example`" with the actual ID you are targeting. The `querySelector` method returns the first element that matches the specified ID. If you need to select multiple elements with the same ID (which is not recommended as IDs should be unique), you can still use `querySelectorAll`, but in practice, IDs are meant to be unique in a document.

Diff. b/w

InnerText &

InnerHTML

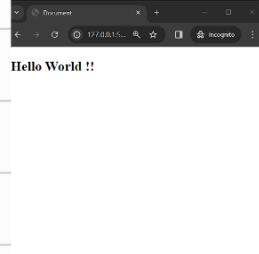


Here we're just using

- JAVASCRIPT

to achieve the same result

Here "h2" tag was created by JS & put into the root



New, we'll do the same thing with REACT!!

But, first let's see how does our Browser understands the code that we normally write in our scripts or in our js files??

Ans- Bcoz of the interpreters that the Browsers have already inside of them. Browsers have JS engines that execute ones JS codes easily & are able to understand  $\rightarrow$  `querySelector`  $\rightarrow$  `document` etc.  $\rightarrow$  various other such methods

But does the Browser know what is REACT. Ans- No & neither our Project is configured

to use REACT as of now!!  
∴ we've to first get REACT in our project

∴ The first way of adding REACT into our project is via - [CDN links]..... But what is CDN?? Let's see

ChatGPT  
CDN stands for Content Delivery Network. It is a distributed network of servers that work together to provide fast delivery of web content, including images, scripts, stylesheets, and other resources, to users based on their geographical location. CDN aims to reduce latency and improve website performance by serving content from servers that are geographically closer to the user.

Here's how a CDN typically works:

- Content Replication:** Content from the origin server (the original server where your website is hosted) is replicated and stored on multiple servers across different locations.
- Geographically Distributed Servers:** These replicated copies are stored on servers strategically located in various regions or even countries.
- Request Routing:** When a user makes a request to access a specific piece of content (like an image or a stylesheet), the CDN uses intelligent algorithms to determine the optimal server to fulfill that request based on the user's location.
- Content Delivery:** The content is then delivered to the user from the nearest CDN server instead of directly from the origin server.
- Benefits of using a CDN include:**
  - Faster Load Times:** Users can access content from a server that is physically closer to them, reducing the time it takes for the content to reach their device.
  - Scalability:** CDNs can handle a large volume of traffic and distribute it across multiple servers, reducing the load on the origin server.
  - Reliability:** CDNs often have multiple servers in different locations, providing redundancy and improving the reliability of content delivery.
  - Improved Security:** Some CDNs offer security features, such as DDoS protection and web application firewall (WAF), helping to protect websites from various online threats.

Popular CDN providers include Cloudflare, Akamai, Amazon CloudFront, and others.

These are the websites where React is hosted & we're just pulling React from the CDN links

so, after going to the CDN links

React Docs Issues Blog Community

## CDN Links

These are old and won't be updated. Go to reactify for the new links.  
See Add React to an Existing Project for the recommended steps to add React.

Both React and ReactDOM are available over a CDN.

`<script src="https://unpkg.com/react@18/umd/react.development.js"></script>`

The version above is only meant for development, and are not suitable for production. Minified and optimized production versions of React are available at:

`<script src="https://unpkg.com/react@18/umd/react.production.min.js"></script>`

To read a specific version of React and ReactDOM, replace 18 with the version number.

**Why the crossorigin Attribute?**

If you serve React from a CDN, we recommend to keep the `crossorigin` attribute set:

`<script crossorigin src="https://unpkg.com/react@18/umd/react.development.js"></script>`

We also recommend to verify that the CDN you are using sets the `Access-Control-Allow-Origin` HTTP header.

& copying these links in our code we've now acquired the power of "REACT"

`</body>
<script crossorigin src="https://unpkg.com/react@18/umd/react.development.js"></script>
<script crossorigin src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"></script>`

These will be present before the end of `</body>`

these are the links where React is hosted !!  
(2) what is cross origin ??

