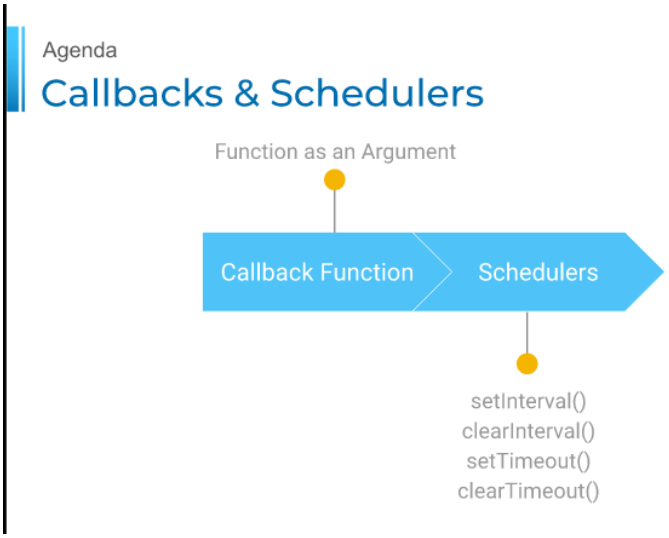


Passing function as an argument:

```JS
function displayGreeting(displayName) {


}
displayGreeting(function() {
  console.log("Rahul");
});
```

Callback:

A callback is a function that is passed as a argument to the other function.example

Callback Function

```JS
function displayGreeting(displayName) {
  console.log("Hello");
  displayName();
  console.log("Good Morning!");
}
function displayRahul() {
  console.log("Rahul");
}
displayGreeting(displayRahul);
```

Callbacks & Schedulers | Cheat Sheet:

1. Callback function: **A Callback is a function that is passed as an argument to another function.**

1.1 Passing a function as an argument

```JS
function displayGreeting(displayName) {
  console.log("Hello");
  displayName();
  console.log("Good Morning!");
}

displayGreeting(function() {
  console.log("Rahul");
});
```

```
Hello
Rahul
Good Morning!
```

1.2 Passing a function name as an argument

```JS
function displayGreeting(displayName) {
  console.log("Hello");
  displayName();
  console.log("Good Morning!");
}
function displayRahul() {
  console.log("Rahul");
}
displayGreeting(displayRahul);
```

Output

```
Hello
Rahul
Good Morning!
```

1.3 Passing a function expression as an argument

```JS
function displayGreeting(displayName) {
  console.log("Hello");
  displayName();
  console.log("Good Morning!");
}
let displayRam = function() {
  console.log("Ram");
}
displayGreeting(displayRam);
```

```
Hello
Ram
Good Morning!
```

2. Schedulers

The Schedulers are used to schedule the execution of a callback function.

There are different scheduler methods.

- **setInterval()**
- **clearInterval()**
- **setTimeout()**
- **clearTimeout(), etc.**

2.1 setInterval():**The setInterval()** method allows us to run a function at the specified interval of time repeatedly.

Syntax: `setInterval(function, delay);`

function - a callback function that is called repeatedly at the specified interval of time (`delay`).

delay - time in milliseconds. (1 second = 1000 milliseconds)

```
let counter = 0;
setInterval(function() {
  console.log(counter);
  counter = counter+1;
}, 1000);
```

In the setInterval() method, the callback function repeatedly executes until the browser tab is closed or the scheduler is cancelled.

When we call the setInterval() method, it returns a unique id. This unique Id is used to cancel the callback function execution.

2.2 clearInterval()

The `clearInterval()` method cancels a schedule previously set up by calling `setInterval()` .

To execute `clearInterval()` method, we need to pass the uniqueId returned by `setInterval()` as an argument.

Syntax: `clearInterval(uniqueId);`

```
let counter = 0;
let uniqueId = setInterval(function() {
  console.log(counter);
  counter = counter+1;
}, 1000);

clearInterval(uniqueId);
```

2.3 setTimeout()

The `setTimeout()` method executes a function after the specified time.

Syntax: `setTimeout(function, delay);`

function - a callback function that is called after the specified time (`delay`).

delay - time in milliseconds.

```
let counter = 0;
setTimeout(function() {
  console.log(counter);
  counter = counter + 1;
}, 1000);
```

2.4 clearTimeout()

We can cancel the `setTimeout()` before it executes the callback function using the `clearTimeout()` method.

