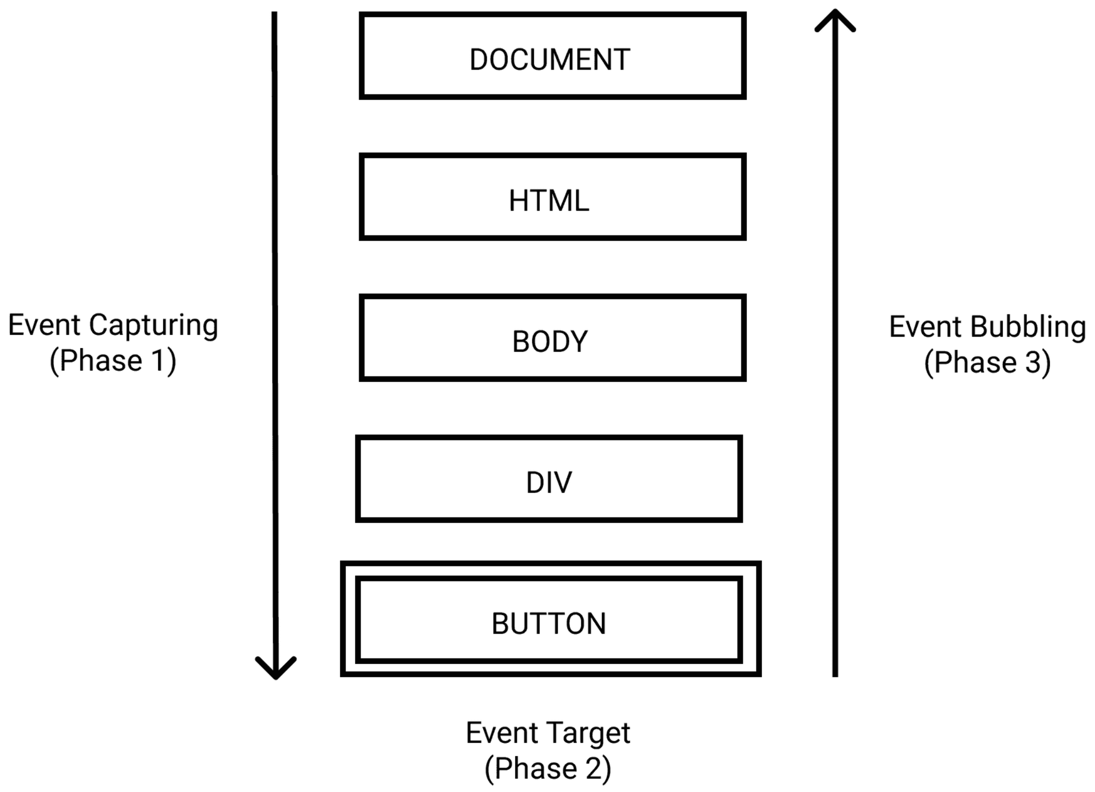
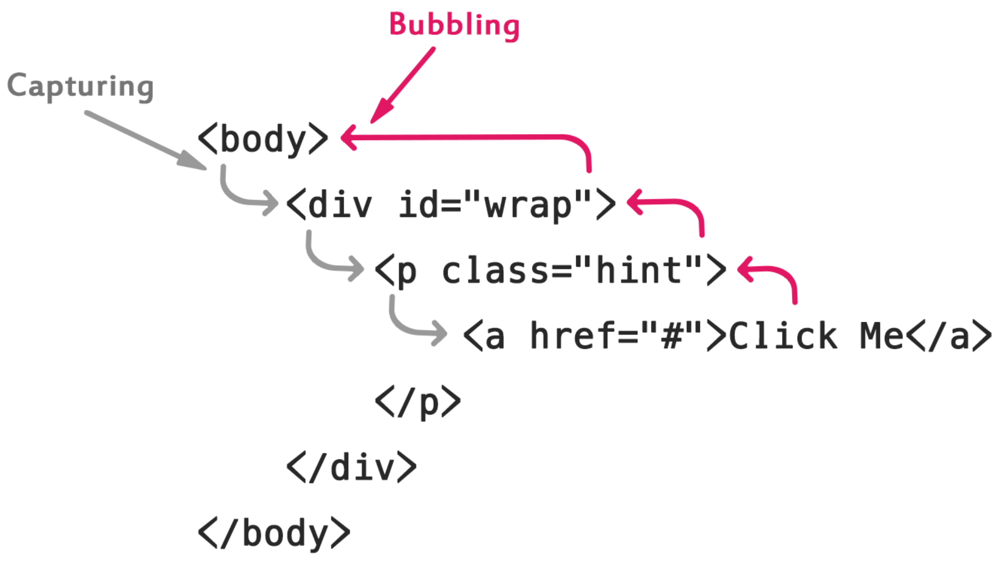
preventdefault vs stoppropagation vs stopimmediatepropagation JS

To know those we must know the event bubbling and event capturing





```
<body>
  <div id="wrap">
    <p class="hint">
      <a href="#">Click Me</a>
    </p>
  </div>
</body>
```

The code :

- Event.preventDefault()
- Event.stopPropagation()
- Event.stopImmediatePropagation()

Definition :

- **preventDefault:** Cancels the event if it is cancelable, without stopping further propagation of the event.
- **stopPropagation:** Prevents further propagation of the current event.
- **stopImmediatePropagation:** Prevents other listeners of the same event from being called.