ChatGPT  
The `crossorigin` attribute is used in HTML to control how the browser handles cross-origin requests for certain elements, such as images (`<img>`), scripts (`<script>`), and stylesheets (`<link>`). Cross-origin requests occur when a user page at one origin (domain, protocol, or port) requests a resource from a different origin.

The `crossorigin` attribute is typically used with elements that load external resources from a different domain. It has two possible values:

- Anonymous:**
  - When `crossorigin="anonymous"`, the browser makes the request without sending any credentials (like cookies or HTTP authentication).
  - This is suitable when the resource (e.g., an image or script) doesn't require credentials for access.
- Use Credentials (or use-credentials):**
  - When `crossorigin="use-credentials"`, the browser includes credentials (like cookies or HTTP authentication) with the request.
  - This is used when the resource requires authentication to be accessed.

The `crossorigin` attribute is a security feature that helps prevent certain types of attacks, such as Cross-Site Request Forgery (CSRF). It allows web developers to control the behavior of cross-origin requests for specific resources.

## Part I . . . continues.

Now, this is the file that opens up when we COPY+PASTE the link of mentioned in src = " " of CDN links.

It tells us that all it is just a whole lot of code that's been rewritten by some FACEBOOK developers

Now, when  
we'll  
see in our  
console  
we'll see  
that -

→ we can now access React object & also there's something cool

these files contains some amazing algos which help us to use the power of React!!

## Part - 2

Now, we'll use **REACT CODE** to print "HELLO WORLD".

Now, in React, stuff happens differently  
Here, we render everything inside Root &  
for this, we need to create "Root"

Cuz "Root" is the place where React does all the "DOM STUFF".

"Creating an element is the core function of React" (ie the first file in the CDN links)

But creating a root is done via  
"ReactDOM" (is the 2nd file in the CDN  
links)

```
<html>
  <head>
    <title>Namste from React</title>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <script></script>
    <head>
      <div class="root"></div>
    </head>
    <body>
      <script>
        import React from 'https://esm.module-loader.com/react@18/umd/react.development.js'
        import ReactDOM from 'https://esm.module-loader.com/react-dom@18/umd/react-dom.development.js'
        import { createRoot } from 'https://esm.module-loader.com/react-dom@18/umd/react-dom.development.js'
        import App from './App'
        const root = ReactDOM.createRoot(document.querySelector('#root'))
        root.render(<App />)
      </script>
    </body>
  </html>

```

- `createElement` - takes 3 things
  - ① tag that "we want to create"
  - ② object
  - ③ 3<sup>rd</sup> argument - "Stuff" that we want to write

Here we're creating "Root" [ Here all the rendering will take place ]  
of React inside ".root" class

Don't PANIC React is Fuzzy!! 

### Part - 3

**COSTLIEST OPERATION** Manipulating DOM Nodes inside a web page & that's what React does. It helps us in achieving manipulation of DOM using JS in the most efficient way.

Now, we'll look at `createElement()` method in detail.

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>Document</title>
</head>
<body>
  <div class="root">

  </div>

  <script>
    crossorigin
    src="https://unpkg.com/react@18/umd/react.development.js"
  </script>
  <script>
    crossorigin
    src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"
  </script>

  <script src="App.js"></script>
</body>
</html>

```

index.html

```

App.js > [heading]
1 const heading = React.createElement(
2   "h1",
3   {},
4   "Namaste from React"
5 );
6
7 const root = ReactDOM.createRoot(document.querySelector(".root"));
8 root.render(heading);
9

```

App.js

```

const heading = React.createElement(
  "h1",
  { id: "heading" },
  "Hello World from React!"
);
any
const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(heading);

```

We can also add attributes like `{ id: "heading", xyz: "abc" }`

Let's try to make our heading in **Red color**



} By adding this file, we'll see that our heading will become "Red" in Color.

Now, let's see something interesting.