To execute `clearTimeout()` , we need to pass the uniqueId returned by `setTimeout()` as an argument.

Syntax: `clearTimeout(uniqueId);`

JAVASCRIPT

```
let counter = 0;
let uniqueId = setTimeout(function() {
  console.log(counter);
  counter = counter+1;
}, 1000);

clearTimeout(uniqueId);
```

Event Listeners and More Events | Cheat Sheet

1. Event Listeners

JavaScript offers three ways to add an Event Listener to a DOM element.

- Inline event listeners
- onevent listeners
- addEventListener()

1.1 Inline event listeners

HTML

```html
<button onclick="greeting()">Greet</button>
```

JAVASCRIPT

```javascript
function greeting() {
  console.log("Hi Rahul");
}
```

1.2 onevent listeners

HTML

```html
<button id="greetBtn">Greet</button>
```

JAVASCRIPT

```javascript
let greetBtnEl = document.getElementById("greetBtn");

greetBtnEl.onclick = function() {
  console.log("Hi Rahul");
};
```

1.3 addEventListener()

It is a modern approach to add an event listener.

Syntax: `element.addEventListener(event, function);`

element - HTML element

event - event name

function - Callback function

HTML

```html
<button id="greetBtn">Greet</button>
```

JAVASCRIPT

```javascript
let greetBtn = document.getElementById("greetBtn");

greetBtn.addEventListener("click", function() {
  console.log("Hi Rahul");
});
```

2. Operators

2.1 Comparison Operators

Operator	Usage	Description
Equal (==)	a == b	returns true if both *a* and *b* values are equal.
Not equal (!=)	a != b	returns true if both *a* and *b* values are not equal.
Strict equal (===)	a === b	returns true if both *a* and *b* values are equal and of the same type.
Strict not equal (!==)	a !== b	returns true if either *a* and *b* values are not equal or of the different type.
Greater than (>)	a > b	returns true if *a* value is greater than *b* value.
Greater than or equal (>=)	a >= b	returns true if *a* value is greater than or equal to *b* value.
Less than (<)	a < b	returns true if *a* value is less than *b* value.
Less than or equal (<=)	a <= b	returns true if *a* value is less than or equal to *b* value.

2.2 Logical Operators

Operator	Usage	Description
AND (&&)	a && b	returns true if both *a* and *b* values are true.
OR (||)	a || b	returns true if either *a* or *b* value is true.
NOT (!)	!a	returns true if *a* value is not true.

3. More Events

Events are the actions by which the user or browser interact with HTML elements.

There are different types of events.

- **Keyboard Events**
- Mouse Events
- Touch Events, and many more.

3.1 Keyboard Events

Keyboard Event is the user interaction with the keyboard.

The keyboard events are

- `keydown`
- `keyup`

3.1.1 Keydown event

The `keydown` event occurs when a key on the keyboard is pressed.

Syntax: `element.addEventListener("keydown", function);`

```
let inputEl = document.createElement("input");

function printKeydown() {
  console.log("key pressed");
}

inputEl.addEventListener("keydown", printKeydown);
document.body.appendChild(inputEl);
```

3.1.2 Keyup event

The `keyup` event occurs when a key on the keyboard is released.

Syntax: `element.addEventListener("keyup", function);`

3.2 Event Object

Whenever an event happens, the browser creates an event object.

It consists of information about the event that has happened.

It consists of many properties and methods.

- `type`
- `target`
- `key`
- `timeStamp`
- `stopPropagation` , and many more.

3.2.1 Properties & Methods

For any event, event-specific properties and methods will be present.

For Example,

The `keydown` event has `key` property, whereas the `onclick` event doesn't have it.

event.type

The `event.type` property contains the type of event occurred like `click` , `keydown` , etc.

```
let inputEl = document.createElement("input");

function printKeydown(event) {
  console.log(event.type);  // keydown
}

inputEl.addEventListener("keydown", printKeydown);
document.body.appendChild(inputEl);
```

event.target

The `event.target` property contains the HTML element that triggered the event.

```
let inputElement = document.createElement("input");

function printKeydown(event) {
  console.log(event.target);  // <input></input>
}

inputElement.addEventListener("keydown", printKeydown);
document.body.appendChild(inputElement);
```

event.key

The `event.key` property contains the value of the key pressed by the user.

```
let inputElement = document.createElement("input");

function printKeydown(event) {
  console.log(event.key);
}

inputElement.addEventListener("keydown", printKeydown);
document.body.appendChild(inputElement);
```

Keyboard key	event.key value
Enter	Enter
a	a
A	A
1	1
*	*
<	<

Hypertext Transfer Protocol (HTTP) | Cheat Sheet

1. Web Resources

A Web Resource is any data that can be obtained via internet.