Example:

The html code

```
<form id="myForm" action="/my-handling-form-page"
method="post">
    Name:


    E-mail:


    Message
    Send your messag
</form>
```

While e.preventDefault() :

```
$('#myForm').on('submit', function(e) {
  e.preventDefault(); // Now nothing will happen
});
```

Note ⇒ Event.preventDefault will ensure that the form is never submitted, but it won’t control or prevent that submit or click event from bubbling up. That’s what the other two are for.

---

whilee.stopPropagation()

Note⇒ stopPropagation ensures that the event doesn’t bubble any further. Let’s look at another code example:

```
    Click Me!
$('.container').on('click', function(e) {
    console.log('container clicked');
});

$('.element').on('click', function(e) {
    e.preventDefault(); // Now link won't go anywhere
    console.log('element clicked');
});
```

⇒ Now if you were to click the link with the console open, you would see:

```
"element was clicked"
"container was clicked"
```

Now let’s add Event.stopPropagation:

```
$('.container').on('click', function(e) {
    console.log('container clicked');
});

$('.element').on('click', function(e) {
    e.preventDefault(); // Now link won't go anywhere
    e.stopPropagation(); // Now the event won't bubble up
    console.log('element clicked');
});
```

And click the link again. This time we see:

"element was clicked"

While e..stopImmediatePropagation()

Note⇒ This gets you 90% of everything you'll need as far as manipulating events. But let's pose a last use case that can prove difficult.

We'll start off with similar markup, except we'll give the anchor two classes. A generic one, *item*, that all anchors in this area will get, and a specific one, *element*, that's very important for our application to work.

example:

Click Me!

And we'll add our nifty Event.stopPropagation we learned about in the last section!

```
$('.item').on('click', function(e) {

    console.log('anm was clicked');
});



$('.element').on('click', function(e) {

    e.preventDefault(); // Now link won't go anywhere

    e.stopPropagation(); // Now the event won't bubble up

    console.log('element clicked');

});
```

But, when we click on the element we see:

```
"an item was clicked"

"element was clicked"
```

The problem here is that item and element are evenly ranked in the DOM. It’s not as though it hits element and then bubbles up to container like we saw in the last example. Since the click event fires on both element and item at the same time, you cannot stopPropagation like you’d expect.

This is where stopImmediatePropagation comes in handy!

```
$('.element').on('click', function(e) {

    e.preventDefault(); // Now link won't go anywhere

    e.stopImmediatePropagation(); // Now item on click won't fire

    console.log('element clicked');

});

$('.item').on('click', function(e) {

    console.log('anm was clicked');

});
```

Now one important thing to note here is: Battles over immediate propagation go to the first one declared in your script (or series of scripts). So as you may have noticed, in order to get stopImmediatePropagation working we had to switch the listeners so that element comes before item!

Let’s run it one last time and we should see:

```
"element was clicked"
```

_____event.preventDefault()

This method is used to stop the browser's default behavior when performing an action. Following are some examples of the default behavior:

Clicking on a checkbox/radio input selects/deselects a checkbox

Clicking on an input/text-area field focuses the input and places the cursor in the input element

By using event.preventDefault(), the default behavior of an HTML element can be prevented.

I have added an event.preventDefault() on the click of a checkbox and notice it prevents the checkbox from getting
checked.____

event.stopPropagation()

This method is used to prevent the propagation of an event as defined in the capturing/bubbling phase of the flow of an event in the browser.

Consider the following HTML pseudocode:

```
<div class="parent" (onClick)="console.log('parent')">
   <button class="child" (onClick)="console.log('child')"></button>
</div>
```

If the browser is supporting event bubbling, clicking on the button will print:

```
child
parent
```

If we change the click event of the button to the following function call …

```
<div class="parent" (onClick)="console.log('parent')">
   <button class="child" (onClick)="buttonClick(event)"></button>
</div>
<script>
   function buttonClick(event) {
      event.stopPropagation();
      console.log('child');
   }
</script>
```

… now clicking on the button will print:

```
child
```

Since we stopped the propagation (event bubbling, in this case) of the event on clicking, it printed only child in the console.

---

event.stopImmediatePropagation()

What if we have multiple-click listeners on a single HTML element?

Consider the following code snippet:

```
<script>
   $("div").click(function(event) {
     event.stopImmediatePropagation();
     alert('First click triggered');
   });
```

```
    $("div").click(function(event) {
      // This function won't be executed
      alert('Second click triggered');
    });
</script>
<div>Click me</div>
```

Also try commenting the stopImmediatePropagation() statement in the Fiddle.

With stopImmediatePropagation(), along with the event propagation, other event handlers will also be prevented from execution.

As a result, clicking on the div element will:

Prevent event bubbling to the parent elements

Prevent the execution of any other event listener attached to the element

Thus we can say that:

stopImmediatePropagation = stopPropagation + (other event listeners removed)