```

0:0 F...
const heading = React.createElement(
  "h1",
  { id: "heading", xyz: "abc" },
  "Hello World from React!"
);

console.log(heading);

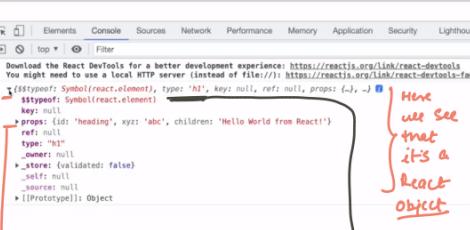
const root = ReactDOM.createRoot(document.getElementById("root"));

root.render(heading);

```

} What you think will be the O/P??

Usually, lot of people think that React will actually create on HTML element "h1" ... but is it true?



} At the end of day we'll see that it's a React element which is nothing but a JS object. Here it's of type: h1

Here we see that props are combinations of - (i) attributes (ii) children

```

0:0 F...
const heading = React.createElement(
  "h1",
  { id: "heading", xyz: "abc" },
  "Hello World from React!"
);

console.log(heading);

const root = ReactDOM.createRoot(document.getElementById("root"));

root.render(heading);

```

} Now, the role of render is to take this object, convert it into one `<h1>` tag & put it up on the top

## PART-4

NORMALLY, we've a nested structure in HTML

HTML

```

/***
 * <div id="parent">
 *   <div id="child">
 *     <h1>I'm h1 tag</h1>
 *   </div>
 * </div>
 */

```

} Now, previously we learnt that

- `createElement()` method have children as its 3rd argument, so

Why don't we use the same idea here.

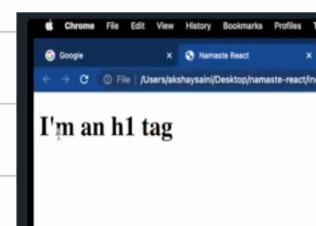
like - `const parent = React.createElement("div", { id: "parent" }, )`

What if we create another "Root.createElement" as its children

```

*<div id="parent">
*  <div id="child">
*    <h1>I'm h1 tag</h1>
*  </div>
*</div>

```



**BASICALLY**  
CREATING  
nesting  
of  
children

```

Download the React DevTools for a better development experience: https://reactjs.org/link/react-devtools
You might need to use a local HTTP server (instead of file://): https://reactjs.org/link/react-devtools-faq
+ $typeof: Symbol/react.element, type: 'div', key: null, ref: null, props: {}, ...
  + $typeof: Symbol/react.element
  + key: null
  + ref: null
  + props:
    + children:
      + $typeof: Symbol/react.element
        + key: null
        + ref: null
        + type: "div"
        + _owner: null
        + _store: {validated: false}
        + _self: null
        + _source: null
        + [[Prototype]]: Object
      + id: 'parent'
      + [[Prototype]]: Object

```

} Set children as its props

∴ Only `render()` method converts this React object into the HTML tags while it's rendering. Until, before than it's just an React object.

```
<div id="parent">
  <div id="child">
    <h1>I'm h1 tag</h1>
    <h2>I'm h1 tag</h2>
  </div>
</div>
```

What if we  
want to create  
siblings??

Ans. this again

in the 3rd argument, if we want to pass more than 1 children, we'll pass more children in an array!! like this

```
const parent = React.createElement('div', { id: 'parent' },
  React.createElement('div', { id: 'child1' },
    React.createElement('h1', {}, 'Hi I am h1 sister'),
    React.createElement('h2', {}, 'I am h1 brother')
  ),
  React.createElement('div', { id: 'child2' },
    React.createElement('h1', {}, 'Hi I am h1 sister child'),
    React.createElement('h2', {}, 'I am h1 brother child')
  )
);

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(parent);
```

we'll  
study about  
this  
'key'  
later

```
const parent = React.createElement('div', { id: 'parent' },
  React.createElement('div', { id: 'child1' },
    React.createElement('h1', {}, 'Hi I am h1 tag'),
    React.createElement('h2', {}, 'I am h1 tag')
  ),
  React.createElement('div', { id: 'child2' },
    React.createElement('h1', {}, 'Hi I am h1 tag'),
    React.createElement('h2', {}, 'I am h1 tag')
  )
);