A resource can be

- HTML document
- CSS document
- JSON Data or Plain text
- Image, Video, etc.

2. Uniform Resource Locator (URL)

URL is a text string that specifies where a resource can be found on the internet.

We can access web resources using the URL.

Syntax: `protocol://domainName/path?query-parameters`

In the URL `http://www.flipkart.com/watches?type=digital&rating=4` ,

- `http` is a **Protocol**
- `www.flipkart.com` is a **Domain Name**
- `/watches` is a **Path**
- `type=digital&rating=4` is the **Query Parameters**

2.1 Protocol

A protocol is a standard set of rules that allow electronic devices to communicate with each other.

There are different types of protocols.

- Hypertext Transfer Protocol (HTTP)
- Hypertext Transfer Protocol Secure (HTTPS)
- Web Sockets, etc.

2.1.1 HTTP

The Hypertext Transfer Protocol (HTTP), is a protocol used to transfer resources over the web.

Examples: Internet forums, Educational sites, etc.

HTTP Request: Message sent by the client

HTTP Response: Message sent by the server

2.1.2 HTTPS

In Hypertext Transfer Protocol Secure (HTTPS), information is transferred in an encrypted format and provides secure communication.

Examples: Banking Websites, Payment gateway, Login Pages, Emails and Corporate Sector Websites, etc.

2.2 Domain Name

It indicates which Web server is being requested.

2.3 Path

The path is to identify the resources on the server.

Examples:

- **/watches** in `http://www.flipkart.com/watches`
- **/electronics/laptops/gaming** in `http://www.flipkart.com/electronics/laptops/gaming`

2.4 Query Parameters

Query parameters add some criteria to the request for the resource.

Multiple query parameters can be added by using an `&` (ampersand) symbol.

For example,

`http://www.flipkart.com/watches?type=digital&rating=4`

3. HTTP

3.1 HTTP Requests

HTTP requests are messages sent by the client to initiate an action on the server.

HTTP request includes

- Start Line
- Headers
- Body

3.1.1 Start Line

A Start Line specifies a

- URL
- HTTP Method
- HTTP Version

HTTP Methods

The HTTP Request methods indicate the desired action to be performed for a given resource.

Methods	Description
GET (Read)	Request for a resource(s) from the server
POST (Create)	Submit data to the server
PUT (Update)	The data within the request must be stored at the URL supplied, replacing any existing data
DELETE (Delete)	Delete a resource(s)

HTTP Version

Year	Version
1991	HTTP/0.9
1994	HTTPS
1996	HTTP/1.0
1997	HTTP/1.1
2015	HTTP/2
2019	HTTP/3

3.1.2 Headers

HTTP Headers let the client and the server to pass **additional information** with an HTTP request or response.

3.1.3 Body

We place the data in the Request body when we want to **send data** to the server.

For example, form details filled by the user.

HTTP Requests

- Start Line
  - URL
    - Protocol
      - HTTP
      - HTTPS
    - Domain Name
    - Path
    - Query Parameters

    - HTTP Method
        - GET (Read)
        - POST (Create)
        - PUT (Update)
        - DELETE (Delete)
    - HTTP Version
- Headers
- Body

3.2 HTTP Responses

HTTP responses are **messages** sent by the server as an **answer** to the **clients request**.

HTTP Response includes

- Status Line
- Headers
- Body

3.2.1 Status Line

A Status line specifies a

- HTTP version
- Status code
- Status text

Status code

Status codes Indicate whether an HTTP request has been successfully completed or not.

Status Code Series	Indicates
1XX	Information
2XX	Success
3XX	Redirection
4XX	Client Error
5XX	Server Error

- 200 (Success) - Indicates that the request has succeeded
- 201 (Created) - The request has succeeded and a new resource has been created as a result

HTTP Requests using JS | Cheat Sheet:

1. Fetch

The `fetch()` method is used to fetch resources across the Internet.

