

Forms

Forms are referred to a document that contains spaces for user or visitors to fill in information. In HTML, forms are refer to different elements that allow you to collect information from visitors to your website.

How forms work?

A form is created using HTML and CSS. On the site, visitors fill up the form and presses a button to submit the information to the server.

JavaScript collect each form control and prepare them to send to the server along with the value the user enters or selects.

The server processes the information using programming language such as PHP, C#, Java, or Python. This process may also store the information in a database.

The server creates a new page to send back to the browser based on the information received.

Form controls elements

There are several types of form controls that can collect information from visitors of a website. The server needs to know which piece of inputted data corresponds with which form element.

<form>

Form control live inside a **<form>** element. This element should always carry the action attribute and will usually have a method and id attribute too.

Action

Every **<form>** element requires an action attribute. Speaking of accessibility, forms should always have a valid action attribute value, allowing the form data to go through to a real resource when JavaScript is not enabled. On live sites, you must always create a fallback page that will handle the form's submission should the user not have JavaScript enabled. You can change the action value dynamically using JavaScript:

```
document.getElementById('theForm').action = 'otherPage.php';
```

By doing so, you can have JavaScript-enabled users head to a different location than the non-JavaScript users upon the form's submission.

Its value is the URL for the page on the server that will receive the information in the form when it is submitted. In other words, is useful for telling the browser which URL should receive and

respond to the form data. We used '.' as its value because we want the form data to be sent to the current URL.x

Method

Forms can be sent using one of two methods: **get or post**

With the **get** method, the values from the form are added to the end of the URL specified in the action attributes. The **get** method is ideal for:

- Short forms (such as search boxes)
- When you are just retrieving data from the web server (not sending information that should be added to or deleted from a database)

GET: This is used to retrieve a resource. You want information? No database update? Use GET!

POST: This is used to insert new data into the server, such as adding a new employee in your database.

With the **post** method the values are sent in what are known as HTTP headers. As a rule of thumb you should use the post method if your form:

Allows user to upload a file

- Is very long
- Contains sensitive data
- Adds information to, or deletes information from, a database
- If the method attributes is not used, the form data will be sent using the get method.

The **method** attribute defines how you want your form data to be sent to the server when it is sent. Its value could be either **get** or **post**. You should use **get**, which is the default, only when the form data is small (a few hundred characters), not sensitive (it doesn't matter if someone else sees it) and there are no files in the form. These requirements exist because when using **get**, all the form data will be appended to the current URL as encoded parameters before being sent.

Id

Id attributes is used to identify the form from other elements on the page, and also used by JS to check validate the form.

<input>

The **<input>** element is used to create several different form controls. The value of the type attribute determines what kind of input they will be crating. For each type it will have different attributes.

type="submit"

input element with type submit is used as a button to send a form to the server.

Attributes:

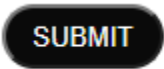
name: identifies the form control and is sent along with the information, if have any, to the server. In this case, since the input type is submit, it does not need to have one.

Value: is used to control the text that appears on a button.

Example: use a submit input to submit a form. When the form is submitted, take the user to a webpage that says that the form has been submitted.

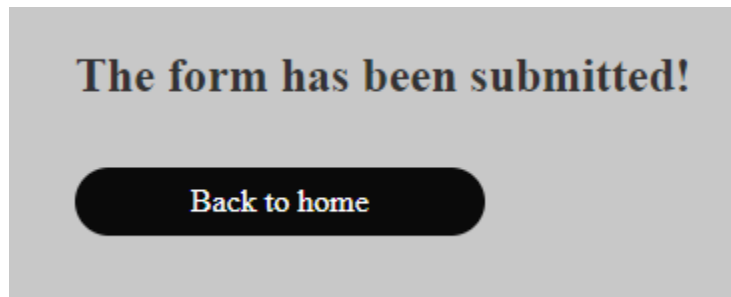
```
<body>
  <form class="" action="formSent.html" method="post">
    <input type="submit" value="SUBMIT">
  </form>
</body>
</html>
```

```
input{
  background-color: rgb(10,10,10);
  color: white;
  padding: 0.5em 1em;
  border-radius: 20px;
}
input:hover{
  background-color:rgb(150,150,150);
  box-shadow: 0px 0px 10px 0.5px black;
  color: black;
}
```



```
<h2>The form has been submitted!</h2>
<a href="forms.html">Back to home</a>
```

```
body{
  background-color: rgb(200,200,200);
  padding: 2em;
}
a{
  margin-top:2em;
  text-decoration:none;
  display:block;
  color:white;
  width: 20%;
  text-align:center;
  padding: 0.5em 1em;
  background-color: rgb(10,10,10);
  border-radius: 20px;
}
a:hover{
  background-color:rgb(150,150,150);
  box-shadow: 0px 0px 10px 0.5px black;
  color: black;
}
h2{
  color: rgb(50,50,50);
}
```



type="text"

attribute text creates a single line text input

name

attribute name is required to identify that the value collected from the text input.

maxlength

You can use the `maxlength` attribute to limit the number of characters a user may enter into the text field.

Value

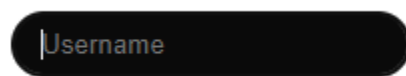
Attribute value inputs a value to the text input.

Placeholder

Creates information in the back of the text input.

Example)

```
<input type="text" name="username" placeholder="Username" maxlength="20">
```



Creating Error message

In order to do everything required, the function needs to take two arguments: the id value of the form element to which the error message is being applied and the error message itself. The function should then create a span containing the message and append the span after the target element. The function also needs to give this new span an id value, so that the function itself can check for its existence upon repeat calls (no need to create it if it's already there), and so that another function can remove that error message when appropriate. The function will also add the *error* class to the form element's label

Temporary Storage

The `localStorage` object allows you to save key/value pairs in the browser.

The `sessionStorage` object let you store key/value pairs in the browser.

The `setItem()` method sets the value of the specified Storage Object item.

The `setItem()` method belongs to the Storage Object, which can be either a [localStorage](#) object or a [sessionStorage](#) object.

Syntax

```
localStorage.setItem(keyname, value)
```

Or:

```
sessionStorage.setItem(keyname, value)
```

Example)

index.html

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title></title>
  </head>
  <body>
    <form action="display.html" method="get">
      <input type="text" name="firstName" id="firstN"/>
      <input type="text" name="lastName" id="lastN"/>
      <input type="submit" value="Submit" onclick="passvalues()"/>
    </form>

    <script type="text/javascript">
      function passvalues(){
        var fName = document.querySelector('#firstN').value;
        var Lname = document.querySelector('#lastN').value
        localStorage.setItem('name1', fName);
        localStorage.setItem('name2', LName);
        return false;
      }
    </script>

  </body>
</html>
```

display.html

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title></title>
  </head>
  <body>
    <h2>Welcome to the program</h2>
    <p>First Name: <span id="result-firstName"></span> </p>
```

```
<p>Last Name: <span id="result-lastName"></span> </p>
<script>
    document.querySelector('#result-firstName').innerHTML =
localStorage.getItem('name1');
    document.querySelector('#result-lastName').innerHTML =
localStorage.getItem('name2');
</script>
</body>
</html>
```

NOTE: NOT ALL BROWSER READ localStorage