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(parent);
```

Our code will look like this, which will destroy our experience of using `React` to resolve this, there's something called `JSX` which is used for creating `tags`. Now, we'll never use `createElement` ever again.

- Summary -
  - `createElement()` have 3 arguments
    - (i) tag that we want to create
    - (ii) attributes
    - (iii) children } together forms props.
  - If we want to create siblings, we'll pass an array of children.

## Part-5

Q: If we change the order of the JS files

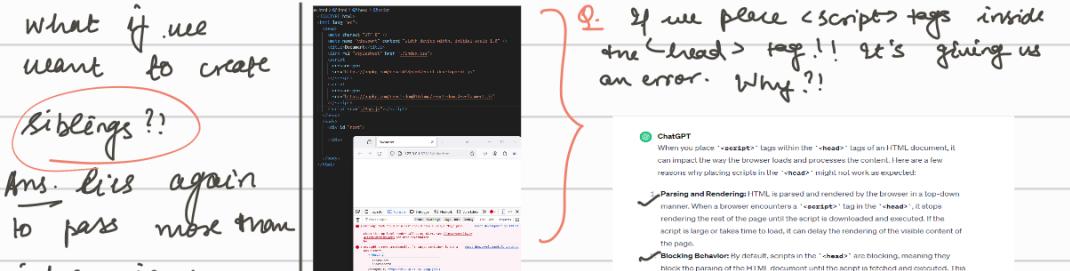
like this → `App.js`  
`React.js`  
`ReactDOM.js`

Result =

nothing

Sequence is imp.

```
3   <head>
4     <meta charset="UTF-8" />
5     <meta http-equiv="X-UA-Compatible" content="IE=edge" />
6     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
7     <title>Nameless React</title>
8   </head>
9   <body>
10    <div id="root">
11      <h1>Akshay is here!</h1>
12    </div>
13    <script src="/App.js"></script> ①
14    <script>
15      crossOrigin="https://unpkg.com/react@18/umd/react.development.js"
16      src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"
17    </script>
18    <script>
19      crossOrigin="https://unpkg.com/react@18/umd/react.development.js"
20      src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"
21    </script>
22  </body>
23 </html>
```



**ChatGPT**  
When you place `<script>` tags within the `<head>` tags of an HTML document, it can impact the way the browser loads and processes the content. Here are a few reasons why placing scripts in the `<head>` might not work as expected:

- Parsing and Rendering**: HTML is parsed and rendered by the browser in a top-down manner. When a browser encounters a `<script>` tag in the `<head>`, it stops the rest of the page until the script is downloaded and executed. If the script is large or takes time to load, it can delay the rendering of the visible content of the page.
- Blocking Behavior**: By default, scripts in the `<head>` are blocking, meaning they won't permit the browser to continue rendering the page until the script has been executed. This can slow down your page load times, especially if the script is located early on.
- DOM Manipulation**: If your script manipulates the DOM (Document Object Model) trees to access elements that haven't been parsed yet (because the browser is still processing the rest of the HTML), it can lead to errors or unexpected behavior.

To address these issues, it's common practice to place `<script>` tags just before the closing `</body>` tag. This allows the browser to render the visible content first, and then load and execute script. This practice is known as "Deferred" script execution.

Example:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <!-- Other head elements -->
    <title>Your Page Title</title>
  </head>
  <body>
    <!-- Your page content -->

    <!-- Scripts at the end of the body -->
    <script>your script...</script>
  </body>
</html>
```

By placing script at the end of the `<body>`, you ensure that the HTML content is loaded and visible before the scripts are executed, improving perceived page load performance.

Q: Diff. b/w Production Link & Developer Link

**You**  
what is diff b/w production link & developer link

**ChatGPT**  
The terms "production link" and "developer link" are not standard terms in web development, and their meaning may depend on the context. However, in a general sense, they might refer to links used in different environments:

**Production Link:**  
This typically refers to a URL or link that points to the live or production version of a website or web application.

It is the publicly accessible link that end-users use to access the finalized, stable version of the application.

**Developer Link:**  
This might refer to a URL or link used during the development phase of a website or application.

Developers often use localhost or specific development servers to test and debug their code before deploying it to production.