Syntax: fetch(URL, OPTIONS)

URL - URL of the resource
OPTIONS - Request Configuration

1.1 Request Configuration

We can configure the fetch request with many options like,

- Request Method
- Headers
- Body
- Credentials
- Cache, etc.

We can configure a request by passing an `options` object with required properties and their values.

Example:

```
let options = {
  method: "GET",
  headers: {
    "Content-Type": "application/json"
  },
  body: JSON.stringify(data)
};
```

2. Making HTTP Requests using Fetch

The `method` property value in the `options` object can be `GET` , `POST` , `PUT` , `DELETE` , etc. The default method is `GET` .

2.1 GET

The `GET` method can be used to retrieve (get) data from a specified resource.

For example,

JAVASCRIPT

```javascript
let options = {
  method: "GET"
};

fetch("https://gorest.co.in/public-api/users", options);
```

2.2 POST

The `POST` method can be used to send data to the server.

```javascript
let data = {
  name: "Rahul",
  gender: "Male",
  email: "rahul@gmail.com",
  status: "Active"
};

let options = {
  method: "POST",
  headers: {
    "Content-Type": "application/json",
    Accept: "application/json",
    Authorization: "Bearer ACCESS-TOKEN"
  },
  body: JSON.stringify(data)
};

fetch("https://gorest.co.in/public-api/users", options)
  .then(function(response) {
    return response.json();
  })
  .then(function(jsonData) {
    console.log(jsonData);
  });
```

2.3 PUT

The `PUT` method can be used to update the existing resource.

```
let data = {
  name: "Rahul Attuluri"
};

let options = {
  method: "PUT",
  headers: {
    "Content-Type": "application/json",
    Accept: "application/json",
    Authorization: "Bearer ACCESS-TOKEN"
  },
  body: JSON.stringify(data)
};

fetch("https://gorest.co.in/public-api/users/1359", options)
  .then(function(response) {
    return response.json();
  })
  .then(function(jsonData) {
    console.log(jsonData);
  });
```

2.4 DELETE

The `DELETE` method can be used to delete the specified resource.

```
let options = {
  method: "DELETE",
  headers: {
    "Content-Type": "application/json",
    Accept: "application/json",
    Authorization: "Bearer ACCESS-TOKEN"
  }
};

fetch("https://gorest.co.in/public-api/users/1359", options)
  .then(function(response) {
    return response.json();
  })
  .then(function(jsonData) {
    console.log(jsonData);
  });
```

3. HTTP Response Object Properties and Methods

Response Object provides multiple properties to give more information about the HTTP Response.

- status (number) - HTTP status code
- statusText (string) - Status text as reported by the server, e.g. "Unauthorized"
- headers
- url
- text() - Getting text from response
- json() - Parses the response as JSON

For example,

```javascript
let options = {
  method: "GET"
};

fetch("https://gorest.co.in/public-api/users", options)
  .then(function(response) {
    return response.status;
  })
  .then(function(status) {
    console.log(status);  // 200
  });
```

Collapse

In the above example, we can get the response status as `200` when the request is success.

Try out accessing the different properties and methods of the HTTP Response Object like `url` , `text()` , `json()` , etc. and check the output in the below Code Playground console.

Wikipedia Search Application | Cheat Sheet

1. HTML Input Element

1.1 Search

The HTML `input` element with the type search is designed for the user to enter the search queries.

```html
<input type="search" />
```

2. Bootstrap Components

2.1 Spinner

The Spinners can be used to show the loading state of the page.

```html
<div class="spinner-border" role="status">
  <span class="sr-only">Loading...</span>
</div>
```

Code:

