Lesson **Wednesday**

Introduction to Programming (/introduction-to-programming) / JavaScript and Web Browsers (/introduction-to-programming/javascript-and-web-browsers) / Practical Tips for Researching Web APIs

Text

The lessons in LearnHowToProgram.com don't cover every tools that's out there. First of all, that would be an overwhelming amount of information. But it would also just be impossible because there's too much out there! Fundamentally, LearnHowToProgram.com is not meant to be a complete reference — that's the job of sites like MDN (https://developer.mozilla.org/en-US/) and W3Schools (https://www.w3schools.com/). What we do provide are guides on computer programming concepts, tools, and best practices, as well as a few (incomplete) references of certain tools. That means it's always up to you to do research online to expand your toolset and figure out how to code the extended functionality that you dream up for your projects.

So, let's continue to build our research and reference skills! In this lesson, we'll build on what we learned in previous lessons on understanding Web APIs and navigating Mozilla Developer Network (MDN) documentation. This time our focus will be on practical tools and tips for general research and referencing MDN docs. Specifically, we'll cover these topics:

- Possible points of confusion about Web APIs.
- Researching based on your goal, referencing the target element or event that you want to learn more about.
- Tips for search engine queries.
- How to use the DevTools console to explore object properties.
- Links to helpful reference pages.
- Certain features on MDN documentation to be aware of.

Please revisit this lesson as necessary.

Possible Points of Confusion

When researching for information online about Web APIs, we need to remember a few things:

- Web API interfaces (the object types) share functionality with each other through inheritance. Inheritance is the mechanism by which one object type, like `HTMLInputElement` , has access to properties that are defined in another object type, like `HTMLElement` . That means that the documentation for a property may be associated with a different object type that you may initially expect.
    - If you want to review the discussion on inheritance, visit this lesson (https://www.learnhowtoprogram.com/introduction-to-programming/javascript-and-web-browsers/understanding-web-apis-interfaces-object-types-and-inheritance).
- Events are always tied to specific object types (Web API interfaces), like `Window` , `Document` , or `Element` . This means that an event is categorized as belonging to the browser window, like when a user prints a page, or as belonging to the DOM, like when a user copies text on a webpage.
- Sometimes multiple Web APIs can respond to the same event type:
    - For the "keyup" keyboard event, there's an `onkeyup` event handler property for `Element`

(https://developer.mozilla.org/en-US/docs/Web/API/Element#keyboard_events) and `Document` (https://developer.mozilla.org/en-US/docs/Web/API/Document#keyboard_events).
    - For the "copy" clipboard event, there's an `oncopy` event handler property for `Element` (https://developer.mozilla.org/en-US/docs/Web/API/Element#clipboard_events), `Window` (https://developer.mozilla.org/en-US/docs/Web/API/Window#clipboard_events), and `Document` (https://developer.mozilla.org/en-US/docs/Web/API/Document#clipboard_events).

While the above points about Web APIs are not inherently confusing, they can make finding the right documentation confusing because the DOM element we may want to manipulate or set up event handling for could have tools that are described in various places in the documentation.

We won't be able to completely avoid confusion, but we can mitigate it by using practical research tips, so continue reading!

Also, always remember to share and discuss your confusions with your peers and your teacher. If you have a question, it's very likely that a fellow student has the same one.

Tool #1: Researching Based on your Goal

Probably the quickest way to find information on something new that you would like to do is by developing a search engine query that focuses on what you want to do. For example, if we wanted to manipulate an HTML link with our JavaScript, we might go through this series of researching steps:

- If we've forgotten what tag we need to make an HTML link, research "how to make a link in HTML", and we'll learn that we need to use an `<a>` tag.

- If we already know we're working with an anchor element, but we've forgotten the name of the attribute for setting the destination of the hyperlink, we would research "HTML a tag attributes" or "HTML anchor attributes" to learn that we want to access the `href` attribute.
- To learn how to set the `href` attribute of a DOM element, we would research something like "DOM anchor href attribute" or "set DOM a tag href property". This will help us find a reference page on the HTML DOM anchor element or the href property belonging to the anchor element.

The same process should be true for applying an inline CSS style, or targeting an event on a specific HTML element. For example, we might use the following search queries:

- "how to hide DOM element"
- "change background color of element with JS"
- "event handler for change in html input"
- "event handler for copy on html element"
- "onevent for scroll"
- "onevent for resizing window"

Note: when researching events and event handling, you will likely see solutions that use **event listeners** with methods like `addEventListener()`. Don't worry about those for now — we'll be covering event listening later in this course section.

