

Alternative Approach to find the Middle

The previous approach didn't work, so we'll try to use a different approach to find the middle of the cursor.

WE'LL COVER THE FOLLOWING ^

- The range property
- Trying out a solution

The range property

Our previous attempt of wrapping the text we want in a span and getting the position of the span didn't work. Time to Google for another lead. Searching for “get position of text”, I find a `range` property which I saw in the `getSelection` API but hadn't paid attention to.

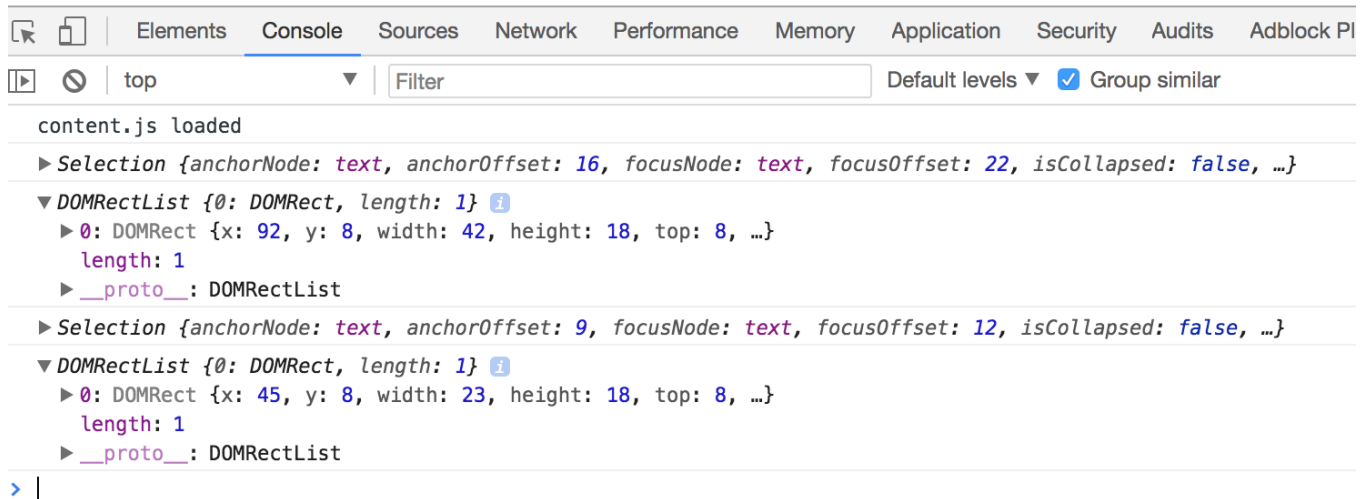
```
document.onmouseup = () => {
  const selection = document.getSelection();
  console.log(selection);
  const anchorNode = selection.anchorNode;
  const focusNode = selection.focusNode;

  if (anchorNode !== focusNode) {
    // Cross-paragraph selection
    return;
  }

  const selectedText = anchorNode.data.substring(selection.anchorOffset, selection.focusOffset);

  console.log(selection.getRangeAt(0).getClientRects());
}
```

we'll begin by scoping the
what what



The screenshot shows the Chrome DevTools Console with the 'Console' tab selected. The filter is set to 'top'. The console output shows the following:

```
content.js loaded
▶ Selection {anchorNode: text, anchorOffset: 16, focusNode: text, focusOffset: 22, isCollapsed: false, ...}
▼ DOMRectList {0: DOMRect, length: 1} ⓘ
  ▶ 0: DOMRect {x: 92, y: 8, width: 42, height: 18, top: 8, ...}
    length: 1
    __proto__: DOMRectList
▶ Selection {anchorNode: text, anchorOffset: 9, focusNode: text, focusOffset: 12, isCollapsed: false, ...}
▼ DOMRectList {0: DOMRect, length: 1} ⓘ
  ▶ 0: DOMRect {x: 45, y: 8, width: 23, height: 18, top: 8, ...}
    length: 1
    __proto__: DOMRectList
> |
```

Ah, this is exactly what we need! Sometimes we do extra work trying to use the tools we're familiar with. Part of being an experienced frontend engineer is just knowing what exists. As you progress, you gain valuable intuition about what's possible and what's impossible, what tasks are straightforward, and which are minefields (ahem- CSS animations), which solutions degrade the user experience and which solutions are cheap on CPU, and then some. This doesn't just apply to frontend development, by the way. Experienced engineers often choose the best solutions partly due to just being **aware** of so many other solutions and their outcomes.

Web development is unique in that it's constantly evolving at a pace much quicker compared to a lot of domains in software engineering like databases or iOS. The "right way" to do things in web development changes rapidly. Frameworks trend and fade on the order of a small number of years (whereas popular databases stay the same for decades). The number of open-source libraries that could potentially fit what you're doing is vast and is growing every day. Even JavaScript itself is iterating drastically in features. A senior JavaScript developer who's been in a coma since 2014 would hardly recognize web code written today.

But that's alright, because if they have the foundation, they can adapt quickly. Frameworks, libraries, and even syntax can change, but keep digging into the

source code, and it often distills to more or less of the same. Whether you're calling `render()` in React, or `$('.class').html()` in jQuery, they rely on JavaScript primitives like `document.getElementsByClassName` and `innerHTML`. The constant change can be a point of contention and scare newcomers away, but we should embrace the change because the improvements are in our favor. There are smart people evaluating changes made by smart people. Change for the sake of change will be disregarded, and change that empowers developers more will be embraced. The former you will never have to hear about, and the latter you won't regret learning. I don't doubt that in subsequent years, there will continue to be changes made to popular JavaScript that allow us to do more with less, paradigm shifts that allow codebases to more sustainably be worked on in large teams, tooling that helps how we catch bugs before they occur, and other improvements.

As you've seen in this lesson, there's no need to memorize exactly how everything is done. Non-programmers looking in would be surprised at the lack of implementation detail we carry in our working memory. Far more important than memorizing the syntax for what you need to do is having a vague awareness of what's possible and how to Google for what you need.

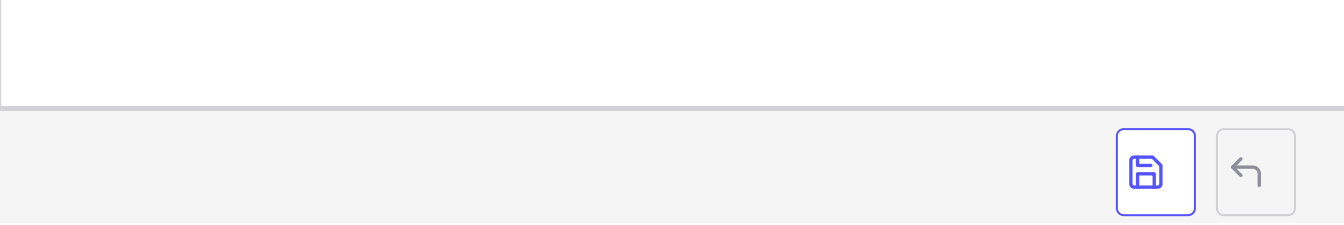
Anyway, we've just discovered `range`. Speaking of which, there it is in the React (the most popular framework as of this writing) source code!

<https://github.com/facebook/react/blob/24f824250fde6418569222f6e33b35ba9c1f1f46/packages/react-dom/src/client/ReactDOMSelection.js#L190>

Trying out a solution

Let's add a div and absolute position it given the information provided by `range`. *Google how to create a div in JavaScript*

| Output |
|---|
| JavaScript |
| HTML |
| CSS (SCSS) |
| we'll begin by scoping the what what |



Try highlighting text. We've finally found the middle!.