Html:

```
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css"
integrity="sha384-JcKb8q3iqJ61gNV9KGb8thSsNjpSL0n8PARn9HuZOnIxN0hoP+VmmDGM
N5t9UJ0Z" crossorigin="anonymous" />
    <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"
integrity="sha384-DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrC
XaRkfj" crossorigin="anonymous"></script>
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.1/dist/umd/popper.min.js"
integrity="sha384-9/reFTGAW83EW2RDu2S0VKaIzap3H66lZH81PoYlFhbGU+6BZp6G7niu7
35Sk7lN" crossorigin="anonymous"></script>
    <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"
integrity="sha384-B4gt1jrGC7Jh4AgTPSdUtOBvfO8shuf57BaghqFfPlYxofvL8/KUEfYiJOMMV
+rV" crossorigin="anonymous"></script>
  </head>
  <body>
    <div class="main-container">
      <div class="wiki-search-header text-center">
        <img class="wiki-logo"
src="https://d1tgh8fmlzexmh.cloudfront.net/ccbp-dynamic-webapps/wiki-logo-img.png" />
        <br />
        <input placeholder="Type a keyword and press Enter to search" type="search"
class="search-input w-100" id="searchInput" />
      </div>
      <div class="d-none" id="spinner">
        <div class="d-flex justify-content-center">
          <div class="spinner-border" role="status">
            <span class="sr-only">Loading...</span>
          </div>
        </div>
      </div>
      <div class="search-results" id="searchResults"></div>
    </div>
```

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

Css:

```
@import
url("https://fonts.googleapis.com/css2?family=Bree+Serif&family=Caveat:wght@400;700&family=Lobster&family=Monoton&family=Open+Sans:ital,wght@0,400;0,700;1,400;1,700&family=Playfair+Display+SC:ital,wght@0,400;0,700;1,700&family=Playfair+Display:ital,wght@0,400;0,700;1,700&family=Roboto:ital,wght@0,400;0,700;1,400;1,700&family=Source+Sans+Pro:ital,wght@0,400;0,700;1,700&family=Work+Sans:ital,wght@0,400;0,700;1,700&display=swap");

.main-container {
  font-family: "Roboto";
}

.wiki-search-header {
  border-style: solid;
  border-width: 1px;
  border-color: #d5cdcd;
  padding-top: 30px;
  padding-right: 20px;
  padding-bottom: 30px;
  padding-left: 20px;
  margin-bottom: 40px;
}

.wiki-logo {
  margin-bottom: 30px;
  width: 150px;
}

.search-input {
  font-size: 18px;
  border-style: solid;
  border-width: 1px;
  border-color: #d5cdcd;
  border-radius: 3px;
  padding: 10px;
}

.search-results {
  width: 100%;
  padding-left: 20px;
}
```

```
.result-item {
  margin-bottom: 20px;
}

.result-title {
  font-size: 22px;
}

.link-description {
  color: #444444;
  font-size: 15px;
}

.result-url {
  color: #006621;
  text-decoration: none;
}
```

Javascript:

```
let searchInputEl = document.getElementById("searchInput");

let searchResultsEl =
document.getElementById("searchResults");

let spinnerEl = document.getElementById("spinner");

function createAndAppendSearchResult(result) {
  let { link, title, description } = result;

  let resultItemEl = document.createElement("div");
  resultItemEl.classList.add("result-item");

  let titleEl = document.createElement("a");
  titleEl.href = link;
```

```
  titleEl.target = "_blank";
  titleEl.textContent = title;
  titleEl.classList.add("result-title");
  resultItemEl.appendChild(titleEl);

  let titleBreakEl = document.createElement("br");
  resultItemEl.appendChild(titleBreakEl);

  let urlEl = document.createElement("a");
  urlEl.classList.add("result-url");
  urlEl.href = link;
  urlEl.target = "_blank";
  urlEl.textContent = link;
  resultItemEl.appendChild(urlEl);

  let linkBreakEl = document.createElement("br");
  resultItemEl.appendChild(linkBreakEl);

  let descriptionEl = document.createElement("p");
  descriptionEl.classList.add("link-description");
  descriptionEl.textContent = description;
  resultItemEl.appendChild(descriptionEl);

  searchResultsEl.appendChild(resultItemEl);
}
```

```
function displayResults(searchResults) {
  spinnerEl.classList.add("d-none");

  for (let result of searchResults) {
    createAndAppendSearchResult(result);
  }
}

function searchWikipedia(event) {
  if (event.key === "Enter") {

    spinnerEl.classList.remove("d-none");
    searchResultsEl.textContent = "";

    let searchInput = searchInputEl.value;
    let url = "https://apis.ccbp.in/wiki-search?search=" +
searchInput;
    let options = {
      method: "GET"
    };

    fetch(url, options)
      .then(function (response) {
        return response.json();
      })
      .then(function (jsonData) {
```

```
        let { search_results } = jsonData;
        displayResults(search_results);
      });
  }
}

searchInputEl.addEventListener("keydown", searchWikipedia);
```

1. HTML Forms

The HTML Forms can be used to collect data from the user.

