# Detailed Explanation of CSS Properties

Of course. Here is a detailed explanation of all the relevant CSS properties and concepts used in your code.

---

## Global Reset & `box-sizing`

\* {

margin: 0;

padding: 0;

box-sizing: border-box;

}

\* `\*` (The Universal Selector): Applies the following styles to \*\*every single HTML element\*\* on the page.

\* `margin: 0;` & `padding: 0;`: This is a "CSS reset." Browsers apply default margins and padding to elements like `<body>`, `<h1>`, `<p>`, etc. This removes all those defaults, giving you a clean slate to work from and ensuring consistency across different browsers.

\* `box-sizing: border-box;`: This is one of the most important concepts in modern CSS layout.

\* \*\*Default (`content-box`)\*\*: An element's `width` and `height` are calculated \*\*only\*\* on the content. If you add `padding` or `border`, it gets added \*on top\* of the width, making the element larger than you intended.

\* \*\*`border-box`\*\*: The element's `width` and `height` properties \*\*include\*\* the content, padding, and border. If you set `width: 100px;` and `padding: 20px;`, the content will shrink to `60px` to make the total element width `100px`. This makes sizing predictable and much easier to manage.

---

## `<body>` Styling

body {

background-image: linear-gradient(to right, #434343 0%, black 100%);

background-blend-mode: multiply, multiply;

min-height: 100vh;

width: 100%;

font-family: "Inter Tight", sans-serif;

}

\* `background-image: linear-gradient(...)`: Creates a smooth color gradient that goes from dark gray (`#434343`) on the left to pure black (`black`) on the right.

\* `background-blend-mode: multiply, multiply;`: This property blends the element's background layers together. In this case, it's blending the gradient with itself (if there were multiple backgrounds) to create a richer, darker effect. (Note: With only one background image, this property may have no visible effect unless another background like a `background-color` is also set).

\* `min-height: 100vh;`: Ensures the body is \*\*at least\*\* as tall as the user's entire viewport (`100vh` = 100% of the Viewport Height). This prevents content from being shorter than the screen.

\* `width: 100%;`: Explicitly sets the body's width to 100% of its container (the viewport). This is often a default but ensures consistency.

\* `font-family: "Inter Tight", sans-serif;`: Sets the default font for all text on the page. The browser will try to use "Inter Tight" first. If that font isn't loaded or available, it will fall back to the system's generic `sans-serif` font.

---

## Navbar & Flexbox

.navbar {

display: flex;

justify-content: flex-end;

align-items: center;

width: 100%;

position: sticky;

top: 0;

height: auto;

min-height: 70px;

z-index: 1000;

padding: 1rem 2rem;

}

\* `display: flex;`: Turns the `.navbar` into a \*\*flex container\*\*. Its direct children (the `.navbar-nav` list) become \*\*flex items\*\* and are laid out in a row by default.

\* `justify-content: flex-end;`: Controls alignment on the \*\*main axis\*\* (horizontal, by default). This pushes the flex items (the navigation list) to the far right (the "end") of the navbar.

\* `align-items: center;`: Controls alignment on the \*\*cross axis\*\* (vertical, by default). This centers the flex items vertically within the navbar.

\* `position: sticky; top: 0;`: This creates a "sticky" element. The navbar will act normally as you scroll until it reaches the top of the viewport (`top: 0`). At that point, it "sticks" there and scrolls with the user, always remaining visible at the top of the screen.

\* `z-index: 1000;`: Controls the stacking order of elements. An element with a higher `z-index` appears on top of elements with a lower one. A high value like `1000` ensures the navbar is always on top of other page content when scrolling.

\* `padding: 1rem 2rem;`: Adds space \*inside\* the navbar. `1rem` on the top/bottom and `2rem` on the left/right.

---

## Navigation Links (`<a>` tags)

.navbar a {

text-decoration: none;

color: rgb(255, 255, 255);

font-size: clamp(0.8rem, 2vw, 0.9rem);

padding: 0.5rem 1rem;

transition: color 0.3s ease;

}

\* `text-decoration: none;`: Removes the default underline from hyperlinks.

\* `color: rgb(255, 255, 255);`: Sets the text color to white.

\* `font-size: clamp(0.8rem, 2vw, 0.9rem);`: This is a \*\*clamp\*\* function for responsive typography. It sets a fluid font size that scales with the viewport width (`2vw` = 2% of the viewport width) but will never be smaller than `0.8rem` or larger than `0.9rem`.

\* `padding: 0.5rem 1rem;`: Adds clickable space around the link text. This improves usability.

\* `transition: color 0.3s ease;`: This animates any change to the `color` property. When a user hovers over the link, the color change will take `0.3` seconds and use an `ease` timing function (start slow, speed up, then end slow) instead of an abrupt change.

---

## Hero Section

.hero {

min-height: calc(100vh - 70px);

display: flex;

justify-content: center;

align-items: center;

margin: 0;

padding: 0;

}

\* `min-height: calc(100vh - 70px);`: This is the most important part. It uses the `calc()` function to dynamically calculate the height. It tells the hero section to be \*\*at least\*\* as tall as the full viewport (`100vh`) \*\*minus\*\* the height of the navbar (`70px`). This ensures the hero section perfectly fills the screen without being hidden behind the sticky navbar.

\* `display: flex; justify-content: center; align-items: center;`: This is a classic centering technique. It turns the hero into a flex container and centers its single child (`.hero-content`) perfectly both horizontally and vertically on the page.

---

## Hero Content (Flexbox Wrapper)

.hero-content {

display: flex;

flex-wrap: wrap;

justify-content: center;

gap: 2rem;

align-items: center;

}

.hero-content > div {

flex: 1 1 300px;

max-width: 500px;

}

\* `.hero-content` is now also a flex container for its child `div`s.

\* `flex-wrap: wrap;`: Allows the child `div`s to wrap onto new lines if there isn't enough horizontal space.

\* `gap: 2rem;`: Puts a `2rem` gap between the child `div`s, both horizontally and vertically.

\* `align-items: center;`: Centers the child `div`s vertically relative to each other in their row.

\* `.hero-content > div` (Direct Child Selector): Targets only the `div`s that are immediate children of `.hero-content`.

\* `flex: 1 1 300px;` (Shorthand for `flex-grow`, `flex-shrink`, `flex-basis`):

\* `flex-grow: 1`: Allows the div to grow to fill any available space in the row.

\* `flex-shrink: 1`: Allows the div to shrink if necessary.

\* `flex-basis: 300px`: Sets the ideal or starting width before growing or shrinking. It acts as a \*\*minimum width\*\*.

\* `max-width: 500px;`: Prevents the divs from growing infinitely and becoming too wide on large screens, setting a \*\*maximum width\*\*. The layout will fluidly scale between `300px` and `500px`.