Tips for Great Search Queries

If you aren't getting the information you need from your search, try reformulating your query to use more specific terminology. For example, we could differentiate between an "HTML element" that might go in our `index.html` file, and a "DOM element" that we would access with our JS.

If you want to use a search engine like Google to search a specific webpage, you can always add `site:www.website.com` to your query, where `www.website.com` is the name of the website you want to search. Alternatively, you could also just add the name of the site to your search query. For example, if you specifically want a resource from MDN, simply add "MDN" to your search query.

If you want to make sure that the search results include the exact text (word or phrase) of your search query, add quotes around it. For example if we use the following search query, any returned results must contain the word "DOM":

```
change background color of "DOM" element with JS
```

The more you practice writing search queries, the better you will become at formulating them!

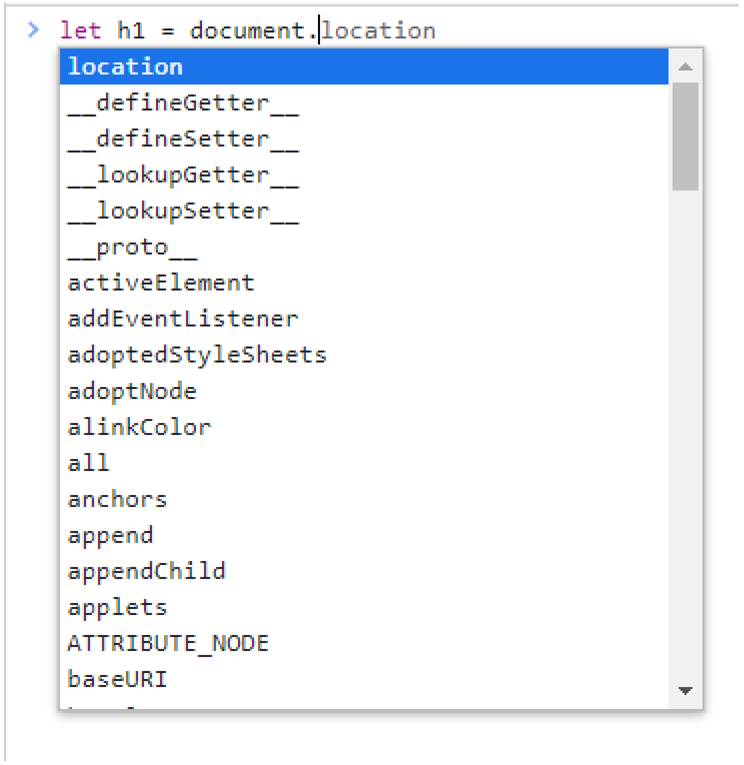
Tool #2: Using the DevTools Console to Explore Objects

If you have autocompletion turned on in your DevTools, this will give you a wealth of information in your DevTools console.

To check on or change your DevTools settings for autocompletion, follow these instructions:

- Click the "Settings" cog icon in the upper-right hand corner of the DevTools window.
- This should open the "Preferences" page, but if not, select the "Preferences" menu option.
- Scroll down to the "Sources" section of the page, and find the checkbox titled "Autocompletion" and uncheck/check the box to set your preference.

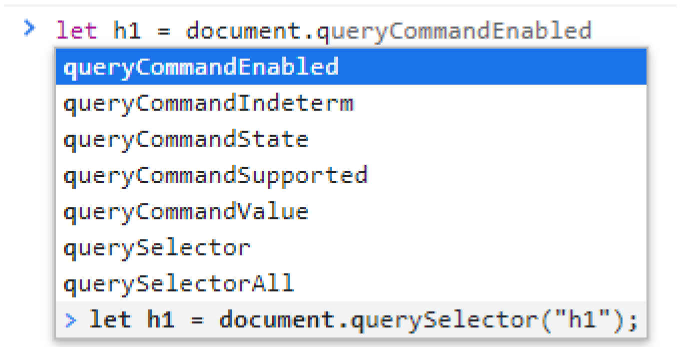
Check out the image below that demonstrates the power of the autocompletion feature.



```
> let h1 = document.location
```

In the image above, we've typed in `let h1 = document.` and a box appears with autocompletion suggestions. Here, suggestions are listed alphabetically, and we can scroll through and look at all of the `document` properties. Take note — there's a limit to the number of properties that the autocompletion box can list, so oftentimes the list in the box won't be complete.