Forms are of different kinds:

- Login/Sign in Form
- Registration Form
- Contact Us Form, etc.

1.1 HTML Form Element

The HTML `form` element can be used to create HTML forms. It is a container that can contain different types of Input elements like Text Fields, Checkboxes, etc.

HTML

```
1  <form></form>
```

Note

Whenever we click a button or press `Enter` key while editing any input field in the form, the `submit` event will be triggered.

2. Event Object Methods

2.1 preventDefault

The `preventDefault()` method prevents the occurrence of default action.

Here in the form, it prevents the default behaviour of the `submit` event.

JAVASCRIPT

```
let myFormEl = document.getElementById("myForm");

myFormEl.addEventListener("submit", function(event) {
  event.preventDefault();
});
```

3. Event Types

There are different types of events.

- Keyboard Events
- Mouse Events
- Touch Events
- Form Events, etc.

3.1 Form Events

A Form Event is an event that can occur within a form.

Some of the form events are:

- blur
- focus
- change, etc.

3.1.1 Blur Event

The `blur` event happens when an HTML element has lost focus.

```
let nameEl = document.getElementById("name");

nameEl.addEventListener("blur", function(event) {
  console.log("blur event triggered");
});
```

Try out the `preventDefault` method and `blur` event in the below Code Playground.

Code:

```
<!DOCTYPE html>
<html>
 <head>
  <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css"
integrity="sha384-JcKb8q3iqJ61gNV9KGb8thSsNjpSL0n8PARn9HuZOnIxN0hoP+VmmDGMN5t9UJ0Z"
crossorigin="anonymous" />
  <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"
integrity="sha384-DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXaRkfj"
crossorigin="anonymous"></script>
  <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.1/dist/umd/popper.min.js"
integrity="sha384-9/reFTGAW83EW2RDu2S0VKaIzap3H66lZH81PoYlFhbGU+6BZp6G7niu735Sk7lN"
crossorigin="anonymous"></script>
  <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"
integrity="sha384-B4gt1jrGC7Jh4AgTPSdUtOBvfO8shuf57BaghqFfPlYxofvL8/KUEfYiJOMMV+rV"
crossorigin="anonymous"></script>
 </head>
 <body>
  <div class="container">
   <h1 class="form-heading">Add User</h1>
   <form id="myForm">
    <div class="mb-3">
     <label for="name">Name</label>
     <input type="text" class="form-control" id="name" />
     <p id="nameErrMsg" class="error-message"></p>
    </div>
    <div class="mb-3">
     <label for="email">Email</label>
     <input type="text" class="form-control" id="email" />
     <p id="emailErrMsg" class="error-message"></p>
    </div>
    <button type="submit" class="btn btn-primary">Submit</button>
   </form>
```

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

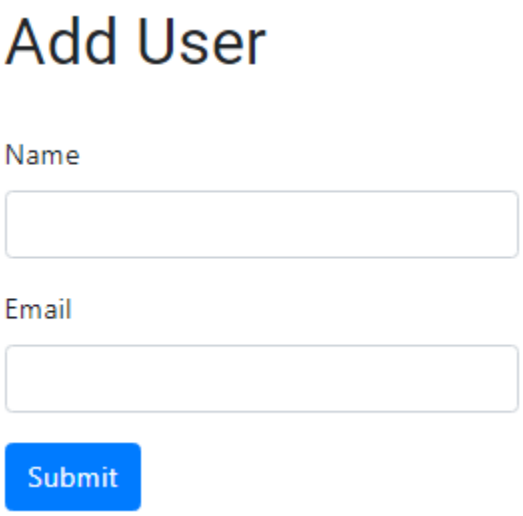
Css:

```
@import
url("https://fonts.googleapis.com/css2?family=Bree+Serif&family=Caveat:wght@400;700&family=Lobster&family=Monoton&family=Open+Sans:ital,wght@0,400;0,700;1,400;1,700&family=Playfair+Display+SC:ital,wght@0,400;0,700;1,700&family=Playfair+Display:ital,wght@0,400;0,700;1,700&family=Roboto:ital,wght@0,400;0,700;1,400;1,700&family=Source+Sans+Pro:ital,wght@0,400;0,700;1,700&family=Work+Sans:ital,wght@0,400;0,700;1,700&display=swap");

.form-heading {
  font-family: "Roboto";
  font-size: 36px;
  padding-top: 40px;
  padding-bottom: 20px;
}

.error-message {
  color: #dc3545;
  font-family: "Roboto";
  font-size: 14px;
}
```