In the context of web development, during the development phase, developers often work locally using a development server, and the links they use might involve localhost or a specific port. On the other hand, the production link is the URL where the final, polished version of the website or application is hosted for public access.

```
1 <html lang="en">
2   <head>
3     <meta charset="UTF-8" />
4     <meta http-equiv="X-UA-Compatible" content="IE=edge" />
5     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
6     <link rel="stylesheet" href="/index.css" />
7     <title>Nameless React</title>
8   </head>
9   <body>
10    <div id="root">
11      <h1>Akshay is here!</h1>
12    </div>
13    <script>
14      crossOrigin="https://unpkg.com/react@18/umd/react.development.js"
15      src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"
16    </script>
17    <script>
18      crossOrigin="https://unpkg.com/react@18/umd/react.development.js"
19      src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"
20    </script>
21  </body>
22 </html>
```

Q: what would happen if we're a tag `h1` already inside `root`??

Ans- we'll see that "Akshay" is here replaced within few milliseconds by the `root.render()` coz after the `scripts` gets loaded it'll take control of the `<div id="root">`

Akshay is here!

Previous

Root.render() → will replace everything

Steps: ① first when browser reads our HTML, it'll point "Akshay is here"

② But as soon as it reads the script, it'll load React into our app & then React DOM & then our App.js & will start executing the code line by line & will

I'm an h1 tag  
I'm an h2 tag  
I'm an h1 tag  
I'm an h2 tag

Current

Result:

Sequence is imp.

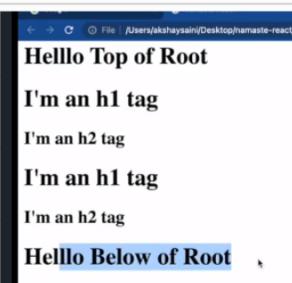
~~replace~~ everything that is present in root div

~~not appendid~~

```
<link rel="stylesheet" href="."/>
<title>Namaste React</title>
</head>
<body>
  <h1>Hello Top of Root</h1>
  <div id="root">
    <h1>Akshay is here!</h1>
  </div>
  <h1>Hello Below of Root</h1>
</body>
<script>
  crossorigin
</script>
```

Ans. This is bcoz root.render() will work only in that area where it's being called.

Q. Suppose if we're something at the top & bottom of the #root div. What will be our ans?



Library: It can work independently in a small portion of our App as well. React is a JS library coz it can work just for Nav, Div & various other tags, without making any huge change in the code.

But Framework works on a certain set of principles & rules.

**ChatGPT**  
The terms "framework" and "library" refer to two different approaches or styles of organizing and structuring code in software development. Here are the key differences:

**Library:**

- A library is a collection of pre-written code or functions that you can use in your program.
- It is a set of tools that you can call to perform specific tasks, but it doesn't dictate the overall structure or flow of your application.
- You include a library in your project and use its functions as needed.
- Libraries are typically focused on specific tasks, such as handling data, working with data structures, etc.

**Framework:**

- A framework is a more comprehensive and predefined structure for building applications.
- It provides a foundation with predefined rules and guidelines on how your application should be structured.
- The framework application within the framework, and it often follows the " inversion of control principle", meaning that the framework calls your code rather than the other way around.
- Frameworks often provide a complete structure, including tools for handling routing, data storage, user interface, etc.

In essence, while a library provides specific tools that you can use, a framework provides a skeleton or structure for your entire application, and you build within that structure.

Here's a simple analogy:

- Library:** You have a toolbox, and you pick and use the tools you need for a specific task.
- Frameworks:** You have a pre-built house structure, and you fill in the rooms and details according to the blueprint.

SUMMARY - ① We learnt to print "HELLO WORLD"

using - ② HTML  
③ JS  
④ React

② we learnt how can we incorporate

React using CDN links

③ we learnt about CDN (Content Delivery Network) & cross-origin

④ we learnt how to use `.createElement()` API which takes 3 args - ② tag that we want to create  
Objects. ④ ⑤ attribute } together known  
⑥ children } as props

⑤ if we want to create sibling tags

we pass array of children for a parent element

⑥ `.createElement()` - returns an object until it remains an object it gets rendered via ReactDOM

⑦ `root.render()` - replace everything, where it's being called