We can type in a letter to narrow down the list of autocompleted suggestions. In the image below, we've typed in `let h1 = document.q`, and we can see that there's grayed out letters offering a suggestion of `let h1 = document.queryCommandEnabled`, and the box of autocompletion suggestions have narrowed down to only `document` properties that start with `q`.

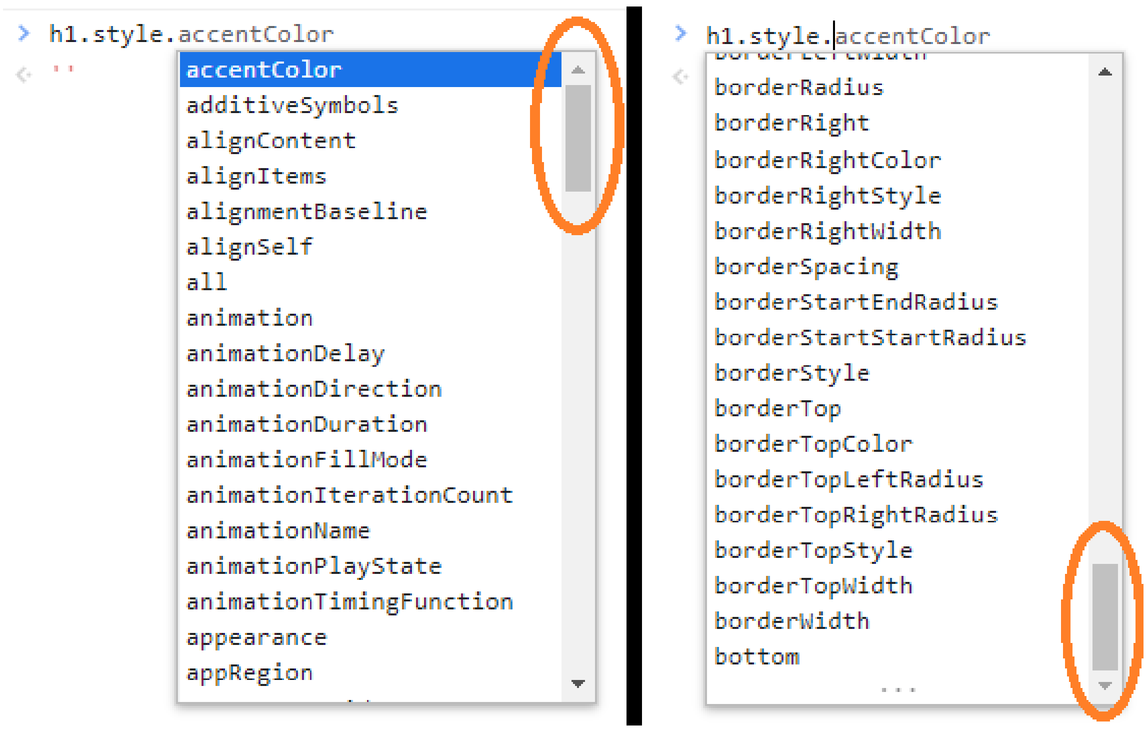


```
> let h1 = document.queryCommandEnabled
  queryCommandEnabled
  queryCommandIndeterm
  queryCommandState
  queryCommandSupported
  queryCommandValue
  querySelector
  querySelectorAll
  > let h1 = document.querySelector("h1");
```

Also notice in the image above the suggestion in the autocompletion pop-up box of `> let h1 = document.querySelector("h1");` — this is a suggestion that matches one of my previously inputted commands to the console.

Everything that we've just done with the autocompletion feature and the `document` object can also be applied to other objects. Take the image below of the `style` object as another example. Here, we've entered `h1.style.` and we've received an alphabetical list of all properties belonging to the `style` object (pictured on the left). Pretty cool!