Javascript:

```
let myFormEl = document.getElementById("myForm");

let nameEl = document.getElementById("name");
let nameErrMsgEl = document.getElementById("nameErrMsg");

let emailEl = document.getElementById("email");
let emailErrMsgEl = document.getElementById("emailErrMsg");

nameEl.addEventListener("blur", function(event) {
  if (event.target.value === "") {
```

```
    nameErrMsgEl.textContent = "Required*";
  } else {
    nameErrMsgEl.textContent = "";
  }
});

emailEl.addEventListener("blur", function(event) {
  if (event.target.value === "") {
    emailErrMsgEl.textContent = "Required*";
  } else {
    emailErrMsgEl.textContent = "";
  }
});

myFormEl.addEventListener("submit", function(event) {
  event.preventDefault();
});
```



Forms | Part-2 | Cheat Sheet

1. HTML Select Element

The HTML `select` element is used to create a drop-down list.

HTML

```html
<select></select>
```

1.1 HTML Option Element

The HTML `option` element is used to create the menu option of a drop-down list.

The text content of the HTML `option` element is used as a label.

HTML

```html
<select>
  <option>Active</option>
</select>
```

1.1.1 The value Attribute

Every HTML `option` element should contain the HTML `value` attribute.

```HTML
<option value="Active">Active</option>
```

2. HTML Input Element

2.1 Radio

The HTML `input` radio element is used to select one option among a list of given options.

```HTML
<input type="radio" id="genderMale" value="Male" />
<input type="radio" id="genderFemale" value="Female" />
```

2.1.1 HTML name attribute

The HTML `name` Attribute specifies the name for an HTML Element.

```HTML
<input type="radio" value="Male" name="gender" />
```

2.1.2 Radio Group

All the radio buttons with same name collectively called as a radio group.

We can select only one radio button within a radio group.

HTML

```html
<input type="radio" value="Male" name="gender" />
<input type="radio" value="Female" name="gender" />
```

3. Boolean Attributes

For the HTML Boolean attributes, we only specify the name of the HTML attribute.

The presence of a boolean attribute represents the `true` value, and the absence represents the `false` value.

3.1 HTML selected attribute

The `selected` attribute specifies that an option should be pre-selected when the page loads.

HTML

```html
<option value="Active" selected>Active</option>
```

3.2 HTML checked attribute

The `checked` attribute specifies that an input element should be pre-selected (checked) when the page loads.

HTML

```html
<input type="radio" id="genderMale" value="Male" name="gender" checked />
```

Try out the HTML `select` element, `input` radio element and the boolean attributes in the below Code Playground.

Code:

```
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css"
integrity="sha384-JcKb8q3iqJ61gNV9KGb8thSsNjpSL0n8PARn9HuZOnIxN0hoP+VmmDGMN5t9UJ0Z"
crossorigin="anonymous" />
    <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"
integrity="sha384-DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXaRkfj"
crossorigin="anonymous"></script>
```

```
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.1/dist/umd/popper.min.js"
integrity="sha384-9/reFTGAW83EW2RDu2S0VKaIzap3H66lZH81PoYlFhbGU+6BZp6G7niu735Sk7lN"
crossorigin="anonymous"></script>
    <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"
integrity="sha384-B4gt1jrGC7Jh4AgTPSdUtOBvfO8shuf57BaghqFfPlYxofvL8/KUEfYiJOMMV+rV"
crossorigin="anonymous"></script>
  </head>
  <body>
    <div class="container">
      <h1 class="form-heading">Add User</h1>
      <form id="myForm">
        <div class="mb-3">
          <label for="name">Name</label>
          <input type="text" class="form-control" id="name" />
          <p id="nameErrMsg" class="error-message"></p>
        </div>
        <div class="mb-3">
          <label for="email">Email</label>
          <input type="text" class="form-control" id="email" />
          <p id="emailErrMsg" class="error-message"></p>
        </div>
        <div class="mb-3">
          <label for="status">Working Status</label>
          <select id="status" class="form-control">
            <option value="Active">Active</option>
            <option value="Inactive">Inactive</option>
          </select>
        </div>
        <div class="mb-3">
          <h1 class="gender-field-heading">Gender</h1>
          <input type="radio" name="gender" id="genderMale" value="Male" checked />
          <label for="genderMale">Male</label>
          <input type="radio" name="gender" id="genderFemale" value="Female" class="ml-2"/>
          <label for="genderFemale">Female</label>
        </div>
        <button class="btn btn-primary">Submit</button>
      </form>
    </div>
  </body>
</html>
```

