Creating a responsive website in pure CSS involves using media queries and flexible layout techniques to make your website adapt to different screen sizes and devices. Here's a breakdown of the key concepts and techniques:

1. Media Queries:

- **Purpose:** Media queries allow you to apply different CSS styles based on the characteristics of the device or screen, such as width, height, orientation, and resolution.
- **Syntax:**

```
    @media (condition) {
  /* CSS styles to apply when the condition is met */
   }
```

- **Common Media Query Features:**
    - width: Matches the width of the viewport.
    - height: Matches the height of the viewport.
    - min-width: Matches a minimum width.
    - max-width: Matches a maximum width (most commonly used for responsive design).
    - orientation: Matches the orientation of the device (portrait or landscape).
    - resolution: Matches the resolution of the device.
- **Example:**

```
/* Styles for screens smaller than 768px wide (typical tablet size) */
 @media (max-width: 767px) {
body {
  font-size: 14px;
}
.container {
  width: 90%;
}
nav ul {
  flex-direction: column; /* Stack navigation items vertically */
}
}

/* Styles for screens larger than 768px wide */
 @media (min-width: 768px) {
.container {
  width: 960px;
}
nav ul {
  flex-direction: row; /* Display navigation items horizontally */
```

```
  }
  }
```

2. Flexible Layouts:

- **Fluid Widths:** Use percentages or viewport units (vw, vh) for widths instead of fixed pixel values. This allows elements to scale proportionally to the screen size.

```
  .container {
    width: 90%; /* Takes up 90% of the screen width */
    max-width: 1200px; /* Sets a maximum width */
    margin: 0 auto; /* Centers the container */
  }

  img {
    max-width: 100%; /* Images scale down as needed */
    height: auto;
  }
```

- **Flexible Heights (Less Common):** Generally, let the content determine the height. However, you might use viewport units (vh) in specific situations.
- **Flexbox:** Flexbox is excellent for creating flexible and responsive layouts, especially for one-dimensional layouts (rows or columns).

```
   nav ul {
    display: flex; /* Enable Flexbox */
    justify-content: space-around; /* Distribute items evenly */
   }

   nav li {
    margin: 10px;
   }
```

- **Grid:** CSS Grid is a powerful layout system for two-dimensional layouts (rows and columns). It's great for creating complex grid-based designs.

```
  .grid-container {
   display: grid;
   grid-template-columns: repeat(auto-fit, minmax(200px, 1fr)); /* Responsive
columns */
   gap: 20px;
 }
```

3. Responsive Images:

- **srcset and sizes Attributes:** The srcset attribute lists multiple image files with different resolutions, and the sizes attribute tells the browser the size of the image relative to the viewport. The browser then chooses the most appropriate image.

```
   <img srcset="image-small.jpg 300w, image-medium.jpg 600w, image-
large.jpg 900w"
   sizes="(max-width: 400px) 300px, (max-width: 800px) 600px, 900px"
  src="image-large.jpg" alt="Description">
```

- **<picture> Element:** The <picture> element allows you to provide different image formats (e.g., WebP) for different browsers.

```
  <picture>
    <source srcset="image.webp" type="image/webp">
    <source srcset="image.jpg" type="image/jpeg">
    <img src="image.jpg" alt="Description">
  </picture>
```

4. Viewport Meta Tag:

- **Purpose:** The viewport meta tag is essential for responsive design. It tells the browser how to scale the page on different devices.
- **Syntax:** Place this tag in the <head> of your HTML document:

```
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

5. Mobile-First Approach:

- Start by designing for the smallest screen size (mobile) and then progressively enhance the layout for larger screens using media queries. This approach often results in a better user experience on mobile devices.

Example (Combining Techniques):

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Responsive Example</title>
<style>
  body {
    font-family: sans-serif;
  }
```

```
  .container {
    width: 90%;
    max-width: 1200px;
    margin: 0 auto;
  }
  nav ul {
    list-style: none;
    padding: 0;
    margin: 0;
    display: flex;
    justify-content: space-around;
  }
  nav li {
    margin: 10px;
  }
  nav a {
    text-decoration: none;
  }
  .grid-container {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));
    gap: 20px;
  }
  .grid-item img {
    max-width: 100%;
    height: auto;
  }

  @media (max-width: 767px) {
    nav ul {
      flex-direction: column;
    }
  }
</style>
</head>
<body>
<div class="container">
  <nav>
    <ul>
      <li><a href="#">Home</a></li>
      <li><a href="#">About</a></li>
      <li><a href="#">Services</a></li>
      <li><a href="#">Contact</a></li>
    </ul>
  </nav>

  <div class="grid-container">
    <div class="grid-item">
```

```
        <img src="image1.jpg" alt="Image 1">
      </div>
      <div class="grid-item">
        <img src="image2.jpg" alt="Image 2">
      </div>
      <div class="grid-item">
        <img src="image3.jpg" alt="Image 3">
      </div>
    </div>
  </div>
</body>
</html>
```

This example demonstrates a basic responsive layout with a navigation menu that adapts to smaller screens and a grid-based image gallery. Remember to replace the placeholder image URLs with your actual image paths. This is a starting point. Real-world responsive designs often require more complex media queries and layout adjustments. Using a CSS framework like Bootstrap or Tailwind CSS can significantly speed up the development process.