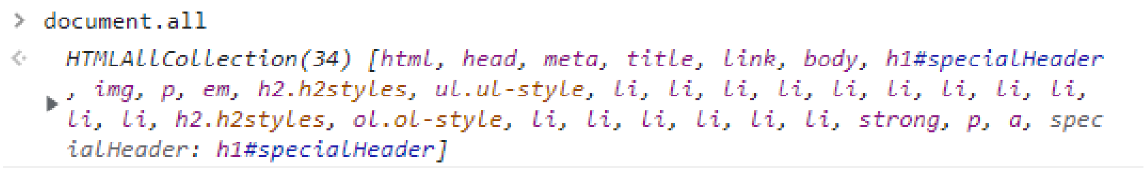
However, keep in mind that the autocompletion suggestion box has a limit to what it can list. If we scroll down to the bottom of the box (pictured on the right), we can see that the list ends in the `b` s and there's a bar with `. . .` . This simply means that it's not a complete list of all `style` properties. In this case, try typing in another letter, like `h1.style.b` , to get a list of all `style` properties beginning with a `b` .



```
> h1.style.accentColor
<· ''
accentColor
additiveSymbols
alignContent
alignItems
alignmentBaseline
alignSelf
all
animation
animationDelay
animationDirection
animationDuration
animationFillMode
animationIterationCount
animationName
animationPlayState
animationTimingFunction
appearance
appRegion
```

```
> h1.style.accentColor
borderRadius
borderRight
borderRightColor
borderRightStyle
borderRightWidth
borderSpacing
borderStartEndRadius
borderStartStartRadius
borderStyle
borderTop
borderTopColor
borderTopLeftRadius
borderTopRightRadius
borderTopStyle
borderTopWidth
borderWidth
bottom
```

There's also a limit to how helpful this information is — we can see all of the property names associated with an object, but we don't know what they do. One option is to simply access the property and see what you get. In the image below, I've done just that — I tried out `document.all` and I got back a large object. However, when I researched the property on MDN (https://developer.mozilla.org/en-US/docs/Web/API/Document/all), I learned that it's a deprecated property that I shouldn't use. The important lesson here is that you always need to research any property that you find via autocompletion to make sure that it is relevant to your code.



```
> document.all
<· ▶ HTMLAllCollection(34) [html, head, meta, title, link, body, h1#specialHeader
    , img, p, em, h2.h2styles, ul.ul-style, li, li, li, li, li, li, li, li, li,
    li, li, h2.h2styles, ol.ol-style, li, li, li, li, li, li, strong, p, a, spec
    ialHeader: h1#specialHeader]
```

Tool #3: Using Reference Pages Effectively

The last tip is to locate references for topics within HTML, CSS, or Events. This is especially helpful for broad exploration. Let's work through a few examples of helpful reference pages.

Adding Styles to DOM Elements

If you want to explore new styles to add to an HTML DOM element, you could find a resource that lists all `style` object properties. Check out these two resources that do just that:

- W3Schools' `style` object properties (https://www.w3schools.com/jsref/dom_obj_style.asp)
- CSS Properties Reference listing CSS and JS equivalents (https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Properties_Reference).

Adding Attributes to DOM Elements

You could peruse a list of global attributes that can be applied to any HTML element to get an idea of what's out there. Check out the following two resources as an example:

- W3School's reference on global attributes (https://www.w3schools.com/tags/ref_standardattributes.asp)
- MDN's reference on global attributes (https://developer.mozilla.org/en-US/docs/Web/HTML/Global_attributes)

Exploring Events to Handle

There are also lists of the most common events to handle in our webpages. MDN has a great reference page for this here:

- Events (https://developer.mozilla.org/en-US/docs/Web/Events)

Exploring HTML Element Object Types

For finding new properties, methods, and events to use/handle with HTML DOM elements, these references will be the most informative:

- `Element` (https://developer.mozilla.org/en-US/docs/Web/API/Element)
- `HTMLElement` (https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement)

To see a list of all DOM Element objects that represent specific HTML elements, like `HTMLHeadingElement` or `HTMLParagraphElement`, visit this link:

- HTML Element Interfaces (https://developer.mozilla.org/en-US/docs/Web/API/HTML_DOM_API#html_element_interfaces_2)

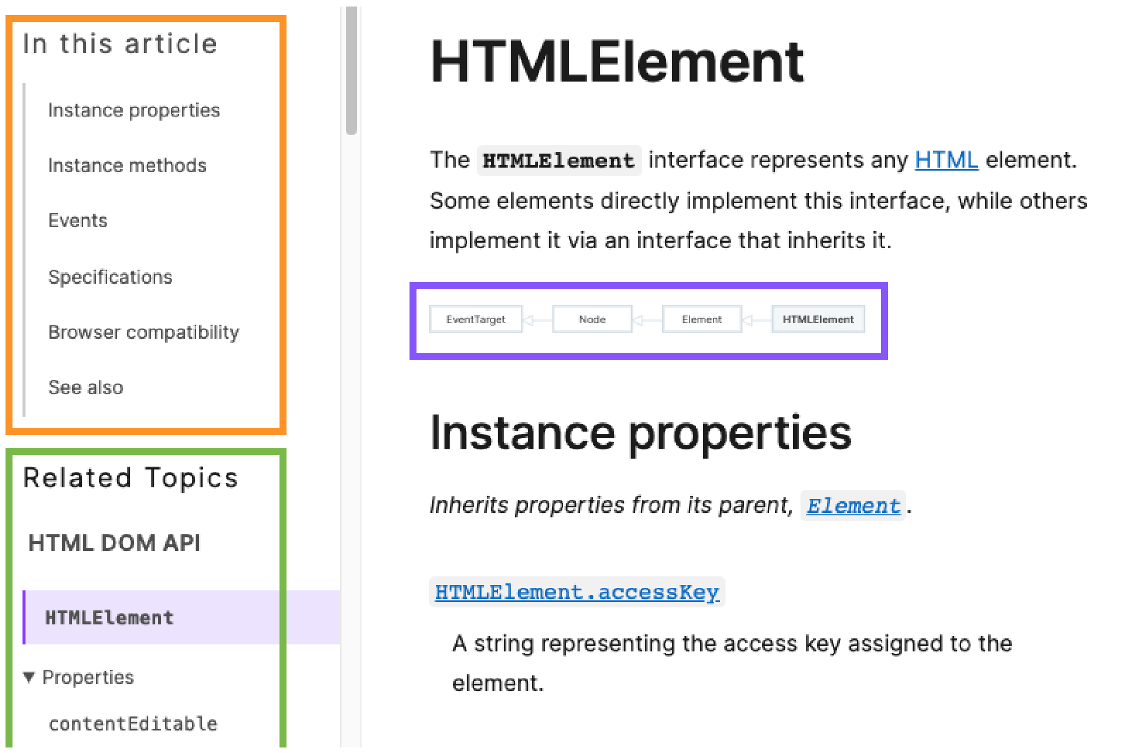
Features of MDN Documentation for Web APIs

Generally speaking, researching the reference pages for the objects you are working with will be the most informative. However, it's not always the most helpful because there usually is information that you won't be able to understand immediately (at least without further research). That said, the only way to get better at using official documentation is by using it! While you won't understand some things, you will be able to understand others. The trick is to be comfortable with not always getting full comprehension.

Let's cover a few features of MDN documentation for Web APIs. Many of these features will be true for all documentation on MDN. We'll look at an example of features for a reference on an Web API interface (an object type), and another for a reference on an Web API interface's property.

HTMLElement Reference (https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement)

This section covers helpful features of the reference pages for Web API interfaces.



In this article

- Instance properties
- Instance methods
- Events
- Specifications
- Browser compatibility
- See also

Related Topics

HTML DOM API

`HTMLElement`

▼ Properties

`contentEditable`

HTMLElement

The `HTMLElement` interface represents any HTML element. Some elements directly implement this interface, while others implement it via an interface that inherits it.

Instance properties

Inherits properties from its parent, `Element`.

`HTMLElement.accessKey`

A string representing the access key assigned to the element.