Css:

```
@import
url("https://fonts.googleapis.com/css2?family=Bree+Serif&family=Caveat:wght@400;700&family=Lobster&family=Monoton&family=Open+Sans:ital,wght@0,400;0,700;1,400;1,700&family=Playfair+Display+SC:ital,wght@0,400;0,700;1,700&family=Playfair+Display:ital,wght@0,400;0,700;1,700&family=Roboto:ital,wght@0,400;0,700;1,400;1,700&family=Source+Sans+Pro:ital,wght@0,400;0,700;1,700&family=Work+Sans:ital,wght@0,400;0,700;1,700&display=swap");
```

```
.form-heading {
  font-family: "Roboto";
  font-size: 36px;
  padding-top: 40px;
  padding-bottom: 20px;
}

.error-message {
  color: #dc3545;
  font-family: "Roboto";
  font-size: 14px;
}

.gender-field-heading {
  color: #212529;
  font-size: 18px;
  margin-bottom: 10px;
}
```

Javascript:

```
let nameEl = document.getElementById("name");
let nameErrMsgEl =
document.getElementById("nameErrMsg");

let emailEl = document.getElementById("email");
let emailErrMsgEl =
document.getElementById("emailErrMsg");

let workingStatusEl =
document.getElementById("status");
```

```
let genderMaleEl =
document.getElementById("genderMale");
let genderFemaleEl =
document.getElementById("genderFemale");

let myFormEl = document.getElementById("myForm");

let formData = {
  name: "",
  email: "",
  status: "Active",
  gender: "Male"
};

nameEl.addEventListener("change", function(event) {
  if (event.target.value === "") {
    nameErrMsgEl.textContent = "Required*";
  } else {
    nameErrMsgEl.textContent = "";
  }

  formData.name = event.target.value;
});
```

```
emailEl.addEventListener("change", function(event) {
  if (event.target.value === "") {
    emailErrMsgEl.textContent = "Required*";
  } else {
    emailErrMsgEl.textContent = "";
  }

  formData.email = event.target.value;
});

workingStatusEl.addEventListener("change",
function(event) {
  formData.status = event.target.value;
});

genderMaleEl.addEventListener("change",
function(event) {
  formData.gender = event.target.value;
});

genderFemaleEl.addEventListener("change",
function(event) {
```

```
  formData.gender = event.target.value;
});

function validateFormData(formData) {
  let {name, email} = formData;
  if (name === "") {
    nameErrMsgEl.textContent = "Required*";
  }
  if (email === ""){
    emailErrMsgEl.textContent = "Required*";
  }
}

function submitFormData(formData) {
  let options = {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
      Accept: "application/json",
      Authorization:
        "Bearer
00f3f8fde06120db02b587cc372c3d85510896e899b457
74068bb750462acd9f",
```

```
    },
    body: JSON.stringify(formData)
  };

  let url = "https://gorest.co.in/public-api/users";

  fetch(url, options)
    .then(function(response) {
      return response.json();
    })
    .then(function(jsonData) {
      console.log(jsonData);
      if (jsonData.code === 422) {
        if (jsonData.data[0].message === "has already
been taken") {
          emailErrMsgEl.textContent = "Email Already
Exists";
        }
      }
    });
}

myFormEl.addEventListener("submit", function(event){
```

```
  event.preventDefault();
  validateFormData(formData);
  submitFormData(formData);
});
```

Output: