An imperative way to build a website!

reinhart1010.id – 1 February 2024 • (Updated 11 February 2024)
From https://reinhart1010.id/blog/2024/02/01/imperative-html. Scan the QR Code to view the article on your device or web browser.

> **(#_)!** This tutorial is definitely inspired by an interesting discussion in choosing a web tech stack in 2024. Of course, Imperative HTML is a more esoteric way of writing websites but still easy to learn for (#-)!

Have you ever:

- Built arcane user interface with C/C++?
- Experienced in using Tcl/Tk (e.g. `tkinter` in Python)?
- Rejected the singularity of XML-based files (including QML and XAML) in user interface development?

HTML itself was primarily based on XML (up to HTML 4.x). And we really hate if we have to write just to display the text “Hello, World!” this way:

```
<!DOCTYPE html>
<html>
  <body>
    <p>Hello, World!</p>
  </body>
</html>
```

Like, there’s no `main` function there. And when you wanted to make these elements interactive, you have to deal with the mess of `document.getElementById()`?

A “Hello, World!” to Imperative HTML.

Imperative HTML is technically JavaScript in disguise of a HTML `<script>` tag. But what makes Imperative HTML different than Node.js-like “back-end JavaScript programming” is that this still produces valid HTML, and even could prevent syntactic HTML errors just because you forgot to put your right bracket or double quotes in the perfect place.

Or in other words, an Imperative HTML ~~program~~ ~~webapp~~ **programmable webpage** like this:

```
<!DOCTYPE HTML>
<script>
```

```
  function main() {
    let text = document.createElement("p");
    text.className = "text-lg";
    text.textContent = "Hello, World (>_ )!";
    document.body.appendChild(text);
  }
  window.onload = main;
</script>
```

is definitely safer to write than:

```
<p class="text-lg>Hello, World (>_ )!</p>
<!-- Oh no, you didn't escape the > in (>_ )! -->
<!-- ...and the missing double-quotes! -->
```

Imperative HTML is powered by JavaScript, so you can still benefit from JavaScript's tighter error-checking than regular HTML, where browsers attempt to guess the meaning and purpose of some HTML and CSS attributes, also known as quirks mode, because of legacy reasons.

And since it is powered by JavaScript, the political dignity of "Imperative HTML" itself can be raised in two ways:

1. By declaring that unlike HTML, Imperative HTML is a true programming language, full with Turing-complete selection and repetition control structures (#-);
2. By giving web developers more power to fight against people who are campaigning against JavaScript, especially for political reasons. Who knows that Imperative HTML gives devs more freedom to do so (#o)?

And of course, you can slap on the power of TypeScript, if you really can, to ensure that you're writing HTML tags and appropriate styles in a standards-compliant way. Isn't this also a benefit of React's JSX and CSS-in-JS?

Imperative HTML *is* the intended way.

This statement is controversial, of course, but the original XML-like HTML truly feels like the conventional way to write proper websites, yet our reinhart1010.id and alterine0101.id websites are still written in *that* conventional way.

But remember, web browsers are still tasked to parse these HTML tags and convert them into internal structs which make up the today's **Document Object Model (DOM)**. Having to learn that HTML tags are objects would never be easier without attempting to learn how to declare HTML tags in the object-oriented way.

Not to mention React, some of the world's most loved web frameworks, still renders your elements this way. Since React features a Virtual DOM by default, they have to re-render the hand-crafted ~~HTML~~ JSX elements in the same way of this imperative tutorial: `document.createElement()` and so.

So far, the only main disadvantage of Imperative HTML is that these webpages will not be good for SEOs, because the document is no longer written in machine-readable format (note: we can still leave some metadata inside `<head>` before `<script>` in Imperative HTML), causing JavaScript

performance overhead, yadda-yadda, and so on. But the main irony is that even modern HTML scraper and parsers like BeautifulSoup parses regular HTML into objects the same way as browsers do, so imperative programming should be the way, right?

Just right to the tutorial!

In Imperative HTML, you (still) write code inside a HTML file, but you have to set up a few tags to make the web browser know that you're writing Imperative that supports HTML5:

```
<!DOCTYPE html><script>
// Your code here...
</script>
```

The ending `</script>` tag is optional, just like the `?>` part of PHP scripts, but we highly recommend it as part of our coding convention.

The `main()` function shown earlier is actually optional, but also recommended to ensure that the function is properly executed after the web browser is ready to load (using `window.onload = main;`.

Declaring a HTML tag

The usual way to write a HTML content is to use one of its supported tags, then adds attributes,

```
<p id="hero-text" class="">
  Hello, World (&gt;_ )!
</p>
```

The `p` is the **Element Name**, `id` and `class` is just some of the attributes, and the final, `Hello, World (>_ )!` text is the `textContent`. In Imperative HTML, it is just as easy as this:

```
// Make sure you have put this code after the <script> tag!
let text = document.createElement("p");
text.className = "text-lg";
text.textContent = "Hello, World (>_ )!";
document.body.appendChild(text);
```

But if you love to explicitly declare the class, you can use this instead:

```
// HTMLParagraphElement only applies to <p> tags.
let text = new HTMLParagraphElement();
text.className = "text-lg";
text.textContent = "Hello, World (>_ )!";
document.body.appendChild(text);
```

In Imperative HTML, you can assign event triggers even before attaching it to the page!

```
// Create a button element that prints the sentence on click
let btn = document.createElement("button");
btn.textContent = "Click Me!";
btn.addEventListener("click", function(e) {
```

```
  alert("Hello, World (>_ )!");
}, false);
```

Displaying tags into the webpage

And now, the most important part of this is to render these elements into the web browser. Now, our simplistic HTML structure of:

```
<!DOCTYPE html><script>
// ...
</script>
```

will be very likely to be rendered as this in most web browsers:

```
<!DOCTYPE html>
<html>
  <head>
    <script>
      // ...
    </script>
  </head>
  <body></body>
</html>
```

We can see that HTML has the `<head>`, the place to put the webpage's metadata, and `<body>`, the actual content displayed in the web browser. As a quick reference,

- Use `document.head` or `document.body` to modify the `<head>` or `<body>`, respectively
- These two parts are still considered as `HTMLElements`, which means you can use `.appendChild()` to append the element child (e.g. add new item to a HTML unordered list / `<ul>` / `HTMLUListElement`)
- Just like `clear()` to clear the Terminal console in other programming languages, you can use `document.body.textcontent = ""` to clear out the contents in the entire page.

Now, using the interactive button example, we can simply add them into the webpage using the following code:

```
<!DOCTYPE html><script>
function main() {
  // Create a button element that prints the sentence on click
  let btn = document.createElement("button");
  btn.textContent = "Click Me!";
  btn.addEventListener("click", function(e) {
    alert("Hello, World (>_ )!");
  }, false);

  // Attach them to the webpage
  document.body.appendChild(btn);
}

// Attach the main function when the webpage is ready to load!
```

```
window.onload = main;
</script>
```

Well, I think that's all for now. Of course, there will be many interesting ways to use Imperative HTML, so stay tuned and follow us! □