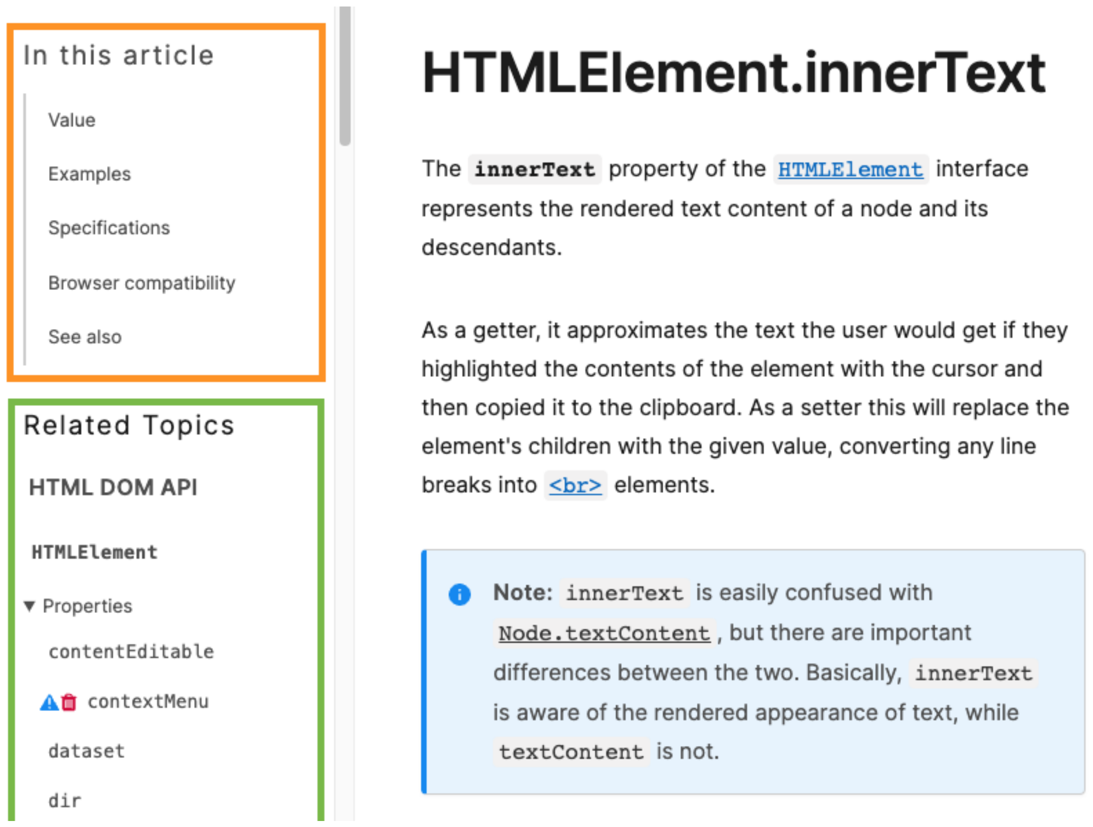
The start of every reference page includes a short description of the object type. You should always start here. What follows are various sections that detail the object's properties, events, and more. Take not of a few features about these reference pages:

- The orange box in the image highlights the topics that are covered within the "article", the reference page for the `HTMLElement` object. You should always scan this section to get a sense of what's in the page:
    - **instance properties** and **instance methods** will always list the methods and properties that belong to this object type that we may want to use in our code. For example, we'd find `HTMLElement.innerText` listed within the "instance properties" section.
    - **events** will list the events that fire on DOM elements of this object type.
    - **see also** lists additional and related resource pages to learn more.

- The green box in the image highlights topics related to the object type we're viewing, as well as links to other reference pages. Perusing this list is a quick way to get an overview of what this object contains. This section may include the following:
    - The Web API specification that this object type belongs to. In this case, we see HTML DOM API listed, which is the specification that the `HTMLElement` interface belongs to.
    - A list of all properties and methods belonging to the object types.
    - A list of events associated with this object.
    - A list of other object types that this object inherits from.
    - A list of related topics.
- The purple box highlights a diagram that shows the chain of inheritance for the object type we're viewing. Note that this diagram is clickable! Just click the name of an object type in the diagram to be taken to its reference page.

HTMLElement.innerText Reference (https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/innerText)

This section covers helpful features of the reference pages for a Web API interface's property.


In this article

Value

Examples

Specifications

Browser compatibility

See also

Related Topics

HTML DOM API

HTMLElement

▼ Properties

contentEditable

contextMenu

dataset

dir

HTMLElement.innerText

The `innerText` property of the `HTMLElement` interface represents the rendered text content of a node and its descendants.

As a getter, it approximates the text the user would get if they highlighted the contents of the element with the cursor and then copied it to the clipboard. As a setter this will replace the element's children with the given value, converting any line breaks into `<br>` elements.

Note: `innerText` is easily confused with `Node.textContent`, but there are important differences between the two. Basically, `innerText` is aware of the rendered appearance of text, while `textContent` is not.

There are many similarities between reference pages for objects and a property of an object, so we'll just highlight the important differences.

- The orange box in the image highlights the topics that are covered in this reference page:
  - The **value** section describes the return value and data type of the property and other helpful details. Always check out this section!
  - The **examples** section covers usage examples. Often there are multiple ways to use a property or method. It's okay not to understand everything in this section.
  - The **see also** section can include links to additional resources.
- The green box in the image highlights related topics with links to reference pages. In this case, the related topics are the object type the property belongs to and other details about the object type in general. It's helpful to peruse this section to get an overview look at related topics.

- Finally notice the "Note" section in a blue box on the reference page. MDN will highlight important notes in boxes like these and you should always take a moment to read these. MDN also has red boxes for warnings.

Previous (/introduction-to-programming/javascript-and-web-browsers/practice-forms)
Next (/introduction-to-programming/javascript-and-web-browsers/branching)

Lesson 58 of 75
Last updated more than 3 months ago.

disable dark mode

Epicodus (http://www.epicodus.com)